Group Ring Module

Hidetsune Kobayashi, L. Chen, H. Murao

April 17, 2016

Abstract

The theory of groups, rings and modules is developed to a great depth. Group theory results include Zassenhaus's theorem and the Jordan-Hoelder theorem. The ring theory development includes ideals, quotient rings and the Chinese remainder theorem. The module development includes the Nakayama lemma, exact sequences and Tensor products.

Contents

1 Preliminaries **4**
- 1.1 Lemmas for logical manipulation . . . 4
- 1.2 Natural numbers and Integers . . . 4
  - 1.2.1 Integers . . . 6
- 1.3 Sets . . . 10
  - 1.3.1 A short notes for proof steps . . . 10
  - 1.3.2 Sets . . . 10
- 1.4 Functions . . . 13
- 1.5 Nsets . . . 25
  - 1.5.1 Lemmas for existence of reduced chain. . . . 39
- 1.6 Lower bounded set of integers . . . 41
- 1.7 Augmented integer: integer and $\infty - \infty$ . . . 43
  - 1.7.1 Ordering of integers and ordering nats . . . 65
  - 1.7.2 The $\leq$ Ordering . . . 65
  - 1.7.3 Aug ordering . . . 70
- 1.8 Amin, amax . . . 76
  - 1.8.1 Maximum element of a set of ants . . . 80
- 1.9 Cardinality of sets . . . 84
  - 1.9.1 Lemmas required in Algebra6.thy . . . 96

2 Ordered Set **100**
- 2.1 Basic Concepts of Ordered Sets . . . 100
  - 2.1.1 Total ordering . . . 106
  - 2.1.2 Two ordered sets . . . 108
  - 2.1.3 Homomorphism of ordered sets . . . 108
- 2.2 Pre elements . . . 171
- 2.3 Transfinite induction . . . 175
- 2.4 *Ordered-set2.* Lemmas to prove Zorn's lemma. . . . 176
- 2.5 Zorn's lemma . . . 183

3 Group Theory. Focused on Jordan Hoelder theorem **231**
- 3.1 Definition of a Group . . . 231
- 3.2 Subgroups . . . 236

3.3 Cosets . . . 245
3.4 Normal subgroups and Quotient groups . . . 253
3.5 Setproducts . . . 263
3.6 Preliminary lemmas for Zassenhaus . . . 268
3.7 Homomorphism . . . 279
3.8 Gkernel . . . 282
3.9 Image . . . 285
3.10 Induced homomorphisms . . . 289
3.10.1 Homomorphism therems . . . 292
3.11 Isomorphims . . . 299
3.11.1 An automorphism groups . . . 301
3.11.2 Complete system of representatives . . . 301
3.12 Zassenhaus . . . 306
3.13 Chain of groups I . . . 310
3.14 Existence of reduced chain . . . 317
3.15 Existence of reduced chain and composition series . . . 331
3.16 Chain of groups II . . . 333
3.17 Jordan Hoelder theorem . . . 347
3.17.1 *Rfn-tools*. Tools to treat refinement of a cmpser, rtos. 347
3.18 Abelian groups . . . 378
3.18.1 Homomorphism of abelian groups . . . 387
3.18.2 Quotient abelian group . . . 395
3.19 Direct product and direct sum of abelian groups, in general case . . . 398
3.19.1 Characterization of a direct product . . . 411

4 Ring theory 420
4.1 Definition of a ring and an ideal . . . 420
4.2 Calculation of elements . . . 429
4.2.1 nscale . . . 429
4.2.2 npow . . . 431
4.2.3 nsum and fSum . . . 433
4.3 Ring homomorphisms . . . 443
4.3.1 Ring of integers . . . 459
4.4 Quotient rings . . . 461
4.5 Primary ideals, Prime ideals . . . 479
4.6 Operation of ideals . . . 516
4.7 Direct product1, general case . . . 523
4.8 Chinese remainder theorem . . . 535
4.9 Addition of finite elements of a ring and *ideal-multiplication* . 550
4.10 Extension and contraction . . . 578
4.11 Complete system of representatives . . . 581
4.12 Polynomial ring . . . 583
4.13 Addition and multiplication of *polyn-exprs* . . . 584

4.13.1 Simple properties of a *polyn-ring* . . . 584
4.13.2 Coefficients of a polynomial . . . 586
4.13.3 Addition of *polyn-exprs* . . . 587
4.13.4 Multiplication of *pol-exprs* . . . 597
4.13.5 Multiplication . . . 597
4.14 The degree of a polynomial . . . 607
4.14.1 Multiplication of polynomials . . . 639
4.14.2 Degree with value in *aug-minf* . . . 643
4.15 Homomorphism of polynomial rings . . . 646
4.16 Relatively prime polynomials . . . 675
4.16.1 Polynomial, coeff mod P . . . 685

5 Modules **727**
5.1 Basic properties of Modules . . . 727
5.2 Injective hom, surjective hom, bijective hom and inverse hom 742
5.3 nsum and Generators . . . 802
5.3.1 Sum up coefficients . . . 810
5.3.2 Free generators . . . 824
5.4 nsum and Generators (continued) . . . 839
5.5 Existence of homomorphism . . . 846
5.6 Nakayama lemma . . . 879
5.7 Direct sum and direct products of modules . . . 886
5.8 Exact sequence . . . 897
5.9 Tensor product . . . 914

6 Construction of an abelian group **921**
6.1 Free generated abelian group I, direct sum and direct product 2921
6.2 Abelian group generated by a singleton (constructive) . . . 928
6.3 Abelian Group generated by one element II (nonconstructive) 951
6.4 Free Generated Modules (constructive) . . . 960
6.5 A fgmodule and a free module . . . 969
6.6 Direct sum, again . . . 976
6.6.1 Existence of the tensor product . . . 983

theory *Algebra1*
imports *Main* ~~/*src*/*HOL*/*Library*/*FuncSet*
begin

Chapter 1

Preliminaries

Some of the lemmas of this section are proved in src/HOL/Integ of Isabelle version 2003.

1.1 Lemmas for logical manipulation

lemma *True-then*:$True \longrightarrow P \Longrightarrow P$
by *simp*

lemma *ex-conjI*:$[\![P\ c;\ Q\ c]\!] \Longrightarrow \exists\, c.\ P\ c \wedge Q\ c$
by *blast*

lemma *forall-spec*:$[\![\ \forall\, b.\ P\ b \longrightarrow Q\ b;\ P\ a]\!] \Longrightarrow Q\ a$
by *simp*

lemma *a-b-exchange*:$[\![a;\ a = b]\!] \Longrightarrow b$
by *simp*

lemma *eq-prop*:$[\![\ P;\ P = Q]\!] \Longrightarrow Q$
by *simp*

lemma *forball-contra*:$[\![\forall\, y{\in}A.\ P\ x\ y \longrightarrow \neg\ Q\ y;\ \forall\, y{\in}A.\ Q\ y \vee R\ y]\!] \Longrightarrow$
$\forall\, y{\in}A.\ (\neg\ P\ x\ y) \vee R\ y$
by *blast*

lemma *forball-contra1*:$[\![\forall\, y{\in}A.\ P\ x\ y \longrightarrow Q\ y;\ \forall\, y{\in}A.\ \neg\ Q\ y]\!] \Longrightarrow \forall\, y{\in}A.\ \neg\ P\ x\ y$
by *blast*

1.2 Natural numbers and Integers

Elementary properties of natural numbers and integers

lemma *nat-nonzero-pos*:$(a{::}nat) \neq 0 \Longrightarrow 0 < a$

by *simp*

lemma *add-both*:$(a::nat) = b \Longrightarrow a + c = b + c$
by *simp*

lemma *add-bothl*:$a = b \Longrightarrow c + a = c + b$
by *simp*

lemma *diff-Suc*:$(n::nat) \leq m \Longrightarrow m - n + Suc\ 0 = Suc\ m - n$
by *arith*

lemma *le-convert*:$[\![a = b;\ a \leq c]\!] \Longrightarrow b \leq c$
by *simp*

lemma *ge-convert*:$[\![a = b;\ c \leq a]\!] \Longrightarrow c \leq b$
by *simp*

lemma *less-convert*:$[\![\ a = b;\ c < b\ ]\!] \Longrightarrow c < a$
by *auto*

lemma *ineq-conv1*:$[\![a = b;\ a < c]\!] \Longrightarrow b < c$
by *simp*

lemma *diff-Suc-pos*:$0 < a - Suc\ 0 \Longrightarrow\ 0 < a$
by *simp*

lemma *minus-SucSuc*:$a - Suc\ (Suc\ 0) = a - Suc\ 0 - Suc\ 0$
by *simp*

lemma *Suc-Suc-Tr*:$Suc\ (Suc\ 0) \leq n \Longrightarrow Suc\ (n - Suc\ (Suc\ 0)) = n - Suc\ 0$
by *arith*

lemma *Suc-Suc-less*:$Suc\ 0 < a \Longrightarrow Suc\ (a - Suc\ (Suc\ 0)) < a$
by *arith*

lemma *diff-zero-eq*:$n = (0::nat) \Longrightarrow m = m - n$
by *simp*

lemma *Suc-less-le*:$x < Suc\ n \Longrightarrow x \leq n$
by *auto*

lemma *less-le-diff*:$x < n \Longrightarrow x \leq n - Suc\ 0$
by *arith*

lemma *le-pre-le*:$x \leq n - Suc\ 0 \Longrightarrow x \leq n$
by *arith*

lemma *nat-not-less*:$\neg\ (m::nat) < n \Longrightarrow n \leq m$
by (*rule contrapos-pp*, *simp+*)

lemma *less-neq*:$n < (m::nat) \Longrightarrow n \neq m$
by (*simp add*:*nat-neq-iff*[*THEN sym*, *of n m*])

lemma *less-le-diff1*:$n \neq 0 \Longrightarrow ((m::nat) < n) = (m \leq (n - Suc\ 0))$
by *arith*

lemma *nat-not-less1*:$n \neq 0 \Longrightarrow (\neg\ (m::nat) < n) = (\neg\ m \leq (n - Suc\ 0))$
by *arith*

lemma *nat-eq-le*:$m = (n::nat) \Longrightarrow m \leq n$
by *simp*

1.2.1 Integers

lemma *non-zero-int*: $(n::int) \neq 0 \Longrightarrow 0 < n \vee n < 0$
by *arith*

lemma *zgt-0-zge-1*:$(0::int) < z \Longrightarrow 1 \leq z$
by *arith*

lemma *not-zle*:$(\neg\ (n::int) \leq m) = \ (m < n)$
by *auto*

lemma *not-zless*:$(\neg\ (n::int) < m) = (m \leq n)$
by *auto*

lemma *zle-imp-zless-or-eq*:$(n::int) \leq m \Longrightarrow n < m \vee n = m$
by *arith*

lemma *zminus-zadd-cancel*: $-\ z + (z + w) = (w::int)$
by *simp*

lemma *int-neq-iff*:$((w::int) \neq z) = (w < z) \vee (z < w)$
by *auto*

lemma *zless-imp-zle*:$(z::int) < z' \Longrightarrow z \leq z'$
by *simp*

lemma *zdiff*:$z - (w::int) = z + (-\ w)$
by *simp*

lemma *zle-zless-trans*:$[\![\ (i::int) \leq j;\ j < k]\!] \Longrightarrow i < k$
by *arith*

lemma *zless-zle-trans*:$[\![\ (i::int) < j;\ j \leq k]\!] \Longrightarrow i < k$
by *arith*

lemma *zless-neq*:$(i::int) < j \Longrightarrow i \neq j$

by *simp*

lemma *int-mult-mono*:$[\![\ i < j;\ (0::int) < k\ ]\!] \Longrightarrow k * i < k * j$
apply (*frule zmult-zless-mono2-lemma* [*of i j nat k*])
apply *simp* **apply** *simp*
done

lemma *int-mult-le*:$[\![i \leq j;\ (0::int) \leq k]\!] \Longrightarrow k * i \leq k * j$
apply (*simp add*:*order-le-less*)
apply (*case-tac i* $<$ *j*)
apply (*case-tac 0* $<$ *k*)
apply *simp*
apply *simp*
apply *simp*
done

lemma *int-mult-le1*:$[\![i \leq j;\ (0::int) \leq k]\!] \Longrightarrow i * k \leq j * k$
apply (*simp add*:*mult.commute*[*of* - *k*])
apply (*simp add*:*int-mult-le*)
done

lemma *zmult-zminus-right*:$(w::int) * (-\ z) = -\ (w * z)$
apply (*insert distrib-left*[*of w z* $-z$])
apply *simp*
done

lemma *zmult-zle-mono1-neg*:$[\![(i::int) \leq j;\ k \leq 0]\!] \Longrightarrow j * k \leq i * k$
apply (*subgoal-tac 0* $\leq -\ k$) **prefer** *2* **apply** *simp*
apply (*frule int-mult-le* [*of i j* $-\ k$], *assumption*+)
apply (*simp add*:*mult.commute*)
done

lemma *zmult-zless-mono-neg*:$[\![(i::int) < j;\ k < 0]\!] \Longrightarrow j * k < i * k$
apply (*subgoal-tac 0* $< -k$,
frule int-mult-mono[*of i j* $-k$], *assumption*+,
simp add:*mult.commute*, *simp*)
done

lemma *zmult-neg-neg*:$[\![i < (0::int);\ j < 0\ ]\!] \Longrightarrow 0 < i * j$
apply (*frule zmult-zless-mono-neg*[*of i 0 j*], *assumption*)
apply *simp*
done

lemma *zmult-pos-pos*:$[\![(0::int) < i;\ 0 < j]\!] \Longrightarrow 0 < i * j$
apply (*frule int-mult-mono*[*of 0 i j*], *assumption*+)
apply (*simp add*:*mult.commute*)
done

lemma *zmult-pos-neg*:$[\![(0::int) < i;\ j < 0]\!] \Longrightarrow i * j < 0$

apply (*frule zmult-zless-mono-neg*[*of 0 i j*], *assumption*+, *simp*)
done

lemma *zmult-neg-pos*:⟦$i < (0::int)$; $0 < j$⟧ ⟹ $i * j < 0$
apply (*frule int-mult-mono*[*of i 0 j*], *assumption*+,
simp add:*mult.commute*)
done

lemma *zle*:$((z::int) \leq w) = (\neg\ (w < z))$
by *auto*

lemma *times-1-both*:⟦$(0::int) < z$; $z * z' = 1$⟧ ⟹ $z = 1 \wedge z' = 1$
apply (*subgoal-tac* $0 < z'$)
apply (*frule zgt-0-zge-1*[*of z'*],
subgoal-tac $z' = 1$, *simp*,
subgoal-tac $1 < z' \vee 1 = z'$, *thin-tac* $1 \leq z'$, *thin-tac* $0 < z'$)
apply (*rule contrapos-pp*, *simp*+,
frule int-mult-mono[*of 1 z' z*], *assumption*+, *simp*, *arith*)
apply (*rule contrapos-pp*, *simp*+, *simp add*:*zle*[*THEN sym*],
frule zless-imp-zle[*of 0 z*], *frule int-mult-le*[*of z' 0 z*],
assumption+, *simp*)
done

lemma *zminus-minus*:$i\ -\ -\ (j::int) = i + j$
by *simp*

lemma *zminus-minus-pos*:$(n::int) < 0 \Longrightarrow 0 < -\ n$
by *simp*

lemma *zadd-zle-mono*:⟦$w' \leq w$; $z' \leq (z::int)$⟧ ⟹ $w' + z' \leq w + z$
by *simp*

lemma *zmult-zle-mono*:⟦$i \leq (j::int)$; $0 < k$⟧ ⟹ $k * i \leq\ k * j$
apply (*case-tac* $i = j$) **apply** *simp*
apply (*frule zle-imp-zless-or-eq*[*of i j*])
apply (*thin-tac* $i \leq j$) **apply** *simp*
done

lemma *zmult-zle-mono-r*:⟦$i \leq (j::int)$; $0 < k$⟧ ⟹ $i * k \leq j * k$
apply (*frule zmult-zle-mono*[*of i j k*], *assumption*)
apply (*simp add*:*mult.commute*)
done

lemma *pos-zmult-pos*:⟦ $0 \leq (a::int)$; $0 < (b::int)$⟧ ⟹ $a \leq a * b$
apply (*case-tac* $a = 0$) **apply** *simp*
apply (*frule zle-imp-zless-or-eq*[*of 0 a*]) **apply** (*thin-tac* $0 \leq a$)
apply *simp*
done

lemma *pos-mult-l-gt*:$[\![(0::int) < w;\ i \leq j;\ 0 \leq i]\!] \Longrightarrow i \leq w * j$
by (*metis not-zless pos-zmult-pos order-trans mult.commute*)

lemma *pos-mult-r-gt*:$[\![(0::int) < w;\ i \leq j;\ 0 \leq i]\!] \Longrightarrow i \leq j * w$
apply (*frule pos-mult-l-gt*[*of w i j*], *assumption*+)
apply (*simp add*:*mult.commute*[*of w j*])
done

lemma *mult-pos-iff*:$[\![(0::int) < i;\ 0 \leq i * j\ ]\!] \Longrightarrow 0 \leq j$
apply (*rule contrapos-pp*, *simp*+)
apply (*cut-tac linorder-linear*[*of 0 j*]) **apply** *simp*
apply (*simp add*:*not-zle*)
apply (*frule int-mult-mono*[*of j 0 i*], *assumption*+) **apply** *simp*
done

lemma *zmult-eq*:$[\![(0::int) < w;\ z = z']\!] \Longrightarrow w * z = w * z'$
by *simp*

lemma *zmult-eq-r*:$[\![(0::int) < w;\ z = z']\!] \Longrightarrow z * w = z' * w$
by *simp*

lemma *zdiv-eq-l*:$[\![(0::int) < w;\ z * w = z' * w\ ]\!] \Longrightarrow z = z'$
by *simp*

lemma *zdiv-eq-r*:$[\![(0::int) < w;\ w * z = w * z'\ ]\!] \Longrightarrow z = z'$
by *simp*

lemma *int-nat-minus*:$0 < (n::int) \Longrightarrow nat\ (n - 1) = (nat\ n) - 1$
by *arith*

lemma *int-nat-add*:$[\![0 < (n::int);\ 0 < (m::int)]\!] \Longrightarrow (nat\ (n - 1)) + (nat\ (m - 1)) + (Suc\ 0) = nat\ (n + m - 1)$
by *arith*

lemma *int-equation*:$(x::int) = y + z \Longrightarrow x - y = z$
by *simp*

lemma *int-pos-mult-monor*:$[\![\ 0 < (n::int);\ 0 \leq n * m\ ]\!] \Longrightarrow 0 \leq m$
by (*rule mult-pos-iff*, *assumption*+)

lemma *int-pos-mult-monol*:$[\![\ 0 < (m::int);\ 0 \leq n * m\ ]\!] \Longrightarrow 0 \leq n$
apply (*rule int-pos-mult-monor*, *assumption*+)
apply (*simp add*:*mult.commute*)
done

lemma *zdiv-positive*:$[\![(0::int) \leq a;\ 0 < b]\!] \Longrightarrow 0 \leq a\ div\ b$
apply (*frule-tac* $a = 0$ **and** $a' = a$ **and** $b = b$ **in** *zdiv-mono1*, *assumption*+)
apply *simp*

done

lemma *zdiv-pos-mono-r*:$[\![(0::int) < w;\ w * z \leq w * z']\!] \Longrightarrow z \leq z'$
apply (*rule contrapos-pp*, *simp+*)
done

lemma *zdiv-pos-mono-l*:$[\![(0::int) < w;\ z * w \leq z' * w]\!] \Longrightarrow z \leq z'$
apply (*simp add*:*mult.commute*)
done

lemma *zdiv-pos-pos-l*:$[\![(0::int) < w;\ 0 \leq z * w]\!] \Longrightarrow 0 \leq z$
by (*simp add*:*mult.commute*, *frule zdiv-pos-mono-r*[*of w 0 z*], *simp*,
assumption)

1.3 Sets

1.3.1 A short notes for proof steps

1.3.2 Sets

lemma *inEx*:$x \in A \Longrightarrow \exists\, y{\in}A.\ y = x$
by *simp*

lemma *inEx-rev*: $\exists\, y{\in}A.\ y = x \Longrightarrow x \in A$
by *blast*

lemma *nonempty-ex*:$A \neq \{\} \Longrightarrow \exists\, x.\ x \in A$
by *blast*

lemma *ex-nonempty*:$\exists\, x.\ x \in A \Longrightarrow A \neq \{\}$
by *blast*

lemma *not-eq-outside*:$a \notin A \Longrightarrow \forall\, b{\in}A.\ b \neq a$
by *blast*

lemma *ex-nonempty-set*:$\exists\, a.\ P\ a \Longrightarrow \{x.\ P\ x\} \neq \{\}$
by *blast*

lemma *nonempty*: $x \in A \Longrightarrow A \neq \{\}$
by *blast*

lemma *subset-self*:$A \subseteq A$
by *simp*

lemma *conditional-subset*:$\{x{\in}A.\ P\ x\} \subseteq A$
by *blast*

lemma *bsubsetTr*:$\{x.\ x \in A \wedge P\ x\} \subseteq A$
by *blast*

lemma *sets-not-eq*:$[\![A \neq B;\ B \subseteq A]\!] \Longrightarrow \exists\, a{\in}A.\ a \notin B$
by *blast*

lemma *diff-nonempty*:$[\![A \neq B;\ B \subseteq A]\!] \Longrightarrow A - B \neq \{\}$
by *blast*

lemma *sub-which1*:$[\![A \subseteq B \vee B \subseteq A;\ x \in A;\ x \notin B]\!] \Longrightarrow B \subseteq A$
by *blast*

lemma *sub-which2*:$[\![A \subseteq B \vee B \subseteq A;\ x \notin A;\ x \in B]\!] \Longrightarrow A \subseteq B$
by *blast*

lemma *diff-sub*:$A - B \subseteq A$
by *blast*

lemma *nonempty-int*: $A \cap B \neq \{\} \Longrightarrow \exists\, x.\ x \in A \cap B$
by *blast*

lemma *no-meet1*:$A \cap B = \{\} \Longrightarrow \forall\, a \in A.\ a \notin B$
by *blast*

lemma *no-meet2*:$A \cap B = \{\} \Longrightarrow \forall\, a \in B.\ a \notin A$
by *blast*

lemma *elem-some*:$x \in A \Longrightarrow \exists\, y{\in}A.\ x = y$
by *blast*

lemma *singleton-sub*:$a \in A \Longrightarrow \{a\} \subseteq A$
by *blast*

lemma *eq-elem-in*: $[\![\ a \in A;\ a = b\ ]\!] \Longrightarrow b \in A$
by *simp*

lemma *eq-set-inc*: $[\![\ a \in A;\ A = B\ ]\!] \Longrightarrow a \in B$
by *simp*

lemma *eq-set-not-inc*:$[\![a \notin A;\ A = B\ ]\!] \Longrightarrow a \notin B$
by *simp*

lemma *int-subsets*: $[\![\ A1 \subseteq A;\ B1 \subseteq B\ ]\!] \Longrightarrow A1 \cap B1 \subseteq A \cap B$
by *blast*

lemma *inter-mono*:$A \subseteq B \Longrightarrow A \cap C \subseteq B \cap C$
by *blast*

lemma *sub-Un1*:$B \subseteq\ B \cup C$
by *blast*

lemma *sub-Un2*:$C \subseteq B \cup C$
by *blast*

lemma *subset-contr*:$[\![A \subset B; B \subseteq A]\!] \Longrightarrow False$
by *blast*

lemma *psubset-contr*:$[\![A \subset B; B \subset A]\!] \Longrightarrow False$
by *blast*

lemma *eqsets-sub*:$A = B \Longrightarrow A \subseteq B$
by *simp*

lemma *not-subseteq*: $\neg A \subseteq B \Longrightarrow \exists a \in A.\ a \notin B$
by *blast*

lemma *in-un1*:$[\![x \in A \cup B; x \notin B]\!] \Longrightarrow x \in A$
by *blast*

lemma *proper-subset*:$[\![A \subseteq B; x \notin A; x \in B]\!] \Longrightarrow A \neq B$
by *blast*

lemma *in-un2*:$[\![x \in A \cup B; x \notin A]\!] \Longrightarrow x \in B$
by *simp*

lemma *diff-disj*:$x \notin A \Longrightarrow A - \{x\} = A$
by *auto*

lemma *in-diff*:$[\![x \neq a; x \in A]\!] \Longrightarrow x \in A - \{a\}$
by *simp*

lemma *in-diff1*:$x \in A - \{a\} \Longrightarrow x \neq a$
by *simp*

lemma *sub-inserted1*:$[\![Y \subseteq insert\ a\ X; \neg\ Y \subseteq X]\!] \Longrightarrow a \notin X \wedge a \in Y$
by *blast*

lemma *sub-inserted2*:$[\![Y \subseteq insert\ a\ X; \neg\ Y \subseteq X]\!] \Longrightarrow Y = (Y - \{a\}) \cup \{a\}$
by *blast*

lemma *insert-sub*:$[\![A \subseteq B; a \in B]\!] \Longrightarrow (insert\ a\ A) \subseteq B$
by *blast*

lemma *insert-diff*:$A \subseteq (insert\ b\ B) \Longrightarrow A - \{b\} \subseteq B$
by *blast*

lemma *insert-inc1*:$A \subseteq insert\ a\ A$
by *blast*

lemma *insert-inc2*:$a \in insert\ a\ A$

by *simp*

lemma *nonempty-some*:$A \neq \{\} \Longrightarrow (SOME\ x.\ x \in A) \in A$
apply (*frule nonempty-ex*[*of A*])
apply (*rule someI2-ex*) **apply** *simp+*
done

lemma *mem-family-sub-Un*:$A \in C \Longrightarrow A \subseteq \bigcup C$
by *blast*

lemma *sub-Union*:$\exists X{\in}C.\ A \subseteq X \Longrightarrow A \subseteq \bigcup C$
by *blast*

lemma *family-subset-Un-sub*:$\forall A{\in}C.\ A \subseteq B \Longrightarrow \bigcup C \subseteq B$
by *blast*

lemma *in-set-with-P*:$P\ x \Longrightarrow x \in \{y.\ P\ y\}$
by *blast*

lemma *sub-single*:$[\![A \neq \{\};\ A \subseteq \{a\}]\!] \Longrightarrow A = \{a\}$
by *blast*

lemma *not-sub-single*:$[\![A \neq \{\};\ A \neq \{a\}]\!] \Longrightarrow \neg\ A \subseteq \{a\}$
by *blast*

lemma *not-sub*:$\neg\ A \subseteq B \Longrightarrow \exists\, a.\ a{\in}A \wedge a \notin B$
by *blast*

1.4 Functions

definition
cmp :: $['b \Rightarrow 'c,\ 'a \Rightarrow 'b] \Rightarrow ('a \Rightarrow 'c)$ **where**
cmp g f = $(\lambda x.\ g\ (f\ x))$

definition
idmap :: $'a\ set \Rightarrow ('a \Rightarrow 'a)$ **where**
idmap A = $(\lambda x{\in}A.\ x)$

definition
constmap :: $['a\ set,\ 'b\ set] \Rightarrow ('a \Rightarrow 'b)$ **where**
constmap A B = $(\lambda x{\in}A.\ SOME\ y.\ y \in B)$

definition
invfun :: $['a\ set,\ 'b\ set,\ 'a \Rightarrow 'b] \Rightarrow ('b \Rightarrow 'a)$ **where**
invfun A B $(f :: 'a \Rightarrow 'b)$ = $(\lambda y{\in}B.(SOME\ x.\ (x \in A \wedge f\ x = y)))$

abbreviation
INVFUN :: $['a \Rightarrow 'b,\ 'b\ set,\ 'a\ set] \Rightarrow ('b \Rightarrow 'a)$ $((3\text{-}^{-1}\text{-},\text{-})\ [82,82,83]82)$ **where**
$f^{-1}{}_{B,A}$ == *invfun A B f*

lemma *eq-fun*:$[\![f \in A \to B; f = g]\!] \Longrightarrow g \in A \to B$
by *simp*

lemma *eq-fun-eq-val*: $f = g \Longrightarrow f\ x = g\ x$
by *simp*

lemma *eq-elems-eq-val*:$x = y \Longrightarrow f\ x = f\ y$
by *simp*

lemma *cmp-fun*:$[\![f \in A \to B; g \in B \to C]\!] \Longrightarrow cmp\ g\ f \in A \to C$
by (*auto simp add*:*cmp-def*)

lemma *cmp-fun-image*:$[\![f \in A \to B; g \in B \to C]\!] \Longrightarrow$
$(cmp\ g\ f)\ `\ A = g\ `\ (f\ `\ A)$
apply (*rule equalityI*)
apply (*rule subsetI*, *simp add*:*image-def*)
apply (*erule bexE*, *simp add*:*cmp-def*, *blast*)
apply (*rule subsetI*, *simp add*:*image-def*[*of g*])
apply (*erule bexE*, *simp*)
apply (*simp add*:*image-def cmp-def*)
apply *blast*
done

lemma *cmp-fun-sub-image*:$[\![f \in A \to B; g \in B \to C; A1 \subseteq A]\!] \Longrightarrow$
$(cmp\ g\ f)\ `\ A1 = g\ `\ (f\ `\ A1)$
apply (*rule equalityI*)
apply (*rule subsetI*, *simp add*:*image-def*)
apply (*erule bexE*, *simp add*:*cmp-def*, *blast*)
apply (*rule subsetI*, *simp add*:*image-def*[*of g*])
apply (*erule bexE*, *simp*)
apply (*simp add*:*image-def cmp-def*)
apply *blast*
done

lemma *restrict-fun-eq*:$\forall x{\in}A.\ f\ x = g\ x \Longrightarrow (\lambda x{\in}A.\ f\ x) = (\lambda x{\in}A.\ g\ x)$
apply (*simp add*:*fun-eq-iff*)
done

lemma *funcset-mem*: $[\![f \in A \to B; x \in A]\!] \Longrightarrow f\ x \in B$
apply (*simp add*: *Pi-def*)
done

lemma *img-subset*:$f \in A \to B \Longrightarrow f\ `\ A \subseteq B$
apply (*rule subsetI*)
apply (*simp add*:*image-def*, *erule bexE*, *simp*)
apply (*simp add*:*funcset-mem*)
done

lemma *funcset-mem1*:⟦∀ *l*∈*A*. *f l* ∈ *B*; *x* ∈ *A*⟧ ⟹ *f x* ∈ *B*
apply *simp*
done

lemma *func-to-img*:*f* ∈ *A* → *B* ⟹ *f* ∈ *A* → *f* ‘ *A*
by (*simp add*:*Pi-def*)

lemma *restrict-in-funcset*: ∀ *x*∈ *A*. *f x* ∈ *B* ⟹
(λ*x*∈*A*. *f x*)∈ *A* → *B*
apply (*simp add*:*Pi-def restrict-def*)
done

lemma *funcset-eq*:⟦ *f* ∈ *extensional A*; *g* ∈ *extensional A*; ∀ *x*∈*A*. *f x* = *g x* ⟧ ⟹
f = *g*
apply (*simp add*:*extensionalityI*)
done

lemma *eq-funcs*:⟦*f* ∈ *A* → *B*; *g* ∈ *A* → *B*; *f* = *g*; *x* ∈ *A*⟧ ⟹ *f x* = *g x*
by *simp*

lemma *restriction-of-domain*:⟦ *f* ∈ *A* → *B*; *A1* ⊆ *A* ⟧ ⟹
restrict f A1 ∈ *A1* → *B*
by *blast*

lemma *restrict-restrict*:⟦ *restrict f A* ∈ *A* → *B*; *A1* ⊆ *A* ⟧ ⟹
restrict (*restrict f A*) *A1* = *restrict f A1*
apply (*rule funcset-eq*[*of* - *A1*])
apply (*simp add*:*restrict-def extensional-def*)
apply (*simp add*:*restrict-def extensional-def*)
apply (*rule ballI*) **apply** *simp*
apply (*simp add*:*subsetD*)
done

lemma *restr-restr-eq*:⟦ *restrict f A* ∈ *A* → *B*; *restrict f A* = *restrict g A*;
A1 ⊆ *A* ⟧ ⟹ *restrict f A1* = *restrict g A1*
apply (*subst restrict-restrict*[*THEN sym*, *of f A B A1*], *assumption*+)
apply (*simp add*:*restrict-restrict*[*THEN sym*, *of g A B A1*])
done

lemma *funcTr*:⟦ *f* ∈ *A* → *B*; *g* ∈ *A* → *B*; *f* = *g*; *a* ∈ *A*⟧ ⟹ *f a* = *g a*
apply *simp*
done

lemma *funcTr1*:⟦*f* = *g*; *a* ∈ *A*⟧ ⟹ *f a* = *g a*
apply *simp*
done

lemma *restrictfun-im*:⟦ (*restrict f A*) ∈ *A* → *B*; *A1* ⊆ *A* ⟧ ⟹

$(restrict\ f\ A)\ `\ A1 = f\ `\ A1$
apply (*subgoal-tac* $\forall x{\in}A1.\ x \in A$)
apply (*simp add*:*image-def*)
apply (*rule ballI*) **apply** (*simp add*:*subsetD*)
done

lemma *mem-in-image*:$[\![f \in A \to B;\ a \in A]\!] \Longrightarrow f\ a \in f\ `\ A$
apply (*simp add*:*image-def*)
apply *blast*
done

lemma *mem-in-image1*:$[\![\forall l{\in}A.\ f\ l \in B;\ a \in A]\!] \Longrightarrow f\ a \in f\ `\ A$
apply *simp*
done

lemma *mem-in-image2*:$a \in A \Longrightarrow f\ a \in f\ `\ A$
apply *simp*
done

lemma *mem-in-image3*:$b \in f\ `\ A \Longrightarrow \exists\, a \in A.\ b = f\ a$
by (*simp add*:*image-def*)

lemma *elem-in-image2*: $[\![f \in A \to B;\ A1 \subseteq A;\ x \in A1]\!] \Longrightarrow f\ x \in f`\ A1$
apply (*simp add*:*image-def*)
apply *blast*
done

lemma *funcs-nonempty*:$[\![A \neq \{\};\ B \neq \{\}\ ]\!] \Longrightarrow (A \to B) \neq \{\}$
apply (*subgoal-tac constmap A B* $\in A \to B$) **apply** (*simp add*:*nonempty*)
apply (*simp add*:*Pi-def*)
apply (*rule allI*) **apply** (*rule impI*)
apply (*simp add*:*constmap-def*)
apply (*frule nonempty-ex*[*of B*])
apply (*rule someI2-ex*) **apply** *assumption*+
done

lemma *idmap-funcs*: *idmap A* $\in A \to A$
apply (*simp add*:*Pi-def restrict-def idmap-def*)
done

lemma *l-idmap-comp*: $[\![f \in extensional\ A;\ f \in A \to B]\!] \Longrightarrow$
compose A (*idmap B*) *f* = *f*
apply (*rule funcset-eq*[*of - A*])
apply (*simp add*:*compose-def*)
apply *assumption*
apply (*rule ballI*)
apply (*simp add*:*funcset-mem*[*of f A B*] *compose-def idmap-def*)
done

lemma *r-idmap-comp*:⟦$f \in$ *extensional* A; $f \in A \to B$⟧ ⟹
compose A f (*idmap* A) = f
apply (*rule funcset-eq*[*of* - A])
apply (*simp add*:*compose-def*)
apply *assumption*
apply (*rule ballI*)
apply (*simp add*:*funcset-mem*[*of* f A B] *compose-def idmap-def*)
done

lemma *extend-fun*: ⟦ $f \in A \to B$; $B \subseteq B1$ ⟧ ⟹ $f \in A \to B1$
apply (*simp add*:*Pi-def restrict-def*)
apply (*rule allI*) **apply** (*rule impI*)
apply (*simp add*:*subsetD*)
done

lemma *restrict-fun*: ⟦ $f \in A \to B$; $A1 \subseteq A$ ⟧ ⟹ *restrict* f $A1 \in A1 \to B$
apply (*simp add*:*Pi-def restrict-def*)
apply (*rule allI*) **apply** (*rule impI*)
apply (*simp add*:*subsetD*)
done

lemma *set-of-hom*: ∀ $x \in A$. $f\ x \in B$ ⟹ *restrict* f $A \in A \to B$
apply (*simp add*:*Pi-def restrict-def*)
done

lemma *composition* : ⟦ $f \in A \to B$; $g \in B \to C$⟧ ⟹ (*compose* A g f) $\in A \to C$
apply (*simp add*:*Pi-def restrict-def compose-def*)
done

lemma *comp-assoc*:⟦$f \in A \to B$; $g \in B \to C$; $h \in C \to D$ ⟧ ⟹
compose A h (*compose* A g f) = *compose* A (*compose* B h g) f
apply (*rule funcset-eq*[*of* - A])
apply (*simp add*:*compose-def*)
apply (*simp add*:*compose-def*)
apply (*rule ballI*)
apply (*simp add*:*funcset-mem*[*of* f A B] *compose-def*)
done

lemma *restrictfun-inj*: ⟦ *inj-on* f A; $A1 \subseteq A$ ⟧ ⟹ *inj-on* (*restrict* f $A1$) $A1$
apply (*simp add*:*inj-on-def*)
apply (*simp add*:*subsetD*)
done

lemma *restrict-inj*:⟦*inj-on* f A; $A1 \subseteq A$⟧ ⟹ *inj-on* f $A1$
apply (*simp add*:*inj-on-def*)
apply ((*rule ballI*)+, *rule impI*)
apply (*frule-tac* $c = x$ **in** *subsetD*[*of* $A1$ A], *assumption*+,

frule-tac $c = y$ **in** *subsetD*[*of A1 A*], *assumption*+)
apply *simp*
done

lemma *injective*:⟦ *inj-on f A*; $x \in A$; $y \in A$; $x \neq y$ ⟧ $\Longrightarrow f\ x \neq f\ y$
apply (*rule contrapos-pp*, *simp*+)
apply (*simp add*:*inj-on-def*)
done

lemma *injective-iff*:⟦ *inj-on f A*; $x \in A$; $y \in A$⟧ $\Longrightarrow$
$(x = y) = (f\ x = f\ y)$
apply (*rule iffI*, *simp*)
apply (*rule contrapos-pp*, *simp*+)
apply (*frule injective*[*of f A x y*], *assumption*+)
apply *simp*
done

lemma *injfun-elim-image*:⟦$f \in A \to B$; *inj-on f A*; $x \in A$⟧ $\Longrightarrow$
$f\ `\ (A - \{x\}) = (f\ `\ A) - \{f\ x\}$
apply (*rule equalityI*)
apply (*rule subsetI*, *simp add*:*image-def*, *erule bexE*)
apply (*simp*, (*erule conjE*)+)
apply (*rule contrapos-pp*, *simp*+)
apply (*erule disjE*, *simp add*:*inj-on-def*, *blast*)
apply (*frule-tac* $x = xaa$ **and** $y = x$ **in** *injective*[*of f A*], *assumption*+,
blast)

apply (*rule subsetI*, *simp add*:*image-def*)
apply (*rule contrapos-pp*, *simp*+, *erule conjE*, *erule bexE*)
apply (*frule-tac* $x = xaa$ **in** *bspec*)
apply (*simp*, *rule contrapos-pp*, *simp*+)
done

lemma *cmp-inj*:⟦$f \in A \to B$; $g \in B \to C$; *inj-on f A*; *inj-on g B* ⟧ $\Longrightarrow$
inj-on (*cmp g f*) *A*
apply (*simp add*:*inj-on-def* [*of cmp g f*])
apply (*rule ballI*)+
apply (*simp add*:*cmp-def*) **apply** (*rule impI*)
apply (*subgoal-tac f x* = *f y*)
apply (*simp add*:*inj-on-def* [*of f*])
apply (*frule-tac* $x = x$ **in** *funcset-mem* [*of f A B*], *assumption*+)
apply (*frule-tac* $x = y$ **in** *funcset-mem* [*of f A B*], *assumption*+)
apply (*simp add*:*inj-on-def* [*of g*])
done

lemma *cmp-assoc*:⟦$f \in A \to B$; $g \in B \to C$; $h \in C \to D$; $x \in A$⟧ $\Longrightarrow$
$(cmp\ h\ (cmp\ g\ f))\ x\ = (cmp\ (cmp\ h\ g)\ f)\ x$
apply (*simp add*:*cmp-def*)
done

lemma *bivar-fun*: $[\![f \in A \to (B \to C);\ a \in A]\!] \Longrightarrow f\ a \in B \to C$
by (*simp add*:*Pi-def*)

lemma *bivar-fun-mem*: $[\![f \in A \to (B \to C);\ a \in A;\ b \in B]\!] \Longrightarrow f\ a\ b \in C$
apply (*frule funcset-mem*[*of f A B* $\to$ *C*], *assumption*+)
apply (*rule funcset-mem*[*of f a B C*], *assumption*+)
done

lemma *bivar-func-eq*:$[\![\forall\, a{\in}A.\ \forall\, b{\in}B.\ f\ a\ b = g\ a\ b]\!] \Longrightarrow$
$(\lambda x{\in}A.\ \lambda y{\in}B.\ f\ x\ y) = (\lambda x{\in}A.\ \lambda y{\in}B.\ g\ x\ y)$
apply (*subgoal-tac* $\forall\, x{\in}A.\ (\lambda y{\in}B.\ f\ x\ y) = (\lambda y{\in}B.\ g\ x\ y)$)
apply (*rule funcset-eq* [*of* - *A*])
apply (*simp add*:*extensional-def restrict-def*)
apply (*simp add*:*extensional-def restrict-def*)
apply (*rule ballI*)
apply *simp*
apply (*rule ballI*)
apply (*rule funcset-eq* [*of* - *B*])
apply (*simp add*:*restrict-def extensional-def*)
apply (*simp add*:*restrict-def extensional-def*)
apply (*rule ballI*) **apply** *simp*
done

lemma *set-image*: $[\![f \in A \to B;\ A1 \subseteq A;\ A2 \subseteq A]\!] \Longrightarrow$
$f`(A1 \cap A2) \subseteq (f` A1) \cap (f` A2)$
apply (*simp add*: *image-def*)
apply *auto*
done

lemma *image-sub*: $[\![f \in A \to B;\ A1 \subseteq A]\!] \Longrightarrow (f`A1) \subseteq B$
by (*auto simp add*:*image-def*)

lemma *image-sub0*: $f \in A \to B \Longrightarrow (f`A) \subseteq B$
by (*simp add*:*image-sub*[*of f A B A*])

lemma *image-nonempty*:$[\![f \in A \to B;\ A1 \subseteq A;\ A1 \neq \{\}]\!] \Longrightarrow f\ `\ A1 \neq \{\}$
by (*frule nonempty-some*[*of A1*],
frule elem-in-image2[*of f A B A1 SOME x*. $x \in A1$],
assumption+, *simp add*:*nonempty*)

lemma *im-set-mono*: $[\![f \in A \to B;\ A1 \subseteq A2;\ A2 \subseteq A]\!] \Longrightarrow (f\ `\ A1) \subseteq (f\ `\ A2)$
apply (*simp add*:*image-def*)
apply *auto*
done

lemma *im-set-un*:$[\![f{\in}A \to B;\ A1 \subseteq A;\ A2 \subseteq A]\!] \Longrightarrow$
$f`(A1 \cup A2) = (f`A1) \cup (f`A2)$
apply (*simp add*:*image-def*)

apply *auto*
done

lemma *im-set-un1*:⟦∀ *l*∈*A*. *f l* ∈ *B*; *A* = *A1* ∪ *A2*⟧ ⟹
f ‘(*A1* ∪ *A2*) = *f* ‘(*A1*) ∪ *f* ‘(*A2*)
apply (*rule equalityI*,
rule subsetI, *simp add*:*image-def*, *erule bexE*)
apply *blast*
apply (*rule subsetI*,
simp add:*image-def*, *erule disjE*, *erule bexE*, *blast*)
apply (*erule bexE*) **apply** *blast*
done

lemma *im-set-un2*:*A* = *A1* ∪ *A2* ⟹ *f* ‘*A* = *f* ‘(*A1*) ∪ *f* ‘(*A2*)
apply (*rule equalityI*,
rule subsetI, *simp add*:*image-def*, *erule bexE*, *blast*)
apply (*rule subsetI*,
simp add:*image-def*, *erule disjE*, *erule bexE*, *blast*, *erule bexE*, *blast*)
done

definition
invim::[*'a* ⇒ *'b*, *'a set*, *'b set*] ⇒ *'a set* **where**
invim f A B = {*x*. *x*∈*A* ∧ *f x* ∈ *B*}

lemma *invim*: ⟦ *f*:*A* → *B*; *B1* ⊆ *B* ⟧ ⟹ *invim f A B1* ⊆ *A*
apply (*simp add*:*invim-def*)
apply *auto*
done

lemma *setim-cmpfn*: ⟦ *f*:*A* → *B*; *g*:*B* → *C*; *A1* ⊆ *A* ⟧ ⟹
(*compose A g f*)‘ *A1* = *g*‘(*f*‘ *A1*)
apply (*simp add*:*image-def compose-def*)
apply *auto*
done

definition
surj-to :: [*'a* ⇒ *'b*, *'a set*, *'b set*] ⇒ *bool* **where**
surj-to f A B ⟷ *f*‘*A* = *B*

lemma *surj-to-test*:⟦ *f* ∈ *A* → *B*; ∀ *b*∈*B*. ∃ *a*∈*A*. *f a* = *b* ⟧ ⟹
surj-to f A B
by (*auto simp add*:*surj-to-def image-def*)

lemma *surj-to-image*:*f* ∈ *A* → *B* ⟹ *surj-to f A* (*f* ‘ *A*)
apply (*rule surj-to-test*[*of f A f* ‘ *A*])
apply (*rule func-to-img*[*of f A B*], *assumption*)
apply (*rule ballI*, *simp add*:*image-def*, *erule bexE*, *simp*)
apply *blast*
done

lemma *surj-to-el*:⟦ $f \in A \to B$; *surj-to f A B* ⟧ $\Longrightarrow \forall\, b{\in}B.\ \exists\, a{\in}A.\ f\ a = b$
apply (*simp add*:*surj-to-def image-def*)
apply *auto*
done

lemma *surj-to-el1*:⟦ $f \in A \to B$; *surj-to f A B*; $b{\in}B$⟧ $\Longrightarrow \exists\, a{\in}A.\ f\ a = b$
apply (*simp add*:*surj-to-el*)
done

lemma *surj-to-el2*:⟦*surj-to f A B*; $b \in B$⟧ $\Longrightarrow \exists\, a{\in}A.\ f\ a = b$
apply (*simp add*:*surj-to-def image-def*)
apply (*frule sym*, *thin-tac* $\{y.\ \exists\, x{\in}A.\ y = f\ x\} = B$, *simp*)
apply (*erule bexE*, *simp*, *blast*)
done

lemma *compose-surj*: ⟦*f*:$A \to B$; *surj-to f A B*; $g : B \to C$; *surj-to g B C* ⟧
$\Longrightarrow$ *surj-to* (*compose A g f*) *A C*
apply (*simp add*:*surj-to-def compose-def image-def*)
apply *auto*
done

lemma *cmp-surj*: ⟦*f*:$A \to B$; *surj-to f A B*; $g : B \to C$; *surj-to g B C* ⟧
$\Longrightarrow$ *surj-to* (*cmp g f*) *A C*
apply (*rule surj-to-test*, *simp add*:*cmp-fun*)
apply (*rule ballI*, *simp add*:*surj-to-def* [*of g*], *frule sym*,
thin-tac g ‘ B = C, *simp*, *simp add*:*image-def*,
simp add:*cmp-def*)
apply *auto*
apply (*simp add*:*surj-to-def*, *frule sym*,
thin-tac f ‘ A = B, *simp add*:*image-def*)
apply *auto*
done

lemma *inj-onTr0*:⟦ $f \in A \to B$; $x \in A$; $y \in A$; *inj-on f A*; $f\ x = f\ y$⟧ $\Longrightarrow x = y$
apply (*simp add*:*inj-on-def*)
done

lemma *inj-onTr1*:⟦*inj-on f A*; $x \in A$; $y \in A$; $f\ x = f\ y$⟧ $\Longrightarrow x = y$
apply (*simp add*:*inj-on-def*)
done

lemma *inj-onTr2*:⟦*inj-on f A*; $x \in A$; $y \in A$; $f\ x \neq f\ y$⟧ $\Longrightarrow x \neq y$
apply (*rule contrapos-pp*, *simp*+)
done

lemma *comp-inj*: ⟦ $f \in A \to B$; *inj-on f A*; $g \in B \to C$; *inj-on g B* ⟧
$\Longrightarrow$ *inj-on* (*compose A g f*) *A*

apply (*simp add*:*inj-on-def* [*of compose A g f*])
apply (*rule ballI*)+ **apply** (*rule impI*)
apply (*rule inj-onTr0* [*of f A B*], *assumption*+)
apply (*frule funcset-mem* [*of f A B* -], *assumption*+)
apply (*rotate-tac* −*3*)
apply (*frule funcset-mem* [*of f A B* -], *assumption*+)
apply (*rule inj-onTr0* [*of g B C* -], *assumption*+)
apply (*simp add*:*compose-def*)
done

lemma *cmp-inj-1*: ⟦ $f \in A \to B$; *inj-on f A*; $g \in B \to C$; *inj-on g B* ⟧
⟹ *inj-on* (*cmp g f*) *A*
apply (*simp add*:*inj-on-def* [*of cmp g f*])
apply (*rule ballI*)+ **apply** (*rule impI*)
apply (*simp add*:*cmp-def*)
apply (*frule-tac* $x = x$ **in** *funcset-mem* [*of f A B*], *assumption*+)
apply (*frule-tac* $x = y$ **in** *funcset-mem* [*of f A B*], *assumption*+)
apply (*frule-tac* $x = f\ x$ **and** $y = f\ y$ **in** *inj-onTr1* [*of g B*],
assumption+)
apply (*rule-tac* $x = x$ **and** $y = y$ **in** *inj-onTr1* [*of f A*], *assumption*+)
done

lemma *cmp-inj-2*: ⟦∀ $l \in A$. $f\ l \in B$; *inj-on f A*; ∀ $k \in B$. $g\ k \in C$; *inj-on g B* ⟧
⟹ *inj-on* (*cmp g f*) *A*
apply (*simp add*:*inj-on-def* [*of cmp g f*])
apply (*rule ballI*)+ **apply** (*rule impI*)
apply (*simp add*:*cmp-def*)
apply (*frule-tac* $x = x$ **in** *funcset-mem1* [*of A f B*], *assumption*+)
apply (*frule-tac* $x = y$ **in** *funcset-mem1* [*of A f B*], *assumption*+)
apply (*frule-tac* $x = f\ x$ **and** $y = f\ y$ **in** *inj-onTr1* [*of g B*],
assumption+)
apply (*rule-tac* $x = x$ **and** $y = y$ **in** *inj-onTr1* [*of f A*], *assumption*+)
done

lemma *invfun-mem*:⟦ $f \in A \to B$; *inj-on f A*; *surj-to f A B*; $b \in B$ ⟧
⟹ (*invfun A B f*) $b \in A$
apply (*simp add*:*invfun-def*)
apply (*simp add*:*surj-to-def image-def*) **apply** (*frule sym*)
apply (*thin-tac* $\{y.\ \exists x \in A.\ y = f\ x\} = B$) **apply** *simp*
apply (*thin-tac* $B = \{y.\ \exists x \in A.\ y = f\ x\}$) **apply** *auto*
apply (*rule someI2-ex*)
apply *blast* **apply** *simp*
done

lemma *inv-func*:⟦ $f \in A \to B$; *inj-on f A*; *surj-to f A B*⟧
⟹ (*invfun A B f*) $\in B \to A$
apply (*simp add*:*Pi-def*)
apply (*rule allI*) **apply** (*rule impI*)

apply (*rule invfun-mem*) **apply** (*rule funcsetI*)
apply *simp+*
done

lemma *invfun-r*:⟦ *f*∈*A* → *B*; *inj-on f A*; *surj-to f A B*; *b* ∈ *B* ⟧
⟹ *f* ((*invfun A B f*) *b*) = *b*
apply (*simp add*:*invfun-def*)
apply (*rule someI2-ex*)
apply (*simp add*:*surj-to-def image-def*)
apply *auto*
done

lemma *invfun-l*:⟦*f* ∈ *A* → *B*; *inj-on f A*; *surj-to f A B*; *a* ∈ *A*⟧
⟹ (*invfun A B f*) (*f a*) = *a*
apply (*simp add*:*invfun-def Pi-def restrict-def*)
apply (*rule someI2-ex*) **apply** *auto*
apply (*simp add*:*inj-on-def*)
done

lemma *invfun-inj*:⟦*f* ∈ *A* → *B*; *inj-on f A*; *surj-to f A B*⟧
⟹ *inj-on* (*invfun A B f*) *B*
apply (*simp add*:*inj-on-def* [*of invfun A B f B*])
apply *auto*
apply (*frule-tac b* = *y* **in** *invfun-r* [*of f A B*], *assumption+*)
apply (*frule-tac b* = *x* **in** *invfun-r* [*of f A B*], *assumption+*)
apply *simp*
done

lemma *invfun-surj*:⟦*f* ∈ *A* → *B*; *inj-on f A*; *surj-to f A B*⟧
⟹ *surj-to* (*invfun A B f*) *B A*
apply (*simp add*:*surj-to-def* [*of invfun A B f B A*] *image-def*)
apply (*rule equalityI*)
apply (*rule subsetI*) **apply** (*simp add*:*CollectI*)
apply *auto*
apply (*simp add*:*invfun-mem*)
apply (*frule funcset-mem* [*of f A B*], *assumption+*)
apply (*frule-tac t* = *x* **in** *invfun-l* [*of f A B*, *THEN sym*], *assumption+*)
apply *auto*
done

definition
bij-to :: [*'a* ⇒ *'b*, *'a set*, *'b set*] ⇒ *bool* **where**
bij-to f A B ⟷ *surj-to f A B* ∧ *inj-on f A*

lemma *idmap-bij*:*bij-to* (*idmap A*) *A A*
apply (*simp add*:*bij-to-def*)
apply (*rule conjI*)
apply (*simp add*:*surj-to-def*, *simp add*:*image-def*, *simp add*:*idmap-def*)

apply (*simp add*:*inj-on-def*, *simp add*:*idmap-def*)
done

lemma *bij-invfun*:⟦*f* ∈ *A* → *B*; *bij-to f A B*⟧ ⟹
bij-to (*invfun A B f*) *B A*
apply (*simp add*:*bij-to-def*)
apply (*simp add*:*invfun-inj invfun-surj*)
done

lemma *l-inv-invfun*:⟦ *f* ∈ *A* → *B*; *inj-on f A*; *surj-to f A B*⟧
⟹ *compose A* (*invfun A B f*) *f* = *idmap A*
apply (*rule ext*)
apply (*simp add*:*compose-def idmap-def*)
apply (*rule impI*)
apply (*simp add*:*invfun-l*)
done

lemma *invfun-mem1*:⟦*f* ∈ *A* → *B*; *bij-to f A B*; *b* ∈ *B*⟧ ⟹
(*invfun A B f*) *b* ∈ *A*
apply (*simp add*:*bij-to-def*, *erule conjE*)
apply (*simp add*:*invfun-mem*)
done

lemma *invfun-r1*:⟦ *f*∈*A* → *B*; *bij-to f A B*; *b* ∈ *B* ⟧
⟹ *f* ((*invfun A B f*) *b*) = *b*
apply (*simp add*:*bij-to-def*, *erule conjE*)
apply (*rule invfun-r*, *assumption*+)
done

lemma *invfun-l1*:⟦*f* ∈ *A* → *B*; *bij-to f A B*; *a* ∈ *A*⟧
⟹ (*invfun A B f*) (*f a*) = *a*
apply (*simp add*:*bij-to-def*, *erule conjE*)
apply (*rule invfun-l*, *assumption*+)
done

lemma *compos-invfun-r*:⟦*f* ∈ *A* → *B*; *bij-to f A B*; *g* ∈ *A* → *C*; *h* ∈ *B* → *C*;
g ∈ *extensional A*; *compose B g* (*invfun A B f*) = *h*⟧ ⟹
g = *compose A h f*
apply (*rule funcset-eq*[*of g A compose A h f*], *assumption*)
apply (*simp add*:*compose-def extensional-def*)
apply (*rule ballI*)
apply (*frule sym*, *thin-tac compose B g* (*invfun A B f*) = *h*, *simp*)
apply (*simp add*:*compose-def Pi-def invfun-l1*)
done

lemma *compos-invfun-l*:⟦*f* ∈ *A* → *B*; *bij-to f A B*; *g* ∈ *C* → *B*; *h* ∈ *C* → *A*;
compose C (*invfun A B f*) *g* = *h*; *g* ∈ *extensional C* ⟧ ⟹
g = *compose C f h*
apply (*rule funcset-eq*[*of g C compose C f h*], *assumption*)

apply (*simp add*:*compose-def extensional-def*)
apply (*rule ballI*)
apply (*frule sym*, *thin-tac compose C* (*invfun A B f*) *g* = *h*,
simp add:*compose-def*)
apply (*frule-tac x* = *x* **in** *funcset-mem*[*of g C B*], *assumption*)
apply (*simp add*:*invfun-r1*)
done

lemma *invfun-set*:⟦*f* ∈ *A* → *B*; *bij-to f A B*; *C* ⊆ *B*⟧ ⟹
f ‘ ((*invfun A B f*)‘ *C*) = *C*
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*simp add*:*image-def*, *erule exE*,
erule conjE, *erule bexE*, *simp*,
frule-tac c = *xb* **in** *subsetD*[*of C B*], *assumption*+)
apply (*simp add*:*bij-to-def*, *erule conjE*,
simp add:*invfun-r*)

apply (*rule subsetI*, *simp add*:*image-def*)
apply (*frule-tac c* = *x* **in** *subsetD*[*of C B*], *assumption*+,
simp add:*bij-to-def*, *erule conjE*,
frule-tac b = *x* **in** *invfun-r*[*of f A B*], *assumption*+)
apply (*frule sym*, *thin-tac f* (*invfun A B f x*) = *x*)
apply *blast*
done

lemma *compos-bij*:⟦*f* ∈ *A* → *B*; *bij-to f A B*; *g* ∈ *B* → *C*; *bij-to g B C*⟧ ⟹
bij-to (*compose A g f*) *A C*
apply (*simp add*:*bij-to-def*, (*erule conjE*)+)
apply (*simp add*:*comp-inj*[*of f A B g C*])
apply (*simp add*:*compose-surj*)
done

1.5 Nsets

definition
nset :: [*nat*, *nat*] ⇒ (*nat*) *set* **where**
nset i j = {*k*. *i* ≤ *k* ∧ *k* ≤ *j*}

definition
slide :: *nat* ⇒ *nat* ⇒ *nat* **where**
slide i j == *i* + *j*

definition
sliden :: *nat* ⇒ *nat* ⇒ *nat* **where**
sliden i j == *j* − *i*

definition
jointfun :: [*nat*, *nat* ⇒ ′*a*, *nat*, *nat* ⇒ ′*a*] ⇒ (*nat* ⇒ ′*a*) **where**

$(jointfun\ n\ f\ m\ g) = (\lambda i.\ if\ i \leq n\ then\ f\ i\ else\ \ g\ ((sliden\ (Suc\ n))\ i))$

definition
$skip :: nat \Rightarrow (nat \Rightarrow nat)$ **where**
$skip\ i = (\lambda x.\ (if\ i = 0\ then\ Suc\ x\ else$
$\quad (if\ x \in \{j.\ j \leq (i - Suc\ 0)\}\ then\ x\ \ else\ Suc\ x)))$

lemma *nat-pos*:$0 \leq (l::nat)$
apply *simp*
done

lemma *Suc-pos*:$Suc\ k \leq r \Longrightarrow 0 < r$
apply *simp*
done

lemma *nat-pos2*:$(k::nat) < r \Longrightarrow 0 < r$
apply *simp*
done

lemma *eq-le-not*:$[\![(a::nat) \leq b; \neg\ a < b\ ]\!] \Longrightarrow a = b$
apply *auto*
done

lemma *im-of-constmap*:$(constmap\ \{0\}\ \{a\})\ `\ \{0\} = \{a\}$
apply $(simp\ add{:}constmap\text{-}def)$
done

lemma *noteq-le-less*:$[\![\ m \leq (n::nat);\ m \neq n\ ]\!] \Longrightarrow m < n$
apply *auto*
done

lemma *nat-not-le-less*:$(\neg\ (n::nat) \leq m) = (m < n)$
by $(simp\ add{:}\ not\text{-}le)$

lemma *self-le*:$(n::nat) \leq n$
apply *simp*
done

lemma *n-less-Suc*:$(n::nat) < Suc\ n$
apply *simp*
done

lemma *less-diff-pos*:$i < (n::nat) \Longrightarrow 0 < n - i$
apply *auto*
done

lemma *less-diff-Suc*:$i < (n::nat) \Longrightarrow n - (Suc\ i) = (n - i) - (Suc\ 0)$
apply *auto*
done

lemma *less-pre-n*:$0 < n \Longrightarrow n - Suc\ 0 < n$
apply *simp*
done

lemma *Nset-inc-0*:$(0::nat) \in \{i.\ i \leq n\}$
apply *simp*
done

lemma *Nset-1*:$\{i.\ i \leq Suc\ 0\} = \{0, Suc\ 0\}$
apply *auto*
done

lemma *Nset-1-1*:$(k \leq Suc\ 0) = (k = 0 \vee k = Suc\ 0)$
apply (*rule iffI*)
apply (*case-tac* $k = 0$, *simp*+)
apply (*erule disjE*, *simp*+)
done

lemma *Nset-2*:$\{i, j\} = \{j, i\}$
apply *auto*
done

lemma *Nset-nonempty*:$\{i.\ i \leq (n::nat)\} \neq \{\}$
apply (*subgoal-tac* $0 \in \{i.\ i \leq n\}$)
apply (*rule nonempty*[*of 0*], *assumption*)
apply *simp*
done

lemma *Nset-le*:$x \in \{i.\ i \leq n\} \Longrightarrow x \leq n$
apply *simp*
done

lemma *n-in-Nsetn*:$(n::nat) \in \{i.\ i \leq n\}$
apply *simp*
done

lemma *Nset-pre*:$[\![\ (x::nat) \in \{i.\ i \leq (Suc\ n)\};\ x \neq Suc\ n\ ]\!] \Longrightarrow x \in \{i.\ i \leq n\}$
apply *simp*
done

lemma *Nset-pre1*:$\{i.\ i \leq (Suc\ n)\} - \{Suc\ n\} = \{i.\ i \leq n\}$
apply (*rule equalityI*)
apply (*rule subsetI*, *simp*)+
done

lemma *le-Suc-mem-Nsetn*:$x \leq Suc\ n \Longrightarrow x - Suc\ 0 \in \{i.\ i \leq n\}$
apply (*frule diff-le-mono*[*of x Suc n Suc 0*],
thin-tac $x \leq Suc\ n$, *simp*)

done

lemma *le-Suc-diff-le*:$x \leq Suc\ n \Longrightarrow x - Suc\ 0 \leq n$
apply (*frule diff-le-mono*[*of x Suc n Suc 0*],
thin-tac $x \leq Suc\ n$, *simp*)
done

lemma *Nset-not-pre*:$[\![\ x \notin \{i.\ i \leq n\};\ x \in \{i.\ i \leq (Suc\ n)\}]\!] \Longrightarrow x = Suc\ n$
by *simp*

lemma *mem-of-Nset*:$x \leq (n::nat) \Longrightarrow x \in \{i.\ i \leq n\}$
apply *simp*
done

lemma *less-mem-of-Nset*:$x < (n::nat) \Longrightarrow x \in \{i.\ i \leq n\}$
apply (*frule less-imp-le* [*of x n*])
apply *simp*
done

lemma *Nset-nset*:$\{i.\ i \leq (Suc\ (n + m))\} = \{i.\ i \leq n\} \cup$
nset (*Suc n*) (*Suc* $(n + m)$)
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*simp add*:*nset-def*)
apply (*auto simp add*: *nset-def*)
done

lemma *Nset-nset-1*:$[\![0 < n;\ i < n]\!] \Longrightarrow \{j.\ j \leq n\} = \{j.\ j \leq i\} \cup$
nset (*Suc i*) *n*
apply *auto*
apply (*simp add*:*nset-def*)
apply (*simp add*:*nset-def*)
done

lemma *Nset-img0*:$[\![f \in \{j.\ j \leq Suc\ n\} \to B;\ (f\ (Suc\ n)) \in f\ `\ \{j.\ j \leq n\}]\!] \Longrightarrow$
$f\ `\ \{j.\ j \leq Suc\ n\} = f\ `\ \{j.\ j \leq n\}$
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*simp add*:*image-def*)
apply (*erule exE*, *erule conjE*)
apply (*erule exE*, *simp*, *erule conjE*)
apply (*case-tac xb* $= Suc\ n$, *simp*, *blast*)
apply (*frule-tac m* $= xb$ **and** $n = Suc\ n$ **in** *noteq-le-less*, *assumption*,
thin-tac xb $\leq Suc\ n$,
frule-tac x $= xb$ **and** $n = Suc\ n$ **in** *less-le-diff*,
thin-tac xb $< Suc\ n$, *simp*, *blast*)
apply (*rule subsetI*, *simp add*:*image-def*, (*erule exE*)+, (*erule conjE*)+)
apply (*simp*,
frule-tac x $= xb$ **and** $y = n$ **and** $z = Suc\ n$ **in** *le-less-trans*,

```
        simp,
        frule-tac x = xb and y = Suc n in less-imp-le,
         blast)
done

lemma Nset-img:f ∈ {j. j ≤ Suc n} → B ⟹
         insert (f (Suc n)) (f ‘ {j. j ≤ n}) = f ‘ {j. j ≤ Suc n}
apply (rule equalityI)
 apply (rule subsetI)
 apply (simp add:image-def,
        erule disjE, blast,
        erule exE, erule conjE,
        frule-tac x = xa and y = n and z = Suc n in le-less-trans,
        simp+,
        frule-tac x = xa and y = Suc n in less-imp-le, blast)
 apply (rule subsetI,
        simp add:image-def,
        erule exE, simp, erule conjE,
        case-tac xa = Suc n, simp)
 apply (frule-tac m = xa and n = Suc n in noteq-le-less, assumption,
        frule-tac x = xa and n = Suc n in less-le-diff,
        thin-tac xa ≤ Suc n, thin-tac xa < Suc n, simp)
 apply blast
done

primrec nasc-seq :: [nat set, nat, nat] ⇒ nat
where
  dec-seq-0: nasc-seq A a 0 = a
| dec-seq-Suc: nasc-seq A a (Suc n) =
                    (SOME b. ((b ∈ A) ∧ (nasc-seq A a n) < b))

lemma nasc-seq-mem:⟦(a::nat) ∈ A; ¬ (∃ m. m∈A ∧ (∀ x∈A. x ≤ m))⟧ ⟹
                      (nasc-seq A a n) ∈ A
apply (induct n)
apply (simp-all add: not-le)
apply (subgoal-tac ∃ x∈A. (nasc-seq A a n) < x) prefer 2 apply blast
 apply (thin-tac ∀ m. m ∈ A ⟶ (∃ x∈A. m < x),
        rule someI2-ex, blast, simp)
done

lemma nasc-seqn:⟦(a::nat) ∈ A; ¬ (∃ m. m∈A ∧ (∀ x∈A. x ≤ m))⟧ ⟹
                               (nasc-seq A a n) < (nasc-seq A a (Suc n))
apply (simp,
       frule nasc-seq-mem [of a A n], simp)
apply (simp add: not-le,
       subgoal-tac ∃ x∈A. (nasc-seq A a n) < x) prefer 2 apply simp
 apply (thin-tac ∀ m. m ∈ A ⟶ (∃ x∈A. m < x),
        rule someI2-ex, blast, simp)
done
```

lemma *nasc-seqn1*:⟦(*a*::*nat*) ∈ *A*; ¬ (∃ *m*. *m*∈*A* ∧ (∀ *x*∈*A*. *x* ≤ *m*))⟧ ⟹
Suc (*nasc-seq A a n*) ≤ (*nasc-seq A a* (*Suc n*))
apply (*frule nasc-seqn* [*of a A n*], *assumption*+)
apply *simp*
done

lemma *ubs-ex-n-maxTr*:⟦(*a*::*nat*) ∈ *A*; ¬ (∃ *m*. *m*∈*A* ∧ (∀ *x*∈*A*. *x* ≤ *m*))⟧
⟹ (*a* + *n*) ≤ (*nasc-seq A a n*)
apply (*induct-tac n*)
apply *simp*
apply (*frule-tac n* = *n* **in** *nasc-seqn1*[*of a A*], *assumption*+,
subgoal-tac Suc (*a* + *n*) ≤ *Suc* (*nasc-seq A a n*),
frule-tac i = *Suc* (*a* + *n*) **and** *j* = *Suc* (*nasc-seq A a n*) **and**
k = *nasc-seq A a* (*Suc n*) **in** *le-trans*, *assumption*+,
simp, *thin-tac Suc* (*nasc-seq A a n*) ≤ *nasc-seq A a* (*Suc n*),
subst Suc-le-mono, *assumption*+)
done

lemma *ubs-ex-n-max*:⟦*A* ≠ {}; *A* ⊆ {*i*. *i* ≤ (*n*::*nat*)}⟧ ⟹
∃!*m*. *m*∈*A* ∧ (∀ *x*∈*A*. *x* ≤ *m*)
apply (*frule nonempty-ex*[*of A*])
apply (*thin-tac A* ≠ {})
apply (*erule exE*)
apply (*rename-tac a*)
apply (*rule ex-ex1I*)
prefer *2*
apply (*erule conjE*)+
apply (*frule-tac x* = *y* **in** *bspec*, *assumption*+,
thin-tac ∀ *x*∈*A*. *x* ≤ *m*,
frule-tac x = *m* **in** *bspec*, *assumption*+,
thin-tac ∀ *x*∈*A*. *x* ≤ *y*, *simp*)

apply (*rule contrapos-pp*, *simp*+)

apply (*frule-tac a* = *a* **and** *A* = *A* **and** *n* = *n* + *1* **in** *ubs-ex-n-maxTr*, *simp*)
apply (*frule-tac a* = *a* **in** *nasc-seq-mem*[*of - A n* + *1*], *simp*)
apply (*frule-tac c* = *nasc-seq A a* (*n* + *1*) **in** *subsetD*[*of A* {*i*. *i* ≤ *n*}],
assumption+, *simp*)
done

definition
n-max :: *nat set* ⇒ *nat* **where**
n-max A = (*THE m*. *m* ∈ *A* ∧ (∀ *x*∈*A*. *x* ≤ *m*))

lemma *n-max*:⟦*A* ⊆ {*i*. *i* ≤ (*n*::*nat*)}; *A* ≠ {}⟧ ⟹
(*n-max A*) ∈ *A* ∧ (∀ *x*∈*A*. *x* ≤ (*n-max A*))
apply (*simp add*:*n-max-def*)
apply (*frule ubs-ex-n-max*[*of A n*], *assumption*)

apply (*rule theI'*)
apply *assumption*
done

lemma *n-max-eq-sets*:⟦$A = B$; $A \neq \{\}$; $\exists n.\ A \subseteq \{j.\ j \leq n\}$⟧ ⟹
n-max A = *n-max* B
by *simp*

lemma *skip-mem*:$l \in \{i.\ i \leq n\} \Longrightarrow (skip\ i\ l) \in \{i.\ i \leq (Suc\ n)\}$
apply (*case-tac* $i = 0$)
apply (*simp add*:*skip-def*)
apply (*simp*)+
apply (*simp add*:*skip-def*)
done

lemma *skip-fun*:$(skip\ i) \in \{i.\ i \leq n\} \to \{i.\ i \leq (Suc\ n)\}$
apply (*rule Pi-I*)
apply (*rule skip-mem*, *assumption*)
done

lemma *skip-im-Tr0*:$x \in \{i.\ i \leq n\} \Longrightarrow skip\ 0\ x = Suc\ x$
apply (*simp add*:*skip-def*)
done

lemma *skip-im-Tr0-1*:$0 < y \Longrightarrow skip\ 0\ (y - Suc\ 0) = y$
apply (*simp add*:*skip-def*)
done

lemma *skip-im-Tr1*:⟦ $i \in \{i.\ i \leq (Suc\ n)\}$; $0 < i$; $x \leq i - Suc\ 0$ ⟧ ⟹
$skip\ i\ x = x$
by (*simp add*:*skip-def*)

lemma *skip-im-Tr1-1*:⟦ $i \in \{i.\ i \leq (Suc\ n)\}$; $0 < i$; $x < i$⟧ ⟹
$skip\ i\ x = x$
apply (*frule less-le-diff*[*of x i*])
apply (*simp add*:*skip-def*)
done

lemma *skip-im-Tr1-2*:⟦ $i \leq (Suc\ n)$; $x < i$⟧ ⟹ $skip\ i\ x = x$
apply (*rule skip-im-Tr1-1*[*of i n x*], *simp*+)
done

lemma *skip-im-Tr2*:⟦ $0 < i$; $i \in \{i.\ i \leq (Suc\ n)\}$; $i \leq x$⟧ ⟹
$skip\ i\ x = Suc\ x$
by (*simp add*:*skip-def*)

lemma *skip-im-Tr2-1*:⟦$i \in \{i.\ i \leq (Suc\ n)\}$; $i \leq x$⟧ ⟹
$skip\ i\ x = Suc\ x$

apply (*case-tac* $i = 0$)
  apply (*simp add*:*skip-def*)
apply (*simp*, *rule skip-im-Tr2*, *assumption*+, *simp*+)
done

lemma *skip-im-Tr3*:$x \in \{i.\ i \leq n\} \Longrightarrow$ *skip* (*Suc n*) $x = x$
apply (*simp add*:*skip-def*)
done

lemma *skip-im-Tr4*:$[\![x \leq$ *Suc n*; $0 < x]\!] \Longrightarrow x -$ *Suc 0* $\leq n$
 apply (*simp add*:*Suc-le-mono* [*of* $x -$ *Suc 0 n*, *THEN sym*])
done

lemma *skip-fun-im*:$i \in \{j.\ j \leq$ (*Suc n*)$\} \Longrightarrow$
        (*skip i*) ‘ $\{j.\ j \leq n\} = (\{j.\ j \leq$ (*Suc n*)$\} - \{i\})$
apply (*rule equalityI*)
 apply (*rule subsetI*)
 apply (*case-tac* $i = 0$, *simp*)
 apply (*simp add*:*image-def*, *erule exE*, *erule conjE*)
 apply (*cut-tac* $x =$ *xa* **in** *skip-im-Tr0*[*of* - *n*], *simp*, *simp*)

 apply (*simp add*:*image-def*, *erule exE*, *erule conjE*, *simp*)
 apply (*case-tac xa* $< i$)
 apply (*frule-tac* $x =$ *xa* **in** *skip-im-Tr1-2*[*of i n*], *simp*+)
 apply (*cut-tac m1* $=$ *xa* **and** *n1* $= i$ **in** *nat-not-le-less*[*THEN sym*], *simp*)
 apply (*cut-tac* $x =$ *xa* **and** $n = n$ **in** *skip-im-Tr2-1*[*of i*], *simp*+)

apply (*rule subsetI*, *simp*, *erule conjE*)
 apply (*cut-tac* $x = x$ **and** $y = i$ **in** *less-linear*, *simp*)
 apply (*erule disjE*)
 apply (*simp add*:*image-def*)
 apply (*frule-tac* $x = x$ **in** *skip-im-Tr1-2*[*of i n*], *assumption*,
       frule-tac $x = x$ **and** $y = i$ **and** $z =$ *Suc n* **in** *less-le-trans*,
       assumption+,
       frule-tac $m = x$ **and** $n =$ *Suc n* **in** *Suc-leI*,
       simp only:*Suc-le-mono*,
       frule sym, *thin-tac skip i* $x = x$, *blast*)
 apply (*cut-tac* $x = x -$ *Suc 0* **in** *skip-im-Tr2-1*[*of i n*],
       simp, *simp add*:*less-le-diff*)
 apply (*cut-tac* $x = i$ **and** $n = x$ **in** *less-le-diff*, *assumption*,
       simp add:*image-def*)
 apply (*frule-tac* $m = x$ **and** $n =$ *Suc n* **and** $l =$ *Suc 0* **in** *diff-le-mono*,
       simp)
 apply (*frule sym*, *thin-tac skip i* $(x -$ *Suc 0*$) = x$, *blast*)
done

lemma *skip-fun-im1*:$[\![i \in \{j.\ j \leq$ (*Suc n*)$\}$; $x \in \{j.\ j \leq n\}]\!] \Longrightarrow$
                (*skip i*) $x \in (\{j.\ j \leq$ (*Suc n*)$\} - \{i\})$
by (*subst skip-fun-im*[*THEN sym*], *assumption*,

```
   simp add:image-def, blast)

lemma skip-id:l < i ⟹ skip i l = l
apply (simp add:skip-def )
 done

lemma Suc-neq:⟦0 < i; i − Suc 0 < l⟧ ⟹ i ≠ Suc l
by (rule contrapos-pp, simp+)

lemma skip-il-neq-i:skip i l ≠ i
apply (auto simp add:skip-def)
done

lemma skip-inj:⟦i ∈ {k. k ≤ n}; j ∈ {k. k ≤ n}; i ≠ j⟧ ⟹
                        skip k i ≠ skip k j
apply (simp add:skip-def)
done

lemma le-imp-add-int: i ≤ (j::nat) ⟹ ∃k. j = i + k
 apply (case-tac i = j)
 apply simp
 apply (frule le-imp-less-or-eq) apply (thin-tac i ≤ j)
 apply simp
 apply (insert less-imp-add-positive [of i j])
 apply simp
 apply blast
 done

lemma jointfun-hom0:⟦ f ∈ {j. j ≤ n} → A; g ∈ {k. k ≤ m} → B ⟧ ⟹
       (jointfun n f m g) ∈ {l. l ≤ (Suc (n + m))} →  (A ∪ B)
by (simp add:jointfun-def sliden-def Pi-def)

lemma jointfun-mem:⟦∀j ≤ (n::nat). f j ∈ A; ∀j ≤ m. g j ∈ B;
            l ≤ (Suc (n + m))⟧ ⟹ (jointfun n f m g) l ∈ (A ∪ B)
apply (rule funcset-mem[of jointfun n f m g {j. j ≤ Suc (n + m)} A ∪ B
       l])
apply (rule jointfun-hom0)
apply simp+
done

lemma jointfun-inj:⟦f ∈ {j. j ≤ n} → B; inj-on f {j. j ≤ n};
      b ∉ f ‘ {j. j ≤ n}⟧ ⟹
      inj-on (jointfun n f 0 (λk∈{0::nat}. b)) {j. j ≤ Suc n}
  apply (simp add:inj-on-def, (rule allI, rule impI)+, rule impI)
  apply (case-tac x = Suc n, simp)
       apply (case-tac y = Suc n, simp)
       apply (frule-tac m = y and n = Suc n in noteq-le-less, assumption)
          apply (
              frule-tac x = y and n = Suc n in less-le-diff,
```

```
        thin-tac y < Suc n, thin-tac y ≤ Suc n,
        simp add:jointfun-def sliden-def)
  apply (case-tac y = Suc n, simp,
      frule-tac m = x and n = Suc n in noteq-le-less, assumption,
      frule-tac x = x and n = Suc n in less-le-diff,
      thin-tac x < Suc n, thin-tac x ≤ Suc n,
      simp add:jointfun-def sliden-def)
  apply (rotate-tac −3, frule sym, thin-tac  f x = b, simp)
  apply (frule-tac m = x and n = Suc n in noteq-le-less, assumption,
      frule-tac x = x and n = Suc n in less-le-diff,
      thin-tac x < Suc n, thin-tac x ≤ Suc n, simp,
      frule-tac m = y and n = Suc n in noteq-le-less, assumption,
      frule-tac x = y and n = Suc n in less-le-diff,
      thin-tac y < Suc n, thin-tac y ≤ Suc n, simp,
      simp add:jointfun-def)
done

lemma slide-hom:i ≤ j ⟹ (slide i) ∈ {l. l ≤ (j − i)} → nset i j
apply (simp add:Pi-def restrict-def)
apply (rule allI) apply (rule impI)
  apply (simp add:slide-def)
apply (simp add:nset-def)
done

lemma slide-mem:⟦ i ≤ j; l ∈ {k. k ≤ (j − i)}⟧ ⟹ slide i l ∈ nset i j
apply (frule slide-hom)
apply (rule funcset-mem, assumption+)
done

lemma slide-iM:(slide i) ‘ {l. 0 ≤ l} = {k. i ≤ k}
apply (simp add:image-def slide-def)
apply (rule equalityI)
 apply (rule subsetI)
 apply simp
 apply auto
 apply (rule le-imp-add-int)
 apply assumption
done

lemma jointfun-hom:⟦ f ∈ {i. i ≤ n} → A; g ∈ {j. j ≤ m} → B ⟧ ⟹
          (jointfun n f m g) ∈ {j. j ≤ (Suc (n + m))} → A ∪ B
by (simp add:sliden-def Pi-def jointfun-def)

lemma im-jointfunTr1:(jointfun n f m g) ‘ {i. i ≤ n} = f ‘ {i. i ≤ n}
apply auto
  apply (simp add:jointfun-def)

  apply (simp add:jointfun-def)
 done
```

lemma *im-jointfunTr2*:(*jointfun n f m g*) ‘ (*nset* (*Suc n*) (*Suc* (*n* + *m*))) =
g ‘ ({*j*. *j* ≤ *m*})
apply *auto*
apply (*simp add*:*nset-def*) **apply** *auto*
apply (*frule-tac m* = *xa* **and** *n* = *Suc* (*n* + *m*) **and** *l* = *Suc n* **in** *diff-le-mono*)
apply *simp*
apply (*simp add*:*jointfun-def sliden-def*)

apply (*simp add*:*image-def*)
apply (*cut-tac le-add1*[*of n m*],
simp only:*Suc-le-mono*[*THEN sym*, *of n n*+*m*])
apply (*frule-tac l* = *xa* **in** *slide-mem*[*of Suc n Suc* (*n* + *m*)])
apply *simp*
apply (*subst jointfun-def*)
apply (*subgoal-tac* ∀ *i*∈*nset* (*Suc n*) (*Suc* (*n*+*m*)). ¬ (*i* ≤ *n*))
apply *simp*
apply (*thin-tac* ∀ *i*∈*nset* (*Suc n*) (*Suc* (*n* + *m*)). ¬ *i* ≤ *n*)
apply (*subgoal-tac g xa* = *g* (*sliden* (*Suc n*) (*slide* (*Suc n*) *xa*)))
apply *blast*
apply (*simp add*:*slide-def sliden-def*)
apply (*auto simp add*: *nset-def*)
done

lemma *im-jointfun*:⟦*f* ∈ {*j*. *j* ≤ *n*} → *A*; *g* ∈ {*j*. *j* ≤ *m*} → *B*⟧ ⟹
(*jointfun n f m g*) ‘({*j*. *j* ≤ (*Suc* (*n* + *m*))}) =
f ‘{*j*. *j* ≤ *n*} ∪ *g* ‘{*j*. *j* ≤ *m*}
apply (*cut-tac im-set-un1* [*of* {*j*. *j* ≤ (*Suc* (*n* + *m*))} *jointfun n f m g*
A ∪ *B* {*i*. *i* ≤ *n*} *nset* (*Suc n*) (*Suc* (*n* + *m*))])
apply (*simp add*:*Nset-nset*[*THEN sym*, *of n m*],
simp add:*im-jointfunTr1*[*of n f m g*],
simp add:*im-jointfunTr2*[*of n f m g*])
apply (*rule ballI*)
apply (*simp add*:*jointfun-def*,
case-tac l ≤ *n*, *simp add*:*Pi-def*,
simp add:*sliden-def*,
simp add:*nat-not-le-less*,
frule-tac m = *n* **and** *n* = *l* **in** *Suc-leI*,
frule-tac m = *l* **and** *n* = *Suc* (*n* + *m*) **and** *l* = *Suc n* **in** *diff-le-mono*,
thin-tac l ≤ *Suc* (*n* + *m*), *simp*,
simp add:*Pi-def*)
apply (*simp add*:*Nset-nset*[*of n m*])
done

lemma *im-jointfun1*:(*jointfun n f m g*) ‘({*j*. *j* ≤ (*Suc* (*n* + *m*))}) =
f ‘{*j*. *j* ≤ *n*} ∪ *g* ‘ {*j*. *j* ≤ *m*}
apply (*cut-tac Nset-nset*[*of n m*])
apply (*subst im-set-un2*[*of* {*j*. *j* ≤ (*Suc* (*n* + *m*))} {*j*. *j* ≤ *n*}
nset (*Suc n*) (*Suc* (*n* + *m*)) *jointfun n f m g*], *assumption*)

apply (*simp add*:*im-jointfunTr1 im-jointfunTr2*)
done

lemma *jointfun-surj*:⟦$f \in \{j.\ j \leq n\} \to A$; *surj-to f* $\{j.\ j \leq (n::nat)\}$ A;
$g \in \{j.\ j \leq (m::nat)\} \to B$; *surj-to g* $\{j.\ j \leq m\}$ B⟧ ⟹
surj-to (*jointfun n f m g*) $\{j.\ j \leq Suc\ (n + m)\}$ $(A \cup B)$
apply (*simp add*:*surj-to-def* [*of jointfun n f m g*])
apply (*simp add*:*im-jointfun*)
apply (*simp add*:*surj-to-def*)
done

lemma *Nset-un*:$\{j.\ j \leq (Suc\ n)\} = \{j.\ j \leq n\} \cup \{Suc\ n\}$
apply (*rule equalityI*)
apply (*rule subsetI*)
apply *simp*
apply *auto*
done

lemma *Nsetn-sub*: $\{j.\ j \leq n\} \subseteq \{j.\ j \leq (Suc\ n)\}$
apply (*rule subsetI*)
apply *simp*
done

lemma *Nset-pre-sub*:$(0::nat) < k \Longrightarrow \{j.\ j \leq (k - Suc\ 0)\} \subseteq \{j.\ j \leq k\}$
apply (*rule subsetI*)
apply *simp*
done

lemma *Nset-pre-un*:$(0::nat) < k \Longrightarrow \{j.\ j \leq k\} = \{j.\ j \leq (k - Suc\ 0)\} \cup \{k\}$
apply (*insert Nset-un* [*of k − Suc 0*])
apply *simp*
done

lemma *Nsetn-sub-mem*: $l \in \{j.\ j \leq n\} \Longrightarrow l \in \{j.\ j \leq (Suc\ n)\}$
apply *simp*
done

lemma *Nsetn-sub-mem1*:$\forall j.\ j \in \{j.\ j \leq n\} \longrightarrow j \in \{j.\ j \leq (Suc\ n)\}$
by (*simp add*:*Nsetn-sub-mem*)

lemma *Nset-Suc*:$\{j.\ j \leq (Suc\ n)\} = insert\ (Suc\ n)\ \{j.\ j \leq n\}$
apply (*rule equalityI*)
apply (*rule subsetI*)
apply *simp*
apply *auto*
done

lemma *nsetnm-sub-mem*:$\forall j.\ j \in nset\ n\ (n + m) \longrightarrow j \in nset\ n\ (Suc\ (n + m))$
by (*rule allI*, *simp add*:*nset-def*)

lemma *Nset-0*:$\{j.\ j \leq (0::nat)\} = \{0\}$
by *simp*

lemma *Nset-Suc0*:$\{i.\ i \leq (Suc\ 0)\} = \{0,\ Suc\ 0\}$
apply (*rule equalityI*)
apply (*rule subsetI*, *simp*)
apply (*case-tac* $x = 0$, *simp*)
apply *simp*+
done

lemma *Nset-Suc-Suc*:$Suc\ (Suc\ 0) \leq n \Longrightarrow$
$\{j.\ j \leq (n - Suc\ (Suc\ 0))\} = \{j.\ j \leq n\} - \{n - Suc\ 0,\ n\}$
apply (*insert Nset-un* [*of* $n - (Suc\ 0)$])
apply (*insert Nset-un* [*of* $n - Suc\ (Suc\ 0)$])
apply (*subgoal-tac* $\{j.\ j \leq (Suc\ (n - Suc\ (Suc\ 0)))\} = \{j.\ j \leq (n - Suc\ 0)\}$)
apply (*simp*,
thin-tac $\{j.\ j \leq n\} =$
insert n (*insert* ($Suc\ (n - Suc\ (Suc\ 0))$) $\{j.\ j \leq n - Suc\ (Suc\ 0)\}$),
thin-tac $\{j.\ j \leq n - Suc\ 0\} =$
insert ($Suc\ (n - Suc\ (Suc\ 0))$) $\{j.\ j \leq n - Suc\ (Suc\ 0)\}$,
thin-tac $\{j.\ j \leq Suc\ (n - Suc\ (Suc\ 0))\} =$
insert ($Suc\ (n - Suc\ (Suc\ 0))$) $\{j.\ j \leq n - Suc\ (Suc\ 0)\}$)
apply (*simp add*:*Suc-Suc-Tr*)
apply (*auto*)
done

lemma *func-pre*:$f \in \{j.\ j \leq (Suc\ n)\} \to A \Longrightarrow f \in \{j.\ j \leq n\} \to A$
by (*simp add*:*Pi-def*)

lemma *image-Nset-Suc*:$f\ `\ (\{j.\ j \leq (Suc\ n)\}) =$
$insert\ (f\ (Suc\ n))\ (f\ `\ \{j.\ j \leq n\})$
apply (*cut-tac Nset-un*[*of* n])
apply (*frule im-set-un2*[*of* $\{j.\ j \leq (Suc\ n)\}$ $\{j.\ j \leq n\}$ $\{Suc\ n\}$ f])
apply (*simp add*:*Un-commute*)
done

definition
Nleast :: *nat set* ⇒ *nat* **where**
Nleast $A = (THE\ a.\ (a \in A \wedge (\forall\, x{\in}A.\ a \leq x)))$

definition
Nlb :: [*nat set*, *nat*] ⇒ *bool* **where**
Nlb $A\ n \longleftrightarrow (\forall\, a{\in}A.\ n \leq a)$

primrec *ndec-seq* :: [*nat set*, *nat*, *nat*] ⇒ *nat* **where**
ndec-seq-0 :*ndec-seq* $A\ a\ 0 = a$
| *ndec-seq-Suc*:*ndec-seq* $A\ a\ (Suc\ n) =$

$$(SOME\ b.\ ((b \in A) \wedge b < (ndec\text{-}seq\ A\ a\ n)))$$

lemma *ndec-seq-mem*:$[\![a \in (A::nat\ set);\ \neg\ (\exists\, m.\ m{\in}A \wedge (\forall\, x{\in}A.\ m \leq x))]\!] \Longrightarrow$
$(ndec\text{-}seq\ A\ a\ n) \in A$
apply (*induct-tac n*)
apply *simp* **apply** *simp*
apply (*simp add*: *not-less* [*symmetric*])
apply (*subgoal-tac* $\exists\, x{\in}A.\ x < (ndec\text{-}seq\ A\ a\ n)$) **prefer** *2* **apply** *blast*
apply (*thin-tac* $\forall\, m.\ m \in A \longrightarrow (\exists\, x{\in}A.\ x < m)$)
apply (*rule someI2-ex*) **apply** *blast*
apply *simp*
done

lemma *ndec-seqn*:$[\![a \in (A::nat\ set);\neg\ (\exists\, m.\ m{\in}A \wedge (\forall\, x{\in}A.\ m \leq x))]\!] \Longrightarrow$
$(ndec\text{-}seq\ A\ a\ (Suc\ n)) < (ndec\text{-}seq\ A\ a\ n)$
apply (*frule ndec-seq-mem* [*of a A n*], *assumption*+)
apply *simp*
apply (*simp add*: *not-less* [*symmetric*])
apply (*subgoal-tac* $\exists\, x{\in}A.\ x < (ndec\text{-}seq\ A\ a\ n)$) **prefer** *2* **apply** *simp*
apply (*thin-tac* $\forall\, m.\ m \in A \longrightarrow (\exists\, x{\in}A.\ x < m)$)
apply (*rule someI2-ex*) **apply** *blast*
apply *simp*
done

lemma *ndec-seqn1*:$[\![a \in (A::nat\ set);\ \neg\ (\exists\, m.\ m{\in}A \wedge (\forall\, x{\in}A.\ m \leq x))]\!] \Longrightarrow$
$(ndec\text{-}seq\ A\ a\ (Suc\ n)) \leq (ndec\text{-}seq\ A\ a\ n) - 1$
apply (*frule ndec-seqn* [*of a A n*], *assumption*+,
thin-tac $\neg\ (\exists\, m.\ m \in A \wedge (\forall\, x{\in}A.\ m \leq x))$)
apply (*simp del*:*ndec-seq-Suc*)
done

lemma *ex-NleastTr*:$[\![a \in (A::nat\ set);\ \neg\ (\exists\, m.\ m{\in}A \wedge (\forall\, x{\in}A.\ m \leq x))]\!] \Longrightarrow$
$(ndec\text{-}seq\ A\ a\ n) \leq (a - n)$
apply (*induct-tac n*)
apply *simp*
apply (*frule-tac* $n = n$ **in** *ndec-seqn1*[*of a A*], *assumption*+)
apply (*subgoal-tac ndec-seq A a n* $-$ *1* $\leq (a - n) - 1$) **prefer** *2*
apply *arith*
apply *arith*
done

lemma *nat-le*:$((a::nat) - (a + 1)) \leq 0$
apply *arith*
done

lemma *ex-Nleast*:$(A::nat\ set) \neq \{\} \Longrightarrow \exists!m.\ m{\in}A \wedge (\forall\, x{\in}A.\ m \leq x)$
apply (*frule nonempty-ex*[*of A*], *thin-tac* $A \neq \{\}$,
erule exE, *rename-tac a*)
apply (*case-tac* $0 \in A$)

apply (*rule ex-ex1I*, *subgoal-tac* $\forall x{\in}A.\ 0 \leq a$, *blast*,
rule ballI, *simp*)
apply ((*erule conjE*)+,
subgoal-tac $m \leq 0$, *thin-tac* $\forall x{\in}A.\ m \leq x$,
subgoal-tac $y \leq 0$, *thin-tac* $\forall x{\in}A.\ y \leq x$,
simp, *blast*, *blast*)
apply (*rule ex-ex1I*)
prefer *2* **apply** (*erule conjE*)+
apply (*subgoal-tac* $m \leq y$, *thin-tac* $\forall x{\in}A.\ m \leq x$,
subgoal-tac $y \leq m$, *thin-tac* $\forall x{\in}A.\ y \leq x$,
simp, *blast*, *blast*)
apply (*rule contrapos-pp*, *simp*,
frule-tac $a = a$ **and** $A = A$ **and** $n = a + 1$ **in** *ex-NleastTr*, *assumption*+)
apply (*subgoal-tac* $(a - (a + 1)) \leq 0$)
prefer *2* **apply** (*rule nat-le*)
apply (*frule-tac* $i = ndec\text{-}seq\ A\ a\ (a + 1)$ **and** $j = a - (a + 1)$ **and** $k = 0$ **in**
le-trans, *assumption*+,
frule-tac $a = a$ **and** $n = a + 1$ **in** *ndec-seq-mem* [*of* - *A*],
assumption+)
apply (*thin-tac* $\neg\ (\exists m.\ m \in A \wedge (\forall x{\in}A.\ m \leq x))$,
thin-tac $ndec\text{-}seq\ A\ a\ (a + 1) \leq a - (a + 1)$,
thin-tac $a - (a + 1) \leq 0$)
apply *simp*
done

lemma *Nleast*:$(A{::}nat\ set) \neq \{\} \Longrightarrow Nleast\ A \in A \wedge (\forall x{\in}A.\ (Nleast\ A) \leq x)$
apply (*frule ex-Nleast* [*of A*])
apply (*simp add*:*Nleast-def*)
apply (*rule theI'*)
apply *simp*
done

1.5.1 Lemmas for existence of reduced chain.

lemma *jointgd-tool1*: $0 < i \Longrightarrow 0 \leq i - Suc\ 0$
by *arith*

lemma *jointgd-tool2*: $0 < i \Longrightarrow i = Suc\ (i - Suc\ 0)$
by *arith*

lemma *jointgd-tool3*:$[\![0 < i;\ \ i \leq m]\!] \Longrightarrow i - Suc\ 0 \leq (m - Suc\ 0)$
by *arith*

lemma *jointgd-tool4*:$n < i \Longrightarrow i - n = Suc(\ i - Suc\ n)$
by *arith*

lemma *pos-prec-less*:$0 < i \Longrightarrow i - Suc\ 0 < i$
by *arith*

lemma *Un-less-Un*:⟦$f \in \{j.\ j \leq (Suc\ n)\} \to (X::'a\ set\ set)$;
$A \subseteq \bigcup (f\ `\ \{j.\ j \leq (Suc\ n)\})$;
$i \in \{j.\ j \leq (Suc\ n)\}$; $j \in \{l.\ l \leq (Suc\ n)\}$; $i \neq j \wedge f\ i \subseteq f\ j$⟧
$\Longrightarrow A \subseteq \bigcup (compose\ \{j.\ j \leq n\}\ f\ (skip\ i)\ `\ \{j.\ j \leq n\})$
apply (*simp add*:*compose-def*)
apply (*rule subsetI*, *simp*)
apply (*frule-tac* $c = x$ **and** $A = A$ **and** $B = \bigcup x{\in}\{j.\ j \leq Suc\ n\}.\ f\ x$ **in**
subsetD, *assumption*+, *simp*)
apply (*erule exE*, (*erule conjE*)+)
apply (*case-tac* $xa = i$, *simp*,
frule-tac $c = x$ **in** *subsetD*[*of f i f j*], *assumption*+)
apply (*cut-tac less-linear*[*of i j*], *simp*, *erule disjE*,
frule less-le-diff[*of i j*],
cut-tac skip-im-Tr2-1[*of i n j* − *Suc 0*],
simp,
frule eq-elems-eq-val[*THEN sym*, *of skip i* (*j* − *Suc 0*) *j f*],
cut-tac $a = x$ **in** *eq-set-inc*[*of - f j f* (*skip i* (*j* − *Suc 0*))],
assumption+,
frule le-Suc-diff-le[*of j n*], *blast*, *simp*, *assumption*, *simp*)
apply (*frule skip-im-Tr1-2*[*of i n j*], *assumption*,
frule eq-elems-eq-val[*THEN sym*, *of skip i j j f*])
apply (*cut-tac* $a = x$ **in** *eq-set-inc*[*of - f j f* (*skip i j*)],
assumption+)
apply (*frule-tac* $x = j$ **and** $y = i$ **and** $z = Suc\ n$ **in** *less-le-trans*,
assumption+,
frule Suc-less-le[*of j n*], *blast*)
apply (*cut-tac* $x = xa$ **and** $y = i$ **in** *less-linear*, *simp*,
erule disjE,
frule-tac $x = xa$ **in** *skip-im-Tr1-2*[*of i n*], *assumption*)
apply (*frule-tac x1* = *skip i xa* **and** *y1* = *xa* **and** *f1* = *f* **in**
eq-elems-eq-val[*THEN sym*],
frule-tac $a = x$ **and** $A = f\ xa$ **and** $B = f\ (skip\ i\ xa)$ **in** *eq-set-inc*,
assumption,
frule-tac $x = xa$ **and** $y = i$ **and** $z = Suc\ n$ **in** *less-le-trans*,
assumption+,
frule-tac $x = xa$ **and** $n = n$ **in** *Suc-less-le*, *blast*)
apply (*frule-tac* $x = i$ **and** $n = xa$ **in** *less-le-diff*,
cut-tac $x = xa - Suc\ 0$ **and** $n = n$ **in** *skip-im-Tr2-1* [*of i*],
simp, *assumption*,
simp,
frule-tac x1 = *skip i* (*xa* − *Suc 0*) **and** *y1* = *xa* **and** *f1* = *f* **in**
eq-elems-eq-val[*THEN sym*],
frule-tac $a = x$ **and** $A = f\ xa$ **and** $B = f\ (skip\ i\ (xa - Suc\ 0))$ **in**
eq-set-inc, *assumption*,
frule-tac $x = xa$ **and** $n = n$ **in** *le-Suc-diff-le*)
apply *blast*
done

1.6 Lower bounded set of integers

definition $Zset = \{x.\ \exists (n::int).\ x = n\}$

definition
$Zleast :: int\ set \Rightarrow int$ **where**
$Zleast\ A = (THE\ a.\ (a \in A \wedge (\forall x{\in}A.\ a \leq x)))$

definition
$LB :: [int\ set,\ int] \Rightarrow bool$ **where**
$LB\ A\ n = (\forall a{\in}A.\ n \leq a)$

lemma *linorder-linear1*:$(m::int) < n \vee n \leq m$
apply $(subgoal\text{-}tac\ m < n \vee n = m \vee n < m)$
apply $(case\text{-}tac\ m < n)$ **apply** *simp* **apply** *simp*
apply $(subgoal\text{-}tac\ m < n \vee m = n \vee n < m)$
apply *blast*
apply $(simp\ add{:}less\text{-}linear)$
done

primrec *dec-seq* :: $[int\ set,\ int,\ nat] \Rightarrow int$
where
dec-seq-0: $dec\text{-}seq\ A\ a\ 0 = a$
| *dec-seq-Suc*: $dec\text{-}seq\ A\ a\ (Suc\ n) = (SOME\ b.\ ((b \in A) \wedge b < (dec\text{-}seq\ A\ a\ n)))$

lemma *dec-seq-mem*:$[\![a \in A;\ A \subseteq Zset;\neg\ (\exists m.\ m{\in}A \wedge (\forall x{\in}A.\ m \leq x))]\!] \Longrightarrow$
$(dec\text{-}seq\ A\ a\ n) \in A$
apply $(induct\text{-}tac\ n)$
apply *simp* **apply** *simp* **apply** $(simp\ add{:}not\text{-}zle)$
apply $(subgoal\text{-}tac\ \exists x{\in}A.\ x < (dec\text{-}seq\ A\ a\ n))$ **prefer** *2* **apply** *blast*
apply $(thin\text{-}tac\ \forall m.\ m \in A \longrightarrow (\exists x{\in}A.\ x < m))$
apply $(rule\ someI2\text{-}ex)$ **apply** *blast*
apply *simp*
done

lemma *dec-seqn*:$[\![a \in A;\ A \subseteq Zset;\neg\ (\exists m.\ m{\in}A \wedge (\forall x{\in}A.\ m \leq x))]\!] \Longrightarrow$
$(dec\text{-}seq\ A\ a\ (Suc\ n)) < (dec\text{-}seq\ A\ a\ n)$
apply *simp*
apply $(frule\ dec\text{-}seq\text{-}mem\ [of\ a\ A\ n],\ assumption{+})$
apply *simp*
apply $(simp\ add{:}not\text{-}zle)$
apply $(subgoal\text{-}tac\ \exists x{\in}A.\ x < (dec\text{-}seq\ A\ a\ n))$ **prefer** *2* **apply** *simp*
apply $(thin\text{-}tac\ \forall m.\ m \in A \longrightarrow (\exists x{\in}A.\ x < m))$
apply $(rule\ someI2\text{-}ex)$ **apply** *blast*
apply *simp*
done

lemma *dec-seqn1*:$[\![a \in A;\ A \subseteq Zset;\neg\ (\exists m.\ m{\in}A \wedge (\forall x{\in}A.\ m \leq x))]\!] \Longrightarrow$
$(dec\text{-}seq\ A\ a\ (Suc\ n)) \leq (dec\text{-}seq\ A\ a\ n) - 1$

apply (*frule dec-seqn* [*of a A n*], *assumption*+)
 apply *simp*
done

lemma *lbs-ex-ZleastTr*:⟦*a* ∈ *A*; *A* ⊆ *Zset*;¬ (∃ *m*. *m*∈*A* ∧ (∀ *x*∈*A*. *m* ≤ *x*))⟧ ⟹
(*dec-seq A a n*) ≤ (*a* − *int*(*n*))
apply (*induct-tac n*)
 apply *simp*
apply (*frule-tac n* = *n* **in** *dec-seqn1*[*of a A*], *assumption*+)
 apply (*subgoal-tac dec-seq A a n* − *1* ≤ *a* − (*int n*) − *1*) **prefer** *2*
 apply *simp* **apply** (*thin-tac dec-seq A a n* ≤ *a* − *int n*)
 apply (*frule-tac x* = *dec-seq A a* (*Suc n*) **and** *y* = *dec-seq A a n* − *1* **and**
z = *a* − *int n* − *1* **in** *order-trans*, *assumption*+)
 apply (*thin-tac* ¬ (∃ *m*. *m* ∈ *A* ∧ (∀ *x*∈*A*. *m* ≤ *x*)))
 apply (*thin-tac dec-seq A a* (*Suc n*) ≤ *dec-seq A a n* − *1*)
 apply (*thin-tac dec-seq A a n* − *1* ≤ *a* − *int n* − *1*)
apply (*subgoal-tac a* − *int n* − *1* = *a* − *int* (*Suc n*)) **apply** *simp*
 apply (*thin-tac dec-seq A a* (*Suc n*) ≤ *a* − *int n* − *1*)
apply *simp*
done

lemma *big-int-less*:*a* − *int*(*nat*(*abs*(*a*) + *abs*(*N*) + *1*)) < *N*
apply (*simp add*:*zabs-def*)
done

lemma *lbs-ex-Zleast*:⟦*A* ≠ {}; *A* ⊆ *Zset*; *LB A n*⟧ ⟹ ∃!*m*. *m*∈*A* ∧ (∀ *x*∈*A*. *m* ≤
x)
apply (*frule nonempty-ex*[*of A*])
 apply (*thin-tac A* ≠ {})
 apply (*erule exE*)
 apply (*rename-tac a*)
apply (*rule ex-ex1I*)
prefer *2*
 apply (*thin-tac LB A n*) **apply** (*erule conjE*)+
 apply (*subgoal-tac m* ≤ *y*) **prefer** *2* **apply** *simp*
 apply (*subgoal-tac y* ≤ *m*) **prefer** *2* **apply** *simp*
 apply (*thin-tac* ∀ *x*∈*A*. *m* ≤ *x*) **apply** (*thin-tac* ∀ *x*∈*A*. *y* ≤ *x*)
 apply *simp*
apply (*rule contrapos-pp*) **apply** *simp*
 apply (*frule-tac a* = *a* **and** *A* = *A* **and** *n* = *nat*(*abs*(*a*) + *abs*(*n*) + *1*) **in**
lbs-ex-ZleastTr, *assumption*+)
 apply (*subgoal-tac a* − *int*(*nat*(*abs*(*a*) + *abs*(*n*) + *1*)) < *n*)
 prefer *2* **apply** (*rule big-int-less*)
 apply (*frule-tac x* = *dec-seq A a* (*nat* (|*a*| + |*n*| + *1*)) **and** *y* = *a* − *int* (*nat*
(|*a*| + |*n*| + *1*)) **and** *z* = *n* **in** *order-le-less-trans*, *assumption*+)
 apply (*frule-tac a* = *a* **and** *n* = *nat* (|*a*| + |*n*| + *1*) **in** *dec-seq-mem* [*of* - *A*],
assumption+)
 apply (*thin-tac* ¬ (∃ *m*. *m* ∈ *A* ∧ (∀ *x*∈*A*. *m* ≤ *x*)))
 apply (*thin-tac dec-seq A a* (*nat* (|*a*| + |*n*| + *1*))

$\leq a - int\ (nat\ (|a| + |n| + 1)))$
apply (*thin-tac* $a - int\ (nat\ (|a| + |n| + 1)) < n$)
apply (*simp add*:*LB-def*)
apply (*subgoal-tac* $n \leq$ *dec-seq A a* (*nat* $(|a| + |n| + 1)))$)
apply (*thin-tac* $\forall\, a{\in}A.\ n \leq a$) **apply** (*simp add*:*not-zle*)
apply *blast*
done

lemma *Zleast*:$[\![A \neq \{\};\ A \subseteq Zset;\ LB\ A\ n]\!] \Longrightarrow Zleast\ A \in A \wedge$
$(\forall\, x{\in}A.\ (Zleast\ A) \leq x)$
apply (*frule lbs-ex-Zleast* [*of A n*], *assumption*+)
apply (*simp add*:*Zleast-def*)
apply (*rule theI'*)
apply *simp*
done

lemma *less-convert1*:$[\![\ a = c;\ a < b\ ]\!] \Longrightarrow c < b$
apply *auto*
done

lemma *less-convert2*:$[\![a = b;\ b < c]\!] \Longrightarrow a < c$
apply *auto*
done

1.7 Augmented integer: integer and $\infty{-}\infty$

definition
zag :: (*int* $*$ *int*) *set* **where**
zag = $\{(x,y) \mid x\ y.\ x * y = (0{::}int) \wedge (y = -1 \vee y = 0 \vee y = 1)\}$

definition
zag-pl::[(*int* $*$ *int*), (*int* $*$ *int*)] $\Rightarrow$ (*int* $*$ *int*) **where**
zag-pl x y == *if* (*snd x* + *snd y*) = *2 then* (*0*, *1*)
else if (*snd x* + *snd y*) = *1 then* (*0*, *1*)
else if (*snd x* + *snd y*) = *0 then* (*fst x* + *fst y*, *0*)
else if (*snd x* + *snd y*) = -1 *then* (*0*, -1)
else if (*snd x* + *snd y*) = -2 *then* (*0*, -1) *else undefined*

definition
zag-t :: [(*int* $*$ *int*), (*int* $*$ *int*)] $\Rightarrow$ (*int* $*$ *int*) **where**
zag-t x y = (*if* (*snd x*)$*$(*snd y*) = *0 then*
(*if 0* $<$ (*fst x*)$*$(*snd y*) + (*snd x*)$*$(*fst y*) *then* (*0*,*1*)
else (*if* (*fst x*)$*$(*snd y*) + (*snd x*)$*$(*fst y*) = *0*
then ((*fst x*)$*$(*fst y*), *0*) *else* (*0*, -1)))
else (*if 0* $<$ (*snd x*)$*$(*snd y*) *then* (*0*, *1*) *else* (*0*, -1)))

definition *Ainteg* = *zag*

typedef *ant* = *Ainteg*

morphisms *Rep-Ainteg Abs-Ainteg*
unfolding *Ainteg-def*
proof
show $(1, 0) \in zag$ **unfolding** *zag-def* **by** *auto*
qed

definition
$ant :: int \Rightarrow ant$ **where**
$ant\ m = Abs\text{-}Ainteg(\ (m, 0))$

definition
$tna :: ant \Rightarrow int$ **where**
$tna\ z = (if\ Rep\text{-}Ainteg(z) \neq (0,1) \wedge Rep\text{-}Ainteg(z) \neq (0,-1)\ then$
$fst\ (Rep\text{-}Ainteg(z))\ else\ undefined)$

instantiation $ant :: \{zero, one, plus, uminus, minus, times, ord\}$
begin

definition
$Zero\text{-}ant\text{-}def\ : 0 == ant\ 0$

definition
$One\text{-}ant\text{-}def\ : 1 == ant\ 1$

definition
add-ant-def:
$z + w ==$
$Abs\text{-}Ainteg\ (zag\text{-}pl\ (Rep\text{-}Ainteg\ z)\ (Rep\text{-}Ainteg\ w))$

definition
$minus\text{-}ant\text{-}def : - z ==$
$Abs\text{-}Ainteg((-\ (fst\ (Rep\text{-}Ainteg\ z)), -\ (snd\ (Rep\text{-}Ainteg\ z))))$

definition
$diff\text{-}ant\text{-}def:\ z - (w::ant) == z + (-w)$

definition
mult-ant-def:
$z * w ==$
$Abs\text{-}Ainteg\ (zag\text{-}t\ (Rep\text{-}Ainteg\ z)\ (Rep\text{-}Ainteg\ w))$

definition
le-ant-def:
$(z::ant) \leq w == if\ (snd\ (Rep\text{-}Ainteg\ w)) = 1\ then\ True$
$else\ (if\ (snd\ (Rep\text{-}Ainteg\ w)) = 0\ then\ (if\ (snd\ (Rep\text{-}Ainteg\ z)) = 1$
$then\ False\ else\ (if\ (snd\ (Rep\text{-}Ainteg\ z)) = 0\ then$
$(fst\ (Rep\text{-}Ainteg\ z)) \leq (fst\ (Rep\text{-}Ainteg\ w))\ \ else\ True))$
$else\ (if\ snd\ (Rep\text{-}Ainteg\ z) = -1\ then\ True\ else\ False))$

definition
less-ant-def: $((z::ant) < (w::ant)) == (z \leq w \wedge z \neq w)$

instance ..

end

definition
$inf\text{-}ant :: ant\ \ (\infty)$ **where**
$\infty = Abs\text{-}Ainteg((0,1))$

definition
$an :: nat \Rightarrow ant$ **where**
$an\ m = ant\ (int\ m)$

definition
$na :: ant \Rightarrow nat$ **where**
$na\ x = (if\ (x < 0)\ then\ 0\ else$
$\quad if\ x \neq \infty\ then\ (nat\ (tna\ x))\ else\ undefined)$

definition
$UBset :: ant \Rightarrow ant\ set$ **where**
$UBset\ z = \{(x::ant).\ \ x \leq z\}$

definition
$LBset :: ant \Rightarrow ant\ set$ **where**
$LBset\ z = \{(x::ant).\ z \leq x\}$

lemma *ant-z-in-Ainteg*:$(z::int,\ 0) \in Ainteg$
apply (*simp add*:*Ainteg-def zag-def*)
done

lemma *ant-inf-in-Ainteg*:$((0::int),\ 1) \in Ainteg$
apply (*simp add*:*Ainteg-def zag-def*)
done

lemma *ant-minf-in-Ainteg*:$((0::int),\ -1) \in Ainteg$
apply (*simp add*:*Ainteg-def zag-def*)
done

lemma *ant-0-in-Ainteg*:$((0::int),\ 0) \in Ainteg$
apply (*simp add*:*Ainteg-def zag-def*)
done

lemma *an-0*[*simp*]:*an 0 = 0*
by (*simp add*:*an-def Zero-ant-def*)

lemma *an-1*[*simp*]:*an 1 = 1*
by (*simp add*:*an-def One-ant-def*)

lemma *mem-ant*:(*a*::*ant*) = $-\infty \vee (\exists (z::int).\ a = ant\ z) \vee a = \infty$
apply (*case-tac* $a = -\infty \vee a = \infty$)
apply *blast*
apply (*simp*, *simp add*:*ant-def*,
cut-tac Rep-Ainteg[*of a*],
simp add:*Ainteg-def zag-def*,
erule conjE, *simp add*:*inf-ant-def*,
simp add:*minus-ant-def*,
cut-tac ant-inf-in-Ainteg,
simp add:*Abs-Ainteg-inverse*)
apply *auto*
apply (*cut-tac Rep-Ainteg*[*of a*],
subgoal-tac Abs-Ainteg (*Rep-Ainteg a*) = *Abs-Ainteg* ((*0*,-1)),
thin-tac Rep-Ainteg a = (*0*, -1),
simp add:*Rep-Ainteg-inverse*, *simp*)
apply (*cut-tac Rep-Ainteg*[*of a*],
subgoal-tac Abs-Ainteg (*Rep-Ainteg a*) = *Abs-Ainteg* ((*0*,*0*)),
thin-tac Rep-Ainteg a = (*0*, *0*),
simp add:*Rep-Ainteg-inverse*, *blast*, *simp*)
apply (*cut-tac Rep-Ainteg*[*of a*],
subgoal-tac Abs-Ainteg (*Rep-Ainteg a*) = *Abs-Ainteg* ((*0*,*1*)),
thin-tac Rep-Ainteg a = (*0*, *1*),
simp add:*Rep-Ainteg-inverse*, *simp*)
apply (*cut-tac Rep-Ainteg*[*of a*],
subgoal-tac Abs-Ainteg (*Rep-Ainteg a*) = *Abs-Ainteg* ((*x*,*0*)),
thin-tac Rep-Ainteg a = (*x*, *0*),
simp add:*Rep-Ainteg-inverse*, *blast*, *simp*)
done

lemma *minf*:$-\infty$ = *Abs-Ainteg*((*0*,-1))
apply (*simp add*:*inf-ant-def minus-ant-def*,
cut-tac ant-inf-in-Ainteg,
simp add:*Abs-Ainteg-inverse*)
done

lemma *z-neq-inf*[*simp*]:(*ant z*) $\neq \infty$
apply (*rule contrapos-pp*, *simp*+)
apply (*simp add*:*ant-def inf-ant-def*)
apply (*subgoal-tac Rep-Ainteg* (*Abs-Ainteg* (*z*,*0*)) =
Rep-Ainteg (*Abs-Ainteg* (*0*,*1*)),
thin-tac Abs-Ainteg (*z*, *0*) = *Abs-Ainteg* (*0*, *1*),
cut-tac ant-z-in-Ainteg[*of z*],
cut-tac ant-inf-in-Ainteg,
simp add:*Abs-Ainteg-inverse*)
apply *simp*
done

lemma *z-neq-minf*[*simp*]:(*ant z*) $\neq -\infty$

apply (*rule contrapos-pp*, *simp+*)
apply (*subgoal-tac ant* $(-z) = \infty$)
apply (*cut-tac z-neq-inf*[*of* $-$ *z*], *simp*)
apply (*simp add*:*ant-def inf-ant-def minus-ant-def*)
apply (*cut-tac ant-inf-in-Ainteg*,
simp add:*Abs-Ainteg-inverse*)
apply (*subgoal-tac* $-$ *Abs-Ainteg* (*z*, *0*) $=$ $-$ *Abs-Ainteg* (*0*, -1),
thin-tac Abs-Ainteg (*z*, *0*) $=$ *Abs-Ainteg* (*0*, -1),
simp add:*minus-ant-def*,
cut-tac ant-z-in-Ainteg[*of z*],
cut-tac ant-inf-in-Ainteg,
cut-tac ant-minf-in-Ainteg,
simp add:*Abs-Ainteg-inverse*)
apply *simp*
done

lemma *minf-neq-inf*[*simp*]:$-\infty \neq \infty$
apply (*cut-tac ant-inf-in-Ainteg*,
simp add:*inf-ant-def minus-ant-def Abs-Ainteg-inverse*)
apply (*rule contrapos-pp*, *simp+*,
subgoal-tac Rep-Ainteg (*Abs-Ainteg* (*0*,-1)) $=$
Rep-Ainteg (*Abs-Ainteg* (*0*,*1*)),
thin-tac Abs-Ainteg (*0*, -1) $=$ *Abs-Ainteg* (*0*, *1*),
cut-tac ant-minf-in-Ainteg,
simp add:*Abs-Ainteg-inverse*)
apply *simp*
done

lemma *a-ipi*[*simp*]:$\infty + \infty = \infty$
apply (*simp add*:*add-ant-def inf-ant-def*,
cut-tac ant-inf-in-Ainteg,
simp add:*Abs-Ainteg-inverse*,
simp add:*zag-pl-def*)
done

lemma *a-zpi*[*simp*]:(*ant z*) $+ \infty = \infty$
apply (*simp add*:*add-ant-def inf-ant-def ant-def*,
cut-tac ant-z-in-Ainteg[*of z*],
cut-tac ant-inf-in-Ainteg,
simp add:*Abs-Ainteg-inverse*,
simp add:*zag-pl-def*)
done

lemma *a-ipz*[*simp*]: $\infty +$ (*ant z*) $= \infty$
apply (*simp add*:*add-ant-def inf-ant-def ant-def*,
cut-tac ant-z-in-Ainteg[*of z*],
cut-tac ant-inf-in-Ainteg,
simp add:*Abs-Ainteg-inverse*,

simp add:*zag-pl-def*)
done

lemma *a-zpz*:(*ant m*) + (*ant n*) = *ant* (*m* + *n*)
apply (*simp add*:*add-ant-def inf-ant-def ant-def*,
cut-tac ant-z-in-Ainteg[*of m*],
cut-tac ant-z-in-Ainteg[*of n*],
simp add:*Abs-Ainteg-inverse*,
simp add:*zag-pl-def*)
done

lemma *a-mpi*[*simp*]:−∞ + ∞ = *0*
apply (*simp add*:*add-ant-def inf-ant-def*,
cut-tac ant-inf-in-Ainteg,
cut-tac ant-minf-in-Ainteg,
simp add:*minus-ant-def*,
simp add:*Abs-Ainteg-inverse*,
simp add:*Zero-ant-def ant-def zag-pl-def*)
done

lemma *a-ipm*[*simp*]:∞ + (−∞) = *0*
apply (*simp add*:*add-ant-def inf-ant-def*,
cut-tac ant-inf-in-Ainteg,
cut-tac ant-minf-in-Ainteg,
simp add:*minus-ant-def*,
simp add:*Abs-Ainteg-inverse*,
simp add:*Zero-ant-def ant-def zag-pl-def*)
done

lemma *a-mpm*[*simp*]:−∞ + (−∞) = −∞
apply (*simp add*:*add-ant-def inf-ant-def*,
cut-tac ant-inf-in-Ainteg,
cut-tac ant-minf-in-Ainteg,
simp add:*minus-ant-def*,
simp add:*Abs-Ainteg-inverse*,
simp add:*Zero-ant-def ant-def zag-pl-def*)
done

lemma *a-mpz*[*simp*]:−∞ + (*ant m*) = −∞
apply (*simp add*:*add-ant-def minus-ant-def inf-ant-def*,
cut-tac ant-inf-in-Ainteg,
cut-tac ant-minf-in-Ainteg,
simp add:*Abs-Ainteg-inverse*,
simp add:*ant-def*,
cut-tac ant-z-in-Ainteg[*of m*],
simp add:*Abs-Ainteg-inverse*)
apply (*simp add*:*zag-pl-def*)
done

lemma *a-zpm*[*simp*]:(*ant m*) + (−∞) = −∞
apply (*simp add*:*add-ant-def minus-ant-def inf-ant-def*,
cut-tac ant-inf-in-Ainteg,
cut-tac ant-minf-in-Ainteg,
simp add:*Abs-Ainteg-inverse*,
simp add:*ant-def*,
cut-tac ant-z-in-Ainteg[*of m*],
simp add:*Abs-Ainteg-inverse*)
apply (*simp add*:*zag-pl-def*)
done

lemma *a-mdi*[*simp*]:−∞ − ∞ = − ∞
apply (*simp add*:*diff-ant-def minus-ant-def inf-ant-def*,
cut-tac ant-inf-in-Ainteg,
simp add:*Abs-Ainteg-inverse*)
apply (*simp add*:*add-ant-def*,
cut-tac ant-minf-in-Ainteg,
simp add:*Abs-Ainteg-inverse*, *simp add*:*zag-pl-def*)
done

lemma *a-zdz*:(*ant m*) − (*ant n*) = *ant* (*m* − *n*)
apply (*simp add*:*diff-ant-def minus-ant-def ant-def*,
cut-tac ant-z-in-Ainteg[*of n*],
simp add:*Abs-Ainteg-inverse*)
apply (*simp add*:*add-ant-def*,
cut-tac ant-z-in-Ainteg[*of m*],
cut-tac ant-z-in-Ainteg[*of* −*n*],
simp add:*Abs-Ainteg-inverse zag-pl-def*)
done

lemma *a-i-i*[*simp*]:∞ ∗ ∞ = ∞
apply (*simp add*:*mult-ant-def inf-ant-def*,
cut-tac ant-inf-in-Ainteg,
simp add:*Abs-Ainteg-inverse*)
apply (*simp add*:*zag-t-def*)
done

lemma *a-0-i*[*simp*]:*0* ∗ ∞ = *0*
by (*simp add*:*mult-ant-def inf-ant-def Zero-ant-def*, *simp add*:*ant-def*,
cut-tac ant-inf-in-Ainteg, *cut-tac ant-0-in-Ainteg*,
simp add:*Abs-Ainteg-inverse*, *simp add*:*zag-t-def*)

lemma *a-i-0*[*simp*]:∞ ∗ *0* = *0*
by (*simp add*:*mult-ant-def inf-ant-def Zero-ant-def*, *simp add*:*ant-def*,
cut-tac ant-inf-in-Ainteg, *cut-tac ant-0-in-Ainteg*,
simp add:*Abs-Ainteg-inverse*, *simp add*:*zag-t-def*)

lemma *a-0-m*[*simp*]:*0* ∗ (−∞) = *0*
by (*simp add*:*mult-ant-def inf-ant-def Zero-ant-def*, *simp add*:*ant-def*,

cut-tac ant-inf-in-Ainteg, *cut-tac ant-0-in-Ainteg*,
simp add:*Abs-Ainteg-inverse*, *simp add*:*zag-t-def*)

lemma *a-m-0*[*simp*]:$(-\infty) * 0 = 0$
by (*simp add*:*mult-ant-def inf-ant-def Zero-ant-def*, *simp add*:*ant-def*,
cut-tac ant-inf-in-Ainteg, *cut-tac ant-0-in-Ainteg*,
simp add:*Abs-Ainteg-inverse*, *simp add*:*zag-t-def*)

lemma *a-m-i*[*simp*]:$(-\infty) * \infty = -\infty$
by (*simp add*:*mult-ant-def inf-ant-def minus-ant-def*,
cut-tac ant-inf-in-Ainteg, *cut-tac ant-minf-in-Ainteg*,
simp add:*Abs-Ainteg-inverse*, *simp add*:*zag-t-def*)

lemma *a-i-m*[*simp*]:$\infty * (-\infty) = -\infty$
by (*simp add*:*mult-ant-def inf-ant-def minus-ant-def*,
cut-tac ant-inf-in-Ainteg, *cut-tac ant-minf-in-Ainteg*,
simp add:*Abs-Ainteg-inverse*, *simp add*:*zag-t-def*)

lemma *a-pos-i*[*simp*]:$0 < m \Longrightarrow (ant\ m) * \infty = \infty$
apply (*simp add*:*mult-ant-def inf-ant-def ant-def*,
cut-tac ant-inf-in-Ainteg,
cut-tac ant-z-in-Ainteg[*of m*],
simp add:*Abs-Ainteg-inverse*)
apply (*simp add*:*zag-t-def*)
done

lemma *a-i-pos*[*simp*]:$0 < m \Longrightarrow \infty * (ant\ m) = \infty$
apply (*simp add*:*mult-ant-def inf-ant-def ant-def*,
cut-tac ant-inf-in-Ainteg,
cut-tac ant-z-in-Ainteg[*of m*],
simp add:*Abs-Ainteg-inverse*)
apply (*simp add*:*zag-t-def*)
done

lemma *a-neg-i*[*simp*]:$m < 0 \Longrightarrow (ant\ m) * \infty = -\infty$
apply (*simp add*:*mult-ant-def inf-ant-def ant-def*,
cut-tac ant-inf-in-Ainteg,
cut-tac ant-minf-in-Ainteg,
cut-tac ant-z-in-Ainteg[*of m*],
simp add:*minus-ant-def*,
simp add:*Abs-Ainteg-inverse*)
apply (*simp add*:*zag-t-def*)
done

lemma *a-i-neg*[*simp*]:$m < 0 \Longrightarrow \infty * (ant\ m) = -\infty$
apply (*simp add*:*mult-ant-def inf-ant-def ant-def*,
cut-tac ant-inf-in-Ainteg,
cut-tac ant-minf-in-Ainteg,
cut-tac ant-z-in-Ainteg[*of m*],

simp add:*minus-ant-def*,
simp add:*Abs-Ainteg-inverse*)
apply (*simp add*:*zag-t-def*)
done

lemma *a-z-z*:(*ant m*) ∗ (*ant n*) = *ant* (*m*∗*n*)
apply (*simp add*:*mult-ant-def ant-def*,
cut-tac ant-z-in-Ainteg[*of m*],
cut-tac ant-z-in-Ainteg[*of n*],
simp add:*Abs-Ainteg-inverse*)
apply (*simp add*:*zag-t-def*)
done

lemma *a-pos-m*[*simp*]:*0* < *m* ⟹ (*ant m*) ∗ (−∞) = −∞
apply (*simp add*:*mult-ant-def inf-ant-def minus-ant-def ant-def*,
cut-tac ant-inf-in-Ainteg,
cut-tac ant-minf-in-Ainteg,
cut-tac ant-z-in-Ainteg[*of m*],
simp add:*Abs-Ainteg-inverse*)
apply (*simp add*:*zag-t-def*)
done

lemma *a-m-pos*[*simp*]:*0* < *m* ⟹ (−∞) ∗ (*ant m*) = −∞
apply (*simp add*:*mult-ant-def inf-ant-def minus-ant-def ant-def*,
cut-tac ant-inf-in-Ainteg,
cut-tac ant-minf-in-Ainteg,
cut-tac ant-z-in-Ainteg[*of m*],
simp add:*Abs-Ainteg-inverse*)
apply (*simp add*:*zag-t-def*)
done

lemma *a-neg-m*[*simp*]:*m* < *0* ⟹ (*ant m*) ∗ (−∞) = ∞
apply (*simp add*:*mult-ant-def inf-ant-def minus-ant-def ant-def*,
cut-tac ant-inf-in-Ainteg,
cut-tac ant-minf-in-Ainteg,
cut-tac ant-z-in-Ainteg[*of m*],
simp add:*Abs-Ainteg-inverse*)
apply (*simp add*:*zag-t-def*)
done

lemma *neg-a-m*[*simp*]:*m* < *0* ⟹ (−∞) ∗ (*ant m*) = ∞
apply (*simp add*:*mult-ant-def inf-ant-def minus-ant-def ant-def*,
cut-tac ant-inf-in-Ainteg,
cut-tac ant-minf-in-Ainteg,
cut-tac ant-z-in-Ainteg[*of m*],
simp add:*Abs-Ainteg-inverse*)
apply (*simp add*:*zag-t-def*)
done

lemma *a-m-m*[*simp*]:$(-\infty) * (-\infty) = \infty$
apply (*simp add*:*mult-ant-def inf-ant-def minus-ant-def ant-def*,
cut-tac ant-inf-in-Ainteg,
cut-tac ant-minf-in-Ainteg,
simp add:*Abs-Ainteg-inverse*)
apply (*simp add*:*zag-t-def*)
done

lemma *inj-on-Abs-Ainteg*:*inj-on Abs-Ainteg Ainteg*
apply (*simp add*:*inj-on-def*)
apply (*rule ballI*)+
apply (*rule impI*,
subgoal-tac Rep-Ainteg (*Abs-Ainteg x*) = *Rep-Ainteg* (*Abs-Ainteg y*),
thin-tac Abs-Ainteg x = *Abs-Ainteg y*,
simp add:*Abs-Ainteg-inverse*, *simp*)
done

lemma *an-Suc*:*an* (*Suc n*) = *an n* + *1*
apply (*subst an-1*[*THEN sym*])
apply (*simp del*:*an-1 add*:*an-def*)
apply (*simp del*:*an-1 add*:*a-zpz*, *simp add*:*add.commute*)
done

lemma *aeq-zeq* [*iff*]: (*ant m* = *ant n*) = (*m* = *n*)
apply (*rule iffI*)
apply (*subgoal-tac Rep-Ainteg* (*ant m*) = *Rep-Ainteg* (*ant n*),
thin-tac ant m = *ant n*,
cut-tac ant-z-in-Ainteg[*of m*],
cut-tac ant-z-in-Ainteg[*of n*],
simp add:*ant-def Abs-Ainteg-inverse*)
apply *simp*+
done

lemma *aminus*:− *ant m* = *ant* (−*m*)
apply (*simp add*:*ant-def minus-ant-def*,
cut-tac ant-z-in-Ainteg[*of m*],
simp add:*Abs-Ainteg-inverse*)
done

lemma *aminusZero*:− *ant 0* = *ant 0*
apply (*simp add*:*aminus*)
done

lemma *ant-0*: *ant 0* = (*0*::*ant*)
by (*simp add*: *Zero-ant-def*)

lemma *inf-neq-0*[*simp*]:$\infty \neq 0$

apply (*cut-tac z-neq-inf*[*of 0*], *frule not-sym*)
apply (*simp add*:*ant-0*)
done

lemma *zero-neq-inf*[*simp*]:$0 \neq \infty$
by (*cut-tac inf-neq-0*, *frule not-sym*, *simp*)

lemma *minf-neq-0*[*simp*]:$-\infty \neq 0$
apply (*cut-tac z-neq-minf*[*of 0*], *frule not-sym*)
apply (*simp add*:*ant-0*)
done

lemma *zero-neq-minf*[*simp*]:$0 \neq -\infty$
by (*cut-tac minf-neq-0*, *frule not-sym*, *simp*)

lemma *a-minus-zero*[*simp*]:$-(0::ant) = 0$
by (*cut-tac aminusZero*, *simp add*:*ant-0*)

lemma *a-minus-minus*: $-(-z) = (z::ant)$
apply (*cut-tac mem-ant*[*of z*])
apply (*erule disjE*, *simp add*:*minf*, *simp add*: *minus-ant-def*,
cut-tac ant-minf-in-Ainteg,
cut-tac ant-inf-in-Ainteg,
simp add:*Abs-Ainteg-inverse*)
apply (*erule disjE*) **apply** (*erule exE*, *simp add*:*aminus*)
apply (*simp add*:*minf*, *simp add*: *minus-ant-def*,
cut-tac ant-minf-in-Ainteg,
cut-tac ant-inf-in-Ainteg,
simp add:*Abs-Ainteg-inverse*,
simp add:*inf-ant-def*)
done

lemma *aminus-0*: $-(-0) = (0::ant)$
apply (*simp add*:*a-minus-minus*)
done

lemma *a-a-z-0*:$[\![\ 0 < z;\ a * ant\ z = 0\ ]\!] \Longrightarrow a = 0$
by (*rule contrapos-pp*, *simp*+, *cut-tac mem-ant*[*of a*], *erule disjE*,
simp, *erule disjE*, *erule exE*, *simp add*:*a-z-z*,
simp only:*ant-0*[*THEN sym*], *simp*, *simp*)

lemma *adiv-eq*:$[\![\ z \neq 0;\ a * (ant\ z) = b * (ant\ z)\ ]\!] \Longrightarrow a = b$
apply (*cut-tac mem-ant*[*of a*], *cut-tac mem-ant*[*of b*],
(*erule disjE*)+, *simp*, *erule disjE*, *erule exE*,
cut-tac less-linear[*of z 0*], *erule disjE*, *simp add*:*a-z-z*,
frule sym, *thin-tac* $\infty = ant\ (za * z)$, *simp*,
simp add:*a-z-z*, *frule sym*, *thin-tac* $-\infty = ant\ (za * z)$, *simp*,
cut-tac less-linear[*of z 0*], *erule disjE*, *simp*,
simp, *erule disjE*, *erule exE*)

apply (*erule disjE*,
cut-tac less-linear[*of z 0*], *simp*,
erule disjE, *simp add*:*a-z-z*, *simp add*:*a-z-z*,
erule disjE, *erule exE*, *simp add*:*a-z-z*,
cut-tac less-linear[*of z 0*], *simp*,
erule disjE, *simp add*:*a-z-z*, *simp add*:*a-z-z*,
erule disjE,
cut-tac less-linear[*of z 0*], *simp*,
erule disjE, *simp*+)
apply (*erule disjE*, *erule exE*, *simp add*:*a-z-z*,
cut-tac less-linear[*of z 0*], *simp*, *erule disjE*, *simp*,
frule sym, *thin-tac* $-\infty = ant\ (za * z)$, *simp*,
simp, *frule sym*, *thin-tac* $\infty = ant\ (za * z)$, *simp*,
cut-tac less-linear[*of z 0*], *simp*)
done

lemma *aminus-add-distrib*: $-(z + w) = (-z) + (-w::ant)$
apply (*cut-tac mem-ant*[*of z*], *cut-tac mem-ant*[*of w*],
(*erule disjE*)+, *simp add*:*a-minus-minus*,
erule disjE, *erule exE*, *simp*,
simp add:*a-minus-minus aminus*, *simp add*:*a-minus-minus*)
apply ((*erule disjE*)+, *erule exE*,
simp add:*a-minus-minus*, *simp add*:*aminus*,
simp add:*a-minus-minus*)
apply ((*erule disjE*)+, (*erule exE*)+, *simp add*:*a-zpz aminus*,
erule exE, *simp add*:*aminus*,
erule disjE, *erule exE*, *simp add*:*aminus*, *simp*)
done

lemma *aadd-commute*:$(x::ant) + y = y + x$
apply (*cut-tac mem-ant*[*of x*], *cut-tac mem-ant*[*of y*])
apply (*erule disjE*, *erule disjE*, *simp*,
erule disjE, *erule exE*, *simp*+,
(*erule disjE*)+, *erule exE*, *simp*+)
apply ((*erule disjE*)+, (*erule exE*)+, *simp add*:*a-zpz*,
erule exE, *simp*, *erule disjE*, *erule exE*, *simp*+)
done

definition
aug-inf :: *ant set* (Z_∞) **where**
$Z_\infty = \{(z::ant).\ z \neq -\infty\ \}$

definition
aug-minf :: *ant set* $(Z_{-\infty})$ **where**
$Z_{-\infty} = \{(z::ant).\ z \neq \infty\ \}$

lemma *z-in-aug-inf*:*ant* $z \in Z_\infty$
apply (*simp add*:*aug-inf-def*)
done

lemma *Zero-in-aug-inf*:*0* $\in Z_{\infty}$
by (*simp only*:*Zero-ant-def*, *simp add*: *aug-inf-def*)

lemma *z-in-aug-minf*:*ant z* $\in Z_{-\infty}$
by (*simp add*:*aug-minf-def*)

lemma *mem-aug-minf*:*a* $\in Z_{-\infty} \Longrightarrow a = -\infty \vee (\exists z.\ a = ant\ z)$
by (*cut-tac mem-ant*[*of a*], *simp add*:*aug-minf-def*)

lemma *minus-an-in-aug-minf*: $-$ *an n* $\in\ Z_{-\infty}$
apply (*simp add*:*an-def*)
apply (*simp add*:*aminus*)
apply (*simp add*:*z-in-aug-minf*)
done

lemma *Zero-in-aug-minf*:*0* $\in Z_{-\infty}$
by (*simp add*:*Zero-ant-def aug-minf-def*)

lemma *aadd-assoc-i*: $[\![x \in Z_{\infty};\ y \in Z_{\infty};\ z \in Z_{\infty}]\!] \Longrightarrow (x + y) + z = x + (y + z)$
apply (*cut-tac mem-ant*[*of x*],
cut-tac mem-ant[*of y*],
cut-tac mem-ant[*of z*], *simp add*:*aug-inf-def*,
(*erule disjE*)+, (*erule exE*)+, (*simp add*:*a-zpz*)+,
(*erule exE*)+, *simp add*:*a-zpz*)
apply ((*erule disjE*)+, (*erule exE*)+, *simp*,
erule exE, *simp*,
(*erule disjE*)+, (*erule exE*)+, *simp add*:*a-zpz*,
erule exE, *simp*, *erule disjE*, *erule exE*, *simp*)
apply *simp*
done

lemma *aadd-assoc-m*: $[\![x \in Z_{-\infty};\ y \in Z_{-\infty};\ z \in Z_{-\infty}]\!] \Longrightarrow (x + y) + z = x + (y + z)$
apply (*cut-tac mem-ant*[*of x*],
cut-tac mem-ant[*of y*],
cut-tac mem-ant[*of z*], *simp add*:*aug-minf-def*)
apply ((*erule disjE*)+, *simp*, *erule exE*, *simp*,
erule disjE, *erule exE*, *simp*, (*erule exE*)+, *simp add*:*a-zpz*)
apply ((*erule disjE*)+, *erule exE*, *simp*, (*erule exE*)+, *simp*,
erule disjE, *erule exE*, *simp*, *erule exE*, *simp add*:*a-zpz*)
apply ((*erule exE*)+, *simp add*:*a-zpz*)
done

lemma *aadd-0-r*: $x + (0::ant) = x$
apply (*cut-tac mem-ant*[*of x*], *simp add*:*Zero-ant-def*)
apply ((*erule disjE*)+, *simp*)
apply (*erule disjE*, *erule exE*, *simp add*:*a-zpz*,

simp)
done

lemma *aadd-0-l*: $(0::ant) + x = x$
apply (*cut-tac mem-ant*[*of x*], *simp add*:*Zero-ant-def*)
apply ((*erule disjE*)+, *simp*)
apply (*erule disjE*, *erule exE*, *simp*, *simp add*:*a-zpz*, *simp*)
done

lemma *aadd-minus-inv*: $(-\ x) + x = (0::ant)$
apply (*cut-tac mem-ant*[*of x*],
erule disjE, *simp add*:*a-minus-minus*,
erule disjE, *erule exE*, *simp add*:*aminus*, *simp add*:*a-zpz*,
simp add:*Zero-ant-def*, *simp*)
done

lemma *aadd-minus-r*: $x + (-\ x) = (0::ant)$
apply (*cut-tac aadd-minus-inv*[*of x*])
apply (*simp add*:*aadd-commute*)
done

lemma *ant-minus-inj*:$ant\ z \neq ant\ w \Longrightarrow -\ ant\ z \neq -\ ant\ w$
by (*simp add*:*aminus*)

lemma *aminus-mult-minus*: $(-\ (ant\ z)) * (ant\ w) = -\ ((ant\ z) * (ant\ w))$
apply (*simp add*:*ant-def minus-ant-def*,
cut-tac ant-z-in-Ainteg[*of z*],
cut-tac ant-z-in-Ainteg[*of* $-z$],
cut-tac ant-z-in-Ainteg[*of w*],
simp add:*Abs-Ainteg-inverse*)
apply (*simp add*:*mult-ant-def*) **apply** (*simp add*:*Abs-Ainteg-inverse*,
simp add:*zag-t-def*,
cut-tac ant-z-in-Ainteg[*of* $z * w$])
apply (*simp add*:*Abs-Ainteg-inverse*)
done

lemma *amult-commute*: $(x::ant) * y = y * x$
apply (*cut-tac mem-ant*[*of x*],
cut-tac mem-ant[*of y*])
apply (*erule disjE*, *erule disjE*, *simp*)
apply (*erule disjE*, *erule exE*, *simp*)
apply (*cut-tac* $x = 0$ **and** $y = z$ **in** *less-linear*)
apply (*erule disjE*, *simp*)
apply (*erule disjE*, *rotate-tac* -1, *frule sym*, *thin-tac* $0 = z$, *simp*)
apply (*simp add*:*inf-ant-def ant-def*, *simp add*:*minus-ant-def*,
cut-tac ant-inf-in-Ainteg,
cut-tac ant-z-in-Ainteg[*of 0*],
cut-tac ant-z-in-Ainteg[*of* -1],
cut-tac ant-minf-in-Ainteg,

simp add:*Abs-Ainteg-inverse*)
apply (*simp add*:*mult-ant-def*, *simp add*:*Abs-Ainteg-inverse*,
simp add:*zag-t-def*, *simp*)
apply (*simp add*:*inf-ant-def*)
apply (*simp add*:*mult-ant-def minus-ant-def*,
cut-tac ant-inf-in-Ainteg,
simp add:*Abs-Ainteg-inverse*,
cut-tac ant-minf-in-Ainteg,
simp add:*Abs-Ainteg-inverse*, *simp add*:*zag-t-def*)
apply (*erule disjE*, *erule disjE*, *simp*)
apply (*erule exE*,
cut-tac $x = 0$ **and** $y = z$ **in** *less-linear*)
apply (*erule disjE*, *simp*)
apply (*erule disjE*, *rotate-tac* -1, *thin-tac* $0 = z$, *simp add*:*mult-ant-def*,
simp add:*ant-def inf-ant-def minus-ant-def*,
cut-tac ant-inf-in-Ainteg,
cut-tac $z = z$ **in** *ant-z-in-Ainteg*,
cut-tac ant-minf-in-Ainteg,
simp add:*Abs-Ainteg-inverse*, *simp add*:*zag-t-def*,
simp)
apply (*simp add*:*inf-ant-def minus-ant-def*,
cut-tac ant-inf-in-Ainteg,
cut-tac $z = z$ **in** *ant-z-in-Ainteg*,
cut-tac ant-minf-in-Ainteg,
simp add:*Abs-Ainteg-inverse*,
simp add:*mult-ant-def*,
simp add:*Abs-Ainteg-inverse*, *simp add*:*zag-t-def*)
apply ((*erule disjE*)+, (*erule exE*)+, *simp add*:*a-z-z*)
apply (*erule exE*,
cut-tac $x = 0$ **and** $y = z$ **in** *less-linear*,
erule disjE, *simp*)
apply (*erule disjE*, *rotate-tac* -1, *frule sym*, *thin-tac* $0 = z$, *simp*,
simp add:*mult-ant-def ant-def inf-ant-def*,
cut-tac ant-inf-in-Ainteg,
cut-tac ant-z-in-Ainteg[*of* 0],
simp add:*Abs-Ainteg-inverse*, *simp add*:*zag-t-def*,
simp)
apply (*erule disjE*, *erule exE*,
cut-tac $x = 0$ **and** $y = z$ **in** *less-linear*,
erule disjE, *simp*,
erule disjE, *rotate-tac* -1, *frule sym*, *thin-tac* $0 = z$, *simp*,
simp add:*mult-ant-def ant-def inf-ant-def*,
cut-tac ant-inf-in-Ainteg,
cut-tac ant-z-in-Ainteg[*of* 0],
simp add:*Abs-Ainteg-inverse*, *simp add*:*zag-t-def*,
simp+)
done

lemma *z-le-i*[*simp*]:$(ant\ x) \leq \infty$

apply (*simp add*:*le-ant-def ant-def*,
cut-tac ant-z-in-Ainteg[*of 0*],
cut-tac ant-z-in-Ainteg[*of x*],
cut-tac ant-inf-in-Ainteg,
simp add:*Abs-Ainteg-inverse*,
simp add:*inf-ant-def*,
simp add:*Abs-Ainteg-inverse*)
done

lemma *z-less-i*[*simp*]:$(ant\ x) < \infty$
apply (*cut-tac z-le-i*[*of x*],
cut-tac z-neq-inf[*of x*],
simp add:*less-ant-def*)
done

lemma *m-le-z*:$-\infty \leq (ant\ x)$
apply (*simp add*:*le-ant-def ant-def*,
cut-tac ant-z-in-Ainteg[*of 0*],
cut-tac ant-z-in-Ainteg[*of x*],
cut-tac ant-minf-in-Ainteg,
cut-tac ant-inf-in-Ainteg,
simp add:*Abs-Ainteg-inverse*,
simp add:*inf-ant-def*,
simp add:*minus-ant-def*,
simp add:*Abs-Ainteg-inverse*)
done

lemma *m-less-z*[*simp*]:$-\infty < (ant\ x)$
apply (*cut-tac m-le-z*[*of x*],
cut-tac z-neq-minf[*of x*],
frule not-sym, *thin-tac ant x* $\neq -\infty$,
simp add:*less-ant-def*)
done

lemma *noninf-mem-Z*:$[\![x \in Z_\infty;\ x \neq \infty]\!] \Longrightarrow \exists (z::int).\ x = ant\ z$
apply (*simp add*:*aug-inf-def*)
apply (*cut-tac mem-ant*[*of x*], *simp*)
done

lemma *z-mem-Z*:*ant z* $\in Z_\infty$
by (*simp add*:*aug-inf-def*)

lemma *inf-ge-any*[*simp*]:$x \leq \infty$
apply (*cut-tac mem-ant*[*of x*], *erule disjE*)
apply (*simp add*:*inf-ant-def minus-ant-def*,
cut-tac ant-minf-in-Ainteg,
cut-tac ant-inf-in-Ainteg,
simp add:*Abs-Ainteg-inverse*,
simp add:*le-ant-def*, *simp add*:*Abs-Ainteg-inverse*)

apply (*erule disjE*, *erule exE*, *simp*)
apply (*simp add*:*inf-ant-def*,
cut-tac ant-inf-in-Ainteg,
simp add:*le-ant-def*, *simp add*:*Abs-Ainteg-inverse*)
done

lemma *zero-lt-inf*:$0 < \infty$
by (*simp add*:*less-ant-def*)

lemma *minf-le-any*[*simp*]:$-\infty \leq x$
apply (*cut-tac mem-ant*[*of x*], *erule disjE*)
apply (*simp add*:*inf-ant-def minus-ant-def*,
cut-tac ant-minf-in-Ainteg,
cut-tac ant-inf-in-Ainteg,
simp add:*Abs-Ainteg-inverse*,
simp add:*le-ant-def*, *simp add*:*Abs-Ainteg-inverse*)
apply (*erule disjE*, *erule exE*, *simp*)
apply (*simp add*:*inf-ant-def*, *simp add*:*minus-ant-def*,
cut-tac ant-inf-in-Ainteg,
cut-tac ant-minf-in-Ainteg,
simp add:*le-ant-def*, *simp add*:*Abs-Ainteg-inverse*)
apply *simp*
done

lemma *minf-less-0*:$-\infty < 0$
by (*simp add*:*less-ant-def*)

lemma *ale-antisym*[*simp*]:$[\![(x::ant) \leq y;\ y \leq x\ ]\!] \Longrightarrow x = y$
apply (*rule contrapos-pp*, *simp*+)
apply (*cut-tac mem-ant*[*of x*], *cut-tac mem-ant*[*of y*])
apply (*erule disjE*, *erule disjE*, *simp*)
apply (*erule disjE*, *erule exE*, *simp*, *simp add*:*ant-def*,
simp add:*minus-ant-def inf-ant-def*,
cut-tac ant-inf-in-Ainteg,
cut-tac ant-minf-in-Ainteg,
cut-tac $z = z$ **in** *ant-z-in-Ainteg*, *simp add*:*Abs-Ainteg-inverse*,
simp add:*le-ant-def Abs-Ainteg-inverse*)
apply (*thin-tac* $x \leq y$,
simp add:*le-ant-def ant-def minus-ant-def inf-ant-def*,
cut-tac ant-inf-in-Ainteg,
cut-tac ant-minf-in-Ainteg,
simp add:*Abs-Ainteg-inverse*)
apply (*erule disjE*, *erule disjE*, *erule exE*,
thin-tac $y \leq x$,
simp add:*le-ant-def ant-def minus-ant-def inf-ant-def*,
cut-tac ant-inf-in-Ainteg,
cut-tac ant-minf-in-Ainteg,
cut-tac $z = z$ **in** *ant-z-in-Ainteg*, *simp add*:*Abs-Ainteg-inverse*)
apply (*thin-tac* $y \leq x$,

simp add:*le-ant-def ant-def minus-ant-def inf-ant-def*,
cut-tac ant-inf-in-Ainteg,
cut-tac ant-minf-in-Ainteg,
simp add:*Abs-Ainteg-inverse*)
apply ((*erule disjE*)+, (*erule exE*)+,
cut-tac $z = z$ **in** *ant-z-in-Ainteg*,
cut-tac $z = za$ **in** *ant-z-in-Ainteg*,
simp add:*le-ant-def ant-def*,
simp add:*Abs-Ainteg-inverse*)
apply (*erule exE*,
simp add:*le-ant-def ant-def inf-ant-def*,
cut-tac ant-inf-in-Ainteg,
cut-tac $z = z$ **in** *ant-z-in-Ainteg*, *simp add*:*Abs-Ainteg-inverse*)
apply (*erule disjE*, *erule exE*, *thin-tac* $y \leq x$,
simp add:*le-ant-def ant-def minus-ant-def inf-ant-def*,
cut-tac ant-inf-in-Ainteg,
cut-tac ant-minf-in-Ainteg,
cut-tac $z = z$ **in** *ant-z-in-Ainteg*, *simp add*:*Abs-Ainteg-inverse*)
apply *simp*
done

lemma *x-gt-inf*[*simp*]:$\infty \leq x \Longrightarrow x = \infty$
apply (*cut-tac inf-ge-any*[*of x*],
rule ale-antisym[*of* x ∞], *assumption*+)
done

lemma *Zinf-pOp-closed*:$[\![x \in Z_\infty;\ y \in Z_\infty]\!] \Longrightarrow x + y \in Z_\infty$
apply (*cut-tac mem-ant*[*of x*], *cut-tac mem-ant*[*of y*],
simp add:*aug-inf-def*,
(*erule disjE*)+, (*erule exE*)+, *simp add*:*a-zpz*,
cut-tac $z = -(z + za)$ **in** *z-neq-inf*,
rule contrapos-pp, *simp*+,
cut-tac m1 $= z+za$ **in** *aminus*[*THEN sym*], *simp add*:*a-minus-minus*,
erule exE, *simp*, *simp add*:*minf-neq-inf*[*THEN not-sym*],
erule disjE, *erule exE*, *simp*,
simp add:*minf-neq-inf*[*THEN not-sym*],
simp)
done

lemma *Zminf-pOp-closed*:$[\![x \in Z_{-\infty};\ y \in Z_{-\infty}]\!] \Longrightarrow x + y \in Z_{-\infty}$
apply (*cut-tac mem-ant*[*of x*], *cut-tac mem-ant*[*of y*],
simp add:*aug-minf-def*,
(*erule disjE*)+, *simp*, *erule exE*, *simp*,
erule disjE, *erule exE*, *simp*,
(*erule exE*)+, *simp add*:*a-zpz*)
done

lemma *amult-distrib1*:$(ant\ z) \neq 0 \Longrightarrow$
$(a + b) * (ant\ z) = a * (ant\ z) + b * (ant\ z)$

apply (*cut-tac mem-ant*[*of a*], *cut-tac mem-ant*[*of b*],
(*erule disjE*)+, *simp*, *cut-tac less-linear*[*of z 0*],
erule disjE, *simp*, *erule disjE*, *simp*, *simp add*:*ant-0*, *simp*,
erule disjE, *erule exE*, *simp*,
cut-tac less-linear[*of z 0*],
erule disjE, *simp add*:*a-z-z*, *erule disjE*, *simp add*:*ant-0*,
simp add:*a-z-z*,
cut-tac less-linear[*of z 0*], *simp*,
erule disjE, *simp add*:*ant-0*[*THEN sym*] *a-z-z*)
apply (*erule disjE*, *simp add*:*ant-0*[*THEN sym*],
simp, *simp add*:*ant-0*[*THEN sym*], *simp add*:*a-z-z*,
(*erule disjE*)+, (*erule exE*)+, *cut-tac less-linear*[*of z 0*], *simp*,
erule disjE, *simp add*:*a-z-z*,
erule disjE, *simp add*:*ant-0*, *simp add*:*a-z-z*,
cut-tac less-linear[*of z 0*],
erule disjE, *simp add*:*ant-0*[*THEN sym*])
apply (*simp add*:*a-z-z*, *simp*,
erule disjE, *simp add*:*ant-0*, *simp add*:*ant-0*[*THEN sym*] *a-z-z*,
(*erule disjE*)+, (*erule exE*)+, *simp add*:*a-zpz a-z-z*,
simp add: *distrib-right*, *erule exE*, *simp add*:*a-z-z*,
cut-tac less-linear[*of z 0*], *erule disjE*, *simp*,
erule disjE, *simp add*:*ant-0*, *simp*)
apply (*erule disjE*, *erule exE*, *simp*,
cut-tac less-linear[*of z 0*], *erule disjE*, *simp add*:*a-z-z*,
erule disjE, *simp add*:*ant-0*, *simp add*:*a-z-z*,
cut-tac less-linear[*of z 0*], *erule disjE*, *simp*,
erule disjE, *simp add*:*ant-0*, *simp*)
done

lemma *amult-0-r*:(*ant z*) $*$ *0* = *0*
by (*simp add*:*ant-0*[*THEN sym*] *a-z-z*)

lemma *amult-0-l*:*0* $*$ (*ant z*) = *0*
by (*simp add*:*ant-0*[*THEN sym*] *a-z-z*)

definition
asprod :: [*int*, *ant*] ⇒ *ant* (**infixl** $*_a$ *200*) **where**
m $*_a$ *x* ==
if x = ∞ *then* (*if 0* < *m then* ∞ *else* (*if m* < *0 then* −∞ *else*
if m = *0 then 0 else undefined*))
else (*if x* = −∞ *then*
(*if 0* < *m then* −∞ *else* (*if m* < *0 then* ∞ *else*
if m = *0 then 0 else undefined*))
else (*ant m*) $*$ *x*)

lemma *asprod-pos-inf*[*simp*]:*0* < *m* ⟹ *m* $*_a$ ∞ = ∞
apply (*simp add*:*asprod-def*)
done

lemma *asprod-neg-inf*[*simp*]:$m < 0 \Longrightarrow m *_a \infty = -\infty$
apply (*simp add*:*asprod-def*)
done

lemma *asprod-pos-minf*[*simp*]:$0 < m \Longrightarrow m *_a (-\infty) = (-\infty)$
apply (*simp add*:*asprod-def*)
done

lemma *asprod-neg-minf*[*simp*]:$m < 0 \Longrightarrow m *_a (-\infty) = \infty$
apply (*simp add*:*asprod-def*)
done

lemma *asprod-mult*: $m *_a (ant\ n) = ant(m * n)$
apply (*cut-tac z-neq-inf*[*of n*],
cut-tac z-neq-minf[*of n*],
simp add:*asprod-def*, *simp add*:*a-z-z*)
done

lemma *asprod-1*:$1 *_a x = x$
by (*cut-tac mem-ant*[*of x*], *erule disjE*, *simp*,
erule disjE, *erule exE*, *simp add*:*asprod-mult*, *simp*)

lemma *agsprod-assoc-a*:$m *_a (n *_a (ant\ x)) = (m * n) *_a (ant\ x)$
apply (*simp add*:*asprod-mult*)
done

lemma *agsprod-assoc*:$[\![m \neq 0; n \neq 0]\!] \Longrightarrow m *_a (n *_a x) = (m * n) *_a x$
apply (*cut-tac less-linear*[*of m 0*], *cut-tac less-linear*[*of n 0*],
cut-tac mem-ant[*of x*],
(*erule disjE*)+, *simp*,
frule zmult-neg-neg[*of m n*], *assumption*+, *simp*)
apply (*erule disjE*, *erule exE*, *simp add*:*asprod-mult*,
frule zmult-neg-neg[*of m n*], *assumption*+, *simp*+,
erule disjE, *simp*,
frule zmult-neg-pos[*of m n*], *assumption*+, *simp*,
erule disjE, *erule exE*, *simp*,
frule zmult-neg-pos[*of m n*], *assumption*+, *simp add*:*asprod-mult*,
frule zmult-neg-pos[*of m n*], *assumption*+, *simp*)
apply (*simp*, (*erule disjE*)+,
frule zmult-pos-neg[*of m n*], *assumption*+,
simp,
erule disjE, *erule exE*, *simp add*:*asprod-mult*,
frule zmult-pos-neg[*of m n*], *assumption*+, *simp*)
apply (*frule zmult-pos-pos*[*of m n*], *assumption*+,
erule disjE, *simp*,
erule disjE, *erule exE*, *simp add*:*asprod-mult*, *simp*)
done

lemma *asprod-distrib1*:$m \neq 0 \Longrightarrow m *_a (x + y) = (m *_a x) + (m *_a y)$
apply (*cut-tac mem-ant*[*of x*], *cut-tac mem-ant*[*of y*])
apply (*cut-tac less-linear*[*of m 0*],
erule disjE,
erule disjE, *erule disjE*, *simp*,
erule disjE, *simp add*:*asprod-def add-ant-def*, *simp*,
simp, (*erule disjE*)+, *erule exE*, *simp add*:*asprod-mult*,
simp add:*Zero-ant-def asprod-mult*)
apply (*erule disjE*, *erule exE*, *simp add*:*asprod-mult*,
simp add: *Zero-ant-def asprod-mult*,
erule disjE, *erule disjE*, *erule disjE*, *erule exE*,
simp add:*asprod-mult*,
simp add:*Zero-ant-def asprod-mult*,
erule disjE, *erule exE*, *simp add*:*asprod-mult*,
simp add:*Zero-ant-def asprod-mult*)
apply (*simp*, *erule disjE*, *erule exE*, *simp*,
(*erule disjE*)+, *erule exE*, *simp add*:*asprod-mult*,
simp add:*a-zpz*, *simp add*:*asprod-mult distrib-left*,
simp add:*asprod-mult*)
apply (*erule disjE*, *erule exE*, *simp add*:*a-zpz asprod-mult*,
simp add: *distrib-left*, *simp add*:*asprod-mult*,
(*erule disjE*)+, *erule exE*, *simp add*:*asprod-mult*, *simp*,
erule disjE, *erule exE*, *simp add*:*asprod-mult*, *simp*)
done

lemma *asprod-0-x*[*simp*]:$0 *_a x = 0$
apply (*simp add*:*asprod-def*, (*rule impI*)+,
cut-tac mem-ant[*of x*], *simp*, *erule exE*,
simp add:*asprod-def a-z-z*, *simp add*:*ant-0*)
done

lemma *asprod-n-0*:$n *_a 0 = 0$
apply (*simp add*:*Zero-ant-def asprod-mult*)
done

lemma *asprod-distrib2*:$[\![0 < i; 0 < j]\!] \Longrightarrow (i + j) *_a x = (i *_a x) + (j *_a x)$
by (*cut-tac mem-ant*[*of x*], *erule disjE*, *simp*,
erule disjE, *erule exE*, *simp add*:*asprod-mult*,
simp add: *distrib-right a-zpz*, *simp*)

lemma *asprod-minus*:$x \neq -\infty \wedge x \neq \infty \Longrightarrow - z *_a x = z *_a (- x)$
apply (*cut-tac mem-ant*[*of x*], *erule disjE*, *simp*+)
apply (*erule exE*, *simp add*:*asprod-mult aminus*)
done

lemma *asprod-div-eq*:$[\![n \neq 0; n *_a x = n *_a y]\!] \Longrightarrow x = y$
apply (*cut-tac less-linear*[*of n 0*], *simp*)
apply (*cut-tac mem-ant*[*of x*], *cut-tac mem-ant*[*of y*])

apply ((*erule disjE*)+, *simp*,
erule disjE, *erule exE*, *rule contrapos-pp*, *simp*+,
simp add:*asprod-mult*)
apply (*cut-tac z1* = $n * z$ **in** *z-neq-inf*[*THEN not-sym*], *simp*+)
apply ((*erule disjE*)+, *erule exE*, *simp add*:*asprod-mult*,
cut-tac z = $n * z$ **in** *z-neq-inf*,
rule contrapos-pp, *simp*, *simp*,
(*erule disjE*)+, (*erule exE*)+, *simp add*:*asprod-mult*,
erule exE, *simp add*: *asprod-mult*)
apply (*erule disjE*, *erule exE*, *simp add*:*asprod-mult*,
simp add:*z-neq-minf*[*THEN not-sym*], *simp*)
apply ((*erule disjE*)+, *simp*,
erule disjE, *erule exE*, *rule contrapos-pp*, *simp*+,
simp add:*asprod-mult*,
cut-tac z1 = $n * z$ **in** *z-neq-minf*[*THEN not-sym*], *simp*,
rule contrapos-pp, *simp*+)
apply ((*erule disjE*)+, (*erule exE*)+, *simp add*:*asprod-mult*,
erule exE, *simp add*:*asprod-mult*,
erule disjE, *erule exE*, *simp add*:*asprod-mult*
z-neq-inf[*THEN not-sym*], *simp*)
apply (*erule disjE*, *simp*, *erule disjE*, *erule exE*, *simp add*:*asprod-mult*
z-neq-inf[*THEN not-sym*], *simp*)
done

lemma *asprod-0*:$[\![z \neq 0;\ z *_a x = 0\ ]\!] \Longrightarrow x = 0$
by (*rule asprod-div-eq*[*of z x 0*], *assumption*, *simp add*:*asprod-n-0*)

lemma *asp-z-Z*:$z *_a ant\ x \in Z_\infty$
by (*simp add*:*asprod-mult z-in-aug-inf*)

lemma *tna-ant*: *tna* (*ant z*) = *z*
apply (*cut-tac z-neq-minf*[*of z*], *cut-tac z-neq-inf*[*of z*],
simp add:*ant-def tna-def*)
apply (*cut-tac ant-z-in-Ainteg*[*of z*], *simp add*:*Abs-Ainteg-inverse*)
done

lemma *ant-tna*:$x \neq \infty \wedge x \neq -\infty \Longrightarrow$ *ant* (*tna x*) = *x*
apply (*cut-tac mem-ant*[*of x*], *simp*, *erule exE*)
apply (*simp add*:*ant-def tna-def*)
apply (*cut-tac z* = *z* **in** *ant-z-in-Ainteg*, *simp add*:*Abs-Ainteg-inverse*)
done

lemma *ant-sol*:$[\![a \in Z_\infty;\ b \in Z_\infty;\ c \in Z_\infty;\ b \neq \infty;\ a = b + c]\!] \Longrightarrow a - b = c$
apply (*subgoal-tac* $-b \in Z_\infty$, *simp add*:*diff-ant-def*,
subgoal-tac a + $(-b)$ = *b* + *c* + $(-b)$,
subst aadd-commute[*of b c*], *subst aadd-assoc-i*, *assumption*+,
simp add:*aadd-minus-r*, *simp add*:*aadd-0-r*, *simp*)
apply (*cut-tac mem-ant*[*of b*], *simp add*:*aug-inf-def*,
erule exE, *simp add*:*aminus*)

done

1.7.1 Ordering of integers and ordering nats

1.7.2 The $\leq$ Ordering

lemma *zneq-aneq*:$(n \neq m) = ((ant\ n) \neq (ant\ m))$
apply (*rule iffI*)
apply (*rule contrapos-pp*, *simp*+)
done

lemma *ale*:$(n \leq m) = ((ant\ n) \leq (ant\ m))$
apply (*rule iffI*)
apply (*simp add*:*ant-def le-ant-def*,
cut-tac ant-z-in-Ainteg[*of n*],
cut-tac ant-z-in-Ainteg[*of m*],
simp add:*Abs-Ainteg-inverse*)+
done

lemma *aless*:$(n < m) = ((ant\ n) < (ant\ m))$
apply (*simp add*:*less-ant-def*,
cut-tac ale[*of n m*], *arith*)
done

lemma *ale-refl*: $w \leq (w::ant)$
apply (*cut-tac mem-ant*[*of w*],
erule disjE, *simp*,
erule disjE, *erule exE*, *simp*,
subst ale[*THEN sym*], *simp*+)
done

lemma *aeq-ale*:$(a::ant) = b \Longrightarrow a \leq b$
by (*simp add*:*ale-refl*)

lemma *ale-trans*: $[\![\ (i::ant) \leq j;\ j \leq k\ ]\!] \Longrightarrow i \leq k$
apply (*cut-tac mem-ant*[*of i*], *cut-tac mem-ant*[*of j*],
cut-tac mem-ant[*of k*],
(*erule disjE*)+, *simp add*:*ale-refl*, *erule disjE*, *erule exE*, *simp*+,
(*erule disjE*)+, *simp add*:*ale-refl*, *simp add*:*ale-refl*)
apply ((*erule disjE*)+, *erule exE*, *simp*+,
erule exE, *simp*,
cut-tac $x = ant\ z$ **in** *minf-le-any*,
frule-tac $x = ant\ z$ **in** *ale-antisym*[*of* - $-\infty$], *assumption*+,
simp+,
cut-tac minf-le-any[*of* ∞], *frule ale-antisym*[*of* $-\infty$ ∞],
simp+)
apply (*erule disjE*, *simp*,
(*erule disjE*)+, (*erule exE*)+, *simp*,
cut-tac $x = ant\ za$ **in** *minf-le-any*,
frule-tac $x = ant\ za$ **in** *ale-antisym*[*of* - $-\infty$], *assumption*+,

simp, *erule exE*,
cut-tac x = ant z **in** *minf-le-any*, *simp*)
apply (*cut-tac minf-le-any*[*of* ∞],
frule-tac ale-antisym[*of* $-\infty$ ∞], *assumption*+,
simp, *erule disjE*, *erule exE*, *simp*,
cut-tac x = ant z **in** *inf-ge-any*,
frule-tac x = ant z **in** *ale-antisym*[*of* - ∞], *assumption*+,
simp)
apply (*cut-tac minf-le-any*[*of* ∞], *frule ale-antisym*[*of* $-\infty$ ∞],
simp+,
(*erule disjE*)+, (*erule exE*)+, *simp add*:*ale*[*THEN sym*],
simp, (*erule disjE*)+, (*erule exE*)+,
cut-tac x = ant za **in** *inf-ge-any*,
frule-tac x = ant za **in** *ale-antisym*[*of* - ∞],
simp+)
apply (*erule disjE*, *erule exE*,
cut-tac inf-ge-any[*of j*],
frule ale-antisym[*of j* ∞], *assumption*+,
cut-tac x = ant z **in** *inf-ge-any*, *simp*+)
done

lemma *aless-le-not-le*: $((w::ant) < z) = (w \leq z \wedge \neg\, z \leq w)$
by (*auto simp add*: *less-ant-def*)

instance *ant* :: *order*
proof qed
(*assumption* |
rule ale-refl ale-trans ale-antisym aless-le-not-le)+

lemma *ale-linear*: $(z::ant) \leq w \vee w \leq z$
apply (*cut-tac mem-ant*[*of z*], *cut-tac mem-ant*[*of w*],
erule disjE, *simp*,
erule disjE, *simp*)
apply ((*erule disjE*)+, (*erule exE*)+, *simp add*:*ale*[*THEN sym*],
simp add:*linorder-linear*)
apply *simp*+
done

instance *ant* :: *linorder*
proof qed (*rule ale-linear*)

lemmas *aless-linear* = *less-linear* [**where** $'a = ant$]

lemma *ant-eq-0-conv* [*simp*]: (*ant n = 0*) = (*n = 0*)
apply (*simp add*:*Zero-ant-def*)
done

lemma *aless-zless*: $(ant\ m < ant\ n) = (m<n)$
by (*simp add*: *ale ant-def linorder-not-le* [*symmetric*])

lemma *a0-less-int-conv* [*simp*]: $(0 < ant\ n) = (0 < n)$
apply (*simp add*:*Zero-ant-def*)
apply (*simp add*:*aless*[*THEN sym*])
done

lemma *a0-less-1*: $0 < (1::ant)$
apply (*simp add*:*Zero-ant-def One-ant-def*)
apply (*subst aless-zless*) **apply** *simp*
done

lemma *a0-neq-1* [*simp*]: $0 \neq (1::ant)$
by (*simp only*:*Zero-ant-def One-ant-def*, *subst zneq-aneq*[*THEN sym*], *simp*)

lemma *ale-zle* [*simp*]: $((ant\ i) \leq (ant\ j)) = (i \leq j)$
by (*subst ale*[*of i j*], *simp*)

lemma *ant-1* [*simp*]: $ant\ 1 = 1$
by (*simp add*: *One-ant-def*)

lemma *zpos-apos*:$(0 \leq n) = (0 \leq (ant\ n))$
apply (*simp only*:*ale*[*of 0 n*], *simp only*:*ant-0*[*THEN sym*])
done

lemma *zposs-aposss*:$(0 < n) = (0 < (ant\ n))$
apply (*rule iffI*)
apply (*unfold Zero-ant-def*,
subst aless[*THEN sym*, *of 0 n*], *simp*,
subst aless[*of 0 n*], *simp*)
done

lemma *an-nat-pos*[*simp*]:$0 \leq an\ n$
by (*simp add*:*ant-0*[*THEN sym*] *an-def*)

lemma *amult-one-l*: $1 * (x::ant) = x$
by (*cut-tac mem-ant*[*of x*], *erule disjE*, *simp*
only:*ant-1*[*THEN sym*], *simp del*:*ant-1*,
erule disjE, *erule exE*, *simp only*:*ant-1*[*THEN sym*],
simp del:*ant-1 add*:*a-z-z*,
simp only:*ant-1*[*THEN sym*], *simp del*:*ant-1*)

lemma *amult-one-r*:$(x::ant) * 1 = x$
by (*cut-tac amult-one-l*[*of x*], *simp add*:*amult-commute*)

lemma *amult-eq-eq-r*:$[\![z \neq 0;\ \ a * ant\ z = b * ant\ z]\!] \Longrightarrow a = b$
apply (*cut-tac less-linear*[*of z 0*], *simp*,

cut-tac mem-ant[*of a*], *cut-tac mem-ant*[*of b*],
(*erule disjE*)+, *simp*,
erule disjE, *erule exE*, *simp add*:*a-z-z*,
frule sym, *thin-tac* $\infty = ant\ (za * z)$, *simp*,
simp, (*erule disjE*)+, *simp*, *erule exE*, *simp add*:*a-z-z*, *simp*)
apply ((*erule disjE*)+, (*erule exE*)+, *simp add*:*a-z-z*,
erule exE, *simp add*:*a-z-z*, *erule disjE*, *erule exE*,
simp add:*a-z-z*,
frule sym, *thin-tac* $-\infty = ant\ (za * z)$, *simp*, *simp*,
(*erule disjE*)+, *simp*, *erule disjE*, *erule exE*, *simp add*:*a-z-z*,
frule sym, *thin-tac* $-\infty = ant\ (za * z)$, *simp*, *simp*)
apply ((*erule disjE*)+, *erule exE*, *simp add*:*a-z-z*, *simp*,
(*erule disjE*)+, (*erule exE*)+, *simp add*:*a-z-z*,
erule exE, *simp add*:*a-z-z*, *erule disjE*, *erule exE*, *simp add*:*a-z-z*,
frule sym, *thin-tac* $\infty = ant\ (za * z)$, *simp*, *simp*)
done

lemma *amult-eq-eq-l*:$[\![z \neq 0;\ (ant\ z) * a = (ant\ z) * b]\!] \Longrightarrow a = b$
by (*simp add*:*amult-commute*, *rule amult-eq-eq-r*, *assumption*+)

lemma *amult-pos*:$[\![0 < b;\ 0 \leq x]\!] \Longrightarrow x \leq (b *_a x)$
apply (*cut-tac mem-ant*[*of x*], *erule disjE*, *simp*,
erule disjE, *erule exE*, *simp add*:*asprod-mult*,
simp add:*zpos-apos*[*THEN sym*],
frule-tac $a = z$ **and** $b = b$ **in** *pos-zmult-pos*, *assumption*+,
simp add:*mult.commute*, *simp*)
done

lemma *asprod-amult*:$0 < z \Longrightarrow z *_a x = (ant\ z) * x$
apply (*simp add*:*asprod-def*)
done

lemma *amult-pos1*:$[\![0 < b;\ 0 \leq x]\!] \Longrightarrow x \leq ((ant\ b) * x)$
by (*frule amult-pos*[*of b x*], *assumption*, *simp add*:*asprod-amult*)

lemma *amult-pos-mono-l*:$0 < w \Longrightarrow (((ant\ w) * x) \leq ((ant\ w) * y)) = (x \leq y)$
apply (*cut-tac mem-ant*[*of x*], *cut-tac mem-ant*[*of y*],
(*erule disjE*)+, *simp*, *erule disjE*, *erule exE*, *simp*, *simp*,
(*erule disjE*)+, *erule exE*, *simp add*:*a-z-z*)
apply (*rule iffI*,
cut-tac $x = ant\ (w * z)$ **in** *minf-le-any*, *frule-tac* $x = ant\ (w * z)$
in *ale-antisym*, *assumption*+, *simp*,
cut-tac $x = ant\ z$ **in** *minf-le-any*, *frule-tac* $x = ant\ z$
in *ale-antisym*, *assumption*+, *simp*)
apply *simp*
apply ((*erule disjE*)+, (*erule exE*)+, *simp add*:*a-z-z*)
apply (*erule exE*, *simp add*:*a-z-z*)
apply (*erule disjE*, *erule exE*, *simp add*:*a-z-z*,
rule iffI,

cut-tac $x = ant\ (w * z)$ **in** *inf-ge-any*,
frule-tac $x = ant\ (w * z)$ **in** *ale-antisym*[*of* - ∞], *assumption*+,
simp,
cut-tac $x = ant\ z$ **in** *inf-ge-any*,
frule-tac $x = ant\ z$ **in** *ale-antisym*[*of* - ∞], *assumption*+,
simp, *simp*)
done

lemma *amult-pos-mono-r*:$0 < w \Longrightarrow ((x * (ant\ w)) \leq (y * (ant\ w))) = (x \leq y)$
apply (*simp add*:*amult-commute*[*of* - *ant w*])
apply (*rule amult-pos-mono-l*, *assumption*)
done

lemma *apos-neq-minf*:$0 \leq a \Longrightarrow a \neq -\infty$
by (*rule contrapos-pp*, *simp*+,
cut-tac minf-le-any[*of 0*],
frule ale-antisym[*of 0* $-\infty$], *assumption*+, *simp*)

lemma *asprod-pos-mono*:$0 < w \Longrightarrow ((w *_a x) \leq (w *_a y)) = (x \leq y)$
by (*simp add*:*asprod-amult*, *simp add*:*amult-pos-mono-l*)

lemma *a-inv*:$(a::ant) + b = 0 \Longrightarrow a = -b$
apply (*cut-tac mem-ant*[*of a*], *cut-tac mem-ant*[*of b*],
(*erule disjE*)+, *frule sym*, *thin-tac* $a + b = 0$,
simp add:*ant-0*[*THEN sym*])
apply (*erule disjE*, *erule exE*, *simp*, *simp*,
(*erule disjE*)+, *erule exE*, *simp*, *simp*,
simp add:*a-minus-minus*,
(*erule disjE*)+, (*erule exE*)+, *simp add*:*aminus a-zpz*,
erule exE, *simp*,
erule disjE, *erule exE*, *simp*, *simp*)
done

lemma *asprod-pos-pos*:$0 \leq x \Longrightarrow 0 \leq int\ n *_a x$
apply (*cases* $n = 0$)
apply *simp-all*
using *asprod-pos-mono* [*THEN sym*, *of int n 0 x*]
apply (*simp add*:*asprod-n-0*)
done

lemma *asprod-1-x*[*simp*]:$1 *_a x = x$
apply (*simp add*:*asprod-def*)
apply (*rule impI*)+
apply (*cut-tac mem-ant*[*of x*], *simp*, *erule exE*, *simp add*:*a-z-z*)
apply (*simp only*:*ant-1*[*THEN sym*], *simp del*:*ant-1 add*:*a-z-z*)
done

lemma *asprod-n-1*[*simp*]:$n *_a 1 = ant\ n$
apply (*simp only*:*ant-1*[*THEN sym*]) **apply** (*simp only*:*asprod-mult*)

apply *simp*
done

1.7.3 Aug ordering

lemma *aless-imp-le*: $x < (y::ant) \Longrightarrow x \leq y$
by (*simp add*:*less-ant-def*)

lemma *gt-a0-ge-1*:$(0::ant) < x \Longrightarrow 1 \leq x$
apply (*cut-tac mem-ant*[*of x*],
erule disjE, *unfold Zero-ant-def*, *simp*)
apply (*cut-tac less-ant-def*[*of 0* $-\infty$], *simp add*:*ant-0*,
cut-tac minf-le-any[*of 0*],
frule ale-antisym[*of 0* $-\infty$], *assumption*+,
simp add:*ant-0*[*THEN sym*], *blast*)
apply (*erule disjE*, *erule exE*, *unfold One-ant-def*, *simp del*:*ant-1*,
simp add:*aless-zless*, *simp*)
done

lemma *gt-a0-ge-aN*:$[\![0 < x;\ N \neq 0]\!] \Longrightarrow (ant\ (int\ N)) \leq (int\ N) *_a x$
apply (*cut-tac mem-ant*[*of x*], *erule disjE*, *simp*)
apply (*cut-tac aless-imp-le*[*of 0* $-\infty$],
cut-tac minf-le-any[*of 0*],
frule ale-antisym[*of 0* $-\infty$], *simp*,
simp only: *Zero-ant-def*, *simp*)
apply (*erule disjE*, *erule exE*, *simp add*:*asprod-mult*, *simp*)
done

lemma *aless-le-trans*:$[\![(x::ant) < y;\ y \leq z]\!] \Longrightarrow x < z$
by *auto*

lemma *ale-less-trans*:$[\![(x::ant) \leq y;\ y < z]\!] \Longrightarrow x < z$
by *auto*

lemma *aless-trans*:$[\![(x::ant) < y;\ y < z]\!] \Longrightarrow x < z$
by *auto*

lemma *ale-neq-less*:$[\![\ (x::ant) \leq y;\ x \neq y]\!] \Longrightarrow x < y$
apply (*simp add*:*less-ant-def*)
done

lemma *aneg-le*:$(\neg\ (x::ant) \leq y) = (y\ <\ x)$
apply (*cut-tac ale-linear*[*of y x*])
apply (*rule iffI*, *simp*)
apply (*rule contrapos-pp*, *simp*+)
done

lemma *aneg-less*:$(\neg\ x < (y::ant)) = (y \leq x)$
by *auto*

lemma *aadd-le-mono*:$x \leq (y::ant) \Longrightarrow x + z \leq y + z$
apply (*cut-tac mem-ant*[*of x*], *cut-tac mem-ant*[*of y*],
cut-tac mem-ant[*of z*],
(*erule disjE*)+, *simp*, *erule disjE*, *erule exE*, *simp*+,
(*erule disjE*)+, *erule exE*, *simp*+,
(*erule disjE*)+, (*erule exE*)+, *simp*, *erule exE*, *simp*,
erule disjE, *erule exE*, *simp*+, (*erule disjE*)+, *simp*,
erule exE, *simp*+,
cut-tac minf-le-any[*of* ∞], *frule ale-antisym*[*of* $-\infty$ ∞],
assumption+, *simp*, (*erule disjE*)+, (*erule exE*)+, *simp*+,
cut-tac x = *ant za* **in** *minf-le-any*,
frule-tac x = *ant za* **in** *ale-antisym*[*of* - $-\infty$], *assumption*+, *simp*)
apply (*erule exE*, *simp*,
cut-tac x = *ant za* **in** *minf-le-any*,
frule-tac x = *ant za* **in** *ale-antisym*[*of* - $-\infty$], *assumption*+, *simp*,
erule disjE, *erule exE*, *simp*+,
cut-tac minf-le-any[*of* ∞], *frule ale-antisym*[*of* $-\infty$ ∞],
assumption+, *simp*, (*erule disjE*)+, (*erule exE*)+, *simp*+,
erule exE, *simp*, *erule disjE*, *erule exE*, *simp*+)
apply (*cut-tac x* = *ant za* **in** *inf-ge-any*, *frule-tac x* = *ant za* **in**
ale-antisym[*of* - ∞], *assumption*+, *simp*+,
(*erule disjE*)+, (*erule exE*)+, *simp add*:*a-zpz*,
(*erule exE*)+, *simp add*:*a-zpz*, (*erule disjE*)+, (*erule exE*)+,
simp add:*a-zpz*, *erule exE*, *simp*,
(*erule disjE*)+, (*erule exE*)+, *simp add*:*a-zpz*)
apply (*cut-tac x* = *ant za* **in** *inf-ge-any*, *frule-tac x* = *ant za* **in**
ale-antisym[*of* - ∞], *assumption*+, *simp*+,
erule exE, *simp*, *erule disjE*, *erule exE*, *simp*+)
done

lemma *aadd-less-mono-z*:$(x::ant) < y \Longrightarrow (x + (ant\ z)) < (y + (ant\ z))$
apply (*simp add*:*less-ant-def*, *simp add*:*aadd-le-mono*)
apply (*cut-tac mem-ant*[*of x*], *cut-tac mem-ant*[*of y*])
apply *auto*
apply (*metis a-inv a-ipi a-ipz a-zpz aadd-minus-r less-le diff-ant-def minf-less-0*)
apply (*metis a-inv a-ipi a-ipz a-zpz aadd-minus-r less-le diff-ant-def minf-less-0*)
apply (*metis a-zpz add-right-cancel aeq-zeq*)
apply (*metis a-zpz less-le z-less-i*)
done

lemma *aless-le-suc*[*simp*]:$(a::ant) < b \Longrightarrow a + 1 \leq b$
apply (*cut-tac mem-ant*[*of b*])
apply (*erule disjE*,
frule aless-imp-le[*of a b*], *simp*,
cut-tac minf-le-any[*of a*], *frule ale-antisym*[*of a* $-\infty$],
assumption, *simp*)
apply (*erule disjE*, *erule exE*, *cut-tac mem-ant*[*of a*], *erule disjE*,
unfold One-ant-def, *simp del*:*ant-1*,

erule disjE, *erule exE*, *simp del*:*ant-1 add*:*a-zpz*, *simp only*:*aless-zless*,
frule aless-imp-le[*of a b*], *simp del*:*ant-1*, *simp*)
done

lemma *aposs-le-1*:$(0::ant) < x \Longrightarrow 1 \leq x$
apply (*frule aless-le-suc*[*of 0 x*],
simp add:*aadd-0-l*)
done

lemma *pos-in-aug-inf*:$(0::ant) \leq x \Longrightarrow x \in Z_\infty$
apply (*simp add*:*aug-inf-def*)
apply (*rule contrapos-pp*, *simp*+)
apply (*cut-tac minf-le-any*[*of 0*],
frule ale-antisym[*of 0* $-\infty$], *assumption*+,
unfold Zero-ant-def,
simp)
done

lemma *aug-inf-noninf-is-z*:$[\![x \in Z_\infty;\ x \neq \infty]\!] \Longrightarrow \exists z.\ x = ant\ z$
apply (*cut-tac mem-ant*[*of x*], *simp add*:*aug-inf-def*)
done

lemma *aadd-two-pos*:$[\![0 \leq (x::ant);\ 0 \leq y]\!] \Longrightarrow 0 \leq x + y$
apply (*cut-tac Zero-in-aug-inf*,
cut-tac pos-in-aug-inf[*of x*],
cut-tac pos-in-aug-inf[*of y*])
apply (*cut-tac aadd-le-mono*[*of 0 x y*], *simp add*:*aadd-0-l*,
assumption+)
done

lemma *aadd-pos-poss*:$[\![(0::ant) \leq x;\ 0 < y]\!] \Longrightarrow 0 < (x + y)$
apply (*frule aless-imp-le*[*of 0 y*],
subst less-ant-def, *rule conjI*, *simp add*:*aadd-two-pos*,
rule contrapos-pp, *simp*+)
apply (*cut-tac Zero-in-aug-inf*,
cut-tac pos-in-aug-inf[*of x*],
cut-tac pos-in-aug-inf[*of y*],
case-tac $y = \infty$, *simp*,
cut-tac mem-ant[*of x*], *erule disjE*,
simp add:*aug-inf-def*)
apply (*erule disjE*, *erule exE*, *simp*, *simp*,
case-tac $x = \infty$, *unfold Zero-ant-def*,
frule aug-inf-noninf-is-z[*of y*], *assumption*, *erule exE*,
simp, *frule sym*, *thin-tac* $\infty = ant\ 0$, *simp*)
apply (*thin-tac ant* $0 \leq y$,
frule aug-inf-noninf-is-z[*of x*], *assumption*, *erule exE*,
frule aug-inf-noninf-is-z[*of y*], *assumption*, *erule exE*,
simp add:*a-zpz*, *simp add*: *aless-zless*)
apply (*simp add*:*aless-imp-le*)+

done

lemma *aadd-poss-pos*:$[\![(0::ant) < x;\ 0 \leq y]\!] \Longrightarrow 0 < (x + y)$
apply (*subst aadd-commute*, *rule aadd-pos-poss*, *assumption*+)
done

lemma *aadd-pos-le*:$0 \leq (a::ant) \Longrightarrow b \leq a + b$
apply (*cut-tac mem-ant*[*of a*], (*erule disjE*)+,
simp, *cut-tac minf-le-any*[*of 0*], *frule ale-antisym*[*of 0* $-\infty$],
assumption+, *simp*)
apply (*erule disjE*, *erule exE*,
simp, *thin-tac* $a = ant\ z$, *cut-tac mem-ant*[*of b*],
erule disjE, *simp*,
erule disjE, *erule exE*, *simp add*:*a-zpz*, *simp only*:*ant-0*[*THEN sym*],
simp only:*ale*, *simp*+)
apply (*cut-tac mem-ant*[*of b*],
erule disjE, *simp*,
erule disjE, *erule exE*, *simp*, *simp*)
done

lemma *aadd-poss-less*:$[\![b \neq \infty;\ b \neq -\infty;\ 0 < a]\!] \Longrightarrow b < a + b$
apply (*cut-tac mem-ant*[*of b*], *simp*)
apply (*erule exE*,
cut-tac mem-ant[*of a*], *erule disjE*, *simp*,
thin-tac $a = -\infty$,
cut-tac minf-le-any[*of 0*],
frule aless-imp-le[*of 0* $-\infty$],
frule ale-antisym[*of 0* $-\infty$], *assumption*+,
simp only:*ant-0*[*THEN sym*], *simp*)
apply (*erule disjE*, *erule exE*, *simp add*:*a-zpz*,
subst aless[*THEN sym*], *simp*, *simp*)
done

lemma *ale-neg*:$(0::ant) \leq x \Longrightarrow (-x) \leq 0$
apply (*frule pos-in-aug-inf*[*of x*])
apply (*case-tac* $x = \infty$, *simp*,
frule aug-inf-noninf-is-z[*of x*], *assumption*, *erule exE*,
simp add:*aminus*, *unfold Zero-ant-def*,
simp only:*ale-zle*)
done

lemma *ale-diff-pos*:$(x::ant) \leq y \Longrightarrow 0 \leq (y - x)$
apply (*case-tac* $y = -\infty$, *simp*,
cut-tac minf-le-any[*of x*],
frule ale-antisym[*of x* $-\infty$], *assumption*+,
simp add:*diff-ant-def a-minus-minus*,
cut-tac mem-ant[*of y*], *simp*, *thin-tac* $y \neq -\infty$,
erule disjE, *erule exE*)
apply (*case-tac* $x = \infty$, *simp*,

cut-tac $x = ant\ z$ **in** *inf-ge-any*,
frule-tac $x = ant\ z$ **in** *ale-antisym*[*of* - ∞], *simp*+,
cut-tac mem-ant[*of* x], *simp*+, *erule disjE*,
simp add:*diff-ant-def a-minus-minus*)
apply (*erule exE*, *simp add*:*a-zdz*, *unfold Zero-ant-def*,
simp only:*ale-zle*,
cut-tac mem-ant[*of* x], *erule disjE*,
simp add:*diff-ant-def a-minus-minus*,
erule disjE, *erule exE*, *simp add*:*diff-ant-def aminus*,
simp add:*diff-ant-def ant-0*)
done

lemma *aless-diff-poss*:$(x::ant) < y \Longrightarrow 0 < (y - x)$
apply (*case-tac* $y = -\infty$, *simp*,
cut-tac minf-le-any[*of* x],
frule less-imp-le[*of* x $-\infty$],
frule antisym[*of* x $-\infty$], *assumption*+,
cut-tac less-le[*of* x $-\infty$], *simp*)
apply (*case-tac* $x = -\infty$, *simp*,
case-tac $y = \infty$, *simp add*:*diff-ant-def a-minus-minus*,
simp add:*zero-lt-inf*,
cut-tac mem-ant[*of* y], *simp*, *erule exE*, *simp add*:*diff-ant-def*
a-minus-minus, *simp add*:*zero-lt-inf*)
apply (*case-tac* $x = \infty$, *simp*,
frule aless-imp-le[*of* ∞ y],
cut-tac inf-ge-any[*of* y], *frule ale-antisym*[*of* y ∞],
assumption+, *simp*,
cut-tac mem-ant[*of* x], *simp*, *erule exE*,
case-tac $y = \infty$, *simp add*:*diff-ant-def aminus*,
simp add:*zero-lt-inf*)
apply (*cut-tac mem-ant*[*of* y], *simp*, *erule exE*, *simp*,
simp add:*diff-ant-def*, *simp add*:*aminus a-zpz*,
simp add:*aless-zless*)
done

lemma *ale-minus*: $(x::ant) \leq y \Longrightarrow - y \leq - x$
apply (*cut-tac mem-ant*[*of* x], *cut-tac mem-ant*[*of* y])
apply ((*erule disjE*)+, *simp*, *erule disjE*, *erule exE*,
simp add:*aminus a-minus-minus*, *simp add*:*a-minus-minus*,
(*erule disjE*)+, (*erule exE*)+,
simp, *cut-tac* $x = ant\ z$ **in** *minf-le-any*, *frule-tac* $x = ant\ z$ **in**
ale-antisym[*of* - $-\infty$], *assumption*+, *simp*,
simp, *cut-tac* $x = \infty$ **in** *minf-le-any*,
frule-tac $x = \infty$ **in** *ale-antisym*[*of* - $-\infty$], *assumption*+, *simp*)
apply ((*erule disjE*)+, (*erule exE*)+, *simp add*:*aminus*, *erule exE*, *simp*,
erule disjE, *erule exE*, *simp*, *cut-tac* $x = ant\ z$ **in** *inf-ge-any*,
frule-tac $x = ant\ z$ **in** *ale-antisym*[*of* - ∞], *assumption*+, *simp*,
simp)
done

lemma *aless-minus*:$(x::ant) < y \Longrightarrow - y < - x$
by (*simp add*:*less-ant-def*, *erule conjE*, *simp add*:*ale-minus*,
rule not-sym, *rule contrapos-pp*, *simp*+,
cut-tac a-minus-minus[*of x*], *simp add*:*a-minus-minus*)

lemma *aadd-minus-le*:$(a::ant) \leq 0 \Longrightarrow a + b \leq b$
apply (*frule ale-minus*[*of a 0*],
cut-tac aadd-pos-le[*of* $-a$ $-b$], *simp add*:*aminus-0*)
apply (*frule ale-minus*[*of* $-b$ $-a + -b$], *simp add*:*aminus-add-distrib*,
simp add:*a-minus-minus*, *simp add*:*aminus-0*)
done

lemma *aadd-minus-less*:$[\![b \neq -\infty \wedge b \neq \infty; (a::ant) < 0]\!] \Longrightarrow a + b < b$
apply (*simp add*:*less-ant-def*, *erule conjE*,
simp add:*aadd-minus-le*)
apply (*rule contrapos-pp*, *simp*+,
cut-tac mem-ant[*of a*], *cut-tac mem-ant*[*of b*],
simp, *erule disjE*, *erule exE*, *simp*,
frule sym, *thin-tac* $- \infty = ant\ z$, *simp*,
erule disjE, (*erule exE*)+, *simp add*:*a-zpz*,
erule exE, *simp*, *frule sym*, *thin-tac* $\infty = ant\ z$, *simp*)
done

lemma *an-inj*:$an\ n = an\ m \Longrightarrow n = m$
apply (*simp add*:*an-def*)
done

lemma *nat-eq-an-eq*:$n = m \Longrightarrow an\ n = an\ m$
apply *simp*
done

lemma *aneq-natneq*:$(an\ n \neq an\ m) = (n \neq m)$
apply (*simp add*:*an-def*)
done

lemma *ale-natle*: $(an\ n \leq an\ m) = (n \leq m)$
apply (*simp add*:*an-def*)
done

lemma *aless-natless*:$(an\ n < an\ m) = (n < m)$
apply (*simp add*:*an-def*)
apply (*simp add*:*aless-zless*)
done

lemma *na-an*:$na\ (an\ n) = n$
by (*simp only*:*na-def an-def*,
subgoal-tac $\neg\ ant\ (int\ n) < 0$, *simp*,
simp add:*tna-ant*, *subst aneg-less*[*of ant* $(int\ n)$ *0*],

simp only:*ant-0*[*THEN sym*], *subst ale-zle*[*of 0 int n*], *simp*)

lemma *asprod-ge*:
  $0 < b \Longrightarrow N \neq 0 \Longrightarrow an\ N \leq int\ N *_a b$
apply (*frule aposs-le-1*[*of b*])
apply *simp*
using *asprod-pos-mono* [*THEN sym*, *of int N 1 b*]
apply (*simp only*: *ant-1* [*THEN sym*], *simp add*: *asprod-amult*, *simp add*:*an-def*)
done

lemma *an-npn*:*an* $(n + m) = an\ n + an\ m$
by (*unfold an-def*, *simp add*:*a-zpz*)

lemma *an-ndn*:$n \leq m \Longrightarrow an\ (m - n) = an\ m - an\ n$
apply (*cut-tac an-npn*[*of m − n n*], *simp*)
apply (*unfold an-def*)
apply (*simp add*:*a-zpz*[*of int* $(m - n)$ *int n*])
apply (*subst a-zdz*[*of int* $(m - n) + int\ n$ *int n*], *simp*)
done

1.8 Amin, amax

definition
  amin :: [*ant*, *ant*] ⇒ *ant* **where**
  amin x y = (*if* $(x \leq y)$ *then x else y*)

definition
  amax :: [*ant*, *ant*] ⇒ *ant* **where**
  amax x y = (*if* $(x \leq y)$ *then y else x*)

primrec *Amin* :: [*nat*, *nat* ⇒ *ant*] ⇒ *ant*
where
  Amin-0 : *Amin 0 f* = (*f 0*)
| *Amin-Suc* :*Amin* (*Suc n*) *f* = *amin* (*Amin n f*) (*f* (*Suc n*))

primrec *Amax* :: [*nat*, *nat* ⇒ *ant*] ⇒ *ant*
where
  Amax-0 : *Amax 0 f* = *f 0*
| *Amax-Suc* :*Amax* (*Suc n*) *f* = *amax* (*Amax n f*) (*f* (*Suc n*))

lemma *amin-ge*:$x \leq amin\ x\ y \vee y \leq amin\ x\ y$
apply (*simp add*:*amin-def*)
done

lemma *amin-le-l*:*amin x y* ≤ *x*
apply (*simp add*:*amin-def*, *cut-tac ale-linear*[*of x y*],
      rule impI, *simp*)
done

lemma *amin-le-r*:*amin x y* $\leq$ *y*
apply (*simp add*:*amin-def*)
done

lemma *amax-le*:*amax x y* $\leq$ *x* $\vee$ *amax x y* $\leq$ *y*
apply (*simp add*:*amax-def*)
done

lemma *amax-le-n*:$[\![x \leq n; y \leq n]\!] \Longrightarrow$ *amax x y* $\leq$ *n*
by (*simp add*:*amax-def*)

lemma *amax-ge-l*:*x* $\leq$ *amax x y*
apply (*simp add*:*amax-def*)
done

lemma *amax-ge-r*:*y* $\leq$ *amax x y*
apply (*simp add*:*amax-def*, *cut-tac ale-linear*[*of x y*],
rule impI, *simp*)
done

lemma *amin-mem-i*:$[\![x \in Z_\infty; y \in Z_\infty]\!] \Longrightarrow$ *amin x y* $\in Z_\infty$
apply (*cut-tac mem-ant*[*of x*], *cut-tac mem-ant*[*of y*], *simp add*:*aug-inf-def*,
(*erule disjE*)+, (*erule exE*)+, *cut-tac amin-ge*[*of x y*],
rule contrapos-pp, *simp*+,
erule disjE,
cut-tac x = *ant z* **in** *minf-le-any*,
frule-tac x = *ant z* **in** *ale-antisym*[*of* - $-\infty$], *assumption*+, *simp*,
cut-tac x = *ant za* **in** *minf-le-any*,
frule-tac x = *ant za* **in** *ale-antisym*[*of* - $-\infty$], *assumption*+, *simp*)
apply (*erule exE*, *simp add*:*amin-def*, *erule disjE*,
erule exE, *simp add*:*amin-def*, *simp add*:*amin-def*)
done

lemma *amax-mem-m*:$[\![x \in Z_{-\infty}; y \in Z_{-\infty}]\!] \Longrightarrow$ *amax x y* $\in Z_{-\infty}$
apply (*cut-tac mem-ant*[*of x*], *cut-tac mem-ant*[*of y*],
simp add:*aug-minf-def*)
apply ((*erule disjE*)+, *simp add*:*amax-def*,
erule exE, *simp add*:*amax-def*,
erule disjE, *erule exE*, *simp add*:*amax-def*)
apply ((*erule exE*)+, *cut-tac amax-le*[*of x y*],
rule contrapos-pp, *simp*+) **apply** (*erule disjE*,
cut-tac x = *ant z* **in** *inf-ge-any*,
frule-tac x = *ant z* **in** *ale-antisym*[*of* - ∞], *assumption*+, *simp*,
cut-tac x = *ant za* **in** *inf-ge-any*,
frule-tac x = *ant za* **in** *ale-antisym*[*of* - ∞], *assumption*+, *simp*)
done

lemma *amin-commute*:*amin x y* = *amin y x*
apply (*cut-tac ale-linear*[*of x y*], *erule disjE*, *simp add*:*amin-def*)

apply (*simp add*:*amin-def*)
done

lemma *amin-mult-pos*:$0 < z \Longrightarrow amin\ (z *_a x)\ (z *_a y) = z *_a amin\ x\ y$
by (*simp add*:*amin-def*, *simp add*:*asprod-pos-mono*)

lemma *amin-amult-pos*:$0 < z \Longrightarrow$
$amin\ ((ant\ z) * x)\ ((ant\ z) * y) = (ant\ z) * amin\ x\ y$
by (*simp add*:*asprod-amult*[*THEN sym*], *simp add*:*amin-mult-pos*)

lemma *times-amin*:$[\![0 < a;\ amin\ (x * (ant\ a))\ (y * (ant\ a)) \leq z * (ant\ a)]\!] \Longrightarrow$
$amin\ x\ y \leq z$
by (*frule amin-mult-pos*[*of a x y*], *simp add*:*asprod-amult*,
simp add:*amult-commute*[*of ant a*], *simp add*:*amult-pos-mono-r*)

lemma *Amin-memTr*:$f \in \{i.\ i \leq n\} \to Z_\infty \longrightarrow Amin\ n\ f \in Z_\infty$
apply (*induct-tac n*,
simp add:*Pi-def*)
apply (*rule impI*,
frule-tac func-pre[*of f - Z*$_\infty$],
simp, *rule amin-mem-i*, *assumption*+,
simp add:*Pi-def*)
done

lemma *Amin-mem*:$f \in \{i.\ i \leq n\} \to Z_\infty \Longrightarrow Amin\ n\ f \in Z_\infty$
apply (*simp add*:*Amin-memTr*)
done

lemma *Amax-memTr*:$f \in \{i.\ i \leq n\} \to Z_{-\infty} \longrightarrow Amax\ n\ f \in Z_{-\infty}$
apply (*induct-tac n*,
simp add:*Pi-def*)
apply (*rule impI*,
frule-tac func-pre[*of f - Z*$_{-\infty}$],
simp, *rule amax-mem-m*, *assumption*+,
simp add:*Pi-def*)
done

lemma *Amax-mem*:$f \in \{i.\ i \leq n\} \to Z_{-\infty} \Longrightarrow Amax\ n\ f \in Z_{-\infty}$
apply (*simp add*:*Amax-memTr*)
done

lemma *Amin-mem-mem*:$\forall j \leq n.\ f\ j \in Z_\infty \Longrightarrow Amin\ n\ f \in Z_\infty$
by (*rule Amin-mem*, *simp*)

lemma *Amax-mem-mem*:$\forall j \leq n.\ f\ j \in Z_{-\infty} \Longrightarrow Amax\ n\ f \in Z_{-\infty}$
by (*rule Amax-mem*, *simp*)

lemma *Amin-leTr*:$f \in \{i.\ i \leq n\} \to Z_\infty \longrightarrow (\forall j \in \{i.\ i \leq n\}.\ Amin\ n\ f \leq (f\ j))$
apply (*induct-tac n*,

rule impI, *rule ballI*,
simp)
apply (*rule impI*, *rule ballI*,
frule func-pre, *simp*)

apply (*case-tac* $j = Suc\ n$, *simp*, *rule amin-le-r*)
apply (*cut-tac* $x = j$ **and** $n = n$ **in** *Nset-pre*, *simp*, *assumption*,
cut-tac $x = Amin\ n\ f$ **and** $y = f\ (Suc\ n)$ **in** *amin-le-l*,
thin-tac $j \leq Suc\ n$, *simp*)
apply (*frule-tac* $x = j$ **in** *spec*,
thin-tac $\forall j{\leq}n.\ Amin\ n\ f \leq f\ j$, *simp*)
done

lemma *Amin-le*:$[\![f \in \{j.\ j \leq n\} \to\ Z_\infty;\ j \in \{k.\ k \leq n\}]\!] \Longrightarrow Amin\ n\ f \leq (f\ j)$
apply (*simp add*:*Amin-leTr*)
done

lemma *Amax-geTr*:$f \in \{j.\ j \leq n\} \to Z_{-\infty} \longrightarrow (\forall j{\in}\{j.\ j \leq n\}.\ (f\ j) \leq Amax\ n\ f)$
apply (*induct-tac* n,
rule impI, *rule ballI*,
simp)
apply (*rule impI*, *rule ballI*,
frule func-pre, *simp*,
case-tac $j = Suc\ n$, *simp*, *rule amax-ge-r*,
cut-tac $x = j$ **and** $n = n$ **in** *Nset-pre*, *simp*, *assumption*,
thin-tac $j \leq Suc\ n$,
simp)
apply (*cut-tac* $x = Amax\ n\ f$ **and** $y = f\ (Suc\ n)$ **in** *amax-ge-l*,
drule-tac $x = j$ **in** *spec*, *simp*)
done

lemma *Amax-ge*:$[\![f \in \{j.\ j \leq n\} \to Z_{-\infty};\ j \in \{j.\ j \leq n\}]\!] \Longrightarrow$
$(f\ j) \leq (Amax\ n\ f)$
apply (*simp add*:*Amax-geTr*)
done

lemma *Amin-mem-le*:$[\![\forall j \leq n.\ (f\ j) \in\ Z_\infty;\ j \in \{j.\ j \leq n\}]\!] \Longrightarrow$
$(Amin\ n\ f) \leq (f\ j)$
by (*rule Amin-le*, *simp*, *simp*)

lemma *Amax-mem-le*:$[\![\forall j \leq n.\ (f\ j) \in\ Z_{-\infty};\ j \in \{j.\ j \leq n\}]\!] \Longrightarrow$
$(f\ j) \leq (Amax\ n\ f)$
by (*rule Amax-ge*, *simp*, *simp*)

lemma *amin-ge1*:$[\![(z{::}ant) \leq x;\ z \leq y\ ]\!] \Longrightarrow z \leq amin\ x\ y$
by (*simp add*:*amin-def*)

lemma *amin-gt*:$[\![(z{::}ant) < x;\ z < y]\!] \Longrightarrow z < amin\ x\ y$

apply (*simp add*:*less-ant-def*, (*erule conjE*)+,
rule conjI, *simp add*:*amin-ge1*)
apply (*rule contrapos-pp*, *simp*+,
case-tac $x \leq y$, *simp add*:*amin-def*, *simp add*:*amin-def*)
done

lemma *Amin-ge1Tr*:$(\forall j{\leq}(Suc\ n).\ (f\ j) \in Z_\infty \wedge z \leq (f\ j)) \longrightarrow$
$z \leq (Amin\ (Suc\ n)\ f)$
apply (*induct-tac n*)
apply (*rule impI*)
apply (*frule-tac* $x = 0$ **in** *spec*,
frule-tac $x = Suc\ 0$ **in** *spec*,
thin-tac $\forall j{\leq}Suc\ 0.\ f\ j \in Z_\infty \wedge z \leq f\ j$, *simp*, (*erule conjE*)+,
simp add:*amin-ge1*)

apply (*rule impI*,
simp,
frule-tac $a = Suc\ (Suc\ n)$ **in** *forall-spec*,
thin-tac $\forall j{\leq}Suc\ (Suc\ n).\ f\ j \in Z_\infty \wedge z \leq f\ j$, *simp*,
thin-tac $\forall j{\leq}Suc\ (Suc\ n).\ f\ j \in Z_\infty \wedge z \leq f\ j$, *erule conjE*)
apply (*rule amin-ge1*, *assumption*+)
done

lemma *Amin-ge1*:$[\![\ \forall j \leq (Suc\ n).\ f\ j \in Z_\infty;\ \forall j \leq (Suc\ n).\ z \leq (f\ j)]\!] \Longrightarrow$
$z \leq (Amin\ (Suc\ n)\ f)$
apply (*simp del*:*Amin-Suc add*:*Amin-ge1Tr*)
done

lemma *amin-trans1*:$[\![x \in Z_\infty;\ y \in Z_\infty;\ z \in Z_\infty;\ z \leq x\ ]\!] \Longrightarrow$
$amin\ z\ y \leq amin\ x\ y$
apply (*case-tac* $z \leq y$, *simp add*:*amin-def*)
apply (*simp add*:*amin-def*)
apply (*simp only*:*aneg-le*[*of z y*], *frule aless-imp-le*[*of y z*],
frule ale-trans[*of y z x*], *assumption*+, *rule impI*,
frule ale-antisym[*of y x*], *assumption*+)
done

lemma *inf-in-aug-inf*:$\infty\ \in Z_\infty$
apply (*simp add*:*aug-inf-def*, *simp add*:*not-sym*)
done

1.8.1 Maximum element of a set of ants

primrec *aasc-seq* :: [*ant set*, *ant*, *nat*] ⇒ *ant*
where
aasc-seq-0 : *aasc-seq A a 0* = *a*
| *aasc-seq-Suc* : *aasc-seq A a* (*Suc n*) =
(*SOME b*. ((*b* ∈ *A*) ∧ (*aasc-seq A a n*) < *b*))

lemma *aasc-seq-mem*:⟦$a \in A$; $\neg\ (\exists\, m.\ m{\in}A \wedge (\forall\, x{\in}A.\ x \leq m))$⟧ ⟹
($aasc\text{-}seq\ A\ a\ n$) $\in A$
apply (*induct-tac n*)
apply (*simp*, *simp*)
apply (*simp add*:*aneg-le*,
frule-tac a = *aasc-seq A a n* **in** *forall-spec*, *assumption*+,
thin-tac $\forall\, m.\ m \in A \longrightarrow (\exists\, x{\in}A.\ m < x)$,
rule someI2-ex, *blast*, *simp*)
done

lemma *aasc-seqn*:⟦$a \in A$; $\neg\ (\exists\, m.\ m{\in}A \wedge (\forall\, x{\in}A.\ x \leq m))$⟧ ⟹
($aasc\text{-}seq\ A\ a\ n$) < ($aasc\text{-}seq\ A\ a\ (Suc\ n)$)
apply (*frule aasc-seq-mem* [*of a A n*], *assumption*+,
simp add:*aneg-le*,
frule-tac a = *aasc-seq A a n* **in** *forall-spec*, *assumption*+,
thin-tac $\forall\, m.\ m \in A \longrightarrow (\exists\, x{\in}A.\ m < x)$, *rule someI2-ex*, *blast*, *simp*)
done

lemma *aasc-seqn1*:⟦$a \in A$; $\neg\ (\exists\, m.\ m{\in}A \wedge (\forall\, x{\in}A.\ x \leq m))$⟧ ⟹
($aasc\text{-}seq\ A\ a\ n$) + *1* ≤ ($aasc\text{-}seq\ A\ a\ (Suc\ n)$)
by (*frule aasc-seqn* [*of a A n*], *assumption*+, *simp*)

lemma *aubs-ex-n-maxTr*:⟦$a \in A$; $\neg\ (\exists\, m.\ m{\in}A \wedge (\forall\, x{\in}A.\ x \leq m))$⟧ ⟹
($a + an\ n$) ≤ ($aasc\text{-}seq\ A\ a\ n$)
apply (*induct-tac n*)
apply (*simp add*:*aadd-0-r*,
frule-tac n = *n* **in** *aasc-seqn1*[*of a A*], *assumption*+,
cut-tac x = *a* + *an n* **and** *y* = *aasc-seq A a n* **in**
aadd-le-mono[*of - - 1*], *assumption*, *simp*,
frule-tac i = *a* + *an n* + *1* **and** *j* = *aasc-seq A a n* + *1* **and**
k = (*SOME b*. *b* ∈ *A* ∧ *aasc-seq A a n* < *b*) **in** *ale-trans*, *assumption*+)
apply (*simp add*:*an-Suc*,
case-tac a = $-\infty$,
subst ant-1[*THEN sym*], *simp del*:*ant-1 add*:*a-zpz an-def*,
subgoal-tac a ∈ Z_∞, *subgoal-tac an n* ∈ Z_∞,
subgoal-tac 1 ∈ Z_∞,
subst aadd-assoc-i[*THEN sym*], *assumption*+)
apply (*subst ant-1*[*THEN sym*], *simp del*:*ant-1 add*:*aug-inf-def*,
(*simp add*:*aug-inf-def an-def*)+)
done

lemma *aubs-ex-AMax*:⟦$A \subseteq$ *UBset* (*ant z*); $A \neq \{\}$⟧ ⟹ $\exists!m.\ m{\in}A \wedge (\forall\, x{\in}A.\ x \leq m)$
apply (*case-tac A* = {$-\infty$}, *simp*,
frule not-sub-single[*of A* $-\infty$], *assumption*+,
frule not-sub[*of A* {$-\infty$}],
erule exE, *erule conjE*, *simp*, *rename-tac a*, *rule ex-ex1I*)
prefer *2*
apply ((*erule conjE*)+,

frule-tac $x = y$ **in** *bspec*, *assumption*+,
thin-tac $\forall x{\in}A.\ x \leq m$,
frule-tac $x = m$ **in** *bspec*, *assumption*+,
thin-tac $\forall x{\in}A.\ x \leq y$, *simp*)
apply (*rule contrapos-pp*, *simp*,
subgoal-tac $\exists w.\ a = ant\ w$, *erule exE*,
frule-tac $a = a$ **and** $A = A$ **and** $n = nat\ ((abs\ w) + (abs\ z) + 1)$ **in**
aubs-ex-n-maxTr, *simp*,
frule-tac $a = a$ **and** $n = nat\ ((abs\ w) + (abs\ z) + 1)$ **in**
aasc-seq-mem[*of* - *A*], *assumption*,
thin-tac $\neg\ (\exists m.\ m \in A \wedge (\forall x{\in}A.\ x \leq m))$,
simp add:*UBset-def*)
apply (*frule-tac* $c = aasc\text{-}seq\ A\ (ant\ w)\ (nat\ (|w| + |z| + 1))$ **in**
subsetD[*of A* $\{x.\ x \leq ant\ z\}$], *assumption*+,
simp)
apply(*frule-tac* $i = ant\ w + an\ (nat\ (|w| + |z| + 1))$ **and**
$j = aasc\text{-}seq\ A\ (ant\ w)\ (nat\ (|w| + |z| + 1))$ **and**
$k = ant\ z$ **in** *ale-trans*, *assumption*+)
apply (*thin-tac* $ant\ w + an\ (nat\ (|w| + |z| + 1))$
$\leq aasc\text{-}seq\ A\ (ant\ w)\ (nat\ (|w| + |z| + 1))$,
thin-tac $aasc\text{-}seq\ A\ (ant\ w)\ (nat\ (|w| + |z| + 1)) \in A$,
thin-tac $aasc\text{-}seq\ A\ (ant\ w)\ (nat\ (|w| + |z| + 1)) \leq ant\ z$,
simp add:*an-def a-zpz*)
apply (*cut-tac* $a = a$ **in** *mem-ant*, *erule disjE*, *simp*, *erule disjE*, *erule exE*,
simp, *simp add*:*UBset-def*, *frule subsetD*[*of A* $\{x.\ x \leq ant\ z\}$ ∞],
assumption+, *simp*, *cut-tac inf-ge-any*[*of ant z*],
frule-tac ale-antisym[*of ant z* ∞], *assumption*+, *simp*)
done

definition
AMax :: *ant set* $\Rightarrow$ *ant* **where**
$AMax\ A = (THE\ m.\ m \in A \wedge (\forall x{\in}A.\ x \leq m))$

definition
AMin::*ant set* $\Rightarrow$ *ant* **where**
$AMin\ A = (THE\ m.\ m \in A \wedge (\forall x{\in}A.\ m \leq x))$

definition
rev-o :: *ant* $\Rightarrow$ *ant* **where**
$rev\text{-}o\ x = -\ x$

lemma *AMax*:$[\![A \subseteq UBset\ (ant\ z);\ A \neq \{\}]\!] \Longrightarrow$
$(AMax\ A) \in A \wedge (\forall x{\in}A.\ x \leq (AMax\ A))$
apply (*simp add*:*AMax-def*)
apply (*frule aubs-ex-AMax*[*of A z*], *assumption*)
apply (*rule theI'*)
apply *assumption*
done

lemma *AMax-mem*:⟦$A \subseteq$ *UBset* (*ant z*); $A \neq \{\}$⟧ ⟹ (*AMax A*) ∈ *A*
apply (*simp add*:*AMax*[*of A z*])
done

lemma *rev-map-nonempty*:$A \neq \{\}$ ⟹ *rev-o* ‘ *A* ≠ {}
by (*rule contrapos-pp*, *simp*+)

lemma *rev-map*:*rev-o* ∈ *LBset* (*ant* (−*z*)) → *UBset* (*ant z*)
by (*rule Pi-I*, *simp add*:*UBset-def LBset-def rev-o-def*,
frule-tac x = *ant* (−*z*) **and** *y* = *x* **in** *ale-minus*, *simp add*:*aminus*)

lemma *albs-ex-AMin*:⟦$A \subseteq$ *LBset* (*ant z*); $A \neq \{\}$⟧ ⟹ ∃!*m*. *m*∈*A* ∧ (∀ *x*∈*A*. *m* ≤ *x*)
apply (*rule ex-ex1I*)
prefer *2* **apply** ((*erule conjE*)+,
frule-tac x = *y* **in** *bspec*, *assumption*+,
thin-tac ∀ *x*∈*A*. *m* ≤ *x*,
frule-tac x = *m* **in** *bspec*, *assumption*+,
thin-tac ∀ *x*∈*A*. *y* ≤ *x*, *simp*)
apply (*subgoal-tac* − *AMax* (*rev-o* ‘ *A*) ∈ *A* ∧
(∀ *x* ∈ *A*. (− *AMax* (*rev-o* ‘ *A*)) ≤ *x*), *blast*,
cut-tac rev-map[*of* −*z*], *simp add*:*a-minus-minus*,
frule rev-map-nonempty[*of A*],
frule image-sub[*of rev-o LBset* (*ant z*) *UBset* (*ant* (−*z*)) *A*],
assumption+, *frule AMax*[*of rev-o* ‘ *A* −*z*], *assumption*+,
erule conjE,
rule conjI, *thin-tac* ∀ *x*∈*rev-o* ‘ *A*. *x* ≤ *AMax* (*rev-o* ‘ *A*),
thin-tac rev-o ∈ *LBset* (*ant z*) → *UBset* (*ant* (− *z*)),
thin-tac rev-o ‘ *A* ≠ {},
thin-tac rev-o ‘ *A* ⊆ *UBset* (*ant* (− *z*)))
apply (*simp add*:*image-def rev-o-def*,
erule bexE, *simp add*:*a-minus-minus*, *rule ballI*,
subgoal-tac rev-o x ∈ *rev-o* ‘ *A*,
frule-tac x = *rev-o x* **in** *bspec*, *assumption*+,
thin-tac ∀ *x*∈*rev-o* ‘ *A*. *x* ≤ *AMax* (*rev-o* ‘ *A*),
thin-tac rev-o ∈ *LBset* (*ant z*) → *UBset* (*ant* (− *z*)),
thin-tac rev-o ‘ *A* ≠ {},
thin-tac rev-o ‘ *A* ⊆ *UBset* (*ant* (− *z*)))
apply (*simp add*:*image-def rev-o-def*, *erule bexE*, *simp add*:*a-minus-minus*,
frule-tac x = −*x* **and** *y* = −*xa* **in** *ale-minus*, *simp add*:*a-minus-minus*,
simp add:*image-def*, *blast*)
done

lemma *AMin*:⟦$A \subseteq$ *LBset* (*ant z*); $A \neq \{\}$⟧ ⟹
(*AMin A*) ∈ *A* ∧ (∀ *x*∈*A*. (*AMin A*) ≤ *x*)
apply (*simp add*:*AMin-def*)
apply (*frule albs-ex-AMin*[*of A z*], *assumption*)
apply (*rule theI'*)
apply *assumption*

done

lemma *AMin-mem*:$[\![A \subseteq LBset\ (ant\ z);\ A \neq \{\}]\!] \Longrightarrow (AMin\ A) \in A$
apply (*simp add*:*AMin*)
done

primrec $ASum :: (nat \Rightarrow ant) \Rightarrow nat \Rightarrow ant$
where
  ASum-0: $ASum\ f\ 0 = f\ 0$
| *ASum-Suc*: $ASum\ f\ (Suc\ n) = (ASum\ f\ n) + (f\ (Suc\ n))$

lemma *age-plus*:$[\![0 \leq (a::ant);\ 0 \leq b;\ a + b \leq c]\!] \Longrightarrow a \leq c$
apply (*frule aadd-le-mono*[*of 0 b a*])
apply (*simp add*:*aadd-commute*[*of b a*] *aadd-0-l*)
done

lemma *age-diff-le*:$[\![(a::ant) \leq c;\ 0 \leq b]\!] \Longrightarrow a - b \leq c$
apply (*frule ale-minus*[*of 0 b*], *thin-tac* $0 \leq b$, *simp*)
apply (*frule aadd-le-mono*[*of a c* $-b$])
apply (*frule aadd-le-mono*[*of* $-b$ *0 c*])
apply (*thin-tac* $a \leq c$, *thin-tac* $-b \leq 0$,
      simp add:*aadd-commute*[*of* $-b$ *c*] *aadd-0-l*)
apply (*simp add*:*diff-ant-def*)
done

lemma *adiff-le-adiff*:$a \leq (a'::ant) \Longrightarrow a - b \leq a' - b$
apply (*simp add*:*diff-ant-def*)
apply (*rule aadd-le-mono*[*of a a'* $-b$], *assumption*+)
done

lemma *aplus-le-aminus*:$[\![\ a \in\ Z_{-\infty};\ b \in\ Z_{-\infty};\ c \in\ Z_{-\infty};\ -b \in\ Z_{-\infty}]\!] \Longrightarrow$
        $((a + b) \leq (c::ant)) = (a \leq c - b)$
apply (*rule iffI*)
apply (*frule aadd-le-mono*[*of a* + *b c* $-b$])
 apply (*simp add*:*aadd-assoc-m*, *simp add*:*aadd-minus-r*)
 apply (*simp add*:*aadd-0-r*, *simp add*:*diff-ant-def*)

apply (*frule aadd-le-mono*[*of a c* − *b b*])
apply (*simp add*:*diff-ant-def*)
apply (*simp add*:*aadd-assoc-m*)
apply (*simp add*:*aadd-minus-inv*[*of b*])
apply (*simp add*: *aadd-0-r*)
done

1.9 Cardinality of sets

cardinality is defined for the finite sets only

lemma *card-eq*:$A = B \Longrightarrow card\ A = card\ B$

apply *simp*
done

lemma *card0*:*card* {} = *0*
by *simp*

lemma *card-nonzero*:⟦*finite A*; *card A* ≠ *0*⟧ ⟹ *A* ≠ {}
by (*rule contrapos-pp*, *simp*+)

lemma *finite1*:*finite* {*a*}
by *simp*

lemma *card1*:*card* {*a*} = *1*
by *simp*

lemma *nonempty-card-pos*:⟦*finite A*; *A* ≠ {}⟧ ⟹ *0* < *card A*
apply (*frule nonempty-ex* [*of A*], *erule exE*,
  frule-tac a = *x* **and** *A* = *A* **in** *singleton-sub*)
apply (*frule-tac B* = *A* **and** *A* = {*x*} **in** *card-mono*, *assumption*+,
  simp add:*card1*)
done

lemma *nonempty-card-pos1*:⟦*finite A*; *A* ≠ {}⟧ ⟹ *Suc 0* ≤ *card A*
apply (*frule nonempty-card-pos*[*of A*], *assumption*+)
apply (*rule Suc-leI*[*of 0 card A*], *assumption*)
done

lemma *card1-tr0*:⟦ *finite A*; *card A* = *Suc 0*; *a* ∈ *A* ⟧ ⟹ {*a*} = *A*
apply (*cut-tac card1*[*of a*])
apply (*rule card-seteq*[*of A* {*a*}], *assumption*)
apply (*rule singleton-sub*[*of a A*], *assumption*)
apply *simp*
done

lemma *card1-tr1*:(*constmap* {*0*::*nat*} {*x*}) ∈ {*0*} → {*x*} ∧
  surj-to (*constmap* {*0*::*nat*} {*x*}) {*0*} {*x*}
apply (*rule conjI*, *simp add*:*constmap-def Pi-def*,
  simp add:*surj-to-def image-def constmap-def*)
done

lemma *card1-Tr2*:⟦*finite A*; *card A* = *Suc 0*⟧ ⟹
  ∃*f*. *f* ∈ {*0*::*nat*} → *A* ∧ *surj-to f* {*0*} *A*
apply (*frule card-nonzero*[*of A*], *simp*)
apply (*cut-tac nonempty-ex*[*of A*], *erule exE*)
apply (*frule-tac a* = *x* **in** *card1-tr0*[*of A*], *assumption*+)
apply (*rotate-tac* −*1*, *frule sym*, *thin-tac* {*x*} = *A*, *simp*)
apply (*cut-tac x* = *x* **in** *card1-tr1*, *blast*, *simp*)
done

lemma *card2*:⟦ *finite A*; $a \in A$; $b \in A$; $a \neq b$ ⟧ ⟹ *Suc* (*Suc 0*) ≤ *card A*
apply (*cut-tac card1*[*of a*])
apply (*frule singleton-sub*[*of b A*])
apply (*frule finite-subset*[*of* {*b*} *A*], *assumption*)
apply (*frule card-insert-disjoint*[*of* {*b*} *a*])
apply *simp*
apply (*simp only*:*card1*)
apply (*frule insert-sub*[*of* {*b*} *A a*], *assumption*+)
apply (*frule card-mono* [*of A* {*a*, *b*}], *assumption*)
apply *simp*
done

lemma *card2-inc-two*:⟦*0* < (*n*::*nat*); *x* ∈ {*j*. *j* ≤ *n*}⟧ ⟹
∃ *y* ∈ {*j*. *j* ≤ *n*}. *x* ≠ *y*
apply (*rule contrapos-pp*, *simp*+)
apply (*frule-tac m = 0* **and** *n = n* **in** *Suc-leI*) **apply** (
frule-tac a = Suc 0 **in** *forall-spec*, *assumption*)
apply (*frule-tac a = 0* **in** *forall-spec*)
apply (*rule less-imp-le*, *assumption*)
apply *simp*
done

lemma *Nset2-prep1*:⟦*finite A*; *card A = Suc* (*Suc n*) ⟧ ⟹ ∃ *x*. *x*∈*A*
apply (*frule card-nonzero*[*of A*])
apply *simp*
apply (*simp add*:*nonempty-ex*)
done

lemma *ex-least-set*:⟦*A* = {*H*. *finite H* ∧ *P H*}; *H* ∈ *A*⟧ ⟹
∃ *K* ∈ *A*. (*LEAST j*. *j* ∈ (*card* ‘ *A*)) = *card K*

by (*simp add*:*image-def*, *rule LeastI*, *rule-tac x = H* **in** *exI*, *simp*)

lemma *Nset2-prep2*:*x* ∈ *A* ⟹ *A* − {*x*} ∪ {*x*} = *A*
by *auto*

lemma *Nset2-finiteTr*:∀ *A*. (*finite A* ∧(*card A = Suc n*) ⟶
(∃ *f*. *f* ∈ {*i*. *i* ≤ *n*} → *A* ∧ *surj-to f* {*i*. *i* ≤ *n*} *A*))
apply (*induct-tac n*, *rule allI*, *rule impI*, *erule conjE*)
apply (*simp add*: *card1-Tr2 del*: *Pi-split-insert-domain*)

apply (*rule allI*, *rule impI*, *erule conjE*, *frule Nset2-prep1*, *assumption*+)
apply (*erule exE*)
apply(*drule-tac a = A − {x}* **in** *forall-spec*)
apply *simp*
apply (*erule exE*)
apply (*cut-tac x = x* **in** *card1-tr1*, (*erule conjE*)+)
apply (*frule-tac f = f* **and** *n = n* **and** *A = A − {x}* **and**

$g = constmap\ \{0\}\ \{x\}$ **and** $m = 0$ **and** $B = \{x\}$ **in** *jointfun-surj*,
assumption+)
apply *simp*+
apply (*frule-tac* $f = f$ **and** $n = n$ **and** $A = A - \{x\}$ **and**
$g = constmap\ \{0\}\ \{x\}$ **and** $m = 0$ **and** $B = \{x\}$ **in** *jointfun-hom0*,
simp,
frule-tac $x = x$ **and** $A = A$ **in** *Nset2-prep2*, *simp*, *blast*)
done

lemma *Nset2-finite*:$[\![$ *finite* A; *card* $A = Suc\ n]\!] \Longrightarrow$
$\exists f.\ f \in \{i.\ i \leq n\} \to A \wedge surj\text{-}to\ f\ \{i.\ i \leq n\}\ A$
by (*simp add*:*Nset2-finiteTr*)

lemma *Nset2finite-inj-tr0*:$j \in \{i.\ i \leq (n::nat)\} \Longrightarrow$
$card\ (\{i.\ i \leq n\} - \{j\}) = n$
by *simp*

lemma *Nset2finite-inj-tr1*:$[\![\ i \leq (n::nat);\ j \leq n;\ f\ i = f\ j;\ i \neq j\ ]\!] \Longrightarrow$
$f\ `\ (\{i.\ i \leq n\} - \{j\}) = f\ `\ \{i.\ i \leq n\}$
apply (*simp add*:*image-def*, *rule equalityI*, *rule subsetI*, *simp add*:*CollectI*,
erule bexE, *case-tac* $xa = j$, *frule sym*, *thin-tac* $f\ i = f\ j$,
simp, *blast*)
apply (*rule subsetI*, *simp*, *erule exE*, *case-tac* $xa = j$, *frule sym*,
thin-tac $f\ i = f\ j$, *blast*, *blast*)
done

lemma *Nset2finite-inj*:$[\![$*finite* A; *card* $A = Suc\ n$; *surj-to* $f\ \{i.\ i \leq n\}\ A\ ]\!] \Longrightarrow$
inj-on $f\ \{i.\ i \leq n\}$
by (*metis card-Collect-le-nat eq-card-imp-inj-on finite-Collect-le-nat surj-to-def*)

definition
$zmax :: [int, int] \Rightarrow int$ **where**
$zmax\ x\ y = (if\ (x \leq y)\ then\ y\ else\ x)$

primrec $Zmax :: [nat, nat \Rightarrow int] \Rightarrow int$
where
Zmax-0 : $Zmax\ 0\ f = f\ 0$
| *Zmax-Suc* :$Zmax\ (Suc\ n)\ f = zmax\ (Zmax\ n\ f)\ (f\ (Suc\ n))$

lemma *Zmax-memTr*:$f \in \{i.\ i \leq (n::nat)\} \to (UNIV::int\ set) \longrightarrow$
$Zmax\ n\ f \in f\ `\ \{i.\ i \leq n\}$
apply (*induct-tac* n)
apply *simp*
apply (*rule impI*)
apply (*frule func-pre*)
apply (*frule-tac* $f = f$ **and** $A = \{i.\ i \leq Suc\ n\}$ **and** $B = UNIV$ **and**
$?A1.0 = \{i.\ i \leq n\}$ **and** $?A2.0 = \{i.\ i \leq Suc\ n\}$ **in** *im-set-mono*)
apply (*rule subsetI*, *simp*, *simp*, *simp*)

apply (*case-tac* (*Zmax n f*) ≤ (*f* (*Suc n*)), *simp add*:*zmax-def*)
apply (*simp add*:*zmax-def*)
apply (*simp add*:*subsetD*)
done

lemma *zmax-ge-r*:*y* ≤ *zmax x y*
by (*simp add*:*zmax-def*)

lemma *zmax-ge-l*:*x* ≤ *zmax x y*
by (*simp add*:*zmax-def*)

lemma *Zmax-geTr*:*f* ∈ {*j*. *j* ≤ (*n*::*nat*)} → (*UNIV*::*int set*) ⟶
(∀ *j*∈{*j*. *j* ≤ *n*}. (*f j*) ≤ *Zmax n f*)
apply (*induct-tac n*,
rule impI, *rule ballI*,
simp)
apply (*rule impI*, *rule ballI*,
frule func-pre, *simp*,
case-tac j = *Suc n*, *simp*, *rule zmax-ge-r*,
cut-tac x = *j* **and** *n* = *n* **in** *Nset-pre*, *simp*, *assumption*,
thin-tac j ≤ *Suc n*,
simp)

apply (*cut-tac x* = *Zmax n f* **and** *y* = *f* (*Suc n*) **in** *zmax-ge-l*,
frule-tac x = *j* **in** *spec*,
thin-tac ∀ *j*≤*n*. *f j* ≤ *Zmax n f*)
apply *simp*
done

lemma *Zmax-plus1*:*f* ∈ {*j*. *j* ≤ (*n*::*nat*)} → (*UNIV*::*int set*) ⟹
((*Zmax n f*) + *1*) ∉ *f* ‘ {*j*. *j* ≤ *n*}
apply (*cut-tac Zmax-geTr*[*of f n*])
apply (*rule contrapos-pp*, *simp*+)
apply (*simp add*:*image-def*, *erule exE*, *erule conjE*)
apply (*frule-tac a* = *x* **in** *forall-spec*, *assumption*,
thin-tac ∀ *j*≤*n*. *f j* ≤ *Zmax n f*)
apply (*frule sym*, *thin-tac Zmax n f* + *1* = *f x*, *simp*)
done

lemma *infinite-Univ-int*:¬ (*finite* (*UNIV* :: *int set*))
apply (*rule contrapos-pp*, *simp*+)
apply (*subgoal-tac* (*0*::*int*) ∈ *UNIV*)
prefer *2* **apply** *simp*
apply (*frule nonempty*[*of* (*0*::*int*) *UNIV*])
apply (*frule-tac nonempty-card-pos*[*of UNIV*], *assumption*)
apply (*frule Nset2-finite*[*of UNIV* (*card UNIV*) − *Suc 0*],
rule Suc-pred[*THEN sym*, *of card UNIV*],*simp*)
apply (*erule exE*, *erule conjE*)
apply (*frule-tac f* = *f* **in**

Nset2finite-inj[*of UNIV* (*card UNIV*) − *Suc 0*],
rule Suc-pred[*THEN sym*, *of card UNIV*], *simp*, *assumption*)
apply (*frule-tac f* = *f* **and** *n* = *card UNIV* − *Suc 0* **in** *Zmax-plus1*)
apply (*simp add*:*surj-to-def*)
done

lemma *image-Nsetn-card-pos*: *0* < *card* (*f* ‘ {*i*. *i* ≤ (*n*::*nat*)})
apply(*rule nonempty-card-pos*)
apply *auto*
done

lemma *card-image-Nsetn-Suc*
:⟦*f* ∈ {*j*. *j* ≤ *Suc n*} → *B*;
f (*Suc n*) ∉ *f* ‘ {*j*. *j* ≤ *n*}⟧ ⟹
card (*f* ‘ {*j*. *j* ≤ *Suc n*}) − *Suc 0* =
Suc (*card* (*f* ‘ {*j*. *j* ≤ *n*}) − *Suc 0*)
apply (*simp add*:*image-Nset-Suc*)
apply (*cut-tac image-Nsetn-card-pos*[*of f n*], *simp*)
done

lemma *slide-surj*:*i* < (*j*::*nat*) ⟹
surj-to (*slide i*) {*l*. *l* ≤ (*j* − *i*)} (*nset i j*)
proof −
assume *p1*:*i* < *j*
from *p1* **show** *?thesis*
apply (*simp add*:*surj-to-def image-def*)
apply (*rule equalityI*,
rule subsetI, *simp*, *erule exE*, *simp add*:*slide-def nset-def*,
frule less-imp-le [*of i j*], *erule conjE*,
thin-tac i < *j*, *frule add-le-mono* [*of - j* − *i i i*],
simp+, *rule subsetI*, *simp*)
apply (*simp add*:*nset-def slide-def*, *erule conjE*,
frule-tac m = *x* **and** *n* = *j* **and** *l* = *i* **in** *diff-le-mono*,
subgoal-tac x = *i* + (*x* − *i*), *blast*, *simp*)
done
qed

lemma *slide-inj*:*i* < *j* ⟹ *inj-on* (*slide i*) {*k*. *k* ≤ (*j* − *i*)}
apply (*simp add*:*inj-on-def*, (*rule allI*)+)
apply (*rule impI*, *rule allI*, *rule impI*, *rule impI*)
apply (*simp add*:*slide-def*)
done

lemma *card-nset*:*i* < (*j* :: *nat*) ⟹ *card* (*nset i j*) = *Suc* (*j* − *i*)
apply (*frule slide-inj* [*of i j*])
apply (*frule card-image* [*of slide i* {*k*. *k* ≤ (*j* − *i*)}])
apply (*simp*, *frule slide-surj* [*of i j*], *simp add*:*surj-to-def*)
done

lemma *sliden-hom*:$i < j \Longrightarrow (sliden\ i) \in nset\ i\ j \rightarrow \{k.\ k \leq (j - i)\}$
by (*simp add*:*Pi-def*, *rule allI*, *rule impI*, *simp add*:*sliden-def*,
simp add:*nset-def*, *erule conjE*, *simp add*:*diff-le-mono*)

lemma *slide-sliden*:$(sliden\ i)\ (slide\ i\ k) = k$
by (*simp add*:*sliden-def slide-def*)

lemma *sliden-surj*:$i < j \Longrightarrow surj\text{-}to\ (sliden\ i)\ (nset\ i\ j)\ \{k.\ k \leq (j - i)\}$
apply (*simp add*:*surj-to-def image-def*, *rule equalityI*)
apply (*rule subsetI*, *simp*, *erule bexE*, *simp add*:*nset-def sliden-def*,
erule conjE, *rule-tac* $m = xa$ **in** *diff-le-mono*[*of* - *j i*],
assumption+)
apply (*rule subsetI*, *simp add*:*nset-def sliden-def*,
frule-tac $i = x$ **in** *add-le-mono*[*of* - $j - i\ i\ i$], *simp*,
simp, *subgoal-tac* $i \leq x + i$, *subgoal-tac* $x = (x + i) - i$,
blast) **apply** *simp*+
done

lemma *sliden-inj*: $i < j \Longrightarrow inj\text{-}on\ (sliden\ i)\ (nset\ i\ j)$
apply (*simp add*:*inj-on-def*, (*rule ballI*)+, *rule impI*, *simp add*:*sliden-def*)
apply (*simp add*:*nset-def*, (*erule conjE*)+,
subgoal-tac $(x - i = y - i) = (x = y)$, *blast*)
apply (*rule eq-diff-iff*, *assumption*+)
done

definition
transpos :: [*nat*, *nat*] ⇒ (*nat* ⇒ *nat*) **where**
transpos i j = (λ*k*. *if k* = *i then j else if k* = *j then i else k*)

lemma *transpos-id*:$[\![\ i \leq n;\ j \leq n;\ i \neq j\ ;\ x \in \{k.\ k \leq n\} - \{i, j\}\ ]\!]$
$\Longrightarrow transpos\ i\ j\ x = x$
proof −
assume *p1*:$i \leq n$ **and** *p2*:$j \leq n$ **and** *p3*: $i \neq j$ **and**
p4:$x \in \{k.\ k \leq n\} - \{i, j\}$
from *p1* **and** *p2* **and** *p3* **and** *p4* **show** *?thesis*
apply (*simp add*:*transpos-def*)
done
qed

lemma *transpos-id-1*:$[\![i \leq n;\ j \leq n;\ i \neq j;\ x \leq n;\ x \neq i;\ x \neq j]\!] \Longrightarrow$
$transpos\ i\ j\ x = x$
proof −
assume *p1*:$i \leq n$ **and** *p2*:$j \leq n$ **and** *p3*:$i \neq j$ **and** *p4*:$x \leq n$ **and** *p5*:$x \neq i$ **and**
p6:$x \neq j$
from *p1* **and** *p2* **and** *p3* **and** *p4* **and** *p5* **and** *p6* **show** *?thesis*
apply (*simp add*:*transpos-def*)
done
qed

lemma *transpos-id-2*:$i \leq n \Longrightarrow$ *transpos i n* (*Suc n*) = *Suc n*
by (*simp add*:*transpos-def*)

lemma *transpos-ij-1*:$[\![i \leq n;\ j \leq n;\ i \neq j\ ]\!] \Longrightarrow$
transpos i j i = *j*
by (*simp add*:*transpos-def*)

lemma *transpos-ij-2*:$[\![i \leq n;\ j \leq n;\ i \neq j]\!] \Longrightarrow$ *transpos i j j* = *i*
by (*simp add*:*transpos-def*)

lemma *transpos-hom*:$[\![i \leq n;\ j \leq n;\ i \neq j]\!] \Longrightarrow$
(*transpos i j*) $\in \{i.\ i \leq n\} \to \{i.\ i \leq n\}$
apply (*simp add*:*Pi-def*, *rule allI*, *rule impI*)
apply (*case-tac x* = *i*, *simp add*:*transpos-def*)
apply (*case-tac x* = *j*, *simp add*:*transpos-def*,
subst transpos-id, *assumption*+, *simp*, *assumption*)
done

lemma *transpos-mem*:$[\![i \leq n;\ j \leq n;\ i \neq j;\ l \leq n]\!] \Longrightarrow$
(*transpos i j l*) $\leq n$
apply (*frule transpos-hom* [*of i n j*], *assumption*+,
cut-tac funcset-mem[*of transpos i j* $\{i.\ i \leq n\}$ $\{i.\ i \leq n\}$ *l*])
apply *simp*+
done

lemma *transpos-inj*:$[\![i \leq n;\ j \leq n;\ i \neq j]\!]$
$\Longrightarrow$ *inj-on* (*transpos i j*) $\{i.\ i \leq n\}$
apply (*simp add*:*inj-on-def*, (*rule allI*, *rule impI*)+, *rule impI*,
case-tac x = *i*, *case-tac y* = *j*,
simp add:*transpos-def*)
apply (*simp add*:*transpos-ij-1*, *rule contrapos-pp*, *simp*+,
frule-tac x = *y* **in** *transpos-id* [*of i n j*], *assumption*+,
simp+)
apply (*case-tac x* = *j*, *simp*,
simp add:*transpos-ij-2*, *rule contrapos-pp*, *simp*+,
frule-tac x = *y* **in** *transpos-id* [*of i n j*], *assumption*+,
simp, *rule contrapos-pp*, *simp*+, *simp add*:*transpos-ij-1*)
apply (*simp*, *simp add*:*transpos-ij-1*, *simp add*:*transpos-id-1*,
thin-tac x = *transpos i j y*,
case-tac y = *i*, *simp add*:*transpos-ij-1*,
case-tac y = *j*, *simp add*:*transpos-ij-2*)
apply (*simp add*:*transpos-id-1*)
done

lemma *transpos-surjec*:$[\![i \leq n;\ j \leq n;\ i \neq j]\!]$
$\Longrightarrow$ *surj-to* (*transpos i j*) $\{i.\ i \leq n\}$ $\{i.\ i \leq n\}$
apply (*simp add*:*surj-to-def*,
frule transpos-hom [*of i n j*], *assumption*+,

frule image-sub [*of transpos i j* {*i*. $i \leq n$} {*i*. $i \leq n$}
{*i*. $i \leq n$}], *simp*)
apply (*frule transpos-inj* [*of i n j*], *assumption*+,
frule card-image [*of transpos i j* {*i*. $i \leq n$}],
simp add:*card-seteq*)
done

lemma *comp-transpos*:⟦$i \leq n$; $j \leq n$; $i \neq j$⟧ ⟹
$\forall k \leq n$. (*compose* {*i*. $i \leq n$} (*transpos i j*) (*transpos i j*)) $k = k$
proof −
assume *p1*:$i \leq n$ **and** *p2*:$j \leq n$ **and** *p3*:$i \neq j$
from *p1* **and** *p2* **and** *p3* **show** *?thesis*
apply (*simp add*:*compose-def*)
apply (*rule allI*)
apply (*case-tac* $k = i$) **apply** *simp*
apply (*subst transpos-ij-1*, *assumption*+)
apply (*rule transpos-ij-2*, *simp*+)
apply (*rule impI*)
apply (*case-tac* $k = j$) **apply** *simp*
apply (*subst transpos-ij-2*, *simp*+)
apply (*rule transpos-ij-1*, *simp*+)
apply (*subst transpos-id-1*, *assumption*+)
apply (*simp add*:*transpos-mem*)
apply (*simp add*:*transpos-id-1*)+
done
qed

lemma *comp-transpos-1*:⟦$i \leq n$; $j \leq n$; $i \neq j$; $k \leq n$⟧ ⟹
(*transpos i j*) ((*transpos i j*) k) $= k$
apply (*frule comp-transpos* [*of i n j*], *assumption*+)
apply (*simp add*:*compose-def*)
done

lemma *cmp-transpos1*:⟦$i \leq n$; $j \leq n$; $i \neq j$; $k \leq n$⟧ ⟹
(*cmp* (*transpos i j*) (*transpos i j*)) $k = k$
apply (*simp add*:*cmp-def*)
apply (*simp add*:*comp-transpos-1*)
done

lemma *cmp-transpos*:⟦$i \leq n$; $i \neq n$; $a \leq$ (*Suc n*)⟧ ⟹
(*cmp* (*transpos i n*) (*cmp* (*transpos n* (*Suc n*)) (*transpos i n*))) $a =$
transpos i (*Suc n*) a
apply (*simp add*:*cmp-def*)
apply (*case-tac* $a =$ *Suc n*, *simp*)
apply (*simp add*:*transpos-id-2*)
apply (*cut-tac transpos-ij-2*[*of n Suc n Suc n*], *simp*,
cut-tac transpos-ij-2[*of i Suc n Suc n*], *simp*,
cut-tac transpos-ij-2[*of i n n*], *simp*+)
apply (*frule le-imp-less-or-eq*[*of a Suc n*],

thin-tac $a \leq Suc\ n$, *simp*,
frule Suc-less-le[*of a n*])
apply (*case-tac* $a = n$, *simp*,
cut-tac transpos-ij-2[*of i n n*], *simp*,
cut-tac transpos-id[*of i Suc n Suc n n*], *simp*,
cut-tac transpos-id[*of n Suc n Suc n i*], *simp*,
cut-tac transpos-ij-1[*of i n n*], *simp*+)
apply (*case-tac* $a = i$, *simp*,
cut-tac transpos-ij-1[*of i n n*], *simp*+,
cut-tac transpos-ij-1[*of i Suc n Suc n*], *simp*,
cut-tac transpos-ij-1[*of n Suc n Suc n*], *simp*,
cut-tac transpos-id[*of i Suc n n Suc n*], *simp*+)
apply (*cut-tac transpos-id*[*of i n n a*], *simp*,
cut-tac transpos-id[*of i Suc n Suc n a*], *simp*,
cut-tac transpos-id[*of n Suc n Suc n a*], *simp*+)
done

lemma *im-Nset-Suc*:*insert* $(f\ (Suc\ n))\ (f\ `\ \{i.\ i \leq n\}) = f\ `\ \{i.\ i \leq (Suc\ n)\}$
apply (*simp add*:*image-def*)
apply (*rule equalityI*)
apply (*rule subsetI*, *simp*)
apply (*erule disjE*, *blast*)
apply (*erule exE*, *erule conjE*, *simp*,
frule-tac $i = xa$ **and** $j = n$ **and** $k = Suc\ n$ **in** *le-trans*,
simp)
apply *blast*
apply (*rule subsetI*, *simp*, *erule exE*, *erule conjE*)
apply (*case-tac* $xa = Suc\ n$, *simp*)
apply (*metis le-SucE*)
done

lemma *Nset-injTr0*:$[\![f \in \{i.\ i \leq (Suc\ n)\} \to \{i.\ i \leq (Suc\ n)\};$
$inj\text{-}on\ f\ \{i.\ i \leq (Suc\ n)\};\ f\ (Suc\ n) = Suc\ n]\!] \Longrightarrow$
$f \in \{i.\ i \leq n\} \to \{i.\ i \leq n\} \wedge inj\text{-}on\ f\ \{i.\ i \leq n\}$
proof −
assume *p1*:$f \in \{i.\ i \leq (Suc\ n)\} \to \{i.\ i \leq (Suc\ n)\}$ **and**
p2:$inj\text{-}on\ f\ \{i.\ i \leq (Suc\ n)\}$ **and** *p3*:$f\ (Suc\ n) = Suc\ n$
have *q1*:$\forall\, l \leq n.\ l \leq (Suc\ n)$ **apply** *simp* **done**
from *p1* **and** *p2* **and** *p3* **and** *q1* **have** *q2*:$f \in \{i.\ i \leq n\} \to \{i.\ i \leq n\}$
apply (*simp add*:*Pi-def*)
apply (*rule allI*, *rule impI*)
apply (*frule-tac* $a = x$ **in** *forall-spec*, *simp*,
thin-tac $\forall\, x \leq Suc\ n.\ f\ x \leq Suc\ n$)
apply (*rule contrapos-pp*, *simp*+)
apply (*simp add*:*nat-not-le-less*)
apply (*frule-tac* $n = f\ x$ **in** *Suc-leI*[*of n*], *thin-tac* $n < (f\ x)$)
apply (*frule-tac* $m = Suc\ n$ **and** $n = f\ x$ **in** *le-antisym*, *assumption*)
apply(*unfold inj-on-def*)
apply (*frule-tac* $x = x$ **in** *bspec*, *simp*,

```
      thin-tac ∀ x∈{i. i ≤ Suc n}. ∀ y∈{i. i ≤ Suc n}. f x = f y ⟶ x = y,
       frule-tac x = Suc n in bspec, simp)
  apply (frule-tac r = f (Suc n) and s = Suc n and t = f x in trans,
        assumption,
        thin-tac f (Suc n) = Suc n, thin-tac Suc n = f x,
        thin-tac ∀ y∈{i. i ≤ Suc n}. f x = f y ⟶ x = y)
  apply simp
done
from p2 have q3:inj-on f {i. i ≤ n}
   apply (simp add:inj-on-def) done
from q2 and q3 show ?thesis apply simp done
qed

lemma inj-surj:⟦f ∈ {i. i ≤ (n::nat)} → {i. i ≤ n};
              inj-on f {i. i ≤ (n::nat)}⟧ ⟹ f ' {i. i ≤ n} = {i. i ≤ n}
proof −
 assume p1:f ∈ {i. i ≤ n} → {i. i ≤ n} and p2:inj-on f {i. i ≤ n}
 have q1:0 < Suc 0 apply simp done
 from p1 and p2 and q1 show ?thesis
 apply simp
 apply (frule image-sub [of f {i. i ≤ n} {i. i ≤ n} {i. i ≤ n}])
 apply simp+
 apply (cut-tac card-image [of f {i. i ≤ n}])
 apply (simp add:card-seteq) apply assumption
 done
qed

lemma Nset-pre-mem:⟦f:{i. i≤(Suc n)} →{i. i≤(Suc n)};
     inj-on f {i. i≤(Suc n)}; f (Suc n) = Suc n; k ≤ n⟧ ⟹ f k ∈ {i. i≤n}
apply (frule Nset-injTr0[of f n], assumption+, erule conjE)
apply (frule-tac x = k in funcset-mem[of f {i. i ≤ n} {i. i ≤ n}],
      simp, assumption)
done

lemma Nset-injTr1:⟦ ∀ l ≤(Suc n). f l ≤ (Suc n); inj-on f {i. i ≤ (Suc n)};
                  f (Suc n) = Suc n ⟧ ⟹ inj-on f {i. i ≤ n}
by (cut-tac Nset-injTr0[of f n], simp, simp)

lemma Nset-injTr2:⟦∀ l≤ (Suc n). f l ≤ (Suc n); inj-on f {i. i ≤ (Suc n)};
                  f (Suc n) = Suc n⟧ ⟹ ∀ l ≤ n. f l ≤ n
apply (rule allI, rule impI)
apply (cut-tac k = l in Nset-pre-mem[of f n])
 apply (simp+)
done

lemma TR-inj-inj:⟦∀ l≤ (Suc n). f l ≤ (Suc n); inj-on f {i. i ≤ (Suc n)};
                  i ≤ (Suc n); j ≤ (Suc n); i < j ⟧ ⟹
     inj-on (compose {i. i ≤ (Suc n)} (transpos i j) f) {i. i ≤ (Suc n)}
apply (frule transpos-inj[of i Suc n j], assumption+,
```

```
      simp )
apply (rule  comp-inj [of f {i. i ≤ (Suc n)} {i. i ≤ (Suc n)}
           transpos i j {i. i ≤ (Suc n)}])
 apply (simp, assumption,
       rule transpos-hom[of i Suc n j], simp+)
done

lemma enumeration:⟦f ∈ {i. i ≤ (n::nat)} → {i. i ≤ m}; inj-on f {i. i ≤ n}⟧
                  ⟹ n ≤ m
apply (frule image-sub[of f {i. i ≤ n} {i. i ≤ m} {i. i ≤ n}])
 apply simp
apply (frule card-image[of f {i. i ≤ n}])
apply(drule card-mono[OF finite-Collect-le-nat])
apply simp
done

lemma enumerate-1:⟦∀ j ≤ (n::nat). f j ∈ A; ∀ j ≤ (m::nat). g j ∈ A;
     inj-on f {i. i ≤ n}; inj-on g {j. j ≤ m}; f ‘{j. j ≤ n} = A;
     g ‘ {j. j ≤ m} = A ⟧ ⟹ n = m
apply (frule card-image[of f {i. i ≤ n}],
       frule card-image[of g {i. i ≤ m}])
apply simp
done

definition
  ninv :: [nat, (nat ⇒ nat)] ⇒ (nat ⇒ nat) where
  ninv n f = (λy∈{i. i ≤ n}. (SOME x. (x ≤ n ∧ y = f x)))

lemma ninv-hom:⟦f ∈ {i. i ≤ n} → {i. i ≤ n}; inj-on f {i. i ≤ n}⟧ ⟹
                    ninv n f ∈ {i. i ≤ n} → {i. i ≤ n}
apply (rule Pi-I)
apply (simp add:ninv-def)
apply (frule inj-surj[of f n], assumption+,
       frule-tac x = x in funcset-mem[of f {i. i ≤ n} {i. i ≤ n}],
       simp)
apply (frule sym, thin-tac f ‘ {i. i ≤ n} = {i. i ≤ n},
       cut-tac a = x and A = {i. i ≤ n} and B = f ‘ {i. i ≤ n} in
       eq-set-inc, simp, assumption,
       thin-tac f x ∈ {i. i ≤ n}, thin-tac {i. i ≤ n} = f ‘ {i. i ≤ n},
       simp add:image-def, rule someI2-ex)
  apply blast+
done

lemma ninv-r-inv:⟦f ∈ {i. i ≤ (n::nat)} → {i. i ≤ n}; inj-on f {i. i ≤ n};
     b ≤ n⟧ ⟹ f (ninv n f b) = b
apply (simp add:ninv-def)
  apply (frule inj-surj, assumption+)
  apply (cut-tac a = b in eq-set-inc[of - {i. i ≤ n} f ‘ {i. i ≤ n}])
  apply (simp, rule sym, assumption)
```

apply (*thin-tac f ' {i. i ≤ n} = {i. i ≤ n}*, *simp add*:*image-def*,
erule exE, *erule conjE*, *frule sym*, *thin-tac b = f x*)
apply (*rule someI2-ex*, *blast*)
apply (*erule conjE*, *rule sym*, *assumption*)
done

lemma *ninv-inj*:⟦*f ∈ {i. i ≤ n} → {i. i ≤ n}*; *inj-on f {i. i ≤ n}*⟧ ⟹
inj-on (*ninv n f*) *{i. i ≤ n}*
apply (*subst inj-on-def*, *simp*)
apply ((*rule allI*, *rule impI*)+, *rule impI*)
apply (*frule ninv-hom*[*of f n*], *assumption*,
frule-tac x = x **in** *funcset-mem*[*of ninv n f {i. i ≤ n} {i. i ≤ n}*], *simp*,
frule-tac x = y **in** *funcset-mem*[*of ninv n f {i. i ≤ n} {i. i ≤ n}*],
simp,
frule-tac b = x **in** *ninv-r-inv* [*of f n*], *assumption*+)
apply (*simp add*:*ninv-r-inv*)
done

1.9.1 Lemmas required in Algebra6.thy

lemma *ge2-zmult-pos*:
*2 ≤ m ⟹ 0 < z ⟹ 1 < int m * z*
proof −
assume *a1*: *0 < z*
assume *a2*: *2 ≤ m*
have *int m + − 1 * (int m * z) ≤ 0*
using *a1* **by** (*simp add*: *pos-zmult-pos*)
then show *?thesis*
using *a2* **by** *linarith*
qed

lemma *zmult-pos-mono*:⟦ (*0*::*int*) < *w*; *w * z ≤ w * z'*⟧ ⟹ *z ≤ z'*
apply (*rule contrapos-pp*, *simp*+)
done

lemma *zmult-pos-mono-r*:
⟦(*0*::*int*) < *w*; *z * w ≤ z' * w*⟧ ⟹ *z ≤ z'*
apply (*simp add*:*mult.commute*)
done

lemma *an-neq-inf*:*an n ≠ ∞*
by (*simp add*:*an-def*)

lemma *an-neq-minf*:*an n ≠ −∞*
by (*simp add*:*an-def*)

lemma *aeq-mult*:⟦*z ≠ 0*; *a = b*⟧ ⟹ *a * ant z = b * ant z*
by *simp*

lemma *tna-0*[*simp*]:*tna 0 = 0*
by (*simp add*:*ant-0*[*THEN sym*] *tna-ant*)

lemma *ale-nat-le*:(*an n* ≤ *an m*) = (*n* ≤ *m*)
by (*simp add*:*an-def*)

lemma *aless-nat-less*:(*an n* < *an m*) = (*n* < *m*)
by (*simp add*:*an-def*, *subst aless-zless*[*of int n int m*], *simp*)

lemma *apos-natpos*:⟦*a* ≠ ∞; *0* ≤ *a*⟧ ⟹ *0* ≤ *na a*
by (*cut-tac ale-nat-le*[*of 0 na a*], *simp add*:*na-def an-def*)

lemma *apos-tna-pos*:⟦*n* ≠ ∞; *0* ≤ *n*⟧ ⟹ *0* ≤ *tna n*
by (*subst tna-0*[*THEN sym*],
subst ale-zle[*THEN sym*, *of tna 0 tna n*],
frule apos-neq-minf[*of n*],
simp add:*ant-tna ant-0*)

lemma *apos-na-pos*:⟦*n* ≠ ∞; *0* ≤ *n*⟧ ⟹ *0* ≤ *na n*
by (*frule apos-tna-pos*[*of n*], *assumption*,
cut-tac tna-0[*THEN sym*], *simp del*:*tna-0*)

lemma *aposs-tna-poss*:⟦*n* ≠ ∞; *0* < *n*⟧ ⟹ *0* < *tna n*
apply (*subst tna-0*[*THEN sym*],
subst aless-zless[*THEN sym*, *of tna 0 tna n*],
frule aless-imp-le[*of 0 n*],
frule apos-neq-minf[*of n*],
simp add:*ant-tna ant-0*)
done

lemma *aposs-na-poss*:⟦*n* ≠ ∞; *0* < *n*⟧ ⟹ *0* < *na n*
apply (*frule aless-imp-le*[*of 0 n*],
simp add:*aneg-less*[*THEN sym*, *of 0 n*],
simp add:*na-def*)
apply (*rule aposs-tna-poss*, *assumption*+)
done

lemma *nat-0-le*: *0* ≤ *z* ==> *int* (*nat z*) = *z*
apply *simp*
done

lemma *int-eq*:*m* = *n* ⟹ *int m* = *int n*
by *simp*

lemma *box-equation*:⟦*a* = *b*; *a* = *c*⟧ ⟹ *b* = *c*
apply *simp*
done

lemma *aeq-nat-eq*:$[\![n \neq \infty;\ 0 \leq n;\ m \neq \infty;\ 0 \leq m]\!] \Longrightarrow$
$(n = m) = (na\ n = na\ m)$
apply (*rule iffI*, *simp*)
apply (*cut-tac aneg-less*[*THEN sym*, *of 0 n*],
cut-tac aneg-less[*THEN sym*, *of 0 m*], *simp*,
simp add:*na-def*,
frule apos-neq-minf[*of n*],
frule apos-neq-minf[*of m*])
apply (*cut-tac mem-ant*[*of m*],
cut-tac mem-ant[*of n*], *simp*,
(*erule exE*)+, *simp*,
simp add:*tna-ant*,
simp only:*ant-0*[*THEN sym*],
simp only:*ale-zle*)
done

lemma *na-minf*:$na\ (-\infty) = 0$
apply (*simp add*:*na-def*, *rule impI*,
cut-tac minf-less-0, *simp*)
done

lemma *an-na*:$[\![a \neq \infty;\ 0 \leq a]\!] \Longrightarrow an\ (na\ a) = a$
apply (*frule apos-tna-pos*[*of a*], *assumption*,
frule apos-neq-minf[*of a*],
cut-tac mem-ant[*of a*], *simp*, *erule exE*,
simp, *simp add*:*an-def na-def*)
apply (*cut-tac* $y = 0$ **and** $x = ant\ z$ **in** *aneg-less*, *simp*,
simp only:*ant-0*[*THEN sym*],
simp only:*ale-zle*, *simp add*:*tna-ant*)
done

lemma *not-na-le-minf*:$\neg\ (an\ n \leq -\infty\ )$
apply (*rule contrapos-pp*, *simp*+)
apply (*cut-tac minf-le-any*[*of an n*], *frule ale-antisym*[*of an n* $-\infty$],
assumption+, *simp add*:*an-def*)
done

lemma *not-na-less-minf*:$\neg\ (an\ n < -\infty)$
apply (*simp add*:*aneg-less*)
done

lemma *not-na-ge-inf*:$\neg\ \infty \leq (an\ n)$
apply (*simp add*:*aneg-le*, *unfold an-def*)
apply (*simp add*:*z-less-i*[*of int n*])
done

lemma *an-na-le*:$j \leq an\ n \Longrightarrow na\ j \leq n$
apply (*case-tac* $j = -\infty$, *simp add*:*na-minf*)
apply (*simp add*:*na-def*)

apply (*case-tac* $j = \infty$, *simp*, *rule impI*)
apply (*cut-tac not-na-ge-inf*[*of n*], *simp*)

apply *simp*
apply (*rule impI*, *simp add*:*aneg-less*)
apply (*frule an-na*[*of j*], *assumption*)
apply (*subgoal-tac nat* (*tna j*) = *na j*, *simp*,
thin-tac nat (*tna j*) = *na j*)
apply (*cut-tac ale-trans*[*of an* (*na j*) *j an n*], *thin-tac* $j \leq$ *an n*,
thin-tac an (*na j*) = *j*, *simp add*:*ale-nat-le*[*of na j n*],
simp add:*ale-refl*[*of j*], *assumption*)
apply (*thin-tac an* (*na j*) = *j*, *simp add*:*na-def*,
rule impI)
apply (*simp add*:*aneg-le*[*THEN sym*, *of j 0*])
done

lemma *aless-neq* :(*x*::*ant*) < *y* $\Longrightarrow$ *x* $\neq$ *y*
by (*rule contrapos-pp*, *simp*+)

Chapter 2

Ordered Set

2.1 Basic Concepts of Ordered Sets

record $'a$ *carrier* =
  carrier :: $'a$ *set*

record $'a$ *Order* = $'a$ *carrier* +
  rel :: $('a \times 'a)$ *set*

locale *Order* =
  fixes *D* (**structure**)
  assumes *closed*: *rel D* $\subseteq$ *carrier D* $\times$ *carrier D*
    and *refl*: *a* $\in$ *carrier D* $\Longrightarrow$ (*a*, *a*) $\in$ *rel D*
    and *antisym*: $[\![a \in carrier\ D;\ b \in carrier\ D;\ (a, b) \in rel\ D;$
        $(b, a) \in rel\ D]\!] \Longrightarrow a = b$
    and *trans*: $[\![a \in carrier\ D;\ b \in carrier\ D;\ c \in carrier\ D;$
        $(a, b) \in rel\ D;\ (b, c) \in rel\ D]\!] \Longrightarrow (a, c) \in rel\ D$

definition
  ole :: - $\Rightarrow$ $'a \Rightarrow 'a \Rightarrow$ *bool* (**infix** $\preceq_1$ *60*) **where**
  $a \preceq_D b \longleftrightarrow (a, b) \in rel\ D$

definition
  oless :: - $\Rightarrow$ $'a \Rightarrow 'a \Rightarrow$ *bool* (**infix** $\prec_1$ *60*) **where**
  $a \prec_D b \equiv a \preceq_D b \wedge a \neq b$

lemma *Order-component*:$(E::'a\ Order) = (\!|\ carrier = carrier\ E,\ rel = rel\ E\ |\!)$
by *simp*

lemma *Order-comp-eq*:$[\![carrier\ (E::'a\ Order) = carrier\ (F::'a\ Order);$
    $rel\ E = rel\ F]\!] \Longrightarrow E = F$
by *simp*

lemma (**in** *Order*) *le-rel*:$[\![a \in carrier\ D;\ b \in carrier\ D]\!] \Longrightarrow$
$(a \preceq b) = ((a,\ b) \in rel\ D)$
by (*simp add*:*ole-def*)

lemma (**in** *Order*) *less-imp-le*:
$[\![a \in carrier\ D;\ b \in carrier\ D;\ a \prec b\ ]\!] \Longrightarrow a \preceq b$
by (*simp add*:*oless-def*)

lemma (**in** *Order*) *le-refl*:$a \in carrier\ D \Longrightarrow a \preceq a$
apply (*unfold ole-def*)
apply (*rule refl*, *assumption*)
done

lemma (**in** *Order*) *le-antisym*:$[\![a \in carrier\ D;\ b \in carrier\ D;$
$a \preceq b;\ b \preceq a\ ]\!] \Longrightarrow a = b$
apply (*unfold ole-def*)
apply (*rule antisym*)
apply *assumption*+
done

lemma (**in** *Order*) *le-trans*:$[\![a \in carrier\ D;\ b \in carrier\ D;\ c \in carrier\ D;$
$a \preceq b;\ b \preceq c\ ]\!] \Longrightarrow a \preceq c$
apply (*unfold ole-def*)
apply (*rule-tac* $a = a$ **and** $b = b$ **and** $c = c$ **in** *trans*)
apply *assumption*+
done

lemma (**in** *Order*) *less-trans*:$[\![a \in carrier\ D;\ b \in carrier\ D;\ c \in carrier\ D;$
$a \prec b;\ b \prec c\ ]\!] \Longrightarrow a \prec c$
apply (*unfold oless-def*)
apply (*erule conjE*)+
apply (*simp add*:*le-trans*[*of a b c*])
apply (*rule contrapos-pp*, *simp*+)
apply (*frule-tac le-antisym*[*of b c*], *assumption*+)
apply *simp*
done

lemma (**in** *Order*) *le-less-trans*:$[\![a \in carrier\ D;\ b \in carrier\ D;\ c \in carrier\ D;$
$a \preceq b;\ b \prec c\ ]\!] \Longrightarrow a \prec c$
apply (*simp add*:*oless-def*)
apply (*erule conjE*)
apply (*simp add*:*le-trans*[*of a b c*])
apply (*rule contrapos-pp*, *simp*+)
apply (*frule le-antisym*[*of b c*])
apply *assumption*+
apply *simp*
done

lemma (**in** *Order*) *less-le-trans*:⟦$a \in$ *carrier* D; $b \in$ *carrier* D; $c \in$ *carrier* D;
$a \prec b$; $b \preceq c$ ⟧ $\Longrightarrow$ $a \prec c$
apply (*simp add*:*oless-def*)
apply (*erule conjE*)
apply (*simp add*:*le-trans*[*of a b c*])
apply (*rule contrapos-pp*, *simp*+)
apply (*frule le-antisym*[*of b c*])
apply *assumption*+
apply *simp*
done

lemma (**in** *Order*) *le-imp-less-or-eq*:
⟦$a \in$ *carrier* D; $b \in$ *carrier* D⟧ $\Longrightarrow$ $(a \preceq b) = (a \prec b \vee a = b)$
apply (*simp add*:*oless-def*)
apply (*rule iffI*)
apply *simp*
apply (*erule disjE*)
apply *simp*
apply *simp*
apply (*rule le-refl*)
apply *assumption*
done

lemma (**in** *Order*) *less-neq*: $a \prec b \Longrightarrow a \neq b$
by (*simp add*: *oless-def*)

lemma (**in** *Order*) *le-neq-less*: ⟦$a \preceq b$; $a \neq b$⟧ $\Longrightarrow$ $a \prec b$
by (*simp add*: *oless-def*)

lemma (**in** *Order*) *less-irrefl*: ⟦$a \in$ *carrier* D; $a \prec a$⟧ $\Longrightarrow$ C
by (*simp add*:*oless-def*)

lemma (**in** *Order*) *less-irrefl'*: $a \in$ *carrier* $D \Longrightarrow \neg\ a \prec a$
by (*simp add*:*oless-def*)

lemma (**in** *Order*) *less-asym*:
$a \in$ *carrier* $D \Longrightarrow b \in$ *carrier* $D \Longrightarrow a \prec b \Longrightarrow b \prec a \Longrightarrow C$
apply (*simp add*:*oless-def*)
apply (*erule conjE*)+
apply (*frule le-antisym*[*of a b*])
apply *assumption*+
apply *simp*
done

lemma (**in** *Order*) *less-asym'*:
$a \in$ *carrier* $D \Longrightarrow b \in$ *carrier* $D \Longrightarrow a \prec b \Longrightarrow \neg\ b \prec a$
apply (*rule contrapos-pp*, *simp*+)
apply (*simp add*:*oless-def*)
apply (*erule conjE*)+

apply (*frule le-antisym*[*of a b*])
apply *assumption*+
apply *simp*
done

lemma (**in** *Order*) *gt-than-any-outside*:⟦$A \subseteq$ *carrier D*; $b \in$ *carrier D*;
$\forall x{\in}A.\ x \prec b$⟧ $\Longrightarrow b \notin A$
apply (*rule contrapos-pp*, *simp*+)
apply (*frule-tac* $x = b$ **in** *bspec*)
apply (*assumption*,
thin-tac $\forall x{\in}A.\ x \prec b$, *simp add*:*oless-def*)
done

definition
Iod :: - ⇒ $'a$ *set* ⇒ - **where**
Iod D T =
D (|*carrier* := *T*, *rel* := {(*a*, *b*). (*a*, *b*) ∈ *rel D* ∧ *a* ∈ *T* ∧ *b* ∈ *T*}|)

definition
SIod :: $'a$ *Order* ⇒ $'a$ *set* ⇒ $'a$ *Order* **where**
SIod D T = (|*carrier* = *T*, *rel* = {(*a*, *b*). (*a*, *b*) ∈ *rel D* ∧ *a* ∈ *T* ∧ *b* ∈ *T*}|)

lemma (**in** *Order*) *Iod-self*: *D* = *Iod D* (*carrier D*)
apply (*unfold Iod-def*)
apply (*cases D*)
apply (*insert closed*)
apply (*simp add*:*Iod-def*)
apply (*rule equalityI*)
apply (*rule subsetI*)
apply *auto*
done

lemma *SIod-self*:*Order D* ⟹ *D* = *SIod D* (*carrier D*)
apply (*unfold SIod-def*)
apply (*cases D*)
apply (*cut-tac Order.closed*[*of D*])
apply *auto*
done

lemma (**in** *Order*) *Od-carrier*:*carrier* (*D*(|*carrier* := *S*, *rel* := *R*|)) = *S*
by *simp*

lemma (**in** *Order*) *Od-rel*:*rel* (*D*(|*carrier* := *S*, *rel* := *R*|)) = *R*
by *simp*

lemma (**in** *Order*) *Iod-carrier*:
T ⊆ *carrier D* ⟹ *carrier* (*Iod D T*) = *T*
by (*simp add*: *Iod-def*)

lemma *SIod-carrier*:$[\![Order\ D;\ T \subseteq carrier\ D]\!] \Longrightarrow carrier\ (SIod\ D\ T) = T$
by (*simp add*:*SIod-def*)

lemma (**in** *Order*) *Od-compare*:$(S = S' \wedge R = R') = (D(\!|carrier := S,\ rel := R|\!)$
$= D(\!|carrier := S',\ rel := R'|\!))$
apply (*rule iffI*)
apply *simp*

apply (*cut-tac Od-carrier*[*of R S*], *cut-tac Od-carrier*[*of R' S'*], *simp*)
apply (*cut-tac Od-rel*[*of R S*], *cut-tac Od-rel*[*of R' S'*])
apply (*thin-tac* $S' = S$)
apply *simp*
done

lemma (**in** *Order*) *Iod-le*:
$[\![T \subseteq carrier\ D;\ a \in T;\ b \in T]\!] \Longrightarrow (a \preceq_{Iod\ D\ T} b) = (a \preceq b)$
apply (*simp add*: *Iod-def*)
apply (*simp add*:*ole-def*)
done

lemma *SIod-le*:$[\![T \subseteq carrier\ D;\ a \in T;\ b \in T]\!] \Longrightarrow$
$(a \preceq_{SIod\ D\ T} b) = (a \preceq_D b)$
apply (*simp add*:*SIod-def*)
apply (*simp add*:*ole-def*)
done

lemma (**in** *Order*) *Iod-less*:
$[\![T \subseteq carrier\ D;\ a \in T;\ b \in T]\!] \Longrightarrow (a \prec_{Iod\ D\ T} b) = (a \prec b)$
apply (*simp add*:*oless-def*)
apply (*simp add*:*Iod-le*)
done

lemma *SIod-less*:$[\![T \subseteq carrier\ D;\ a \in T;\ b \in T]\!] \Longrightarrow$
$(a \prec_{SIod\ D\ T} b) = (a \prec_D b)$
by (*simp add*:*oless-def SIod-le*)

lemma (**in** *Order*) *Iod-Order*:
$T \subseteq carrier\ D \Longrightarrow Order\ (Iod\ D\ T)$
apply (*rule Order.intro*)
apply (*simp add*:*Iod-def*)
apply (*rule subsetI*)
apply (*unfold split-paired-all*)
apply *simp*
apply (*simp add*:*Iod-carrier*)
apply (*simp add*:*Iod-def*)
apply (*rule refl*)
apply (*rule subsetD*, *assumption*+)
apply (*simp add*:*Iod-carrier*)

apply (*simp add*:*Iod-def*)
apply (*rule-tac* $a = a$ **and** $b = b$ **in** *antisym*)
apply (*simp add*:*subsetD*[*of* T *carrier* D])+
apply (*simp add*:*Iod-def*)
apply (*rule-tac* $a = a$ **and** $b = b$ **and** $c = c$ **in** *trans*)
apply (*simp add*:*subsetD*[*of* T *carrier* D])+
done

lemma *SIod-Order*:$[\![$ *Order* D; $T \subseteq$ *carrier* $D]\!] \Longrightarrow$ *Order* (*SIod* D T)
apply (*rule Order.intro*)
apply (*rule subsetI*)
apply (*simp add*:*SIod-def*)
apply (*unfold split-paired-all*)
apply *simp*
apply (*simp add*:*SIod-def*)
apply (*frule-tac* $c = a$ **in** *subsetD*[*of* T *carrier* D], *assumption*+)
apply (*simp add*:*Order.refl*[*of* D])

apply (*simp add*:*SIod-def*)
apply (*rule Order.antisym*[*of* D], *assumption*+)
apply (*simp add*:*subsetD*)+

apply (*simp add*:*SIod-def*)
apply (*frule-tac* $c = a$ **in** *subsetD*[*of* T *carrier* D], *assumption*+,
frule-tac $c = b$ **in** *subsetD*[*of* T *carrier* D], *assumption*+,
frule-tac $c = c$ **in** *subsetD*[*of* T *carrier* D], *assumption*+)
apply (*rule-tac* $a = a$ **and** $b = b$ **and** $c = c$ **in** *Order.trans*[*of* D], *assumption*+)
done

lemma (**in** *Order*) *emptyset-Iod*:*Order* (*Iod* D {})
apply (*rule Iod-Order*)
apply *simp*
done

lemma (**in** *Order*) *Iod-sub-sub*:
$[\![S \subseteq T;\ T \subseteq$ *carrier* $D]\!] \Longrightarrow$ *Iod* (*Iod* D T) S = *Iod* D S
apply (*simp add*:*Iod-def*)
apply (*subst Od-compare*[*THEN sym*])
apply *simp*
apply *blast*
done

lemma *SIod-sub-sub*:
$[\![S \subseteq T;\ T \subseteq$ *carrier* $D]\!] \Longrightarrow$ *SIod* (*SIod* D T) S = *SIod* D S
apply (*simp add*:*SIod-def*)
apply *blast*
done

lemma *rel-SIod*:$[\![$*Order* D; *Order* E; *carrier* $E \subseteq$ *carrier* D;

$\forall\, a{\in}carrier\ E.\ \forall\, b{\in}carrier\ E.\ (a \preceq_E b) = (a \preceq_D b)]\!] \Longrightarrow$
$rel\ E = rel\ (SIod\ D\ (carrier\ E))$
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*unfold split-paired-all*)
apply (*simp add*:*ole-def*)
apply (*simp add*:*SIod-def*)
apply (*cut-tac Order.closed*[*of E*])
apply *blast*
apply *assumption*
apply (*rule subsetI*)
apply (*unfold split-paired-all*)
apply (*simp add*:*SIod-def*)
apply (*simp add*:*ole-def*)
done

lemma *SIod-self-le*:[![*Order D*; *Order E*;
carrier E $\subseteq$ *carrier D*;
$\forall\, a{\in}carrier\ E.\ \forall\, b{\in}carrier\ E.\ (a \preceq_E b) = (a \preceq_D b)$]!] $\Longrightarrow$
$E = SIod\ D\ (carrier\ E)$
apply (*rule Order-comp-eq*[*of E SIod D* (*carrier E*)])
apply (*simp add*:*SIod-carrier*)
apply (*rule rel-SIod*[*of D E*], *assumption*+)
done

2.1.1 Total ordering

locale *Torder* = *Order* +
assumes *le-linear*: [![$a \in carrier\ D$; $b \in carrier\ D$]!] $\Longrightarrow$
$a \preceq b \vee b \preceq a$

lemma (**in** *Order*) *Iod-empty-Torder*:*Torder* (*Iod D* {})
apply (*rule Torder.intro*)
apply(*simp add*:*emptyset-Iod*)
apply (*rule Torder-axioms.intro*)
apply (*simp add*:*Iod-carrier*)
done

lemma (**in** *Torder*) *le-cases*:
[![$a \in carrier\ D$; $b \in carrier\ D$; $(a \preceq b \Longrightarrow C)$; $(b \preceq a \Longrightarrow C)$]!] $\Longrightarrow C$
by (*cut-tac le-linear*[*of a b*], *blast*, *assumption*+)

lemma (**in** *Torder*) *Order*:*Order D*
apply (*rule Order-axioms*)
done

lemma (**in** *Torder*) *less-linear*:
$a \in carrier\ D \Longrightarrow b \in carrier\ D \Longrightarrow a \prec b \vee a = b \vee b \prec a$
apply (*simp add*:*oless-def*)

apply (*rule le-cases*[*of a b*])
apply *assumption*+
apply *blast*
apply *blast*
done

lemma (**in** *Torder*) *not-le-less*:
  $a \in$ *carrier* $D \Longrightarrow b \in$ *carrier* $D \Longrightarrow$
    $(\neg\ a \preceq b) = (b \prec a)$
apply (*unfold oless-def*)
apply (*cut-tac le-linear*[*of a b*])
apply (*rule iffI*)
apply *simp*
apply (*rule contrapos-pp*, *simp*+)
apply (*rule contrapos-pp*, *simp*+)
apply (*erule conjE*)
apply (*frule le-antisym*[*of b a*])
apply *assumption*+
apply *simp*+
done

lemma (**in** *Torder*) *not-less-le*:
  $a \in$ *carrier* $D \Longrightarrow b \in$ *carrier* $D \Longrightarrow$
    $(\neg\ a \prec b) = (b \preceq a)$
apply (*unfold oless-def*)
apply (*rule iffI*)
 apply (*simp only*:*de-Morgan-conj*[*of a* $\preceq$ *b a* $\neq$ *b*])
 apply (*simp only*:*not-le-less*[*of a b*])
 apply (*erule disjE*)
  apply (*simp add*:*less-imp-le*)
apply (*simp add*:*le-imp-less-or-eq*)
apply (*rule contrapos-pp*, *simp*+)
 apply (*erule conjE*)
 apply (*frule le-antisym*[*of a b*])
 apply *assumption*+
 apply *simp*
done

lemma (**in** *Order*) *Iod-not-le-less*:⟦$T \subseteq$ *carrier* D; $a \in T$; $b \in T$;
      Torder (*Iod D T*)⟧ $\Longrightarrow$ $(\neg\ a \preceq_{(Iod\ D\ T)} b) = b \prec_{(Iod\ D\ T)} a$
apply (*subst Torder.not-le-less*)
apply *assumption*+
apply (*simp add*:*Iod-carrier*)+
done

lemma (**in** *Order*) *Iod-not-less-le*:⟦$T \subseteq$ *carrier* D; $a \in T$; $b \in T$;
      Torder (*Iod D T*)⟧ $\Longrightarrow$ $(\neg\ a \prec_{(Iod\ D\ T)} b) = b \preceq_{(Iod\ D\ T)} a$
apply (*subst Torder.not-less-le*)
apply *assumption*+

apply (*simp add*:*Iod-carrier*)+
done

2.1.2 Two ordered sets

definition
Order-Pow :: *'a set* ⇒ *'a set Order* ((*po* -) [*999*] *1000*) **where**
po A =
(|*carrier* = *Pow A*,
rel = {(*X*, *Y*). *X* ∈ *Pow A* ∧ *Y* ∈ *Pow A* ∧ *X* ⊆ *Y*}|)

interpretation *order-Pow*: *Order po A*
apply (*unfold Order-Pow-def*)
apply (*rule Order.intro*)
apply (*rule subsetI*)
apply (*unfold split-paired-all*)
apply *simp*
apply *simp*
apply *simp*
apply *simp*
done

definition
Order-fs :: *'a set* ⇒ *'b set* ⇒ (*'a set* ∗ (*'a* ⇒ *'b*)) *Order* **where**
Order-fs A B =
(|*carrier* = {*Z*. ∃ *A1 f*. *A1* ∈ *Pow A* ∧ *f* ∈ *A1* → *B* ∧
f ∈ *extensional A1* ∧ *Z* = (*A1*, *f*)},
rel = {*Y*. *Y* ∈ ({*Z*. ∃ *A1 f*. *A1* ∈ *Pow A* ∧ *f* ∈ *A1* → *B* ∧ *f* ∈ *extensional A1*
∧ *Z* = (*A1*, *f*)}) × ({*Z*. ∃ *A1 f*. *A1* ∈ *Pow A* ∧ *f* ∈ *A1* → *B* ∧ *f* ∈ *extensional*
A1
∧ *Z* = (*A1*, *f*)}) ∧ *fst* (*fst Y*) ⊆ *fst* (*snd Y*) ∧
(∀ *a*∈ (*fst* (*fst Y*)). (*snd* (*fst Y*)) *a* = (*snd* (*snd Y*)) *a*)}|)

lemma *Order-fs*:*Order* (*Order-fs A B*)
apply (*simp add*:*Order-fs-def*)
apply (*rule Order.intro*)
apply (*rule subsetI*)
apply (*unfold split-paired-all*)
apply (*auto intro*: *funcset-eq*)
done

2.1.3 Homomorphism of ordered sets

definition
ord-inj :: [(*'a*, *'m0*) *Order-scheme*, (*'b*, *'m1*) *Order-scheme*,
'a ⇒ *'b*] ⇒ *bool* **where**
ord-inj D E f ⟷ *f* ∈ *extensional* (*carrier D*) ∧
f ∈ (*carrier D*) → (*carrier E*) ∧
(*inj-on f* (*carrier D*)) ∧
(∀ *a*∈*carrier D*. ∀ *b*∈*carrier D*. (*a* ≺$_D$ *b*) = ((*f a*) ≺$_E$ (*f b*)))

definition
ord-isom :: [('*a*, '*m0*) *Order-scheme*, ('*b*, '*m1*) *Order-scheme*,
'*a* ⇒ '*b*] ⇒ *bool* **where**
ord-isom D E f ⟷ *ord-inj D E f* ∧
(*surj-to f* (*carrier D*) (*carrier E*))

lemma (**in** *Order*) *ord-inj-func*:⟦*Order E*; *ord-inj D E f*⟧ ⟹
f ∈ *carrier D* → *carrier E*
by (*simp add*:*ord-inj-def*)

lemma (**in** *Order*) *ord-isom-func*:⟦*Order E*; *ord-isom D E f*⟧ ⟹
f ∈ *carrier D* → *carrier E*
by (*simp add*:*ord-isom-def ord-inj-func*)

lemma (**in** *Order*) *ord-inj-restrict-isom*:⟦*Order E*; *ord-inj D E f*; *T* ⊆ *carrier D*⟧
⟹ *ord-isom* (*Iod D T*) (*Iod E* (*f* ‘ *T*)) (*restrict f T*)
apply (*subst ord-isom-def*)
apply (*frule ord-inj-func*[*of E f*], *assumption*,
frule image-sub[*of f carrier D carrier E T*], *assumption*+)

apply (*rule conjI*)
apply (*subst ord-inj-def*)
apply (*simp add*:*Iod-carrier Order.Iod-carrier*)

apply (*rule conjI*)
apply (*rule restrict-inj*[*of f carrier D T*])
apply (*simp add*:*ord-inj-def*, *assumption*+)

apply (*rule ballI*)+
apply (*frule-tac x* = *a* **in** *elem-in-image2*[*of f carrier D carrier E T*],
assumption+,
frule-tac x = *b* **in** *elem-in-image2*[*of f carrier D carrier E T*],
assumption+)
apply (*simp add*:*Iod-less Order.Iod-less*)
apply (*frule-tac c* = *a* **in** *subsetD*[*of T carrier D*], *assumption*+,
frule-tac c = *b* **in** *subsetD*[*of T carrier D*], *assumption*+)
apply (*simp add*:*ord-inj-def*)

apply (*subst surj-to-def*)
apply (*simp add*:*Iod-carrier Order.Iod-carrier*)
done

lemma *ord-inj-Srestrict-isom*:⟦*Order D*; *Order E*; *ord-inj D E f*; *T* ⊆ *carrier D*⟧
⟹ *ord-isom* (*SIod D T*) (*SIod E* (*f* ‘ *T*)) (*restrict f T*)
apply (*subst ord-isom-def*)
apply (*frule Order.ord-inj-func*[*of D E f*], *assumption*+,
frule image-sub[*of f carrier D carrier E T*], *assumption*+)

apply (*rule conjI*)
apply (*subst ord-inj-def*)
apply (*simp add*:*SIod-carrier*)

apply (*rule conjI*)
  apply (*rule restrict-inj*[*of f carrier D T*])
apply (*simp add*:*ord-inj-def*, *assumption*+)

apply (*rule ballI*)+
apply (*frule-tac x* = *a* **in** *elem-in-image2*[*of f carrier D carrier E T*],
    assumption+,
    frule-tac x = *b* **in** *elem-in-image2*[*of f carrier D carrier E T*],
    assumption+)
apply (*simp add*:*SIod-less*)
apply (*frule-tac c* = *a* **in** *subsetD*[*of T carrier D*], *assumption*+,
    frule-tac c = *b* **in** *subsetD*[*of T carrier D*], *assumption*+)
apply (*simp add*:*ord-inj-def*)

apply (*simp add*:*SIod-carrier*)
apply (*simp add*:*surj-to-def*)
done

lemma (**in** *Order*) *id-ord-isom*:*ord-isom D D* (*idmap* (*carrier D*))
apply (*simp add*:*ord-isom-def*)
apply (*cut-tac idmap-bij*[*of carrier D*])
apply (*simp add*:*bij-to-def*)
apply (*simp add*:*ord-inj-def*)
apply (*simp add*:*idmap-def*[*of carrier D*])
done

lemma (**in** *Order*) *ord-isom-bij-to*:⟦*Order E*; *ord-isom D E f*⟧ ⟹
                bij-to f (*carrier D*) (*carrier E*)
by (*simp add*:*bij-to-def ord-isom-def*,
    simp add:*ord-inj-def*)

lemma (**in** *Order*) *ord-inj-mem*:⟦*Order E*; *ord-inj D E f*; *a* ∈ *carrier D*⟧ ⟹
    (*f a*) ∈ *carrier E*
apply (*simp add*:*ord-inj-def*, (*erule conjE*)+)
apply (*simp add*:*Pi-def*)
done

lemma (**in** *Order*) *ord-isom-mem*:⟦*Order E*; *ord-isom D E f*; *a* ∈ *carrier D*⟧ ⟹
        (*f a*) ∈ *carrier E*
apply (*simp add*:*ord-isom-def*, (*erule conjE*)+)
apply (*simp add*:*ord-inj-mem*)
done

lemma (**in** *Order*) *ord-isom-surj*:⟦*Order E*; *ord-isom D E f*; *b* ∈ *carrier E*⟧ ⟹
    ∃ *a*∈*carrier D*. *b* = *f a*

apply (*simp add*:*ord-isom-def*, (*erule conjE*)+)
apply (*simp add*:*surj-to-def image-def*)
apply (*frule sym*, *thin-tac* $\{y.\ \exists x{\in}carrier\ D.\ y = f\ x\} = carrier\ E$,
simp)
done

lemma (**in** *Order*) *ord-isom-surj-forall*:$[\![Order\ E;\ ord\text{-}isom\ D\ E\ f]\!] \Longrightarrow$
$\forall\, b \in carrier\ E.\ \exists\, a{\in}carrier\ D.\ b = f\ a$
apply (*rule ballI*)
apply (*rule ord-isom-surj*[*of E f*], *assumption*+)
done

lemma (**in** *Order*) *ord-isom-onto*:$[\![Order\ E;\ ord\text{-}isom\ D\ E\ f]\!] \Longrightarrow$
$f\ `\ (carrier\ D) = carrier\ E$
apply (*frule ord-isom-bij-to*[*of E f*], *assumption*+)
apply(*simp add*:*bij-to-def surj-to-def*)
done

lemma (**in** *Order*) *ord-isom-inj-on*:$[\![Order\ E;\ ord\text{-}isom\ D\ E\ f]\!] \Longrightarrow$
$inj\text{-}on\ f\ (carrier\ D)$
by (*simp add*:*ord-isom-def ord-inj-def*)

lemma (**in** *Order*) *ord-isom-inj*:$[\![Order\ E;\ ord\text{-}isom\ D\ E\ f;$
$a \in carrier\ D;\ b \in carrier\ D]\!] \Longrightarrow (a = b) = ((f\ a) = (f\ b))$
apply (*frule ord-isom-inj-on*[*of E f*], *assumption*)
apply (*simp add*:*injective-iff*)
done

lemma (**in** *Order*) *ord-isom-surj-to*:$[\![Order\ E;\ ord\text{-}isom\ D\ E\ f]\!] \Longrightarrow$
$surj\text{-}to\ f\ (carrier\ D)\ (carrier\ E)$
by (*simp add*:*ord-isom-def*)

lemma (**in** *Order*) *ord-inj-less*:$[\![Order\ E;\ ord\text{-}inj\ D\ E\ f;\ a \in carrier\ D;$
$b \in carrier\ D]\!] \Longrightarrow (a \prec_D b) = ((f\ a) \prec_E (f\ b))$
by (*simp add*:*ord-inj-def*)

lemma (**in** *Order*) *ord-isom-less*:$[\![Order\ E;\ ord\text{-}isom\ D\ E\ f;$
$a \in carrier\ D;\ b \in carrier\ D]\!] \Longrightarrow (a \prec_D b) = ((f\ a) \prec_E (f\ b))$
by (*simp add*:*ord-isom-def ord-inj-less*)

lemma (**in** *Order*) *ord-isom-less-forall*:$[\![Order\ E;\ ord\text{-}isom\ D\ E\ f]\!] \Longrightarrow$
$\forall\, a \in carrier\ D.\ \forall\ \ b \in carrier\ D.\ (a \prec_D b) = ((f\ a) \prec_E (f\ b))$
by ((*rule ballI*)+,
simp add:*ord-isom-less*)

lemma (**in** *Order*) *ord-isom-le*:$[\![Order\ E;\ ord\text{-}isom\ D\ E\ f;$
$a \in carrier\ D;\ b \in carrier\ D]\!] \Longrightarrow (a \preceq_D b) = ((f\ a) \preceq_E (f\ b))$
apply (*frule-tac a = a* **in** *ord-isom-mem*[*of E f*], *assumption*+,
frule-tac a = b **in** *ord-isom-mem*[*of E f*], *assumption*+)

apply (*simp add*:*le-imp-less-or-eq Order*.*le-imp-less-or-eq*[*of E*])
apply (*simp add*:*ord-isom-less ord-isom-inj*)
done

lemma (**in** *Order*) *ord-isom-le-forall*:⟦*Order E*; *ord-isom D E f*⟧ ⟹
∀ *a* ∈ *carrier D*. ∀ *b* ∈ *carrier D*. (*a* ⪯ *b*) = ((*f a*) ⪯$_E$ (*f b*))
by ((*rule ballI*)+,
rule ord-isom-le, *assumption*+)

lemma (**in** *Order*) *ord-isom-convert*:⟦*Order E*; *ord-isom D E f*;
x ∈ *carrier D*; *a* ∈ *carrier D*⟧ ⟹ (∀ *y*∈*carrier D*. (*x* ≺ *y* ⟶ ¬ *y* ≺ *a*)) =
(∀ *z*∈*carrier E*. ((*f x*) ≺$_E$ *z* ⟶ ¬ *z* ≺$_E$ (*f a*)))
apply (*rule iffI*)
apply (*rule ballI*, *rule impI*)
apply (*frule-tac b* = *z* **in** *ord-isom-surj*[*of E f*], *assumption*+,
erule bexE)
apply (*simp add*:*ord-isom-less*[*THEN sym*, *of E f*])
apply (*rule ballI*, *rule impI*)
apply (*simp add*:*ord-isom-less*[*of E f*])
apply (*frule-tac a* = *y* **in** *ord-isom-mem*[*of E f*], *assumption*+)
apply *simp*
done

lemma (**in** *Order*) *ord-isom-sym*:⟦*Order E*; *ord-isom D E f*⟧ ⟹
ord-isom E D (*invfun* (*carrier D*) (*carrier E*) *f*)
apply (*frule ord-isom-func*[*of E f*], *assumption*+,
frule ord-isom-inj-on[*of E f*], *assumption*+,
frule ord-isom-surj-to[*of E f*], *assumption*+)

apply (*subst ord-isom-def*, *subst ord-inj-def*)
apply (*simp add*:*inv-func*)
apply (*simp add*:*invfun-inj*)
apply (*simp add*:*invfun-surj*)
apply (*rule conjI*)
apply (*simp add*:*invfun-def extensional-def*)

apply (*rule ballI*)+
apply (*frule-tac b* = *a* **in** *invfun-mem*[*of f carrier D carrier E*],
assumption+,
frule-tac b = *b* **in** *invfun-mem*[*of f carrier D carrier E*],
assumption+)
apply (*frule-tac a* = (f^{-1} *carrier E*,*carrier D*) *a* **and** *b* = (f^{-1} *carrier E*,*carrier D*)
b
in *ord-isom-less*[*of E f*], *assumption*+)
apply (*simp add*:*invfun-r*)
done

lemma (**in** *Order*) *ord-isom-trans*:⟦*Order E*; *Order F*; *ord-isom D E f*;
ord-isom E F g ⟧ ⟹ *ord-isom D F* (*compose* (*carrier D*) *g f*)

apply (*frule ord-isom-func*[*of E f*], *assumption*+,
frule ord-isom-inj-on[*of E f*], *assumption*+,
frule ord-isom-surj-to[*of E f*], *assumption*+,
frule Order.ord-isom-func[*of E F g*], *assumption*+,
frule Order.ord-isom-inj-on[*of E F g*], *assumption*+,
frule Order.ord-isom-surj-to[*of E F g*], *assumption*+)

apply (*subst ord-isom-def*, *subst ord-inj-def*)
apply (*simp add*:*composition*)
apply (*simp add*:*comp-inj*[*of f carrier D carrier E g carrier F*])
apply (*simp add*:*compose-surj*)
apply (*rule ballI*)+

apply (*frule-tac x* = *a* **in** *funcset-mem*[*of f carrier D carrier E*],
assumption+,
frule-tac x = *b* **in** *funcset-mem*[*of f carrier D carrier E*],
assumption+)
apply (*frule-tac a* = *a* **and** *b* = *b* **in** *ord-isom-less*[*of E f*], *assumption*+,
frule-tac a = *f a* **and** *b* = *f b* **in** *Order.ord-isom-less*[*of E F g*],
assumption+)
apply (*simp add*:*compose-def*)
done

definition
ord-equiv :: [-, ('*b*, '*m1*) *Order-scheme*] ⇒ *bool* **where**
ord-equiv D E ⟷ (∃*f*. *ord-isom D E f*)

lemma (**in** *Order*) *ord-equiv*:⟦*Order E*; *ord-isom D E f*⟧ ⟹ *ord-equiv D E*
by (*simp add*:*ord-equiv-def*, *blast*)

lemma (**in** *Order*) *ord-equiv-isom*:⟦*Order E*; *ord-equiv D E*⟧ ⟹
∃*f*. *ord-isom D E f*
by (*simp add*:*ord-equiv-def*)

lemma (**in** *Order*) *ord-equiv-reflex*:*ord-equiv D D*
apply (*simp add*:*ord-equiv-def*)
apply (*cut-tac id-ord-isom*, *blast*)
done

lemma (**in** *Order*) *eq-ord-equiv*:⟦*Order E*; *D* = *E*⟧ ⟹ *ord-equiv D E*
apply (*frule sym*, *thin-tac D* = *E*)
apply (*simp add*:*ord-equiv-reflex*)
done

lemma (**in** *Order*) *ord-equiv-sym*:⟦*Order E*; *ord-equiv D E* ⟧ ⟹ *ord-equiv E D*
apply (*simp add*:*ord-equiv-def*)
apply (*erule exE*,

frule-tac $E = E$ **and** $f = f$ **in** *ord-isom-sym*, *assumption*+, *blast*)
done

lemma (**in** *Order*) *ord-equiv-trans*:⟦*Order E*; *Order F*; *ord-equiv D E*;
ord-equiv E F⟧ ⟹ *ord-equiv D F*
apply (*simp add*:*ord-equiv-def*)
apply (*erule exE*)+
apply (*frule-tac f* = *f* **and** *g* = *fa* **in** *ord-isom-trans* [*of E F*],
assumption+, *blast*)
done

lemma (**in** *Order*) *ord-equiv-box*:⟦*Order E*; *Order F*; *ord-equiv D E*;
ord-equiv D F⟧ ⟹ *ord-equiv E F*
apply (*rule Order.ord-equiv-trans*[*of E D F*])
apply *assumption*
apply (*rule Order-axioms*)
apply *assumption*
apply (*rule ord-equiv-sym*) **apply** *assumption*+
done

lemma *SIod-isom-Iod*:⟦*Order D*; *T* ⊆ *carrier D* ⟧ ⟹
ord-isom (*SIod D T*) (*Iod D T*) (λ*x*∈*T*. *x*)
apply (*simp add*:*ord-isom-def ord-inj-def*)
apply (*simp add*:*SIod-carrier Order.Iod-carrier*)
apply (*rule conjI*)
apply (*fold idmap-def*[*of T*])

apply (*simp add*:*SIod-less Order.Iod-less*)

apply (*cut-tac A* = *T* **in** *idmap-bij*,
simp add:*bij-to-def*)
done

definition
minimum-elem :: [- , *'a set*, *'a*] ⇒ *bool* **where**
minimum-elem = (λ*D X a*. *a* ∈ *X* ∧ (∀ *x*∈*X*. *a* ≼$_D$ *x*))

locale *Worder* = *Torder* +
assumes *ex-minimum*: ∀ *X*. *X* ⊆ (*carrier D*) ∧ *X* ≠ {} ⟶
(∃ *x*. *minimum-elem D X x*)

lemma (**in** *Worder*) *Order*:*Order D*
by (*rule Order*)

lemma (**in** *Worder*) *Torder*:*Torder D*
apply (*rule Torder-axioms*)
done

lemma (**in** *Worder*) *Worder*:*Worder D*

apply (*rule Worder-axioms*)
done

lemma (**in** *Worder*) *equiv-isom*:⟦*Worder E*; *ord-equiv D E*⟧ ⟹
∃ *f*. *ord-isom D E f*
by (*insert Order*, *frule Worder.Order*[*of E*], *simp add*:*ord-equiv-def*)

lemma (**in** *Order*) *minimum-elem-mem*:⟦*X* ⊆ *carrier D*; *minimum-elem D X a*⟧
⟹ *a* ∈ *X*
by (*simp add*:*minimum-elem-def*)

lemma (**in** *Order*) *minimum-elem-unique*:⟦*X* ⊆ *carrier D*; *minimum-elem D X a1*;
minimum-elem D X a2⟧ ⟹ *a1* = *a2*
apply (*frule minimum-elem-mem*[*of X a1*], *assumption*+,
frule minimum-elem-mem[*of X a2*], *assumption*+)
apply (*simp add*:*minimum-elem-def*)
apply (*drule-tac x* = *a2* **in** *bspec*, *assumption*)
apply (*drule-tac x* = *a1* **in** *bspec*, *assumption*)
apply (*rule le-antisym*[*of a1 a2*])
apply (*simp add*:*subsetD*)+
done

lemma (**in** *Order*) *compare-minimum-elements*:⟦*S* ⊆ *carrier D*; *T* ⊆ *carrier D*;
S ⊆ *T*; *minimum-elem D S s*; *minimum-elem D T t* ⟧ ⟹ *t* ≼ *s*
apply (*frule minimum-elem-mem*[*of S s*], *assumption*+)
apply (*frule subsetD*[*of S T s*], *assumption*+)
apply (*simp add*:*minimum-elem-def*)
done

lemma (**in** *Order*) *minimum-elem-sub*:⟦*T* ⊆ *carrier D*; *X* ⊆ *T*⟧
⟹ *minimum-elem D X a* = *minimum-elem* (*Iod D T*) *X a*
apply (*simp add*:*minimum-elem-def*)
apply (*simp add*:*subset-eq*[*of X T*])
apply (*rule iffI*, *erule conjE*)
apply *simp*
apply (*rule ballI*)
apply (*simp add*:*Iod-le*)
apply *simp*
apply (*rule ballI*)
apply (*erule conjE*)
apply (*simp add*:*Iod-le*)
done

lemma *minimum-elem-Ssub*:⟦*Order D*; *T* ⊆ *carrier D*; *X* ⊆ *T*⟧
⟹ *minimum-elem D X a* = *minimum-elem* (*SIod D T*) *X a*
apply (*simp add*:*minimum-elem-def*)

apply (*rule iffI*)

apply *simp*
apply (*rule ballI*, *erule conjE*)
apply (*drule-tac* $x = x$ **in** *bspec*, *assumption*)

apply (*frule-tac* $c = x$ **in** *subsetD*[*of* X T], *assumption*+,
frule-tac $c = a$ **in** *subsetD*[*of* X T], *assumption*+)
apply (*simp add*:*SIod-le*)

apply *simp*
apply (*rule ballI*, *erule conjE*)
apply (*drule-tac* $x = x$ **in** *bspec*, *assumption*)

apply (*frule-tac* $c = x$ **in** *subsetD*[*of* X T], *assumption*+,
frule-tac $c = a$ **in** *subsetD*[*of* X T], *assumption*+)
apply (*simp add*:*SIod-le*)
done

lemma (**in** *Order*) *augmented-set-minimum*:$[\![X \subseteq$ *carrier* D; $a \in$ *carrier* D;
$Y - \{a\} \subseteq X$; $y - \{a\} \neq \{\}$; *minimum-elem* (*Iod* D X) ($Y - \{a\}$) x;
$\forall x{\in}X.\ x \preceq a]\!] \Longrightarrow$ *minimum-elem* (*Iod* D (*insert* a X)) Y x
apply (*frule insert-mono*[*of* $Y - \{a\}$ X a])
apply *simp*
apply (*frule insert-sub*[*of* X *carrier* D a], *assumption*+)
apply (*simp add*:*minimum-elem-sub*[*THEN sym*, *of insert* a X Y],
simp add:*minimum-elem-sub*[*THEN sym*, *of* X $Y - \{a\}$])

apply (*simp add*:*subset-eq*[*of* $Y - \{a\}$ X])

apply (*simp add*:*minimum-elem-def*, (*erule conjE*)+)
apply (*rule ballI*)
apply *blast*
done

lemma *augmented-Sset-minimum*:$[\![$*Order* D; $X \subseteq$ *carrier* D; $a \in$ *carrier* D;
$Y - \{a\} \subseteq X$; $y - \{a\} \neq \{\}$; *minimum-elem* (*SIod* D X) ($Y - \{a\}$) x;
$\forall x{\in}X.\ x \preceq_D a]\!] \Longrightarrow$ *minimum-elem* (*SIod* D (*insert* a X)) Y x
apply (*frule insert-mono*[*of* $Y - \{a\}$ X a])
apply *simp*
apply (*frule insert-sub*[*of* X *carrier* D a], *assumption*+)
apply (*simp add*:*minimum-elem-Ssub*[*THEN sym*, *of* D *insert* a X Y],
simp add:*minimum-elem-Ssub*[*THEN sym*, *of* D X $Y - \{a\}$])

apply (*simp add*:*subset-eq*[*of* $Y - \{a\}$ X])

apply (*simp add*:*minimum-elem-def*, (*erule conjE*)+)
apply (*rule ballI*)
apply *blast*
done

lemma (**in** *Order*) *ord-isom-minimum*:⟦*Order E*; *ord-isom D E f*;
S ⊆ *carrier D*; *a* ∈ *carrier D*; *minimum-elem D S a*⟧ ⟹
minimum-elem E (*f*‘*S*) (*f a*)
apply (*subst minimum-elem-def*,
frule minimum-elem-mem[*of S a*], *assumption*+)
apply (*simp add*:*ord-isom-mem*)
apply (*rule ballI*)
apply (*simp add*:*minimum-elem-def*)
apply (*frule-tac x* = *x* **in** *bspec*, *assumption*,
thin-tac Ball S (*op* ≼ *a*))
apply (*frule-tac b* = *x* **in** *ord-isom-le*[*of E f a*], *assumption*+)
apply (*simp add*:*subsetD*)
apply *simp*
done

lemma (**in** *Worder*) *pre-minimum*:⟦*T* ⊆ *carrier D*; *minimum-elem D T t*;
s ∈ *carrier D*; *s* ≺$_D$ *t* ⟧ ⟹ ¬ *s* ∈ *T*
apply (*rule contrapos-pp*, *simp*+)
apply (*simp add*:*minimum-elem-def*, (*erule conjE*)+)
apply (*frule-tac x* = *s* **in** *bspec*, *assumption*+,
thin-tac ∀ *x*∈*T*. *t* ≼$_D$ *x*)
apply (*simp add*:*oless-def*, *erule conjE*)
apply (*frule le-antisym*[*of s t*])
apply (*simp add*:*subsetD*[*of T carrier D*], *assumption*+)
apply *simp*
done

lemma *bex-nonempty-subset*:∃ *a*. *a* ∈ *A* ∧ *P a* ⟹
{*x*. *x* ∈ *A* ∧ *P x*} ⊆ *A* ∧ {*x*. *x* ∈ *A* ∧ *P x*} ≠ {}
apply (*erule exE*, *rule conjI*)
apply (*rule subsetI*, *simp*)
apply (*rule-tac A* = {*x* ∈ *A*. *P x*} **in** *nonempty*, *simp*)
done

lemma (**in** *Worder*) *to-subset*:⟦*T* ⊆ *carrier D*; *ord-isom D* (*Iod D T*) *f*⟧ ⟹
∀ *a*. *a* ∈ *carrier D* ⟶ *a* ≼ (*f a*)
apply (*rule contrapos-pp*, *simp*+)
apply (*cut-tac ex-minimum*)
apply (*drule-tac a* = {*a*. *a* ∈ *carrier D* ∧ ¬ *a* ≼ *f a*} **in** *forall-spec*)
apply (*rule conjI*)
apply (*rule subsetI*, *simp*)
apply (*rule ex-nonempty*, *simp*)

apply ((*erule exE*)+, *simp add*:*minimum-elem-def*, (*erule conjE*)+)
apply (*frule Iod-Order*[*of T*],
frule-tac a = *x* **in** *ord-isom-mem*[*of Iod D T f*], *assumption*+)
apply (*frule-tac a* = *x* **and** *b* = *f x* **in** *ord-isom-le*[*of Iod D T f*],
assumption+)
apply (*simp add*:*Iod-carrier subsetD*)

apply (*frule Iod-carrier*[*of T*],
frule-tac a = *f x* **in** *eq-set-inc*[*of - carrier* (*Iod D T*) *T*],
assumption+)
apply (*frule-tac c* = *f x* **in** *subsetD*[*of T carrier D*], *assumption*+)
apply (*frule-tac a* = *f x* **in** *ord-isom-mem*[*of Iod D T f*], *assumption*+)
apply (*frule-tac a* = *f* (*f x*) **in** *eq-set-inc*[*of - carrier* (*Iod D T*) *T*],
assumption+)
apply (*drule-tac a* = *f x* **in** *forall-spec*)

apply (*simp add*:*subsetD Iod-le*)
apply *simp*
done

lemma *to-subsetS*:⟦*Worder D*; *T* ⊆ *carrier D*; *ord-isom D* (*SIod D T*) *f*⟧ ⟹
∀ *a*. *a* ∈ *carrier D* ⟶ *a* ≼$_D$ (*f a*)
apply (*frule Worder.Order*[*of D*],
frule SIod-isom-Iod[*of D T*], *assumption*+,
frule Order.ord-isom-trans[*of D SIod D T Iod D T f* λ*x*∈*T*. *x*])
apply (*simp add*:*SIod-Order*, *simp add*:*Order.Iod-Order*, *assumption*+)

apply (*frule-tac D* = *D* **and** *T* = *T* **and** *f* = *compose* (*carrier D*) (λ*x*∈*T*. *x*)
f
in *Worder.to-subset*, *assumption*+)
apply (*rule allI*, *rule impI*)
apply (*drule-tac a* = *a* **in** *forall-spec*, *simp*)

apply (*frule-tac a* = *a* **in** *Order.ord-isom-mem*[*of D SIod D T f*])
apply (*simp add*:*SIod-Order*, *assumption*+)
apply (*simp add*:*SIod-carrier*)
apply (*simp add*:*compose-def*)
done

lemma (**in** *Worder*) *isom-Worder*:⟦*Order T*; *ord-isom D T f*⟧ ⟹ *Worder T*
apply (*rule Worder.intro*)
apply (*rule Torder.intro*)
apply *assumption*
apply (*rule Torder-axioms.intro*)
apply (*frule-tac b* = *a* **in** *ord-isom-surj*[*of T f*], *assumption*+,
frule-tac b = *b* **in** *ord-isom-surj*[*of T f*], *assumption*+,
(*erule bexE*)+)
apply (*cut-tac Torder-axioms*, *simp add*:*Torder-axioms-def*)
apply (*meson le-cases ord-isom-le*)
apply (*rule Worder-axioms.intro*)
apply (*rule allI*, *rule impI*, *erule conjE*)

apply (*frule ord-isom-func*[*of T f*], *assumption*+)
apply (*frule ord-isom-bij-to*[*of T f*], *assumption*+)
apply (*frule ord-isom-sym*[*of T f*], *assumption*+,
frule Order.ord-isom-func[*of T D*

```
            invfun (carrier D) (carrier T) f])
 apply (rule Order, assumption)
 apply (frule-tac ?A1.0 = X in  image-sub[of
       invfun (carrier D) (carrier T) f carrier T carrier D],
       assumption+,
       frule-tac ?A1.0 = X in image-nonempty[of invfun (carrier D)
       (carrier T) f carrier T carrier D], assumption+)
apply (cut-tac ex-minimum)
apply (drule-tac a = invfun (carrier D) (carrier T) f ' X in forall-spec,

       simp) apply (

      erule exE)
apply (frule-tac S = invfun (carrier D) (carrier T) f ' X and a = x in
      ord-isom-minimum[of T f], assumption+)
 apply (frule-tac X = invfun (carrier D) (carrier T) f ' X and a = x in
       minimum-elem-mem, assumption+)
 apply (simp add:subsetD) apply assumption
 apply (simp add:invfun-set, blast)
done

lemma (in Worder) equiv-Worder:[[Order T; ord-equiv D T]] ==> Worder T
by (simp add:ord-equiv-def,
      erule exE, simp add:isom-Worder)

lemma (in Worder) equiv-Worder1:[[Order T; ord-equiv T D]] ==> Worder T
apply (cut-tac Worder,
      frule Worder.Order[of D],
      frule Order.ord-equiv-sym[of T D], assumption+)
apply (rule equiv-Worder, assumption+)
done

lemma (in Worder) ord-isom-self-id:ord-isom D D f ==> f = idmap (carrier D)
apply (cut-tac Order,
      frule ord-isom-sym [of D f], assumption+,
      frule ord-isom-func[of D f], assumption+)
apply (rule funcset-eq[of f carrier D idmap (carrier D)])
 apply (simp add:ord-isom-def ord-inj-def, simp add:idmap-def)
apply (rule ballI)
 apply (simp add:idmap-def)
 apply (cut-tac subset-self[of carrier D],
       frule to-subset [of carrier D f],
       simp add:Iod-self[THEN sym])

 apply (drule-tac a = x in forall-spec, assumption
     )
 apply (frule to-subset [of carrier D invfun (carrier D) (carrier D) f])
 apply (simp add:Iod-self[THEN sym])
 apply (drule-tac a = x in forall-spec, assumption)
```

apply (*frule-tac* $x = x$ **in** *funcset-mem* [*of f carrier D carrier D*],
assumption+)
apply (*frule-tac* $a = x$ **in** *ord-isom-mem*[*of D*
invfun (*carrier D*) (*carrier D*) *f*], *assumption*+)
apply (*frule-tac* $a = x$ **and** $b =$ *invfun* (*carrier D*) (*carrier D*) *f x* **in**
ord-isom-le[*of D f*], *assumption*+)
apply *simp*

apply (*frule ord-isom-bij-to*[*of D f*], *assumption*+,
simp add:*bij-to-def*, *erule conjE*)
apply (*simp add*:*invfun-r*[*of f carrier D carrier D*])
apply (*rule-tac* $a = f\ x$ **and** $b = x$ **in** *le-antisym*,
assumption+)
done

lemma (**in** *Worder*) *isom-unique*:⟦*Worder E*; *ord-isom D E f*; *ord-isom D E g*⟧
$\Longrightarrow f = g$
apply (*frule Worder.Order*[*of E*])
apply (*insert Order*,
frule ord-isom-sym[*of E g*], *assumption*+,
frule ord-isom-trans [*of E D f*
invfun (*carrier D*) (*carrier E*) *g*], *assumption*+,
frule ord-isom-func[*of D*
compose (*carrier D*) (*invfun* (*carrier D*) (*carrier E*) *g*) *f*], *assumption*+)
apply (*frule ord-isom-self-id* [*of*
compose (*carrier D*) (*invfun* (*carrier D*) (*carrier E*) *g*) *f*])
apply (*thin-tac ord-isom E D* (*invfun* (*carrier D*) (*carrier E*) *g*))
apply (*cut-tac id-ord-isom*, *insert Order*,
frule ord-isom-func[*of D idmap* (*carrier D*)], *assumption*+)

apply (*rule funcset-eq*[*of f carrier D g*])
apply (*simp add*:*ord-isom-def ord-inj-def*)
apply (*simp add*:*ord-isom-def ord-inj-def*)
apply (*rule ballI*)
apply (*frule-tac* $x = x$ **in** *eq-funcs*[*of*
compose (*carrier D*) (*invfun* (*carrier D*) (*carrier E*) *g*) *f*
carrier D carrier D idmap (*carrier D*)], *assumption*+)
apply (*frule-tac* $a = x$ **in** *ord-isom-mem* [*of E f*], *assumption*+,
thin-tac compose (*carrier D*) (*invfun* (*carrier D*) (*carrier E*) *g*) *f* =
idmap (*carrier D*),
simp add:*idmap-def compose-def*)
apply (*simp add*:*ord-isom-def*[*of - E g*] *ord-inj-def*, (*erule conjE*)+)
apply (*frule-tac* $b = f\ x$ **in** *invfun-r*[*of g carrier D carrier E*],
assumption+)
apply *simp*
done

definition
segment :: - ⇒ *'a* ⇒ *'a set* **where**

segment D a = (if a ∉ carrier D then carrier D else
{x. x ≺_D a ∧ x ∈ carrier D})

definition
Ssegment :: 'a Order ⇒ 'a ⇒ 'a set **where**
Ssegment D a = (if a ∉ carrier D then carrier D else
{x. x ≺_D a ∧ x ∈ carrier D})

lemma (**in** *Order*) *segment-sub*:*segment D a ⊆ carrier D*
apply (*rule subsetI*, *simp add*:*segment-def*)
apply (*case-tac a ∉ carrier D*, *simp*)
apply (*simp add*:*segment-def*)
done

lemma *Ssegment-sub*:*Ssegment D a ⊆ carrier D*
by (*rule subsetI*, *simp add*:*Ssegment-def*,
case-tac a ∉ carrier D, *simp*, *simp add*:*Ssegment-def*)

lemma (**in** *Order*) *segment-free*:*a ∉ carrier D ⟹*
segment D a = carrier D
by (*simp add*:*segment-def*)

lemma *Ssegment-free*:*a ∉ carrier D ⟹*
Ssegment D a = carrier D
by (*simp add*:*Ssegment-def*)

lemma (**in** *Order*) *segment-sub-sub*:*⟦S ⊆ carrier D*; *d ∈ S⟧ ⟹*
segment (Iod D S) d ⊆ segment D d
apply (*rule subsetI*)
apply (*frule-tac c = d* **in** *subsetD*[*of S carrier D*], *assumption*+)
apply (*simp add*:*segment-def*)
apply (*simp add*:*Iod-carrier*)
apply (*erule conjE*, *simp add*:*Iod-less*[*of S*])
apply (*simp add*:*subsetD*)
done

lemma *Ssegment-sub-sub*:*⟦Order D*; *S ⊆ carrier D*; *d ∈ S⟧ ⟹*
Ssegment (SIod D S) d ⊆ Ssegment D d
apply (*rule subsetI*)
apply (*frule-tac c = d* **in** *subsetD*[*of S carrier D*], *assumption*+)
apply (*simp add*:*Ssegment-def*)
apply (*simp add*:*SIod-carrier*, *erule conjE*, *simp add*:*SIod-less*[*of S*])
apply (*simp add*:*subsetD*)
done

lemma (**in** *Order*) *a-notin-segment*:*a ∉ segment D a*
by (*simp add*:*segment-def oless-def*)

lemma *a-notin-Ssegment*:*a ∉ Ssegment D a*

by (*simp add*:*Ssegment-def oless-def*)

lemma (**in** *Order*) *Iod-carr-segment*:
  carrier (*Iod D* (*segment D a*)) = *segment D a*
by (*cut-tac segment-sub*[*of a*], *simp add*:*Iod-carrier*)

lemma *SIod-carr-Ssegment*:*Order D* ⟹
  carrier (*SIod D* (*Ssegment D a*)) = *Ssegment D a*
apply (*cut-tac Ssegment-sub*[*of D a*])
apply (*simp add*:*SIod-carrier*)
done

lemma (**in** *Order*) *segment-inc*:⟦*a* ∈ *carrier D*; *b* ∈ *carrier D*⟧ ⟹
    (*a* ≺ *b*) = (*a* ∈ *segment D b*)
by (*simp add*:*segment-def*)

lemma *Ssegment-inc*:⟦*Order D*; *a* ∈ *carrier D*; *b* ∈ *carrier D*⟧ ⟹
    (*a* ≺$_D$ *b*) = (*a* ∈ *Ssegment D b*)
by (*simp add*:*Ssegment-def*)

lemma (**in** *Order*) *segment-inc1*:*b* ∈ *carrier D* ⟹
    (*a* ≺ *b* ∧ *a* ∈ *carrier D*) = (*a* ∈ *segment D b*)
by (*simp add*:*segment-def*)

lemma *Ssegment-inc1*:⟦*Order D*; *b* ∈ *carrier D*⟧ ⟹
    (*a* ≺$_D$ *b* ∧ *a* ∈ *carrier D*) = (*a* ∈ *Ssegment D b*)
by (*simp add*:*Ssegment-def*)

lemma (**in** *Order*) *segment-inc-if*:⟦*b* ∈ *carrier D*;*a* ∈ *segment D b*⟧ ⟹
    a ≺ *b*
by (*simp add*:*segment-def*)

lemma *Ssegment-inc-if*:⟦*Order D*; *b* ∈ *carrier D*; *a* ∈ *Ssegment D b*⟧ ⟹
    a ≺$_D$ *b*
by (*simp add*:*Ssegment-def*)

lemma (**in** *Order*) *segment-inc-less*:⟦*W* ⊆ *carrier D*; *a* ∈ *carrier D*;
  y ∈ *W*; *x* ∈ *segment* (*Iod D W*) *a*; *y* ≺ *x*⟧ ⟹ *y* ∈ *segment* (*Iod D W*) *a*
apply (*frule Iod-Order*[*of W*],
  frule Order.segment-sub[*of Iod D W a*],
  frule subsetD[*of segment* (*Iod D W*) *a carrier* (*Iod D W*) *x*],
    assumption+, *simp add*:*Iod-carrier*)
apply (*case-tac a* ∈ *carrier* (*Iod D W*))
apply (*subst Order.segment-inc*[*THEN sym*, *of Iod D W y a*], *assumption*,
  simp add:*Iod-carrier*, *simp add*:*Iod-carrier*)
apply (*simp add*:*Iod-carrier*, *simp add*:*Iod-less*)
apply (*rule less-trans*[*of y x a*], (*simp add*:*subsetD*)+)
apply (*frule Order.segment-inc*[*THEN sym*, *of Iod D W x a*],
  (*simp add*:*Iod-carrier*)+,

frule-tac Order.segment-sub[*of Iod D W x*],
frule subsetD[*of segment* (*Iod D W*) *a W x*], *assumption*+,
simp add:*Iod-carrier*,
frule-tac subsetD[*of segment* (*Iod D W*) *a W x*], *assumption*+,
simp add:*Iod-less*)
apply (*simp add*:*Order.segment-free*[*of Iod D W a*], *simp add*:*Iod-carrier*)
done

lemma (**in** *Order*) *segment-order-less*:$\forall\, b{\in}carrier\ D.\ \forall\, x{\in}\ segment\ D\ b.\ \forall\, y{\in}\ segment\ D\ b.\ (x \prec y) = (x \prec_{(Iod\ D\ (segment\ D\ b))}\ y)$
by ((*rule ballI*)+,
cut-tac a = *b* **in** *segment-sub*, *simp add*:*Iod-less*)

lemma *Ssegment-order-less*:*Order D* $\Longrightarrow$
$\forall\, b{\in}carrier\ D.\ \forall\, x{\in}\ Ssegment\ D\ b.\ \forall\, y{\in}\ Ssegment\ D\ b.$
$(x \prec_D y) = (x \prec_{(SIod\ D\ (Ssegment\ D\ b))}\ y)$
by ((*rule ballI*)+,
cut-tac a = *b* **in** *Ssegment-sub*[*of D*], *simp add*:*SIod-less*)

lemma (**in** *Order*) *segment-order-le*:$\forall\, b{\in}carrier\ D.\ \forall\, x{\in}\ segment\ D\ b.$
$\forall\, y{\in}\ segment\ D\ b.\ (x \preceq y) = (x \preceq_{(Iod\ D\ (segment\ D\ b))}\ y)$
by ((*rule ballI*)+,
cut-tac a = *b* **in** *segment-sub*, *simp add*:*Iod-le*)

lemma *Ssegment-order-le*:$\forall\, b{\in}carrier\ D.\ \forall\, x{\in}\ Ssegment\ D\ b.$
$\forall\, y{\in}\ Ssegment\ D\ b.\ (x \preceq_D y) = (x \preceq_{(SIod\ D\ (Ssegment\ D\ b))}\ y)$
by ((*rule ballI*)+,
cut-tac a = *b* **in** *Ssegment-sub*[*of D*], *simp add*:*SIod-le*)

lemma (**in** *Torder*) *Iod-Torder*:$X \subseteq carrier\ D \Longrightarrow Torder\ (Iod\ D\ X)$
apply (*rule Torder.intro*)
apply (*simp add*:*Iod-Order*)
apply (*rule Torder-axioms.intro*)
apply (*simp add*:*Iod-carrier Iod-le*)
apply (*meson contra-subsetD le-cases*)
done

lemma *SIod-Torder*:$[\![Torder\ D;\ X \subseteq carrier\ D]\!] \Longrightarrow Torder\ (SIod\ D\ X)$
apply (*simp add*:*Torder-def*, *simp add*:*SIod-Order*, *simp add*:*Torder-axioms-def*)
apply ((*rule allI*, *rule impI*)+,
simp add:*SIod-carrier SIod-le*) **apply** (*erule conjE*)
apply (*frule-tac c* = *a* **in** *subsetD*[*of X carrier D*], *assumption*+,
frule-tac c = *b* **in** *subsetD*[*of X carrier D*], *assumption*+)
apply *blast*
done

lemma (**in** *Order*) *segment-not-inc*:$[\![a \in carrier\ D;\ b \in carrier\ D;$
$a \prec b]\!] \Longrightarrow b \notin segment\ D\ a$

apply (*rule contrapos-pp*, *simp*+, *simp add*:*segment-def*)
apply (*simp add*:*oless-def*, (*erule conjE*)+)
apply (*frule le-antisym*[*of a b*], *assumption*+, *simp*)
done

lemma *Ssegment-not-inc*:⟦*Order D*; *a* ∈ *carrier D*; *b* ∈ *carrier D*; *a* ≺$_D$ *b*⟧ ⟹
b ∉ *Ssegment D a*
apply (*rule contrapos-pp*, *simp*+, *simp add*:*Ssegment-def*)
apply (*simp add*:*oless-def*, (*erule conjE*)+)
apply (*frule Order.le-antisym*[*of D a b*], *assumption*+, *simp*)
done

lemma (**in** *Torder*) *segment-not-inc-iff*:⟦*a* ∈ *carrier D*; *b* ∈ *carrier D*⟧ ⟹
(*a* ≼ *b*) = (*b* ∉ *segment D a*)
apply (*rule iffI*)
apply (*simp add*:*le-imp-less-or-eq*,
erule disjE, *simp add*:*segment-not-inc*, *simp add*:*a-notin-segment*)
apply (*simp add*:*segment-def*, *simp add*:*not-less-le*[*THEN sym*, *of b a*])
done

lemma *Ssegment-not-inc-iff*:⟦*Torder D*; *a* ∈ *carrier D*; *b* ∈ *carrier D*⟧ ⟹
(*a* ≼$_D$ *b*) = (*b* ∉ *Ssegment D a*)
apply (*rule iffI*)
apply (*frule Torder.Order*[*of D*])
apply (*simp add*:*Order.le-imp-less-or-eq*,
erule disjE, *rule Ssegment-not-inc*, *assumption*+)

apply (*simp add*: *a-notin-Ssegment*)
apply (*simp add*:*Ssegment-def*)
apply (*simp add*:*Torder.not-less-le*[*THEN sym*, *of D b a*])
done

lemma (**in** *Torder*) *minimum-segment-of-sub*:⟦*X* ⊆ *carrier D*;
minimum-elem D (*segment* (*Iod D X*) *d*) *m* ⟧ ⟹ *minimum-elem D X m*
apply (*case-tac d* ∉ *carrier* (*Iod D X*))
apply (*simp add*:*segment-def*)
apply (*simp add*:*Iod-carrier*)

apply (*simp add*:*Iod-carrier*)
apply (*subst minimum-elem-def*)
apply (*frule Iod-Order*[*of X*],
frule Order.segment-sub[*of Iod D X d*],
simp add:*Iod-carrier*,
frule subset-trans[*of segment* (*Iod D X*) *d X carrier D*],
assumption+,
frule minimum-elem-mem[*of segment* (*Iod D X*) *d m*], *assumption*)
apply (*simp add*:*subsetD*[*of segment* (*Iod D X*) *d X m*])
apply (*rule ballI*)
apply (*simp add*:*minimum-elem-def*)

apply (*case-tac* $x \in$ *segment* (*Iod D X*) *d*)
apply (*frule-tac a1* $= x$ **in** *Order.segment-inc*[*THEN sym*, *of Iod D X - d*])
apply (*simp add*:*Iod-carrier subsetD*)
apply (*simp add*:*Iod-carrier*)
apply (*simp add*:*Iod-less*)
apply (*frule Iod-Torder*[*of X*])
apply (*frule-tac b1* $= x$ **in** *Torder.segment-not-inc-iff*[*THEN sym*,
of Iod D X d])
apply (*simp add*:*Iod-carrier*)
apply (*simp add*:*Iod-carrier*)
apply *simp*
apply (*frule Order.segment-inc*[*THEN sym*, *of Iod D X m d*],
thin-tac $x \notin$ *segment* (*Iod D X*) *d*,
frule Order.segment-sub[*of Iod D X d*])
apply (*simp add*:*Iod-carrier subsetD*)
apply (*simp add*:*Iod-carrier*)
apply *simp*
apply (*frule subsetD*[*of segment* (*Iod D X*) *d X m*], *assumption*)
apply (*simp add*:*Iod-le Iod-less*)
apply (*frule subsetD*[*of X carrier D m*], *assumption*+,
frule subsetD[*of X carrier D d*], *assumption*+,
frule-tac c $= x$ **in** *subsetD*[*of X carrier D*], *assumption*+)
apply (*frule-tac c* $= x$ **in** *less-le-trans*[*of m d*], *assumption*+)
apply (*simp add*:*less-imp-le*)
done

lemma (**in** *Torder*) *segment-out*:⟦$a \in$ *carrier D*; $b \in$ *carrier D*;
$a \prec b$⟧ $\Longrightarrow$ *segment* (*Iod D* (*segment D a*)) b = *segment D a*
apply (*subst segment-def*[*of Iod D* (*segment D a*)])
apply (*frule segment-not-inc*[*of a b*], *assumption*+)
apply (*cut-tac segment-sub*[*of a*])
apply (*simp add*:*Iod-carrier*)
done

lemma (**in** *Torder*) *segment-minimum-minimum*:⟦$X \subseteq$ *carrier D*; $d \in X$;
minimum-elem (*Iod D* (*segment D d*)) ($X \cap$ (*segment D d*)) m⟧ $\Longrightarrow$
minimum-elem D X m
apply (*cut-tac segment-sub*[*of d*])
apply (*subst minimum-elem-def*)
apply (*cut-tac Order.minimum-elem-mem*[*of Iod D* (*segment D d*)
$X \cap$ (*segment D d*) m])
apply (*cut-tac Int-lower1*[*of X segment D d*],
frule-tac subsetD[*of X* $\cap$ *segment D d X m*], *assumption*+, *simp*)
apply (*rule ballI*)
apply (*case-tac* $x \in$ *segment D d*)
apply (*simp add*:*minimum-elem-def*)
apply (*drule-tac* $x = x$ **in** *bspec*,

simp) **apply** (

simp add:*Iod-le*)
apply (*frule subsetD*[*of X carrier D d*], *assumption*+,
frule subsetD[*of X carrier D m*], *assumption*+,
frule-tac c = *x* **in** *subsetD*[*of X carrier D*], *assumption*+)
apply (*simp add*:*segment-inc*[*THEN sym*, *of* - *d*],
simp add:*not-less-le*)
apply (*frule-tac c* = *x* **in** *less-le-trans*[*of m d*], *assumption*+)
apply (*simp add*:*less-imp-le*)

apply (*simp add*:*Iod-Order*)
apply (*simp add*:*Iod-carrier*)
apply (*simp add*:*Int-lower2*)
done

lemma (**in** *Torder*) *segment-mono*:⟦*a* ∈ *carrier D*; *b* ∈ *carrier D*⟧ ⟹
(*a* ≺ *b*) = (*segment D a* ⊂ *segment D b*)
apply (*rule iffI*)
apply (*rule psubsetI*, *rule subsetI*)
apply (*simp add*:*segment-def*, *erule conjE*)
apply (*rule-tac a* = *x* **and** *b* = *a* **and** *c* = *b* **in** *less-trans*,
assumption+)
apply (*cut-tac a-notin-segment*[*of a*],
simp add:*segment-inc*[*of a b*], *blast*)
apply (*simp add*:*psubset-eq*, *erule conjE*,
frule not-sym[*of segment D a segment D b*],
thin-tac segment D a ≠ *segment D b*,
frule sets-not-eq[*of segment D b segment D a*], *assumption*+)
apply (*erule bexE*)
apply (*thin-tac segment D a* ⊆ *segment D b*,
thin-tac segment D b ≠ *segment D a*)
apply (*simp add*:*segment-def*, (*erule conjE*)+)
apply (*frule-tac a* = *aa* **and** *b* = *a* **in** *not-less-le*, *assumption*+,
simp, *simp add*:*oless-def*, (*erule conjE*)+)
apply (*frule-tac a* = *a* **and** *b* = *aa* **and** *c* = *b* **in** *le-trans*,
assumption+, *simp*)
apply (*rule contrapos-pp*, *simp*+)
done

lemma *Ssegment-mono*:⟦*Torder D*; *a* ∈ *carrier D*; *b* ∈ *carrier D*⟧ ⟹
(*a* ≺$_D$ *b*) = (*Ssegment D a* ⊂ *Ssegment D b*)
apply (*frule Torder.Order*)
apply (*rule iffI*)
apply (*rule psubsetI*, *rule subsetI*)
apply (*simp add*:*Ssegment-def*, *erule conjE*)
apply (*rule-tac a* = *x* **and** *b* = *a* **and** *c* = *b* **in** *Order.less-trans*,
assumption+)
apply (*cut-tac a-notin-Ssegment*[*of a*],
simp add:*Ssegment-inc*[*of D a b*], *blast*)
apply (*simp add*:*psubset-eq*, *erule conjE*,

frule not-sym[*of Ssegment D a Ssegment D b*],
thin-tac Ssegment D a $\neq$ *Ssegment D b*,
frule sets-not-eq[*of Ssegment D b Ssegment D a*], *assumption*+)
apply (*erule bexE*)
apply (*thin-tac Ssegment D a* $\subseteq$ *Ssegment D b*,
thin-tac Ssegment D b $\neq$ *Ssegment D a*)
apply (*simp add*:*Ssegment-def*, (*erule conjE*)+)
apply (*frule-tac* $a = aa$ **and** $b = a$ **in** *Torder.not-less-le*, *assumption*+,
simp, *simp add*:*oless-def*, (*erule conjE*)+)
apply (*frule-tac* $a = a$ **and** $b = aa$ **and** $c = b$ **in** *Order.le-trans*,
assumption+, *simp*)
apply (*rule contrapos-pp*, *simp*+)
done

lemma (**in** *Torder*) *segment-le-mono*:$[\![a \in carrier\ D;\ b \in carrier\ D]\!] \Longrightarrow$
$(a \preceq b) = (segment\ D\ a \subseteq segment\ D\ b)$
apply (*simp add*:*le-imp-less-or-eq*[*of a b*])

apply (*rule iffI*)
apply (*erule disjE*)
apply (*simp add*:*segment-mono*[*of a b*], *simp*)
apply (*frule segment-mono*[*THEN sym*, *of a b*], *assumption*+)
apply (*simp add*:*psubset-eq*)
apply (*case-tac segment D a* $\neq$ *segment D b*, *simp*)
apply *simp*
apply (*rule contrapos-pp*, *simp*+,
frule less-linear[*of a b*], *assumption*+, *simp*,
simp add:*segment-mono*[*of b a*])
done

lemma *Ssegment-le-mono*:$[\![Torder\ D;\ a \in carrier\ D;\ b \in carrier\ D]\!] \Longrightarrow$
$(a \preceq_D b) = (Ssegment\ D\ a \subseteq Ssegment\ D\ b)$
apply (*cut-tac Torder.Order*[*of D*])
apply (*simp add*:*Order.le-imp-less-or-eq*[*of D a b*])

apply (*rule iffI*)
apply (*erule disjE*)
apply (*simp add*: *Ssegment-mono*[*of D a b*])

apply (*frule Ssegment-mono*[*THEN sym*, *of D a b*], *assumption*+)
apply (*simp add*:*psubset-eq*)
apply (*case-tac Ssegment D a* $\neq$ *Ssegment D b*)
apply (*cut-tac Ssegment-mono*[*THEN sym*, *of D a b*])
apply (*simp add*:*psubset-eq*, *assumption*+)
apply *simp*
apply (*cut-tac a-notin-Ssegment*[*of a D*], *simp*)
apply (*simp add*:*Ssegment-not-inc-iff*[*THEN sym*, *of D b a*])
apply (*frule sym*, *thin-tac Ssegment D a* $=$ *Ssegment D b*)
apply (*cut-tac a-notin-Ssegment*[*of b D*], *simp*)

apply (*simp add*:*Ssegment-not-inc-iff*[*THEN sym*, *of D a b*])
apply (*frule Order.le-antisym*[*of D a b*], *assumption*+, *simp*+)
done

lemma (**in** *Torder*) *segment-inj*:⟦*a* ∈ *carrier D*; *b* ∈ *carrier D*⟧ ⟹
(*a* = *b*) = (*segment D a* = *segment D b*)
apply (*rule iffI*)
apply *simp*
apply (*rule equalityE*[*of segment D a segment D b*], *assumption*)
apply (*thin-tac segment D a* = *segment D b*)
apply (*simp add*:*segment-le-mono*[*THEN sym*, *of a b*])
apply (*simp add*:*segment-le-mono*[*THEN sym*, *of b a*])

apply (*simp add*:*le-antisym*)
done

lemma *Ssegment-inj*:⟦*Torder D*; *a* ∈ *carrier D*; *b* ∈ *carrier D*⟧ ⟹
(*a* = *b*) = (*Ssegment D a* = *Ssegment D b*)
apply (*rule iffI*)
apply *simp*
apply (*rule equalityE*[*of Ssegment D a Ssegment D b*], *assumption*)

apply (*thin-tac Ssegment D a* = *Ssegment D b*)
apply (*simp add*:*Ssegment-le-mono*[*THEN sym*, *of D a b*])
apply (*simp add*:*Ssegment-le-mono*[*THEN sym*, *of D b a*])
apply (*cut-tac Torder.Order*[*of D*])
apply (*simp add*:*Order.le-antisym*, *assumption*)
done

lemma (**in** *Torder*) *segment-inj-neq*:⟦*a* ∈ *carrier D*; *b* ∈ *carrier D*⟧ ⟹
(*a* ≠ *b*) = (*segment D a* ≠ *segment D b*)
by (*simp add*:*segment-inj*)

lemma *Ssegment-inj-neq*:⟦*Torder D*; *a* ∈ *carrier D*; *b* ∈ *carrier D*⟧ ⟹
(*a* ≠ *b*) = (*Ssegment D a* ≠ *Ssegment D b*)
by (*simp add*:*Ssegment-inj*)

lemma (**in** *Order*) *segment-inc-psub*:⟦*x* ∈ *segment D a*⟧ ⟹
segment D x ⊂ *segment D a*
apply (*simp add*:*psubset-eq*)
apply (*rule conjI*, *rule subsetI*)
apply (*simp add*:*segment-def*)
apply (*case-tac a* ∉ *carrier D*, *simp*)
apply (*simp*, (*erule conjE*)+)
apply (*rule-tac a* = *xa* **and** *b* = *x* **and** *c* = *a* **in** *less-trans*, *assumption*+)
apply (*cut-tac a-notin-segment*[*of x*]) **apply** *blast*
done

lemma *Ssegment-inc-psub*:⟦*Order D*; *x* ∈ *Ssegment D a*⟧ ⟹

Ssegment D x ⊂ Ssegment D a

apply (*simp add*:*psubset-eq*)
apply (*rule conjI*, *rule subsetI*)
apply (*simp add*:*Ssegment-def*)
apply (*case-tac a ∉ carrier D*, *simp*)
apply (*simp*, (*erule conjE*)+)

apply (*rule-tac a = xa* **and** *b = x* **and** *c = a* **in** *Order.less-trans*[*of D*],
assumption+)

apply (*cut-tac a-notin-Ssegment*[*of x*]) **apply** *blast*
done

lemma (**in** *Order*) *segment-segment*:⟦*b ∈ carrier D*; *a ∈ segment D b*⟧ ⟹
segment (*Iod D* (*segment D b*)) *a = segment D a*
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*simp add*:*segment-def*[*of Iod D* (*segment D b*) *a*])
apply (*cut-tac segment-sub*[*of b*], *simp add*:*Iod-carrier*)
apply (*erule conjE*)
apply (*simp add*:*Iod-less*)
apply (*frule-tac c = x* **in** *subsetD*[*of segment D b carrier D*], *assumption*+,
frule-tac c = a **in** *subsetD*[*of segment D b carrier D*], *assumption*+)
apply (*simp add*:*segment-inc*[*of* - *a*])

apply (*rule subsetI*)
apply (*simp add*:*segment-def*[*of Iod D* (*segment D b*) *a*])
apply (*cut-tac segment-sub*[*of b*], *simp add*:*Iod-carrier*)
apply (*frule segment-inc-psub*[*of a b*],
frule psubset-imp-subset[*of segment D a segment D b*],
thin-tac segment D a ⊂ segment D b,
frule-tac c = x **in** *subsetD*[*of segment D a segment D b*],
assumption+)
apply (*simp add*:*Iod-less*) **apply** (*simp add*:*segment-def*)
done

lemma *Ssegment-Ssegment*:⟦*Order D*; *b ∈ carrier D*; *a ∈ Ssegment D b*⟧ ⟹
Ssegment (*SIod D* (*Ssegment D b*)) *a = Ssegment D a*
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*simp add*:*Ssegment-def*[*of SIod D* (*Ssegment D b*) *a*])
apply (*cut-tac Ssegment-sub*[*of D b*], *simp add*:*SIod-carrier*)
apply (*erule conjE*)
apply (*simp add*:*SIod-less*)
apply (*frule-tac c = x* **in** *subsetD*[*of Ssegment D b carrier D*], *assumption*+,
frule-tac c = a **in** *subsetD*[*of Ssegment D b carrier D*], *assumption*+)
apply (*simp add*:*Ssegment-inc*[*of D*- *a*])

apply (*rule subsetI*)

apply (*simp add*:*Ssegment-def*[*of SIod D* (*Ssegment D b*) *a*])
 apply (*cut-tac Ssegment-sub*[*of D b*], *simp add*:*SIod-carrier*)
 apply (*frule Ssegment-inc-psub*[*of D a b*], *assumption*,
 frule psubset-imp-subset[*of Ssegment D a Ssegment D b*],
 thin-tac Ssegment D a ⊂ *Ssegment D b*,
 frule-tac c = *x* **in** *subsetD*[*of Ssegment D a Ssegment D b*],
 assumption+)
 apply (*simp add*:*SIod-less*) **apply** (*simp add*:*Ssegment-def*)
done

lemma (**in** *Order*) *Iod-segment-segment*:*a* ∈ *carrier* (*Iod D* (*segment D b*)) ⟹
 Iod (*Iod D* (*segment D b*)) (*segment* (*Iod D* (*segment D b*)) *a*) =
 Iod D (*segment D a*)
apply (*case-tac b* ∈ *carrier D*)
apply (*cut-tac segment-sub*[*of b*])
 apply (*simp add*:*Iod-carrier*)
 apply (*frule segment-inc-psub*[*of a b*],
 frule psubset-imp-subset[*of segment D a segment D b*],
 thin-tac segment D a ⊂ *segment D b*)
 apply (*simp add*:*segment-segment*[*of b a*])
 apply (*simp add*:*Iod-sub-sub*[*of segment D a segment D b*])
apply (*simp add*:*segment-def*[*of D b*])
 apply (*simp add*:*Iod-self*[*THEN sym*])
done

lemma *SIod-Ssegment-Ssegment*:⟦*Order D*; *a* ∈ *carrier* (*SIod D* (*Ssegment D b*))⟧

 ⟹
 SIod (*SIod D* (*Ssegment D b*)) (*Ssegment* (*SIod D* (*Ssegment D b*)) *a*) =
 SIod D (*Ssegment D a*)
apply (*case-tac b* ∈ *carrier D*)
apply (*cut-tac Ssegment-sub*[*of D b*])
 apply (*simp add*:*SIod-carrier*[*of D*])
 apply (*frule Ssegment-inc-psub*[*of D a b*], *simp add*:*subsetD*) **apply** (
 frule psubset-imp-subset[*of Ssegment D a Ssegment D b*],
 thin-tac Ssegment D a ⊂ *Ssegment D b*)
 apply (*simp add*:*Ssegment-Ssegment*[*of D b a*])
 apply (*simp add*:*SIod-sub-sub*[*of Ssegment D a Ssegment D b*])
apply (*simp add*:*Ssegment-def*[*of D b*], *simp add*:*SIod-self*[*THEN sym*])
done

lemma (**in** *Order*) *ord-isom-segment-mem*:⟦*Order E*;
 ord-isom D E f; *a* ∈ *carrier D*; *x* ∈ *segment D a* ⟧ ⟹
 (*f x*) ∈ *segment E* (*f a*)

apply (*frule segment-inc-if*[*of a x*], *assumption*+)
apply (*frule ord-isom-less*[*of E f x a*], *assumption*+)
 apply (*simp add*:*segment-def*, *assumption*, *simp*)

apply (*frule ord-isom-mem*[*of E f x*], *assumption*+,
simp add:*segment-def*,
frule ord-isom-mem[*of E f a*], *assumption*+)
apply (*simp add*:*Order.segment-inc*[*of E f x f a*])
done

lemma *ord-isom-Ssegment-mem*:⟦*Order D*; *Order E*;
ord-isom D E f; *a* ∈ *carrier D*; *x* ∈ *Ssegment D a*⟧ ⟹
(*f x*) ∈ *Ssegment E* (*f a*)
apply (*frule Ssegment-inc-if*[*of D a x*], *assumption*+)
apply (*frule Order.ord-isom-less*[*of D E f x a*], *assumption*+)
apply (*simp add*:*Ssegment-def*, *assumption*, *simp*)

apply (*frule Order.ord-isom-mem*[*of D E f x*], *assumption*+,
simp add:*Ssegment-def*,
frule Order.ord-isom-mem[*of D E f a*], *assumption*+)
apply (*simp add*:*Ssegment-def*)
done

lemma (**in** *Order*) *ord-isom-segment-segment*:⟦*Order E*;
ord-isom D E f; *a* ∈ *carrier D* ⟧ ⟹
ord-isom (*Iod D* (*segment D a*)) (*Iod E* (*segment E* (*f a*)))
(λ*x*∈*carrier* (*Iod D* (*segment D a*)). *f x*)
apply (*frule ord-isom-inj-on*[*of E f*], *assumption*+)
apply (*cut-tac segment-sub*[*of a*])
apply (*frule restrict-inj*[*of f carrier D segment D a*], *assumption*)
apply (*frule ord-isom-surj-to*[*of E f*], *assumption*+)

apply (*subst ord-isom-def*, *subst ord-inj-def*)
apply (*simp add*:*Iod-carr-segment Order.Iod-carr-segment*)

apply (*subgoal-tac restrict f* (*segment D a*) ∈
segment D a → *segment E* (*f a*), *simp*)
defer
apply (*simp add*:*ord-isom-segment-mem*)

apply (*rule conjI*)
defer
apply (*rule surj-to-test*, *assumption*+)
apply (*rule ballI*, *simp*)
apply (*frule ord-isom-func*[*of E f*], *assumption*+)
apply (*frule surj-to-el*[*of f carrier D carrier E*], *assumption*+,

frule ord-isom-mem[*of E f a*], *assumption*+,
frule Order.segment-sub[*of E f a*],
frule-tac c = *b* **in** *subsetD*[*of segment E* (*f a*) *carrier E*],
assumption+,

drule-tac x = *b* **in** *bspec*, *assumption*,

```
        erule bexE)
 apply (simp add:Order.segment-inc[THEN sym, of E - f a],
        rotate-tac −1, frule sym, thin-tac f aa = b, simp,
        frule-tac a1 = aa and b1 = a in ord-isom-less[THEN sym, of E f],
        assumption+, simp,
        simp add:segment-inc[of - a], blast)

 apply (rule ballI)+
 apply (frule ord-isom-mem[of E f a], assumption+,
        frule Order.segment-sub[of E f a])
 apply (frule-tac x = aa in ord-isom-segment-mem[of E f a], assumption+,
        frule-tac x = b in ord-isom-segment-mem[of E f a], assumption+,

        simp add:Iod-less Order.Iod-less,
        subst ord-isom-less[of E f], assumption+, (simp add:subsetD)+)
done

lemma ord-isom-Ssegment-Ssegment:⟦Order D; Order E;
      ord-isom D E f; a ∈ carrier D ⟧ ⟹
      ord-isom (SIod D (Ssegment D a)) (SIod E (Ssegment E (f a)))
                                 (λx∈carrier (SIod D (Ssegment D a)). f x)
apply (frule-tac a = a in Order.ord-isom-mem[of D E f], assumption+)
apply (cut-tac Ssegment-sub[of D a],
       cut-tac Ssegment-sub[of E f a])

apply (subst ord-isom-def, simp add:ord-inj-def)
apply (rule conjI)
 apply (rule Pi-I)
 apply (simp add:SIod-carrier)
 apply (frule-tac c = x in subsetD[of Ssegment D a carrier D], assumption+)
  apply (frule-tac a = x in Order.ord-isom-mem[of D E f], assumption+)
 apply (subst Ssegment-inc[THEN sym, of E - f a], assumption+)
 apply (subst Order.ord-isom-less[THEN sym, of D E f - a], assumption+)
 apply (subst Ssegment-inc[of D - a], assumption+)
 apply (rule conjI)
  apply (simp add:SIod-carrier)
  apply (simp add:ord-isom-def bij-to-def, (erule conjE)+)
  apply (simp add:ord-inj-def, (erule conjE)+)
  apply (rule restrict-inj[of f carrier D Ssegment D a], assumption+)
apply (rule conjI)
 apply (rule ballI)+
 apply (simp add:SIod-carrier)
 apply (frule-tac c = aa in subsetD[of Ssegment D a carrier D],
        assumption+,
        frule-tac c = b in subsetD[of Ssegment D a carrier D], assumption+)
 apply (frule-tac a1 = aa and b1 = a in Ssegment-inc[THEN sym], assumption+,
        frule-tac a1 = b and b1 = a in Ssegment-inc[THEN sym], assumption+,
        simp)
 apply (simp add:Order.ord-isom-less[of D E f])
```

apply (*frule-tac a = a* **in** *Order.ord-isom-mem*[*of D E f*], *assumption*+,
frule-tac a = aa **in** *Order.ord-isom-mem*[*of D E f*], *assumption*+,
frule-tac a = b **in** *Order.ord-isom-mem*[*of D E f*], *assumption*+)
apply (*simp add*:*Ssegment-inc*[*of E*])
apply (*simp add*:*SIod-less Order.ord-isom-less*)
apply (*simp add*:*surj-to-def*,
simp add:*SIod-carrier*)
apply (*rule equalityI*)
apply (*rule subsetI*, *simp add*:*image-def*, *erule bexE*)
apply (*frule-tac c = xa* **in** *subsetD*[*of Ssegment D a carrier D*],
assumption+)
apply (*frule-tac a = xa* **in** *Ssegment-inc*[*of D - a*], *assumption*+, *simp*)
apply (*simp add*:*Order.ord-isom-less*[*of D E f - a*])
apply (*frule-tac a = xa* **in** *Order.ord-isom-mem*[*of D E f*], *assumption*+)
apply (*subst Ssegment-inc*[*THEN sym*], *assumption*+)

apply (*rule subsetI*)
apply (*frule-tac c = x* **in** *subsetD*[*of Ssegment E* (*f a*) *carrier E*],
assumption+)
apply (*simp add*:*Ssegment-inc*[*THEN sym*])
apply (*frule-tac b = x* **in** *Order.ord-isom-surj*[*of D E f*], *assumption*+,
erule bexE, *simp*, *thin-tac x = f aa*)
apply (*simp add*:*Order.ord-isom-less*[*THEN sym*])
apply (*simp add*:*Ssegment-inc*[*of D*])
done

lemma (**in** *Order*) *ord-equiv-segment-segment*:
⟦*Order E*; *ord-equiv D E*; *a* ∈ *carrier D*⟧
⟹ ∃ *t*∈*carrier E*. *ord-equiv* (*Iod D* (*segment D a*)) (*Iod E* (*segment E t*))

apply (*simp add*:*ord-equiv-def*, *erule exE*)
apply (*frule-tac f = f* **in** *ord-isom-segment-segment*[*of E - a*], *assumption*+)
apply (*frule-tac f = f* **in** *ord-isom-mem*[*of E - a*], *assumption*+)
apply *blast*
done

lemma *ord-equiv-Ssegment-Ssegment*:
⟦*Order D*; *Order E*; *ord-equiv D E*; *a* ∈ *carrier D*⟧
⟹ ∃ *t*∈*carrier E*. *ord-equiv* (*SIod D* (*Ssegment D a*)) (*SIod E* (*Ssegment E t*))
apply (*simp add*:*ord-equiv-def*, *erule exE*)
apply (*frule-tac f = f* **in** *ord-isom-Ssegment-Ssegment*[*of D E - a*],
assumption+)
apply (*frule-tac f = f* **in** *Order.ord-isom-mem*[*of D E - a*], *assumption*+)
apply *blast*
done

lemma (**in** *Order*) *ord-isom-restricted*:
⟦*Order E*; *ord-isom D E f*; *D1* ⊆ *carrier D*⟧ ⟹
ord-isom (*Iod D D1*) (*Iod E* (*f* ‘ *D1*)) (λ*x*∈*D1*. *f x*)

apply (*simp add*:*ord-isom-def*[*of D E f*], *erule conjE*)
apply (*simp add*:*ord-inj-restrict-isom*[*of E f D1*])
done

lemma *ord-isom-restrictedS*:
⟦*Order D*; *Order E*; *ord-isom D E f*; *D1* ⊆ *carrier D*⟧ ⟹
ord-isom (*SIod D D1*) (*SIod E* (*f* ‘ *D1*)) (λ*x*∈*D1*. *f x*)
apply (*simp add*:*ord-isom-def*[*of D E f*], *erule conjE*)
apply (*simp add*:*ord-inj-Srestrict-isom*[*of D E f D1*])
done

lemma (**in** *Order*) *ord-equiv-induced*:
⟦*Order E*; *ord-isom D E f*; *D1* ⊆ *carrier D* ⟧ ⟹
ord-equiv (*Iod D D1*) (*Iod E* (*f* ‘ *D1*))
apply (*simp add*:*ord-equiv-def*)
apply (*frule ord-isom-restricted* [*of E f D1*], *assumption*+)
apply *blast*
done

lemma *ord-equiv-inducedS*:
⟦*Order D*; *Order E*; *ord-isom D E f*; *D1* ⊆ *carrier D* ⟧ ⟹
ord-equiv (*SIod D D1*) (*SIod E* (*f* ‘ *D1*))
apply (*simp add*:*ord-equiv-def*)
apply (*frule ord-isom-restrictedS* [*of D E f D1*], *assumption*+)
apply *blast*
done

lemma (**in** *Order*) *equiv-induced-by-inj*:⟦*Order E*; *ord-inj D E f*;
D1 ⊆ *carrier D*⟧ ⟹ *ord-equiv* (*Iod D D1*) (*Iod E* (*f* ‘ *D1*))
apply (*simp add*:*ord-equiv-def*)
apply (*frule ord-inj-restrict-isom* [*of E f D1*], *assumption*+)
apply *blast*
done

lemma *equiv-induced-by-injS*:⟦*Order D*; *Order E*; *ord-inj D E f*;
D1 ⊆ *carrier D*⟧ ⟹ *ord-equiv* (*SIod D D1*) (*SIod E* (*f* ‘ *D1*))
apply (*simp add*:*ord-equiv-def*)
apply (*frule ord-inj-Srestrict-isom*[*of D E f D1*], *assumption*+)
apply *blast*
done

lemma (**in** *Torder*) *le-segment-segment*:⟦*a* ∈ *carrier D*; *b* ∈ *carrier D*⟧ ⟹
(*a* ≼ *b*) = (*segment* (*Iod D* (*segment D b*)) *a* = *segment D a*)
apply (*cut-tac segment-sub*[*of b*],
frule Iod-Order[*of segment D b*])
apply (*case-tac a* = *b*) **apply** *simp*
apply (*simp add*:*le-refl*)
apply (*cut-tac a-notin-segment*[*of b*])
apply (*subst Order.segment-free*[*of Iod D* (*segment D b*) *b*], *assumption*)

apply (*simp add*:*Iod-carrier*)
apply (*simp add*:*Iod-carrier*)
apply (*subst le-imp-less-or-eq*[*of a b*], *assumption*+, *simp*)

apply (*rule iffI*)
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*frule-tac a1* = *x* **in** *Order.segment-inc*[*THEN sym*,
of Iod D (*segment D b*) - *a*])
apply (*frule-tac Order.segment-sub*[*of Iod D* (*segment D b*) *a*])
apply (*rule subsetD*, *assumption*+)
apply (*simp add*:*Iod-carrier*) **apply** (*simp add*:*segment-inc*)
apply *simp*
apply (*subst segment-inc*[*THEN sym*])
apply (*simp add*:*segment-def Iod-def*) **apply** *assumption*
apply (*simp add*:*segment-inc*)
apply (*frule Order.segment-sub*[*of Iod D* (*segment D b*) *a*])
apply (*simp add*:*Iod-carrier*)
apply (*simp add*:*subsetD Iod-less*)
apply (*rule subsetI*)
apply (*subst Order.segment-inc*[*THEN sym*, *of Iod D* (*segment D b*)],
assumption+)
apply (*simp add*:*Iod-carrier*)
apply (*simp add*:*segment-mono*[*of a b*] *psubset-eq*, *erule conjE*)
apply (*rule subsetD*[*of segment D a segment D b*], *assumption*+)
apply (*simp add*:*Iod-carrier segment-inc*)
apply (*frule segment-inc*[*of a b*], *assumption*, *simp*)
apply (*frule segment-mono*[*of a b*], *assumption*, *simp*)
apply (*simp add*:*psubset-eq*, (*erule conjE*)+)
apply (*frule-tac c* = *x* **in** *subsetD*[*of segment D a segment D b*],
assumption+)
apply (*simp add*:*Iod-less*)
apply (*subst segment-inc*) **apply** (*simp add*:*subsetD*) **apply** *assumption*+
apply (*rule contrapos-pp*, *simp*+)
apply (*simp add*:*not-less-le*)
apply (*simp add*:*le-imp-less-or-eq*)
apply (*frule segment-not-inc*[*of b a*], *assumption*+)
apply (*frule Order.segment-free*[*of Iod D* (*segment D b*) *a*])
apply (*simp add*:*Iod-carrier*)
apply (*simp add*:*Iod-carrier*)
apply (*simp add*:*segment-inj*[*THEN sym*, *of b a*])
done

lemma *le-Ssegment-Ssegment*:⟦*Torder D*; *a* ∈ *carrier D*; *b* ∈ *carrier D*⟧ ⟹
(*a* ⪯$_D$ *b*) = (*Ssegment* (*SIod D* (*Ssegment D b*)) *a* = *Ssegment D a*)
apply (*frule Torder.Order*[*of D*])
apply (*case-tac a* = *b*) **apply** *simp*
apply (*simp add*:*Order.le-refl*)

apply (*cut-tac Ssegment-sub*[*of D b*])
apply (*frule SIod-Order*[*of D Ssegment D b*], *assumption*)

apply (*cut-tac a-notin-Ssegment*[*of b D*])

apply (*frule SIod-carrier*[*THEN sym*, *of D Ssegment D b*], *assumption*+)
apply (*frule eq-set-not-inc*[*of b Ssegment D b*
carrier (*SIod D* (*Ssegment D b*))], *assumption*+)
apply (*thin-tac b* $\notin$ *Ssegment D b*,
thin-tac Ssegment D b = *carrier* (*SIod D* (*Ssegment D b*)))
apply (*cut-tac Ssegment-free*[*of b SIod D* (*Ssegment D b*)])
apply (*simp add*:*SIod-carrier*) **apply** *assumption*+

apply (*subst Order.le-imp-less-or-eq*[*of D a b*], *assumption*+)
apply *simp*

apply (*cut-tac Ssegment-sub*[*of D b*])
apply (*subst Ssegment-def*[*of SIod D* (*Ssegment D b*)],
subst SIod-carrier[*of D Ssegment D b*], *assumption*+)
apply (*subst Ssegment-inc*[*of D a b*], *assumption*+)

apply (*rule iffI*) **apply** *simp*
apply (*simp add*:*SIod-carrier*)
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*simp*, *erule conjE*)
apply (*simp add*:*SIod-less*)
apply (*subst Ssegment-def*, *simp add*:*Ssegment-def*)

apply (*rule subsetI*, *simp*)
apply (*simp add*:*Ssegment-inc*[*THEN sym*, *of D a b*])
apply (*cut-tac a1* = *x* **in** *Ssegment-inc*[*THEN sym*, *of D - a*], *assumption*+)
apply (*simp add*:*Ssegment-def*, *assumption*, *simp*)
apply (*cut-tac a* = *x* **in** *Order.less-trans*[*of D - a b*], *assumption*)
apply (*simp add*:*Ssegment-def*, *assumption*+)

apply (*cut-tac a* = *x* **in** *Ssegment-inc*[*of D - b*], *assumption*)
apply (*simp add*:*Ssegment-def*)
apply *assumption*+
apply *simp*
apply (*cut-tac a* = *a* **in** *Ssegment-inc*[*of D - b*])
apply *assumption*+
apply *simp*
apply (*simp add*:*SIod-less*)

apply (*rule contrapos-pp*, *simp*+)
apply (*simp add*:*SIod-carrier*)
apply (*frule sym*, *thin-tac Ssegment D b* = *Ssegment D a*, *simp*)

apply (*simp add*:*Ssegment-inc*[*THEN sym*, *of D a b*])
apply (*simp add*:*Torder.not-less-le*[*of D a b*])

apply (*frule not-sym*, *thin-tac* $a \neq b$)
apply (*simp add*:*Order.le-imp-less-or-eq*[*of D b a*])
apply (*simp add*:*Ssegment-inc*[*of D b a*])
apply (*simp add*:*a-notin-Ssegment*[*of b D*])
done

lemma (**in** *Torder*) *inc-segment-segment*:⟦$b \in$ *carrier D*;
$a \in$ *segment D b*⟧ $\Longrightarrow$ *segment* (*Iod D* (*segment D b*)) a = *segment D a*

apply (*cut-tac segment-sub*[*of b*],
frule subsetD[*of segment D b carrier D a*], *assumption*)
apply (*subst le-segment-segment*[*THEN sym*, *of a b*],
assumption+)
apply (*simp add*:*segment-inc*[*THEN sym*])
apply (*simp add*:*less-imp-le*)
done

lemma (**in** *Torder*) *segment-segment*:⟦$a \in$ *carrier D*; $b \in$ *carrier D*⟧ $\Longrightarrow$
(*segment* (*Iod D* (*segment D b*)) a = *segment D a*) =
((*segment D a*) $\subseteq$ (*segment D b*))
apply (*subst le-segment-segment*[*THEN sym*, *of a b*],
assumption+)
apply (*simp add*:*segment-le-mono*[*of a b*])
done

lemma (**in** *Torder*) *less-in-Iod*:⟦$a \in$ *carrier D*; $b \in$ *carrier D*; $a \prec b$⟧
$\Longrightarrow$ ($a \prec b$) = ($a \in$ *carrier* (*Iod D* (*segment D b*)))
apply (*simp add*:*Iod-def segment-inc*)
done

definition
SS :: - $\Rightarrow$ *'a set Order* **where**
SS D = (|*carrier* = {*X*. $\exists$ *a*∈*carrier D*. *X* = *segment D a*}, *rel* =
{*XX*. *XX* ∈ {*X*. $\exists$ *a*∈*carrier D*. *X* = *segment D a*} ×
{*X*. $\exists$ *a*∈*carrier D*. *X* = *segment D a*} ∧ ((*fst XX*) ⊆ (*snd XX*))} |)

definition
segmap::- $\Rightarrow$ *'a* $\Rightarrow$ *'a set* **where**
segmap D = (λ*x*∈(*carrier D*). *segment D x*)

lemma *segmap-func*:*segmap D* ∈ *carrier D* → *carrier* (*SS D*)
by (*simp add*:*SS-def segmap-def Pi-def*) *blast*

lemma (**in** *Worder*) *ord-isom-segmap*: *ord-isom D* (*SS D*) (*segmap D*)

apply (*simp add*:*ord-isom-def*)
apply (*rule conjI*)
 apply (*simp add*:*ord-inj-def*)
apply (*rule conjI*)
 apply (*simp add*:*segmap-def*)

apply (*rule conjI*)
 apply (*simp add*:*segmap-func*)

apply (*rule conjI*)
 apply (*simp add*:*inj-on-def*)
 apply ((*rule ballI*)+, *rule impI*, *simp add*:*segmap-def*,
 simp add:*segment-inj*[*THEN sym*])
 apply (*rule ballI*)+
 apply (*simp add*:*oless-def*[*of SS D*]) **apply** (*simp add*:*ole-def SS-def*)
 apply (*rule iffI*)
  apply (*simp add*:*oless-def*, *erule conjE*)
  apply (*frule-tac a* = *a* **and** *b* = *b* **in** *segment-le-mono*, *assumption*+)
  apply (*simp add*:*segment-inj segmap-def*)
  apply *blast*
 apply (*erule conjE*)+
  apply (*thin-tac* ∃ *aa*∈*carrier D*. *segmap D a* = *segment D aa*,
 thin-tac ∃ *a*∈*carrier D*. *segmap D b* = *segment D a*)
  apply (*simp add*:*segmap-def segment-inj*[*THEN sym*])
  apply (*simp add*:*segment-le-mono*[*THEN sym*])
  apply (*simp add*:*oless-def*)
 apply (*rule surj-to-test*[*of segmap D carrier D carrier* (*SS D*)])
 apply (*simp add*:*segmap-func*)
 apply (*rule ballI*)
 apply (*simp add*:*SS-def*, *erule bexE*, *simp*)
 apply (*simp add*:*segmap-def*, *blast*)
done

lemma (**in** *Worder*) *nonequiv-segment*:*a* ∈ *carrier D* ⟹
 ¬ *ord-equiv D* (*Iod D* (*segment D a*))
apply (*rule contrapos-pp*, *simp*+)
 apply (*simp add*:*ord-equiv-def*)
 apply (*erule exE*)
 apply (*cut-tac segment-sub*[*of a*])
 apply (*frule Iod-Order*[*of segment D a*])
 apply (*frule-tac f* = *f* **in** *ord-isom-func*[*of Iod D* (*segment D a*)],
 assumption+)
 apply (*frule-tac f* = *f* **and** *a* = *a* **in** *ord-isom-mem*[*of Iod D* (*segment D a*)]
 , *assumption*+)
 apply (*frule-tac f* = *f* **in** *to-subset* [*of segment D a*], *assumption*+)
 apply (*drule-tac a* = *a* **in** *forall-spec*, *assumption*)

 apply (*simp add*:*Iod-carrier*)
 apply (*frule-tac c* = *f a* **in** *subsetD*[*of segment D a carrier D*],

assumption+)
apply (*simp add*:*segment-inc*[*THEN sym*])
apply (*simp add*:*not-le-less*[*THEN sym*, *of a* -])
done

lemma *nonequiv-Ssegment*:⟦*Worder D*; *a* ∈ *carrier D*⟧ ⟹
¬ *ord-equiv D* (*SIod D* (*Ssegment D a*))
apply (*frule Worder.Order*[*of D*], *frule Worder.Torder*[*of D*])
apply (*rule contrapos-pp*, *simp*+)
apply (*simp add*:*ord-equiv-def*)
apply (*erule exE*)

apply (*cut-tac Ssegment-sub*[*of D a*])
apply (*frule SIod-Order*[*of D Ssegment D a*], *assumption*)
apply (*frule-tac f* = *f* **in** *Order.ord-isom-func*[*of D SIod D* (*Ssegment D a*)],
assumption+,
frule-tac f = *f* **and** *a* = *a* **in** *Order.ord-isom-mem*[*of D*
SIod D (*Ssegment D a*)], *assumption*+)
apply (*frule-tac f* = *f* **in** *to-subsetS* [*of D Ssegment D a*], *assumption*+)
apply (*drule-tac a* = *a* **in** *forall-spec*, *assumption*)

apply (*simp add*:*SIod-carrier*)
apply (*frule-tac c* = *f a* **in** *subsetD*[*of Ssegment D a carrier D*],
assumption+)
apply (*simp add*:*Ssegment-inc*[*THEN sym*])
apply (*simp add*:*Torder.not-le-less*[*THEN sym*, *of D a* -])
done

lemma (**in** *Worder*) *subset-Worder*: *T* ⊆ *carrier D* ⟹
Worder (*Iod D T*)
apply (*rule Worder.intro*)
apply (*simp add*: *Iod-Torder*)
apply (*rule Worder-axioms.intro*)
apply (*rule allI*, *rule impI*)
apply (*simp add*:*Iod-carrier*, *erule conjE*)
apply (*cut-tac ex-minimum*)
apply (*frule-tac A* = *X* **and** *B* = *T* **and** *C* = *carrier D* **in** *subset-trans*,
assumption+)
apply (*frule-tac a* = *X* **in** *forall-spec*, *simp*,
thin-tac ∀ *X*. *X* ⊆ *carrier D* ∧ *X* ≠ {} ⟶ (∃ *x*. *minimum-elem D X x*))
apply (*erule exE*)
apply (*simp add*:*minimum-elem-sub*)
apply *blast*
done

lemma *SIod-Worder*:⟦*Worder D*; *T* ⊆ *carrier D*⟧ ⟹ *Worder* (*SIod D T*)
apply (*frule Worder.Order*[*of D*],
frule Worder.Torder[*of D*])
apply (*rule Worder.intro*)

apply (*simp add*: *SIod-Torder*)
apply (*rule Worder-axioms.intro*)
apply (*rule allI*, *rule impI*, *erule conjE*, *simp add*:*SIod-carrier*)
apply (*frule Worder.ex-minimum*)
apply (*frule-tac* $A = X$ **and** $B = T$ **and** $C =$ *carrier* D **in** *subset-trans*,
assumption+)
apply (*frule-tac* $a = X$ **in** *forall-spec*, *simp*,
thin-tac $\forall X.\ X \subseteq$ *carrier* $D \wedge X \neq \{\} \longrightarrow (\exists x.$ *minimum-elem* $D\ X\ x))$
apply (*simp add*:*minimum-elem-Ssub*)
done

lemma (**in** *Worder*) *segment-Worder*:*Worder* (*Iod* D (*segment* $D\ a$))
apply (*rule subset-Worder* [*of segment* $D\ a$])
apply (*rule segment-sub*[*of* a])
done

lemma *Ssegment-Worder*:*Worder* $D \Longrightarrow$ *Worder* (*SIod* D (*Ssegment* $D\ a$))
apply (*rule SIod-Worder*, *assumption*)
apply (*rule Ssegment-sub*[*of* $D\ a$])
done

lemma (**in** *Worder*) *segment-unique1*:$[\![a \in$ *carrier* D; $b \in$ *carrier* D; $a \prec b]\!] \Longrightarrow$
$\neg$ *ord-equiv* (*Iod* D (*segment* $D\ b$)) (*Iod* D (*segment* $D\ a$))
apply (*cut-tac segment-Worder*[*of* b],
cut-tac segment-sub[*of* b],
frule segment-mono[*of* $a\ b$], *assumption*, *simp add*:*psubset-eq*,
erule conjE)
apply (*simp add*:*segment-inc*,
frule Worder.nonequiv-segment[*of Iod* D (*segment* $D\ b$) a],
simp add:*Iod-carrier*)
apply (*frule segment-segment*[*THEN sym*, *of* $a\ b$], *assumption*, *simp*)
apply (*simp add*:*Iod-sub-sub*[*of segment* $D\ a$ *segment* $D\ b$])
done

lemma *Ssegment-unique1*:$[\![$*Worder* D; $a \in$ *carrier* D; $b \in$ *carrier* D; $a \prec_D b]\!] \Longrightarrow$
$\neg$ *ord-equiv* (*SIod* D (*Ssegment* $D\ b$)) (*SIod* D (*Ssegment* $D\ a$))
apply (*frule Worder.Order*[*of* D], *frule Worder.Torder*[*of* D],
frule Ssegment-inc[*of* $D\ a\ b$], *assumption*+, *simp*,
frule Ssegment-Worder [*of* $D\ b$])

apply (*cut-tac Ssegment-sub*[*of* $D\ b$]) **apply** (
frule Ssegment-mono[*of* $D\ a\ b$], *assumption*+, *simp*)
apply (*frule nonequiv-Ssegment*[*of SIod* D (*Ssegment* $D\ b$) a])
apply (*simp add*:*SIod-carrier*)
apply (*frule le-Ssegment-Ssegment*[*of* $D\ a\ b$], *assumption*+)
apply (*simp add*:*oless-def psubset-eq*, (*erule conjE*)+)
apply (*simp add*:*SIod-sub-sub*[*of Ssegment* $D\ a$ *Ssegment* $D\ b$])
done

lemma (**in** *Worder*) *segment-unique*:⟦$a \in$ *carrier D*; $b \in$ *carrier D*;
ord-equiv (*Iod D* (*segment D a*)) (*Iod D* (*segment D b*)) ⟧ ⟹ $a = b$
apply (*cut-tac segment-sub*[*of a*],
frule-tac Iod-Order[*of segment D a*],
cut-tac segment-sub[*of b*],
frule-tac Iod-Order[*of segment D b*])
apply (*rule contrapos-pp*, *simp*+)
apply (*frule less-linear*[*of a b*], *assumption*+)
apply *simp*
apply (*erule disjE*)
apply (*frule segment-unique1*[*of a b*], *assumption*+)
apply (*simp add*:*Order.ord-equiv-sym*[*of Iod D* (*segment D a*)
Iod D (*segment D b*)])

apply (*simp add*:*segment-unique1*[*of b a*])
done

lemma *Ssegment-unique*:⟦*Worder D*; $a \in$ *carrier D*; $b \in$ *carrier D*;
ord-equiv (*SIod D* (*Ssegment D a*)) (*SIod D* (*Ssegment D b*)) ⟧ ⟹ $a = b$
apply (*frule Worder.Order*[*of D*], *frule Worder.Torder*[*of D*],
cut-tac Ssegment-sub[*of D b*],
cut-tac Ssegment-sub[*of D a*],
frule SIod-Order[*of D Ssegment D a*], *assumption*,
frule SIod-Order[*of D Ssegment D b*], *assumption*)

apply (*rule contrapos-pp*, *simp*+)
apply (*frule Torder.less-linear*[*of D a b*], *assumption*+)
apply *simp*

apply (*erule disjE*)
apply (*frule Ssegment-unique1*[*of D a b*], *assumption*+)
apply (*simp add*:*Order.ord-equiv-sym*[*of SIod D* (*Ssegment D a*)
SIod D (*Ssegment D b*)])

apply (*simp add*:*Ssegment-unique1*[*of D b a*])
done

lemma (**in** *Worder*) *subset-segment*:⟦$T \subseteq$ *carrier D*;
$\forall\, b{\in}T.\ \forall x.\ x \prec b \wedge x \in$ *carrier D* $\longrightarrow x \in T$;
minimum-elem D (*carrier D* − *T*) *a*⟧ ⟹ $T =$ *segment D a*
apply (*cut-tac diff-sub*[*of carrier D T*],
frule minimum-elem-mem [*of carrier D* − *T a*], *assumption*,
simp, *erule conjE*)
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*frule-tac c* = *x* **in** *subsetD*[*of T carrier D*], *assumption*+)
apply (*subst segment-inc*[*THEN sym*], *assumption*+)
apply (*frule-tac x* = *x* **in** *bspec*, *assumption*,
thin-tac $\forall\, b{\in}T.\ \forall x.\ x \prec b \wedge x \in$ *carrier D* $\longrightarrow x \in T$)

apply (*rule contrapos-pp*, *simp*+)
apply (*frule-tac* $a = x$ **and** $b = a$ **in** *not-less-le*, *assumption*+)
apply (*simp add*:*le-imp-less-or-eq*, *thin-tac* $\neg\ x \prec a$)
apply (*erule disjE*)
apply (*frule-tac* $a = a$ **in** *forall-spec*) **apply** (
thin-tac $\forall xa.\ xa \prec x \wedge xa \in carrier\ D \longrightarrow xa \in T$)
apply *simp* **apply** *simp* **apply** *simp*

apply (*rule subsetI*)
apply (*cut-tac* $a = a$ **in** *segment-sub*)
apply (*frule-tac* $c = x$ **and** $A = segment\ D\ a$ **in** *subsetD*[*of* - *carrier D*],
assumption+)
apply (*thin-tac* $\forall b{\in}T.\ \forall x.\ x \prec b \wedge x \in carrier\ D \longrightarrow x \in T$)
apply (*rule contrapos-pp*, *simp*+)
apply (*simp add*:*minimum-elem-def*)
apply (*frule-tac* $x = x$ **in** *bspec*, *simp*)
apply (*simp add*:*segment-inc*[*THEN sym*])
apply (*simp add*:*not-le-less*[*THEN sym*])
done

lemma *subset-Ssegment*:⟦*Worder D*; $T \subseteq carrier\ D$;
$\forall b{\in}T.\ \forall x.\ x \prec_D b \wedge x \in carrier\ D \longrightarrow x \in T$;
minimum-elem D (*carrier D* − *T*) *a*⟧ ⟹ $T = Ssegment\ D\ a$
apply (*cut-tac diff-sub*[*of carrier D T*],
frule Worder.Torder[*of D*],
frule Worder.Order[*of D*],
frule Order.minimum-elem-mem [*of D carrier D* − *T a*], *assumption*+,
simp, *erule conjE*)
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*frule-tac* $c = x$ **in** *subsetD*[*of T carrier D*], *assumption*+)
apply (*subst Ssegment-inc*[*THEN sym*], *assumption*+)
apply (*frule-tac* $x = x$ **in** *bspec*, *assumption*,
thin-tac $\forall b{\in}T.\ \forall x.\ x \prec_D b \wedge x \in carrier\ D \longrightarrow x \in T$)
apply (*rule contrapos-pp*, *simp*+)
apply (*frule-tac* $a = x$ **and** $b = a$ **in** *Torder.not-less-le*, *assumption*+)
apply (*simp add*:*Order.le-imp-less-or-eq*, *thin-tac* $\neg\ x \prec_D a$)
apply (*erule disjE*)
apply (*frule-tac* $a = a$ **in** *forall-spec*) **apply** (
thin-tac $\forall xa.\ xa \prec_D x \wedge xa \in carrier\ D \longrightarrow xa \in T$)
apply *simp* **apply** *simp* **apply** *simp*

apply (*rule subsetI*)
apply (*cut-tac* $a = a$ **in** *Ssegment-sub*[*of D*])
apply (*frule-tac* $c = x$ **and** $A = Ssegment\ D\ a$ **in** *subsetD*[*of* - *carrier D*],
assumption+)
apply (*thin-tac* $\forall b{\in}T.\ \forall x.\ x \prec_D b \wedge x \in carrier\ D \longrightarrow x \in T$)
apply (*rule contrapos-pp*, *simp*+)
apply (*simp add*:*minimum-elem-def*)

apply (*frule-tac* $x = x$ **in** *bspec*, *simp*,
thin-tac Ball (*carrier* $D - T$) (*op* $\preceq_D$ a))
apply (*simp add*:*Ssegment-inc*[*THEN sym*])
apply (*simp add*:*Torder.not-le-less*[*THEN sym*])
done

lemma (**in** *Worder*) *segmentTr*:⟦$T \subseteq$ *carrier* D;
$\forall\, b \in T.\ (\forall\, x.\ \ (x \prec b \wedge x \in (\mathit{carrier}\ D) \longrightarrow x \in T))$⟧ ⟹
$(T = \mathit{carrier}\ D) \vee (\exists\, a.\ a \in (\mathit{carrier}\ D) \wedge T = \mathit{segment}\ D\ a)$
apply (*case-tac* $T =$ *carrier* D)
apply *simp*

apply *simp*

apply (*frule not-sym*, *thin-tac* $T \neq$ *carrier* D,
frule diff-nonempty[*of carrier* D T], *assumption*)
apply (*cut-tac ex-minimum*)
apply (*frule-tac* $a =$ *carrier* $D - T$ **in** *forall-spec*, *simp*)
apply (*simp add*:*diff-sub*)
apply (*thin-tac* $\forall\, X.\ X \subseteq \mathit{carrier}\ D \wedge X \neq \{\} \longrightarrow (\exists\, x.\ \mathit{minimum\text{-}elem}\ D\ X\ x)$)

apply (*erule exE*, *rename-tac a*)
apply (*thin-tac carrier* $D \neq T$, *thin-tac carrier* $D - T \neq \{\}$)
apply (*cut-tac diff-sub*[*of carrier* D T])
apply (*frule-tac* $a = a$ **in** *minimum-elem-mem*[*of carrier* $D - T$],
assumption+,
thin-tac carrier $D - T \subseteq$ *carrier* D)
apply (*simp only*:*Diff-iff*, *erule conjE*)
apply (*frule-tac* $a = a$ **in** *subset-segment*[*of* T], *assumption*+)
apply *blast*
done

lemma *SsegmentTr*:⟦*Worder* D; $T \subseteq$ *carrier* D;
$\forall\, b \in T.\ (\forall\, x.\ \ (x \prec_D b \wedge x \in (\mathit{carrier}\ D) \longrightarrow x \in T))$⟧ ⟹
$(T = \mathit{carrier}\ D) \vee (\exists\, a.\ a \in (\mathit{carrier}\ D) \wedge T = \mathit{Ssegment}\ D\ a)$
apply (*case-tac* $T =$ *carrier* D)
apply *simp*

apply *simp*
apply (*frule not-sym*, *thin-tac* $T \neq$ *carrier* D,
frule diff-nonempty[*of carrier* D T], *assumption*)
apply (*cut-tac Worder.ex-minimum*[*of* D])
apply (*frule-tac* $a =$ *carrier* $D - T$ **in** *forall-spec*, *simp*)
apply (*simp add*:*diff-sub*)
apply (*thin-tac* $\forall\, X.\ X \subseteq \mathit{carrier}\ D \wedge X \neq \{\} \longrightarrow (\exists\, x.\ \mathit{minimum\text{-}elem}\ D\ X\ x)$)

apply (*erule exE*, *rename-tac a*)
apply (*thin-tac carrier D* $\neq$ *T*, *thin-tac carrier D* $-$ *T* $\neq$ {})
apply (*cut-tac diff-sub*[*of carrier D T*])
apply (*frule Worder.Order*[*of D*])
apply (*frule-tac a* = *a* **in** *Order.minimum-elem-mem*[*of D carrier D* $-$ *T*],
assumption+,
thin-tac carrier D $-$ *T* $\subseteq$ *carrier D*)
apply (*simp only*:*Diff-iff*, *erule conjE*)
apply (*subgoal-tac T* = *Ssegment D a*)
apply *blast*

apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*frule-tac c* = *x* **in** *subsetD*[*of T carrier D*], *assumption*+)
apply (*subst Ssegment-inc*[*THEN sym*], *assumption*+)
apply (*frule-tac x* = *x* **in** *bspec*, *assumption*,
thin-tac $\forall$ *b*$\in$*T*. $\forall$ *x*. *x* $\prec_D$ *b* $\wedge$ *x* $\in$ *carrier D* $\longrightarrow$ *x* $\in$ *T*)
apply (*rule contrapos-pp*, *simp*+)
apply (*frule Worder.Torder*[*of D*],
frule-tac a = *x* **and** *b* = *a* **in** *Torder.not-less-le*[*of D*], *assumption*+)
apply (*simp add*:*Order.le-imp-less-or-eq*, *thin-tac* $\neg$ *x* $\prec_D$ *a*)
apply (*erule disjE*)
apply (*frule-tac a* = *a* **in** *forall-spec*) **apply** (
thin-tac $\forall$ *xa*. *xa* $\prec_D$ *x* $\wedge$ *xa* $\in$ *carrier D* $\longrightarrow$ *xa* $\in$ *T*)
apply *simp* **apply** *simp* **apply** *simp*

apply (*rule subsetI*)
apply (*frule Worder.Torder*[*of D*],
frule Torder.Order[*of D*])
apply (*cut-tac a* = *a* **in** *Ssegment-sub*[*of D*])
apply (*frule-tac c* = *x* **and** *A* = *Ssegment D a* **in** *subsetD*[*of - carrier D*],
assumption+)
apply (*thin-tac* $\forall$ *b*$\in$*T*. $\forall$ *x*. *x* $\prec_D$ *b* $\wedge$ *x* $\in$ *carrier D* $\longrightarrow$ *x* $\in$ *T*)
apply (*rule contrapos-pp*, *simp*+)
apply (*simp add*:*minimum-elem-def*)
apply (*frule-tac x* = *x* **in** *bspec*, *simp*)
apply (*simp add*:*Ssegment-inc*[*THEN sym*])
apply (*simp add*:*Torder.not-le-less*[*THEN sym*])
apply *assumption*
done

lemma (**in** *Worder*) *ord-isom-segment-segment*:⟦*Worder E*;
ord-isom D E f; *a* $\in$ *carrier D* ⟧ $\Longrightarrow$
ord-isom (*Iod D* (*segment D a*)) (*Iod E* (*segment E* (*f a*)))
(λ*x*$\in$*carrier* (*Iod D* (*segment D a*)). *f x*)
by (*frule Worder.Order*[*of E*],
rule ord-isom-segment-segment[*of E f a*], *assumption*+)

definition

```
  Tw :: [- , ('b, 'm1) Order-scheme] ⇒ 'a ⇒ 'b  ((2Tw_-,-) [60,61]60) where
  Tw_{D,T} = (λa∈ carrier D. SOME x. x∈carrier T ∧
    ord-equiv (Iod D (segment D a)) (Iod T (segment T x)))

lemma (in Worder) Tw-func:⟦Worder T;
    ∀ a∈carrier D. ∃ b∈carrier T. ord-equiv (Iod D (segment D a))
       (Iod T (segment T b))⟧ ⟹ Tw_{D,T} ∈ carrier D → carrier T
apply (rule Pi-I)
 apply (simp add:Tw-def)
 apply (rule someI2-ex) apply blast apply simp
done

lemma (in Worder) Tw-mem:⟦Worder E; x ∈ carrier D;
    ∀ a∈carrier D. ∃ b∈carrier E. ord-equiv (Iod D (segment D a))
       (Iod E (segment E b))⟧ ⟹ (Tw_{D,E}) x ∈ carrier E
by (frule Tw-func[of E], assumption,
      simp add:Pi-def)

lemma (in Worder) Tw-equiv:⟦Worder T;
     ∀ a∈carrier D. ∃ b∈carrier T. ord-equiv (Iod D (segment D a))
                    (Iod T (segment T b)); x ∈ carrier D ⟧ ⟹
     ord-equiv (Iod D (segment D x)) (Iod T (segment T ((Tw_{D,T}) x)))
apply (frule-tac x = x in bspec, assumption+,
     thin-tac ∀ a∈carrier D.
     ∃ b∈carrier T. ord-equiv (Iod D (segment D a)) (Iod T (segment T b)))

apply (simp add:Tw-def)
apply (rule someI2-ex)
 apply blast apply simp
done

lemma (in Worder) Tw-inj:⟦Worder E;
     ∀ a∈carrier D. ∃ b∈carrier E.  ord-equiv (Iod D (segment D a))
      (Iod E (segment E b))⟧ ⟹ inj-on (Tw_{D,E}) (carrier D)

 apply (simp add:inj-on-def)
 apply (rule ballI)+ apply (rule impI)

 apply (frule-tac x = x in Tw-equiv [of E], assumption+)
 apply simp

apply (frule Tw-func[of E], assumption)
 apply (frule-tac x = x in funcset-mem[of Tw D E carrier D carrier E],
                 assumption+,
       frule-tac x = y in funcset-mem[of Tw D E carrier D carrier E],
                 assumption+)
 apply (frule Worder.Order[of E],
       cut-tac a = x in segment-sub,
       cut-tac a = y in segment-sub,
```

cut-tac a = Tw D E y **in** *Order.segment-sub*[*of E*], *assumption*)

apply (*frule-tac T = segment D x* **in** *Iod-Order*,
frule-tac T = segment D y **in** *Iod-Order*,
frule-tac T = segment E (Tw D E y) **in** *Order.Iod-Order*[*of E*],
assumption)

apply (*thin-tac Tw D E x = Tw D E y*)
apply (*frule-tac x = y* **in** *Tw-equiv*[*of E*], *assumption*+)
apply (*frule-tac D = Iod D (segment D y)* **and**
E = Iod E (segment E (Tw D E y)) **in** *Order.ord-equiv-sym*,
assumption+,
thin-tac ord-equiv (Iod D (segment D y))
(Iod E (segment E (Tw D E y))))
apply (*frule-tac D = Iod D (segment D x)* **and**
E = Iod E (segment E (Tw D E y)) **and**
F = Iod D (segment D y) **in** *Order.ord-equiv-trans*, *assumption*+)
apply (*simp add*:*segment-unique*)
done

lemma (**in** *Worder*) *Tw-eq-ord-isom*:⟦*Worder E*;
∀ *a*∈*carrier D*. ∃ *b*∈*carrier E*.
ord-equiv (Iod D (segment D a)) (Iod E (segment E b)); *a* ∈ *carrier D*;
ord-isom (Iod D (segment D a)) (Iod E (segment E (Tw D E a))) *f*;
x ∈ *segment D a* ⟧ ⟹ *f x = Tw D E x*
apply (*cut-tac segment-sub*[*of a*])

apply (*frule-tac c = x* **in** *subsetD*[*of segment D a carrier D*], *assumption*+,
frule Tw-equiv[*of E x*], *assumption*+)

apply (*frule Worder.Torder*[*of E*],
frule Torder.Order[*of E*])
apply (*cut-tac a = x* **in** *segment-Worder*,
frule-tac D = Iod D (segment D x) **in** *Worder.Torder*,
frule-tac D = Iod D (segment D x) **in** *Worder.Order*)
apply (*frule-tac T = segment D a* **in** *Iod-Order*)
apply (*frule-tac x = a* **in** *Tw-mem*[*of E*], *assumption*+)
apply (*frule-tac a = Tw D E x* **in** *Order.segment-sub*[*of E*])
apply (*frule-tac a = Tw D E a* **in** *Worder.segment-Worder*,
frule-tac D = Iod E (segment E (Tw D E a)) **in** *Worder.Order*)
apply (*frule-tac f = f* **and** *a = x* **in** *Order.ord-isom-segment-segment*[*of*
Iod D (segment D a) Iod E (segment E (Tw D E a))], *assumption*+)
apply (*simp add*:*Iod-carrier*)

apply (*frule-tac a = x* **and** *b = a* **in** *segment-le-mono*, *assumption*+)
apply (*frule-tac a1 = x* **and** *b1 = a* **in** *segment-inc*[*THEN sym*], *assumption*+)
apply (*simp add*:*oless-def*)
apply (*frule-tac a1 = x* **and** *b1 = a* **in** *segment-segment*[*THEN sym*], *assumption*+)

apply *simp*
apply (*simp add*:*Iod-sub-sub*)

apply (*frule-tac f* = *f* **and** *a* = *x* **in** *Order.ord-isom-mem*[*of*
Iod D (*segment D a*) *Iod E* (*segment E* (*Tw D E a*))],
simp add:*Iod-carrier*,
frule Order.segment-sub[*of E Tw D E a*],
simp add:*Order.Iod-carrier*, *simp add*:*Iod-carrier*,
frule Order.segment-sub[*of E Tw D E a*],
simp add:*Order.Iod-carrier*[*of E*],
frule-tac c = *f x* **in** *subsetD*[*of segment E* (*Tw D E a*)
carrier E], *assumption*+)
apply (*frule-tac a1* = *f x* **in** *Order.segment-inc*[*THEN sym*, *of E* -
Tw D E a], *assumption*+, *simp*)
apply (*simp add*:*oless-def*, (*erule conjE*)+)
apply (*frule-tac a* = *f x* **and** *b* = *Tw D E a* **in**
Torder.segment-le-mono [*of E*], *assumption*+, *simp*)
apply (*frule-tac a* = *f x* **and** *b* = *Tw D E a* **in**
Order.segment-segment[*of E*], *assumption*+)
apply *simp*
apply (*simp add*:*Order.Iod-sub-sub*)

apply (*frule-tac D* = *Iod D* (*segment D x*) **in** *Torder.Order*)
apply (*frule-tac D* = *Iod D* (*segment D x*) **and** *E* = *Iod E* (*segment E* (*f x*))
and *F* = *Iod E* (*segment E* (*Tw D E x*)) **in** *Order.ord-equiv-box*)
apply (*frule-tac a* = *f x* **in** *Order.segment-sub*[*of E*])
apply (*frule-tac T* = *segment E* (*f x*) **in** *Order.Iod-Order*[*of E*], *assumption*+)
apply (*frule-tac a* = *f x* **in** *Order.segment-sub*[*of E*])
apply (*frule Tw-mem*[*of E x*], *assumption*+)
apply (*frule Order.segment-sub*[*of E Tw D E x*])
apply (*rule Order.Iod-Order*[*of E*], *assumption*+)

apply (*simp add*:*ord-equiv-def*, *blast*)
apply *assumption*
apply (*frule-tac a* = *f x* **and** *b* = *Tw D E x* **in**
Worder.segment-unique[*of E*], *assumption*+)
apply (*frule-tac x* = *x* **in** *Tw-mem*[*of E*], *assumption*+)
done

lemma (**in** *Worder*) *Tw-ord-injTr*:⟦*Worder E*;
∀ *a*∈*carrier D*. ∃ *b*∈*carrier E*.
ord-equiv (*Iod D* (*segment D a*)) (*Iod E* (*segment E b*));
a ∈ *carrier D*; *b* ∈ *carrier D*; *a* ≺ *b*⟧ ⟹
Tw D E a ≺$_E$ (*Tw D E b*)
apply (*frule-tac x* = *b* **in** *Tw-equiv* [*of E*], *assumption*+)
apply (*simp add*:*segment-inc*)
apply (*simp add*:*ord-equiv-def*, *erule exE*, *fold ord-equiv-def*)
apply (*frule-tac f* = *f* **in** *Tw-eq-ord-isom*[*of E b* - *a*], *assumption*+)
apply (*cut-tac segment-sub*[*of b*])

apply (*frule Iod-Order*[*of segment D b*])
apply (*frule Worder.Order*[*of E*],
frule Tw-mem[*of E b*], *assumption*+,
frule Order.segment-sub[*of E Tw D E b*],
frule Order.Iod-Order[*of E segment E* (*Tw D E b*)], *assumption*)
apply (*frule-tac f* = *f* **and** *a* = *a* **in** *Order.ord-isom-mem*[*of*
Iod D (*segment D b*) *Iod E* (*segment E* (*Tw D E b*))], *assumption*+)
apply (*simp add*:*Iod-carrier*)
apply (*simp add*:*Order.Iod-carrier*)
apply (*subst Order.segment-inc*[*of E*], *assumption*+)
apply (*simp add*:*Tw-mem*)+
done

lemma (**in** *Worder*) *Tw-ord-inj*:⟦*Worder E*;
∀ *a*∈*carrier D*. ∃ *b*∈*carrier E*. *ord-equiv* (*Iod D* (*segment D a*))
(*Iod E* (*segment E b*))⟧ ⟹ *ord-inj D E* (*Tw D E*)
apply (*simp add*:*ord-inj-def*)
apply (*rule conjI*)
apply (*simp add*:*Tw-def extensional-def*)
apply (*simp add*:*Tw-func*)
apply (*rule conjI*)
apply (*simp add*:*Tw-inj*)
apply (*rule ballI*)+

apply (*rule iffI*)
apply (*simp add*:*Tw-ord-injTr*)

apply (*rule contrapos-pp*, *simp*+)
apply (*simp add*:*not-less-le*)
apply (*simp add*:*le-imp-less-or-eq*)
apply (*erule disjE*)

apply (*frule-tac a* = *b* **and** *b* = *a* **in** *Tw-ord-injTr*[*of E*], *assumption*+)
apply (*frule Tw-func* [*of E*], *assumption*+)
apply (*frule-tac x* = *a* **in** *funcset-mem*[*of Tw D E carrier D carrier E*],
assumption+,
frule-tac x = *b* **in** *funcset-mem*[*of Tw D E carrier D carrier E*],
assumption+)

apply (*frule Worder.Torder*[*of E*],
frule-tac a1 = *Tw D E b* **and** *b1* = *Tw D E a* **in**
Torder.not-le-less[*THEN sym*, *of E*], *assumption*+, *simp*)

apply (*frule Worder.Order*[*of E*],
frule-tac a = *Tw D E b* **and** *b* = *Tw D E a* **in**
Order.less-imp-le[*of E*], *assumption*+, *simp*)
apply (*simp add*:*oless-def*)
done

lemma (**in** *Worder*) *ord-isom-restricted-by-Tw*:⟦*Worder E*;
∀ *a*∈*carrier D*. ∃ *b*∈*carrier E*.
ord-equiv (*Iod D* (*segment D a*)) (*Iod E* (*segment E b*));
D1 ⊆ *carrier D*⟧ ⟹
ord-isom (*Iod D D1*) (*Iod E* ((*Tw D E*) ‘ *D1*))
(*restrict* (*Tw D E*) *D1*)
apply (*frule Tw-ord-inj* [*of E*], *assumption*+)
apply (*frule Worder.Order*[*of E*])
apply (*rule ord-inj-restrict-isom* [*of E Tw D E D1*], *assumption*+)
done

lemma (**in** *Worder*) *Tw-segment-segment*:⟦*Worder E*;
∀ *a*∈*carrier D*.∃ *b*∈*carrier E*.
ord-equiv (*Iod D* (*segment D a*)) (*Iod E* (*segment E b*)); *a* ∈ *carrier D*⟧
⟹ *Tw D E* ‘ (*segment D a*) = *segment E* (*Tw D E a*)
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*simp add*:*image-def*, *erule bexE*)
apply (*frule Tw-equiv*[*of E a*], *assumption*+)
apply (*simp add*:*ord-equiv-def*, *erule exE*, *fold ord-equiv-def*)
apply (*frule-tac x* = *xa* **in** *Tw-eq-ord-isom*[*of E a*], *assumption*+)
apply (*rotate-tac* −*1*, *frule sym*, *thin-tac f xa* = *Tw D E xa*, *simp*)
apply (*cut-tac segment-sub*[*of a*],
frule Iod-Order[*of segment D a*])
apply (*frule Worder.Order*[*of E*],
frule-tac a = *Tw D E a* **in** *Order.segment-sub*[*of E*],
frule Tw-mem[*of E a*], *assumption*+,
frule Order.segment-sub[*of E Tw D E a*])
apply (*frule-tac T* = *segment E* (*Tw D E a*) **in** *Order.Iod-Order*[*of E*],
assumption+)
apply (*frule-tac a* = *xa* **and** *f* = *f* **and** *D* = *Iod D* (*segment D a*) **and**
E = *Iod E* (*segment E* (*Tw D E a*)) **in** *Order.ord-isom-mem*,
assumption+)
apply (*simp add*:*Iod-carrier*)
apply (*simp add*:*Order.Iod-carrier*)

apply (*rule subsetI*)
apply (*simp add*:*image-def*)
apply (*frule Tw-equiv*[*of E a*], *assumption*+)
apply (*simp add*:*ord-equiv-def*, *erule exE*, *fold ord-equiv-def*)
apply (*cut-tac segment-sub*[*of a*],
frule Iod-Order[*of segment D a*])
apply (*frule Worder.Order*[*of E*],
frule-tac a = *Tw D E a* **in** *Order.segment-sub*[*of E*],
frule Tw-mem[*of E a*], *assumption*+,
frule Order.segment-sub[*of E Tw D E a*])
apply (*frule-tac T* = *segment E* (*Tw D E a*) **in** *Order.Iod-Order*[*of E*],
assumption+)
apply (*frule Iod-Order*[*of segment D a*])

```
 apply (frule-tac b = x in Order.ord-isom-surj [of Iod D (segment D a)
       Iod E (segment E (Tw D E a))], assumption+)
       apply (simp add:Order.Iod-carrier)
 apply (erule bexE, simp add:Iod-carrier)
 apply (frule-tac f = f and x = aa in Tw-eq-ord-isom[of E a], assumption+)
 apply (simp, blast)
done

lemma (in Worder) ord-isom-Tw-segment:⟦Worder E;
 ∀ a∈carrier D. ∃ b∈carrier E.
       ord-equiv (Iod D (segment D a)) (Iod E (segment E b)); a∈carrier D⟧ ⟹
  ord-isom (Iod D (segment D a)) (Iod E (segment E (Tw D E a)))
             (restrict (Tw D E) (segment D a))
apply (cut-tac segment-sub[of a],
       frule ord-isom-restricted-by-Tw[of E segment D a], assumption+,
       simp add:Tw-segment-segment[of E a])
done

lemma (in Worder) well-ord-compare1:⟦Worder E;
      ∀ a∈carrier D. ∃ b∈carrier E.
          ord-equiv (Iod D (segment D a)) (Iod E (segment E b))⟧ ⟹
    (ord-equiv D E) ∨ (∃ c∈carrier E. ord-equiv D (Iod E (segment E c)))
apply (frule Tw-ord-inj [of E], assumption+)
apply (frule Tw-func[of E], assumption+)

apply (frule ord-isom-restricted-by-Tw [of E carrier D], assumption+,
       simp)
      apply (simp add:Iod-self[THEN sym])

apply (frule image-sub0[of Tw D E carrier D carrier E],
       frule Worder.segmentTr [of E (Tw D E) ‘ (carrier D)],
       assumption)

 apply (rule ballI, rule allI, rule impI, erule conjE)
 apply (thin-tac ord-isom D (Iod E (Tw D E ‘ carrier D))
      (restrict (Tw D E) (carrier D)))

 apply (thin-tac Tw D E ‘ carrier D ⊆ carrier E,
        simp add:image-def, erule bexE)
 apply (frule-tac a = xa in ord-isom-Tw-segment[of E], assumption+)
 apply (rename-tac b x c)
  apply (frule-tac x = c in funcset-mem[of Tw D E carrier D carrier E],
        assumption, simp, thin-tac b = Tw D E c)
 apply (frule Worder.Order[of E],
        frule-tac a = Tw D E c in Order.segment-sub[of E],
        cut-tac a = c in segment-Worder,
        cut-tac a = Tw D E c in Worder.segment-Worder[of E],
        assumption,
        frule-tac D = Iod D (segment D c) in Worder.Order,
```

```
      frule-tac D = Iod E (segment E (Tw D E c)) in Worder.Order)
 apply (frule-tac D = Iod D (segment D c) and
      E = Iod E (segment E (Tw D E c)) and
      f = restrict (Tw D E) (segment D c) and b = x in
      Order.ord-isom-surj, assumption+)
      apply (simp add:Order.Iod-carrier[of E])
 apply (frule-tac a = x and b = Tw D E c in Order.segment-inc[of E],
      assumption+, simp)
 apply (insert Order,
      cut-tac a = c in segment-sub,
      simp add:Iod-carrier, erule bexE, blast)

apply (erule disjE)
 apply simp
 apply (frule Worder.Order[of E],
      simp add:Order.Iod-self[THEN sym, of E],
      simp add:ord-equiv)

apply (erule exE, erule conjE, simp,
      frule Worder.Order[of E],
      frule-tac a = a in Order.segment-sub[of E],
      cut-tac a = a in Worder.segment-Worder[of E],
      assumption,
      frule-tac D = Iod E (segment E a) in Worder.Order,
      frule-tac E = Iod E (segment E a) in ord-equiv, simp, blast)
done

lemma bex-nonempty-set:∃ x ∈ A. P x ⟹ {x. x ∈ A ∧ P x } ≠ {}
by blast

lemma nonempty-set-sub:{x. x ∈ A ∧ P x } ≠ {} ⟹
                                {x. x ∈ A ∧ P x} ⊆ A
by (rule subsetI, simp)

lemma (in Torder) less-minimum:⟦minimum-elem D {x. x ∈ carrier D ∧ P x}
d⟧
      ⟹ ∀ a. (((a ≺ d) ∧ a ∈ carrier D) ⟶ ¬ (P a))
apply (rule allI, rule impI, erule conjE)
apply (rule contrapos-pp, simp+)
apply (simp add:minimum-elem-def, (erule conjE)+)
apply (frule-tac a = a in forall-spec, simp,
      thin-tac ∀ x. x ∈ carrier D ∧ P x ⟶ d ⪯ x)
apply (simp add:not-le-less[THEN sym, of d])
done

lemma (in Torder) segment-minimum-empty:⟦X ⊆ carrier D; d ∈ X⟧ ⟹
          (minimum-elem D X d) = (segment (Iod D X) d = {})
apply (rule iffI)
apply (rule contrapos-pp, simp+)
```

apply (*frule nonempty-ex*[*of segment* (*Iod D X*) *d*], *erule exE*,
thin-tac segment (*Iod D X*) *d* ≠ {},
frule minimum-elem-mem[*of X d*], *assumption*+,
frule-tac c = *d* **in** *subsetD*[*of X carrier D*], *assumption*+)
apply (*simp add*:*segment-def*,
simp add:*Iod-carrier*, *erule conjE*,
simp add:*Iod-less*[*of X*])
apply (*simp add*:*minimum-elem-def*,
frule-tac x = *x* **in** *bspec*, *assumption*,
frule-tac c = *x* **in** *subsetD*[*of X carrier D*], *assumption*+,
frule-tac a1 = *x* **and** *b1* = *d* **in** *not-less-le*[*THEN sym*], *assumption*+)
apply *simp*

apply (*rule contrapos-pp*, *simp*+)
apply (*simp add*:*minimum-elem-def*)
apply (*erule bexE*)
apply (*frule-tac c* = *d* **in** *subsetD*[*of X carrier D*], *assumption*+,
frule-tac c = *x* **in** *subsetD*[*of X carrier D*], *assumption*+,
simp add:*not-le-less*)
apply (*simp add*:*segment-def Iod-carrier*,
simp add:*Iod-less*[*THEN sym*, *of X*])
done

end

theory *Algebra2*
imports *Algebra1*
begin

lemma (**in** *Order*) *less-and-segment*:*b* ∈ *carrier D* ⟹
(∀ *a*.((*a* ≺ *b* ∧ *a* ∈ *carrier D*) ⟶ (*Q a*))) =
(∀ *a*∈*carrier* (*Iod D* (*segment D b*)).(*Q a*))
apply (*rule iffI*)
apply (*rule ballI*)
apply (*cut-tac segment-sub*[*of b*], *simp add*:*Iod-carrier*,
thin-tac segment D b ⊆ *carrier D*,
simp add:*segment-def*)
apply (*rule allI*, *rule impI*, *erule conjE*)
apply (*cut-tac segment-sub*[*of b*], *simp add*:*Iod-carrier*,
thin-tac segment D b ⊆ *carrier D*,
simp add:*segment-def*)
done

lemma (**in** *Worder*) *Word-compare2*:⟦*Worder E*;
¬ (∀ *a*∈*carrier D*. ∃ *b*∈*carrier E*. *ord-equiv* (*Iod D* (*segment D a*))
(*Iod E* (*segment E b*)))⟧ ⟹
∃ *c*∈*carrier D*. *ord-equiv* (*Iod D* (*segment D c*)) *E*
apply *simp*

apply (*frule bex-nonempty-set*[*of carrier D*],
frule nonempty-set-sub[*of carrier D* -],
thin-tac $\exists\, a{\in}$*carrier D*. $\forall\, b{\in}$*carrier E*.
$\neg$ *ord-equiv* (*Iod D* (*segment D a*)) (*Iod E* (*segment E b*)))

apply (*insert ex-minimum*,
frule-tac a = $\{x \in$ *carrier D*. $\forall\, b{\in}$*carrier E*.
$\neg$ *ord-equiv* (*Iod D* (*segment D x*)) (*Iod E* (*segment E b*))$\}$ **in**
forall-spec, *simp*,
thin-tac $\{x \in$ *carrier D*. $\forall\, b{\in}$*carrier E*.
$\neg$ *ord-equiv* (*Iod D* (*segment D x*)) (*Iod E* (*segment E b*))$\} \neq \{\}$,
thin-tac $\forall\, X.\ X \subseteq$ *carrier D* $\wedge\ X \neq \{\} \longrightarrow (\exists\, x.$ *minimum-elem D X x*),
erule exE)
apply (*frule-tac X* = $\{x \in$ *carrier D*. $\forall\, b{\in}$*carrier E*.
$\neg$ *ord-equiv* (*Iod D* (*segment D x*)) (*Iod E* (*segment E b*))$\}$ **and** *a* = *x* **in**
minimum-elem-mem, *assumption*, *simp*)
apply (*rename-tac d*)
apply (*erule conjE*, *thin-tac* $\forall\, b{\in}$*carrier E*.
$\neg$ *ord-equiv* (*Iod D* (*segment D d*)) (*Iod E* (*segment E b*)))
apply (*frule-tac d* = *d* **in** *less-minimum*) **apply** *simp*

apply (*simp add*:*less-and-segment*)
apply (*cut-tac a* = *d* **in** *segment-Worder*)
apply (*frule-tac D* = *Iod D* (*segment D d*) **in** *Worder.well-ord-compare1* [*of* - *E*], *assumption*+)
apply (*rule ballI*)
apply (*frule-tac x* = *a* **in** *bspec*, *assumption*+)
apply (*simp add*:*Iod-segment-segment*)

apply (*erule disjE*) **apply** *blast*
apply (*frule-tac X* = $\{x \in$ *carrier D*. $\forall\, b{\in}$*carrier E*.
$\neg$ *ord-equiv* (*Iod D* (*segment D x*)) (*Iod E* (*segment E b*))$\}$ **and**
a = *d* **in** *minimum-elem-mem*, *assumption*, *simp*)
done

lemma (**in** *Worder*) *Worder-equiv*:$[\![$*Worder E*;
$\forall\, a{\in}$*carrier D*. $\exists\, b{\in}$*carrier E*. *ord-equiv* (*Iod D* (*segment D a*))
(*Iod E* (*segment E b*));
$\forall\, c{\in}$*carrier E*. $\exists\, d{\in}$*carrier D*. *ord-equiv* (*Iod E* (*segment E c*))
(*Iod D* (*segment D d*))$]\!] \Longrightarrow$ *ord-equiv D E*
apply (*frule well-ord-compare1* [*of E*], *assumption*+,
erule disjE, *assumption*)
apply (*erule bexE*,
insert Worder,
frule-tac x = *c* **in** *bspec*, *assumption*+,
thin-tac $\forall\, c{\in}$*carrier E*. $\exists\, d{\in}$*carrier D*.
ord-equiv (*Iod E* (*segment E c*)) (*Iod D* (*segment D d*)),
erule bexE)

apply (*cut-tac* $a = d$ **in** *segment-Worder*,
  cut-tac $D = E$ **and** $a = c$ **in** *Worder.segment-Worder*, *assumption*+,
  frule-tac $D =$ *Iod* D (*segment* D d) **in** *Worder.Order*,
  frule-tac $D =$ *Iod* E (*segment* E c) **in** *Worder.Order*,
  insert Order)
apply (*frule-tac* $D = D$ **and** $E =$ *Iod* E (*segment* E c) **and**
  $F =$ *Iod* D (*segment* D d) **in** *Order.ord-equiv-trans*, *assumption*+,
  frule-tac $a = d$ **in** *nonequiv-segment*)
apply *simp*
done

lemma (**in** *Worder*) *Worder-equiv1*:⟦*Worder E*; ¬ *ord-equiv D E*⟧ ⟹
  ¬ ((∀ *a*∈*carrier D*. ∃ *b*∈*carrier E*.
    ord-equiv (*Iod D* (*segment D a*)) (*Iod E* (*segment E b*))) ∧
    (∀ *c*∈*carrier E*. ∃ *d*∈*carrier D*.
    ord-equiv (*Iod E* (*segment E c*)) (*Iod D* (*segment D d*))))
apply (*rule contrapos-pp*, *simp*+) **apply** (*erule conjE*)
apply (*frule Worder-equiv* [*of E*], *assumption*+)
apply *simp*
done

lemma (**in** *Worder*) *Word-compare*:*Worder E* ⟹
 (∃ *a*∈*carrier D*. *ord-equiv* (*Iod D* (*segment D a*)) *E*) ∨ *ord-equiv D E* ∨
 (∃ *b*∈*carrier E*. *ord-equiv D* (*Iod E* (*segment E b*)))
apply (*frule Worder.Order*[*of E*],
  case-tac ord-equiv D E, *simp*)
 apply (*frule Worder-equiv1* [*of E*], *assumption*+)
 apply *simp*
 apply (*erule disjE*)
 apply (*frule Word-compare2* [*of E*], *simp*)
 apply *blast*
 apply (*cut-tac Worder.Word-compare2* [*of E D*])
apply (*thin-tac* ∃ *c*∈*carrier E*. ∀ *d*∈*carrier D*.
    ¬ *ord-equiv* (*Iod E* (*segment E c*)) (*Iod D* (*segment D d*)))
 apply (*erule bexE*,
  cut-tac $a = c$ **in** *Worder.segment-Worder*[*of E*], *assumption*+,
  frule-tac $D =$ *Iod* E (*segment* E c) **in** *Worder.Order*,
  insert Order,
  frule-tac $D =$ *Iod* E (*segment* E c) **and** $E = D$ **in** *Order.ord-equiv-sym*,
  assumption+, *blast*)
 apply *assumption* **apply** (*simp add*:*Worder*)
 apply *simp*
done

lemma (**in** *Worder*) *Word-compareTr1*:⟦*Worder E*;
  ∃ *a*∈*carrier D*. *ord-equiv* (*Iod D* (*segment D a*)) *E*; *ord-equiv D E* ⟧ ⟹
  False
apply (*erule bexE*,
  cut-tac $a = a$ **in** *segment-Worder*,

frule-tac $D = E$ **in** *Worder.Order*,
frule-tac $D =$ *Iod* D (*segment* D a) **in** *Worder.Order*,
frule-tac $D =$ *Iod* D (*segment* D a) **and** $E = E$ **in** *Order.ord-equiv-sym*,
assumption+)
apply (*insert Order*,
frule-tac $D = D$ **and** $E = E$ **and** $F =$ *Iod* D (*segment* D a) **in**
Order.ord-equiv-trans, *assumption*+)
apply (*frule-tac* $a = a$ **in** *nonequiv-segment*, *simp*)
done

lemma (**in** *Worder*) *Word-compareTr2*:⟦*Worder E*; *ord-equiv D E*;
$\exists\, b{\in}$*carrier E*. *ord-equiv D* (*Iod E* (*segment E b*))⟧ ⟹ *False*
apply (*erule bexE*)
apply (*cut-tac* $a = b$ **in** *Worder.segment-Worder* [*of E*], *assumption*+)
apply (*cut-tac Worder*,
frule Worder.Order[*of E*])
apply (*frule-tac* $D =$ *Iod E* (*segment E b*) **in** *Worder.Order*)
apply (*frule-tac* $E = E$ **in** *ord-equiv-sym*, *assumption*)
apply (*meson Worder Worder.Word-compareTr1 ord-equiv-sym*)
done

lemma (**in** *Worder*) *Word-compareTr3*:⟦*Worder E*;
$\exists\, b{\in}$*carrier E*. *ord-equiv D* (*Iod E* (*segment E b*));
$\exists\, a{\in}$*carrier D*. *ord-equiv* (*Iod D* (*segment D a*)) *E*⟧ ⟹ *False*
apply (*erule bexE*)+
apply (*simp add*:*ord-equiv-def*[*of D*], *erule exE*)

apply (*frule Worder.Torder*[*of E*])
apply (*cut-tac* $a = b$ **in** *Worder.segment-Worder* [*of E*], *assumption*+,
frule-tac $D =$ *Iod E* (*segment E b*) **in** *Worder.Torder*,
frule-tac $D =$ *Iod E* (*segment E b*) **in** *Worder.Order*)
apply (*frule-tac* $E =$ *Iod E* (*segment E b*) **and** $f = f$ **and** $a = a$ **in**
ord-isom-segment-segment, *assumption*+)
apply (*frule-tac* $f = f$ **and** $a = a$ **and** $E =$ *Iod E* (*segment E b*) **in**
ord-isom-mem, *assumption*+,
frule Worder.Order[*of E*],
frule-tac $b = b$ **and** $a = f\ a$ **in** *Order.Iod-segment-segment*[*of E*],
assumption+, *simp*,
thin-tac Iod (*Iod E* (*segment E b*)) (*segment* (*Iod E* (*segment E b*))
($f\ a$)) = *Iod E* (*segment E* ($f\ a$)))

apply (*frule-tac* $D = E$ **and** $a = b$ **in** *Order.segment-sub*)
apply (*simp add*:*Order.Iod-carrier*[*of E*])
apply (*frule-tac* $c = f\ a$ **and** $A =$ *segment E b* **and** $B =$ *carrier E*
in *subsetD*, *assumption*+)
apply (*cut-tac* $a = a$ **in** *segment-Worder*,
frule-tac $D =$ *Iod D* (*segment D a*) **in** *Worder.Order*,
cut-tac $a = f\ a$ **in** *Worder.segment-Worder* [*of E*], *assumption*+,
frule-tac $D =$ *Iod E* (*segment E* ($f\ a$)) **in** *Worder.Order*)

```
apply (
        frule-tac D = Iod D (segment D a) and E = Iod E (segment E (f a)) in
Order.ord-equiv, assumption, simp)
 apply (frule-tac D = Iod D (segment D a) and E = E in
                                        Order.ord-equiv-sym, assumption+)
 apply (frule-tac D = E and E = Iod D (segment D a) and
       F = Iod E (segment E (f a)) in Order.ord-equiv-trans, assumption+)

 apply (simp add:Worder.nonequiv-segment[of E])
done

lemma (in Worder) subset-equiv-segment:S ⊆ carrier D ⟹
      ord-equiv D (Iod D S) ∨
      (∃ a∈carrier D. ord-equiv (Iod D S) (Iod D (segment D a)))
apply (frule subset-Worder [of S])
 apply (frule Word-compare [of Iod D S])

 apply (erule disjE)
 apply (erule bexE)
 apply (cut-tac a = a in segment-Worder,
       frule-tac D = Iod D (segment D a) in Worder.Order,
       frule Worder.Order[of Iod D S],
       frule-tac D = Iod D (segment D a) in Order.ord-equiv-sym[of -
             Iod D S], assumption+, blast)

apply (erule disjE) apply simp
 apply (erule bexE)
 apply (frule Worder.Order[of Iod D S],
       frule-tac D = Iod D S and a = b in Order.segment-sub)
 apply (frule-tac S = segment (Iod D S) b in Order.Iod-sub-sub[of Iod D S - S])
 apply (simp add:Iod-carrier) apply (simp add:Iod-carrier)
 apply (simp add:Iod-sub-sub[of S S])
 apply (simp add:Iod-carrier)
 apply (frule-tac S = segment (Iod D S) b in Iod-sub-sub[of - S],
        assumption+, simp,
       thin-tac Iod (Iod D S) (segment (Iod D S) b) =
                                        Iod D (segment (Iod D S) b))

 apply (simp add:ord-equiv-def, erule exE)
 apply (frule-tac A = segment (Iod D S) b and B = S and C = carrier D in
        subset-trans, assumption+,
       frule-tac c = b in subsetD[of S carrier D], assumption+)
 apply (frule-tac T = segment (Iod D S) b and f = f in to-subset,
        assumption+,
       frule-tac a = b in forall-spec, assumption,
       thin-tac ∀ a. a ∈ carrier D ⟶ a ≼ f a,
       frule-tac T = segment (Iod D S) b in Iod-Order,
       cut-tac D = Iod D S and a = b in Worder.segment-Worder)
 apply (simp add:Iod-carrier)
```

apply (*frule-tac* S = *segment* (*Iod* D S) b **in** *Iod-sub-sub*[*of* - S],
assumption+, *simp*,
thin-tac Iod (*Iod* D S) (*segment* (*Iod* D S) b) =
Iod D (*segment* (*Iod* D S) b))
apply (*insert Order*,
frule-tac E = *Iod* D (*segment* (*Iod* D S) b) **and** f = f **and** a = b **in**
ord-isom-mem, *assumption*+,
simp add:*Iod-carrier*,
frule-tac c = f b **and** A = *segment* (*Iod* D S) b **in**
subsetD[*of* - S], *assumption*+,
simp add:*segment-def*[*of Iod* D S],
simp add:*Iod-carrier*,
simp add:*Iod-less*)
apply (*frule-tac* c = b **in** *subsetD*[*of* S *carrier* D], *assumption*+,
frule-tac c = f b **in** *subsetD*[*of* S *carrier* D], *assumption*+,
frule-tac a1 = f b **and** *b1* = b **in** *not-less-le*[*THEN sym*], *assumption*+)
apply *simp*
done

definition
ordinal-number :: *'a Order* ⇒ *'a Order set* **where**
ordinal-number S = {X. *Worder* X ∧ *ord-equiv* X S}

definition
ODnums :: *'a Order set set* **where**
ODnums = {X. ∃ S. *Worder* S ∧ X = *ordinal-number* S}

definition
ODord :: [*'a Order set*, *'a Order set*] ⇒ *bool* (**infix** ⊏ *60*) **where**
X ⊏ Y ⟷ (∃ x ∈ X. ∃ y ∈ Y. (∃ c∈*carrier* y. *ord-equiv* x (*Iod* y (*segment* y c))))

definition
ODord-le :: [*'a Order set*, *'a Order set*] ⇒ *bool* (**infix** ⊑ *60*) **where**
X ⊑ Y ⟷ X = Y ∨ *ODord* X Y

definition
ODrel :: (((*'a Order*) *set*) ∗ ((*'a Order*) *set*)) *set* **where**
ODrel = {Z. Z ∈ *ODnums* × *ODnums* ∧ *ODord-le* (*fst* Z) (*snd* Z)}

definition
ODnods :: (*'a Order set*) *Order* **where**
ODnods = (|*carrier* = *ODnums*, *rel* = *ODrel* |)

lemma *Worder-ord-equivTr*:⟦*Worder* S; *Worder* T⟧ ⟹
ord-equiv S T = (∃ f. *ord-isom* S T f)
by (*frule Worder.Order*[*of* S], *frule Worder.Order*[*of* T],
simp add:*ord-equiv-def*)

lemma *Worder-ord-isom-mem*:⟦*Worder S*; *Worder T*; *ord-isom S T f*; *a* ∈ *carrier S*⟧
⟹ *f a* ∈ *carrier T*
by (*frule Worder.Order*[*of S*], *frule Worder.Order*[*of T*],
simp add:*Order.ord-isom-mem*)

lemma *Worder-refl*:*Worder S* ⟹ *ord-equiv S S*
apply (*frule-tac Worder.Order* [*of S*])
apply (*rule Order.ord-equiv-reflex* [*of S*], *assumption*+)
done

lemma *Worder-sym*:⟦*Worder S*; *Worder T*; *ord-equiv S T* ⟧ ⟹ *ord-equiv T S*
apply (*frule-tac Worder.Order* [*of S*])
apply (*frule-tac Worder.Order* [*of T*])
apply (*rule Order.ord-equiv-sym* [*of S T*], *assumption*+)
done

lemma *Worder-trans*:⟦*Worder S*; *Worder T*; *Worder U*; *ord-equiv S T*; *ord-equiv T U*⟧ ⟹ *ord-equiv S U*
apply (*frule Worder.Order* [*of S*],
frule Worder.Order [*of T*],
frule Worder.Order [*of U*])
apply (*rule Order.ord-equiv-trans* [*of S T U*], *assumption*+)
done

lemma *ordinal-inc-self*:*Worder S* ⟹ *S* ∈ *ordinal-number S*
by (*simp add*:*ordinal-number-def*, *simp add*:*Worder-refl*)

lemma *ordinal-number-eq*:⟦*Worder D*; *Worder E*⟧ ⟹
(*ord-equiv D E*) = (*ordinal-number D* = *ordinal-number E*)
apply (*rule iffI*)
apply (*simp add*:*ordinal-number-def*)
apply (*rule equalityI*)
apply (*rule subsetI*) **apply** *simp* **apply** (*erule conjE*)
apply (*rule-tac S* = *x* **and** *T* = *D* **and** *U* = *E* **in** *Worder-trans*,
assumption+)
apply (*rule subsetI*, *simp*, *erule conjE*)
apply (*rule-tac S* = *x* **and** *T* = *E* **and** *U* = *D* **in** *Worder-trans*,
assumption+)
apply (*rule Worder-sym*, *assumption*+)
apply (*simp add*:*ordinal-number-def*)
apply (*frule Worder-refl*[*of D*],
frule Worder-refl[*of E*])
apply *blast*
done

lemma *mem-ordinal-number-equiv*:⟦*Worder D*;
X ∈ *ordinal-number D*⟧ ⟹ *ord-equiv X D*
by (*simp add*:*ordinal-number-def*)

lemma *mem-ordinal-number-Worder*:⟦*Worder D*;
X ∈ *ordinal-number D*⟧ ⟹ *Worder X*
by (*simp add*:*ordinal-number-def*)

lemma *mem-ordinal-number-Worder1*:⟦*x* ∈ *ODnums*; *X* ∈ *x*⟧ ⟹ *Worder X*
apply (*simp add*:*ODnums-def*, *erule exE*, *erule conjE*, *simp*)
apply (*rule mem-ordinal-number-Worder*, *assumption*+)
done

lemma *mem-ODnums-nonempty*:*X* ∈ *ODnums* ⟹ ∃ *x*. *x* ∈ *X*
apply (*simp add*:*ODnums-def*, *simp add*:*ordinal-number-def*,
erule exE, *erule conjE*)
apply (*frule-tac S* = *S* **in** *Worder-refl*, *blast*)
done

lemma *carr-ODnods*:*carrier ODnods* = *ODnums*
by (*simp add*:*ODnods-def*)

lemma *ordinal-number-mem-carrier-ODnods*:
Worder D ⟹ *ordinal-number D* ∈ *carrier ODnods*
by (*simp add*:*ODnods-def ODnums-def*, *blast*)

lemma *ordinal-number-mem-ODnums*:
Worder D ⟹ *ordinal-number D* ∈ *ODnums*
by (*simp add*:*ODnums-def*, *blast*)

lemma *ODordTr1*: ⟦*Worder D*; *Worder E*⟧ ⟹
(*ODord* (*ordinal-number D*) (*ordinal-number E*)) =
(∃ *b*∈ *carrier E*. *ord-equiv D* (*Iod E* (*segment E b*)))
apply (*rule iffI*)
apply (*simp add*:*ODord-def*)
apply ((*erule bexE*)+, *simp add*:*ordinal-number-def*, (*erule conjE*)+)
apply (*rename-tac X Y c*)

apply (*frule-tac S* = *Y* **and** *T* = *E* **in** *Worder-ord-equivTr*, *assumption*,
simp, *erule exE*)
apply (*frule-tac D* = *Y* **in** *Worder.Order*,
frule-tac D = *E* **in** *Worder.Order*,
frule-tac D = *Y* **and** *E* = *E* **and** *f* = *f* **and** *a* = *c* **in**
Order.ord-isom-segment-segment, *assumption*+,
frule-tac S = *Y* **and** *T* = *E* **and** *f* = *f* **and** *a* = *c* **in**
Worder-ord-isom-mem, *assumption*+)
apply (*cut-tac D* = *Y* **and** *a* = *c* **in** *Worder.segment-Worder*, *assumption*,
cut-tac D = *E* **and** *a* = *f c* **in** *Worder.segment-Worder*, *assumption*)
apply (*frule-tac S1* = *Iod Y* (*segment Y c*) **and** *T1* = *Iod E* (*segment E* (*f c*))
in *Worder-ord-equivTr*[*THEN sym*], *assumption*+)
apply (*frule-tac S* = *X* **and** *T* = *D* **in** *Worder-sym*, *assumption*+,
thin-tac ord-equiv X D,

frule-tac $S = D$ **and** $T = X$ **and** $U = Iod\ Y\ (segment\ Y\ c)$ **in**
Worder-trans, *assumption*+,
frule-tac $S = D$ **and** $T = Iod\ Y\ (segment\ Y\ c)$ **and**
$U = Iod\ E\ (segment\ E\ (f\ c))$ **in** *Worder-trans*, *assumption*+)
apply *blast* **apply** *blast*

apply (*simp add*:*ODord-def*)
apply (*frule ordinal-inc-self*[*of D*],
frule ordinal-inc-self[*of E*], *blast*)
done

lemma *ODord*: $[\![Worder\ D;\ d \in carrier\ D]\!] \Longrightarrow$
$ODord\ (ordinal\text{-}number\ (Iod\ D\ (segment\ D\ d)))\ (ordinal\text{-}number\ D)$
apply (*cut-tac Worder.segment-Worder*[*of D d*],
subst ODordTr1[*of Iod D* (*segment D d*) *D*], *assumption*+,
frule Worder-refl[*of Iod D* (*segment D d*)], *blast*, *assumption*)
done

lemma *ord-less-ODord*:[$[\![Worder\ D;\ c \in carrier\ D;\ d \in carrier\ D;$
$a = ordinal\text{-}number\ (Iod\ D\ (segment\ D\ c));$
$b = ordinal\text{-}number\ (Iod\ D\ (segment\ D\ d))]\!] \Longrightarrow$
$c \prec_D d = \ a \sqsubset b$
apply (*rule iffI*)
apply (*frule Worder.Order*[*of D*])
apply (*simp add*:*Order.segment-inc*)
apply (*frule Order.Iod-carr-segment*[*THEN sym*, *of D d*],
frule eq-set-inc[*of c segment D d carrier* (*Iod D* (*segment D d*))],
assumption+,
thin-tac segment D d = carrier (*Iod D* (*segment D d*)),
thin-tac c $\in$ *segment D d*)
apply (*cut-tac Worder.segment-Worder*[*of D d*],
frule ODord[*of Iod D* (*segment D d*) *c*], *assumption*+,
simp add:*Order.Iod-segment-segment*, *assumption*)

apply *simp*
apply (*frule Worder.segment-Worder*[*of D c*],
frule Worder.segment-Worder[*of D d*])
apply (*simp add*:*ODordTr1*[*of Iod D* (*segment D c*) *Iod D* (*segment D d*)],
thin-tac a = ordinal-number (*Iod D* (*segment D c*)),
thin-tac b = ordinal-number (*Iod D* (*segment D d*)))
apply (*erule bexE*)
apply (*frule Worder.Order*[*of D*])

apply (*simp add*:*Order.Iod-segment-segment*)
apply (*simp add*:*Order.Iod-carr-segment*,
frule Order.segment-sub[*of D d*],
frule-tac c = b **in** *subsetD*[*of segment D d carrier D*], *assumption*+)
apply (*frule-tac b = b* **in** *Worder.segment-unique*[*of D c*], *assumption*+, *simp*)
apply (*simp add*:*Order.segment-inc*[*THEN sym*])

done

lemma *ODord-le-ref*:⟦$X \in ODnums$; $Y \in ODnums$; *ODord-le* X Y; $Y \sqsubseteq X$ ⟧
$\Longrightarrow$
$X = Y$
apply (*simp add*:*ODnums-def*)
apply ((*erule exE*)+, (*erule conjE*)+, *rename-tac S T*)
apply (*simp add*:*ODord-le-def*)
apply (*erule disjE*, *simp*)
apply (*erule disjE*, *simp*)

apply (*simp add*:*ODordTr1*)
apply (*frule-tac* $D = T$ **and** $E = S$ **in** *Worder.Word-compareTr3*, *assumption*+)
apply (*erule bexE*)
apply (*frule-tac* $D = T$ **and** $a = b$ **in** *Worder.segment-Worder*)
apply (*frule-tac* $S = S$ **and** $T = Iod\ T\ (segment\ T\ b)$ **in** *Worder-sym*,
assumption+) **apply** *blast*

apply *simp*
done

lemma *ODord-le-trans*:⟦$X \in ODnums$; $Y \in ODnums$; $Z \in ODnums$; $X \sqsubseteq Y$; $Y \sqsubseteq Z$ ⟧
$\Longrightarrow X \sqsubseteq Z$
apply (*simp add*:*ODord-le-def*)
apply (*erule disjE*, *simp*)
apply (*erule disjE*, *simp*)

apply (*simp add*:*ODnums-def*, (*erule exE*)+, (*erule conjE*)+)
apply (*rename-tac A B C*, *simp*)
apply (*simp add*:*ODordTr1*,
thin-tac $X = ordinal\text{-}number\ A$,
thin-tac $Y = ordinal\text{-}number\ B$,
thin-tac $Z = ordinal\text{-}number\ C$)
apply (*erule bexE*)+

apply (*frule-tac* $D = B$ **in** *Worder.Order*,
frule-tac $D = C$ **in** *Worder.Order*,
frule-tac $D = C$ **and** $a = ba$ **in** *Worder.segment-Worder*,
frule-tac $D = Iod\ C\ (segment\ C\ ba)$ **in** *Worder.Order*,
frule-tac $D = B$ **and** $E = Iod\ C\ (segment\ C\ ba)$ **in**
Order.ord-equiv-isom, *assumption*+, *erule exE*)
apply (*frule-tac* $D = B$ **and** $E = Iod\ C\ (segment\ C\ ba)$ **and** $f = f$ **in**
Order.ord-isom-segment-segment, *assumption*+,
frule-tac $D = B$ **and** $E = Iod\ C\ (segment\ C\ ba)$ **and** $f = f$ **and** $a = b$ **in**
Order.ord-isom-mem, *assumption*+)

apply (*simp add*:*Order.Iod-segment-segment*)
apply (*frule-tac* $D = B$ **and** $a = b$ **in** *Worder.segment-Worder*,
frule-tac $D =$ *Iod* B (*segment* B b) **in** *Worder.Order*,
frule-tac $D = C$ **and** $a = f\ b$ **in** *Worder.segment-Worder*,
frule-tac $D =$ *Iod* C (*segment* C (f b)) **in** *Worder.Order*)
apply (*frule-tac* $D =$ *Iod* B (*segment* B b) **and** $E =$ *Iod* C (*segment* C (f b))
in *Order.ord-equiv*, *assumption*+)
apply (*frule-tac* $S = A$ **and** $T =$ *Iod* B (*segment* B b) **and**
$U =$ *Iod* C (*segment* C (f b)) **in** *Worder-trans*, *assumption*+)
apply (*simp add*:*Order.Iod-carr-segment*)
apply (*frule-tac* $D = C$ **and** $a = ba$ **in** *Order.segment-sub*,
frule-tac $c = f\ b$ **and** $A =$ *segment* C ba **and** $B =$ *carrier* C **in**
subsetD, *assumption*+)
apply *blast*
done

lemma *ordinal-numberTr1*: $X \in$ *carrier ODnods* $\Longrightarrow \exists D.$ *Worder* $D \wedge D \in X$
apply (*simp add*:*ODnods-def*, *simp add*:*ODnums-def*)
apply (*erule exE*, *erule conjE*)
apply (*simp add*:*ordinal-number-def*)
apply (*frule-tac* $S = S$ **in** *Worder-refl*, *blast*)
done

lemma *ordinal-numberTr1-1*: $X \in$ *ODnums* $\Longrightarrow \exists D.$ *Worder* $D \wedge D \in X$
apply (*simp add*:*ODnums-def*, *erule exE*, *erule conjE*,
simp add:*ordinal-number-def*)
apply (*frule-tac* $S = S$ **in** *Worder-refl*, *blast*)
done

lemma *ordinal-numberTr1-2*:$[\![x \in$ *ODnums*; $S \in x$; $T \in x]\!] \Longrightarrow$
ord-equiv S T
by (*simp add*:*ODnums-def*, *erule exE*, *erule conjE*,
simp add:*ordinal-number-def*, (*erule conjE*)+,
frule-tac $S = T$ **and** $T = Sa$ **in** *Worder-sym*, *assumption*, *assumption*,
rule-tac $S = S$ **and** $T = Sa$ **and** $U = T$ **in** *Worder-trans*, *assumption*+)

lemma *ordinal-numberTr2*:$[\![$*Worder* D; $x =$ *ordinal-number* $D]\!] \Longrightarrow$
$D \in x$
by (*simp add*:*ordinal-inc-self*)

lemma *ordinal-numberTr3*:$[\![$*Worder* D; *Worder* F; *ord-equiv* D F;
$x =$ *ordinal-number* $D]\!] \Longrightarrow x =$ *ordinal-number* F
apply (*simp add*:*ordinal-number-def*,
thin-tac $x = \{X.$ *Worder* $X \wedge$ *ord-equiv* X $D\}$)
apply (*rule equalityI*)
apply (*rule subsetI*, *simp*, *erule conjE*)
apply (*rule-tac* $S = x$ **and** $T = D$ **and** $U = F$ **in** *Worder-trans*, *assumption*+)
apply (*rule subsetI*, *simp*, *erule conjE*)
apply (*frule Worder-sym* [*of* D F], *assumption*+,

rule-tac $S = x$ **and** $T = F$ **and** $U = D$ **in** *Worder-trans*, *assumption*+)
done

lemma *ordinal-numberTr4*:⟦*Worder D*; $X \in$ *carrier ODnods*; $D \in X$ ⟧ ⟹
$X =$ *ordinal-number D*
apply (*simp add*:*ODnods-def ODnums-def*, *erule exE*, *erule conjE*)
apply *simp*
apply (*frule-tac* $D = S$ **in** *mem-ordinal-number-equiv*[*of* - *D*], *assumption*+,
frule-tac $D = D$ **and** $E = S$ **in** *ordinal-number-eq*, *assumption*+)
apply *simp*
done

lemma *ordinal-numberTr5*:⟦$x \in$ *ODnums*; $D \in x$⟧ ⟹ $x =$ *ordinal-number D*
apply (*frule mem-ordinal-number-Worder1*[*of x D*], *assumption*+)
apply (*rule ordinal-numberTr4*[*of D x*], *assumption*,
simp add:*ODnods-def*, *assumption*)
done

lemma *ordinal-number-ord*:⟦ $X \in$ *carrier ODnods*; $Y \in$ *carrier ODnods*⟧ ⟹
ODord X Y ∨ $X = Y$ ∨ *ODord Y X*
apply (*simp add*:*ODord-def*)
apply (*frule ordinal-numberTr1* [*of X*],
frule ordinal-numberTr1 [*of Y*], (*erule exE*)+, *rename-tac D E*)
apply (*erule conjE*)+
apply (*frule-tac* $D = D$ **and** $E = E$ **in** *Worder.Word-compare*, *assumption*+)
apply (*erule disjE*)+
apply (*erule bexE*,
cut-tac $D = D$ **and** $a = a$ **in** *Worder.segment-Worder*, *assumption*+,
frule-tac $S =$ *Iod D* (*segment D a*) **and** $T = E$ **in** *Worder-sym*,
assumption+, *blast*)

apply (*erule disjE*)
apply (*frule-tac* $D = D$ **and** $X = X$ **in** *ordinal-numberTr4*, *assumption*+,
frule-tac $D = E$ **and** $X = Y$ **in** *ordinal-numberTr4*, *assumption*+,
frule-tac $D = D$ **and** $E = E$ **in** *ordinal-number-eq*, *assumption*+)
apply *simp*

apply *blast*
done

lemma *ODnum-subTr*:⟦*Worder D*; $x =$ *ordinal-number D*; $y \in$*ODnums*; $y \sqsubset x$;
$Y \in y$⟧
⟹ ∃ *c*∈*carrier D*. *ord-equiv Y* (*Iod D* (*segment D c*))
apply *simp*
apply (*thin-tac* $x =$ *ordinal-number D*)
apply (*simp add*:*ODnums-def*, *erule exE*, *erule conjE*, *simp*,
thin-tac $y =$ *ordinal-number S*)
apply (*frule-tac* $D = S$ **and** $X = Y$ **in** *mem-ordinal-number-Worder*,
assumption+)

apply (*frule-tac* $D = Y$ **and** $X =$ *ordinal-number* S **in** *ordinal-numberTr4*,
simp add:*ODnods-def ODnums-def*, *blast*, *simp*)
apply *simp*
apply (*thin-tac* $Y \in$ *ordinal-number* Y,
thin-tac ordinal-number $S =$ *ordinal-number* Y)
apply (*simp add*:*ODordTr1*[*of* Y D])
done

lemma *ODnum-segmentTr*:⟦*Worder* D; $x =$ *ordinal-number* D; $y \in$*ODnums*; $y \sqsubset x$⟧ ⟹
$\exists\, c.\ c \in$*carrier* $D \wedge (\forall\, Y \in y.$ *ord-equiv* Y (*Iod* D (*segment* D c)))
apply (*frule ordinal-numberTr1-1*[*of* y], *erule exE*, *erule conjE*,
frule-tac $x = x$ **and** $y = y$ **and** $Y = Da$ **in** *ODnum-subTr*[*of* D],
assumption+, *erule bexE*)
apply (*rule ex-conjI*, *simp*+)

apply (*rule ballI*)
apply (*frule-tac* $D = D$ **and** $a = c$ **in** *Worder.segment-Worder*)
apply (*frule-tac* $X = Y$ **in** *mem-ordinal-number-Worder1*[*of* y], *assumption*)
apply (*subst ordinal-number-eq*, *assumption*+)
apply (*simp add*:*ordinal-number-eq*)
apply (*subst ordinal-numberTr5*[*THEN sym*, *of* y], *assumption*+)
apply (*frule-tac* $D = Da$ **in** *ordinal-numberTr5*[*of* y], *assumption*, *simp*)
done

lemma *ODnum-segmentTr1*:⟦*Worder* D; $x =$ *ordinal-number* D; $y \in$ *ODnums*; $y \sqsubset x$⟧
⟹ $\exists\, c.\ c \in$ *carrier* $D \wedge (y =$ *ordinal-number* (*Iod* D (*segment* D c)))
apply (*frule ODnum-segmentTr*[*of* D x y], *assumption*+, *erule exE*, *erule conjE*)
apply (*frule mem-ODnums-nonempty*[*of* y], *erule exE*,
frule-tac $x = xa$ **in** *bspec*, *assumption*,
thin-tac $\forall\, Y \in y.$ *ord-equiv* Y (*Iod* D (*segment* D c)))

apply (*frule-tac* $D = xa$ **in** *ordinal-numberTr5*[*of* y], *assumption*, *simp*,
frule-tac $a = c$ **in** *Worder.segment-Worder*[*of* D],
rotate-tac -2, *frule sym*, *thin-tac* $y =$ *ordinal-number* xa, *simp*,
frule-tac $X = xa$ **in** *mem-ordinal-number-Worder1*[*of* y], *assumption*+)
apply (*simp add*:*ordinal-number-eq*) **apply** *blast*
done

lemma *ODnods-less*:⟦$x \in$ *carrier ODnods*; $y \in$ *carrier ODnods*⟧ ⟹
$x \prec_{ODnods} y = x \sqsubset y$
apply (*simp add*:*ODnods-def ole-def oless-def ODrel-def ODord-le-def*)
apply (*rule iffI*)
apply (*erule conjE*, *erule disjE*, *simp*, *assumption*, *simp*)
apply (*simp add*:*ODord-def*, (*erule bexE*)+)
apply (*rule contrapos-pp*, *simp*+,
frule-tac $x = y$ **and** $S = ya$ **and** $T = xa$ **in** *ordinal-numberTr1-2*,
assumption+)

apply (*simp add*:*ODnums-def*, *erule exE*, *erule conjE*, *simp*,
frule-tac $D = S$ **and** $X = xa$ **in** *mem-ordinal-number-Worder*, *assumption*+,
frule-tac $D = S$ **and** $X = ya$ **in** *mem-ordinal-number-Worder*, *assumption*+,
cut-tac $D = ya$ **and** $a = c$ **in** *Worder.segment-Worder*, *assumption*+)

apply (*frule-tac* $S = ya$ **and** $T = xa$ **and** $U = Iod\ ya\ (segment\ ya\ c)$ **in**
Worder-trans, *assumption*+,
frule-tac $D = ya$ **and** $a = c$ **in** *Worder.nonequiv-segment*, *assumption*+)
apply *simp*
done

lemma *ODord-less-not-eq*:$[\![x \in carrier\ ODnods;\ y \in carrier\ ODnods;\ x \sqsubset y]\!] \Longrightarrow$
$x \neq y$
apply (*rule contrapos-pp*, *simp*+)
apply (*frule ordinal-numberTr1*[*of y*], *erule exE*, *erule conjE*,
simp add:*ODnods-def*)

apply (*frule-tac* $D = D$ **in** *ordinal-numberTr5*[*of y*], *assumption*+,
frule-tac $D = D$ **and** $x = y$ **and** $y = y$ **in** *ODnum-segmentTr1*, *assumption*+,
erule exE, *erule conjE*, *simp*,
frule-tac $a = c$ **and** $D = D$ **in** *Worder.segment-Worder*)
apply (*rotate-tac* -4, *frule sym*,
thin-tac ordinal-number $(Iod\ D\ (segment\ D\ c)) = ordinal\text{-}number\ D$,
simp add:*ordinal-number-eq*[*THEN sym*])
apply (*frule-tac* $D = D$ **and** $a = c$ **in** *Worder.nonequiv-segment*, *assumption*)

apply *simp*
done

lemma *not-ODord*:$[\![a \in ODnums;\ b \in ODnums;\ a \sqsubset b]\!] \Longrightarrow \neg\ (b \sqsubseteq a)$
apply (*rule contrapos-pp*, *simp*+)

apply (*simp add*:*ODord-le-def*)
apply (*cut-tac* $x = a$ **and** $y = b$ **in** *ODord-less-not-eq*,
simp add:*ODnods-def*, *simp add*:*ODnods-def*, *assumption*)
apply (*erule disjE*) **apply** *simp*

apply (*frule ordinal-numberTr1-1*[*of a*],
frule ordinal-numberTr1-1[*of b*], (*erule exE*)+, (*erule conjE*)+)
apply (*frule-tac* $D = D$ **in** *ordinal-numberTr5*[*of a*], *assumption*,
frule-tac $D = Da$ **in** *ordinal-numberTr5*[*of b*], *assumption*, *simp*)
apply (*frule-tac* $D = D$ **and** $E = Da$ **in** *ODordTr1*, *assumption*+,
frule-tac $D = Da$ **and** $E = D$ **in** *ODordTr1*, *assumption*+, *simp*)
apply (*frule-tac* $D = D$ **and** $E = Da$ **in** *Worder.Word-compareTr3*, *assumption*+)
apply (*erule bexE*)+
apply (*frule-tac* $D = D$ **and** $a = baa$ **in** *Worder.segment-Worder*,
frule-tac $S = Da$ **and** $T = Iod\ D\ (segment\ D\ baa)$ **in** *Worder-sym*,

```
      assumption+) apply blast
 apply assumption
done

lemma Order-ODnods:Order ODnods
apply (rule Order.intro)
 apply (simp add:ODnods-def ODrel-def)
 apply (rule subsetI) apply simp

 apply (simp add:ODnods-def ODrel-def, simp add:ODord-le-def)

 apply (simp add:ODnods-def ODrel-def, simp add:ODord-le-def)
 apply (erule disjE, assumption)
 apply (erule disjE, rule sym, assumption)
 apply (frule-tac a = a and b = b in not-ODord, assumption+)
 apply (simp add:ODord-le-def)

 apply (simp add:ODnods-def ODrel-def)
 apply (rule-tac X = a and Y = b and Z = c in ODord-le-trans, assumption+)
done

lemma Torder-ODnods:Torder ODnods
apply (rule Torder.intro)
apply (cut-tac Order-ODnods, assumption)


 apply (simp add:Torder-axioms-def)
  apply ((rule allI, rule impI)+)
apply (cut-tac Order-ODnods)
apply (subst Order.le-imp-less-or-eq[of ODnods], assumption+,
      subst Order.le-imp-less-or-eq[of ODnods], assumption+)
apply (simp add:ODnods-less,
      frule-tac X = a and Y = b in ordinal-number-ord, assumption+,
        blast)
done

definition
  ODNmap :: 'a Order ⇒ ('a Order) set ⇒ 'a where
  ODNmap D y = (SOME z. (z ∈ carrier D ∧
                    (∀ Y∈y. ord-equiv Y (Iod D (segment D z)))))

lemma ODNmap-mem:⟦Worder D; x = ordinal-number D; y ∈ ODnums; ODord
y x⟧ ⟹
     ODNmap D y ∈ carrier D ∧
          (∀ Y∈y. ord-equiv Y (Iod D (segment D (ODNmap D y))))
apply (frule ODnum-segmentTr [of D x y], assumption+)
apply (simp add:ODNmap-def)
apply (rule someI2-ex, assumption+)
done
```

lemma *ODNmapTr1*:⟦*Worder D*; *x* = *ordinal-number D*; *y* ∈ *ODnums*; *ODord y x*⟧ ⟹
y = *ordinal-number* (*Iod D* (*segment D* (*ODNmap D y*)))
apply (*frule ODNmap-mem* [*of D x y*], *assumption*+, *erule conjE*,
frule ODnum-segmentTr1 [*of D x y*], *assumption*+)
apply (*erule exE*, *erule conjE*,
cut-tac D = *D* **and** *a* = *c* **in** *Worder.segment-Worder*, *assumption*+,
frule-tac D = *Iod D* (*segment D c*) **and** *x* = *y* **in** *ordinal-numberTr2*,
assumption+,
frule-tac x = *Iod D* (*segment D c*) **in** *bspec*, *assumption*,
thin-tac ∀ *Y*∈*y*. *ord-equiv Y* (*Iod D* (*segment D* (*ODNmap D y*))),
cut-tac a = *ODNmap D y* **in** *Worder.segment-Worder*[*of D*],
assumption+)
apply (*frule-tac D* = *Iod D* (*segment D c*) **and** *E* = *Iod D* (*segment D* (*ODNmap D y*)) **in** *ordinal-number-eq*, *assumption*+) **apply** *simp*
done

lemma *ODNmap-self*:⟦*Worder D*; *c* ∈ *carrier D*;
a = *ordinal-number* (*Iod D* (*segment D c*))⟧ ⟹ *ODNmap D a* = *c*
apply (*simp add*:*ODNmap-def*)
apply (*rule someI2-ex*, *rule ex-conjI*, *simp*)
apply (*rule ballI*,
cut-tac Worder.segment-Worder[*of D c*],
rule-tac X = *Y* **in** *mem-ordinal-number-equiv*[*of Iod D* (*segment D c*)],
assumption+)

apply (*erule conjE*)
apply (*cut-tac Worder.segment-Worder*[*of D c*],
frule ordinal-inc-self[*of Iod D* (*segment D c*)],
frule-tac x = *Iod D* (*segment D c*) **in** *bspec*, *assumption*)
apply (*frule-tac b* = *x* **in** *Worder.segment-unique*[*of D c* -], *assumption*+,
rule sym, *assumption*, *assumption*)
done

lemma *ODord-ODNmap-less*:⟦*Worder D*; *c* ∈ *carrier D*;
a = *ordinal-number* (*Iod D* (*segment D c*)); *d* ∈ *carrier D*;
b = *ordinal-number* (*Iod D* (*segment D d*)); *a* ⊏ *b*⟧ ⟹
ODNmap D a ≺$_D$ (*ODNmap D b*)
apply (*frule ODNmap-self* [*of D c a*], *assumption*+,
frule ODNmap-self [*of D d b*], *assumption*+)
apply *simp*
apply (*subst ord-less-ODord*[*of D c d a b*], *assumption*+)
apply *simp*
done

lemma *ODNmap-mem1*: ⟦*Worder D*; *y* ∈ *segment ODnods* (*ordinal-number D*)⟧
⟹ *ODNmap D y* ∈ *carrier D*
apply (*frule-tac D* = *D* **and** *x* = *ordinal-number D* **and** *y* = *y* **in** *ODNmap-mem*,

```
      simp,
      frule ordinal-number-mem-carrier-ODnods[of D],
      simp add:ODnods-def segment-def)
 apply (simp add:segment-def,
       frule ordinal-number-mem-carrier-ODnods[of D], simp,
       erule conjE, simp add:ODnods-def oless-def ole-def ODrel-def)
 apply (simp add:ODord-le-def, blast, simp)
done

lemma ODnods-segment-inc-ODord:⟦Worder D;
      y ∈ segment ODnods (ordinal-number D)⟧ ⟹ ODord y (ordinal-number D)
 apply (simp add:segment-def,
       frule ordinal-number-mem-carrier-ODnods[of D], simp,
       erule conjE, simp add:ODnods-def oless-def ole-def ODrel-def)
 apply ((erule conjE)+, simp add:ODord-le-def)
done

lemma restict-ODNmap-func:⟦Worder D; x = ordinal-number D⟧ ⟹
     restrict (ODNmap D) (segment ODnods (ordinal-number D))
           ∈ segment ODnods (ordinal-number D) → carrier D
apply (cut-tac Order-ODnods,
       frule Order.Iod-carr-segment[of ODnods ordinal-number D],
       frule Order.segment-sub[of ODnods ordinal-number D])
apply (rule Pi-I, simp,
       (frule Order.Iod-carr-segment[of ODnods ordinal-number D],
        simp,
        thin-tac carrier (Iod ODnods (segment ODnods (ordinal-number D))) =
                        segment ODnods (ordinal-number D) ),
       rule ODNmap-mem1[of D], assumption+)
done

lemma ODNmap-ord-isom:⟦Worder D; x = ordinal-number D⟧ ⟹
           ord-isom (Iod ODnods (segment ODnods x)) D
     (λx∈(carrier (Iod ODnods (segment ODnods x))). (ODNmap D x))
apply (cut-tac Order-ODnods,
       frule Order.Iod-carr-segment[of ODnods ordinal-number D],
       frule ordinal-number-mem-carrier-ODnods[of D],
       frule Order.segment-sub[of ODnods ordinal-number D])

apply (simp add:ord-isom-def)
 apply (rule conjI)
 apply (simp add:ord-inj-def)
 apply (simp add:restict-ODNmap-func[of D x])
 apply (rule conjI)
 apply (subst inj-on-def)
 apply ((rule ballI)+, rule impI)
  apply (thin-tac carrier (Iod ODnods (segment ODnods (ordinal-number D))) =
       segment ODnods (ordinal-number D))
  apply (frule-tac y = xa in ODnods-segment-inc-ODord[of D], assumption+,
```

```
        frule-tac y = y in ODnods-segment-inc-ODord[of D], assumption+)
  apply (frule-tac y = y in ODNmapTr1[of D x], assumption+)
  apply (frule-tac c = y in subsetD[of segment ODnods (ordinal-number D)
                carrier ODnods], assumption+, simp add:carr-ODnods,
                simp)
  apply (frule-tac y = xa in ODNmapTr1[of D x], assumption+)
  apply (frule-tac c = xa in subsetD[of segment ODnods (ordinal-number D)
                carrier ODnods], assumption+, simp add:carr-ODnods,
                simp, simp)
  apply (rule ballI)+
   apply (thin-tac carrier (Iod ODnods (segment ODnods (ordinal-number D))) =
       segment ODnods (ordinal-number D))
  apply (frule-tac y = a in ODnods-segment-inc-ODord[of D], assumption+,
        frule-tac y = b in ODnods-segment-inc-ODord[of D], assumption+)
  apply (frule-tac y = a in ODNmapTr1[of D x], assumption+)
  apply (frule-tac c = a in subsetD[of segment ODnods (ordinal-number D)
                carrier ODnods], assumption+, simp add:carr-ODnods,
                simp)
  apply (frule-tac y = b in ODNmapTr1[of D x], assumption+)
  apply (frule-tac c = b in subsetD[of segment ODnods (ordinal-number D)
                carrier ODnods], assumption+, simp add:carr-ODnods,
                simp)
   apply (frule-tac c = ODNmap D a and d = ODNmap D b and a = a and b
= b in
            ord-less-ODord[of D],
        frule-tac  x = x and y = a in ODNmap-mem[of D], assumption,
        frule-tac c = a in subsetD[of segment ODnods (ordinal-number D)
                carrier ODnods], assumption+, simp add:carr-ODnods,
        simp, simp)
  apply (frule-tac  x = x and y = b in ODNmap-mem[of D], assumption,
        frule-tac c = b in subsetD[of segment ODnods (ordinal-number D)
                carrier ODnods], assumption+, simp add:carr-ODnods,
        simp, simp) apply simp+
   apply (frule-tac c = a in subsetD[of segment ODnods (ordinal-number D)
                carrier ODnods], assumption+,
         frule-tac c = b in subsetD[of segment ODnods (ordinal-number D)
                carrier ODnods], assumption+)
  apply (simp add:Order.Iod-less[of ODnods])
   apply (simp add:ODnods-less)

 apply (rule surj-to-test)
    apply (simp add:restict-ODNmap-func)
    apply (rule ballI,
          cut-tac D = D and a = b in Worder.segment-Worder, assumption+,
          frule-tac D = Iod D (segment D b) in
                              ordinal-number-mem-carrier-ODnods,
          frule-tac c = b and a = ordinal-number (Iod D (segment D b))
           in ODNmap-self, assumption, simp,
          frule-tac d = b in ODord[of D], assumption,
```

```
        frule-tac a = ordinal-number (Iod D (segment D b)) in
         Order.segment-inc[of ODnods - ordinal-number D], assumption+,
        cut-tac x = ordinal-number (Iod D (segment D b)) in
                  ODnods-less[of - ordinal-number D], assumption+,
        simp)
   apply (frule-tac c = b and a = ordinal-number (Iod D (segment D b)) in
         ODNmap-self[of D], assumption)
    apply simp
   apply  blast
done

lemma ODnum-equiv-segment:⟦Worder D; x = ordinal-number D⟧ ⟹
       ord-equiv (Iod ODnods (segment ODnods x)) D
apply (simp add: ord-equiv-def)
apply (frule ODNmap-ord-isom[of D x], assumption+, blast)
done

lemma ODnods-sub-carrier:S ⊆ ODnums ⟹ carrier (Iod ODnods S) = S
by (simp add:Iod-def)

lemma ODnum-sub-well-ordered:S ⊆ ODnums ⟹ Worder (Iod ODnods S)
apply (cut-tac Torder-ODnods,
      cut-tac Order-ODnods)
 apply (frule Torder.Iod-Torder[of ODnods S],
       simp add:carr-ODnods)
apply intro-locales
 apply (simp add:Torder-def, simp add:Torder-def)

 apply (simp add:Worder-axioms-def,
       rule allI, rule impI, erule conjE)


 apply (frule Order.Iod-carrier[of ODnods S],
       simp add:carr-ODnods, simp)
 apply (frule-tac A = X in nonempty-ex, erule exE)
 apply (frule-tac c = x and A = X and B = S in subsetD, assumption+,
       frule-tac c = x and A = S and B = ODnums in subsetD, assumption+)
 apply (frule-tac X = x in ordinal-numberTr1-1, erule exE, erule conjE)
 apply (frule-tac D = D and x = x in ODnum-equiv-segment)
 apply (rule ordinal-numberTr4, assumption+, simp add:carr-ODnods, assump-
tion)
 apply (frule-tac D = D and T = Iod ODnods (segment ODnods x) in
       Worder.equiv-Worder1,
       rule Order.Iod-Order[of ODnods], assumption,
       rule Order.segment-sub, assumption+)
 apply (frule-tac D = Iod ODnods (segment ODnods x) in Worder.ex-minimum)
 apply (case-tac (segment ODnods x) ∩ X ≠ {})
 apply (frule-tac a = segment ODnods x ∩ X in forall-spec)
  apply (simp add:Order.Iod-carr-segment)
```

apply (*thin-tac* $\forall X.\ X \subseteq carrier\ (Iod\ ODnods\ (segment\ ODnods\ x)) \wedge X \neq \{\} \longrightarrow (\exists xa.\ minimum\text{-}elem\ (Iod\ ODnods\ (segment\ ODnods\ x))\ X\ xa)$)
apply (*erule exE*)

apply (*simp add*:*carr-ODnods*[*THEN sym*])
apply (*frule-tac* $A = X$ **and** $B = S$ **and** $C = carrier\ ODnods$ **in** *subset-trans*, *assumption*+)
apply (*frule-tac* $d = x$ **and** $m = xa$ **and** $X = X$ **in**
Torder.segment-minimum-minimum[*of ODnods*], *assumption*+,
simp add:*Int-commute*,
simp add:*Order.minimum-elem-sub*[*of ODnods S*], *blast*)
apply (*simp*,
thin-tac $\forall X.\ X \subseteq carrier\ (Iod\ ODnods\ (segment\ ODnods\ x)) \wedge X \neq \{\} \longrightarrow (\exists xa.\ minimum\text{-}elem\ (Iod\ ODnods\ (segment\ ODnods\ x))\ X\ xa)$)
apply (*frule-tac* $A = segment\ ODnods\ x$ **and** $B = X$ **in** *no-meet2*)
apply (*simp add*:*carr-ODnods*[*THEN sym*])
apply (*frule-tac* $A = X$ **and** $B = S$ **and** $C = carrier\ ODnods$ **in** *subset-trans*, *assumption*+)
apply (*frule-tac* $A = X$ **and** $B = S$ **and** $C = carrier\ ODnods$ **in** *subset-trans*, *assumption*+)
apply (*simp add*:*Order.segment-inc*[*THEN sym*, *of ODnods*])
apply (*rule contrapos-pp*, *simp*+)
apply (*frule-tac* $x = x$ **in** *spec*,
thin-tac $\forall x.\ \neg\ minimum\text{-}elem\ (Iod\ ODnods\ S)\ X\ x$)
apply (*simp add*:*minimum-elem-def*, *erule bexE*)
apply (*frule-tac* $x = xa$ **in** *bspec*, *assumption*,
thin-tac $\forall a{\in}X.\ a \notin segment\ ODnods\ x$,
frule-tac $c = xa$ **and** $A = X$ **and** $B = carrier\ ODnods$ **in** *subsetD*, *assumption*+)
apply (*simp add*:*Order.segment-inc*[*of ODnods*, *THEN sym*],
frule-tac $c = xa$ **and** $A = X$ **and** $B = S$ **in** *subsetD*, *assumption*+)
apply (*simp add*:*Order.Iod-le*[*of ODnods S*])
apply (*simp add*:*Torder.not-le-less*)
done

2.2 Pre elements

definition
ExPre :: - ⇒ *'a* ⇒ *bool* **where**
$ExPre\ D\ a \longleftrightarrow (\exists x.\ x \in carrier\ D \wedge x \prec_D a \wedge \neg\ (\exists y{\in}carrier\ D.\ x \prec_D y \wedge y \prec_D a))$

definition
Pre :: [- , *'a*] ⇒ *'a* **where**
$Pre\ D\ a = (SOME\ x.\ x \in carrier\ D \wedge x \prec_D a \wedge \neg\ (\exists y{\in}carrier\ D.\ x \prec_D y \wedge y \prec_D a))$

lemma (**in** *Order*) *Pre-mem*:⟦*a* ∈ *carrier D*; *ExPre D a*⟧ ⟹
Pre D a ∈ *carrier D*
apply (*simp add*:*ExPre-def*)
apply (*subst Pre-def*, *rule someI2-ex*, *blast*, *simp*)
done

lemma (**in** *Order*) *Not-ExPre*:*a* ∈ *carrier D* ⟹ ¬ *ExPre* (*Iod D* {*a*}) *a*
apply (*simp add*:*ExPre-def*,
rule allI, *rule impI*, *rule impI*,
frule singleton-sub[*of a carrier D*])
apply (*simp add*:*Iod-less Iod-carrier*)
done

lemma (**in** *Worder*) *UniquePre*:⟦*a* ∈ *carrier D*; *ExPre D a*;
a1 ∈ *carrier D* ∧ *a1* ≺ *a* ∧ ¬ (∃ *y*∈*carrier D*. (*a1* ≺ *y* ∧ *y* ≺ *a*)) ⟧ ⟹
Pre D a = *a1*
apply (*simp add*:*ExPre-def*)
apply (*subst Pre-def*)
apply (*rule someI2-ex*, *blast*)
apply (*erule conjE*)+
apply (*thin-tac* ∃ *x*. *x* ∈ *carrier D* ∧ *x* ≺ *a* ∧
(∀ *y*∈*carrier D*. *x* ≺ *y* ⟶ ¬ *y* ≺ *a*),
rename-tac z)
apply (*rule contrapos-pp*, *simp*+)
apply (*frule-tac a* = *z* **and** *b* = *a1* **in** *less-linear*, *assumption*+,
simp)
apply (*erule disjE*)
apply (*rotate-tac* −*4*,
frule-tac x = *a1* **in** *bspec*, *assumption*+,
thin-tac ∀ *y*∈*carrier D*. *z* ≺ *y* ⟶ ¬ *y* ≺ *a*,
thin-tac ∀ *y*∈*carrier D*. *a1* ≺ *y* ⟶ ¬ *y* ≺ *a*, *simp*)

apply (*frule-tac x* = *z* **in** *bspec*, *assumption*+, *simp*)
done

lemma (**in** *Order*) *Pre-element*:⟦*a* ∈ *carrier D*; *ExPre D a*⟧ ⟹
Pre D a ∈ *carrier D* ∧ (*Pre D a*) ≺ *a* ∧
¬ (∃ *y*∈*carrier D*. ((*Pre D a*) ≺ *y* ∧ *y* ≺ *a*))
apply (*simp add*:*ExPre-def*)
apply (*subst Pre-def*)+
apply (*rule someI2-ex*)
apply *simp*+
done

lemma (**in** *Order*) *Pre-in-segment*:⟦*a* ∈ *carrier D*; *ExPre D a*⟧ ⟹
Pre D a ∈ *segment D a*
by (*frule Pre-element*[*of a*], *assumption*+, (*erule conjE*)+,
simp add:*segment-inc*)

lemma (**in** *Worder*) *segment-forall*:⟦$a \in$ *segment D b*; $b \in$ *carrier D*;
$x \in$ *segment D b*; $x \prec a$; $\forall\, y{\in}$*segment D b*. $x \prec y \longrightarrow \neg\ y \prec a$⟧ $\Longrightarrow$
$\forall\, y{\in}$*carrier D*. $x \prec y \longrightarrow \neg\ y \prec a$
apply (*cut-tac segment-sub*[*of b*])
apply (*rule ballI*, *rule impI*)
apply (*case-tac* $y \in$ *segment D b*,
frule-tac $x = y$ **in** *bspec*, *assumption*+, *simp*)

apply (*thin-tac* $\forall\, y{\in}$*segment D b*. $x \prec y \longrightarrow \neg\ y \prec a$,
frule subsetD[*of segment D b carrier D a*], *assumption*+,
frule-tac $c = x$ **in** *subsetD*[*of segment D b carrier D*], *assumption*+,
thin-tac segment D b $\subseteq$ *carrier D*,
thin-tac $x \in$ *segment D b*)
apply (*simp add*:*segment-inc*[*THEN sym*, *of* - *b*]) **apply** (
simp add:*not-less-le*)
apply (*frule-tac* $c = y$ **in** *less-le-trans*[*of a b*], *assumption*+)
apply (*simp add*:*less-imp-le*)
done

lemma (**in** *Worder*) *segment-Expre*:$a \in$ *segment D b* $\Longrightarrow$
ExPre (*Iod D* (*segment D b*)) a = *ExPre D a*
apply (*case-tac* $b \notin$ *carrier D*)
apply (*simp add*:*segment-def Iod-self*[*THEN sym*])

apply *simp*
apply (*cut-tac segment-sub*[*of b*],
frule subsetD[*of segment D b carrier D a*], *assumption*+)
apply (*rule iffI*)
apply (*simp add*:*ExPre-def*, *erule exE*, (*erule conjE*)+)
apply (*simp add*:*Iod-carrier Iod-less*)
apply (*frule-tac* $x = x$ **in** *segment-forall*[*of a b*], *assumption*+)
apply *blast*

apply (*simp add*:*ExPre-def*, *erule exE*, (*erule conjE*)+)
apply (*frule-tac* $a = a$ **in** *segment-inc*[*of* - *b*], *assumption*, *simp*)
apply (*rule contrapos-pp*, *simp*+)
apply (*frule-tac* $x = x$ **in** *spec*,
thin-tac $\forall\, x.\ x \prec_{Iod\ D\ (segment\ D\ b)}\ a \longrightarrow$
$x \in$ *carrier* (*Iod D* (*segment D b*)) $\longrightarrow$
($\exists\, y{\in}$*carrier* (*Iod D* (*segment D b*))).
$x \prec_{Iod\ D\ (segment\ D\ b)}\ y \wedge y \prec_{Iod\ D\ (segment\ D\ b)}\ a$),
frule-tac $a = x$ **in** *less-trans*[*of* - *a b*], *assumption*+,
frule-tac $a = x$ **in** *segment-inc*[*of* - *b*], *assumption*+, *simp*)
apply (*simp add*:*Iod-carrier Iod-less*)
apply (*erule bexE*, *erule conjE*,
frule-tac $c = y$ **in** *subsetD*[*of segment D b carrier D*], *assumption*+)
apply (*frule-tac* $x = y$ **in** *bspec*, *assumption*,
thin-tac $\forall\, y{\in}$*carrier D*. $x \prec y \longrightarrow \neg\ y \prec a$, *simp*)

done

lemma (**in** *Worder*) *Pre-segment*:⟦*a* ∈ *segment D b*;
ExPre (*Iod D* (*segment D b*)) *a*⟧ ⟹
ExPre D a ∧ *Pre D a* = *Pre* (*Iod D* (*segment D b*)) *a*
apply (*frule segment-Expre*[*of a b*], *simp*)
apply (*case-tac b* ∉ *carrier D*)
apply (*simp add*:*segment-def*, *simp add*:*Iod-self*[*THEN sym*])
apply *simp*

apply (*cut-tac segment-sub*[*of b*],
frule subsetD[*of segment D b carrier D a*], *assumption*+)
apply (*frule-tac a* = *a* **and** *?a1.0* = *Pre* (*Iod D* (*segment D b*)) *a* **in**
UniquePre, *assumption*+)
apply *simp*
apply (*cut-tac segment-Worder*[*of b*])
apply (*frule-tac D* = *Iod D* (*segment D b*) **in** *Worder.Order*)
apply (*frule Order.Pre-element*[*of Iod D* (*segment D b*) *a*],
simp add:*Iod-carrier*, *assumption*, *erule conjE*,
simp add:*Iod-carrier*, *simp add*:*subsetD*, *simp add*:*Iod-less*)
apply (*erule conjE*, *rule ballI*)
apply (*case-tac y* ∈ *segment D b*,
frule-tac x = *y* **in** *bspec*, *assumption*,
thin-tac ∀ *y*∈*segment D b*. *Pre* (*Iod D* (*segment D b*)) *a* ≺ *y*
⟶ ¬ *y* ≺ *a*,
simp)
apply (*rule impI*)
apply (*frule subsetD*[*of segment D b carrier D a*], *assumption*+)
apply (*frule-tac a* = *y* **in** *segment-inc*[*of - b*], *assumption*+, *simp*,
frule segment-inc[*of a b*], *assumption*+, *simp*)
apply (*simp add*:*not-less-le*)
apply (*frule-tac c* = *y* **in** *less-le-trans*[*of a b*], *assumption*+,
simp add:*less-imp-le*)
apply *assumption*
done

lemma (**in** *Worder*) *Pre2segment*:⟦*a* ∈ *carrier D*; *b* ∈ *carrier D*; *b* ≺ *a*;
ExPre D b⟧ ⟹ *ExPre* (*Iod D* (*segment D a*)) *b*
apply (*frule segment-inc* [*of b a*], *assumption*+, *simp*)
apply (*simp add*:*segment-Expre*[*of b a*])
done

lemma (**in** *Worder*) *ord-isom-Pre1*:⟦*Worder E*; *a* ∈ *carrier D*; *ExPre D a*;
ord-isom D E f⟧ ⟹ *ExPre E* (*f a*)
apply (*simp add*:*ExPre-def*)
apply (*erule exE*,
frule Worder.Order[*of E*],
erule conjE,
frule-tac a = *x* **in** *ord-isom-mem*[*of E f*], *assumption*+,

frule-tac $a = a$ **in** *ord-isom-mem*[*of E f*], *assumption*+,
erule conjE)
apply (*frule-tac* $a = x$ **in** *ord-isom-less*[*of E f - a*], *assumption*+, *simp*)
apply (*frule ord-isom-less-forall*[*of E f*], *assumption*+)
apply (*frule-tac* $x = x$ **and** $a = a$ **in** *ord-isom-convert*[*of E f*],
assumption+, *simp*) **apply** *blast*
done

lemma (**in** *Worder*) *ord-isom-Pre11*:⟦*Worder E*; $a \in$ *carrier D*; *ord-isom D E f*⟧
$\Longrightarrow$ *ExPre D a* = *ExPre E* (*f a*)
apply (*rule iffI*)
apply (*simp add*:*ord-isom-Pre1*)
apply (*frule Worder.Order*[*of E*],
frule ord-isom-sym[*of E f*], *assumption*+)
apply (*cut-tac Worder*)
apply (*frule Worder-ord-isom-mem*[*of D E f a*], *assumption*+,
frule Worder.ord-isom-Pre1[*of E D f a*
invfun (*carrier D*) (*carrier E*) *f*], *assumption*+)
apply (*frule ord-isom-func*[*of E f*], *assumption*+)
apply (*simp add*:*ord-isom-def*[*of D E f*] *ord-inj-def*, (*erule conjE*)+,
thin-tac $\forall\, a{\in}$*carrier D*. $\forall\, b{\in}$*carrier D*. $a \prec b = f\ a \prec_E f\ b$)
apply (*simp add*:*invfun-l*)
done

lemma (**in** *Worder*) *ord-isom-Pre2*:⟦*Worder E*; $a \in$ *carrier D*; *ExPre D a*;
ord-isom D E f⟧ $\Longrightarrow$ *f* (*Pre D a*) = *Pre E* (*f a*)
apply (*frule-tac* $E = E$ **and** $a = a$ **and** $f = f$ **in** *ord-isom-Pre1*, *assumption*+,
frule-tac $a = a$ **in** *Pre-element*, *assumption*+, (*erule conjE*)+)
apply (*frule Worder.Order*[*of E*],
frule ord-isom-mem[*of E f a*], *assumption*+,
frule ord-isom-mem[*of E f Pre D a*], *assumption*+,
simp add:*ord-isom-less*[*of E f Pre D a a*])
apply (*simp add*:*ord-isom-convert*[*of E f Pre D a a*])
apply (*rule Worder.UniquePre*[*THEN sym*, *of E f a f* (*Pre D a*)],
assumption+, *simp*)
done

2.3 Transfinite induction

lemma (**in** *Worder*) *transfinite-induction*:⟦*minimum-elem D* (*carrier D*) *s0*; *P s0*;
$\forall\, t{\in}$*carrier D*. ((∀ *u*∈ *segment D t*. *P u*) ⟶ *P t*)⟧ $\Longrightarrow$ $\forall\, x{\in}$*carrier D*. *P x*
apply (*rule contrapos-pp*, *simp*+)
apply (*frule bex-nonempty-set*[*of carrier D*],
frule nonempty-set-sub[*of carrier D*])
apply (*cut-tac ex-minimum*)
apply (*frule-tac* $a = \{x \in$ *carrier D*. $\neg$ *P x*$\}$ **in** *forall-spec*,
simp,
thin-tac $\forall\, X.\ X \subseteq$ *carrier D* $\wedge\ X \neq \{\} \longrightarrow (\exists\, x.$ *minimum-elem D X x*))
apply (*thin-tac* $\exists\, x{\in}$*carrier D*. $\neg$ *P x*)

apply (*erule exE*)
apply (*frule-tac* $d = x$ **in** *less-minimum*)
apply (*frule-tac* $X = \{x \in carrier\ D.\ \neg\ P\ x\}$ **and** $a = x$ **in** *minimum-elem-mem*, *assumption*+)
apply (*frule-tac* $c = x$ **and** $A = \{x \in carrier\ D.\ \neg\ P\ x\}$ **and** $B = carrier\ D$ **in** *subsetD*, *assumption*+)
apply (*frule-tac* $x = x$ **in** *bspec*, *assumption*+, *thin-tac* $\forall\ t{\in}carrier\ D.\ (\forall\ u{\in}segment\ D\ t.\ P\ u) \longrightarrow P\ t$)
apply (*simp add*:*minimum-elem-def*, (*erule conjE*)+)
apply (*erule bexE*, *simp add*:*segment-def*)
done

2.4 *Ordered-set2*. Lemmas to prove Zorn's lemma.

definition
adjunct-ord ::[- , 'a] ⇒ - **where**
adjunct-ord D a = *D* (|*carrier* := *carrier D* ∪ {*a*}, *rel* := {(*x*,*y*). (*x*, *y*) ∈ *rel D* ∨ (*x* ∈ (*carrier D* ∪ {*a*}) ∧ *y* = *a*)}|)

lemma (**in** *Order*) *carrier-adjunct-ord*:
carrier (*adjunct-ord D a*) = *carrier D* ∪ {*a*}
by (*simp add*:*adjunct-ord-def*)

lemma (**in** *Order*) *Order-adjunct-ord*:*a* ∉ *carrier D* ⟹ *Order* (*adjunct-ord D a*)
apply (*cut-tac closed*)
apply (*rule Order.intro*)
apply (*rule subsetI*)
apply (*unfold split-paired-all*)
apply *simp*
apply (*simp add*:*adjunct-ord-def insert-absorb*)
apply *blast*

apply (*simp add*:*carrier-adjunct-ord*)
apply (*erule disjE*)
apply (*simp add*:*adjunct-ord-def*)
apply (*simp add*:*adjunct-ord-def*)
apply (*simp add*:*refl*)

apply (*simp add*:*adjunct-ord-def*)
apply (*erule disjE*)+
apply *simp*+
apply (*erule disjE*)
apply (*frule-tac* $c = (a, b)$ **in** *subsetD*[*of rel D carrier D* × *carrier D*], *assumption*+)
apply *blast*
apply *simp*

apply (*erule disjE*)+
apply (*cut-tac closed*, *simp*+)
apply (*frule-tac c* = (*a*, *aa*) **in** *subsetD*[*of rel D*
carrier D × *carrier D*], *assumption*+, *simp*)
apply (*frule-tac c* = (*aa*, *b*) **in** *subsetD*[*of rel D*
carrier D × *carrier D*], *assumption*+, *simp*)

apply (*erule disjE*)
apply (*frule-tac c* = (*b*, *aa*) **in** *subsetD*[*of rel D*
carrier D × *carrier D*], *assumption*+)
apply *simp*+

apply (*erule disjE*)+
apply (*rule antisym*, *assumption*+)

apply (*frule-tac c* = (*aa*, *b*) **in** *subsetD*[*of rel D*
carrier D × *carrier D*], *assumption*+)
apply *simp*

apply (*erule disjE*)
apply (*frule-tac c* = (*b*, *aa*) **in** *subsetD*[*of rel D*
carrier D × *carrier D*], *assumption*+,
simp, *simp*)

apply (*simp add*:*adjunct-ord-def*)
apply (*erule disjE*)+
apply *blast*
apply *blast*
apply *blast*

apply (*erule disjE*)
apply *simp*
apply (*erule disjE*)
apply (*frule-tac c* = (*b*, *c*) **in** *subsetD*[*of rel D*
carrier D × *carrier D*], *assumption*+)
apply *simp* **apply** *blast*
apply (*erule disjE*, *blast*)
apply (*erule disjE*)
apply *simp*
apply (*erule disjE*)
apply (*frule-tac c* = (*a*, *b*) **in** *subsetD*[*of rel D*
carrier D × *carrier D*], *assumption*+, *simp*)
apply (*frule-tac c* = (*a*, *b*) **in** *subsetD*[*of rel D*
carrier D × *carrier D*], *assumption*+, *simp*)
apply (*erule disjE*, *simp*) **apply** *blast*
apply (*erule disjE*) **apply** *simp*
apply (*erule disjE*) **apply** *blast*
apply (*erule disjE*)
apply (*frule-tac c* = (*a*, *c*) **in** *subsetD*[*of rel D*

carrier D × *carrier D*], *assumption*+, *simp*)
apply *blast*
apply (*erule disjE*)+
apply *blast* **apply** *blast* **apply** *blast*
apply (*erule disjE*)
apply (*erule disjE*)
apply (*frule-tac a* = *aa* **and** *b* = *b* **and** *c* = *c* **in** *trans*, *assumption*+)
apply *simp*
apply *blast*
apply (*erule disjE*) **apply** *simp* **apply** *blast*
done

lemma (**in** *Order*) *adjunct-ord-large-a*:⟦*Order D*; *a* ∉ *carrier D*⟧ ⟹
∀ *x*∈*carrier D*. *x* ≺$_{adjunct\text{-}ord\ D\ a}$ *a*
apply (*rule ballI*)
apply (*subst oless-def*)
apply (*rule conjI*)
apply (*simp add*:*ole-def adjunct-ord-def*)
apply (*rule contrapos-pp*, *simp*+)
done

lemma *carr-Ssegment-adjunct-ord*:⟦*Order D*; *a* ∉ *carrier D*⟧ ⟹
carrier D = (*Ssegment* (*adjunct-ord D a*) *a*)
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*simp add*:*Ssegment-def Order.carrier-adjunct-ord*)
apply (*simp add*:*Order.adjunct-ord-large-a*)

apply (*rule subsetI*)
apply (*simp add*:*Ssegment-def Order.carrier-adjunct-ord*)
apply (*erule conjE*)
apply (*erule disjE*, *simp add*:*oless-def*)
apply *assumption*
done

lemma (**in** *Order*) *adjunct-ord-selfD*:*a* ∉ *carrier D* ⟹
D = *Iod* (*adjunct-ord D a*) (*carrier D*)
apply (*simp add*:*Iod-def*)
apply (*simp add*:*adjunct-ord-def*)
apply (*subgoal-tac rel D* = {(*aa*, *b*).
((*aa*, *b*) ∈ *rel D* ∨ (*aa* = *a* ∨ *aa* ∈ *carrier D*) ∧ *b* = *a*) ∧
aa ∈ *carrier D* ∧ *b* ∈ *carrier D*})
apply *simp*
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*cut-tac closed*,
frule-tac c = *x* **in** *subsetD*[*of rel D carrier D* × *carrier D*],
assumption+)
apply *auto*

done

lemma *Ssegment-adjunct-ord*:⟦*Order D*; *a* ∉ *carrier D*⟧ ⟹
D = *SIod* (*adjunct-ord D a*) (*Ssegment* (*adjunct-ord D a*) *a*)
apply (*simp add*: *carr-Ssegment-adjunct-ord*[*THEN sym*, *of D a*])
apply (*frule Order.Order-adjunct-ord*[*of D a*], *assumption*+)
apply (*cut-tac Order.carrier-adjunct-ord*[*THEN sym*, *of D a*])
apply (*cut-tac Un-upper1*[*of carrier D* {*a*}], *simp*)
apply (*subst SIod-self-le*[*THEN sym*, *of adjunct-ord D a D*],
assumption+)
apply (*rule ballI*)+
apply (*frule-tac c* = *aa* **in** *subsetD*[*of carrier D carrier* (*adjunct-ord D a*)],
assumption,
frule-tac c = *b* **in** *subsetD*[*of carrier D carrier* (*adjunct-ord D a*)],
assumption)
apply (*simp add*:*Order.le-rel*[*of adjunct-ord D a*])
apply (*subst adjunct-ord-def*, *simp*)
apply (*case-tac b* = *a*, *simp*) **apply** *simp*
apply (*simp add*:*Order.le-rel*[*of D*])
apply *simp*+
done

lemma (**in** *Order*) *Torder-adjunction*:⟦*X* ⊆ *carrier D*; *a* ∈ *carrier D*;
∀ *x*∈*X*. *x* ≼ *a*; *Torder* (*Iod D X*)⟧ ⟹ *Torder* (*Iod D* (*X* ∪ {*a*}))
apply (*frule insert-sub*[*of X carrier D a*], *assumption*)
apply (*subst Torder-def*)
apply (*frule singleton-sub*[*of a carrier D*],
frule Un-least[*of X carrier D* {*a*}], *assumption*)

apply (*rule conjI*)
apply (*simp add*:*Iod-Order*[*of insert a X*])

apply (*subst Torder-axioms-def*)
apply ((*rule allI*)+, (*rule impI*)+)
apply (*simp only*:*Iod-carrier*, *simp add*:*Iod-le*)
apply (*erule disjE*, *simp*) **apply** (*erule disjE*, *simp*)
apply (*simp add*:*le-refl*) **apply** *blast*
apply (*erule disjE*, *simp*)
apply (*simp add*:*Torder-def*, *simp add*:*Torder-axioms-def*)
apply (*simp add*:*Iod-carrier*, *erule conjE*)
apply (*frule-tac a* = *aa* **in** *forall-spec*, *assumption*,
thin-tac ∀ *a*. *a* ∈ *X* ⟶ (∀ *b*. *b* ∈ *X* ⟶ *a* ≼$_{Iod\ D\ X}$ *b* ∨ *b* ≼$_{Iod\ D\ X}$ *a*),
frule-tac a = *b* **in** *forall-spec*, *assumption*,
thin-tac ∀ *b*. *b* ∈ *X* ⟶ *aa* ≼$_{Iod\ D\ X}$ *b* ∨ *b* ≼$_{Iod\ D\ X}$ *aa*,
simp add:*Iod-le*)
done

lemma *Torder-Sadjunction*:⟦*Order D*; *X* ⊆ *carrier D*; *a* ∈ *carrier D*;
∀ *x*∈*X*. *x* ≼$_D$ *a*; *Torder* (*SIod D X*)⟧ ⟹ *Torder* (*SIod D* (*X* ∪ {*a*}))

apply (*frule insert-sub*[*of X carrier D a*], *assumption*)
apply (*subst Torder-def*)
apply (*frule singleton-sub*[*of a carrier D*],
frule Un-least[*of X carrier D* {*a*}], *assumption*)

apply (*rule conjI*)
apply (*simp add*:*SIod-Order*[*of - insert a X*])

apply (*subst Torder-axioms-def*)
apply ((*rule allI*)+, (*rule impI*)+)
apply (*simp only*:*SIod-carrier*, *simp add*:*SIod-le*)
apply (*erule disjE*, *simp*) **apply** (*erule disjE*, *simp*)
apply (*simp add*:*Order.le-refl*) **apply** *blast*
apply (*erule disjE*, *simp*)
apply (*simp add*:*Torder-def*, *simp add*:*Torder-axioms-def*)
apply (*simp add*:*SIod-carrier*, *erule conjE*)
apply (*frule-tac a = aa* **in** *forall-spec*, *assumption*,
thin-tac $\forall\, a.\ a \in X \longrightarrow (\forall\, b.\ b \in X \longrightarrow a \preceq_{SIod\ D\ X} b \vee b \preceq_{SIod\ D\ X} a)$,
frule-tac a = b **in** *forall-spec*, *assumption*,
thin-tac $\forall\, b.\ b \in X \longrightarrow aa \preceq_{SIod\ D\ X} b \vee b \preceq_{SIod\ D\ X} aa$,
simp add:*SIod-le*)
done

lemma (**in** *Torder*) *Torder-adjunct-ord*:$a \notin$ *carrier D* $\Longrightarrow$
Torder (*adjunct-ord D a*)
apply (*frule Order-adjunct-ord*[*of a*],
cut-tac carrier-adjunct-ord[*THEN sym*, *of a*],
cut-tac Un-upper1[*of carrier D* {*a*}], *simp*,
frule Order.Torder-adjunction[*of adjunct-ord D a carrier D a*],
assumption+)
apply (*simp add*:*carrier-adjunct-ord*)
apply (*cut-tac adjunct-ord-large-a*[*of a*],
rule ballI,
frule-tac x = x **in** *bspec*, *assumption*,
thin-tac $\forall\, x{\in}$*carrier D*. $x \prec_{adjunct\text{-}ord\ D\ a} a$,
cut-tac insertI1[*of a carrier D*], *simp*,
frule-tac c = x **in** *subsetD*[*of carrier D carrier* (*adjunct-ord D a*)],
assumption+,
simp add:*Order.less-imp-le*, *rule Order*,
assumption+)
apply (*simp add*:*adjunct-ord-selfD*[*THEN sym*])
prefer *2*
apply (*simp add*:*Order.Iod-self*[*THEN sym*, *of adjunct-ord D a*])
apply (*unfold-locales*)
done

lemma (**in** *Order*) *well-ord-adjunction*:$[\![X \subseteq$ *carrier D*; $a \in$ *carrier D*;
$\forall\, x{\in}X.\ x \preceq a$; *Worder* (*Iod D X*)$]\!] \Longrightarrow$ *Worder* (*Iod D* ($X \cup \{a\}$))
apply (*frule insert-sub*[*of X carrier D a*], *assumption*)

apply (*subst Worder-def*)
 apply (*simp add*:*Iod-Order*)
 apply (*frule-tac Torder-adjunction*[*of X a*], *assumption*+)
 apply (*simp add*:*Worder.Torder*)
 apply (*simp add*:*Torder-def*)

apply (*subst Worder-axioms-def*)
 apply (*rule allI*, *rule impI*, *erule conjE*)
 apply (*simp add*:*Iod-carrier*)
 apply (*cut-tac insert-inc1*[*of X a*])

 apply (*case-tac Xa* ⊆ *X*)
 apply (*simp only*:*Worder-def*, (*erule conjE*)+) **apply** (
 simp only:*Worder-axioms-def*)
 apply (*frule-tac a* = *Xa* **in** *forall-spec*,
 thin-tac ∀ *Xa*. *Xa* ⊆ *carrier* (*Iod D X*) ∧ *Xa* ≠ {} ⟶
 (∃ *x*. *minimum-elem* (*Iod D X*) *Xa x*),
 simp add:*Iod-carrier*)
 apply (*thin-tac* ∀ *Xa*. *Xa* ⊆ *carrier* (*Iod D X*) ∧ *Xa* ≠ {} ⟶
 (∃ *x*. *minimum-elem* (*Iod D X*) *Xa x*),
 erule exE)
 apply (*frule-tac X* = *Xa* **and** *a* = *x* **in** *Order.minimum-elem-sub*[*of*
 Iod D (*insert a X*) *X*])
 apply (*cut-tac insert-inc1*[*of X a*], *simp add*:*Iod-carrier*, *assumption*+)
 apply (*cut-tac insert-inc1*[*of X a*],
 simp add:*Iod-sub-sub*) **apply** *blast*
 apply (*erule conjE*,
 frule-tac A = *Xa* **in** *insert-diff*[*of* - *a X*])
 apply (*case-tac Xa* − {*a*} = {})
 apply (*frule-tac A* = *Xa* **in** *nonempty-ex*, *erule exE*, *simp*,
 frule-tac c = *x* **and** *A* = *Xa* **and** *B* = {*a*} **in** *subsetD*, *assumption*+,
 simp,
 frule-tac A = *Xa* **in** *singleton-sub*[*of a*],
 frule-tac A = *Xa* **and** *B* = {*a*} **in** *equalityI*, *assumption*+, *simp*)
 apply (*simp add*:*minimum-elem-def*)
 apply (*cut-tac insert-inc2*[*of a X*],
 simp add:*Iod-le le-refl*)

 apply (*simp only*:*Worder-def*, (*erule conjE*)+,
 simp only:*Worder-axioms-def*)
 apply (*frule-tac a* = *Xa* − {*a*} **in** *forall-spec*,
 thin-tac ∀ *Xa*. *Xa* ⊆ *carrier* (*Iod D X*) ∧ *Xa* ≠ {} ⟶
 (∃ *x*. *minimum-elem* (*Iod D X*) *Xa x*), *simp add*:*Iod-carrier*)
 apply (*thin-tac* ∀ *Xa*. *Xa* ⊆ *carrier* (*Iod D X*) ∧ *Xa* ≠ {} ⟶
 (∃ *x*. *minimum-elem* (*Iod D X*) *Xa x*), *erule exE*)
 apply (*frule-tac Y* = *Xa* **and** *x* = *x* **in** *augmented-set-minimum*[*of X a*],
 assumption+, *blast*)
done

lemma *well-ord-Sadjunction*:⟦*Order D*; *X* ⊆ *carrier D*; *a* ∈ *carrier D*;
∀*x*∈*X*. *x* ≼$_D$ *a*; *Worder* (*SIod D X*)⟧ ⟹ *Worder* (*SIod D* (*X* ∪ {*a*}))
apply (*frule insert-sub*[*of X carrier D a*], *assumption*)
apply (*subst Worder-def*)
apply (*simp add*:*SIod-Order*)
apply (*frule Torder-Sadjunction*[*of D X a*], *assumption*+)
apply (*simp add*:*Worder.Torder*)
apply (*simp add*:*Torder-def*)

apply (*subst Worder-axioms-def*)
apply (*rule allI*, *rule impI*, *erule conjE*)
apply (*simp add*:*SIod-carrier*)
apply (*cut-tac insert-inc1*[*of X a*])

apply (*case-tac Xa* ⊆ *X*)
apply (*simp only*:*Worder-def*, (*erule conjE*)+) **apply** (
simp only:*Worder-axioms-def*)
apply (*frule-tac a* = *Xa* **in** *forall-spec*,
thin-tac ∀ *Xa*. *Xa* ⊆ *carrier* (*SIod D X*) ∧ *Xa* ≠ {} ⟶
(∃ *x*. *minimum-elem* (*SIod D X*) *Xa x*),
simp add:*SIod-carrier*)
apply (*thin-tac* ∀ *Xa*. *Xa* ⊆ *carrier* (*SIod D X*) ∧ *Xa* ≠ {} ⟶
(∃ *x*. *minimum-elem* (*SIod D X*) *Xa x*),
erule exE)
apply (*frule-tac X* = *Xa* **and** *a* = *x* **in** *minimum-elem-Ssub*[*of*
SIod D (*insert a X*) *X*])
apply (*cut-tac insert-inc1*[*of X a*], *simp add*:*SIod-carrier*, *assumption*+)
apply (*cut-tac insert-inc1*[*of X a*],
simp add:*SIod-sub-sub*) **apply** *blast*

apply (*erule conjE*, *frule-tac A* = *Xa* **in** *insert-diff*[*of* - *a X*])
apply (*case-tac Xa* − {*a*} = {})
apply (*frule-tac A* = *Xa* **in** *nonempty-ex*, *erule exE*, *simp*,
frule-tac c = *x* **and** *A* = *Xa* **and** *B* = {*a*} **in** *subsetD*, *assumption*+,
simp,
frule-tac A = *Xa* **in** *singleton-sub*[*of a*],
frule-tac A = *Xa* **and** *B* = {*a*} **in** *equalityI*, *assumption*+, *simp*)
apply (*simp add*:*minimum-elem-def*)
apply (*cut-tac insert-inc2*[*of a X*],
simp add:*SIod-le Order.le-refl*)

apply (*simp only*:*Worder-def*, (*erule conjE*)+,
simp only:*Worder-axioms-def*)
apply (*frule-tac a* = *Xa* − {*a*} **in** *forall-spec*,
thin-tac ∀ *Xa*. *Xa* ⊆ *carrier* (*SIod D X*) ∧ *Xa* ≠ {} ⟶
(∃ *x*. *minimum-elem* (*SIod D X*) *Xa x*), *simp add*:*SIod-carrier*)
apply (*thin-tac* ∀ *Xa*. *Xa* ⊆ *carrier* (*SIod D X*) ∧ *Xa* ≠ {} ⟶
(∃ *x*. *minimum-elem* (*SIod D X*) *Xa x*), *erule exE*)
apply (*frule-tac Y* = *Xa* **and** *x* = *x* **in** *augmented-Sset-minimum*[*of D X a*],

assumption+, *blast*)
done

lemma (**in** *Worder*) *Worder-adjunct-ord*:$a \notin$ *carrier* $D \Longrightarrow$
Worder (*adjunct-ord* D a)
apply (*frule Torder-adjunct-ord*[*of a*])
apply (*intro-locales*)
apply (*simp add*:*Torder-def*)
apply (*simp add*:*Torder-def*)

apply (*cut-tac carrier-adjunct-ord*[*THEN sym*, *of a*],
cut-tac Un-upper1[*of carrier D* {*a*}], *simp*)
apply (*simp add*:*Torder-def*, *erule conjE*)
apply (*cut-tac insertI1*[*of a carrier D*])
apply (*frule Order.well-ord-adjunction*[*of adjunct-ord D a carrier D a*],
assumption+)
apply (*frule sym*, *thin-tac insert a* (*carrier D*) = *carrier* (*adjunct-ord D a*),
simp)
apply (*cut-tac adjunct-ord-large-a*[*of a*],
rule ballI,
frule-tac $x = x$ **in** *bspec*, *assumption*,
thin-tac $\forall x \in$ *carrier* D. $x \prec_{\textit{adjunct-ord } D\ a}\ a$,
cut-tac insertI1[*of a carrier D*], *simp*,
frule-tac $c = x$ **in** *subsetD*[*of carrier D carrier* (*adjunct-ord D a*)],
assumption+,
simp add:*Order.less-imp-le*, *rule Order*,
assumption+)
apply (*simp add*:*adjunct-ord-selfD*[*THEN sym*])
prefer *2*
apply (*simp add*:*Order.Iod-self*[*THEN sym*, *of adjunct-ord D a*] *Worder-def*)
apply *unfold-locales*
done

2.5 Zorn's lemma

definition
Chain :: - $\Rightarrow$ *'a set* $\Rightarrow$ *bool* **where**
Chain D $C \longleftrightarrow C \subseteq$ *carrier* $D \wedge$ *Torder* (*Iod* D C)

definition
upper-bound :: [-, *'a set*, *'a*] $\Rightarrow$ *bool*
((*3ub*$_1$/ -/ -) [*100*,*101*]*100*) **where**
$ub_D\ S\ b \longleftrightarrow b \in$ *carrier* $D \wedge (\forall s \in S.\ s \preceq_D b)$

definition
inductive-set :: - $\Rightarrow$ *bool* **where**
inductive-set $D \longleftrightarrow (\forall C.\ (\textit{Chain } D\ C \longrightarrow (\exists b.\ ub_D\ C\ b)))$

definition

maximal-element :: [-, 'a] ⇒ *bool* ((*maximal*ı/ -) [*101*]*100*) **where**
maximal$_D$ *m* ⟷ *m* ∈ *carrier D* ∧ (∀ *b*∈*carrier D*. *m* ≼$_D$ *b* ⟶ *m* = *b*)

definition
upper-bounds::[-, 'a *set*] ⇒ 'a *set* **where**
upper-bounds D H = {*x*. *ub*$_D$ *H x*}

definition
Sup :: [-, 'a *set*] ⇒ 'a **where**
Sup D X = (*THE x*. *minimum-elem D* (*upper-bounds D X*) *x*)

definition
S-inductive-set :: - ⇒ *bool* **where**
S-inductive-set D ⟷ (∀ *C*. *Chain D C* ⟶
(∃ *x*∈*carrier D*. *minimum-elem D* (*upper-bounds D C*) *x*))

lemma (**in** *Order*) *mem-upper-bounds*:⟦*X* ⊆ *carrier D*; *b* ∈ *carrier D*;
∀ *x*∈*X*. *x* ≼ *b*⟧ ⟹ *ub X b*
apply (*simp add*:*upper-bounds-def upper-bound-def*)
done

lemma (**in** *Order*) *Torder-Chain*:⟦*X* ⊆ *carrier D*; *Torder* (*Iod D X*)⟧
⟹ *Chain D X*
apply (*simp add*:*Chain-def Torder-def*)
done

lemma (**in** *Order*) *Chain-Torder*:*Chain D X* ⟹
Torder (*Iod D X*)
apply (*simp add*:*Chain-def*)
done

lemma (**in** *Order*) *Chain-sub*:*Chain D X* ⟹ *X* ⊆ *carrier D*
apply (*simp add*:*Chain-def*)
done

lemma (**in** *Order*) *Chain-sub-Chain*:⟦*Chain D X*; *Y* ⊆ *X* ⟧ ⟹ *Chain D Y*
apply (*frule Chain-sub*[*of X*],
frule Chain-Torder[*of X*],
frule Torder.Iod-Torder[*of Iod D X Y*], *simp add*:*Iod-carrier*)
apply (*simp add*:*Iod-sub-sub*[*of Y X*],
frule subset-trans[*of Y X carrier D*], *assumption*+)
apply (*simp add*:*Torder-Chain*[*of Y*])
done

lemma (**in** *Order*) *upper-bounds-sub*:*X* ⊆ *carrier D* ⟹
upper-bounds D X ⊆ *carrier D*
by (*rule subsetI*, *simp add*:*upper-bounds-def upper-bound-def*)

lemma (**in** *Order*) *Sup*:⟦*X* ⊆ *carrier D*; *minimum-elem D* (*upper-bounds D X*) *a*⟧
⟹ *Sup D X* = *a*
apply (*subst Sup-def*)
apply (*rule the-equality*, *assumption*,
rule-tac ?a1.0 = *x* **in** *minimum-elem-unique*[*of upper-bounds D X - a*])
apply (*rule subsetI*, *thin-tac minimum-elem D* (*upper-bounds D X*) *a*,
thin-tac minimum-elem D (*upper-bounds D X*) *x*,
simp add:*upper-bounds-def upper-bound-def*)
apply *assumption*+
done

lemma (**in** *Worder*) *Sup-mem*:⟦*X* ⊆ *carrier D*; ∃ *b*. *ub X b*⟧ ⟹
Sup D X ∈ *carrier D*
apply (*frule upper-bounds-sub*[*of X*],
frule minimum-elem-mem[*of upper-bounds D X Sup D X*],
simp add:*Sup-def*, *rule theI'*)
apply (*rule ex-ex1I*)
apply (*cut-tac ex-minimum*)
apply (*frule-tac a* = *upper-bounds D X* **in** *forall-spec*,
thin-tac ∀ *X*. *X* ⊆ *carrier D* ∧ *X* ≠ {} ⟶ (∃ *x*. *minimum-elem D X x*),
simp, *erule exE*)
apply (*rule-tac x* = *b* **in** *nonempty*[*of - upper-bounds D X*],
simp add:*upper-bounds-def*, *assumption*,
rule-tac ?a1.0 = *x* **and** *?a2.0* = *y* **in** *minimum-elem-unique*[*of*
upper-bounds D X], *assumption*+,
simp add:*upper-bounds-def upper-bound-def*)
done

lemma (**in** *Order*) *S-inductive-sup*:⟦*S-inductive-set D*; *Chain D X*⟧ ⟹
minimum-elem D (*upper-bounds D X*) (*Sup D X*)
apply (*simp add*:*S-inductive-set-def*)
apply (*frule-tac a* = *X* **in** *forall-spec*, *assumption*,
thin-tac ∀ *C*. *Chain D C* ⟶ (∃ *x*∈*carrier D*. *minimum-elem D*
(*upper-bounds D C*) *x*))
apply (*erule bexE*)
apply (*frule Chain-sub*[*of X*])
apply (*frule-tac a* = *x* **in** *Sup*[*of X*], *assumption*+)
apply *simp*
done

lemma (**in** *Order*) *adjunct-Chain*:⟦*Chain D X*; *b* ∈ *carrier D*; ∀ *x*∈*X*. *x* ≼ *b*⟧ ⟹
Chain D (*insert b X*)
apply (*simp add*:*Chain-def*, *erule conjE*)
apply (*frule Torder-adjunction*[*of X b*], *assumption*+)
apply *simp*
done

lemma (**in** *Order*) *S-inductive-sup-mem*:⟦*S-inductive-set D*; *Chain D X*⟧ ⟹
Sup D X ∈ *carrier D*

apply (*frule-tac* $X = X$ **in** *S-inductive-sup*, *assumption*)
apply (*simp add*:*minimum-elem-def*, (*erule conjE*)+,
simp add:*upper-bounds-def*, *simp add*:*upper-bound-def*)
done

lemma (**in** *Order*) *S-inductive-Sup-min-bounds*:⟦*S-inductive-set D*; *Chain D X*;
ub X b⟧ ⟹ *Sup D X* ⪯ *b*
apply (*frule S-inductive-sup*[*of X*], *assumption*+,
simp add:*minimum-elem-def*, *erule conjE*)
apply (*frule-tac x* = *b* **in** *bspec*,
simp add:*upper-bounds-def*, *assumption*)
done

lemma (**in** *Order*) *S-inductive-sup-bound*:⟦*S-inductive-set D*; *Chain D X*⟧ ⟹
∀ *x*∈*X*. *x* ⪯ (*Sup D X*)
apply (*frule-tac X* = *X* **in** *S-inductive-sup*, *assumption*+)
apply (*rule ballI*)
apply (*simp add*:*minimum-elem-def*) **apply** (*erule conjE*)
apply (*thin-tac* ∀ *x*∈*upper-bounds D X*. *Sup D X* ⪯ *x*)
apply (*simp add*:*upper-bounds-def upper-bound-def*)
done

lemma (**in** *Order*) *S-inductive-Sup-in-ChainTr*:
⟦*S-inductive-set D*; *Chain D X*; *c* ∈ *carrier* (*Iod D* (*insert* (*Sup D X*) *X*));
Sup D X ∉ *X*;
∀ *y*∈*carrier* (*Iod D* (*insert* (*Sup D X*) *X*)).
c ≺*Iod D* (*insert* (*Sup D X*) *X*) *y* ⟶ ¬ *y* ≺*Iod D* (*insert* (*Sup D X*) *X*) *Sup D*
X⟧ ⟹
c ∈ *upper-bounds D X*
apply (*subst upper-bounds-def*, *simp add*:*upper-bound-def*)
apply (*frule Chain-sub*[*of X*],
frule S-inductive-sup-mem[*of X*], *assumption*+,
frule insert-sub[*of X carrier D Sup D X*], *assumption*)
apply (*rule conjI*)
apply (*simp add*:*Iod-carrier*,
frule Chain-sub[*of X*],
frule insert-sub[*of X carrier D Sup D X*], *assumption*,
erule disjE, *simp*, *simp add*:*subsetD*)

apply (*rule ballI*)
apply (*simp add*:*Iod-carrier*, (*erule conjE*)+,
frule S-inductive-sup-bound[*of X*], *assumption*+)
apply (*erule disjE*, *simp*)
apply (*frule-tac x* = *s* **in** *bspec*, *assumption*,
thin-tac ∀ *y*∈*X*. *c* ≺*Iod D* (*insert* (*Sup D X*) *X*) *y* ⟶
¬ *y* ≺*Iod D* (*insert* (*Sup D X*) *X*) *Sup D X*,
frule-tac x = *s* **in** *bspec*, *assumption*,
thin-tac ∀ *x*∈*X*. *x* ⪯ *Sup D X*)
apply (*frule insert-sub*[*of X carrier D Sup D X*], *assumption*+,

cut-tac subset-insertI[*of X Sup D X*],
frule-tac c = s **in** *subsetD*[*of X insert* (*Sup D X*) *X*], *assumption*+,
frule-tac c = c **in** *subsetD*[*of X insert* (*Sup D X*) *X*], *assumption*+,
frule Iod-Order[*of insert* (*Sup D X*) *X*])
apply (*subst Iod-le*[*THEN sym*, *of insert* (*Sup D X*) *X*], *assumption*+,
rule contrapos-pp, (*simp del*:*insert-subset*)+)
apply (*frule Torder-adjunction* [*of X Sup D X*], *assumption*+,
rule S-inductive-sup-bound[*of X*], *assumption*+, *simp add*:*Chain-Torder*)
apply (*frule-tac a = s* **and** *b = c* **in**
Torder.not-le-less[*of Iod D* (*X* ∪ {*Sup D X*})])
apply (*simp add*:*Iod-carrier*, *simp add*:*subsetD*,
simp add:*Iod-carrier*)
apply (*thin-tac c* ≺*Iod D* (*insert* (*Sup D X*) *X*) *Sup D X* ⟶
¬ *Sup D X* ≺*Iod D* (*insert* (*Sup D X*) *X*) *Sup D X*)
apply (*simp del*:*insert-sub*,
frule-tac a = s **in**
Torder.not-less-le[*of Iod D* (*insert* (*Sup D X*) *X*) - *Sup D X*])
apply (*frule insert-sub*[*of X carrier D Sup D X*], *assumption*,
simp add:*Iod-carrier subsetD*,
frule insert-sub[*of X carrier D Sup D X*], *assumption*,
simp add:*Iod-carrier*)
apply *simp*
apply (*simp add*:*Iod-le*)
apply (*frule-tac c = s* **in** *subsetD*[*of X carrier D*], *assumption*+,
frule-tac a = s **and** *b = Sup D X* **in** *le-antisym*, *assumption*+)
apply *simp*
done

lemma (**in** *Order*) *S-inductive-Sup-in-Chain*:⟦*S-inductive-set D*; *Chain D X*;
ExPre (*Iod D* (*insert* (*Sup D X*) *X*)) (*Sup D X*)⟧ ⟹ *Sup D X* ∈ *X*
apply (*frule S-inductive-sup-mem*[*of X*], *assumption*+)
apply (*frule Chain-sub*[*of X*],
frule insert-sub[*of X carrier D Sup D X*], *assumption*)
apply (*rule contrapos-pp*, (*simp del*:*insert-subset*)+)
apply (*frule Iod-Order*[*of insert* (*Sup D X*) *X*])
apply (*frule Order.Pre-element*[*of Iod D* (*insert* (*Sup D X*) *X*) *Sup D X*])
apply (*simp add*:*Iod-carrier*) **apply** *assumption*
apply ((*erule conjE*)+, *simp del*:*insert-subset*)
apply (*frule S-inductive-Sup-in-ChainTr*[*of X
Pre* (*Iod D* (*insert* (*Sup D X*) *X*)) (*Sup D X*)], *assumption*+)
apply (*simp add*:*upper-bounds-def*)
apply (*frule S-inductive-Sup-min-bounds*[*of X
Pre* (*Iod D* (*insert* (*Sup D X*) *X*)) (*Sup D X*)], *assumption*+,
thin-tac ∀ *y*∈*carrier* (*Iod D* (*insert* (*Sup D X*) *X*)).
Pre (*Iod D* (*insert* (*Sup D X*) *X*)) (*Sup D X*) ≺*Iod D* (*insert* (*Sup D X*) *X*)
y ⟶ ¬ *y* ≺*Iod D* (*insert* (*Sup D X*) *X*) *Sup D X*)
apply (*frule Order.less-le-trans*[*of Iod D* (*insert* (*Sup D X*) *X*)
Pre (*Iod D* (*insert* (*Sup D X*) *X*)) (*Sup D X*)
Sup D X Pre (*Iod D* (*insert* (*Sup D X*) *X*)) (*Sup D X*)])

apply *assumption+*

apply (*frule insert-sub*[*of X carrier D Sup D X*], *assumption*,
simp add:*Iod-carrier*) **apply** *assumption+*
apply (*frule insert-sub*[*of X carrier D Sup D X*], *assumption*,
simp add:*Iod-carrier Iod-le*)
apply (*simp add*:*oless-def*)
done

lemma (**in** *Order*) *S-inductive-bounds-compare*:⟦*S-inductive-set D*; *Chain D X1*;
Chain D X2; *X1* ⊆ *X2*⟧ ⟹ *upper-bounds D X2* ⊆ *upper-bounds D X1*
apply (*rule subsetI*,
simp add:*upper-bounds-def upper-bound-def*,
erule conjE, *rule ballI*,
frule-tac c = *s* **in** *subsetD*[*of X1 X2*], *assumption+*)
apply *simp*
done

lemma (**in** *Order*) *S-inductive-sup-compare*:⟦*S-inductive-set D*; *Chain D X1*;
Chain D X2; *X1* ⊆ *X2*⟧ ⟹ *Sup D X1* ≼ *Sup D X2*
apply (*frule S-inductive-bounds-compare*[*of X1 X2*], *assumption+*,
frule Chain-sub[*of X1*], *frule Chain-sub*[*of X2*],
frule upper-bounds-sub[*of X1*], *frule upper-bounds-sub*[*of X2*])
apply (*rule-tac s* = *Sup D X2* **and** *t* = *Sup D X1* **in**
compare-minimum-elements[*of upper-bounds D X2 upper-bounds D X1*],
assumption+,
simp add:*S-inductive-sup*, *simp add*:*S-inductive-sup*)
done

definition
Wa :: [-, *'a set*, *'a* ⇒ *'a*, *'a*] ⇒ *bool* **where**
Wa D W g a ⟷ *W* ⊆ *carrier D* ∧ *Worder* (*Iod D W*) ∧ *a* ∈ *W* ∧ (∀ *x*∈*W*. *a*
≼$_D$ *x*) ∧
(∀ *x*∈*W*. (*if* (*ExPre* (*Iod D W*) *x*) *then* *g* (*Pre* (*Iod D W*) *x*) = *x else*
(*if a* ≠ *x then Sup D* (*segment* (*Iod D W*) *x*) = *x else a* = *a*)))

definition
WWa :: [-, *'a* ⇒ *'a*, *'a*] ⇒ *'a set set* **where**
WWa D g a = {*W*. *Wa D W g a*}

lemma (**in** *Order*) *mem-of-WWa*:⟦*W* ⊆ *carrier D*; *Worder* (*Iod D W*); *a* ∈ *W*;
(∀ *x*∈*W*. *a* ≼ *x*);
(∀ *x*∈*W*. (*if* (*ExPre* (*Iod D W*) *x*) *then* *g* (*Pre* (*Iod D W*) *x*) = *x else*
(*if a* ≠ *x then Sup D* (*segment* (*Iod D W*) *x*) = *x else a* = *a*)))⟧ ⟹
W ∈ *WWa D g a*
by (*simp add*:*WWa-def*, *simp add*:*Wa-def*)

lemma (**in** *Order*) *mem-WWa-then*:*W* ∈ *WWa D g a* ⟹ *W* ⊆ *carrier D* ∧
Worder (*Iod D W*) ∧ *a* ∈ *W* ∧ (∀ *x*∈*W*. *a* ≼ *x*) ∧

$(\forall\, x{\in}W.\ (\mathit{if}\ (\mathit{ExPre}\ (\mathit{Iod}\ D\ W)\ x)\ \mathit{then}\ \ g\ (\mathit{Pre}\ (\mathit{Iod}\ D\ W)\ x) = x\ \mathit{else}$
$(\mathit{if}\ a \neq x\ \mathit{then}\ \mathit{Sup}\ D\ (\mathit{segment}\ (\mathit{Iod}\ D\ W)\ x) = x\ \mathit{else}\ a = a)))$
by (*simp add*:*WWa-def Wa-def*)

lemma (**in** *Order*) *mem-wwa-Worder*:$W \in \mathit{WWa}\ D\ g\ a \Longrightarrow \mathit{Worder}\ (\mathit{Iod}\ D\ W)$
by (*simp add*:*WWa-def Wa-def*)

lemma (**in** *Order*) *mem-WWa-sub-carrier*:$W \in \mathit{WWa}\ D\ g\ a \Longrightarrow W \subseteq \mathit{carrier}\ D$
by (*simp add*:*WWa-def Wa-def*)

lemma (**in** *Order*) *Union-WWa-sub-carrier*:$\bigcup\ \mathit{WWa}\ D\ g\ a \subseteq \mathit{carrier}\ D$
by (*rule Union-least*[*of WWa D g a carrier D*], *simp add*:*mem-WWa-sub-carrier*)

lemma (**in** *Order*) *mem-WWa-inc-a*:$W \in \mathit{WWa}\ D\ g\ a \Longrightarrow a \in W$
by (*simp add*:*WWa-def Wa-def*)

lemma (**in** *Order*) *mem-WWa-Chain*:$W \in \mathit{WWa}\ D\ g\ a \Longrightarrow \mathit{Chain}\ D\ W$
apply (*simp add*:*Chain-def*)
apply (*simp add*:*mem-WWa-sub-carrier*)
apply (*frule mem-wwa-Worder*[*of W*])
apply (*simp add*:*Worder.Torder*)
done

lemma (**in** *Order*) *Sup-adjunct-Sup*:$[\![\mathit{S\text{-}inductive\text{-}set}\ D;$
$f \in \mathit{carrier}\ D \to \mathit{carrier}\ D;\ a \in \mathit{carrier}\ D;\ \forall\, x{\in}\mathit{carrier}\ D.\ x \preceq f\ x;$
$W \in \mathit{WWa}\ D\ f\ a;\ \mathit{Sup}\ D\ W \notin W]\!]$
$\Longrightarrow \mathit{Sup}\ D\ (\mathit{insert}\ (\mathit{Sup}\ D\ W)\ W) = \mathit{Sup}\ D\ W$

apply (*frule mem-WWa-Chain*[*of W*],
frule S-inductive-sup-bound[*of W*], *assumption*,
frule mem-wwa-Worder[*of W*],
frule mem-WWa-sub-carrier[*of W*],
frule S-inductive-sup-mem[*of W*], *assumption*+)
apply (*frule well-ord-adjunction*[*of W Sup D W*], *assumption*+, *simp*,
frule insert-sub[*of W carrier D Sup D W*], *assumption*+,
frule Worder.Torder[*of Iod D* (*insert* (*Sup D W*) *W*)],
frule Torder-Chain[*of insert* (*Sup D W*) *W*], *assumption*+,
frule S-inductive-sup-mem[*of insert* (*Sup D W*) *W*], *assumption*+)
apply (*rule le-antisym*[*of Sup D* (*insert* (*Sup D W*) *W*) *Sup D W*], *assumption*+,
rule S-inductive-Sup-min-bounds[*of insert* (*Sup D W*) *W Sup D W*],
assumption+,
simp add:*upper-bound-def*, *simp add*:*le-refl*)
apply (*rule S-inductive-sup-compare*[*of W insert* (*Sup D W*) *W*], *assumption*+)
apply (*simp add*:*subset-insertI*[*of W Sup D W*])
done

lemma (**in** *Order*) *BNTr1*:$a \in \mathit{carrier}\ D \Longrightarrow \mathit{Worder}\ (\mathit{Iod}\ D\ \{a\})$
apply *intro-locales*
apply (*frule singleton-sub*[*of a carrier D*],

rule Iod-Order[*of* {*a*}], *assumption*)
apply (*simp add*:*Torder-axioms-def*)
apply (*rule allI*, *rule impI*)+
apply (*simp add*:*Iod-carrier*, *simp add*:*Iod-le le-refl*)
apply (*simp add*:*Worder-axioms-def*)
apply (*rule allI*, *rule impI*, *erule conjE*, *simp add*:*Iod-carrier*)
apply (*frule-tac* $A = X$ **in** *nonempty-ex*, *erule exE*,
frule-tac $c = x$ **and** $A = X$ **and** $B = \{a\}$ **in** *subsetD*, *assumption*+,
simp,
frule-tac $A = X$ **in** *singleton-sub*[*of a*],
frule-tac $A = X$ **and** $B = \{a\}$ **in** *equalityI*, *assumption*+, *simp*)
apply (*simp add*:*minimum-elem-def Iod-le le-refl*)
done

lemma (**in** *Order*) *BNTr2*:⟦$f \in$ *carrier* $D \to$ *carrier* D; $a \in$ *carrier* D;
$\forall x \in$ *carrier* D. $x \preceq (f\ x)$⟧ $\Longrightarrow \{a\} \in$ *WWa* $D\ f\ a$
apply (*simp add*:*WWa-def Wa-def*)
apply (*simp add*:*Not-ExPre*[*of a*])
apply (*simp add*:*BNTr1 le-refl*)
done

lemma (**in** *Order*) *BNTr2-1*:⟦$f \in$ *carrier* $D \to$ *carrier* D; $a \in$ *carrier* D;
$\forall x \in$ *carrier* D. $x \preceq (f\ x)$; *W* $\in$ *WWa* $D\ f\ a$⟧ $\Longrightarrow \forall x \in W$. $a \preceq x$
by (*rule ballI*, *simp add*:*WWa-def Wa-def*)

lemma (**in** *Order*) *BNTr3*:⟦$f \in$ *carrier* $D \to$ *carrier* D; $a \in$ *carrier* D;
$\forall x \in$ *carrier* D. $x \preceq (f\ x)$; *W* $\in$ *WWa* $D\ f\ a$⟧ $\Longrightarrow$ *minimum-elem* (*Iod* D *W*)
W a

apply (*frule mem-WWa-inc-a*[*of W*])
apply (*subst minimum-elem-def*)
apply *simp*
apply (*rule ballI*)
apply (*frule mem-WWa-sub-carrier*[*of W f a*])
apply (*frule BNTr2-1*[*of f a W*], *assumption*+)
apply (*simp add*:*Iod-le*)
done

lemma (**in** *Order*) *Adjunct-segment-sub*:⟦*S-inductive-set D*; *Chain D X*⟧ $\Longrightarrow$
segment (*Iod D* (*insert* (*Sup D X*) *X*)) (*Sup D X*) $\subseteq$ *X*
apply (*frule S-inductive-sup-mem*[*of X*], *assumption*+,
frule Chain-sub[*of X*],
frule insert-sub[*of X carrier D Sup D X*], *assumption*)
apply (*rule subsetI*)
apply (*simp add*:*segment-def*)
apply (*case-tac Sup D X* $\notin$ *carrier* (*Iod D* (*insert* (*Sup D X*) *X*)), *simp*)
apply (*simp add*:*Iod-carrier*)

apply (*simp add*:*Iod-carrier*, *erule conjE*, *simp add*:*oless-def*)

done

lemma (**in** *Order*) *Adjunct-segment-eq*:⟦*S-inductive-set D*; *Chain D X*;
Sup D X ∉ *X*⟧ ⟹
segment (*Iod D* (*insert* (*Sup D X*) *X*)) (*Sup D X*) = *X*
apply (*frule Chain-sub*[*of X*],
frule Adjunct-segment-sub[*of X*], *assumption*)
apply (*rule equalityI*, *assumption*)
apply (*frule S-inductive-sup-mem*[*of X*], *assumption*+,
frule insert-sub[*of X carrier D Sup D X*], *assumption*+,
rule subsetI,
simp add:*segment-def Iod-carrier*,
cut-tac subset-insertI[*of X Sup D X*],
cut-tac insertI1[*of Sup D X X*],
frule-tac c = *x* **in** *subsetD*[*of X insert* (*Sup D X*) *X*], *assumption*+)
apply (*simp add*:*Iod-less*[*of insert* (*Sup D X*) *X*],
frule S-inductive-sup-bound[*of X*], *assumption*+,
frule-tac x = *x* **in** *bspec*, *assumption*+,
thin-tac ∀ *x*∈*X*. *x* ≼ *Sup D X*,
simp add:*oless-def*)
apply (*rule contrapos-pp*, *simp*+)
done

definition
fixp :: [′*a* ⇒ ′*a*, ′*a*] ⇒ *bool* **where**
fixp f a ⟷ *f a* = *a*

lemma (**in** *Order*) *fixp-same*:⟦*W1* ⊆ *carrier D*; *W2* ⊆ *carrier D*; *t* ∈ *W1*;
b ∈ *carrier D*; *ord-isom* (*Iod D W1*) (*Iod* (*Iod D W2*) (*segment* (*Iod D W2*) *b*))
g;
∀ *u*∈*segment* (*Iod D W1*) *t*. *fixp g u*⟧ ⟹
segment (*Iod D W1*) *t* = *segment* (*Iod D W2*) (*g t*)

apply (*frule Iod-Order*[*of W1*],
frule Iod-Order[*of W2*],
frule Order.segment-sub[*of Iod D W1 t*],
frule Order.segment-sub[*of Iod D W2 b*],
frule Order.Iod-Order[*of Iod D W2 segment* (*Iod D W2*) *b*],
assumption+)
apply (*frule Order.ord-isom-segment-segment*[*of Iod D W1*
Iod (*Iod D W2*) (*segment* (*Iod D W2*) *b*) *g t*], *assumption*+)
apply (*simp add*:*Iod-carrier*, *simp add*:*Iod-carrier*)
apply (*frule Order.ord-isom-mem*[*of Iod D W1*
Iod (*Iod D W2*) (*segment* (*Iod D W2*) *b*) *g t*], *assumption*+,
simp add:*Iod-carrier*)
apply (*frule Order.Iod-segment-segment*[*of Iod D W2 g t b*],
assumption, *simp*)
apply (*simp add*:*Iod-sub-sub*[*of segment* (*Iod D W1*) *t W1*])
apply (*frule Order.segment-sub* [*of Iod* (*Iod D W2*)

```
                                    (segment (Iod D W2) b) g t])
      apply (simp add:Iod-sub-sub)
      apply (frule subset-trans[of segment (Iod D W2) b W2 carrier D],
                  assumption+)
      apply (simp add:Iod-carrier)
      apply (frule Order.segment-sub[of Iod D W2 g t],
            simp add:Iod-carrier[of W2])
      apply (simp add:Iod-sub-sub[of segment (Iod D W2) (g t) W2],
            thin-tac Iod (Iod D (segment (Iod D W2) b))
                  (segment (Iod D (segment (Iod D W2) b)) (g t)) =
                   Iod D (segment (Iod D W2) (g t)))
      apply (frule subset-trans[of segment (Iod D W1) t W1 carrier D],
            assumption+,
            frule subset-trans[of segment (Iod D W2) (g t) W2 carrier D],
            assumption+,
            frule Iod-Order[of segment (Iod D W1) t],
            frule Iod-Order[of segment (Iod D W2) (g t)])
      apply (thin-tac segment (Iod D (segment (Iod D W2) b)) (g t) ⊆
                        segment (Iod D W2) b,
            thin-tac segment (Iod D W2) b ⊆ carrier D,
            thin-tac segment (Iod D W2) (g t) ⊆ W2,
            thin-tac ord-isom (Iod D W1) (Iod D (segment (Iod D W2) b)) g)
apply (rule equalityI)
 apply (rule subsetI)
 apply (frule-tac a = x in Order.ord-isom-mem[of Iod D (segment (Iod D W1) t)
       Iod D (segment (Iod D W2) (g t))
         restrict g (carrier (Iod D (segment (Iod D W1) t)))], assumption+,
        simp add:Iod-carrier, simp add:Iod-carrier, simp add:fixp-def)
apply (metis Order.Iod-carrier [OF Order-axioms])
 apply (rule subsetI)
 apply (frule-tac b = x in Order.ord-isom-surj[of
        Iod D (segment (Iod D W1) t) Iod D (segment (Iod D W2) (g t))
        restrict g (carrier (Iod D (segment (Iod D W1) t)))])
apply assumption
apply (metis Order.Iod-carrier [OF Order-axioms])
apply (metis Order.Iod-carrier [OF Order-axioms] Order.segment-free [OF Order-axioms])
apply (metis Order.Iod-carrier [OF Order-axioms] fixp-def restrict-apply)
done

lemma (in Order) BNTr4-1:⟦f ∈ carrier D → carrier D;  a ∈ carrier D;
    b ∈ carrier D; ∀ x∈carrier D. x ≼ (f x); W1 ∈ WWa D f a; W2 ∈ WWa D f
a;
    ord-isom (Iod D W1) (Iod D (segment (Iod D W2) b)) g⟧ ⟹
    ∀ x∈W1. g x = x
 apply (frule mem-wwa-Worder[of W1 f a],
       frule mem-wwa-Worder[of W2 f a],
       frule Worder.Order[of Iod D W2],
       frule mem-WWa-sub-carrier[of W2 f a],
       frule mem-WWa-sub-carrier[of W1 f a],
```

cut-tac Worder.segment-Worder[*of Iod D W2 b*],
simp add:*Worder.Order*)
apply (*cut-tac Order.segment-sub*[*of Iod D W2 b*],
simp add:*Iod-carrier*,
frule subset-trans[*of segment* (*Iod D W2*) *b W2 carrier D*],
assumption+,
frule Iod-Order[*of segment* (*Iod D W2*) *b*])
apply (*frule Worder.Order*[*of Iod D W1*],
frule Order.ord-isom-onto[*of Iod D W1*
Iod D (*segment* (*Iod D W2*) *b*) *g*], *assumption*+)

apply (*frule Order.ord-isom-minimum*[*of Iod D W1*
Iod D (*segment* (*Iod D W2*) *b*) *g W1 a*], *assumption*+,
simp add:*Iod-carrier*, *simp add*:*Iod-carrier mem-WWa-inc-a*,
simp add:*BNTr3*)
apply (*frule Order.ord-isom-onto*[*of Iod D W1*
Iod D (*segment* (*Iod D W2*) *b*) *g*], *assumption*+,
simp add:*Iod-carrier*, *frule Worder.Torder*[*of Iod D W2*])
apply (*simp add*:*minimum-elem-sub*[*THEN sym*, *of segment* (*Iod D W2*) *b*
segment (*Iod D W2*) *b*])

apply (*simp add*:*minimum-elem-sub*[*of W2 segment* (*Iod D W2*) *b*])
apply (*frule Torder.minimum-segment-of-sub*[*of Iod D W2 W2 b g a*],
simp add:*Iod-carrier*, *cut-tac subset-self*[*of W2*],
simp add:*Iod-sub-sub*[*of W2 W2*],
thin-tac minimum-elem (*Iod D W2*) (*segment* (*Iod D W2*) *b*) (*g a*),
frule BNTr3[*of f a W2*], *assumption*+)
apply (*frule Worder.Order*[*of Iod D W2*],
frule Order.minimum-elem-unique[*of Iod D W2 W2 g a a*],
simp add:*Iod-carrier*, *assumption*+)

apply (*simp add*:*Iod-sub-sub*[*THEN sym*, *of segment* (*Iod D W2*) *b W2*],
frule Worder.transfinite-induction[*of Iod D W1 a fixp g*],
simp add:*Iod-carrier*, *simp add*:*BNTr3*, *simp add*:*fixp-def*,
rule ballI, *rule impI*)

apply (*frule-tac a* = *t* **in** *Worder-ord-isom-mem*[*of Iod D W1*
Iod (*Iod D W2*) (*segment* (*Iod D W2*) *b*) *g*], *assumption*+,
frule Iod-carrier[*THEN sym*, *of W2*],
frule subset-trans[*of segment* (*Iod D W2*) *b W2*
carrier (*Iod D W2*)], *simp*,
thin-tac W2 = *carrier* (*Iod D W2*),
simp add:*Order.Iod-carrier*[*of Iod D W2 segment* (*Iod D W2*) *b*],
frule-tac c = *g t* **in** *subsetD*[*of segment* (*Iod D W2*) *b*
carrier (*Iod D W2*)], *assumption*+,
simp add:*Iod-carrier*)
apply (*case-tac t* = *a*)
apply (*simp add*:*fixp-def*)
apply (*case-tac ExPre* (*Iod D W1*) *t*,

frule-tac $a = t$ **in** *Worder.ord-isom-Pre1*[*of Iod D W1*
Iod (*Iod D W2*) (*segment* (*Iod D W2*) *b*) - *g*], *assumption*+,
simp add:*Iod-carrier*, *assumption*+)
apply (*frule-tac* $a = t$ **in** *Worder.ord-isom-Pre2*[*of Iod D W1*
Iod (*Iod D W2*) (*segment* (*Iod D W2*) *b*) - *g*], *assumption*+,
simp add:*Iod-carrier*, *assumption*+)
apply (*frule-tac* $a = t$ **in** *Order.Pre-in-segment*[*of Iod D W1*],
simp add:*Iod-carrier*, *assumption*)
apply (*frule-tac* $x =$ *Pre* (*Iod D W1*) t **in** *bspec*, *assumption*,
thin-tac $\forall u \in$ *segment* (*Iod D W1*) t. *fixp g u*)
apply (*simp add*:*fixp-def*)
apply (*frule-tac* $a = g\ t$ **in** *Worder.Pre-segment*[*of Iod D W2* - *b*],
simp add:*Iod-carrier*, *assumption*+)

apply (*rotate-tac* -2, *frule sym*, *thin-tac Pre* (*Iod D W1*) $t =$
Pre (*Iod* (*Iod D W2*) (*segment* (*Iod D W2*) *b*)) (*g t*), *simp*,
thin-tac Pre (*Iod* (*Iod D W2*) (*segment* (*Iod D W2*) *b*)) (*g t*) =
Pre (*Iod D W1*) *t*)
apply (*erule conjE*)
apply (*simp add*:*WWa-def Wa-def*, (*erule conjE*)+,
thin-tac $\forall x \in W1.\ a \preceq x$, *thin-tac* $\forall x \in W2.\ a \preceq x$,
thin-tac $\forall x \in$ *carrier D*. $x \preceq f\ x$)
apply (*frule-tac* $x = t$ **in** *bspec*, *assumption*+,
thin-tac $\forall x \in W1$.
if ExPre (*Iod D W1*) *x then f* (*Pre* (*Iod D W1*) *x*) = *x*
else if a $\neq$ *x then Sup D* (*segment* (*Iod D W1*) *x*) = *x else a* = *a*,
frule-tac $x = g\ t$ **in** *bspec*, *assumption*+,
thin-tac $\forall x \in W2$.
if ExPre (*Iod D W2*) *x then f* (*Pre* (*Iod D W2*) *x*) = *x*
else if a $\neq$ *x then Sup D* (*segment* (*Iod D W2*) *x*) = *x else a* = *a*)
apply *simp*
apply (*frule-tac* $a = t$ **in** *Worder.ord-isom-Pre11*[*of Iod D W1*
Iod (*Iod D W2*) (*segment* (*Iod D W2*) *b*) - *g*], *assumption*+,
simp add:*Iod-carrier*, *assumption*, *simp*)

apply (*frule-tac* $a = g\ t$ **in** *Worder.segment-Expre*[*of Iod D W2* - *b*],
assumption, *simp*,
thin-tac $\neg$ *ExPre* (*Iod* (*Iod D W2*) (*segment* (*Iod D W2*) *b*)) (*g t*))
apply (*simp add*:*WWa-def Wa-def*, (*erule conjE*)+)
apply (*rotate-tac* -3,
frule-tac $x = t$ **in** *bspec*, *assumption*,
thin-tac $\forall x \in W1$.
if ExPre (*Iod D W1*) *x then f* (*Pre* (*Iod D W1*) *x*) = *x*
else if a $\neq$ *x then Sup D* (*segment* (*Iod D W1*) *x*) = *x else a* = *a*)
apply (*rotate-tac* 1,
frule-tac $x = g\ t$ **in** *bspec*, *assumption*,
thin-tac $\forall x \in W2$.
if ExPre (*Iod D W2*) *x then f* (*Pre* (*Iod D W2*) *x*) = *x*
else if a $\neq$ *x then Sup D* (*segment* (*Iod D W2*) *x*) = *x else a* = *a*,

simp)
apply (*frule-tac* $t = t$ **and** $s = a$ **in** *not-sym*, *thin-tac* $t \neq a$)
apply (*frule-tac* $b = t$ **in** *Order.ord-isom-inj*[*of Iod D W1*
Iod (*Iod D W2*) (*segment* (*Iod D W2*) *b*) *g a*], *assumption*+,
simp add:*Iod-carrier*, *simp add*:*Iod-carrier*, *simp*)
apply (*frule-tac t1* $= t$ **in** *fixp-same*[*THEN sym*, *of W1 W2 - b g*],
assumption+, *simp*, *simp add*:*fixp-def*)
apply (*rule ballI*,
frule-tac $x = x$ **in** *bspec*,
simp add:*subsetD*[*of W1 carrier D*],
simp add:*Iod-carrier fixp-def*)
apply (*simp add*:*Worder.Order*, *assumption*)
done

lemma (**in** *Order*) *BNTr4-2*:⟦$f \in$ *carrier D* $\to$ *carrier D*; $a \in$ *carrier D*;
$b \in$ *carrier D*; $\forall x \in$*carrier D*. $x \preceq (f\ x)$; *W1* $\in$ *WWa D f a*; *W2* $\in$ *WWa D f a*;
ord-equiv (*Iod D W1*) (*Iod D* (*segment* (*Iod D W2*) *b*))⟧ $\Longrightarrow$
W1 $=$ *segment* (*Iod D W2*) *b*
apply (*simp add*:*ord-equiv-def*,
erule exE)
apply (*rename-tac g*)
apply (*frule mem-wwa-Worder*[*of W1 f a*],
frule mem-wwa-Worder[*of W2 f a*],
frule Worder.Order[*of Iod D W2*],
frule mem-WWa-sub-carrier[*of W2 f a*],
frule mem-WWa-sub-carrier[*of W1 f a*],
cut-tac $a = b$ **in** *Worder.segment-Worder*[*of Iod D W2*], *assumption*)
apply (*frule Worder.Order*[*of Iod D W1*])
apply (*frule-tac* $D =$ *Iod* (*Iod D W2*) (*segment* (*Iod D W2*) *b*) **in** *Worder.Order*)
apply (*frule Worder.Order*[*of Iod D W2*])
apply (*frule-tac* $a = b$ **in** *Order.segment-sub*[*of Iod D W2*],
simp add:*Iod-carrier*)
apply (*frule-tac* $A =$ *segment* (*Iod D W2*) *b* **in** *subset-trans*[*of - W2*
carrier D], *assumption*+)
apply (*frule-tac* $T =$ *segment* (*Iod D W2*) *b* **in** *Iod-Order*)
apply (*frule-tac* $E =$ *Iod D* (*segment* (*Iod D W2*) *b*) **and** $f = g$ **in**
Order.ord-isom-func[*of Iod D W1*], *assumption*+)
apply (*frule-tac* $f = g$ **in** *Order.ord-isom-onto*[*of Iod D W1*
Iod D (*segment* (*Iod D W2*) *b*)], *assumption*+)
apply (*simp only*:*Iod-carrier*)
apply (*frule-tac* $b = b$ **and** $g = g$ **in** *BNTr4-1*[*of f a - W1 W2*],
assumption+)
apply (*simp add*:*image-def*)
done

lemma (**in** *Order*) *BNTr4*:⟦$f \in$ *carrier D* $\to$ *carrier D*; $a \in$ *carrier D*;
$\forall x \in$*carrier D*. $x \preceq (f\ x)$; *W1* $\in$ *WWa D f a*; *W2* $\in$ *WWa D f a*;
$\exists b \in$*carrier D*. *ord-equiv* (*Iod D W1*) (*Iod D* (*segment* (*Iod D W2*) *b*))⟧ $\Longrightarrow$

```
      W1 ⊆ W2
apply (erule bexE)
 apply (rename-tac b)
 apply (frule-tac b = b in BNTr4-2[of f a - W1 W2], assumption+)
 apply (frule mem-WWa-sub-carrier[of W2 f a],
       frule Iod-Order[of W2])
 apply (frule-tac a = b in Order.segment-sub[of Iod D W2],
       simp add:Iod-carrier)
done

lemma (in Order) Iod-same:A = B ⟹ Iod D A = Iod D B
by simp

lemma (in Order) eq-ord-equivTr:⟦ord-equiv D E; E = F⟧ ⟹ ord-equiv D F
by simp

lemma (in Order) BNTr5:⟦f ∈ carrier D → carrier D; a ∈ carrier D;
     ∀x∈carrier D. x ≼ (f x); W1 ∈ WWa D f a; W2 ∈ WWa D f a;
     ord-equiv (Iod D W1) (Iod D W2)⟧ ⟹ W1 ⊆ W2
 apply (frule mem-WWa-sub-carrier[of W1 f a],
       frule mem-WWa-sub-carrier[of W2 f a])
 apply (case-tac W2 = carrier D)
  apply simp
 apply (frule not-sym, thin-tac W2 ≠ carrier D)
 apply (frule sets-not-eq[of carrier D W2], assumption, erule bexE)
 apply (frule Iod-Order[of W2],
       frule Iod-Order[of W1],
       frule Iod-carrier[THEN sym, of W2])
       apply (frule-tac a = aa and A = W2 and B = carrier (Iod D W2) in
            eq-set-not-inc, assumption)
       apply (thin-tac W2 = carrier (Iod D W2))
  apply (frule-tac a = aa in Order.segment-free[of Iod D W2],
         assumption, simp add:Iod-carrier)
  apply (rule BNTr4[of f a W1 W2], assumption+)
  apply (frule-tac a = aa in Order.segment-free[of Iod D W2])
        apply (simp add:Iod-carrier)
  apply (simp only:Iod-carrier,
        rotate-tac −1, frule sym, thin-tac segment (Iod D W2) aa = W2)
  apply (frule-tac B = segment (Iod D W2) aa in
                Iod-same[of W2])
  apply (frule-tac F = Iod D (segment (Iod D W2) aa) in
        Order.eq-ord-equivTr[of Iod D W1 Iod D W2], assumption+)
  apply blast
done

lemma (in Order) BNTr6:⟦f ∈ carrier D → carrier D; a ∈ carrier D;
∀x∈carrier D. x ≼ (f x); W1 ∈ WWa D f a; W2 ∈ WWa D f a; W1 ⊂ W2⟧ ⟹
 (∃b∈carrier (Iod D W2). ord-equiv (Iod D W1) (Iod D (segment (Iod D W2) b)))
apply (frule mem-wwa-Worder[of W1 f a],
```

frule mem-wwa-Worder[*of W2 f a*])
apply (*frule-tac D = Iod D W1* **and** *E = Iod D W2* **in** *Worder.Word-compare*,
assumption+)
apply (*erule disjE*)
apply (*erule bexE*,
frule-tac a = aa **in** *Worder.segment-Worder*[*of Iod D W1*],
frule-tac S = Iod (*Iod D W1*) (*segment* (*Iod D W1*) *aa*) **in**
Worder-sym[*of - Iod D W2*], *assumption+*,
thin-tac ord-equiv (*Iod* (*Iod D W1*) (*segment* (*Iod D W1*) *aa*)) (*Iod D W2*),
frule BNTr4[*of f a W2 W1*], *assumption+*,
frule mem-WWa-sub-carrier[*of W1*],
simp add:*Iod-carrier*,
frule-tac c = aa **in** *subsetD*[*of W1 carrier D*], *assumption+*,
frule Iod-Order[*of W1*],
frule-tac a = aa **in** *Order.segment-sub*[*of Iod D W1*],
simp add:*Iod-carrier*)
apply (*frule-tac S = segment* (*Iod D W1*) *aa* **and** *T = W1* **in** *Iod-sub-sub*,
assumption+, *simp*, *blast*)
apply (*simp add*:*subset-contr*[*of W1 W2*])

apply (*erule disjE*)
apply (*frule Worder-sym*[*of Iod D W1 Iod D W2*], *assumption+*,
thin-tac ord-equiv (*Iod D W1*) (*Iod D W2*),
frule BNTr5[*of f a W2 W1*], *assumption+*)
apply (*simp add*:*subset-contr*[*of W1 W2*])

apply (*erule bexE*,
frule mem-WWa-sub-carrier[*of W2*],
simp add:*Iod-carrier*,
frule-tac c = b **in** *subsetD*[*of W2 carrier D*], *assumption+*,
frule Iod-Order[*of W2*],
frule-tac a = b **in** *Order.segment-sub*[*of Iod D W2*],
simp add:*Iod-carrier*,
frule-tac S = segment (*Iod D W2*) *b* **and** *T = W2* **in** *Iod-sub-sub*,
assumption+, *simp*)
apply *blast*
done

lemma (**in** *Order*) *BNTr6-1*:⟦*f* ∈ *carrier D* → *carrier D*; *a* ∈ *carrier D*;
∀ *x*∈*carrier D*. *x* ⪯ (*f x*); *W1* ∈ *WWa D f a*; *W2* ∈ *WWa D f a*; *W1* ⊂ *W2*⟧ ⟹
(∃ *b*∈*carrier* (*Iod D W2*). *W1* = (*segment* (*Iod D W2*) *b*))
by (*frule-tac BNTr6*[*of f a W1 W2*], *assumption+*, *erule bexE*,
frule mem-WWa-sub-carrier[*of W2*], *simp add*:*Iod-carrier*,
frule-tac c = b **in** *subsetD*[*of W2 carrier D*], *assumption+*,
frule-tac b = b **in** *BNTr4-2*[*of f a - W1 W2*], *assumption+*,
blast)

lemma (**in** *Order*) *BNTr7*:⟦*f* ∈ *carrier D* → *carrier D*; *a* ∈ *carrier D*;
∀ *x*∈*carrier D*. *x* ⪯ (*f x*); *W1* ∈ *WWa D f a*; *W2* ∈ *WWa D f a*⟧ ⟹

```
                W1 ⊆ W2 ∨ W2 ⊆ W1
apply (frule mem-wwa-Worder[of W1 f a],
       frule mem-wwa-Worder[of W2 f a])
 apply (frule-tac D = Iod D W1 and E = Iod D W2 in Worder.Word-compare,
        assumption+)
 apply (erule disjE, erule bexE)
apply (frule mem-WWa-sub-carrier[of W1],
       frule mem-WWa-sub-carrier[of W2],
       frule Iod-Order[of W1],
       frule Iod-Order[of W2],
       frule-tac a = aa in Order.segment-sub[of Iod D W1],
                 simp add:Iod-carrier,
       frule-tac S = segment (Iod D W1) aa and T = W1 in Iod-sub-sub,
                 assumption+, simp,
       thin-tac Iod (Iod D W1) (segment (Iod D W1) aa) =
          Iod D (segment (Iod D W1) aa),
       frule-tac A = segment (Iod D W1) aa in subset-trans[of - W1
          carrier D], assumption+,
       frule-tac T = segment (Iod D W1) aa in Iod-Order)

 apply (frule-tac D = Iod D (segment (Iod D W1) aa) in
          Order.ord-equiv-sym[of - Iod D W2], assumption+)
 apply (frule-tac c = aa in subsetD[of W1 carrier D], assumption+)
 apply (frule BNTr4[of f a W2 W1], assumption+, blast, simp)

apply (erule disjE)
 apply (simp add:BNTr5)

apply (frule mem-WWa-sub-carrier[of W2], simp add:Iod-carrier)
apply (frule BNTr4[of f a W1 W2], assumption+)
apply (erule bexE,
       frule mem-WWa-sub-carrier[of W2],
       simp add:Iod-carrier,
       frule-tac c = b in subsetD[of W2 carrier D], assumption+,
       frule Iod-Order[of W2],
       frule-tac a = b in Order.segment-sub[of Iod D W2],
       simp add:Iod-carrier,
       frule-tac S = segment (Iod D W2) b and T = W2 in Iod-sub-sub,
          assumption+, simp,
       frule-tac c = b in subsetD[of W2 carrier D], assumption+)
apply blast
apply simp
done

lemma (in Order) BNTr7-1:⟦f ∈ carrier D → carrier D; a ∈ carrier D;
       ∀ x∈carrier D. x ≼ f x; x ∈ W; W ∈ WWa D f a; xa ∈ ⋃ WWa D f a;
         xa ≺Iod D (⋃ WWa D f a) x⟧ ⟹ xa ∈ W
apply (cut-tac Union-WWa-sub-carrier[of f a],
      frule-tac X = W and C = WWa D f a and A = x in UnionI, assumption+,
```

simp del:*Union-iff add*:*Iod-less*)

apply (*simp only*:*Union-iff*[*of xa WWa D f a*], *erule bexE*, *rename-tac W'*)
apply (*frule-tac ?W1.0* = *W* **and** *?W2.0* = *W'* **in** *BNTr7*[*of f a*],
assumption+)
apply (*case-tac W'* ⊆ *W*, *simp add*:*subsetD*[*of* - *W*])

apply (*simp del*:*Union-iff*)
apply (*frule-tac B* = *W'* **in** *psubsetI*[*of W*])
apply (*rule not-sym*, *assumption*)
apply (*thin-tac W* ⊆ *W'*, *thin-tac W'* ≠ *W*)
apply (*frule-tac ?W2.0* = *W'* **in** *BNTr6-1*[*of f a W*], *assumption*+)
apply (*erule bexE*)
apply (*frule-tac W* = *W'* **in** *mem-WWa-sub-carrier*)
apply (*simp add*:*Iod-carrier*)
apply (*frule-tac c* = *b* **and** *A* = *W'* **in** *subsetD*[*of* - *carrier D*],
assumption+)
apply (*frule-tac W* = *W'* **and** *a* = *b* **and** *y* = *xa* **and** *x* = *x* **in** *segment-inc-less*,
assumption+)
done

lemma (**in** *Order*) *BNTr7-1-1*:⟦*f* ∈ *carrier D* → *carrier D*; *a* ∈ *carrier D*;
∀ *x*∈*carrier D*. *x* ≼ *f x*; *x* ∈ *W*; *W* ∈ *WWa D f a*; *xa* ∈ ⋃ *WWa D f a*;
xa ≺ *x*⟧ ⟹ *xa* ∈ *W*
apply (*cut-tac Union-WWa-sub-carrier*[*of f a*],
frule Iod-Order[*of* ⋃ *WWa D f a*],
frule Iod-less[*THEN sym*, *of* ⋃ *WWa D f a xa x*], *assumption*+,
rule UnionI[*of W WWa D f a x*], *assumption*+)
apply (*simp del*:*Union-iff*, *rule BNTr7-1*, *assumption*+)
done

lemma (**in** *Order*) *BNTr7-2*: ⟦*f* ∈ *carrier D* → *carrier D*; *a* ∈ *carrier D*;
∀ *x*∈*carrier D*. *x* ≼ *f x*; *x* ∈ ⋃ *WWa D f a*; *ExPre* (*Iod D* (⋃ *WWa D f a*)) *x* ⟧
⟹ ∀ *W*∈*WWa D f a*. (*x* ∈ *W* ⟶ *ExPre* (*Iod D W*) *x*)
apply (*cut-tac Union-WWa-sub-carrier*[*of f a*])
apply (*rule ballI*, *rule impI*)
apply (*simp only*:*ExPre-def*)
apply (*erule exE*, (*erule conjE*)+)
apply (*simp only*:*Iod-carrier*)
apply (*frule-tac X* = *W* **and** *C* = *WWa D f a* **and** *A* = *x* **in** *UnionI*, *assumption*+)
apply (*simp del*:*Union-iff*)
apply (*frule-tac x* = *W* **in** *bspec*, *assumption*,
thin-tac ∀ *y*∈*WWa D f a*.
∀ *y*∈*y*. *xa* ≺*Iod D* (⋃ *WWa D f a*) *y* ⟶ ¬ *y* ≺*Iod D* (⋃ *WWa D f a*) *x*)
apply (*frule-tac W* = *W* **and** *xa* = *xa* **in** *BNTr7-1*[*of f a x*], *assumption*+,
frule-tac W = *W* **in** *mem-WWa-sub-carrier*)
apply (*simp del*:*Union-iff add*:*Iod-less*)
apply (*subgoal-tac* ∀ *y*∈*W*. *xa* ≺*Iod D W y* ⟶ ¬ *y* ≺*Iod D W x*)

apply (*simp only*:*Iod-carrier*)
apply (*subgoal-tac xa* $\prec_{Iod\ D\ W}$ *x*, *blast*)
apply (*simp add*:*Iod-less*)

apply (*rule ballI*, *rule impI*)
apply (*frule-tac x* = *y* **in** *bspec*, *assumption*,
thin-tac $\forall y{\in}W.\ xa \prec_{Iod\ D\ (\bigcup WWa\ D\ f\ a)} y \longrightarrow \neg\ y \prec_{Iod\ D\ (\bigcup WWa\ D\ f\ a)} x$,
frule-tac X = *W* **and** *C* = *WWa D f a* **and** *A* = *y* **in** *UnionI*, *assumption*+)
apply (*simp add*:*Iod-less*)
done

lemma (**in** *Order*) *BNTr7-3*: $[\![f \in carrier\ D \to carrier\ D;\ a \in carrier\ D;$
$\forall x{\in}carrier\ D.\ x \preceq f\ x;\ x \in \bigcup WWa\ D\ f\ a;\ ExPre\ (Iod\ D\ (\bigcup WWa\ D\ f\ a))\ x\ ]\!]$
$\Longrightarrow \forall W{\in}WWa\ D\ f\ a.\ (x \in W \longrightarrow Pre\ (Iod\ D\ (\bigcup WWa\ D\ f\ a))\ x = Pre\ (Iod\ D\ W)\ x)$
apply (*rule ballI*)
apply (*rule impI*)
apply (*rule-tac s* = *Pre* (*Iod D W*) *x* **in**
sym[*of* - *Pre* (*Iod D* ($\bigcup$ *WWa D f a*)) *x*],
frule-tac W = *W* **in** *mem-wwa-Worder*,
frule-tac W = *W* **in** *mem-WWa-sub-carrier*,
frule BNTr7-2[*of f a x*], *assumption*+,
frule-tac x = *W* **in** *bspec*, *assumption*,
thin-tac $\forall W{\in}WWa\ D\ f\ a.\ x \in W \longrightarrow ExPre\ (Iod\ D\ W)\ x$,
simp)

apply (*rule-tac D* = *Iod D W* **and** *a* = *x* **and**
?a1.0 = *Pre* (*Iod D* ($\bigcup$ *WWa D f a*)) *x* **in** *Worder.UniquePre*,
assumption,
simp add:*Iod-carrier*, *assumption*, *simp add*:*Iod-carrier*)
apply (*frule-tac D* = *Iod D W* **in** *Worder.Order*,
frule-tac D = *Iod D W* **and** *a* = *x* **in** *Order.Pre-element*,
simp add:*Iod-carrier*, *assumption*,
(*erule conjE*)+, *simp add*:*Iod-carrier*)
apply (*cut-tac Union-WWa-sub-carrier*[*of f a*],
frule Iod-Order[*of* $\bigcup$ *WWa D f a*],
frule-tac X = *W* **and** *A* = *x* **in** *UnionI*[*of* - *WWa D f a*], *assumption*+,
frule-tac D = *Iod D* ($\bigcup$ *WWa D f a*) **and** *a* = *x* **in** *Order.Pre-element*,
simp del:*Union-iff add*:*Iod-carrier*, *assumption*)
apply (*erule conjE*)+
apply (*frule-tac W* = *W* **and** *xa* = *Pre* (*Iod D* ($\bigcup$ *WWa D f a*)) *x* **in**
BNTr7-1[*of f a x*], *assumption*+,
simp only:*Iod-carrier*, *assumption*,
simp del:*Union-iff*)

apply (*rule conjI*,
simp del:*Union-iff add*:*Iod-carrier Iod-less*,
rule ballI,

frule-tac $X = W$ **and** $A = y$ **in** *UnionI*[*of* - *WWa D f a*], *assumption*+,
thin-tac $\forall y{\in}W.\ Pre\ (Iod\ D\ W)\ x \prec_{Iod\ D\ W} y \longrightarrow \neg\ y \prec_{Iod\ D\ W} x$)
apply (*simp only*:*Iod-carrier*,
frule-tac $x = y$ **in** *bspec*, *assumption*,
thin-tac $\forall y{\in}\bigcup WWa\ D\ f\ a.$
$Pre\ (Iod\ D\ (\bigcup WWa\ D\ f\ a))\ x \prec_{Iod\ D\ (\bigcup WWa\ D\ f\ a)} y \longrightarrow$
$\neg\ y \prec_{Iod\ D\ (\bigcup WWa\ D\ f\ a)} x$)
apply (*simp del*:*Union-iff add*:*Iod-less*)
done

lemma (**in** *Order*) *BNTr7-4*:$[\![f \in carrier\ D \to carrier\ D;\ a \in carrier\ D;$
$\forall x{\in}carrier\ D.\ x \preceq f\ x;\ x \in W;\ \ W \in WWa\ D\ f\ a]\!] \Longrightarrow$
$ExPre\ (Iod\ D\ (\bigcup WWa\ D\ f\ a))\ x = ExPre\ (Iod\ D\ W)\ x$
apply (*rule iffI*)
apply (*frule BNTr7-2*[*of f a x*], *assumption*+)
apply (*frule-tac* $X = W$ **and** $A = x$ **in** *UnionI*[*of* - *WWa D f a*], *assumption*+,
simp)

apply (*simp only*:*ExPre-def*, *erule exE*, (*erule conjE*)+)
apply (*cut-tac Union-WWa-sub-carrier*[*of f a*])
apply (*frule mem-WWa-sub-carrier*[*of W*], *simp only*:*Iod-carrier*,
frule Iod-Order[*of W*]) **apply** (
frule Iod-Order[*of* $\bigcup WWa\ D\ f\ a$]) **apply** (
simp only:*Iod-less*)
apply (*frule-tac* $X = W$ **and** $A = xa$ **in** *UnionI*[*of* - *WWa D f a*], *assumption*+,
frule-tac $X = W$ **and** $A = x$ **in** *UnionI*[*of* - *WWa D f a*], *assumption*+)
apply (*frule-tac* $T1 = \bigcup WWa\ D\ f\ a$ **and** $a1 = xa$ **and** $b1 = x$ **in**
Iod-less[*THEN sym*], *assumption*+)
apply (*subgoal-tac* $\neg\ (\exists y{\in}\bigcup WWa\ D\ f\ a.$
$xa \prec_{Iod\ D\ (\bigcup WWa\ D\ f\ a)} y \wedge y \prec_{Iod\ D\ (\bigcup WWa\ D\ f\ a)} x)$)
apply *blast*

apply (*rule contrapos-pp*, (*simp del*:*Union-iff*)+)
apply (*erule bexE*, *rename-tac xa W'*, *erule bexE*, *erule conjE*,
frule-tac $X = W'$ **and** $A = y$ **in** *UnionI*[*of* - *WWa D f a*], *assumption*+)
apply (*frule-tac* $xa = y$ **in** *BNTr7-1*[*of f a x W*], *assumption*+)
apply (*frule-tac* $x = y$ **in** *bspec*, *assumption*+,
thin-tac $\forall y{\in}W.\ xa \prec_{Iod\ D\ W} y \longrightarrow \neg\ y \prec_{Iod\ D\ W} x$)
apply (*simp add*:*Iod-less*)
done

lemma (**in** *Order*) *BNTr7-5*: $[\![f \in carrier\ D \to carrier\ D;\ a \in carrier\ D;$
$\forall x{\in}carrier\ D.\ x \preceq f\ x;\ x \in W;\ W \in WWa\ D\ f\ a]\!]$
$\Longrightarrow (segment\ (Iod\ D\ (\bigcup WWa\ D\ f\ a))\ x) = segment\ (Iod\ D\ W)\ x$
apply (*cut-tac Union-WWa-sub-carrier*[*of f a*])
apply (*frule-tac* $W = W$ **in** *mem-WWa-sub-carrier*)
apply (*frule Iod-Order*[*of* $\bigcup WWa\ D\ f\ a$],
frule Iod-Order[*of W*])
apply (*rule equalityI*)

apply (*rule subsetI*,
frule Order.segment-sub[*of Iod D* ($\bigcup$ *WWa D f a*) *x*],
frule-tac c = *xa* **in** *subsetD*[*of segment* (*Iod D* ($\bigcup$ *WWa D f a*)) *x*
carrier (*Iod D* ($\bigcup$ *WWa D f a*))], *assumption*+,
frule-tac X = *W* **and** *A* = *x* **in** *UnionI*[*of* - *WWa D f a*], *assumption*+)
apply (*frule-tac a1* = *xa* **in** *Order.segment-inc*[*THEN sym*,
of Iod D ($\bigcup$ *WWa D f a*) - *x*],
assumption, *simp add*:*Iod-carrier*, *simp del*:*Union-iff*,
frule-tac xa = *xa* **in** *BNTr7-1*[*of f a x*], *assumption*+,
simp only:*Iod-carrier*, *assumption*, *simp only*:*Iod-carrier*,
simp only:*Iod-less*,
frule-tac a1 = *xa* **in** *Iod-less*[*THEN sym*, *of W* - *x*],
assumption+, *simp del*:*Union-iff*,
subst Order.segment-inc[*THEN sym*, *of Iod D W* - *x*], *assumption*+,
simp add:*Iod-carrier*, *simp add*:*Iod-carrier*, *assumption*)

apply (*rule subsetI*,
frule-tac a1 = *xa* **in** *Order.segment-inc*[*THEN sym*, *of Iod D W* - *x*])
apply (*frule Order.segment-sub*[*of Iod D W x*],
rule-tac c = *xa* **in** *subsetD*[*of segment* (*Iod D W*) *x carrier* (*Iod D W*)],
assumption+, *simp only*:*Iod-carrier*,
frule-tac W = *W* **in** *mem-WWa-sub-carrier*, *frule Iod-Order*[*of W*],
frule Order.segment-sub[*of Iod D W x*], *simp only*:*Iod-carrier*,
frule-tac c = *xa* **in** *subsetD*[*of segment* (*Iod D W*) *x W*],
simp add:*Iod-less*)

apply (*frule-tac X* = *W* **and** *A* = *xa* **in** *UnionI*[*of* - *WWa D f a*], *assumption*+,
frule-tac X = *W* **and** *A* = *x* **in** *UnionI*[*of* - *WWa D f a*], *assumption*+,
subst Order.segment-inc[*THEN sym*, *of Iod D* ($\bigcup$ *WWa D f a*) - *x*],
assumption+, *simp only*:*Iod-carrier*, *simp only*:*Iod-carrier*,
simp only:*Iod-less*)
done

lemma (**in** *Order*) *BNTr7-6*:⟦*f* ∈ *carrier D* → *carrier D*;
a ∈ *carrier D*; ∀ *x*∈*carrier D*. *x* ≼ (*f x*)⟧ ⟹ *a* ∈ $\bigcup$ *WWa D f a*
apply (*frule BNTr2*[*of f a*], *assumption*+,
frule UnionI[*of* {*a*} *WWa D f a a*])
apply (*simp*, *assumption*)
done

lemma (**in** *Order*) *BNTr7-7*:⟦*S-inductive-set D*; *f* ∈ *carrier D* → *carrier D*;
a ∈ *carrier D*; ∀ *x*∈*carrier D*. *x* ≼ (*f x*); ∃ *xa*. *Wa D xa f a* ∧ *x* ∈ *xa*⟧ ⟹
x ∈ $\bigcup$ *WWa D f a*
apply (*subst Union-iff*[*of x WWa D f a*])
apply (*subst WWa-def*, *blast*)
done

lemma (**in** *Order*) *BNTr7-8*:⟦*S-inductive-set D*; *f* ∈ *carrier D* → *carrier D*; *a* ∈
carrier D; ∀ *x*∈*carrier D*. *x* ≼ (*f x*); ∃ *xa*. *Wa D xa f a* ∧ *x* ∈ *xa*⟧ ⟹ *x* ∈ *carrier*

```
D
apply (erule exE) apply (rename-tac W, erule conjE)
 apply (rule-tac A = W and B = carrier D and c = x in subsetD,
       rule-tac W = W in mem-WWa-sub-carrier[of - f a])
 apply (simp add:WWa-def, assumption)
done


lemma (in Order) BNTr7-9:⟦f ∈ carrier D → carrier D; a ∈ carrier D;
        ∀ x∈carrier D. x ≼ (f x); x ∈ ⋃ WWa D f a ⟧ ⟹ x ∈ carrier D
by (cut-tac Union-WWa-sub-carrier[of f a],
      rule subsetD[of ⋃ WWa D f a carrier D x], assumption+)

lemma (in Order) BNTr7-10:⟦S-inductive-set D; f ∈ carrier D → carrier D;
     a ∈ carrier D; ∀ x∈carrier D. x ≼ (f x); W ∈ WWa D f a; Sup D W ∉ W⟧
⟹
            ¬ ExPre (Iod D (insert (Sup D W) W)) (Sup D W)
apply (frule mem-WWa-sub-carrier[of W f a])
apply (frule mem-WWa-Chain[of W f a],
      frule S-inductive-sup-mem[of W], assumption+,
      frule insert-sub[of W carrier D Sup D W], assumption+,
      cut-tac insertI1[of Sup D W W],
      cut-tac subset-insertI[of W Sup D W],
      frule insert-sub[of W carrier D Sup D W], assumption+)
apply (rule contrapos-pp, simp+)
apply (simp add:ExPre-def)
apply (erule exE, (erule conjE)+)
apply (frule forball-contra[of carrier (Iod D (insert (Sup D W) W)) - - -
      op = (Sup D W)],
      thin-tac ∀ y∈carrier (Iod D (insert (Sup D W) W)).
          x ≺Iod D (insert (Sup D W) W) y ⟶
          ¬ y ≺Iod D (insert (Sup D W) W) Sup D W)
apply (frule well-ord-adjunction[of W Sup D W], assumption+,
      simp add:S-inductive-sup-bound[of W],
      simp add:mem-wwa-Worder[of W],
      frule insert-sub[of W carrier D Sup D W], assumption+,
           frule Worder.Torder[of Iod D (W ∪ {Sup D W})], simp,
      frule insert-sub[of W carrier D Sup D W], assumption+,
           frule Torder-Chain[of insert (Sup D W) W], assumption+,
      frule S-inductive-sup-bound[of insert (Sup D W) W], assumption+)
      apply (simp only:Iod-carrier)
apply (rule ballI)
apply (rotate-tac −2,
      frule-tac x = y in bspec, assumption+,
      thin-tac ∀ x∈insert (Sup D W) W. x ≼ Sup D (insert (Sup D W) W),
      cut-tac insertI1[of Sup D W W],
      simp only:Iod-less)
apply (simp add:Sup-adjunct-Sup, erule disjE)
     apply (frule sym, thin-tac y = Sup D W, simp)
```

apply (*frule-tac* $c = y$ **in** *subsetD*[*of W carrier D*], *assumption*+)
apply (*frule-tac* $a = y$ **and** $b = Sup\ D\ W$ **in** *le-imp-less-or-eq*, *assumption*+,
simp) **apply** (
thin-tac $y \preceq Sup\ D\ W$,
thin-tac $x = Sup\ D\ W \vee x \in W$)
apply (*erule disjE*, *simp*, *simp*)

apply (*thin-tac* $\forall y{\in}carrier\ (Iod\ D\ (insert\ (Sup\ D\ W)\ W)).$
$x \prec_{Iod\ D\ (insert\ (Sup\ D\ W)\ W)} y \longrightarrow$
$\neg\ y \prec_{Iod\ D\ (insert\ (Sup\ D\ W)\ W)} Sup\ D\ W$)
apply (*frule insert-sub*[*of W carrier D Sup D W*], *assumption*+,
cut-tac insertI1[*of Sup D W W*],
simp only:*Iod-carrier Iod-less*)
apply (*frule mem-wwa-Worder*[*of W*],
frule S-inductive-sup-bound[*of W*], *assumption*+,
frule well-ord-adjunction[*of W Sup D W*], *assumption*+,
frule Worder.Torder[*of Iod D* $(W \cup \{Sup\ D\ W\})$], *simp*)
apply (*frule forball-contra1*,
thin-tac $\forall y{\in}W.\ x \prec_{Iod\ D\ (insert\ (Sup\ D\ W)\ W)} y \longrightarrow Sup\ D\ W = y$)
apply (*rule ballI*)
apply (*rule contrapos-pp*, *simp*+)
apply (*frule-tac* $b = x$ **in** *S-inductive-Sup-min-bounds*[*of W*], *assumption*+)
apply (*simp add*:*upper-bound-def*)
apply (*erule disjE*, *simp add*:*oless-def*)
apply (*simp add*:*subsetD*[*of W carrier D*])
apply (*rule ballI*)
apply (*thin-tac* $\forall x{\in}carrier\ D.\ x \preceq f\ x$,
thin-tac $\forall x{\in}W.\ x \preceq Sup\ D\ W$)
apply (*frule-tac* $x = s$ **in** *bspec*, *assumption*+,
thin-tac $\forall y{\in}W.\ \neg\ x \prec_{Iod\ D\ (insert\ (Sup\ D\ W)\ W)} y$)
apply (*frule-tac* $c = x$ **in** *subsetD*[*of W insert* $(Sup\ D\ W)\ W$], *assumption*+,
frule-tac $c = s$ **in** *subsetD*[*of W insert* $(Sup\ D\ W)\ W$], *assumption*+)
apply (*simp add*:*Iod-not-less-le Iod-le*)
apply (*erule disjE*, *simp add*:*oless-def*,
frule-tac $c = x$ **in** *subsetD*[*of W insert* $(Sup\ D\ W)\ W$], *assumption*+,
cut-tac insertI1[*of Sup D W W*],
frule insert-sub[*of W carrier D Sup D W*], *assumption*+)
apply (*frule-tac* $a1 = Sup\ D\ W$ **and** $b1 = x$ **in** *Iod-le*[*THEN sym*,
of insert $(Sup\ D\ W)\ W$], *assumption*+, *simp*)
apply (*frule-tac* $c = x$ **in** *subsetD*[*of W insert* $(Sup\ D\ W)\ W$], *assumption*+,
cut-tac insertI1[*of Sup D W W*],
frule insert-sub[*of W carrier D Sup D W*], *assumption*+)
apply (*simp only*:*Iod-not-less-le*[*THEN sym*])
apply (*frule-tac* $c = x$ **in** *subsetD*[*of W insert* $(Sup\ D\ W)\ W$], *assumption*+,
cut-tac insertI1[*of Sup D W W*],
frule insert-sub[*of W carrier D Sup D W*], *assumption*+)
apply (*simp add*:*Iod-less*)
done

lemma (**in** *Order*) *BNTr7-11*:⟦*S-inductive-set D*; $f \in$ *carrier D* $\to$ *carrier D*;
$a \in$ *carrier D*; $b \in$ *carrier D*; $\forall x{\in}$*carrier D*. $x \preceq f\ x$; $W \in$ *WWa D f a*;
$\forall x{\in}W.\ x \preceq b$; $x \in W$⟧ $\Longrightarrow$
ExPre (*Iod D* (*insert b W*)) *x* = *ExPre* (*Iod D W*) *x*
apply (*case-tac* $b \in W$,
simp add:*insert-absorb*[*of b W*])
apply (*frule mem-WWa-sub-carrier*[*of W*],
frule subsetD[*of W carrier D x*], *assumption*+,
frule mem-wwa-Worder[*of W*],
frule mem-WWa-Chain[*of W f a*],
frule well-ord-adjunction[*of W b*], *assumption*+,
frule insert-sub[*of W carrier D b*], *assumption*+,
cut-tac insertI1[*of b W*],
cut-tac subset-insertI[*of W b*])
apply (*simp del*:*insert-iff insert-subset add*:*Un-commute*,
frule Worder.Torder[*of Iod D* (*insert b W*)])
apply (*rule iffI*)
apply (*frule subsetD*[*of W insert b W x*], *assumption*+,
frule Iod-Order[*of insert b W*],
unfold ExPre-def)
apply (*erule exE*, (*erule conjE*)+)
apply (*rule contrapos-pp*, (*simp del*:*insert-iff insert-subset*)+)
apply (*frule-tac* $a = xa$ **in** *forall-spec*,
thin-tac $\forall xa.\ xa \prec_{Iod\ D\ W} x \longrightarrow xa \in$ *carrier* (*Iod D W*) $\longrightarrow$
($\exists y{\in}$*carrier* (*Iod D W*). $xa \prec_{Iod\ D\ W} y \wedge y \prec_{Iod\ D\ W} x$))
apply (*simp only*:*Iod-carrier*)
apply (*simp del*:*insert-subset*) **apply** (*cut-tac insertI1*[*of b W*])
apply (*erule disjE*) **apply** (*simp del*:*insert-iff insert-subset*)
apply (*thin-tac* $\forall x{\in}$*carrier D*. $x \preceq f\ x$)
apply (*frule-tac* $x = x$ **in** *bspec*, *assumption*,
thin-tac $\forall x{\in}W.\ x \preceq b$)
apply (*frule subsetD*[*of W insert b W x*], *assumption*+,
frule Iod-carrier[*THEN sym*, *of insert b W*],
frule eq-set-inc[*of x insert b W carrier* (*Iod D* (*insert b W*))],
assumption,
frule eq-set-inc[*of b insert b W carrier* (*Iod D* (*insert b W*))],
assumption,
frule Torder.not-le-less[*THEN sym*, *of Iod D* (*insert b W*) *x b*],
assumption+,
thin-tac insert b W = *carrier* (*Iod D* (*insert b W*)))
apply (*simp del*:*insert-iff insert-subset add*:*Iod-le*)
apply (*frule-tac* $c = xa$ **in** *subsetD*[*of W insert b W*], *assumption*+,
frule subsetD[*of W insert b W x*], *assumption*+,
simp only:*Iod-less*, *simp only*:*Iod-carrier*)
apply (*cut-tac* $a = xa$ **in** *insert-iff*[*of - b W*],
simp del:*insert-iff insert-subset*)
apply (*erule disjE*)
apply (*frule Iod-carrier*[*THEN sym*, *of insert b W*])
apply (*frule-tac* $a = b$ **in** *eq-set-inc*[*of - insert b W*

```
                carrier (Iod D (insert b W))], assumption+,
        frule-tac a = x in eq-set-inc[of - insert b W
                carrier (Iod D (insert b W))], assumption+,
        thin-tac insert b W = carrier (Iod D (insert b W)))
  apply (simp del:insert-iff insert-subset add:Torder.not-le-less[THEN sym,
          of Iod D (insert b W) x b])
  apply (rotate-tac 5,
        frule-tac x = x in bspec, assumption,
        thin-tac ∀ x∈W. x ≼ b, simp add:Iod-le)
  apply (thin-tac xa ∈ insert b W, erule conjE,
        simp del:insert-iff insert-subset,
        thin-tac ∀ xa. xa ≺Iod D W x ⟶
             xa ∈ W ⟶ (∃ y∈W. xa ≺Iod D W y ∧ y ≺Iod D W x))
  apply (erule bexE, erule conjE)
  apply (thin-tac ∀ x∈carrier D. x ≼ f x,
        thin-tac ∀ x∈W. x ≼ b,
        frule-tac x = y in bspec, assumption,
        thin-tac ∀ y∈W. xa ≺Iod D (insert b W) y ⟶ ¬ y ≺Iod D (insert b W) x,
        frule-tac c = xa in subsetD[of W insert b W], assumption+,
        frule-tac c = y in subsetD[of W insert b W], assumption+)
  apply (simp del:insert-iff insert-subset add:Iod-less)

apply (erule exE, (erule conjE)+)
 apply (rule contrapos-pp, (simp del:insert-iff insert-subset)+)
 apply (frule-tac a = xa in forall-spec,
       simp only:Iod-carrier,
       frule-tac c = xa in subsetD[of W insert b W], assumption+,
       frule-tac c = x in subsetD[of W insert b W], assumption+,
       simp add:Iod-less)
 apply (simp only:Iod-carrier,
       frule-tac c = xa in subsetD[of W insert b W], assumption,
       simp del:insert-iff insert-subset add:Iod-less)
 apply (erule disjE, erule conjE) apply (
       frule-tac a = xa in forall-spec,
       thin-tac ∀ xa. xa ≺Iod D (insert b W) x ⟶ xa ∈ insert b W ⟶
     xa ≺ b ∧ b ≺Iod D (insert b W) x ∨ (∃ y∈W. xa ≺ y ∧ y ≺Iod D (insert b W)
x))
 apply (thin-tac ∀ x∈carrier D. x ≼ f x,
       frule-tac x = x in bspec, assumption,
       thin-tac ∀ x∈W. x ≼ b,
       frule-tac c = xa in subsetD[of W insert b W], assumption+,
       frule-tac c = x in subsetD[of W insert b W], assumption+,
       simp only:Iod-less)
 apply (thin-tac ∀ xa. xa ≺Iod D (insert b W) x ⟶ xa ∈ insert b W ⟶
     xa ≺ b ∧ b ≺Iod D (insert b W) x ∨ (∃ y∈W. xa ≺ y ∧ y ≺Iod D (insert b W)
x))
 apply (simp del:insert-iff insert-subset)
 apply (thin-tac ∀ x∈carrier D. x ≼ f x,
```

frule-tac $x = x$ **in** *bspec*, *assumption*,
thin-tac $\forall x{\in}W.\ x \preceq b$,
frule-tac subsetD[*of W insert b W x*], *assumption*+,
frule-tac $c = xa$ **in** *subsetD*[*of W insert b W*], *assumption*+,
frule Iod-carrier[*THEN sym*, *of insert b W*],
frule eq-set-inc[*of x insert b W carrier* (*Iod D* (*insert b W*))],
assumption,
frule eq-set-inc[*of b insert b W carrier* (*Iod D* (*insert b W*))],
assumption,
frule Torder.not-le-less[*THEN sym*, *of Iod D* (*insert b W*) *x b*],
assumption+,
thin-tac insert b W = *carrier* (*Iod D* (*insert b W*)),
simp add:*Iod-le*,
thin-tac $\forall xa.\ xa \prec_{Iod\ D\ (insert\ b\ W)} x \longrightarrow xa \in insert\ b\ W \longrightarrow$
$xa \prec b \wedge b \prec_{Iod\ D\ (insert\ b\ W)} x \vee (\exists y{\in}W.\ xa \prec y \wedge y \prec_{Iod\ D\ (insert\ b\ W)} x)$)
apply (*erule bexE*, *erule conjE*,
thin-tac $\forall x{\in}W.\ x \preceq b$,
frule-tac $x = y$ **in** *bspec*, *assumption*,
thin-tac $\forall y{\in}W.\ xa \prec y \longrightarrow \neg\ y \prec x$,
frule-tac $c = y$ **in** *subsetD*[*of W insert b W*], *assumption*+,
frule-tac $c = x$ **in** *subsetD*[*of W insert b W*], *assumption*+,
simp add:*Iod-less*)
done

lemma (**in** *Order*) *BNTr7-12*:$[\![$*S-inductive-set D*; $f \in carrier\ D \to carrier\ D$;
$a \in carrier\ D$; $b \in carrier\ D$; $\forall x{\in}carrier\ D.\ x \preceq f\ x$; $W \in WWa\ D\ f\ a$;
$\forall x{\in}W.\ x \preceq b$; $x \in W$; *ExPre* (*Iod D W*) $x]\!] \Longrightarrow$
Pre (*Iod D* (*insert b W*)) x = *Pre* (*Iod D W*) x
apply (*case-tac* $b \in W$, *simp only*:*insert-absorb*)
apply (*frule mem-WWa-Chain*[*of W*],
frule mem-wwa-Worder[*of W*],
frule mem-WWa-sub-carrier[*of W*],
frule well-ord-adjunction[*of W b*], *assumption*+,
simp add:*Un-commute*[*of W* {*b*}],
frule insert-sub[*of W carrier D b*], *assumption*+,
cut-tac subset-insertI[*of W b*],
frule subsetD[*of W insert b W x*], *assumption*+,
cut-tac insertI1[*of b W*])
apply (*rule Worder.UniquePre*[*of Iod D* (*insert b W*) *x*
Pre (*Iod D W*) *x*], *assumption*+,
simp add:*Iod-carrier*,
subst BNTr7-11[*of f a b W x*], *assumption*+,
frule Worder.Order[*of Iod D W*])
apply (*frule Order.Pre-element*[*of Iod D W x*],
simp add:*Iod-carrier*, *assumption*)
apply (*erule conjE*)+
apply (*rule conjI*)
apply (*simp add*:*Iod-carrier*)

apply (*rule conjI*)
 apply (*simp only*:*Iod-carrier*,
 frule subsetD[*of W insert b W x*], *assumption*+,
 frule subsetD[*of W insert b W Pre* (*Iod D W*) *x*],
 assumption+)
 apply (*simp only*:*Iod-less*)
apply (*rule contrapos-pp*, (*simp del*:*insert-iff insert-subset*)+)
 apply (*erule bexE*)
 apply (*simp only*:*Iod-carrier*,
 frule subsetD[*of W insert b W Pre* (*Iod D W*) *x*],
 assumption+)
 apply (*cut-tac a* = *y* **in** *insert-iff*[*of - b W*])
 apply (*frule-tac P* = *y* ∈ *insert b W* **and** *Q* = *y* = *b* ∨ *y* ∈ *W* **in**
 eq-prop, *assumption*+,
 thin-tac (*y* ∈ *insert b W*) = (*y* = *b* ∨ *y* ∈ *W*),
 thin-tac y ∈ *insert b W*)
 apply (*erule disjE*,
 thin-tac ∀ *x*∈*carrier D*. *x* ≼ *f x*,
 frule-tac x = *x* **in** *bspec*, *assumption*,
 thin-tac ∀ *x*∈*W*. *x* ≼ *b*, *erule conjE*,
 frule Iod-carrier[*THEN sym*, *of insert b W*],
 frule eq-set-inc[*of x insert b W carrier* (*Iod D* (*insert b W*))],
 assumption,
 frule eq-set-inc[*of b insert b W carrier* (*Iod D* (*insert b W*))],
 assumption,
 thin-tac insert b W = *carrier* (*Iod D* (*insert b W*)),
 simp del:*insert-iff insert-subset*,
 frule Worder.Torder[*of Iod D* (*insert b W*)],
 frule Torder.not-le-less[*THEN sym*, *of Iod D* (*insert b W*) *x b*],
 assumption+, *simp add*:*Iod-le*)
 apply (*rotate-tac* −4,
 frule-tac x = *y* **in** *bspec*, *assumption*,
 thin-tac ∀ *y*∈*W*. *Pre* (*Iod D W*) *x* ≺$_{Iod\ D\ W}$ *y* ⟶ ¬ *y* ≺$_{Iod\ D\ W}$ *x*,
 frule-tac c = *y* **in** *subsetD*[*of W insert b W*], *assumption*,
 simp add:*Iod-less*)
done

lemma (**in** *Order*) *BNTr7-13*:⟦*S-inductive-set D*; *f* ∈ *carrier D* → *carrier D*;
 a ∈ *carrier D*; *b* ∈ *carrier D*; ∀ *x*∈*carrier D*. *x* ≼ *f x*; *W* ∈ *WWa D f a*;
 ∀ *x*∈*W*. *x* ≼ *b*; *x* ∈ *W*⟧ ⟹
 (*segment* (*Iod D* (*insert b W*)) *x*) = *segment* (*Iod D W*) *x*
apply (*case-tac b* ∈ *W*, *simp add*:*insert-absorb*)
apply (*frule mem-wwa-Worder*[*of W*],
 frule mem-WWa-sub-carrier[*of W*],
 frule mem-WWa-Chain[*of W*],
 frule insert-sub[*of W carrier D b*], *assumption*+,
 frule well-ord-adjunction[*of W b*], *assumption*+,
 simp del:*insert-subset add*:*Un-commute*,
 cut-tac subset-insertI[*of W b*],

cut-tac insertI1[*of b W*],
frule Worder.Torder[*of Iod D* (*insert b W*)])
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*simp del*:*insert-iff insert-subset add*:*segment-def Iod-carrier*)
apply (*frule subsetD*[*of W insert b W x*], *assumption*+,
simp del:*insert-iff insert-subset*)
apply (*erule conjE*, *simp only*:*Iod-carrier*)
apply (*cut-tac a* = *xa* **in** *insert-iff*[*of - b W*],
frule-tac P = *xa* ∈ *insert b W* **and** *Q* = *xa* = *b* ∨ *xa* ∈ *W* **in**
eq-prop, *assumption*+)
apply (*thin-tac* (*xa* ∈ *insert b W*) = (*xa* = *b* ∨ *xa* ∈ *W*),
erule disjE, *simp del*:*insert-iff insert-subset*,
frule Iod-carrier[*THEN sym*, *of insert b W*],
frule eq-set-inc[*of x insert b W carrier* (*Iod D* (*insert b W*))],
assumption,
frule eq-set-inc[*of b insert b W carrier* (*Iod D* (*insert b W*))],
assumption,
thin-tac insert b W = *carrier* (*Iod D* (*insert b W*)),
frule Torder.not-le-less[*THEN sym*, *of Iod D* (*insert b W*) *x b*],
assumption+, *simp add*:*Iod-le*)
apply (*frule-tac c* = *xa* **in** *subsetD*[*of W insert b W*], *assumption*+,
frule-tac c = *x* **in** *subsetD*[*of W insert b W*], *assumption*+,
simp add:*Iod-less*)

apply (*rule subsetI*)
apply (*simp del*:*insert-iff insert-subset add*:*segment-def*,
simp only:*Iod-carrier*)
apply (*frule-tac subsetD*[*of W insert b W x*], *assumption*+,
simp del:*insert-iff insert-subset*)
apply (*erule conjE*,
frule-tac c = *xa* **in** *subsetD*[*of W insert b W*], *assumption*+,
simp add:*Iod-less*)
done

lemma (**in** *Order*) *BNTr7-14*:⟦*S-inductive-set D*; *f* ∈ *carrier D* → *carrier D*;
a ∈ *carrier D*; ∀ *x*∈*carrier D*. *x* ≼ (*f x*); *W* ∈ *WWa D f a*⟧ ⟹
(*insert* (*Sup D W*) *W*) ∈ *WWa D f a*
apply (*case-tac Sup D W* ∈ *W*,
simp add:*insert-absorb*[*of Sup D W W*])
apply (*frule mem-WWa-sub-carrier*[*of W f a*],
frule mem-WWa-Chain[*of W f a*],
frule S-inductive-sup-mem[*of W*], *assumption*+,
frule insert-sub[*of W carrier D Sup D W*], *assumption*+,
rule mem-of-WWa [*of insert* (*Sup D W*) *W a f*], *assumption*+,
frule S-inductive-sup-bound[*of W*], *assumption*+)
apply (*frule well-ord-adjunction*[*of W Sup D W*], *assumption*+,
simp add:*mem-wwa-Worder*, *simp*,

frule mem-WWa-inc-a[*of W f a*], *simp*)

apply (*rule ballI*)
apply (*simp add*:*WWa-def Wa-def*, (*erule conjE*)+, *erule disjE*,
frule S-inductive-sup-bound[*of W*], *assumption*+,
simp, *simp*)

apply (*rule ballI*)
apply (*simp only*:*insert-iff*)
apply (*erule disjE*)
apply (*frule mem-WWa-inc-a*[*of W f a*])
apply (*frule not-eq-outside* [*of Sup D W W*])
apply (*rotate-tac* −*1*,
frule-tac x = *a* **in** *bspec*, *assumption*+,
thin-tac ∀ *b*∈*W*. *b* ≠ *Sup D W*)
apply (*frule BNTr7-10*[*of f a W*], *assumption*+)
apply (*simp del*:*insert-iff insert-subset add*:*Adjunct-segment-eq*)
apply (*frule S-inductive-sup-bound*[*of W*], *assumption*+)
apply (*subst BNTr7-11*[*of f a Sup D W W*], *assumption*+)
apply (*case-tac ExPre* (*Iod D W*) *x*,
subst BNTr7-12[*of f a Sup D W W*], *assumption*+)
apply (*simp del*:*insert-iff insert-subset add*:*WWa-def*)
apply (*unfold Wa-def*, *simp*)
apply (*simp del*:*insert-iff insert-subset*)
apply (*rule impI*)
apply (*subst BNTr7-13*[*of f a Sup D W W*], *assumption*+)
apply (*simp add*:*WWa-def Wa-def*)
done

lemma (**in** *Order*) *BNTr7-15*:⟦*S-inductive-set D*; *f* ∈ *carrier D* → *carrier D*;
a ∈ *carrier D*; ∀ *x*∈*carrier D*. *x* ≼ (*f x*); *W* ∈ *WWa D f a*;
f (*Sup D W*) ≠ *Sup D W*⟧ ⟹
ExPre (*Iod D* (*insert* (*f* (*Sup D W*)) (*insert* (*Sup D W*) *W*))) (*f* (*Sup D W*))
apply (*simp add*:*ExPre-def*)
apply (*rule contrapos-pp*, *simp*+)
apply (*frule BNTr7-14*[*of f a W*], *assumption*+)
apply (*frule mem-WWa-sub-carrier*[*of insert* (*Sup D W*) *W f a*],
frule mem-WWa-Chain[*of insert* (*Sup D W*) *W f a*],
frule mem-wwa-Worder[*of insert* (*Sup D W*) *W f a*],
frule mem-WWa-Chain[*of W f a*],
frule S-inductive-sup-mem[*of W*], *assumption*+,
frule funcset-mem[*of f carrier D carrier D Sup D W*], *assumption*+,
frule insert-sub[*of insert* (*Sup D W*) *W carrier D f* (*Sup D W*)],
assumption+,
frule S-inductive-sup-bound[*of W*], *assumption*+)
apply (*frule well-ord-adjunction*[*of insert* (*Sup D W*) *W f* (*Sup D W*)],
assumption+,
rule ballI,
simp only:*insert-iff*, *erule disjE*, *simp del*:*insert-iff insert-subset*)

apply (*frule-tac* $x = Sup\ D\ W$ **in** *bspec*, *assumption*,
thin-tac $\forall x{\in}carrier\ D.\ x \preceq f\ x$) **apply** (
rotate-tac -3,
frule-tac $x = x$ **in** *bspec*, *assumption*,
thin-tac $\forall x{\in}W.\ x \preceq Sup\ D\ W$) **apply** (
frule mem-WWa-sub-carrier[*of W*],
frule-tac $c = x$ **in** *subsetD*[*of W carrier D*], *assumption*+,
rule-tac $a = x$ **in** *le-trans*[*of - Sup D W f* (*Sup D W*)], *assumption*+,
cut-tac insertI1[*of Sup D W W*],
cut-tac insertI1[*of f* (*Sup D W*) *insert* (*Sup D W*) *W*],
cut-tac subset-insertI[*of insert* (*Sup D W*) *W f* (*Sup D W*)],
frule subsetD[*of insert* (*Sup D W*) *W insert* (*f* (*Sup D W*))
(*insert* (*Sup D W*) *W*) *Sup D W*], *assumption*+)
apply (*frule-tac* $a = Sup\ D\ W$ **in** *forall-spec*)
apply (*frule-tac* $x = Sup\ D\ W$ **in** *bspec*, *assumption*,
thin-tac $\forall x{\in}carrier\ D.\ x \preceq f\ x$,
frule not-sym,
thin-tac $f\ (Sup\ D\ W) \neq Sup\ D\ W$,
simp del:*insert-iff insert-subset*
add:*le-imp-less-or-eq*[*of Sup D W f* (*Sup D W*)],
simp only:*Iod-less*)
apply (*thin-tac* $\forall x.\ x \prec_{Iod\ D\ (insert\ (f\ (Sup\ D\ W))\ (insert\ (Sup\ D\ W)\ W))}\ f\ (Sup\ D\ W) \longrightarrow$
$x \in carrier\ (Iod\ D\ (insert\ (f\ (Sup\ D\ W))\ (insert\ (Sup\ D\ W)\ W))) \longrightarrow$
$(\exists y{\in}carrier\ (Iod\ D\ (insert\ (f\ (Sup\ D\ W))\ (insert\ (Sup\ D\ W)\ W))).$
$x \prec_{Iod\ D\ (insert\ (f\ (Sup\ D\ W))\ (insert\ (Sup\ D\ W)\ W))}\ y\ \wedge$
$y \prec_{Iod\ D\ (insert\ (f\ (Sup\ D\ W))\ (insert\ (Sup\ D\ W)\ W))}$
$f\ (Sup\ D\ W))$,
simp only:*Iod-carrier*,
frule True-then[*of* $\exists y{\in}insert\ (f\ (Sup\ D\ W))\ (insert\ (Sup\ D\ W)\ W).$
$Sup\ D\ W \prec_{Iod\ D\ (insert\ (f\ (Sup\ D\ W))\ (insert\ (Sup\ D\ W)\ W))}\ y\ \wedge$
$y \prec_{Iod\ D\ (insert\ (f\ (Sup\ D\ W))\ (insert\ (Sup\ D\ W)\ W))}\ f\ (Sup\ D\ W)$],
thin-tac True $\longrightarrow$
$(\exists y{\in}insert\ (f\ (Sup\ D\ W))\ (insert\ (Sup\ D\ W)\ W).$
$Sup\ D\ W \prec_{Iod\ D\ (insert\ (f\ (Sup\ D\ W))\ (insert\ (Sup\ D\ W)\ W))}\ y\ \wedge$
$y \prec_{Iod\ D\ (insert\ (f\ (Sup\ D\ W))\ (insert\ (Sup\ D\ W)\ W))}\ f\ (Sup\ D\ W))$,
erule bexE, *erule conjE*)
apply (*cut-tac* $a = y$ **in** *insert-iff*[*of - f* (*Sup D W*) *insert* (*Sup D W*) *W*],
frule-tac $P = y \in insert\ (f\ (Sup\ D\ W))\ (insert\ (Sup\ D\ W)\ W)$ **and**
$Q = y = f\ (Sup\ D\ W) \vee y \in insert\ (Sup\ D\ W)\ W$ **in** *eq-prop*,
assumption+,
thin-tac $y \in insert\ (f\ (Sup\ D\ W))\ (insert\ (Sup\ D\ W)\ W)$,
thin-tac $(y \in insert\ (f\ (Sup\ D\ W))\ (insert\ (Sup\ D\ W)\ W)) =$
$(y = f\ (Sup\ D\ W) \vee y \in insert\ (Sup\ D\ W)\ W))$
apply (*erule disjE*, *simp add*:*oless-def*)
apply (*cut-tac* $a = y$ **in** *insert-iff*[*of - Sup D W W*],
frule-tac $P = y \in insert\ (Sup\ D\ W)\ W$ **and**
$Q = y = (Sup\ D\ W) \vee y \in W$ **in** *eq-prop*, *assumption*+,

thin-tac $y \in insert\ (Sup\ D\ W)\ W$,
thin-tac $y \in insert\ (Sup\ D\ W)\ W = (y = (Sup\ D\ W) \vee y \in W)$,
erule disjE, *simp add*:*oless-def*,
simp del:*insert-iff insert-subset*)
apply (*frule-tac* $x = y$ **in** *bspec*, *assumption*,
thin-tac $\forall x{\in}W.\ x \preceq Sup\ D\ W$)
apply (*frule-tac* $x = Sup\ D\ W$ **in** *bspec*, *assumption*,
thin-tac $\forall x{\in}carrier\ D.\ x \preceq f\ x$,
cut-tac subset-insertI[*of W Sup D W*],
frule-tac $c = y$ **in** *subsetD*[*of W insert* (*Sup D W*) *W*], *assumption*+,
frule-tac $c = y$ **in** *subsetD*[*of insert* (*Sup D W*) *W*
insert (*f* (*Sup D W*)) (*insert* (*Sup D W*) *W*)], *assumption*+,
frule-tac Worder.Torder[*of Iod D* (*insert* (*f* (*Sup D W*))
(*insert* (*Sup D W*) *W*))])
apply (*frule-tac a1* $= y$ **in** *Torder.not-le-less*[*THEN sym*, *of*
Iod D (*insert* (*f* (*Sup D W*)) (*insert* (*Sup D W*) *W*)) - *Sup D W*],
simp add:*Iod-carrier*, *simp add*:*Iod-carrier*)
apply (*simp add*:*Iod-le*)
done

lemma (**in** *Order*) *BNTr7-16*:⟦*S-inductive-set D*; $f \in carrier\ D \to carrier\ D$;
$a \in carrier\ D$; $\forall x{\in}carrier\ D.\ x \preceq (f\ x)$; $W \in WWa\ D\ f\ a$;
$f\ (Sup\ D\ W) \neq (Sup\ D\ W)$⟧ ⟹
Pre (*Iod D* (*insert* (*f* (*Sup D W*)) (*insert* (*Sup D W*) *W*))) (*f* (*Sup D W*))
=
(*Sup D W*)
apply (*frule BNTr7-14*[*of f a W*], *assumption*+,
frule mem-WWa-sub-carrier[*of insert* (*Sup D W*) *W f a*],
frule mem-WWa-Chain[*of insert* (*Sup D W*) *W f a*],
frule mem-wwa-Worder[*of insert* (*Sup D W*) *W f a*],
frule mem-WWa-Chain[*of W f a*],
frule S-inductive-sup-mem[*of W*], *assumption*+,
frule funcset-mem[*of f carrier D carrier D Sup D W*], *assumption*+,
frule insert-sub[*of insert* (*Sup D W*) *W carrier D f* (*Sup D W*)],
assumption+,
frule S-inductive-sup-bound[*of W*], *assumption*+,
frule well-ord-adjunction[*of insert* (*Sup D W*) *W f* (*Sup D W*)],
assumption+, *rule ballI*,
simp only:*insert-iff*, *erule disjE*, *simp del*:*insert-iff insert-subset*,
frule-tac $x = Sup\ D\ W$ **in** *bspec*, *assumption*,
thin-tac $\forall x{\in}carrier\ D.\ x \preceq f\ x$,
rotate-tac -3,
frule-tac $x = x$ **in** *bspec*, *assumption*,
thin-tac $\forall x{\in}W.\ x \preceq Sup\ D\ W$,
frule mem-WWa-sub-carrier[*of W*],
frule-tac $c = x$ **in** *subsetD*[*of W carrier D*], *assumption*+,
rule-tac $a = x$ **in** *le-trans*[*of - Sup D W f* (*Sup D W*)], *assumption*+,
cut-tac insertI1[*of Sup D W W*],
cut-tac insertI1[*of f* (*Sup D W*) *insert* (*Sup D W*) *W*],

cut-tac subset-insertI[*of insert* (*Sup D W*) *W f* (*Sup D W*)],
frule subsetD[*of insert* (*Sup D W*) *W insert* (*f* (*Sup D W*))
(*insert* (*Sup D W*) *W*) *Sup D W*], *assumption*+)
apply (*simp only*:*Un-commute*[*of insert* (*Sup D W*) *W* {*f* (*Sup D W*)}],
simp only:*insert-is-Un*[*THEN sym*])
apply (*rule Worder.UniquePre*[*of Iod D* (*insert* (*f* (*Sup D W*))
(*insert* (*Sup D W*) *W*)) *f* (*Sup D W*) *Sup D W*], *assumption*+,
simp only:*Iod-carrier*,
rule BNTr7-15, *assumption*+,
rule conjI, *simp only*:*Iod-carrier*, *rule conjI*, *simp only*:*Iod-less*,
frule-tac x = *Sup D W* **in** *bspec*, *assumption*,
thin-tac ∀ *x*∈*carrier D*. *x* ≼ *f x*, *frule not-sym*,
thin-tac f (*Sup D W*) ≠ *Sup D W*, *simp add*:*oless-def*)
apply (*rule contrapos-pp*, (*simp del*:*insert-iff insert-subset*)+,
erule bexE, *erule conjE*)
apply (*simp only*:*Iod-carrier*) **apply** (
cut-tac a = *y* **in** *insert-iff*[*of* - *f* (*Sup D W*) *insert* (*Sup D W*) *W*],
frule-tac P = *y* ∈ *insert* (*f* (*Sup D W*)) (*insert* (*Sup D W*) *W*) **and**
Q = *y* = *f* (*Sup D W*) ∨ *y* ∈ *insert* (*Sup D W*) *W* **in** *eq-prop*,
assumption,
thin-tac y ∈ *insert* (*f* (*Sup D W*)) (*insert* (*Sup D W*) *W*),
thin-tac y ∈ *insert* (*f* (*Sup D W*)) (*insert* (*Sup D W*) *W*) =
(*y* = *f* (*Sup D W*) ∨ *y* ∈ *insert* (*Sup D W*) *W*),
erule disjE, *simp add*:*oless-def*) **apply** (
cut-tac a = *y* **in** *insert-iff*[*of* - *Sup D W W*],
frule-tac P = *y* ∈ *insert* (*Sup D W*) *W* **and**
Q = *y* = (*Sup D W*) ∨ *y* ∈ *W* **in** *eq-prop*, *assumption*,
thin-tac y ∈ *insert* (*Sup D W*) *W*,
thin-tac y ∈ (*insert* (*Sup D W*) *W*) = (*y* = (*Sup D W*) ∨ *y* ∈ *W*),
erule disjE, *simp add*:*oless-def*)
apply (*frule-tac x* = *y* **in** *bspec*, *assumption*,
thin-tac ∀ *x*∈*W*. *x* ≼ *Sup D W*,
cut-tac subset-insertI[*of W Sup D W*],
frule-tac c = *y* **in** *subsetD*[*of W insert* (*Sup D W*) *W*], *assumption*,
frule Worder.Torder[*of Iod D* (*insert* (*f* (*Sup D W*))
(*insert* (*Sup D W*) *W*))])
apply (*frule mem-WWa-sub-carrier*[*of W*],
frule-tac c = *y* **in** *subsetD*[*of W carrier D*], *assumption*+,
frule-tac c = *y* **in** *subsetD*[*of insert* (*Sup D W*) *W*
insert (*f* (*Sup D W*)) (*insert* (*Sup D W*) *W*)], *assumption*+,
frule-tac a1 = *y* **in** *Torder.not-le-less*[*THEN sym*,
of Iod D (*insert* (*f* (*Sup D W*)) (*insert* (*Sup D W*) *W*)) - *Sup D W*],
simp add:*Iod-carrier*, *simp add*:*Iod-carrier*)
apply (*simp add*:*Iod-le*)
done

lemma (**in** *Order*) *BNTr7-17*:⟦*S-inductive-set D*; *f* ∈ *carrier D* → *carrier D*;
a ∈ *carrier D*; ∀ *x*∈*carrier D*. *x* ≼ (*f x*); *W* ∈ *WWa D f a*⟧ ⟹
(*insert* (*f* (*Sup D W*)) (*insert* (*Sup D W*) *W*)) ∈ *WWa D f a*

apply (*frule mem-WWa-Chain*[*of W*],
frule S-inductive-sup-mem[*of W*], *assumption*+)
apply (*case-tac f* (*Sup D W*) = *Sup D W*,
simp add:*insert-absorb*, *simp add*: *BNTr7-14*,
frule not-sym, *thin-tac f* (*Sup D W*) ≠ *Sup D W*)
apply (*frule BNTr7-14*[*of f a W*], *assumption*+,
frule mem-WWa-sub-carrier[*of insert* (*Sup D W*) *W f a*],
frule mem-WWa-Chain[*of insert* (*Sup D W*) *W f a*],
frule funcset-mem[*of f carrier D carrier D Sup D W*], *assumption*+,
frule mem-wwa-Worder[*of insert* (*Sup D W*) *W*])
apply (*frule insert-sub*[*of insert* (*Sup D W*) *W carrier D f* (*Sup D W*)],
assumption+,
rule mem-of-WWa [*of* (*insert* (*f* (*Sup D W*)) (*insert* (*Sup D W*) *W*)) *a*
f], *assumption*+)
apply (*frule well-ord-adjunction*[*of insert* (*Sup D W*) *W f* (*Sup D W*)],
assumption+,
frule S-inductive-sup-bound[*of W*], *assumption*+,
rule ballI,
simp only:*insert-iff*, *erule disjE*, *simp*,
frule-tac x = *x* **in** *bspec*, *assumption*,
thin-tac ∀ *x*∈*W*. *x* ≼ *Sup D W*,
frule-tac x = *Sup D W* **in** *bspec*, *assumption*,
thin-tac ∀ *x*∈*carrier D*. *x* ≼ *f x*,
frule mem-WWa-sub-carrier[*of W*],
frule-tac c = *x* **in** *subsetD*[*of W carrier D*], *assumption*+,
rule-tac a = *x* **in** *le-trans*[*of - Sup D W f* (*Sup D W*)],
assumption+, *simp*)
apply (*frule mem-WWa-inc-a*[*of insert* (*Sup D W*) *W f a*],
simp)
apply (*rule ballI*)
apply (*frule BNTr2-1*[*of f a insert* (*Sup D W*) *W*], *assumption*+,
cut-tac a = *x* **in** *insert-iff*[*of - f* (*Sup D W*) *insert* (*Sup D W*) *W*],
frule-tac P = *x* ∈ *insert* (*f* (*Sup D W*)) (*insert* (*Sup D W*) *W*) **and**
Q = *x* = (*f* (*Sup D W*)) ∨ *x* ∈ (*insert* (*Sup D W*) *W*) **in** *eq-prop*,
assumption+,
thin-tac x ∈ *insert* (*f* (*Sup D W*)) (*insert* (*Sup D W*) *W*),
thin-tac (*x* ∈ *insert* (*f* (*Sup D W*)) (*insert* (*Sup D W*) *W*)) =
(*x* = *f* (*Sup D W*) ∨ *x* ∈ *insert* (*Sup D W*) *W*),
erule disjE)
apply (*frule-tac x* = *Sup D W* **in** *bspec*, *assumption*,
thin-tac ∀ *x*∈*carrier D*. *x* ≼ *f x*,
frule-tac x = *Sup D W* **in** *bspec*, *simp*)
apply (
thin-tac ∀ *x*∈*insert* (*Sup D W*) *W*. *a* ≼ *x*,
simp add:*le-trans*[*of a Sup D W f* (*Sup D W*)])
apply (*frule-tac x* = *x* **in** *bspec*, *assumption*,
thin-tac ∀ *x*∈*insert* (*Sup D W*) *W*. *a* ≼ *x*,
simp)
apply (*rule ballI*)

apply (*cut-tac* $a = x$ **in** *insert-iff*[*of* - f (*Sup* D W) *insert* (*Sup* D W) W],
frule-tac $P = x \in$ *insert* (f (*Sup* D W)) (*insert* (*Sup* D W) W) **and**
$Q = x =$ (f (*Sup* D W)) $\vee$ $x \in$ (*insert* (*Sup* D W) W) **in** *eq-prop*,
assumption+,
thin-tac $x \in$ *insert* (f (*Sup* D W)) (*insert* (*Sup* D W) W),
thin-tac ($x \in$ *insert* (f (*Sup* D W)) (*insert* (*Sup* D W) W)) =
($x = f$ (*Sup* D W) $\vee$ $x \in$ *insert* (*Sup* D W) W),
erule disjE)
apply (*frule not-sym*, *thin-tac Sup* D $W \neq f$ (*Sup* D W),
frule BNTr7-15[*of* f a W], *assumption*+,
simp del:*insert-iff insert-subset*)
apply (*subst BNTr7-16*[*of* f a W], *assumption*+, *simp*,
frule S-inductive-sup-bound[*of* W], *assumption*+)
apply (*subst BNTr7-11*[*of* f a f (*Sup* D W)
insert (*Sup* D W) W], *assumption*+,
rule ballI,
thin-tac $x \in$ *insert* (*Sup* D W) W, *simp only*:*insert-iff*,
erule disjE,
frule-tac $x =$ *Sup* D W **in** *bspec*, *assumption*,
thin-tac $\forall x \in$*carrier* D. $x \preceq f\, x$,
simp add:*le-antisym*,
frule-tac $x = xa$ **in** *bspec*, *assumption*+,
thin-tac $\forall x \in W$. $x \preceq$ *Sup* D W,
frule-tac $x =$ *Sup* D W **in** *bspec*, *assumption*,
thin-tac $\forall x \in$*carrier* D. $x \preceq f\, x$,
frule mem-WWa-sub-carrier[*of* W],
frule-tac $c = xa$ **in** *subsetD*[*of* W *carrier* D], *assumption*+,
rule-tac $a = xa$ **in** *le-trans*[*of* - *Sup* D W f (*Sup* D W)],
assumption+)
apply (*case-tac ExPre* (*Iod* D (*insert* (*Sup* D W) W)) x,
simp del:*insert-iff insert-subset*,
subst BNTr7-12[*of* f a f (*Sup* D W) *insert* (*Sup* D W) W],
assumption+,
rule ballI, *thin-tac* $x \in$ *insert* (*Sup* D W) W,
simp only:*insert-iff*, *erule disjE*,
frule-tac $x =$ *Sup* D W **in** *bspec*, *assumption*,
thin-tac $\forall x \in$*carrier* D. $x \preceq f\, x$,
simp add:*le-antisym*,
frule-tac $x = xa$ **in** *bspec*, *assumption*+,
thin-tac $\forall x \in W$. $x \preceq$ *Sup* D W,
frule-tac $x =$ *Sup* D W **in** *bspec*, *assumption*,
thin-tac $\forall x \in$*carrier* D. $x \preceq f\, x$,
frule mem-WWa-sub-carrier[*of* W],
frule-tac $c = xa$ **in** *subsetD*[*of* W *carrier* D], *assumption*+,
rule-tac $a = xa$ **in** *le-trans*[*of* - *Sup* D W f (*Sup* D W)],
assumption+)
apply (*frule mem-WWa-then*[*of insert* (*Sup* D W) W f a],
(*erule conjE*)+,
thin-tac $\forall x \in$*insert* (*Sup* D W) W. $a \preceq x$,

```
        frule-tac x = x in bspec, assumption,
        thin-tac ∀x∈insert (Sup D W) W.
       if ExPre (Iod D (insert (Sup D W) W)) x
       then f (Pre (Iod D (insert (Sup D W) W)) x) = x
       else if a ≠ x
            then Sup D (segment (Iod D (insert (Sup D W) W)) x) = x
            else a = a, simp)
  apply (simp del:insert-iff insert-subset, rule impI)
 apply (subst BNTr7-13[of f a f (Sup D W) insert (Sup D W) W],
         assumption+,
    rule ballI, thin-tac x ∈ insert (Sup D W) W,
        simp only:insert-iff,
    erule disjE,
    frule-tac x = Sup D W in bspec, assumption,
        thin-tac ∀x∈carrier D. x ≼ f x,
    simp add:le-antisym,
    frule-tac x = xa in bspec, assumption+,
        thin-tac ∀x∈W. x ≼ Sup D W,
    frule-tac x = Sup D W in bspec, assumption,
        thin-tac ∀x∈carrier D. x ≼ f x,
        frule mem-WWa-sub-carrier[of W],
        frule-tac c = xa in subsetD[of W carrier D], assumption+,
        rule-tac a = xa in le-trans[of - Sup D W f (Sup D W)],
         assumption+)
  apply (frule mem-WWa-then[of insert (Sup D W) W f a],
        (erule conjE)+,
        thin-tac ∀x∈insert (Sup D W) W. a ≼ x,
       frule-tac x = x in bspec, assumption,
        thin-tac ∀x∈insert (Sup D W) W.
       if ExPre (Iod D (insert (Sup D W) W)) x
       then f (Pre (Iod D (insert (Sup D W) W)) x) = x
       else if a ≠ x
            then Sup D (segment (Iod D (insert (Sup D W) W)) x) = x
            else a = a, simp)
done

lemma (in Order) BNTr8:⟦f ∈ carrier D → carrier D; a ∈ carrier D;
      ∀x∈carrier D. x ≼ (f x)⟧ ⟹ ⋃ (WWa D f a) ∈ (WWa D f a)
apply (cut-tac Union-WWa-sub-carrier[of f a])
apply (rule mem-of-WWa[of ⋃ WWa D f a a f], simp)
 apply (simp add:Worder-def Torder-def)
 apply (simp add:Iod-Order[of ⋃ WWa D f a])
 apply (rule conjI)
 apply (simp add:Torder-axioms-def)
 apply ((rule allI, rule impI)+, simp add:Iod-carrier,
       (erule bexE)+)
 apply (rename-tac b c X1 X2)
 apply (frule-tac ?W1.0 = X1 and ?W2.0 = X2 in BNTr7[of f a],
         assumption+)
```

apply (*erule disjE*)
apply ((*frule-tac* $c = b$ **and** $A = X1$ **and** $B = X2$ **in** *subsetD*, *assumption*+),
(*frule-tac* $W = X2$ **in** *mem-wwa-Worder*[*of* - f a]),
(*simp add*:*Worder-def Torder-def*, (*erule conjE*)+,
simp add:*Torder-axioms-def*),

(*frule-tac* $W = X1$ **in** *mem-WWa-sub-carrier*[*of* - f a],
frule-tac $W = X2$ **in** *mem-WWa-sub-carrier*[*of* - f a],
simp add:*Iod-carrier*),
(*frule-tac* $a = b$ **in** *forall-spec*, *assumption*,
thin-tac $\forall a.\ a \in X2 \longrightarrow (\forall b.\ b \in X2 \longrightarrow$
$a \preceq_{Iod\ D\ X2}\ b \vee b \preceq_{Iod\ D\ X2}\ a)$,
frule-tac $a = c$ **in** *forall-spec*, *assumption*,
thin-tac $\forall ba.\ ba \in X2 \longrightarrow b \preceq_{Iod\ D\ X2}\ ba \vee ba \preceq_{Iod\ D\ X2}\ b$,
frule-tac $X = X1$ **and** $A = b$ **in** *UnionI*[*of* - *WWa D f a*], *assumption*+,
frule-tac $X = X2$ **and** $A = c$ **in** *UnionI*[*of* - *WWa D f a*], *assumption*+,
simp add:*Iod-le*))
apply (*frule-tac* $c = c$ **and** $A = X2$ **and** $B = X1$ **in** *subsetD*, *assumption*+,
frule-tac $W = X1$ **in** *mem-wwa-Worder*[*of* - f a]),
(*simp add*:*Worder-def Torder-def*, (*erule conjE*)+,
simp add:*Torder-axioms-def*),

(*frule-tac* $W = X2$ **in** *mem-WWa-sub-carrier*[*of* - f a],
frule-tac $W = X1$ **in** *mem-WWa-sub-carrier*[*of* - f a],
simp add:*Iod-carrier*,
frule-tac $a = b$ **in** *forall-spec*, *assumption*),
thin-tac $\forall a.\ a \in X1 \longrightarrow (\forall b.\ b \in X1 \longrightarrow$
$a \preceq_{Iod\ D\ X1}\ b \vee b \preceq_{Iod\ D\ X1}\ a)$,
(*frule-tac* $a = c$ **in** *forall-spec*, *assumption*,
thin-tac $\forall ba.\ ba \in X1 \longrightarrow b \preceq_{Iod\ D\ X1}\ ba \vee ba \preceq_{Iod\ D\ X1}\ b$,
frule-tac $X = X1$ **and** $A = b$ **in** *UnionI*[*of* - *WWa D f a*], *assumption*+,
frule-tac $X = X1$ **and** $A = c$ **in** *UnionI*[*of* - *WWa D f a*], *assumption*+),
simp add:*Iod-le*)

apply (*subst Worder-axioms-def*)
apply (*rule allI*, *rule impI*, *erule conjE*)
apply (*frule-tac* $A = X$ **in** *nonempty-ex*, *erule exE*)
apply (*simp only*:*Iod-carrier*,
frule-tac $c = x$ **and** $A = X$ **and** $B = \bigcup WWa\ D\ f\ a$ **in** *subsetD*,
assumption, *simp*, *erule bexE*)
apply (*rename-tac X x W*)
apply (*frule-tac* $W = W$ **in** *mem-wwa-Worder*[*of* - f a],
simp add:*Worder-def Torder-def*, *erule conjE*,
simp add:*Worder-axioms-def*, *erule conjE*)
apply (*frule-tac* $a = X \cap W$ **in** *forall-spec*,
thin-tac $\forall X.\ X \subseteq carrier\ (Iod\ D\ W) \wedge X \neq \{\} \longrightarrow$
$(\exists x.\ minimum\text{-}elem\ (Iod\ D\ W)\ X\ x)$,
frule-tac $W = W$ **in** *mem-WWa-sub-carrier*[*of* - f a],
simp only:*Iod-carrier*, *simp add*:*Int-lower2*, *blast*,

thin-tac $\forall X.\ X \subseteq carrier\ (Iod\ D\ W) \wedge X \neq \{\} \longrightarrow$
$(\exists x.\ minimum\text{-}elem\ (Iod\ D\ W)\ X\ x)$, *erule exE*)
apply (*frule-tac* $W = W$ **in** *mem-WWa-sub-carrier*)
apply (*frule-tac* $D = Iod\ D\ W$ **and** $X = X \cap W$ **and** $a = xa$ **in**
Order.minimum-elem-mem)
apply (*simp add*:*Iod-carrier*)
apply *simp*
apply (*rule contrapos-pp*, (*simp del*:*Union-iff*)+, *erule conjE*)
apply (*simp add*:*minimum-elem-def*)
apply (*frule-tac* $a = xa$ **in** *forall-spec*, *assumption*+,
thin-tac $\forall x.\ x \in X \longrightarrow (\exists xa{\in}X.\ \neg\ x \preceq_{Iod\ D\ (\bigcup WWa\ D\ f\ a)}\ xa)$,
erule bexE)
apply (*frule-tac* $c = xb$ **and** $A = X$ **and** $B = \bigcup WWa\ D\ f\ a$ **in** *subsetD*,
assumption+)
apply (*cut-tac* $A = xb$ **and** $C = WWa\ D\ f\ a$ **in** *Union-iff*, *simp del*:*Union-iff*,
erule bexE, *rename-tac X x W xa xb W'*,
frule-tac ?W1.0 $= W$ **and** *?W2.0* $= W'$ **in** *BNTr7*[*of f a*], *assumption*+)
apply (*case-tac* $W' \subseteq W$,
frule-tac $c = xb$ **and** $A = W'$ **and** $B = W$ **in** *subsetD*, *assumption*+,
rotate-tac 4,
frule-tac $x = xb$ **in** *bspec*, *simp*,
thin-tac $\forall x{\in}X \cap W.\ xa \preceq_{Iod\ D\ W}\ x$)
apply (*frule Iod-Order*[*of* $\bigcup WWa\ D\ f\ a$],
frule-tac $X = W$ **and** $A = xa$ **in** *UnionI*[*of - WWa D f a*], *assumption*+,
simp del:*Union-iff add*:*Iod-le*)
apply (*simp del*:*Union-iff*)
apply (*frule-tac* $c = xa$ **and** $A = W$ **and** $B = W'$ **in** *subsetD*, *assumption*+,
frule-tac $X = W'$ **and** $A = xa$ **in** *UnionI*[*of - WWa D f a*], *assumption*+,
frule-tac $X = W'$ **and** $A = xb$ **in** *UnionI*[*of - WWa D f a*], *assumption*+)
apply (*simp only*:*Iod-le*)
apply (*frule-tac* $W = W'$ **in** *mem-wwa-Worder*,
frule-tac $D = Iod\ D\ W'$ **in** *Worder.Torder*,
frule-tac $D = Iod\ D\ W'$ **in** *Worder.Order*)
apply (*frule-tac* $W = W'$ **in** *mem-WWa-sub-carrier*,
frule-tac $T1 = W'$ **and** $a1 = xa$ **and** $b1 = xb$ **in** *Iod-le*[*THEN sym*],
assumption+, *simp del*:*Union-iff*)
apply (*frule-tac* $D = Iod\ D\ W'$ **and** $a = xa$ **and** $b = xb$ **in** *Torder.not-le-less*)
apply (*simp add*:*Iod-carrier*) **apply** (*simp add*:*Iod-carrier*)
apply (*simp del*:*Union-iff add*:*Iod-less*, *thin-tac* $\neg\ xa \preceq xb$,
thin-tac $\neg\ xa \preceq_{Iod\ D\ W'}\ xb$)
apply (*frule Iod-Order*[*of* $\bigcup WWa\ D\ f\ a$])
apply (*frule-tac* $a1 = xb$ **and** $b1 = xa$ **in** *Iod-less*[*THEN sym*, *of* $\bigcup\ WWa\ D\ f\ a$],
assumption+, *simp del*:*Union-iff*,
frule-tac $x = xa$ **and** $xa = xb$ **and** $W = W$ **in** *BNTr7-1*[*of f a*],
assumption+,
frule-tac $a1 = xb$ **and** $b1 = xa$ **and** $T1 = W$ **in** *Iod-less*[*THEN sym*],
assumption+, *simp del*:*Union-iff*,
frule-tac $W = W$ **in** *mem-wwa-Worder*,

frule-tac $D = Iod\ D\ W$ **in** *Worder.Torder*)
apply (*frule-tac* $D1 = Iod\ D\ W$ **and** $a1 = xa$ **and** $b1 = xb$ **in**
Torder.not-le-less[*THEN sym*],
simp add:*Iod-carrier*, *simp add*:*Iod-carrier*, *simp del*:*Union-iff*)
apply (*rule BNTr7-6*, *assumption*+)

apply (*rule ballI*)
apply (*cut-tac* $A = x$ **in** *Union-iff*[*of - WWa D f a*], *simp del*:*Union-iff*)
apply (*erule bexE*, *rename-tac x W*)
apply (*simp add*:*WWa-def Wa-def*[*of D - f a*])

apply (*rule ballI*)
apply (*case-tac ExPre* (*Iod D* ($\bigcup$ *WWa D f a*)) *x*, *simp*,
erule bexE, *rename-tac x W*,
frule-tac $X = W$ **and** $A = x$ **in** *UnionI*[*of - WWa D f a*], *assumption*+,
frule-tac $x = x$ **in** *BNTr7-2*[*of f a*], *assumption*+,
frule-tac $x = W$ **in** *bspec*, *assumption*,
thin-tac $\forall\ W \in WWa\ D\ f\ a.\ x \in W \longrightarrow ExPre\ (Iod\ D\ W)\ x$,
simp del:*Union-iff*)
apply (*frule-tac* $x = x$ **in** *BNTr7-3*[*of f a*], *assumption*+,
frule-tac $x = W$ **in** *bspec*, *assumption*,
thin-tac $\forall\ W \in WWa\ D\ f\ a.$
$x \in W \longrightarrow Pre\ (Iod\ D\ (\bigcup WWa\ D\ f\ a))\ x = Pre\ (Iod\ D\ W)\ x$,
simp del:*Union-iff*)
apply (*simp add*:*WWa-def Wa-def*)

apply (*simp del*:*Union-iff*, *rule impI*)
apply (*cut-tac* $A = x$ **in** *Union-iff*[*of - WWa D f a*], *simp del*:*Union-iff*,
erule bexE, *rename-tac x W*,
frule-tac $x = x$ **and** $W = W$ **in** *BNTr7-5*[*of f a -*], *assumption*+,
simp)
apply (*frule-tac* $x = x$ **and** $W = W$ **in** *BNTr7-4*[*of f a*], *assumption*+,
simp del:*Union-iff*,
thin-tac $\bigcup WWa\ D\ f\ a \subseteq carrier\ D$,
thin-tac $\exists\ X \in WWa\ D\ f\ a.\ x \in X$,
thin-tac segment (*Iod D* ($\bigcup$ *WWa D f a*)) *x* = *segment* (*Iod D W*) *x*,
thin-tac $\neg$ *ExPre* (*Iod D* ($\bigcup$ *WWa D f a*)) *x*)
apply (*simp add*:*WWa-def Wa-def*)
done

lemma (**in** *Order*) *BNTr10*:⟦*S-inductive-set D*; $f \in carrier\ D \to carrier\ D$;
$a \in carrier\ D$; $\forall\ x \in carrier\ D.\ x \preceq (f\ x)$⟧ ⟹
(*Sup D* ($\bigcup$ *WWa D f a*)) ∈ ($\bigcup$ *WWa D f a*)
apply (*frule-tac* $f = f$ **and** $a = a$ **in** *BNTr8*, *assumption*+,
frule BNTr7-14[*of f a* $\bigcup$ *WWa D f a*], *assumption*+,
frule mem-family-sub-Un[*of insert* (*Sup D* ($\bigcup$ *WWa D f a*)) ($\bigcup$ *WWa D f a*)
WWa D f a],

cut-tac insertI1[*of Sup D* ($\bigcup$ *WWa D f a*) $\bigcup$ *WWa D f a*])
apply (*simp add*:*subsetD*)
done

lemma (**in** *Order*) *BNTr11*:⟦*S-inductive-set D*; *f* ∈ *carrier D* → *carrier D*;
a ∈ *carrier D*; ∀ *x*∈*carrier D*. *x* ⪯ (*f x*)⟧ ⟹
f (*Sup D* ($\bigcup$ *WWa D f a*)) = (*Sup D* ($\bigcup$ *WWa D f a*))
apply (*frule-tac f* = *f* **and** *a* = *a* **in** *BNTr8*, *assumption*+,
frule mem-WWa-Chain[*of* $\bigcup$ *WWa D f a*],
frule BNTr10[*of f a*], *assumption*+,
frule S-inductive-sup-mem[*of* $\bigcup$ *WWa D f a*], *assumption*)
apply (*frule BNTr7-17*[*of f a* $\bigcup$ *WWa D f a*], *assumption*+)
apply (*cut-tac insertI1*[*of f* (*Sup D* ($\bigcup$ *WWa D f a*)) *insert* (*Sup D* ($\bigcup$ *WWa D f a*)) ($\bigcup$ *WWa D f a*)],
frule mem-family-sub-Un[*of insert* (*f* (*Sup D* ($\bigcup$ *WWa D f a*)))
(*insert* (*Sup D* ($\bigcup$ *WWa D f a*)) ($\bigcup$ *WWa D f a*)) *WWa D f a*])
apply (*frule subsetD*[*of insert* (*f* (*Sup D* ($\bigcup$ *WWa D f a*)))
(*insert* (*Sup D* ($\bigcup$ *WWa D f a*)) ($\bigcup$ *WWa D f a*)) $\bigcup$ *WWa D f a f* (*Sup D* ($\bigcup$ *WWa D f a*))],
assumption+)
apply (*frule S-inductive-sup-bound*[*of* $\bigcup$ *WWa D f a*], *assumption*)
apply (*frule-tac x* = *f* (*Sup D* ($\bigcup$ *WWa D f a*)) **in** *bspec*,
assumption,
thin-tac ∀ *x*∈$\bigcup$ *WWa D f a*. *x* ⪯ *Sup D* ($\bigcup$ *WWa D f a*))
apply (*frule-tac x* = *Sup D* ($\bigcup$ *WWa D f a*) **in** *bspec*, *assumption*,
thin-tac ∀ *x*∈*carrier D*. *x* ⪯ *f x*)
apply (*frule funcset-mem*[*of f carrier D carrier D Sup D* ($\bigcup$ *WWa D f a*)],
assumption+,
rule le-antisym[*of f* (*Sup D* ($\bigcup$ *WWa D f a*)) *Sup D* ($\bigcup$ *WWa D f a*)],
assumption+)
done

lemma (**in** *Order*) *Bourbaki-Nakayama*:⟦*S-inductive-set D*;
f ∈ *carrier D* → *carrier D*; *a* ∈ *carrier D*; ∀ *x*∈*carrier D*. *x* ⪯ (*f x*)⟧ ⟹
∃ *x0*∈*carrier D*. *f x0* = *x0*
apply (*frule BNTr8*[*of f a*], *assumption*+,
frule mem-WWa-Chain[*of* $\bigcup$ *WWa D f a f a*],
frule S-inductive-sup-mem[*of* $\bigcup$ *WWa D f a*], *assumption*+,
frule BNTr11[*of f a*], *assumption*+)
apply *blast*
done

definition
maxl-fun :: - ⇒ *'a* ⇒ *'a* **where**
maxl-fun D = (λ*x*∈*carrier D*. *if* ∃ *y*. *y*∈(*upper-bounds D* {*x*}) ∧ *y* ≠ *x then*
SOME z. *z* ∈ (*upper-bounds D* {*x*}) ∧ *z* ≠ *x else x*)

lemma (**in** *Order*) *maxl-funTr*:⟦*x* ∈ *carrier D*;
∃ *y*. *y* ∈ *upper-bounds D* {*x*} ∧ *y* ≠ *x*⟧ ⟹

(*SOME z. z ∈ upper-bounds D {x} ∧ z ≠ x*) ∈ *carrier D*
apply (*rule someI2-ex*, *assumption+*,
simp add:*upper-bounds-def upper-bound-def*)
done

lemma (**in** *Order*) *maxl-fun-func*:*maxl-fun D* ∈ *carrier D* → *carrier D*
by (*simp add*:*maxl-fun-def maxl-funTr*)

lemma (**in** *Order*) *maxl-fun-gt*:⟦*x* ∈ *carrier D*;
∃ *y*∈ *carrier D*. *x* ≼ *y* ∧ *x* ≠ *y* ⟧ ⟹
x ≼ (*maxl-fun D x*) ∧ (*maxl-fun D x*) ≠ *x*
apply (*simp add*:*maxl-fun-def upper-bounds-def upper-bound-def*,
rule conjI, *rule impI*)
apply (*rule someI2-ex*, *assumption+*, *simp*,
erule bexE, *blast*)
done

lemma (**in** *Order*) *maxl-fun-maxl*:⟦*x* ∈ *carrier D*; *maxl-fun D x* = *x* ⟧
⟹ *maximal x*
apply (*rule contrapos-pp*, *simp+*, *simp add*:*maximal-element-def*)
apply (*frule maxl-fun-gt*[*of x*], *assumption*, *erule conjE*, *simp*)
done

lemma (**in** *Order*) *maxl-fun-asc*:∀ *x*∈*carrier D*. *x* ≼ (*maxl-fun D x*)
apply (*rule ballI*)
apply (*simp add*:*maxl-fun-def*, *rule conjI*, *rule impI*)
apply (*rule someI2-ex*, *assumption*, *simp add*:*upper-bounds-def upper-bound-def*,
rule impI, *rule le-refl*, *assumption*)
done

lemma (**in** *Order*) *S-inductive-maxl*:⟦*S-inductive-set D*; *carrier D* ≠ {}⟧ ⟹
∃ *m*. *maximal m*
apply (*frule nonempty-ex* [*of carrier D*],
erule exE, *rename-tac a*)
apply (*cut-tac maxl-fun-asc*, *cut-tac maxl-fun-func*,
frule-tac a = *a* **in** *Bourbaki-Nakayama*[*of maxl-fun D* -], *assumption+*)
apply (*erule bexE*,
frule-tac x = *x0* **in** *maxl-fun-maxl*, *assumption+*)
apply *blast*
done

lemma (**in** *Order*) *maximal-mem*:*maximal m* ⟹ *m* ∈ *carrier D*
by (*simp add*:*maximal-element-def*)

definition
Chains :: - ⇒ ('*a set*) *set* **where**
Chains D == {*C*. *Chain D C*}

definition
family-Torder:: - ⇒ ('*a set*) *Order*
((*fTo* -) [*999*]*1000*) **where**
fTo D = (|*carrier* = *Chains D* , *rel* = {*Z*. *Z* ∈ (*Chains D*) × (*Chains D*) ∧ (*fst Z*) ⊆ (*snd Z*)}|)

lemma (**in** *Order*) *Chain-mem-fTo*:*Chain D C* ⟹ *C* ∈ *carrier* (*fTo D*)
by (*simp add*:*family-Torder-def Chains-def*)

lemma (**in** *Order*) *fTOrder*:*Order* (*fTo D*)
apply (*subst Order-def*)
apply (*simp add*:*family-Torder-def*)
apply *auto*
done

lemma (**in** *Order*) *fTo-Order-sub*:⟦*A* ∈ *carrier* (*fTo D*); *B* ∈ *carrier* (*fTo D*)⟧
⟹ (*A* ≼(*fTo D*) *B*) = (*A* ⊆ *B*)
by (*subst ole-def*, *simp add*:*family-Torder-def*)

lemma (**in** *Order*) *mem-fTo-Chain*:*X* ∈ *carrier* (*fTo D*) ⟹ *Chain D X*
by (*simp add*:*family-Torder-def Chains-def*)

lemma (**in** *Order*) *mem-fTo-sub-carrier*:*X* ∈ *carrier* (*fTo D*) ⟹ *X* ⊆ *carrier D*
by (*frule mem-fTo-Chain*[*of X*], *simp add*:*Chain-sub*)

lemma (**in** *Order*) *Un-fTo-Chain*:*Chain* (*fTo D*) *CC* ⟹ *Chain D* (⋃ *CC*)
apply (*cut-tac fTOrder*,
frule Order.Chain-sub[*of fTo D CC*], *assumption*+,
cut-tac family-subset-Un-sub[*of CC carrier D*],
subst Chain-def, *simp*, *simp add*:*Torder-def*, *simp add*:*Iod-Order*,
simp add:*Torder-axioms-def*)
apply ((*rule allI*, *rule impI*)+, *simp add*:*Iod-carrier*)
apply ((*erule bexE*)+,
frule-tac A = *X* **in** *mem-family-sub-Un*[*of* - *CC*],
frule-tac A = *Xa* **in** *mem-family-sub-Un*[*of* - *CC*],
frule-tac c = *a* **and** *A* = *X* **in** *subsetD*[*of* - ⋃ *CC*], *assumption*+,
frule-tac c = *b* **and** *A* = *Xa* **in** *subsetD*[*of* - ⋃ *CC*], *assumption*+,
simp only:*Iod-le*,
frule-tac c = *X* **in** *subsetD*[*of CC carrier fTo D*], *assumption*+,
frule-tac c = *Xa* **in** *subsetD*[*of CC carrier fTo D*], *assumption*+)
apply (*simp add*:*Chain-def Torder-def Torder-axioms-def*,
thin-tac ∃ *X*∈*CC*. *a* ∈ *X*, *thin-tac* ∃ *X*∈*CC*. *b* ∈ *X*,
(*erule conjE*)+,
frule-tac a = *X* **in** *forall-spec*,
simp only:*Order.Iod-carrier*[*of fTo D CC*],
thin-tac ∀ *a*. *a* ∈ *carrier* (*Iod* (*fTo D*) *CC*) ⟶
(∀ *b*. *b* ∈ *carrier* (*Iod* (*fTo D*) *CC*) ⟶
a ≼*Iod* (*fTo D*) *CC* *b* ∨ *b* ≼*Iod* (*fTo D*) *CC* *a*),

frule-tac $a = Xa$ **in** *forall-spec*,
simp only:*Order.Iod-carrier*[*of fTo D CC*],
thin-tac $\forall\, b.\ b \in$ *carrier* (*Iod* (*fTo D*) *CC*) $\longrightarrow$
$X \preceq_{Iod\ (fTo\ D)\ CC}\ b \vee b \preceq_{Iod\ (fTo\ D)\ CC}\ X$)
apply (*simp add*:*Order.Iod-le*) **apply** (*simp add*:*fTo-Order-sub*)
apply (*frule-tac* $X = X$ **in** *mem-fTo-Chain*,
frule-tac $X = Xa$ **in** *mem-fTo-Chain*,
frule-tac $X = X$ **in** *Chain-Torder*,
frule-tac $X = Xa$ **in** *Chain-Torder*,
frule-tac $X = X$ **in** *Chain-sub*,
frule-tac $X = Xa$ **in** *Chain-sub*)
apply (*simp add*:*Torder-def Torder-axioms-def*, (*erule conjE*)+,
simp only:*Iod-carrier*)
apply (*erule disjE*,
frule-tac $c = a$ **and** $A = X$ **and** $B = Xa$ **in** *subsetD*, *assumption*+,
thin-tac $\forall\, a.\ a \in X \longrightarrow (\forall\, b.\ b \in X \longrightarrow a \preceq_{Iod\ D\ X}\ b \vee b \preceq_{Iod\ D\ X}\ a)$,
frule-tac $a = a$ **in** *forall-spec*,
thin-tac $\forall\, a.\ a \in Xa \longrightarrow (\forall\, b.\ b \in Xa \longrightarrow a \preceq_{Iod\ D\ Xa}\ b \vee b \preceq_{Iod\ D\ Xa}\ a)$,
assumption,
thin-tac $\forall\, a.\ a \in Xa \longrightarrow (\forall\, b.\ b \in Xa \longrightarrow a \preceq_{Iod\ D\ Xa}\ b \vee b \preceq_{Iod\ D\ Xa}\ a)$,
frule-tac $a = b$ **in** *forall-spec*, *assumption*,
thin-tac $\forall\, b.\ b \in Xa \longrightarrow a \preceq_{Iod\ D\ Xa}\ b \vee b \preceq_{Iod\ D\ Xa}\ a$,
simp add:*Iod-le*)
apply (
frule-tac $c = b$ **and** $A = Xa$ **and** $B = X$ **in** *subsetD*, *assumption*+,
thin-tac $\forall\, a.\ a \in Xa \longrightarrow (\forall\, b.\ b \in Xa \longrightarrow a \preceq_{Iod\ D\ Xa}\ b \vee b \preceq_{Iod\ D\ Xa}\ a)$,
frule-tac $a = a$ **in** *forall-spec*,
thin-tac $\forall\, a.\ a \in X \longrightarrow (\forall\, b.\ b \in X \longrightarrow a \preceq_{Iod\ D\ X}\ b \vee b \preceq_{Iod\ D\ X}\ a)$,
assumption,
thin-tac $\forall\, a.\ a \in X \longrightarrow (\forall\, b.\ b \in X \longrightarrow a \preceq_{Iod\ D\ X}\ b \vee b \preceq_{Iod\ D\ X}\ a)$,
frule-tac $a = b$ **in** *forall-spec*, *assumption*,
thin-tac $\forall\, b.\ b \in X \longrightarrow a \preceq_{Iod\ D\ X}\ b \vee b \preceq_{Iod\ D\ X}\ a$,
simp add:*Iod-le*)

apply (*rule ballI*,
frule-tac $c = A$ **in** *subsetD*[*of CC carrier* (*fTo D*)], *assumption*+,
rule mem-fTo-sub-carrier, *assumption*+)
done

lemma (**in** *Order*) *Un-fTo-Chain-mem-fTo*:*Chain* (*fTo D*) *CC* $\Longrightarrow$
$(\bigcup\ CC) \in$ *carrier* (*fTo D*)
apply (*frule Un-fTo-Chain*[*of CC*], *thin-tac Chain* (*fTo D*) *CC*)
apply (*simp add*:*family-Torder-def Chains-def*)
done

lemma (**in** *Order*) *Un-upper-bound*:*Chain* (*fTo D*) *C* $\Longrightarrow$
$\bigcup C \in$ *upper-bounds* (*fTo D*) *C*
apply (*simp add*:*upper-bounds-def upper-bound-def*,
simp add:*Un-fTo-Chain-mem-fTo*)

apply (*rule ballI*,
simp add:*ole-def*,
subst family-Torder-def, *simp*)

apply (*cut-tac fTOrder*,
frule Order.Chain-sub[*of fTo D C*], *assumption*,
frule-tac c = *s* **in** *subsetD*[*of C carrier* (*fTo D*)], *assumption*+,
cut-tac Un-fTo-Chain-mem-fTo[*of C*],
simp add:*mem-fTo-Chain Chains-def*,
rule-tac A = *s* **in** *mem-family-sub-Un*[*of* - *C*], *assumption*)
apply *assumption*
done

lemma (**in** *Order*) *fTo-conditional-inc-C*:*C* ∈ *carrier* (*fTo D*) ⟹
C ∈ *carrier* (*Iod* (*fTo D*) {*S* ∈ *carrier fTo D*. *C* ⊆ *S*})
apply (*cut-tac fTOrder*,
cut-tac conditional-subset[*of carrier* (*fTo D*) *op* ⊆ *C*])
apply (*simp add*:*Order.Iod-carrier*)
done

lemma (**in** *Order*) *fTo-conditional-Un-Chain-mem1*: ⟦*C* ∈ *carrier* (*fTo D*);
Chain (*Iod* (*fTo D*) {*S* ∈ *carrier* (*fTo D*). *C* ⊆ *S*}) *Ca*; *Ca* ≠ {}⟧ ⟹
⋃ *Ca* ∈ *upper-bounds* (*Iod* (*fTo D*) {*S* ∈ *carrier fTo D*. *C* ⊆ *S*}) *Ca*

apply (*cut-tac fTOrder*,
cut-tac conditional-subset[*of carrier* (*fTo D*) *op* ⊆ *C*])

apply (*simp add*:*upper-bounds-def upper-bound-def*)
apply (*subgoal-tac* ⋃ *Ca* ∈*carrier* (*Iod* (*fTo D*) {*S* ∈ *carrier* (*fTo D*). *C* ⊆ *S*}))
apply *simp*
apply (*rule ballI*)
apply (*simp only*:*Order.Iod-carrier*)
apply (*frule Order.Iod-Order*[*of fTo D* {*x* ∈ *carrier fTo D*. *C* ⊆ *x*}],
assumption+,
frule Order.Chain-sub[*of Iod* (*fTo D*) {*S* ∈ *carrier fTo D*. *C* ⊆ *S*}
Ca], *assumption*+, *simp only*:*Order.Iod-carrier*,
frule-tac c = *s* **in** *subsetD*[*of Ca* {*S* ∈ *carrier fTo D*. *C* ⊆ *S*}],
assumption+,
frule-tac c = *s* **in** *subsetD*[*of* {*S* ∈ *carrier fTo D*. *C* ⊆ *S*}
carrier (*fTo D*)], *assumption*+)
apply (*subst Order.Iod-le*[*of fTo D* {*x* ∈ *carrier fTo D*. *C* ⊆ *x*}],
assumption+)
apply (*subst Order.fTo-Order-sub*, *rule Order-axioms*, *assumption*, *simp*)
apply (*rule-tac A* = *s* **in** *mem-family-sub-Un*[*of* - *Ca*], *assumption*)

apply (*simp add*:*Order.Iod-carrier*[*of fTo D*])
apply (*rule conjI*)
apply (*rule Un-fTo-Chain-mem-fTo*[*of Ca*])
apply (*simp add*:*Chain-def*, *erule conjE*)

apply (*simp add*:*Order.Iod-carrier*,
frule subset-trans[*of Ca* {$S \in$ *carrier* (*fTo D*). $C \subseteq S$}
carrier (*fTo D*)], *assumption*+, *simp*)
apply (*frule-tac* $T = \{S \in$ *carrier* (*fTo D*). $C \subseteq S\}$ **in**
Order.Iod-sub-sub[*of fTo D Ca*], *assumption*+,
simp)
apply (*rule sub-Union*[*of Ca C*])
apply (*frule nonempty-ex*[*of Ca*], *erule exE*)
apply (*frule Order.Iod-Order*[*of fTo D* {$x \in$ *carrier* (*fTo D*). $C \subseteq x$}],
assumption+,
frule Order.Chain-sub[*of Iod* (*fTo D*) {$S \in$ *carrier fTo D*. $C \subseteq S$} *Ca*],
assumption+) **apply** (*simp add*:*Order.Iod-carrier*)
apply (*frule-tac* $c = x$ **in** *subsetD*[*of Ca* {$S \in$ *carrier fTo D*. $C \subseteq S$}],
assumption+, *simp*, *erule conjE*, *blast*)
done

lemma (**in** *Order*) *fTo-conditional-min1*: $[\![C \in$ *carrier* (*fTo D*);
Chain (*Iod* (*fTo D*) {$S \in$ *carrier* (*fTo D*). $C \subseteq S$}) *Ca*; $Ca \neq \{\}]\!] \Longrightarrow$
minimum-elem (*Iod* (*fTo D*) {$S \in$ *carrier fTo D*. $C \subseteq S$})
(*upper-bounds* (*Iod* (*fTo D*) {$S \in$ *carrier* (*fTo D*). $C \subseteq S$}) *Ca*) ($\bigcup Ca$)
apply (*frule fTo-conditional-Un-Chain-mem1*[*of C Ca*], *assumption*+,
simp add:*minimum-elem-def*)
apply (*rule ballI*)
apply (*simp add*:*upper-bounds-def upper-bound-def*, (*erule conjE*)+,
cut-tac fTOrder,
cut-tac conditional-subset[*of carrier* (*fTo D*) *op* $\subseteq$ *C*])
apply (*simp only*:*Order.Iod-carrier*,
simp only:*Order.Iod-le*[*of fTo D*])
apply (*frule-tac* $c = \bigcup Ca$ **in** *subsetD*[*of* {$S \in$ *carrier fTo D*. $C \subseteq S$}
carrier (*fTo D*)], *assumption*+,
frule-tac $c = x$ **in** *subsetD*[*of* {$S \in$ *carrier fTo D*. $C \subseteq S$}
carrier (*fTo D*)], *assumption*+)
apply (*subst Order.fTo-Order-sub*, *rule Order-axioms*, *assumption*, *simp*,
rule-tac $C = Ca$ **and** $B = x$ **in** *family-subset-Un-sub*)
apply (*rule ballI*)
apply (*thin-tac* $\forall s{\in}Ca.\ s \preceq_{Iod\ (fTo\ D)} \{S \in$ *carrier* (*fTo D*). $C \subseteq S\} \bigcup Ca$,
frule-tac $x = A$ **in** *bspec*, *assumption*,
thin-tac $\forall s{\in}Ca.\ s \preceq_{Iod\ (fTo\ D)} \{S \in$ *carrier* (*fTo D*). $C \subseteq S\}\ x$)
apply (*frule Order.Iod-Order*[*of fTo D* {$S \in$ *carrier fTo D*. $C \subseteq S$}],
assumption,
frule Order.Chain-sub[*of Iod* (*fTo D*) {$S \in$ *carrier* (*fTo D*). $C \subseteq S$}
Ca], *assumption*,
simp only:*Order.Iod-carrier*[*of fTo D* {$S \in$ *carrier fTo D*. $C \subseteq S$}])
apply (*frule-tac* $c = x$ **in** *subsetD*[*of* {$x \in$ *carrier fTo D*. $C \subseteq x$}
carrier (*fTo D*)], *simp*,
frule-tac $c = A$ **in** *subsetD*[*of Ca* {$x \in$ *carrier fTo D*. $C \subseteq x$}],
assumption+,
frule-tac $a = A$ **and** $b = x$ **in** *Order.Iod-le*[*of fTo D*
{$x \in$ *carrier fTo D*. $C \subseteq x$}], *assumption*+)

apply *simp* **apply** (*simp add*:*fTo-Order-sub*)
done

lemma (**in** *Order*) *fTo-conditional-Un-Chain-mem2*: ⟦$C \in$ *carrier* (*fTo D*);
Chain (*Iod* (*fTo D*) {$S \in$ *carrier fTo D*. $C \subseteq S$}) *Ca*; $Ca = \{\}$⟧ ⟹
$C \in$ *upper-bounds* (*Iod* (*fTo D*) {$S \in$ *carrier* (*fTo D*). $C \subseteq S$}) *Ca*
apply (*cut-tac fTOrder*,
cut-tac conditional-subset[*of carrier* (*fTo D*) *op* $\subseteq$ C])
apply (*simp add*:*upper-bounds-def upper-bound-def*, *simp add*:*Order.Iod-carrier*)
done

lemma (**in** *Order*) *fTo-conditional-min2*: ⟦$C \in$ *carrier* (*fTo D*);
Chain (*Iod* (*fTo D*) {$S \in$ *carrier* (*fTo D*). $C \subseteq S$}) *Ca*; $Ca = \{\}$⟧ ⟹
minimum-elem (*Iod* (*fTo D*) {$S \in$ *carrier fTo D*. $C \subseteq S$})
(*upper-bounds* (*Iod* (*fTo D*) {$S \in$ *carrier* (*fTo D*). $C \subseteq S$}) *Ca*) C
apply (*simp add*:*minimum-elem-def upper-bounds-def upper-bound-def*)
apply (*cut-tac fTOrder*,
cut-tac conditional-subset[*of carrier* (*fTo D*) *op* $\subseteq$ C],
simp add:*Order.Iod-carrier*)
apply (*rule allI*, *rule impI*, *erule conjE*,
subst Order.Iod-le[*of fTo D* {$x \in$ *carrier* (*fTo D*). $C \subseteq x$}],
assumption+, *simp*, *simp*)
apply (*subst fTo-Order-sub*, *assumption*+)
done

lemma (**in** *Order*) *fTo-S-inductive*:*S-inductive-set* (*fTo D*)
apply (*simp add*:*S-inductive-set-def*,
rule allI, *rule impI*)
apply (*rule contrapos-pp*, *simp*+)
apply (*simp add*:*minimum-elem-def*,
frule-tac CC = C **in** *Un-fTo-Chain-mem-fTo*,
frule-tac x = $\bigcup C$ **in** *bspec*, *assumption*,
thin-tac ∀ $x \in$ *carrier* (*fTo D*).
$x \in$ *upper-bounds* (*fTo D*) C ⟶
(∃ $xa \in$ *upper-bounds* (*fTo D*) C. ¬ $x \preceq_{(fTo\ D)} xa$))
apply (*frule-tac C* = C **in** *Un-upper-bound*, *simp*,
erule bexE,
thin-tac $\bigcup C \in$ *upper-bounds* (*fTo D*) C)
apply (*simp add*:*upper-bounds-def upper-bound-def*,
erule conjE)
apply (*cut-tac C* = C **and** $B = x$ **in** *family-subset-Un-sub*)
apply (*rule ballI*)
apply (*frule-tac x* = A **in** *bspec*, *assumption*,
thin-tac ∀ $s \in C$. $s \preceq_{fTo\ D} x$,
cut-tac fTOrder,
frule-tac X = C **in** *Order.Chain-sub*[*of fTo D*], *assumption*+,
frule-tac c = A **and** $A = C$ **in** *subsetD*[*of* - *carrier* (*fTo D*)],
assumption+,

simp add:*fTo-Order-sub*)
apply (*simp add*:*fTo-Order-sub*)
done

lemma (**in** *Order*) *conditional-min-upper-bound*: ⟦*C* ∈ *carrier* (*fTo D*);
Chain (*Iod* (*fTo D*) {*S* ∈ *carrier fTo D*. *C* ⊆ *S*}) *Ca*⟧ ⟹
∃ *X*. *minimum-elem* (*Iod* (*fTo D*) {*S* ∈ *carrier* (*fTo D*). *C* ⊆ *S*})
(*upper-bounds* (*Iod* (*fTo D*) {*S* ∈ *carrier* (*fTo D*). *C* ⊆ *S*}) *Ca*) *X*
apply (*case-tac Ca* = {},
frule fTo-conditional-min2[*of C*], *assumption*+, *blast*,
frule fTo-conditional-min1[*of C*], *assumption*+, *blast*)
done

lemma (**in** *Order*) *Hausdorff-acTr*:*C* ∈ *carrier* (*fTo D*) ⟹
S-inductive-set (*Iod* (*fTo D*) {*S*. *S* ∈ (*carrier* (*fTo D*)) ∧ *C* ⊆ *S*})
apply (*simp add*:*S-inductive-set-def*)
apply (*rule allI*, *rule impI*)
apply (*frule-tac Ca* = *Ca* **in** *conditional-min-upper-bound*[*of C*],
assumption+)
apply (*erule exE*, *cut-tac fTOrder*)
apply (*cut-tac conditional-subset*[*of carrier* (*fTo D*) *op* ⊆ *C*])
apply (*frule Order.Iod-Order*[*of fTo D* {*S* ∈ *carrier fTo D*. *C* ⊆ *S*}],
assumption+)
apply (*frule-tac X* = *upper-bounds* (*Iod* (*fTo D*)
{*S* ∈ *carrier* (*fTo D*). *C* ⊆ *S*}) *Ca*
and *a* = *X* **in**
Order.minimum-elem-mem[*of Iod* (*fTo D*) {*S* ∈ *carrier* (*fTo D*). *C* ⊆ *S*}])
apply (*simp only*:*Order.Iod-carrier*)
apply (*rule subsetI*,
thin-tac minimum-elem (*Iod* (*fTo D*) {*S* ∈ *carrier* (*fTo D*). *C* ⊆ *S*})
(*upper-bounds* (*Iod* (*fTo D*) {*S* ∈ *carrier* (*fTo D*). *C* ⊆ *S*}) *Ca*) *X*)
apply (*simp add*:*upper-bounds-def upper-bound-def*, *erule conjE*)
apply (*simp add*:*Order.Iod-carrier*) **apply** *assumption*
apply (*subgoal-tac X*∈*carrier* (*Iod* (*fTo D*) {*S* ∈ *carrier* (*fTo D*). *C* ⊆ *S*}))
apply *blast*
apply (*frule-tac X* = *Ca* **in** *Order.Chain-sub*[*of Iod* (*fTo D*)
{*S* ∈ *carrier* (*fTo D*). *C* ⊆ *S*}], *assumption*,
frule-tac X = *Ca* **in** *Order.upper-bounds-sub*[*of*
Iod (*fTo D*) {*S* ∈ *carrier* (*fTo D*). *C* ⊆ *S*}], *assumption*)
apply (*thin-tac minimum-elem* (*Iod* (*fTo D*) {*S* ∈ *carrier* (*fTo D*). *C* ⊆ *S*})
(*upper-bounds* (*Iod* (*fTo D*) {*S* ∈ *carrier* (*fTo D*). *C* ⊆ *S*}) *Ca*) *X*)
apply (*rule-tac A* = *upper-bounds* (*Iod* (*fTo D*) {*S* ∈ *carrier* (*fTo D*). *C* ⊆ *S*})
Ca **and** *B* = *carrier* (*Iod* (*fTo D*) {*S* ∈ *carrier* (*fTo D*). *C* ⊆ *S*}) **and**
c = *X* **in** *subsetD*, *assumption*+)
done

lemma *satisfy-cond-mem-set*:⟦ *x* ∈ *A*; *P x* ⟧ ⟹ *x* ∈ {*y* ∈ *A*. *P y*}
by *blast*

lemma (**in** *Order*) *maximal-conditional-maximal*: $[\![C \in carrier\ (fTo\ D);$
$maximal_{Iod\ (fTo\ D)}\ \{S \in carrier\ (fTo\ D).\ C \subseteq S\}\ m]\!] \Longrightarrow maximal_{(fTo\ D)}\ m$
apply (*unfold maximal-element-def*, *erule conjE*)
apply (*cut-tac conditional-subset*[*of carrier* (*fTo D*) *op* $\subseteq$ *C*],
cut-tac fTOrder,
frule Order.Iod-Order[*of fTo D* $\{x \in carrier\ fTo\ D.\ C \subseteq x\}$],
assumption+,
simp only:*Order.Iod-carrier*,
frule subsetD[*of* $\{S \in carrier\ fTo\ D.\ C \subseteq S\}$
carrier (*fTo D*) *m*], *assumption*+, *simp*)
apply (*rule ballI*, *rule impI*)
apply (*simp only*:*fTo-Order-sub*,
frule-tac a = *b* **in** *forall-spec*, *simp*,
thin-tac $\forall b.\ b \in carrier\ (fTo\ D) \wedge C \subseteq b \longrightarrow$
$m \preceq_{Iod\ (fTo\ D)}\ \{S \in carrier\ (fTo\ D).\ C \subseteq S\}\ b \longrightarrow m = b$)
apply (*frule-tac x* = *m* **in**
satisfy-cond-mem-set[*of - carrier* (*fTo D*) *op* $\subseteq$ *C*], *assumption*+,
frule-tac A = *C* **and** *B* = *m* **and** *C* = *b* **in** *subset-trans*, *assumption*+,
frule-tac x = *b* **in**
satisfy-cond-mem-set[*of - carrier* (*fTo D*) *op* $\subseteq$ *C*], *assumption*+,
simp only:*Order.Iod-le*, *simp only*:*fTo-Order-sub*)
done

lemma (**in** *Order*) *Hausdorff-ac*:$C \in carrier\ (fTo\ D) \Longrightarrow$
$\exists M \in carrier\ (fTo\ D).\ C \subseteq M \wedge maximal_{(fTo\ D)}\ M$
apply (*frule-tac Hausdorff-acTr*[*of C*],
cut-tac conditional-subset[*of carrier* (*fTo D*) *op* $\subseteq$ *C*],
cut-tac fTOrder,
frule Order.Iod-Order[*of fTo D* $\{x \in carrier\ fTo\ D.\ C \subseteq x\}$],
assumption+)
apply (*frule Order.S-inductive-maxl*[*of Iod* (*fTo D*)
$\{S \in carrier\ (fTo\ D).\ C \subseteq S\}$], *assumption*+,
frule fTo-conditional-inc-C[*of C*], *simp add*:*nonempty*,
erule exE,
frule-tac m = *m* **in** *Order.maximal-mem*[*of*
Iod (*fTo D*) $\{x \in carrier\ (fTo\ D).\ C \subseteq x\}$], *assumption*+,
simp add:*Order.Iod-carrier*, *erule conjE*,
frule-tac m = *m* **in** *maximal-conditional-maximal*[*of C*], *assumption*+)
apply *blast*
done

lemma (**in** *Order*) *Zorn-lemmaTr*:$[\![Chain\ D\ C;\ M \in carrier\ fTo\ D;\ C \subseteq M;$
$maximal_{fTo\ D}\ M;\ b \in carrier\ D;\ \forall s \in M.\ s \preceq b\ ]\!] \Longrightarrow$
maximal b $\wedge$ *b* $\in$ *upper-bounds D C*
apply (*simp add*:*upper-bounds-def upper-bound-def*)
apply (*rule conjI*)
prefer *2*
apply (*rule ballI*, *simp add*:*subsetD*,

rule contrapos-pp, *simp*+, *simp add*:*maximal-element-def*,
erule bexE, *erule conjE*)
apply (*frule-tac* $X = M$ **in** *mem-fTo-Chain*,
frule-tac $X = M$ **and** $b = ba$ **in** *adjunct-Chain*, *assumption*+,
rule ballI)
apply (*frule-tac* $x = x$ **in** *bspec*, *assumption*+,
thin-tac $\forall s{\in}M.\ s \preceq b$,
frule-tac $X = M$ **in** *Chain-sub*,
frule-tac $c = x$ **in** *subsetD*[*of M carrier D*], *assumption*+,
rule-tac $a = x$ **and** $b = b$ **and** $c = ba$ **in** *le-trans*, *assumption*+)
apply (*cut-tac* $B = M$ **and** $a = ba$ **in** *subset-insertI*,
cut-tac $a = ba$ **in** *insertI1*[*of - M*],
cut-tac $C =$ *insert ba M* **in** *Chain-mem-fTo*, *assumption*)
apply (*frule-tac* $x =$ *insert ba M* **in** *bspec*, *assumption*,
thin-tac $\forall b{\in}$*carrier fTo D*. $M \preceq_{fTo\ D} b \longrightarrow M = b$,
simp del:*insert-iff insert-subset add*:*fTo-Order-sub*)
apply (*frule-tac* $x = ba$ **in** *bspec*, *assumption*,
thin-tac $\forall s{\in}M.\ s \preceq b$)
apply (*frule-tac* $a = b$ **and** $b = ba$ **in** *le-antisym*, *assumption*+, *simp*)
done

lemma (**in** *Order*) *g-Zorn-lemma1*:$[\![$*inductive-set D*; *Chain D C*$]\!] \Longrightarrow \exists m.$ *maximal* $m \wedge m \in$ *upper-bounds D C*
apply (*frule Chain-mem-fTo* [*of C*],
frule Hausdorff-ac[*of C*])
apply (*erule bexE*, *erule conjE*)
apply (*frule-tac* $X = M$ **in** *mem-fTo-Chain*)
apply (*simp add*:*inductive-set-def*)
apply (*frule-tac* $a = M$ **in** *forall-spec*, *assumption*,
thin-tac $\forall C.$ *Chain D C* $\longrightarrow (\exists b.$ *ub C b*),
erule exE, *simp add*:*upper-bound-def*, *erule conjE*)
apply (*frule-tac* $M = M$ **and** $b = b$ **in** *Zorn-lemmaTr*[*of C*], *assumption*+)
apply *blast*
done

lemma (**in** *Order*) *g-Zorn-lemma2*:$[\![$*inductive-set D*; $a \in$ *carrier D*$]\!] \Longrightarrow$
$\exists m{\in}$*carrier D*. *maximal* $m \wedge a \preceq m$
apply (*frule BNTr1* [*of a*],
frule singleton-sub[*of a carrier D*],
frule-tac $X = \{a\}$ **in** *Torder-Chain*,
simp add:*Worder*.*Torder*)
apply (*frule-tac* $C = \{a\}$ **in** *g-Zorn-lemma1*, *assumption*+,
erule exE, *erule conjE*,
simp add:*upper-bounds-def upper-bound-def*)
apply *blast*
done

lemma (**in** *Order*) *g-Zorn-lemma3*:*inductive-set D* $\Longrightarrow \exists m{\in}$*carrier D*. *maximal* m

apply (*cut-tac Iod-empty-Torder*,
cut-tac empty-subsetI[*of carrier D*],
frule Torder-Chain[*of* {}], *assumption*+)
apply (*frule-tac* $C = \{\}$ **in** *g-Zorn-lemma1*, *assumption*+,
simp add:*upper-bounds-def upper-bound-def*,
blast)
done

Chapter 3

Group Theory. Focused on Jordan Hoelder theorem

3.1 Definition of a Group

record *'a Group = 'a carrier +*
  top :: *['a, 'a] ⇒ 'a* (**infixl** ·$_1$ *70*)
  iop :: *'a ⇒ 'a* (*ϱ*$_1$ - *[81] 80*)
  one :: *'a* (**1**$_1$)

locale *Group =*
 fixes *G* (**structure**)
 assumes *top-closed*: *top G ∈ carrier G → carrier G → carrier G*
 and *tassoc* : *⟦a ∈ carrier G; b ∈ carrier G; c ∈ carrier G⟧ ⟹*
 (a · b) · c = a · (b · c)
 and *iop-closed*:*iop G ∈ carrier G → carrier G*
 and *l-i* :*a ∈ carrier G ⟹ (ϱ a) · a = 1*
 and *unit-closed*: *1 ∈ carrier G*
 and *l-unit*:*a ∈ carrier G ⟹ 1 · a = a*

lemma (**in** *Group*) *mult-closed*:*⟦a ∈ carrier G; b ∈ carrier G⟧ ⟹*
 a · b ∈ carrier G
apply (*cut-tac top-closed*)
apply (*frule funcset-mem*[*of op · carrier G carrier G → carrier G a*],
 assumption+,
 frule funcset-mem[*of op · a carrier G carrier G b*],
 assumption+)
done

lemma (**in** *Group*) *i-closed*:*a ∈ carrier G ⟹ (ϱ a) ∈ carrier G*
apply (*cut-tac iop-closed*,
 frule funcset-mem[*of iop G carrier G carrier G a*])
apply *assumption*+
done

lemma (**in** *Group*) *r-mult-eqn*:⟦$a \in$ *carrier* G; $b \in$ *carrier* G;
$c \in$ *carrier* G; $a = b$⟧ $\Longrightarrow a \cdot c = b \cdot c$
by *simp*

lemma (**in** *Group*) *l-mult-eqn*:⟦$a \in$ *carrier* G; $b \in$ *carrier* G;
$c \in$ *carrier* G; $a = b$⟧ $\Longrightarrow c \cdot a = c \cdot b$
by *simp*

lemma (**in** *Group*) *r-i*:$a \in$ *carrier* $G \Longrightarrow$
$a \cdot (\varrho\ a) = \mathbf{1}$
apply (*frule mult-closed*[*of* a ϱ a],
simp add:*i-closed*,
cut-tac l-unit[*of* a],
cut-tac unit-closed,
frule mult-closed[*of* $\mathbf{1}$ a], *assumption*+,
frule i-closed[*of* a],
frule mult-closed[*of* ϱ a a], *assumption*+)

apply (*frule r-mult-eqn*[*of* $(\varrho\ a) \cdot a$ $\mathbf{1}$ ϱ a], *assumption*+,
simp add:*l-i*,
simp add:*l-unit*[*of* ϱ a])

apply (*frule i-closed*[*of* a],
frule i-closed[*of* ϱ a],
frule mult-closed[*of* ϱ a a], *assumption*+,
frule mult-closed[*of* $(\varrho\ a) \cdot a$ ϱ a], *assumption*+,
frule l-mult-eqn[*of* $(\varrho\ a) \cdot a \cdot (\varrho\ a)$ ϱ a ϱ $(\varrho\ a)$],
assumption+)
apply (*thin-tac* $(\varrho\ a) \cdot a \cdot (\varrho\ a) = (\varrho\ a)$,
simp add:*l-i*[*of* ϱ a],
simp add:*tassoc*[*THEN sym*, *of* ϱ $(\varrho\ a)$ $(\varrho\ a) \cdot a$ ϱ a],
simp add:*tassoc*[*THEN sym*, *of* ϱ $(\varrho\ a)$ ϱ a a])
apply (*simp add*:*l-i*[*of* ϱ a])
apply *assumption*
done

lemma (**in** *Group*) *r-unit*:$a \in$ *carrier* $G \Longrightarrow a \cdot \mathbf{1} = a$
apply (*cut-tac unit-closed*,
frule i-closed[*of* a],
frule mult-closed[*of* a $\mathbf{1}$], *assumption*+)
apply (*cut-tac l-i*[*THEN sym*, *of* a],
simp,
thin-tac $\mathbf{1} = \varrho\ a \cdot a$)

apply (*simp add*:*tassoc*[*THEN sym*] *r-i l-unit*)
apply *assumption*
done

lemma (**in** *Group*) *l-i-unique*:⟦$a \in$ *carrier* G; $b \in$ *carrier* G;
$b \cdot a = \mathbf{1}$ ⟧ $\Longrightarrow$ $(\varrho\ a) = b$
apply (*cut-tac unit-closed*,
frule i-closed[*of a*],
frule mult-closed[*of b a*], *assumption*+)
apply (*frule r-mult-eqn*[*of* $b \cdot a\ \mathbf{1}\ \varrho\ a$], *assumption*+)
apply (*thin-tac* $b \cdot a = \mathbf{1}$,
simp add:*tassoc*[*of b a* ϱ *a*] *r-i*)
apply (*simp add*:*l-unit r-unit*)
done

lemma (**in** *Group*) *l-i-i*:$a \in$ *carrier* $G \Longrightarrow (\varrho\ (\varrho\ a)) \cdot (\varrho\ a) = \mathbf{1}$
by (*frule i-closed*[*of a*],
cut-tac l-i[*of* ϱ *a*], *assumption*+)

lemma (**in** *Group*) *l-div-eqn*:⟦$a \in$ *carrier* G; $x \in$ *carrier* G; $y \in$ *carrier* G;
$a \cdot x = a \cdot y$ ⟧ $\Longrightarrow x = y$
apply (*frule mult-closed*[*of a x*], *assumption*+,
frule mult-closed[*of a y*], *assumption*+,
frule i-closed[*of a*],
frule l-mult-eqn[*of* $a \cdot x\ a \cdot y\ \varrho\ a$], *assumption*+)
apply (*thin-tac* $a \cdot x = a \cdot y$,
simp add:*tassoc*[*THEN sym*])
apply (*simp add*:*l-i l-unit*)
done

lemma (**in** *Group*) *r-div-eqn*:⟦$a \in$ *carrier* G; $x \in$ *carrier* G; $y \in$ *carrier* G;
$x \cdot a = y \cdot a$ ⟧ $\Longrightarrow x = y$
apply (*frule mult-closed*[*of x a*], *assumption*+,
frule mult-closed[*of y a*], *assumption*+,
frule i-closed[*of a*],
frule r-mult-eqn[*of* $x \cdot a\ y \cdot a\ \varrho\ a$], *assumption*+)
apply (*thin-tac* $x \cdot a = y \cdot a$,
simp add:*tassoc*, *simp add*:*r-i r-unit*)
done

lemma (**in** *Group*) *l-mult-eqn1*:⟦$a \in$ *carrier* G; $x \in$ *carrier* G; $y \in$ *carrier* G;
$(\varrho\ a) \cdot x = (\varrho\ a) \cdot y$⟧ $\Longrightarrow x = y$
by (*frule i-closed*[*of a*], *rule l-div-eqn*[*of* ϱ *a x y*], *assumption*+)

lemma (**in** *Group*) *tOp-assocTr41*:⟦$a \in$ *carrier* G; $b \in$ *carrier* G; $c \in$ *carrier* G;
$d \in$ *carrier* G⟧ $\Longrightarrow a \cdot b \cdot c \cdot d = a \cdot b \cdot (c \cdot d)$
by (*frule mult-closed*[*of a b*], *assumption*+,
simp add:*tassoc*[*of* $a \cdot b\ c\ d$])

lemma (**in** *Group*) *tOp-assocTr42*:⟦$a \in$ *carrier* G; $b \in$ *carrier* G; $c \in$ *carrier* G;
$d \in$ *carrier* G⟧ $\Longrightarrow a \cdot b \cdot c \cdot d = a \cdot (b \cdot c) \cdot d$
by (*simp add*:*tassoc*[*of a b c*])

lemma (**in** *Group*) *tOp-assocTr44*:⟦$a \in$ *carrier* G; $b \in$ *carrier* G; $c \in$ *carrier* G;
$d \in$ *carrier* G ⟧ ⟹ $(\varrho\ a) \cdot b \cdot ((\varrho\ c) \cdot d) =$
$(\varrho\ \ a) \cdot ((b \cdot (\varrho\ c)) \cdot d)$
apply (*frule i-closed*[*of a*],
frule i-closed[*of c*])
apply (*simp add*:*tassoc*[*of b* ϱ *c d*],
frule mult-closed[*of* ϱ *c d*], *assumption*+,
simp add:*tassoc*[*THEN sym*, *of* ϱ *a b* $(\varrho\ c) \cdot d$])
done

lemma (**in** *Group*) *tOp-assocTr45*:⟦$a \in$ *carrier* G; $b \in$ *carrier* G; $c \in$ *carrier* G;
$d \in$ *carrier* G⟧ ⟹ $a \cdot b \cdot c \cdot d = a \cdot (b \cdot (c \cdot d))$
apply (*frule mult-closed*[*of c d*], *assumption*+)
apply (*simp add*:*tassoc*[*of a b c* $\cdot$ *d*, *THEN sym*])
apply (*simp add*:*tOp-assocTr41*)
done

lemma (**in** *Group*) *one-unique*:⟦$a \in$ *carrier* G; $x \in$ *carrier* G; $x \cdot a = x$⟧ ⟹
$a = \mathbf{1}$
apply (*frule mult-closed*[*of x a*], *assumption*+,
frule i-closed[*of x*],
frule l-mult-eqn[*of* $x \cdot a\ x\ \varrho\ x$], *assumption*+)
apply (*thin-tac* $x \cdot a = x$,
simp add:*tassoc*[*THEN sym*, *of* $\varrho\ x\ x\ a$],
simp add:*l-i l-unit*)
done

lemma (**in** *Group*) *i-one*:$\varrho\ \mathbf{1} = \mathbf{1}$
by (*cut-tac unit-closed*, *frule l-i*[*of* **1**],
frule i-closed[*of* **1**], *simp add*:*r-unit*)

lemma (**in** *Group*) *eqn-inv1*:⟦$a \in$ *carrier* G; $x \in$ *carrier* G; $a = (\varrho\ x)$ ⟧ ⟹
$x = (\varrho\ a)$
apply (*frule i-closed*[*of x*],
frule l-mult-eqn[*of a* ϱ *x x*], *assumption*+,
thin-tac $a = \varrho\ x$, *simp add*:*r-i*,
simp add:*l-i-unique*[*of a x*])
done

lemma (**in** *Group*) *eqn-inv2*:⟦$a \in$ *carrier* G; $x \in$ *carrier* G; $x \cdot a = x \cdot (\varrho\ x)$⟧
⟹
$x = (\varrho\ a)$
apply (*rule eqn-inv1*, *assumption*+)
apply (*rule l-div-eqn*[*of x a* ϱ *x*], *assumption*+,
simp add:*i-closed*, *assumption*)
done

lemma (**in** *Group*) *r-one-unique*:⟦$a \in$ *carrier* G; $x \in$ *carrier* G; $a \cdot x = a$⟧ ⟹

$x = \mathbf{1}$
apply (*frule mult-closed*[*of a x*], *assumption*+,
frule i-closed[*of a*],
frule l-mult-eqn[*of a* · *x a* ϱ *a*], *assumption*+,
thin-tac a · *x* = *a*,
simp add:*tassoc*[*THEN sym*] *l-i l-unit*)
done

lemma (**in** *Group*) *r-i-unique*:⟦*a* ∈ *carrier G*; *x* ∈ *carrier G*; *a* · *x* = **1**⟧ ⟹
x = (ϱ *a*)
apply (*cut-tac unit-closed*,
frule mult-closed[*of a x*], *assumption*+,
frule i-closed[*of x*],
frule r-mult-eqn[*of a* · *x* **1** ϱ *x*], *assumption*+,
thin-tac a · *x* = **1**,
simp add:*tassoc*[*of a x* ϱ *x*] *r-i r-unit l-unit*)
apply (*simp add*:*eqn-inv1*)
done

lemma (**in** *Group*) *iop-i-i*:*a* ∈ *carrier G* ⟹ ϱ (ϱ *a*) = *a*
apply (*frule i-closed*[*of a*], *frule i-closed*[*of* ϱ *a*],
frule l-i[*of* ϱ *a*],
frule mult-closed[*of* ϱ (ϱ *a*) ϱ *a*], *assumption*+)
apply (*cut-tac unit-closed*,
frule r-mult-eqn[*of* ϱ (ϱ *a*) · ϱ *a* **1** *a*], *assumption*+,
thin-tac ϱ (ϱ *a*) · ϱ *a* = **1**,
simp only:*tassoc*)
apply (*simp add*:*l-i r-unit l-unit*)
done

lemma (**in** *Group*) *i-ab*:⟦*a* ∈ *carrier G*; *b* ∈ *carrier G*⟧ ⟹
ϱ (*a* · *b*) = (ϱ *b*) · (ϱ *a*)
apply (*frule mult-closed*[*of a b*], *assumption*+,
frule i-closed[*of a* · *b*],
frule i-closed[*of a*], *frule i-closed*[*of b*],
frule l-div-eqn[*of a* · *b* ϱ (*a* · *b*) (ϱ *b*) · (ϱ *a*)], *assumption*+,
simp add:*mult-closed*, *simp add*:*r-i*[*of a* · *b*],
simp add:*tOp-assocTr41*[*THEN sym*], *simp add*:*tOp-assocTr42*,
simp add:*r-i r-unit*)
apply *assumption*
done

lemma (**in** *Group*) *sol-eq-l*:⟦*a* ∈ *carrier G*; *b* ∈ *carrier G*; *x* ∈ *carrier G*;
a · *x* = *b*⟧ ⟹ *x* = (ϱ *a*) · *b*
apply (*frule mult-closed*[*of a x*], *assumption*+,
frule i-closed[*of a*],
frule l-mult-eqn[*of a* · *x b* ϱ *a*], *assumption*+)
apply (*thin-tac a* · *x* = *b*,
simp add:*tassoc*[*THEN sym*],

simp add:*l-i l-unit*)
done

lemma (**in** *Group*) *sol-eq-r*:⟦$a \in$ *carrier* G; $b \in$ *carrier* G; $x \in$ *carrier* G;
x $\cdot$ *a* = *b* ⟧ ⟹ *x* = *b* $\cdot$ (ϱ *a*)
apply (*frule mult-closed*[*of x a*], *assumption*+,
frule i-closed[*of a*],
frule r-mult-eqn[*of x* $\cdot$ *a b* ϱ *a*], *assumption*+)
apply (*thin-tac x* $\cdot$ *a* = *b*,
simp add:*tassoc*,
simp add:*r-i r-unit*)
done

lemma (**in** *Group*) *r-div-eq*:⟦$a \in$ *carrier* G; $b \in$ *carrier* G; *a* $\cdot$ (ϱ *b*) = **1**⟧ ⟹
a = *b*
apply (*frule i-closed*[*of b*],
frule mult-closed[*of a* ϱ *b*], *assumption*+,
cut-tac unit-closed,
frule r-mult-eqn[*of a* $\cdot$ (ϱ *b*) **1** *b*], *assumption*+)
apply (*thin-tac a* $\cdot$ ϱ *b* = **1**,
simp add:*tassoc l-i r-i*, *simp add*:*l-unit r-unit*)
done

lemma (**in** *Group*) *l-div-eq*:⟦$a \in$ *carrier* G; $b \in$ *carrier* G; (ϱ *a*) $\cdot$ *b* = **1**⟧ ⟹
a = *b*
apply (*frule i-closed*[*of a*],
frule mult-closed[*of* ϱ *a b*], *assumption*+,
cut-tac unit-closed,
frule l-mult-eqn[*of* (ϱ *a*) $\cdot$ *b* **1** *a*], *assumption*+)
apply (*thin-tac* ϱ *a* $\cdot$ *b* = **1**,
simp add:*tassoc*[*THEN sym*] *r-i l-unit r-unit*)
done

lemma (**in** *Group*) *i-m-closed*:⟦$a \in$ *carrier* G ; $b \in$ *carrier* G⟧ ⟹
(ϱ *a*) $\cdot$ *b* $\in$ *carrier* G
by (*rule mult-closed*,
simp add:*i-closed*, *assumption*)

3.2 Subgroups

definition
sg ::[- , 'a *set*] ⇒ *bool* (- ≫ - [60, 61]60) **where**
$G \gg H \longleftrightarrow H \neq \{\} \wedge H \subseteq$ *carrier* $G \wedge (\forall a \in H.\ \forall b \in H.\ a \cdot_G (\varrho_G\ b) \in H)$

definition
Gp :: - ⇒ 'a *set* ⇒ - ((↳1-) 70) **where**
$\natural_G H \equiv G$ (| *carrier* := *H*, *top* := *top* *G*, *iop* := *iop* *G*, *one* := *one* *G*|)

definition
rcs :: [- , *'a set*, *'a*] ⇒ *'a set* (**infix** ·₁ *70*) **where**
$H \cdot_G a = \{b.\ \exists\ h \in H.\ h \cdot_G a = b\}$

definition
lcs :: [- , *'a*, *'a set*] ⇒ *'a set* (**infix** ◇₁ *70*) **where**
$a \diamondsuit_G H = \{b.\ \exists\ h \in H.\ a \cdot_G h = b\}$

definition
nsg :: - ⇒ *'a set* ⇒ *bool* (- ▷ - [*60*,*61*]*60*) **where**
$G \triangleright H \longleftrightarrow G \gg H \wedge (\forall x \in carrier\ G.\ H \cdot_G x = x \diamondsuit_G H)$

definition
set-rcs :: [- , *'a set*] ⇒ *'a set set* **where**
set-rcs G H = $\{C.\ \exists a \in carrier\ G.\ C = H \cdot_G a\}$

definition
c-iop :: [- , *'a set*] ⇒ *'a set* ⇒ *'a set* **where**
c-iop G H = $(\lambda X \in set\text{-}rcs\ G\ H.\ \{z.\ \exists\ x \in X\ .\ \exists h \in H.\ h \cdot_G (\varrho_G\ x) = z\})$

definition
c-top :: [-, *'a set*] ⇒ (['*a set*, *'a set*] ⇒ *'a set*) **where**
c-top G H = $(\lambda X \in set\text{-}rcs\ G\ H.\ \lambda Y \in set\text{-}rcs\ G\ H.$
$\{z.\ \exists x \in X.\ \exists\ y \in Y.\ x \cdot_G y = z\})$

lemma (**in** *Group*) *sg-subset*:$G \gg H \Longrightarrow H \subseteq carrier\ G$
by (*simp add*:*sg-def*)

lemma (**in** *Group*) *one-Gp-one*:$G \gg H \Longrightarrow \mathbf{1}_{(Gp\ G\ H)} = \mathbf{1}$
by (*simp add*:*Gp-def*)

lemma (**in** *Group*) *carrier-Gp*:$G \gg H \Longrightarrow (carrier\ (\natural H)) = H$
by (*simp add*:*Gp-def*)

lemma (**in** *Group*) *sg-subset-elem*:$[\![G \gg H;\ h \in H\ ]\!] \Longrightarrow h \in carrier\ G$
by (*frule sg-subset* [*of H*], *simp only*:*subsetD*)

lemma (**in** *Group*) *sg-mult-closedr*:$[\![G \gg H;\ x \in carrier\ G;\ h \in H]\!] \Longrightarrow$
$x \cdot h \in carrier\ G$
apply (*frule sg-subset-elem* [*of H h*], *assumption*+)
apply (*simp add*:*mult-closed*)
done

lemma (**in** *Group*) *sg-mult-closedl*:$[\![G \gg H;\ x \in carrier\ G;\ h \in H]\!] \Longrightarrow$
$h \cdot x \in carrier\ G$

apply (*frule sg-subset-elem*[*of H h*], *assumption*+)
apply (*simp add*:*mult-closed*)
done

lemma (**in** *Group*) *sg-condTr1*:⟦$H \subseteq$ *carrier G*; $H \neq \{\}$;
$\forall a.\ \forall b.\ a \in H \wedge b \in H \longrightarrow\ a \cdot (\varrho\ b) \in H$⟧ $\Longrightarrow$ $\mathbf{1} \in H$
apply (*frule nonempty-ex* [*of H*], *erule exE*)
apply (*frule-tac* $x = x$ **in** *spec*,
thin-tac $\forall a\ b.\ a \in H \wedge b \in H \longrightarrow a \cdot \varrho\ b \in H$,
frule-tac $x = x$ **in** *spec*,
thin-tac $\forall b.\ x \in H \wedge b \in H \longrightarrow x \cdot \varrho\ b \in H$)
apply (*frule-tac* $c = x$ **in** *subsetD*[*of H carrier G*], *assumption*+,
simp add:*r-i*)
done

lemma (**in** *Group*) *sg-unit-closed*:$G \gg H \Longrightarrow \mathbf{1} \in H$
apply (*simp add*:*sg-def*, (*erule conjE*)+,
rule sg-condTr1, *assumption*+, *blast*)
done

lemma (**in** *Group*) *sg-i-closed*:⟦$G \gg H$; $x \in H$⟧ $\Longrightarrow$ $(\varrho\ x) \in H$
apply (*frule sg-unit-closed*,
frule sg-subset-elem[*of H x*], *assumption*,
simp add:*sg-def*, (*erule conjE*)+)
apply (*frule-tac* $x = \mathbf{1}$ **in** *bspec*, *assumption*,
rotate-tac -1,
frule-tac $x = x$ **in** *bspec*, *assumption*,
thin-tac $\forall b{\in}H.\ \mathbf{1} \cdot \varrho\ b \in H$,
thin-tac $\forall a{\in}H.\ \forall b{\in}H.\ a \cdot \varrho\ b \in H$)
apply (*frule i-closed*[*of x*],
simp add:*l-unit*)
done

lemma (**in** *Group*) *sg-mult-closed*:⟦$G \gg H$; $x \in H$; $y \in H$⟧ $\Longrightarrow$
$x \cdot y \in H$
apply (*frule sg-i-closed*[*of H y*], *assumption*,
simp add:*sg-def*, (*erule conjE*)+)
apply (*frule-tac* $x = x$ **in** *bspec*, *assumption*,
rotate-tac -1,
frule-tac $x = \varrho\ y$ **in** *bspec*, *assumption*,
thin-tac $\forall b{\in}H.\ x \cdot \varrho\ b \in H$,
thin-tac $\forall a{\in}H.\ \forall b{\in}H.\ a \cdot \varrho\ b \in H$)
apply (*frule subsetD*[*of H carrier G*], *assumption*+,
simp add:*iop-i-i*)
done

lemma (**in** *Group*) *nsg-sg*: $G \triangleright H \Longrightarrow G \gg H$
by (*simp add*:*nsg-def*)

lemma (**in** *Group*) *nsg-subset*:$G \triangleright N \implies N \subseteq$ *carrier* G
apply (*simp add*:*nsg-def*, (*erule conjE*)+)
apply (*simp add*:*sg-subset*)
done

lemma (**in** *Group*) *nsg-lr-cst-eq*:$[\![G \triangleright N;\ a \in$ *carrier* $G]\!] \implies$
$a \diamondsuit N = N \cdot a$
by (*simp add*: *nsg-def*)

lemma (**in** *Group*) *sg-i-m-closed*:$[\![G \gg H;\ a \in H\ ;\ b \in H]\!] \implies (\varrho\ a) \cdot b \in H$
apply (*rule sg-mult-closed*, *assumption*+,
simp add:*sg-i-closed*, *assumption*)
done

lemma (**in** *Group*) *sg-m-i-closed*:$[\![G \gg H;\ a \in H\ ;\ b \in H\ ]\!] \implies a \cdot (\varrho\ b) \in H$
apply (*simp add*:*sg-def*)
done

definition
sg-gen :: [- , *'a set*] ⇒ *'a set* **where**
sg-gen $G\ A = \bigcap\{H.\ G \gg H \wedge A \subseteq H\}$

lemma (**in** *Group*) *smallest-sg-gen*:$[\![A \subseteq$ *carrier* $G;\ G \gg H;\ A \subseteq H]\!] \implies$
sg-gen $G\ A \subseteq H$
apply (*simp add*:*sg-gen-def*)
apply *auto*
done

lemma (**in** *Group*) *special-sg-G*: $G \gg$ (*carrier* G)
apply (*simp add*:*sg-def*,
cut-tac unit-closed, *simp add*:*nonempty*)
apply ((*rule ballI*)+, *simp add*:*mult-closed i-closed*)
done

lemma (**in** *Group*) *special-sg-self*: $G = \natural$(*carrier* G)
by (*simp add*:*Gp-def*)

lemma (**in** *Group*) *special-sg-e*: $G \gg \{\mathbf{1}\}$
apply (*simp add*:*sg-def*)
apply (*simp add*:*unit-closed i-one l-unit*)
done

lemma (**in** *Group*) *inter-sgs*:$[\![G \gg H;\ G \gg K]\!] \implies G \gg (H \cap K)$
apply (*frule sg-unit-closed*[*of H*],
frule sg-unit-closed[*of K*])
apply (*simp add*:*sg-def*)
apply *auto*
done

lemma (**in** *Group*) *subg-generated*:$A \subseteq$ *carrier* $G \implies G \gg$ (*sg-gen* G A)
apply (*simp add*:*sg-def*)
apply (*rule conjI*,
simp add:*sg-gen-def*,
rule ex-nonempty, *simp*)
apply (*rule contrapos-pp*, *simp*+,
frule-tac $x = \mathbf{1}$ **in** *spec*,
thin-tac $\forall x.\ \exists xa.\ G \gg xa\ \wedge A \subseteq xa \wedge x \notin xa$,
erule exE, (*erule conjE*)+,
frule-tac $H = x$ **in** *sg-unit-closed*, *simp*)

apply (*rule conjI*)
apply (*cut-tac special-sg-G*,
simp add:*sg-gen-def*,
rule subsetI,
blast)

apply ((*rule ballI*)+,
simp add:*sg-gen-def*,
rule allI, *rule impI*,
frule-tac $a = x$ **in** *forall-spec*, *assumption*,
thin-tac $\forall x.\ G \gg x\ \wedge A \subseteq x \longrightarrow a \in x$,
frule-tac $a = x$ **in** *forall-spec*, *assumption*,
thin-tac $\forall x.\ G \gg x\ \wedge A \subseteq x \longrightarrow b \in x$)
apply (*simp add*:*sg-m-i-closed*)
done

definition
Qg :: [- , *'a set*] ⇒
(⦇*carrier*:: *'a set set*, *top*:: [*'a set*, *'a set*] ⇒ *'a set*,
iop:: *'a set* ⇒ *'a set*, *one*:: *'a set* ⦈) **where**
Qg G H = (⦇*carrier* = *set-rcs* G H, *top* = *c-top* G H, *iop* = *c-iop* G H, *one* = H ⦈)

definition
Pj :: [- , *'a set*] ⇒ (*'a* => *'a set*) **where**
Pj G H = ($\lambda x \in$ *carrier* G. H ·$_G$ x)

no-notation *inverse-divide* (**infixl** *'/ 70*)

abbreviation
QGRP :: ([(*'a*, *'more*) *Group-scheme*, *'a set*] => (*'a set*) *Group*)
(**infixl** *'/ 70*) **where**
G / H == *Qg* G H

definition
gHom ::[(*'a*, *'more*) *Group-scheme*, (*'b*, *'more1*) *Group-scheme*] ⇒

$('a \Rightarrow 'b)$ *set* **where**
gHom G F = $\{f.\ (f \in$ *extensional* $($*carrier G*$) \wedge f \in$ *carrier G* $\rightarrow$ *carrier F*$) \wedge$
$(\forall x \in$ *carrier G*$.\ \forall y \in$ *carrier G*$.\ f\ (x \cdot_G y) = (f\ x) \cdot_F (f\ y))\}$

definition
gkernel ::$[('a, 'more)$ *Group-scheme* , $('b, 'more1)$ *Group-scheme*, $'a \Rightarrow 'b]$
$\Rightarrow 'a$ *set* **where**
gkernel G F f = $\{x.\ (x \in$ *carrier G*$) \wedge (f\ x = \mathbf{1}_F)\}$

definition
iim :: $[('a, 'more)$ *Group-scheme*, $('b, 'more1)$ *Group-scheme*, $'a \Rightarrow 'b$,
$'b$ *set*$]\ \Rightarrow 'a$ *set* **where**
iim G F f K = $\{x.\ (x \in$ *carrier G*$) \wedge (f\ x \in K)\}$

abbreviation
GKER :: $[('a, 'more)$ *Group-scheme*, $('b, 'more1)$ *Group-scheme*, $'a \Rightarrow 'b\ ] \Rightarrow 'a$
set
$((3ger_{_,_}\ _)\ [88,88,89]88)$ **where**
$ger_{G,F}\ f$ == *gkernel G F f*

definition
gsurjec :: $[('a, 'more)$ *Group-scheme*, $('b, 'more1)$ *Group-scheme*,
$'a \Rightarrow 'b] \Rightarrow$ *bool* $((3gsurj_{_,_}\ _)\ [88,88,89]88)$ **where**
$gsurj_{F,G}\ f \longleftrightarrow f \in$ *gHom F G* $\wedge$ *surj-to f* $($*carrier F*$)$ $($*carrier G*$)$

definition
ginjec :: $[('a, 'more)$ *Group-scheme*, $('b, 'more1)$ *Group-scheme*,
$'a \Rightarrow 'b]\ \Rightarrow$ *bool* $((3ginj_{_,_}\ _)\ [88,88,89]88)$ **where**
$ginj_{F,G}\ f \longleftrightarrow f \in$ *gHom F G* $\wedge$ *inj-on f* $($*carrier F*$)$

definition
gbijec :: $[('a, 'm)$ *Group-scheme*, $('b, 'm1)$ *Group-scheme*, $'a \Rightarrow 'b]$
$\Rightarrow$ *bool* $((3gbij_{_,_}\ _)\ [88,88,89]88)$ **where**
$gbij_{F,G}\ f \longleftrightarrow gsurj_{F,G}\ f \wedge ginj_{F,G}\ f$

definition
Ug :: - $\Rightarrow$ $('a, 'more)$ *Group-scheme* **where**
Ug G = $\natural_G\ \{\mathbf{1}_G\}$

definition
Ugp :: - $\Rightarrow$ *bool* **where**
Ugp G == *Group G* $\wedge$ *carrier G* = $\{\mathbf{1}_G\}$

definition
isomorphic :: $[('a, 'm)$ *Group-scheme*, $('b, 'm1)$ *Group-scheme*$]$
$\Rightarrow$ *bool* $($**infix** $\cong$ *100*$)$ **where**
$F \cong G \longleftrightarrow (\exists f.\ gbij_{F,G}\ f)$

definition
constghom :: [('*a*, '*m*) *Group-scheme*, ('*b*, '*m1*) *Group-scheme*]
⇒ ('*a* ⇒ '*b*) (('*1*'-,-) [*88*,*89*]*88*) **where**
$1_{F,G}$ = (λ*x*∈*carrier F*. $\mathbf{1}_G$)

definition
cmpghom :: [('*a*, '*m*) *Group-scheme*, '*b* ⇒ '*c*, '*a* ⇒ '*b*] ⇒ '*a* ⇒ '*c* **where**
cmpghom F g f = *compose* (*carrier F*) *g f*

abbreviation
GCOMP :: ['*b* ⇒ '*c*, ('*a*, '*m*) *Group-scheme*, '*a* ⇒ '*b*] ⇒ '*a* ⇒ '*c*
((*3*- ∘_ -) [*88*, *88*, *89*]*88*) **where**
g $\circ_F$ *f* == *cmpghom F g f*

lemma *Group-Ugp*:*Ugp G* ⟹ *Group G*
by (*simp add*:*Ugp-def*)

lemma (**in** *Group*) *r-mult-in-sg*:⟦*G* ⋗ *H*; *a* ∈ *carrier G*; *x* ∈ *carrier G*; *x* · *a* ∈ *H*⟧
⟹ ∃ *h* ∈ *H*. *h* · (ϱ *a*) = *x*
apply (*frule inEx*[*of x* · *a H*],
erule bexE)
apply (*rotate-tac* −*1*, *frule sym*, *thin-tac y* = *x* · *a*,
frule-tac h = *y* **in** *sg-subset-elem*[*of H*], *assumption*+,
frule-tac b1 = *y* **in** *sol-eq-r*[*THEN sym*, *of a* - *x*], *assumption*+)
apply *blast*
done

lemma (**in** *Group*) *r-unit-sg*:⟦*G* ⋗ *H*; *h* ∈ *H*⟧ ⟹ *h* · **1** = *h*
by (*frule sg-subset-elem* [*of H h*], *assumption*,
simp add:*r-unit*)

lemma (**in** *Group*) *sg-l-unit*:⟦*G* ⋗ *H*; *h* ∈ *H*⟧ ⟹ **1** · *h* = *h*
by (*frule sg-subset-elem* [*of H h*], *assumption*+, *simp add*:*l-unit*)

lemma (**in** *Group*) *sg-l-i*:⟦*G* ⋗ *H*; *x* ∈ *H* ⟧ ⟹ (ϱ *x*) · *x* = **1**
by (*frule sg-subset-elem*[*of H x*], *assumption*+,
simp add:*l-i*)

lemma (**in** *Group*) *sg-tassoc*:⟦*G* ⋗ *H*; *x* ∈ *H*; *y* ∈ *H*; *z* ∈ *H*⟧ ⟹
x · *y* · *z* = *x* · (*y* · *z*)
apply (*frule sg-subset-elem*[*of H x*], *assumption*+,
frule sg-subset-elem[*of H y*], *assumption*+,
frule sg-subset-elem[*of H z*], *assumption*+)
apply (*simp add*:*tassoc*)
done

lemma (**in** *Group*) *sg-condition*:⟦*H* ⊆ *carrier G*; *H* ≠ {};

$\forall\, a.\ \forall\, b.\ a \in H \wedge b \in H \longrightarrow\ a \cdot (\varrho\ b) \in H\rrbracket \Longrightarrow G \gg H$
by (*simp add*:*sg-def*)

definition
Gimage :: [('*a*, '*m*) *Group-scheme*, ('*b*, '*m1*) *Group-scheme*, '*a* ⇒ '*b*] ⇒
('*b*, '*m1*) *Group-scheme* **where**
Gimage F G f = *Gp G* (*f* ‘(*carrier F*))

abbreviation
GIMAGE :: [('*a*, '*m*) *Group-scheme*, ('*b*, '*m1*) *Group-scheme*,
'*a* ⇒ '*b*] ⇒ ('*b*, '*m1*) *Group-scheme* ((*3Img*$_{-,-}$ -) [*88*,*88*,*89*]*88*) **where**
$Img_{F,G}\ f == Gimage\ F\ G\ f$

lemma (**in** *Group*) *Group-Gp*:*G* ≫ *H* ⟹ *Group* (♮ *H*)
apply (*simp add*:*Group-def Gp-def*)
apply (*simp add*:*sg-tassoc sg-l-i sg-unit-closed sg-l-unit*
sg-mult-closed sg-i-closed)
done

lemma (**in** *Group*) *Gp-carrier*:*G* ≫ *H* ⟹ *carrier* (*Gp G H*) = *H*
by (*simp add*:*Gp-def*)

lemma (**in** *Group*) *sg-sg*:⟦*G* ≫ *K*; *G* ≫ *H*; *H* ⊆ *K*⟧ ⟹ *Gp G K* ≫ *H*
apply (*simp add*:*sg-def* [*of Gp G K H*])
apply (*rule conjI*, *simp add*:*sg-def*)
apply (*simp add*:*Gp-def*)
apply ((*rule ballI*)+, *simp add*:*sg-m-i-closed*)
done

lemma (**in** *Group*) *sg-subset-of-subG*:⟦*G* ≫ *K*; *Gp G K* ≫ *H*⟧ ⟹ *H* ⊆ *K*
by (*simp add*:*sg-def*[*of* ♮ *K*], *simp add*:*Gp-def*)

lemma *const-ghom*:⟦*Group F*; *Group G*⟧ ⟹ $1_{F,G} \in gHom\ F\ G$
apply (*simp add*:*gHom-def constghom-def*)
apply (*simp add*:*Pi-def Group.unit-closed*)
apply ((*rule ballI*)+,
cut-tac Group.unit-closed[*of G*],
simp add:*Group.mult-closed Group.l-unit*)
apply *assumption*
done

lemma (**in** *Group*) *const-gbij*:$gbij_{(♮\ \{\mathbf{1}\}),(♮\ \{\mathbf{1}\})}\ (1_{(♮\{\mathbf{1}\}),(♮\{\mathbf{1}\})})$
apply (*cut-tac special-sg-e*,
frule Group-Gp[*of* {**1**}],
frule const-ghom[*of* ♮ {**1**} ♮ {**1**}], *assumption*)
apply (*simp add*:*gbijec-def*)
apply (*rule conjI*, *simp add*:*gsurjec-def*,
simp add:*surj-to-def Gp-def constghom-def*)
apply (*simp add*:*ginjec-def inj-on-def Gp-def*)

done

lemma (**in** *Group*) *unit-Groups-isom*: $(\natural\ \{\mathbf{1}\}) \cong (\natural\ \{\mathbf{1}\})$
apply (*unfold isomorphic-def*,
  cut-tac const-gbij, *blast*)
done

lemma *Ugp-const-gHom*:$[\![Ugp\ G;\ Ugp\ E]\!] \Longrightarrow (\lambda x{\in}carrier\ G.\ \mathbf{1}_E) \in gHom\ G\ E$
apply (*simp add*:*gHom-def*)
apply (*rule conjI*)
apply (*rule Pi-I*)
apply (*simp add*:*Group.unit-closed*[*of E*] *Ugp-def*)
apply (*simp add*:*Ugp-def*, (*erule conjE*)+)
apply (*cut-tac Group.unit-closed*[*of G*], *cut-tac Group.unit-closed*[*of E*])
apply (*simp add*:*Group.l-unit*) **apply** *assumption*+
done

lemma *Ugp-const-gbij*:$[\![Ugp\ G;\ Ugp\ E]\!] \Longrightarrow gbij_{G,E}\ (\lambda x{\in}carrier\ G.\ \mathbf{1}_E)$
apply (*simp add*:*gbijec-def*)
apply (*simp add*:*gsurjec-def ginjec-def*)
apply (*simp add*:*Ugp-const-gHom*)
apply (*rule conjI*)
apply (*simp add*:*surj-to-def*, *simp add*:*Ugp-def*)

apply (*simp add*:*inj-on-def*)
apply ((*rule ballI*)+, *simp add*:*Ugp-def*)
done

lemma *Ugps-isomorphic*:$[\![Ugp\ G;\ Ugp\ E]\!] \Longrightarrow G \cong E$
apply (*simp add*:*isomorphic-def*)
apply (*frule-tac Ugp-const-gbij*[*of G E*], *assumption*+)
apply *blast*
done

lemma (**in** *Group*) *Gp-mult-induced*:$[\![G \gg L;\ a \in L;\ b \in L]\!] \Longrightarrow$
  $a\ \cdot_{(Gp\ G\ L)}\ b = a \cdot b$
by (*simp add*:*Gp-def*)

lemma (**in** *Group*) *sg-i-induced*:$[\![G \gg L;\ a \in L]\!] \Longrightarrow \varrho_{(Gp\ G\ L)}\ a = \varrho\ a$
by (*simp add*:*Gp-def*)

lemma (**in** *Group*) *Gp-mult-induced1*:$[\![G \gg H\ ;\ G \gg K;\ a \in H \cap K;\ b \in H \cap K]\!]$

  $\Longrightarrow a\ \cdot_{\natural(H \cap K)}\ b = a\ \cdot_{(\natural H)}\ b$
by (*simp add*:*Gp-def*)

lemma (**in** *Group*) *Gp-mult-induced2*:$[\![G \gg H\ ;\ G \gg K;\ a \in H \cap K;\ b \in H \cap K]\!]$

$\Longrightarrow a \cdot_{\natural(H \cap K)} b = a \cdot_{(\natural K)} b$
by (*simp add*:*Gp-def*)

lemma (**in** *Group*) *sg-i-induced1*:⟦$G \gg H$; $G \gg K$; $a \in H \cap K$⟧
$\Longrightarrow \varrho_{\natural(H \cap K)}\ a = \varrho_{(\natural H)}\ a$
by (*simp add*:*Gp-def*)

lemma (**in** *Group*) *sg-i-induced2*:⟦$G \gg H$; $G \gg K$; $a \in H \cap K$⟧
$\Longrightarrow \varrho_{\natural(H \cap K)}\ a = \varrho_{\natural K}\ a$
by (*simp add*:*Gp-def*)

lemma (**in** *Group*) *subg-sg-sg*:⟦$G \gg K$; $(Gp\ G\ K) \gg H$ ⟧ $\Longrightarrow G \gg H$
apply (*frule sg-subset-of-subG*[*of K H*], *assumption*+,
simp add:*sg-def* [*of* - *H*])
apply (*simp add*:*Gp-def*[*of* - *K*],
frule sg-subset[*of K*], *simp add*:*subset-trans*[*of H K carrier G*])
done

lemma (**in** *Group*) *Gp-inherited*:⟦$G \gg K$; $G \gg L$; $K \subseteq L$⟧ $\Longrightarrow$
$Gp\ (Gp\ G\ L)\ K = Gp\ G\ K$
by (*simp add*:*Gp-def*)

3.3 Cosets

lemma (**in** *Group*) *mem-lcs*:⟦$G \gg H$; $a \in carrier\ G$; $x \in a \diamondsuit H$⟧ $\Longrightarrow$
$x \in carrier\ G$
by (*simp add*: *lcs-def*, *erule bexE*,
frule-tac h = *h* **in** *sg-mult-closedr*[*of H a*], *assumption*+, *simp*)

lemma (**in** *Group*) *lcs-subset*:⟦$G \gg H$; $a \in carrier\ G$⟧ $\Longrightarrow a \diamondsuit H \subseteq carrier\ G$
apply (*simp add*:*lcs-def*,
rule subsetI, *simp*, *erule bexE*)
apply (*frule-tac h* = *h* **in** *sg-subset-elem*[*of H*], *assumption*+,
frule-tac a = *a* **and** *b* = *h* **in** *mult-closed*, *assumption*+)
apply *simp*
done

lemma (**in** *Group*) *a-in-lcs*:⟦$G \gg H$; $a \in carrier\ G$⟧ $\Longrightarrow a \in a \diamondsuit H$
apply (*simp add*: *lcs-def*,
rule bexI [*of* - **1**],
subst r-unit, *assumption*+, *simp*)
apply (*simp add*:*sg-unit-closed*)
done

lemma (**in** *Group*) *eq-lcs1*:⟦$G \gg H$; $a \in carrier\ G$; $b \in carrier\ G$;
$x \in a \diamondsuit H$; $a \diamondsuit H = b \diamondsuit H$⟧ $\Longrightarrow\ x \in b \diamondsuit H$
by *simp*

lemma (**in** *Group*) *eq-lcs2*:⟦$G \gg H$; $a \in$ *carrier* G; $b \in$ *carrier* G;
$a \diamondsuit H = b \diamondsuit H$⟧ $\Longrightarrow a \in b \diamondsuit H$
by (*frule a-in-lcs*[*of H a*], *assumption*+, *simp*)

lemma (**in** *Group*) *lcs-mem-ldiv*:⟦$G \gg H$; $a \in$ *carrier* G; $b \in$ *carrier* G⟧ $\Longrightarrow$
$(a \in b \diamondsuit H) = ((\varrho\ b) \cdot a \in H)$
apply (*rule iffI*)
apply (*simp add*: *lcs-def*, *erule bexE*)
apply (*frule-tac* $x = h$ **in** *sol-eq-l*[*of b a*], *assumption*+,
simp add:*sg-subset-elem*[*of H*], *assumption*+)
apply (*thin-tac* $b \cdot h = a$, *simp*)

apply (*frule-tac* $x = (\varrho\ b) \cdot a$ **and** $A = H$ **in** *inEx*,
erule bexE,
frule-tac $h = y$ **in** *sg-subset-elem*[*of H*], *assumption*+,
frule sg-subset-elem[*of H* $(\varrho\ b) \cdot a$], *assumption*+)
apply (*frule-tac* $a = y$ **in** *l-mult-eqn*[*of* - $(\varrho\ b) \cdot a\ b$], *assumption*+,
frule i-closed[*of b*],
thin-tac $y = \varrho\ b \cdot a$,
simp add:*tassoc*[*THEN sym*], *simp add*:*r-i l-unit*)
apply (*rotate-tac* -2, *frule sym*, *thin-tac* $b \cdot y = a$, *simp add*:*lcs-def*,
blast)
done

lemma (**in** *Group*) *lcsTr5*:⟦$G \gg H$; $a \in$ *carrier* G; $b \in$ *carrier* G;
$(\varrho\ a) \cdot b \in H$; $x \in a \diamondsuit H$⟧ $\Longrightarrow ((\varrho\ b) \cdot x) \in H$
apply (*frule mem-lcs*[*of H a x*], *assumption*+,
subst lcs-mem-ldiv[*THEN sym*, *of H x b*], *assumption*+,
simp add:*lcs-def*, *erule bexE*)
apply (*frule-tac* $h = h$ **in** *sg-subset-elem*[*of H*], *assumption*+,
frule-tac $x = h$ **in** *sol-eq-l*[*of a x*], *assumption*+,
frule sg-i-m-closed[*of H* $(\varrho\ a) \cdot b\ (\varrho\ a) \cdot x$], *assumption*+,
rotate-tac -1, *frule sym*, *thin-tac* $h = \varrho\ a \cdot x$, *simp*)
apply (*frule i-closed*[*of a*],
simp add:*i-ab iop-i-i*, *frule i-closed*[*of b*],
simp add:*tassoc*[*of* $\varrho\ b\ a$],
simp add:*tassoc*[*THEN sym*, *of a* $\varrho\ a\ x$] *r-i l-unit*)
apply (*frule inEx*[*of* $(\varrho\ b) \cdot x$], *erule bexE*,
rotate-tac -1, *frule sym*, *thin-tac* $y = \varrho\ b \cdot x$,
frule-tac $b = y$ **in** *sol-eq-l*[*of* $\varrho\ b$ - x],
simp add:*sg-subset-elem*, *assumption*+,
simp add:*iop-i-i*, *blast*)
done

lemma (**in** *Group*) *lcsTr6*:⟦$G \gg H$; $a \in$ *carrier* G; $b \in$ *carrier* G;
$(\varrho\ a) \cdot b \in H$; $x \in a \diamondsuit H$⟧ $\Longrightarrow\ x \in b \diamondsuit H$
by (*frule lcsTr5*[*of H a b x*], *assumption*+,

subst lcs-mem-ldiv, *assumption+*, *rule mem-lcs*, *assumption+*)

lemma (**in** *Group*) *lcs-Unit1*:$G \gg H \Longrightarrow \mathbf{1} \diamondsuit H = H$
apply (*rule equalityI*)
apply (*rule subsetI*, *simp add*:*lcs-def*, *erule bexE*,
frule-tac $h = h$ **in** *sg-subset-elem*[*of H*], *assumption+*,
simp add:*l-unit*)
apply (*rule subsetI*,
simp add:*lcs-def*,
frule-tac $h = x$ **in** *sg-subset-elem*[*of H*], *assumption+*,
frule-tac $a = x$ **in** *l-unit*)
apply *blast*
done

lemma (**in** *Group*) *lcs-Unit2*:$[\![G \gg H;\ h \in H]\!] \Longrightarrow h \diamondsuit H = H$
apply (*rule equalityI*)
apply (*rule subsetI*, *simp add*:*lcs-def*, *erule bexE*,
frule-tac $x = h$ **and** $y = ha$ **in** *sg-mult-closed*, *assumption+*, *simp*)
apply (*rule subsetI*,
simp add:*lcs-def*,
rule-tac $x = (\varrho\ h) \cdot x$ **in** *bexI*[*of - - H*])
apply (*frule sg-subset-elem*[*of H h*], *assumption+*,
frule i-closed[*of h*],
frule-tac $h = x$ **in** *sg-subset-elem*[*of H*], *assumption+*,
simp add:*tassoc*[*THEN sym*, *of h* ϱ *h*] *r-i l-unit*)
apply (*simp add*:*sg-i-m-closed*[*of H h*])
done

lemma (**in** *Group*) *lcsTr7*:$[\![G \gg H;\ a \in carrier\ G;\ b \in carrier\ G;\ (\varrho\ a) \cdot b \in H]\!]$
$\Longrightarrow a \diamondsuit H \subseteq\ b \diamondsuit H$
apply (*rule subsetI*)
apply (*simp add*:*lcsTr6* [*of H a b -*])
done

lemma (**in** *Group*) *lcsTr8*:$[\![G \gg H;\ a \in carrier\ G;\ h \in H]\!] \Longrightarrow a \cdot h \in a \diamondsuit H$
apply (*simp add*:*lcs-def*)
apply *blast*
done

lemma (**in** *Group*) *lcs-tool1*:$[\![G \gg H;\ a \in carrier\ G;\ b \in carrier\ G;$
$(\varrho\ a) \cdot b \in H]\!] \Longrightarrow (\varrho\ b) \cdot a \in H$
by (*frule sg-i-closed* [*of H* $(\varrho\ a) \cdot b$], *assumption+*,
frule i-closed[*of a*], *simp add*:*i-ab iop-i-i*)

theorem (**in** *Group*) *lcs-eq*:$[\![G \gg H;\ a \in carrier\ G;\ b \in carrier\ G]\!] \Longrightarrow$
$((\varrho\ a) \cdot b \in H) = (a \diamondsuit H = b \diamondsuit H)$
apply (*rule iffI*)
apply ((*rule equalityI*),
(*rule lcsTr7* [*of H a b*], *assumption+*),

(*frule lcs-tool1* [*of H a b*], *assumption*+),
(*rule lcsTr7* [*of H b a*], *assumption*+))
apply (*subst lcs-mem-ldiv*[*THEN sym*, *of H b a*], *assumption*+,
simp add:*a-in-lcs*[*of H b*])
done

lemma (**in** *Group*) *rcs-subset*:⟦*G* ≫ *H*; *a* ∈ *carrier G*⟧ ⟹ *H* · *a* ⊆ *carrier G*
apply (*rule subsetI*,
simp add:*rcs-def*, *erule bexE*,
frule-tac h = *h* **in** *sg-mult-closedl*[*of H a*], *assumption*+)
apply *simp*
done

lemma (**in** *Group*) *mem-rcs*:⟦*G* ≫ *H*; *x* ∈ *H* · *a*⟧ ⟹ ∃ *h*∈*H*. *h* · *a* = *x*
by (*simp add*:*rcs-def*)

lemma (**in** *Group*) *rcs-subset-elem*:⟦*G* ≫ *H*; *a* ∈ *carrier G*; *x* ∈ *H* · *a*⟧ ⟹
x ∈ *carrier G*
apply (*simp add*:*rcs-def*, *erule bexE*)
apply (*frule-tac h* = *h* **in** *sg-mult-closedl*[*of H a*], *assumption*+,
simp)
done

lemma (**in** *Group*) *rcs-in-set-rcs*:⟦*G* ≫ *H*; *a* ∈ *carrier G*⟧ ⟹
H · *a* ∈ *set-rcs G H*
apply (*simp add*:*set-rcs-def*)
apply *blast*
done

lemma (**in** *Group*) *rcsTr0*:⟦*G* ≫ *H*; *a* ∈ *carrier G*; *b* ∈ *carrier G*⟧ ⟹
H · (*a* · *b*) ∈ *set-rcs G H*
apply (*rule rcs-in-set-rcs* [*of H a* · *b*], *assumption*)
apply (*simp add*:*mult-closed*)
done

lemma (**in** *Group*) *a-in-rcs*:⟦*G* ≫ *H*; *a* ∈ *carrier G*⟧ ⟹ *a* ∈ *H* · *a*
apply (*simp add*: *rcs-def*)
apply (*frule l-unit*[*of a*],
frule sg-unit-closed[*of H*], *blast*)
done

lemma (**in** *Group*) *rcs-nonempty*:⟦*G* ≫ *H*; *X* ∈ *set-rcs G H*⟧ ⟹ *X* ≠ {}
apply (*simp add*:*set-rcs-def*, *erule bexE*)
apply (*frule-tac a* = *a* **in** *a-in-rcs*[*of H*], *assumption*+, *simp*)
apply (*simp add*:*nonempty*)
done

lemma (**in** *Group*) *rcs-tool0*:⟦*G* ≫ *H*; *a* ∈ *carrier G*; *b* ∈ *carrier G*;
a · (*ϱ b*) ∈ *H*⟧ ⟹ *b* · (*ϱ a*) ∈ *H*

by (*frule sg-i-closed* [*of H a* · (ϱ *b*)], *assumption*+,
frule i-closed[*of b*], *simp add*:*i-ab iop-i-i*)

lemma (**in** *Group*) *rcsTr1*:⟦*G* » *H*; *a* ∈ *carrier G*; *b* ∈ *carrier G*;
x ∈ *H* · *a*; *H* · *a* = *H* · *b*⟧ ⟹ *x* ∈ *H* · *b*
by *simp*

lemma (**in** *Group*) *rcs-eqTr*:⟦*G* » *H*; *a* ∈ *carrier G*; *b* ∈ *carrier G*;
H · *a* = *H* · *b*⟧ ⟹ *a* ∈ *H* · *b*
apply (*rule rcsTr1*, *assumption*+)
apply (*rule a-in-rcs*, *assumption*+)
done

lemma (**in** *Group*) *rcs-eqTr1*:⟦*G* » *H*; *a* ∈ *carrier G*; *b* ∈ *carrier G*⟧ ⟹
(*a* ∈ *H* · *b*) = (*a* · (ϱ *b*) ∈ *H*)
apply (*rule iffI*)
apply (*simp add*:*rcs-def*, *erule bexE*,
frule-tac h = *h* **in** *sg-subset-elem*[*of H*], *assumption*+)
apply (*frule-tac x* = *h* **in** *sol-eq-r*[*of b a* -], *assumption*+, *simp*)
apply (*frule inEx*[*of a* · (ϱ *b*) *H*], *erule bexE*)
apply (*frule-tac h* = *y* **in** *sg-subset-elem*[*of H*], *assumption*+,
frule i-closed[*of b*])
apply (*rotate-tac* −*3*, *frule sym*, *thin-tac y* = *a* · ϱ *b*,
frule-tac b = *y* **in** *sol-eq-r*[*of* ϱ *b* - *a*], *assumption*+,
simp add:*iop-i-i*)
apply (*simp add*:*rcs-def*, *blast*)
done

lemma (**in** *Group*) *rcsTr2*:⟦*G* » *H*; *a* ∈ *carrier G*; *b* ∈ *H* · (ϱ *a*)⟧ ⟹
b · *a* ∈ *H*
apply (*frule i-closed*[*of a*],
frule-tac rcs-subset-elem[*of H* ϱ *a b*], *assumption*+,
frule rcs-eqTr1[*of H b* ϱ *a*], *assumption*+)
apply (*simp add*:*iop-i-i*)
done

lemma (**in** *Group*) *rcsTr5*:⟦*G* » *H*; *a* ∈ *carrier G*; *b* ∈ *carrier G*;
b · (ϱ *a*) ∈ *H*; *x* ∈ *H* · *a*⟧ ⟹ *x* · (ϱ *b*) ∈ *H*
apply (*frule rcs-subset-elem*[*of H a x*], *assumption*+,
simp add:*rcs-def*, *erule bexE*)
apply (*frule-tac h* = *h* **in** *sg-subset-elem*[*of H*], *assumption*,
frule-tac a = *h* **and** *b* = *a* **in** *mult-closed*, *assumption*+,
frule i-closed[*of b*])
apply (*frule-tac a* = *h* · *a* **in** *r-mult-eqn*[*of* - *x* ϱ *b*], *assumption*+,
thin-tac h · *a* = *x*,
simp add:*tassoc*[*of* - *a* ϱ *b*],
frule-tac x = *h* **in** *sg-mult-closed*[*of H* - *a* · (ϱ *b*)], *assumption*+,
rule rcs-tool0[*of H b a*], *assumption*+)
apply *simp*

done

lemma (**in** *Group*) *rcsTr6*:⟦$G \gg H$; $a \in$ *carrier* G; $b \in$ *carrier* G;
$b \cdot (\varrho\ a) \in H$; $x \in H \cdot a$⟧ ⟹ $x \in H \cdot b$
apply (*frule rcsTr5* [*of H a b x*], *assumption*+)
apply (*subst rcs-eqTr1*[*of H x b*], *assumption*+)
apply (*rule rcs-subset-elem*[*of H a x*], *assumption*+)
done

lemma (**in** *Group*) *rcs-Unit1*:$G \gg H$ ⟹ $H \cdot \mathbf{1} = H$
apply (*rule equalityI*)
apply (*rule subsetI*,
simp add:*rcs-def*, *erule bexE*,
frule-tac $h = h$ **in** *sg-subset-elem*[*of H*], *assumption*+,
simp add:*r-unit*)
apply (*rule subsetI*)
apply (*simp add*:*rcs-def*,
frule-tac $h = x$ **in** *sg-subset-elem*[*of H*], *assumption*+,
frule-tac $a = x$ **in** *r-unit*, *blast*)
done

lemma (**in** *Group*) *unit-rcs-in-set-rcs*:$G \gg H$ ⟹ $H \in$ *set-rcs* $G\ H$
apply (*cut-tac unit-closed*)
apply (*frule rcs-in-set-rcs*[*of H* **1**], *assumption*+)
apply (*simp add*:*rcs-Unit1*)
done

lemma (**in** *Group*) *rcs-Unit2*:⟦$G \gg H$; $h \in H$⟧ ⟹ $H \cdot h = H$
apply (*rule equalityI*)
apply (*rule subsetI*,
simp add:*rcs-def*, *erule bexE*,
frule-tac $x = ha$ **and** $y = h$ **in** *sg-mult-closed*[*of H*], *assumption*+,
simp)

apply (*rule subsetI*,
simp add:*rcs-def*)
apply (*frule-tac* $h = h$ **in** *sg-subset-elem*[*of H*], *assumption*+,
frule-tac $h = x$ **in** *sg-subset-elem*[*of H*], *assumption*+,
frule i-closed[*of h*],
frule-tac $a = x$ **in** *tassoc*[*of* - ϱ h h], *assumption*+,
simp add:*l-i r-unit*)
apply (*frule-tac* $a = x$ **in** *sg-m-i-closed*[*of H* - *h*], *assumption*+,
blast)
done

lemma (**in** *Group*) *rcsTr7*:⟦$G \gg H$; $a \in$ *carrier* G; $b \in$ *carrier* G; $b \cdot (\varrho\ a) \in H$⟧
⟹ $H \cdot a \subseteq H \cdot b$
apply (*rule subsetI*)
apply (*rule rcsTr6*[*of H a b*], *assumption*+)

done

lemma (**in** *Group*) *rcs-tool1*:⟦$G \gg H$; $a \in$ *carrier* G; $b \in$ *carrier* G;
$b \cdot (\varrho\ a) \in H$⟧ ⟹ $a \cdot (\varrho\ b) \in H$
apply (*frule sg-i-closed*[*of* H $b \cdot (\varrho\ a)$], *assumption*+)
apply (*frule i-closed*[*of* a], *simp add*:*i-ab iop-i-i*)
done

lemma (**in** *Group*) *rcs-tool2*:⟦$G \gg H$; $a \in$ *carrier* G; $x \in H \cdot a$⟧ ⟹
$\exists\ h \in H.\ h \cdot a = x$
apply (*simp add*:*rcs-def*)
done

theorem (**in** *Group*) *rcs-eq*:⟦$G \gg H$; $a \in$ *carrier* G; $b \in$ *carrier* G⟧ ⟹
$(b \cdot (\varrho\ a) \in H) = (H \cdot a = H \cdot b)$
apply (*rule iffI*)
apply (*rule equalityI*)
apply (*frule rcsTr7*[*of* H a b], *assumption*+)
apply (*frule rcs-tool1*[*of* H a b], *assumption*+)
apply (*rule rcsTr7*[*of* H b a], *assumption*+)
apply (*frule a-in-rcs*[*of* H a], *assumption*, *simp*)
apply (*simp add*:*rcs-eqTr1*[*of* H a b])
apply (*rule rcs-tool1*, *assumption*+)
done

lemma (**in** *Group*) *rcs-eq1*:⟦$G \gg H$; $a \in$ *carrier* G; $x \in H \cdot a$⟧ ⟹
$H \cdot a = H \cdot x$
apply (*frule rcs-subset-elem*[*of* H a x], *assumption*+)
apply (*subst rcs-eq*[*THEN sym*, *of* H a x], *assumption*+)
apply (*subst rcs-eqTr1*[*THEN sym*, *of* H x a], *assumption*+)
done

lemma (**in** *Group*) *rcs-eq2*:⟦$G \gg H$; $a \in$ *carrier* G; $b \in$ *carrier* G;
$(H \cdot a) \cap (H \cdot b) \neq \{\}$⟧ ⟹ $(H \cdot a) = (H \cdot b)$
apply (*frule nonempty-int* [*of* $H \cdot a$ $H \cdot b$], *erule exE*)
apply (*simp*, *erule conjE*)
apply (*frule-tac* $x = x$ **in** *rcs-eq1*[*of* H a], *assumption*+,
frule-tac $x = x$ **in** *rcs-eq1*[*of* H b], *assumption*+)
apply *simp*
done

lemma (**in** *Group*) *rcs-meet*:⟦$G \gg H$; $X \in$ *set-rcs* G H; $Y \in$ *set-rcs* G H;
$X \cap Y \neq \{\}$⟧ ⟹ $X = Y$
apply (*simp add*:*set-rcs-def*, (*erule bexE*)+, *simp*)
apply (*rule-tac* $a = a$ **and** $b = aa$ **in** *rcs-eq2*[*of* H], *assumption*+)
done

lemma (**in** *Group*) *rcsTr8*:⟦$G \gg H$; $a \in$ *carrier* G; $h \in H$; $x \in H \cdot a$⟧ ⟹
$h \cdot x \in H \cdot a$

apply (*frule rcs-subset-elem*[*of H a x*], *assumption*+,
frule sg-subset-elem[*of H h*], *assumption*+)
apply (*simp add*:*rcs-def*, *erule bexE*,
frule-tac h = *ha* **in** *sg-subset-elem*[*of H*], *assumption*+,
frule-tac a = *ha* **and** *b* = *a* **in** *mult-closed*, *assumption*,
frule-tac a = *ha* · *a* **and** *b* = *x* **and** *c* = *h* **in** *l-mult-eqn*, *assumption*+,
thin-tac ha · *a* = *x*, *simp add*:*tassoc*[*THEN sym*, *of h - a*])
apply (*frule-tac x* = *h* **and** *y* = *ha* **in** *sg-mult-closed*[*of H*], *assumption*+,
blast)
done

lemma (**in** *Group*) *rcsTr9*:⟦*G* ≫ *H*; *a* ∈ *carrier G*; *h* ∈ *H*; (*ϱ x*) ∈ *H* · *a*⟧ ⟹
h · (*ϱ x*) ∈ *H* · *a*
by (*insert rcsTr8* [*of H a h ϱ x*], *simp*)

lemma (**in** *Group*) *rcsTr10*:⟦*G* ≫ *H*; *a* ∈ *carrier G*; *x* ∈ *H* · *a*; *y* ∈ *H* · *a*⟧ ⟹
x · (*ϱ y*) ∈ *H*
apply (*simp add*:*rcs-def*)
apply (*erule bexE*)+
apply (*rotate-tac* −*1*, *frule sym*, *thin-tac ha* · *a* = *y*,
frule sym, *thin-tac h* · *a* = *x*, *simp*,
thin-tac y = *ha* · *a*, *thin-tac x* = *h* · *a*)
apply (*frule-tac h* = *ha* **in** *sg-subset-elem*[*of H*], *assumption*+,
frule-tac h = *h* **in** *sg-subset-elem*[*of H*], *assumption*+,
simp add:*i-ab*)
apply (*frule-tac a* = *a* **in** *i-closed*, *frule-tac a* = *ha* **in** *i-closed*,
simp add:*tOp-assocTr41*[*THEN sym*], *simp add*:*tOp-assocTr42*,
simp add:*r-i r-unit*)
apply (*simp add*:*sg-m-i-closed*[*of H*])
done

lemma (**in** *Group*) *PrSubg4-2*:⟦*G* ≫ *H*; *a* ∈ *carrier G*; *x* ∈ *H* · (*ϱ a*)⟧ ⟹
x ∈ {*z*. ∃ *v*∈(*H* · *a*). ∃ *h*∈*H*. *h* · (*ϱ v*) = *z*}
apply (*simp add*:*rcs-def*[*of - H ϱ a*], *erule bexE*,
frule-tac h = *h* **in** *sg-subset-elem*[*of H*], *assumption*+,
frule i-closed[*of a*])
apply (*frule a-in-rcs*[*of H a*], *assumption*, *blast*)
done

lemma (**in** *Group*) *rcs-fixed*:⟦*G* ≫ *H*; *a* ∈ *carrier G*; *H* · *a* = *H*⟧ ⟹ *a* ∈ *H*
by (*frule a-in-rcs*[*of H a*], *assumption*+, *simp*)

lemma (**in** *Group*) *rcs-fixed1*:⟦*G* ≫ *H*; *a* ∈ *carrier G*; *h* ∈ *H*⟧ ⟹
H · *a* = (*H* · (*h* · *a*))
apply (*rule rcs-eq1*[*of H a h* · *a*], *assumption*+)
apply (*simp add*:*rcs-def*, *blast*)
done

lemma (**in** *Group*) *rcs-fixed2*:*G* ≫ *H* ⟹ ∀ *h*∈*H*. *H* · *h* = *H*

apply (*rule ballI*)
apply (*simp add*:*rcs-Unit2*)
done

lemma (**in** *Group*) *Gp-rcs*:$[\![G \gg H;\ G \gg K;\ H \subseteq K;\ x \in K]\!] \Longrightarrow$
$H \cdot_{(Gp\ G\ K)} x = (H \cdot x)$
by (*simp add*:*rcs-def*, *simp add*:*Gp-def*)

lemma (**in** *Group*) *subg-lcs*:$[\![G \gg H;\ G \gg K;\ H \subseteq K;\ x \in K]\!] \Longrightarrow$
$x \diamondsuit_{(Gp\ G\ K)} H = x \diamondsuit H$
by (*simp add*:*lcs-def*, *simp add*:*Gp-def*)

3.4 Normal subgroups and Quotient groups

lemma (**in** *Group*) *nsg1*:$[\![G \gg H;\ b \in carrier\ G;\ h \in H;$
$\forall a \in carrier\ G.\ \forall h \in H.\ (a \cdot h) \cdot (\varrho\ a) \in H]\!] \Longrightarrow b \cdot h \cdot (\varrho\ b) \in H$
by *blast*

lemma (**in** *Group*) *nsg2*:$[\![G \gg H;\ b \in carrier\ G;\ h \in H;$
$\forall a \in carrier\ G.\ \forall h \in H.\ (a \cdot h) \cdot (\varrho\ a) \in H]\!] \Longrightarrow\ (\varrho\ b) \cdot h \cdot b \in H$
apply (*frule i-closed*[*of b*],
frule-tac $x = \varrho\ b$ **in** *bspec*, *assumption*,
thin-tac $\forall a \in carrier\ G.\ \forall h \in H.\ a \cdot h \cdot \varrho\ a \in H$,
frule-tac $x = h$ **in** *bspec*, *assumption*,
thin-tac $\forall h \in H.\ \varrho\ b \cdot h \cdot \varrho\ (\varrho\ b) \in H$)
apply (*simp add*:*iop-i-i*)
done

lemma (**in** *Group*) *nsg-subset-elem*:$[\![G \triangleright H;\ h \in H]\!] \Longrightarrow h \in carrier\ G$
by (*insert nsg-sg*[*of H*], *simp add*:*sg-subset-elem*)

lemma (**in** *Group*) *nsg-l-rcs-eq*:$[\![G \triangleright N;\ a \in carrier\ G]\!] \Longrightarrow a \diamondsuit N = N \cdot a$
by (*simp add*: *nsg-def*)

lemma (**in** *Group*) *sg-nsg1*:$[\![G \gg H;\ \forall a \in carrier\ G.\ \forall h \in H.\ (a \cdot h) \cdot (\varrho\ a) \in H;$
$b \in carrier\ G\ ]\!] \Longrightarrow H \cdot b = b \diamondsuit H$
apply (*rule equalityI*)

apply (*rule subsetI*, *simp add*:*rcs-def*, *erule bexE*, *frule i-closed*[*of b*])
apply (*frule-tac* $x = \varrho\ b$ **in** *bspec*, *assumption*,
thin-tac $\forall a \in carrier\ G.\ \forall h \in H.\ a \cdot h \cdot \varrho\ a \in H$,
frule-tac $x = h$ **in** *bspec*, *assumption*,
thin-tac $\forall h \in H.\ \varrho\ b \cdot h \cdot \varrho\ (\varrho\ b) \in H$,
simp add:*iop-i-i*)
apply (*frule-tac* $h = h$ **in** *sg-mult-closedl*[*of H b*], *assumption*+, *simp*,
frule-tac $h = h$ **in** *sg-subset-elem*[*of H*], *assumption*+,
simp only:*tassoc*[*of* $\varrho\ b$ - *b*], *thin-tac* $h \cdot b = x$)
apply (*subst lcs-mem-ldiv*[*of H* - *b*], *assumption*+)

apply (*rule subsetI*, *simp add*:*lcs-def*, *erule bexE*)
apply (*frule-tac* $x = b$ **in** *bspec*, *assumption*,
thin-tac $\forall\, a{\in}carrier\ G.\ \forall\, h{\in}H.\ a \cdot h \cdot \varrho\ a \in H$,
frule-tac $x = h$ **in** *bspec*, *assumption*,
thin-tac $\forall\, h{\in}H.\ \ b \cdot h \cdot (\varrho\ b) \in H$,
simp add:*iop-i-i*)
apply (*frule-tac* $h = h$ **in** *sg-mult-closedr*[*of H b*], *assumption*+, *simp*)
apply (*subst rcs-eqTr1*[*of H - b*], *assumption*+)
done

lemma (**in** *Group*) *cond-nsg*:$[\![G \gg H;\ \forall\, a{\in}carrier\ G.\ \forall\, h{\in}H.\ a \cdot h \cdot (\varrho\ a) \in H\ ]\!]$
$\Longrightarrow G \triangleright H$
apply (*subst nsg-def*, *simp*)
apply (*rule ballI*, *rule sg-nsg1*, *assumption*+)
done

lemma (**in** *Group*) *special-nsg-e*:$G \gg H \Longrightarrow Gp\ G\ H \triangleright \{\mathbf{1}\}$
apply (*simp add*:*nsg-def*,
frule Group-Gp[*of H*])
apply (*rule conjI*)
apply (*frule Group.special-sg-e*[*of* $\natural H$],
simp add:*one-Gp-one*[*THEN sym*])

apply (*rule ballI*,
simp add:*lcs-def rcs-def*, *simp add*:*Gp-def*,
frule-tac $h = x$ **in** *sg-subset-elem*[*of H*], *assumption*+,
simp add:*l-unit r-unit*)
done

lemma (**in** *Group*) *special-nsg-G*:$G \triangleright (carrier\ G)$
apply (*simp add*:*nsg-def*,
simp add:*special-sg-G*)
apply (*rule ballI*, *rule equalityI*)
apply (*rule subsetI*,
simp add:*rcs-def lcs-def*, *erule bexE*)
apply (*frule-tac* $a = h$ **and** $b = x$ **in** *mult-closed*, *assumption*+,
frule-tac $a = x$ **in** *i-closed*,
frule-tac $a1 = x$ **and** $b1 = \varrho\ x$ **and** $c1 = h \cdot x$ **in** *tassoc*[*THEN sym*],
assumption+, *simp add*:*r-i l-unit*,
frule-tac $a = \varrho\ x$ **and** $b = xa$ **in** *mult-closed*, *assumption*+, *blast*)

apply (*rule subsetI*, *simp add*:*rcs-def lcs-def*, *erule bexE*,
frule-tac $a = x$ **and** $b = h$ **in** *mult-closed*, *assumption*+,
frule-tac $a = x$ **in** *i-closed*,
frule-tac $a = x \cdot h$ **and** $b = \varrho\ x$ **and** $c = x$ **in** *tassoc*,
assumption+, *simp add*:*l-i r-unit*,
frule-tac $a = xa$ **and** $b = \varrho\ x$ **in** *mult-closed*, *assumption*+,
blast)

done

lemma (**in** *Group*) *special-nsg-G1*:$G \gg H \Longrightarrow Gp\ G\ H \triangleright H$
apply (*frule Group-Gp*[*of H*], *frule Group.special-nsg-G*[*of* $\natural H$])
apply (*simp add*:*Gp-carrier*)
done

lemma (**in** *Group*) *nsgTr0*:$[\![G \triangleright N;\ a \in carrier\ G;\ b \in carrier\ G;\ b \in N \cdot a\ ]\!]$
$\Longrightarrow\ (a \cdot (\varrho\ b) \in N) \wedge ((\varrho\ a) \cdot b \in N)$
apply (*frule nsg-sg*[*of N*],
frule rcs-eqTr1[*of N b a*], *assumption*+, *simp*,
frule i-closed[*of a*],
frule sg-i-closed[*of N b* $\cdot$ ϱ *a*], *assumption*, *simp add*:*i-ab iop-i-i*)
apply (*simp add*:*nsg-l-rcs-eq*[*THEN sym*, *of N a*],
subst lcs-mem-ldiv[*THEN sym*, *of N b a*], *assumption*+)
done

lemma (**in** *Group*) *nsgTr1*:$[\![G \triangleright N;\ a \in carrier\ G;\ b \in carrier\ G;\ b \cdot (\varrho\ a) \in N]\!]$
$\Longrightarrow (\varrho\ b) \cdot a \in N$
apply (*frule nsg-sg*[*of N*],
simp add:*rcs-eqTr1*[*THEN sym*, *of N b a*],
simp add:*nsg-l-rcs-eq*[*THEN sym*])
apply (*simp add*:*lcs-mem-ldiv*[*of N b a*],
rule lcs-tool1, *assumption*+)
done

lemma (**in** *Group*) *nsgTr2*:$[\![a \in carrier\ G;\ b \in carrier\ G;\ a1 \in carrier\ G;$
b1 $\in$ *carrier G* $]\!] \Longrightarrow (a \cdot b) \cdot (\varrho\ (a1 \cdot b1)) =$
$a \cdot (((b \cdot (\varrho\ b1)) \cdot ((\varrho\ a1) \cdot a)) \cdot (\varrho\ a))$
apply (*frule i-closed*[*of a1*], *frule i-closed*[*of b1*],
frule i-closed[*of a*],
frule mult-closed[*of b* ϱ *b1*], *assumption*+,
frule mult-closed[*of* ϱ *a1 a*], *assumption*+,
subst tassoc[*of b* $\cdot$ $(\varrho\ b1)$ $(\varrho\ a1) \cdot a\ \varrho\ a$], *assumption*+,
simp add:*tassoc*[*of* ϱ *a1 a* ϱ *a*] *r-i r-unit*)
apply (*simp add*:*i-ab*,
subst tassoc[*of a b* $(\varrho\ b1) \cdot (\varrho\ a1)$], *assumption*+,
rule mult-closed, *assumption*+,
simp add:*tassoc*[*THEN sym*, *of b* ϱ *b1* ϱ *a1*])
done

lemma (**in** *Group*) *nsgPr1*:$[\![G \triangleright N;\ a \in carrier\ G;\ h \in N]\!] \Longrightarrow$
$a \cdot (h \cdot (\varrho\ a)) \in N$
apply (*frule nsg-sg*[*of N*],
frule sg-subset-elem[*of N h*], *assumption*+,
frule i-closed[*of a*],
frule rcs-fixed1[*THEN sym*, *of N* ϱ *a h*], *assumption*+,
frule mult-closed[*of h* ϱ *a*], *assumption*+)
apply (*frule a-in-rcs*[*of N h* $\cdot$ $(\varrho\ a)$], *assumption*+, *simp*,

thin-tac $N \cdot (h \cdot \varrho\ a) = N \cdot \varrho\ a$)
apply (*simp add*:*nsg-l-rcs-eq*[*THEN sym*, *of N* ϱ *a*],
simp add:*lcs-mem-ldiv*[*of N h* $\cdot$ (ϱ *a*) ϱ *a*] *iop-i-i*)
done

lemma (**in** *Group*) *nsgPr1-1*:$[\![G \triangleright N;\ a \in carrier\ G\ ;\ h \in N]\!] \Longrightarrow$
$(a \cdot h) \cdot (\varrho\ a) \in N$
apply (*frule nsgPr1*[*of N a h*], *assumption*+)
apply (*frule nsg-sg*[*of N*],
frule sg-subset-elem[*of N h*], *assumption*+,
frule i-closed[*of a*],
simp add:*tassoc*[*THEN sym*, *of a h* ϱ *a*])
done

lemma (**in** *Group*) *nsgPr2*:$[\![G \triangleright N;\ a \in carrier\ G;\ h \in N]\!] \Longrightarrow$
$(\varrho\ a) \cdot (h \cdot a) \in N$
apply (*frule i-closed*[*of a*],
frule nsgPr1[*of N* ϱ *a h*], *assumption*+)
apply (*simp add*:*iop-i-i*)
done

lemma (**in** *Group*) *nsgPr2-1*:$[\![G \triangleright N;\ a \in carrier\ G;\ h \in N]\!] \Longrightarrow$
$(\varrho\ a) \cdot h \cdot a \in N$
apply (*frule i-closed*[*of a*],
frule nsgPr1-1[*of N* ϱ *a h*], *assumption*+, *simp add*:*iop-i-i*)
done

lemma (**in** *Group*) *nsgTr3*:$[\![G \triangleright N;\ a \in carrier\ G;\ b \in carrier\ G;$
$a1 \in carrier\ G;\ b1 \in carrier\ G;\ \ a \cdot (\varrho\ a1) \in N;\ b \cdot (\varrho\ b1) \in N]\!] \Longrightarrow$
$(a \cdot b) \cdot (\varrho\ (a1 \cdot b1)) \in N$
apply (*frule nsg-sg*[*of N*])
apply (*frule nsgTr2* [*of a b a1 b1*], *assumption*+)
apply (*frule nsgTr1* [*of N a1 a*], *assumption*+)

apply (*frule i-closed*[*of a*],
frule sg-i-closed [*of N* (ϱ *a*) $\cdot$ *a1*], *assumption*+,
simp add:*i-ab*[*of* ϱ *a a1*] *iop-i-i*,
frule sg-mult-closed[*of N b* $\cdot$ (ϱ *b1*) (ϱ *a1*) $\cdot$ *a*], *assumption*+)
apply (*rule nsgPr1*[*of N a b* $\cdot$ (ϱ *b1*) $\cdot$ ((ϱ *a1*) $\cdot$ *a*)], *assumption*+)
done

lemma (**in** *Group*) *nsg-in-Gp*:$[\![G \triangleright N;\ G \gg H;\ N \subseteq H]\!] \Longrightarrow (Gp\ G\ H) \triangleright N$
apply (*frule Group-Gp* [*of H*],
frule nsg-sg[*of N*])
apply (*rule Group.cond-nsg* [*of* $\natural H$ *N*], *assumption*+,
simp add:*sg-sg*[*of H N*])
apply (*rule ballI*, *rule ballI*,
(*subst Gp-def*)+, *simp add*:*Gp-carrier*)
apply (*frule-tac h* = *a* **in** *sg-subset-elem*[*of H*], *assumption*+,

rule-tac $a = a$ **and** $h = h$ **in** *nsgPr1-1*[*of* N], *assumption*+)
done

lemma (**in** *Group*) *nsgTr4*:⟦$G \triangleright N$; $a \in$ *carrier* G; $x \in N \cdot a$⟧ ⟹
($\varrho\ x) \in N \cdot (\varrho\ a)$
apply (*frule nsgTr0* [*of* N], *assumption*+)
apply (*frule nsg-sg*[*of* N], *rule a-in-rcs*[*of* N], *assumption*+,
thin-tac $a \cdot \varrho\ a \in N \wedge \varrho\ a \cdot a \in N$,
frule nsg-sg[*of* N],
frule rcs-subset-elem[*of* N a x], *assumption*+,
simp add:*rcs-eqTr1*[*of* N x a])
apply (*frule nsgTr1*[*of* N a x], *assumption*+,
frule i-closed[*of* x],
subst rcs-eqTr1[*of* N ϱ x ϱ a], *assumption*+,
simp add:*i-closed*, *simp add*:*iop-i-i*)
done

lemma (**in** *Group*) *c-topTr1*:⟦$G \triangleright N$; $a \in$ *carrier* G; $b \in$ *carrier* G;
$a1 \in$ *carrier* G; $b1 \in$ *carrier* G; $N \cdot a = N \cdot a1$; $N \cdot b = N \cdot b1$⟧ ⟹
$N \cdot (a \cdot b) = N \cdot (a1 \cdot b1)$
apply (*frule nsg-sg*[*of* N],
frule mult-closed[*of* a b], *assumption*+,
frule mult-closed[*of* $a1$ $b1$], *assumption*+,
simp add:*rcs-eq*[*THEN sym*, *of* N a $a1$],
simp add:*rcs-eq*[*THEN sym*, *of* N b $b1$])
apply (*subst rcs-eq*[*THEN sym*, *of* N $a \cdot b$ $a1 \cdot b1$], *assumption*+)
apply (*rule nsgTr3*[*of* N $a1$ $b1$ a b], *assumption*+)
done

lemma (**in** *Group*) *c-topTr2*:⟦$G \triangleright N$; $a \in$ *carrier* G; $a1 \in$ *carrier* G;
$N \cdot a = N \cdot a1$ ⟧ ⟹ $N \cdot (\varrho\ a) = N \cdot (\varrho\ a1)$
apply (*frule nsg-sg*[*of* N],
simp add:*rcs-eq*[*THEN sym*, *of* N a $a1$])
apply (*subst rcs-eq*[*THEN sym*, *of* N ϱ a ϱ $a1$], *assumption*+,
simp add:*i-closed*, *simp add*:*i-closed*, *simp add*:*iop-i-i*)
apply (*rule nsgTr1*[*of* N a $a1$], *assumption*+)
done

lemma (**in** *Group*) *c-iop-welldefTr1*:⟦$G \triangleright N$; $a \in$ *carrier* G⟧ ⟹
c-iop G N $(N \cdot a) \subseteq\ N \cdot (\varrho\ a)$
apply (*frule nsg-sg*[*of* N],
frule i-closed[*of* a])
apply (*rule subsetI*)
apply (*simp add*:*c-iop-def rcs-in-set-rcs*)
apply (*erule bexE*)+
apply (*simp add*:*rcs-def*[*of* - N a], *erule bexE*,
rotate-tac -1, *frule sym*, *thin-tac* $ha \cdot a = xa$, *simp*,
thin-tac $xa = ha \cdot a$)
apply (*frule-tac* $h = ha$ **in** *sg-subset-elem*[*of* N], *assumption*+,

frule-tac $a = ha$ **and** $b = a$ **in** *mult-closed*, *assumption*+,
frule-tac $a = ha \cdot a$ **in** *i-closed*,
frule-tac $h = h$ **in** *sg-subset-elem*[*of N*], *assumption*+,
frule-tac $a = h$ **and** $b = \varrho\ (ha \cdot a)$ **in** *mult-closed*, *assumption*+)
apply (*frule-tac* $a = h \cdot \varrho\ (ha \cdot a)$ **and** $b = x$ **in** *r-mult-eqn*[*of* - -
$a \cdot (\varrho\ a)$], *simp*, *simp add*:*r-i unit-closed*, *assumption*)
apply (*frule-tac* $a = h \cdot \varrho\ (ha \cdot a)$ **in** *tassoc*[*of* - $a\ \varrho\ a$], *assumption*+,
frule-tac $a = h \cdot \varrho\ (ha \cdot a)$ **and** $A = carrier\ G$ **and** $b = x$ **in**
eq-elem-in, *assumption*+,
thin-tac $h \cdot \varrho\ (ha \cdot a) = x$, *simp*,
thin-tac $h \cdot \varrho\ (ha \cdot a) \cdot (a \cdot \varrho\ a) = x \cdot (a \cdot \varrho\ a)$)
apply (*frule-tac* $a = h$ **and** $b = \varrho\ (ha \cdot a)$ **and** $c = a$ **in** *tassoc*, *assumption*+,
simp, *thin-tac* $h \cdot \varrho\ (ha \cdot a) \cdot a = h \cdot (\varrho\ (ha \cdot a) \cdot a)$)
apply (*simp add*:*i-ab r-i r-unit*)
apply (*frule-tac* $x = ha$ **in** *sg-i-closed*[*of N*], *assumption*+,
frule sym, *thin-tac* $h \cdot (\varrho\ a \cdot \varrho\ ha \cdot a) \cdot \varrho\ a = x$, *simp*,
thin-tac $x = h \cdot (\varrho\ a \cdot \varrho\ ha \cdot a) \cdot \varrho\ a$)
apply (*frule-tac* $h = \varrho\ ha$ **in** *nsgPr2-1*[*of N a*], *assumption*+,
frule-tac $x = h$ **and** $y = \varrho\ a \cdot \varrho\ ha \cdot a$ **in** *sg-mult-closed*, *assumption*+)
apply (*simp add*:*rcs-def*, *blast*)
done

lemma (**in** *Group*) *c-iop-welldefTr2*:$[\![G \triangleright N;\ a \in carrier\ G]\!] \Longrightarrow$
$N \cdot (\varrho\ a) \subseteq\ c\text{-}iop\ G\ N\ (N \cdot a)$
apply (*rule subsetI*)
apply (*frule nsg-sg*[*of N*],
frule i-closed[*of a*],
frule-tac $x = x$ **in** *rcs-subset-elem*[*of N* ϱ *a*], *assumption*+)
apply (*simp add*:*c-iop-def*,
simp add:*rcs-in-set-rcs*[*of N a*],
simp add:*rcs-def*[*of* - *N* ϱ *a*])
apply (*frule a-in-rcs*[*of N a*], *assumption*, *blast*)
done

lemma (**in** *Group*) *c-iop-welldef*:$[\![G \triangleright N;\ a \in carrier\ G]\!] \Longrightarrow$
$c\text{-}iop\ G\ N\ (N \cdot a) =\ N \cdot (\varrho\ a)$
apply (*rule equalityI*)
apply (*simp only*:*c-iop-welldefTr1*[*of N a*])
apply (*simp only*:*c-iop-welldefTr2*[*of N a*])
done

lemma (**in** *Group*) *c-top-welldefTr1*:$[\![G \triangleright N;\ a \in carrier\ G;$
$b \in carrier\ G;\ x \in N \cdot a;\ y \in N \cdot b]\!] \Longrightarrow x \cdot y \in N \cdot (a \cdot b)$
apply (*frule nsg-sg*[*of N*])
apply (*frule-tac* $x = x$ **in** *rcs-subset-elem*[*of N a*], *assumption*+,
frule-tac $x = y$ **in** *rcs-subset-elem*[*of N b*], *assumption*+,
frule rcs-eqTr1[*of N x a*], *assumption*+,
frule rcs-eqTr1[*of N y b*], *assumption*+, *simp*)
apply (*frule-tac mult-closed*[*of a b*], *assumption*+,

frule-tac mult-closed[*of x y*], *assumption*+)
apply (*subst rcs-eqTr1*[*of N x* · *y a* · *b*], *assumption*+,
rule nsgTr3[*of N x y a b*], *assumption*+)
done

lemma (**in** *Group*) *c-top-welldefTr2*:⟦*G* ▷ *N*; *a* ∈ *carrier G*; *b* ∈ *carrier G* ⟧
⟹ *c-top G N* (*N* · *a*) (*N* · *b*) ⊆ *N* · (*a* · *b*)
apply (*frule nsg-sg*[*of N*],
simp add:*c-top-def*, *simp add*:*rcs-in-set-rcs*)
apply (*rule subsetI*, *simp*, (*erule bexE*)+,
frule-tac x = *xa* **and** *y* = *y* **in** *c-top-welldefTr1*[*of N a b*],
assumption+, *simp*)
done

lemma (**in** *Group*) *c-top-welldefTr4*:⟦*G* ▷ *N*; *a* ∈ *carrier G*; *b* ∈ *carrier G*;
x ∈ *N* · (*a* · *b*)⟧ ⟹ *x* ∈ *c-top G N* (*N* · *a*) (*N* · *b*)
apply (*frule nsg-sg*[*of N*])
apply (*simp add*:*c-top-def*, *simp add*:*rcs-in-set-rcs*,
simp add:*rcs-def*[*of - N a* · *b*], *erule bexE*,
frule-tac h = *h* **in** *sg-subset-elem*[*of N*], *assumption*+,
simp add:*tassoc*[*THEN sym*, *of - a b*])
apply (*frule a-in-rcs*[*of N b*], *assumption*,
frule-tac h1 = *h* **in** *rcs-fixed1*[*THEN sym*, *of N a*], *assumption*+,
frule-tac a = *h* **and** *b* = *a* **in** *mult-closed*, *assumption*+,
frule-tac a = *h* · *a* **in** *a-in-rcs*[*of N*], *assumption*+, *simp*)
apply *blast*
done

lemma (**in** *Group*) *c-top-welldefTr5*:⟦*G* ▷ *N*; *a* ∈ *carrier G*; *b* ∈ *carrier G*⟧ ⟹
N · (*a* · *b*) ⊆ *c-top G N* (*N* · *a*) (*N* · *b*)
by (*rule subsetI*,
rule c-top-welldefTr4[*of N a b -*], *assumption*+)

lemma (**in** *Group*) *c-top-welldef*:⟦*G* ▷ *N*; *a* ∈ *carrier G*; *b* ∈ *carrier G*⟧ ⟹
N · (*a* · *b*) = *c-top G N* (*N* · *a*) (*N* · *b*)
by (*rule equalityI*, *simp only*:*c-top-welldefTr5*, *simp only*:*c-top-welldefTr2*)

lemma (**in** *Group*) *Qg-unitTr*:⟦*G* ▷ *N*; *a* ∈ *carrier G*⟧ ⟹
c-top G N N (*N* · *a*) = *N* · *a*
apply (*frule nsg-sg*[*of N*])
apply (*rule equalityI*)
apply (*rule subsetI*, *simp add*:*c-top-def*)
apply (*simp add*:*unit-rcs-in-set-rcs rcs-in-set-rcs*)
apply (*erule bexE*)+
apply (*simp add*:*rcs-def*, *erule bexE*)
apply (*frule sym*, *thin-tac xa* · *y* = *x*, *frule sym*, *thin-tac h* · *a* = *y*,
simp, *thin-tac x* = *xa* · (*h* · *a*), *thin-tac y* = *h* · *a*,
frule-tac h = *xa* **in** *sg-subset-elem*[*of N*], *assumption*+,
frule-tac h = *h* **in** *sg-subset-elem*[*of N*], *assumption*+,

simp add:*tassoc*[*THEN sym*],
frule-tac $x = xa$ **and** $y = h$ **in** *sg-mult-closed*[*of N*], *assumption*+,
blast)
apply (*rule subsetI*,
simp add:*c-top-def*, *simp add*:*unit-rcs-in-set-rcs rcs-in-set-rcs*)
apply (*frule-tac* $x = x$ **in** *mem-rcs* [*of N - a*], *assumption*, *erule bexE*,
frule a-in-rcs[*of N a*], *assumption*, *blast*)
done

lemma (**in** *Group*) *Qg-unit*:$G \triangleright N \Longrightarrow \forall x{\in}$*set-rcs G N*. *c-top G N N x* $= x$
by (*rule ballI*,
simp add:*set-rcs-def*, *erule bexE*, *simp*,
simp add:*Qg-unitTr*)

lemma (**in** *Group*) *Qg-iTr*:$[\![G \triangleright N;\ a \in$ *carrier G*$]\!] \Longrightarrow$
c-top G N (*c-iop G N* (*N* $\cdot$ *a*)) (*N* $\cdot$ *a*) $= N$
apply (*simp add*:*c-iop-welldef* [*of N a*])
apply (*frule i-closed*[*of a*],
simp add:*c-top-welldef*[*THEN sym*, *of N* ϱ *a a*],
simp add:*l-i*)
apply (*frule nsg-sg*[*of N*],
simp add:*rcs-Unit1*[*of N*])
done

lemma (**in** *Group*) *Qg-i*:$G \triangleright N \Longrightarrow$
$\forall x \in$ *set-rcs G N*. *c-top G N* (*c-iop G N x*) $x = N$
apply (*rule ballI*, *simp add*:*set-rcs-def*, *erule bexE*)
apply (*simp add*:*Qg-iTr*)
done

lemma (**in** *Group*) *Qg-tassocTr*:
$[\![G \triangleright N;\ a \in$ *carrier G*; $b \in$ *carrier G*; $c \in$ *carrier G* $]\!] \Longrightarrow$
c-top G N (*N* $\cdot$ *a*) (*c-top G N* (*N* $\cdot$ *b*) (*N* $\cdot$ *c*)) =
c-top G N (*c-top G N* (*N* $\cdot$ *a*) (*N* $\cdot$ *b*)) (*N* $\cdot$ *c*)
apply (*frule mult-closed*[*of b c*], *assumption*+,
frule mult-closed[*of a b*], *assumption*+,
simp add:*c-top-welldef*[*THEN sym*])
apply (*simp add*:*tassoc*)
done

lemma (**in** *Group*) *Qg-tassoc*: $G \triangleright N \Longrightarrow$
$\forall X{\in}$*set-rcs G N*. $\forall Y{\in}$*set-rcs G N*. $\forall Z{\in}$*set-rcs G N*. *c-top G N X* (*c-top G N Y Z*)
$=$ *c-top G N* (*c-top G N X Y*) *Z*
apply (*rule ballI*)+ **apply** (*simp add*:*set-rcs-def*, (*erule bexE*)+)
apply (*simp add*:*Qg-tassocTr*)
done

lemma (**in** *Group*) *Qg-top*:$G \triangleright N \Longrightarrow$

c-top G N : set-rcs G N → set-rcs G N → set-rcs G N
apply (*rule Pi-I*, *rule Pi-I*, *simp add*:*set-rcs-def*, (*erule bexE*)+,
simp add:*c-top-welldef*[*THEN sym*])
apply (*metis mult-closed*)
done

lemma (**in** *Group*) *Qg-top-closed*:⟦*G ▷ N*; *A ∈ set-rcs G N*; *B ∈ set-rcs G N*⟧ ⟹
c-top G N A B ∈ set-rcs G N
apply (*frule Qg-top*[*of N*])
apply (*frule funcset-mem* [*of c-top G N set-rcs G N - A*], *assumption*)
apply (*rule funcset-mem*[*of c-top G N A set-rcs G N set-rcs G N B*],
assumption, *assumption*)
done

lemma (**in** *Group*) *Qg-iop*: *G ▷ N* ⟹
c-iop G N :*set-rcs G N → set-rcs G N*
apply (*rule Pi-I*)
apply (*simp add*:*set-rcs-def*, *erule bexE*)
apply (*simp add*:*c-iop-welldef*)
apply (*frule-tac a = a* **in** *i-closed*, *blast*)
done

lemma (**in** *Group*) *Qg-iop-closed*:⟦*G ▷ N*; *A ∈ set-rcs G N*⟧ ⟹
c-iop G N A ∈ set-rcs G N
by (*frule Qg-iop*[*of N*],
erule funcset-mem, *assumption*)

lemma (**in** *Group*) *Qg-unit-closed*: *G ▷ N* ⟹ *N ∈ set-rcs G N*
by (*frule nsg-sg*[*of N*],
simp only:*unit-rcs-in-set-rcs*)

theorem (**in** *Group*) *Group-Qg*:*G ▷ N* ⟹ *Group* (*Qg G N*)
apply (*frule Qg-top* [*of N*], *frule Qg-iop* [*of N*],
frule Qg-unit[*of N*], *frule Qg-i*[*of N*],
frule Qg-tassoc[*of N*], *frule Qg-unit-closed*[*of N*])
apply (*simp add*:*Qg-def Group-def*)
done

lemma (**in** *Group*) *Qg-one*:*G ▷ N* ⟹ *one* (*G / N*) = *N*
by (*simp add*:*Qg-def*)

lemma (**in** *Group*) *Qg-carrier*:*carrier* (*G / N*::*'a set*)) = *set-rcs G N*
by (*simp add*:*Qg-def*)

lemma (**in** *Group*) *Qg-unit-group*:*G ▷ N* ⟹
(*set-rcs G N* = {*N*}) = (*carrier G* = *N*)
apply (*rule iffI*)
apply (*rule contrapos-pp*, *simp*+,
frule nsg-sg[*of N*],

frule sg-subset[*of N*],
frule sets-not-eq[*of carrier G N*], *assumption*, *erule bexE*,
frule-tac a = *a* **in** *rcs-in-set-rcs*[*of N*], *assumption*+,
simp)
apply (*frule-tac a* = *a* **in** *a-in-rcs*[*of N*], *assumption*+,
simp)
apply (*simp add*:*set-rcs-def*, *frule nsg-sg*[*of N*],
frule rcs-fixed2[*of N*], *frule-tac sg-unit-closed*[*of N*], *blast*)
done

lemma (**in** *Group*) *Gp-Qg*:$G \triangleright N \Longrightarrow Gp(G\ /\ N)\ (carrier(G\ /\ N)) = G\ /\ N$
by (*simp add*:*Gp-def Qg-def*)

lemma (**in** *Group*) *Pj-hom0*:$[\![G \triangleright N;\ x \in carrier\ G;\ y \in carrier\ G]\!]$
$\Longrightarrow Pj\ G\ N\ (x \cdot y) = (Pj\ G\ N\ x) \cdot_{(G\ /\ N)} (Pj\ G\ N\ y)$
apply (*simp add*:*Pj-def mult-closed*)
apply (*simp add*:*Qg-def*, *simp add*:*c-top-welldef*)
done

lemma (**in** *Group*) *Pj-ghom*:$G \triangleright N \Longrightarrow (Pj\ G\ N) \in gHom\ G\ (G\ /\ N)$
apply (*simp add*:*gHom-def*)
apply (*rule conjI*,
simp add:*restrict-def Pj-def extensional-def*)
apply (*rule conjI*, *simp add*:*Pi-def*,
rule allI, *rule impI*,
simp add:*Qg-def set-rcs-def*, *simp add*:*Pj-def*, *blast*)
apply ((*rule ballI*)+, *simp add*:*Pj-hom0*)
done

lemma (**in** *Group*) *Pj-mem*:$[\![G \triangleright N;\ x \in carrier\ G]\!] \Longrightarrow (Pj\ G\ N)\ x = N \cdot x$
by (*simp add*:*Pj-def*)

lemma (**in** *Group*) *Pj-gsurjec*:$G \triangleright N \Longrightarrow gsurjec\ G\ (G/N)\ (Pj\ G\ N)$
apply (*simp add*:*gsurjec-def*)
apply (*simp add*:*Pj-ghom*)
apply (*rule surj-to-test*[*of Pj G N carrier G carrier* $(G\ /\ N)$],
frule Pj-ghom[*of N*], *simp add*:*gHom-def*,
rule ballI,
simp add:*Qg-def set-rcs-def*, *erule bexE*)
apply (*frule-tac x* = *a* **in** *Pj-mem*[*of N*], *assumption*, *simp*, *blast*)
done

lemma (**in** *Group*) *lcs-in-Gp*:$[\![G \gg H;\ G \gg K;\ K \subseteq H;\ a \in H]\!] \Longrightarrow$
$a \diamondsuit K = a \diamondsuit_{(Gp\ G\ H)} K$
by (*simp add*:*lcs-def*, *simp add*:*Gp-def*)

lemma (**in** *Group*) *rcs-in-Gp*:$[\![G \gg H;\ G \gg K;\ K \subseteq H;\ a \in H\ ]\!] \Longrightarrow$
$K \cdot a = K \cdot_{(Gp\ G\ H)} a$
apply (*simp add*:*rcs-def*)

apply (*simp add*:*Gp-def*)
done

end

theory *Algebra3* **imports** *Algebra2* **begin**

3.5 Setproducts

definition
commutators:: - ⇒ 'a *set* **where**
commutators G = {*z*. ∃ *a* ∈ *carrier G*. ∃ *b* ∈ *carrier G*.
((*a* ·$_G$ *b*) ·$_G$ (ϱ$_G$ *a*)) ·$_G$ (ϱ$_G$ *b*) = *z*}

lemma (**in** *Group*) *contain-commutator*:⟦*G* » *H*; (*commutators G*) ⊆ *H*⟧ ⟹ *G* ▷ *H*
apply (*rule cond-nsg*[*of H*], *assumption*)
apply (*rule ballI*)+
apply (*frule-tac h* = *h* **in** *sg-subset-elem*[*of H*], *assumption*,
frule-tac a = *h* **in** *i-closed*,
frule-tac a = *a* **in** *i-closed*,
frule-tac a = *a* **and** *b* = *h* **in** *mult-closed*, *assumption*+,
frule-tac a = *a* · *h* **and** *b* = ϱ *a* **in** *mult-closed*, *assumption*+)
apply (*frule-tac a* = *a* · *h* · (ϱ *a*) **and** *b* = ϱ *h* **and** *c* = *h* **in** *tassoc*,
assumption+)
apply (*simp add*:*l-i r-unit*)
apply (*rule-tac a* = *a* · *h* · ϱ *a* · ϱ *h* · *h* **and** *A* = *H* **and** *b* = *a* · *h* · ϱ *a*
in *eq-elem-in*,
rule-tac x = *a* · *h* · ϱ *a* · ϱ *h* **and** *y* = *h* **in** *sg-mult-closed*[*of H*],
assumption,
rule-tac c = *a* · *h* · ϱ *a* · ϱ *h* **in** *subsetD*[*of commutators G H*],
assumption,
thin-tac commutators G ⊆ *H*,
simp add:*commutators-def*, *blast*)
apply *assumption*+
done

definition
s-top :: [- , 'a *set*, 'a *set*] ⇒ 'a *set* **where**
s-top G H K = {*z*. ∃ *x* ∈ *H*. ∃ *y* ∈ *K*. (*x* ·$_G$ *y* = *z*)}

abbreviation
S-TOP :: [('a, 'm) *Group-scheme*, 'a *set*, 'a *set*] ⇒ 'a *set*
((3- ⋄$_1$ -) [66,67]66) **where**
H ⋄$_G$ *K* == *s-top G H K*

lemma (**in** *Group*) *s-top-induced*:⟦*G* » *L*; *H* ⊆ *L*; *K* ⊆ *L*⟧ ⟹
H ⋄$_{Gp}$ *G L K* = *H* ⋄$_G$ *K*

by (*simp add*:*s-top-def Gp-def*)

lemma (**in** *Group*) *s-top-l-unit*:$G \gg K \Longrightarrow \{\mathbf{1}\} \diamond_G K = K$
apply (*rule equalityI*)
apply (*rule subsetI*, *simp add*:*s-top-def*, *erule bexE*,
frule-tac $h = y$ **in** *sg-subset-elem*[*of K*], *assumption*+,
simp add:*l-unit*)
apply (*rule subsetI*,
simp add:*s-top-def*)
apply (*frule-tac* $h = x$ **in** *sg-subset-elem*, *assumption*,
frule-tac $a = x$ **in** *l-unit*, *blast*)
done

lemma (**in** *Group*) *s-top-r-unit*:$G \gg K \Longrightarrow K \diamond_G \{\mathbf{1}\} = K$
apply (*rule equalityI*)
apply (*rule subsetI*, *simp add*:*s-top-def*, *erule bexE*,
frule-tac $h = xa$ **in** *sg-subset-elem*[*of K*], *assumption*+,
simp add:*r-unit*)
apply (*rule subsetI*,
simp add:*s-top-def*,
frule-tac $h = x$ **in** *sg-subset-elem*[*of K*], *assumption*+,
frule-tac $a = x$ **in** *r-unit*, *blast*)
done

lemma (**in** *Group*) *s-top-sub*:$[\![G \gg H;\ G \gg K]\!] \Longrightarrow\ H \diamond_G K \subseteq$ *carrier G*
apply (*rule subsetI*) **apply** (*simp add*:*s-top-def*)
apply (*erule bexE*)+
apply (*frule-tac* $h = xa$ **in** *sg-subset-elem* [*of H*], *assumption*+,
frule-tac $h = y$ **in** *sg-subset-elem*[*of K*], *assumption*+,
frule-tac $a = xa$ **and** $b = y$ **in** *mult-closed*, *assumption*+, *simp*)
done

lemma (**in** *Group*) *sg-inc-set-mult*:$[\![G \gg L;\ H \subseteq L;\ K \subseteq L]\!] \Longrightarrow H \diamond_G K \subseteq L$
apply (*rule subsetI*)
apply (*simp add*:*s-top-def*, (*erule bexE*)+)
apply (*frule-tac* $c = xa$ **in** *subsetD* [*of H L*], *assumption*+,
frule-tac $c = y$ **in** *subsetD* [*of K L*], *assumption*+,
frule-tac $x = xa$ **and** $y = y$ **in** *sg-mult-closed*[*of L*], *assumption*+)
apply *simp*
done

lemma (**in** *Group*) *s-top-sub1*:$[\![H \subseteq$ (*carrier G*); $K \subseteq$ (*carrier G*)$]\!] \Longrightarrow$
$H \diamond_G K \subseteq$ *carrier G*
apply (*rule subsetI*)
apply (*simp add*:*s-top-def*)
apply (*erule bexE*)+
apply (*frule-tac* $c = xa$ **in** *subsetD*[*of H carrier G*], *assumption*+,
frule-tac $c = y$ **in** *subsetD*[*of K carrier G*], *assumption*+,
frule-tac $a = xa$ **and** $b = y$ **in** *mult-closed*, *assumption*+, *simp*)

done

lemma (**in** *Group*) *s-top-elem*:$[\![G \gg H;\ G \gg K;\ a \in H;\ b \in K]\!] \Longrightarrow a \cdot b \in H \diamond_G K$
by (*simp add*:*s-top-def*, *blast*)

lemma (**in** *Group*) *s-top-elem1*:$[\![H \subseteq carrier\ G;\ K \subseteq carrier\ G;\ a \in H;\ b \in K]\!] \Longrightarrow$
$a \cdot b \in H \diamond_G K$
by (*simp add*:*s-top-def*, *blast*)

lemma (**in** *Group*) *mem-s-top*:$[\![H \subseteq carrier\ G;\ K \subseteq carrier\ G;\ u \in H \diamond_G K]\!] \Longrightarrow$
$\exists a \in H.\ \exists b \in K.\ (a \cdot b = u)$
by (*simp add*:*s-top-def*)

lemma (**in** *Group*) *s-top-mono*:$[\![H \subseteq carrier\ G;\ K \subseteq carrier\ G;\ H1 \subseteq H;\ K1 \subseteq K]\!]$
$\Longrightarrow\ H1 \diamond_G K1 \subseteq H \diamond_G K$
by (*rule subsetI*, *simp add*:*s-top-def*, *blast*)

lemma (**in** *Group*) *s-top-unit-closed*:$[\![G \gg H;\ G \gg K]\!] \Longrightarrow\ \mathbf{1} \in H \diamond_G K$
apply (*frule sg-unit-closed* [*of H*],
frule sg-unit-closed [*of K*])
apply (*cut-tac unit-closed*,
frule l-unit[*of* **1**])
apply (*simp add*:*s-top-def*, *blast*)
done

lemma (**in** *Group*) *s-top-commute*:$[\![G \gg H;\ G \gg K;\ K \diamond_G H = H \diamond_G K;$
$u \in H \diamond_G K;\ \ v \in\ H \diamond_G K]\!] \Longrightarrow\ u \cdot v \in H \diamond_G K$
apply (*frule sg-subset*[*of H*], *frule sg-subset*[*of K*],
frule mem-s-top[*of H K u*], *assumption*+, (*erule bexE*)+,
frule mem-s-top[*of H K v*], *assumption*+, (*erule bexE*)+)
apply (*rotate-tac 4*, *frule sym*, *thin-tac* $a \cdot b = u$, *frule sym*,
thin-tac $aa \cdot ba = v$, *simp*,
thin-tac $u = a \cdot b$, *thin-tac* $v = aa \cdot ba$)
apply (*frule-tac* $h = a$ **in** *sg-subset-elem*[*of H*], *assumption*+,
frule-tac $h = aa$ **in** *sg-subset-elem*[*of H*], *assumption*+,
frule-tac $h = b$ **in** *sg-subset-elem*[*of K*], *assumption*+,
frule-tac $h = ba$ **in** *sg-subset-elem*[*of K*], *assumption*+)
apply (*simp add*:*tOp-assocTr41*[*THEN sym*], *simp add*:*tOp-assocTr42*)
apply (*frule-tac* $a = b$ **and** $b = aa$ **in** *s-top-elem1*[*of K H*], *assumption*+,
simp, *thin-tac* $K \diamond_G H = H \diamond_G K$)
apply (*frule-tac* $u = b \cdot aa$ **in** *mem-s-top*[*of H K*], *assumption*+,
(*erule bexE*)+, *frule sym*, *thin-tac* $ab \cdot bb = b \cdot aa$, *simp*,
thin-tac $b \cdot aa = ab \cdot bb$)
apply (*frule-tac* $h = ab$ **in** *sg-subset-elem*[*of H*], *assumption*+,
frule-tac $h = bb$ **in** *sg-subset-elem*[*of K*], *assumption*+)
apply (*simp add*:*tOp-assocTr42*[*THEN sym*], *simp add*:*tOp-assocTr41*)

apply (*frule-tac* $x = a$ **and** $y = ab$ **in** *sg-mult-closed*[*of* H], *assumption*+,
frule-tac $x = bb$ **and** $y = ba$ **in** *sg-mult-closed*[*of* K], *assumption*+,
simp add:*s-top-elem1*)
done

lemma (**in** *Group*) *s-top-commute1*:$[\![G \gg H;\ G \gg K;\ K \diamond_G H = H \diamond_G K;$
$u \in H \diamond_G K]\!] \Longrightarrow (\varrho\ u) \in H \diamond_G K$
apply (*frule sg-subset*[*of* H], *frule sg-subset*[*of* K],
frule mem-s-top[*of* H K u], *assumption*+, (*erule bexE*)+)
apply (*frule-tac* $h = a$ **in** *sg-subset-elem*[*of* H], *assumption*+,
frule-tac $h = b$ **in** *sg-subset-elem*[*of* K], *assumption*+,
frule-tac $a = a$ **and** $b = b$ **in** *i-ab*, *assumption*+,
rotate-tac 4, *frule sym*, *thin-tac* $a \cdot b = u$, *simp*,
thin-tac $\varrho\ (a \cdot b) = \varrho\ b \cdot \varrho\ a$)
apply (*frule-tac* $x = a$ **in** *sg-i-closed*[*of* H], *assumption*+,
frule-tac $x = b$ **in** *sg-i-closed*[*of* K], *assumption*+,
frule-tac $a = \varrho\ b$ **and** $b = \varrho\ a$ **in** *s-top-elem1*[*of* K H],
assumption+, *simp*)
done

lemma (**in** *Group*) *s-top-commute-sg*:$[\![G \gg H;\ G \gg K;\ K \diamond_G H = H \diamond_G K]\!] \Longrightarrow$
$G \gg (H \diamond_G K)$
apply (*subst sg-def*)
apply (*frule s-top-unit-closed*[*of* H K], *assumption*,
simp add:*nonempty*, *simp add*:*s-top-sub*)
apply ((*rule ballI*)+,
frule-tac $u = b$ **in** *s-top-commute1*[*of* H K], *assumption*+,
rule-tac $u = a$ **and** $v = \varrho\ b$ **in** *s-top-commute*[*of* H K],
assumption+)
done

lemma (**in** *Group*) *s-top-assoc*:$[\![G \gg H;\ G \gg K;\ G \gg L]\!] \Longrightarrow$
$(H \diamond_G K) \diamond_G L = H \diamond_G (K \diamond_G L)$
apply (*rule equalityI*)
apply (*rule subsetI*, *simp add*:*s-top-def*) **apply** (*erule exE*)
apply (*erule conjE*)
apply (*erule bexE*)+
apply (*rotate-tac* -1, *frule sym*, *thin-tac* $xb \cdot ya = xa$, *simp*,
thin-tac $xa = xb \cdot ya$, *frule sym*, *thin-tac* $xb \cdot ya \cdot y = x$,
simp)
apply (*frule-tac* $h = xb$ **in** *sg-subset-elem*[*of* H], *assumption*+,
frule-tac $h = y$ **in** *sg-subset-elem*[*of* L], *assumption*+,
frule-tac $h = ya$ **in** *sg-subset-elem*[*of* K], *assumption*+,
simp add:*tassoc*, *blast*)
apply (*rule subsetI*, *simp add*:*s-top-def*,
erule bexE, *erule exE*, *erule conjE*, (*erule bexE*)+,
rotate-tac -1, *frule sym*, *thin-tac* $xb \cdot ya = y$, *simp*,
thin-tac $y = xb \cdot ya$)
apply (*frule-tac* $h = xa$ **in** *sg-subset-elem*[*of* H], *assumption*+,

frule-tac $h = ya$ **in** *sg-subset-elem*[*of L*], *assumption*+,
frule-tac $h = xb$ **in** *sg-subset-elem*[*of K*], *assumption*+,
simp add:*tassoc*[*THEN sym*],
frule sym, *thin-tac* $xa \cdot xb \cdot ya = x$, *simp*, *blast*)
done

lemma (**in** *Group*) *s-topTr6*:$[\![G \gg H1;\ G \gg H2;\ G \gg K;\ H1 \subseteq K]\!] \Longrightarrow$
$(H1 \diamond_G H2) \cap K = H1 \diamond_G (H2 \cap K)$
apply (*rule equalityI*)
apply (*rule subsetI*,
simp add:*s-top-def*, *erule conjE*, (*erule bexE*)+,
frule sym, *thin-tac* $xa \cdot y = x$, *simp*,
frule-tac $c = xa$ **in** *subsetD*[*of H1 K*], *assumption*+,
frule-tac $x = xa \cdot y$ **in** *inEx*[*of* - *K*], *erule bexE*,
frule-tac $x = xa$ **in** *sg-i-closed*[*of K*], *assumption*+,
frule-tac $x = \varrho\ xa$ **and** $y = ya$ **in** *sg-mult-closed*[*of K*],
assumption+, *simp*)
apply (*frule-tac* $h = xa$ **in** *sg-subset-elem*[*of K*], *assumption*+,
frule-tac $h = \varrho\ xa$ **in** *sg-subset-elem*[*of K*], *assumption*+,
frule-tac $h = y$ **in** *sg-subset-elem*[*of H2*], *assumption*+,
simp add:*tassoc*[*THEN sym*] *l-i l-unit*)
apply *blast*

apply (*rule subsetI*,
simp add:*s-top-def*, (*erule bexE*)+, *simp*,
frule sym, *thin-tac* $xa \cdot y = x$, *simp*,
frule-tac $c = xa$ **in** *subsetD*[*of H1 K*], *assumption*+,
frule-tac $x = xa$ **and** $y = y$ **in** *sg-mult-closed*[*of K*], *assumption*+,
simp)
apply *blast*
done

lemma (**in** *Group*) *s-topTr6-1*:$[\![G \gg H1;\ G \gg H2;\ G \gg K;\ H2 \subseteq K]\!] \Longrightarrow$
$(H1 \diamond_G H2) \cap K = (H1 \cap K) \diamond_G H2$
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*simp add*:*s-top-def*, *erule conjE*, (*erule bexE*)+)
apply (*frule-tac* $c = y$ **in** *subsetD* [*of H2 K*], *assumption*+)
apply (*frule-tac* $x = y$ **in** *sg-i-closed* [*of K*], *assumption*)
apply (*frule-tac* $h = xa$ **in** *sg-subset-elem*[*of H1*], *assumption*+,
frule-tac $h = x$ **in** *sg-subset-elem*[*of K*], *assumption*+,
frule-tac $h = y$ **in** *sg-subset-elem*[*of K*], *assumption*+,
frule-tac $h = \varrho\ y$ **in** *sg-subset-elem*[*of K*], *assumption*+,
frule sym, *thin-tac* $xa \cdot y = x$,
frule-tac $x = x$ **and** $y = \varrho\ y$ **in** *sg-mult-closed*[*of K*], *assumption*+,
simp add:*tassoc r-i r-unit*, *blast*)

apply (*rule subsetI*, *simp add*:*s-top-def*, (*erule bexE*)+,
simp, *erule conjE*,

frule-tac $c = y$ **in** *subsetD*[*of H2 K*], *assumption*+,
frule-tac $x = xa$ **and** $y = y$ **in** *sg-mult-closed*[*of K*], *assumption*+, *simp*,
blast)
done

lemma (**in** *Group*) *l-sub-smult*:$[\![G \gg H;\ G \gg K]\!] \Longrightarrow H \subseteq H \diamond_G K$
apply (*rule subsetI*,
simp add:*s-top-def*)
apply (*frule sg-unit-closed*[*of K*],
frule-tac $h = x$ **in** *sg-subset-elem*[*of H*], *assumption*+,
frule-tac $a = x$ **in** *r-unit*)
apply *blast*
done

lemma (**in** *Group*) *r-sub-smult*:$[\![G \gg H;\ G \gg K]\!] \Longrightarrow K \subseteq H \diamond_G K$
apply (*rule subsetI*,
simp add:*s-top-def*)
apply (*frule sg-unit-closed*[*of H*],
frule-tac $h = x$ **in** *sg-subset-elem*[*of K*], *assumption*+,
frule-tac $a = x$ **in** *l-unit*)
apply *blast*
done

lemma (**in** *Group*) *s-topTr8*:$G \gg H \Longrightarrow H = H \diamond_G H$
apply (*frule l-sub-smult*[*of H H*], *assumption*)
apply (*rule equalityI*, *assumption*)
apply (*rule subsetI*)
apply (*thin-tac* $H \subseteq H \diamond_G H$,
simp add:*s-top-def*, (*erule bexE*)+)
apply (*frule-tac* $x = xa$ **and** $y = y$ **in** *sg-mult-closed*[*of H*], *assumption*+,
simp)
done

3.6 Preliminary lemmas for Zassenhaus

lemma (**in** *Group*) *Gp-sg-subset*:$[\![G \gg H;\ Gp\ G\ H \gg K]\!] \Longrightarrow K \subseteq H$
by (*frule Group-Gp*[*of H*],
frule Group.sg-subset[*of* $\natural H\ K$], *assumption*,
thin-tac $(\natural H) \gg K$, *thin-tac Group* $(\natural H)$,
simp add:*Gp-def*)

lemma (**in** *Group*) *inter-Gp-nsg*:$[\![G \triangleright N;\ G \gg H\ ]\!] \Longrightarrow (\natural H) \triangleright (H \cap N)$
apply (*frule Group-Gp*[*of H*],
rule Group.cond-nsg[*of Gp G H H* $\cap$ *N*], *assumption*+,
frule nsg-sg[*of N*], *frule inter-sgs*[*of H N*], *assumption*+,
rule sg-sg [*of H H* $\cap$ *N*], *assumption*+)
apply (*rule subsetI*, *simp*)
apply ((*rule ballI*)+, *simp*,
simp add:*Gp-carrier*,

simp add:*Gp-mult-induced*[*of H*],
simp add:*sg-i-induced*[*of H*])
apply (*erule conjE*,
frule-tac x = *a* **in** *sg-i-closed*[*of H*], *assumption*+,
frule-tac x = *a* **and** *y* = *h* **in** *sg-mult-closed*, *assumption*+,
simp add:*Gp-mult-induced*[*of H*],
simp add:*sg-mult-closed*)
apply (*frule-tac h* = *a* **in** *sg-subset-elem*[*of H*], *assumption*+,
simp add:*nsgPr1-1*[*of N*])
done

lemma (**in** *Group*) *ZassenhausTr0*:⟦*G* ⋗ *H*; *G* ⋗ *H1*; *G* ⋗ *K*; *G* ⋗ *K1*;
Gp G H ▷ *H1*; *Gp G K* ▷ *K1*⟧ ⟹ *Gp G* (*H* ∩ *K*) ▷ (*H* ∩ *K1*)
apply (*frule inter-sgs*[*of H K*], *assumption*,
frule inter-sgs[*of H K1*], *assumption*,
frule Group-Gp[*of H*],
frule Group-Gp[*of K*],
frule Group.nsg-sg[*of* ♮*H H1*], *assumption*+,
frule Group.nsg-sg[*of* ♮*K K1*], *assumption*+)
apply (*rule Group.cond-nsg*[*of* ♮(*H* ∩ *K*) *H* ∩ *K1*],
simp add:*Group-Gp*[*of H* ∩ *K*])
apply (*rule sg-sg*[*of H* ∩ *K H* ∩ *K1*], *assumption*+)
apply (*frule Gp-sg-subset*[*of K K1*], *assumption*+,
rule subsetI, *simp add*:*subsetD*)
apply ((*rule ballI*)+, *simp*)

apply (*frule Gp-sg-subset*[*of K K1*], *assumption*+,
erule conjE, *frule-tac c* = *h* **in** *subsetD*[*of K1 K*], *assumption*+)
apply (*rule conjI*)
apply (*simp only*:*Gp-carrier*,
subst Gp-mult-induced1[*of H K*], *assumption*+, *simp*,
simp only:*sg-i-induced1*)
apply (*frule-tac a* = *a* **and** *b* = *h* **in** *Group.mult-closed*[*of* ♮*H*],
simp add:*Gp-carrier*, *simp add*:*Gp-carrier*,
simp only:*Gp-carrier*)
apply (*frule-tac a* = *a* **in** *Group.i-closed*[*of* ♮*H*],
simp add:*Gp-carrier*)
apply (*simp add*:*Gp-mult-induced1*[*of H K*], *simp add*:*Gp-carrier*,
subst Gp-mult-induced1[*of H K*], *assumption*+,
simp add:*Gp-mult-induced sg-mult-closed*,
simp add:*sg-i-induced sg-i-closed*)
apply (*simp add*:*Gp-mult-induced sg-i-induced*, *simp add*:*sg-mult-closed*)

apply (*subst Gp-mult-induced2*[*of H K*], *assumption*+,
simp add:*Gp-carrier*, *simp*, *subst sg-i-induced2*, *assumption*+,
simp add:*Gp-carrier*)
apply (*frule-tac a* = *a* **and** *b* = *h* **in** *Group.mult-closed*[*of* ♮*K*],
simp add:*Gp-carrier*, *simp add*:*Gp-carrier*,
frule-tac a = *a* **in** *Group.i-closed*[*of* ♮*K*], *simp add*:*Gp-carrier*)

apply (*subst Gp-mult-induced2*, *assumption*+,
simp add:*Gp-carrier*, *simp add*:*Gp-mult-induced sg-mult-closed*,
simp add:*Gp-carrier*, *simp add*:*sg-i-induced sg-i-closed*)
apply (*rule-tac* $a = a$ **and** $h = h$ **in** *Group.nsgPr1-1*[*of* $\natural K$ *K1*], *assumption*+,
simp add:*Gp-carrier*, *assumption*)
done

lemma (**in** *Group*) *lcs-sub-s-mult*:$[\![G \gg H;\ G \gg N;\ a \in H]\!] \Longrightarrow a \diamondsuit N \subseteq H \diamond_G N$
apply (*rule subsetI*)
apply (*simp add*:*lcs-def s-top-def*, *blast*)
done

lemma (**in** *Group*) *rcs-sub-smult*:$[\![G \gg H;\ G \gg N;\ a \in H]\!] \Longrightarrow N \cdot a \subseteq N \diamond_G H$
apply (*rule subsetI*)
apply (*simp add*:*rcs-def s-top-def*, *blast*)
done

lemma (**in** *Group*) *smult-commute-sg-nsg*:$[\![G \gg H;\ G \triangleright N]\!] \Longrightarrow H \diamond_G N = N \diamond_G H$
apply (*frule nsg-sg*[*of N*])
apply (*rule equalityI*)
apply (*rule subsetI*,
simp add:*s-top-def*, (*erule bexE*)+,
frule-tac $h = xa$ **in** *sg-subset-elem*, *assumption*+,
frule-tac $h = y$ **in** *sg-subset-elem*, *assumption*,
frule-tac $a = xa$ **and** $b = y$ **in** *mult-closed*, *assumption*,
frule-tac $a = xa$ **in** *i-closed*,
frule-tac $a = xa \cdot y$ **and** $b = \varrho\ xa$ **and** $c = xa$ **in** *tassoc*,
assumption+,
frule sym, *thin-tac* $xa \cdot y = x$, *simp*,
thin-tac $x = xa \cdot y$, *simp add*:*l-i r-unit*,
frule-tac $a = xa$ **and** $h = y$ **in** *nsgPr1-1*[*of N*], *assumption*+)
apply *blast*

apply (*rule subsetI*)
apply (*simp add*:*s-top-def*, (*erule bexE*)+,
frule-tac $h = xa$ **in** *sg-subset-elem*, *assumption*+,
frule-tac $h = y$ **in** *sg-subset-elem*, *assumption*,
frule-tac $a = xa$ **and** $b = y$ **in** *mult-closed*, *assumption*,
frule-tac $a = y$ **in** *i-closed*,
frule-tac $a = y$ **and** $b = \varrho\ y$ **and** $c = xa \cdot y$ **in** *tassoc*,
assumption+,
frule sym, *thin-tac* $xa \cdot y = x$, *simp*,
thin-tac $x = xa \cdot y$, *simp add*:*r-i l-unit*,
frule-tac $a = y$ **and** $h = xa$ **in** *nsgPr2*[*of N*], *assumption*+,
frule sym, *thin-tac* $xa \cdot y = y \cdot (\varrho\ y \cdot (xa \cdot y))$)
apply *blast*
done

lemma (**in** *Group*) *smult-sg-nsg*:$[\![G \gg H;\ G \triangleright N]\!] \Longrightarrow G \gg H \diamond_G N$
apply (*frule smult-commute-sg-nsg*[*of H N*], *assumption*+,
frule nsg-sg[*of N*],
rule s-top-commute-sg[*of H N*], *assumption*+,
rule sym, *assumption*)
done

lemma (**in** *Group*) *smult-nsg-sg*:$[\![G \gg H;\ G \triangleright N]\!] \Longrightarrow G \gg N \diamond_G H$
apply (*frule smult-commute-sg-nsg*[*THEN sym*, *of H N*], *assumption*+)
apply (*simp add*:*smult-sg-nsg*)
done

lemma (**in** *Group*) *Gp-smult-sg-nsg*:$[\![G \gg H;\ G \triangleright N]\!] \Longrightarrow$ *Group* (*Gp G* (*H* $\diamond_G$ *N*))
apply (*frule smult-sg-nsg*[*of H N*], *assumption*+)
apply (*simp add*:*Group-Gp*)
done

lemma (**in** *Group*) *N-sg-HN*:$[\![G \gg H;\ G \triangleright N]\!] \Longrightarrow Gp\ G\ (H \diamond_G N) \gg N$
apply (*frule smult-sg-nsg*[*of H N*], *assumption*+,
frule nsg-sg[*of N*],
frule r-sub-smult[*of H N*], *assumption*+)
apply (*rule sg-sg*[*of H* $\diamond_G$ *N N*], *assumption*+)
done

lemma (**in** *Group*) *K-absorb-HK*:$[\![G \gg H;\ G \gg K;\ H \subseteq K]\!] \Longrightarrow\ H \diamond_G K = K$
apply (*frule r-sub-smult*[*of H K*], *assumption*+)
apply (*rule equalityI*)
apply (*thin-tac* $K \subseteq H \diamond_G K$,
rule subsetI, *simp add*:*s-top-def*, (*erule bexE*)+,
frule-tac c = *xa* **in** *subsetD*[*of H K*], *assumption*+,
frule-tac x = *xa* **and** *y* = *y* **in** *sg-mult-closed*[*of K*], *assumption*+,
simp)
apply *assumption*
done

lemma (**in** *Group*) *nsg-Gp-nsg*:$[\![G \gg H;\ G \triangleright N;\ N \subseteq H]\!] \Longrightarrow Gp\ G\ H \triangleright N$
apply (*frule Group-Gp*[*of H*],
frule nsg-sg[*of N*],
frule sg-sg[*of H N*], *assumption*+,
rule Group.cond-nsg[*of* $\natural H$ *N*], *assumption*+)
apply ((*rule ballI*)+,
frule-tac c = *h* **in** *subsetD*[*of N H*], *assumption*+,
simp add:*Gp-carrier*,
simp add:*Gp-mult-induced*[*of H*] *sg-i-induced*[*of H*]
sg-mult-closed sg-i-closed)
apply (*rule-tac a* = *a* **and** *h* = *h* **in** *nsgPr1-1*[*of N*], *assumption*+,
rule-tac h = *a* **in** *sg-subset-elem*[*of H*], *assumption*+)

done

lemma (**in** *Group*) *Gp-smult-nsg*:$[\![G \gg H;\ G \triangleright N]\!] \Longrightarrow Gp\ G\ (H \diamond_G N) \triangleright N$
apply (*frule smult-sg-nsg*[*of H N*], *assumption*+,
  frule nsg-sg[*of N*],
  frule N-sg-HN[*of H N*], *assumption*+,
  frule Gp-smult-sg-nsg[*of H N*], *assumption*+,
  rule Group.cond-nsg[*of* $\natural(H \diamond_G N)\ N$], *assumption*+)
apply ((*rule ballI*)+,
  frule-tac a = a **in** *Group.i-closed*[*of* $\natural(H \diamond_G N)$], *assumption*+,
  simp add:*Gp-carrier*)

apply (*frule r-sub-smult*[*of H N*], *assumption*+,
  frule-tac c = h **in** *subsetD*[*of N H* $\diamond_G$ *N*], *assumption*+,
  simp add:*Gp-mult-induced*[*of H* $\diamond_G$ *N*] *sg-i-induced*[*of H* $\diamond_G$ *N*])
apply (*frule-tac x = a* **and** *y = h* **in** *sg-mult-closed*[*of H* $\diamond_G$ *N*], *assumption*+,
  simp add:*Gp-mult-induced*)
apply (*rule-tac a = a* **and** *h = h* **in** *nsgPr1-1*[*of N*], *assumption*+,
  frule sg-subset[*of H* $\diamond_G$ *N*], *frule-tac c = a* **in** *subsetD*[*of H* $\diamond_G$ *N*
  carrier G], *assumption*+)
done

lemma (**in** *Group*) *Gp-smult-nsg1*:$[\![G \gg H;\ G \triangleright N]\!] \Longrightarrow Gp\ G\ (N \diamond_G H) \triangleright N$
apply (*simp add*:*smult-commute-sg-nsg*[*THEN sym*, *of H N*],
  simp only:*Gp-smult-nsg*)
done

lemma (**in** *Group*) *ZassenhausTr2-3*:$[\![G \gg H;\ G \gg H1;\ Gp\ G\ H \triangleright H1]\!] \Longrightarrow H1 \subseteq H$
apply (*frule Group-Gp*[*of H*],
  frule Group.nsg-sg[*of* $\natural H\ H1$], *assumption*,
  frule Group.sg-subset[*of* $\natural H\ H1$], *assumption*, *simp add*:*Gp-carrier*)
done

lemma (**in** *Group*) *ZassenhausTr2-4*:$[\![G \gg H;\ G \gg H1;\ Gp\ G\ H \triangleright H1;\ h \in H;\ h1 \in H1]\!] \Longrightarrow h \cdot h1 \cdot (\varrho\ h) \in H1$
apply (*frule Group-Gp*[*of H*])
apply (*frule-tac a = h* **and** *h = h1* **in** *Group.nsgPr1-1*[*of* $\natural H\ H1$], *assumption*)
  apply (*simp add*:*Gp-carrier*) **apply** *assumption*
apply (*simp add*:*Gp-def*)
done

lemma (**in** *Group*) *ZassenhausTr1*:$[\![G \gg H;\ G \gg H1;\ G \gg K;\ G \gg K1;\ Gp\ G\ H \triangleright H1;\ Gp\ G\ K \triangleright K1]\!] \Longrightarrow H1 \diamond_G (H \cap K1) = (H \cap K1) \diamond_G H1$
apply (*frule Group-Gp*[*of H*],
  frule Group.nsg-sg[*of* $\natural H\ H1$], *assumption*,
  frule Group.sg-subset[*of* $\natural H\ H1$], *assumption*, *simp add*:*Gp-carrier*)
apply (*frule Group-Gp*[*of K*],
  frule Group.nsg-sg[*of* $\natural K\ K1$], *assumption*,

frule Group.sg-subset[*of* ♮*K K1*], *assumption*, *simp add*:*Gp-carrier*)
apply (*rule equalityI*)
apply (*rule subsetI*,
simp add:*s-top-def*, (*erule bexE*)+,
frule-tac h = *xa* **in** *sg-subset-elem*[*of H1*], *assumption*+,
frule-tac h = *y* **in** *sg-subset-elem*[*of H*], *simp*,
frule-tac a = *y* **in** *i-closed*,
frule-tac a = *xa* **and** *b* = *y* **in** *mult-closed*, *assumption*+,
frule-tac a1 = *y* **and** *b1* = *ϱ y* **and** *c1* = *xa* · *y* **in** *tassoc*[*THEN sym*],
assumption+)
apply (*frule sym*, *thin-tac xa* · *y* = *x*, *simp*, *thin-tac x* = *xa* · *y*,
simp add:*r-i l-unit*,
frule-tac x = *y* **in** *sg-i-closed*[*of H*], *simp*)
apply (*frule-tac a1* = *ϱ y* **and** *b1* = *xa* **and** *c1* = *y* **in** *tassoc*[*THEN sym*],
assumption+, *simp*, *thin-tac ϱ y* · (*xa* · *y*) = *ϱ y* · *xa* · *y*)
apply (*frule-tac h* = *ϱ y* **and** *?h1.0* = *xa* **in** *ZassenhausTr2-4*[*of H H1*],
assumption+, *simp add*:*iop-i-i*)
apply *blast*

apply (*rule subsetI*)
apply (*simp add*:*s-top-def*, (*erule bexE*)+,
frule-tac h = *xa* **in** *sg-subset-elem*[*of H*], *simp*,
frule-tac h = *y* **in** *sg-subset-elem*[*of H1*], *assumption*,
frule sym, *thin-tac xa* · *y* = *x*, *simp*, *thin-tac x* = *xa* · *y*)
apply (*frule-tac a* = *xa* **in** *i-closed*,
frule-tac a = *xa* **and** *b* = *y* **in** *mult-closed*, *assumption*+,
frule-tac a = *xa* · *y* **and** *b* = *ϱ xa* **and** *c* = *xa* **in** *tassoc*, *assumption*+)
apply (*simp add*:*l-i r-unit*,
frule-tac h = *xa* **and** *?h1.0* = *y* **in** *ZassenhausTr2-4*[*of H H1*],
assumption+, *simp*, *assumption*, *blast*)
done

lemma (**in** *Group*) *ZassenhausTr1-1*:⟦*G* ≫ *H*; *G* ≫ *H1*; *G* ≫ *K*; *G* ≫ *K1*;
Gp G H ▷ *H1*; *Gp G K* ▷ *K1*⟧ ⟹ *G* ≫ (*H1* ⋄$_G$ (*H* ∩ *K1*))
apply (*rule s-top-commute-sg*, *assumption*)
apply (*simp add*:*inter-sgs*[*of H K1*])
apply (*rule ZassenhausTr1* [*THEN sym*, *of H H1 K K1*], *assumption*+)
done

lemma (**in** *Group*) *ZassenhausTr2*:⟦*G* ≫ *H*; *G* ≫ *H1*; *G* ≫ *K*; *Gp G H* ▷ *H1*⟧ ⟹
H1 ⋄$_G$ (*H* ∩ *K*) = (*H* ∩ *K*) ⋄$_G$ *H1*
apply (*frule special-nsg-G1*[*of K*])
apply (*simp add*: *ZassenhausTr1* [*of H H1 K K*])
done

lemma (**in** *Group*) *ZassenhausTr2-1*:⟦*G* ≫ *H*; *G* ≫ *H1*; *G* ≫ *K*; *Gp G H* ▷ *H1*⟧
⟹ *G* ≫ *H1* ⋄$_G$ (*H* ∩ *K*)
apply (*frule ZassenhausTr2* [*of H H1 K*], *assumption*+,
frule inter-sgs [*of H K*], *assumption*+,

rule s-top-commute-sg, *assumption*+)
apply (*rule sym*, *assumption*)
done

lemma (**in** *Group*) *ZassenhausTr2-2*:⟦$G \gg H$; $G \gg H1$; $G \gg K$; $G \gg K1$;
Gp G H ▷ *H1*; *Gp G K* ▷ *K1*⟧ ⟹ $H1 \diamond_G (H \cap K1) \subseteq H1 \diamond_G (H \cap K)$
apply (*frule Group-Gp*[*of K*],
frule Group.nsg-sg[*of* ♮*K K1*], *assumption*,
frule Group.sg-subset[*of* ♮*K K1*], *assumption*, *simp add*:*Gp-carrier*,
frule Group-Gp[*of H*],
frule Group.nsg-sg[*of* ♮*H H1*], *assumption*,
frule Group.sg-subset[*of* ♮*H H1*], *assumption*, *simp add*:*Gp-carrier*,
frule sg-subset[*of H*], *frule sg-subset*[*of K*])
apply (*rule s-top-mono*[*of H1 H* ∩ *K H1 H* ∩ *K1*],
rule subset-trans[*of H1 H carrier G*], *assumption*+,
blast, *simp*, *blast*)
done

lemma (**in** *Group*) *ZassenhausTr2-5*:⟦$G \gg H$; $G \gg H1$; $G \gg K$; $G \gg K1$; *Gp G H* ▷ *H1*;
Gp G K ▷ *K1*; $a \in H1$; $b \in H \cap K1$; $c \in H1$⟧ ⟹
$a \cdot b \cdot c \in H1 \diamond_G (H \cap K1)$
apply (*simp*, *erule conjE*)
apply (*frule sg-subset-elem*[*of H1 a*], *assumption*+,
frule sg-subset-elem[*of H1 c*], *assumption*+,
frule sg-subset-elem[*of H b*], *assumption*+,
frule i-closed[*of b*],
frule mult-closed[*of a b*], *assumption*+,
frule mult-closed[*of* $a \cdot b$ *c*], *assumption*+,
frule tassoc[*of* $a \cdot b \cdot c$ ϱ *b b*], *assumption*+,
simp add:*l-i r-unit*)
apply (*rule eq-elem-in*[*of* $a \cdot b \cdot c \cdot \varrho\ b \cdot b$ *H1* $\diamond_G$ *H* ∩ *K1 a* · *b* · *c*],
thin-tac $a \cdot b \cdot c \cdot \varrho\ b \cdot b = a \cdot b \cdot c$,
frule inter-sgs[*of H K1*], *assumption*+,
rule s-top-elem[*of H1 H* ∩ *K1* $a \cdot b \cdot c \cdot \varrho\ b$ *b*], *assumption*+,
subst tOp-assocTr42, *assumption*+,
frule mult-closed[*of b c*], *assumption*+,
simp add:*tassoc*[*of a b* · *c* ϱ *b*])
apply (*rule sg-mult-closed*[*of H1 a b* · *c* ϱ *b*], *assumption*+,
rule ZassenhausTr2-4[*of H H1 b c*], *assumption*+) **apply** *blast*
apply *assumption*
done

lemma (**in** *Group*) *ZassenhausTr2-6*:⟦$u \in$ *carrier G*; $v \in$ *carrier G*;
$x \in$ *carrier G*; $y \in$ *carrier G*⟧ ⟹
$(u \cdot v) \cdot (x \cdot y) \cdot (\varrho\ (u \cdot v)) =$
$u \cdot v \cdot x \cdot (\varrho\ v) \cdot (v \cdot y \cdot (\varrho\ v)) \cdot (\varrho\ u)$
apply (*simp add*:*i-ab*)
apply (*frule i-closed*[*of u*], *frule i-closed*[*of v*])

apply (*frule mult-closed*[*of u v*], *assumption*+,
frule mult-closed[*of u · v x*], *assumption*+,
frule mult-closed[*of v y*], *assumption*+,
frule mult-closed[*of v · y ϱ v*], *assumption*+,
frule mult-closed[*of u · v · x ϱ v*], *assumption*+,
simp add:*tOp-assocTr42*[*THEN sym*, *of u · v · x · ϱ v*
v · y ϱ v ϱ u])
apply (*frule mult-closed*[*of x y*], *assumption*+,
frule mult-closed[*of u · v x · y*], *assumption*+)
apply (*simp add*:*tassoc*[*THEN sym*, *of u · v · (x · y) ϱ v ϱ u*])
apply (*rule r-mult-eqn*[*of - - ϱ u*],
rule mult-closed[*of u · v · (x · y) ϱ v*], *assumption*+,
(*rule mult-closed*)+, *assumption*+)
apply (*rule r-mult-eqn*[*of - - ϱ v*], *assumption*+,
(*rule mult-closed*)+, *assumption*+)
apply (*simp add*:*tOp-assocTr41*[*THEN sym*, *of u · v · x ϱ v v y*],
simp add:*tOp-assocTr42*[*of u · v · x ϱ v v y*],
simp add:*l-i r-unit*)
apply (*simp add*:*tOp-assocTr41*)
done

lemma (**in** *Group*) *ZassenhausTr2-7*:⟦*a* ∈ *carrier G*; *x* ∈ *carrier G*; *y* ∈ *carrier G*⟧
⟹ *a · (x · y) · (ϱ a) = a · x · (ϱ a) · (a · y · (ϱ a))*
apply (*frule i-closed*[*of a*],
frule mult-closed[*of a y*], *assumption*+,
frule mult-closed[*of a · y ϱ a*], *assumption*+)
apply (*simp add*:*tOp-assocTr41*[*of a x ϱ a a · y · (ϱ a)*],
simp add:*tassoc*[*THEN sym*, *of ϱ a a · y ϱ a*],
simp add:*tassoc*[*THEN sym*, *of ϱ a a y*] *l-i l-unit*,
simp add:*tOp-assocTr41*[*THEN sym*],
simp add:*tOp-assocTr42*[*THEN sym*])
done

lemma (**in** *Group*) *ZassenhausTr3*:⟦*G ⋗ H*; *G ⋗ H1*; *G ⋗ K*; *G ⋗ K1*; *Gp G H ▷ H1*;
Gp G K ▷ K1⟧ ⟹ *Gp G (H1 ⋄$_G$ (H ∩ K)) ▷ (H1 ⋄$_G$ (H ∩ K1))*
apply (*frule ZassenhausTr2-1* [*of H H1 K*], *assumption*+,
frule ZassenhausTr2-1 [*of H H1 K1*], *assumption*+,
frule ZassenhausTr2-2 [*of H H1 K K1*], *assumption*+,
frule sg-sg [*of H1 ⋄$_G$ (H ∩ K) H1 ⋄$_G$ (H ∩ K1)*], *assumption*+,
frule Group-Gp[*of H1 ⋄$_G$ H ∩ K*])
apply (*rule Group.cond-nsg*[*of ♮(H1 ⋄$_G$ H ∩ K) H1 ⋄$_G$ H ∩ K1*], *assumption*+,
(*rule ballI*)+,
simp add:*Gp-carrier*)
apply (*frule-tac c = h* **in** *subsetD*[*of H1 ⋄$_G$ H ∩ K1 H1 ⋄$_G$ H ∩ K*],
assumption+,
simp add:*Gp-mult-induced*[*of H1 ⋄$_G$ H ∩ K*],
simp add:*sg-i-induced*[*of H1 ⋄$_G$ H ∩ K*],

frule-tac $x = a$ **in** *sg-i-closed*[*of H1* $\diamond_G$ $H \cap K$], *assumption*+,
frule-tac $x = a$ **and** $y = h$ **in** *sg-mult-closed*[*of H1* $\diamond_G$ $H \cap K$],
assumption+,
simp add:*Gp-mult-induced*[*of H1* $\diamond_G$ $H \cap K$],
thin-tac $\varrho\ a \in H1 \diamond_G H \cap K$, *thin-tac* $a \cdot h \in H1 \diamond_G H \cap K$)
apply (*simp add*:*s-top-def*[*of G H1 H* $\cap$ *K*], (*erule bexE*)+,
simp add:*s-top-def*[*of G H1 H* $\cap$ *K1*], *fold s-top-def*,
(*erule bexE*)+, *thin-tac* $xa \cdot ya = h$, (*erule conjE*)+,
thin-tac $xa \in H1$, *thin-tac* $ya \in H$,
frule sym, *thin-tac* $x \cdot y = a$,
frule sym, *thin-tac* $xb \cdot yb = h$, *simp*, (*erule conjE*)+,
thin-tac $a = x \cdot y$, *thin-tac* $h = xb \cdot yb$)
apply (*frule-tac* $h = x$ **in** *sg-subset-elem*[*of H1*], *assumption*+,
frule-tac $h = y$ **in** *sg-subset-elem*[*of H*], *assumption*+,
frule-tac $h = xb$ **in** *sg-subset-elem*[*of H1*], *assumption*+,
frule-tac $h = yb$ **in** *sg-subset-elem*[*of H*], *assumption*+,
subst ZassenhausTr2-6, *assumption*+)

apply (*frule-tac* $a = y$ **and** $b = xb$ **in** *mult-closed*, *assumption*+,
frule-tac $a = y$ **in** *i-closed*,
frule-tac $a = y \cdot xb$ **and** $b = \varrho\ y$ **in** *mult-closed*, *assumption*+,
frule-tac $a = x$ **and** $b = y \cdot xb \cdot \varrho\ y$ **in** *mult-closed*, *assumption*+,
frule-tac $a = y$ **and** $b = yb$ **in** *mult-closed*, *assumption*+,
frule-tac $a = y \cdot yb$ **and** $b = \varrho\ y$ **in** *mult-closed*, *assumption*+,
frule-tac $a = x \cdot y \cdot xb \cdot \varrho\ y$ **and** $b = y \cdot yb \cdot \varrho\ y$ **and** $c = \varrho\ x$
in *ZassenhausTr2-5*[*of H H1 K K1*], *assumption*+,
frule-tac $a = x$ **and** $b = y$ **in** *mult-closed*, *assumption*+,
frule-tac $a = x \cdot y$ **and** $b = xb$ **and** $c = \varrho\ y$ **in** *tassoc*, *assumption*+,
simp, *thin-tac* $x \cdot y \cdot xb \cdot \varrho\ y = x \cdot y \cdot (xb \cdot \varrho\ y)$,
frule-tac $a = xb$ **and** $b = \varrho\ y$ **in** *mult-closed*, *assumption*+,
simp add:*tassoc*)
apply (*rule-tac* $x = x$ **and** $y = y \cdot (xb \cdot \varrho\ y)$ **in** *sg-mult-closed*, *assumption*+,
simp add:*tassoc*[*THEN sym*],
rule-tac $h = y$ **and** *?h1.0* $= xb$ **in** *ZassenhausTr2-4*[*of H H1*],
assumption+)
apply (*frule-tac* $x = y$ **and** $y = yb$ **in** *sg-mult-closed*[*of H*], *assumption*+,
frule-tac $x = y$ **in** *sg-i-closed*[*of H*], *assumption*+,
frule-tac $x = y \cdot yb$ **and** $y = \varrho\ y$ **in** *sg-mult-closed*[*of H*],
assumption+, *simp*,
rule-tac $h = y$ **and** *?h1.0* $= yb$ **in** *ZassenhausTr2-4*[*of K K1*],
assumption+,
rule-tac $x = x$ **in** *sg-i-closed*[*of H1*], *assumption*+)
apply (*simp add*:*s-top-def*[*of G H1 H* $\cap$ *K1*])
done

lemma (**in** *Group*) *ZassenhausTr3-2*:⟦$G \gg H$; $G \gg H1$; $G \gg K$; $G \gg K1$; *Gp G H* $\triangleright$ *H1*;
Gp G K $\triangleright$ *K1*⟧ $\Longrightarrow$ $G \gg H1 \diamond_G (H \cap K1) \diamond_G (H \cap K)$
apply (*frule s-top-assoc*[*of H1 H* $\cap$ *K1 H* $\cap$ *K*],

(*simp add*:*inter-sgs*)+,
frule inter-sgs[*of H K1*], *assumption*+,
frule inter-sgs[*of H K*], *assumption*+)
apply (*frule K-absorb-HK*[*of* $H \cap K1$ $H \cap K$], *assumption*+,
frule ZassenhausTr2-3[*of K K1*], *assumption*+, *blast*,
simp, *simp add*:*ZassenhausTr2-1*)
done

lemma (**in** *Group*) *ZassenhausTr3-3*:⟦$G \gg H$; $G \gg H1$; $G \gg K$; $G \gg K1$; *Gp G H* ▷ *H1*;
Gp G K ▷ *K1*⟧ ⟹ $(H1 \cap K) \diamond_G (H \cap K1) = (K1 \cap H) \diamond_G (K \cap H1)$
apply (*rule equalityI*)
apply (*rule subsetI*, *simp add*:*s-top-def*, (*erule bexE*)+)
apply (*frule sym*, *thin-tac* $xa \cdot y = x$, *simp*, (*erule conjE*)+)
apply (*frule-tac h* = *xa* **in** *sg-subset-elem*[*of H1*], *assumption*+,
frule-tac h = *y* **in** *sg-subset-elem*[*of K1*], *assumption*+,
frule-tac a = *xa* **in** *i-closed*,
frule-tac a = *xa* **and** *b* = *y* **and** *c* = *ϱ xa* **and** *d* = *xa* **in** *tOp-assocTr41*,
assumption+,
frule-tac a = *xa* **and** *b* = *y* **in** *mult-closed*, *assumption*,
simp add:*l-i r-unit*)
apply (*frule-tac h* = *xa* **and** *?h1.0* = *y* **in** *ZassenhausTr2-4*[*of K K1*],
assumption+)
apply (*frule ZassenhausTr2-3*[*of H H1*], *assumption*+,
frule-tac c = *xa* **in** *subsetD*[*of H1 H*], *assumption*+)
apply (*frule-tac x* = *xa* **and** *y* = *y* **in** *sg-mult-closed*[*of H*], *assumption*+)
apply (*frule-tac x* = *xa* **in** *sg-i-closed*[*of H*], *assumption*+,
frule-tac x = $xa \cdot y$ **and** *y* = *ϱ xa* **in** *sg-mult-closed*[*of H*],
assumption+)
apply *blast*

apply (*rule subsetI*, *simp add*:*s-top-def*, (*erule bexE*)+)
apply (*frule sym*, *thin-tac* $xa \cdot y = x$, *simp*, (*erule conjE*)+)
apply (*frule-tac h* = *xa* **in** *sg-subset-elem*[*of K1*], *assumption*+,
frule-tac h = *y* **in** *sg-subset-elem*[*of H1*], *assumption*+,
frule-tac a = *xa* **in** *i-closed*,
frule-tac a = *xa* **and** *b* = *y* **and** *c* = *ϱ xa* **and** *d* = *xa* **in** *tOp-assocTr41*,
assumption+,
frule-tac a = *xa* **and** *b* = *y* **in** *mult-closed*, *assumption*,
simp add:*l-i r-unit*)

apply (*frule-tac h* = *xa* **and** *?h1.0* = *y* **in** *ZassenhausTr2-4*[*of H H1*],
assumption+)
apply (*frule ZassenhausTr2-3*[*of K K1*], *assumption*+)
apply (*frule-tac c* = *xa* **in** *subsetD*[*of K1 K*], *assumption*+)
apply (*frule-tac x* = *xa* **and** *y* = *y* **in** *sg-mult-closed*[*of K*], *assumption*+)
apply (*frule-tac x* = *xa* **in** *sg-i-closed*[*of K*], *assumption*+,
frule-tac x = $xa \cdot y$ **and** *y* = *ϱ xa* **in** *sg-mult-closed*[*of K*],
assumption+)

apply *blast*
done

lemma (**in** *Group*) *ZassenhausTr3-4*:⟦$G \gg H$; $G \gg H1$; $G \gg K$; $G \gg K1$; $Gp\ G\ H \triangleright H1$;
$Gp\ G\ K \triangleright K1$; $g \in H \cap K$; $h \in H \cap K1$⟧ ⟹ $g \cdot h \cdot (\varrho\ g) \in H \cap K1$
apply (*simp*, (*erule conjE*)+)
apply (*frule-tac* $x = g$ **and** $y = h$ **in** *sg-mult-closed*, *assumption*+,
frule-tac $x = g$ **in** *sg-i-closed*[*of H*], *assumption*+,
simp add:*sg-mult-closed*[*of H* $g \cdot h\ \varrho\ g$])
apply (*rule ZassenhausTr2-4*[*of K K1 g h*], *assumption*+)
done

lemma (**in** *Group*) *ZassenhausTr3-5*:⟦$G \gg H$; $G \gg H1$; $G \gg K$; $G \gg K1$; $Gp\ G\ H \triangleright H1$;
$Gp\ G\ K \triangleright K1$⟧ ⟹ $(Gp\ G\ (H \cap K)) \triangleright (H1 \cap K) \diamond_G (H \cap K1)$
apply (*frule inter-sgs*[*of H K*], *assumption*,
frule inter-sgs[*of H1 K*], *assumption*,
frule inter-sgs[*of K H*], *assumption*,
frule inter-sgs[*of H K1*], *assumption*+)
apply (*frule ZassenhausTr3*[*of* $H \cap K$ $H1 \cap K$ $K \cap H$ $H \cap K1$],
assumption+,
frule ZassenhausTr0[*of K K1 H H1*], *assumption*+,
simp add:*Int-commute*,
frule ZassenhausTr0[*of H H1 K K1*], *assumption*+,
simp add:*Int-commute*)
apply (*frule ZassenhausTr2-3* [*of K K1*], *assumption*+,
frule ZassenhausTr2-3 [*of H H1*], *assumption*+)
apply (*simp add*:*Int-commute*[*of K H*])
apply (*cut-tac Int-mono*[*of H H K1 K*])
apply (*cut-tac Int-mono*[*of H1 H K K*])
apply (*simp only*:*Int-absorb1*[*of* $H \cap K1$ $H \cap K$],
simp only:*K-absorb-HK*[*of* $H1 \cap K$ $H \cap K$]) **apply** *simp*+
done

lemma (**in** *Group*) *ZassenhausTr4*:⟦$G \gg H$; $G \gg H1$; $G \gg K$; $G \gg K1$; $Gp\ G\ H \triangleright H1$;
$Gp\ G\ K \triangleright K1$⟧ ⟹ $(H1 \diamond_G (H \cap K1)) \diamond_G (H1 \diamond_G (H \cap K)) = H1 \diamond_G (H \cap K)$
apply (*frule ZassenhausTr2* [*of H H1 K*], *assumption*+,
frule ZassenhausTr2 [*of H H1 K1*], *assumption*+,
frule ZassenhausTr1-1 [*of H H1 K K1*], *assumption*+,
frule ZassenhausTr2-1 [*of H H1 K*], *assumption*+,
frule ZassenhausTr2-2 [*of H H1 K K1*], *assumption*+)
apply (*rule K-absorb-HK*[*of* $H1 \diamond_G H \cap K1$ $H1 \diamond_G H \cap K$], *assumption*+)
done

lemma (**in** *Group*) *ZassenhausTr4-0*: ⟦$G \gg H$; $G \gg H1$; $G \gg K$; $G \gg K1$; $Gp\ G\ H \triangleright H1$;

$Gp\ G\ K \triangleright K1\rrbracket \Longrightarrow\ H1 \diamond_G (H \cap K) = (H1 \diamond_G (H \cap K1)) \diamond_G (H \cap K)$
apply (*frule inter-sgs* [*of H K1*], *assumption*+,
frule inter-sgs [*of H K*], *assumption*+)
apply (*subst s-top-assoc* [*of H1 H* $\cap$ *K1 H* $\cap$ *K*], *assumption*+,
subst K-absorb-HK[*of H* $\cap$ *K1 H* $\cap$ *K*], *assumption*+)
apply (*frule ZassenhausTr2-3*[*of K K1*], *assumption*+,
rule Int-mono[*of H H K1 K*])
apply *simp*+
done

lemma (**in** *Group*) *ZassenhausTr4-1*:$\llbracket G \gg H$; $(Gp\ G\ H) \triangleright H1$; $(Gp\ G\ H) \gg (H \cap K)\rrbracket$
$\Longrightarrow (Gp\ G\ (H1 \diamond_G (H \cap K))) \triangleright H1$
apply (*frule Group-Gp* [*of H*],
frule Group.nsg-sg[*of Gp G H H1*], *assumption*+,
frule Group.Gp-smult-nsg1[*of* $\natural H\ H \cap K\ H1$], *assumption*+,
frule subg-sg-sg [*of H H1*], *assumption*+,
frule Group.sg-subset[*of* $\natural H\ H1$], *assumption*,
frule Group.sg-subset[*of* $\natural H\ H \cap K$], *assumption*+,
frule Group.smult-nsg-sg[*of* $\natural H\ H \cap K\ H1$], *assumption*+,
frule Group.s-top-sub[*of* $\natural H\ H1\ H \cap K$], *assumption*+)
apply (*simp only*: *Gp-carrier s-top-induced* [*of H H1 H* $\cap$ *K*])
apply (*frule subg-sg-sg*[*of H H1* $\diamond_G$ *H* $\cap$ *K*], *assumption*+,
simp add:*Gp-inherited*[*of H1* $\diamond_G$ *H* $\cap$ *K H*])
done

3.7 Homomorphism

lemma *gHom*: $\llbracket$*Group F*; *Group G*; $f \in gHom\ F\ G$; $x \in carrier\ F$;
$y \in carrier\ F\rrbracket \Longrightarrow f\ (x \cdot_F y) = (f\ x) \cdot_G (f\ y)$
apply (*simp add*: *gHom-def*)
done

lemma *gHom-mem*:$\llbracket$*Group F*; *Group G*; $f \in gHom\ F\ G$; $x \in carrier\ F\rrbracket \Longrightarrow$
$(f\ x) \in carrier\ G$
apply (*simp add*:*gHom-def*, (*erule conjE*)+)
apply (*simp add*:*Pi-def*)
done

lemma *gHom-func*:$\llbracket$*Group F*; *Group G*; $f \in gHom\ F\ G\rrbracket \Longrightarrow$
$f \in carrier\ F \to carrier\ G$
by (*simp add*:*gHom-def*)

lemma *gHomcomp*:$\llbracket$*Group F*; *Group G*; *Group H*; $f \in gHom\ F\ G$; $g \in gHom\ G\ H\rrbracket$
$\Longrightarrow (g \circ_F f) \in gHom\ F\ H$
apply (*simp add*:*gHom-def*)
apply (*erule conjE*)+

apply (*simp add*:*cmpghom-def composition*)
apply (*rule ballI*)+
apply (*simp add*:*compose-def*)
apply (*frule-tac* $x = x$ **in** *funcset-mem* [*of f carrier F carrier G*],
assumption+)
apply (*frule-tac* $x = y$ **in** *funcset-mem* [*of f carrier F carrier G*],
assumption+)
apply (*simp add*:*Group.mult-closed*[*of F*])
done

lemma *gHom-comp-gsurjec*:⟦*Group F*; *Group G*; *Group H*; $gsurj_{F,G}\ f$;
$gsurj_{G,H}\ g$⟧ ⟹ $gsurj_{F,H}\ (g \circ_F f)$
apply (*simp add*:*gsurjec-def*,
simp add:*gHomcomp*,
(*erule conjE*)+)
apply (*simp add*:*cmpghom-def*,
simp add:*gHom-def*, (*erule conjE*)+,
rule compose-surj, *assumption*+)
done

lemma *gHom-comp-ginjec*:⟦*Group F*; *Group G*; *Group H*; $ginj_{F,G}\ f$; $ginj_{G,H}\ g$⟧
⟹
$ginj_{F,H}\ (g \circ_F f)$
apply (*simp add*:*ginjec-def*,
simp add:*gHomcomp*,
simp add:*gHom-def*,
(*erule conjE*)+)
apply (*simp add*:*cmpghom-def*,
simp add:*comp-inj*)
done

lemma *ghom-unit-unit*:⟦*Group F*; *Group G*; $f \in gHom\ F\ G$ ⟧ ⟹
$f\ (\mathbf{1}_F) = \mathbf{1}_G$
apply (*frule Group.unit-closed*[*of F*])
apply (*frule gHom* [*of F G f* $\mathbf{1}_F$ $\mathbf{1}_F$], *assumption*+)
apply (*simp add*:*Group.l-unit*[*of F*])
apply (*frule gHom-mem*[*of F G f* $\mathbf{1}_F$], *assumption*+)
apply (*frule sym*)
apply (*rule Group.one-unique*[*of G f* $\mathbf{1}_F$ *f* $\mathbf{1}_F$], *assumption*+)
done

lemma *ghom-inv-inv*:⟦*Group F*; *Group G*; $f \in gHom\ F\ G$; $x \in$ *carrier F*⟧ ⟹
$f\ (\varrho_F\ x) = \varrho_G\ (f\ x)$
apply (*frule Group.i-closed*[*of F x*], *assumption*+,
frule gHom [*of F G f* ϱ_F *x x*], *assumption*+)
apply (*simp add*:*Group.l-i*, *simp add*:*ghom-unit-unit*)
apply (*frule sym*,
frule gHom-mem[*of F G f x*], *assumption*+ ,
frule gHom-mem[*of F G f* ϱ_F *x*], *assumption*+,

rule Group.l-i-unique[*THEN sym*, *of G f x f* (ϱ_F *x*)], *assumption*+)
done

lemma *ghomTr3*:⟦*Group F*; *Group G*; *f* ∈ *gHom F G* ; *x* ∈ *carrier F*;
y ∈ *carrier F*; *f* (*x* ·$_F$ (ϱ_F *y*)) = $\mathbf{1}_G$ ⟧ ⟹ *f x* = *f y*
apply (*frule Group.i-closed*[*of F y*], *assumption*+,
simp only:*gHom ghom-inv-inv*)
apply (*rule Group.r-div-eq*[*of G f x f y*], *assumption*,
(*simp add*:*gHom-mem*)+)
done

lemma *iim-nonempty*:⟦*Group F*; *Group G*; *f* ∈ *gHom F G*; *G* ≫ *K*⟧ ⟹
(*iim F G f K*) ≠ {}
apply (*frule Group.sg-unit-closed*[*of G K*], *assumption*+,
frule Group.unit-closed[*of F*])
apply (*frule ghom-unit-unit*[*of F G f*], *assumption*+, *simp add*:*iim-def*,
frule sym, *thin-tac f* $\mathbf{1}_F$ = $\mathbf{1}_G$, *simp*)
apply *blast*
done

lemma *ghomTr4*:⟦*Group F*; *Group G*; *f* ∈ *gHom F G*; *G* ≫ *K*⟧ ⟹
F ≫ (*iim F G f K*)
apply (*rule Group.sg-condition*[*of F iim F G f K*], *assumption*+,
rule subsetI, *simp add*:*iim-def*,
simp add:*iim-nonempty*)
apply ((*rule allI*)+, *rule impI*, *erule conjE*)
apply (*simp add*:*iim-def*) **apply** (*erule conjE*)+
apply (*frule-tac a* = *b* **in** *Group.i-closed*[*of F*], *assumption*+,
frule-tac a = *a* **and** *b* = ϱ_F *b* **in** *Group.mult-closed*[*of F*],
assumption+, *simp*)
apply (*simp add*:*gHom ghom-inv-inv*)
apply (*frule-tac x* = *f b* **in** *Group.sg-i-closed*[*of G K*], *assumption*+)
apply (*simp add*:*gHom-mem Group.sg-mult-closed*)
done

lemma (**in** *Group*) *IdTr0*: *idmap* (*carrier G*) ∈ *gHom G G*
apply (*simp add*:*gHom-def*)
apply (*simp add*:*idmap-def extensional-def*)
apply (*simp add*:*Pi-def mult-closed*)
done

abbreviation
IDMAP ((*I*_) [*999*]*1000*) **where**
I_F == *idmap* (*carrier F*)

abbreviation
INVFUN ((*3Ifn* - - -) [*88*,*88*,*89*]*88*) **where**
Ifn F G f == *invfun* (*carrier F*) (*carrier G*) *f*

lemma *IdTr1*:⟦*Group F*; $x \in$ *carrier F*⟧ $\Longrightarrow$ $(I_F)\ x = x$
apply (*simp add*:*idmap-def*)
done

lemma *IdTr2*:*Group F* $\Longrightarrow$ $gbij_{F,F}\ (I_F)$
apply (*simp add*:*gbijec-def*)
apply (*rule conjI*)

apply (*simp add*:*gsurjec-def*)
apply (*rule conjI*)
apply (*simp add*:*idmap-def gHom-def Group.mult-closed*)
apply (*simp add*:*surj-to-def image-def idmap-def*)

apply (*simp add*:*ginjec-def*)
apply (*simp add*:*idmap-def gHom-def Group.mult-closed*)
done

lemma *Id-l-unit*:⟦*Group G*; $gbij_{G,G}\ f$⟧ $\Longrightarrow$ $I_G \circ_G f = f$
apply (*rule funcset-eq*[*of* - *carrier G*])
apply (*simp add*:*cmpghom-def*)
apply (*simp add*:*gbijec-def gsurjec-def gHom-def*)
apply (*rule ballI*)
apply (*simp add*:*cmpghom-def compose-def*)
apply (*frule-tac* $x = x$ **in** *gHom-mem*[*of G G f*], *assumption*+)
apply (*simp add*:*gbijec-def gsurjec-def*, *assumption*)
apply (*simp add*:*IdTr1*)
done

3.8 Gkernel

lemma *gkernTr1*:⟦*Group F*; *Group G*; $f \in$ *gHom F G*; $x \in gker_{F,G}\ f$⟧ $\Longrightarrow$ $x \in$ *carrier F*
apply (*simp add*: *gkernel-def*)
done

lemma *gkernTr1-1*:⟦*Group F*; *Group G*; $f \in$ *gHom F G*⟧ $\Longrightarrow$ $gker_{F,G}\ f \subseteq$ *carrier F*
by (*rule subsetI*, *simp add*:*gkernTr1*)

lemma *gkernTr2*:⟦*Group F*; *Group G*; $f \in$ *gHom F G*; $x \in gker_{F,G}\ f$; $y \in gker_{F,G}\ f$⟧
$\Longrightarrow$ $(x \cdot_F y) \in gker_{F,G}\ f$
apply (*simp add*:*gkernel-def*)
apply (*simp add*:*gHom*, (*erule conjE*)+)
apply (*simp add*:*Group.mult-closed*,
frule Group.unit-closed[*of G*],
simp add:*Group.l-unit*[*of G*])
done

lemma *gkernTr3*:⟦*Group F*; *Group G*; $f \in gHom\ F\ G$; $x \in gker_{F,G}\ f$⟧ ⟹
($\varrho_F\ x$) $\in gker_{F,G}\ f$
apply (*simp add*:*gkernel-def*)

apply (*simp add*:*ghom-inv-inv* [*of F G f x*])
apply (*simp add*:*Group.i-closed*) **apply** (*simp add*:*Group.i-one*)
done

lemma *gkernTr6*:⟦*Group F*; *Group G*; $f \in gHom\ F\ G$⟧ ⟹ ($\mathbf{1}_F$) $\in gker_{F,G}\ f$
apply (*simp add*:*gkernel-def*)
apply (*simp add*:*Group.unit-closed ghom-unit-unit* [*of F G f*])
done

lemma *gkernTr7*:⟦*Group F*; *Group G*; $f \in gHom\ F\ G$ ⟧ ⟹ $F \gg gker_{F,G}\ f$
apply (*rule Group.sg-condition*[*of F* $gker_{F,G}\ f$], *assumption*+,
rule subsetI, *simp add*:*gkernTr1*,
frule gkernTr6[*of F G f*], *assumption*+, *simp add*:*nonempty*)

apply ((*rule allI*)+, *rule impI*, *erule conjE*,
frule-tac x = *b* **in** *gkernTr3*[*of F G f*], *assumption*+)
apply (*simp add*:*gkernTr2*)
done

lemma *gker-normal*:⟦*Group F*; *Group G*; $f \in gHom\ F\ G$⟧ ⟹ $F \triangleright gker_{F,G}\ f$
apply (*rule Group.cond-nsg*[*of F* $gker_{F,G}\ f$], *assumption*)
apply (*simp add*:*gkernTr7*)
apply (*rule ballI*)+
apply (*simp add*:*gkernel-def*, *erule conjE*)
apply (*frule-tac a* = *a* **in** *Group.i-closed*[*of F*], *assumption*)
apply (*subst gHom* [*of F G f* -], *assumption*+)
apply (*simp add*:*Group.mult-closed*[*of F*])
apply *assumption*
apply (*simp add*:*gHom*)
apply (*simp add*:*Group.mult-closed*[*of F*])+
apply (*frule-tac x* = *a* **in** *gHom-mem*[*of F G f*], *assumption*+,
simp add:*Group.r-unit*[*of G*])
apply (*simp add*:*ghom-inv-inv*, *simp add*:*Group.r-i*)
done

lemma *Group-coim*:⟦*Group F*; *Group G*; $f \in gHom\ F\ G$⟧ ⟹ *Group* (*F* / $gker_{F,G}$ *f*)
by (*frule gker-normal*[*of F G f*], *assumption*+,
simp add:*Group.Group-Qg*[*of F* $gker_{F,G}\ f$])

lemma *gkern1*:⟦*Group F*; *Ugp E*; $f \in gHom\ F\ E$⟧ ⟹ $gker_{F,E}\ f$ = *carrier F*
apply (*frule Group-Ugp*[*of E*])
apply (*frule gkernTr1-1*[*of F E f*], *assumption*+)
apply (*rule equalityI*, *assumption*)
apply (*rule subsetI*,

thin-tac $gker_{F,E}\ f \subseteq carrier\ F$,
simp add:*gkernel-def*,
frule-tac $x = x$ **in** *gHom-mem*[*of F E f*], *assumption*+,
simp add:*Ugp-def*)
done

lemma *gkern2*:⟦*Group F*; *Group G*; $f \in gHom\ F\ G$; $ginj_{F,G}\ f$⟧ ⟹
$gker_{F,G}\ f = \{\mathbf{1}_F\}$
apply (*frule gkernTr6*[*of F G f*], *assumption*+,
frule singleton-sub[*of* $\mathbf{1}_F$ $gker_{F,G}\ f$],
rule equalityI)
apply (*rule subsetI*,
thin-tac $\{\mathbf{1}_F\} \subseteq gker_{F,G}\ f$,
simp add:*gkernel-def*, (*erule conjE*)+)
apply (*frule sym*, *thin-tac* $f\ \mathbf{1}_F = \mathbf{1}_G$, *simp*, *thin-tac* $\mathbf{1}_G = f\ \mathbf{1}_F$,
simp add:*ginjec-def*, *simp add*:*inj-on-def*)
apply *assumption*
done

lemma *gkernTr9*:⟦*Group F*; *Group G*; $f \in gHom\ F\ G$; $a \in carrier\ F$; $b \in carrier\ F$⟧
⟹ $((gker_{F,G}\ f) \cdot_F a = (gker_{F,G}\ f) \cdot_F b) = (f\ a = f\ b)$

apply (*frule gkernTr7*[*of F G f*], *assumption*+)
apply (*subst Group.rcs-eq*[*THEN sym*, *of F* $gker_{F,G}\ f\ a\ b$], *assumption*+)
apply (*thin-tac* $F \gg gker_{F,G}\ f$)
apply (*simp add*:*gkernel-def*)
apply (*frule Group.i-closed*[*of F a*], *assumption*)
apply (*simp add*:*Group.mult-closed*[*of F*])
apply (*simp add*:*gHom*, *simp add*:*ghom-inv-inv*)
apply (*frule gHom-mem*[*of F G f a*], *assumption*+,
frule gHom-mem[*of F G f b*], *assumption*+)
apply (*rule iffI*)
apply (*rule Group.r-div-eq*[*THEN sym*, *of G f b f a*], *assumption*+)
apply (*simp add*:*Group.r-i*[*of G*])
done

lemma *gkernTr11*:⟦*Group F*; *Group G*; $f \in gHom\ F\ G$; $a \in carrier\ F$⟧ ⟹
$(iim\ F\ G\ f\ \{f\ a\}) = (gker_{F,G}\ f) \cdot_F a$
apply (*frule gkernTr7*[*of F G f*], *assumption*+)
apply (*rule equalityI*)
apply (*rule subsetI*,
simp add:*iim-def*,
erule conjE)
apply (*frule-tac a1* $= x$ **in** *gkernTr9*[*THEN sym*, *of F G f - a*],
assumption+, *simp*,
frule-tac a $= x$ **in** *Group.a-in-rcs*[*of F* $gker_{F,G}\ f$], *assumption*+,
simp)

apply (*rule subsetI*)
apply (*simp add*:*gkernel-def rcs-def iim-def*, *erule exE*, (*erule conjE*)+,
rotate-tac −*1*, *frule sym*, *thin-tac* $h \cdot_F a = x$, *simp add*:*gHom*,
thin-tac $x = h \cdot_F a$,
frule gHom-mem[*of F G f a*], *assumption*+,
simp add:*Group.mult-closed Group.l-unit*)
done

lemma *gbij-comp-bij*:⟦*Group F*; *Group G*; *Group H*; $gbij_{F,G}\ f$; $gbij_{G,H}\ g$⟧
$\Longrightarrow gbij_{F,H}\ (g \circ_F f)$
apply (*simp add*:*gbijec-def*)
apply (*simp add*:*gHom-comp-gsurjec gHom-comp-ginjec*)
done

lemma *gbij-automorph*:⟦*Group G*; $gbij_{G,G}\ f$; $gbij_{G,G}\ g$⟧ ⟹
$gbij_{G,G}\ (g \circ_G f)$
apply (*simp add*:*gbij-comp-bij*)
done

lemma *l-unit-gHom*:⟦*Group F*; *Group G*; $f \in gHom\ F\ G$⟧ ⟹ $(I_G) \circ_F f = f$
apply (*simp add*:*cmpghom-def*)
apply (*rule funcset-eq*[*of* - *carrier F*],
simp add:*compose-def*, *simp add*:*gHom-def*)

apply (*rule ballI*,
simp add:*idmap-def compose-def*,
simp add:*gHom-mem*[*of F G f*])
done

lemma *r-unit-gHom*:⟦*Group F*; *Group G*; $f \in gHom\ F\ G$ ⟧ ⟹ $f \circ_F (I_F) = f$
apply (*simp add*:*cmpghom-def*)
apply (*rule funcset-eq*[*of* - *carrier F*],
simp add:*compose-def*, *simp add*:*gHom-def*)
apply (*rule ballI*,
simp add:*idmap-def compose-def*)
done

3.9 Image

lemma *inv-gHom*:⟦*Group F*; *Group G*; $gbij_{F,G}\ f$⟧ ⟹ $(Ifn\ F\ G\ f) \in gHom\ G\ F$
apply (*simp add*:*gHom-def*)
apply (*rule conjI*)

apply (*simp add*:*invfun-def restrict-def extensional-def*)
apply (*rule conjI*)
apply (*rule inv-func*)
apply (*simp add*:*gbijec-def gsurjec-def gHom-def*)
apply (*simp add*:*gbijec-def gsurjec-def ginjec-def*)+

apply (*rule ballI*)+ **apply** (*erule conjE*)+
apply (*frule gHom-func*[*of F G f*], *assumption*+,
frule inv-func[*of f carrier F carrier G*], *assumption*+)
apply (*frule-tac b* = *x* **in** *invfun-mem*[*of f carrier F carrier G*],
assumption+,
frule-tac b = *y* **in** *invfun-mem*[*of f carrier F carrier G*],
assumption+)
apply (*frule-tac x* = (*Ifn F G f*) *x* **and** *y* = (*Ifn F G f*) *y* **in**
gHom[*of F G f*], *assumption*+)
apply (*simp only*:*invfun-r*)
apply (*frule sym*, *thin-tac f* ((*Ifn F G f*) *x* $\cdot_F$ (*Ifn F G f*) *y*) = *x* $\cdot_G$ *y*)
apply (*frule-tac a* = *x* **and** *b* = *y* **in** *Group.mult-closed*[*of G*], *assumption*+)
apply (*frule-tac b* = *x* $\cdot_G$ *y* **in** *invfun-r*[*of f carrier F*
carrier G], *assumption*+)
apply (*frule-tac r* = *f* ((*Ifn F G f*) (*x* $\cdot_G$ *y*)) **and** *s* = *x* $\cdot_G$ *y* **and** *t* = *f* ((*Ifn F G f*) *x* $\cdot_F$ (*Ifn F G f*) *y*) **in** *trans*, *assumption*+)
apply (*thin-tac f* ((*Ifn F G f*) (*x* $\cdot_G$ *y*)) = *x* $\cdot_G$ *y*,
thin-tac x $\cdot_G$ *y* = *f* ((*Ifn F G f*) *x* $\cdot_F$ (*Ifn F G f*) *y*))
apply (*frule-tac b* = *x* $\cdot_G$ *y* **in** *invfun-mem*[*of f carrier F carrier G*],
assumption+,
frule-tac a = (*Ifn F G f*) *x* **and** *b* = (*Ifn F G f*) *y* **in**
Group.mult-closed[*of F*], *assumption*+)
apply (*simp add*:*inj-on-def*)
done

lemma *inv-gbijec-gbijec*:⟦*Group F*; *Group G*; *gbij*$_{F,G}$ *f*⟧ ⟹ *gbij*$_{G,F}$ (*Ifn F G f*)
apply (*frule inv-gHom* [*of F G f*], *assumption*+)
apply (*simp add*:*gbijec-def*)
apply (*rule conjI*)

apply (*simp add*:*gsurjec-def ginjec-def*, (*erule conjE*)+)
apply (*frule gHom-func*[*of F G f*], *simp add*:*invfun-surj*,
assumption+, *simp add*:*invfun-surj*)

apply (*erule conjE*, *simp add*:*gsurjec-def ginjec-def*, *erule conjE*,
frule gHom-func[*of F G f*], *assumption*+,
rule invfun-inj, *assumption*+)
done

lemma *l-inv-gHom*:⟦*Group F*; *Group G*; *gbij*$_{F,G}$ *f*⟧ ⟹ (*Ifn F G f*) $\circ_F$ *f* = (*I*$_F$)
apply (*rule funcset-eq*[*of - carrier F*],
simp add:*cmpghom-def*, *simp add*:*idmap-def*,
rule ballI)
apply (*simp add*:*idmap-def cmpghom-def compose-def*,
simp add:*gbijec-def gsurjec-def ginjec-def*, (*erule conjE*)+,
frule gHom-func[*of F G f*], *assumption*+)
apply (*rule invfun-l*, *assumption*+)
done

lemma *img-mult-closed*:⟦*Group F*; *Group G*; $f \in$ *gHom F G*; $u \in f$ ‘(*carrier F*); $v \in f$ ‘(*carrier F*)⟧ ⟹ $u \cdot_G v \in f$ ‘(*carrier F*)
apply (*simp add*:*image-def*)
apply ((*erule bexE*)+, *simp*)
apply (*subst gHom* [*of F G f*, *THEN sym*], *assumption*+)
apply (*frule-tac* $a = x$ **and** $b = xa$ **in** *Group.mult-closed* [*of F*], *assumption*+)
apply *blast*
done

lemma *img-unit-closed*:⟦*Group F*; *Group G*; $f \in$ *gHom F G*⟧ ⟹ $\mathbf{1}_G \in f$ ‘(*carrier F*)
apply (*cut-tac Group.unit-closed*[*of F*],
frule ghom-unit-unit[*THEN sym*, *of F G f*], *assumption*+,
simp add:*image-def*, *blast*,
assumption)
done

lemma *imgTr7*:⟦*Group F*; *Group G*; $f \in$ *gHom F G*; $u \in f$ ‘(*carrier F*)⟧ ⟹ $\varrho_G\ u \in f$ ‘(*carrier F*)
apply (*simp add*:*image-def*, *erule bexE*, *simp*)
apply (*subst ghom-inv-inv*[*THEN sym*, *of F G f*], *assumption*+)
apply (*frule-tac* $a = x$ **in** *Group.i-closed*[*of F*], *assumption*+)
apply *blast*
done

lemma *imgTr8*:⟦*Group F*; *Group G*; $f \in$ *gHom F G*; $F \gg H$; $u \in f$ ‘ H; $v \in f$‘ H ⟧ ⟹ $u \cdot_G v \in f$ ‘ H
apply (*simp add*:*image-def*, (*erule bexE*)+, *simp*,
subst gHom [*of F G f* -, *THEN sym*], *assumption*+)
apply (*simp only*:*Group.sg-subset-elem*[*of F*],
simp only:*Group.sg-subset-elem*[*of F*])
apply (*frule-tac* $x = x$ **and** $y = xa$ **in** *Group.sg-mult-closed*[*of F H*],
assumption+)
apply *blast*
done

lemma *imgTr9*:⟦*Group F*; *Group G*; $f \in$ *gHom F G*; $F \gg H$; $u \in f$ ‘ H⟧ ⟹ $\varrho_G\ u \in f$ ‘ H
apply (*simp add*:*image-def*, *erule bexE*, *simp*)
apply (*frule-tac* $h = x$ **in** *Group.sg-subset-elem*[*of F H*], *assumption*+,
simp add:*ghom-inv-inv*[*THEN sym*])
apply (*frule-tac* $x = x$ **in** *Group.sg-i-closed* [*of F H*], *assumption*+,
blast)
done

lemma *imgTr10*:⟦*Group F*; *Group G*; $f \in$ *gHom F G*; $F \gg H$⟧ ⟹ $\mathbf{1}_G \in f$ ‘ H
apply (*frule Group.sg-unit-closed*[*of F H*], *assumption*+,
subst ghom-unit-unit[*THEN sym*, *of F G f*], *assumption*+)
apply (*simp add*:*image-def*, *blast*)

done

lemma *imgTr11*:⟦*Group F*; *Group G*; $f \in gHom\ F\ G$; $F \gg H$⟧ $\Longrightarrow$ $G \gg (f\ `\ H)$
apply (*frule gHom-func*[*of F G f*], *assumption*+,
  frule Group.sg-subset[*of F H*], *assumption*+,
  frule image-sub[*of f carrier F carrier G H*], *assumption*+)
apply (*rule Group.sg-condition*[*of G f ` H*], *assumption*+,
  frule imgTr10[*of F G f H*], *assumption*+,
  rule nonempty, *assumption*)
apply ((*rule allI*)+, *rule impI*, *erule conjE*,
  frule-tac u = b **in** *imgTr9*[*of F G f H*], *assumption*+,
  frule-tac u = a **and** *v* = ϱ_G *b* **in** *imgTr8*[*of F G f*], *assumption*+)
done

lemma *sg-gimg*:⟦*Group F*; *Group G*; $f \in gHom\ F\ G$ ⟧ $\Longrightarrow$ $G \gg f`(carrier\ F)$
apply (*frule Group.special-sg-G* [*of F*])
apply (*simp add*:*imgTr11*)
done

lemma *Group-Img*:⟦*Group F*; *Group G*; $f \in gHom\ F\ G$ ⟧ $\Longrightarrow$ *Group* $(Img_{F,G}\ f)$
apply (*frule sg-gimg* [*of F G f*], *assumption*+,
  simp add:*Gimage-def Group.Group-Gp*[*of G*])
done

lemma *Img-carrier*:⟦*Group F*; *Group G*; $f \in gHom\ F\ G$ ⟧ $\Longrightarrow$
  carrier $(Img_{F,G}\ f) = f\ `\ (carrier\ F)$
by (*simp add*:*Gimage-def Gp-def*)

lemma *hom-to-Img*:⟦*Group F*; *Group G*; $f \in gHom\ F\ G$⟧ $\Longrightarrow$ $f \in gHom\ F$ $(Img_{F,G}\ f)$
by (*simp add*:*gHom-def Gimage-def Gp-def*)

lemma *gker-hom-to-img*:⟦*Group F*; *Group G*; $f \in gHom\ F\ G$ ⟧ $\Longrightarrow$
  $gker_{F,(Img_{F,G}\ f)}\ f = gker_{F,G}\ f$
by (*simp add*:*gkernel-def Gimage-def*,
  frule sg-gimg[*of F G f*], *assumption*+,
  simp add:*Group.one-Gp-one*[*of G f ` (carrier F)*])

lemma *Pj-im-subg*:⟦*Group G*; $G \gg H$; $G \triangleright K$; $K \subseteq H$⟧ $\Longrightarrow$
  Pj G K ` H = *carrier* ((*Gp G H*) / *K*)
apply (*simp add*:*Qg-def* [*of Gp G H K*])
apply (*rule equalityI*, *rule subsetI*)
apply (*simp add*:*image-def Pj-def set-rcs-def*)
apply (*erule bexE*,
  frule-tac h = xa **in** *Group.sg-subset-elem*[*of G H*], *assumption*+,
  simp)
apply (*simp add*:*Group.Gp-carrier*)
apply (*frule Group.nsg-sg*[*of G K*], *assumption*+)
apply (*frule-tac x1 = xa* **in** *Group.Gp-rcs*[*THEN sym*, *of G K H*],

assumption+, *blast*)

apply (*rule subsetI*)
apply (*simp add*:*image-def Pj-def*)
apply (*simp add*:*set-rcs-def*)
apply (*simp add*:*Group.Gp-carrier*, *erule bexE*)
apply (*frule Group.nsg-sg*[*of G K*], *assumption*+)
apply (*frule-tac x* = *a* **in** *Group.Gp-rcs*[*of G K H*], *assumption*+,
simp)
apply (*frule-tac h* = *a* **in** *Group.sg-subset-elem*[*of G H*], *assumption*+)
apply *blast*
done

lemma (**in** *Group*) *subg-Qsubg*:⟦*G* ≫ *H*; *G* ▷ *K*; *K* ⊆ *H*⟧ ⟹
(*G* / *K*) ≫ *carrier* ((*Gp G H*) / *K*)
apply (*frule Pj-ghom*[*of K*])
apply (*frule nsg-sg* [*of K*])
apply (*frule Group-Qg*[*of K*])
apply (*cut-tac imgTr11* [*of G G* / *K Pj G K H*])
apply (*cut-tac Pj-im-subg* [*of G H K*])
apply *simp* **apply** (*rule Group-axioms* | *assumption*)+
done

3.10 Induced homomorphisms

lemma *inducedhomTr*:⟦*Group F*; *Group G*; *f* ∈ *gHom F G*;
S ∈ *set-rcs F* (*gker*$_{F,G}$ *f*); *s1* ∈ *S*; *s2* ∈ *S* ⟧ ⟹ *f s1* = *f s2*
apply (*simp add*:*set-rcs-def*, *erule bexE*)
apply (*frule-tac a1* = *a* **in** *gkernTr11*[*THEN sym*, *of F G f*], *assumption*+,
simp, *thin-tac S* = *iim F G f* {*f a*},
thin-tac gker$_{F,G}$ *f* ·$_F$ *a* = *iim F G f* {*f a*})
apply (*simp add*:*iim-def*)
done

definition
induced-ghom :: [('*a*, '*more*) *Group-scheme*, ('*b*, '*more1*) *Group-scheme*,
('*a* ⇒ '*b*)] ⇒ ('*a set* ⇒ '*b*) **where**
induced-ghom F G f = (λ*X*∈ (*set-rcs F* (*gker*$_{F,G}$ *f*)). *f* (*SOME x*. *x* ∈ *X*))

abbreviation
INDUCED-GHOM :: ['*a* ⇒ '*b*, ('*a*, '*m*) *Group-scheme*, ('*b*, '*m1*) *Group-scheme*]
⇒ ('*a set* ⇒ '*b*) ((*3*-¨-,-) [*82*,*82*,*83*]*82*) **where**
f¨$_{F,G}$ == *induced-ghom F G f*

lemma *induced-ghom-someTr*:⟦*Group F*; *Group G*; *f* ∈ *gHom F G*;
X ∈ *set-rcs F* (*gker*$_{F,G}$ *f*)⟧ ⟹ *f* (*SOME xa*. *xa* ∈ *X*) ∈ *f* ‘(*carrier F*)
apply (*simp add*:*set-rcs-def*, *erule bexE*, *simp*)
apply (*frule gkernTr7* [*of F G f*], *assumption*+)
apply (*rule someI2-ex*)

apply (*frule-tac* $a = a$ **in** *Group.a-in-rcs*[*of* F $gker_{F,G}\ f$], *assumption*+)
apply *blast*
apply (*frule-tac* $a = a$ **and** $x = x$ **in** *Group.rcs-subset-elem*[*of* F $gker_{F,G}\ f$],
assumption+)
apply (*simp add*:*image-def*, *blast*)
done

lemma *induced-ghom-someTr1*:⟦*Group* F; *Group* G; $f \in gHom\ F\ G$; $a \in carrier\ F$⟧ ⟹
$f\ (SOME\ xa.\ xa \in (gker_{F,G}\ f)\ \cdot_F\ a) = f\ a$
apply (*rule someI2-ex*)
apply (*frule gkernTr7* [*of* F G f], *assumption*+)
apply (*frule Group.a-in-rcs* [*of* F $gker_{F,G}\ f$ a], *assumption*+)
apply *blast*
apply (*simp add*:*gkernTr11* [*THEN sym*])
apply (*simp add*:*iim-def*)
done

lemma *inducedHOMTr0*:⟦*Group* F; *Group* G; $f \in gHom\ F\ G$; $a \in carrier\ F$⟧ ⟹
$(f^{\cdot\cdot}{}_{F,G})\ ((gker_{F,G}\ f)\ \cdot_F\ a) = f\ a$
apply (*simp add*:*induced-ghom-def*)
apply (*frule gkernTr7*[*of* F G f], *assumption*+)
apply (*simp add*:*Group.rcs-in-set-rcs*[*of* F])
apply (*simp add*:*induced-ghom-someTr1*)
done

lemma *inducedHOMTr0-1*:⟦*Group* F; *Group* G; $f \in gHom\ F\ G$⟧ ⟹
$(f^{\cdot\cdot}{}_{F,G}) \in set\text{-}rcs\ F\ (gker_{F,G}\ f) \to carrier\ G$
apply (*rule Pi-I*)
apply (*simp add*:*set-rcs-def*, *erule bexE*, *simp*)
apply (*subst inducedHOMTr0*[*of* F G f], *assumption*+)
apply (*simp add*:*gHom-mem*)
done

lemma *inducedHOMTr0-2*:⟦*Group* F; *Group* G; $f \in gHom\ F\ G$⟧ ⟹
$(f^{\cdot\cdot}{}_{F,G}) \in set\text{-}rcs\ F\ (gker_{F,G}\ f) \to f\ `\ (carrier\ F)$
apply (*rule Pi-I*)
apply (*simp add*:*set-rcs-def*, *erule bexE*, *simp*)
apply (*subst inducedHOMTr0*[*of* F G f], *assumption*+)
apply (*simp add*:*image-def*, *blast*)
done

lemma *inducedHom*:⟦*Group* F; *Group* G; $f \in gHom\ F\ G$⟧ ⟹
$(f^{\cdot\cdot}{}_{F,G}) \in gHom\ (F/(gker_{F,G}\ f))\ G$
apply (*simp add*: *gHom-def* [*of* F/ $gker_{F,G}\ f\ G$])
apply (*rule conjI*)
apply (*simp add*:*induced-ghom-def extensional-def*)
apply (*rule allI*) **apply** (*rule impI*)+ **apply** (*simp add*:*Qg-def*)

apply (*rule conjI*)
 apply (*simp add*:*Qg-def inducedHOMTr0-1*)
apply (*rule ballI*)+
apply (*simp add*:*Qg-def set-rcs-def*, (*erule bexE*)+)
apply *simp*
 apply (*frule gker-normal*[*of F G f*], *assumption*+)
 apply (*simp add*:*Group.c-top-welldef* [*THEN sym*, *of F* $gker_{F,G}$ *f*])
 apply (*frule-tac a* = *a* **and** *b* = *aa* **in** *Group.mult-closed*[*of F*], *assumption*+)
 apply (*simp add*:*inducedHOMTr0*)
apply (*simp add*:*gHom*)
done

lemma *induced-ghom-ginjec*: ⟦*Group F*; *Group G*; *f* ∈ *gHom F G* ⟧ ⟹
 $ginj_{(F/(gker_{F,G}\ f)),G}\ (f^{\cdot\cdot}{}_{F,G})$
apply (*simp add*:*ginjec-def*)
apply (*simp add*:*inducedHom*)
apply (*simp add*:*inj-on-def*)
apply (*rule ballI*)+
apply (*simp add*:*Qg-def*)
apply (*simp add*:*set-rcs-def*)
apply ((*erule bexE*)+, *rule impI*, *simp*)
apply (*simp add*:*inducedHOMTr0*)
apply (*simp add*: *gkernTr11*[*THEN sym*])
done

lemma *inducedhomgsurjec*:⟦*Group F*; *Group G*; $gsurj_{F,G}\ f$⟧ ⟹
 $gsurj_{(F/(gker_{F,G}\ f)),G}\ (f^{\cdot\cdot}{}_{F,G})$
apply (*simp add*:*gsurjec-def*)
apply (*simp add*:*inducedHom*)
 apply (*erule conjE*)
 apply (*frule gHom-func*[*of F G f*], *assumption*+)
 apply (*rule surj-to-test*)
 apply (*simp add*:*Qg-def*)
 apply (*frule inducedHOMTr0-2*[*of F G f*], *assumption*+)
 apply (*simp add*:*surj-to-def*[*of f carrier F carrier G*])

apply (*rule ballI*)
 apply (*simp add*:*surj-to-def*[*of f carrier F carrier G*],
 frule sym, *thin-tac f* ‘ *carrier F* = *carrier G*, *simp*,
 thin-tac carrier G = *f* ‘ *carrier F*)
 apply (*simp add*:*image-def*, *erule bexE*, *simp*,
 thin-tac b = *f x*)
 apply (*simp add*:*Qg-def*)
 apply (*frule-tac a* = *x* **in** *inducedHOMTr0*[*of F G f*], *assumption*+)
 apply (*frule gkernTr7*[*of F G f*], *assumption*+)
 apply (*frule-tac a* = *x* **in** *Group.rcs-in-set-rcs*[*of F* $gker_{F,G}$ *f*],
 assumption+)
 apply *blast*

done

lemma *homomtr*: ⟦*Group F*; *Group G*; *f* ∈ *gHom F G*⟧ ⟹
(*f*¨$_{F,G}$) ∈ *gHom* (*F* / (*gker*$_{F,G}$ *f*)) (*Img*$_{F,G}$ *f*)
apply (*simp add*:*gHom-def* [*of F* / (*gker*$_{F,G}$ *f*) -])
apply (*rule conjI*)
apply (*simp add*:*induced-ghom-def extensional-def*)
apply (*rule allI*, (*rule impI*)+, *simp add*:*Qg-def*)
apply (*rule conjI*)
apply (*rule Pi-I*)
apply (*simp add*:*Gimage-def Qg-def Gp-def*, *simp add*:*set-rcs-def*, *erule bexE*)
apply *simp*
apply (*simp add*:*inducedHOMTr0*)

apply (*rule ballI*)+
apply (*simp add*:*Qg-def set-rcs-def*, (*erule bexE*)+, *simp*)
apply (*frule gker-normal*[*of F G f*], *assumption*+)
apply (*simp add*:*Group.c-top-welldef*[*THEN sym*, *of F gker*$_{F,G}$ *f*])
apply (*frule-tac a* = *a* **and** *b* = *aa* **in** *Group.mult-closed*[*of F*], *assumption*+)
apply (*simp add*:*inducedHOMTr0*)
apply (*simp add*:*Gimage-def*)
apply (*subst Group.Gp-mult-induced*[*of G f* ‘ *carrier F*], *assumption*+)
apply (*simp add*:*sg-gimg*)
apply (*simp add*:*image-def*, *blast*)
apply (*simp add*:*image-def*, *blast*)
apply (*simp add*:*gHom*)
done

lemma *homom2img*: ⟦*Group F*; *Group G*; *f* ∈ *gHom F G* ⟧ ⟹
(*f*¨$_{F,(Img_{F,G}\ f)}$) ∈ *gHom* (*F* / (*gker*$_{F,G}$ *f*)) (*Img*$_{F,G}$ *f*)
apply (*frule hom-to-Img*[*of F G f*], *assumption*+)
apply (*frule inducedHom*[*of F Img*$_{F,G}$ *f f*])
apply (*simp add*:*Group-Img*) **apply** *assumption*
apply (*simp add*:*gker-hom-to-img*[*of F G f*])
done

lemma *homom2img1*:⟦*Group F*; *Group G*; *f* ∈ *gHom F G*; *X* ∈ *set-rcs F* (*gker*$_{F,G}$ *f*)⟧
⟹ (*f*¨$_{F,(Img_{F,G}\ f)}$) *X* = (*f*¨$_{F,G}$) *X*
apply (*frule gker-hom-to-img* [*of F G f*], *assumption*+)
apply (*simp add*:*induced-ghom-def*)
done

3.10.1 Homomorphism therems

definition
iota :: (′*a*, ′*m*) *Group-scheme* ⇒ (′*a* ⇒ ′*a*)

((ι_) [*1000*]*999*) **where**

$\iota_F = (\lambda x \in carrier\ F.\ x)$

lemma *iotahomTr0*:$[\![Group\ G;\ G \gg H;\ h \in H\ ]\!] \Longrightarrow (\iota_{(Gp\ G\ H)})\ h = h$
apply (*simp add*:*iota-def*)
apply (*simp add*:*Gp-def*)
done

lemma *iotahom*:$[\![Group\ G;\ G \gg H;\ G \triangleright N]\!] \Longrightarrow$
$\iota_{(Gp\ G\ H)} \in gHom\ (Gp\ G\ H)\ (Gp\ G\ (H \diamond_G N))$
apply (*simp add*:*gHom-def*)
apply (*rule conjI*)
apply (*simp add*:*iota-def extensional-def*)
apply (*rule conjI*)
apply (*simp add*:*Pi-def restrict-def iota-def*)
apply (*rule allI*, *rule impI*)
apply (*simp add*:*Gp-def*)
apply (*frule Group.nsg-sg*[*of G N*], *assumption*+)
apply (*frule Group.l-sub-smult*[*of G H N*], *assumption*+,
simp add: *subsetD*)
apply (*rule ballI*)+
apply (*simp add*:*iota-def*, *simp add*:*Group.Gp-carrier*)
apply (*frule Group.smult-sg-nsg*[*of G H N*], *assumption*+,
frule Group.l-sub-smult[*of G H N*], *assumption*+,
simp add:*Group.nsg-sg*,
frule-tac c = *x* **in** *subsetD*[*of H H* $\diamond_G$ *N*], *assumption*+,
frule-tac c = *y* **in** *subsetD*[*of H H* $\diamond_G$ *N*], *assumption*+)
apply (*simp add*:*Group.Gp-mult-induced*[*of G*])
apply (*simp add*:*Group.sg-mult-closed*)
done

lemma *iotaTr0*: $[\![Group\ G;\ G \gg H;\ G \triangleright N]\!] \Longrightarrow$
$ginj_{(Gp\ G\ H),(Gp\ G\ (H \diamond_G N))}\ (\iota_{(Gp\ G\ H)})$
apply (*simp add*:*ginjec-def*)
apply (*simp add*:*iotahom*)
apply (*simp add*:*inj-on-def iota-def Gp-def*)
done

theorem *homomthm1*:$[\![Group\ F;\ Group\ G;\ f \in gHom\ F\ G\ ]\!] \Longrightarrow$
$gbij_{(F/\ (gkernel\ F\ G\ f)),\ (Gimage\ F\ G\ f)}\ (f^{\cdot\cdot}{}_F,\ (Gimage\ F\ G\ f))$
apply (*frule homom2img* [*of F G f*], *assumption*+)
apply (*simp add*:*gbijec-def*)
apply (*frule hom-to-Img* [*of F G f*], *assumption*+)
apply (*frule Group-coim*[*of F G f*], *assumption*+,
frule gHom-func[*of F / (gker*$_{F,G}$ *f*) *Img*$_{F,G}$ *f*
$f^{\cdot\cdot}{}_{F,Img_{F,G}}$ *f*], *simp add*:*Group-Img*, *assumption*)
apply (*rule conjI*)

apply (*simp add*:*gsurjec-def*,

rule surj-to-test[*of* $f^{\cdot\cdot}{}_{F,Img_{F,G}\ f}$
carrier $(F\ /\ gker_{F,G}\ f)$ *carrier* $(Img_{F,G}\ f)$], *assumption*+,
rule ballI)
apply (*thin-tac* $f^{\cdot\cdot}{}_{F,Img_{F,G}\ f} \in gHom\ (F\ /\ gker_{F,G}\ f)$
$(Img_{F,G}\ f)$)
apply (*simp add*:*Img-carrier*, *simp add*:*image-def*, *erule bexE*, *simp*,
simp add:*Qg-def*,
frule gkernTr7[*of F G f*], *assumption*+)
apply (*frule-tac a* = *x* **in** *Group.rcs-in-set-rcs*[*of F* $gker_{F,G}\ f$],
assumption+)
apply (*simp add*:*homom2img1*)
apply (*frule-tac a* = *x* **in** *inducedHOMTr0*[*of F G f*], *assumption*+)
apply *blast*

apply (*frule induced-ghom-ginjec*[*of F G f*], *assumption*+)
apply (*simp add*:*ginjec-def*)
apply (*frule conjunct2*)
apply (*thin-tac* $f^{\cdot\cdot}{}_{F,G} \in gHom\ (F\ /\ gker_{F,G}\ f)\ G \wedge$
inj-on $(f^{\cdot\cdot}{}_{F,G})$ $(carrier\ (F\ /\ gker_{F,G}\ f))$)
apply (*simp add*:*inj-on-def*)
apply ((*rule ballI*)+, *rule impI*)
apply (*simp add*:*Qg-def*, *fold Qg-def*)
apply (*simp add*:*homom2img1*)
done

lemma *isomTr0* [*simp*]:*Group F* $\Longrightarrow F \cong F$
by (*frule IdTr2* [*of F*], *simp add*:*isomorphic-def*,
blast)

lemma *isomTr1*:$[\![$*Group F*; *Group G*; $F \cong G\ ]\!] \Longrightarrow G \cong F$
apply (*simp add*:*isomorphic-def*, *erule exE*)
apply (*frule-tac f* = *f* **in** *inv-gbijec-gbijec*[*of F G*], *assumption*+)
apply *blast*
done

lemma *isomTr2*:$[\![$*Group F*; *Group G*; *Group H*; $F \cong G$; $G \cong H\ ]\!] \Longrightarrow F \cong H$
apply (*simp add*:*isomorphic-def*)
apply (*erule exE*)+
apply (*simp add*:*gbijec-def*)
apply (*erule conjE*)+
apply (*frule gHom-comp-gsurjec* [*of F G H - -*], *assumption*+)
apply (*frule gHom-comp-ginjec* [*of F G H - -*], *assumption*+)
apply *blast*
done

lemma *gisom1*: $[\![$*Group F*; *Group G*; $f \in gHom\ F\ G\ ]\!] \Longrightarrow$
$(F/\ (gker_{F,G}\ f)) \cong (Img_{F,G}\ f)$
apply (*simp add*:*isomorphic-def*)
apply (*frule homomthm1* [*of F G f*], *assumption*+)

apply *blast*
done

lemma *homomth2Tr0*: $[\![Group\ F;\ Group\ G;\ f \in gHom\ F\ G;\ G \triangleright N]\!] \Longrightarrow$
$F \triangleright (iim\ F\ G\ f\ N)$
apply (*frule Group.cond-nsg*[*of F iim F G f N*],
frule Group.nsg-sg[*of G N*], *assumption*+,
simp add:*ghomTr4*[*of F G f N*])
apply ((*rule ballI*)+,
simp add:*iim-def*, *erule conjE*,
frule-tac a = *a* **in** *Group.i-closed*[*of F*], *assumption*+,
frule-tac a = *a* **and** *b* = *h* **in** *Group.mult-closed*[*of F*], *assumption*+)
apply (*simp add*:*gHom ghom-inv-inv Group.mult-closed*)
apply (*frule-tac x* = *a* **in** *gHom-mem*[*of F G f*], *assumption*+,
simp add:*Group.nsgPr1-1*,
assumption)
done

lemma *kern-comp-gHom*:$[\![Group\ F;\ Group\ G;\ gsurj_{F,G}\ f;\ G \triangleright N]\!] \Longrightarrow$
$gker_{F,\ (G/N)}\ ((Pj\ G\ N) \circ_F f) = iim\ F\ G\ f\ N$
apply (*simp add*:*gkernel-def iim-def*)
apply (*simp add*:*Group.Qg-one*[*of G N*] *cmpghom-def compose-def*)
apply (*rule equalityI*)
apply (*rule subsetI*, *simp*, *erule conjE*, *simp*)
apply (*rule subsetI*, *simp*, *erule conjE*, *fold gsurjec-def*)
apply (*frule-tac x* = *x* **in** *gHom-mem*[*of F G f*], *assumption*+)
apply (*simp add*:*Group.Pj-mem*[*of G N*])
apply (*frule Group.nsg-sg*[*of G N*], *assumption*+)
apply (*frule-tac a* = *f x* **in** *Group.a-in-rcs*[*of G N*], *assumption*+)
apply *simp*
apply (*rule subsetI*)
apply (*simp*, *erule conjE*)
apply (*frule Group.nsg-sg*[*of G N*], *assumption*,
frule-tac h = *f x* **in** *Group.sg-subset-elem*[*of G N*], *assumption*+)
apply (*simp add*:*Group.Pj-mem*[*of G N*])
apply (*simp add*:*Group.rcs-fixed2*[*of G N*])
done

lemma *QgrpUnit-1*:$[\![Group\ G;\ Ugp\ E;\ G \triangleright H;\ (G\ /\ H) \cong E\ ]\!] \Longrightarrow carrier\ G = H$
apply (*simp add*:*isomorphic-def*, *erule exE*)
apply (*frule Group.Group-Qg*[*of G H*], *assumption*+,
simp add:*gbijec-def*, *erule conjE*)
apply (*frule-tac f* = *f* **in** *gkern2*[*of G / H E*],
simp add:*Ugp-def*, *simp add*:*gsurjec-def*, *assumption*,
simp add:*gsurjec-def*, *frule conjunct1*, *fold gsurjec-def*,
frule-tac f = *f* **in** *gkern1*[*of G*/*H E*], *assumption*+)
apply (*simp*, *thin-tac* $gker_{(G\ /\ H),E}\ f = \{\mathbf{1}_{G\ /\ H}\}$,
thin-tac $gsurj_{(G\ /\ H),E}\ f$, *thin-tac* $ginj_{(G\ /\ H),E}\ f$,

thin-tac Group (G / H), thin-tac f ∈ gHom (G / H) E,
simp add:Group.Qg-carrier)
apply (*rule contrapos-pp*, *simp*+,
frule Group.nsg-sg[*of G H*], *assumption*+,
frule Group.sg-subset[*of G H*], *assumption*+,
frule sets-not-eq[*of carrier G H*], *assumption*, *erule bexE*,
frule-tac a = a **in** *Group.rcs-in-set-rcs*[*of G H*], *assumption*+,
simp)
apply (*thin-tac set-rcs G H* = {$\mathbf{1}_{G\ /\ H}$}, *simp add:Qg-def*,
frule-tac a = a **in** *Group.a-in-rcs*[*of G H*], *assumption*+,
simp)
done

lemma *QgrpUnit-2*:⟦*Group G*; *Ugp E*; *G ▷ H*; *carrier G* = *H*⟧ ⟹ (*G*/*H*) ≅ *E*
apply (*frule Group.Group-Qg* [*of G H*], *assumption*+)
apply (*simp add:Group.Qg-unit-group*[*THEN sym*, *of G H*])
apply (*simp add:Ugp-def*)
apply (*frule Group.Qg-carrier*[*of G H*], *simp*)
apply (*thin-tac set-rcs G H* = {*H*})
apply (*frule Group.Qg-one*[*of G H*], *assumption*+, *erule conjE*)
apply (*rule Ugps-isomorphic*[*of G / H E*])
apply (*simp add:Ugp-def*)+
done

lemma *QgrpUnit-3*:⟦*Group G*; *Ugp E*; *G ≫ H*; *G ≫ H1*; (*Gp G H*) ▷ *H1*;
((*Gp G H*) / *H1*) ≅ *E* ⟧ ⟹ *H* = *H1*
apply (*frule Group.Group-Gp*[*of G H*], *assumption*+)
apply (*frule QgrpUnit-1* [*of Gp G H E H1*], *assumption*+)
apply (*simp add:Group.Gp-carrier*)
done

lemma *QgrpUnit-4*:⟦*Group G*; *Ugp E*; *G ≫ H*; *G ≫ H1*; (*Gp G H*) ▷ *H1*;
¬ ((*Gp G H*) / *H1*) ≅ *E* ⟧ ⟹ *H* ≠ *H1*
apply (*frule Group.Group-Gp*[*of G H*], *assumption*+)
apply (*rule contrapos-pp*, *simp*) **apply** *simp*
apply (*frule QgrpUnit-2* [*of Gp G H1 E H1*], *assumption*+)
apply (*simp add:Group.Gp-carrier*)
apply *simp*
done

definition
Qmp :: [('*a*, '*m*) *Group-scheme*, '*a set*, '*a set*] ⇒ ('*a set* ⇒ '*a set*) **where**
Qmp G H N = (λ*X*∈ *set-rcs G H*. {*z*. ∃ *x* ∈ *X*. ∃ *y* ∈ *N*. (*y* ·$_G$ *x* = *z*)})

abbreviation
QP :: [-, '*a set*, '*a set*] ⇒ ('*a set* ⇒ '*a set*)
((*3Qm*_ _,_) [*82*,*82*,*83*]*82*) **where**
Qm$_{G\ H,N}$ == *Qmp G H N*

lemma (**in** *Group*) *QmpTr0*:⟦*G* ≫ *H*; *G* ≫ *N*; *H* ⊆ *N* ; *a* ∈ *carrier G*⟧ ⟹
Qmp G H N (*H* ⋅ *a*) = (*N* ⋅ *a*)
apply (*frule-tac a* = *a* **in** *rcs-in-set-rcs*[*of H*], *assumption*,
simp add:*Qmp-def*)
apply (*rule equalityI*)
apply (*rule subsetI*, *simp*, (*erule bexE*)+,
thin-tac H ⋅ *a* ∈ *set-rcs G H*,
simp add:*rcs-def*, *erule bexE*)
apply (*frule sym*, *thin-tac y* ⋅ *xa* = *x*, *frule sym*, *thin-tac h* ⋅ *a* = *xa*,
simp,
frule-tac h = *y* **in** *sg-subset-elem*[*of N*], *assumption*+,
frule-tac h = *h* **in** *sg-subset-elem*[*of H*], *assumption*+,
frule-tac c = *h* **in** *subsetD*[*of H N*], *assumption*+,
frule-tac x = *y* **and** *y* = *h* **in** *sg-mult-closed*[*of N*], *assumption*+,
subst tassoc[*THEN sym*], *assumption*+, *blast*)

apply (*rule subsetI*,
thin-tac H ⋅ *a* ∈ *set-rcs G H*,
simp add:*rcs-def*, *erule bexE*,
frule sg-unit-closed[*of H*],
frule l-unit[*of a*], *blast*)
done

lemma (**in** *Group*) *QmpTr1*:⟦*G* ≫ *H*; *G* ≫ *N*; *H* ⊆ *N*; *a* ∈ *carrier G*; *b* ∈ *carrier G*;
H ⋅ *a* = *H* ⋅ *b*⟧ ⟹ *N* ⋅ *a* = *N* ⋅ *b*
apply (*simp add*:*rcs-eq*[*THEN sym*, *of H a b*],
frule subsetD[*of H N b* ⋅ *ϱ a*], *assumption*+,
simp add:*rcs-eq*[*of N a b*])
done

lemma (**in** *Group*) *QmpTr2*:⟦*G* ≫ *H*; *G* ≫ *N*; *H* ⊆ *N* ; *X* ∈ *carrier* (*G*/*H*)⟧
⟹ (*Qmp G H N*) *X* ∈ *carrier* (*G*/*N*)
by (*simp add*:*Qg-carrier*[*of H*] *set-rcs-def*, *erule bexE*, *simp add*: *QmpTr0*,
simp add:*Qg-carrier rcs-in-set-rcs*)

lemma (**in** *Group*) *QmpTr2-1*:⟦*G* ≫ *H*; *G* ≫ *N*; *H* ⊆ *N* ⟧ ⟹
Qmp G H N ∈ *carrier* (*G*/*H*) → *carrier* (*G*/*N*)
by (*simp add*:*QmpTr2* [*of H N*])

lemma (**in** *Group*) *QmpTr3*:⟦*G* ▷ *H*; *G* ▷ *N*; *H* ⊆ *N*; *X* ∈ *carrier* (*G*/*H*);
Y ∈ *carrier* (*G*/*H*)⟧ ⟹
(*Qmp G H N*) (*c-top G H X Y*) = *c-top G N* ((*Qmp G H N*) *X*) ((*Qmp G H N*) *Y*)

apply (*frule nsg-sg*[*of H*], *frule nsg-sg*[*of N*])
apply (*simp add*:*Qg-carrier*)
apply (*simp add*:*set-rcs-def*, (*erule bexE*)+, *simp*)
apply (*subst c-top-welldef* [*THEN sym*], *assumption*+)
apply (*frule-tac* $a = a$ **and** $b = aa$ **in** *mult-closed*, *assumption*+)
apply (*simp add*:*QmpTr0*)+
apply (*subst c-top-welldef* [*THEN sym*], *assumption*+)
apply *simp*
done

lemma (**in** *Group*) *Gp-s-mult-nsg*:$[\![G \triangleright H;\ G \triangleright N;\ H \subseteq N;\ a \in N\ ]\!] \Longrightarrow$
$H \cdot_{(Gp\ G\ N)} a = H \cdot a$
by (*frule nsg-sg*[*of H*], *frule nsg-sg*[*of N*],
rule Gp-rcs[*of H N a*], *assumption*+)

lemma (**in** *Group*) *QmpTr5*:$[\![G \triangleright H;\ G \triangleright N;\ H \subseteq N;\ X \in carrier\ (G/H);$
$Y \in carrier\ (G/H)\ ]\!] \Longrightarrow (Qmp\ G\ H\ N)\ (\ X \cdot_{(G\ /\ H)}\ Y) =$
$((Qmp\ G\ H\ N)\ X) \cdot_{(G\ /\ N)} ((Qmp\ G\ H\ N)\ Y)$
by (*frule nsg-sg*[*of H*], *frule nsg-sg*[*of N*],
(*subst Qg-def*)+, *simp*,
simp add:*QmpTr3* [*of H N X Y*])

lemma (**in** *Group*) *QmpTr*:$[\![G \triangleright H;\ G \triangleright N;\ H \subseteq N\ ]\!] \Longrightarrow$
$(Qm_{G\ H,N}) \in gHom\ (G\ /\ H)\ (G\ /\ N)$
apply (*simp add*:*gHom-def*)
apply (*rule conjI*)
apply (*simp add*:*Qmp-def extensional-def*)
apply (*rule allI*, (*rule impI*)+, *simp add*:*Qg-def*)
apply (*rule conjI*)
apply (*rule QmpTr2-1*[*of H N*])
apply (*simp add*:*nsg-sg*)+
apply (*rule ballI*)+
apply (*simp add*:*QmpTr5*)
done

lemma (**in** *Group*) *Qmpgsurjec*:$[\![G \triangleright H;\ G \triangleright N;\ H \subseteq N\ ]\!] \Longrightarrow$
$gsurj_{(G\ /\ H),(G\ /\ N)}\ (Qm_{G\ H,N})$
apply (*frule nsg-sg*[*of H*], *frule nsg-sg*[*of N*])
apply (*frule QmpTr* [*of H N*], *assumption*+)
apply (*simp add*:*gsurjec-def*)
apply (*rule surj-to-test*)
apply (*simp add*:*QmpTr2-1*)
apply (*rule ballI*)
apply (*simp add*:*Qg-carrier*,
simp add:*set-rcs-def*[*of G N*], *erule bexE*,
frule-tac $a = a$ **in** *QmpTr0*[*of H N*], *assumption*+, *simp*)
apply (*frule-tac* $a = a$ **in** *rcs-in-set-rcs*[*of H*], *assumption*+,
blast)
done

lemma (**in** *Group*) *gkerQmp*:⟦$G \triangleright H$; $G \triangleright N$; $H \subseteq N$ ⟧ ⟹
$gker_{(G\ /\ H),(G\ /\ N)}\ (Qm_{G\ H,N}) = carrier\ ((Gp\ G\ N)/\ H)$
apply (*frule nsg-sg*[*of H*], *frule nsg-sg*[*of N*])
apply (*simp add*:*gkernel-def*)
apply (*rule equalityI*)
apply (*rule subsetI*,
simp add:*Qg-carrier set-rcs-def*, *erule conjE*, *erule bexE*, *simp*,
simp add:*Qg-one*)
apply (*simp add*:*QmpTr0*,
frule-tac a = *a* **in** *a-in-rcs*[*of N*], *assumption*+, *simp*,
frule Group-Gp[*of N*])
apply (*simp add*:*Group.Qg-carrier*, *simp add*:*set-rcs-def*, *simp add*:*Gp-carrier*,
simp add:*Gp-rcs*, *blast*)

apply (*rule subsetI*)
apply (*frule Group-Gp*[*of N*],
simp add:*Group.Qg-carrier Qg-one set-rcs-def*, *erule bexE*,
simp add:*Qg-carrier*, *thin-tac x* = *H* ·$_{\natural N}$ *a*)
apply (*simp add*:*Gp-carrier*, *simp add*:*Gp-rcs*,
frule-tac h = *a* **in** *sg-subset-elem*[*of N*], *assumption*,
simp add:*rcs-in-set-rcs*, *simp add*:*QmpTr0*,
simp add:*rcs-fixed2*[*of N*])
done

theorem (**in** *Group*) *homom2*:⟦$G \triangleright H$; $G \triangleright N$; $H \subseteq N$⟧ ⟹
$gbij_{((G/H)/(carrier\ ((Gp\ G\ N)/H))),(G/N)}\ ((Qm_{G\ H,N})^{\cdot\cdot}{}_{(G/H),(G/N)})$
apply (*frule QmpTr* [*of H N*], *assumption*+)
apply (*simp add*:*gbijec-def*)
apply (*rule conjI*)
apply (*frule Group-Qg*[*of H*], *frule Group-Qg*[*of N*])
apply (*frule inducedHom* [*of G*/*H G*/*N Qmp G H N*], *assumption*+)

apply (*frule Qmpgsurjec* [*of H N*], *assumption*+)
apply (*frule inducedhomgsurjec* [*of G*/*H G*/*N* $Qm_{G\ H,N}$], *assumption*+)
apply (*simp add*:*gkerQmp* [*of H N*])

apply (*frule QmpTr* [*of H N*], *assumption*+)
apply (*frule Group-Qg*[*of H*], *frule Group-Qg*[*of N*])
apply (*frule induced-ghom-ginjec* [*of G*/*H G*/*N Qmp G H N*], *assumption*+)
apply (*simp add*:*gkerQmp* [*of H N*])
done

3.11 Isomorphims

theorem (**in** *Group*) *isom2*:⟦$G \triangleright H$; $G \triangleright N$; $H \subseteq N$⟧ ⟹
$((G/H)/(carrier\ ((Gp\ G\ N)/H))) \cong\ (G/N)$

apply (*frule homom2* [*of H N*], *assumption*+)
apply (*simp add*:*isomorphic-def*)
apply *blast*
done

theorem *homom3*:⟦ *Group F*; *Group G*; $G \triangleright N$; $gsurj_{F,G}\ f$;
N1 = (*iim F G f*) *N* ⟧ ⟹ (*F* / *N1*) ≅ (*G* / *N*)
apply (*frule Group.Pj-gsurjec* [*of G N*], *assumption*+)
apply (*frule Group.Group-Qg*[*of G N*], *assumption*)
apply (*frule gHom-comp-gsurjec* [*of F G G* / *N f Pj G N*], *assumption*+)
apply (*frule inducedhomgsurjec* [*of F G* / *N* (*Pj G N*) $\circ_F$ *f*], *assumption*+)
apply (*frule induced-ghom-ginjec* [*of F G* / *N* (*Pj G N*) $\circ_F$ *f*], *assumption*+)
apply (*simp add*:*gsurjec-def* [*of F G* / *N* (*Pj G N*) $\circ_F$ *f*])
apply (*simp add*:*kern-comp-gHom*[*of F G f N*])
apply (*frule sym*, *thin-tac N1* = *iim F G f N*, *simp*)
apply (*simp add*:*isomorphic-def gbijec-def*, *blast*)
done

lemma (**in** *Group*) *homom3Tr1*:⟦$G \gg H$; $G \triangleright N$⟧ ⟹ $H \cap N$ =
$gker_{(Gp\ G\ H),((Gp\ G\ (H \diamond_G N))/N)}$
((*Pj* (*Gp G* (*H* $\diamond_G$ *N*)) *N*) $\circ_{(Gp\ G\ H)}$ ($\iota_{(Gp\ G\ H)}$))
apply (*simp add*:*gkernel-def*, *frule nsg-sg*,
simp add:*Gp-carrier*[*of H*],
frule smult-sg-nsg, *assumption*+,
frule Gp-smult-nsg[*of H N*], *assumption*,
frule Group-Gp [*of H* $\diamond_G$ *N*])
apply (*simp add*:*Group.Qg-one*[*of Gp G* (*H* $\diamond_G$ *N*) *N*],
simp add:*iota-def Gp-carrier*, *simp add*:*cmpghom-def compose-def*,
simp add:*Gp-carrier*)
apply (*rule equalityI*)
apply (*rule subsetI*, *simp*, *erule conjE*)
apply (*frule-tac h* = *x* **in** *sg-subset-elem*[*of H*], *assumption*+,
subst Group.Pj-mem, *assumption*+,
simp add:*Gp-carrier*,
frule l-sub-smult[*of H N*], *assumption*+,
rule-tac c = *x* **in** *subsetD*[*of H H* $\diamond_G$ *N*], *assumption*+)
apply (*frule r-sub-smult*[*of H N*], *assumption*+,
frule-tac c = *x* **in** *subsetD*[*of N H* $\diamond_G$ *N*], *assumption*+,
simp add:*Gp-rcs*[*of N H* $\diamond_G$ *N*])
apply (*simp add*:*rcs-fixed2*)

apply (*rule subsetI*, *simp*, *erule conjE*, *simp*)
apply (*frule-tac h* = *x* **in** *sg-subset-elem*[*of H*], *assumption*+)
apply (*frule l-sub-smult*[*of H N*], *assumption*+,
frule r-sub-smult[*of H N*], *assumption*+)
apply (*frule-tac x* = *x* **in** *Group.Pj-mem*[*of Gp G* (*H* $\diamond_G$ *N*) *N*], *assumption*+)
apply (*simp add*:*Gp-carrier*)
apply (
rule-tac c = *x* **in** *subsetD*[*of H H* $\diamond_G$ *N*], *assumption*+)

apply (*frule-tac* $c = x$ **in** *subsetD*[*of* H H $\diamond_G$ N], *assumption*+)
apply (*simp only*:*Group.Gp-rcs*)
apply (*simp only*:*Gp-rcs*[*of* N H $\diamond_G$ N])
apply (*frule-tac* $a = x$ **in** *a-in-rcs*[*of* N], *assumption*+, *simp*)
done

3.11.1 An automorphism groups

definition
automg :: - ⇒
(⦇ *carrier* :: ('*a* ⇒ '*a*) *set*, *top* :: ['*a* ⇒ '*a*,'*a* ⇒ '*a*] ⇒ ('*a* ⇒ '*a*),
iop :: ('*a* ⇒ '*a*) ⇒ ('*a* ⇒ '*a*), *one* :: ('*a* ⇒ '*a*)⦈ **where**
automg G = ⦇ *carrier* = {f. *gbij*$_{G,G}$ f},
top = λg∈{f. *gbij*$_{G,G}$ f}. λf∈{f. *gbij*$_{G,G}$ f}. (g ∘$_G$ f),
iop = λf∈{f. *gbij*$_{G,G}$ f}. (*Ifn* G G f), *one* = I_G ⦈

lemma *automgroupTr1*:⟦*Group* G; *gbij*$_{G,G}$ f; *gbij*$_{G,G}$ g; *gbij*$_{G,G}$ h⟧ ⟹
(h ∘$_G$ g) ∘$_G$ f = h ∘$_G$ (g ∘$_G$ f)
apply (*simp add*:*cmpghom-def*,
unfold gbijec-def)
apply (*frule conjunct1*, *rotate-tac 2*, *frule conjunct1*,
rotate-tac 1, *frule conjunct1*, *fold gbijec-def*)
apply (*simp add*:*gsurjec-def*, (*erule conjE*)+,
frule gHom-func[*of* G G f], *assumption*+,
frule gHom-func[*of* G G g], *assumption*+,
frule gHom-func[*of* G G h], *assumption*+)
apply (*simp add*:*compose-assoc*)
done

lemma *automgroup*:*Group* G ⟹ *Group* (*automg* G)
apply (*unfold Group-def* [*of automg* G])
apply(*auto simp*: *automg-def Pi-def gbij-comp-bij automgroupTr1 IdTr2 Id-l-unit l-inv-gHom inv-gbijec-gbijec*)
done

3.11.2 Complete system of representatives

definition
gcsrp :: - ⇒ '*a set* ⇒ '*a set* ⇒ *bool* **where**
gcsrp G H S == ∃f. (*bij-to* f (*set-rcs* G H) S)

definition
gcsrp-map::- ⇒ '*a set* ⇒ '*a set* ⇒ '*a* **where**
gcsrp-map G H == λX∈(*set-rcs* G H). *SOME* x. x ∈ X

lemma (**in** *Group*) *gcsrp-func*:G ≫ H ⟹ *gcsrp-map* G H ∈ *set-rcs* G H → *UNIV*
by (*simp add*:*set-rcs-def*)

lemma (**in** *Group*) *gcsrp-func1*:G ≫ H ⟹

```
      gcsrp-map G H ∈ set-rcs G H → (gcsrp-map G H) ‘ (set-rcs G H)
by (simp add:set-rcs-def)

lemma (in Group) gcsrp-map-bij:G » H ⟹
       bij-to (gcsrp-map G H) (set-rcs G H) ((gcsrp-map G H) ‘(set-rcs G H))
apply (simp add:bij-to-def, rule conjI)
 apply (rule surj-to-test)
 apply (rule Pi-I)
 apply (simp add:image-def, blast)
apply (rule ballI, simp add:image-def, erule bexE, simp, blast)

apply (simp add:inj-on-def)
 apply ((rule ballI)+, rule impI)
 apply (simp add:gcsrp-map-def)
 apply (frule-tac X = x in rcs-nonempty, assumption+,
       frule-tac X = y in rcs-nonempty, assumption+)
 apply (frule-tac A = x in nonempty-some,
       frule-tac A = y in nonempty-some, simp)
apply (rule-tac X = x and Y = y in rcs-meet[of H], assumption+)
apply blast
done

lemma (in Group) image-gcsrp:G » H ⟹
                gcsrp G H ((gcsrp-map G H) ‘(set-rcs G H))
apply (simp add:gcsrp-def)
apply (frule gcsrp-map-bij[of H], blast)
done

lemma (in Group) gcsrp-exists:G » H ⟹ ∃ S. gcsrp G H S
by (frule image-gcsrp[of H], blast)

definition
 gcsrp-top :: [- , 'a set] ⇒  'a ⇒  'a  ⇒ 'a where
 gcsrp-top G H == λx ∈ ((gcsrp-map G H) ‘(set-rcs G H)).
                  λy ∈ ((gcsrp-map G H) ‘(set-rcs G H)).
 gcsrp-map G H
  (c-top G H
  ((invfun (set-rcs G H) ((gcsrp-map G H) ‘(set-rcs G H)) (gcsrp-map G H)) x)
  ((invfun (set-rcs G H) ((gcsrp-map G H) ‘(set-rcs G H)) (gcsrp-map G H)) y))

definition
  gcsrp-iop::[- , 'a set] ⇒ 'a ⇒ 'a where
  gcsrp-iop G H = (λx ∈ ((gcsrp-map G H) ‘(set-rcs G H)).
    gcsrp-map G H
    (c-iop G H
    ((invfun (set-rcs G H) ((gcsrp-map G H) ‘(set-rcs G H)) (gcsrp-map G H))
x)))

definition
```

gcsrp-one::[- , *'a set*] ⇒ *'a* **where**
gcsrp-one G H = *gcsrp-map G H H*

definition
Gcsrp :: - ⇒ *'a set* ⇒ *'a Group* **where**
Gcsrp G N = (⌊*carrier* = (*gcsrp-map G N*) ‘(*set-rcs G N*),
top = *gcsrp-top G N*, *iop* = *gcsrp-iop G N*, *one* = *gcsrp-one G N*⌋)

lemma (**in** *Group*) *gcsrp-top-closed*:⟦*Group G*; *G* ▷ *N*;
a ∈ ((*gcsrp-map G N*) ‘(*set-rcs G N*)); *b* ∈ ((*gcsrp-map G N*) ‘(*set-rcs G N*))⟧
⟹ *gcsrp-top G N a b* ∈ (*gcsrp-map G N*) ‘(*set-rcs G N*)
apply (*frule nsg-sg*[*of N*],
frule gcsrp-func1[*of N*],
frule gcsrp-map-bij[*of N*])
apply (*frule invfun-mem1*[*of gcsrp-map G N set-rcs G N*
(*gcsrp-map G N*) ‘ (*set-rcs G N*) *a*], *assumption*+,
frule invfun-mem1[*of gcsrp-map G N set-rcs G N*
(*gcsrp-map G N*) ‘ (*set-rcs G N*) *b*], *assumption*+)
apply (*frule Qg-top-closed*[*of N invfun* (*set-rcs G N*)
(*gcsrp-map G N* ‘ *set-rcs G N*) (*gcsrp-map G N*) *a*
invfun (*set-rcs G N*)
(*gcsrp-map G N* ‘ *set-rcs G N*) (*gcsrp-map G N*) *b*], *assumption*+)
apply (*simp add*:*gcsrp-top-def*)
done

lemma (**in** *Group*) *gcsrp-tassoc*:⟦*Group G*; *G* ▷ *N*;
a ∈ ((*gcsrp-map G N*) ‘(*set-rcs G N*));
b ∈ ((*gcsrp-map G N*) ‘(*set-rcs G N*));
c ∈ ((*gcsrp-map G N*) ‘(*set-rcs G N*))⟧ ⟹
(*gcsrp-top G N* (*gcsrp-top G N a b*) *c*) =
(*gcsrp-top G N a* (*gcsrp-top G N b c*))
apply (*frule nsg-sg*[*of N*],
frule gcsrp-func1[*of N*],
frule gcsrp-map-bij[*of N*])
apply (*frule invfun-mem1*[*of gcsrp-map G N set-rcs G N*
(*gcsrp-map G N*) ‘ (*set-rcs G N*) *a*], *assumption*+,
frule invfun-mem1[*of gcsrp-map G N set-rcs G N*
(*gcsrp-map G N*) ‘ (*set-rcs G N*) *b*], *assumption*+,
frule invfun-mem1[*of gcsrp-map G N set-rcs G N*
(*gcsrp-map G N*) ‘ (*set-rcs G N*) *c*], *assumption*+)
apply (*frule Qg-top-closed*[*of N invfun* (*set-rcs G N*)
(*gcsrp-map G N* ‘ *set-rcs G N*) (*gcsrp-map G N*) *a*
invfun (*set-rcs G N*)
(*gcsrp-map G N* ‘ *set-rcs G N*) (*gcsrp-map G N*) *b*], *assumption*+,
frule Qg-top-closed[*of N invfun* (*set-rcs G N*)
(*gcsrp-map G N* ‘ *set-rcs G N*) (*gcsrp-map G N*) *b*
invfun (*set-rcs G N*)
(*gcsrp-map G N* ‘ *set-rcs G N*) (*gcsrp-map G N*) *c*], *assumption*+)
apply (*simp add*:*gcsrp-top-def*)

apply (*simp add*:*invfun-l1*)
apply (*simp add*:*Qg-tassoc*[*of N*])
done

lemma (**in** *Group*) *gcsrp-l-one*:⟦*Group G*; *G* ▷ *N*;
a ∈ ((*gcsrp-map G N*) ‘(*set-rcs G N*))⟧ ⟹
(*gcsrp-top G N* (*gcsrp-one G N*) *a*) = *a*
apply (*frule nsg-sg*[*of N*],
frule gcsrp-func1[*of N*],
frule gcsrp-map-bij[*of N*],
frule invfun-mem1[*of gcsrp-map G N set-rcs G N*
(*gcsrp-map G N*) ‘ (*set-rcs G N*) *a*], *assumption*+)
apply (*simp add*:*gcsrp-top-def gcsrp-one-def*)
apply (*frule Qg-unit-closed*[*of N*])
apply (*simp add*:*Pi-def invfun-l1 Qg-unit invfun-r1*)
done

lemma (**in** *Group*) *gcsrp-l-i*:⟦*G* ▷ *N*; *a* ∈ ((*gcsrp-map G N*) ‘(*set-rcs G N*))⟧ ⟹
gcsrp-top G N (*gcsrp-iop G N a*) *a* = *gcsrp-one G N*
apply (*frule nsg-sg*[*of N*],
frule gcsrp-func1[*of N*],
frule gcsrp-map-bij[*of N*],
frule invfun-mem1[*of gcsrp-map G N set-rcs G N*
(*gcsrp-map G N*) ‘ (*set-rcs G N*) *a*], *assumption*+)
apply (*frule Qg-iop-closed*[*of N invfun* (*set-rcs G N*) (*gcsrp-map G N* ‘
set-rcs G N) (*gcsrp-map G N*) *a*], *assumption*+)
apply (*simp add*:*gcsrp-top-def gcsrp-one-def gcsrp-iop-def*)
apply (*simp add*:*invfun-l1 Qg-i*)
done

lemma (**in** *Group*) *gcsrp-i-closed*:⟦*G* ▷ *N*; *a* ∈ ((*gcsrp-map G N*) ‘(*set-rcs G N*))⟧
⟹ *gcsrp-iop G N a* ∈ ((*gcsrp-map G N*) ‘(*set-rcs G N*))
apply (*frule nsg-sg*[*of N*],
frule gcsrp-func1[*of N*],
frule gcsrp-map-bij[*of N*],
frule invfun-mem1[*of gcsrp-map G N set-rcs G N*
(*gcsrp-map G N*) ‘ (*set-rcs G N*) *a*], *assumption*+)
apply (*frule Qg-iop-closed*[*of N invfun* (*set-rcs G N*) (*gcsrp-map G N* ‘
set-rcs G N) (*gcsrp-map G N*) *a*], *assumption*+)
apply (*simp add*:*gcsrp-iop-def*)
done

lemma (**in** *Group*) *Group-Gcsrp*:*G* ▷ *N* ⟹ *Group* (*Gcsrp G N*)
apply (*simp add*:*Group-def*)
apply (*rule conjI*)
apply (*rule Pi-I*)
apply (*simp add*:*Gcsrp-def*)
apply (*rule Pi-I*)
apply (*rule-tac a* = *x* **and** *b* = *xa* **in** *gcsrp-top-closed*[*of N*], *rule Group-axioms*,

assumption+)
apply (*rule conjI*)
apply (*rule allI*, *rule impI*)+
apply (*simp add*:*Gcsrp-def*)
apply (*rule-tac* $a = a$ **and** $b = b$ **and** $c = c$ **in** *gcsrp-tassoc*[*of N*], *rule Group-axioms*, *assumption*+)
apply (*rule conjI*)
apply (*rule Pi-I*)
apply (*simp add*:*Gcsrp-def*, *rule gcsrp-i-closed*[*of N*], *assumption*+)
apply (*rule conjI*)
apply (*rule allI*, *rule impI*)
apply (*simp add*:*Gcsrp-def*,
rule gcsrp-l-i[*of N*], *assumption*+)
apply (*rule conjI*)
apply (*frule Qg-unit-closed*[*of N*],
simp add:*Gcsrp-def gcsrp-one-def*)
apply (*rule allI*, *rule impI*)
apply (*simp add*:*Gcsrp-def*)
apply (*rule gcsrp-l-one*[*of N*], *rule Group-axioms*, *assumption*+)
done

lemma (**in** *Group*) *gcsrp-map-gbijec*:$G \triangleright N \Longrightarrow$
$gbij_{(G/N),\ (Gcsrp\ G\ N)}\ (gcsrp\text{-}map\ G\ N)$
apply (*simp add*:*gbijec-def gsurjec-def ginjec-def Qg-carrier Gcsrp-def*)
apply (*frule nsg-sg*[*of N*],
frule gcsrp-map-bij[*of N*], *simp add*:*bij-to-def*)
apply (*fold Gcsrp-def*)
apply (*simp add*:*gHom-def*)

apply (*rule conjI*)
apply (*simp add*:*Qg-carrier gcsrp-map-def*)
apply (*rule conjI*)
apply (*simp add*:*Qg-carrier Gcsrp-def*)
apply (*fold bij-to-def*)
apply (*rule ballI*)+
apply (*simp add*:*Qg-def Gcsrp-def gcsrp-top-def*)
apply (*frule gcsrp-func1*[*of N*])
apply (*simp add*:*invfun-l1*[*of gcsrp-map G N set-rcs G N*
gcsrp-map G N ‘ set-rcs G N])
done

lemma (**in** *Group*) *Qg-equiv-Gcsrp*:$G \triangleright N \Longrightarrow (G\ /\ N) \cong Gcsrp\ G\ N$
apply (*simp add*:*isomorphic-def*)
apply (*frule gcsrp-map-gbijec*[*of N*], *blast*)
done

3.12 Zassenhaus

we show $H \to H\,N/N$ is gsurjective

lemma (**in** *Group*) *homom4Tr1*:$[\![G \triangleright N;\ G \gg H]\!] \Longrightarrow$ *Group* (($Gp\ G\ (H \diamond_G N)$) / N)
apply (*frule Gp-smult-sg-nsg*[*of H N*], *assumption*+)
apply (*frule Gp-smult-nsg* [*of H N*], *assumption*+)
apply (*simp add*:*Group.Group-Qg*)
done

lemma *homom3Tr2*:$[\![$*Group* G; $G \gg H$; $G \triangleright N]\!] \Longrightarrow$
gsurj$_{(Gp\ G\ H),((Gp\ G\ (H \diamond_G N))/N)}$
$((Pj\ (Gp\ G\ (H \diamond_G N))\ N) \circ_{(Gp\ G\ H)} (\iota_{(Gp\ G\ H)}))$
apply (*frule iotahom*[*of G H N*], *assumption*+,
frule Group.Gp-smult-nsg[*of G H N*], *assumption*+,
frule Group.smult-sg-nsg[*of G H N*], *assumption*+,
frule Group.Gp-smult-sg-nsg[*of G H N*], *assumption*+,
frule Group.Pj-gsurjec [*of Gp G* ($H \diamond_G N$) *N*], *assumption*,
frule Group.Group-Gp[*of G H*], *assumption*+,
frule Group.Group-Qg[*of Gp G* ($H \diamond_G N$) *N*], *assumption*+,
frule gHomcomp[*of Gp G H Gp G* ($H \diamond_G N$) ($Gp\ G\ (H \diamond_G N)$) / N
$\iota_{(\natural_G H)}$ $Pj\ (Gp\ G\ (H \diamond_G N))\ N$], *assumption*+)
apply (*simp add*:*gsurjec-def*)
apply (*subst gsurjec-def*, *simp*)

apply (*rule surj-to-test*,
simp add:*gHom-def*)

apply (*rule ballI*)
apply (*simp add*:*Group.Qg-carrier*[*of Gp G* ($H \diamond_G N$) *N*],
simp add:*set-rcs-def*, *erule bexE*,
frule Group.nsg-sg[*of G N*], *assumption*,
frule Group.r-sub-smult[*of G H N*], *assumption*+,
simp add:*Group.Gp-carrier*)
apply (*simp add*:*Group.Gp-rcs*[*of G N H* $\diamond_G$ *N*])

apply (*thin-tac* $\iota_{(\natural_G H)} \in gHom\ (\natural_G H)\ (\natural_G(H \diamond_G N))$,
thin-tac gsurj$_{(Gp\ G\ (H \diamond_G N)),((\natural_G(H \diamond_G N))\ /\ N)}$ $Pj\ (\natural_G(H \diamond_G N))\ N$,
thin-tac $Pj\ (\natural_G(H \diamond_G N))\ N \circ_{(\natural_G H)} \iota_{(\natural_G H)} \in gHom\ (\natural_G H)\ ((\natural_G(H \diamond_G N))\ /\ N))$
apply (*simp add*:*cmpghom-def compose-def*,
simp add:*Group.Gp-carrier iota-def*,
frule Group.smult-commute-sg-nsg[*of G H N*], *assumption*+,
frule-tac $a = a$ **in** *eq-set-inc*[*of - H* $\diamond_G$ *N N* $\diamond_G$ *H*], *assumption*+,
thin-tac $H \diamond_G N = N \diamond_G H$)
apply (*simp add*:*s-top-def*[*of G N H*], (*erule bexE*)+,
rotate-tac -1, *frule sym*, *thin-tac* $x \cdot_G y = a$,
frule-tac $h = y$ **in** *Group.sg-subset-elem*[*of G H*], *assumption*+,

simp add:*Group.rcs-fixed1*[*THEN sym*])
apply (*frule Group.l-sub-smult*[*of G H N*], *assumption*+,
frule-tac x1 = y **in** *Group.Pj-mem*[*THEN sym*, *of Gp G* (*H* $\diamond_G$ *N*) *N*],
assumption+, *simp add*:*Group.Gp-carrier*, *simp add*: *subsetD*)
apply (*frule-tac c = y* **in** *subsetD*[*of H H* $\diamond_G$ *N*], *assumption*+,
simp add:*Group.Gp-rcs*[*of G N H* $\diamond_G$ *N*], *blast*)
done

theorem *homom4*:⟦*Group G*; *G* ▷ *N*; *G* ≫ *H*⟧ ⟹*gbij*$_{((Gp\ G\ H)/(H \cap N)),((Gp\ G\ (H \diamond_G N))\ /\ N)}$
$(((Pj\ (Gp\ G\ (H \diamond_G N))\ N) \circ_{(Gp\ G\ H)} (\iota_{(Gp\ G\ H)}))^{\cdot\cdot}_{(Gp\ G\ H),((Gp\ G\ (H \diamond_G N))\ /\ N)})$

apply (*frule homom3Tr2* [*of G H N*], *assumption*+)
apply (*frule Group.Gp-smult-sg-nsg*, *assumption*+)
apply (*frule Group.homom4Tr1*[*of G N H*], *assumption*+)
apply (*frule Group.Group-Gp* [*of G H*], *assumption*+)
apply (*frule induced-ghom-ginjec* [*of Gp G H* (*Gp G* (*H* $\diamond_G$ *N*)/*N*) (*Pj* (*Gp G* (*H* $\diamond_G$ *N*)) *N*) $\circ_{(Gp\ G\ H)}$ ($\iota_{(Gp\ G\ H)}$)], *assumption*+)
apply (*simp add*:*gsurjec-def*)
apply (*frule inducedhomgsurjec* [*of Gp G H* (*Gp G* (*H* $\diamond_G$ *N*))/*N* (*Pj* (*Gp G* (*H* $\diamond_G$ *N*)) *N*) $\circ_{(Gp\ G\ H)}$ ($\iota_{(Gp\ G\ H)}$)], *assumption*+)
apply (*frule Group.homom3Tr1*[*of G H N*], *assumption*+)
apply *simp*
apply (*simp add*:*gbijec-def*)
done

lemma (**in** *Group*) *homom4-2*:⟦*G* ▷ *N*; *G* ≫ *H*⟧ ⟹ *Group* ((*Gp G H*) / (*H* ∩ *N*))
by (*frule Group-Gp*[*of H*],
frule inter-Gp-nsg[*of N H*], *assumption*,
rule Group.Group-Qg, *assumption*+)

lemma *isom4*:⟦*Group G*; *G* ▷ *N*; *G* ≫ *H*⟧ ⟹
((*Gp G H*)/(*H* ∩ *N*)) ≅ ((*Gp G* (*N* $\diamond_G$ *H*)) / *N*)
apply (*frule homom4* [*of G N H*], *assumption*+,
frule Group.smult-sg-nsg[*of G H N*], *assumption*+,
frule Group.smult-commute-sg-nsg[*of G H N*], *assumption*+)
apply (*simp add*:*isomorphic-def*, *blast*)
done

lemma *ZassenhausTr5*:⟦*Group G*; *G* ≫ *H*; *G* ≫ *H1*; *G* ≫ *K*; *G* ≫ *K1*; *Gp G H* ▷ *H1*;
Gp G K ▷ *K1*⟧ ⟹
((*Gp G* (*H* ∩ *K*))/((*H1* ∩ *K*) $\diamond_G$ (*H* ∩ *K1*))) ≅
((*Gp G* (*H1* $\diamond_G$ (*H* ∩ *K*)))/(*H1* $\diamond_G$ (*H* ∩ *K1*)))
apply (*frule Group.ZassenhausTr2-1* [*of G H H1 K*], *assumption*+,
frule Group.Group-Gp [*of G H1* $\diamond_G$ (*H* ∩ *K*)], *assumption*+,
frule Group.ZassenhausTr3 [*of G H H1 K K1*], *assumption*+,
frule Group.inter-sgs [*of G H K*], *assumption*+,

frule Group.r-sub-smult[*of G H1 H* $\cap$ *K*], *assumption*+,
frule Group.sg-sg[*of G H1* $\diamond_G$ *H* $\cap$ *K H* $\cap$ *K*], *assumption*+,
frule isom4 [*of Gp G* (*H1* $\diamond_G$ *H* $\cap$ *K*) *H1* $\diamond_G$ *H* $\cap$ *K1 H* $\cap$ *K*],
assumption+)
apply (*simp add*:*Int-commute*[*of H* $\cap$ *K H1* $\diamond_G$ *H* $\cap$ *K1*])
apply (*frule Group.Group-Gp*[*of G H*], *assumption*,
frule Group.Group-Gp[*of G K*], *assumption*,
frule Group.nsg-sg[*of Gp G H H1*], *assumption*+,
frule Group.sg-subset[*of Gp G H H1*], *assumption*+,
frule Group.nsg-sg[*of Gp G K K1*], *assumption*+,
frule Group.sg-subset[*of Gp G K K1*], *assumption*+,
simp add:*Group.Gp-carrier*,
frule Group.inter-sgs[*of G H K1*], *assumption*+,
cut-tac subset-self[*of H*],
frule Int-mono[*of H H K1 K*], *assumption*)
apply (*simp add*:*Group.s-topTr6-1*[*of G H1 H* $\cap$ *K1 H* $\cap$ *K*],
simp add:*Int-assoc*[*THEN sym*, *of H1 H K*])

apply (*simp add*:*Int-absorb2*[*of H1 H*],
simp add:*Group.Gp-inherited*[*of G H* $\cap$ *K H1* $\diamond_G$ *H* $\cap$ *K*])
apply (*frule Group.s-top-mono*[*of G H1 H* $\cap$ *K H1 H* $\cap$ *K1*],
frule Group.sg-subset[*of G H*], *assumption*+,
rule subset-trans[*of H1 H carrier G*], *assumption*+)
apply (*rule Group.sg-subset*[*of G H* $\cap$ *K*], *assumption*+, *simp*,
simp,
(*frule Group.ZassenhausTr2-1*[*of G H H1 K*], *assumption*+,
frule Group.subg-sg-sg[*of G H1* $\diamond_G$ *H* $\cap$ *K H1* $\diamond_G$ *H* $\cap$ *K1*],
assumption+, *simp add*:*Group.nsg-sg*))
apply (*simp add*:*Group.s-top-induced*[*of G H1* $\diamond_G$ *H* $\cap$ *K H1* $\diamond_G$ *H* $\cap$ *K1 H* $\cap$ *K*],
simp add:*Group.s-top-assoc*[*of G H1 H* $\cap$ *K1 H* $\cap$ *K*],
cut-tac subset-self[*of H*],
frule Int-mono[*of H H K1 K*], *assumption*)
apply (*simp add*:*Group.K-absorb-HK*[*of G H* $\cap$ *K1 H* $\cap$ *K*])
apply (*cut-tac subset-self*[*of H1* $\diamond_G$ *H* $\cap$ *K*],
simp add:*Group.Gp-inherited*[*of G H1* $\diamond_G$ *H* $\cap$ *K H1* $\diamond_G$ *H* $\cap$ *K*])
done

lemma *ZassenhausTr5-1*:⟦*Group G*; *G* $\gg$ *H*; *G* $\gg$ *H1*; *G* $\gg$ *K*; *G* $\gg$ *K1*; *Gp G H* $\triangleright$ *H1*;
Gp G K $\triangleright$ *K1*⟧ $\Longrightarrow$ ((*Gp G* (*K* $\cap$ *H*))/((*K1* $\cap$ *H*) $\diamond_G$ (*K* $\cap$ *H1*))) $\cong$
((*Gp G* (*K1* $\diamond_G$ (*K* $\cap$ *H*)))/(*K1* $\diamond_G$ (*K* $\cap$ *H1*)))

apply (*simp add*:*ZassenhausTr5* [*of G K K1 H H1*])
done

lemma *ZassenhausTr5-2*: ⟦*Group G*; *G* $\gg$ *H*; *G* $\gg$ *H1*; *G* $\gg$ *K*; *G* $\gg$ *K1*; *Gp G H* $\triangleright$ *H1*;
Gp G K $\triangleright$ *K1*⟧ $\Longrightarrow$

$((Gp\ G\ (H \cap K))/((H1 \cap K) \diamond_G (H \cap K1))) =$
$((Gp\ G\ (K \cap H))/((K1 \cap H) \diamond_G (K \cap H1)))$
by (*simp add*:*Group.ZassenhausTr3-3*[*of G H H1 K K1*],
simp add:*Int-commute*[*of H K*])

lemma *ZassenhausTr6-1*:⟦*Group G*; $G \gg H$; $G \gg H1$; $G \gg K$; $G \gg K1$; $Gp\ G\ H \triangleright H1$;
$Gp\ G\ K \triangleright K1$⟧ ⟹ *Group* $(Gp\ G\ (H \cap K) / (H1 \cap K \diamond_G H \cap K1))$
apply (*frule Group.inter-sgs* [*of G H K*], *assumption*+,
frule Group.Group-Gp [*of G H* ∩ *K*], *assumption*+,
frule Group.ZassenhausTr3-5 [*of G H H1 K K1*], *assumption*+)
apply (*rule Group.Group-Qg*, *assumption*+)
done

lemma *ZassenhausTr6-2*:⟦*Group G*; $G \gg H$; $G \gg H1$; $G \gg K$; $G \gg K1$; $Gp\ G\ H \triangleright H1$;
$Gp\ G\ K \triangleright K1$⟧ ⟹ *Group* $(Gp\ G\ (H1 \diamond_G H \cap K) / (H1 \diamond_G H \cap K1))$
apply (*frule Group.ZassenhausTr2-1* [*of G H H1 K*], *assumption*+,
frule Group.Group-Gp [*of G H1* $\diamond_G$ *H* ∩ *K*], *assumption*+,
frule Group.ZassenhausTr3 [*of G H H1 K K1*], *assumption*+)
apply (*simp add*:*Group.Group-Qg*)
done

lemma *ZassenhausTr6-3*:⟦*Group G*; $G \gg H$; $G \gg H1$; $G \gg K$; $G \gg K1$; $Gp\ G\ H \triangleright H1$;
$Gp\ G\ K \triangleright K1$⟧ ⟹ *Group* $(Gp\ G\ (K1 \diamond_G K \cap H) / (K1 \diamond_G K \cap H1))$
apply (*frule Group.ZassenhausTr2-1* [*of G K K1 H*], *assumption*+,
frule Group.Group-Gp [*of G K1* $\diamond_G$ *K* ∩ *H*], *assumption*+,
frule Group.ZassenhausTr3[*of G K K1 H H1*], *assumption*+)
apply (*simp add*:*Group.Group-Qg*)
done

theorem *Zassenhaus*:⟦*Group G*; $G \gg H$; $G \gg H1$; $G \gg K$; $G \gg K1$; $Gp\ G\ H \triangleright H1$;
$Gp\ G\ K \triangleright K1$⟧ ⟹ $(Gp\ G\ (H1 \diamond_G H \cap K) / (H1 \diamond_G H \cap K1)) \cong$
$(Gp\ G\ (K1 \diamond_G K \cap H) / (K1 \diamond_G K \cap H1))$
apply (*frule ZassenhausTr6-1*[*of G K K1 H H1*], *assumption*+)
apply (*frule ZassenhausTr6-3* [*of G H H1 K K1*], *assumption*+)
apply (*frule ZassenhausTr6-2* [*of G H H1 K K1*], *assumption*+)
apply (*rule isomTr2*[*of* $(\natural_G(H1 \diamond_G H \cap K)) / (H1 \diamond_G H \cap K1)$
$(\natural_G(K \cap H)) / (K1 \cap H \diamond_G K \cap H1)$
$(\natural_G(K1 \diamond_G K \cap H)) / (K1 \diamond_G K \cap H1)$], *assumption*+)
apply (*frule ZassenhausTr5-1*[*of G K K1 H H1*], *assumption*+)
apply (*simp add*:*Int-commute*[*of K H*])
apply (*simp add*:*Group.ZassenhausTr3-3*[*THEN sym*, *of G H H1 K K1*])
apply (*rule isomTr1*[*of* $(\natural_G(H \cap K)) / (H1 \cap K \diamond_G H \cap K1)$
$(\natural_G(H1 \diamond_G H \cap K)) / (H1 \diamond_G H \cap K1)$], *assumption*+)
apply (*rule ZassenhausTr5-1*[*of G H H1 K K1*], *assumption*+)
done

3.13 Chain of groups I

definition
d-gchain :: [- , *nat*, (*nat* ⇒ *'a set*)] ⇒ *bool* **where**
d-gchain G n g = (*if n=0 then G* ≫ *g 0 else* (∀ *l*≤ *n*. *G* ≫ (*g l*) ∧
(∀ *l* ≤ (*n* − *Suc 0*). *g* (*Suc l*) ⊆ *g l*)))

definition
D-gchain :: [- , *nat*, (*nat* ⇒ *'a set*)] ⇒ *bool* **where**
D-gchain G n g = (*if n* = *0 then G* ≫ (*g 0*) *else* (*d-gchain G n g*) ∧
(∀ *l* ≤ (*n* − *Suc 0*). (*g* (*Suc l*)) ⊂ (*g l*)))

definition
td-gchain :: [- , *nat*, (*nat* ⇒ *'a set*)] ⇒ *bool* **where**
td-gchain G n g = (*if n=0 then g 0* = *carrier G* ∧ *g 0* = {**1**$_G$} *else*
d-gchain G n g ∧ *g 0* = *carrier G* ∧ *g n* = {**1**$_G$})

definition
tD-gchain :: [-, *nat*, (*nat* ⇒ *'a set*)] ⇒ *bool* **where**
tD-gchain G n g = (*if n=0 then g 0* = *carrier G* ∧ *g 0* = {**1**$_G$} *else*
D-gchain G n g ∧ (*g 0* = *carrier G*) ∧ (*g n* = {**1**$_G$}))

definition
w-cmpser :: [- , *nat*, (*nat* ⇒ *'a set*)] ⇒ *bool* **where**
w-cmpser G n g = (*if n* = *0 then d-gchain G n g else d-gchain G n g* ∧
(∀ *l* ≤ (*n* − *1*). (*Gp G* (*g l*)) ▷ (*g* (*Suc l*))))

definition
W-cmpser :: [- , *nat*, (*nat* ⇒ *'a set*)] ⇒ *bool* **where**
W-cmpser G n g = (*if n* = *0 then d-gchain G 0 g else D-gchain G n g* ∧
(∀ *l* ≤ (*n* − *1*). (*Gp G* (*g l*)) ▷ (*g* (*Suc l*))))

definition
tw-cmpser :: [- , *nat*, (*nat* ⇒ *'a set*)] ⇒ *bool* **where**
tw-cmpser G n g = (*if n* = *0 then td-gchain G 0 g else td-gchain G n g* ∧
(∀ *l* ≤ (*n* − *1*). (*Gp G* (*g l*)) ▷ (*g* (*Suc l*))))

definition
tW-cmpser :: [- , *nat*, (*nat* ⇒ *'a set*)] ⇒ *bool* **where**
tW-cmpser G n g = (*if n* = *0 then td-gchain G 0 g else tD-gchain G n g* ∧
(∀ *l* ≤ (*n* − *1*). (*Gp G* (*g l*)) ▷ (*g* (*Suc l*))))

definition
Qw-cmpser :: [- , *nat* ⇒ *'a set*] ⇒ (*nat* ⇒ (*'a set*) *Group*) **where**
Qw-cmpser G f l = ((*Gp G* (*f l*)) / (*f* (*Suc l*)))

definition
red-chn :: [- , *nat*, (*nat* ⇒ *'a set*)] ⇒ (*nat* ⇒ *'a set*) **where**
red-chn G n f = (*SOME g*. *g* ∈ {*h*.(*tW-cmpser G* (*card* (*f* ‘ {*i*. *i* ≤ *n*}) − *1*) *h*)
∧ *h* ‘{*i*. *i* ≤ (*card* (*f* ‘ {*i*. *i* ≤ *n*}) − *1*)} = *f* ‘{*i*. *i* ≤ *n*}})

definition
chain-cutout :: [*nat*, (*nat* ⇒ *'a set*)] ⇒ (*nat* ⇒ *'a set*) **where**
chain-cutout l f = (λ*j*. *f* (*slide l j*))

lemma (**in** *Group*) *d-gchainTr0*:⟦*0* < *n*; *d-gchain G n f*; *k* ≤ (*n* − *1*)⟧
⟹ *f* (*Suc k*) ⊆ *f k*
apply (*simp add*:*d-gchain-def*)
apply (*frule-tac a* = *k* **in** *forall-spec*)
apply (*rule Nat.le-trans*, *assumption*+, *simp*)
apply (*erule conjE*, *rotate-tac 2*,
frule-tac a = *k* **in** *forall-spec*, *assumption*,
thin-tac ∀ *l*≤*n* − *Suc 0*. *f* (*Suc l*) ⊆ *f l*,
thin-tac ∀ *l*≤*n*. *G* ≫ *f l* ∧ (∀ *l*≤*n* − *Suc 0*. *f* (*Suc l*) ⊆ *f l*))
apply *assumption*
done

lemma (**in** *Group*) *d-gchain-mem-sg*:*d-gchain G n f* ⟹ ∀ *i*≤ *n*. *G* ≫ (*f i*)
apply (*rule allI*)
apply (*rule impI*, *simp add*:*d-gchain-def*)
apply (*case-tac n* = *0*, *simp*)
apply *simp*
done

lemma (**in** *Group*) *d-gchain-pre*:*d-gchain G* (*Suc n*) *f* ⟹ *d-gchain G n f*
apply (*simp add*:*d-gchain-def*, *rule impI*, *rule impI*)
apply (*rule allI*, *rule impI*)
apply (*frule-tac a* = *l* **in** *forall-spec*, *arith*)
apply (*erule conjE*)
apply (*thin-tac* ∀ *l*≤*Suc n*. *G* ≫ *f l* ∧ (∀ *l*≤*n*. *f* (*Suc l*) ⊆ *f l*),
frule-tac a = *l* **in** *forall-spec*, *arith*, *assumption*)
done

lemma (**in** *Group*) *d-gchainTr1*:*0* < *n* ⟶ (∀ *f*. *d-gchain G n f* ⟶
(∀ *l* ≤ *n*. ∀ *j* ≤ *n*. *l* < *j* ⟶ *f j* ⊆ *f l*))
apply (*induct-tac n*)
apply (*rule impI*, *simp*)

apply (*rule impI*, *rule allI*, *rule impI*)
apply ((*rule allI*, *rule impI*)+, *rule impI*)

apply (*case-tac n = 0*, *simp*)
apply (*case-tac j = 0*, *simp*,
frule le-imp-less-or-eq, *thin-tac j ≤ Suc 0*,
simp, *simp add*:*d-gchain-def*)
apply (*frule-tac a = 0* **in** *forall-spec*, *simp*, *simp*)

apply *simp*

apply (*case-tac j = Suc n*)
apply (*frule d-gchain-pre*,
frule-tac a = f **in** *forall-spec*, *assumption*,
thin-tac ∀*f*. *d-gchain G n f* ⟶ (∀*l*≤*n*. ∀*j*≤*n*. *l* < *j* ⟶ *f j* ⊆ *f l*),
thin-tac d-gchain G n f,
simp add:*d-gchain-def*)
apply (*frule-tac a = n* **in** *forall-spec*, *simp*,
thin-tac ∀*l*≤*Suc n*. *G* ≫ *f l* ∧ (∀*l*≤*n*. *f* (*Suc l*) ⊆ *f l*),
erule conjE,
rotate-tac −*1*,
frule-tac a = n **in** *forall-spec*, *simp*,
thin-tac ∀*l*≤*n*. *f* (*Suc l*) ⊆ *f l*,
frule-tac x = l **and** *n = n* **in** *Suc-less-le*)
apply (*case-tac l = n*, *simp*,
thin-tac l < *Suc n*,
frule-tac x = l **and** *y = n* **in** *le-imp-less-or-eq*,
thin-tac l ≤ *n*, *simp*)
apply (*frule-tac a = l* **in** *forall-spec*, *simp*,
thin-tac ∀*l*≤*n*. ∀*j*≤*n*. *l* < *j* ⟶ *f j* ⊆ *f l*) **apply** (
frule-tac a = n **in** *forall-spec*) **apply** (*simp*,
thin-tac ∀*j*≤*n*. *l* < *j* ⟶ *f j* ⊆ *f l*, *simp*)
apply (*metis less-le-trans d-gchain-pre less-Suc-eq-le linorder-antisym-conv1 mod-if mod-le-divisor*)
done

lemma (**in** *Group*) *d-gchainTr2*:⟦*0* < *n*; *d-gchain G n f*; *l* ≤ *n*; *j* ≤ *n*; *l* ≤ *j* ⟧
⟹ *f j* ⊆ *f l*
apply (*case-tac l = j*, *simp*)
apply (*metis Group.d-gchainTr1* [*OF Group-axioms*] *linorder-antisym-conv2*)
done

lemma (**in** *Group*) *im-d-gchainTr1*:⟦*d-gchain G n f*;
f l ∈ (*f* ‘ {*i*. *i* ≤ *n*}) − {*f 0*}⟧ ⟹
f (*LEAST j*. *f j* ∈ (*f* ‘ {*i*. *i* ≤ *n*}) − {*f 0*}) ∈ (*f* ‘ {*i*. *i* ≤ *n*} − {*f 0*})
apply (*rule LeastI*)
apply *simp*
done

lemma (**in** *Group*) *im-d-gchainTr1-0*:⟦*d-gchain G n f*;
f l ∈ (*f* ‘ {*i*. *i* ≤ *n*}) − {*f 0*}⟧ ⟹
0 < (*LEAST j*. *f j* ∈ (*f* ‘ {*i*. *i* ≤ *n*}) − {*f 0*})
apply (*frule im-d-gchainTr1* [*of n f*], *assumption*+)
apply (*rule contrapos-pp*, *simp*)
apply *simp*
done

lemma (**in** *Group*) *im-d-gchainTr1-1*:
⟦*d-gchain G n f*; ∃ *i*. *f i* ∈ (*f* ‘ {*i*. *i* ≤ *n*}) − {*f 0*}⟧ ⟹
f (*LEAST j*. *f j* ∈ ((*f* ‘ {*i*. *i* ≤ *n*}) − {*f 0*})) ∈ ((*f*‘ {*i*. *i* ≤ *n*}) − {*f 0*})
apply (*subgoal-tac* ∀ *i*. *f i* ∈ *f* ‘ {*i*. *i* ≤ *n*} − {*f 0*} ⟶
f (*LEAST j*. *f j* ∈ *f* ‘{*i*. *i* ≤ *n*} − {*f 0*}) ∈ *f* ‘ {*i*. *i*≤ *n*} − {*f 0*})
apply *blast*
apply (*thin-tac* ∃ *i*. *f i* ∈ *f* ‘{*i*. *i* ≤ *n*} − {*f 0*})
apply (*rule allI*) **apply** (*rule impI*)
apply (*rule im-d-gchainTr1* [*of n f* -], *assumption*+)
done

lemma (**in** *Group*) *im-d-gchainsTr1-2*:
⟦*d-gchain G n f*; *i* ≤ *n*; *f i* ∈ *f* ‘{*i*. *i* ≤ *n*} − {*f 0*}⟧ ⟹
(*LEAST j*. *f j* ∈ (*f* ‘{*i*. *i* ≤ *n*} − {*f 0*})) ≤ *i*
by (*rule Least-le*, *assumption*)

lemma (**in** *Group*) *im-d-gchainsTr1-3*:⟦*d-gchain G n f*; ∃ *i* ≤ *n*.
f i ∈ *f*‘ {*i*. *i* ≤ *n*} − {*f 0*};
k < (*LEAST j*. *f j* ∈ (*f* ‘{*i*. *i* ≤ *n*} − {*f 0*}))⟧ ⟹ *f k* = *f 0*
apply (*erule exE*)
apply (*rule contrapos-pp*, *simp*+)
apply (*frule-tac i* = *i* **in** *im-d-gchainsTr1-2* [*of n f* -], *simp*+)
apply (*erule conjE*)+
apply (*frule-tac x* = *k* **and**
y = *LEAST j*. *f j* ∈ *f* ‘ {*i*. *i* ≤ *n*} ∧ *f j* ≠ *f 0* **and**
z = *i* **in** *less-le-trans*, *assumption*,
frule-tac x = *k* **and**
y = *i* **and**
z = *n* **in** *less-le-trans*, *assumption*)
apply (*frule-tac i* = *k* **in** *im-d-gchainsTr1-2* [*of n f* -], *simp*+)
done

lemma (**in** *Group*) *im-gdchainsTr1-4*:⟦*d-gchain G n f*;
∃ *v*∈*f* ‘{*i*. *i* ≤ *n*}. *v* ∉ {*f 0*}; *i* < (*LEAST j*. *f j* ∈ (*f* ‘{*i*. *i* ≤ *n*}) ∧
f j ≠ *f 0*) ⟧ ⟹ *f i* = *f 0*
apply (*rule im-d-gchainsTr1-3* [*of n f i*], *assumption*,
thin-tac i < (*LEAST j*. *f j* ∈ *f* ‘ {*i*. *i* ≤ *n*} ∧ *f j* ≠ *f 0*),
simp add:*image-def*, *blast*)
apply *simp*
done

lemma (**in** *Group*) *im-d-gchainsTr1-5*:⟦*0* < *n*; *d-gchain G n f*; *i* ≤ *n*;
f i ∈ (*f* ‘ {*i*. *i* ≤ *n*} − {*f 0*}); (*LEAST j*. *f j* ∈ (*f* ‘{*i*. *i* ≤ *n*} − {*f 0*})) = *j*⟧
⟹ *f* ‘{*i*. *i* ≤ (*j* − (*Suc 0*))} = {*f 0*}
apply (*frule im-d-gchainTr1-0* [*of n f i*], *assumption*+)
apply (*subst image-def*)
apply (*rule equalityI*)
apply (*rule subsetI*, *simp*, *erule exE*, *erule conjE*)
apply (*frule-tac x* = *xa* **and** *y* = *j* − *Suc 0* **and**
z = (*LEAST j*. *f j* ∈ *f* ‘ {*i*. *i* ≤ *n*} ∧ *f j* ≠ *f 0*) **in** *le-less-trans*,
simp,
frule-tac k = *xa* **in** *im-d-gchainsTr1-3*[*of n f*])
apply (*simp*, *blast*, *simp*, *simp*)
apply (*rule subsetI*, *blast*)
done

lemma (**in** *Group*) *im-d-gchains1*:⟦*0* < *n*; *d-gchain G n f*; *i* ≤ *n*;
f i ∈ (*f* ‘ {*i*. *i* ≤ *n*} − {*f 0*});
(*LEAST j*. *f j* ∈ (*f* ‘{*i*. *i* ≤ *n*} − {*f 0*})) = *j* ⟧ ⟹
f ‘{*i*. *i* ≤ *n*} = {*f 0*} ∪ {*f i* |*i*. *j* ≤ *i* ∧ *i* ≤ *n*}
apply (*frule im-d-gchainTr1-0* [*of n f i*], *assumption*+,
frule im-d-gchainsTr1-2 [*of n f i*], *assumption*+,
frule Nset-nset-1 [*of n j* − *Suc 0*])
apply *simp*
apply (*subst im-set-un2*, *simp*)
apply (*subst im-d-gchainsTr1-5*[*of n f i j*])
apply (*simp*, *assumption*, *simp*+)

apply (*rule equalityI*)
apply (*rule subsetI*, *simp*, *erule disjE*, *simp*,
simp add:*image-def nset-def*, *erule exE*, *blast*)
apply (*rule subsetI*)
apply (*simp*, *erule disjE*, *simp*)
apply (*erule exE*, *simp add*:*nset-def*)
done

lemma (**in** *Group*) *im-d-gchains1-1*:⟦*d-gchain G n f*; *f n* ≠ *f 0*⟧ ⟹
f ‘{*i*. *i* ≤ *n*} = {*f 0*} ∪
{*f i* |*i*. (*LEAST j*. *f j* ∈ (*f* ‘{*i*. *i* ≤ *n*} − {*f 0*})) ≤ *i* ∧ *i* ≤ *n*}
apply (*case-tac n* = *0*) **apply** *simp*
apply (*simp*)
apply (*frule im-d-gchains1* [*of n f n*
(*LEAST j*. *f j* ∈ *f* ‘ {*i*. *i* ≤ *n*} − {*f 0*})], *assumption*+,
simp add:*n-in-Nsetn*)
apply (*cut-tac n-in-Nsetn*[*of n*], *simp*,
simp)
apply *simp*
done

lemma (**in** *Group*) *d-gchains-leastTr*:⟦*d-gchain G n f*; *f n* ≠ *f 0*⟧ ⟹
(*LEAST j*. *f j* ∈ (*f* ‘{*i*. *i* ≤ *n*} − {*f 0*})) ∈ {*i*. *i* ≤ *n*} ∧
f (*LEAST j*. *f j* ∈ (*f* ‘{*i*. *i* ≤ *n*} − {*f 0*})) ≠ *f 0*
apply (*frule im-d-gchainsTr1-2* [*of n f n*],
simp add:*n-in-Nsetn*,
simp add:*image-def*, *blast*,
frule mem-of-Nset[*of LEAST j*. *f j* ∈ *f* ‘ {*i*. *i* ≤ *n*} − {*f 0*} *n*],
simp)
apply (*frule im-d-gchainTr1*[*of n f n*],
simp add:*image-def*, *cut-tac n-in-Nsetn*[*of n*], *blast*)
apply (*simp add*:*image-def*)
done

lemma (**in** *Group*) *im-d-gchainTr2*:⟦*d-gchain G n f*; *j* ≤ *n*; *f j* ≠ *f 0*⟧ ⟹
∀ *i* ≤ *n*. *f 0* = *f i* ⟶ ¬ *j* ≤ *i*
apply (*case-tac n* = *0*, *simp*, *simp*)
apply (*rule allI*, *rule impI*)
apply (*rule contrapos-pp*, *simp*+)
apply (*case-tac j* = *i*, *simp*)
apply (*frule d-gchainTr2* [*of n f 0 j*], *assumption*+,
simp, (*erule conjE*)+,
rule-tac i = *j* **and** *j* = *i* **and** *k* = *n* **in** *le-trans*, *assumption*+,
simp,
(*erule conjE*)+,
frule-tac l = *j* **and** *j* = *i* **in** *d-gchainTr2* [*of n f*], *assumption*+)

apply *simp*+
done

lemma (**in** *Group*) *D-gchain-pre*:⟦*D-gchain G* (*Suc n*) *f*⟧ ⟹ *D-gchain G n f*
apply (*simp add*:*D-gchain-def*, *erule conjE*)
apply (*case-tac n* = *0*, *rotate-tac* −*1*)
apply (*simp*, *insert lessI* [*of 0*::*nat*], *simp*)
apply (*simp add*:*d-gchain-def*, *simp*)
apply (*frule d-gchain-pre* [*of n*])
apply *simp*
done

lemma (**in** *Group*) *D-gchain0*:⟦*D-gchain G n f*; *i* ≤ *n*; *j* ≤ *n*; *i* < *j*⟧ ⟹
f j ⊂ *f i*
apply (*case-tac n* = *0*)
apply (*simp*, *simp*)
apply (*cut-tac d-gchainTr1*[*of n*], *simp*)
apply (*simp add*:*D-gchain-def*, *frule conjunct1*)
apply (*frule-tac a* = *f* **in** *forall-spec*, *assumption*,
thin-tac ∀ *f*. *d-gchain G n f* ⟶ (∀ *l*≤*n*. ∀ *j*≤*n*. *l* < *j* ⟶ *f j* ⊆ *f l*))
apply (*frule-tac a* = *i* **in** *forall-spec*,
frule-tac x = *i* **and** *y* = *j* **and** *z* = *n* **in** *less-le-trans*, *assumption*+,
simp)

apply (*thin-tac* $\forall l{\leq}n.\ \forall j{\leq}n.\ l < j \longrightarrow f\ j \subseteq f\ l$,
frule-tac $a = j$ **in** *forall-spec*, *assumption*,
thin-tac $\forall j{\leq}n.\ i < j \longrightarrow f\ j \subseteq f\ i$, *simp*)
apply (*frule Suc-leI*[*of i j*],
frule less-le-trans[*of i j n*], *assumption*+,
frule less-le-diff[*of i n*])
apply (*frule-tac* $a = i$ **in** *forall-spec*, *assumption*,
thin-tac $\forall l{\leq}n - Suc\ 0.\ f\ (Suc\ l) \subset f\ l$)
apply (*cut-tac d-gchainTr2*[*of n f Suc i j*])
apply *blast* **apply** *simp*+
done

lemma (**in** *Group*) *D-gchain1*:*D-gchain G n f* $\Longrightarrow$ *inj-on f* $\{i.\ i \leq n\}$
apply (*case-tac* $n = 0$)
apply (*simp add*:*inj-on-def*)
apply (*simp*)
apply (*simp add*:*inj-on-def*)
apply ((*rule allI*)+, *rule impI*, *rule contrapos-pp*, *simp*+, *erule exE*)
apply (*cut-tac* $x = x$ **and** $y = y$ **in** *less-linear*, *simp*)
apply (*erule disjE*, (*erule conjE*)+)
apply (*frule-tac* $i = x$ **and** $j = y$ **in** *D-gchain0* [*of n f*], *assumption*+,
simp,
simp, *frule-tac* $i = y$ **and** $j = x$ **in** *D-gchain0* [*of n f*],
simp+)
done

lemma (**in** *Group*) *card-im-D-gchain*:$[\![0 < n;\ D\text{-}gchain\ G\ n\ f]\!]$
$\Longrightarrow$ *card* $(f\ `\{i.\ i \leq n\}) = Suc\ n$
apply (*frule D-gchain1* [*of n*])
apply (*subst card-image*, *assumption*+, *simp*)
done

lemma (**in** *Group*) *w-cmpser-gr*:$[\![0 < r;\ w\text{-}cmpser\ G\ r\ f;\ i \leq r]\!]$
$\Longrightarrow G \gg (f\ i)$
by (*simp add*:*w-cmpser-def*, *erule conjE*, *simp add*:*d-gchain-def*)

lemma (**in** *Group*) *w-cmpser-ns*:$[\![0 < r;\ w\text{-}cmpser\ G\ r\ f;\ i \leq (r - 1)]\!] \Longrightarrow$
$(Gp\ G\ (f\ i)) \triangleright (f\ (Suc\ i))$
apply (*simp add*:*w-cmpser-def*)
done

lemma (**in** *Group*) *w-cmpser-pre*:*w-cmpser G* $(Suc\ n)$ *f* $\Longrightarrow$ *w-cmpser G n f*
apply (*simp add*:*w-cmpser-def*)
apply (*erule conjE*)
apply (*case-tac* $n = 0$, *rotate-tac* -1, *simp*)
apply (*rule d-gchain-pre* [*of 0 f*], *assumption*+)
apply *simp*
apply (*frule d-gchain-pre* [*of n f*])
apply *simp*

done

lemma (**in** *Group*) *W-cmpser-pre*:*W-cmpser G* (*Suc n*) *f* ⟹ *W-cmpser G n f*
apply (*simp add*:*W-cmpser-def*)
apply (*erule conjE*)
apply (*case-tac n* = *0*, *simp*,
simp add:*D-gchain-def*, *erule conjE*,
rule d-gchain-pre, *assumption*+, *simp*)

apply (*frule D-gchain-pre*, *simp*)
done

lemma (**in** *Group*) *td-gchain-n*:⟦*td-gchain G n f*; *carrier G* ≠ {**1**}⟧ ⟹ *0* < *n*
apply (*simp add*:*td-gchain-def*)
apply (*rule contrapos-pp*, *simp*+)
apply (*erule conjE*, *simp*)
done

3.14 Existence of reduced chain

lemma (**in** *Group*) *D-gchain-is-d-gchain*:*D-gchain G n f* ⟹ *d-gchain G n f*
apply (*simp add*:*D-gchain-def*)
apply (*case-tac n* = *0*) **apply** (*rotate-tac* −*1*)
apply (*simp add*:*d-gchain-def*) **apply** (*rotate-tac* −*1*)
apply *simp*
done

lemma (**in** *Group*) *joint-d-gchains*:⟦*d-gchain G n f*; *d-gchain G m g*;
g 0 ⊆ *f n* ⟧ ⟹ *d-gchain G* (*Suc* (*n* + *m*)) (*jointfun n f m g*)
apply (*case-tac n* = *0*)
apply (*case-tac m* = *0*)
apply (*simp add*:*d-gchain-def* [*of G Suc* (*n* + *m*) -])
apply (*simp add*:*jointfun-def sliden-def d-gchain-def*)
apply (*simp add*:*jointfun-def sliden-def d-gchain-def*)
apply (*rule allI*) **apply** (*rule conjI*) **apply** (*rule impI*)
apply (*rule allI*) **apply** (*rule impI*)+ **apply** *simp*
apply (*frule-tac a* = *la* **in** *forall-spec*, *assumption*,
thin-tac ∀ *l*≤*m*. *G* ↠ *g l* ∧ (∀ *l*≤*m* − *Suc 0*. *g* (*Suc l*) ⊆ *g l*),
erule conjE)
apply (*frule-tac a* = *la* − *Suc 0* **in** *forall-spec*,
thin-tac ∀ *l*≤*m* − *Suc 0*. *g* (*Suc l*) ⊆ *g l*,
rule diff-le-mono, *assumption*, *simp*)
apply (*rule impI*, *rule impI*)
apply (*frule-tac m* = *l* **and** *n* = *Suc m* **and** *l* = *Suc 0* **in** *diff-le-mono*)
apply *simp*
apply (*rule allI*, *rule impI*, *rule impI*)
apply (*frule-tac a* = *la* **in** *forall-spec*,*assumption*,
thin-tac ∀ *l*≤*m*. *G* ↠ *g l* ∧ (∀ *l*≤*m* − *Suc 0*. *g* (*Suc l*) ⊆ *g l*),
erule conjE)

```
      apply (frule-tac a = la − Suc 0 in forall-spec, simp)
apply simp-all
 apply (simp add:d-gchain-def [of G - jointfun n f m g])
 apply (rule allI, rule impI) apply (rule conjI)
 apply (case-tac l ≤ n, simp add:jointfun-def d-gchain-def[of - n f])
 apply (simp add:jointfun-def sliden-def,
       frule-tac m = l and n = Suc (n + m) and l = Suc n in diff-le-mono,
       thin-tac l ≤ Suc (n + m), simp add:d-gchain-def[of - m g])
 apply (case-tac m = 0, simp, simp)

apply (rule allI, rule impI)
 apply (case-tac Suc la ≤ n)
 apply (simp add:jointfun-def)
 apply (rule-tac l = la and j = Suc la in d-gchainTr2[of n f],
       simp+)
 apply (simp add:jointfun-def)
 apply (cut-tac y = Suc la and x = n in not-less [symmetric], simp)
 apply (frule-tac m = n and n = Suc la in Suc-leI,
       thin-tac n < Suc la, simp)
 apply (case-tac la = n, simp add:sliden-def)
apply (frule not-sym, thin-tac la ≠ n,
      frule-tac x = n and y = la in le-imp-less-or-eq,
      thin-tac n ≤ la, simp,
      frule-tac m = n and n = la in Suc-leI, simp add:sliden-def)
 apply (simp add:d-gchain-def[of - m g])
 apply (cut-tac m = la and n = n + m and l = Suc n in diff-le-mono,
        assumption, simp)
 apply (frule-tac a = m in forall-spec, simp,
       thin-tac ∀ l≤m. G ⋙ g l ∧ (∀ l≤m − Suc 0. g (Suc l) ⊆ g l),
       erule conjE) apply (
       frule-tac a = la − Suc n in forall-spec, assumption,
       thin-tac ∀ l≤m − Suc 0. g (Suc l) ⊆ g l)
 apply (cut-tac n1 = n and i1 = la  in jointgd-tool4[THEN sym], simp+)
done

lemma (in Group) joint-D-gchains:⟦D-gchain G n f; D-gchain G m g;
      g 0 ⊂ f n ⟧ ⟹  D-gchain G (Suc (n + m)) (jointfun n f m g)
apply (simp add:D-gchain-def [of G Suc (n + m) -])
apply (rule conjI)
apply (rule joint-d-gchains[of n f m g],
      simp add:D-gchain-is-d-gchain,
      simp add:D-gchain-is-d-gchain,
      simp add:psubset-imp-subset)
apply (rule allI, rule impI)
 apply (case-tac Suc l ≤ n)
 apply (simp add:jointfun-def)
 apply (rule-tac i = l and j = Suc l in D-gchain0[of n f], assumption,
       cut-tac x = l and y = Suc l and z = n in less-le-trans)
       apply simp+
```

apply (*simp add*:*nat-not-le-less*,
frule-tac $m = n$ **and** $n = Suc\ l$ **in** *Suc-leI*, *thin-tac* $n < Suc\ l$, *simp*)
apply (*case-tac* $l = n$, *simp add*:*jointfun-def sliden-def*)
apply (*frule not-sym*, *thin-tac* $l \neq n$,
frule-tac $x = n$ **and** $y = l$ **in** *le-imp-less-or-eq*,
thin-tac $n \leq l$, *simp*)
apply (*simp add*:*jointfun-def sliden-def*)
apply (*frule-tac* $m = l$ **and** $n = n + m$ **and** $l = n$ **in** *diff-le-mono*)
apply (*simp add*:*diff-add-assoc*)
apply (*rule-tac* $i = l - Suc\ n$ **and** $j = l - n$ **in** *D-gchain0*[*of m g*],
assumption)
apply (*arith*, *assumption*, *arith*)
done

lemma (**in** *Group*) *w-cmpser-is-d-gchain*:*w-cmpser G n f* $\Longrightarrow$ *d-gchain G n f*
apply (*simp add*:*w-cmpser-def*)
apply (*case-tac n=0*) **apply** (*rotate-tac −1*) **apply** *simp*
apply (*rotate-tac −1*) **apply** *simp*
done

lemma (**in** *Group*) *joint-w-cmpser*:⟦*w-cmpser G n f*; *w-cmpser G m g*;
Gp G (*f n*) ▷ (*g 0*)⟧ $\Longrightarrow$ *w-cmpser G* (*Suc* (*n* + *m*)) (*jointfun n f m g*)
apply (*simp add*:*w-cmpser-def* [*of* - *Suc* (*n* + *m*) -])
apply (*rule conjI*)
apply (*frule w-cmpser-is-d-gchain*[*of n f*],
frule w-cmpser-is-d-gchain[*of m g*])
apply (*rule joint-d-gchains*, *assumption*+)
apply (*frule d-gchain-mem-sg*[*of n f*],
cut-tac n-in-Nsetn[*of n*],
frule-tac $a = n$ **in** *forall-spec*, *simp*,
thin-tac $\forall\, i \leq n.\ G \gg f\ i$)
apply (*frule Group-Gp*[*of f n*],
frule Group.nsg-sg[*of Gp G* (*f n*) *g 0*], *assumption*,
frule Group.sg-subset[*of Gp G* (*f n*) *g 0*], *assumption*,
simp add:*Gp-carrier*)
apply (*rule allI*, *rule impI*)
apply (*case-tac* $n = 0$) **apply** *simp*
apply (*simp add*:*jointfun-def*)
apply (*case-tac* $l = 0$)
apply *simp* **apply** (*simp add*:*sliden-def*)
apply *simp*
apply (*simp add*:*w-cmpser-def* [*of* - *m g*])
apply (*case-tac* $m = 0$) **apply** (*simp add*:*sliden-def*)
apply (*erule conjE*)
apply (*simp add*:*sliden-def*)
apply (*frule-tac* $x = 0$ **and** $y = l$ **and** $z = m$ **in** *less-le-trans*, *assumption*+)
apply (*frule-tac* $m = l$ **and** $n = m$ **and** $l = Suc\ 0$ **in** *diff-le-mono*)
apply (*frule-tac* $a = l - Suc\ 0$ **in** *forall-spec*, *assumption*,
thin-tac $\forall\, l{\leq}(m - Suc\ 0).\ (\natural(g\ l)) \triangleright (g\ (Suc\ l))$)

apply *simp*
apply (*case-tac* $l \leq n - Suc\ 0$, *simp*)
apply (*frule less-pre-n* [*of n*])
apply (*frule-tac* $x = l$ **in** *le-less-trans*[*of - n* $-$ *Suc 0 n*], *assumption*+)
apply (*simp add*:*jointfun-def w-cmpser-def* [*of - n*])
apply (*simp add*:*nat-not-le-less*)
apply (*frule-tac* $n = l$ **in** *Suc-leI* [*of n* $-$ *Suc 0 -*], *simp*)
apply (*case-tac* $n = l$)
apply (*frule sym*) **apply** (*thin-tac* $n = l$)
apply *simp*
apply (*simp add*:*jointfun-def sliden-def*)
apply (*frule-tac* $m = n$ **and** $n = l$ **in** *noteq-le-less*, *assumption*+)
apply (*frule-tac* $m = n$ **and** $n = l$ **in** *Suc-leI*)
apply (*simp add*:*jointfun-def*)
apply (*frule-tac* $m = l$ **and** $n = n + m$ **and** $l = Suc\ n$ **in** *diff-le-mono*)
apply *simp*
apply (*simp add*:*sliden-def w-cmpser-def* [*of - m -*])
apply (*erule conjE*)
apply (*simp add*:*jointgd-tool4*)
done

lemma (**in** *Group*) *W-cmpser-is-D-gchain*:*W-cmpser G n f* $\Longrightarrow$ *D-gchain G n f*
apply (*simp add*:*W-cmpser-def*)
apply (*case-tac* $n = 0$) **apply** (*rotate-tac* -1) **apply** *simp*
apply (*simp add*:*D-gchain-def d-gchain-def*)
apply (*rotate-tac* -1) **apply** *simp*
done

lemma (**in** *Group*) *W-cmpser-is-w-cmpser*:*W-cmpser G n f* $\Longrightarrow$ *w-cmpser G n f*
apply (*simp add*:*W-cmpser-def*)
apply (*case-tac* $n = 0$) **apply** (*rotate-tac* -1)
apply *simp*
apply (*simp add*:*w-cmpser-def*)
apply (*rotate-tac* -1)
apply *simp* **apply** (*erule conjE*)
apply (*frule D-gchain-is-d-gchain*)
apply (*simp add*:*w-cmpser-def*)
done

lemma (**in** *Group*) *tw-cmpser-is-w-cmpser*:*tw-cmpser G n f* $\Longrightarrow$ *w-cmpser G n f*
apply (*simp add*:*tw-cmpser-def*)
apply (*case-tac* $n = 0$)
apply (*rotate-tac* -1) **apply** *simp*
apply (*simp add*:*td-gchain-def w-cmpser-def*)
apply (*simp add*:*d-gchain-def*) **apply** (*simp add*:*special-sg-G*)
apply (*rotate-tac* -1) **apply** *simp*
apply (*erule conjE*) **apply** (*simp add*:*td-gchain-def*)
apply (*erule conjE*)+
apply (*simp add*:*w-cmpser-def*)

done

lemma (**in** *Group*) *tW-cmpser-is-W-cmpser*:*tW-cmpser G n f* $\Longrightarrow$ *W-cmpser G n f*
apply (*simp add*:*tW-cmpser-def*)
apply (*case-tac n = 0*) **apply** (*rotate-tac −1*)
apply *simp*
apply (*simp add*:*td-gchain-def*)
apply (*simp add*: *W-cmpser-def d-gchain-def*) **apply** (*simp add*:*special-sg-G*)
apply (*rotate-tac −1*) **apply** *simp*
apply (*erule conjE*)
apply (*simp add*:*tD-gchain-def*)
apply (*erule conjE*)+
apply (*simp add*: *W-cmpser-def*)
done

lemma (**in** *Group*) *joint-W-cmpser*:⟦*W-cmpser G n f*; *W-cmpser G m g*;
(*Gp G* (*f n*)) ▷ (*g 0*); *g 0* ⊂ *f n*⟧ $\Longrightarrow$
W-cmpser G (*Suc* (*n* + *m*)) (*jointfun n f m g*)
apply (*simp add*: *W-cmpser-def* [*of - Suc* (*n* + *m*) -])
apply (*frule W-cmpser-is-D-gchain* [*of n f*],
frule W-cmpser-is-D-gchain [*of m g*])
apply (*simp add*:*joint-D-gchains*)
apply (*frule W-cmpser-is-w-cmpser* [*of n* -],
frule W-cmpser-is-w-cmpser [*of m* -])
apply (*frule joint-w-cmpser* [*of n f m g*], *assumption*+)
apply (*simp add*:*w-cmpser-def* [*of - Suc* (*n* + *m*) -])
done

lemma (**in** *Group*) *joint-d-gchain-n0*:⟦*d-gchain G n f*; *d-gchain G 0 g*;
g 0 ⊆ *f n* ⟧ $\Longrightarrow$ *d-gchain G* (*Suc n*) (*jointfun n f 0 g*)
apply (*frule joint-d-gchains* [*of n f 0 g*], *assumption*+)
apply *simp*
done

lemma (**in** *Group*) *joint-D-gchain-n0*:⟦*D-gchain G n f*; *D-gchain G 0 g*;
g 0 ⊂ *f n* ⟧ $\Longrightarrow$ *D-gchain G* (*Suc n*) (*jointfun n f 0 g*)
apply (*frule joint-D-gchains* [*of n f 0 g*], *assumption*+)
apply *simp*
done

lemma (**in** *Group*) *joint-w-cmpser-n0*:⟦*w-cmpser G n f*; *w-cmpser G 0 g*;
(*Gp G* (*f n*)) ▷ (*g 0*)⟧ $\Longrightarrow$ *w-cmpser G* (*Suc n*) (*jointfun n f 0 g*)
apply (*frule joint-w-cmpser* [*of n f 0 g*], *assumption*+)
apply *simp*
done

lemma (**in** *Group*) *joint-W-cmpser-n0*:⟦*W-cmpser G n f*; *W-cmpser G 0 g*;
(*Gp G* (*f n*)) ▷ (*g 0*); *g 0* ⊂ *f n* ⟧ $\Longrightarrow$

W-cmpser G (Suc n) (jointfun n f 0 g)
apply (*frule joint-W-cmpser* [*of n f 0 g*], *assumption*+)
apply *simp*
done

definition
simple-Group :: - ⇒ *bool* **where**
simple-Group G ⟷ {*N*. *G* ≫ *N*} = {*carrier G*, {**1**$_G$}}

definition
compseries:: [- , *nat*, *nat* ⇒ *'a set*] ⇒ *bool* **where**
compseries G n f ⟷ *tW-cmpser G n f* ∧ (*if n* = *0 then f 0* = {**1**$_G$} *else*
(∀ *i* ≤ (*n* − *1*). (*simple-Group* ((*Gp G* (*f i*))/(*f* (*Suc i*))))))

definition
length-twcmpser :: [- , *nat*, *nat* ⇒ *'a set*] ⇒ *nat* **where**
length-twcmpser G n f = *card* (*f* ‘{*i*. *i* ≤ *n*}) − *Suc 0*

lemma (**in** *Group*) *compseriesTr0*:⟦*compseries G n f*; *i* ≤ *n*⟧ ⟹
G ≫ (*f i*)
apply (*simp add*:*compseries-def*)
apply (*frule conjunct1*)
apply (*fold compseries-def*)
apply (*frule tW-cmpser-is-W-cmpser*,
frule W-cmpser-is-w-cmpser,
frule w-cmpser-is-d-gchain)
apply (*frule d-gchain-mem-sg*[*of n f*])
apply *simp*
done

lemma (**in** *Group*) *compseriesTr1*:*compseries G n f* ⟹ *tW-cmpser G n f*
apply (*simp add*:*compseries-def*)
done

lemma (**in** *Group*) *compseriesTr2*:*compseries G n f* ⟹ *f 0* = *carrier G*
apply (*frule compseriesTr1*, *simp add*:*tW-cmpser-def*)
apply (*case-tac n* = *0*)
apply (*simp add*:*td-gchain-def*)
apply *simp*
apply (*erule conjE*, *simp add*:*tD-gchain-def*)
done

lemma (**in** *Group*) *compseriesTr3*:*compseries G n f* ⟹ *f n* = {**1**}
apply (*frule compseriesTr1*)
apply (*simp add*:*tW-cmpser-def*)
apply (*case-tac n* = *0*)
apply (*simp add*:*td-gchain-def*)
apply (*auto del*:*equalityI*)

apply (*simp add*:*tD-gchain-def*)
done

lemma (**in** *Group*) *compseriesTr4*:*compseries G n f* $\Longrightarrow$ *w-cmpser G n f*
apply (*frule compseriesTr1*,
  frule tW-cmpser-is-W-cmpser)
apply (*rule W-cmpser-is-w-cmpser*, *assumption*)
done

lemma (**in** *Group*) *im-jointfun1Tr1*:$\forall\, l \leq n.\ G \gg (f\ l) \Longrightarrow$
  $f \in \{i.\ i \leq n\} \to$ *Collect* (*sg G*)
apply (*simp add*:*Pi-def*)
done

lemma (**in** *Group*) *Nset-Suc-im*:$\forall\, l \leq (Suc\ n).\ G \gg (f\ l) \Longrightarrow$
  insert (*f* (*Suc n*)) (*f* ‘ $\{i.\ i \leq n\}$) = *f* ‘ $\{i.\ i \leq (Suc\ n)\}$
apply (*rule equalityI*)
 apply (*rule subsetI*)
  apply (*simp add*:*image-def*)
  apply (*erule disjE*) **apply** *blast*
  apply (*erule exE*, *erule conjE*)
  apply (*frule-tac x* = *xa* **and** *y* = *n* **and** *z* = *Suc n* **in** *le-less-trans*,
    simp,
    frule-tac x = *xa* **and** *y* = *Suc n* **in** *less-imp-le*, *blast*)
 apply (*cut-tac Nset-Suc* [*of n*], *simp*)
done

definition
  NfuncPair-neq-at::[*nat* $\Rightarrow$ *'a set*, *nat* $\Rightarrow$ *'a set*, *nat*] $\Rightarrow$ *bool* **where**
  NfuncPair-neq-at f g i $\longleftrightarrow$ *f i* $\neq$ *g i*

lemma *LeastTr0*:$[\![\ (i::nat) < (LEAST\ l.\ P\ (l))]\!] \Longrightarrow \neg\ P\ (i)$
apply (*rule not-less-Least*)
apply *simp*
done

lemma (**in** *Group*) *funeq-LeastTr1*:$[\![\forall\, l \leq n.\ G \gg f\ l;\ \forall\, l \leq n.\ G \gg g\ l;$
  $(l :: nat) < (LEAST\ k.\ (NfuncPair\text{-}neq\text{-}at\ f\ g\ k))\ ]\!] \Longrightarrow f\ l = g\ l$
apply (*rule contrapos-pp*, *simp*+)
apply (*frule LeastTr0* [*of l NfuncPair-neq-at f g*])
apply (*simp add*:*NfuncPair-neq-at-def*)
done

lemma (**in** *Group*) *funeq-LeastTr1-1*:$[\![\forall\, l \leq (n::nat).\ G \gg f\ l;\ \forall\, l \leq n.\ G \gg\ g\ l;$
  $(l :: nat) < (LEAST\ k.\ (f\ k \neq\ g\ k))\ ]\!] \Longrightarrow f\ l = g\ l$
apply (*rule funeq-LeastTr1*[*of n f g l*], *assumption*+)
apply (*simp add*:*NfuncPair-neq-at-def*)
done

lemma (**in** *Group*) *Nfunc-LeastTr2-1*:⟦$i \leq n$; $\forall\, l \leq n.\ G \gg f\ l$; $\forall\, l \leq n.\ G \gg g\ l$;
NfuncPair-neq-at f g i⟧ ⟹
NfuncPair-neq-at f g (*LEAST k.* (*NfuncPair-neq-at f g k*))
apply (*simp add*: *LeastI* [*of NfuncPair-neq-at f g i*])
done

lemma (**in** *Group*) *Nfunc-LeastTr2-2*:⟦$i \leq n$; $\forall\, l \leq n.\ G \gg f\ l$; $\forall\, l \leq n.\ G \gg g\ l$;
NfuncPair-neq-at f g i⟧ ⟹
(*LEAST k.* (*NfuncPair-neq-at f g k*)) $\leq i$
apply (*simp add*: *Least-le* [*of NfuncPair-neq-at f g i*])
done

lemma (**in** *Group*) *Nfunc-LeastTr2-2-1*:⟦$i \leq$ (*n*::*nat*); $\forall\, l \leq n.\ G \gg f\ l$;
$\forall\, l \leq n.\ G \gg g\ l$; $f\ i \neq g\ i$⟧ ⟹ (*LEAST k.* ($f\ k \neq g\ k$)) $\leq i$
apply (*rule contrapos-pp*, *simp*+)
apply (*simp add*:*nat-not-le-less*)
apply (*frule Nfunc-LeastTr2-2* [*of i n f g*], *assumption*+)
apply (*simp add*:*NfuncPair-neq-at-def*)+
done

lemma (**in** *Group*) *Nfunc-LeastTr2-3*:⟦$\forall\, l \leq$ (*n*::*nat*). $G \gg f\ l$; $\forall\, l \leq n.\ G \gg g\ l$;
$i \leq n$; $f\ i \neq g\ i$⟧ ⟹
f (*LEAST k.* ($f\ k \neq\ g\ k$)) $\neq g$ (*LEAST k.* ($f\ k \neq\ g\ k$))
apply (*frule Nfunc-LeastTr2-1* [*of i n f g*], *assumption*+)
apply (*simp add*:*NfuncPair-neq-at-def*)+
done

lemma (**in** *Group*) *Nfunc-LeastTr2-4*:⟦$\forall\, l \leq$ (*n*::*nat*). $G \gg f\ l$; $\forall\, l \leq n.\ G \gg g\ l$;
$i \leq n$; $f\ i \neq g\ i$⟧ ⟹(*LEAST k.* ($f\ k \neq g\ k$)) $\leq n$
apply (*frule-tac* $i = i$ **in** *Nfunc-LeastTr2-2* [*of* - *n f g*],
assumption+)
apply (*simp add*:*NfuncPair-neq-at-def*)
apply (*frule le-trans* [*of* (*LEAST k. NfuncPair-neq-at f g k*) *i n*],
assumption+)
apply (*simp add*:*NfuncPair-neq-at-def*)
done

lemma (**in** *Group*) *Nfunc-LeastTr2-5*:⟦$\forall\, l \leq$ (*n*::*nat*). $G \gg f\ l$; $\forall\, l \leq n.\ G \gg g\ l$;
$\exists\, i \leq n.\ (f\ i \neq g\ i)$⟧ ⟹
f (*LEAST k.* ($f\ k \neq g\ k$)) $\neq g$ ((*LEAST k.* $f\ k \neq g\ k$))
apply (*erule exE*)
apply (*rule-tac* $i = i$ **in** *Nfunc-LeastTr2-3*[*of n f g*], *assumption*+, *simp*+)
done

lemma (**in** *Group*) *Nfunc-LeastTr2-6*:⟦$\forall\, l \leq$ (*n*::*nat*). $G \gg f\ l$; $\forall\, l \leq n.\ G \gg g\ l$;
$\exists\, i \leq n.\ (f\ i \neq g\ i)$⟧ ⟹ (*LEAST k.* ($f\ k \neq g\ k$)) $\leq n$
apply (*erule exE*)
apply (*rule-tac* $i = i$ **in** *Nfunc-LeastTr2-4*, *assumption*+, *simp*+)
done

lemma (**in** *Group*) *Nfunc-Least-sym*:⟦∀ *l* ≤ (*n*::*nat*). *G* ≫ *f l*; ∀ *l* ≤ *n*. *G* ≫ *g l*;
∃ *i* ≤ *n*. (*f i* ≠ *g i*)⟧ ⟹
(*LEAST k*. (*f k* ≠ *g k*)) = (*LEAST k*. (*g k* ≠ *f k*))
apply (*erule exE*)
apply (*frule-tac i* = *i* **in** *Nfunc-LeastTr2-4* [*of n f g*], *assumption*+,
simp+,
frule-tac i = *i* **in** *Nfunc-LeastTr2-3* [*of n f g* -], *assumption*+,
simp+,
frule-tac i = *i* **in** *Nfunc-LeastTr2-4* [*of n g f*], *assumption*+,
simp+, *rule not-sym*, *simp*,
frule-tac i = *i* **in** *Nfunc-LeastTr2-3* [*of n g f* -], *assumption*+,
simp+, *rule not-sym*, *simp*)
apply (*frule-tac i* = (*LEAST k*. *f k* ≠ *g k*) **in**
Nfunc-LeastTr2-2-1 [*of* - *n g f*], *assumption*+,
rule not-sym, *simp*) **apply** (
frule-tac i = (*LEAST k*. *g k* ≠ *f k*) **in**
Nfunc-LeastTr2-2-1 [*of* - *n f g*], *assumption*+,
rule not-sym, *simp*)
apply (*rule le-antisym*, *assumption*+)
done

lemma *Nfunc-iNJTr*:⟦*inj-on g* {*i*. *i* ≤ (*n*::*nat*)}; *i* ≤ *n*; *j* ≤ *n*; *i* < *j* ⟧ ⟹ *g i* ≠ *g j*
apply (*unfold inj-on-def*)
apply (*simp add*:*CollectI*)
apply (*rule contrapos-pp*, *simp*+)
apply (*frule-tac a* = *i* **in** *forall-spec*)
apply (*frule-tac x* = *i* **and** *y* = *j* **and** *z* = *n* **in** *less-le-trans*, *assumption*+,
simp add:*less-imp-le*,
thin-tac ∀ *x*≤*n*. ∀ *y*≤*n*. *g x* = *g y* ⟶ *x* = *y*,
rotate-tac −*1*,
frule-tac a = *j* **in** *forall-spec*, *assumption*,
thin-tac ∀ *y*≤*n*. *g i* = *g y* ⟶ *i* = *y*)
apply *simp*
done

lemma (**in** *Group*) *Nfunc-LeastTr2-7*:⟦∀ *l* ≤ (*n*::*nat*). *G* ≫ *f l*; ∀ *l* ≤ *n*. *G* ≫ *g l*;
inj-on g {*i*. *i* ≤ *n*}; ∃ *i* ≤ *n*. (*f i* ≠ *g i*);
f k = *g* (*LEAST k*.(*f k* ≠ *g k*))⟧ ⟹(*LEAST k*.(*f k* ≠ *g k*)) < *k*
apply (*rule contrapos-pp*, *simp*+)
apply (*simp add*:*nat-not-le-less*[*THEN sym*, *of LEAST k*. *f k* ≠ *g k k*])
apply (*frule le-imp-less-or-eq*)
apply (*case-tac k* = (*LEAST k*. *f k* ≠ *g k*))
apply *simp*
apply (*frule Nfunc-LeastTr2-5* [*of n f g*], *assumption*+)
apply *simp*
apply (*frule funeq-LeastTr1-1* [*of n f g k*], *assumption*+)
apply *simp*

apply (*frule Nfunc-LeastTr2-6* [*of n f g*], *assumption*+)
apply *simp*
apply (*frule-tac le-trans*[*of k LEAST k. f k* ≠ *g k n*], *assumption*+)
apply (*frule mem-of-Nset*[*of k n*])
apply (*simp add*:*inj-on-def*[*of g*])
done

lemma (**in** *Group*) *Nfunc-LeastTr2-8*:⟦∀ *l* ≤ *n*. *G* ⋗ *f l*; ∀ *l* ≤ *n*. *G* ⋗ *g l*;
inj-on g {*i*. *i* ≤ *n*}; ∃ *i* ≤ *n*. *f i* ≠ *g i*; *f* ‘{*i*. *i* ≤ *n*} = *g* ‘{*i*. *i* ≤ *n*}⟧
⟹
∃ *k* ∈(*nset* (*Suc* (*LEAST i*. (*f i* ≠ *g i*))) *n*). *f k* = *g* (*LEAST i*. (*f i* ≠ *g i*))
apply (*frule-tac Nfunc-LeastTr2-6* [*of n f g*], *assumption*+)
apply (*cut-tac mem-in-image2*[*of LEAST k. f k* ≠ *g k* {*i*. *i* ≤ *n*} *g*])
apply (*frule sym*, *thin-tac f* ‘ {*i*. *i* ≤ *n*} = *g* ‘ {*i*. *i* ≤ *n*}, *simp*)
apply (*thin-tac g* ‘ {*i*. *i* ≤ *n*} = *f* ‘ {*i*. *i* ≤ *n*})
apply (*simp add*:*image-def*)
apply (*rotate-tac* −*1*, *erule exE*)
apply (*frule-tac k* = *x* **in** *Nfunc-LeastTr2-7*[*of n f g*], *assumption*+)
apply (*erule conjE*) **apply** (*rule sym*, *assumption*)
apply (*frule-tac m* = (*LEAST k. f k* ≠ *g k*) **and** *n* = *x* **in** *Suc-leI*)
apply (*simp add*:*nset-def*)
apply *blast* **apply** *simp*
done

lemma (**in** *Group*) *ex-redchainTr1*:⟦*d-gchain G n f*;
D-gchain G (*card* (*f* ‘ {*i*. *i* ≤ *n*}) − *Suc 0*) *g*;
g ‘ {*i*. *i* ≤ (*card* (*f* ‘ {*i*. *i* ≤ *n*}) − *Suc 0*)} = *f* ‘ {*i*. *i* ≤ *n*}⟧ ⟹
g (*card* (*f* ‘ {*i*. *i* ≤ *n*}) − *Suc 0*) = *f n*
apply (*case-tac n* = *0*, *simp*, *simp*)
apply (*rule contrapos-pp*, *simp*+)
apply (*cut-tac n-in-Nsetn*[*of card* (*f* ‘ {*i*. *i* ≤ *n*}) − *Suc 0*])
apply (*frule mem-in-image2*[*of card* (*f* ‘ {*i*. *i* ≤ *n*}) − *Suc 0*
{*i*. *i* ≤ (*card* (*f* ‘ {*i*. *i* ≤ *n*}) − *Suc 0*)} *g*])
apply (*cut-tac n-in-Nsetn*[*of n*])
apply (*frule mem-in-image2*[*of n* {*i*. *i* ≤ *n*} *f*])
apply *simp*
apply (*simp add*:*image-def*[*of f* {*i*. *i* ≤ *n*}])
apply (*erule exE*)
apply (*frule-tac l* = *x* **in** *d-gchainTr2*[*of n f* - *n*], *assumption*+)
apply *simp*+
apply (*erule conjE*)
apply (*rotate-tac* −*1*, *frule sym*,
thin-tac g (*card* {*y*. ∃ *x*≤*n*. *y* = *f x*} − *Suc 0*) = *f x*,
simp,
thin-tac f x = *g* (*card* {*y*. ∃ *x*≤*n*. *y* = *f x*} − *Suc 0*))
apply (*cut-tac mem-in-image2*[*of n* {*i*. *i* ≤ *n*} *f*],
unfold image-def)
apply (*frule sym*,
thin-tac {*y*. ∃ *x*∈{*i*. *i* ≤ *card* {*y*. ∃ *x*≤*n*. *y* = *f x*} − *Suc 0*}. *y* = *g x*} =

```
        {y. ∃x≤n. y = f x})
 apply (cut-tac eq-set-inc[of f n {y. ∃x ≤ n. y = f x}
        {y. ∃x∈{i. i ≤ card {y. ∃x≤n. y = f x} − Suc 0}. y = g x}])
 apply (thin-tac {y. ∃x≤n. y = f x} =
        {y. ∃x∈{i. i ≤ card {y. ∃x≤n. y = f x} − Suc 0}. y = g x},
        thin-tac f n ∈ {y. ∃x∈{i. i ≤ n}. y = f x})
apply  (simp, erule exE, erule conjE)
 apply (case-tac xa ≤ card {y. ∃x≤n. y = f x} − Suc 0, simp)
 apply (frule D-gchain-is-d-gchain[of card {y. ∃x≤n. y = f x} − Suc 0 g])
 apply (case-tac card {y. ∃x ≤ n. y = f x} − Suc 0 = 0,
        simp)
 apply (frule nat-nonzero-pos[of card {y. ∃x ≤ n. y = f x} − Suc 0],
        thin-tac card {y. ∃x≤n. y = f x} − Suc 0 ≠ 0)
 apply (frule-tac l = xa in d-gchainTr2[of
  card {y. ∃x ≤ n. y = f x} − Suc 0 g -
                card {y. ∃x ≤ n. y = f x} − Suc 0], assumption+)
 apply simp apply simp
 apply (frule-tac A = g xa and
        B = g (card {y. ∃x ≤ n. y = f x} − Suc 0) in equalityI,
        assumption+, simp) apply simp+
done

lemma (in Group) ex-redchainTr1-1:⟦d-gchain G (n::nat) f;
       D-gchain G (card (f ‘ {i. i ≤ n}) − Suc 0) g;
       g ‘ {i. i ≤ (card (f ‘ {i. i ≤ n}) − Suc 0)} = f ‘ {i. i ≤ n}⟧ ⟹
       g 0 = f 0
apply (cut-tac Nset-inc-0[of n],
        frule mem-in-image2[of 0 {i. i ≤ n} f]) apply (
        frule sym) apply (
       thin-tac g ‘ {i. i ≤ card (f ‘ {i. i ≤ n}) − Suc 0} = f ‘ {i. i ≤ n})
  apply (
       frule eq-set-inc[of f 0 f ‘ {i. i ≤ n}
                g ‘ {i. i ≤ card (f ‘ {i. i ≤ n}) − Suc 0}], assumption)
apply (
 thin-tac f 0 ∈ f ‘ {i. i ≤ n},
        thin-tac 0 ∈ {i. i ≤ n})
apply (cut-tac Nset-inc-0[of card (f ‘ {i. i ≤ n}) − Suc 0],
        frule mem-in-image2[of 0 {i. i ≤ (card (f ‘ {i. i ≤ n}) − Suc 0)}
         g],
        frule sym) apply (
       frule eq-set-inc[of g 0 g ‘ {i. i ≤ card (f ‘ {i. i ≤ n}) − Suc 0}
             f ‘ {i. i ≤ n}], assumption) apply (
       thin-tac f ‘ {i. i ≤ n} = g ‘ {i. i ≤ card (f ‘ {i. i ≤ n}) − Suc 0})
 apply (
       thin-tac 0 ∈ {i. i ≤ card (f ‘ {i. i ≤ n}) − Suc 0}) apply (
       thin-tac g ‘ {i. i ≤ card (f ‘ {i. i ≤ n}) − Suc 0} = f ‘ {i. i ≤ n})
 apply (
       thin-tac g 0 ∈ g ‘ {i. i ≤ card (f ‘ {i. i ≤ n}) − Suc 0})
apply (case-tac n = 0, simp)
```

apply (*simp*)
apply (*cut-tac mem-in-image3*[*of f 0 g*
{*i*. *i* ≤ *card* (*f* ‘ {*i*. *i* ≤ *n*}) − *Suc 0*}],
frule mem-in-image3[*of g 0 f*
{*i*. *i* ≤ *n*}]) **apply** (
thin-tac f 0 ∈ *g* ‘ {*i*. *i* ≤ *card* (*f* ‘ {*i*. *i* ≤ *n*}) − *Suc 0*},
thin-tac g 0 ∈ *f* ‘ {*i*. *i* ≤ *n*}) **apply** (*erule bexE*)+ **apply** (
frule-tac j = *aa* **in** *d-gchainTr2*[*of n f 0*], *assumption*+)
apply *simp*+
apply (*rotate-tac* −*2*, *frule sym*, *thin-tac g 0* = *f aa*, *simp*)
apply (*case-tac a* = *0*, *simp*)
apply (*simp*,
frule-tac j = *a* **in** *D-gchain0*[*of card* (*f* ‘ {*i*. *i* ≤ *n*}) − *Suc 0 g 0*],
simp add:*Nset-inc-0*, *assumption*+,
simp add:*psubset-contr*)
apply *simp*
done

lemma (**in** *Group*) *ex-redchainTr2*:*d-gchain G* (*Suc n*) *f*
⟹ *D-gchain G 0* (*constmap* {*0*::*nat*} {*f* (*Suc n*)})
apply (*simp add*:*D-gchain-def constmap-def*)
apply (*simp add*:*d-gchain-def*)
done

lemma (**in** *Group*) *last-mem-excluded*:⟦*d-gchain G* (*Suc n*) *f*; *f n* ≠ *f* (*Suc n*)⟧
⟹
f (*Suc n*) ∉ *f* ‘ {*i*. *i* ≤ *n*}
apply (*rule contrapos-pp*, *simp*+)
apply (*frule mem-in-image3*[*of f* (*Suc n*) *f* {*i*. *i* ≤ *n*}], *erule bexE*)
apply (*cut-tac zero-less-Suc*[*of n*])
apply (*frule-tac l* = *a* **in** *d-gchainTr2*[*of Suc n f* - *n*], *assumption*+)
apply *simp*+
apply (*frule sym*, *thin-tac f* (*Suc n*) = *f a*, *simp*)
apply (*cut-tac l* = *n* **and** *j* = *Suc n* **in** *d-gchainTr2*[*of Suc n f*])
apply *simp*+
done

lemma (**in** *Group*) *ex-redchainTr4*:⟦*d-gchain G* (*Suc n*) *f*; *f n* ≠ *f* (*Suc n*)⟧ ⟹
card (*f* ‘ {*i*. *i* ≤ (*Suc n*)}) = *Suc* (*card* (*f* ‘ {*i*. *i* ≤ *n*}))
apply (*cut-tac image-Nset-Suc* [*of f n*])
apply *simp*
apply (*rule card-insert-disjoint*)
apply (*simp*)
apply (*simp add*:*last-mem-excluded*)
done

lemma (**in** *Group*) *ex-redchainTr5*:*d-gchain G n f* ⟹ *0* < *card* (*f* ‘ {*i*. *i*≤ *n*})
apply (*simp add*:*image-Nsetn-card-pos*)
done

lemma (**in** *Group*) *ex-redchainTr6*:$\forall f.$ *d-gchain G n f* $\longrightarrow$
(∃ *g*. *D-gchain G* (*card* (*f* ‘{*i*. *i* ≤ *n*}) − *1*) *g* ∧
(*g* ‘ {*i*. *i* ≤ (*card* (*f* ‘{*i*. *i* ≤ *n*}) − *1*)} = *f* ‘{*i*. *i* ≤ *n*}))
apply (*induct-tac n*)
apply (*rule allI*, *rule impI*)
apply (*simp add*:*image-def*)
apply (*simp add*:*D-gchain-def d-gchain-def*)
apply *blast*

apply (*rule allI*, *rule impI*)
apply (*case-tac f* (*Suc n*) = *f n*)
apply (*cut-tac n* = *n* **in** *Nset-Suc*)
apply (*cut-tac n* = *n* **in** *n-in-Nsetn*,
frule-tac a = *n* **and** *A* = {*i*. *i* ≤ *n*} **and** *f* = *f* **in** *mem-in-image2*)
apply (*frule sym*) **apply** (*thin-tac f* (*Suc n*) = *f n*, *simp*)
apply (*subst insert-absorb*, *assumption*)+
apply (*frule-tac n* = *n* **and** *f* = *f* **in** *d-gchain-pre*, *blast*)

apply (*frule-tac n* = *n* **and** *f* = *f* **in** *d-gchain-pre*)
apply (*frule-tac a* = *f* **in** *forall-spec*, *assumption*,
thin-tac ∀*f*. *d-gchain G n f* $\longrightarrow$
(∃ *g*. *D-gchain G* (*card* (*f* ‘ {*i*. *i* ≤ *n*}) − *1*) *g* ∧
g ‘ {*i*. *i* ≤ *card* (*f* ‘ {*i*. *i* ≤ *n*}) − *1*} =
f ‘ {*i*. *i* ≤ *n*}))
apply (*erule exE*, *erule conjE*)
apply (*simp add*:*image-Nset-Suc*)

apply (*frule-tac n* = *n* **and** *f* = *f* **in** *ex-redchainTr2*)
apply (*frule-tac n* = *card* (*f* ‘ {*i*. *i* ≤ *n*}) − *Suc 0* **and** *f* = *g* **and**
g = *constmap* {*0*} {*f* (*Suc n*)} **in** *joint-D-gchain-n0*, *assumption*+)
apply (*simp add*: *ex-redchainTr1*)
apply (*simp add*: *constmap-def Nset-inc-0*)
apply (*cut-tac n* = *n* **in** *zero-less-Suc*)
apply (*frule-tac n* = *Suc n* **and** *f* = *f* **and** *l* = *n* **and** *j* = *Suc n* **in**
d-gchainTr2, *assumption*+)
apply *simp* **apply** *simp* **apply** *simp*
apply (*simp add*:*psubset-eq*)
apply (*cut-tac f* = *f* **and** *n* = *n* **in** *image-Nsetn-card-pos*,
cut-tac k = *n* **in** *finite-Collect-le-nat*,
frule-tac F = {*i*. *i* ≤ *n*} **and** *h* = *f* **in** *finite-imageI*,
frule-tac n = *n* **and** *f* = *f* **in** *last-mem-excluded*,
rule not-sym, *assumption*)
apply *simp*+
apply (*cut-tac n* = *card* (*f* ‘ {*i*. *i* ≤ *n*}) − *Suc 0* **and** *f* = *g* **and** *m* = *0* **and**
g = *constmap* {*0*} {*f* (*Suc n*)} **in** *im-jointfun1*)
apply *simp*
apply (*simp add*:*Nset-0 constmap-def*)
apply *blast*

done

lemma (**in** *Group*) *ex-redchain:d-gchain G n f* $\Longrightarrow$
$(\exists g.$ *D-gchain G* (*card* (*f* ‘ $\{i.\ i \leq n\}$) $-$ *1*) *g* $\wedge$
g ‘ $\{i.\ i \leq$ (*card* (*f* ‘ $\{i.\ i \leq n\}$) $-$ *1*)$\}$ $=$ *f* ‘ $\{i.\ i \leq n\}$)
apply (*cut-tac ex-redchainTr6* [*of n*])
apply *simp*
done

lemma (**in** *Group*) *const-W-cmpser:d-gchain G* (*Suc n*) *f* $\Longrightarrow$
W-cmpser G 0 (*constmap* $\{0$::*nat*$\}$ $\{f$ (*Suc n*)$\}$)
apply (*simp add*:*W-cmpser-def d-gchain-def constmap-def*)
done

lemma (**in** *Group*) *ex-W-cmpserTr0m*:$\forall f.$ *w-cmpser G m f* $\longrightarrow$
$(\exists g.$ (*W-cmpser G* (*card* (*f* ‘$\{i.\ i \leq m\}$) $-$ *1*) *g* $\wedge$
g ‘$\{i.\ i \leq$ (*card* (*f* ‘$\{i.\ i \leq m\}$) $-$ *1*)$\}$ $=$ *f* ‘$\{i.\ i \leq m\}$))
apply (*induct-tac m*)

apply (*rule allI*, *rule impI*)
apply *simp*
apply (*simp add*:*w-cmpser-def W-cmpser-def*)
apply *blast*

apply (*rule allI*, *rule impI*)
apply (*case-tac f* (*Suc n*) $=$ *f n*)
apply (*cut-tac n* $=$ *n* **in** *Nset-Suc*)
apply (*cut-tac n* $=$ *n* **in** *n-in-Nsetn*,
frule-tac a $=$ *n* **and** *A* $=$ $\{i.\ i \leq n\}$ **and** *f* $=$ *f* **in** *mem-in-image2*)
apply (*frule sym*) **apply** (*thin-tac f* (*Suc n*) $=$ *f n*, *simp*)
apply (*subst insert-absorb*, *assumption*)+
apply (*frule-tac n* $=$ *n* **and** *f* $=$ *f* **in** *w-cmpser-pre*, *blast*)

apply (*frule-tac n* $=$ *n* **and** *f* $=$ *f* **in** *w-cmpser-pre*)
apply (*frule-tac a* $=$ *f* **in** *forall-spec*, *assumption*,
thin-tac $\forall f.$ *w-cmpser G n f* $\longrightarrow$
$(\exists g.$ *W-cmpser G* (*card* (*f* ‘ $\{i.\ i \leq n\}$) $-$ *1*) *g* $\wedge$
g ‘ $\{i.\ i \leq$ *card* (*f* ‘ $\{i.\ i \leq n\}$) $-$ *1*$\}$ $=$
f ‘ $\{i.\ i \leq n\}$))
apply (*erule exE*, *erule conjE*)
apply (*simp add*:*image-Nset-Suc*,
frule-tac n $=$ *Suc n* **and** *f* $=$ *f* **in** *w-cmpser-is-d-gchain*,
frule-tac n $=$ *n* **and** *f* $=$ *f* **in** *const-W-cmpser*)
apply (*frule-tac n* $=$ *card* (*f* ‘ $\{i.\ i \leq n\}$) $-$ *Suc 0* **and** *f* $=$ *g* **and**
g $=$ *constmap* $\{0$::*nat*$\}$ $\{f$ (*Suc n*)$\}$ **in** *joint-W-cmpser-n0*, *assumption*+)
apply (*frule-tac n* $=$ *card* (*f* ‘ $\{i.\ i \leq n\}$) $-$ *Suc 0* **and** *f* $=$ *g* **in**
W-cmpser-is-D-gchain)
apply (*frule d-gchain-pre*)
apply (*subst ex-redchainTr1*, *assumption*+)

apply (*simp add*:*constmap-def Nset-inc-0*)
apply (*simp add*:*w-cmpser-def*)
apply (*frule d-gchain-pre*)
apply (*frule-tac n* = *card* (*f* ‘ {*i*. *i* ≤ *n*}) − *Suc 0* **and** *f* = *g* **in**
W-cmpser-is-D-gchain)
apply (*frule-tac n* = *n* **and** *f* = *f* **and** *g* = *g* **in** *ex-redchainTr1*, *assumption*+)
apply *simp*
apply (*simp add*:*constmap-def Nset-inc-0*,
thin-tac d-gchain G n f, *simp add*:*d-gchain-def*)
apply (*cut-tac n* = *Suc n* **in** *n-in-Nsetn*,
frule-tac x = *Suc n* **in** *spec*, *simp*,
simp add:*psubset-eq*)
apply (*cut-tac f* = *f* **and** *n* = *n* **in** *image-Nsetn-card-pos*,
cut-tac k = *n* **in** *finite-Collect-le-nat*,
frule-tac F = {*i*. *i* ≤ *n*} **and** *h* = *f* **in** *finite-imageI*,
frule-tac n = *n* **and** *f* = *f* **in** *last-mem-excluded*,
rule not-sym, *assumption*)
apply *simp*+
apply (*cut-tac n* = *card* (*f* ‘ {*i*. *i* ≤ *n*}) − *Suc 0* **and** *f* = *g* **and** *m* = *0* **and**
g = *constmap* {*0*::*nat*} {*f* (*Suc n*)} **in** *im-jointfun1*)
apply *simp*
apply (*simp add*:*Nset-0 constmap-def*)
apply *blast*
done

lemma (**in** *Group*) *ex-W-cmpser*:*w-cmpser G m f* ⟹
∃ *g*. *W-cmpser G* (*card* (*f* ‘ {*i*. *i* ≤ *m*}) − *1*) *g* ∧
g ‘ {*i*. *i* ≤ (*card* (*f* ‘ {*i*. *i* ≤ *m*}) − *1*)} = *f* ‘ {*i*. *i* ≤ *m*}
apply (*cut-tac ex-W-cmpserTr0m* [*of m*])
apply *simp*
done

3.15 Existence of reduced chain and composition series

lemma (**in** *Group*) *ex-W-cmpserTr3m1*:⟦*tw-cmpser G* (*m*::*nat*) *f*;
W-cmpser G ((*card* (*f* ‘ {*i*. *i* ≤ *m*})) − *1*) *g*;
g ‘ {*i*. *i* ≤ ((*card* (*f* ‘ {*i*. *i* ≤ *m*})) − *1*)} = *f* ‘{*i*. *i* ≤ *m*}⟧ ⟹
tW-cmpser G ((*card* (*f* ‘ {*i*. *i* ≤ *m*})) − *1*) *g*
apply (*frule-tac tw-cmpser-is-w-cmpser* [*of m f*],
frule-tac w-cmpser-is-d-gchain [*of m f*],
frule-tac W-cmpser-is-D-gchain [*of* (*card* (*f* ‘ {*i*. *i* ≤ *m*}) − *1*) *g*])

apply (*frule ex-redchainTr1* [*of m f g*])
apply *simp*+
apply (*frule-tac ex-redchainTr1-1* [*of m f g*])
apply (*simp add*:*tW-cmpser-def tw-cmpser-def*)
apply (*case-tac m* = *0*) **apply** *simp*

apply (*cut-tac card-image-le* [*of* {*0*::*nat*} *f*])
apply (*simp*, *simp*)
apply (*simp add*:*tW-cmpser-def*)
apply (*case-tac card* (*f* ‘ {*i*. *i* ≤ *m*}) ≤ *Suc 0*) **apply** *simp*
apply (*simp add*:*td-gchain-def tw-cmpser-def*)
apply (*case-tac m* = *0*)
apply (*thin-tac f 0* = *f m*, *thin-tac g 0* = *f m*) **apply** *simp*
apply (*simp add*:*td-gchain-def*) **apply** (*erule conjE*, *simp*)
apply *simp*
apply (*simp add*:*td-gchain-def*[*of G m f*], (*erule conjE*)+, *simp*)
apply *simp*
apply (*simp add*:*tD-gchain-def tw-cmpser-def td-gchain-def* [*of* - *m f*])
apply (*case-tac m* = *0*, *simp add*:*card1*)
apply (*simp*, *erule conjE*, *simp add*:*td-gchain-def*)
apply (*simp add*:*W-cmpser-def*)
done

lemma (**in** *Group*) *ex-W-cmpserTr3m*:*tw-cmpser G m f* ⟹
∃ *g*. *tW-cmpser G* ((*card* (*f* ‘ {*i*. *i* ≤ *m*})) − *1*) *g* ∧
g ‘{ *i*. *i* ≤ (*card* (*f* ‘{*i*. *i* ≤ *m*}) − *1*)} = *f* ‘ {*i*. *i* ≤ *m*}
apply (*frule tw-cmpser-is-w-cmpser*[*of m f*])
apply (*frule ex-W-cmpser* [*of m f*])
apply (*auto del*:*equalityI*)
apply (*frule-tac g* = *g* **in** *ex-W-cmpserTr3m1* [*of m f*])
apply *simp*+ **apply** *blast*
done

definition
red-ch-cd :: [- , *nat* ⇒ *'a set*, *nat*, *nat* ⇒ *'a set*] ⇒ *bool* **where**
red-ch-cd G f m g ⟷ *tW-cmpser G* (*card* (*f* ‘ {*i*. *i* ≤ *m*}) − *1*) *g* ∧
(*g* ‘{*i* . *i* ≤ (*card* (*f* ‘ {*i*. *i* ≤ *m*}) − *1*)} = *f*‘ {*i*. *i* ≤ *m*})

definition
red-chain :: [- , *nat*, *nat* ⇒ *'a set*] ⇒ (*nat* ⇒ *'a set*) **where**
red-chain G m f = (*SOME g*. *g* ∈ {*h*. *red-ch-cd G f m h*})

lemma (**in** *Group*) *red-chainTr0m1-1*:*tw-cmpser G m f* ⟹
(*SOME g*. *g* ∈ {*h*. *red-ch-cd G f m h*}) ∈ {*h*. *red-ch-cd G f m h*}
apply (*rule nonempty-some* [*of* {*h*. *red-ch-cd G f m h*}])
apply (*frule ex-W-cmpserTr3m* [*of m f*])
apply *simp*
apply (*simp add*:*red-ch-cd-def*)
done

lemma (**in** *Group*) *red-chain-m*:*tw-cmpser G m f* ⟹
tW-cmpser G (*card* (*f* ‘ {*i*. *i* ≤ *m*}) − *1*) (*red-chain G m f*) ∧
(*red-chain G m f*) ‘{*i*. *i* ≤ (*card* (*f* ‘{*i*. *i* ≤ *m*}) − *1*)} = *f* ‘ {*i*. *i* ≤ *m*}
apply (*frule red-chainTr0m1-1* [*of m f*])
apply *simp*

apply (*simp add*:*red-ch-cd-def*)
apply (*simp add*:*red-chain-def*)
done

3.16 Chain of groups II

definition
Gchain :: [*nat*, *nat* ⇒ ((*'a set*), *'more*) *Group-scheme*] ⇒ *bool* **where**
Gchain n g ⟷ (∀ *l* ≤ *n*. *Group* (*g l*))

definition
isom-Gchains :: [*nat*, *nat* ⇒ *nat*, *nat* ⇒ ((*'a set*), *'more*) *Group-scheme*,
nat ⇒ ((*'a set*), *'more*) *Group-scheme*] ⇒ *bool* **where**
isom-Gchains n f g h ⟷ (∀ *i* ≤ *n*. (*g i*) ≅ (*h* (*f i*)))

definition
Gch-bridge :: [*nat*, *nat* ⇒ ((*'a set*), *'more*) *Group-scheme*, *nat* ⇒
((*'a set*), *'more*) *Group-scheme*, *nat* ⇒ *nat*] ⇒ *bool* **where**
Gch-bridge n g h f ⟷ (∀ *l* ≤ *n*. *f l* ≤ *n*) ∧ *inj-on f* {*i*. *i* ≤ *n*} ∧
isom-Gchains n f g h

lemma *Gchain-pre*:*Gchain* (*Suc n*) *g* ⟹ *Gchain n g*
apply (*simp add*:*Gchain-def*)
done

lemma (**in** *Group*) *isom-unit*:⟦*G* ⋗ *H*; *G* ⋗ *K*; *H* = {**1**}⟧ ⟹
Gp G H ≅ *Gp G K* ⟶ *K* = {**1**}
apply (*simp add*:*isomorphic-def*)
apply (*rule impI*)
apply (*erule exE*)
apply (*simp add*:*gbijec-def*)
apply (*erule conjE*)
apply (*simp add*:*gsurjec-def ginjec-def*)
apply (*erule conjE*)
apply (*simp add*:*Gp-carrier*)
apply (*simp add*:*surj-to-def*)
apply (*cut-tac a* = *f* **1** **in** *finite1*)

apply (*frule sg-unit-closed* [*of K*])
apply (*frule singleton-sub*[*of* **1** *K*])
apply (*rotate-tac 4*, *frule sym*, *thin-tac* {*f* **1**} = *K*)
apply (*rule card-seteq*[*THEN sym*, *of K* {**1**}])
apply (*simp add*:*finite1*) **apply** *assumption*
apply (*simp add*:*card1*)
done

lemma *isom-gch-unitsTr4*:⟦*Group F*; *Group G*; *Ugp E*; *F* ≅ *G*; *F* ≅ *E*⟧ ⟹
G ≅ *E*

apply (*simp add*:*Ugp-def*)
apply (*erule conjE*)
apply (*frule isomTr1* [*of F G*], *assumption*+)
apply (*rule isomTr2* [*of G F E*], *assumption*+)
done

lemma *isom-gch-cmp*:⟦*Gchain n g*; *Gchain n h*; *f1* ∈ {*i*. *i* ≤ *n*} → {*i*. *i* ≤ *n*};
f2 ∈ {*i*. *i* ≤ *n*} → {*i*. *i* ≤ *n*}; *isom-Gchains n* (*cmp f2 f1*) *g h*⟧ ⟹
isom-Gchains n f1 g (*cmp h f2*)
apply (*simp add*:*isom-Gchains-def*)
apply (*simp add*:*cmp-def*)
done

lemma *isom-gch-transp*:⟦*Gchain n f*; *i* ≤ *n*; *j* ≤ *n*; *i* < *j*⟧ ⟹
isom-Gchains n (*transpos i j*) *f* (*cmp f* (*transpos i j*))
apply (*rule isom-gch-cmp* [*of n f - transpos i j transpos i j*],
assumption+)
apply (*rule transpos-hom*, *assumption*+) **apply** *simp*
apply (*rule transpos-hom*, *assumption*+) **apply** *simp*
apply (*simp add*:*isom-Gchains-def*)
apply (*rule allI*, *rule impI*)
apply (*frule less-le-trans* [*of i j n*], *assumption*+)
apply (*frule less-imp-le* [*of i n*])
apply (*frule-tac k* = *ia* **in** *cmp-transpos1* [*of i n j*], *assumption*+)
apply *simp*+
apply (*simp add*:*Gchain-def*)
done

lemma *isom-gch-units-transpTr0*:⟦*Ugp E*; *Gchain n g*; *Gchain n h*; *i* ≤ *n*; *j* ≤ *n*;
i < *j*; *isom-Gchains n* (*transpos i j*) *g h*⟧ ⟹
{*i*. *i* ≤ *n* ∧ *g i* ≅ *E*} − {*i*, *j*} ={*i*. *i* ≤ *n* ∧ *h i* ≅ *E*} − {*i*, *j*}
apply (*simp add*:*isom-Gchains-def*)
apply (*rule equalityI*)
apply (*rule subsetI*, *simp add*:*CollectI*)
apply (*erule conjE*)+
apply (*cut-tac x* = *x* **in** *transpos-id-1* [*of i n j*], *simp*+)
apply (*subgoal-tac g x* ≅ *h* (*transpos i j x*),
thin-tac ∀ *ia*≤*n*. *g ia* ≅ *h* (*transpos i j ia*), *simp*)
apply (*subgoal-tac Group* (*g x*), *subgoal-tac Group* (*h x*))
apply (*simp add*:*Ugp-def*) **apply** (*erule conjE*)
apply (*frule-tac F* = *g x* **and** *G* = *h x* **in** *isomTr1*, *assumption*+)
apply (*rule-tac F* = *h x* **and** *G* = *g x* **and** *H* = *E* **in** *isomTr2*, *assumption*+)
apply (*simp add*:*Gchain-def*, *simp add*:*Gchain-def*)
apply (*thin-tac transpos i j x* = *x*)
apply *simp*
apply (*rule subsetI*, *simp add*:*CollectI*)
apply (*erule conjE*)+
apply (*cut-tac x* = *x* **in** *transpos-id-1* [*of i n j*], *simp*+)
apply (*subgoal-tac g x* ≅ *h* (*transpos i j x*),

thin-tac $\forall ia{\leq}n.\ g\ ia \cong h\ (transpos\ i\ j\ ia)$)
apply *simp*
apply (*subgoal-tac Group* $(g\ x)$,
subgoal-tac Group $(h\ x)$)
apply (*simp add*:*Ugp-def*) **apply** (*erule conjE*)
apply (*rule-tac* $F = g\ x$ **and** $G = h\ x$ **and** $H = E$ **in** *isomTr2*, *assumption*+)
apply (*simp add*:*Gchain-def*, *simp add*:*Gchain-def*)
apply (*thin-tac transpos i j x* $= x$)
apply *simp*
done

lemma *isom-gch-units-transpTr1*:$[\![Ugp\ E;\ Gchain\ n\ g;\ i \leq n;\ j \leq n;\ g\ j \cong E;$
$i \neq j]\!] \Longrightarrow$
insert $j\ (\{i.\ i \leq n \wedge g\ i \cong E\} - \{i, j\}) = \{i.\ i \leq n \wedge g\ i \cong E\} - \{i\}$
apply (*rule equalityI*)
apply (*rule subsetI*) **apply** (*simp add*:*CollectI*)
apply *blast*
apply (*rule subsetI*) **apply** (*simp add*:*CollectI*)
done

lemma *isom-gch-units-transpTr2*:$[\![Ugp\ E;\ Gchain\ n\ g;\ i \leq n;\ j \leq n;\ i < j;$
$g\ i \cong E]\!] \Longrightarrow$
$\{i.\ i \leq n \wedge g\ i \cong E\} = insert\ i\ (\{i.\ i \leq n \wedge\ \ g\ i \cong E\} - \{i\})$
apply (*rule equalityI*)
apply (*rule subsetI*, *simp add*:*CollectI*)
apply (*rule subsetI*, *simp add*:*CollectI*)
apply (*erule disjE*, *simp*)
apply *simp*
done

lemma *isom-gch-units-transpTr3*:$[\![Ugp\ E;\ Gchain\ n\ g;\ i \leq n]\!]$
$\Longrightarrow finite\ (\{i.\ i \leq n \wedge g\ i \cong E\} - \{i\})$
apply (*rule finite-subset*[*of* $\{i.\ i \leq n \wedge g\ i \cong E\} - \{i\}\ \{i.\ i \leq n\}$])
apply (*rule subsetI*, *simp*+)
done

lemma *isom-gch-units-transpTr4*:$[\![Ugp\ E;\ Gchain\ n\ g;\ i \leq n]\!]$
$\Longrightarrow finite\ (\{i.\ i \leq n \wedge g\ i \cong E\} - \{i, j\})$
apply (*rule finite-subset*[*of* $\{i.\ i \leq n \wedge g\ i \cong E\} - \{i, j\}\ \{i.\ i \leq n\}$])
apply (*rule subsetI*, *simp*+)
done

lemma *isom-gch-units-transpTr5-1*:$[\![Ugp\ E;\ Gchain\ n\ g;\ Gchain\ n\ h;\ i \leq (n{::}nat);$
$j \leq n;\ i < j;\ isom\text{-}Gchains\ n\ (transpos\ i\ j)\ g\ h]\!] \Longrightarrow g\ i \cong h\ j$
apply (*simp add*:*isom-Gchains-def*)
apply (*frule-tac* $a = i$ **in** *forall-spec*,
frule-tac $x = i$ **and** $y = j$ **and** $z = n$ **in** *less-le-trans*,
assumption+, *simp*,
thin-tac $\forall ia \leq n.\ g\ ia \cong h\ (transpos\ i\ j\ ia)$)

apply (*simp add*:*transpos-ij-1* [*of i n j*])
done

lemma *isom-gch-units-transpTr5-2*:⟦*Ugp E*; *Gchain n g*; *Gchain n h*; $i \leq n$;
$j \leq n$; $i < j$; *isom-Gchains n* (*transpos i j*) *g h*⟧ ⟹ $g\ j \cong h\ i$
apply (*simp add*:*isom-Gchains-def*)
apply (*frule-tac* $x = j$ **in** *spec*,
thin-tac $\forall ia{\leq}\ n.\ g\ ia \cong h$ (*transpos i j ia*))
apply (*simp add*:*transpos-ij-2* [*of i n j*])
done

lemma *isom-gch-units-transpTr6*:⟦*Gchain n g*; $i \leq n$⟧ ⟹ *Group* (*g i*)
apply (*simp add*:*Gchain-def*)
done

lemma *isom-gch-units-transpTr7*:⟦*Ugp E*; $i \leq n$; $j \leq n$; $g\ j \cong h\ i$;
Group (*h i*); *Group* (*g j*); $\neg\ g\ j \cong E$⟧ ⟹ $\neg\ h\ i \cong E$
apply (*rule contrapos-pp*, *simp*+)
apply (*frule isomTr2* [*of g j h i E*], *assumption*+)
apply (*simp add*:*Ugp-def*)
apply *assumption*+
apply *simp*
done

lemma *isom-gch-units-transpTr8-1*:⟦*Ugp E*; *Gchain n g*; $i \leq n$; $j \leq n$;
$g\ i \cong E$; $\neg\ g\ j \cong E$⟧ ⟹
$\{i.\ i \leq n \wedge g\ i \cong E\} = \{i.\ i \leq n \wedge g\ i \cong E\} - \{\ j\ \}$
by *auto*

lemma *isom-gch-units-transpTr8-2*:⟦*Ugp E*; *Gchain n g*; $i \leq n$; $j \leq n$;
$\neg\ g\ i \cong E$; $\neg\ g\ j \cong E$⟧ ⟹
$\{i.\ i \leq n \wedge g\ i \cong E\} = \{i.\ i \leq n \wedge g\ i \cong E\} - \{i, j\ \}$
by *auto*

lemma *isom-gch-units-transp*:⟦*Ugp E*; *Gchain n g*; *Gchain n h*; $i \leq n$; $j \leq n$;
$i < j$; *isom-Gchains n* (*transpos i j*) *g h*⟧ ⟹
card $\{i.\ i \leq n \wedge g\ i \cong E\}$ = *card* $\{i.\ i \leq n \wedge h\ i \cong E\}$
apply (*frule isom-gch-units-transpTr0* [*of E n g h i j*],
assumption+)
apply (*frule isom-gch-units-transpTr6* [*of n g i*], *assumption*+)
apply (*frule isom-gch-units-transpTr6* [*of n h i*], *assumption*+)
apply (*frule isom-gch-units-transpTr6* [*of n g j*], *assumption*+)
apply (*frule isom-gch-units-transpTr6* [*of n h j*], *assumption*+)
apply (*unfold Ugp-def*) **apply** (*frule conjunct1*) **apply** (*fold Ugp-def*)
apply (*frule isom-gch-units-transpTr5-1* [*of E n g h i j*],
assumption+)
apply (*frule isom-gch-units-transpTr5-2* [*of E n g h i j*],
assumption+)
apply (*case-tac g i* ≅ *E*)

apply (*case-tac g j* ≅ *E*)
apply (*subst isom-gch-units-transpTr2* [*of E n g i j*], *assumption*+)
apply (*subst isom-gch-units-transpTr2* [*of E n h i j*], *assumption*+)
apply (*rule isom-gch-unitsTr4* [*of g j h i E*], *assumption*+)
apply (*subst card-insert-disjoint*)
apply (*rule isom-gch-units-transpTr3*, *assumption*+)
apply *simp*
apply (*subst card-insert-disjoint*)
apply (*rule isom-gch-units-transpTr3*, *assumption*+)
apply *simp*
apply (*subst isom-gch-units-transpTr1*[*THEN sym*, *of E n g i j*], *assumption*+)
apply *simp*
apply (*subst isom-gch-units-transpTr1*[*THEN sym*, *of E n h i j*], *assumption*+)
apply (*rule isom-gch-unitsTr4* [*of g i h j E*], *assumption*+)
apply *simp*
apply *simp*
apply (*subst isom-gch-units-transpTr8-1* [*of E n g i j*], *assumption*+)
apply (*subst isom-gch-units-transpTr8-1* [*of E n h j i*], *assumption*+)
apply (*rule isom-gch-unitsTr4* [*of g i h j E*], *assumption*+)
apply (*rule isom-gch-units-transpTr7* [*of E i n j g h*],
assumption+)
apply (*subst isom-gch-units-transpTr1* [*THEN sym*, *of E n g j i*], *assumption*+)
apply *simp*
apply (*subst card-insert-disjoint*)
apply (*rule isom-gch-units-transpTr4*, *assumption*+)
apply *simp*
apply (*subst isom-gch-units-transpTr1* [*THEN sym*, *of E n h i j*],
assumption+)
apply (*rule isom-gch-unitsTr4* [*of g i h j E*], *assumption*+)
apply *simp*
apply (*subst card-insert-disjoint*)
apply (*rule isom-gch-units-transpTr4*, *assumption*+)
apply *simp*
apply (*insert Nset-2* [*of j i*])
apply *simp*
apply (*case-tac g j* ≅ *E*)
apply (*subst isom-gch-units-transpTr8-1* [*of E n g j i*], *assumption*+)
apply (*subst isom-gch-units-transpTr8-1* [*of E n h i j*], *assumption*+)
apply (*rule isom-gch-unitsTr4* [*of g j h i E*], *assumption*+)
apply (*rule isom-gch-units-transpTr7* [*of E j n i g h*], *assumption*+)
apply (*subst isom-gch-units-transpTr1* [*THEN sym*, *of E n g i j*],
assumption+)
apply *simp*
apply (*subst isom-gch-units-transpTr1* [*THEN sym*, *of E n h j i*],
assumption+)
apply (*rule isom-gch-unitsTr4* [*of g j h i E*], *assumption*+)
apply *simp* **apply** *simp*
apply (*subst isom-gch-units-transpTr8-2* [*of E n g i j*], *assumption*+)
apply (*subst isom-gch-units-transpTr8-2* [*of E n h i j*], *assumption*+)

apply (*rule isom-gch-units-transpTr7*[*of E i n j g h*], *assumption*+)
apply (*rule isom-gch-units-transpTr7*[*of E j n i g h*], *assumption*+)
apply *simp*
done

lemma *TR-isom-gch-units*:⟦*Ugp E*; *Gchain n f*; $i \leq n$; $j \leq n$; $i < j$⟧ ⟹
$card\ \{k.\ k \leq n \wedge f\ k \cong E\} = card\ \{k.\ k \leq n \wedge$
$(cmp\ f\ (transpos\ i\ j))\ k \cong E\}$
apply (*frule isom-gch-transp* [*of n f i j*], *assumption*+)

apply (*rule isom-gch-units-transp* [*of E n f - i j*], *assumption*+)
apply (*simp add*:*Gchain-def*)
apply (*rule allI*, *rule impI*)
apply (*simp add*:*cmp-def*)
apply (*cut-tac* $l = l$ **in** *transpos-mem* [*of i n j*],
frule-tac $x = i$ **and** $y = j$ **and** $z = n$ **in** *less-le-trans*, *assumption*+,
simp)
apply *simp*+
done

lemma *TR-isom-gch-units-1*:⟦*Ugp E*; *Gchain n f*; $i \leq n$; $j \leq n$; $i < j$⟧ ⟹
$card\ \{k.\ k \leq n \wedge f\ k \cong E\} = card\ \{k.\ k \leq n \wedge f\ (transpos\ i\ j\ k) \cong E\}$
apply (*frule TR-isom-gch-units* [*of E n f i j*], *assumption*+)
apply (*simp add*:*cmp-def*)
done

lemma *isom-tgch-unitsTr0-1*:⟦*Ugp E*; *Gchain* (*Suc n*) *g*; *g* (*Suc n*) ≅ *E*⟧ ⟹
$\{i.\ i \leq (Suc\ n) \wedge g\ i \cong E\} = insert\ (Suc\ n)\ \{i.\ i \leq n \wedge g\ i \cong E\}$
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*simp add*:*CollectI*)
apply (*case-tac* $x = Suc\ n$) **apply** *simp*
apply (*erule conjE*) **apply** *simp*
apply (*rule subsetI*)
apply (*simp add*:*CollectI*)
apply (*erule disjE*) **apply** *simp*
apply *simp*
done

lemma *isom-tgch-unitsTr0-2*:*Ugp E* ⟹ *finite* ($\{i.\ i \leq (n::nat) \wedge g\ i \cong E\}$)
apply (*rule finite-subset*[*of* $\{i.\ i \leq n \wedge g\ i \cong E\}$ $\{i.\ i \leq n\}$])
apply (*rule subsetI*, *simp*+)
done

lemma *isom-tgch-unitsTr0-3*:⟦*Ugp E*; *Gchain* (*Suc n*) *g*; ¬ *g* (*Suc n*) ≅ *E*⟧
⟹ $\{i.\ i \leq (Suc\ n) \wedge g\ i \cong E\} = \{i.\ i \leq n \wedge g\ i \cong E\}$
apply (*rule equalityI*)
apply (*rule subsetI*, *simp add*:*CollectI*)
apply (*case-tac* $x = Suc\ n$, *simp*, *erule conjE*)

apply *arith*
apply (*rule subsetI*, *simp add*:*CollectI*)
done

lemma *isom-tgch-unitsTr0*:⟦*Ugp E*;
card {*i*. *i* ≤ *n* ∧ *g i* ≅ *E*} = *card* {*i*. *i* ≤ *n* ∧ *h i* ≅ *E*};
Gchain (*Suc n*) *g* ∧ *Gchain* (*Suc n*) *h* ∧ *Gch-bridge* (*Suc n*) *g h f*;
f (*Suc n*) = *Suc n*⟧ ⟹
card {*i*. *i* ≤ (*Suc n*) ∧ *g i* ≅ *E*} =
card {*i*. *i* ≤ (*Suc n*) ∧ *h i* ≅ *E*}
apply (*erule conjE*)+
apply (*frule isom-gch-units-transpTr6* [*of Suc n g Suc n*])
apply (*simp add*:*n-in-Nsetn*)
apply (*frule isom-gch-units-transpTr6* [*of Suc n h Suc n*])
apply (*simp add*:*n-in-Nsetn*)
apply (*simp add*:*Gch-bridge-def isom-Gchains-def*)
apply (*erule conjE*)+
apply (*rotate-tac* −*1*,
frule-tac a = *Suc n* **in** *forall-spec*,
thin-tac ∀ *i*≤*Suc n*. *g i* ≅ *h* (*f i*), *simp*+,
thin-tac ∀ *i*≤*Suc n*. *g i* ≅ *h* (*f i*))
apply (*case-tac g* (*Suc n*) ≅ *E*)
apply (*subst isom-tgch-unitsTr0-1* [*of E n g*], *assumption*+)
apply (*subst isom-tgch-unitsTr0-1* [*of E n h*], *assumption*+)
apply (*frule isom-gch-unitsTr4* [*of g* (*Suc n*) *h* (*Suc n*) *E*], *assumption*+)
apply (*subst card-insert-disjoint*)
apply (*rule finite-subset*[*of* {*i*. *i* ≤ *n* ∧ *g i* ≅ *E*} {*i*. *i* ≤ *n*}])
apply (*rule subsetI*, *simp*) **apply** (*simp*)
apply *simp*
apply (*subst card-insert-disjoint*)
apply (*rule finite-subset*[*of* {*i*. *i* ≤ *n* ∧ *h i* ≅ *E*} {*i*. *i* ≤ *n*}])
apply (*rule subsetI*, *simp*) **apply** *simp* **apply** *simp*
apply *simp*
apply (*cut-tac isom-gch-units-transpTr7*[*of E Suc n Suc n Suc n g h*])
apply (*subgoal-tac* {*i*. *i* ≤ *Suc n* ∧ *g i* ≅ *E*} = {*i*. *i* ≤ *n* ∧ *g i* ≅ *E*},
subgoal-tac {*i*. *i* ≤ *Suc n* ∧ *h i* ≅ *E*} = {*i*. *i* ≤ *n* ∧ *h i* ≅ *E*},
simp)
apply (*rule equalityI*, *rule subsetI*, *simp*,
erule conjE, *case-tac x* = *Suc n*, *simp*,
frule-tac x = *x* **and** *y* = *Suc n* **in** *le-imp-less-or-eq*,
thin-tac x ≤ *Suc n*, *simp*,
rule subsetI, *simp*)
apply (*rule equalityI*, *rule subsetI*, *simp*,
erule conjE, *case-tac x* = *Suc n*, *simp*,
frule-tac m = *x* **and** *n* = *Suc n* **in** *noteq-le-less*, *assumption*+,
simp,
rule subsetI, *simp*)
apply *simp*+
done

lemma *isom-gch-unitsTr1-1*: ⟦*Ugp E*; *Gchain* (*Suc n*) *g* ∧ *Gchain* (*Suc n*) *h*
∧ *Gch-bridge* (*Suc n*) *g h f*; *f* (*Suc n*) = *Suc n*⟧ ⟹
Gchain n g ∧ *Gchain n h* ∧ *Gch-bridge n g h f*
apply (*erule conjE*)+
apply (*frule Gchain-pre* [*of n g*])
apply (*frule Gchain-pre* [*of n h*])
apply *simp*
apply (*simp add*:*Gch-bridge-def*) **apply** (*erule conjE*)+
apply (*rule conjI*)
apply (*rule Nset-injTr2*, *assumption*+)
apply (*rule conjI*)
apply (*rule Nset-injTr1*, *assumption*+)
apply (*simp add*:*isom-Gchains-def*)
done

lemma *isom-gch-unitsTr1-2*:⟦*Ugp E*; *f* (*Suc n*) ≠ *Suc n*; *inj-on f* {*i*. *i*≤(*Suc n*)};
∀ *l* ≤ (*Suc n*). *f l* ≤ (*Suc n*)⟧ ⟹
(*cmp* (*transpos* (*f* (*Suc n*)) (*Suc n*)) *f*) (*Suc n*) = *Suc n*
apply (*simp add*:*cmp-def*)
apply (*cut-tac n-in-Nsetn*[*of Suc n*], *simp*)
apply (*frule-tac a* = *Suc n* **in** *forall-spec*,
simp,
thin-tac ∀ *l*≤*Suc n*. *f l* ≤ *Suc n*)
apply (*rule transpos-ij-1*, *assumption*+, *simp*+)
done

lemma *isom-gch-unitsTr1-3*:⟦*Ugp E*; *f* (*Suc n*) ≠ *Suc n*;
∀ *l* ≤ (*Suc n*). *f l* ≤ (*Suc n*); *inj-on f* {*i*. *i* ≤ (*Suc n*)}⟧ ⟹
inj-on (*cmp* (*transpos* (*f* (*Suc n*)) (*Suc n*)) *f*) {*i*. *i* ≤ (*Suc n*)}
apply (*cut-tac n-in-Nsetn*[*of Suc n*], *simp*)
apply (*frule-tac a* = *Suc n* **in** *forall-spec*, *simp*)
apply (*frule transpos-hom* [*of f* (*Suc n*) *Suc n Suc n*], *simp*+)
thm *transpos-inj*
apply (*cut-tac transpos-inj* [*of f* (*Suc n*) *Suc n Suc n*])
apply (*cut-tac cmp-inj-1* [*of f* {*i*. *i* ≤ (*Suc n*)} {*i*. *i* ≤ (*Suc n*)}
transpos (*f* (*Suc n*)) (*Suc n*) {*i*. *i* ≤ (*Suc n*)}])
apply *simp*+
done

lemma *isom-gch-unitsTr1-4*:⟦*Ugp E*; *f* (*Suc n*) ≠ *Suc n*; *inj-on f* {*i*. *i*≤(*Suc n*)};
∀ *l* ≤ (*Suc n*). *f l* ≤ (*Suc n*)⟧ ⟹
inj-on (*cmp* (*transpos* (*f* (*Suc n*)) (*Suc n*)) *f*) {*i*. *i* ≤ *n*}
apply (*frule isom-gch-unitsTr1-3* [*of E f n*], *assumption*+)
apply (*frule isom-gch-unitsTr1-2* [*of E f n*], *assumption*+)
apply (*rule Nset-injTr1* [*of n* (*cmp* (*transpos* (*f* (*Suc n*)) (*Suc n*)) *f*)])
apply (*rule allI*, *rule impI*)
apply (*simp add*:*cmp-def*)
apply (*cut-tac l* = *f l* **in** *transpos-mem*[*of f* (*Suc n*) *Suc n Suc n*],

```
    simp+)
done

lemma isom-gch-unitsTr1-5:⟦Ugp E; Gchain (Suc n) g ∧ Gchain (Suc n) h ∧
    Gch-bridge (Suc n) g h f; f (Suc n) ≠ Suc n ⟧ ⟹
    Gchain n g ∧ Gchain n (cmp h (transpos (f (Suc n)) (Suc n))) ∧
    Gch-bridge n g (cmp h (transpos (f (Suc n)) (Suc n)))
                        (cmp (transpos (f (Suc n)) (Suc n)) f)
apply (erule conjE)+
apply (simp add:Gchain-pre [of n g])
apply (rule conjI)
apply (simp add:Gchain-def) apply (rule allI, rule impI)
apply (simp add:Gch-bridge-def) apply (frule conjunct1)
apply (fold Gch-bridge-def)
apply (cut-tac n-in-Nsetn[of Suc n])
apply (cut-tac l = l in transpos-mem [of f (Suc n) Suc n Suc n])
    apply simp+
apply (simp add:cmp-def)

apply (simp add:Gch-bridge-def)
apply (erule conjE)+
apply (rule conjI)
apply (cut-tac n-in-Nsetn[of Suc n])
apply (rule allI, rule impI, simp add:cmp-def)
apply (frule isom-gch-unitsTr1-2 [of E f n], assumption+)
apply (frule isom-gch-unitsTr1-3 [of E f n], assumption+)
apply (cut-tac l = f l in transpos-mem[of f (Suc n) Suc n Suc n])
 apply simp+
 apply (simp add:inj-on-def[of cmp (transpos (f (Suc n)) (Suc n)) f])
 apply (rotate-tac −2,
       frule-tac a = Suc n in forall-spec, simp) apply (
       thin-tac ∀ x≤Suc n.
          ∀ y≤Suc n.
             cmp (transpos (f (Suc n)) (Suc n)) f x =
             cmp (transpos (f (Suc n)) (Suc n)) f y ⟶
             x = y) apply (
       rotate-tac −1,
       frule-tac x = l in spec) apply (
       thin-tac ∀ y≤Suc n.
          cmp (transpos (f (Suc n)) (Suc n)) f (Suc n) =
          cmp (transpos (f (Suc n)) (Suc n)) f y ⟶
          Suc n = y)
apply (metis Nfunc-iNJTr comp-transpos-1 le-SucE le-SucI le-refl less-Suc-eq-le
transpos-ij-2)
apply (simp add:isom-gch-unitsTr1-4)
apply (simp add:isom-Gchains-def[of n])
apply (rule allI, rule impI)
apply (simp add:cmp-def)
apply (cut-tac l = f i in transpos-mem[of f (Suc n) Suc n Suc n])
```

apply *simp+*
apply (*cut-tac k = f i* **in** *cmp-transpos1* [*of f* (*Suc n*) *Suc n Suc n*])
apply *simp+*
apply (*simp add*:*cmp-def*)
apply (*thin-tac transpos* (*f* (*Suc n*)) (*Suc n*) (*transpos* (*f* (*Suc n*)) (*Suc n*) (*f i*)) = *f i*)
apply (*simp add*:*isom-Gchains-def*)
done

lemma *isom-gch-unitsTr1-6*:⟦*Ugp E*; *f* (*Suc n*) ≠ *Suc n*; *Gchain* (*Suc n*) *g* ∧ *Gchain* (*Suc n*) *h* ∧ *Gch-bridge* (*Suc n*) *g h f*⟧ ⟹ *Gchain* (*Suc n*) *g* ∧ *Gchain* (*Suc n*) (*cmp h* (*transpos* (*f* (*Suc n*)) (*Suc n*))) ∧ *Gch-bridge* (*Suc n*) *g* (*cmp h* (*transpos* (*f* (*Suc n*)) (*Suc n*))) (*cmp* (*transpos* (*f* (*Suc n*)) (*Suc n*)) *f*)
apply (*erule conjE*)+
apply *simp*
apply (*simp add*:*Gch-bridge-def*, *frule conjunct1*)
apply (*frule conjunct2*, *fold Gch-bridge-def*, *erule conjE*)
apply (*rule conjI*)
apply (*thin-tac Gchain* (*Suc n*) *g*)
apply (*simp add*:*Gchain-def*, *rule allI*, *rule impI*)
apply (*simp add*:*cmp-def*)
apply (*cut-tac n-in-Nsetn*[*of Suc n*])
apply (*cut-tac l = l* **in** *transpos-mem* [*of f* (*Suc n*) *Suc n Suc n*], *simp+*)
apply (*simp add*:*Gch-bridge-def*)
apply (*cut-tac n-in-Nsetn*[*of Suc n*])
apply (*rule conjI*)
apply (*rule allI*, *rule impI*)
apply (*simp add*:*cmp-def*)
apply (*rule-tac l = f l* **in** *transpos-mem* [*of f* (*Suc n*) *Suc n Suc n*], *simp+*)
apply (*rule conjI*)
apply (*rule isom-gch-unitsTr1-3* [*of E f n*], *assumption+*)
apply (*simp add*:*isom-Gchains-def*, *rule allI*, *rule impI*)
apply (*simp add*:*cmp-def*)
apply (*cut-tac k = f i* **in** *cmp-transpos1* [*of f* (*Suc n*) *Suc n Suc n*], *simp+*)
apply (*simp add*:*cmp-def*)
done

lemma *isom-gch-unitsTr1-7-0*:⟦*Gchain* (*Suc n*) *h*; *k* ≠ *Suc n*; *k* ≤ (*Suc n*)⟧ ⟹ *Gchain* (*Suc n*) (*cmp h* (*transpos k* (*Suc n*)))
apply (*simp add*:*Gchain-def*)
apply (*rule allI*, *rule impI*)
apply (*simp add*:*cmp-def*)
apply (*insert n-in-Nsetn* [*of Suc n*])
apply (*cut-tac l = l* **in** *transpos-mem* [*of k Suc n Suc n*])
apply *simp+*

done

lemma *isom-gch-unitsTr1-7-1*:⟦*Ugp E*; *Gchain* (*Suc n*) *h*; $k \neq Suc\ n$; $k \leq (Suc\ n)$⟧
$\Longrightarrow \{i.\ i \leq (Suc\ n) \wedge cmp\ h\ (transpos\ k\ (Suc\ n))\ i \cong E\} - \{k\ ,\ Suc\ n\} =$
$\{i.\ i \leq (Suc\ n) \wedge h\ i \cong E\} - \{k\ ,\ Suc\ n\}$
apply (*cut-tac n-in-Nsetn*[*of Suc n*])
apply *auto*
apply (*frule-tac* $x = x$ **in** *transpos-id-1* [*of k Suc n Suc n*], *simp*+)
apply (*simp add*:*cmp-def*)
apply (*simp add*:*cmp-def*)
apply (*cut-tac* $x = x$ **in** *transpos-id-1* [*of k Suc n Suc n*], *simp*+)
done

lemma *isom-gch-unitsTr1-7-2*:⟦*Ugp E*; *Gchain* (*Suc n*) *h*; $k \neq Suc\ n$;
$k \leq (Suc\ n)$; $h\ (Suc\ n) \cong E$⟧ $\Longrightarrow$
$cmp\ h\ (transpos\ k\ (Suc\ n))\ k \cong E$
apply (*simp add*:*cmp-def*)
apply (*subst transpos-ij-1* [*of k Suc n Suc n*], *simp*+)
done

lemma *isom-gch-unitsTr1-7-3*:⟦*Ugp E*; *Gchain* (*Suc n*) *h*; $k \neq Suc\ n$;
$k \leq (Suc\ n)$; $h\ k \cong E$⟧ $\Longrightarrow$ $cmp\ h\ (transpos\ k\ (Suc\ n))\ (Suc\ n) \cong E$
apply (*simp add*:*cmp-def*)
apply (*subst transpos-ij-2* [*of k Suc n Suc n*], *assumption*+)
apply *simp* **apply** *assumption*+
done

lemma *isom-gch-unitsTr1-7-4*:⟦*Ugp E*; *Gchain* (*Suc n*) *h*; $k \neq Suc\ n$;
$k \leq (Suc\ n)$; $\neg\ h\ (Suc\ n) \cong E$⟧ $\Longrightarrow$
$\neg\ \ cmp\ h\ (transpos\ k\ (Suc\ n))\ k \cong E$
apply (*rule contrapos-pp*, *simp*+)
apply (*simp add*:*cmp-def*)
apply (*insert n-in-Nsetn* [*of Suc n*])
apply (*simp add*: *transpos-ij-1* [*of k Suc n Suc n*])
done

lemma *isom-gch-unitsTr1-7-5*:⟦*Ugp E*; *Gchain* (*Suc n*) *h*; $k \neq Suc\ n$;
$k \leq (Suc\ n)$; $\neg\ h\ k \cong E$⟧ $\Longrightarrow$
$\neg\ \ cmp\ h\ (transpos\ k\ (Suc\ n))\ (Suc\ n) \cong E$
apply (*rule contrapos-pp*, *simp*+)
apply (*simp add*:*cmp-def*)
apply (*insert n-in-Nsetn* [*of Suc n*])
apply (*simp add*:*transpos-ij-2* [*of k Suc n Suc n*])
done

lemma *isom-gch-unitsTr1-7-6*:⟦*Ugp E*; *Gchain* (*Suc n*) *h*; $k \neq Suc\ n$;
$k \leq (Suc\ n)$; $h\ (Suc\ n) \cong E$; $h\ k \cong E$⟧ $\Longrightarrow$
$\{i.\ i \leq (Suc\ n) \wedge\ \ h\ i \cong E\} =$

insert k (insert (Suc n) ({i. i ≤ (Suc n) ∧ h i ≅ E} − {k, Suc n}))
apply *auto*
done

lemma *isom-gch-unitsTr1-7-7*:⟦*Ugp E*; *Gchain (Suc n) h*; *k ≠ Suc n*;
k ≤ (Suc n); *h (Suc n) ≅ E*; *¬ h k ≅ E*⟧ ⟹
{*i. i ≤ (Suc n) ∧ h i ≅ E*} =
insert (Suc n) ({i. i ≤ (Suc n) ∧ h i ≅ E} − {k, Suc n})
apply *auto*
done

lemma *isom-gch-unitsTr1-7-8*:⟦*Ugp E*; *Gchain (Suc n) h*; *k ≠ Suc n*;
k ≤ (Suc n); *¬ h (Suc n) ≅ E*; *h k ≅ E*⟧ ⟹
{*i. i ≤ (Suc n) ∧ h i ≅ E*} =
insert k ({i. i ≤ (Suc n) ∧ h i ≅ E} − {k, Suc n})
apply *auto*
done

lemma *isom-gch-unitsTr1-7-9*:⟦*Ugp E*; *Gchain (Suc n) h*; *k ≠ Suc n*;
k ≤ (Suc n); *¬ h (Suc n) ≅ E*; *¬ h k ≅ E*⟧ ⟹
{*i. i ≤ (Suc n) ∧ h i ≅ E*} =
{*i. i ≤ (Suc n) ∧ h i ≅ E*} − {*k*, *Suc n*}
apply *auto*
done

lemma *isom-gch-unitsTr1-7*:⟦*Ugp E*; *Gchain (Suc n) h*; *k ≠ Suc n*;
k ≤ (Suc n)⟧ ⟹ *card* {*i. i ≤ (Suc n) ∧
cmp h (transpos k (Suc n)) i ≅ E*} = *card* {*i. i ≤ (Suc n) ∧ h i ≅ E*}
apply (*cut-tac finite-Collect-le-nat*[*of Suc n*])
apply (*frule isom-gch-unitsTr1-7-1* [*of E n h k*], *assumption*+)
apply (*cut-tac n-in-Nsetn*[*of Suc n*])
apply (*case-tac h (Suc n) ≅ E*)
apply (*case-tac h k ≅ E*)
apply (*subst isom-gch-unitsTr1-7-6* [*of E n h k*], *assumption*+)
apply (*frule isom-gch-unitsTr1-7-2* [*of E n h k*], *assumption*+)
apply (*frule isom-gch-unitsTr1-7-3* [*of E n h k*], *assumption*+)
apply (*frule isom-gch-unitsTr1-7-0* [*of n h k*], *assumption*+)
apply (*subst isom-gch-unitsTr1-7-6* [*of E n cmp h (transpos k (Suc n)) k*], *assumption*+)
apply (*subst card-insert-disjoint*)
apply (*rule finite-subset*[*of insert (Suc n)*
({*i. i ≤ (Suc n) ∧ cmp h (transpos k (Suc n)) i ≅ E*} −
{*k*, *Suc n*}) {*i. i ≤ (Suc n)*}])
apply (*rule subsetI*, *simp*)
apply (*erule disjE*)
apply *simp* **apply** *simp* **apply** *assumption*
apply *simp*

apply (*subst card-insert-disjoint*)+

apply (*rule finite-subset*[*of* $\{i.\ i \leq (Suc\ n) \wedge cmp\ h\ (transpos\ k\ (Suc\ n))\ i \cong E\} - \{k, Suc\ n\}\ \{i.\ i \leq (Suc\ n)\}$])
apply (*rule subsetI*, *simp*) **apply** *assumption*
apply *simp*

apply (*subst card-insert-disjoint*)+
apply (*rule finite-subset*[*of insert* $(Suc\ n)\ (\{i.\ i \leq (Suc\ n) \wedge h\ i \cong E\} - \{k, Suc\ n\})\ \{i.\ i \leq (Suc\ n)\}$])
apply (*rule subsetI*, *simp*) **apply** (*erule disjE*, *simp add*:*n-in-Nsetn*) **apply** *simp*
apply *assumption* **apply** *simp*

apply (*subst card-insert-disjoint*)
apply (*rule finite-subset*[*of* $\{i.\ i \leq (Suc\ n) \wedge h\ i \cong E\} - \{k, Suc\ n\}\ \{i.\ i \leq (Suc\ n)\}$])
apply (*rule subsetI*, *simp*) **apply** *assumption* **apply** *simp*
apply *simp*

apply (*subst isom-gch-unitsTr1-7-7*[*of E n h k*], *assumption*+)
apply (*frule isom-gch-unitsTr1-7-0*[*of n h k*], *assumption*+)
apply (*subst isom-gch-unitsTr1-7-8*[*of E n cmp h* (*transpos k* (*Suc n*)) *k*], *assumption*+)
apply (*subst cmp-def*)
apply (*subst transpos-ij-2*[*of k Suc n Suc n*], *assumption*+, *simp*+)
apply (*simp add*:*cmp-def*, *simp add*:*transpos-ij-1*)

apply (*subst card-insert-disjoint*)
apply (*rule finite-subset*[*of* $\{i.\ i \leq (Suc\ n) \wedge cmp\ h\ (transpos\ k\ (Suc\ n))\ i \cong E\} - \{k, Suc\ n\}\ \{i.\ i \leq (Suc\ n)\}$])
apply (*rule subsetI*, *simp*) **apply** *assumption* **apply** *simp*
apply (*subst card-insert-disjoint*)
apply (*rule finite-subset*[*of* $\{i.\ i \leq (Suc\ n) \wedge h\ i \cong E\} - \{k, Suc\ n\}\ \{i.\ i \leq (Suc\ n)\}$])
apply (*rule subsetI*, *simp*) **apply** *assumption* **apply** *simp*
apply *simp*

apply (*case-tac h k* $\cong$ *E*)
apply (*subst isom-gch-unitsTr1-7-8* [*of E n h k*], *assumption*+)
apply (*frule isom-gch-unitsTr1-7-3* [*of E n h k*], *assumption*+)
apply (*frule isom-gch-unitsTr1-7-4* [*of E n h k*], *assumption*+)
apply (*frule isom-gch-unitsTr1-7-0* [*of n h k*], *assumption*+)
apply (*subst isom-gch-unitsTr1-7-7* [*of E n cmp h* (*transpos k* (*Suc n*)) *k*], *assumption*+)
apply (*subst card-insert-disjoint*)
apply (*rule isom-gch-units-transpTr4*, *assumption*+) **apply** *simp*
apply (*subst card-insert-disjoint*)
apply (*rule isom-gch-units-transpTr4*, *assumption*+) **apply** *simp* **apply** *simp*
apply (*subst isom-gch-unitsTr1-7-9* [*of E n h k*], *assumption*+)
apply (*frule-tac isom-gch-unitsTr1-7-4* [*of E n h k*], *assumption*+)
apply (*frule-tac isom-gch-unitsTr1-7-5* [*of E n h k*], *assumption*+)

apply (*frule isom-gch-unitsTr1-7-0* [*of n h k*], *assumption*+)
apply (*subst isom-gch-unitsTr1-7-9* [*of E n cmp h* (*transpos k* (*Suc n*)) *k*], *assumption*+) **apply** *simp*
done

lemma *isom-gch-unitsTr1*:*Ugp E* $\Longrightarrow \forall g.\ \forall h.\ \forall f.$ *Gchain n g* $\wedge$
Gchain n h $\wedge$ *Gch-bridge n g h f* $\longrightarrow$ *card* $\{i.\ i \leq n \wedge g\ i \cong E\} =$
card $\{i.\ i \leq n \wedge h\ i \cong E\}$
apply (*induct-tac n*)
apply (*clarify*)
apply (*simp add*:*Gch-bridge-def isom-Gchains-def Collect-conv-if*)
apply *rule*
apply (*simp add*:*Gchain-def*)
apply(*metis isom-gch-unitsTr4*)
apply (*simp add*:*Gchain-def*)
apply (*metis Ugp-def isomTr2*)

apply (*rule allI*)+ **apply** (*rule impI*)

apply (*case-tac f* (*Suc n*) = *Suc n*)
apply (*subgoal-tac card* $\{i.\ i \leq n \wedge g\ i \cong E\}$ = *card* $\{i.\ i \leq n \wedge h\ i \cong E\}$)
apply (*thin-tac* $\forall g\ h\ f.$
Gchain n g $\wedge$ *Gchain n h* $\wedge$ *Gch-bridge n g h f* $\longrightarrow$
card $\{i.\ i \leq n \wedge g\ i \cong E\} =$
card $\{i.\ i \leq n \wedge h\ i \cong E\}$)
apply (*rule isom-tgch-unitsTr0*, *assumption*+)
apply (*frule-tac n* = *n* **and** *g* = *g* **and** *h* = *h* **and** *f* = *f* **in**
isom-gch-unitsTr1-1 [*of E*], *assumption*+)
apply (*rotate-tac* −*1*)
apply (*thin-tac Gchain* (*Suc n*) *g* $\wedge$ *Gchain* (*Suc n*) *h* $\wedge$ *Gch-bridge* (*Suc n*) *g h f*)
apply *blast*

apply (*frule-tac n* = *n* **and** *g* = *g* **and** *h* = *h* **and** *f* = *f* **in** *isom-gch-unitsTr1-5* [*of E*], *assumption*+)
apply (*subgoal-tac card* $\{i.\ i \leq n \wedge g\ i \cong E\}$ = *card* $\{i.\ i \leq n \wedge$
(*cmp h* (*transpos* (*f* (*Suc n*)) (*Suc n*))) *i* $\cong E\}$)
prefer *2* **apply** *blast*
apply (*thin-tac* $\forall g\ h\ f.$
Gchain n g $\wedge$ *Gchain n h* $\wedge$ *Gch-bridge n g h f* $\longrightarrow$
card $\{i.\ i \leq n \wedge g\ i \cong E\} =$
card $\{i.\ i \leq n \wedge h\ i \cong E\}$)
apply (*thin-tac Gchain n g* $\wedge$
Gchain n (*cmp h* (*transpos* (*f* (*Suc n*)) (*Suc n*))) $\wedge$
Gch-bridge n g (*cmp h* (*transpos* (*f* (*Suc n*)) (*Suc n*)))
(*cmp* (*transpos* (*f* (*Suc n*)) (*Suc n*)) *f*))
apply (*subgoal-tac cmp* (*transpos* (*f* (*Suc n*)) (*Suc n*)) *f* (*Suc n*) = *Suc n*)
apply (*frule-tac n* = *n* **and** *g* = *g* **and** *h* = *cmp h* (*transpos* (*f* (*Suc n*)) (*Suc n*)) **and** *f* = *cmp* (*transpos* (*f* (*Suc n*)) (*Suc n*)) *f* **in** *isom-tgch-unitsTr0* [*of E*],

assumption+)
apply (*rule isom-gch-unitsTr1-6*, *assumption*+)
apply (*thin-tac card* {*i*. *i* ≤ *n* ∧ *g i* ≅ *E*} = *card* {*i*. *i* ≤ *n* ∧
cmp h (*transpos* (*f* (*Suc n*)) (*Suc n*)) *i* ≅ *E*})
prefer *2*
apply (*erule conjE*)+
apply (*simp add*:*Gch-bridge-def*) **apply** (*erule conjE*)+
apply (*rule isom-gch-unitsTr1-2*, *assumption*+)
apply *simp*
apply (*erule conjE*)+
apply (*rule isom-gch-unitsTr1-7*, *assumption*+)
apply (*simp add*:*Gch-bridge-def*)
done

lemma *isom-gch-units*:⟦*Ugp E*; *Gchain n g*; *Gchain n h*; *Gch-bridge n g h f*⟧ ⟹

card {*i*. *i* ≤ *n* ∧ *g i* ≅ *E*} = *card* {*i*. *i* ≤ *n* ∧ *h i* ≅ *E*}
apply (*simp add*:*isom-gch-unitsTr1*)
done

lemma *isom-gch-units-1*:⟦*Ugp E*; *Gchain n g*; *Gchain n h*; ∃*f*. *Gch-bridge n g h f*⟧
⟹ *card* {*i*. *i* ≤ *n* ∧ *g i* ≅ *E*} = *card* {*i*. *i* ≤ *n* ∧ *h i* ≅ *E*}
apply (*auto del*:*equalityI*)
apply (*simp add*:*isom-gch-units*)
done

3.17 Jordan Hoelder theorem

3.17.1 *Rfn-tools*. Tools to treat refinement of a cmpser, rtos.

lemma *rfn-tool1*:⟦ *0* < (*r*::*nat*); (*k*::*nat*) = *i* ∗ *r* + *j*; *j* < *r* ⟧
⟹ (*k div r*) = *i*
proof −
assume *p1*: *0* < *r* **and** *p2*: *k* = *i* ∗ *r* + *j* **and** *p3*: *j* < *r*
from *p1* **and** *p2* **have** *q1*: (*j* + *i* ∗ *r*) *div r* = *i* + *j div r*
apply (*simp add*:*div-mult-self1* [*of r j i*]) **done**
from *p1* **and** *p3* **have** *q2*: *j div r* = *0*
apply (*simp add*:*div-if*) **done**
from *q1* **and** *q2* **have** *q3*:(*j* + *i* ∗ *r*) *div r* = *i*
apply *simp* **done**
from *q3* **have** *q4*: (*i* ∗ *r* + *j*) *div r* = *i* **apply** (*simp add*:*add.commute*)
done
from *p2* **and** *q4* **show** *?thesis*
apply *simp*
done
qed

lemma *pos-mult-pos*:⟦ *0* < (*r*::*nat*); *0* < *s*⟧ ⟹ *0* < *r* ∗ *s*

by *simp*

lemma *rfn-tool1-1*:⟦ *0* < (*r*::*nat*); *j* < *r* ⟧
⟹ (*i* * *r* + *j*) *div r* = *i*
apply (*rule rfn-tool1* [*of r i* * *r* + *j i j*], *assumption*+)
apply *simp*+
done

lemma *rfn-tool2*:(*a*::*nat*) < *s* ⟹ *a* ≤ *s* − *Suc 0*
apply (*rule less-le-diff*) **apply** *simp*+
done

lemma *rfn-tool3*:(*0*::*nat*) ≤ *m* ⟹ (*m* + *n*) − *n* = *m*
apply *auto*
done

lemma *rfn-tool11*:⟦*0* < *b*; (*a*::*nat*) ≤ *b* − *Suc 0*⟧ ⟹ *a* < *b*
apply (*frule le-less-trans* [*of a b* − *Suc 0 b*])
apply *simp*+
done

lemma *rfn-tool12*:⟦*0* < (*s*::*nat*); (*i*::*nat*) *mod s* = *s* − *1* ⟧ ⟹
Suc (*i div s*) = (*Suc i*) *div s*
proof −
assume *p1*:*0* < *s* **and** *p2*:*i mod s* = *s* − *1*
have *q1*:*i div s* * *s* + *i mod s* = *i*
apply (*insert mod-div-equality* [*of i s*])
apply *simp* **done**
have *q2*:*Suc* (*i div s* * *s* + *i mod s*) = *i div s* * *s* + *Suc* (*i mod s*)
apply (*insert add-Suc-right* [*THEN sym*, *of i div s* * *s i mod s*])
apply *assumption* **done**
from *p1* **and** *p2* **and** *q2* **have** *q3*:*Suc* (*i div s* * *s* + *i mod s*) = *i div s* * *s* + *s*
apply *simp* **done**
from *q3* **have** *q4*:*Suc* (*i div s* * *s* + *i mod s*) = *Suc* (*i div s*) * *s*
apply *simp* **done**
from *p1* **and** *q1* **and** *q4* **show** *?thesis*
apply *auto*
done
qed

lemma *rfn-tool12-1*:⟦*0* < (*s*::*nat*); (*l*::*nat*) *mod s* < *s* − *1* ⟧ ⟹
Suc (*l mod s*) = (*Suc l*) *mod s*
apply (*insert mod-div-equality* [*of l s*])
apply (*insert add-Suc-right* [*THEN sym*, *of l div s* * *s l mod s*])
apply (*insert mod-mult-self1* [*of Suc* (*l mod s*) *l div s s*])
apply (*frule Suc-mono* [*of l mod s s* − *1*]) **apply** *simp*
done

lemma *rfn-tool12-2*:⟦*0* < (*s*::*nat*); (*i*::*nat*) *mod s* = *s* − *Suc 0*⟧ ⟹

$(Suc\ i)\ mod\ s = 0$
apply (*insert mod-div-equality* [*THEN sym*, *of i s*])
apply (*insert add-Suc-right* [*THEN sym*, *of i div s * s i mod s*])
apply *simp*
done

lemma *rfn-tool13*:⟦ (*0*::*nat*) < *r*; *a* = *b* ⟧ ⟹ *a mod r* = *b mod r*
apply *simp*
done

lemma *rfn-tool13-1*:⟦ (*0*::*nat*) < *r*; *a* = *b* ⟧ ⟹ *a div r* = *b div r*
apply *simp*
done

lemma *div-Tr1*:⟦ (*0*::*nat*) < *r*; *0* < *s*; *l* ≤ *s* * *r*⟧ ⟹ *l div s* ≤ *r*
apply (*frule-tac m* = *l* **and** *n* = *s* * *r* **and** *k* = *s* **in** *div-le-mono*)
apply *simp*
done

lemma *div-Tr2*:⟦(*0*::*nat*) < *r*; *0* < *s*; *l* < *s* * *r*⟧ ⟹ *l div s* ≤ *r* − *Suc 0*
apply (*rule contrapos-pp*, *simp*+)
apply (*simp add*: *not-less* [*symmetric*, *of l div s r* − *Suc 0*])
apply (*frule Suc-leI* [*of r* − *Suc 0 l div s*])
apply *simp*
apply (*frule less-imp-le* [*of l s* * *r*])
apply (*frule div-le-mono* [*of l s* * *r s*]) **apply** *simp*
apply (*insert mod-div-equality* [*THEN sym*, *of l s*])
apply (*frule sym*) **apply** (*thin-tac r* = *l div s*)
apply *simp* **apply** (*simp add*:*mult.commute* [*of r s*])
done

lemma *div-Tr3*:⟦(*0*::*nat*) < *r*; *0* < *s*; *l* < *s* * *r*⟧ ⟹ *Suc* (*l div s*) ≤ *r*
apply (*frule-tac div-Tr2*[*of r s*], *assumption*+,
cut-tac n1 = *l div s* **and** *m1* = *r* − *Suc 0* **in** *Suc-le-mono*[*THEN sym*])
apply *simp*
done

lemma *div-Tr3-1*:⟦(*0*::*nat*) < *r*; *0* < *s*; *l mod s* = *s* − *1*⟧ ⟹ *Suc l div s* = *Suc* (*l div s*)
apply (*frule rfn-tool12* [*of s l*], *assumption*+)
apply (*rotate-tac* −*1*) **apply** *simp*
done

lemma *div-Tr3-2*:⟦(*0*::*nat*) < *r*; *0* < *s*; *l mod s* < *s* − *1*⟧ ⟹
l div s = *Suc l div s*
apply (*frule Suc-mono* [*of l mod s s* − *1*]) **apply** *simp*
apply (*cut-tac mod-div-equality* [*of l s*])
apply (*cut-tac add-Suc-right* [*THEN sym*, *of l div s * s l mod s*])
apply (*cut-tac zero-less-Suc*[*of l mod s*],

frule less-trans[*of 0 Suc* (*l mod s*) *s*], *assumption*+)
apply (*frule rfn-tool13-1* [*of s Suc* (*l div s* ∗ *s* + *l mod s*) *l div s* ∗ *s* + *Suc* (*l mod s*)], *assumption*+)
apply (*subgoal-tac s* ≠ *0*)
apply (*frule div-mult-self1* [*of s Suc* (*l mod s*) *l div s*])
apply *simp-all*
done

lemma *mod-div-injTr*:⟦(*0*::*nat*) < *r*; *x mod r* = *y mod r*; *x div r* = *y div r*⟧
⟹ *x* = *y*
apply (*cut-tac mod-div-equality*[*of x r*])
apply *simp*
done

definition
rtos :: [*nat*, *nat*] ⇒ (*nat* ⇒ *nat*) **where**
rtos r s i = (*if i* < *r* ∗ *s then* (*i mod s*) ∗ *r* + *i div s else r* ∗ *s*)

lemma *rtos-hom0*:⟦(*0*::*nat*) < *r*; (*0*::*nat*) < *s*; *i* ≤ (*r* ∗ *s* − *Suc 0*)⟧ ⟹
i div s < *r*
apply (*frule div-Tr2* [*of r s i*], *assumption*+)
apply (*simp add*:*mult.commute* [*of r s*])
apply (*rule le-less-trans*, *assumption*+) **apply** *simp*
apply (*rule le-less-trans*, *assumption*+) **apply** *simp*
done

lemma *rtos-hom1*:⟦(*0*::*nat*) < *r*; *0* < *s*; *l* ≤ (*r* ∗ *s* − *Suc 0*)⟧ ⟹
(*rtos r s*) *l* ≤ (*s* ∗ *r* − *Suc 0*)
apply (*simp add*:*rtos-def*)
apply (*frule le-less-trans* [*of l r* ∗ *s* − *Suc 0 r* ∗ *s*])
apply *simp*
apply *simp*
apply (*frule mod-less-divisor* [*of s l*])
apply (*frule less-le-diff* [*of l mod s s*])
apply (*frule-tac i* = *l mod s* **and** *j* = *s* − *Suc 0* **and** *k* = *r* **and** *l* = *r* **in**
mult-le-mono, *simp*)
apply (*frule-tac i* = *l mod s* ∗ *r* **and** *j* = (*s* − *Suc 0*) ∗ *r* **and** *k* = *l div s* **and**
l = *r* − *Suc 0* **in** *add-le-mono*)
apply (*rule div-Tr2*, *assumption*+, *simp add*:*mult.commute*)
apply (*simp add*:*diff-mult-distrib*)
done

lemma *rtos-hom2*:⟦(*0*::*nat*) < *r*; (*0*::*nat*) < *s*; *l* ≤ (*r* ∗ *s* − *Suc 0*)⟧ ⟹
rtos r s l ≤ (*r* ∗ *s* − *Suc 0*)
apply (*insert rtos-hom1* [*of r s*]) **apply** *simp*
apply (*simp add*:*mult.commute*)
done

lemma *rtos-hom3*:⟦(*0*::*nat*) < *r*; *0* < *s*; *i* ≤ (*r* ∗ *s* − *Suc 0*) ⟧ ⟹

```
                (rtos r s i div r) = i mod s
apply (simp add:rtos-def)
 apply (frule le-less-trans [of i r * s - Suc 0 r * s])
 apply simp apply simp
apply (auto simp add: div-mult2-eq [symmetric] mult.commute)
done

lemma rtos-hom3-1:⟦(0::nat) < r; (0::nat) < s; i ≤ (r * s - Suc 0) ⟧ ⟹
  (rtos r s i mod  r) = i div s
apply (simp add:rtos-def)
apply (simp add:rfn-tool11 [of r * s i])
 apply (frule rtos-hom0 [of r s i], assumption+)
 apply (simp add:mem-of-Nset)
done

lemma rtos-hom5:⟦(0::nat) < r; (0::nat) < s; i ≤ (r *s - Suc 0);
i div s = r - Suc 0 ⟧ ⟹ Suc (rtos r s i) div r = Suc (i mod s)
 apply (frule mult-less-mono2[of 0 s r],
       simp only:mult.commute,
       simp only:mult-0-right)
 apply (frule rfn-tool11 [of r * s i], assumption+)
 apply (simp add: rtos-def)
done

lemma rtos-hom7:⟦(0::nat) < r; (0::nat) < s; i ≤ (r * s - Suc 0);
                i div s = r - Suc 0 ⟧ ⟹ Suc (rtos r s i) mod r = 0
apply (frule rtos-hom0 [of r s i], assumption+)
apply (simp add: rtos-def)
apply (frule mult-less-mono2[of 0 s r],
       simp only:mult.commute,
       simp only:mult-0-right)
apply (simp add: rfn-tool11 [of r * s i])
done

lemma rtos-inj:⟦ (0::nat) < r; (0::nat) < s ⟧ ⟹
                inj-on (rtos r s) {i. i ≤ (r * s - Suc 0)}
apply (simp add:inj-on-def)
apply ((rule allI, rule impI)+, rule impI)
 apply (frule-tac i1 = x in rtos-hom3-1[THEN sym, of r s], assumption+,
       frule-tac i1 = x in rtos-hom3[THEN sym, of r s], assumption+,
       frule-tac i = y in rtos-hom3-1[of r s], assumption+,
       frule-tac i = y in rtos-hom3[of r s], assumption+)
 apply simp
 apply (rule-tac x = x and y = y in mod-div-injTr[of s], assumption+)
done

lemma rtos-rs-Tr1:⟦(0::nat) < r; 0 < s ⟧ ⟹ rtos r s (r * s) = r * s
apply (simp add:rtos-def)
done
```

lemma *rtos-rs-Tr2*:⟦(*0*::*nat*) < *r*; *0* < *s* ⟧ ⟹
∀ *l* ≤ (*r* ∗ *s*). *rtos r s l* ≤ (*r* ∗ *s*)
apply (*rule allI*, *rule impI*)
apply (*case-tac l* = *r* ∗ *s*, *simp add*:*rtos-rs-Tr1*)
apply (*frule le-imp-less-or-eq*)
apply (*thin-tac l* ≤ *r* ∗ *s*, *simp*)
apply (*frule mult-less-mono2*[*of 0 s r*],
simp only:*mult.commute*,
simp only:*mult-0-right*)
apply (*frule-tac r* = *r* **and** *s* = *s* **and** *l* = *l* **in** *rtos-hom2*, *assumption*+)
apply (*rule less-le-diff*)
apply *simp*
apply (*metis le-pre-le*)
done

lemma *rtos-rs-Tr3*:⟦(*0*::*nat*) < *r*; *0* < *s* ⟧ ⟹
inj-on (*rtos r s*) {*i*. *i* ≤ (*r* ∗ *s*)}
apply (*frule rtos-inj* [*of r s*], *assumption*+)
apply (*simp add*:*inj-on-def* [*of* - {*i*. *i* ≤ (*r* ∗ *s*)}])
apply ((*rule allI*, *rule impI*)+, *rule impI*)
apply (*case-tac x* = *r* ∗ *s*)
apply (*rule contrapos-pp*, *simp*+)
apply (*frule not-sym*)
apply (*frule mult-less-mono2*[*of 0 s r*],
simp only:*mult.commute*, *simp only*:*mult-0-right*)
apply (*cut-tac x* = *y* **and** *n* = *r* ∗ *s* − *Suc 0* **in** *Nset-pre*, *simp*+)
apply (*frule-tac l* = *y* **in** *rtos-hom1*[*of r s*], *assumption*+)
apply (*simp only*: *rtos-rs-Tr1*)
apply (*frule sym*, *thin-tac r* ∗ *s* = *rtos r s y*, *simp*)
apply (*simp add*:*mult.commute*[*of s r*])
apply (*simp add*: *not-less* [*symmetric*, *of r* ∗ *s r* ∗ *s* − *Suc 0*])

apply (*frule mult-less-mono2*[*of 0 s r*],
simp only:*mult.commute*, *simp only*:*mult-0-right*,
cut-tac x = *x* **in** *Nset-pre*[*of* - *r* ∗ *s* − *Suc 0*], *simp*+)
apply (*case-tac y* = *r* ∗ *s*)
apply (*simp add*: *rtos-rs-Tr1*)
apply (*frule-tac l* = *x* **in** *rtos-hom1*[*of r s*], *assumption*+)
apply (*simp add*:*mult.commute*[*of s r*])
apply (*simp add*: *not-less* [*symmetric*, *of r* ∗ *s r* ∗ *s* − *Suc 0*])

apply (*cut-tac x* = *y* **in** *Nset-pre*[*of* - *r* ∗ *s* − *Suc 0*], *simp*+)
apply (*frule rtos-inj*[*of r s*], *assumption*+)
apply (*simp add*:*inj-on-def*)
done

lemma *Qw-cmpser*:⟦*Group G*; *w-cmpser G* (*Suc n*) *f* ⟧ ⟹
Gchain n (*Qw-cmpser G f*)

apply (*simp add*:*Gchain-def*)
apply (*rule allI*, *rule impI*)
apply (*simp add*:*Qw-cmpser-def*)
apply (*simp add*:*w-cmpser-def*)
apply (*erule conjE*)
apply (*simp add*:*d-gchain-def*)
apply (*cut-tac* $c = l$ **in** *subsetD*[*of* $\{i.\ i \leq n\}$ $\{i.\ i \leq (Suc\ n)\}$],
rule subsetI, *simp*, *simp*)
apply (*frule-tac* $H = f\ l$ **in** *Group.Group-Gp*[*of G*],
frule-tac $a = l$ **in** *forall-spec*, *simp*+)
apply (*frule-tac* $G = Gp\ G\ (f\ l)$ **and** $N = f\ (Suc\ l)$ **in** *Group.Group-Qg*)
apply *blast* **apply** *assumption*
done

definition
wcsr-rfns :: [- , *nat*, *nat* ⇒ *'a set*, *nat*] ⇒ (*nat* ⇒ *'a set*) *set* **where**
wcsr-rfns G r f s = {*h*. *tw-cmpser G* (*s* ∗ *r*) *h* ∧
(∀ *i* ≤ *r*. *h* (*i* ∗ *s*) = *f i*)}

definition
trivial-rfn :: [- , *nat*, *nat* ⇒ *'a set*, *nat*] ⇒ (*nat* ⇒ *'a set*) **where**
trivial-rfn G r f s k == *if k* < (*s* ∗ *r*) *then f* (*k div s*) *else f r*

lemma (**in** *Group*) *rfn-tool8*:⟦*compseries G r f*; 0 < *r*⟧ ⟹ *d-gchain G r f*
apply (*simp add*:*compseries-def tW-cmpser-def*)
apply (*erule conjE*)+
apply (*simp add*:*tD-gchain-def*) **apply** (*erule conjE*)+
apply (*simp add*:*D-gchain-is-d-gchain*)
done

lemma (**in** *Group*) *rfn-tool16*:⟦0 < *r*; 0 < *s*; *i* ≤ (*s* ∗ *r* − *Suc 0*);
G ⋗ *f* (*i div s*); (*Gp G* (*f* (*i div s*))) ▷ *f* (*Suc* (*i div s*));
(*Gp G* (*f* (*i div s*))) ⋗ (*f* (*i div s*) ∩ *g* (*s* − *Suc 0*))⟧ ⟹
(*Gp G* ((*f* (*Suc* (*i div s*)) ⋄$_G$ (*f* (*i div s*) ∩ *g* (*s* − *Suc 0*)))))) ▷
(*f* (*Suc* (*i div s*)))
apply (*rule ZassenhausTr4-1* [*of f* (*i div s*) *f* (*Suc* (*i div s*)) *g* (*s* − *Suc 0*)],
assumption+)
done

Show existence of the trivial refinement. This is not necessary to prove JHS

lemma *rfn-tool30*:⟦0 < *r*; 0 < *s*; *l div s* ∗ *s* + *s* < *s* ∗ *r*⟧
⟹ *Suc* (*l div s*) < *r*
apply (*simp add*:*mult.commute*[*of s r*])
apply (*cut-tac add-mult-distrib*[*THEN sym*, *of l div s s 1*])
apply (*simp only*:*nat-mult-1*)
apply (*thin-tac l div s* ∗ *s* + *s* = (*l div s* + *1*) ∗ *s*)

apply (*cut-tac mult-less-cancel2*[*of l div s + 1 s r*])
  apply *simp*
done

lemma (**in** *Group*) *simple-grouptr0*:⟦*G ⋗ H*; *G ▷ K*; *K ⊆ H*; *simple-Group* (*G / K*)⟧
 ⟹ *H = carrier G ∨ H = K*
apply (*simp add*:*simple-Group-def*)
apply (*frule subg-Qsubg*[*of H K*], *assumption*+)
apply (*rotate-tac −1*)
apply (*frule in-set-with-P*[*of - carrier* ((*Gp G H*) / *K*)])
  apply *simp*
  apply (*thin-tac* {*N*. *G / K ⋗ N* } = {*carrier* (*G / K*), {**1**$_{G\ /\ K}$}})
 apply (*erule disjE*)
 apply (*frule sg-subset*[*of H*])
 apply (*frule equalityI*[*of H carrier G*])
 apply (*rule subsetI*)
 apply (*simp add*:*Qg-carrier*)
 apply (*frule nsg-sg*[*of K*])
 apply (*frule-tac a = x* **in** *rcs-in-set-rcs*[*of K*], *assumption*+)
 apply (*frule sym*, *thin-tac carrier* ((♮*H*)/ *K*) = *set-rcs G K*, *simp*,
   thin-tac set-rcs G K = carrier ((♮*H*) / *K*))
 apply (*frule Group-Gp*[*of H*], *simp add*:*Group.Qg-carrier*[*of Gp G H K*],
   simp add:*set-rcs-def*, *erule bexE*, *simp add*:*Gp-carrier*)

 apply (*simp add*:*rcs-in-Gp*[*THEN sym*])
 apply (*frule-tac a = x* **in** *a-in-rcs*[*of K*], *assumption*+, *simp*,
   thin-tac K ⋅ x = K ⋅ a,
   thin-tac G / K ⋗ {*C*. ∃ *a*∈*H*. *C = K ⋅ a*},
   simp add:*rcs-def*, *erule bexE*,
   frule-tac c = h **in** *subsetD*[*of K H*], *assumption*+)
 apply (*frule-tac x = h* **and** *y = a* **in** *sg-mult-closed*[*of H*], *assumption*+, *simp*)
 apply *simp*

apply (*frule equalityI*[*THEN sym*, *of K H*])
 apply (*rule subsetI*)
 apply (*frule Group-Gp*[*of H*], *simp add*:*Group.Qg-carrier*)
 apply (*simp add*:*Qg-one*)
 apply (*frule nsg-sg*[*of K*])
 apply (*frule-tac a = x* **in** *Group.rcs-in-set-rcs*[*of Gp G H K*])
  apply (*simp add*:*sg-sg*) **apply** (*simp add*:*Gp-carrier*)
  apply *simp*
  apply (*frule-tac a = x* **in** *Group.rcs-fixed*[*of Gp G H K*])
   apply (*simp add*:*sg-sg*) **apply** (*simp add*:*Gp-carrier*) **apply** *assumption*+
 apply *simp*
done

lemma (**in** *Group*) *compser-nsg*:⟦*0 < n*; *compseries G n f*; *i ≤ (n − 1)*⟧
  ⟹ *Gp G* (*f i*) ▷ (*f* (*Suc i*))

apply (*simp add*:*compseries-def tW-cmpser-def*)
done

lemma (**in** *Group*) *compseriesTr5*:⟦*0* < *n*; *compseries G n f*; *i* ≤ (*n* − *Suc 0*)⟧
⟹ (*f* (*Suc i*)) ⊆ (*f i*)
apply (*frule compseriesTr4*[*of n f*])
apply (*frule w-cmpser-is-d-gchain*[*of n f*])
apply (*simp add*:*d-gchain-def*,
cut-tac n-in-Nsetn[*of n*],
frule-tac a = *n* **in** *forall-spec*, *simp*,
thin-tac ∀ *l* ≤ *n*. *G* ≫ *f l* ∧ (∀ *l* ≤ (*n* − *Suc 0*). *f* (*Suc l*) ⊆ *f l*),
erule conjE, *blast*)
done

lemma (**in** *Group*) *refine-cmpserTr0*:⟦*0* < *n*; *compseries G n f*; *i* ≤ (*n* − *1*);
G ≫ *H*; *f* (*Suc i*) ⊆ *H* ∧ *H* ⊆ *f i*⟧ ⟹ *H* = *f* (*Suc i*) ∨ *H* = *f i*
apply (*frule compseriesTr0* [*of n f i*])
apply (*cut-tac lessI*[*of n* − *Suc 0*], *simp only*:*Suc-pred*, *simp*)
apply (*frule Group-Gp* [*of f i*])
apply (*erule conjE*)
apply (*frule compseriesTr4*[*of n f*])
apply (*frule w-cmpser-ns*[*of n f i*], *simp*+)
apply (*unfold compseries-def*, *frule conjunct2*, *fold compseries-def*, *simp*)
apply (*frule-tac x* = *i* **in** *spec*,
thin-tac ∀ *i*≤*n* − *Suc 0*. *simple-Group* ((♮*f i*) / *f* (*Suc i*)),
simp)
apply (*frule Group.simple-grouptr0* [*of Gp G* (*f i*) *H f* (*Suc i*)])
apply (*simp add*:*sg-sg*) **apply** *assumption*+
apply (*simp add*:*Gp-carrier*)
apply *blast*
done

lemma *div-Tr4*:⟦ (*0*::*nat*) < *r*; *0* < *s*; *j* < *s* ∗ *r* ⟧ ⟹ *j div s* ∗ *s* + *s* ≤ *r* ∗ *s*
apply (*frule div-Tr2* [*of r s j*], *assumption*+)
apply (*frule mult-le-mono* [*of j div s r* − *Suc 0 s s*])
apply *simp*
apply (*frule add-le-mono* [*of j div s* ∗ *s* (*r* − *Suc 0*) ∗ *s s s*])
apply *simp*
apply (*thin-tac j div s* ∗ *s* ≤ (*r* − *Suc 0*) ∗ *s*)
apply (*cut-tac add-mult-distrib*[*THEN sym*, *of r* − *Suc 0 s 1*])
apply (*simp only*:*nat-mult-1*)
apply *simp*
done

lemma (**in** *Group*) *compseries-is-tW-cmpser*:⟦*0* < *r*; *compseries G r f*⟧ ⟹
tW-cmpser G r f
by (*simp add*:*compseries-def*)

lemma (**in** *Group*) *compseries-is-td-gchain*:⟦*0* < *r*; *compseries G r f*⟧ ⟹

td-gchain G r f
apply (*frule compseries-is-tW-cmpser*, *assumption*+)
apply (*simp add*:*tW-cmpser-def*, *erule conjE*)
apply (*thin-tac* $\forall l{\leq}(r - Suc\ 0).\ (\natural f\ l) \triangleright (f\ (Suc\ l))$)
apply (*simp add*:*tD-gchain-def*, (*erule conjE*)+)
apply (*frule D-gchain-is-d-gchain*)
apply (*simp add*:*td-gchain-def*)
done

lemma (**in** *Group*) *compseries-is-D-gchain*:$[\![0 < r;\ compseries\ G\ r\ f]\!] \Longrightarrow$
D-gchain G r f
apply (*frule compseriesTr1*)
apply (*frule tW-cmpser-is-W-cmpser*)
apply (*rule W-cmpser-is-D-gchain*, *assumption*)
done

lemma *divTr5*:$[\![0 < r;\ 0 < s;\ l < (r * s)]\!] \Longrightarrow$
$l\ div\ s * s \leq l \wedge l\ \leq (Suc\ (l\ div\ s)) * s$
apply (*insert mod-div-equality* [*THEN sym*, *of l s*])
apply (*rule conjI*)
apply (*insert le-add1* [*of l div s * s l mod s*])
apply *simp*
apply (*frule mod-less-divisor* [*of s l*])
apply (*frule less-imp-le* [*of l mod s s*])
apply (*insert self-le* [*of l div s * s*])
apply (*frule add-le-mono* [*of l div s * s l div s * s l mod s s*])
apply *assumption*+
apply (*thin-tac* $l\ div\ s * s \leq l\ div\ s * s + l\ mod\ s$)
apply (*thin-tac* $l\ div\ s * s \leq l\ div\ s * s$)
apply *simp*
done

lemma (**in** *Group*) *rfn-compseries-iMTr1*:$[\![0 < r;\ 0 < s;\ compseries\ G\ r\ f;$
$h \in wcsr\text{-}rfns\ G\ r\ f\ s]\!] \Longrightarrow\ f\ `\ \{i.\ i \leq r\} \subseteq\ h\ `\ \{i.\ i \leq (s * r)\}$
apply (*simp add*:*wcsr-rfns-def*) **apply** (*rule subsetI*)
apply (*simp add*:*image-def*)
apply (*auto del*:*equalityI*)
apply (*frule-tac* $i = xa$ **in** *mult-le-mono* [*of - r s s*])
apply *simp*
apply (*simp add*:*mult.commute* [*of r s*])
apply (*frule-tac* $a = xa$ **in** *forall-spec*, *assumption*,
thin-tac $\forall i{\leq}r.\ h\ (i * s) = f\ i$)
apply (*frule sym*, *thin-tac* $h\ (xa * s) = f\ xa$)
apply (*cut-tac* $a = xa$ **in** *mult-mono*[*of - r s s*], *simp*, *simp*, *simp*, *simp*)
apply (*simp only*:*mult.commute*[*of r s*], *blast*)
done

lemma *rfn-compseries-iMTr2*:$[\![0 < r;\ 0 < s;\ xa < s * r\ ]\!] \Longrightarrow$
$xa\ div\ s * s \leq r * s \wedge Suc\ (xa\ div\ s) * s \leq r * s$

apply (*frule div-Tr1* [*of r s xa*], *assumption*+)
 apply (*simp add*:*less-imp-le*)
apply (*rule conjI*)
 apply (*simp add*:*mult-le-mono* [*of xa div s r s s*])
apply (*thin-tac xa div s* ≤ *r*)
apply (*frule div-Tr2*[*of r s xa*], *assumption*+)
 apply (*thin-tac xa* < *s* ∗ *r*)
 apply (*frule le-less-trans* [*of xa div s r* − *Suc 0 r*])
 apply *simp*
 apply (*frule Suc-leI* [*of xa div s r*])
 apply (*rule mult-le-mono* [*of Suc* (*xa div s*) *r s s*], *assumption*+)
apply *simp*
done

lemma (**in** *Group*) *rfn-compseries-iMTr3*:⟦*0* < *r*; *0* < *s*; *compseries G r f*;
 j ≤ *r*; ∀ *i* ≤ *r*. *h* (*i* ∗ *s*) = *f i*⟧ ⟹ *h* (*j* ∗ *s*) = *f j*
apply *blast*
done

lemma (**in** *Group*) *rfn-compseries-iM*:⟦*0* < *r*; *0* < *s*; *compseries G r f*;
 h ∈ *wcsr-rfns G r f s*⟧ ⟹ *card* (*h* ‘{*i*. *i* ≤ (*s* ∗ *r*)}) = *r* + *1*
apply (*frule compseries-is-D-gchain*, *assumption*+)
apply (*frule D-gchain1*)
 apply *simp*
 apply (*subst card-Collect-le-nat*[*THEN sym*, *of r*])
 apply (*subst card-image*[*THEN sym*, *of f* {*i*. *i* ≤ *r*}], *assumption*+)
 apply (*rule card-eq*[*of h* ‘ {*i*. *i* ≤ (*s* ∗ *r*)} *f* ‘ {*i*. *i* ≤ *r*}])
apply (*frule rfn-compseries-iMTr1*[*of r s f h*], *assumption*+)
apply (*rule equalityI*[*of h* ‘ {*i*. *i* ≤ (*s* ∗ *r*)} *f* ‘ {*i*. *i* ≤ *r*}])
prefer *2* **apply** *simp*
apply (*rule subsetI*,
 thin-tac f ‘ {*i*. *i* ≤ *r*} ⊆ *h* ‘ {*i*. *i* ≤ *s* ∗ *r*})
apply (*frule-tac b* = *x* **and** *f* = *h* **and** *A* = {*i*. *i* ≤ (*s* ∗ *r*)} **in** *mem-in-image3*,
 erule bexE) **apply** (*simp add*:*mult.commute*[*of s r*])
apply (*simp add*:*wcsr-rfns-def*, (*erule conjE*)+)
apply (*frule rfn-compseries-iMTr3*[*of r s f r h*], *assumption*+,
 simp add:*n-in-Nsetn*, *assumption*+, *subst image-def*, *simp*)
 apply (*case-tac a* = *s* ∗ *r*, *simp add*:*mult.commute*[*of s r*],
 cut-tac n-in-Nsetn[*of r*], *blast*)
 apply (*simp add*:*mult.commute*[*of s r*])
 apply (*frule-tac m* = *a* **and** *n* = *r* ∗ *s* **in** *noteq-le-less*, *assumption*+)
 apply (*frule tw-cmpser-is-w-cmpser*, *frule w-cmpser-is-d-gchain*)
 apply (*frule-tac xa* = *a* **in** *rfn-compseries-iMTr2*[*of r s*], *assumption*+)
 apply (*simp add*:*mult.commute*)
 apply (*erule conjE*)
 apply (*frule-tac l* = *a* **in** *divTr5*[*of r s*], *assumption*+)
 apply (*frule pos-mult-pos*[*of r s*], *assumption*+)
 apply (*erule conjE*,
 frule-tac l = *a div s* ∗ *s* **and** *j* = *a* **in** *d-gchainTr2*[*of r* ∗ *s h*],

assumption+)
apply (*frule-tac* $l = a$ **and** $j = Suc\ (a\ div\ s) * s$ **in** *d-gchainTr2*[*of* $r * s$ *h*], *assumption*+)
apply (*frule-tac* $i = a$ **in** *rtos-hom0*[*of r s*], *assumption*+)
apply (*rule less-le-diff*)
apply *simp*
apply (*frule-tac* $x = a\ div\ s$ **and** $y = r$ **in** *less-imp-le*,
frule-tac $a = a\ div\ s$ **in** *forall-spec*, *assumption*,
frule-tac $a = Suc\ (a\ div\ s)$ **in** *forall-spec*)
apply (*cut-tac* $m = Suc\ (a\ div\ s)$ **and** $k = s$ **and** $n = r$ **in** *mult-le-cancel2*)
apply *simp*
apply (*thin-tac* $\forall i \leq r.\ h\ (i * s) = f\ i$) **apply** *simp*
apply (*cut-tac* $i = a\ div\ s$ **and** $H = h\ a$**in** *refine-cmpserTr0*[*of r f*],
simp, *assumption*+,
cut-tac $x = a\ div\ s$ **and** $n = r$ **in** *less-le-diff*, *assumption*, *simp*)
apply (*simp add*:*d-gchain-mem-sg*[*of* $r * s$ *h*])
apply *blast*
apply (*erule disjE*)
apply (*frule-tac* $m = a\ div\ s$ **and** $n = r$ **in** *Suc-leI*, *blast*)
apply (*frule-tac* $x = a\ div\ s$ **and** $y = r$ **in** *less-imp-le*, *blast*)
done

definition
cmp-rfn :: [- , *nat*, *nat* ⇒ *'a set*, *nat*, *nat* ⇒ *'a set*] ⇒ (*nat* ⇒ *'a set*) **where**
cmp-rfn G r f s g = (λ*i*. (*if* $i < s * r$ *then*
$f\ (Suc\ (i\ div\ s))\ \diamond_G\ (f\ (i\ div\ s) \cap g\ (i\ mod\ s))$ *else* $\{\mathbf{1}_G\}$))

lemma (**in** *Group*) *cmp-rfn0*:⟦$0 < r$; $0 < s$; *compseries G r f*; *compseries G s g*;
$i \leq (r - 1)$; $j \leq (s - 1)$⟧ ⟹ $G \gg f\ (Suc\ i)\ \diamond_G\ ((f\ i\ ) \cap (g\ j))$
apply (*rule ZassenhausTr2-1*[*of f i f* (*Suc i*) *g j*], *simp*,
rule compseriesTr0[*of r f i*], *assumption*+,
frule-tac le-less-trans[*of i r* − *Suc 0 r*], *simp*+)
apply (*rule compseriesTr0*[*of r f Suc i*], *assumption*+, *arith*)
apply(*rule compseriesTr0*[*of s g j*], *assumption*+, *simp*)
apply (*frule compseries-is-tW-cmpser*[*of r f*], *assumption*+,
simp add:*tW-cmpser-def*)
done

lemma (**in** *Group*) *cmp-rfn1*:⟦$0 < r$; $0 < s$; *compseries G r f*; *compseries G s g*⟧
⟹ $f\ (Suc\ 0)\ \diamond_G\ ((f\ 0\ ) \cap (g\ 0)) = carrier\ G$
apply (*frule compseriesTr2* [*of r f*])
apply (*frule compseriesTr2* [*of s g*])
apply (*frule compseriesTr0* [*of - f Suc 0*])
apply *simp*
apply *simp*
apply (*rule K-absorb-HK*[*of f* (*Suc 0*) *carrier G*], *assumption*+,
simp add:*special-sg-G*)

apply (*rule sg-subset*, *assumption*)
done

lemma (**in** *Group*) *cmp-rfn2*:⟦*0* < *r*; *0* < *s*; *compseries G r f*; *compseries G s g*;
l ≤ (*s* ∗ *r*)⟧ ⟹ *G* ≫ *cmp-rfn G r f s g l*
apply (*simp add*:*cmp-rfn-def*)
apply (*case-tac l* < *s* ∗ *r*)
apply *simp*
apply (*frule-tac i* = *l div s* **and** *j* = *l mod s* **in** *cmp-rfn0* [*of r s*],
assumption+)
apply (*simp add*:*div-Tr2*)
apply (*frule-tac m* = *l* **in** *mod-less-divisor* [*of s*])
apply (*frule-tac x* = *l mod s* **and** *n* = *s* **in** *less-le-diff*)
apply *simp* **apply** *assumption*
apply *simp*
apply (*rule special-sg-e*)
done

lemma (**in** *Group*) *cmp-rfn3*:⟦*0* < *r*; *0* < *s*; *compseries G r f*; *compseries G s g*⟧
⟹ *cmp-rfn G r f s g 0* = *carrier G* ∧ *cmp-rfn G r f s g* (*s* ∗ *r*) = {**1**}
apply (*rule conjI*)
apply (*simp add*:*cmp-rfn-def*)
apply (*rule cmp-rfn1* [*of r s f g*], *assumption*+)
apply (*simp add*:*cmp-rfn-def*)
done

lemma *rfn-tool20*:⟦(*0*::*nat*) < *m*; *a* = *b* ∗ *m* + *c*; *c* < *m* ⟧ ⟹ *a mod m* = *c*
apply (*simp add*:*add.commute*)
done

lemma *Suci-mod-s-2*:⟦*0* < *r*; *0* < *s*; *i* ≤ *r* ∗ *s* − *Suc 0*; *i mod s* < *s* − *Suc 0*⟧
⟹ (*Suc i*) *mod s* = *Suc* (*i mod s*)
apply (*insert mod-div-equality* [*of i s*, *THEN sym*])
apply (*subgoal-tac Suc i* = *i div s* ∗ *s* + *Suc* (*i mod s*))
apply (*subgoal-tac Suc i mod s* = (*i div s* ∗ *s* + *Suc* (*i mod s*)) *mod s*)
apply (*subgoal-tac Suc* (*i mod s*) < *s*)
apply (*frule-tac m* = *s* **and** *a* = *Suc i* **and** *b* = *i div s* **and** *c* = *Suc* (*i mod s*)
in *rfn-tool20*, *assumption*+)
apply (*subgoal-tac Suc* (*i mod s*) < *Suc* (*s* − *Suc 0*)) **apply** *simp*
apply (*simp del*:*Suc-pred*)
apply *simp*+
done

lemma (**in** *Group*) *inter-sgsTr1*:⟦*0* < *r*; *0* < *s*; *compseries G r f*; *compseries G s*
g; *i* < *r* ∗ *s*⟧ ⟹ *G* ≫ *f* (*i div s*) ∩ *g* (*s* − *Suc 0*)
apply (*rule inter-sgs*)
apply (*rule compseriesTr0*, *assumption*+)
apply (*frule less-imp-le* [*of i r* ∗ *s*])
apply (*frule div-Tr1* [*of r s i*], *assumption*+)

apply (*simp add*:*mult.commute*, *simp*)
apply (*rule compseriesTr0*, *simp*+)
done

lemma (**in** *Group*) *JHS-Tr0-2*:⟦*0* < *r*; *0* < *s*; *compseries G r f*; *compseries G s g*⟧
⟹ ∀ *i* ≤ (*s* ∗ *r* − *Suc 0*). *Gp G* (*cmp-rfn G r f s g i*) ▷
cmp-rfn G r f s g (*Suc i*)

apply (*frule compseriesTr4* [*of r f*], *frule compseriesTr4* [*of s g*])
apply (*rule allI*, *rule impI*)
apply (*frule pos-mult-pos* [*of s r*], *assumption*+,
frule-tac a = *i* **in** *rfn-tool11* [*of s* ∗ *r*], *assumption*+,
frule-tac l = *i* **in** *div-Tr1* [*of r s*], *assumption*+,
simp add:*less-imp-le*)
apply (*frule-tac x* = *i div s* **in** *mem-of-Nset* [*of - r*],
thin-tac i div s ≤ *r*, *thin-tac i* ≤ *s* ∗ *r* − *Suc 0*,
frule-tac l = *i* **in** *div-Tr2* [*of r s*], *assumption*+,
frule-tac x = *i div s* **in** *mem-of-Nset* [*of - r* − *Suc 0*],
frule-tac a = *i div s* **in** *rfn-tool11* [*of r*], *assumption*+,
frule-tac m = *i div s* **in** *Suc-leI* [*of - r*],
frule-tac x = *Suc* (*i div s*) **in** *mem-of-Nset* [*of - r*],
thin-tac i div s < *r*, *thin-tac Suc* (*i div s*) ≤ *r*)
apply (*simp add*:*cmp-rfn-def*)
apply (*case-tac Suc i* < *s* ∗ *r*, *simp*,
case-tac i mod s = *s* − *1*,
cut-tac l = *i* **in** *div-Tr3-1* [*of r s*],
simp+,
cut-tac l = *Suc i* **in** *div-Tr2* [*of r s*], *simp*+,
cut-tac x = *Suc i div s* **in** *mem-of-Nset* [*of - r* − *Suc 0*],
simp,
cut-tac a = *Suc i div s* **in** *rfn-tool11* [*of r*], *simp*+,
cut-tac x = *Suc i div s* **in** *less-mem-of-Nset* [*of - r*], *simp*,
cut-tac m = *Suc i div s* **in** *Suc-leI* [*of - r*], *simp*,
frule-tac x = *Suc* (*Suc i div s*) **in** *mem-of-Nset* [*of - r*],
frule w-cmpser-is-d-gchain [*ofr f*],
frule-tac rfn-tool12-2 [*of s*], *assumption*+,
thin-tac i mod s = *s* − *Suc 0*,
thin-tac Suc i div s ∈ {*i*. *i* ≤ *r*},
thin-tac Suc (*Suc i div s*) ≤ *r*)
apply (*simp*,
cut-tac l = *i div s* **and** *j* = *Suc* (*i div s*) **in**
d-gchainTr2 [*of r f*], *simp*, *assumption*+,
cut-tac x = *i div s* **and** *y* = *Suc* (*i div s*) **and** *z* = *r* **in**
less-trans, *simp*, *assumption*, *simp add*:*less-imp-le*,
simp add:*less-imp-le*, *simp*,
cut-tac l = *Suc* (*i div s*) **and** *j* = *Suc* (*Suc* (*i div s*)) **in**
d-gchainTr2 [*of r f*], *simp*+,
thin-tac Suc i div s = *Suc* (*i div s*), *thin-tac Suc i mod s* = *0*,

cut-tac $i = i$ *div* s **in** *compseriesTr0* [*of r f*], *assumption*,
simp,
cut-tac $i = Suc\ (i\ div\ s)$ **in** *compseriesTr0* [*of r f*],
assumption, *simp add*:*less-imp-le*,
cut-tac $i = Suc\ (Suc\ (i\ div\ s))$ **in** *compseriesTr0* [*of r f*],
assumption, *simp*)

apply (*subst compseriesTr2* [*of s g*], *assumption*,
frule-tac $H = f\ (i\ div\ s)$ **in** *sg-subset*,
frule-tac $H = f\ (Suc\ (i\ div\ s))$ **in** *sg-subset*,
frule-tac $A = f\ (Suc\ (i\ div\ s))$ **in** *Int-absorb2* [*of - carrier G*],
simp,
thin-tac $f\ (Suc\ (i\ div\ s)) \cap carrier\ G = f\ (Suc\ (i\ div\ s))$,
frule-tac $H = f\ (Suc\ (Suc\ (i\ div\ s)))$ **and** $K = f\ (Suc\ (i\ div\ s))$ **in**
K-absorb-HK, *assumption*+,
simp,
thin-tac $f\ (Suc\ (Suc\ (i\ div\ s))) \diamond_G (f\ (Suc\ (i\ div\ s))) = f\ (Suc\ (i\ div\ s)))$
apply (*rule rfn-tool16* [*of r s -*], *simp*+,
cut-tac $x = Suc\ i$ **and** $y = s * r$ **and** $z = Suc\ (s * r)$ **in**
less-trans, *assumption*, *simp*,
thin-tac $Suc\ i < s * r$, *simp*)
apply (*rule compser-nsg*[*of r f*], *simp*+)
apply (*rule-tac* $H = f\ (i\ div\ s) \cap g\ (s - Suc\ 0)$ **and** $K = f\ (i\ div\ s)$ **in**
sg-sg, *assumption*+,
cut-tac $i = i$ **in** *inter-sgsTr1* [*of r s f g*], *simp*+,
cut-tac $x = i$ **and** $y = Suc\ i$ **and** $z = s * r$ **in** *less-trans*,
simp+,
cut-tac $x = i$ **and** $y = Suc\ i$ **and** $z = r * s$ **in** *less-trans*,
simp+,
simp add:*mult.commute*, *assumption*+,
simp add:*Int-lower1*)
apply (*frule-tac* $m = i$ **in** *mod-less-divisor* [*of s*],
frule-tac $x = i\ mod\ s$ **in** *less-le-diff* [*of - s*],
simp,
frule-tac $m = i\ mod\ s$ **in** *noteq-le-less* [*of - s − Suc 0*],
assumption+,
thin-tac $i\ mod\ s \neq s - Suc\ 0$, *thin-tac* $i\ mod\ s \leq s - Suc\ 0$)
apply (*frule-tac* $x = i\ mod\ s$ **and** $y = s - Suc\ 0$ **and** $z = s$ **in**
less-trans, *simp*,
frule-tac $k = i\ mod\ s$ **in** *nat-pos2* [*of - s*],
cut-tac l1 $= i$ **in** *div-Tr3-2* [*THEN sym*, *of r s*], *simp*+,
frule-tac $i = i\ mod\ s$ **in** *compser-nsg* [*of s g*], *assumption*+,
simp,
frule-tac $i = Suc\ (i\ div\ s)$ **in** *compseriesTr0* [*of r f*],
assumption+,
frule-tac $m = i\ mod\ s$ **in** *Suc-mono* [*of - s − Suc 0*],
simp only:*Suc-pred*)
apply (*frule-tac* $x = Suc\ (i\ mod\ s)$ **in** *less-mem-of-Nset* [*of - s*],

cut-tac $i = Suc\ (i\ mod\ s)$ **in** *compseriesTr0* [*of s g*], *simp*+,
cut-tac $H = f\ (i\ div\ s)$ **and** *?H1.0* $= f\ (Suc\ (i\ div\ s))$ **and**
$K = g\ (i\ mod\ s)$ **and** *?K1.0* $= g\ (Suc\ (i\ mod\ s))$ **in** *ZassenhausTr3*,
rule-tac $i = i\ div\ s$ **in** *compseriesTr0* [*of r f*], *assumption*+,
frule-tac $x = i\ div\ s$ **and** $y = r - Suc\ 0$ **and** $z = r$ **in**
le-less-trans, *simp*, *simp*, *assumption*+,
frule-tac $i = i\ mod\ s$ **in** *compseriesTr0* [*of s g*],
simp, *simp*, *simp*)
apply (*rule-tac* $i = i\ div\ s$ **in** *compser-nsg*[*of r f*], *simp*+,
cut-tac $i = i$ **in** *Suci-mod-s-2*[*of r s*], *simp*+,
cut-tac $x = Suc\ i$ **and** $y = s * r$ **and** $z = Suc\ (s * r)$ **in**
less-trans, *assumption*, *simp*,
thin-tac $Suc\ i < s * r$, *simp*,
frule-tac $x = i$ **and** $n = s * r$ **in** *less-le-diff*,
simp add:*mult.commute*[*of s r*], *simp*+)
apply (*cut-tac* $a = i\ mod\ s$ **in** *rfn-tool11* [*of s*], *simp*+,
frule-tac $m = i$ **in** *mod-less-divisor* [*of s*],
frule-tac $x = i\ mod\ s$ **in** *less-le-diff* [*of - s*], *simp*,
rule special-nsg-e,
cut-tac $i = i\ div\ s$ **and** $j = i\ mod\ s$ **in** *cmp-rfn0*[*of r s f g*],
simp+)
done

lemma (**in** *Group*) *cmp-rfn4*:⟦$0 < r$; $0 < s$; *compseries G r f*;
compseries G s g; $l \leq (s * r - Suc\ 0)$⟧ ⟹
cmp-rfn G r f s g $(Suc\ l) \subseteq$ *cmp-rfn G r f s g l*
apply (*frule JHS-Tr0-2* [*of r s f g*], *assumption*+)
apply (*frule-tac* $a = l$ **in** *forall-spec*, *simp*,
thin-tac $\forall i \leq (s * r - Suc\ 0)$.
(♮(*cmp-rfn G r f s g i*)) ▷ (*cmp-rfn G r f s g* (*Suc i*)))
apply (*frule cmp-rfn2* [*of r s f g l*], *assumption*+,
frule le-less-trans [*of l s* $* r - Suc\ 0\ s*$ *r*], *simp*,
simp add:*less-imp-le*)
apply (*frule Group-Gp* [*of cmp-rfn G r f s g l*],
frule Group.nsg-subset [*of Gp G* (*cmp-rfn G r f s g l*)],
assumption+, *simp add*:*Gp-carrier*)
done

lemma (**in** *Group*) *cmp-rfn5*:⟦$0 < r$; $0 < s$; *compseries G r f*; *compseries G s g*⟧
⟹ $\forall i \leq r$. *cmp-rfn G r f s g* $(i * s) = f\ i$
apply (*rule allI*, *rule impI*)
apply (*simp add*:*cmp-rfn-def*)
apply (*case-tac* $i < r$, *simp*,
frule-tac $x = i$ **in** *less-imp-le* [*of - r*],
frule-tac $x = i$ **in** *mem-of-Nset*[*of - r*],
frule-tac $i = i$ **in** *compseriesTr0* [*of r f*], *assumption*+,
thin-tac $i \leq r$,
frule-tac $m = i$ **in** *Suc-leI* [*of - r*],
frule-tac $x = Suc\ i$ **in** *mem-of-Nset*[*of - r*],

frule-tac $i = Suc\ i$ **in** *compseriesTr0* [*of r f*], *assumption*+)
apply (*subst compseriesTr2* [*of s g*], *assumption*+,
frule-tac $H = f\ i$ **in** *sg-subset*,
subst Int-absorb2, *assumption*+,
frule rfn-tool8, *assumption*+, *simp*)
apply (*cut-tac* $n = i$ **in** *zero-less-Suc*,
frule-tac $x = 0$ **and** $y = Suc\ i$ **and** $z = r$ **in** *less-le-trans*, *assumption*+,
frule-tac $l = i$ **and** $j = Suc\ i$ **in** *d-gchainTr2* [*of r f*],
frule compseries-is-D-gchain[*of r f*], *assumption*,
rule D-gchain-is-d-gchain, *assumption*+,
cut-tac $x = i$ **and** $y = Suc\ i$ **and** $z = r$ **in** *less-le-trans*, *simp*+,
rule-tac $K = f\ i$ **and** $H = f\ (Suc\ i)$ **in** *K-absorb-HK*, *assumption*+,
simp, *frule compseries-is-td-gchain*, *assumption*+,
simp add:*td-gchain-def*)
done

lemma (**in** *Group*) *JHS-Tr0*:$[\![(0::nat) < r;\ 0 < s;$ *compseries G r f*;
compseries G s g$]\!] \Longrightarrow$ *cmp-rfn G r f s g* $\in$ *wcsr-rfns G r f s*
apply (*simp add*:*wcsr-rfns-def*,
frule cmp-rfn5 [*of r s f g*], *assumption*+,
simp,
thin-tac $\forall i{\le}r.$ *cmp-rfn G r f s g* $(i * s) = f\ i$)
apply (*simp add*:*tw-cmpser-def*, *rule conjI*)
prefer *2* **apply** (*rule JHS-Tr0-2* [*of r s f g*], *assumption*+)
apply (*simp add*:*td-gchain-def*, *rule conjI*)
prefer *2*
apply (*rule cmp-rfn3*, *assumption*+)
apply (*simp add*:*d-gchain-def*,
rule allI, *rule impI*, *rule conjI*, *rule cmp-rfn2*, *assumption*+,
rule allI, *rule impI*, *rule cmp-rfn4*, *assumption*+)
done

lemma *rfn-tool17*:$(a::nat) = b \Longrightarrow a - c = b - c$
by *simp*

lemma *isom4b*:$[\![$*Group G*; $G \triangleright N$; $G \gg H]\!] \Longrightarrow$
$(Gp\ G\ (N \diamond_G H)\ /\ N) \cong (Gp\ G\ H\ /\ (H \cap N))$
apply (*frule isom4* [*of G N H*], *assumption*+)
apply (*rule isomTr1*)
apply (*simp add*:*Group.homom4-2*)
apply (*frule Group.smult-sg-nsg*[*of G H N*], *assumption*+)
apply (*simp add*:*Group.smult-commute-sg-nsg*[*THEN sym*, *of G H N*])
apply (*rule Group.homom4Tr1*, *assumption*+)
done

lemma *Suc-rtos-div-r-1*:$[\![\ 0 < r;\ 0 < s;\ i \le r * s - Suc\ 0;$
$Suc\ (rtos\ r\ s\ i) < r * s;\ i\ mod\ s = s - Suc\ 0;$
$i\ div\ s < r - Suc\ 0]\!] \Longrightarrow\ Suc\ (rtos\ r\ s\ i)\ div\ r = i\ mod\ s$
apply (*simp add*:*rtos-def*,

frule le-less-trans [*of i r * s - Suc 0 r * s*], *simp*)
apply *simp*
apply (*subgoal-tac Suc* ((*s - Suc 0*) * *r* + *i div s*) *div r* =
((*s - Suc 0*) * *r* + *Suc* (*i div s*)) *div r*,
simp del: *add-Suc add-Suc-right*)
apply (*subgoal-tac Suc* (*i div s*) < *Suc* (*r - Suc 0*))
apply *simp-all*
done

lemma *Suc-rtos-mod-r-1*:⟦*0 < r*; *0 < s*; *i ≤ r * s - Suc 0*; *Suc* (*rtos r s i*) < *r* * *s*; *i mod s = s - Suc 0*; *i div s < r - Suc 0*⟧
⟹ *Suc* (*rtos r s i*) *mod r* = *Suc* (*i div s*)
apply (*simp add*:*rtos-def*)
done

lemma *i-div-s-less*:⟦*0 < r*; *0 < s*; *i ≤ r * s - Suc 0*; *Suc* (*rtos r s i*) < *r * s*; *i mod s = s - Suc 0*; *Suc i < s * r* ⟧ ⟹ *i div s < r - Suc 0*
apply (*frule le-less-trans* [*of i r * s - Suc 0 r * s*], *simp*)
apply (*frule-tac r = r* **and** *s = s* **and** *l = i* **in** *div-Tr2*, *assumption*+)
apply (*simp add*:*mult.commute*)
apply (*rule contrapos-pp*, *simp*+,
subgoal-tac i div s = r - Suc 0,
thin-tac i div s = r - Suc 0)
apply (*simp add*:*rtos-def*,
subgoal-tac (*s - Suc 0*) * *r* + *r* = *r * s*, *simp*)
apply (*thin-tac* (*s - Suc 0*) * *r* + *r* < *r * s*)
apply (*simp add*:*mult.commute*, *simp add*:*diff-mult-distrib2*)
apply *simp*
done

lemma *rtos-mod-r-1*:⟦ *0 < r*; *0 < s*; *i ≤ r * s - Suc 0*; *rtos r s i < r * s*; *i mod s = s - Suc 0* ⟧ ⟹ *rtos r s i mod r = i div s*
apply (*simp add*:*rtos-def*)
apply (*frule le-less-trans* [*of i r * s - Suc 0 r * s*], *simp*)
apply *simp*
apply (*rule rfn-tool20*, *assumption*+, *simp*)
apply (*frule-tac r = r* **and** *s = s* **and** *l = i* **in** *div-Tr2*, *assumption*+)
apply (*rule le-less-trans*, *assumption*+, *simp add*:*mult.commute*)
apply (*rule le-less-trans*, *assumption*+, *simp*)
done

lemma *Suc-i-mod-s-0-1*:⟦*0 < r*; *0 < s*; *i ≤ r * s - Suc 0*; *i mod s = s - Suc 0*⟧
⟹ *Suc i mod s = 0*
apply (*insert mod-div-equality* [*of i s*, *THEN sym*])
apply *simp*
apply (*thin-tac i mod s = s - Suc 0*)
apply (*subgoal-tac Suc i mod s = Suc* (*i div s * s + s - Suc 0*) *mod s*)
apply (*thin-tac i = i div s * s + s - Suc 0*, *simp del*: *add-Suc*)
apply (*subgoal-tac Suc* (*i div s * s + s - Suc 0*) *mod s* = (*i div s * s + s*) *mod*

```
s)
 apply (simp del: add-Suc)
 apply (subgoal-tac Suc (i div s * s + s - Suc 0) = i div s * s + s)
 apply (simp del: add-Suc)
apply (rule Suc-pred [of i div s * s + s], simp)
done


lemma Suci-div-s-2:⟦0 < r; 0 < s; i ≤ r * s - Suc 0; i mod s < s - Suc 0⟧
       ⟹ Suc i div s = i div s
apply (rule div-Tr3-2 [THEN sym], assumption+)
 apply simp
done

lemma rtos-i-mod-r-2:⟦0 < r; 0 < s; i ≤ r * s - Suc 0⟧ ⟹ rtos r s i mod r =
i div s
apply (simp add:rtos-def)
apply (frule le-less-trans [of i r * s - Suc 0 r * s], simp)
apply simp
apply (frule-tac  r = r and s = s and l = i in div-Tr2, assumption+)
 apply (simp add:mult.commute)
apply (subgoal-tac i div s < r)
 apply (rule rfn-tool20, assumption+, simp)
 apply assumption
 apply (rule le-less-trans, assumption+, simp)
done

lemma Suc-rtos-i-mod-r-2:⟦0 < r; 0 < s; i ≤ r * s - Suc 0;
      i div s = r - Suc 0⟧ ⟹ Suc (rtos r s i) mod r = 0
apply (frule le-less-trans [of i r * s - Suc 0 r * s], simp)
apply (simp add:rtos-def)
done

lemma Suc-rtos-i-mod-r-3:⟦0 < r; 0 < s; i ≤ r * s - Suc 0;
     i div s < r - Suc 0⟧ ⟹ Suc (rtos r s i) mod r = Suc (i div s)
apply (frule le-less-trans [of i r * s - Suc 0 r * s], simp)
apply (simp add:rtos-def)
done

lemma Suc-rtos-div-r-3:⟦0 < r; 0 < s; i mod s < s - Suc 0; i ≤ r * s - Suc 0;
     Suc (rtos r s i) < r * s; i div s < r - Suc 0⟧ ⟹
           Suc (rtos r s i) div r = i mod s
apply (simp add:rtos-def)
apply (frule le-less-trans [of i r * s - Suc 0 r * s])
apply simp
apply simp
apply (subst add-Suc-right[THEN sym, of i mod s * r i div s])
 apply (subst add.commute[of i mod s * r Suc (i div s)])
 apply (frule Suc-leI[of i div s r - Suc 0])
```

apply (*frule le-less-trans*[*of Suc* (*i div s*) *r* − *Suc 0 r*], *simp*)
apply (*subst div-mult-self1* [*of r Suc* (*i div s*) *i mod s*])
apply *auto*
done

lemma *r-s-div-s*:⟦*0* < *r*; *0* < *s*⟧ ⟹ (*r* ∗ *s* − *Suc 0*) *div s* = *r* − *Suc 0*
apply (*frule rfn-tool1-1* [*of s s* − *Suc 0 r* − *Suc 0*])
apply *simp*
apply (*simp add*:*diff-mult-distrib*)
done

lemma *r-s-mod-s*:⟦*0* < *r*; *0* < *s*⟧ ⟹ (*r* ∗ *s* − *Suc 0*) *mod s* = *s* − *Suc 0*
apply (*frule rfn-tool20* [*of s* (*r* − *Suc 0*) ∗ *s* + (*s* − *Suc 0*) *r* − *Suc 0*
s − *Suc 0*], *simp*, *simp*)
apply (*simp add*:*diff-mult-distrib*)
done

lemma *rtos-r-s*:⟦*0* < *r*; *0* < *s*⟧ ⟹ *rtos r s* (*r* ∗ *s* − *Suc 0*) = *r* ∗ *s* − *Suc 0*
apply (*simp add*:*rtos-def*)
apply (*frule r-s-div-s* [*of r s*], *assumption*+)
apply (*frule r-s-mod-s* [*of r s*], *assumption*+)
apply (*simp*, *simp add*:*diff-mult-distrib*, *simp add*:*mult.commute*)
done

lemma *rtos-rs-1*:⟦ *0* < *r*; *0* < *s*; *rtos r s i* < *r* ∗ *s*;
¬ *Suc* (*rtos r s i*) < *r* ∗ *s*⟧ ⟹ *rtos r s i* = *r* ∗ *s* − *Suc 0*
apply (*frule pos-mult-pos* [*of r s*], *assumption*+)
apply (*frule less-le-diff* [*of rtos r s i r* ∗ *s*])
apply (*simp add*:*nat-not-le-less*[*THEN sym*, *of Suc* (*rtos r s i*) *r* ∗ *s*])
done

lemma *rtos-rs-i-rs*:⟦ *0* < *r*; *0* < *s*; *i* ≤ *r* ∗ *s* − *Suc 0*;
rtos r s i = *r* ∗ *s* − *Suc 0*⟧ ⟹ *i* = *r* ∗ *s* − *Suc 0*
apply (*frule mem-of-Nset* [*of i r* ∗ *s* − *Suc 0*])
apply (*frule rtos-r-s* [*of r s*], *assumption*+)
apply (*frule rtos-inj*[*of r s*], *assumption*+)
apply (*cut-tac n-in-Nsetn*[*of r* ∗ *s* − *Suc 0*])
apply (*simp add*:*inj-on-def*)
done

lemma *JHS-Tr1-1*:⟦*Group G*; *0* < *r*; *0* < *s*; *compseries G r f*; *compseries G s g*⟧
⟹ *f* (*Suc* ((*r* ∗ *s* − *Suc 0*) *div s*)) ⋄$_G$ (*f* ((*r* ∗ *s* − *Suc 0*) *div s*) ∩ *g* ((*r* ∗ *s* − *Suc 0*) *mod s*)) = *f* (*r* − *Suc 0*) ∩ *g* (*s* − *Suc 0*)
apply (*frule r-s-div-s* [*of r s*], *assumption*+)
apply (*frule r-s-mod-s* [*of r s*], *assumption*+)
apply *simp*
apply (*subst Group.compseriesTr3* [*of G r f*], *assumption*+)
apply (*frule Group.compseriesTr0* [*of G r f r* − *Suc 0*], *assumption*+)
apply (*simp add*:*less-mem-of-Nset*)

apply (*frule Group.compseriesTr0* [*of G s g s − Suc 0*], *assumption*+)
apply (*simp add*:*less-mem-of-Nset*)
apply (*frule Group.inter-sgs* [*of G f* (*r* − (*Suc 0*)) *g* (*s* − *Suc 0*)],
assumption+)
apply (*rule Group.s-top-l-unit*, *assumption*+)
done

lemma *JHS-Tr1-2*:⟦*Group G*; *0* < *r*; *0* < *s*; *compseries G r f*; *compseries G s g*;
k < *r* − *Suc 0*⟧ ⟹ ((*Gp G* (*f* (*Suc k*) ⋄$_G$ (*f k* ∩ *g* (*s* − *Suc 0*)))) /
(*f* (*Suc* (*Suc k*)) ⋄$_G$ (*f* (*Suc k*) ∩ *g 0*))) ≅
((*Gp G* (*g s* ⋄$_G$ (*g* (*s* − *Suc 0*) ∩ *f k*))) /
(*g s* ⋄$_G$ (*g* (*s* − *Suc 0*) ∩ *f* (*Suc k*))))
apply (*frule Group.compseriesTr0* [*of G r f k*], *assumption*+)
apply (*frule less-trans* [*of k r* − *Suc 0 r*]) **apply** *simp*
apply *simp*
apply (*frule Suc-leI* [*of k r* − *Suc 0*])
apply (*frule le-less-trans* [*of Suc k r* − *Suc 0 r*]) **apply** *simp*
apply (*frule Group.compseriesTr0* [*of G r f Suc k*], *assumption*+)
apply *simp*
apply (*thin-tac Suc k* ≤ *r* − *Suc 0*)
apply (*frule Suc-leI*[*of Suc k r*])
apply (*frule Group.compseriesTr0* [*of G r f Suc* (*Suc k*)], *assumption*+)
apply (*frule Group.compseriesTr0* [*of G s g s* − *Suc 0*], *assumption*+)
apply *simp*
apply (*subst Group.compseriesTr2* [*of G s g*], *assumption*+)
apply (*subst Group.compseriesTr3* [*of G s g*], *assumption*+)
apply (*subst Int-absorb2* [*of f* (*Suc k*) *carrier G*])
apply (*rule Group.sg-subset*, *assumption*+)
apply (*subst Group.K-absorb-HK*[*of G f* (*Suc* (*Suc k*)) *f* (*Suc k*)], *assumption*+)

apply (*rule Group.compseriesTr5* [*of G r f Suc k*], *assumption*+)
apply (*frule Suc-leI* [*of k r* − *Suc 0*])
apply *simp*
apply (*subst Group.s-top-l-unit*[*of G g* (*s* − *Suc 0*) ∩ *f k*], *assumption*+)
apply (*simp add*:*Group.inter-sgs*)
apply (*subst Group.compseriesTr3*[*of G s g*], *assumption*+)
apply (*subst Group.s-top-l-unit*[*of G g* (*s* − *Suc 0*) ∩ *f* (*Suc k*)], *assumption*+)
apply (*simp add*:*Group.inter-sgs*)
apply (*frule Group.Group-Gp* [*of G f k*], *assumption*+)
apply (*frule Group.compser-nsg* [*of G r f k*], *assumption*+)
apply (*simp add*:*less-mem-of-Nset* [*of k r* − *Suc 0*])
apply (*frule isom4b* [*of Gp G* (*f k*) *f* (*Suc k*) *f k* ∩ *g* (*s* − *Suc 0*)],
assumption+)
apply (*rule Group.sg-sg*, *assumption*+)
apply (*rule Group.inter-sgs*, *assumption*+)
apply (*simp add*:*Int-lower1*)
apply (*frule Group.compseriesTr5*[*of G r f k*], *assumption*+)
apply *simp*
apply (*frule Group.s-top-induced*[*of G f k f* (*Suc k*) *f k* ∩ *g* (*s* − *Suc 0*)], *as-*

sumption+)
apply (*simp add*:*Int-lower1*) **apply** *simp*
apply (*thin-tac f* (*Suc k*) $\diamond_{\natural_G(f\ k)}$ (*f k* ∩ *g* (*s* − *Suc 0*)) =
f (*Suc k*) $\diamond_G$ (*f k* ∩ *g* (*s* − *Suc 0*)))
apply (*frule Suc-pos* [*of Suc k r*])
apply (*frule Group.cmp-rfn0* [*of G r s f g k s* − *Suc 0*], *assumption+*)
apply *simp*
apply *simp*
apply (*frule Group.sg-inc-set-mult*[*of G f k f* (*Suc k*) *f k* ∩ *g* (*s* − *Suc 0*)], *assumption+*) **apply** (*simp add*:*Int-lower1*)

apply (*simp add*:*Group.Gp-inherited* [*of G f* (*Suc k*) $\diamond_G$ (*f k* ∩ *g* (*s* − *Suc 0*)) *f k*])
apply (*frule Group.inter-sgs* [*of G f k g* (*s* − *Suc 0*)], *assumption+*)
apply (*frule Group.Gp-inherited* [*of G f k* ∩ *g* (*s* − *Suc 0*) *f k*], *assumption+*)
apply (*rule Int-lower1*) **apply** *simp*
apply (*thin-tac* (*Gp* (*Gp G* (*f k*)) ((*f k*) ∩ (*g* (*s* − *Suc 0*)))) =
(*Gp G* ((*f k*) ∩ (*g* (*s* − *Suc 0*)))))
apply (*thin-tac f* (*Suc k*) $\diamond_G$ *f k* ∩ *g* (*s* − *Suc 0*) ⊆ *f k*)
apply (*thin-tac G* ≫ *f* (*Suc k*) $\diamond_G$ *f k* ∩ *g* (*s* − *Suc 0*))
apply (*simp add*:*Int-assoc* [*of f k g* (*s* − *Suc 0*) *f* (*Suc k*)])
apply (*simp add*:*Int-commute* [*of g* (*s* − *Suc 0*) *f* (*Suc k*)])
apply (*simp add*:*Int-assoc* [*of f k f* (*Suc k*) *g* (*s* − *Suc 0*), *THEN sym*])
apply (*simp add*:*Int-absorb1*[*of f* (*Suc k*) *f k*])
apply (*simp add*:*Int-commute*)
done

lemma *JHS-Tr1-3*:⟦*Group G*; *0* < *r*; *0* < *s*; *compseries G r f*; *compseries G s g*;
i ≤ *s* ∗ *r* − *Suc 0*; *Suc* (*rtos r s i*) < *s* ∗ *r*; *Suc i* < *s* ∗ *r*;
i mod s < *s* − *Suc 0*; *Suc i div s* ≤ *r* − *Suc 0*; *i div s* = *r* − *Suc 0*⟧
⟹ *Group* (*Gp G* (*f r* $\diamond_G$ (*f* (*r* − *Suc 0*) ∩ *g* (*i mod s*))) /
(*f r* $\diamond_G$ (*f* (*r* − *Suc 0*) ∩ *g* (*Suc* (*i mod s*)))))
apply (*frule nat-eq-le*[*of i div s r* − *Suc 0*])
apply (*frule Group.compser-nsg* [*of G r f i div s*], *assumption+*)
apply *simp*
apply (*frule Group.compser-nsg* [*of G s g i mod s*], *assumption+*)
apply *simp*
apply (*frule Group.compseriesTr0* [*of G r f r* − *Suc 0*], *assumption+*)
apply *simp*
apply (*frule Group.compseriesTr0* [*of G r f r*], *assumption+*)
apply *simp*
apply (*frule Group.compseriesTr0* [*of G s g i mod s*], *assumption+*)
apply (*simp add*:*less-imp-le*)
apply (*frule Group.compseriesTr0* [*of G s g Suc* (*i mod s*)], *assumption+*)
apply (*frule Suc-mono* [*of i mod s s* − (*Suc 0*)],
simp add:*less-mem-of-Nset*)
apply (*frule Group.cmp-rfn0* [*of G r s f g i div s i mod s*],
assumption+, *simp*, *simp*)
apply (*frule Group.ZassenhausTr3* [*of G f* (*r* − *Suc 0*) *f r g* (*i mod s*)

g (Suc (i mod s))], *assumption+*, *simp*, *simp*)
apply (*cut-tac Group.Group-Gp* [*of G f r* $\diamond_G$ (*f (r − Suc 0*) ∩ *g (i mod s)*)])
apply (*rule Group.Group-Qg*, *assumption+*)
apply *simp*
done

lemma *JHS-Tr1-4*:⟦*Group G*; *0 < r*; *0 < s*; *compseries G r f*; *compseries G s g*;
*i ≤ s * r − Suc 0*; *Suc (rtos r s i) < s * r*; *Suc i < s * r*;
i mod s < s − Suc 0; *Suc i div s ≤ r − Suc 0*; *i div s = r − Suc 0*⟧ ⟹
Group (Gp G (g (Suc (i mod s)) $\diamond_G$ *(g (i mod s) ∩ f (r − Suc 0))) /*
(g (Suc (Suc (i mod s))) $\diamond_G$ *(g (Suc (i mod s)) ∩ f 0)))*
apply (*subst Group.compseriesTr2* [*of G r f*], *assumption+*)
apply (*frule Group.compseriesTr0* [*of G s g Suc (i mod s)*], *assumption+*)
apply (*frule Suc-mono* [*of i mod s s − Suc 0*], *simp*)
apply (*frule Group.sg-subset* [*of G g (Suc (i mod s))*], *assumption+*)
apply (*subst Int-absorb2*, *assumption+*)
apply (*frule Suc-mono* [*of i mod s s − Suc 0*])
apply (*frule less-mem-of-Nset* [*of i mod s s − Suc 0*], *simp*)
apply (*frule Suc-leI* [*of Suc (i mod s) s*])
apply (*frule Group.compseriesTr0* [*of G s g Suc (Suc (i mod s))*],
assumption+)
apply (*frule less-le-diff* [*of Suc (i mod s) s*])
apply (*frule Suc-pos* [*of Suc (i mod s) s*])
apply (*frule Group.compseriesTr5*[*of G s g Suc (i mod s)*], *assumption+*)
apply (*subst Group.K-absorb-HK*[*of G g (Suc (Suc (i mod s)))*
g (Suc (i mod s))], *assumption+*)
apply (*frule Group.compseriesTr0* [*of G s g i mod s*], *assumption+*)
apply (*frule mod-less-divisor* [*of s i*], *simp*)
apply (*frule Group.cmp-rfn0* [*of G s r g f i mod s r − Suc 0*],
assumption+, *simp*, *simp*, *simp*)
apply (*cut-tac Group.compser-nsg* [*of G s g i mod s*])
apply (*frule Group.ZassenhausTr4-1* [*of G g (i mod s) g (Suc (i mod s))*
f (r − Suc 0)], *assumption+*)
apply (*rule Group.sg-sg*, *assumption+*)
apply (*rule Group.inter-sgs*, *assumption+*)
apply (*rule Group.compseriesTr0* [*of G r f r − Suc 0*], *assumption+*)
apply *simp*
apply (*rule Int-lower1*)
apply (*rule Group.Group-Qg*)
apply (*rule Group.Group-Gp*, *assumption+*, *simp+*)
done

lemma *JHS-Tr1-5*:⟦*Group G*; *0 < r*; *0 < s*; *compseries G r f*; *compseries G s g*;
*i ≤ s * r − Suc 0*; *Suc (rtos r s i) < s * r*; *Suc i < s * r*;
i mod s < s − Suc 0; *i div s < r − Suc 0*⟧
⟹ (*Gp G (f (Suc (i div s))* $\diamond_G$ *(f (i div s) ∩ g (i mod s))) /*
(f (Suc (i div s)) $\diamond_G$ *(f (i div s) ∩ g (Suc (i mod s))))) ≅*
(Gp G (g (Suc (i mod s)) $\diamond_G$ *(g (i mod s) ∩ f (i div s))) /*
(g (Suc (Suc (rtos r s i) div r)) $\diamond_G$

```
            (g (Suc (rtos r s i) div r) ∩ f (Suc (rtos r s i) mod r))))
apply (frule Group.compseriesTr0 [of G s g i mod s], assumption+)
 apply simp
apply (frule Group.compseriesTr0 [of G s g Suc (i mod s)], assumption+)
apply (frule Suc-mono [of i mod s s − Suc 0], simp)
apply (frule Group.compseriesTr0 [of G r f i div s], assumption+)
 apply (frule less-trans [of i div s r − Suc 0 r], simp)
 apply simp
apply (frule Group.compseriesTr0 [of G r f Suc (i div s)], assumption+)
 apply (frule Suc-mono [of i div s r − Suc 0])
 apply simp
 apply (frule Group.compser-nsg [of G r f i div s], assumption+)
 apply simp
 apply (frule Group.compser-nsg [of G s g i mod s], assumption+, simp)
apply (subst Suc-rtos-i-mod-r-3 [of r s i], assumption+)
 apply (simp add:mult.commute, assumption)
 apply (subst Suc-rtos-div-r-3 [of r s i ], assumption+)+
 apply (simp add:mult.commute)+
 apply (subst Suc-rtos-div-r-3[of r s i], assumption+)
apply (rule Zassenhaus [of G f (i div s) f (Suc (i div s)) g (i mod s)
 g (Suc (i mod s))], assumption+)
done

lemma JHS-Tr1-6: ⟦Group G; 0 < r; 0 < s; compseries G r f; compseries G s g;
 i ≤ r ∗ s − Suc 0; Suc (rtos r s i) < r ∗ s⟧ ⟹
  ((Gp G (cmp-rfn G r f s g i)) / (cmp-rfn G r f s g (Suc i))) ≅
  ((Gp G (g (Suc (rtos r s i div r)) ⋄G
      (g (rtos r s i div r) ∩ f (rtos r s i mod r)))) /
      (g (Suc (Suc (rtos r s i) div r)) ⋄G
            (g (Suc (rtos r s i) div r) ∩ f (Suc (rtos r s i) mod r))))
apply (simp add:cmp-rfn-def)
 apply (frule le-less-trans [of i r ∗ s − Suc 0 r ∗ s], simp)
 apply (simp add:mult.commute [of r s])
 apply (frule Suc-leI [of i s ∗ r], thin-tac i < s ∗ r)
apply (case-tac ¬ Suc i < s ∗ r, simp)

 apply (frule rfn-tool17 [of Suc i s ∗ r Suc 0])
 apply (thin-tac  Suc i = s ∗ r)
 apply simp
 apply (frule rtos-r-s [of r s], assumption+)
 apply (simp add:mult.commute [of r s])
apply simp
apply (frule mod-less-divisor [of s i])
apply (frule less-le-diff [of i mod s s], thin-tac i mod s < s)
 apply (case-tac i mod s = s − Suc 0, simp)
 apply (frule-tac div-Tr2 [of r s Suc i], assumption+)
 apply (simp add:le-imp-less-or-eq)
 apply (subst div-Tr3-1[of r s i], assumption+, simp)
 apply (subst rtos-hom3 [of r s i], assumption+)
```

apply (*simp add*:*mult.commute*)
apply (*subst rtos-hom3-1* [*of r s i*], *assumption*+)
apply (*simp add*:*mult.commute*)
apply (*frule div-Tr3-1* [*of r s i*], *assumption*+, *simp*)
apply (*simp*, *thin-tac Suc i div s* = *Suc* (*i div s*))
apply (*insert n-less-Suc* [*of i div s*])
apply (*frule less-le-trans* [*of i div s Suc* (*i div s*) *r* − *Suc 0*], *assumption*+)
apply (*subst Suc-rtos-div-r-1* [*of r s i*], *assumption*+)
apply (*simp add*:*mult.commute*[*of s r*])+
apply (*subst Suc-rtos-mod-r-1* [*of r s i*], *assumption*+)
apply (*subst Suc-i-mod-s-0-1* [*of r s i*], *assumption*+)
apply (*simp only*:*Suc-rtos-div-r-1* [*of r s i*])
apply (*subst rtos-hom3*[*of r s i*], *assumption*+, *simp*)

apply (*frule JHS-Tr1-2* [*of G r s f g i div s*], *assumption*+, *simp*)
apply (*frule noteq-le-less* [*of i mod s s* − *Suc 0*], *assumption*+)
apply (*thin-tac i mod s* ≤ *s* − *Suc 0*)
apply (*thin-tac i mod s* ≠ *s* − *Suc 0*)
apply (*frule div-Tr2* [*of r s Suc i*], *assumption*+, *rule noteq-le-less*, *assumption*+)
apply (*subst div-Tr3-2* [*THEN sym*, *of r s i*], *assumption*+)
apply *simp*
apply (*subst rfn-tool12-1* [*THEN sym*, *of s i*], *assumption*+)
apply *simp*
apply (*subst rtos-hom3* [*of r s i*], *assumption*+)
apply (*simp add*:*mult.commute*)
apply (*subst rtos-hom3-1* [*of r s i*], *assumption*+)
apply (*simp add*:*mult.commute*)
apply (*case-tac i div s* = *r* − *Suc 0*)
apply (*subst rtos-hom5* [*of r s i*], *assumption*+)
apply (*simp add*:*mult.commute*)
apply *assumption*
apply (*subst rtos-hom7* [*of r s i*], *assumption*+)
apply (*simp add*:*mult.commute*)
apply (*assumption*, *simp*)
apply (*frule JHS-Tr1-3* [*of G r s f g i*], *assumption*+, *simp only*:*noteq-le-less*, *assumption*+)
apply (*frule JHS-Tr1-4* [*of G r s f g i*], *assumption*+, *simp only*:*noteq-le-less*, *assumption*+)
apply (*subst rtos-hom3* [*of r s i*], *assumption*+, *simp only*:*mult.commute*[*of s r*])
apply (*subst rtos-hom5* [*of r s i*], *assumption*+, *simp only*:*mult.commute*[*of s r*], *simp add*:*mem-of-Nset*)
apply (*rule isomTr1*, *assumption*+)
apply (*frule Suci-div-s-2*[*of r s i*], *assumption*+, *simp only*:*mult.commute*, *assumption*)

apply *simp*
apply (*frule Suci-div-s-2*[*of r s i*], *assumption*+,
simp only:*mult.commute*, *assumption*, *simp*)
apply (*rule JHS-Tr1-2* [*of G s r g f i mod s*], *assumption*+)
apply (*frule div-Tr2* [*of r s i*], *assumption*+)
apply (*rule le-less-trans* [*of i s * r − Suc 0 s * r*], *assumption*+)
apply *simp*
apply (*frule noteq-le-less* [*of i div s r − Suc 0*], *assumption*+)
apply (*frule Suci-div-s-2*[*of r s i*], *assumption*+,
simp only:*mult.commute*[*of s r*], *assumption*, *simp*)
apply (*frule JHS-Tr1-5*[*of G r s f g i*], *assumption*+)
apply (*simp add*:*noteq-le-less*[*of Suc i*], *assumption*+)
apply (*frule mem-of-Nset*[*of i s*r − Suc 0*],
simp add:*mult.commute*[*of s r*])
apply (*simp add*:*rtos-hom3* [*of r s i*])
done

lemma *JHS-Tr1*:⟦ *Group G*; *0 < r*; *0 < s*; *compseries G r f*; *compseries G s g*⟧
⟹ *isom-Gchains* (*r * s − 1*) (*rtos r s*) (*Qw-cmpser G* (*cmp-rfn G r f s g*))
(*Qw-cmpser G* (*cmp-rfn G s g r f*))
apply (*simp add*:*isom-Gchains-def*)
apply (*rule allI*, *rule impI*)
apply (*frule pos-mult-pos* [*of r s*], *assumption*+)
apply (*frule-tac b = r * s* **and** *a = i* **in** *rfn-tool11*, *assumption*+)
apply (*simp add*:*Qw-cmpser-def*)
apply (*simp only*:*cmp-rfn-def* [*of G s g*])
apply (*frule-tac l = i* **in** *rtos-hom1* [*of r s*], *assumption*+)
apply (*frule-tac x = rtos r s i* **and** *y = s * r − Suc 0* **and** *z = s * r* **in**
le-less-trans, *simp*)
apply (*simp add*:*mult.commute* [*of s r*])
apply (*case-tac Suc* (*rtos r s i*) *< r * s*, *simp*)
prefer *2* **apply** *simp*
apply (*frule-tac i = i* **in** *rtos-rs-1* [*of r s*], *assumption*+)
apply (*frule-tac i = i* **in** *rtos-rs-i-rs* [*of r s*], *assumption*+)
apply (*rule less-le-diff*, *assumption*+)
apply (*simp add*:*cmp-rfn-def*)
apply (*simp add*:*mult.commute*)
apply (*subst JHS-Tr1-1* [*of G r s f g*], *assumption*+)
apply (*frule JHS-Tr1-1* [*of G s r g f*], *assumption*+)
apply (*simp add*:*mult.commute* [*of r s*])
apply (*simp add*:*Int-commute*)
apply (*frule Group.compseriesTr0* [*of G r f r − Suc 0*], *assumption*+,
simp)
apply (*frule Group.compseriesTr0* [*of G s g s − Suc 0*], *assumption*+,
simp)
apply (*frule Group.inter-sgs* [*of G f* (*r − (Suc 0*)) *g* (*s − Suc 0*)],
assumption+)
apply (*frule Group.special-sg-e* [*of G*])
apply (*frule Group.special-nsg-e* [*of G f* (*r − Suc 0*) ∩ *g* (*s − Suc 0*)],

assumption+)
apply (*frule Group.Group-Gp* [*of G f* (*r* − *Suc 0*) ∩ *g* (*s* − *Suc 0*)], *assumption*+)
apply (*frule Group.Group-Qg*[*of Gp G* (*f* (*r* − *Suc 0*) ∩ *g* (*s* − *Suc 0*)) {$\mathbf{1}_G$}],
assumption+)
apply (*simp add*:*isomTr0*[*of* ($\natural_G$(*f* (*r* − *Suc 0*) ∩ *g* (*s* − *Suc 0*))) / {$\mathbf{1}_G$}])
apply (*rule JHS-Tr1-6*, *assumption*+)
done

lemma *abc-SucTr0*:⟦(*0*::*nat*) < *a*; *c* ≤ *b*; *a* − *Suc 0* = *b* − *c*⟧ ⟹ *a* = (*Suc b*) − *c*
apply (*subgoal-tac Suc 0* ≤ *a*)
apply (*frule le-add-diff-inverse2* [*of Suc 0 a*, *THEN sym*])
apply *auto*
done

lemma *length-wcmpser0-0*:⟦*Group G*; *Ugp E*; *w-cmpser G* (*Suc 0*) *f*⟧ ⟹
f ‘ {*i*. *i* ≤ (*Suc 0*)} = {*f 0*, *f* (*Suc 0*)}
apply (*simp add*:*Nset-1*)
done

lemma *length-wcmpser0-1*:⟦*Group G*; *Ugp E*; *w-cmpser G* (*Suc n*) *f*; *i*∈{*i*. *i* ≤ *n*};
(*Qw-cmpser G f*) *i* ≅ *E*⟧ ⟹ *f i* = *f* (*Suc i*)
apply (*subgoal-tac 0* < *Suc n*)
apply (*frule Group.w-cmpser-gr* [*of G Suc n f i*], *assumption*+)
prefer *2* **apply** *simp*
apply (*frule Group.w-cmpser-gr* [*of G Suc n f Suc i*], *simp*+)
apply (*frule Group.w-cmpser-ns* [*of G Suc n f i*], *simp*+)
apply (*simp add*:*Qw-cmpser-def*)
apply (*rule QgrpUnit-3* [*of G E f i f* (*Suc i*)], *assumption*+, *simp*+)
done

lemma *length-wcmpser0-2*:⟦*Group G*; *Ugp E*; *w-cmpser G* (*Suc n*) *f*; *i* ≤ *n*;
¬ (*Qw-cmpser G f*) *i* ≅ *E*⟧ ⟹ *f i* ≠ *f* (*Suc i*)
apply (*cut-tac zero-less-Suc*[*of n*])
apply (*frule Group.w-cmpser-gr* [*of G Suc n f i*], *assumption*+)
apply *simp*
apply (*frule Group.w-cmpser-gr* [*of G Suc n f Suc i*], *assumption*+)
apply *simp*
apply (*frule Group.w-cmpser-ns* [*of G Suc n f i*], *assumption*+, *simp*)
apply (*simp add*:*Qw-cmpser-def*)
apply (*rule QgrpUnit-4* [*of G E f i f* (*Suc i*)], *assumption*+)
done

lemma *length-wcmpser0-3*:⟦*Group G*; *Ugp E*; *w-cmpser G* (*Suc* (*Suc n*)) *f*;
f (*Suc n*) ≠ *f* (*Suc* (*Suc n*))⟧ ⟹ *f* (*Suc* (*Suc n*)) ∉ *f* ‘ {*i*. *i* ≤ (*Suc n*)}
apply (*frule Group.w-cmpser-is-d-gchain*, *assumption*+)
apply (*thin-tac w-cmpser G* (*Suc* (*Suc n*)) *f*)
apply (*rule contrapos-pp*, *simp*+)

apply (*frule Group.d-gchainTr2* [*of G Suc* ((*Suc n*)) *f Suc n Suc* (*Suc n*)])
apply *simp* **apply** *assumption*+ **apply** *simp*+
apply (*frule psubsetI* [*of f* (*Suc* (*Suc n*)) *f* (*Suc n*)])
apply (*rule not-sym*, *assumption*+)
apply (*thin-tac f* (*Suc* (*Suc n*)) ⊆ *f* (*Suc n*))
apply (*simp add*:*image-def*)
apply (*erule exE*)
apply (*cut-tac zero-less-Suc*[*of Suc n*])
apply (*frule-tac f* = *f* **and** *l* = *x* **in** *Group.d-gchainTr2* [*of G Suc* ((*Suc n*))
- - *Suc* (*Suc n*)], *assumption*+)
apply *simp*+
apply (*frule-tac f* = *f* **and** *l* = *x* **in** *Group.d-gchainTr2* [*of G Suc* (*Suc n*)
- - *Suc n*], *simp*+)
done

lemma *length-wcmpser0-4*:⟦*Group G*; *Ugp E*; *w-cmpser G* (*Suc 0*) *f*⟧ ⟹
card (*f* ‘ {*i*. *i* ≤ *Suc 0*}) − *1* = *Suc 0* − *card* {*i*. *i* = *0* ∧
Qw-cmpser G f i ≅ *E*}

apply (*auto simp add*: *length-wcmpser0-0 Collect-conv-if*)
apply (*frule-tac n* = *0* **and** *f* = *f* **and** *i* = *0* **in** *length-wcmpser0-1* [*of G E*],
assumption+, *simp*+)
apply (*frule-tac f* = *f* **and** *i* = *0* **in** *length-wcmpser0-2* [*of G E 0*],
(*assumption* | *simp*)+)
done

lemma *length-wcmpser0-5*: ⟦*Group G*; *Ugp E*; *w-cmpser G* (*Suc* (*Suc n*)) *f*;
w-cmpser G (*Suc n*) *f*;
card (*f* ‘ {*i*. *i* ≤ (*Suc n*)}) − *1* = *Suc n* −
card {*i*. *i* ≤ *n* ∧ *Qw-cmpser G f i* ≅ *E*};
Qw-cmpser G f (*Suc n*) ≅ *E*⟧ ⟹
card (*f* ‘ {*i* . *i* ≤ (*Suc* (*Suc n*))}) − *1* =
Suc (*Suc n*) − *card* {*i*. *i* ≤ (*Suc n*) ∧ *Qw-cmpser G f i* ≅ *E*}
apply (*frule-tac n* = *Suc n* **and** *f* = *f* **and** *i* = *Suc n* **in**
length-wcmpser0-1 [*of G E*], *assumption*+)
apply (*simp*, *assumption*)
apply (*subgoal-tac f* ‘ {*i*. *i* ≤ (*Suc* (*Suc n*))} = *f* ‘ {*i*. *i* ≤ (*Suc n*)})
apply *simp*
prefer *2* **apply** (*rule equalityI*)
apply (*simp add*:*image-def*)
apply (*auto del*:*equalityI*)
apply (*case-tac xa* = *Suc* (*Suc n*), *simp*)
apply (*thin-tac xa* = *Suc* (*Suc n*), *rotate-tac* −*2*)
apply (*frule sym*, *thin-tac f* (*Suc n*) = *f* (*Suc* (*Suc n*)),
simp, *thin-tac f* (*Suc* (*Suc n*)) = *f* (*Suc n*))
apply (*cut-tac n-in-Nsetn*[*of Suc n*], *blast*)
apply (*frule-tac m* = *xa* **and** *n* = *Suc* (*Suc n*) **in** *noteq-le-less*, *assumption*,
frule-tac x = *xa* **in** *Suc-less-le*[*of* - *Suc n*], *blast*)
apply (*subgoal-tac* {*i*. *i* ≤ (*Suc n*) ∧ *Qw-cmpser G f i* ≅ *E*} =

$insert\ (Suc\ n)\ \{i.\ i \leq n \wedge Qw\text{-}cmpser\ G\ f\ i \cong E\}$)
apply *simp*
apply *fastforce*
done

lemma *length-wcmpser0-6*:⟦*Group G*; *w-cmpser G* (*Suc* (*Suc n*)) *f*⟧ ⟹
$0 < card\ (f\ `\ \{i.\ i \leq (Suc\ n)\})$
apply (*insert finite-Collect-le-nat* [*of Suc n*])
apply (*frule finite-imageI* [*of* $\{i.\ i \leq (Suc\ n)\}$ *f*])
apply (*subgoal-tac* $\{f\ 0\} \subseteq f\ `\ \{i.\ i \leq (Suc\ n)\}$)
apply (*frule card-mono* [*of f* $`\ \{i.\ i \leq (Suc\ n)\}\ \{f\ 0\}$], *assumption*+)
apply (*simp add*:*card1* [*of f 0*])
apply (*rule subsetI*, *simp add*:*image-def*)
apply (*subgoal-tac* $0 \in \{i.\ i \leq (Suc\ n)\}$, *blast*)
apply *simp*
done

lemma *length-wcmpser0-7*:⟦*Group G*; *w-cmpser G* (*Suc* (*Suc n*)) *f*⟧ ⟹
$card\ \{i.\ i \leq n \wedge Qw\text{-}cmpser\ G\ f\ i \cong E\} \leq Suc\ n$
apply (*insert finite-Collect-le-nat* [*of n*])
apply (*subgoal-tac* $\{i.\ i \leq n \wedge Qw\text{-}cmpser\ G\ f\ i \cong E\} \subseteq \{i.\ i \leq n\}$)
apply (*frule card-mono* [*of* $\{i.\ i \leq n\}$ $\{i.\ i \leq n \wedge Qw\text{-}cmpser\ G\ f\ i \cong E\}$])
apply (*assumption*, *simp*)
apply (*rule subsetI*, *simp add*:*CollectI*)
done

lemma *length-wcmpser0*:⟦*Group G*; *Ugp E*⟧ ⟹∀ *f*. *w-cmpser G* (*Suc n*) *f* ⟶
card (*f* ‘ {*i*. *i* ≤ (*Suc n*)}) − *1* = (*Suc n*) − (*card* {*i*. *i* ≤ *n* ∧
((*Qw-cmpser G f*) *i* ≅ *E*)})
apply (*induct-tac n*)

apply (*rule allI*) **apply** (*rule impI*)
apply (*frule-tac f* = *f* **in** *length-wcmpser0-4*[*of G E*], *assumption*+, *simp*)

apply (*rule allI*) **apply** (*rule impI*)
apply (*frule-tac n* = *Suc n* **and** *f* = *f* **in** *Group.w-cmpser-pre* [*of G*], *assumption*+)
apply (*subgoal-tac card* (*f* ‘ {*i*. *i* ≤ (*Suc n*)}) − *1* =
Suc n − *card* {*i*. *i* ≤ *n* ∧ *Qw-cmpser G f i* ≅ *E*})
prefer *2* **apply** *simp*
apply (*thin-tac* ∀ *f*. *w-cmpser G* (*Suc n*) *f* ⟶
card (*f* ‘ {*i*. *i* ≤ (*Suc n*)}) − *1* =
Suc n − *card* {*i*. *i* ≤ *n* ∧ *Qw-cmpser G f i* ≅ *E*})
apply (*case-tac Qw-cmpser G f* (*Suc n*) ≅ *E*)
apply (*rule length-wcmpser0-5*, *assumption*+)
apply (*frule-tac n* = *Suc n* **and** *f* = *f* **and** *i* = *Suc n* **in**
length-wcmpser0-2 [*of G E*], *assumption*+)
apply *simp* **apply** *assumption*

apply (*subgoal-tac f* ‘ $\{i.\ i \leq (Suc\ (Suc\ n))\} =$
insert (*f* (*Suc* (*Suc n*))) (*f* ‘ $\{i.\ i \leq (Suc\ n)\}$))
apply *simp* **apply** (*thin-tac f* ‘ $\{i.\ i \leq (Suc\ (Suc\ n))\} =$
insert (*f* (*Suc* (*Suc n*))) (*f* ‘ $\{i.\ i \leq (Suc\ n)\}$))
apply (*subgoal-tac finite* (*f* ‘ $\{i.\ i \leq (Suc\ n)\}$))
apply (*subgoal-tac f* (*Suc* (*Suc n*)) $\notin$ *f* ‘ $\{i.\ i \leq (Suc\ n)\}$)
apply (*subst card-insert-disjoint*, *assumption*)
apply *assumption*
prefer *2* **apply** (*rule length-wcmpser0-3*, *assumption+*)
prefer *2*
apply (*subgoal-tac finite* $\{i.\ i \leq (Suc\ n)\}$)
apply (*rule finite-imageI*, *assumption+*, *simp*)
prefer *2*
apply (*thin-tac* $\neg$ *Qw-cmpser G f* (*Suc n*) $\cong$ *E*,
thin-tac w-cmpser G (*Suc n*) *f*,
thin-tac f (*Suc n*) $\neq$ *f* (*Suc* (*Suc n*)))
apply (*subgoal-tac* $\{i.\ i \leq (Suc\ (Suc\ n))\} = \{i.\ i \leq (Suc\ n)\} \cup \{Suc\ (Suc\ n)\}$)
prefer *2* **apply** (*rule-tac n* = *Suc n* **in** *Nset-un*, *simp*)
apply (*subgoal-tac card* $\{i.\ i \leq (Suc\ n) \wedge$ *Qw-cmpser G f i* $\cong E\} =$
card $\{i.\ i \leq n \wedge$ *Qw-cmpser G f i* $\cong E\}$)
apply *simp*
apply (*thin-tac card* $\{i.\ i \leq (Suc\ n) \wedge$ *Qw-cmpser G f i* $\cong E\} =$
card $\{i.\ i \leq n \wedge$ *Qw-cmpser G f i* $\cong E\}$,
thin-tac $\neg$ *Qw-cmpser G f* (*Suc n*) $\cong$ *E*,
thin-tac f (*Suc n*) $\neq$ *f* (*Suc* (*Suc n*)),
thin-tac f (*Suc* (*Suc n*)) $\notin$ *f* ‘ $\{i.\ i \leq (Suc\ n)\}$)
apply (*frule-tac n* = *n* **and** *f* = *f* **in** *length-wcmpser0-6* [*of G*], *assumption+*,
frule-tac n = *n* **and** *f* = *f* **in** *length-wcmpser0-7* [*of G*], *assumption+*)
apply (*rule abc-SucTr0*, *assumption+*)
apply (*rule card-eq*)
apply (*thin-tac card* (*f* ‘ $\{i.\ i \leq (Suc\ n)\}$) $-$ *Suc 0* =
Suc n $-$ *card* $\{i.\ i \leq n \wedge$ *Qw-cmpser G f i* $\cong E\}$,
thin-tac f (*Suc n*) $\neq$ *f* (*Suc* (*Suc n*)),
thin-tac f (*Suc* (*Suc n*)) $\notin$ *f* ‘ $\{i.\ i \leq (Suc\ n)\}$)
apply (*rule equalityI*)
apply (*rule subsetI*, *simp add*:*CollectI*, *erule conjE*)
apply (*case-tac x* = *Suc n*, *simp*, *simp*)
apply (*rule subsetI*, *simp add*:*CollectI*)
done

lemma *length-of-twcmpser*:$[\![$*Group G*; *Ugp E*; *tw-cmpser G* (*Suc n*) *f* $]\!] \Longrightarrow$
length-twcmpser G (*Suc n*) *f* =
(*Suc n*) $-$ (*card* $\{i.\ i \leq n \wedge$ ((*Qw-cmpser G f*) *i* $\cong E)\}$)
apply (*unfold length-twcmpser-def*)
apply (*insert length-wcmpser0* [*of G E n*])
apply (*subgoal-tac w-cmpser G* (*Suc n*) *f*, *rotate-tac* -1,
simp, *simp*,
thin-tac $\forall f.$ *w-cmpser G* (*Suc n*) *f* $\longrightarrow$

```
        card (f ‘ {i. i ≤ (Suc n)}) − Suc 0 =
        Suc n − card {i. i ≤ n ∧ Qw-cmpser G f i ≅ E})
 apply (simp add:tw-cmpser-def w-cmpser-def, erule conjE)
 apply (thin-tac ∀ i≤ n. Gp G (f i) ▷ f (Suc i))
 apply (simp add:td-gchain-def)
done

lemma JHS-1:⟦Group G; Ugp E; compseries G r f; compseries G s g; 0<r; 0 <
s⟧
 ⟹ r =  r ∗ s − card {i. i ≤ (r ∗ s − Suc 0) ∧
          Qw-cmpser G (cmp-rfn G r f s g) i ≅ E}
apply (frule-tac r = r and s = s and G = G and f = f and g = g in
                Group.JHS-Tr0, assumption+)
apply (simp add:wcsr-rfns-def, erule conjE)
apply (frule-tac length-of-twcmpser [of G E r ∗ s − Suc 0
       cmp-rfn G r f s g], assumption+, simp add:mult.commute)
apply (simp add:length-twcmpser-def)
apply (frule Group.rfn-compseries-iM [of G r s f cmp-rfn G r f s g], assumption+,
rule Group.JHS-Tr0 [of G r s f g], assumption+)
apply (simp add:mult.commute [of s r])
done

lemma J-H-S:⟦Group G; Ugp E; compseries G r f; compseries G s g; 0<r;
            (0::nat)<s ⟧ ⟹ r = s
apply (frule JHS-1 [of G E r f s g], assumption+,
       frule JHS-1 [of G E s g r f], assumption+,
       frule JHS-Tr1 [of G r s f g], assumption+,
       frule Group.JHS-Tr0 [of G r s f g], assumption+,
       frule Group.JHS-Tr0 [of G s r g f], assumption+)
 apply (simp add:wcsr-rfns-def, (erule conjE)+,
       frule Group.tw-cmpser-is-w-cmpser [of G s ∗ r cmp-rfn G r f s g],
                                          assumption+,
       frule Qw-cmpser [of G s ∗ r − Suc 0 cmp-rfn G r f s g],
       simp add:pos-mult-pos [of s r])
 apply (frule Group.tw-cmpser-is-w-cmpser [of G r ∗ s cmp-rfn G s g r f],
                                          assumption+,
        frule Qw-cmpser [of G r ∗ s − Suc 0 cmp-rfn G s g r f],
        simp add:pos-mult-pos [of r s],
        simp add:mult.commute [of s r])
apply (frule isom-gch-units [of E r ∗ s − Suc 0
 Qw-cmpser G (cmp-rfn G r f s g) Qw-cmpser G (cmp-rfn G s g r f)
                                          rtos r s], assumption+)
prefer 2 apply simp
apply (simp add:Gch-bridge-def)
apply (rule conjI) apply (rule allI, rule impI)
 apply (frule-tac l = l in rtos-hom1 [of r s], assumption+,
       simp add:mult.commute [of s r])
apply (rule rtos-inj, assumption+)
done
```

end

theory *Algebra4*
imports *Algebra3* ~~/*src/HOL/Binomial* ~~/*src/HOL/Library/Zorn*
begin

3.18 Abelian groups

record $'a$ *aGroup* = $'a$ *carrier* +
  pop :: [$'a$, $'a$] ⇒ $'a$ (**infixl** ±1 *62*)
  mop :: $'a$ ⇒ $'a$ ((−$_{a}$1 -) [*64*]*63*)
  zero :: $'a$ (**0**1)

locale *aGroup* =
  fixes *A* (**structure**)
 assumes
    pop-closed: *pop A* ∈ *carrier A* → *carrier A* → *carrier A*
 and *aassoc* : ⟦*a* ∈ *carrier A*; *b* ∈ *carrier A*; *c* ∈ *carrier A*⟧ ⟹
    (*a* ± *b*) ± *c* = *a* ± (*b* ± *c*)
 and *pop-commute*:⟦*a* ∈ *carrier A*; *b* ∈ *carrier A*⟧ ⟹ *a* ± *b* = *b* ± *a*
 and *mop-closed*:*mop A* ∈ *carrier A* → *carrier A*
 and *l-m* :*a* ∈ *carrier A* ⟹ (−$_{a}$ *a*) ± *a* = **0**
 and *ex-zero*: **0** ∈ *carrier A*
 and *l-zero*:*a* ∈ *carrier A* ⟹ **0** ± *a* = *a*

definition
  b-ag :: - ⇒
   (|*carrier*:: $'a$ *set*, *top*:: [$'a$, $'a$] ⇒ $'a$, *iop*:: $'a$ ⇒ $'a$, *one*:: $'a$ |) **where**
  b-ag A = (|*carrier* = *carrier A*, *top* = *pop A*, *iop* = *mop A*, *one* = *zero A* |)

definition
  asubGroup :: [- , $'a$ *set*] ⇒ *bool* **where**
  asubGroup A H ⟷ (*b-ag A*) ⋟ *H*

definition
  aqgrp :: [- , $'a$ *set*] ⇒
      (| *carrier*::$'a$ *set set*, *pop*::[$'a$ *set*, $'a$ *set*] ⇒ $'a$ *set*,
        mop::$'a$ *set* ⇒ $'a$ *set*, *zero* :: $'a$ *set* |) **where**
  aqgrp A H = (|*carrier* = *set-rcs* (*b-ag A*) *H*,
      pop = λ*X*. λ*Y*. (*c-top* (*b-ag A*) *H X Y*),
      mop = λ*X*. (*c-iop* (*b-ag A*) *H X*), *zero* = *H* |)

definition
  ag-idmap :: - ⇒ ($'a$ ⇒ $'a$) ((*aI*_)) **where**
  aI$_{A}$ = (λ*x*∈*carrier A*. *x*)

abbreviation

ASubG :: [('*a*, '*more*) *aGroup-scheme*, '*a set*] => *bool* (**infixl** +> *58*) **where**
A +> *H* == *asubGroup A H*

definition
Ag-ind :: [- , '*a* ⇒ '*d*] ⇒ '*d aGroup* **where**
Ag-ind A f = (|*carrier* = *f*'(*carrier A*),
pop = λ*x* ∈ *f*'(*carrier A*). λ*y* ∈ *f*'(*carrier A*).
f(((*invfun* (*carrier A*) (*f*'(*carrier A*)) *f*) *x*) ±$_A$
((*invfun* (*carrier A*) (*f*'(*carrier A*)) *f*) *y*)),
mop = λ*x*∈(*f*'(*carrier A*)). *f* (−$_a$ $_A$ ((*invfun* (*carrier A*) (*f*'(*carrier A*)) *f*) *x*)),
zero = *f* (**0**$_A$)|)

definition
Agii :: [- , '*a* ⇒ '*d*] ⇒ ('*a* ⇒ '*d*) **where**
Agii A f = (λ*x*∈*carrier A*. *f x*)

lemma (**in** *aGroup*) *ag-carrier-carrier*:*carrier* (*b-ag A*) = *carrier A*
by (*simp add*:*b-ag-def*)

lemma (**in** *aGroup*) *ag-pOp-closed*:⟦*x* ∈ *carrier A*; *y* ∈ *carrier A*⟧ ⟹
pop A x y ∈ *carrier A*
apply (*cut-tac pop-closed*)
apply (*frule funcset-mem*[*of op* ± *carrier A carrier A* → *carrier A x*],
assumption+)
apply (*rule funcset-mem*[*of op* ± *x carrier A carrier A y*], *assumption*+)
done

lemma (**in** *aGroup*) *ag-mOp-closed*:*x* ∈ *carrier A* ⟹ (−$_a$ *x*) ∈ *carrier A*
apply (*cut-tac mop-closed*)
apply (*rule funcset-mem*[*of mop A carrier A carrier A x*], *assumption*+)
done

lemma (**in** *aGroup*) *asubg-subset*:*A* +> *H* ⟹ *H* ⊆ *carrier A*
apply (*simp add*:*asubGroup-def*)
apply (*simp add*:*sg-def*, (*erule conjE*)+)
apply (*simp add*:*ag-carrier-carrier*)
done

lemma (**in** *aGroup*) *ag-pOp-commute*:⟦*x* ∈ *carrier A*; *y* ∈ *carrier A*⟧ ⟹
pop A x y = *pop A y x*
by (*simp add*:*pop-commute*)

lemma (**in** *aGroup*) *b-ag-group*:*Group* (*b-ag A*)
apply (*unfold Group-def*)
apply (*simp add*:*b-ag-def*)
apply (*simp add*:*pop-closed mop-closed ex-zero*)
apply (*rule conjI*)
apply (*rule allI*, *rule impI*)+
apply (*simp add*:*aassoc*)

apply (*rule conjI*)
apply (*rule allI*, *rule impI*)
apply (*simp add*:*l-m*)

apply (*rule allI*, *rule impI*)
apply (*simp add*:*l-zero*)
done

lemma (**in** *aGroup*) *agop-gop*:*top* (*b-ag A*) = *pop A*
apply (*simp add*:*b-ag-def*)
done

lemma (**in** *aGroup*) *agiop-giop*:*iop* (*b-ag A*) = *mop A*
apply (*simp add*:*b-ag-def*)
done

lemma (**in** *aGroup*) *agunit-gone*:*one* (*b-ag A*) = **0**
apply (*simp add*:*b-ag-def*)
done

lemma (**in** *aGroup*) *ag-pOp-add-r*:⟦*a* ∈ *carrier A*; *b* ∈ *carrier A*; *c* ∈ *carrier A*;
a = *b*⟧ ⟹ *a* ± *c* = *b* ± *c*
apply *simp*
done

lemma (**in** *aGroup*) *ag-add-commute*:⟦*a* ∈ *carrier A*; *b* ∈ *carrier A*⟧ ⟹
a ± *b* = *b* ± *a*
by (*simp add*:*pop-commute*)

lemma (**in** *aGroup*) *ag-pOp-add-l*:⟦*a* ∈ *carrier A*; *b* ∈ *carrier A*; *c* ∈ *carrier A*;
a = *b*⟧ ⟹ *c* ± *a* = *c* ± *b*
apply *simp*
done

lemma (**in** *aGroup*) *asubg-pOp-closed*:⟦*asubGroup A H*; *x* ∈ *H*; *y* ∈ *H*⟧
⟹ *pop A x y* ∈ *H*
apply (*simp add*:*asubGroup-def*)
apply (*cut-tac b-ag-group*)
apply (*frule Group.sg-mult-closed* [*of b-ag A H x y*], *assumption*+)
apply (*simp only*:*agop-gop*)
done

lemma (**in** *aGroup*) *asubg-mOp-closed*:⟦*asubGroup A H*; *x* ∈ *H*⟧ ⟹ −$_a$ *x* ∈ *H*
apply (*simp add*:*asubGroup-def*)
apply (*cut-tac b-ag-group*)
apply (*frule Group.sg-i-closed*[*of b-ag A H x*], *assumption*+)
apply (*simp add*:*agiop-giop*)
done

lemma (**in** *aGroup*) *asubg-subset1*:⟦*asubGroup A H*; $x \in H$⟧ ⟹ $x \in$ *carrier A*
apply (*simp add*:*asubGroup-def*)
apply (*cut-tac b-ag-group*)
apply (*frule Group.sg-subset-elem*[*of b-ag A H x*], *assumption*+)
apply (*simp add*:*ag-carrier-carrier*)
done

lemma (**in** *aGroup*) *asubg-inc-zero*:*asubGroup A H* ⟹ $\mathbf{0} \in H$
apply (*simp add*:*asubGroup-def*)
apply (*cut-tac b-ag-group*)
apply (*frule Group.sg-unit-closed*[*of b-ag A H*], *assumption*)
apply (*simp add*:*b-ag-def*)
done

lemma (**in** *aGroup*) *ag-inc-zero*:$\mathbf{0} \in$ *carrier A*
by (*simp add*:*ex-zero*)

lemma (**in** *aGroup*) *ag-l-zero*:$x \in$ *carrier A* ⟹ $\mathbf{0} \pm x = x$
by (*simp add*:*l-zero*)

lemma (**in** *aGroup*) *ag-r-zero*:$x \in$ *carrier A* ⟹ $x \pm \mathbf{0} = x$
apply (*cut-tac ex-zero*)
apply (*subst pop-commute*, *assumption*+)
apply (*rule ag-l-zero*, *assumption*)
done

lemma (**in** *aGroup*) *ag-l-inv1*:$x \in$ *carrier A* ⟹ $(-_a\ x) \pm x = \mathbf{0}$
by (*simp add*:*l-m*)

lemma (**in** *aGroup*) *ag-r-inv1*:$x \in$ *carrier A* ⟹ $x \pm (-_a\ x) = \mathbf{0}$
by (*frule ag-mOp-closed*[*of x*],
subst ag-pOp-commute, *assumption*+,
simp add:*ag-l-inv1*)

lemma (**in** *aGroup*) *ag-pOp-assoc*:⟦$x \in$ *carrier A*; $y \in$ *carrier A*; $z \in$ *carrier A*⟧
⟹ $(x \pm y) \pm z = x \pm (y \pm z)$
by (*simp add*:*aassoc*)

lemma (**in** *aGroup*) *ag-inv-unique*:⟦$x \in$ *carrier A*; $y \in$ *carrier A*; $x \pm y = \mathbf{0}$⟧ ⟹
$y = -_a\ x$
apply (*frule ag-mOp-closed*[*of x*],
frule aassoc[*of* $-_a\ x\ x\ y$], *assumption*+,
simp add:*l-m l-zero ag-r-zero*)
done

lemma (**in** *aGroup*) *ag-inv-inj*:⟦$x \in$ *carrier A*; $y \in$ *carrier A*; $x \neq y$⟧ ⟹
$(-_a\ x) \neq (-_a\ y)$
apply (*rule contrapos-pp*, *simp*+)
apply (*frule ag-mOp-closed*[*of y*],

frule aassoc[*of y* $-_a$ *y x*], *assumption*+)
apply (*simp only*:*ag-r-inv1*,
frule sym, *thin-tac* $-_a$ *x* = $-_a$ *y*, *simp add*:*l-m*)
apply (*simp add*:*l-zero ag-r-zero*)
done

lemma (**in** *aGroup*) *pOp-assocTr41*:⟦*a* ∈ *carrier A*; *b* ∈ *carrier A*; *c* ∈ *carrier A*;
d ∈ *carrier A*⟧ ⟹ *a* ± *b* ± *c* ± *d* = *a* ± *b* ± (*c* ± *d*)
by (*frule ag-pOp-closed*[*of a b*], *assumption*+,
rule aassoc[*of a* ± *b c d*], *assumption*+)

lemma (**in** *aGroup*) *pOp-assocTr42*:⟦*a* ∈ *carrier A*; *b* ∈ *carrier A*;
c ∈ *carrier A*; *d* ∈ *carrier A*⟧ ⟹ *a* ± *b* ± *c* ± *d* = *a* ± (*b* ± *c*) ± *d*
by (*simp add*:*aassoc*[*THEN sym*, *of a b c*])

lemma (**in** *aGroup*) *pOp-assocTr43*:⟦*a* ∈ *carrier A*; *b* ∈ *carrier A*;
c ∈ *carrier A*; *d* ∈ *carrier A*⟧ ⟹ *a* ± *b* ± (*c* ± *d*) = *a* ± (*b* ± *c*) ± *d*
by (*subst pOp-assocTr41*[*THEN sym*], *assumption*+,
rule pOp-assocTr42, *assumption*+)

lemma (**in** *aGroup*) *pOp-assoc-cancel*:⟦*a* ∈ *carrier A*; *b* ∈ *carrier A*;
c ∈ *carrier A*⟧ ⟹ *a* ± $-_a$ *b* ± (*b* ± $-_a$ *c*) = *a* ± $-_a$ *c*
apply (*subst pOp-assocTr43*, *assumption*)
apply (*simp add*:*ag-l-inv1 ag-mOp-closed*)+
apply (*simp add*:*ag-r-zero*)
done

lemma (**in** *aGroup*) *ag-p-inv*:⟦*x* ∈ *carrier A*; *y* ∈ *carrier A*⟧ ⟹
($-_a$ (*x* ± *y*)) = ($-_a$ *x*) ± ($-_a$ *y*)
apply (*frule ag-mOp-closed*[*of x*], *frule ag-mOp-closed*[*of y*],
frule ag-pOp-closed[*of x y*], *assumption*+)
apply (*frule aassoc*[*of x* ± *y* $-_a$ *x* $-_a$ *y*], *assumption*+,
simp add:*pOp-assocTr43*, *simp add*:*pop-commute*[*of y* $-_a$ *x*],
simp add:*aassoc*[*THEN sym*, *of x* $-_a$ *x y*],
simp add:*ag-r-inv1 l-zero*)
apply (*frule ag-pOp-closed*[*of* $-_a$ *x* $-_a$ *y*], *assumption*+,
simp add:*pOp-assocTr41*,
rule ag-inv-unique[*THEN sym*, *of x* ± *y* $-_a$ *x* ± $-_a$ *y*], *assumption*+)
done

lemma (**in** *aGroup*) *gEQAddcross*: ⟦*l1* ∈ *carrier A*; *l2* ∈ *carrier A*;
r1 ∈ *carrier A*; *r1* ∈ *carrier A*; *l1* = *r2*; *l2* = *r1*⟧ ⟹
l1 ± *l2* = *r1* ± *r2*
apply (*simp add*:*ag-pOp-commute*)
done

lemma (**in** *aGroup*) *ag-eq-sol1*:⟦*a* ∈ *carrier A*; *x*∈ *carrier A*; *b*∈ *carrier A*;
a ± *x* = *b*⟧ ⟹ *x* = ($-_a$ *a*) ± *b*
apply (*frule ag-mOp-closed*[*of a*])

apply (*frule aassoc*[*of* $-_a$ *a a x*], *assumption*+)
apply (*simp add*:*l-m l-zero*)
done

lemma (**in** *aGroup*) *ag-eq-sol2*:⟦$a \in$ *carrier A*; $x \in$ *carrier A*; $b \in$ *carrier A*;
$x \pm a = b$⟧ $\Longrightarrow x = b \pm (-_a\ a)$
apply (*frule ag-mOp-closed*[*of a*],
frule aassoc[*of x a* $-_a$ *a*], *assumption*+,
simp add:*ag-r-inv1 ag-r-zero*)
done

lemma (**in** *aGroup*) *ag-add4-rel*:⟦$a \in$ *carrier A*; $b \in$ *carrier A*; $c \in$ *carrier A*;
$d \in$ *carrier A* ⟧ $\Longrightarrow a \pm b \pm (c \pm d) = a \pm c \pm (b \pm d)$
apply (*simp add*:*pOp-assocTr43*[*of a b c d*],
simp add:*ag-pOp-commute*[*of b c*],
simp add:*pOp-assocTr43*[*THEN sym*, *of a c b d*])
done

lemma (**in** *aGroup*) *ag-inv-inv*:$x \in$ *carrier A* $\Longrightarrow -_a\ (-_a\ x) = x$
by (*frule ag-l-inv1*[*of x*], *frule ag-mOp-closed*[*of x*],
rule ag-inv-unique[*THEN sym*, *of* $-_a$ *x x*], *assumption*+)

lemma (**in** *aGroup*) *ag-inv-zero*:$-_a\ \mathbf{0} = \mathbf{0}$
apply (*cut-tac ex-zero*)
apply (*frule l-zero*[*of* **0**])
apply (*rule ag-inv-unique*[*THEN sym*], *assumption*+)
done

lemma (**in** *aGroup*) *ag-diff-minus*:⟦$a \in$ *carrier A*; $b \in$ *carrier A*; $c \in$ *carrier A*;
$a \pm (-_a\ b) = c$⟧ $\Longrightarrow b \pm (-_a\ a) = (-_a\ c)$
apply (*frule sym*, *thin-tac* $a \pm -_a\ b = c$, *simp*, *thin-tac* $c = a \pm -_a\ b$)
apply (*frule ag-mOp-closed*[*of b*], *frule ag-mOp-closed*[*of a*],
subst ag-p-inv, *assumption*+, *subst ag-inv-inv*, *assumption*)
apply (*simp add*:*ag-pOp-commute*)
done

lemma (**in** *aGroup*) *pOp-cancel-l*:⟦$a \in$ *carrier A*; $b \in$ *carrier A*; $c \in$ *carrier A*;
$c \pm a = c \pm b$ ⟧ $\Longrightarrow a = b$
apply (*frule ag-mOp-closed*[*of c*],
frule aassoc[*of* $-_a$ *c c a*], *assumption*+,
simp only:*l-m l-zero*)
apply (*simp only*:*aassoc*[*THEN sym*, *of* $-_a$ *c c b*],
simp only:*l-m l-zero*)
done

lemma (**in** *aGroup*) *pOp-cancel-r*:⟦$a \in$ *carrier A*; $b \in$ *carrier A*; $c \in$ *carrier A*;
$a \pm c = b \pm c$ ⟧ $\Longrightarrow a = b$
by (*simp add*:*ag-pOp-commute pOp-cancel-l*)

lemma (**in** *aGroup*) *ag-eq-diffzero*:⟦*a* ∈ *carrier A*; *b* ∈ *carrier A*⟧ ⟹
(*a* = *b*) = (*a* ± (−$_a$ *b*) = **0**)
apply (*rule iffI*)
apply (*simp add*:*ag-r-inv1*)
apply (*frule ag-mOp-closed*[*of b*])
apply (*simp add*:*ag-pOp-commute*[*of a* −$_a$ *b*])
apply (*subst ag-inv-unique*[*of* −$_a$ *b a*], *assumption*+,
simp add:*ag-inv-inv*)
done

lemma (**in** *aGroup*) *ag-eq-diffzero1*:⟦*a* ∈ *carrier A*; *b* ∈ *carrier A*⟧ ⟹
(*a* = *b*) = ((−$_a$ *a*) ± *b* = **0**)
apply (*frule ag-mOp-closed*[*of a*],
simp add:*ag-pOp-commute*)
apply (*subst ag-eq-diffzero*[*THEN sym*], *assumption*+)
apply (*rule iffI*, *rule sym*, *assumption*)
apply (*rule sym*, *assumption*)
done

lemma (**in** *aGroup*) *ag-neq-diffnonzero*:⟦*a* ∈ *carrier A*; *b* ∈ *carrier A*⟧ ⟹
(*a* ≠ *b*) = (*a* ± (−$_a$ *b*) ≠ **0**)
apply (*rule iffI*)
apply (*rule contrapos-pp*, *simp*+)
apply (*simp add*:*ag-eq-diffzero*[*THEN sym*])
apply (*rule contrapos-pp*, *simp*+)
apply (*simp add*:*ag-r-inv1*)
done

lemma (**in** *aGroup*) *ag-plus-zero*:⟦*x* ∈ *carrier A*; *y* ∈ *carrier A*⟧ ⟹
(*x* = −$_a$ *y*) = (*x* ± *y* = **0**)
apply (*rule iffI*)
apply (*simp add*:*ag-l-inv1*)
apply (*simp add*:*ag-pOp-commute*[*of x y*])
apply (*rule ag-inv-unique*[*of y x*], *assumption*+)
done

lemma (**in** *aGroup*) *asubg-nsubg*:*A* +> *H* ⟹ (*b-ag A*) ▷ *H*
apply (*cut-tac b-ag-group*)
apply (*simp add*:*asubGroup-def*)
apply (*rule Group.cond-nsg*[*of b-ag A H*], *assumption*+)
apply (*rule ballI*)+
apply(*simp add*:*agop-gop agiop-giop*)
apply (*frule Group.sg-subset*[*of b-ag A H*], *assumption*)
apply (*simp add*:*ag-carrier-carrier*)
apply (*frule-tac c* = *h* **in** *subsetD*[*of H carrier A*], *assumption*+)
apply (*subst ag-pOp-commute*, *assumption*+)
apply (*frule-tac x* = *a* **in** *ag-mOp-closed*)
apply (*subst aassoc*, *assumption*+, *simp add*:*ag-r-inv1 ag-r-zero*)
done

lemma (**in** *aGroup*) *subg-asubg*:*b-ag* $G \gg H \Longrightarrow G +> H$
apply (*simp add*:*asubGroup-def*)
done

lemma (**in** *aGroup*) *asubg-test*:$[\![H \subseteq carrier\ A;\ H \neq \{\};$
$\forall a{\in}H.\ \forall b{\in}H.\ (a \pm (-_a\ b) \in H)]\!] \Longrightarrow A +> H$
apply (*simp add*:*asubGroup-def*) **apply** (*cut-tac b-ag-group*)
apply (*rule Group.sg-condition* [*of b-ag A H*], *assumption*+)
apply (*simp add*:*ag-carrier-carrier*) **apply** *assumption*
apply (*rule allI*)+ **apply** (*rule impI*)
apply (*simp add*:*agop-gop agiop-giop*)
done

lemma (**in** *aGroup*) *asubg-zero*:$A +> \{\mathbf{0}\}$
apply (*rule asubg-test*[*of* $\{\mathbf{0}\}$])
apply (*simp add*:*ag-inc-zero*)
apply *simp*
apply (*simp*, *cut-tac ag-inc-zero*, *simp add*:*ag-r-inv1*)
done

lemma (**in** *aGroup*) *asubg-whole*:$A +>$ *carrier A*
apply (*rule asubg-test*[*of carrier A*])
apply (*simp*,
cut-tac ag-inc-zero, *simp add*:*nonempty*)
apply ((*rule ballI*)+,
rule ag-pOp-closed, *assumption*,
rule-tac x = *b* **in** *ag-mOp-closed*, *assumption*)
done

lemma (**in** *aGroup*) *Ag-ind-carrier*:*bij-to f* (*carrier A*) (D::$'d$ *set*) $\Longrightarrow$
carrier (*Ag-ind A f*) = *f* ‘ (*carrier A*)
by (*simp add*:*Ag-ind-def*)

lemma (**in** *aGroup*) *Ag-ind-aGroup*:$[\![f \in carrier\ A \to D;$
bij-to f (*carrier A*) (D::$'d$ *set*)$]\!] \Longrightarrow$ *aGroup* (*Ag-ind A f*)
apply (*simp add*:*bij-to-def*, *frule conjunct1*, *frule conjunct2*, *fold bij-to-def*)
apply (*simp add*:*aGroup-def*)
apply (*rule conjI*)
apply (*rule Pi-I*)+
apply (*simp add*:*Ag-ind-carrier surj-to-def*)
apply (*frule-tac b* = *x* **in** *invfun-mem1*[*of f carrier A D*], *assumption*+,
frule-tac b = *xa* **in** *invfun-mem1*[*of f carrier A D*], *assumption*+)
apply (*simp add*:*Ag-ind-def*)
apply (*rule funcset-mem*[*of f carrier A D*], *assumption*)
apply (*simp add*:*ag-pOp-closed*)

apply (*rule conjI*)
apply (*rule allI*, *rule impI*)+

apply (*simp add*: *Ag-ind-carrier surj-to-def*)
apply (*frule-tac b = a* **in** *invfun-mem1*[*of f carrier A D*], *assumption*+,
frule-tac b = b **in** *invfun-mem1*[*of f carrier A D*], *assumption*+,
frule-tac b = c **in** *invfun-mem1*[*of f carrier A D*], *assumption*+)
apply (*simp add*:*Ag-ind-def*)
apply (*frule-tac x = invfun* (*carrier A*) *D f a* **and**
y = invfun (*carrier A*) *D f b* **in** *ag-pOp-closed*, *assumption*+,
frule-tac x = invfun (*carrier A*) *D f b* **and**
y = invfun (*carrier A*) *D f c* **in** *ag-pOp-closed*, *assumption*+)
apply (*simp add*:*Pi-def*)
apply (*unfold bij-to-def*, *frule conjunct1*, *fold bij-to-def*)
apply (*simp add*:*invfun-l*)
apply (*subst injective-iff*[*of f carrier A*, *THEN sym*], *assumption*)
apply (*simp add*:*ag-pOp-closed*)+
apply (*simp add*:*ag-pOp-assoc*)

apply (*rule conjI*)
apply (*rule allI*, *rule impI*)+
apply (*simp add*:*Ag-ind-def*)
apply (*subst injective-iff*[*of f carrier A*, *THEN sym*], *assumption*)
apply (*frule-tac b = a* **in** *invfun-mem1*[*of f carrier A D*], *assumption*+,
frule-tac b = b **in** *invfun-mem1*[*of f carrier A D*], *assumption*+)
apply (*simp add*:*surj-to-def*) **apply** (*simp add*:*surj-to-def*)

apply (*simp add*:*surj-to-def*)
apply (*frule-tac b = b* **in** *invfun-mem1*[*of f carrier A D*], *assumption*+)
apply (*simp add*:*ag-pOp-closed*)

apply (*simp add*:*surj-to-def*)
apply (*frule-tac b = a* **in** *invfun-mem1*[*of f carrier A D*], *assumption*+,
frule-tac b = b **in** *invfun-mem1*[*of f carrier A D*], *assumption*+)
apply (*simp add*:*ag-pOp-closed*)

apply (*simp add*:*surj-to-def*)
apply (*frule-tac b = a* **in** *invfun-mem1*[*of f carrier A D*], *assumption*+,
frule-tac b = b **in** *invfun-mem1*[*of f carrier A D*], *assumption*+)
apply (*simp add*:*ag-pOp-commute*)

apply (*rule conjI*)
apply (*rule Pi-I*)
apply (*simp add*:*Ag-ind-def surj-to-def*)
apply (*rule funcset-mem*[*of f carrier A D*], *assumption*)
apply (*frule-tac b = x* **in** *invfun-mem1*[*of f carrier A D*], *assumption*+)
apply (*simp add*:*ag-mOp-closed*)

apply (*rule conjI*)
apply (*rule allI*, *rule impI*)
apply (*simp add*:*Ag-ind-def surj-to-def*)
apply (*frule-tac b = a* **in** *invfun-mem1*[*of f carrier A D*], *assumption*+)

apply (*frule-tac x = invfun (carrier A) D f a* **in** *ag-mOp-closed*)
apply (*simp add*:*Pi-def*)
apply (*subst injective-iff*[*of f carrier A*, *THEN sym*], *assumption*)
apply (*unfold bij-to-def*, *frule conjunct1*, *fold bij-to-def*)
apply (*simp add*:*invfun-l*)
apply (*simp add*:*ag-pOp-closed*)
apply (*simp add*:*ag-inc-zero*)
apply (*unfold bij-to-def*, *frule conjunct1*, *fold bij-to-def*)
apply (*simp add*:*invfun-l l-m*)

apply (*rule conjI*)
apply (*simp add*:*Ag-ind-def surj-to-def*)
apply (*rule funcset-mem*[*of f carrier A D*], *assumption*)
apply (*simp add*:*ag-inc-zero*)

apply (*rule allI*, *rule impI*)
apply (*simp add*:*Ag-ind-def surj-to-def*)
apply (*cut-tac ag-inc-zero*, *simp add*:*funcset-mem del*:*Pi-I*)
apply (*unfold bij-to-def*, *frule conjunct1*, *fold bij-to-def*)
apply (*simp add*:*invfun-l*)
apply (*frule-tac b = a* **in** *invfun-mem1*[*of f carrier A D*], *assumption*+)
apply (*simp add*:*l-zero*)
apply (*simp add*:*invfun-r*)
done

3.18.1 Homomorphism of abelian groups

definition
aHom :: [('*a*, '*m*) *aGroup-scheme*, ('*b*, '*m1*) *aGroup-scheme*] ⇒ ('*a* ⇒ '*b*) *set*
where
aHom A B = {*f*. *f* ∈ *carrier A* → *carrier B* ∧ *f* ∈ *extensional* (*carrier A*) ∧
(∀ *a*∈*carrier A*. ∀ *b*∈*carrier A*. *f* (*a* ±$_A$ *b*) = (*f a*) ±$_B$ (*f b*))}

definition
compos :: [('*a*, '*m*) *aGroup-scheme*, '*b* ⇒ '*c*, '*a* ⇒ '*b*] ⇒ '*a* ⇒ '*c* **where**
compos A g f = *compose* (*carrier A*) *g f*

definition
ker :: [('*a*, '*m*) *aGroup-scheme*, ('*b*, '*m1*) *aGroup-scheme*] ⇒ ('*a* ⇒ '*b*)
⇒ '*a set* ((*3ker*$_{-,-}$ -) [*82*,*82*,*83*]*82*) **where**
ker$_{F,G}$ *f* = {*a*. *a* ∈ *carrier F* ∧ *f a* = (**0**$_G$)}

definition
injec :: [('*a*, '*m*) *aGroup-scheme*, ('*b*, '*m1*) *aGroup-scheme*, '*a* ⇒ '*b*]
⇒ *bool* ((*3injec*$_{-,-}$ -) [*82*,*82*,*83*]*82*) **where**
injec$_{F,G}$ *f* ⟷ *f* ∈ *aHom F G* ∧ *ker*$_{F,G}$ *f* = {**0**$_F$}

definition

surjec :: [('*a*, '*m*) *aGroup-scheme*, ('*b*, '*m1*) *aGroup-scheme*, '*a* ⇒ '*b*]
⇒ *bool* ((*3surjec*$_{-,-}$ -) [*82*,*82*,*83*]*82*) **where**
surjec$_{F,G}$ *f* ⟷ *f* ∈ *aHom F G* ∧ *surj-to f* (*carrier F*) (*carrier G*)

definition
bijec :: [('*a*, '*m*) *aGroup-scheme*, ('*b*, '*m1*) *aGroup-scheme*, '*a* ⇒ '*b*]
⇒ *bool* ((*3bijec*$_{-,-}$ -) [*82*,*82*,*83*]*82*) **where**
bijec$_{F,G}$ *f* ⟷ *injec*$_{F,G}$ *f* ∧ *surjec*$_{F,G}$ *f*

definition
ainvf :: [('*a*, '*m*) *aGroup-scheme*, ('*b*, '*m1*) *aGroup-scheme*, '*a* ⇒ '*b*]
⇒ ('*b* ⇒ '*a*) ((*3ainvf*$_{-,-}$ -) [*82*,*82*,*83*]*82*) **where**
ainvf$_{F,G}$ *f* = *invfun* (*carrier F*) (*carrier G*) *f*

lemma *aHom-mem*:⟦*aGroup F*; *aGroup G*; *f* ∈ *aHom F G*; *a* ∈ *carrier F*⟧ ⟹
f a ∈ *carrier G*
apply (*simp add*:*aHom-def*) **apply** (*erule conjE*)+
apply (*simp add*:*Pi-def*)
done

lemma *aHom-func*:*f* ∈ *aHom F G* ⟹ *f* ∈ *carrier F* → *carrier G*
by (*simp add*:*aHom-def*)

lemma *aHom-add*:⟦*aGroup F*; *aGroup G*; *f* ∈ *aHom F G*; *a* ∈ *carrier F*;
b ∈ *carrier F*⟧ ⟹ *f* (*a* ±$_F$ *b*) = (*f a*) ±$_G$ (*f b*)
apply (*simp add*:*aHom-def*)
done

lemma *aHom-0-0*:⟦*aGroup F*; *aGroup G*; *f* ∈ *aHom F G*⟧ ⟹ *f* (**0**$_F$) = **0**$_G$
apply (*frule aGroup.ag-inc-zero* [*of F*])
apply (*subst aGroup.ag-l-zero* [*THEN sym*, *of F* **0**$_F$], *assumption*+)
apply (*simp add*:*aHom-add*)
apply (*frule aGroup.ag-l-zero* [*THEN sym*, *of F* **0**$_F$], *assumption*+)
apply (*subgoal-tac f* (**0**$_F$) = *f* (**0**$_F$ ±$_F$ **0**$_F$)) **prefer** *2* **apply** *simp*
apply (*thin-tac* **0**$_F$ = **0**$_F$ ±$_F$ **0**$_F$)
apply (*simp add*:*aHom-add*) **apply** (*frule sym*)
apply (*thin-tac f* **0**$_F$ = *f* **0**$_F$ ±$_G$ *f* **0**$_F$)
apply (*frule aHom-mem*[*of F G f* **0**$_F$], *assumption*+)
apply (*frule aGroup.ag-mOp-closed*[*of G f* **0**$_F$], *assumption*+)
apply (*frule aGroup.aassoc*[*of G* −$_{a\,G}$ (*f* **0**$_F$) *f* **0**$_F$ *f* **0**$_F$], *assumption*+)
apply (*simp add*:*aGroup.l-m aGroup.l-zero*)
done

lemma *ker-inc-zero*:⟦*aGroup F*; *aGroup G*; *f* ∈ *aHom F G*⟧ ⟹ **0**$_F$ ∈ *ker*$_{F,G}$ *f*
by (*frule aHom-0-0*[*of F G f*], *assumption*+,
simp add:*ker-def*, *simp add*:*aGroup.ag-inc-zero* [*of F*])

lemma *aHom-inv-inv*:⟦*aGroup F*; *aGroup G*; *f* ∈ *aHom F G*; *a* ∈ *carrier F*⟧ ⟹
f (−$_{a\,F}$ *a*) = −$_{a\,G}$ (*f a*)

apply (*frule aGroup.ag-l-inv1* [*of F a*], *assumption*+,
frule sym, *thin-tac* $-_{a\,F}\ a \pm_F a = \mathbf{0}_F$,
frule aHom-0-0[*of F G f*], *assumption*+,
frule aGroup.ag-mOp-closed[*of F a*], *assumption*+)
apply (*simp add*:*aHom-add*, *thin-tac* $\mathbf{0}_F = -_{a\,F}\ a \pm_F a$)
apply (*frule aHom-mem*[*of F G f* $-_{a\,F}$ *a*], *assumption*+,
frule aHom-mem[*of F G f a*], *assumption*+,
simp only:*aGroup.ag-pOp-commute*[*of G f* $(-_{a\,F}\ a)$ *f a*])
apply (*rule aGroup.ag-inv-unique*[*of G*], *assumption*+)
done

lemma *aHom-compos*:⟦*aGroup L*; *aGroup M*; *aGroup N*; *f* ∈ *aHom L M*; *g* ∈ *aHom M N* ⟧
⟹ *compos L g f* ∈ *aHom L N*
apply (*simp add*:*aHom-def* [*of L N*])
apply (*rule conjI*)
apply (*rule Pi-I*)
apply (*simp add*:*compos-def compose-def*)
apply (*rule aHom-mem* [*of M N g*], *assumption*+)
apply (*simp add*:*aHom-mem* [*of L M f*])
apply (*rule conjI*)
apply (*simp add*:*compos-def compose-def extensional-def*)
apply (*rule ballI*)+
apply (*simp add*:*compos-def compose-def*)
apply (*simp add*:*aGroup.ag-pOp-closed*)
apply (*simp add*:*aHom-add*)
apply (*rule aHom-add*, *assumption*+)
apply (*simp add*:*aHom-mem*)+
done

lemma *aHom-compos-assoc*:⟦*aGroup K*; *aGroup L*; *aGroup M*; *aGroup N*; *f* ∈ *aHom K L*;
g ∈ *aHom L M*; *h* ∈ *aHom M N* ⟧ ⟹
compos K h (*compos K g f*) = *compos K* (*compos L h g*) *f*
apply (*simp add*:*compos-def compose-def*)
apply (*rule funcset-eq*[*of* - *carrier K*])
apply (*simp add*:*restrict-def extensional-def*)
apply (*simp add*:*restrict-def extensional-def*)
apply (*rule ballI*, *simp*)
apply (*simp add*:*aHom-mem*)
done

lemma *injec-inj-on*:⟦*aGroup F*; *aGroup G*; *injec*$_{F,G}$ *f*⟧ ⟹ *inj-on f* (*carrier F*)
apply (*simp add*:*inj-on-def*)
apply (*rule ballI*)+ **apply** (*rule impI*)
apply (*simp add*:*injec-def*, *erule conjE*)
apply (*frule-tac a* = *x* **in** *aHom-mem*[*of F G f*], *assumption*+,
frule-tac a = *x* **in** *aHom-mem*[*of F G f*], *assumption*+)
apply (*frule-tac x* = *f x* **in** *aGroup.ag-r-inv1*[*of G*], *assumption*+)

apply (*simp only*:*aHom-inv-inv*[*THEN sym*, *of F G f*])
apply (*frule sym*, *thin-tac f x* = *f y*, *simp*)
apply (*frule-tac x* = *y* **in** *aGroup.ag-mOp-closed*[*of F*], *assumption*+)
apply (*simp add*:*aHom-add*[*THEN sym*], *simp add*:*ker-def*)
apply (*subgoal-tac x* $\pm_F$ $-_{a\,F}$ *y* ∈ {*a* ∈ *carrier F*. *f a* = $\mathbf{0}_G$},
simp)
apply (*subst aGroup.ag-eq-diffzero*[*of F*], *assumption*+)
apply (*frule-tac x* = *x* **and** *y* = $-_{a\,F}$ *y* **in** *aGroup.ag-pOp-closed*[*of F*],
assumption+)
apply *simp* **apply** *blast*
done

lemma *surjec-surj-to*:*surjec*$_{R,S}$ *f* ⟹ *surj-to f* (*carrier R*) (*carrier S*)
by (*simp add*:*surjec-def*)

lemma *compos-bijec*:⟦*aGroup E*; *aGroup F*; *aGroup G*; *bijec*$_{E,F}$ *f*; *bijec*$_{F,G}$ *g*⟧
⟹
bijec$_{E,G}$ (*compos E g f*)
apply (*simp add*:*bijec-def*, (*erule conjE*)+)
apply (*rule conjI*)
apply (*simp add*:*injec-def*, (*erule conjE*)+)
apply (*simp add*:*aHom-compos*[*of E F G f g*])
apply (*rule equalityI*, *rule subsetI*, *simp add*:*ker-def*, *erule conjE*)
apply (*simp add*:*compos-def compose-def*)
apply (*frule-tac a* = *x* **in** *aHom-mem*[*of E F f*], *assumption*+)
apply (*subgoal-tac* (*f x*) ∈ {*a* ∈ *carrier F*. *g a* = $\mathbf{0}_G$}, *simp*)
apply (*subgoal-tac x* ∈ {*a* ∈ *carrier E*. *f a* = $\mathbf{0}_F$}, *simp*)
apply *blast* **apply** *blast*
apply (*rule subsetI*, *simp*)
apply (*simp add*:*ker-def compos-def compose-def*)
apply (*simp add*:*aGroup.ag-inc-zero*) **apply** (*simp add*:*aHom-0-0*)

apply (*simp add*:*surjec-def*, (*erule conjE*)+)
apply (*simp add*:*aHom-compos*)
apply (*simp add*:*aHom-def*, (*erule conjE*)+) **apply** (*simp add*:*compos-def*)
apply (*rule compose-surj*[*of f carrier E carrier F g carrier G*],
assumption+)
done

lemma *ainvf-aHom*:⟦*aGroup F*; *aGroup G*; *bijec*$_{F,G}$ *f*⟧ ⟹
ainvf$_{F,G}$ *f* ∈ *aHom G F*
apply (*subst aHom-def*, *simp*)
apply (*simp add*:*ainvf-def*)
apply (*simp add*:*bijec-def*, *erule conjE*)
apply (*frule injec-inj-on*[*of F G f*], *assumption*+)
apply (*simp add*:*surjec-def*, (*erule conjE*)+)
apply (*simp add*:*aHom-def*, (*erule conjE*)+)
apply (*frule inv-func*[*of f carrier F carrier G*], *assumption*+, *simp*)
apply (*rule conjI*)

apply (*simp add*:*invfun-def*)
apply (*rule ballI*)+
apply (*frule-tac* $x = a$ **in** *funcset-mem*[*of Ifn F G f carrier G carrier F*],
assumption+,
frule-tac $x = b$ **in** *funcset-mem*[*of Ifn F G f carrier G carrier F*],
assumption+,
frule-tac $x = a$ **and** $y = b$ **in** *aGroup.ag-pOp-closed*[*of G*], *assumption*+,
frule-tac $x = a \pm_G b$ **in** *funcset-mem*[*of Ifn F G f carrier G*
carrier F], *assumption*+)
apply (*frule-tac* $a = (Ifn\ F\ G\ f)\ a$ **and** $b = (Ifn\ F\ G\ f)\ b$ **in**
aHom-add[*of F G f*], *assumption*+, *simp add*:*injec-def*,
assumption+,
thin-tac $\forall a{\in}carrier\ F.\ \forall b{\in}carrier\ F.\ f\ (a \pm_F b) = f\ a \pm_G f\ b$)
apply (*simp add*:*invfun-r*[*of f carrier F carrier G*])
apply (*frule-tac* $x = a$ **and** $y = b$ **in** *aGroup.ag-pOp-closed*[*of G*], *assumption*+)
apply (*frule-tac* $b = a \pm_G b$ **in** *invfun-r*[*of f carrier F carrier G*],
assumption+)
apply (*simp add*:*inj-on-def*)
apply (*frule-tac* $x = (Ifn\ F\ G\ f)\ a$ **and** $y = (Ifn\ F\ G\ f)\ b$ **in**
aGroup.ag-pOp-closed, *assumption*+)
apply (*frule-tac* $x = (Ifn\ F\ G\ f)\ (a \pm_G b)$ **in** *bspec*, *assumption*,
thin-tac $\forall x{\in}carrier\ F.\ \forall y{\in}carrier\ F.\ f\ x = f\ y \longrightarrow x = y$)
apply (*frule-tac* $x = (Ifn\ F\ G\ f)\ a \pm_F (Ifn\ F\ G\ f)\ b$ **in** *bspec*,
assumption,
thin-tac $\forall y{\in}carrier\ F.$
$f\ ((Ifn\ F\ G\ f)\ (a \pm_G b)) = f\ y \longrightarrow (Ifn\ F\ G\ f)\ (a \pm_G b) = y$)
apply *simp*
done

lemma *ainvf-bijec*:$[\![aGroup\ F;\ aGroup\ G;\ bijec_{F,G}\ f]\!] \Longrightarrow bijec_{G,F}\ (ainvf_{F,G}\ f)$
apply (*subst bijec-def*)
apply (*simp add*:*injec-def surjec-def*)
apply (*simp add*:*ainvf-aHom*)
apply (*rule conjI*)
apply (*rule equalityI*)
apply (*rule subsetI*, *simp add*:*ker-def*, *erule conjE*)
apply (*simp add*:*ainvf-def*)
apply (*simp add*:*bijec-def*,(*erule conjE*)+, *simp add*:*surjec-def*,
(*erule conjE*)+, *simp add*:*aHom-def*, (*erule conjE*)+)
apply (*frule injec-inj-on*[*of F G f*], *assumption*+)
apply (*subst invfun-r*[*THEN sym*, *of f carrier F carrier G*], *assumption*+)
apply (*simp add*:*injec-def*, (*erule conjE*)+, *simp add*:*aHom-0-0*)

apply (*rule subsetI*, *simp add*:*ker-def*)
apply (*simp add*:*aGroup.ex-zero*)
apply (*frule ainvf-aHom*[*of F G f*], *assumption*+)
apply (*simp add*:*aHom-0-0*)

apply (*frule ainvf-aHom*[*of F G f*], *assumption*+,

simp add:*aHom-def*, (*erule conjE*)+,
rule surj-to-test[*of ainvf*$_{F,G}$ *f carrier G carrier F*],
assumption+)
apply (*rule ballI*,
thin-tac $\forall\, a{\in}$*carrier G*. $\forall\, b{\in}$*carrier G*.
(*ainvf*$_{F,G}$ *f*) (*a* $\pm_G$ *b*) = (*ainvf*$_{F,G}$ *f*) *a* $\pm_F$ (*ainvf*$_{F,G}$ *f*) *b*)
apply (*simp add*:*bijec-def*, *erule conjE*)
apply (*frule injec-inj-on*[*of F G f*], *assumption*+)
apply (*simp add*:*surjec-def aHom-def*, (*erule conjE*)+)
apply (*subst ainvf-def*)
apply (*frule-tac a* = *b* **in** *invfun-l*[*of f carrier F carrier G*],
assumption+,
frule-tac x = *b* **in** *funcset-mem*[*of f carrier F carrier G*],
assumption+, *blast*)
done

lemma *ainvf-l*:⟦*aGroup E*; *aGroup F*; *bijec*$_{E,F}$ *f*; *x* ∈ *carrier E*⟧ ⟹
(*ainvf*$_{E,F}$ *f*) (*f x*) = *x*
apply (*simp add*:*bijec-def*, *erule conjE*)
apply (*frule injec-inj-on*[*of E F f*], *assumption*+)
apply (*simp add*:*surjec-def aHom-def*, (*erule conjE*)+)
apply (*frule invfun-l*[*of f carrier E carrier F x*], *assumption*+)
apply (*simp add*:*ainvf-def*)
done

lemma (**in** *aGroup*) *aI-aHom*:*aI*$_A$ ∈ *aHom A A*
by (*simp add*:*aHom-def ag-idmap-def ag-idmap-def ag-pOp-closed*)

lemma *compos-aI-l*:⟦*aGroup A*; *aGroup B*; *f* ∈ *aHom A B*⟧ ⟹ *compos A aI*$_B$ *f* = *f*
apply (*simp add*:*compos-def*)
apply (*rule funcset-eq*[*of* - *carrier A*])
apply (*simp add*:*compose-def extensional-def*)
apply (*simp add*:*aHom-def*)
apply (*rule ballI*)
apply (*frule-tac a* = *x* **in** *aHom-mem*[*of A B f*], *assumption*+)
apply (*simp add*:*compose-def ag-idmap-def*)
done

lemma *compos-aI-r*:⟦*aGroup A*; *aGroup B*; *f* ∈ *aHom A B*⟧ ⟹ *compos A f aI*$_A$ = *f*
apply (*simp add*:*compos-def*)
apply (*rule funcset-eq*[*of* - *carrier A*])
apply (*simp add*:*compose-def extensional-def*)
apply (*simp add*:*aHom-def*)
apply (*rule ballI*)
apply (*simp add*:*compose-def ag-idmap-def*)
done

lemma *compos-aI-surj*:⟦*aGroup A*; *aGroup B*; $f \in aHom\ A\ B$; $g \in aHom\ B\ A$;
compos A g f = aI_A⟧ ⟹ $surjec_{B,A}\ g$
apply (*simp add*:*surjec-def*)
apply (*rule surj-to-test*[*of g carrier B carrier A*])
apply (*simp add*:*aHom-def*)
apply (*rule ballI*)
apply (*subgoal-tac compos A g f b* = aI_A *b*,
thin-tac compos A g f = aI_A)
apply (*simp add*:*compos-def compose-def ag-idmap-def*)
apply (*frule-tac a* = *b* **in** *aHom-mem*[*of A B f*], *assumption*+, *blast*)
apply *simp*
done

lemma *compos-aI-inj*:⟦*aGroup A*; *aGroup B*; $f \in aHom\ A\ B$; $g \in aHom\ B\ A$;
compos A g f = aI_A⟧ ⟹ $injec_{A,B}\ f$
apply (*simp add*:*injec-def*)
apply (*simp add*:*ker-def*)
apply (*rule equalityI*)
apply (*rule subsetI*, *simp*, *erule conjE*)
apply (*subgoal-tac compos A g f x* = aI_A *x*,
thin-tac compos A g f = aI_A)
apply (*simp add*:*compos-def compose-def*)
apply (*simp add*:*aHom-0-0 ag-idmap-def*) **apply** *simp*

apply (*rule subsetI*, *simp*)
apply (*simp add*:*aGroup.ag-inc-zero aHom-0-0*)
done

lemma (**in** *aGroup*) *Ag-ind-aHom*:⟦$f \in carrier\ A \to D$;
bij-to f (*carrier A*) (*D*::'*d set*)⟧ ⟹ $Agii\ A\ f \in aHom\ A\ (Ag\text{-}ind\ A\ f)$
apply (*simp add*:*aHom-def*)
apply (*unfold bij-to-def*, *frule conjunct1*, *frule conjunct2*, *fold bij-to-def*)
apply (*simp add*:*Ag-ind-carrier surj-to-def*)
apply (*rule conjI*)
apply (*simp add*:*Agii-def Pi-def*)
apply (*simp add*:*Agii-def*)
apply (*simp add*:*Ag-ind-def Pi-def*)
apply (*unfold bij-to-def*, *frule conjunct1*, *fold bij-to-def*)
apply (*simp add*:*invfun-l*)
apply (*simp add*:*ag-pOp-closed*)
done

lemma (**in** *aGroup*) *Agii-mem*:⟦$f \in carrier\ A \to D$; $x \in carrier\ A$;
bij-to f (*carrier A*) (*D*::'*d set*)⟧ ⟹ $Agii\ A\ f\ x \in carrier\ (Ag\text{-}ind\ A\ f)$
apply (*simp add*:*Agii-def Ag-ind-carrier*)
done

lemma *Ag-ind-bijec*:⟦*aGroup A*; $f \in carrier\ A \to D$;
bij-to f (*carrier A*) (*D*::'*d set*)⟧ ⟹ $bijec_{A,\ (Ag\text{-}ind\ A\ f)}\ (Agii\ A\ f)$

apply (*frule aGroup.Ag-ind-aHom*[*of A f D*], *assumption*+)
apply (*frule aGroup.Ag-ind-aGroup*[*of A f D*], *assumption*+)
apply (*simp add*:*bijec-def*)
apply (*rule conjI*)
apply (*simp add*:*injec-def*)
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*simp add*:*ker-def*, *erule conjE*)
apply (*frule aHom-0-0*[*of A Ag-ind A f Agii A f*], *assumption*+)
apply (*rotate-tac* −2, *frule sym*, *thin-tac Agii A f x* = $\mathbf{0}_{Ag\text{-}ind\ A\ f}$, *simp*)
apply (*frule aGroup.ag-inc-zero*[*of A*], *simp add*:*Agii-def*)
apply (*unfold bij-to-def*, *frule conjunct2*, *fold bij-to-def*)
apply (*frule aGroup.ag-inc-zero*[*of A*])
apply (*simp add*:*injective-iff*[*THEN sym*, *of f carrier A* $\mathbf{0}_A$])
apply (*rule subsetI*, *simp*)
apply (*subst ker-def*, *simp*)
apply (*simp add*:*aGroup.ag-inc-zero*, *simp add*:*aHom-0-0*)

apply (*subst surjec-def*)
apply (*unfold bij-to-def*, *frule conjunct1*, *fold bij-to-def*, *simp*)
apply (*simp add*:*aGroup.Ag-ind-carrier surj-to-def Agii-def*)
done

definition
aimg :: [('*b*, '*m1*) *aGroup-scheme*, -, '*b* ⇒ '*a*]
⇒ '*a aGroup* ((*3aimg*$_{-,-}$ -) [*82*,*82*,*83*]*82*) **where**
$aimg_{F,A}\ f = A$ (| *carrier* := *f* ‘ (*carrier F*), *pop* := *pop A*, *mop* := *mop A*,
zero := *zero A*|)

lemma *ker-subg*:⟦*aGroup F*; *aGroup G*; *f* ∈ *aHom F G* ⟧ ⟹ *F* +> $ker_{F,G}\ f$
apply (*rule aGroup.asubg-test*, *assumption*+)
apply (*rule subsetI*)
apply (*simp add*:*ker-def*)
apply (*simp add*:*ker-def*)
apply (*frule aHom-0-0* [*of F G f*], *assumption*+)
apply (*frule aGroup.ex-zero* [*of F*]) **apply** *blast*
apply (*rule ballI*)+
apply (*simp add*:*ker-def*) **apply** (*erule conjE*)+
apply (*frule-tac x* = *b* **in** *aGroup.ag-mOp-closed*[*of F*], *assumption*+)
apply (*rule conjI*)
apply (*rule aGroup.ag-pOp-closed*, *assumption*+)
apply (*simp add*:*aHom-add*)
apply (*simp add*:*aHom-inv-inv*)
apply (*simp add*:*aGroup.ag-inv-zero*[*of G*])
apply (*cut-tac aGroup.ex-zero*[*of G*], *simp add*:*aGroup.l-zero*)
apply *assumption*
done

3.18.2 Quotient abelian group

definition
ar-coset :: [*'a*, - , *'a set*] ⇒ *'a set*
((*3*- ⊎_ -) [*66*,*66*,*67*]*66*) **where**
ar-coset a A H = *H* •$_{(b\text{-}ag\ A)}$ *a*

definition
set-ar-cos :: [- , *'a set*] ⇒ *'a set set* **where**
set-ar-cos A I = {*X*. ∃ *a*∈*carrier A*. *X* = *ar-coset a A I*}

definition
aset-sum :: [- , *'a set*, *'a set*] ⇒ *'a set* **where**
aset-sum A H K = *s-top* (*b-ag A*) *H K*

abbreviation
ASBOP1 (**infix** ∓1 *60*) **where**
H ∓$_A$ *K* == *aset-sum A H K*

lemma (**in** *aGroup*) *ag-a-in-ar-cos*:⟦*A* +> *H*; *a* ∈ *carrier A*⟧ ⟹ *a* ∈ *a* ⊎$_A$ *H*
apply (*simp add*:*ar-coset-def*)
apply (*simp add*:*asubGroup-def*)
apply (*cut-tac b-ag-group*)
apply (*rule Group.a-in-rcs*[*of b-ag A H a*], *assumption*+)
apply (*simp add*:*ag-carrier-carrier*[*THEN sym*])
done

lemma (**in** *aGroup*) *r-cos-subset*:⟦*A* +> *H*; *X* ∈ *set-rcs* (*b-ag A*) *H*⟧ ⟹
X ⊆ *carrier A*
apply (*simp add*:*asubGroup-def set-rcs-def*)
apply (*erule bexE*)
apply (*cut-tac b-ag-group*)
apply (*frule-tac a* = *a* **in** *Group.rcs-subset*[*of b-ag A H*], *assumption*+)
apply (*simp add*:*ag-carrier-carrier*)
done

lemma (**in** *aGroup*) *asubg-costOp-commute*:⟦*A* +> *H*; *x* ∈ *set-rcs* (*b-ag A*) *H*;
y ∈ *set-rcs* (*b-ag A*) *H*⟧ ⟹
c-top (*b-ag A*) *H x y* = *c-top* (*b-ag A*) *H y x*
apply (*simp add*:*set-rcs-def*, (*erule bexE*)+, *simp*)
apply (*cut-tac b-ag-group*)
apply (*subst Group.c-top-welldef*[*THEN sym*], *assumption*+,
simp add:*asubg-nsubg*,
(*simp add*:*ag-carrier-carrier*)+)
apply (*subst Group.c-top-welldef*[*THEN sym*], *assumption*+,
simp add:*asubg-nsubg*,
(*simp add*:*ag-carrier-carrier*)+)
apply (*simp add*:*agop-gop*)
apply (*simp add*:*ag-pOp-commute*)
done

lemma (**in** *aGroup*) *Subg-Qgroup*:A +> $H \Longrightarrow$ *aGroup* (*aqgrp* A H)
apply (*frule asubg-nsubg*[*of* H])
apply (*cut-tac b-ag-group*)
apply (*simp add*:*aGroup-def*)
apply (*simp add*:*aqgrp-def*)
apply (*simp add*:*Group.Qg-top* [*of b-ag* A H])
apply (*simp add*:*Group.Qg-iop* [*of b-ag* A H])
apply (*frule Group.nsg-sg*[*of b-ag* A H], *assumption*+,
simp add:*Group.unit-rcs-in-set-rcs*[*of b-ag* A H])
apply (*simp add*:*Group.Qg-tassoc*)
apply (*simp add*:*asubg-costOp-commute*)
apply (*simp add*:*Group.Qg-i*[*of b-ag* A H])
apply (*simp add*:*Group.Qg-unit*[*of b-ag* A H])
done

lemma (**in** *aGroup*) *plus-subgs*:$[\![A$ +> *H1*; A +> *H2*$]\!] \Longrightarrow A$ +> *H1* $\mp$ *H2*
apply (*simp add*:*aset-sum-def*)
apply (*frule asubg-nsubg*[*of H2*])
apply (*simp add*:*asubGroup-def*[*of* - *H1*])
apply (*cut-tac b-ag-group*)
apply (*frule Group.smult-sg-nsg*[*of b-ag* A *H1 H2*], *assumption*+)
apply (*simp add*:*asubGroup-def*)
done

lemma (**in** *aGroup*) *set-sum*:$[\![H \subseteq$ *carrier* A; $K \subseteq$ *carrier* $A]\!] \Longrightarrow$
$H \mp K = \{x.\ \exists h{\in}H.\ \exists k{\in}K.\ x = h \pm k\}$
apply (*cut-tac b-ag-group*)
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*simp add*:*aset-sum-def*)
apply (*simp add*:*agop-gop*[*THEN sym*] *s-top-def*, (*erule bexE*)+,
frule sym, *thin-tac xa* $\cdot_{b\text{-}ag\ A}$ $y = x$, *simp*, *blast*)
apply (*rule subsetI*, *simp add*:*aset-sum-def*, (*erule bexE*)+)
apply (*frule-tac* $c = h$ **in** *subsetD*[*of H carrier A*], *assumption*+,
frule-tac $c = k$ **in** *subsetD*[*of K carrier A*], *assumption*+)
apply (*simp add*:*agop-gop*[*THEN sym*], *simp add*:*s-top-def*, *blast*)
done

lemma (**in** *aGroup*) *mem-set-sum*:$[\![H \subseteq$ *carrier* A; $K \subseteq$ *carrier* A;
$x \in H \mp K\ ]\!] \Longrightarrow \exists h{\in}H.\ \exists k{\in}K.\ x = h \pm k$
by (*simp add*:*set-sum*)

lemma (**in** *aGroup*) *mem-sum-subgs*:$[\![A$ +> H; A +> K; $h \in H$; $k \in K]\!] \Longrightarrow$
$h \pm k \in H \mp K$
apply (*frule asubg-subset*[*of H*],
frule asubg-subset[*of K*],
simp add:*set-sum*, *blast*)
done

lemma (**in** *aGroup*) *aqgrp-carrier*:A +> H $\Longrightarrow$
set-rcs (*b-ag A*) H = *set-ar-cos A H*
apply (*simp add*:*set-ar-cos-def*)
apply (*simp add*:*ag-carrier-carrier* [*THEN sym*])
apply (*simp add*:*ar-coset-def set-rcs-def*)
done

lemma (**in** *aGroup*) *unit-in-set-ar-cos*:A +> H $\Longrightarrow$ $H \in$ *set-ar-cos A H*
apply (*simp add*:*aqgrp-carrier*[*THEN sym*])
apply (*cut-tac b-ag-group*) **apply** (*simp add*:*asubGroup-def*)
apply (*simp add*:*Group.unit-rcs-in-set-rcs*[*of b-ag A H*])
done

lemma (**in** *aGroup*) *aqgrp-pOp-maps*:⟦A +> H; $a \in$ *carrier A*; $b \in$ *carrier A*⟧ $\Longrightarrow$
pop (*aqgrp A H*) ($a \uplus_A H$) ($b \uplus_A H$) = ($a \pm b$) $\uplus_A H$
apply (*simp add*:*aqgrp-def ar-coset-def*)
apply (*cut-tac b-ag-group*)
apply (*frule asubg-nsubg*)
apply (*simp add*:*ag-carrier-carrier* [*THEN sym*])
apply (*subst Group.c-top-welldef* [*THEN sym*], *assumption*+)
apply (*simp add*:*agop-gop*)
done

lemma (**in** *aGroup*) *aqgrp-mOp-maps*:⟦A +> H; $a \in$ *carrier A*⟧ $\Longrightarrow$
mop (*aqgrp A H*) ($a \uplus_A H$) = ($-_a$ a) $\uplus_A H$
apply (*simp add*:*aqgrp-def ar-coset-def*)
apply (*cut-tac b-ag-group*)
apply (*frule asubg-nsubg*)
apply (*simp add*:*ag-carrier-carrier* [*THEN sym*])
apply (*subst Group.c-iop-welldef*, *assumption*+)
apply (*simp add*:*agiop-giop*)
done

lemma (**in** *aGroup*) *aqgrp-zero*:A +> H $\Longrightarrow$ *zero* (*aqgrp A H*) = H
apply (*simp add*:*aqgrp-def*)
done

lemma (**in** *aGroup*) *arcos-fixed*:⟦A +> H; $a \in$ *carrier A*; $h \in H$ ⟧ $\Longrightarrow$
$a \uplus_A H$ = ($h \pm a$) $\uplus_A H$
apply (*cut-tac b-ag-group*)
apply (*simp add*:*agop-gop*[*THEN sym*])
apply (*simp add*:*ag-carrier-carrier*[*THEN sym*])
apply (*simp add*:*ar-coset-def*)
apply (*simp add*:*asubGroup-def*)
apply (*simp add*:*Group.rcs-fixed1*[*of b-ag A H*])
done

definition

rind-hom :: [('*a*, '*more*) *aGroup-scheme*, ('*b*, '*more1*) *aGroup-scheme*,
('*a* ⇒ '*b*)] ⇒ ('*a set* ⇒ '*b*) **where**
rind-hom A B f = (λ*X*∈(*set-ar-cos A* (*ker*$_{A,B}$ *f*)). *f* (*SOME x*. *x* ∈ *X*))

abbreviation
RIND-HOM ((*3*-°-,-) [*82*,*82*,*83*]*82*) **where**
f°$_{F,G}$ == *rind-hom F G f*

3.19 Direct product and direct sum of abelian groups, in general case

definition
Un-carrier :: ['*i set*, '*i* ⇒ ('*a*, '*more*) *aGroup-scheme*] ⇒ '*a set* **where**
Un-carrier I A = ⋃{*X*. ∃*i*∈*I*. *X* = *carrier* (*A i*)}

definition
carr-prodag :: ['*i set*, '*i* ⇒ ('*a*, '*more*) *aGroup-scheme*] ⇒ ('*i* ⇒ '*a*) *set* **where**
carr-prodag I A = {*f*. *f* ∈ *extensional I* ∧ *f* ∈ *I* → (*Un-carrier I A*) ∧
(∀ *i*∈*I*. *f i* ∈ *carrier* (*A i*))}

definition
prod-pOp :: ['*i set*, '*i* ⇒ ('*a*, '*more*) *aGroup-scheme*] ⇒
('*i* ⇒ '*a*) ⇒ ('*i* ⇒ '*a*) ⇒ ('*i* ⇒ '*a*) **where**
prod-pOp I A = (λ*f*∈*carr-prodag I A*. λ*g*∈*carr-prodag I A*.
λ*x*∈*I*. (*f x*) ±$_{(A\ x)}$ (*g x*))

definition
prod-mOp :: ['*i set*, '*i* ⇒ ('*a*, '*more*) *aGroup-scheme*] ⇒
('*i* ⇒ '*a*) ⇒ ('*i* ⇒ '*a*) **where**
prod-mOp I A = (λ*f*∈*carr-prodag I A*. λ*x*∈*I*. (−$_{a(A\ x)}$ (*f x*)))

definition
prod-zero :: ['*i set*, '*i* ⇒ ('*a*, '*more*) *aGroup-scheme*] ⇒ ('*i* ⇒ '*a*) **where**
prod-zero I A = (λ*x*∈*I*. **0**$_{(A\ x)}$)

definition
prodag :: ['*i set*, '*i* ⇒ ('*a*, '*more*) *aGroup-scheme*] ⇒ ('*i* ⇒ '*a*) *aGroup* **where**
prodag I A = (| *carrier* = *carr-prodag I A*,
pop = *prod-pOp I A*, *mop* = *prod-mOp I A*,
zero = *prod-zero I A*|)

definition
PRoject :: ['*i set*, '*i* ⇒ ('*a*, '*more*) *aGroup-scheme*, '*i*]
⇒ ('*i* ⇒ '*a*) ⇒ '*a* ((*3*π-,-,-) [*82*,*82*,*83*]*82*) **where**
PRoject I A x = (λ*f* ∈ *carr-prodag I A*. *f x*)

abbreviation
PRODag ((*a*Π- -) [*72*,*73*]*72*) **where**

$a\Pi_I\ A == prodag\ I\ A$

lemma *prodag-comp-i*:$[\![a \in carr\text{-}prodag\ I\ A;\ i \in I]\!] \Longrightarrow (a\ i) \in carrier\ (A\ i)$
by (*simp add*:*carr-prodag-def*)

lemma *prod-pOp-func*:$\forall\, k{\in}I.\ aGroup\ (A\ k) \Longrightarrow$
  $prod\text{-}pOp\ I\ A \in carr\text{-}prodag\ I\ A \to carr\text{-}prodag\ I\ A \to carr\text{-}prodag\ I\ A$
apply (*rule Pi-I*)+
apply(*rename-tac a b*)
apply (*subst carr-prodag-def*) **apply** (*simp add*:*CollectI*)
apply (*rule conjI*)
apply (*simp add*:*prod-pOp-def restrict-def extensional-def*)
apply (*rule conjI*)
apply (*rule Pi-I*)
apply (*simp add*:*prod-pOp-def*)
apply (*subst Un-carrier-def*) **apply** (*simp add*:*CollectI*)
apply (*frule-tac x = x* **in** *bspec*, *assumption*,
    thin-tac $\forall\, k{\in}I.\ aGroup\ (A\ k)$)
apply (*simp add*:*carr-prodag-def*) **apply** (*erule conjE*)+
apply (*thin-tac a* $\in I \to$ *Un-carrier I A*)
apply (*thin-tac b* $\in I \to$ *Un-carrier I A*)
apply (*frule-tac x = x* **in** *bspec*, *assumption*,
    thin-tac $\forall\, i{\in}I.\ a\ i \in carrier\ (A\ i)$,
    frule-tac x = x **in** *bspec*, *assumption*,
    thin-tac $\forall\, i{\in}I.\ b\ i \in carrier\ (A\ i)$)
apply (*frule-tac x = a x* **and** *y = b x* **in** *aGroup.ag-pOp-closed*, *assumption*+)
apply *blast*
apply (*rule ballI*)
apply (*simp add*:*prod-pOp-def*)
apply (*rule-tac A = A i* **and** *x = a i* **and** *y = b i* **in** *aGroup.ag-pOp-closed*)
apply *simp*
apply (*simp add*:*carr-prodag-def*)+
done

lemma *prod-pOp-mem*:$[\![\forall\, k{\in}I.\ aGroup\ (A\ k);\ X \in carr\text{-}prodag\ I\ A;$
  $Y \in carr\text{-}prodag\ I\ A]\!] \Longrightarrow prod\text{-}pOp\ I\ A\ X\ Y \in carr\text{-}prodag\ I\ A$
apply (*frule prod-pOp-func*)
apply (*frule funcset-mem*[*of prod-pOp I A*
      $carr\text{-}prodag\ I\ A\ carr\text{-}prodag\ I\ A \to carr\text{-}prodag\ I\ A$
      X], *assumption*+)
apply (*rule funcset-mem*[*of prod-pOp I A X carr-prodag I A*
      carr-prodag I A Y], *assumption*+)
done

lemma *prod-pOp-mem-i*:$[\![\forall\, k{\in}I.\ aGroup\ (A\ k);\ X \in carr\text{-}prodag\ I\ A;$
  $Y \in carr\text{-}prodag\ I\ A;\ i \in I]\!] \Longrightarrow prod\text{-}pOp\ I\ A\ X\ Y\ i = (X\ i) \pm_{(A\ i)} (Y\ i)$
apply (*simp add*:*prod-pOp-def*)
done

lemma *prod-mOp-func*:$\forall\, k{\in}I.\ aGroup\ (A\ k) \Longrightarrow$
$\quad prod\text{-}mOp\ I\ A \in carr\text{-}prodag\ I\ A \to carr\text{-}prodag\ I\ A$
apply (*rule Pi-I*)
apply (*simp add*:*prod-mOp-def carr-prodag-def*)
apply (*erule conjE*)+
apply (*rule conjI*)
apply (*rule Pi-I*) **apply** *simp*
apply (*rename-tac f j*)
apply (*frule-tac f = f* **and** *x = j* **in** *funcset-mem* [*of - I Un-carrier I A*],
$\quad$ *assumption*+)
apply (*thin-tac f ∈ I → Un-carrier I A*)
apply (*frule-tac x = j* **in** *bspec*, *assumption*,
$\quad$ *thin-tac* $\forall\, k{\in}I.$ *aGroup* (*A k*),
$\quad$ *frule-tac x = j* **in** *bspec*, *assumption*,
$\quad$ *thin-tac* $\forall\, i{\in}I.$ *f i ∈ carrier* (*A i*))
apply (*thin-tac f j ∈ Un-carrier I A*)
apply (*simp add*:*Un-carrier-def*)
apply (*frule aGroup.ag-mOp-closed*, *assumption*+)
apply *blast*
apply (*rule ballI*)
apply (*rule-tac A = A i* **and** *x = x i* **in** *aGroup.ag-mOp-closed*)
apply *simp*+
done

lemma *prod-mOp-mem*:$[\![\forall\, j{\in}I.\ aGroup\ (A\ j);\ X \in carr\text{-}prodag\ I\ A]\!] \Longrightarrow$
$\quad prod\text{-}mOp\ I\ A\ X \in carr\text{-}prodag\ I\ A$
apply (*frule prod-mOp-func*)
apply (*simp add*:*Pi-def*)
done

lemma *prod-mOp-mem-i*:$[\![\forall\, j{\in}I.\ aGroup\ (A\ j);\ X \in carr\text{-}prodag\ I\ A;\ i \in I]\!] \Longrightarrow$
$\quad prod\text{-}mOp\ I\ A\ X\ i = -_{a(A\ i)}\ (X\ i)$
apply (*simp add*:*prod-mOp-def*)
done

lemma *prod-zero-func*:$\forall\, k{\in}I.\ aGroup\ (A\ k) \Longrightarrow$
$\quad prod\text{-}zero\ I\ A \in carr\text{-}prodag\ I\ A$
apply (*simp add*:*prod-zero-def prodag-def*)
apply (*simp add*:*carr-prodag-def*)
apply (*rule conjI*)
apply (*rule Pi-I*) **apply** *simp*
apply (*subgoal-tac aGroup* (*A x*)) **prefer** *2* **apply** *simp*
apply (*thin-tac* $\forall\, k{\in}I.$ *aGroup* (*A k*))
apply (*simp add*:*Un-carrier-def*)
apply (*frule aGroup.ex-zero*)
apply *auto*
apply (*frule-tac x = i* **in** *bspec*, *assumption*,
$\quad$ *thin-tac* $\forall\, k{\in}I.$ *aGroup* (*A k*))
apply (*simp add*:*aGroup.ex-zero*)

done

lemma *prod-zero-i*:⟦∀ *k*∈*I*. *aGroup* (*A k*); *i* ∈ *I*⟧ ⟹
prod-zero I A i = $\mathbf{0}_{(A\ i)}$
by (*simp add*:*prod-zero-def*)

lemma *carr-prodag-mem-eq*:⟦∀ *k*∈*I*. *aGroup* (*A k*); *X* ∈ *carr-prodag I A*;
Y ∈ *carr-prodag I A*; ∀ *l*∈*I*. (*X l*) = (*Y l*) ⟧ ⟹ *X* = *Y*
apply (*simp add*:*carr-prodag-def*)
apply (*erule conjE*)+
apply (*simp add*:*funcset-eq*)
done

lemma *prod-pOp-assoc*:⟦∀ *k*∈*I*. *aGroup* (*A k*); *a* ∈ *carr-prodag I A*;
b ∈ *carr-prodag I A*; *c* ∈ *carr-prodag I A*⟧ ⟹
prod-pOp I A (*prod-pOp I A a b*) *c* =
prod-pOp I A a (*prod-pOp I A b c*)
apply (*frule-tac X* = *a* **and** *Y* = *b* **in** *prod-pOp-mem*[*of I A*], *assumption*+,
frule-tac X = *b* **and** *Y* = *c* **in** *prod-pOp-mem*[*of I A*], *assumption*+,
frule-tac X = *prod-pOp I A a b* **and** *Y* = *c* **in** *prod-pOp-mem*[*of I A*], *assumption*+,
frule-tac X = *a* **and** *Y* = *prod-pOp I A b c* **in** *prod-pOp-mem*[*of I A*], *assumption*+)
apply (*rule carr-prodag-mem-eq*[*of I A*], *assumption*+,
rule ballI)
apply (*simp add*:*prod-pOp-mem-i*)
apply (*frule-tac x* = *l* **in** *bspec*, *assumption*,
thin-tac ∀ *k*∈*I*. *aGroup* (*A k*))
apply (*rule aGroup.ag-pOp-assoc*, *assumption*)
apply (*simp add*:*prodag-comp-i*)+
done

lemma *prod-pOp-commute*:⟦∀ *k*∈*I*. *aGroup* (*A k*); *a* ∈ *carr-prodag I A*;
b ∈ *carr-prodag I A*⟧ ⟹
prod-pOp I A a b = *prod-pOp I A b a*
apply (*frule-tac X* = *a* **and** *Y* = *b* **in** *prod-pOp-mem*[*of I A*], *assumption*+,
frule-tac X = *b* **and** *Y* = *a* **in** *prod-pOp-mem*[*of I A*], *assumption*+)
apply (*rule carr-prodag-mem-eq*[*of I A*], *assumption*+,
rule ballI)
apply (*simp add*:*prod-pOp-mem-i*)
apply (*frule-tac x* = *l* **in** *bspec*, *assumption*,
thin-tac ∀ *k*∈*I*. *aGroup* (*A k*),
rule aGroup.ag-pOp-commute, *assumption*)
apply (*simp add*:*prodag-comp-i*)+
done

lemma *prodag-aGroup*:∀ *k*∈*I*. *aGroup* (*A k*) ⟹ *aGroup* (*prodag I A*)
apply (*simp add*:*aGroup-def* [*of* (*prodag I A*)])
apply (*simp add*:*prodag-def*)

apply (*simp add*:*prod-pOp-func*)
apply (*simp add*:*prod-mOp-func*)
apply (*simp add*:*prod-zero-func*)
apply (*rule conjI*)
apply (*rule allI*, *rule impI*)+
apply (*simp add*:*prod-pOp-assoc*)
apply (*rule conjI*)
apply (*rule allI*, *rule impI*)+
apply (*simp add*:*prod-pOp-commute*)
apply (*rule conjI*)
apply (*rule allI*, *rule impI*)
apply (*frule-tac* $X = a$ **in** *prod-mOp-mem* [*of* I A], *assumption*+)
apply (*frule-tac* $X = \textit{prod-mOp}\ I\ A\ a$ **and** $Y = a$ **in** *prod-pOp-mem*[*of* I A], *assumption*+)
apply (*rule carr-prodag-mem-eq*[*of* I A], *assumption*+)
apply (*simp add*:*prod-zero-func*)
apply (*rule ballI*)
apply (*simp add*:*prod-pOp-mem-i*, *simp add*:*prod-zero-i*) **apply** (*simp add*:*prod-mOp-mem-i*)
apply (*frule-tac* $x = l$ **in** *bspec*, *assumption*, *thin-tac* $\forall k{\in}I.\ aGroup\ (A\ k)$, *rule aGroup.l-m*, *assumption*+, *simp add*:*prodag-comp-i*)
apply (*rule allI*, *rule impI*)
apply (*frule-tac prod-zero-func*[*of* I A], *frule-tac* $Y = a$ **in** *prod-pOp-mem*[*of* I A *prod-zero* I A], *assumption*+)
apply (*rule carr-prodag-mem-eq*[*of* I A], *assumption*+)
apply (*rule ballI*)
apply (*subst prod-pOp-mem-i*[*of* I A], *assumption*+, *subst prod-zero-i*[*of* I A], *assumption*+)
apply (*frule-tac* $x = l$ **in** *bspec*, *assumption*, *rule aGroup.l-zero*, *assumption*+, *simp add*:*prodag-comp-i*)
done

lemma *prodag-carrier*:$\forall k{\in}I.\ aGroup\ (A\ k) \Longrightarrow$
$carrier\ (prodag\ I\ A) = carr\text{-}prodag\ I\ A$
by (*simp add*:*prodag-def*)

lemma *prodag-elemfun*:$[\![\forall k{\in}I.\ aGroup\ (A\ k);\ f \in carrier\ (prodag\ I\ A)]\!] \Longrightarrow$
$f \in extensional\ I$
apply (*simp add*:*prodag-carrier*)
apply (*simp add*:*carr-prodag-def*)
done

lemma *prodag-component*:$[\![f \in carrier\ (prodag\ I\ A);\ i \in I\ ]\!] \Longrightarrow$
$f\ i \in carrier\ (A\ i)$
by (*simp add*:*prodag-def carr-prodag-def*)

lemma *prodag-pOp*:$\forall\, k{\in}I.\ aGroup\ (A\ k) \Longrightarrow$
$\quad pop\ (prodag\ I\ A) = prod\text{-}pOp\ I\ A$
apply (*simp add*:*prodag-def*)
done

lemma *prodag-iOp*:$\forall\, k{\in}I.\ aGroup\ (A\ k) \Longrightarrow$
$\quad mop\ (prodag\ I\ A) = prod\text{-}mOp\ I\ A$
apply (*simp add*:*prodag-def*)
done

lemma *prodag-zero*:$\forall\, k{\in}I.\ aGroup\ (A\ k) \Longrightarrow$
$\quad zero\ (prodag\ I\ A) = prod\text{-}zero\ I\ A$
apply (*simp add*:*prodag-def*)
done

lemma *prodag-sameTr0*:$[\![\forall\, k{\in}I.\ aGroup\ (A\ k);\ \forall\, k{\in}I.\ A\ k = B\ k]\!]$
$\quad \Longrightarrow Un\text{-}carrier\ I\ A = Un\text{-}carrier\ I\ B$
apply (*simp add*:*Un-carrier-def*)
done

lemma *prodag-sameTr1*:$[\![\forall\, k{\in}I.\ aGroup\ (A\ k);\ \forall\, k{\in}I.\ A\ k = B\ k]\!]$
$\quad \Longrightarrow carr\text{-}prodag\ I\ A = carr\text{-}prodag\ I\ B$
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*simp add*:*carr-prodag-def*, (*erule conjE*)+)
apply (*rule Pi-I*)
apply (*subst Un-carrier-def*, *simp*, *blast*)

apply (*rule subsetI*)
apply (*simp add*:*carr-prodag-def*, (*erule conjE*)+)
apply (*rule Pi-I*)
apply (*subst Un-carrier-def*, *simp*)
apply *blast*
done

lemma *prodag-sameTr2*:$[\![\forall\, k{\in}I.\ aGroup\ (A\ k);\ \forall\, k{\in}I.\ A\ k = B\ k]\!]$
$\quad \Longrightarrow prod\text{-}pOp\ I\ A = prod\text{-}pOp\ I\ B$
apply (*frule prodag-sameTr1* [*of I A B*], *assumption*+)
apply (*simp add*:*prod-pOp-def*)
apply (*rule bivar-func-eq*)
apply (*rule ballI*)+
apply (*rule funcset-eq* [*of - I*])
apply (*simp add*:*restrict-def extensional-def*)+
done

lemma *prodag-sameTr3*:$[\![\forall\, k{\in}I.\ aGroup\ (A\ k);\ \forall\, k{\in}I.\ A\ k = B\ k]\!]$
$\quad \Longrightarrow prod\text{-}mOp\ I\ A = prod\text{-}mOp\ I\ B$
apply (*frule prodag-sameTr1* [*of I A B*], *assumption*+)

apply (*simp add*:*prod-mOp-def*)
apply (*rule funcset-eq* [*of* - *carr-prodag I B*])
apply (*simp add*:*restrict-def extensional-def*)
apply (*simp add*:*restrict-def extensional-def*)
apply (*rule ballI*)
apply (*rename-tac g*) **apply** *simp*
apply (*rule funcset-eq* [*of* - *I*])
apply (*simp add*:*restrict-def extensional-def*)+
done

lemma *prodag-sameTr4*:⟦∀ *k*∈*I*. *aGroup* (*A k*); ∀ *k*∈*I*. *A k* = *B k*⟧
⟹ *prod-zero I A* = *prod-zero I B*
apply (*simp add*:*prod-zero-def*)
apply (*rule funcset-eq* [*of* - *I*])
apply (*simp add*:*restrict-def extensional-def*)+
done

lemma *prodag-same*:⟦∀ *k*∈*I*. *aGroup* (*A k*); ∀ *k*∈*I*. *A k* = *B k*⟧
⟹ *prodag I A* = *prodag I B*
apply (*frule prodag-sameTr1*, *assumption*+)
apply (*frule prodag-sameTr2*, *assumption*+)
apply (*frule prodag-sameTr3*, *assumption*+)
apply (*frule prodag-sameTr4*, *assumption*+)
apply (*simp add*:*prodag-def*)
done

lemma *project-mem*:⟦∀ *k*∈*I*. *aGroup* (*A k*); *j* ∈ *I*; *x* ∈ *carrier* (*prodag I A*)⟧ ⟹
(*PRoject I A j*) *x* ∈ *carrier* (*A j*)
apply (*simp add*:*PRoject-def*)
apply (*simp add*:*prodag-def*)
apply (*simp add*:*carr-prodag-def*)
done

lemma *project-aHom*:⟦∀ *k*∈*I*. *aGroup* (*A k*); *j* ∈ *I*⟧ ⟹
PRoject I A j ∈ *aHom* (*prodag I A*) (*A j*)
apply (*simp add*:*aHom-def*)
apply (*rule conjI*)
apply (*simp add*:*project-mem*)
apply (*rule conjI*)
apply (*simp add*:*PRoject-def restrict-def extensional-def*)
apply (*rule allI*, *rule impI*, *simp add*:*prodag-def*)
apply (*rule ballI*)+
apply (*simp add*:*prodag-def*)
apply (*simp add*:*prod-pOp-def*)
apply (*frule-tac X* = *a* **and** *Y* = *b* **in** *prod-pOp-mem*[*of I A*], *assumption*+)
apply (*simp add*:*prod-pOp-def*)
apply (*simp add*:*PRoject-def*)
done

lemma *project-aHom1*:$\forall k{\in}I.\ aGroup\ (A\ k) \Longrightarrow$
$\forall j \in I.\ PRoject\ I\ A\ j \in aHom\ (prodag\ I\ A)\ (A\ j)$
apply (*rule ballI*)
apply (*rule project-aHom*, *assumption*+)
done

definition
A-to-prodag :: [('*a*, '*m*) *aGroup-scheme*, '*i set*, '*i* ⇒('*a* ⇒ '*b*),
'*i* ⇒ ('*b*, '*m1*) *aGroup-scheme*] ⇒ ('*a* ⇒ ('*i* ⇒'*b*)) **where**
A-to-prodag A I S B = (λ*a*∈*carrier A*. λ*k*∈*I*. *S k a*)

lemma *A-to-prodag-mem*:⟦*aGroup A*; ∀ *k*∈*I*. *aGroup* (*B k*); ∀ *k*∈*I*. (*S k*) ∈
aHom A (*B k*); *x* ∈ *carrier A* ⟧ ⟹ *A-to-prodag A I S B x* ∈ *carr-prodag I B*
apply (*simp add*:*carr-prodag-def*)
apply (*rule conjI*)
apply (*simp add*:*A-to-prodag-def extensional-def restrict-def*)
apply (*simp add*:*Pi-def restrict-def A-to-prodag-def*)
apply (*rule conjI*)
apply (*rule allI*) **apply** (*rule impI*)
apply (*simp add*:*Un-carrier-def*)
apply (*rotate-tac 2*,
frule-tac x = *xa* **in** *bspec*, *assumption*,
thin-tac ∀ *k*∈*I*. *S k* ∈ *aHom A* (*B k*))
apply (*simp add*:*aHom-def*) **apply** (*erule conjE*)+
apply (*frule-tac f* = *S xa* **and** *A* = *carrier A* **and** *B* = *carrier* (*B xa*)
and *x* = *x* **in** *funcset-mem*, *assumption*+)
apply *blast*
apply (*rule ballI*)
apply (*rotate-tac 2*,
frule-tac x = *i* **in** *bspec*, *assumption*,
thin-tac ∀ *k*∈*I*. *S k* ∈ *aHom A* (*B k*))
apply (*simp add*:*aHom-def*) **apply** (*erule conjE*)+
apply (*simp add*:*Pi-def*)
done

lemma *A-to-prodag-aHom*:⟦*aGroup A*; ∀ *k*∈*I*. *aGroup* (*B k*); ∀ *k*∈*I*. (*S k*) ∈
aHom A (*B k*) ⟧ ⟹ *A-to-prodag A I S B* ∈ *aHom A* (*a*Π$_I$ *B*)
apply (*simp add*:*aHom-def* [*of A a*Π$_I$ *B*])
apply (*rule conjI*)
apply (*simp add*:*prodag-def A-to-prodag-mem*)

apply (*rule conjI*)
apply (*simp add*:*A-to-prodag-def restrict-def extensional-def*)
apply (*rule ballI*)+
apply (*frule-tac x* = *a* **and** *y* = *b* **in** *aGroup.ag-pOp-closed*, *assumption*+)
apply (*frule-tac x* = *a* ±$_A$ *b* **in** *A-to-prodag-mem* [*of A I B S*],
assumption+)

apply (*frule-tac* $x = a$ **in** *A-to-prodag-mem* [*of A I B S*],
assumption+)
apply (*frule-tac* $x = b$ **in** *A-to-prodag-mem* [*of A I B S*],
assumption+)
apply (*frule prodag-aGroup* [*of I B*])
apply (*frule-tac* $x = a$ **in** *A-to-prodag-mem*[*of A I B S*], *assumption*+,
frule-tac $x = b$ **in** *A-to-prodag-mem*[*of A I B S*], *assumption*+,
frule-tac $x = a \pm_A b$ **in** *A-to-prodag-mem*[*of A I B S*],
assumption+)
apply (*frule prodag-aGroup*[*of I B*],
frule-tac $x = A$*-to-prodag A I S B a* **and**
$y = A$*-to-prodag A I S B b* **in** *aGroup.ag-pOp-closed* [*of* $a\Pi_I$ *B*])
apply (*simp add*:*prodag-carrier*)
apply (*simp add*:*prodag-carrier*)
apply (*rule carr-prodag-mem-eq*, *assumption*+)
apply (*simp add*:*prodag-carrier*)
apply (*rule ballI*)
apply (*simp add*:*A-to-prodag-def prod-pOp-def*)
apply (*rotate-tac 2*,
frule-tac $x = l$ **in** *bspec*, *assumption*,
thin-tac $\forall k{\in}I.\ S\ k \in aHom\ A\ (B\ k)$)
apply (*simp add*:*prodag-def prod-pOp-def*)
apply (*frule-tac* $x = l$ **in** *bspec*, *assumption*,
thin-tac $\forall k{\in}I.\ aGroup\ (B\ k)$)
apply (*simp add*: *aHom-add*)
done

definition
finiteHom :: [*'i set*, *'i* ⇒ (*'a*, *'more*) *aGroup-scheme*, *'i* ⇒ *'a*] ⇒ *bool* **where**
finiteHom I A f ⟷ $f \in$ *carr-prodag I A* ∧ (∃ $H.\ H \subseteq I \wedge$ *finite H* ∧ (
$\forall j \in (I - H).\ (f\ j) = \mathbf{0}_{(A\ j)}$))

definition
carr-dsumag :: [*'i set*, *'i* ⇒ (*'a*, *'more*) *aGroup-scheme*] ⇒ (*'i* ⇒ *'a*) *set* **where**
carr-dsumag I A = {f. *finiteHom I A f*}

definition
dsumag :: [*'i set*, *'i* ⇒ (*'a*, *'more*) *aGroup-scheme*] ⇒ (*'i* ⇒ *'a*) *aGroup* **where**
dsumag I A = (| *carrier* = *carr-dsumag I A*,
pop = *prod-pOp I A*, *mop* = *prod-mOp I A*,
zero = *prod-zero I A*|)

definition
dProj :: [*'i set*, *'i* ⇒ (*'a*, *'more*) *aGroup-scheme*, *'i*]
⇒ (*'i* ⇒ *'a*) ⇒ *'a* **where**
dProj I A x = (λ$f{\in}$*carr-dsumag I A*. $f\ x$)

abbreviation
DSUMag ((*a*⨁ - -) [*72*,*73*]*72*) **where**

$a\bigoplus_I A == dsumag\ I\ A$

lemma *dsum-pOp-func*:$\forall\, k{\in}I.\ aGroup\ (A\ k) \Longrightarrow$
$\quad prod\text{-}pOp\ I\ A \in carr\text{-}dsumag\ I\ A \to carr\text{-}dsumag\ I\ A \to carr\text{-}dsumag\ I\ A$
apply (*rule Pi-I*)+
apply (*subst carr-dsumag-def*) **apply** (*simp add*:*CollectI*)
apply (*simp add*:*finiteHom-def*)
apply (*rule conjI*)
apply (*simp add*:*carr-dsumag-def*) **apply** (*simp add*:*finiteHom-def*)
apply (*erule conjE*)+ **apply** (*simp add*:*prod-pOp-mem*)
apply (*simp add*:*carr-dsumag-def finiteHom-def*) **apply** (*erule conjE*)+
apply ((*erule exE*)+, (*erule conjE*)+)
apply (*frule-tac* $F = H$ **and** $G = Ha$ **in** *finite-UnI*, *assumption*+)
apply (*subgoal-tac* $\forall\, j{\in}I - (H \cup Ha).\ prod\text{-}pOp\ I\ A\ x\ xa\ j = \mathbf{0}_{A\ j}$)
apply (*frule-tac* $A = H$ **and** $B = Ha$ **in** *Un-least*[*of* - *I*], *assumption*+)
apply *blast*

apply (*rule ballI*)
apply (*simp*, (*erule conjE*)+)
apply (*frule-tac* $x = j$ **in** *bspec*, *assumption*,
thin-tac $\forall\, k{\in}I.\ aGroup\ (A\ k)$,
frule-tac $x = j$ **in** *bspec*, *simp*,
thin-tac $\forall\, j{\in}I - H.\ x\ j = \mathbf{0}_{A\ j}$,
frule-tac $x = j$ **in** *bspec*, *simp*,
thin-tac $\forall\, j{\in}I - Ha.\ xa\ j = \mathbf{0}_{A\ j}$)
apply (*simp add*:*prod-pOp-def*)
apply (*rule aGroup.ag-l-zero*) **apply** *simp*
apply (*rule aGroup.ex-zero*) **apply** *assumption*
done

lemma *dsum-pOp-mem*:$[\![\forall\, k{\in}I.\ aGroup\ (A\ k);\ X \in carr\text{-}dsumag\ I\ A;$
$Y \in carr\text{-}dsumag\ I\ A]\!] \Longrightarrow prod\text{-}pOp\ I\ A\ X\ Y \in carr\text{-}dsumag\ I\ A$
apply (*frule dsum-pOp-func*[*of I A*])
apply (*frule funcset-mem*[*of prod-pOp I A carr-dsumag I A*
$carr\text{-}dsumag\ I\ A \to carr\text{-}dsumag\ I\ A\ X$], *assumption*+)
apply (*rule funcset-mem*[*of prod-pOp I A X carr-dsumag I A*
carr-dsumag I A Y], *assumption*+)
done

lemma *dsum-iOp-func*:$\forall\, k{\in}I.\ aGroup\ (A\ k) \Longrightarrow$
$\quad prod\text{-}mOp\ I\ A \in carr\text{-}dsumag\ I\ A \to carr\text{-}dsumag\ I\ A$
apply (*rule Pi-I*)
apply (*simp add*:*carr-dsumag-def*) **apply** (*simp add*:*finiteHom-def*)
apply (*erule conjE*)+ **apply** (*simp add*:*prod-mOp-mem*)
apply (*erule exE*, (*erule conjE*)+)
apply (*simp add*:*prod-mOp-def*)
apply (*subgoal-tac* $\forall\, j{\in}I - H.\ -_{a\,A\ j}\ (x\ j) = \mathbf{0}_{A\ j}$)
apply *blast*

apply (*rule ballI*)
apply (*frule-tac* $x = j$ **in** *bspec*, *simp*,
thin-tac $\forall k{\in}I.\ aGroup\ (A\ k)$,
frule-tac $x = j$ **in** *bspec*, *simp*,
thin-tac $\forall j{\in}I - H.\ x\ j = \mathbf{0}_{A\ j}$, *simp add*:*aGroup.ag-inv-zero*)
done

lemma *dsum-iOp-mem*:$[\![\forall j{\in}I.\ aGroup\ (A\ j);\ X \in carr\text{-}dsumag\ I\ A]\!] \Longrightarrow$
$prod\text{-}mOp\ I\ A\ X \in carr\text{-}dsumag\ I\ A$
apply (*frule dsum-iOp-func*)
apply (*simp add*:*Pi-def*)
done

lemma *dsum-zero-func*:$\forall k{\in}I.\ aGroup\ (A\ k) \Longrightarrow$
$prod\text{-}zero\ I\ A \in carr\text{-}dsumag\ I\ A$
apply (*simp add*:*carr-dsumag-def*) **apply** (*simp add*:*finiteHom-def*)
apply (*rule conjI*) **apply** (*simp add*:*prod-zero-func*)
apply (*subgoal-tac* $\{\} \subseteq I$) **prefer** *2* **apply** *simp*
apply (*subgoal-tac finite* $\{\}$) **prefer** *2* **apply** *simp*
apply (*subgoal-tac* $\forall j{\in}I - \{\}.\ prod\text{-}zero\ I\ A\ j = \mathbf{0}_{A\ j}$)
apply *blast*
apply (*rule ballI*) **apply** *simp*
apply (*simp add*:*prod-zero-def*)
done

lemma *dsumag-sub-prodag*:$\forall k{\in}I.\ aGroup\ (A\ k) \Longrightarrow$
$carr\text{-}dsumag\ I\ A \subseteq carr\text{-}prodag\ I\ A$
by (*rule subsetI*,
simp add:*carr-dsumag-def finiteHom-def*)

lemma *carrier-dsumag*:$\forall k{\in}I.\ aGroup\ (A\ k) \Longrightarrow$
$carrier\ (dsumag\ I\ A) = carr\text{-}dsumag\ I\ A$
apply (*simp add*:*dsumag-def*)
done

lemma *dsumag-elemfun*:$[\![\forall k{\in}I.\ aGroup\ (A\ k);\ f \in carrier\ (dsumag\ I\ A)]\!] \Longrightarrow$
$f \in extensional\ I$
apply (*simp add*:*carrier-dsumag*)
apply (*simp add*:*carr-dsumag-def*) **apply** (*simp add*:*finiteHom-def*)
apply (*erule conjE*) **apply** (*simp add*:*carr-prodag-def*)
done

lemma *dsumag-aGroup*:$\forall k{\in}I.\ aGroup\ (A\ k) \Longrightarrow aGroup\ (dsumag\ I\ A)$
apply (*simp add*:*aGroup-def* [*of dsumag I A*])
apply (*simp add*:*dsumag-def*)
apply (*simp add*:*dsum-pOp-func*)
apply (*simp add*:*dsum-iOp-func*)
apply (*simp add*:*dsum-zero-func*)
apply (*frule dsumag-sub-prodag*[*of I A*])

apply (*rule conjI*)
 apply (*rule allI*, *rule impI*)+
 apply (*frule-tac* $X = a$ **and** $Y = b$ **in** *dsum-pOp-mem*, *assumption*+)
 apply (*frule-tac* $X = b$ **and** $Y = c$ **in** *dsum-pOp-mem*, *assumption*+)
 apply (*frule-tac* $X =$ *prod-pOp I A a b* **and** $Y = c$ **in** *dsum-pOp-mem*,
 assumption+)
 apply (*frule-tac* $Y =$ *prod-pOp I A b c* **and** $X = a$ **in** *dsum-pOp-mem*,
 assumption+)
 apply (*rule carr-prodag-mem-eq* [*of I A*], *assumption*+)
 apply (*simp add*:*subsetD*) **apply** (*simp add*:*subsetD*)
 apply (*rule ballI*)
 apply (*subst prod-pOp-mem-i*, *assumption*+, (*simp add*:*subsetD*)+)
 apply (*subst prod-pOp-mem-i*, *assumption*+)
 apply (*simp add*:*subsetD*)+
 apply (*subst prod-pOp-mem-i*, *assumption*+, (*simp add*:*subsetD*)+)
 apply (*subst prod-pOp-mem-i*, *assumption*+) **apply** (*simp add*:*subsetD*)+
 apply (*thin-tac prod-pOp I A a b* $\in$ *carr-dsumag I A*,
 thin-tac prod-pOp I A b c $\in$ *carr-dsumag I A*,
 thin-tac prod-pOp I A (*prod-pOp I A a b*) $c \in$ *carr-dsumag I A*,
 thin-tac prod-pOp I A a (*prod-pOp I A b c*) $\in$ *carr-dsumag I A*,
 thin-tac carr-dsumag I A $\subseteq$ *carr-prodag I A*)

 apply (*frule-tac* $x = l$ **in** *bspec*, *assumption*,
 thin-tac $\forall k{\in}I.$ *aGroup* (*A k*),
 simp add:*carr-dsumag-def finiteHom-def*, (*erule conjE*)+,
 simp add:*carr-prodag-def*, (*erule conjE*)+)
 apply (*frule-tac* $x = l$ **in** *bspec*, *assumption*,
 thin-tac $\forall i{\in}I.$ *a i* $\in$ *carrier* (*A i*),
 frule-tac $x = l$ **in** *bspec*, *assumption*,
 thin-tac $\forall i{\in}I.$ *b i* $\in$ *carrier* (*A i*),
 frule-tac $x = l$ **in** *bspec*, *assumption*,
 thin-tac $\forall i{\in}I.$ *c i* $\in$ *carrier* (*A i*))
 apply (*simp add*:*aGroup.aassoc*)

apply (*rule conjI*)
 apply (*rule allI*, *rule impI*)+
 apply (*rule carr-prodag-mem-eq* [*of I A*], *assumption*+)
 apply (*frule-tac* $X = a$ **and** $Y = b$ **in** *prod-pOp-mem*[*of I A*],
 (*simp add*:*subsetD*)+)
 apply (*frule-tac* $X = b$ **and** $Y = a$ **in** *prod-pOp-mem*[*of I A*],
 (*simp add*:*subsetD*)+)
 apply (*rule ballI*,
 subst prod-pOp-mem-i, *assumption*+, (*simp add*:*subsetD*)+)
 apply (*subst prod-pOp-mem-i*, *assumption*+, (*simp add*:*subsetD*)+)
 apply (*frule-tac* $x = l$ **in** *bspec*, *assumption*,
 thin-tac $\forall k{\in}I.$ *aGroup* (*A k*))
 apply (*frule-tac* $c = a$ **in** *subsetD*[*of carr-dsumag I A carr-prodag I A*],
 assumption+, *thin-tac* $a \in$ *carr-dsumag I A*,

frule-tac $c = b$ **in** *subsetD*[*of carr-dsumag I A carr-prodag I A*],
assumption+, *thin-tac* $b \in$ *carr-dsumag I A*,
thin-tac carr-dsumag I A $\subseteq$ *carr-prodag I A*)
apply (*simp add*:*carr-prodag-def*, (*erule conjE*)+,
simp add:*aGroup.ag-pOp-commute*)

apply (*rule conjI*)
apply (*rule allI*, *rule impI*)
apply (*frule-tac* $X = a$ **in** *prod-mOp-mem*[*of I A*],
simp add:*subsetD*)
apply (*frule-tac* $X =$ *prod-mOp I A a* **and** $Y = a$ **in** *prod-pOp-mem*[*of I A*],
simp add:*subsetD*, *simp add*:*subsetD*)
apply (*rule carr-prodag-mem-eq* [*of I A*], *assumption*+,
simp add:*prod-zero-func*)
apply (*rule ballI*)
apply (*subst prod-pOp-mem-i*, *assumption*+,
simp add:*subsetD*, *assumption*)
apply (*subst prod-mOp-mem-i*, *assumption*+, *simp add*:*subsetD*, *assumption*)
apply (*simp add*:*prod-zero-i*)
apply (*frule-tac* $x = l$ **in** *bspec*, *assumption*,
thin-tac $\forall k{\in}I.$ *aGroup* (*A k*),
thin-tac prod-mOp I A a $\in$ *carr-prodag I A*,
thin-tac prod-pOp I A (*prod-mOp I A a*) $a \in$ *carr-prodag I A*,
frule-tac $c = a$ **in** *subsetD*[*of carr-dsumag I A carr-prodag I A*],
assumption,
thin-tac carr-dsumag I A $\subseteq$ *carr-prodag I A*,
simp add:*carr-prodag-def*, (*erule conjE*)+)
apply (*frule-tac* $x = l$ **in** *bspec*, *assumption*,
thin-tac $\forall i{\in}I.$ $a\ i \in$ *carrier* (*A i*))
apply (*rule aGroup.l-m*, *assumption*+)

apply (*rule allI*, *rule impI*)
apply (*frule prod-zero-func*[*of I A*])
apply (*frule-tac* $X =$ *prod-zero I A* **and** $Y = a$ **in** *prod-pOp-mem*[*of I A*],
assumption+, *simp add*:*subsetD*)
apply (*rule carr-prodag-mem-eq* [*of I A*], *assumption*+,
simp add:*subsetD*)
apply (*rule ballI*)
apply (*subst prod-pOp-mem-i*, *assumption*+)
apply (*simp add*:*subsetD*, *assumption*)
apply (*simp add*:*prod-zero-i*,
frule-tac $x = l$ **in** *bspec*, *assumption*,
thin-tac $\forall k{\in}I.$ *aGroup* (*A k*),
frule-tac $c = a$ **in** *subsetD*[*of carr-dsumag I A carr-prodag I A*],
assumption+,
thin-tac carr-dsumag I A $\subseteq$ *carr-prodag I A*,
thin-tac $a \in$ *carr-dsumag I A*,
thin-tac prod-pOp I A (*prod-zero I A*) $a \in$ *carr-prodag I A*)
apply (*simp add*:*carr-prodag-def*, (*erule conjE*)+)

apply (*rule aGroup.l-zero*, *assumption*)
apply *blast*
done

lemma *dsumag-pOp*:$\forall\, k{\in}I.\ aGroup\ (A\ k) \Longrightarrow$
pop (*dsumag I A*) = *prod-pOp I A*
apply (*simp add*:*dsumag-def*)
done

lemma *dsumag-mOp*:$\forall\, k{\in}I.\ aGroup\ (A\ k) \Longrightarrow$
mop (*dsumag I A*) = *prod-mOp I A*
apply (*simp add*:*dsumag-def*)
done

lemma *dsumag-zero*:$\forall\, k{\in}I.\ aGroup\ (A\ k) \Longrightarrow$
zero (*dsumag I A*) = *prod-zero I A*
apply (*simp add*:*dsumag-def*)
done

3.19.1 Characterization of a direct product

lemma *direct-prod-mem-eq*:$[\![\forall\, j{\in}I.\ aGroup\ (A\ j);\ f \in carrier\ (a\Pi_I\ A);$
$g \in carrier\ (a\Pi_I\ A);\ \forall\, j{\in}I.\ (PRoject\ I\ A\ j)\ f = (PRoject\ I\ A\ j)\ g]\!] \Longrightarrow$
$f = g$
apply (*rule funcset-eq*[*of f I g*])
apply (*thin-tac* $\forall\, j{\in}I.\ aGroup\ (A\ j)$,
thin-tac $g \in carrier\ (a\Pi_I\ A)$,
thin-tac $\forall\, j{\in}I.\ (\pi_{I,A,j})\ f = (\pi_{I,A,j})\ g$,
simp add:*prodag-def carr-prodag-def*)
apply (*thin-tac* $\forall\, j{\in}I.\ aGroup\ (A\ j)$,
thin-tac $f \in carrier\ (a\Pi_I\ A)$,
thin-tac $\forall\, j{\in}I.\ (\pi_{I,A,j})\ f = (\pi_{I,A,j})\ g$,
simp add:*prodag-def carr-prodag-def*)
apply (*simp add*:*PRoject-def prodag-def*)
done

lemma *map-family-fun*:$[\![\forall\, j{\in}I.\ aGroup\ (A\ j);\ aGroup\ S;$
$\forall\, j{\in}I.\ ((g\ j) \in aHom\ S\ (A\ j));\ x \in carrier\ S]\!] \Longrightarrow$
$(\lambda y \in carrier\ S.\ (\lambda j{\in}I.\ (g\ j)\ y))\ x \in carrier\ (a\Pi_I\ A)$
apply (*simp add*:*prodag-def carr-prodag-def*)
apply (*simp add*:*aHom-mem*)
apply (*rule Pi-I*, *simp add*:*Un-carrier-def*)
apply (*frule-tac* $x = xa$ **in** *bspec*, *assumption*,
thin-tac $\forall\, j{\in}I.\ aGroup\ (A\ j)$,
frule-tac $x = xa$ **in** *bspec*, *assumption*,
thin-tac $\forall\, j{\in}I.\ g\ j \in aHom\ S\ (A\ j)$)
apply (*frule-tac* $G = A\ xa$ **and** $f = g\ xa$ **and** $a = x$ **in** *aHom-mem*[*of S*],
assumption+, *blast*)
done

lemma *map-family-aHom*:⟦∀ *j*∈*I*. *aGroup* (*A j*); *aGroup S*;
∀ *j*∈*I*. ((*g j*) ∈ *aHom S* (*A j*))⟧ ⟹
(λ*y* ∈ *carrier S*. (λ*j*∈*I*. (*g j*) *y*)) ∈ *aHom S* (*a*Π$_I$ *A*)
apply (*subst aHom-def*, *simp*)
apply (*simp add*:*aGroup.ag-pOp-closed*)

apply (*rule conjI*)
apply (*rule Pi-I*)
apply (*rule map-family-fun*[*of I A S g*], *assumption*+)
apply (*rule ballI*)+
apply (*frule-tac x* = *a* **and** *y* = *b* **in** *aGroup.ag-pOp-closed*[*of S*],
assumption+)
apply (*frule-tac x* = *a* ±$_S$ *b* **in** *map-family-fun*[*of I A S g*],
assumption+, *simp*)
apply (*frule-tac x* = *a* **in** *map-family-fun*[*of I A S g*],
assumption+, *simp*,
frule-tac x = *b* **in** *map-family-fun*[*of I A S g*],
assumption+, *simp*)
apply (*frule prodag-aGroup*[*of I A*])
apply (*frule-tac x* = (λ*j*∈*I*. *g j a*) **and** *y* = (λ*j*∈*I*. *g j b*) **in**
aGroup.ag-pOp-closed[*of a*Π$_I$ *A*], *assumption*+)
apply (*simp only*:*prodag-carrier*)

apply (*rule carr-prodag-mem-eq*, *assumption*+)
apply (*rule ballI*)
apply (*subst prodag-def*, *simp add*:*prod-pOp-def*)
apply (*simp add*:*aHom-add*)
done

lemma *map-family-triangle*:⟦∀ *j*∈*I*. *aGroup* (*A j*); *aGroup S*;
∀ *j*∈*I*. ((*g j*) ∈ *aHom S* (*A j*))⟧ ⟹ ∃!*f*. *f* ∈ *aHom S* (*a*Π$_I$ *A*) ∧
(∀ *j*∈*I*. *compos S* (*PRoject I A j*) *f* = (*g j*))
apply (*rule ex-ex1I*)
apply (*frule map-family-aHom*[*of I A S g*], *assumption*+)
apply (*subgoal-tac* ∀ *j*∈*I*. *compos S* (π$_{I,A,j}$) (λ*y*∈*carrier S*. λ*j*∈*I*. *g j y*) = *g j*)
apply *blast*
apply (*rule ballI*)
apply (*simp add*:*compos-def*)
apply (*rule funcset-eq*[*of - carrier S*])
apply (*simp add*:*compose-def*) **apply** (*simp add*:*aHom-def*)
apply (*rule ballI*)
apply (*frule prodag-aGroup*[*of I A*])
apply (*frule prodag-carrier*[*of I A*])
apply (*frule-tac f* = λ*y*∈*carrier S*. λ*j*∈*I*. *g j y* **and** *a* = *x* **in**
aHom-mem[*of S a*Π$_I$ *A*], *assumption*+)
apply (*simp add*:*compose-def*, *simp add*:*PRoject-def*)
apply (*rename-tac f f1*)
apply (*erule conjE*)+

apply (*rule funcset-eq*[*of* - *carrier S*])
apply (*simp add*:*aHom-def*, *simp add*:*aHom-def*)
apply (*rule ballI*)
apply (*frule prodag-aGroup*[*of I A*])
apply (*frule-tac f* = *f* **and** *a* = *x* **in** *aHom-mem*[*of S* $a\Pi_I$ *A*], *assumption*+,
frule-tac f = *f1* **and** *a* = *x* **in** *aHom-mem*[*of S* $a\Pi_I$ *A*], *assumption*+)
apply (*rule-tac f* = *f x* **and** *g* = *f1 x* **in** *direct-prod-mem-eq*[*of I A*],
assumption+)
apply (*rule ballI*)
apply (*rotate-tac 4*,
frule-tac x = *j* **in** *bspec*, *assumption*,
thin-tac ∀ *j*∈*I*. *compos S* ($\pi_{I,A,j}$) *f* = *g j*,
frule-tac x = *j* **in** *bspec*, *assumption*,
thin-tac ∀ *j*∈*I*. *compos S* ($\pi_{I,A,j}$) *f1* = *g j*,
simp add:*compos-def compose-def*)
apply (*subgoal-tac* (λ*x*∈*carrier S*. ($\pi_{I,A,j}$) (*f x*)) *x* = *g j x*,
subgoal-tac (λ*x*∈*carrier S*. ($\pi_{I,A,j}$) (*f1 x*)) *x* = *g j x*,
thin-tac (λ*x*∈*carrier S*. ($\pi_{I,A,j}$) (*f x*)) = *g j*,
thin-tac (λ*x*∈*carrier S*. ($\pi_{I,A,j}$) (*f1 x*)) = *g j*,
simp+)
done

lemma *Ag-ind-triangle*:⟦∀ *j*∈*I*. *aGroup* (*A j*); *j* ∈ *I*; *f* ∈ *carrier* ($a\Pi_I$ *A*) → *B*;
bij-to f (*carrier* ($a\Pi_I$ *A*)) (*B*::'*d set*); *j* ∈ *I*⟧ ⟹
compos ($a\Pi_I$ *A*) (*compos* (*Ag-ind* ($a\Pi_I$ *A*) *f*)(*PRoject I A j*) ($ainvf_{(a\Pi_I\ A)}$, (*Ag-ind* ($a\Pi_I$ *A*) *f*)
(*Agii* ($a\Pi_I$ *A*) *f*))) (*Agii* ($a\Pi_I$ *A*) *f*) =
PRoject I A j
apply (*frule prodag-aGroup*[*of I A*])
apply (*frule aGroup.Ag-ind-aGroup*[*of* $a\Pi_I$ *A f B*], *assumption*+)
apply (*simp add*:*compos-def*)
apply (*rule funcset-eq*[*of* - *carrier* ($a\Pi_I$ *A*)])
apply *simp*
apply (*simp add*:*PRoject-def prodag-carrier extensional-def*)
apply (*rule ballI*)
apply (*simp add*:*compose-def invfun-l*)
apply (*simp add*:*aGroup.Agii-mem*)
apply (*frule Ag-ind-bijec*[*of* $a\Pi_I$ *A f B*], *assumption*+)
apply (*frule-tac x* = *x* **in** *ainvf-l*[*of* $a\Pi_I$ *A Ag-ind* ($a\Pi_I$ *A*) *f*
Agii ($a\Pi_I$ *A*) *f*], *assumption*+)
apply *simp*
done

definition
ProjInd :: ['*i set*, '*i* ⇒ ('*a*, '*m*) *aGroup-scheme*, ('*i* ⇒ '*a*) ⇒ '*d*, '*i*] ⇒
('*d* ⇒ '*a*) **where**
ProjInd I A f j = *compos* (*Ag-ind* ($a\Pi_I$ *A*) *f*)(*PRoject I A j*) ($ainvf_{(a\Pi_I\ A)}$, (*Ag-ind* ($a\Pi_I$ *A*) *f*)
(*Agii* ($a\Pi_I$ *A*) *f*))

lemma *ProjInd-aHom*:⟦∀ *j*∈ *I*. *aGroup* (*A j*); *j* ∈ *I*; *f* ∈ *carrier* (*a*Π$_I$ *A*) → *B*;
bij-to f (*carrier* (*a*Π$_I$ *A*)) (*B*::'*d set*); *j* ∈ *I*⟧ ⟹
(*ProjInd I A f j*) ∈ *aHom* (*Ag-ind* (*a*Π$_I$ *A*) *f*) (*A j*)
apply (*frule prodag-aGroup*[*of I A*])
apply (*frule aGroup.Ag-ind-aGroup*[*of a*Π$_I$ *A f B*], *assumption*+)
apply (*frule-tac x* = *j* **in** *bspec*, *assumption*)
apply (*frule aGroup.Ag-ind-aHom*[*of a*Π$_I$ *A f B*], *assumption*+)
apply (*simp add*:*ProjInd-def*)
apply (*frule Ag-ind-bijec*[*of a*Π$_I$ *A f B*], *assumption*+)
apply (*frule ainvf-aHom*[*of a*Π$_I$ *A Ag-ind* (*a*Π$_I$ *A*) *f Agii* (*a*Π$_I$ *A*) *f*],
assumption+)
apply (*frule project-aHom*[*of I A j*], *assumption*)
apply (*simp add*:*aHom-compos*)
done

lemma *ProjInd-aHom1*:⟦∀ *j*∈ *I*. *aGroup* (*A j*); *f* ∈ *carrier* (*a*Π$_I$ *A*) → *B*;
bij-to f (*carrier* (*a*Π$_I$ *A*)) (*B*::'*d set*)⟧ ⟹
∀ *j*∈*I*. (*ProjInd I A f j*) ∈ *aHom* (*Ag-ind* (*a*Π$_I$ *A*) *f*) (*A j*)
apply (*rule ballI*)
apply (*simp add*:*ProjInd-aHom*)
done

lemma *ProjInd-mem-eq*:⟦∀ *j*∈*I*. *aGroup* (*A j*); *f* ∈ *carrier* (*a*Π$_I$ *A*) → *B*;
bij-to f (*carrier* (*a*Π$_I$ *A*)) *B*; *aGroup S*; *x* ∈ *carrier* (*Ag-ind* (*a*Π$_I$ *A*) *f*);
y ∈ *carrier* (*Ag-ind* (*a*Π$_I$ *A*) *f*);
∀ *j*∈*I*. (*ProjInd I A f j x* = *ProjInd I A f j y*)⟧ ⟹ *x* = *y*
apply (*simp add*:*ProjInd-def*)
apply (*simp add*:*compos-def compose-def*)
apply (*frule prodag-aGroup*[*of I A*])
apply (*frule aGroup.Ag-ind-aGroup*[*of a*Π$_I$ *A f B*], *assumption*+)
apply (*frule aGroup.Ag-ind-aHom*[*of a*Π$_I$ *A f B*], *assumption*+)
apply (*frule Ag-ind-bijec*[*of a*Π$_I$ *A f B*], *assumption*+)
apply (*frule ainvf-aHom*[*of a*Π$_I$ *A Ag-ind* (*a*Π$_I$ *A*) *f Agii* (*a*Π$_I$ *A*) *f*],
assumption+)
apply (*frule aHom-mem*[*of Ag-ind* (*a*Π$_I$ *A*) *f a*Π$_I$ *A ainvf*$_{(a\Pi_I\ A),Ag\text{-}ind\ (a\Pi_I\ A)\ f}$ *Agii* (*a*Π$_I$ *A*) *f x*], *assumption*+,
frule aHom-mem[*of Ag-ind* (*a*Π$_I$ *A*) *f a*Π$_I$ *A ainvf*$_{(a\Pi_I\ A),Ag\text{-}ind\ (a\Pi_I\ A)\ f}$ *Agii* (*a*Π$_I$ *A*) *f y*], *assumption*+)

apply (*frule direct-prod-mem-eq*[*of I A* (*ainvf*$_{(a\Pi_I\ A),Ag\text{-}ind\ (a\Pi_I\ A)\ f}$ *Agii* (*a*Π$_I$ *A*) *f*) *x* (*ainvf*$_{(a\Pi_I\ A),Ag\text{-}ind\ (a\Pi_I\ A)\ f}$ *Agii* (*a*Π$_I$ *A*) *f*) *y*], *assumption*+)
apply (*thin-tac ainvf*$_{(a\Pi_I\ A),Ag\text{-}ind\ (a\Pi_I\ A)\ f}$ *Agii* (*a*Π$_I$ *A*) *f*
∈ *aHom* (*Ag-ind* (*a*Π$_I$ *A*) *f*) (*a*Π$_I$ *A*))
apply (*frule ainvf-bijec*[*of a*Π$_I$ *A Ag-ind* (*a*Π$_I$ *A*) *f Agii* (*a*Π$_I$ *A*) *f*],
assumption+)

apply (*thin-tac bijec*$_{(a\Pi_I\ A),Ag\text{-}ind\ (a\Pi_I\ A)\ f}$ *Agii* $(a\Pi_I\ A)\ f$)
apply (*unfold bijec-def*, *frule conjunct1*, *fold bijec-def*)
apply (*frule injec-inj-on*[*of Ag-ind* $(a\Pi_I\ A)\ f\ a\Pi_I\ A$ *ainvf*$_{(a\Pi_I\ A),Ag\text{-}ind\ (a\Pi_I\ A)\ f}$ *Agii* $(a\Pi_I\ A)\ f$], *assumption*+)
apply (*simp add*:*injective-iff*[*THEN sym*, *of ainvf*$_{(a\Pi_I\ A),Ag\text{-}ind\ (a\Pi_I\ A)\ f}$ *Agii* $(a\Pi_I\ A)\ f$ *carrier* $(Ag\text{-}ind\ (a\Pi_I\ A)\ f)\ x\ y$])
done

lemma *ProjInd-mem-eq1*:⟦$\forall j{\in}I.$ *aGroup* $(A\ j)$; $f \in$ *carrier* $(a\Pi_I\ A) \to B$;
bij-to f (*carrier* $(a\Pi_I\ A)$) B; *aGroup* S;
$h \in$ *aHom* $(Ag\text{-}ind\ (a\Pi_I\ A)\ f)\ (Ag\text{-}ind\ (a\Pi_I\ A)\ f)$;
$\forall j{\in}I.$ *compos* $(Ag\text{-}ind\ (a\Pi_I\ A)\ f)\ (ProjInd\ I\ A\ f\ j)\ h = ProjInd\ I\ A\ f\ j$⟧
$\Longrightarrow h =$ *ag-idmap* $(Ag\text{-}ind\ (a\Pi_I\ A)\ f)$
apply (*rule funcset-eq*[*of* - *carrier* $(Ag\text{-}ind\ (a\Pi_I\ A)\ f)$])
apply (*simp add*:*aHom-def*)
apply (*simp add*:*ag-idmap-def*)
apply (*rule ballI*)
apply (*simp add*:*ag-idmap-def*)
apply (*frule prodag-aGroup*[*of I A*],
frule aGroup.Ag-ind-aGroup[*of* $a\Pi_I\ A\ f\ B$], *assumption*+)
apply (*frule-tac* $a = x$ **in** *aHom-mem*[*of Ag-ind* $(a\Pi_I\ A)\ f$ *Ag-ind* $(a\Pi_I\ A)\ f$ h], *assumption*+)
apply (*rule-tac* $x = h\ x$ **and** $y = x$ **in** *ProjInd-mem-eq*[*of I A f B S*], *assumption*+)
apply (*rotate-tac 1*,
rule ballI,
frule-tac $x = j$ **in** *bspec*, *assumption*,
thin-tac $\forall j{\in}I.$ *compos* $(Ag\text{-}ind\ (a\Pi_I\ A)\ f)\ (ProjInd\ I\ A\ f\ j)\ h =$
ProjInd I A f j)
apply (*simp add*:*compos-def compose-def*)
apply (*subgoal-tac* $(\lambda x{\in}carrier\ (Ag\text{-}ind\ (a\Pi_I\ A)\ f).\ ProjInd\ I\ A\ f\ j\ (h\ x))\ x$
$= ProjInd\ I\ A\ f\ j\ x$,
thin-tac $(\lambda x{\in}carrier\ (Ag\text{-}ind\ (a\Pi_I\ A)\ f).\ ProjInd\ I\ A\ f\ j\ (h\ x)) =$
ProjInd I A f j)
apply *simp*+
done

lemma *Ag-ind-triangle1*:⟦$\forall j{\in}I.$ *aGroup* $(A\ j)$; $f \in$ *carrier* $(a\Pi_I\ A) \to B$;
bij-to f (*carrier* $(a\Pi_I\ A)$) $(B{::}'d\ set)$; $j \in I$⟧ $\Longrightarrow$
compos $(a\Pi_I\ A)$ $(ProjInd\ I\ A\ f\ j)$ $(Agii\ (a\Pi_I\ A)\ f) =$ *PRoject I A j*
apply (*simp add*:*ProjInd-def*)
apply (*simp add*:*Ag-ind-triangle*)
done

lemma *map-family-triangle1*:⟦$\forall j{\in}I.$ *aGroup* $(A\ j)$; $f \in$ *carrier* $(a\Pi_I\ A) \to B$;
bij-to f (*carrier* $(a\Pi_I\ A)$) $(B{::}'d\ set)$; *aGroup* S;
$\forall j{\in}I.\ ((g\ j) \in aHom\ S\ (A\ j))$⟧ $\Longrightarrow \exists!h.\ h \in aHom\ S\ (Ag\text{-}ind\ (a\Pi_I\ A)\ f) \wedge$
$(\forall j{\in}I.\ compos\ S\ (ProjInd\ I\ A\ f\ j)\ h = (g\ j))$
apply (*frule prodag-aGroup*[*of I A*])

apply (*frule aGroup.Ag-ind-aGroup*[*of* $a\Pi_I$ *A f B*], *assumption*+)
apply (*frule Ag-ind-bijec*[*of* $a\Pi_I$ *A f B*], *assumption*+)
apply (*rule ex-ex1I*)
apply (*frule map-family-triangle*[*of I A S g*], *assumption*+)
apply (*frule ex1-implies-ex*)
apply (*erule exE*)
apply (*erule conjE*)
apply (*unfold bijec-def*, *frule conjunct2*, *fold bijec-def*)
apply (*unfold surjec-def*, *frule conjunct1*, *fold surjec-def*)
apply (*rename-tac fa*,
frule-tac f = *fa* **in** *aHom-compos*[*of S* $a\Pi_I$ *A Ag-ind* (*a*Π_I *A*) *f* -
Agii (*a*Π_I *A*) *f*], *assumption*+)
apply (*subgoal-tac* $\forall j{\in}I$. *compos S* (*ProjInd I A f j*)
(*compos S* (*Agii* (*a*Π_I *A*) *f*) *fa*) = *g j*)
apply *blast*
apply (*rule ballI*)
apply (*frule-tac N* = *A j* **and** *f* = *fa* **and** *g* = *Agii* (*a*Π_I *A*) *f* **and**
h = *ProjInd I A f j* **in** *aHom-compos-assoc*[*of S* $a\Pi_I$ *A Ag-ind* (*a*Π_I *A*) *f*],
assumption+) **apply** *simp* **apply** *assumption*+
apply (*simp add*:*ProjInd-aHom*)
apply *simp*
apply (*thin-tac compos S* (*ProjInd I A f j*) (*compos S* (*Agii* (*a*Π_I *A*) *f*) *fa*) =
compos S (*compos* (*a*Π_I *A*) (*ProjInd I A f j*) (*Agii* (*a*Π_I *A*) *f*)) *fa*)
apply (*simp add*:*Ag-ind-triangle1*)
apply (*rename-tac h h1*)
apply (*erule conjE*)+
apply (*rule funcset-eq*[*of* - *carrier S*])
apply (*simp add*:*aHom-def*, *simp add*:*aHom-def*)
apply (*rule ballI*)
apply (*simp add*:*compos-def*)

apply (*frule-tac f* = *h* **and** *a* = *x* **in** *aHom-mem*[*of S Ag-ind* (*a*Π_I *A*) *f*],
assumption+,
frule-tac f = *h1* **and** *a* = *x* **in** *aHom-mem*[*of S Ag-ind* (*a*Π_I *A*) *f*],
assumption+)
apply (*rule-tac x* = *h x* **and** *y* = *h1 x* **in** *ProjInd-mem-eq*[*of I A f*
B S], *assumption*+)
apply (*rule ballI*)
apply (*rotate-tac 5*,
frule-tac x = *j* **in** *bspec*, *assumption*,
thin-tac $\forall j{\in}I$. *compose* (*carrier S*) (*ProjInd I A f j*) *h* = *g j*,
frule-tac x = *j* **in** *bspec*, *assumption*,
thin-tac $\forall j{\in}I$. *compose* (*carrier S*) (*ProjInd I A f j*) *h1* = *g j*)
apply (*simp add*:*compose-def*,
subgoal-tac ($\lambda x{\in}$*carrier S*. *ProjInd I A f j* (*h x*)) *x* = *g j x*,
thin-tac ($\lambda x{\in}$*carrier S*. *ProjInd I A f j* (*h x*)) = *g j*,
subgoal-tac ($\lambda x{\in}$*carrier S*. *ProjInd I A f j* (*h1 x*)) *x* = *g j x*,
thin-tac ($\lambda x{\in}$*carrier S*. *ProjInd I A f j* (*h1 x*)) = *g j*, *simp*+)
done

lemma *map-family-triangle2*:⟦$I \neq \{\}$; $\forall j{\in}I.\ aGroup\ (A\ j)$; $aGroup\ S$;
$\forall j{\in}I.\ g\ j \in aHom\ S\ (A\ j)$; $ff \in carrier\ (a\Pi_I\ A) \to B$;
bij-to ff (*carrier* ($a\Pi_I\ A$)) B;
$h1 \in aHom\ (Ag\text{-}ind\ (a\Pi_I\ A)\ ff)\ S$;
$\forall j{\in}I.\ compos\ (Ag\text{-}ind\ (a\Pi_I\ A)\ ff)\ (g\ j)\ h1 = ProjInd\ I\ A\ ff\ j$;
$h2 \in aHom\ S\ (Ag\text{-}ind\ (a\Pi_I\ A)\ ff)$;
$\forall j{\in}I.\ compos\ S\ (ProjInd\ I\ A\ ff\ j)\ h2 = g\ j$⟧
$\Longrightarrow \forall j{\in}I.\ compos\ (Ag\text{-}ind\ (a\Pi_I\ A)\ ff)\ (ProjInd\ I\ A\ ff\ j)$
$(compos\ (Ag\text{-}ind\ (a\Pi_I\ A)\ ff)\ h2\ h1) =$
$ProjInd\ I\ A\ ff\ j$
apply (*rule ballI*)
apply (*frule prodag-aGroup*[*of I A*])
apply (*frule-tac* $f = ff$ **in** *aGroup.Ag-ind-aGroup*[*of* $a\Pi_I\ A$ - B], *assumption*+)

apply (*frule-tac* $N = A\ j$ **and** $h = ProjInd\ I\ A\ ff\ j$ **in** *aHom-compos-assoc*[*of Ag-ind* ($a\Pi_I\ A$) *ff S Ag-ind* ($a\Pi_I\ A$) *ff* - *h1 h2*], *assumption*+)
apply *simp* **apply** *assumption*+ **apply** (*simp add*:*ProjInd-aHom*)
apply *simp*
done

lemma *map-family-triangle3*:⟦$\forall j{\in}I.\ aGroup\ (A\ j)$; $aGroup\ S$; $aGroup\ S1$;
$\forall j{\in}I.\ f\ j \in aHom\ S\ (A\ j)$; $\forall j{\in}I.\ g\ j \in aHom\ S1\ (A\ j)$;
$h1 \in aHom\ S1\ S$; $h2 \in aHom\ S\ S1$;
$\forall j{\in}I.\ compos\ S\ (g\ j)\ h2 = f\ j$;
$\forall j{\in}I.\ compos\ S1\ (f\ j)\ h1 = g\ j$⟧
$\Longrightarrow \forall j{\in}I.\ compos\ S\ (f\ j)\ (compos\ S\ h1\ h2) = f\ j$
apply (*rule ballI*)
apply (*frule-tac* $h = f\ j$ **and** $N = A\ j$ **in** *aHom-compos-assoc*[*of S S1 S* - *h2 h1*], *assumption*+)
apply *simp* **apply** *assumption*+ **apply** *simp*
apply *simp*
done

lemma *map-family-triangle4*:⟦$\forall j{\in}I.\ aGroup\ (A\ j)$; $aGroup\ S$;
$\forall j{\in}I.\ f\ j \in aHom\ S\ (A\ j)$⟧ $\Longrightarrow$
$\forall j{\in}I.\ compos\ S\ (f\ j)\ (ag\text{-}idmap\ S) = f\ j$
apply (*rule ballI*)
apply (*frule-tac* $x = j$ **in** *bspec*, *assumption*,
thin-tac $\forall j{\in}I.\ aGroup\ (A\ j)$,
frule-tac $x = j$ **in** *bspec*, *assumption*,
thin-tac $\forall j{\in}I.\ f\ j \in aHom\ S\ (A\ j)$)
apply (*simp add*:*compos-aI-r*)
done

lemma *prod-triangle*:⟦$I \neq \{\}$; $\forall j{\in}I.\ aGroup\ (A\ j)$; $aGroup\ S$;
$\forall j{\in}I.\ g\ j \in aHom\ S\ (A\ j)$; $ff \in carrier\ (a\Pi_I\ A) \to B$;
bij-to ff (*carrier* ($a\Pi_I\ A$)) B;
$h1 \in aHom\ (Ag\text{-}ind\ (a\Pi_I\ A)\ ff)\ S$;

$\forall j{\in}I.\ compos\ (Ag\text{-}ind\ (a\Pi_I\ A)\ ff)\ (g\ j)\ h1 = ProjInd\ I\ A\ ff\ j;$
$h2 \in aHom\ S\ (Ag\text{-}ind\ (a\Pi_I\ A)\ ff);$
$\forall j{\in}I.\ compos\ S\ (ProjInd\ I\ A\ ff\ j)\ h2 = g\ j\rrbracket$
$\Longrightarrow (compos\ (Ag\text{-}ind\ (a\Pi_I\ A)\ ff)\ h2\ h1) = ag\text{-}idmap\ (Ag\text{-}ind\ (a\Pi_I\ A)\ ff)$
apply (*frule map-family-triangle2*[*of I A S g ff B h1 h2*], *assumption*+)
apply (*frule prodag-aGroup*[*of I A*],
frule aGroup.Ag-ind-aGroup[*of* $a\Pi_I$ *A ff B*], *assumption*+)
apply (*frule aHom-compos*[*of Ag-ind* $(a\Pi_I\ A)$ *ff S Ag-ind* $(a\Pi_I\ A)$ *ff h1 h2*], *assumption*+)
apply (*rule ProjInd-mem-eq1*[*of I A ff B S compos* (*Ag-ind* $(a\Pi_I\ A)$ *ff*) *h2 h1*], *assumption*+)
done

lemma *characterization-prodag*:$\llbracket I \neq \{\};\ \forall j{\in}(I::'i\ set).\ aGroup\ ((A\ j)::('a, 'm)\ aGroup\text{-}scheme);\ aGroup\ (S::'d\ aGroup);\ \forall j{\in}I.\ ((g\ j) \in aHom\ S\ (A\ j));\ \exists ff.\ ff \in carrier\ (a\Pi_I\ A) \to (B::'d\ set) \wedge bij\text{-}to\ ff\ (carrier\ (a\Pi_I\ A))\ B;\ \forall (S'::'d\ aGroup).\ aGroup\ S' \longrightarrow (\forall g'.\ (\forall j{\in}I.\ (g'\ j) \in aHom\ S'\ (A\ j)) \longrightarrow (\exists!\ f.\ f \in aHom\ S'\ S \wedge (\forall j{\in}I.\ compos\ S'\ (g\ j)\ f = (g'\ j)))))\rrbracket \Longrightarrow \exists h.\ bijec_{(prodag\ I\ A),S}\ h$
apply (*frule prodag-aGroup*[*of I A*])
apply (*erule exE*)
apply (*frule-tac f = ff* **in** *aGroup.Ag-ind-aGroup*[*of* $a\Pi_I$ *A - B*], *erule conjE*, *assumption*, *simp*, *erule conjE*)
apply (*frule aGroup.Ag-ind-aGroup*[*of* $a\Pi_I$ *A - B*], *assumption*+, *frule-tac a = S* **in** *forall-spec*, *assumption*+)
apply (*rotate-tac −1*, *frule-tac x = g* **in** *spec*, *thin-tac* $\forall g'.\ \forall j{\in}I.\ g'\ j \in aHom\ S\ (A\ j) \longrightarrow (\exists!f.\ f \in aHom\ S\ S \wedge (\forall j{\in}I.\ compos\ S\ (g\ j)\ f = g'\ j))$)
apply (*frule-tac a = Ag-ind* $(a\Pi_I\ A)$ *ff* **in** *forall-spec*, *assumption*+, *thin-tac* $\forall S'.\ aGroup\ S' \longrightarrow (\forall g'.\ \forall j{\in}I.\ g'\ j \in aHom\ S'\ (A\ j) \longrightarrow (\exists!f.\ f \in aHom\ S'\ S \wedge (\forall j{\in}I.\ compos\ S'\ (g\ j)\ f = g'\ j)))$)
apply (*frule-tac x = ProjInd I A ff* **in** *spec*, *thin-tac* $\forall g'.\ \forall j{\in}I.\ g'\ j \in aHom\ (Ag\text{-}ind\ (a\Pi_I\ A)\ ff)\ (A\ j) \longrightarrow (\exists!f.\ f \in aHom\ (Ag\text{-}ind\ (a\Pi_I\ A)\ ff)\ S \wedge (\forall j{\in}I.\ compos\ (Ag\text{-}ind\ (a\Pi_I\ A)\ ff)\ (g\ j)\ f = g'\ j)))$)
apply (*frule-tac f = ff* **in** *ProjInd-aHom1*[*of I A - B*], *assumption*+)
apply (*simp add*:*nonempty-ex*[*of I*], *rotate-tac −2*, *frule ex1-implies-ex*, *thin-tac* $\exists!f.\ f \in aHom\ (Ag\text{-}ind\ (a\Pi_I\ A)\ ff)\ S \wedge (\forall j{\in}I.\ compos\ (Ag\text{-}ind\ (a\Pi_I\ A)\ ff)\ (g\ j)\ f = ProjInd\ I\ A\ ff\ j)$, *rotate-tac −1*, *erule exE*, *erule conjE*)
apply (*rename-tac ff h1*, *frule-tac f = ff* **in** *map-family-triangle1*[*of I A - B S g*], *assumption*+,

rotate-tac -1,
frule ex1-implies-ex,
thin-tac $\exists!h.\ h \in aHom\ S\ (Ag\text{-}ind\ (a\Pi_I\ A)\ ff) \wedge$
$(\forall j{\in}I.\ compos\ S\ (ProjInd\ I\ A\ ff\ j)\ h = g\ j)$,
rotate-tac -1,
erule exE, *erule conjE*)
apply (*rename-tac ff h1 h2*)
apply (*frule-tac ff* $=$ *ff* **and** *?h1.0* $=$ *h1* **and** *?h2.0* $=$ *h2* **in** *prod-triangle*[*of I A S g - B*], *assumption*+,
frule-tac ?S1.0 $=$ *Ag-ind* $(a\Pi_I\ A)$ *ff* **in** *map-family-triangle3*[*of I A S - g*],
assumption+,
frule-tac f $=$ *h2* **and** *g* $=$ *h1* **and** *M* $=$ *Ag-ind* $(a\Pi_I\ A)$ *ff* **in**
aHom-compos[*of S - S*], *assumption*+)
apply (*erule ex1E*)
apply (*rotate-tac* -1,
frule-tac x $=$ *compos S h1 h2* **in** *spec*,
frule map-family-triangle4[*of I A S g*], *assumption*+,
frule aGroup.aI-aHom[*of S*])
apply (*frule-tac x* $= aI_S$ **in** *spec*,
thin-tac $\forall y.\ y \in aHom\ S\ S \wedge (\forall j{\in}I.\ compos\ S\ (g\ j)\ y = g\ j) \longrightarrow y = f$,
simp,
thin-tac $\forall j{\in}I.\ compos\ S\ (ProjInd\ I\ A\ ff\ j)\ h2 = g\ j$,
thin-tac $\forall j{\in}I.\ compos\ S\ (g\ j)\ f = g\ j$,
thin-tac $\forall j{\in}I.\ compos\ (Ag\text{-}ind\ (a\Pi_I\ A)\ ff)\ (g\ j)\ h1 = ProjInd\ I\ A\ ff\ j$)
apply (*rotate-tac* -1, *frule sym*, *thin-tac* $aI_S = f$, *simp*,
frule-tac A $=$ *Ag-ind* $(a\Pi_I\ A)$ *ff* **and** *f* $=$ *h1* **and** *g* $=$ *h2* **in**
compos-aI-inj[*of - S*], *assumption*+,
frule-tac B $=$ *Ag-ind* $(a\Pi_I\ A)$ *ff* **and** *f* $=$ *h2* **and** *g* $=$ *h1* **in**
compos-aI-surj[*of S*], *assumption*+)
apply (*frule-tac f* $=$ *ff* **in** *Ag-ind-bijec*[*of* $a\Pi_I\ A$ *- B*], *assumption*+,
frule-tac F $=$ *Ag-ind* $(a\Pi_I\ A)$ *ff* **and** *f* $=$ *Agii* $(a\Pi_I\ A)$ *ff* **and** *g* $=$ *h1*
in *compos-bijec*[*of* $a\Pi_I\ A$ *- S*], *assumption*+)
apply (*subst bijec-def*, *simp*)
apply (*thin-tac bijec*$_{(a\Pi_I\ A),Ag\text{-}ind\ (a\Pi_I\ A)\ ff}$ *Agii* $(a\Pi_I\ A)$ *ff*,
thin-tac injec$_{Ag\text{-}ind\ (a\Pi_I\ A)\ ff,S}$ *h1*,
thin-tac surjec$_{Ag\text{-}ind\ (a\Pi_I\ A)\ ff,S}$ *h1*)
apply (*rule exI*, *simp*)
done

Chapter 4

Ring theory

4.1 Definition of a ring and an ideal

record $'a$ *Ring* = $'a$ *aGroup* +
  tp :: $['a, 'a] \Rightarrow 'a$ (**infixl** $\cdot_r 1$ *70*)
  un :: $'a$ $(1_r 1)$

locale *Ring* =
 fixes R (**structure**)

 assumes
  pop-closed: $pop\ R \in carrier\ R \to carrier\ R \to carrier\ R$
 and *pop-aassoc* : $[\![a \in carrier\ R;\ b \in carrier\ R;\ c \in carrier\ R]\!] \Longrightarrow$
  $(a \pm b) \pm c = a \pm (b \pm c)$
 and *pop-commute*:$[\![a \in carrier\ R;\ b \in carrier\ R]\!] \Longrightarrow a \pm b = b \pm a$
 and *mop-closed*:$mop\ R \in carrier\ R \to carrier\ R$
 and *l-m* :$a \in carrier\ R \Longrightarrow\ (-_a\ a) \pm a = \mathbf{0}$
 and *ex-zero*: $\mathbf{0} \in carrier\ R$
 and *l-zero*:$a \in carrier\ R \Longrightarrow \mathbf{0} \pm a = a$
 and *tp-closed*: $tp\ R \in carrier\ R \to carrier\ R \to carrier\ R$
 and *tp-assoc* : $[\![a \in carrier\ R;\ b \in carrier\ R;\ c \in carrier\ R]\!] \Longrightarrow$
  $(a \cdot_r b) \cdot_r c = a \cdot_r (b \cdot_r c)$
 and *tp-commute*: $[\![a \in carrier\ R;\ b \in carrier\ R]\!] \Longrightarrow a \cdot_r b = b \cdot_r a$
 and *un-closed*: $(1_r) \in carrier\ R$
 and *rg-distrib*: $[\![a \in carrier\ R;\ b \in carrier\ R;\ c \in carrier\ R]\!] \Longrightarrow$
  $a \cdot_r (b \pm c) = a \cdot_r b \pm a \cdot_r c$
 and *rg-l-unit*: $a \in carrier\ R \Longrightarrow (1_r) \cdot_r a = a$

definition
  zeroring :: $('a, 'more)$ *Ring-scheme* $\Rightarrow$ *bool* **where**
  $zeroring\ R \longleftrightarrow Ring\ R \wedge carrier\ R = \{\mathbf{0}_R\}$

primrec *nscal* :: $('a, 'more)$ *Ring-scheme* => $'a$ => *nat* => $'a$
where
  nscal-0: $nscal\ R\ x\ 0 = \mathbf{0}_R$

| *nscal-suc*: $nscal\ R\ x\ (Suc\ n) = (nscal\ R\ x\ n) \pm_R x$

primrec *npow* :: $('a, 'more)\ Ring\text{-}scheme \Rightarrow 'a \Rightarrow nat \Rightarrow 'a$
where
npow-0: $npow\ R\ x\ 0 = 1_{r\,R}$
| *npow-suc*: $npow\ R\ x\ (Suc\ n) = (npow\ R\ x\ n) \cdot_{r\,R} x$

primrec *nprod* :: $('a, 'more)\ Ring\text{-}scheme \Rightarrow (nat \Rightarrow 'a) \Rightarrow nat \Rightarrow 'a$
where
nprod-0: $nprod\ R\ f\ 0 = f\ 0$
| *nprod-suc*: $nprod\ R\ f\ (Suc\ n) = (nprod\ R\ f\ n) \cdot_{r\,R} (f\ (Suc\ n))$

primrec *nsum* :: $('a, 'more)\ aGroup\text{-}scheme \Rightarrow (nat \Rightarrow 'a) \Rightarrow nat \Rightarrow 'a$
where
nsum-0: $nsum\ R\ f\ 0 = f\ 0$
| *nsum-suc*: $nsum\ R\ f\ (Suc\ n) = (nsum\ R\ f\ n) \pm_R (f\ (Suc\ n))$

abbreviation
$NSCAL :: [nat, ('a, 'more)\ Ring\text{-}scheme, 'a] \Rightarrow 'a$
$((3\ -\ \times_-\ -)\ [75,75,76]75)$ **where**
$n \times_R x == nscal\ R\ x\ n$

abbreviation
$NPOW :: ['a, ('a, 'more)\ Ring\text{-}scheme, nat] \Rightarrow 'a$
$((3-\hat{}_-\ -)\ [77,77,78]77)$ **where**
$a\hat{}^R\ n == npow\ R\ a\ n$

abbreviation
$SUM :: ('a, 'more)\ aGroup\text{-}scheme \Rightarrow (nat \Rightarrow 'a) \Rightarrow nat \Rightarrow 'a$
$((3\Sigma_e\ -\ -\ -)\ [85,85,86]85)$ **where**
$\Sigma_e\ G\ f\ n == nsum\ G\ f\ n$

abbreviation
$NPROD :: [('a, 'm)\ Ring\text{-}scheme, nat, nat \Rightarrow 'a] \Rightarrow 'a$
$((3e\Pi_{-,-}\ -)\ [98,98,99]98)$ **where**
$e\Pi_{R,n}\ f == nprod\ R\ f\ n$

definition
$fSum :: [-, (nat \Rightarrow 'a), nat, nat] \Rightarrow 'a$ **where**
$fSum\ A\ f\ n\ m = (if\ n \leq m\ then\ nsum\ A\ (cmp\ f\ (slide\ n))(m - n)$
$else\ \mathbf{0}_A)$

abbreviation
$FSUM :: [('a, 'more)\ aGroup\text{-}scheme, (nat \Rightarrow 'a), nat, nat] \Rightarrow 'a$
$((4\Sigma_f\ -\ -\ -\ -)\ [85,85,85,86]85)$ **where**
$\Sigma_f\ G\ f\ n\ m == fSum\ G\ f\ n\ m$

lemma (**in** *aGroup*) *nsum-zeroGTr*:$(\forall j \leq n.\ f\ j = \mathbf{0}) \longrightarrow nsum\ A\ f\ n = \mathbf{0}$
apply (*induct-tac n*)

apply (*rule impI*, *simp*)

apply (*rule impI*)
apply (*cut-tac* $n = n$ **in** *Nsetn-sub-mem1*, *simp*)
apply (*cut-tac ex-zero*)
apply (*simp add*:*l-zero*[*of* **0**])
done

lemma (**in** *aGroup*) *nsum-zeroA*:$\forall j \leq n.\ f\ j = \mathbf{0} \Longrightarrow \quad nsum\ A\ f\ n = \mathbf{0}$
apply (*simp add*:*nsum-zeroGTr*)
done

definition
sr :: [- , $'a$ *set*] ⇒ *bool* **where**
sr R S == $S \subseteq carrier\ R \wedge 1_{r}R \in S \wedge (\forall x{\in}S.\ \forall y \in S.\ x\ \pm_{R}\ (-_{a}R\ y) \in S \wedge$
$x\ \cdot_{r}R\ y \in S)$

definition
Sr :: [- , $'a$ *set*] ⇒ - **where**
Sr R S = *R* (|*carrier* := *S*, *pop* := $\lambda x{\in}S.\ \lambda y{\in}S.\ x\ \pm_{R}\ y$, *mop* := $\lambda x{\in}S.\ (-_{a}R\ x)$,
zero := $\mathbf{0}_{R}$, *tp* := $\lambda x{\in}S.\ \lambda y{\in}S.\ x\ \cdot_{r}R\ y$, *un* := $1_{r}R$ |)

lemma (**in** *Ring*) *Ring*: *Ring R* **..**

lemma (**in** *Ring*) *ring-is-ag*:*aGroup R*
apply (*rule aGroup.intro*,
rule pop-closed,
rule pop-aassoc, *assumption*+,
rule pop-commute, *assumption*+,
rule mop-closed,
rule l-m, *assumption*,
rule ex-zero,
rule l-zero, *assumption*)
done

lemma (**in** *Ring*) *ring-zero*:$\mathbf{0} \in carrier\ R$
by (*simp add*: *ex-zero*)

lemma (**in** *Ring*) *ring-one*:$1_{r} \in carrier\ R$
by (*simp add*:*un-closed*)

lemma (**in** *Ring*) *ring-tOp-closed*:$[\![\ x \in carrier\ R;\ y \in carrier\ R]\!] \Longrightarrow$
$x\ \cdot_{r}\ y \in carrier\ R$
apply (*cut-tac tp-closed*)
apply (*frule funcset-mem*[*of op* $\cdot_{r}$ *carrier R carrier R* → *carrier R*

$x]$, *assumption*+,
thin-tac op $\cdot_r \in$ *carrier* $R \to$ *carrier* $R \to$ *carrier* R)
apply (*rule funcset-mem*[*of op* $\cdot_r$ x *carrier* R *carrier* R y],
assumption+)
done

lemma (**in** *Ring*) *ring-tOp-commute*:$[\![x \in$ *carrier* R; $y \in$ *carrier* $R]\!] \Longrightarrow$
$x \cdot_r y = y \cdot_r x$
by (*simp add*:*tp-commute*)

lemma (**in** *Ring*) *ring-distrib1*:$[\![x \in$ *carrier* R; $y \in$ *carrier* R; $z \in$ *carrier* R $]\!]$
$\Longrightarrow x \cdot_r (y \pm z) = x \cdot_r y \pm x \cdot_r z$
by (*simp add*:*rg-distrib*)

lemma (**in** *Ring*) *ring-distrib2*:$[\![x \in$ *carrier* R; $y \in$ *carrier* R; $z \in$ *carrier* R $]\!]$
$\Longrightarrow (y \pm z) \cdot_r x = y \cdot_r x \pm \ z \cdot_r x$
apply (*subst tp-commute*[*of* $y \pm z$ x])
apply (*cut-tac ring-is-ag*, *simp add*:*aGroup.ag-pOp-closed*)
apply *assumption*
apply (*subst ring-distrib1*, *assumption*+)
apply (*simp add*:*tp-commute*)
done

lemma (**in** *Ring*) *ring-distrib3*:$[\![a \in$ *carrier* R; $b \in$ *carrier* R; $x \in$ *carrier* R;
$y \in$ *carrier* R $]\!] \Longrightarrow (a \pm b) \cdot_r (x \pm y) =$
$a \cdot_r x \pm a \cdot_r y \pm b \cdot_r x \pm b \cdot_r y$
apply (*subst ring-distrib2*)+
apply (*cut-tac ring-is-ag*)
apply (*rule aGroup.ag-pOp-closed*, *assumption*+)
apply ((*subst ring-distrib1*)+, *assumption*+)
apply (*subst ring-distrib1*, *assumption*+)
apply (*rule pop-aassoc* [*THEN sym*, *of* $a \cdot_r x \pm a \cdot_r y$ $b \cdot_r x$ $b \cdot_r y$])
apply (*cut-tac ring-is-ag*, *rule aGroup.ag-pOp-closed*, *assumption*)
apply (*simp add*:*ring-tOp-closed*)+
done

lemma (**in** *Ring*) *rEQMulR*:
$[\![x \in$ *carrier* R; $y \in$ *carrier* R; $z \in$ *carrier* R; $x = y$ $]\!]$
$\Longrightarrow x \cdot_r z = y \cdot_r z$
by *simp*

lemma (**in** *Ring*) *ring-tOp-assoc*:$[\![x \in$ *carrier* R; $y \in$ *carrier* R; $z \in$ *carrier* R $]\!]$
$\Longrightarrow (x \cdot_r y) \cdot_r z = x \cdot_r (y \cdot_r z)$
by (*simp add*:*tp-assoc*)

lemma (**in** *Ring*) *ring-l-one*:$x \in$ *carrier* $R \Longrightarrow 1_r \cdot_r x = x$
by (*simp add*:*rg-l-unit*)

lemma (**in** *Ring*) *ring-r-one*:$x \in$ *carrier* $R \ \Longrightarrow x \cdot_r 1_r = x$

apply (*subst ring-tOp-commute*, *assumption+*)
apply (*simp add*:*un-closed*)
apply (*simp add*:*ring-l-one*)
done

lemma (**in** *Ring*) *ring-times-0-x*:$x \in$ *carrier* $R \Longrightarrow \mathbf{0} \cdot_r x = \mathbf{0}$
apply (*cut-tac ring-is-ag*)
apply (*cut-tac ring-zero*)
apply (*frule ring-distrib2* [*of* x $\mathbf{0}$ $\mathbf{0}$], *assumption+*)
apply (*simp add*:*aGroup.ag-l-zero* [*of* R $\mathbf{0}$])
apply (*frule ring-tOp-closed* [*of* $\mathbf{0}$ x], *assumption+*)
apply (*frule sym*, *thin-tac* $\mathbf{0} \cdot_r x = \mathbf{0} \cdot_r x \pm \mathbf{0} \cdot_r x$)
apply (*frule aGroup.ag-eq-sol2* [*of* R $\mathbf{0} \cdot_r x$ $\mathbf{0} \cdot_r x$ $\mathbf{0} \cdot_r x$],
assumption+)
apply (*thin-tac* $\mathbf{0} \cdot_r x \pm \mathbf{0} \cdot_r x = \mathbf{0} \cdot_r x$)
apply (*simp add*:*aGroup.ag-r-inv1*)
done

lemma (**in** *Ring*) *ring-times-x-0*:$x \in$ *carrier* $R \Longrightarrow x \cdot_r \mathbf{0} = \mathbf{0}$
apply (*cut-tac ring-zero*)
apply (*subst ring-tOp-commute*, *assumption+*, *simp add*:*ring-zero*)
apply (*simp add*:*ring-times-0-x*)
done

lemma (**in** *Ring*) *rMulZeroDiv*:
$[\![x \in$ *carrier* R; $y \in$ *carrier* R; $x = \mathbf{0} \vee y = \mathbf{0}]\!] \Longrightarrow x \cdot_r y = \mathbf{0}$
apply (*erule disjE*, *simp*)
apply (*rule ring-times-0-x*, *assumption+*)
apply (*simp*, *rule ring-times-x-0*, *assumption+*)
done

lemma (**in** *Ring*) *ring-inv1*:$[\![a \in$ *carrier* R; $b \in$ *carrier* $R]\!] \Longrightarrow$
$-_a (a \cdot_r b) = (-_a a) \cdot_r b \wedge -_a (a \cdot_r b) = a \cdot_r (-_a b)$
apply (*cut-tac ring-is-ag*)
apply (*rule conjI*)
apply (*frule ring-distrib2* [*THEN sym*, *of* b a $-_a a$], *assumption+*)
apply (*frule aGroup.ag-mOp-closed* [*of* R a], *assumption+*)
apply (*simp add*:*aGroup.ag-r-inv1* [*of* R a])
apply (*simp add*:*ring-times-0-x*)
apply (*frule aGroup.ag-mOp-closed* [*of* R a], *assumption+*)
apply (*frule ring-tOp-closed* [*of* a b], *assumption+*)
apply (*frule ring-tOp-closed* [*of* $-_a a$ b], *assumption+*)
apply (*frule aGroup.ag-eq-sol1* [*of* R $a \cdot_r b$ $(-_a a) \cdot_r b$ $\mathbf{0}$],
assumption+)
apply (*rule ring-zero*, *assumption+*)
apply (*thin-tac* $a \cdot_r b \pm (-_a a) \cdot_r b = \mathbf{0}$)
apply (*frule sym*) **apply** (*thin-tac* $(-_a a) \cdot_r b = -_a (a \cdot_r b) \pm \mathbf{0}$)
apply (*frule aGroup.ag-mOp-closed* [*of* R $a \cdot_r b$], *assumption+*)
apply (*simp add*:*aGroup.ag-r-zero*)

apply (*frule ring-distrib1* [*THEN sym*, *of a b* $-_a$ *b*], *assumption*+)
apply (*simp add*:*aGroup.ag-mOp-closed*)
apply (*simp add*:*aGroup.ag-r-inv1* [*of R b*])
apply (*simp add*:*ring-times-x-0*)
apply (*frule aGroup.ag-mOp-closed* [*of R b*], *assumption*+)
apply (*frule ring-tOp-closed* [*of a b*], *assumption*+)
apply (*frule ring-tOp-closed* [*of a* $-_a$ *b*], *assumption*+)
apply (*frule aGroup.ag-eq-sol1* [*THEN sym*, *of R a* $\cdot_r$ *b a* $\cdot_r$ ($-_a$ *b*) **0**], *assumption*+)
apply (*simp add*:*ring-zero*) **apply** *assumption*
apply (*frule aGroup.ag-mOp-closed* [*of R a* $\cdot_r$ *b*], *assumption*+)
apply (*simp add*:*aGroup.ag-r-zero*)
done

lemma (**in** *Ring*) *ring-inv1-1*:⟦*a* ∈ *carrier R*; *b* ∈ *carrier R* ⟧ ⟹
$-_a$ (*a* $\cdot_r$ *b*) = ($-_a$ *a*) $\cdot_r$ *b*
apply (*simp add*:*ring-inv1*)
done

lemma (**in** *Ring*) *ring-inv1-2*:⟦ *a* ∈ *carrier R*; *b* ∈ *carrier R* ⟧ ⟹
$-_a$ (*a* $\cdot_r$ *b*) = *a* $\cdot_r$ ($-_a$ *b*)
apply (*frule ring-inv1* [*of a b*], *assumption*+)
apply (*frule conjunct2*)
apply (*thin-tac* $-_a$ *a* $\cdot_r$ *b* = ($-_a$ *a*) $\cdot_r$ *b* ∧ $-_a$ (*a* $\cdot_r$ *b*) = *a* $\cdot_r$ ($-_a$ *b*))
apply *simp*
done

lemma (**in** *Ring*) *ring-times-minusl*:*a* ∈ *carrier R* ⟹ $-_a$ *a* = ($-_a$ 1_r) $\cdot_r$ *a*
apply (*cut-tac ring-one*)
apply (*frule ring-inv1-1*[*of* 1_r *a*], *assumption*+)
apply (*simp add*:*ring-l-one*)
done

lemma (**in** *Ring*) *ring-times-minusr*:*a* ∈ *carrier R* ⟹ $-_a$ *a* = *a* $\cdot_r$ ($-_a$ 1_r)
apply (*cut-tac ring-one*)
apply (*frule ring-inv1-2*[*of a* 1_r], *assumption*+)
apply (*simp add*:*ring-r-one*)
done

lemma (**in** *Ring*) *ring-inv1-3*:⟦*a* ∈ *carrier R*; *b* ∈ *carrier R*⟧ ⟹
a $\cdot_r$ *b* = ($-_a$ *a*) $\cdot_r$ ($-_a$ *b*)
apply (*cut-tac ring-is-ag*)
apply (*subst aGroup.ag-inv-inv*[*THEN sym*], *assumption*+)
apply (*frule aGroup.ag-mOp-closed*[*of R a*], *assumption*+)
apply (*subst ring-inv1-1*[*THEN sym*, *of* $-_a$ *a b*], *assumption*+)
apply (*subst ring-inv1-2*[*of* $-_a$ *a b*], *assumption*+, *simp*)
done

lemma (**in** *Ring*) *ring-distrib4*:⟦*a* ∈ *carrier R*; *b* ∈ *carrier R*;

$x \in carrier\ R;\ y \in carrier\ R\ ⟧ \Longrightarrow$
$a \cdot_r b \pm (-_a\ x \cdot_r y) = a \cdot_r (b \pm (-_a\ y)) \pm (a \pm (-_a\ x)) \cdot_r y$
apply (*cut-tac ring-is-ag*)
apply (*subst ring-distrib1*, *assumption+*)
apply (*rule aGroup.ag-mOp-closed*, *assumption+*)
apply (*subst ring-distrib2*, *assumption+*)
apply (*rule aGroup.ag-mOp-closed*, *assumption+*)
apply (*subst aGroup.pOp-assocTr43*, *assumption+*)
apply (*rule ring-tOp-closed*, *assumption+*)+
apply (*rule aGroup.ag-mOp-closed*, *assumption+*)
apply (*rule ring-tOp-closed*, *assumption+*)
apply (*rule ring-tOp-closed*)
apply (*simp add*:*aGroup.ag-mOp-closed*)+
apply (*subst ring-distrib1* [*THEN sym*, *of a* -], *assumption+*)
apply (*rule aGroup.ag-mOp-closed*, *assumption+*)
apply (*simp add*:*aGroup.ag-l-inv1*)
apply (*simp add*:*ring-times-x-0*)
apply (*subst aGroup.ag-r-zero*, *assumption+*)
apply (*simp add*:*ring-tOp-closed*)
apply (*simp add*: *ring-inv1-1*)
done

lemma (**in** *Ring*) *rMulLC*:
$⟦x \in carrier\ R;\ y \in carrier\ R;\ z \in carrier\ R⟧$
$\Longrightarrow x \cdot_r (y \cdot_r z) = y \cdot_r (x \cdot_r z)$
apply (*subst ring-tOp-assoc* [*THEN sym*], *assumption+*)
apply (*subst ring-tOp-commute* [*of x y*], *assumption+*)
apply (*subst ring-tOp-assoc*, *assumption+*)
apply *simp*
done

lemma (**in** *Ring*) *Zero-ring*:$1_r = \mathbf{0} \Longrightarrow$ *zeroring R*
apply (*simp add*:*zeroring-def*)
apply (*rule conjI*)
apply (*rule Ring-axioms*)
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*frule-tac x* = *x* **in** *ring-r-one*, *simp add*:*ring-times-x-0*)

apply (*simp add*:*ring-zero*)
done

lemma (**in** *Ring*) *Zero-ring1*:$\neg$ (*zeroring R*) $\Longrightarrow$ $1_r \neq \mathbf{0}$
apply (*rule contrapos-pp*, *simp+*,
cut-tac Zero-ring, *simp+*)
done

lemma (**in** *Ring*) *Sr-one*:*sr R S* $\Longrightarrow 1_r \in S$
apply (*simp add*:*sr-def*)

done

lemma (**in** *Ring*) *Sr-zero*:*sr R S* $\Longrightarrow$ $\mathbf{0} \in S$
apply (*cut-tac ring-is-ag*, *frule Sr-one*[*of S*])
apply (*simp add*:*sr-def*) **apply** (*erule conjE*)+
apply (*frule-tac* $x = 1_r$ **in** *bspec*, *assumption*,
  thin-tac $\forall x{\in}S.\ \forall y{\in}S.\ x \pm -_a\ y \in S \wedge x \cdot_r y \in S$,
  frule-tac $x = 1_r$ **in** *bspec*, *assumption*,
  thin-tac $\forall y{\in}S.\ 1_r \pm -_a\ y \in S \wedge 1_r \cdot_r y \in S$,
  erule conjE)
apply (*cut-tac ring-one*,
  simp add:*aGroup.ag-r-inv1*[*of R* 1_r])
done

lemma (**in** *Ring*) *Sr-mOp-closed*:$[\![$*sr R S*; $x \in S]\!] \Longrightarrow -_a\ x \in S$
apply (*frule Sr-zero*[*of S*])
apply (*simp add*:*sr-def*, (*erule conjE*)+)
apply (*cut-tac ring-is-ag*)
apply (*frule-tac* $x = \mathbf{0}$ **in** *bspec*, *assumption*,
  thin-tac $\forall x{\in}S.\ \forall y{\in}S.\ x \pm -_a\ y \in S \wedge x \cdot_r y \in S$,
  frule-tac $x = x$ **in** *bspec*, *assumption*,
  thin-tac $\forall y{\in}S.\ \mathbf{0} \pm -_a\ y \in S \wedge \mathbf{0} \cdot_r y \in S$, *erule conjE*)
apply (*frule subsetD*[*of S carrier R* $\mathbf{0}$], *assumption*+,
  frule subsetD[*of S carrier R x*], *assumption*+)
apply (*frule aGroup.ag-mOp-closed* [*of R x*], *assumption*)
apply (*simp add*:*aGroup.ag-l-zero*)
done

lemma (**in** *Ring*) *Sr-pOp-closed*:$[\![$*sr R S*; $x \in S$; $y \in S]\!] \Longrightarrow x \pm y \in S$
apply (*frule Sr-mOp-closed*[*of S y*], *assumption*+)
apply (*unfold sr-def*, (*erule conjE*)+)
apply (*frule-tac* $x = x$ **in** *bspec*, *assumption*,
  thin-tac $\forall x{\in}S.\ \forall y{\in}S.\ x \pm -_a\ y \in S \wedge x \cdot_r y \in S$,
  frule-tac $x = -_a\ y$ **in** *bspec*, *assumption*,
  thin-tac $\forall y{\in}S.\ x \pm -_a\ y \in S \wedge x \cdot_r y \in S$, *erule conjE*)

apply (*cut-tac ring-is-ag*)
apply (*frule subsetD*[*of S carrier R y*], *assumption*+)
apply (*simp add*:*aGroup.ag-inv-inv*)
done

lemma (**in** *Ring*) *Sr-tOp-closed*:$[\![$*sr R S*; $x \in S$; $y \in S]\!] \Longrightarrow x \cdot_r y \in S$
by (*simp add*:*sr-def*)

lemma (**in** *Ring*) *Sr-ring*:*sr R S* $\Longrightarrow$ *Ring* (*Sr R S*)
apply (*simp add*:*Ring-def* [*of Sr R S*],
  cut-tac ring-is-ag)
apply (*rule conjI*)
apply (*simp add*:*Sr-def Sr-pOp-closed*)

apply (*rule conjI*)
 apply (*rule allI*, *rule impI*)+
 apply (*simp add*:*Sr-def*,
       frule-tac x = *a* **and** *y* = *b* **in** *Sr-pOp-closed*, *assumption*+,
       frule-tac x = *b* **and** *y* = *c* **in** *Sr-pOp-closed*, *assumption*+,
       simp add:*Sr-def sr-def*, (*erule conjE*)+)
 apply (*frule-tac c* = *a* **in** *subsetD*[*of S carrier R*], *assumption*+,
       frule-tac c = *b* **in** *subsetD*[*of S carrier R*], *assumption*+,
       frule-tac c = *c* **in** *subsetD*[*of S carrier R*], *assumption*+)
 apply (*simp add*:*aGroup.ag-pOp-assoc*)

apply (*rule conjI*)
 apply (*rule allI*, *rule impI*)+
 apply (*simp add*:*Sr-def sr-def*, (*erule conjE*)+,
       frule-tac c = *a* **in** *subsetD*[*of S carrier R*], *assumption*+,
       frule-tac c = *b* **in** *subsetD*[*of S carrier R*], *assumption*+)
 apply (*simp add*:*aGroup.ag-pOp-commute*)

apply (*rule conjI*)
  apply ((*subst Sr-def*)+, *simp*)
  apply (*simp add*:*Sr-mOp-closed*)

apply (*rule conjI*)
  apply (*rule allI*)
  apply ((*subst Sr-def*)+, *simp add*:*Sr-mOp-closed*, *rule impI*)
  apply (*unfold sr-def*, *frule conjunct1*, *fold sr-def*,
       frule-tac c = *a* **in** *subsetD*[*of S carrier R*], *assumption*+,
       simp add:*aGroup.ag-l-inv1*)

apply (*rule conjI*)
  apply (*simp add*:*Sr-def Sr-zero*)

apply (*rule conjI*)
  apply (*rule allI*, *simp add*:*Sr-def Sr-zero*)
  apply (*rule impI*)
  apply (*unfold sr-def*, *frule conjunct1*, *fold sr-def*,
       frule-tac c = *a* **in** *subsetD*[*of S carrier R*], *assumption*+,
       simp add:*aGroup.ag-l-zero*)

apply (*rule conjI*)
  apply (*simp add*:*Sr-def Sr-tOp-closed*)

apply (*rule conjI*)
  apply (*rule allI*, *rule impI*)+
  apply (*simp add*:*Sr-def Sr-tOp-closed*)
  apply (*unfold sr-def*, *frule conjunct1*, *fold sr-def*,
       frule-tac c = *a* **in** *subsetD*[*of S carrier R*], *assumption*+,
       frule-tac c = *b* **in** *subsetD*[*of S carrier R*], *assumption*+,

frule-tac $c = c$ **in** *subsetD*[*of S carrier R*], *assumption*+)
apply (*simp add*:*ring-tOp-assoc*)

apply (*rule conjI*)
apply ((*rule allI*, *rule impI*)+, *simp add*:*Sr-def*)
apply (*unfold sr-def*, *frule conjunct1*, *fold sr-def*,
frule-tac $c = a$ **in** *subsetD*[*of S carrier R*], *assumption*+,
frule-tac $c = b$ **in** *subsetD*[*of S carrier R*], *assumption*+,
simp add:*ring-tOp-commute*)

apply (*rule conjI*)
apply (*simp add*:*Sr-def Sr-one*)

apply (*rule conjI*)
apply (*simp add*:*Sr-def Sr-pOp-closed Sr-tOp-closed*)
apply (*rule allI*, *rule impI*)+
apply (*unfold sr-def*, *frule conjunct1*, *fold sr-def*,
frule-tac $c = a$ **in** *subsetD*[*of S carrier R*], *assumption*+,
frule-tac $c = b$ **in** *subsetD*[*of S carrier R*], *assumption*+,
frule-tac $c = c$ **in** *subsetD*[*of S carrier R*], *assumption*+)
apply (*simp add*:*ring-distrib1*)

apply (*simp add*:*Sr-def Sr-one*)
apply (*rule allI*, *rule impI*)
apply (*unfold sr-def*, *frule conjunct1*, *fold sr-def*,
frule-tac $c = a$ **in** *subsetD*[*of S carrier R*], *assumption*+)
apply (*simp add*:*ring-l-one*)
done

4.2 Calculation of elements

4.2.1 nscale

lemma (**in** *Ring*) *ring-tOp-rel*:⟦$x \in$*carrier R*; $xa \in$*carrier R*; $y \in$*carrier R*; $ya \in$ *carrier R* ⟧ ⟹ $(x \cdot_r xa) \cdot_r (y \cdot_r ya) = (x \cdot_r y) \cdot_r (xa \cdot_r ya)$
apply (*frule ring-tOp-closed*[*of y ya*], *assumption*+,
simp add:*ring-tOp-assoc*[*of x xa*])
apply (*simp add*:*ring-tOp-assoc*[*THEN sym*, *of xa y ya*],
simp add:*ring-tOp-commute*[*of xa y*],
simp add:*ring-tOp-assoc*[*of y xa ya*])
apply (*frule ring-tOp-closed*[*of xa ya*], *assumption*+,
simp add:*ring-tOp-assoc*[*THEN sym*, *of x y*])
done

lemma (**in** *Ring*) *nsClose*:
⋀ *n*. ⟦ $x \in$ *carrier R* ⟧ ⟹ *nscal R x n* $\in$ *carrier R*
apply (*induct-tac n*)
apply (*simp add*:*ring-zero*)
apply (*cut-tac ring-is-ag*, *simp add*:*aGroup.ag-pOp-closed*)

done

lemma (**in** *Ring*) *nsZero*:
  nscal R **0** *n* = **0**
 apply (*cut-tac ring-is-ag*)
 apply (*induct-tac n*)
 apply *simp*

 apply *simp*
  apply (*cut-tac ring-zero*, *simp add*:*aGroup.ag-l-zero*)
 done

lemma (**in** *Ring*) *nsZeroI*: ⋀ *n*. *x* = **0** ⟹ *nscal R x n* = **0**
 by (*simp only*:*nsZero*)

lemma (**in** *Ring*) *nsEqElm*: ⟦ *x* ∈ *carrier R*; *y* ∈ *carrier R*; *x* = *y* ⟧
    ⟹ (*nscal R x n*) = (*nscal R y n*)
 by *simp*

lemma (**in** *Ring*) *nsDistr*: *x* ∈ *carrier R*
    ⟹ (*nscal R x n*) ± (*nscal R x m*) = *nscal R x* (*n* + *m*)
apply (*cut-tac ring-is-ag*)
 apply (*induct-tac m*)
 apply *simp*
 apply (*frule nsClose*[*of x n*])
 apply (*simp add*:*aGroup.ag-r-zero*)

 apply *simp*
 apply (*frule-tac x* = *x* **and** *n* = *n* **in** *nsClose*,
     frule-tac x = *x* **and** *n* = *na* **in** *nsClose*)
 apply (*subst aGroup.ag-pOp-assoc*[*THEN sym*], *assumption*+, *simp*)
 done

lemma (**in** *Ring*) *nsDistrL*: ⟦*x* ∈ *carrier R*; *y* ∈ *carrier R* ⟧
    ⟹ (*nscal R x n*) ± (*nscal R y n*) = *nscal R* (*x* ± *y*) *n*
 apply (*cut-tac ring-is-ag*)
 apply (*induct-tac n*)
 apply *simp*
 apply (*cut-tac ring-zero*,
     simp add:*aGroup.ag-l-zero*)

 apply *simp*
 apply (*frule-tac x* = *x* **and** *n* = *n* **in** *nsClose*,
     frule-tac x = *y* **and** *n* = *n* **in** *nsClose*)
 apply (*subst aGroup.pOp-assocTr43*[*of R - x - y*], *assumption*+)
 apply (*frule-tac x* = *x* **and** *y* = *n* ×$_R$ *y* **in** *aGroup.ag-pOp-commute*[*of R*],
     assumption+)
  apply *simp*
  apply (*subst aGroup.pOp-assocTr43*[*THEN sym*, *of R - - x y*], *assumption*+)

apply *simp*
done

lemma (**in** *Ring*) *nsMulDistrL*:⟦ $x \in carrier\ R$; $y \in carrier\ R$ ⟧
$\Longrightarrow x \cdot_r (nscal\ R\ y\ n) = nscal\ R\ (x \cdot_r y)\ n$
apply (*induct-tac n*)
apply *simp*
apply (*simp add*:*ring-times-x-0*)

apply *simp* **apply** (*subst ring-distrib1*, *assumption*+)
apply (*rule nsClose*, *assumption*+)
apply *simp*
done

lemma (**in** *Ring*) *nsMulDistrR*:⟦ $x \in carrier\ R$; $y \in carrier\ R$⟧
$\Longrightarrow (nscal\ R\ y\ n) \cdot_r x = nscal\ R\ (y \cdot_r x)\ n$
apply (*frule-tac* $x = y$ **and** $n = n$ **in** *nsClose*,
simp add:*ring-tOp-commute*[*of* $n \times_R y\ x$],
simp add:*nsMulDistrL*,
simp add:*ring-tOp-commute*[*of* $y\ x$])
done

4.2.2 npow

lemma (**in** *Ring*) *npClose*:$x \in carrier\ R \Longrightarrow npow\ R\ x\ n \in carrier\ R$
apply (*induct-tac n*)
apply *simp* **apply** (*simp add*:*ring-one*)

apply *simp*
apply (*rule ring-tOp-closed*, *assumption*+)
done

lemma (**in** *Ring*) *npMulDistr*:$\bigwedge n\ m.\ x \in carrier\ R \Longrightarrow$
$(npow\ R\ x\ n) \cdot_r (npow\ R\ x\ m) = npow\ R\ x\ (n + m)$
apply (*induct-tac m*)
apply *simp* **apply** (*rule ring-r-one*, *simp add*:*npClose*)

apply *simp*
apply (*frule-tac* $x = x$ **and** $n = n$ **in** *npClose*,
frule-tac $x = x$ **and** $n = na$ **in** *npClose*)
apply (*simp add*:*ring-tOp-assoc*[*THEN sym*])
done

lemma (**in** *Ring*) *npMulExp*:$\bigwedge n\ m.\ x \in carrier\ R$
$\Longrightarrow\ npow\ R\ (npow\ R\ x\ n)\ m = npow\ R\ x\ (n * m)$
apply (*induct-tac m*)
apply *simp*
apply *simp*
apply (*simp add*:*npMulDistr*)

apply (*simp add*:*add.commute*)
done

lemma (**in** *Ring*) *npGTPowZero-sub*:
  $\bigwedge$ *n*. ⟦ *x* ∈ *carrier R*; *npow R x m* = **0** ⟧
    ⟹(*m* ≤ *n*) ⟶ (*npow R x n* = **0**)
 apply (*rule impI*)
 apply (*subgoal-tac npow R x n* = (*npow R x* (*n*−*m*)) ·$_r$ (*npow R x m*))
 apply *simp*
 apply (*rule ring-times-x-0*) **apply** (*simp add*:*npClose*)
 apply (*thin-tac x*^*R m* = **0**)
 apply (*subst npMulDistr*, *assumption*)
 apply *simp*
 done

lemma (**in** *Ring*) *npGTPowZero*:
 $\bigwedge$ *n*. ⟦ *x* ∈ *carrier R*; *npow R x m* = **0**; *m* ≤ *n* ⟧
    ⟹ *npow R x n* = **0**
 apply (*cut-tac x* = *x* **and** *m* = *m* **and** *n* = *n* **in** *npGTPowZero-sub*, *assumption*+)
 apply *simp*
 done

lemma (**in** *Ring*) *npOne*: *npow R* (1$_r$) *n* = 1$_r$
 apply (*induct-tac n*) **apply** *simp*

 apply *simp*
  apply (*rule ring-r-one*, *simp add*:*ring-one*)
done

lemma (**in** *Ring*) *npZero-sub*: *0* < *n* ⟶ *npow R* **0** *n* = **0**
 apply (*induct-tac n*)
 apply *simp*

 apply *simp*
  apply (*cut-tac ring-zero*,
      frule-tac n = *n* **in** *npClose*[*of* **0**])
  apply (*simp add*:*ring-times-x-0*)
done

lemma (**in** *Ring*) *npZero*: *0* < *n* ⟹ *npow R* **0** *n* = **0**
 apply (*simp add*:*npZero-sub*)
done

lemma (**in** *Ring*) *npMulElmL*: $\bigwedge$ *n*. ⟦ *x* ∈ *carrier R*; *0* ≤ *n*⟧
    ⟹ *x* ·$_r$ (*npow R x n*) = *npow R x* (*Suc n*)
apply (*simp only*:*npow-suc*,

frule-tac $n = n$ **and** $x = x$ **in** *npClose*,
simp add:*ring-tOp-commute*)
done

lemma (**in** *Ring*) *npMulEleL*: $\bigwedge n.\ x \in carrier\ R \Longrightarrow (npow\ R\ x\ n) \cdot_r x = npow\ R\ x\ (Suc\ n)$
by (*simp add*:*npMulElmL*[*THEN sym*])

lemma (**in** *Ring*) *npMulElmR*: $\bigwedge n.\ x \in carrier\ R \Longrightarrow (npow\ R\ x\ n) \cdot_r x = npow\ R\ x\ (Suc\ n)$
apply (*frule-tac* $n = n$ **in** *npClose*[*of x*])
apply (*simp only*:*ring-tOp-commute*,
subst npMulElmL, *assumption*, *simp*, *simp*)
done

lemma (**in** *Ring*) *np-1*:$a \in carrier\ R \Longrightarrow npow\ R\ a\ (Suc\ 0) = a$
apply *simp*
apply (*simp add*:*ring-l-one*)
done

4.2.3 nsum and fSum

lemma (**in** *aGroup*) *nsum-memTr*: $(\forall j \le n.\ f\ j \in carrier\ A) \longrightarrow nsum\ A\ f\ n \in carrier\ A$
apply (*induct-tac n*)
apply *simp*
apply (*rule impI*)
apply (*cut-tac* $n = n$ **in** *Nsetn-sub-mem1*, *simp*)
apply (*frule-tac* $a = Suc\ n$ **in** *forall-spec*, *simp*,
thin-tac $\forall j \le Suc\ n.\ f\ j \in carrier\ A$)
apply (*rule ag-pOp-closed*, *assumption*+)
done

lemma (**in** *aGroup*) *nsum-mem*:$\forall j \le n.\ f\ j \in carrier\ A \Longrightarrow nsum\ A\ f\ n \in carrier\ A$
apply (*simp add*:*nsum-memTr*)
done

lemma (**in** *aGroup*) *nsum-eqTr*:$(\forall j \le n.\ f\ j \in carrier\ A \wedge g\ j \in carrier\ A \wedge f\ j = g\ j) \longrightarrow nsum\ A\ f\ n = nsum\ A\ g\ n$
apply (*induct-tac n*)
apply *simp*
apply (*rule impI*)
apply (*cut-tac* $n = n$ **in** *Nsetn-sub-mem1*, *simp*)
done

lemma (**in** *aGroup*) *nsum-eq*:$[\![\forall j \le n.\ f\ j \in carrier\ A;\ \forall j \le n.\ g\ j \in carrier\ A;$

$\forall j \leq n.\ f\ j = g\ j\rrbracket \Longrightarrow\ nsum\ A\ f\ n = nsum\ A\ g\ n$
by (*simp add*:*nsum-eqTr*)

lemma (**in** *aGroup*) *nsum-cmp-assoc*:⟦$\forall j \leq n.\ f\ j \in carrier\ A$;
$g \in \{j.\ j \leq n\} \to \{j.\ j \leq n\}$; $h \in \{j.\ j \leq n\} \to \{j.\ j \leq n\}$⟧ ⟹
nsum A (*cmp* (*cmp f h*) *g*) *n* = *nsum A* (*cmp f* (*cmp h g*)) *n*
apply (*rule nsum-eq*)
apply (*rule allI*, *rule impI*, *simp add*:*cmp-def*)
apply (*frule-tac x* = *j* **in** *funcset-mem*[*of g* {*j*. *j* ≤ *n*} {*j*. *j* ≤ *n*}], *simp*,
frule-tac x = *g j* **in** *funcset-mem*[*of h* {*j*. *j* ≤ *n*} {*j*. *j* ≤ *n*}],
assumption, *simp*)
apply (*rule allI*, *rule impI*, *simp add*:*cmp-def*,
frule-tac x = *j* **in** *funcset-mem*[*of g* {*j*. *j* ≤ *n*} {*j*. *j* ≤ *n*}], *simp*,
frule-tac x = *g j* **in** *funcset-mem*[*of h* {*j*. *j* ≤ *n*} {*j*. *j* ≤ *n*}],
assumption, *simp*)
apply (*rule allI*, *simp add*:*cmp-def*)
done

lemma (**in** *aGroup*) *fSum-Suc*:$\forall j \in nset\ n\ (n + Suc\ m).\ f\ j \in carrier\ A \Longrightarrow$
fSum A f n (*n* + *Suc m*) = *fSum A f n* (*n* + *m*) ± *f* (*n* + *Suc m*)
by (*simp add*:*fSum-def*, *simp add*:*cmp-def slide-def*)

lemma (**in** *aGroup*) *fSum-eqTr*:($\forall j \in nset\ n\ (n + m).\ f\ j \in carrier\ A \wedge$
$g\ j \in carrier\ A \wedge\ f\ j = g\ j) \longrightarrow$
fSum A f n (*n* + *m*) = *fSum A g n* (*n* + *m*)
apply (*induct-tac m*)
apply (*simp add*:*fSum-def*,
simp add:*cmp-def slide-def*,
simp add:*nset-def*)

apply (*rule impI*)
apply (*subst fSum-Suc*,
rule ballI, *simp*, *simp*)
apply (*cut-tac n* = *n* **and** *m* = *na* **and** *f* = *g* **in** *fSum-Suc*,
rule ballI, *simp*, *simp*,
thin-tac $\Sigma_f\ A\ g\ n\ (Suc\ (n + na)) =$
$\Sigma_f\ A\ g\ n\ (n + na) \pm g\ (Suc\ (n + na))$

apply (*cut-tac n* = *n* **and** *m* = *na* **in** *nsetnm-sub-mem*, *simp*,
thin-tac $\forall j.\ j \in nset\ n\ (n + na) \longrightarrow j \in nset\ n\ (Suc\ (n + na))$)
apply (*frule-tac x* = *Suc* (*n* + *na*) **in** *bspec*,
simp add:*nset-def*, *simp*)
done

lemma (**in** *aGroup*) *fSum-eq*:⟦ $\forall j \in nset\ n\ (n + m).\ f\ j \in carrier\ A$;
$\forall j \in nset\ n\ (n + m).\ g\ j \in carrier\ A$; ($\forall j \in nset\ n\ (n + m).\ f\ j = g\ j$)⟧
⟹
fSum A f n (*n* + *m*) = *fSum A g n* (*n* + *m*)
by (*simp add*:*fSum-eqTr*)

lemma (**in** *aGroup*) *fSum-eq1*:⟦$n \leq m$; $\forall j{\in}nset\ n\ m.\ f\ j \in carrier\ A$;
$\forall j{\in}nset\ n\ m.\ g\ j \in carrier\ A$; $\forall j{\in}nset\ n\ m.\ f\ j = g\ j$⟧ ⟹
fSum A f n m = *fSum A g n m*
apply (*cut-tac fSum-eq*[*of n m* − *n f g*])
apply *simp*+
done

lemma (**in** *aGroup*) *fSum-zeroTr*:($\forall j \in nset\ n\ (n + m).\ f\ j = \mathbf{0}$) ⟶
fSum A f n (*n* + *m*) = **0**
apply (*induct-tac m*)
apply (*simp add*:*fSum-def cmp-def slide-def nset-def*)
apply (*rule impI*)
apply (*subst fSum-Suc*)
apply (*rule ballI*, *simp add*:*ag-inc-zero*)
apply (*frule-tac x* = *n* + *Suc na* **in** *bspec*, *simp add*:*nset-def*,
simp)
apply (*simp add*:*nset-def*)
apply (*cut-tac ag-inc-zero*, *simp add*:*ag-l-zero*)
done

lemma (**in** *aGroup*) *fSum-zero*:$\forall j \in nset\ n\ (n + m).\ f\ j = \mathbf{0}$ ⟹
fSum A f n (*n* + *m*) = **0**
by (*simp add*:*fSum-zeroTr*)

lemma (**in** *aGroup*) *fSum-zero1*:⟦$n < m$; $\forall j \in nset\ (Suc\ n)\ m.\ f\ j = \mathbf{0}$⟧ ⟹
fSum A f (*Suc n*) *m* = **0**
apply (*cut-tac fSum-zero*[*of Suc n m* − *Suc n f*])
apply *simp*+
done

lemma (**in** *Ring*) *nsumMulEleL*: ⋀ *n*. ⟦ ∀ *i*. *f i* ∈ *carrier R*; *x* ∈ *carrier R* ⟧
⟹ *x* ·$_r$ (*nsum R f n*) = *nsum R* (λ *i*. *x* ·$_r$ (*f i*)) *n*
apply (*cut-tac ring-is-ag*)
apply (*induct-tac n*)
apply *simp*

apply *simp*
apply (*subst ring-distrib1*, *assumption*)
apply (*rule aGroup.nsum-mem*, *assumption*)
apply (*rule allI*, *simp*+)
done

lemma (**in** *Ring*) *nsumMulElmL*:
⋀ *n*. ⟦ ∀ *i*. *f i* ∈ *carrier R*; *x* ∈ *carrier R* ⟧
⟹ *x* ·$_r$ (*nsum R f n*) = *nsum R* (λ *i*. *x* ·$_r$ (*f i*)) *n*
apply (*cut-tac ring-is-ag*)
apply (*induct-tac n*)
apply *simp*

```
 apply simp
 apply (subst ring-distrib1, assumption+)
  apply (simp add:aGroup.nsum-mem)+
 done

lemma (in aGroup) nsumTailTr:
       (∀ j≤(Suc n). f j ∈ carrier A) ⟶
        nsum A f (Suc n) = (nsum A (λ i. (f (Suc i))) n) ± (f 0)
 apply (induct-tac n)
 apply simp
 apply (rule impI,
       rule ag-pOp-commute)
 apply (cut-tac Nset-inc-0[of Suc 0],
       simp add:Pi-def,
       cut-tac n-in-Nsetn[of Suc 0],
       simp add:Pi-def)

 apply (rule impI)
  apply (cut-tac n = Suc n in Nsetn-sub-mem1, simp)
  apply (frule-tac a = 0 in forall-spec, simp,
        frule-tac a = Suc (Suc n) in forall-spec, simp)
   apply (cut-tac n = n in nsum-mem[of - λi. f (Suc i)],
        rule allI, rule impI,
        frule-tac a = Suc j in forall-spec, simp, simp,
        thin-tac ∀ j≤Suc (Suc n). f j ∈ carrier A)
   apply (subst ag-pOp-assoc, assumption+)
     apply (simp add:ag-pOp-commute[of f 0])
   apply (subst ag-pOp-assoc[THEN sym], assumption+)
   apply simp
 done

lemma (in aGroup) nsumTail:
     ∀ j ≤ (Suc n). f j ∈ carrier A ⟹
          nsum A f (Suc n) = (nsum A (λ i. (f (Suc i))) n) ± (f 0)
 by (cut-tac nsumTailTr[of n f], simp)

lemma (in aGroup) nsumElmTail:
 ∀ i. f i ∈ carrier A
      ⟹ nsum A f (Suc n) = (nsum A (λ i. (f (Suc i))) n) ± (f 0)
 apply (cut-tac n = n and f = f in nsumTail,
       rule allI, simp, simp)
done

lemma (in aGroup) nsum-addTr:
 (∀ j ≤ n. f j ∈ carrier A ∧ g j ∈ carrier A) ⟶
  nsum A (λ i. (f i) ± (g i)) n = (nsum A f n) ± (nsum A g n)
 apply (induct-tac n)
 apply simp
```

apply (*simp*, *rule impI*)
apply (*cut-tac* $n = n$ **in** *Nsetn-sub-mem1*, *simp*)
apply (*thin-tac* $\Sigma_e\ A\ (\lambda i.\ f\ i \pm g\ i)\ n = \Sigma_e\ A\ f\ n \pm \Sigma_e\ A\ g\ n$)
apply (*rule aGroup.ag-add4-rel*, *rule aGroup-axioms*)
apply (*rule aGroup.nsum-mem*, *rule aGroup-axioms*, *rule allI*, *simp*)
apply (*rule aGroup.nsum-mem*, *rule aGroup-axioms*, *rule allI*, *simp*)
apply *simp*+
done

lemma (**in** *aGroup*) *nsum-add*:
$[\![\ \forall j \le n.\ f\ j \in carrier\ A;\ \forall j \le n.\ g\ j \in carrier\ A]\!] \Longrightarrow$
$nsum\ A\ (\lambda\ i.\ (f\ i) \pm (g\ i))\ n = (nsum\ A\ f\ n) \pm (nsum\ A\ g\ n)$
by (*cut-tac nsum-addTr*[*of n f g*], *simp*)

lemma (**in** *aGroup*) *nsumElmAdd*:
$[\![\ \forall\ i.\ f\ i \in carrier\ A;\ \forall\ i.\ g\ i \in carrier\ A]\!]$
$\Longrightarrow nsum\ A\ (\lambda\ i.\ (f\ i) \pm (g\ i))\ n = (nsum\ A\ f\ n) \pm (nsum\ A\ g\ n)$
apply (*cut-tac nsum-add*[*of n f g*])
apply *simp*
apply (*rule allI*, *simp*)+
done

lemma (**in** *aGroup*) *nsum-add-nmTr*:
$(\forall j \le n.\ f\ j \in carrier\ A) \wedge (\forall j \le m.\ g\ j \in carrier\ A) \longrightarrow$
$nsum\ A\ (jointfun\ n\ f\ m\ g)\ (Suc\ (n + m)) = (nsum\ A\ f\ n) \pm (nsum\ A\ g\ m)$
apply (*induct-tac m*)
apply (*simp add*:*jointfun-def sliden-def*)
apply (*rule impI*)
apply (*rule ag-pOp-add-r*)
apply (*rule nsum-mem*, *rule allI*, *erule conjE*, *rule impI*, *simp*)
apply (*erule conjE*, *simp add*:*nsum-mem*, *simp*)
apply (*rule nsum-eq*[*of n*], *simp*+)
apply (*simp add*:*jointfun-def*)
apply (*rule impI*, *simp*)
apply (*erule conjE*, *simp add*:*sliden-def*)
apply (*thin-tac* $\Sigma_e\ A\ (\lambda i.$ *if* $i \le n$ *then* $f\ i$ *else* $g\ (sliden\ (Suc\ n)\ i))$
$(n + na) \pm g\ na = \Sigma_e\ A\ f\ n \pm \Sigma_e\ A\ g\ na$)
apply (*subst ag-pOp-assoc*)
apply (*simp add*:*nsum-mem*)
apply (*simp add*:*nsum-mem*, *simp*)
apply *simp*
done

lemma (**in** *aGroup*) *nsum-add-nm*:
$[\![\forall j \le n.\ f\ j \in carrier\ A;\ \forall j \le m.\ g\ j \in carrier\ A]\!] \Longrightarrow$
$nsum\ A\ (jointfun\ n\ f\ m\ g)\ (Suc\ (n + m)) = (nsum\ A\ f\ n) \pm (nsum\ A\ g\ m)$
apply (*cut-tac nsum-add-nmTr*[*of n f m g*])
apply *simp*

done

lemma (**in** *Ring*) *npeSum2-sub-muly*:
 ⟦ $x \in carrier\ R$; $y \in carrier\ R$ ⟧ ⟹
 $y \cdot_r (nsum\ R\ (\lambda i.\ nscal\ R\ ((npow\ R\ x\ (n-i)) \cdot_r (npow\ R\ y\ i))$
 $(n\ choose\ i))\ n)$
 $= nsum\ R\ (\lambda i.\ nscal\ R\ ((npow\ R\ x\ (n-i)) \cdot_r (npow\ R\ y\ (i+1)))$
 $(n\ choose\ i))\ n$
 apply (*cut-tac ring-is-ag*)
 apply (*subst nsumMulElmL*)
 apply (*rule allI*)
 apply (*simp only*:*nsClose add*:*ring-tOp-closed*
 add:*npClose*)
 apply *assumption*
 apply (*simp only*:*nsMulDistrL add*:*nsClose add*:*ring-tOp-closed*
 add:*npClose*)
 apply (*simp only*: *rMulLC* [*of y*] *add*:*npClose*)

 apply (*simp del*:*npow-suc add*:*ring-tOp-commute*[*of y*])
 apply (*rule aGroup.nsum-eq*, *assumption*)
 apply (*rule allI*, *rule impI*, *rule nsClose*,
 rule ring-tOp-closed, *simp add*:*npClose*,
 rule ring-tOp-closed, *assumption*, *simp add*:*npClose*)
 apply (*rule allI*, *rule impI*, *rule nsClose*,
 rule ring-tOp-closed, *simp add*:*npClose*,
 rule npClose, *assumption*)
 apply (*rule allI*, *rule impI*)
 apply (*frule-tac* $n = j$ **in** *npClose*[*of y*])
 apply (*simp add*:*ring-tOp-commute*[*of y*])
done

lemma *binomial-n0*: $(Suc\ n\ choose\ 0) = (n\ choose\ 0)$
 by *simp*

lemma *binomial-ngt-diff*:
 $(n\ choose\ Suc\ n) = (Suc\ n\ choose\ Suc\ n) - (n\ choose\ n)$
 by (*subst binomial-Suc-Suc*, *arith*)

lemma *binomial-ngt-0*: $(n\ choose\ Suc\ n) = 0$
 apply (*subst binomial-ngt-diff*,
 (*subst binomial-n-n*)+)
 apply *simp*
 done

lemma *diffLessSuc*: $m \leq n \Longrightarrow Suc\ (n-m) = Suc\ n - m$
 by *arith*

lemma (**in** *Ring*) *npow-suc-i*:
 $[\![\ x \in carrier\ R;\ i \leq n\ ]\!]$
  $\Longrightarrow npow\ R\ x\ (Suc\ n - i) = \ x \cdot_r (npow\ R\ x\ (n-i))$
 apply (*subst diffLessSuc* [*THEN sym*, *of i n*], *assumption*)
 apply (*frule-tac* $n = n - i$ **in** *npClose*,
   simp add:*ring-tOp-commute*[*of x*])
 done

lemma (**in** *Ring*) *npeSum2-sub-mulx*: $[\![\ x \in carrier\ R;\ y \in carrier\ R\ ]\!] \Longrightarrow$
 $x \cdot_r (nsum\ R\ (\lambda\ i.\ nscal\ R\ ((npow\ R\ x\ (n-i)) \cdot_r (npow\ R\ y\ i))$
  $(n\ choose\ i))\ n)$
 $= (nsum\ R\ (\lambda i.\ nscal\ R$
  $((npow\ R\ x\ (Suc\ n - Suc\ i)) \cdot_r (npow\ R\ y\ (Suc\ i)))$
  $(n\ choose\ Suc\ i))\ n) \pm$
  $(nscal\ R\ ((npow\ R\ x\ (Suc\ n - 0)) \cdot_r (npow\ R\ y\ 0))$
  $(Suc\ n\ choose\ 0))$
 apply (*cut-tac ring-is-ag*)
 apply (*simp only*: *binomial-n0*)
 apply (*subst aGroup.nsumElmTail* [*THEN sym*, *of R* λ *i. nscal R* ((*npow R x*
(*Suc n* $-$ *i*)) $\cdot_r$ (*npow R y i*)) (*n choose i*)], *assumption*+)
 apply (*rule allI*)
  apply (*simp only*:*nsClose add*:*ring-tOp-closed add*:*npClose*)

 apply (*simp only*:*nsum-suc*)
 apply (*subst binomial-ngt-0*)
 apply (*simp only*:*nscal-0*)
 apply (*subst aGroup.ag-r-zero*, *assumption*)
  apply (*simp add*:*aGroup.nsum-mem nsClose ring-tOp-closed npClose*)
 apply (*subst nsumMulElmL* [*of - x*])
  apply (*rule allI*, *rule nsClose*, *rule ring-tOp-closed*, *simp add*:*npClose*,
   simp add:*npClose*, *assumption*)

 apply (*simp add*: *nsMulDistrL* [*of x*] *ring-tOp-closed npClose*)
 apply (*simp add*:*ring-tOp-assoc* [*THEN sym*, *of x*] *npClose*)
 apply (*rule aGroup.nsum-eq*, *assumption*)
 apply (*rule allI*, *rule impI*,
   rule nsClose, (*rule ring-tOp-closed*)+, *assumption*,
   simp add:*npClose*, *simp add*:*npClose*)
 apply (*rule allI*, *rule impI*,
   rule nsClose, *rule ring-tOp-closed*,
   simp add:*npClose*, *simp add*:*npClose*)
 apply (*rule allI*, *rule impI*)
 apply (*frule-tac* $n = n - j$ **in** *npClose*[*of x*],
  simp add:*ring-tOp-commute*[*of x*],
  subst npow-suc[*THEN sym*])
 apply (*simp add*:*Suc-diff-le*)
done

lemma (**in** *Ring*) *npeSum2-sub-mulx2*:
 ⟦ $x \in$ *carrier* R; $y \in$ *carrier* R ⟧ ⟹
  $x \cdot_r$ (*nsum* R (λ i. *nscal* R ((*npow* R x ($n-i$)) $\cdot_r$ (*npow* R y i))
   (n *choose* i)) n)
  = (*nsum* R (λi. *nscal* R
   ((*npow* R x (n − i)) $\cdot_r$ ((*npow* R y i) $\cdot_r$ y))
   (n *choose Suc* i)) n) ±
   (**0** ± ((*x* $\cdot_r$ (*npow* R x n)) $\cdot_r$ (1_r)))
apply (*subst* *npeSum2-sub-mulx*, *assumption*+, *simp*)
apply (*frule npClose*[*of* x n])
apply (*subst ring-tOp-commute*[*of* x], *assumption*+)
 apply (*cut-tac ring-is-ag*)
 apply (*cut-tac aGroup.nsum-eq*[*of* R n
  λi. (n *choose Suc* i) $\times_R$ (x ^R (n − i) $\cdot_r$ y ^R (*Suc* i))
  λi. (n *choose Suc* i) $\times_R$ (x ^R (n − i) $\cdot_r$ (y ^R i $\cdot_r$ y))])
 apply (*simp del*:*npow-suc*)+
  apply (*rule allI*, *rule impI*,
   rule nsClose, *rule ring-tOp-closed*, *simp add*:*npClose*,
   simp only:*npClose*)
  apply (*rule allI*, *rule impI*,
   rule nsClose, *rule ring-tOp-closed*, *simp add*:*npClose*,
   rule ring-tOp-closed, *simp add*:*npClose*, *assumption*)
  apply (*rule allI*, *rule impI*)
 apply (*frule-tac* $n = j$ **in** *npClose*[*of* y])
 apply *simp*
done

lemma (**in** *Ring*) *npeSum2*:
 ⋀ n. ⟦ $x \in$ *carrier* R; $y \in$ *carrier* R ⟧
  ⟹ *npow* R ($x \pm y$) n =
   nsum R (λ i. *nscal* R ((*npow* R x ($n-i$)) $\cdot_r$ (*npow* R y i))
    (n *choose* i)) n
 apply (*cut-tac ring-is-ag*)
 apply (*induct-tac* n)

 apply *simp*
  apply (*cut-tac ring-one*, *simp add*:*ring-r-one*, *simp add*:*aGroup.ag-l-zero*)

 apply (*subst aGroup.nsumElmTail*, *assumption*+)
  apply (*rule allI*)
  apply (*simp add*:*nsClose ring-tOp-closed npClose*)

 apply (*simp only*:*binomial-Suc-Suc*)
 apply (*simp only*: *nsDistr* [*THEN sym*] *add*:*npClose ring-tOp-closed*)

apply (*subst aGroup.nsumElmAdd*, *assumption*+)
  apply (*rule allI*,
        simp add:*nsClose ring-tOp-closed npClose*)
  apply (*rule allI*,
        simp add:*nsClose add*:*ring-tOp-closed npClose*)
apply (*subst aGroup.ag-pOp-assoc*, *assumption*)
  apply (*rule aGroup.nsum-mem*, *assumption*,
        rule allI, *rule impI*, *simp add*:*nsClose ring-tOp-closed npClose*)
  apply (*rule aGroup.nsum-mem*, *assumption*,
        rule allI, *rule impI*, *simp add*:*nsClose ring-tOp-closed npClose*)
  apply (*simp add*:*nsClose ring-tOp-closed npClose*)
  apply (*rule aGroup.ag-pOp-closed*, *assumption*)
  apply (*simp add*:*aGroup.ag-inc-zero*)
  apply (*rule ring-tOp-closed*)+
  apply (*simp add*:*npClose*, *assumption*, *simp add*:*ring-one*)

apply (*subst npMulElmL* [*THEN sym*, *of x* ± *y*],
      simp add:*aGroup.ag-pOp-closed*, *simp*)
 apply *simp*
apply (*subst ring-distrib2* [*of - x y*])
apply (*rule aGroup.nsum-mem*,*assumption*,
      rule allI, *rule impI*, *rule nsClose*, *rule ring-tOp-closed*,
      simp add:*npClose*, *simp add*:*npClose*, *assumption*+)
apply (*rule aGroup.gEQAddcross* [*THEN sym*], *assumption*+,
      rule aGroup.nsum-mem, *assumption*, *rule allI*, *rule impI*, *rule nsClose*,
      (*rule ring-tOp-closed*)+, *simp add*:*npClose*,
      rule ring-tOp-closed, *simp add*:*npClose*, *assumption*)
  apply (*rule aGroup.ag-pOp-closed*, *assumption*)
  apply (*rule aGroup.nsum-mem*, *assumption*,
        rule allI, *rule impI*, *rule nsClose*, *rule ring-tOp-closed*,
       simp add:*npClose*, *rule ring-tOp-closed*, *simp add*:*npClose*, *assumption*)
  apply (*rule aGroup.ag-pOp-closed*, *assumption*, *simp add*:*ring-zero*)
  apply ((*rule ring-tOp-closed*)+,
        simp add:*npClose*,*assumption*, *simp add*:*ring-one*)
  apply (*rule ring-tOp-closed*, *assumption*,
        rule aGroup.nsum-mem, *assumption*, *rule allI*, *rule impI*,
        rule nsClose, *rule ring-tOp-closed*,
        (*simp add*:*npClose*)+)
  apply (*rule ring-tOp-closed*, *assumption*+,
        rule aGroup.nsum-mem, *assumption*, *rule allI*, *rule impI*,
        rule nsClose,
        rule ring-tOp-closed,
        simp add:*npClose*, *simp add*:*npClose*)
  apply (*subst npeSum2-sub-muly* [*of x y*], *assumption*+, *simp*)

apply (*subst npeSum2-sub-mulx2* [*of x y*], *assumption*+)
apply (*frule-tac n* = *na* **in** *npClose*[*of x*],
      simp add:*ring-tOp-commute*[*of - x*])

done

lemma (**in** *aGroup*) *nsum-zeroTr*:
$\bigwedge n.\ (\forall\ i.\ i \leq n \longrightarrow\ f\ i = \mathbf{0}) \longrightarrow (nsum\ A\ f\ n = \mathbf{0})$
apply (*induct-tac n*)
apply *simp*

apply (*rule impI*)
apply (*cut-tac n = na* **in** *Nsetn-sub-mem1*, *simp*)
apply (*subst aGroup.ag-l-zero*, *rule aGroup-axioms*)
apply (*simp add*:*ag-inc-zero*)
apply *simp*
done

lemma (**in** *Ring*) *npAdd*:
$[\![\ x \in carrier\ R;\ y \in carrier\ R;$
$npow\ R\ x\ m = \mathbf{0};\ npow\ R\ y\ n = \mathbf{0}\ ]\!]$
$\Longrightarrow npow\ R\ (x \pm y)\ (m + n) = \mathbf{0}$
apply (*subst npeSum2*, *assumption*+)

apply (*rule aGroup.nsum-zeroTr* [*THEN mp*])
apply (*simp add*:*ring-is-ag*)
apply (*rule allI*, *rule impI*)
apply (*rule nsZeroI*)
apply (*rule rMulZeroDiv*, *simp add*:*npClose*, *simp add*:*npClose*)

apply (*case-tac* $i \leq n$)

apply (*rule disjI1*)
apply (*rule npGTPowZero* [*of x m*], *assumption*+)
apply *arith*

apply (*rule disjI2*)
apply (*rule npGTPowZero* [*of y n*], *assumption*+)
apply (*arith*)
done

lemma (**in** *Ring*) *npInverse*:
$\bigwedge n.\ x \in carrier\ R$
$\Longrightarrow npow\ R\ (-_a\ x)\ n = npow\ R\ x\ n$
$\vee\ npow\ R\ (-_a\ x)\ n = -_a\ (npow\ R\ x\ n)$
apply (*induct-tac n*)

apply *simp*

apply (*erule disjE*)
apply *simp*
apply (*subst ring-inv1-2*,
simp add:*npClose*, *assumption*, *simp*)

apply (*cut-tac ring-is-ag*)

apply *simp*
apply (*subst ring-inv1-2*[*THEN sym*, *of* - *x*])
apply (*rule aGroup.ag-mOp-closed*, *assumption*+,
    simp add:*npClose*, *assumption*)
apply (*thin-tac* $(-_a\ x)\ \hat{}_R\ na = -_a\ (x\ \hat{}_R\ na)$,
    frule-tac $n = na$ **in** *npClose*[*of x*],
    frule-tac $x = x\ \hat{}_R\ na$ **in** *aGroup.ag-mOp-closed*[*of R*], *simp add*:*npClose*)
apply (*simp add*: *ring-inv1-1*[*of* - *x*])
apply (*simp add*:*aGroup.ag-inv-inv*[*of R*])
done

lemma (**in** *Ring*) *npMul*:
$\bigwedge n.\ [\![\ x \in carrier\ R;\ y \in carrier\ R\ ]\!]$
    $\Longrightarrow npow\ R\ (x \cdot_r y)\ n = (npow\ R\ x\ n) \cdot_r (npow\ R\ y\ n)$
apply (*induct-tac n*)

apply *simp*
apply (*rule ring-r-one* [*THEN sym*]) **apply** (*simp add*:*ring-one*)

apply (*simp only*:*npow-suc*)
apply (*rule ring-tOp-rel*[*THEN sym*])
apply (*rule npClose*, *assumption*+)+
done

4.3 Ring homomorphisms

definition
rHom :: [('*a*, '*m*) *Ring-scheme*, ('*b*, '*m1*) *Ring-scheme*]
    $\Rightarrow$ ('*a* $\Rightarrow$ '*b*) *set* **where**
$rHom\ A\ R = \{f.\ f \in aHom\ A\ R \wedge$
$(\forall x{\in}carrier\ A.\ \forall y{\in}carrier\ A.\ f\ (\ x \cdot_{r\,A}\ y) = (f\ x) \cdot_{r\,R} (f\ y))$
$\wedge\ f\ (1_{r\,A}) = (1_{r\,R})\}$

definition
rInvim :: [('*a*, '*m*) *Ring-scheme*, ('*b*, '*m1*) *Ring-scheme*, '*a* $\Rightarrow$ '*b*, '*b set*]
    $\Rightarrow$ '*a set* **where**
$rInvim\ A\ R\ f\ K = \{a.\ a \in carrier\ A \wedge f\ a \in K\}$

definition
rimg :: [('*a*, '*m*) *Ring-scheme*, ('*b*, '*m1*) *Ring-scheme*, '*a* $\Rightarrow$ '*b*] $\Rightarrow$
    '*b Ring* **where**
$rimg\ A\ R\ f = (\!|carrier{=}\ f\ `(carrier\ A),\ pop = pop\ R,\ mop = mop\ R,$
zero = *zero R*, *tp* = *tp R*, *un* = *un R* $|\!)$

definition
ridmap :: ('*a*, '*m*) *Ring-scheme* $\Rightarrow$ ('*a* $\Rightarrow$ '*a*) **where**
$ridmap\ R = (\lambda x{\in}carrier\ R.\ x)$

definition
r-isom :: [('*a*, '*m*) *Ring-scheme*, ('*b*, '*m1*) *Ring-scheme*] ⇒ *bool*
(**infixr** ≅$_r$ *100*) **where**
r-isom R R' ⟷ (∃*f*∈*rHom R R'*. *bijec*$_{R,R'}$ *f*)

definition
Subring :: [('*a*, '*m*) *Ring-scheme*, ('*a*, '*m1*) *Ring-scheme*] ⇒ *bool* **where**
Subring R S == *Ring S* ∧ (*carrier S* ⊆ *carrier R*) ∧ (*ridmap S*) ∈ *rHom S R*

lemma *ridmap-surjec*:*Ring A* ⟹ *surjec*$_{A,A}$ (*ridmap A*)
by(*simp add*:*surjec-def aHom-def ridmap-def Ring.ring-is-ag aGroup.ag-pOp-closed surj-to-def*)

lemma *rHom-aHom*:*f* ∈ *rHom A R* ⟹ *f* ∈ *aHom A R*
by (*simp add*:*rHom-def*)

lemma *rimg-carrier*:*f* ∈ *rHom A R* ⟹ *carrier* (*rimg A R f*) = *f* ‘ (*carrier A*)
by (*simp add*:*rimg-def*)

lemma *rHom-mem*:⟦ *f* ∈ *rHom A R*; *a* ∈ *carrier A* ⟧ ⟹ *f a* ∈ *carrier R*
apply (*simp add*:*rHom-def*, *frule conjunct1*)
apply (*thin-tac f* ∈ *aHom A R* ∧
(∀ *x*∈*carrier A*. ∀ *y*∈*carrier A*. *f* (*x* ·$_{r\,A}$ *y*) = *f x* ·$_{r\,R}$ *f y*) ∧ *f* 1$_{r\,A}$ = 1$_{r\,R}$)
apply (*simp add*:*aHom-def*, *frule conjunct1*)
apply (*thin-tac f* ∈ *carrier A* → *carrier R* ∧
f ∈ *extensional* (*carrier A*) ∧
(∀ *a*∈*carrier A*. ∀ *b*∈*carrier A*. *f* (*a* ±$_A$ *b*) = *f a* ±$_R$ *f b*))
apply (*simp add*:*funcset-mem*)
done

lemma *rHom-func*:*f* ∈ *rHom A R* ⟹ *f* ∈ *carrier A* → *carrier R*
by (*simp add*:*rHom-def aHom-def*)

lemma *ringhom1*:⟦ *Ring A*; *Ring R*; *x* ∈ *carrier A*; *y* ∈ *carrier A*;
f ∈ *rHom A R* ⟧ ⟹ *f* (*x* ±$_A$ *y*) = (*f x*) ±$_R$ (*f y*)
apply (*simp add*:*rHom-def*) **apply** (*erule conjE*)
apply (*frule Ring.ring-is-ag* [*of A*])
apply (*frule Ring.ring-is-ag* [*of R*])
apply (*rule aHom-add*, *assumption*+)
done

lemma *rHom-inv-inv*:⟦ *Ring A*; *Ring R*; *x* ∈ *carrier A*; *f* ∈ *rHom A R* ⟧
⟹ *f* (−$_{a\,A}$ *x*) = −$_{a\,R}$ (*f x*)
apply (*frule Ring.ring-is-ag* [*of A*],
frule Ring.ring-is-ag [*of R*])
apply (*simp add*:*rHom-def*, *erule conjE*)
apply (*simp add*:*aHom-inv-inv*)
done

lemma *rHom-0-0*:⟦ *Ring A*; *Ring R*; $f \in rHom\ A\ R$ ⟧ $\Longrightarrow f\ (\mathbf{0}_A) = \mathbf{0}_R$
apply (*frule Ring.ring-is-ag* [*of A*], *frule Ring.ring-is-ag* [*of R*])
apply (*simp add*:*rHom-def*, (*erule conjE*)+, *simp add*:*aHom-0-0*)
done

lemma *rHom-tOp*:⟦ *Ring A*; *Ring R*; $x \in carrier\ A$; $y \in carrier\ A$;
$f \in rHom\ A\ R$ ⟧ $\Longrightarrow f\ (x \cdot_{r A}\ y) = (f\ x) \cdot_{r R}\ (f\ y)$
by (*simp add*:*rHom-def*)

lemma *rHom-add*:⟦$f \in rHom\ A\ R$; $x \in carrier\ A$; $y \in carrier\ A$⟧ $\Longrightarrow$
$f\ (x \pm_A\ y) = (f\ x) \pm_R\ (f\ y)$
by (*simp add*:*rHom-def aHom-def*)

lemma *rHom-one*:⟦ *Ring A*; *Ring R*;$f \in rHom\ A\ R$ ⟧ $\Longrightarrow f\ (1_{r A}) = (1_{r R})$
by (*simp add*:*rHom-def*)

lemma *rHom-npow*:⟦ *Ring A*; *Ring R*; $x \in carrier\ A$; $f \in rHom\ A\ R$ ⟧ $\Longrightarrow$
$f\ (x \hat{}_A\ n) = (f\ x) \hat{}_R\ n$
apply (*induct-tac n*)
apply (*simp add*:*rHom-one*)
apply (*simp*,
frule-tac n = n **in** *Ring.npClose*[*of A x*], *assumption*+,
subst rHom-tOp[*of A R - x f*], *assumption*+, *simp*)
done

lemma *rHom-compos*:⟦*Ring A*; *Ring B*; *Ring C*; $f \in rHom\ A\ B$; $g \in rHom\ B\ C$⟧
$\Longrightarrow$
$compos\ A\ g\ f \in rHom\ A\ C$
apply (*subst rHom-def*, *simp*)
apply (*frule Ring.ring-is-ag*[*of A*], *frule Ring.ring-is-ag*[*of B*],
frule Ring.ring-is-ag[*of C*],
frule rHom-aHom[*of f A B*], *frule rHom-aHom*[*of g B C*],
simp add:*aHom-compos*)
apply (*rule conjI*)
apply ((*rule ballI*)+, *simp add*:*compos-def compose-def*,
frule-tac x = x **and** *y = y* **in** *Ring.ring-tOp-closed*[*of A*], *assumption*+,
simp)
apply (*simp add*:*rHom-tOp*)
apply (*frule-tac a = x* **in** *rHom-mem*[*of f A B*], *assumption*+,
frule-tac a = y **in** *rHom-mem*[*of f A B*], *assumption*+,
simp add:*rHom-tOp*)
apply (*frule Ring.ring-one*[*of A*], *frule Ring.ring-one*[*of B*],
simp add:*compos-def compose-def*, *simp add*:*rHom-one*)
done

lemma *rimg-ag*:⟦*Ring A*; *Ring R*; $f \in rHom\ A\ R$⟧ $\Longrightarrow aGroup\ (rimg\ A\ R\ f)$
apply (*frule Ring.ring-is-ag* [*of A*],
frule Ring.ring-is-ag [*of R*])

apply (*simp add*:*rHom-def*, (*erule conjE*)+)
apply (*subst aGroup-def*)
apply (*simp add*:*rimg-def*)
apply (*rule conjI*)
 apply (*rule Pi-I*)+
 apply (*simp add*:*image-def*)
 apply (*erule bexE*)+
 apply *simp*
 apply (*subst aHom-add* [*THEN sym*, *of A R f*], *assumption*+)
 apply (*blast dest*: *aGroup.ag-pOp-closed*)
apply (*rule conjI*)
 apply ((*rule allI*, *rule impI*)+, *simp add*:*image-def*, (*erule bexE*)+, *simp*)
 apply (*frule-tac x* = *x* **and** *y* = *xa* **in** *aGroup.ag-pOp-closed*, *assumption*+,
 frule-tac x = *xa* **and** *y* = *xb* **in** *aGroup.ag-pOp-closed*, *assumption*+)
 apply (*simp add*:*aHom-add*[*of A R f*, *THEN sym*] *aGroup.ag-pOp-assoc*)
apply (*rule conjI*)
 apply ((*rule allI*, *rule impI*)+, *simp add*:*image-def*, (*erule bexE*)+, *simp*)
 apply (*simp add*:*aHom-add*[*of A R f*, *THEN sym*] *aGroup.ag-pOp-commute*)
apply (*rule conjI*)
 apply (*rule Pi-I*)
 apply (*simp add*:*image-def*, *erule bexE*, *simp*)
 apply (*simp add*:*aHom-inv-inv*[*THEN sym*],
 frule-tac x = *xa* **in** *aGroup.ag-mOp-closed*[*of A*], *assumption*+, *blast*)
apply (*rule conjI*)
 apply (*rule allI*, *rule impI*, *simp add*:*image-def*, (*erule bexE*)+, *simp*)
 apply (*simp add*:*aHom-inv-inv*[*THEN sym*],
 frule-tac x = *x* **in** *aGroup.ag-mOp-closed*[*of A*], *assumption*+,
 simp add:*aHom-add*[*of A R f*, *THEN sym*])
 apply (*simp add*:*aGroup.ag-l-inv1 aHom-0-0*)
apply (*rule conjI*)
 apply (*simp add*:*image-def*)
 apply (*frule aHom-0-0*[*THEN sym*, *of A R f*], *assumption*+,
 frule Ring.ring-zero[*of A*], *blast*)

apply (*rule allI*, *rule impI*,
 simp add:*image-def*, *erule bexE*,
 frule-tac a = *x* **in** *aHom-mem*[*of A R f*], *assumption*+, *simp*)
 apply (*simp add*:*aGroup.ag-l-zero*)
done

lemma *rimg-ring*:⟦*Ring A*; *Ring R*; *f* ∈ *rHom A R* ⟧ ⟹ *Ring* (*rimg A R f*)
apply (*unfold Ring-def* [*of rimg A R f*])
apply (*frule rimg-ag*[*of A R f*], *assumption*+)
 apply (*rule conjI*, *simp add*:*aGroup-def*[*of rimg A R f*])
apply(*rule conjI*)
 apply (*rule conjI*, *rule allI*, *rule impI*)
 apply (*frule aGroup.ag-inc-zero*[*of rimg A R f*],
 subst aGroup.ag-pOp-commute, *assumption*+,
 simp add:*aGroup.ag-r-zero*[*of rimg A R f*])

apply (*rule conjI*)
apply (*rule Pi-I*)+
apply (*thin-tac aGroup* (*rimg A R f*),
simp add:*rimg-def*, *simp add*:*image-def*, (*erule bexE*)+,
simp add:*rHom-tOp*[*THEN sym*])
apply (*blast dest*:*Ring.ring-tOp-closed*)
apply ((*rule allI*)+, (*rule impI*)+)
apply (*thin-tac aGroup* (*rimg A R f*), *simp add*:*rimg-def*,
simp add:*image-def*, (*erule bexE*)+, *simp*)
apply (*frule-tac x* = *x* **and** *y* = *xa* **in** *Ring.ring-tOp-closed*, *assumption*+,
frule-tac x = *xa* **and** *y* = *xb* **in** *Ring.ring-tOp-closed*, *assumption*+,
simp add:*rHom-tOp*[*THEN sym*],
simp add:*Ring.ring-tOp-assoc*)
apply (*rule conjI*, *rule conjI*, (*rule allI*)+, (*rule impI*)+)
apply (*thin-tac aGroup* (*rimg A R f*), *simp add*:*rimg-def*,
simp add:*image-def*, (*erule bexE*)+, *simp*,
simp add:*rHom-tOp*[*THEN sym*],
simp add:*Ring.ring-tOp-commute*)
apply (*thin-tac aGroup* (*rimg A R f*), *simp add*:*rimg-def*,
simp add:*image-def*)
apply (*subst rHom-one* [*THEN sym*, *of A R f*], *assumption*+,
frule Ring.ring-one[*of A*], *blast*)
apply (*rule conjI*, (*rule allI*)+, (*rule impI*)+)
apply (*simp add*:*rimg-def*, *fold rimg-def*,
simp add:*image-def*, (*erule bexE*)+, *simp*)
apply (*frule rHom-aHom*[*of f A R*],
frule Ring.ring-is-ag [*of A*],
frule Ring.ring-is-ag [*of R*],
simp add:*aHom-add*[*THEN sym*],
simp add:*rHom-tOp*[*THEN sym*])
apply (*frule-tac x* = *xa* **and** *y* = *xb* **in** *aGroup.ag-pOp-closed*[*of A*],
assumption+,
frule-tac x = *x* **and** *y* = *xa* **in** *Ring.ring-tOp-closed*[*of A*],
assumption+,
frule-tac x = *x* **and** *y* = *xb* **in** *Ring.ring-tOp-closed*[*of A*],
assumption+,
simp add:*aHom-add*[*THEN sym*],
simp add:*rHom-tOp*[*THEN sym*],
simp add:*Ring.ring-distrib1*)
apply (*rule allI*, *rule impI*,
thin-tac aGroup (*rimg A R f*))
apply (*simp add*:*rimg-def*,
simp add:*image-def*, *erule bexE*, *simp add*:*rHom-tOp*[*THEN sym*],
frule-tac a = *x* **in** *rHom-mem*[*of f A R*], *assumption*+,
simp add:*Ring.ring-l-one*)
done

definition

ideal :: [- , *'a set*] ⇒ *bool* **where**
ideal R I ⟷ (*R* +> *I*) ∧ (∀ *r*∈*carrier R*. ∀ *x*∈*I*. (*r* ·$_{r\,R}$ *x* ∈ *I*))

lemma (**in** *Ring*) *ideal-asubg*:*ideal R I* ⟹ *R* +> *I*
by (*simp add*:*ideal-def*)

lemma (**in** *Ring*) *ideal-pOp-closed*:⟦*ideal R I*; *x* ∈ *I*; *y* ∈ *I* ⟧
⟹ *x* ± *y* ∈ *I*
apply (*unfold ideal-def*, *frule conjunct1*, *fold ideal-def*)
apply (*cut-tac ring-is-ag*,
simp add:*aGroup.asubg-pOp-closed*)
done

lemma (**in** *Ring*) *ideal-nsum-closedTr*:*ideal R I* ⟹
(∀ *j* ≤ *n*. *f j* ∈ *I*) ⟶ *nsum R f n* ∈ *I*
apply (*induct-tac n*)
apply (*rule impI*)
apply *simp*

apply (*rule impI*)
apply (*cut-tac n* = *n* **in** *Nsetn-sub-mem1*, *simp*)
apply (*rule ideal-pOp-closed*, *assumption*+)
apply *simp*
done

lemma (**in** *Ring*) *ideal-nsum-closed*:⟦*ideal R I*; ∀ *j* ≤ *n*. *f j* ∈ *I*⟧ ⟹
nsum R f n ∈ *I*
by (*simp add*:*ideal-nsum-closedTr*)

lemma (**in** *Ring*) *ideal-subset1*:*ideal R I* ⟹ *I* ⊆ *carrier R*
apply (*unfold ideal-def*, *frule conjunct1*, *fold ideal-def*)
apply (*simp add*:*asubGroup-def sg-def*, (*erule conjE*)+)
apply (*cut-tac ring-is-ag*,
simp add:*aGroup.ag-carrier-carrier*)
done

lemma (**in** *Ring*) *ideal-subset*:⟦*ideal R I*; *h* ∈ *I*⟧ ⟹ *h* ∈ *carrier R*
by (*frule ideal-subset1*[*of I*],
simp add:*subsetD*)

lemma (**in** *Ring*) *ideal-ring-multiple*:⟦*ideal R I*; *x* ∈ *I*; *r* ∈ *carrier R*⟧ ⟹
r ·$_r$ *x* ∈ *I*
by (*simp add*:*ideal-def*)

lemma (**in** *Ring*) *ideal-ring-multiple1*:⟦*ideal R I*; *x* ∈ *I*; *r* ∈ *carrier R* ⟧ ⟹
x ·$_r$ *r* ∈ *I*
apply (*frule ideal-subset*[*of I x*], *assumption*+)
apply (*simp add*:*ring-tOp-commute ideal-ring-multiple*)

done

lemma (**in** *Ring*) *ideal-npow-closedTr*:⟦*ideal R I*; $x \in I$⟧ ⟹
$0 < n \longrightarrow x \hat{}_R\ n \in I$
apply (*induct-tac n*,
simp)
apply (*rule impI*)
apply *simp*
apply (*case-tac* $n = 0$, *simp*)
apply (*frule ideal-subset*[*of I x*], *assumption*+,
simp add:*ring-l-one*)

apply *simp*
apply (*frule ideal-subset*[*of I x*], *assumption*+,
rule ideal-ring-multiple, *assumption*+,
simp add:*ideal-subset*)
done

lemma (**in** *Ring*) *ideal-npow-closed*:⟦*ideal R I*; $x \in I$; $0 < n$⟧ ⟹ $x \hat{}_R\ n \in I$
by (*simp add*:*ideal-npow-closedTr*)

lemma (**in** *Ring*) *times-modTr*:⟦$a \in$ *carrier R*; $a' \in$ *carrier R*; $b \in$ *carrier R*;
$b' \in$ *carrier R*; *ideal R I*; $a \pm (-_a\ b) \in I$; $a' \pm (-_a\ b') \in I$⟧ ⟹
$a \cdot_r\ a' \pm (-_a\ (b \cdot_r\ b')) \in I$
apply (*cut-tac ring-is-ag*)
apply (*subgoal-tac* $a \cdot_r\ a' \pm (-_a\ (b \cdot_r\ b')) = a \cdot_r\ a' \pm (-_a\ (a \cdot_r\ b'))$
$\pm\ (a \cdot_r\ b' \pm (-_a\ (b \cdot_r\ b')))$)
apply *simp*
apply (*simp add*:*ring-inv1-2*[*of a b'*], *simp add*:*ring-inv1-1*[*of b b'*])
apply (*frule aGroup.ag-mOp-closed*[*of R b'*], *assumption*+)
apply (*simp add*:*ring-distrib1*[*THEN sym*, *of a a'* $-_a$ *b'*])
apply (*frule aGroup.ag-mOp-closed*[*of R b*], *assumption*+)
apply (*frule ring-distrib2*[*THEN sym*, *of b' a* $-_a$ *b*], *assumption*+)
apply *simp*

apply (*thin-tac* $a \cdot_r\ a' \pm (-_a\ b) \cdot_r\ b' = a \cdot_r\ (a' \pm -_a\ b') \pm (a \pm -_a\ b) \cdot_r\ b'$,
thin-tac $a \cdot_r\ b' \pm (-_a\ b) \cdot_r\ b' = (a \pm -_a\ b) \cdot_r\ b'$)
apply (*frule ideal-ring-multiple*[*of I a'* $\pm (-_a\ b')\ a$], *assumption*+,
frule ideal-ring-multiple1[*of I a* $\pm (-_a\ b)\ b'$], *assumption*+)
apply (*simp add*:*ideal-pOp-closed*)

apply (*frule ring-tOp-closed*[*of a a'*], *assumption*+,
frule ring-tOp-closed[*of a b'*], *assumption*+,
frule ring-tOp-closed[*of b b'*], *assumption*+,
frule aGroup.ag-mOp-closed[*of R b* $\cdot_r\ b'$], *assumption*+,
frule aGroup.ag-mOp-closed[*of R a* $\cdot_r\ b'$], *assumption*+)

apply (*subst aGroup.ag-pOp-assoc*[*of R*], *assumption*+)
apply (*rule aGroup.ag-pOp-closed*, *assumption*+)

apply (*simp add*:*aGroup.ag-pOp-assoc*[*THEN sym*, *of R* $-_a$ (a $\cdot_r$ b') a $\cdot_r$ b'
$-_a$ (b $\cdot_r$ b')],
simp add:*aGroup.ag-l-inv1 aGroup.ag-l-zero*)
done

lemma (**in** *Ring*) *ideal-inv1-closed*:⟦ *ideal R I*; $x \in I$ ⟧ ⟹ $-_a$ $x \in I$
apply (*cut-tac ring-is-ag*)
apply (*unfold ideal-def*, *frule conjunct1*, *fold ideal-def*)
apply (*simp add*:*aGroup.asubg-mOp-closed*[*of R I*])
done

lemma (**in** *Ring*) *ideal-zero*:*ideal R I* ⟹ **0** $\in$ *I*

apply (*cut-tac ring-is-ag*)
apply (*unfold ideal-def*, *frule conjunct1*, *fold ideal-def*)
apply (*simp add*:*aGroup.asubg-inc-zero*)
done

lemma (**in** *Ring*) *ideal-zero-forall*:∀ *I*. *ideal R I* ⟶ **0** $\in$ *I*
by (*simp add*:*ideal-zero*)

lemma (**in** *Ring*) *ideal-ele-sumTr1*:⟦ *ideal R I*; $a \in$ *carrier R*; $b \in$ *carrier R*;
$a \pm b \in I$; $a \in I$ ⟧ ⟹ $b \in I$
apply (*frule ideal-inv1-closed*[*of I a*], *assumption*+)
apply (*frule ideal-pOp-closed*[*of I* $-_a$ *a a* $\pm$ *b*], *assumption*+)
apply (*frule ideal-subset*[*of I* $-_a$ *a*], *assumption*+)
apply (*cut-tac ring-is-ag*,
simp add:*aGroup.ag-pOp-assoc*[*THEN sym*],
simp add:*aGroup.ag-l-inv1*,
simp add:*aGroup.ag-l-zero*)
done

lemma (**in** *Ring*) *ideal-ele-sumTr2*:⟦*ideal R I*; $a \in$ *carrier R*; $b \in$ *carrier R*;
$a \pm b \in I$; $b \in I$⟧ ⟹ $a \in I$
apply (*cut-tac ring-is-ag*,
simp add:*aGroup.ag-pOp-commute*[*of R a b*])
apply (*simp add*:*ideal-ele-sumTr1*[*of I b a*])
done

lemma (**in** *Ring*) *ideal-condition*:⟦$I \subseteq$ *carrier R*; $I \neq \{\}$;
∀ x∈I. ∀ y∈I. $x \pm (-_a\ y) \in I$; ∀ r∈*carrier R*. ∀ x∈I. $r \cdot_r x \in I$ ⟧ ⟹
ideal R I
apply (*simp add*:*ideal-def*)
apply (*cut-tac ring-is-ag*)
apply (*rule aGroup.asubg-test*[*of R I*], *assumption*+)
done

lemma (**in** *Ring*) *ideal-condition1*:⟦$I \subseteq$ *carrier R*; $I \neq \{\}$;
∀ x∈I. ∀ y∈I. $x \pm y \in I$; ∀ r∈*carrier R*. ∀ x∈I. $r \cdot_r x \in I$ ⟧ ⟹ *ideal R I*

apply (*rule ideal-condition*[*of I*], *assumption*+)
apply (*rule ballI*)+
apply (*cut-tac ring-is-ag*,
  cut-tac ring-one,
  frule aGroup.ag-mOp-closed[*of R* 1_r], *assumption*+)
apply (*frule-tac* $x = -_a\ 1_r$ **in** *bspec*, *assumption*+,
  thin-tac $\forall r{\in}carrier\ R.\ \forall x{\in}I.\ r \cdot_r x \in I$,
  rotate-tac -1,
  frule-tac $x = y$ **in** *bspec*, *assumption*,
  thin-tac $\forall x{\in}I.\ (-_a\ 1_r) \cdot_r x \in I$)
apply (*frule-tac* $c = y$ **in** *subsetD*[*of I carrier R*], *assumption*+,
  simp add:*ring-times-minusl*[*THEN sym*], *simp add*:*ideal-pOp-closed*)
done

lemma (**in** *Ring*) *zero-ideal*:*ideal R* {**0**}
apply (*cut-tac ring-is-ag*)
apply (*rule ideal-condition1*)
apply (*simp add*:*ring-zero*)
apply *simp*
apply *simp*
apply (*cut-tac ring-zero*, *simp add*:*aGroup.ag-l-zero*)
apply *simp*
apply (*rule ballI*, *simp add*:*ring-times-x-0*)
done

lemma (**in** *Ring*) *whole-ideal*:*ideal R* (*carrier R*)
apply (*rule ideal-condition1*)
apply *simp*
apply (*cut-tac ring-zero*, *blast*)
apply (*cut-tac ring-is-ag*,
  simp add:*aGroup.ag-pOp-closed*,
  simp add:*ring-tOp-closed*)
done

lemma (**in** *Ring*) *ideal-inc-one*:$[\![$*ideal R I*; $1_r \in I\ ]\!] \Longrightarrow I = carrier\ R$
apply (*rule equalityI*)
apply (*simp add*:*ideal-subset1*)
apply (*rule subsetI*,
  frule-tac $r = x$ **in** *ideal-ring-multiple*[*of I* 1_r], *assumption*+,
  simp add:*ring-r-one*)
done

lemma (**in** *Ring*) *ideal-inc-one1*:*ideal R I* $\Longrightarrow$
  $(1_r \in I) = (I = carrier\ R)$
apply (*rule iffI*)
apply (*simp add*:*ideal-inc-one*)
apply (*frule sym*, *thin-tac I* = *carrier R*,
  cut-tac ring-one, *simp*)
done

definition
Unit :: - ⇒ *'a* ⇒ *bool* **where**
Unit R a ⟷ *a* ∈ *carrier R* ∧ (∃ *b*∈*carrier R*. *a* ·$_{r\,R}$ *b* = 1$_{r\,R}$)

lemma (**in** *Ring*) *ideal-inc-unit*:⟦*ideal R I*; *a* ∈ *I*; *Unit R a*⟧ ⟹ 1$_r$ ∈ *I*
by (*simp add*:*Unit-def*, *erule conjE*, *erule bexE*,
frule-tac r = *b* **in** *ideal-ring-multiple1*[*of I a*], *assumption*+,
simp)

lemma (**in** *Ring*) *proper-ideal*:⟦*ideal R I*; 1$_r$ ∉ *I*⟧ ⟹ *I* ≠ *carrier R*
apply (*rule contrapos-pp*, *simp*+)
apply (*simp add*: *ring-one*)
done

lemma (**in** *Ring*) *ideal-inc-unit1*:⟦*a* ∈ *carrier R*; *Unit R a*; *ideal R I*; *a* ∈ *I*⟧
⟹ *I* = *carrier R*
apply (*frule ideal-inc-unit*[*of I a*], *assumption*+)
apply (*rule ideal-inc-one*[*of I*], *assumption*+)
done

lemma (**in** *Ring*) *int-ideal*:⟦*ideal R I*; *ideal R J*⟧ ⟹ *ideal R* (*I* ∩ *J*)
apply (*rule ideal-condition1*)
apply (*frule ideal-subset1*[*of I*], *frule ideal-subset1*[*of J*])
apply *blast*
apply (*frule ideal-zero*[*of I*], *frule ideal-zero*[*of J*], *blast*)

apply ((*rule ballI*)+, *simp*, (*erule conjE*)+,
simp add:*ideal-pOp-closed*)
apply ((*rule ballI*)+, *simp*, (*erule conjE*)+)
apply (*simp add*:*ideal-ring-multiple*)
done

definition
ideal-prod::[-, *'a set*, *'a set*] ⇒ *'a set* (**infix** ◇$_{r1}$ *90*) **where**
ideal-prod R I J == ⋂ {*L*. *ideal R L* ∧
{*x*.(∃ *i*∈*I*. ∃ *j*∈*J*. *x* = *i* ·$_{r\,R}$ *j*)} ⊆ *L*}

lemma (**in** *Ring*) *set-sum-mem*:⟦*a* ∈ *I*; *b* ∈ *J*; *I* ⊆ *carrier R*; *J* ⊆ *carrier R*⟧ ⟹
a ± *b* ∈ *I* ∓ *J*
apply (*cut-tac ring-is-ag*)
apply (*simp add*:*aGroup.set-sum*, *blast*)
done

lemma (**in** *Ring*) *sum-ideals*:⟦*ideal R I1*; *ideal R I2*⟧ ⟹ *ideal R* (*I1* ∓ *I2*)
apply (*cut-tac ring-is-ag*)
apply (*frule ideal-subset1*[*of I1*], *frule ideal-subset1*[*of I2*])
apply (*rule ideal-condition1*)
apply (*rule subsetI*, *simp add*:*aGroup.set-sum*, (*erule bexE*)+)

apply (*frule-tac h = h* **in** *ideal-subset*[*of I1*], *assumption*+,
frule-tac h = k **in** *ideal-subset*[*of I2*], *assumption*+,
cut-tac ring-is-ag,
simp add:*aGroup.ag-pOp-closed*)
apply (*frule ideal-zero*[*of I1*], *frule ideal-zero*[*of I2*],
frule set-sum-mem[*of* **0** *I1* **0** *I2*], *assumption*+, *blast*)
apply (*rule ballI*)+
apply (*simp add*:*aGroup.set-sum*, (*erule bexE*)+, *simp*)
apply (*rename-tac x y i ia j ja*)
apply (*frule-tac h = i* **in** *ideal-subset*[*of I1*], *assumption*+,
frule-tac h = ia **in** *ideal-subset*[*of I1*], *assumption*+,
frule-tac h = j **in** *ideal-subset*[*of I2*], *assumption*+,
frule-tac h = ja **in** *ideal-subset*[*of I2*], *assumption*+)
apply (*subst aGroup.pOp-assocTr43*, *assumption*+)
apply (*frule-tac x = j* **and** *y = ia* **in** *aGroup.ag-pOp-commute*[*of R*],
assumption+, *simp*)
apply (*subst aGroup.pOp-assocTr43*[*THEN sym*], *assumption*+)
apply (*frule-tac x = i* **and** *y = ia* **in** *ideal-pOp-closed*[*of I1*], *assumption*+,
frule-tac x = j **and** *y = ja* **in** *ideal-pOp-closed*[*of I2*], *assumption*+,
blast)
apply (*rule ballI*)+
apply (*simp add*:*aGroup.set-sum*, (*erule bexE*)+, *simp*)
apply (*rename-tac r x i j*)
apply (*frule-tac h = i* **in** *ideal-subset*[*of I1*], *assumption*+,
frule-tac h = j **in** *ideal-subset*[*of I2*], *assumption*+)
apply (*simp add*:*ring-distrib1*)
apply (*frule-tac x = i* **and** *r = r* **in** *ideal-ring-multiple*[*of I1*], *assumption*+,
frule-tac x = j **and** *r = r* **in** *ideal-ring-multiple*[*of I2*], *assumption*+,
blast)
done

lemma (**in** *Ring*) *sum-ideals-la1*:⟦*ideal R I1*; *ideal R I2*⟧ ⟹ *I1* ⊆ (*I1* ∓ *I2*)
apply (*cut-tac ring-is-ag*)
apply (*rule subsetI*)
apply (*frule ideal-zero*[*of I2*],
frule-tac h = x **in** *ideal-subset*[*of I1*], *assumption*+,
frule-tac x = x **in** *aGroup.ag-r-zero*[*of R*], *assumption*+)
apply (*subst aGroup.set-sum*, *assumption*,
simp add:*ideal-subset1*, *simp add*:*ideal-subset1*, *simp*,
frule sym, *thin-tac x* ± **0** = *x*, *blast*)
done

lemma (**in** *Ring*) *sum-ideals-la2*:⟦*ideal R I1*; *ideal R I2* ⟧ ⟹ *I2* ⊆ (*I1* ∓ *I2*)
apply (*cut-tac ring-is-ag*)
apply (*rule subsetI*)
apply (*frule ideal-zero*[*of I1*],
frule-tac h = x **in** *ideal-subset*[*of I2*], *assumption*+,
frule-tac x = x **in** *aGroup.ag-l-zero*[*of R*], *assumption*+)
apply (*subst aGroup.set-sum*, *assumption*,

simp add:*ideal-subset1*, *simp add*:*ideal-subset1*, *simp*,
frule sym, *thin-tac* $\mathbf{0} \pm x = x$, *blast*)
done

lemma (**in** *Ring*) *sum-ideals-cont*:$[\![$*ideal R I*; $A \subseteq I$; $B \subseteq I$ $]\!] \Longrightarrow A \mp B \subseteq I$
apply (*cut-tac ring-is-ag*)
apply (*rule subsetI*)
apply (*frule ideal-subset1*[*of I*],
frule subset-trans[*of A I carrier R*], *assumption*+,
frule subset-trans[*of B I carrier R*], *assumption*+)
apply (*simp add*:*aGroup.set-sum*[*of R*], (*erule bexE*)+, *simp*)
apply (*frule-tac* $c = h$ **in** *subsetD*[*of A I*], *assumption*+,
frule-tac $c = k$ **in** *subsetD*[*of B I*], *assumption*+)
apply (*simp add*:*ideal-pOp-closed*)
done

lemma (**in** *Ring*) *ideals-set-sum*:$[\![$*ideal R A*; *ideal R B*; $x \in A \mp B$$]\!] \Longrightarrow$
$\exists h{\in}A.\ \exists k{\in}B.\ x = h \pm k$
apply (*frule ideal-subset1*[*of A*],
frule ideal-subset1[*of B*])
apply (*cut-tac ring-is-ag*,
simp add:*aGroup.set-sum*)
done

definition
Rxa :: [-, $'a$] $\Rightarrow$ $'a$ *set* (**infixl** $\diamondsuit_p$ *200*) **where**
Rxa R a = $\{x.\ \exists r{\in}$*carrier R*. $x = (r \cdot_{r\,R} a)\}$

lemma (**in** *Ring*) *a-in-principal*:$a \in$ *carrier R* $\Longrightarrow a \in$ *Rxa R a*
apply (*cut-tac ring-one*,
frule ring-l-one[*THEN sym*, *of a*])
apply (*simp add*:*Rxa-def*, *blast*)
done

lemma (**in** *Ring*) *principal-ideal*:$a \in$ *carrier R* $\Longrightarrow$ *ideal R* (*Rxa R a*)
apply (*rule ideal-condition1*)
apply (*rule subsetI*,
simp add:*Rxa-def*, *erule bexE*, *simp add*:*ring-tOp-closed*)
apply (*frule a-in-principal*[*of a*], *blast*)
apply ((*rule ballI*)+,
simp add:*Rxa-def*, (*erule bexE*)+, *simp*,
subst ring-distrib2[*THEN sym*], *assumption*+,
cut-tac ring-is-ag,
frule-tac $x = r$ **and** $y = ra$ **in** *aGroup.ag-pOp-closed*, *assumption*+,
blast)
apply ((*rule ballI*)+,
simp add:*Rxa-def*, (*erule bexE*)+, *simp*,
simp add:*ring-tOp-assoc*[*THEN sym*])
apply (*frule-tac* $x = r$ **and** $y = ra$ **in** *ring-tOp-closed*, *assumption*, *blast*)

done

lemma (**in** *Ring*) *rxa-in-Rxa*:⟦*a* ∈ *carrier R*; *r* ∈ *carrier R*⟧ ⟹
r ·$_r$ *a* ∈ *Rxa R a*
by (*simp add*:*Rxa-def*, *blast*)

lemma (**in** *Ring*) *Rxa-one*:*Rxa R* 1$_r$ = *carrier R*
apply (*rule equalityI*)
apply (*rule subsetI*, *simp add*:*Rxa-def*, *erule bexE*)
apply (*simp add*:*ring-r-one*)

apply (*rule subsetI*, *simp add*:*Rxa-def*)
apply (*frule-tac t* = *x* **in** *ring-r-one*[*THEN sym*], *blast*)
done

lemma (**in** *Ring*) *Rxa-zero*:*Rxa R* **0** = {**0**}
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*simp add*:*Rxa-def*, *erule bexE*, *simp add*:*ring-times-x-0*)
apply (*rule subsetI*)
apply (*simp add*:*Rxa-def*)
apply (*cut-tac ring-zero*,
frule ring-times-x-0[*THEN sym*, *of* **0**], *blast*)
done

lemma (**in** *Ring*) *Rxa-nonzero*:⟦*a* ∈ *carrier R*; *a* ≠ **0**⟧ ⟹ *Rxa R a* ≠ {**0**}
apply (*rule contrapos-pp*, *simp*+)
apply (*frule a-in-principal*[*of a*])
apply *simp*
done

lemma (**in** *Ring*) *ideal-cont-Rxa*:⟦*ideal R I*; *a* ∈ *I*⟧ ⟹ *Rxa R a* ⊆ *I*
apply (*rule subsetI*)
apply (*simp add*:*Rxa-def*, *erule bexE*, *simp*)
apply (*simp add*:*ideal-ring-multiple*)
done

lemma (**in** *Ring*) *Rxa-mult-smaller*:⟦ *a* ∈ *carrier R*; *b* ∈ *carrier R*⟧ ⟹
Rxa R (*a* ·$_r$ *b*) ⊆ *Rxa R b*
apply (*frule rxa-in-Rxa*[*of b a*], *assumption*,
frule principal-ideal[*of b*])
apply (*rule ideal-cont-Rxa*[*of R* ◇$_p$ *b a* ·$_r$ *b*], *assumption*+)
done

lemma (**in** *Ring*) *id-ideal-psub-sum*:⟦*ideal R I*; *a* ∈ *carrier R*; *a* ∉ *I*⟧ ⟹
I ⊂ *I* ∓ *Rxa R a*
apply (*cut-tac ring-is-ag*)
apply (*simp add*:*psubset-eq*)
apply (*frule principal-ideal*)

apply (*rule conjI*)
apply (*rule sum-ideals-la1*, *assumption*+)
apply (*rule contrapos-pp*) **apply** *simp*+
apply (*frule sum-ideals-la2*[*of I Rxa R a*], *assumption*+)
apply (*frule a-in-principal*[*of a*],
frule subsetD[*of Rxa R a I* $\mp$ *Rxa R a a*], *assumption*+)
apply *simp*
done

lemma (**in** *Ring*) *mul-two-principal-idealsTr*:⟦$a \in$ *carrier R*; $b \in$ *carrier R*;
$x \in$ *Rxa R a*; $y \in$ *Rxa R b*⟧ $\Longrightarrow \exists\, r{\in}$*carrier R*. $x \cdot_r y = r \cdot_r (a \cdot_r b)$
apply (*simp add*:*Rxa-def*, (*erule bexE*)+)
apply *simp*
apply (*frule-tac* $x = ra$ **and** $y = b$ **in** *ring-tOp-closed*, *assumption*+)
apply (*simp add*:*ring-tOp-assoc*)
apply (*simp add*:*ring-tOp-assoc*[*THEN sym*, *of a - b*])
apply (*simp add*:*ring-tOp-commute*[*of a*], *simp add*:*ring-tOp-assoc*)
apply (*frule-tac* $x = a$ **and** $y = b$ **in** *ring-tOp-closed*, *assumption*+,
thin-tac ra $\cdot_r$ $b \in$ *carrier R*,
simp add:*ring-tOp-assoc*[*THEN sym*, *of - - a* $\cdot_r$ *b*],
frule-tac $x = r$ **and** $y = ra$ **in** *ring-tOp-closed*, *assumption*+)
apply (*simp add*:*ring-tOp-commute*[*of b a*])
apply *blast*
done

primrec *sum-pr-ideals*::[(*'a*, *'m*) *Ring-scheme*, *nat* $\Rightarrow$ *'a*, *nat*] $\Rightarrow$ *'a set*
where
sum-pr0: *sum-pr-ideals R f 0* = *Rxa R* (*f 0*)
| *sum-prn*: *sum-pr-ideals R f* (*Suc n*) =
(*Rxa R* (*f* (*Suc n*))) $\mp_R$ (*sum-pr-ideals R f n*)

lemma (**in** *Ring*) *sum-of-prideals0*:
$\forall f.\ (\forall\, l \le n.\ f\ l \in$ *carrier R*) $\longrightarrow$ *ideal R* (*sum-pr-ideals R f n*)
apply (*induct-tac n*)
apply (*rule allI*) **apply** (*rule impI*)
apply *simp*
apply (*rule Ring.principal-ideal*, *rule Ring-axioms*, *assumption*)

apply (*rule allI*, *rule impI*)
apply (*frule-tac* $x = f$ **in** *spec*,
thin-tac $\forall f.\ (\forall\, l \le n.\ f\ l \in$ *carrier R*) $\longrightarrow$
ideal R (*sum-pr-ideals R f n*))
apply (*cut-tac* $n = n$ **in** *Nsetn-sub-mem1*, *simp*)
apply (*cut-tac* $a = f$ (*Suc n*) **in** *principal-ideal*,
simp)
apply (*rule-tac ?I1.0* = *Rxa R* (*f* (*Suc n*)) **and**
?I2.0 = *sum-pr-ideals R f n* **in** *Ring.sum-ideals*, *rule Ring-axioms*, *assumption*+)

done

lemma (**in** *Ring*) *sum-of-prideals*:⟦∀ *l* ≤ *n*. *f l* ∈ *carrier R*⟧ ⟹
ideal R (*sum-pr-ideals R f n*)
apply (*simp add*:*sum-of-prideals0*)
done

later, we show *sum-pr-ideals* is the least ideal containing {*f 0*, *f 1*,. . ., *f n*}

lemma (**in** *Ring*) *sum-of-prideals1*:∀ *f*. (∀ *l* ≤ *n*. *f l* ∈ *carrier R*) ⟶
f ‘ {*i*. *i* ≤ *n*} ⊆ (*sum-pr-ideals R f n*)
apply (*induct-tac n*)
apply (*rule allI*, *rule impI*)
apply (*simp*, *simp add*:*a-in-principal*)

apply (*rule allI*, *rule impI*)
apply (*frule-tac a* = *f* **in** *forall-spec*,
thin-tac ∀ *f*. (∀ *l* ≤ *n*. *f l* ∈ *carrier R*) ⟶
f ‘ {*i*. *i* ≤ *n*} ⊆ *sum-pr-ideals R f n*)
apply (*rule allI*, *cut-tac n* = *n* **in** *Nset-un*, *simp*)

apply (*subst Nset-un*)
apply (*cut-tac A* = {*i*. *i* ≤ (*Suc n*)} **and** *f* = *f* **and** *B* = *carrier R* **and**
?A1.0 = {*i*. *i* ≤ *n*} **and** *?A2.0* = {*Suc n*} **in** *im-set-un1*,
simp, *rule Nset-un*)
apply (*thin-tac* ∀ *f*. (∀ *l*≤*n*. *f l* ∈ *carrier R*) ⟶
f ‘ {*i*. *i* ≤ *n*} ⊆ *sum-pr-ideals R f n*,
simp)
apply (*cut-tac n* = *n* **and** *f* = *f* **in** *sum-of-prideals*,
cut-tac n = *n* **in** *Nsetn-sub-mem1*, *simp*)
apply (*cut-tac a* = *f* (*Suc n*) **in** *principal-ideal*, *simp*)
apply (*frule-tac ?I1.0* = *Rxa R* (*f* (*Suc n*)) **and** *?I2.0* = *sum-pr-ideals R f n*
in *sum-ideals-la1*, *assumption*+,
cut-tac a = *f* (*Suc n*) **in** *a-in-principal*, *simp*,
frule-tac A = *R* ♢$_p$ *f* (*Suc n*) **and**
B = *R* ♢$_p$ *f* (*Suc n*) ∓ *sum-pr-ideals R f n* **and** *c* = *f* (*Suc n*) **in**
subsetD, *simp*+)
apply (*frule-tac ?I1.0* = *Rxa R* (*f* (*Suc n*)) **and**
?I2.0 = *sum-pr-ideals R f n* **in** *sum-ideals-la2*, *assumption*+)
apply (*rule-tac A* = *f* ‘ {*j*. *j* ≤ *n*} **and** *B* = *sum-pr-ideals R f n* **and**
C = *Rxa R* (*f* (*Suc n*)) ∓ *sum-pr-ideals R f n* **in** *subset-trans*,
assumption+)
done

lemma (**in** *Ring*) *sum-of-prideals2*:∀ *l* ≤ *n*. *f l* ∈ *carrier R*
⟹ *f* ‘ {*i*. *i* ≤ *n*} ⊆ (*sum-pr-ideals R f n*)
apply (*simp add*:*sum-of-prideals1*)
done

lemma (**in** *Ring*) *sum-of-prideals3*:*ideal R I* $\Longrightarrow$
$\forall f.\ (\forall\, l \leq n.\ f\ l \in \mathit{carrier}\ R) \wedge (f\ `\ \{i.\ i \leq n\} \subseteq I) \longrightarrow$
(*sum-pr-ideals R f n* $\subseteq I$)
apply (*induct-tac n*)
apply (*rule allI*, *rule impI*, *erule conjE*)
apply *simp*
apply (*rule ideal-cont-Rxa*[*of I*], *assumption*+)

apply (*rule allI*, *rule impI*, *erule conjE*)
apply (*frule-tac a* = *f* **in** *forall-spec*,
thin-tac $\forall f.\ (\forall\, l \leq n.\ f\ l \in \mathit{carrier}\ R) \wedge f\ `\{i.\ i \leq n\} \subseteq I \longrightarrow$
sum-pr-ideals R f n $\subseteq I$)
apply (*simp add*:*Nset-un*)
apply (*thin-tac* $\forall f.\ (\forall\, l \leq n.\ f\ l \in \mathit{carrier}\ R) \wedge f\ `\ \{i.\ i \leq n\} \subseteq I \longrightarrow$
sum-pr-ideals R f n $\subseteq I$)
apply (*frule-tac x* = *Suc n* **in** *spec*,
thin-tac $\forall\, l \leq (\mathit{Suc}\ n).\ f\ l \in \mathit{carrier}\ R$, *simp*)
apply (*cut-tac a* = *Suc n* **and** $A = \{i.\ i \leq \mathit{Suc}\ n\}$ **and**
f = *f* **in** *mem-in-image2*, *simp*)
apply (*frule-tac A* = *f* ` $\{i.\ i \leq \mathit{Suc}\ n\}$ **and** *B* = *I* **and** *c* = *f* (*Suc n*) **in**
subsetD, *assumption*+)
apply (*rule-tac A* = *Rxa R* (*f* (*Suc n*)) **and** *B* = *sum-pr-ideals R f n* **in**
sum-ideals-cont[*of I*], *assumption*)
apply (*rule ideal-cont-Rxa*[*of I*], *assumption*+)
done

lemma (**in** *Ring*) *sum-of-prideals4*:⟦*ideal R I*; $\forall\, l \leq n.\ f\ l \in \mathit{carrier}\ R$;
(*f* ` $\{i.\ i \leq n\} \subseteq I$)⟧ $\Longrightarrow$ *sum-pr-ideals R f n* $\subseteq I$
apply (*simp add*:*sum-of-prideals3*)
done

lemma *ker-ideal*:⟦*Ring A*; *Ring R*; *f* $\in$ *rHom A R*⟧ $\Longrightarrow$ *ideal A* ($ker_{A,R}\ f$)
apply (*frule Ring.ring-is-ag*[*of A*], *frule Ring.ring-is-ag*[*of R*])
apply (*rule Ring.ideal-condition1*, *assumption*+)
apply (*rule subsetI*,
simp add:*ker-def*)
apply (*simp add*:*rHom-def*, *frule conjunct1*)
apply (*frule ker-inc-zero*[*of A R f*], *assumption*+, *blast*)

apply (*rule ballI*)+
apply (*simp add*:*ker-def*, (*erule conjE*)+)
apply (*simp add*:*aGroup.ag-pOp-closed*)
apply (*simp add*:*rHom-def*, *frule conjunct1*,
simp add:*aHom-add*,
frule Ring.ring-zero[*of R*],
simp add:*aGroup.ag-l-zero*)
apply (*rule ballI*)+
apply (*simp add*:*ker-def*, (*erule conjE*)+)
apply (*simp add*:*Ring.ring-tOp-closed*)

apply (*simp add*:*rHom-tOp*)
apply (*frule-tac a* = *r* **in** *rHom-mem*[*of f A R*], *assumption*+,
simp add:*Ring.ring-times-x-0*)
done

4.3.1 Ring of integers

definition
Zr :: *int Ring* **where**
Zr = (| *carrier* = *Zset*, *pop* = λ*n*∈*Zset*. λ*m*∈*Zset*. (*m* + *n*),
mop = λ*l*∈*Zset*. −*l*, *zero* = *0*, *tp* = λ*m*∈*Zset*. λ*n*∈*Zset*. *m* ∗ *n*, *un* = *1* |)

lemma *ring-of-integers*:*Ring Zr*
apply (*simp add*:*Ring-def*)
apply (*rule conjI*)
apply (*simp add*:*Zr-def Zset-def*)
apply (*rule conjI*)
apply (*simp add*:*Zr-def Zset-def*)
apply (*rule conjI*)
apply (*rule allI*, *rule impI*)+
apply (*simp add*:*Zr-def Zset-def*)
apply (*rule conjI*)
apply (*simp add*:*Zr-def Zset-def*)
apply (*rule conjI*,
rule allI, *rule impI*, *simp add*:*Zr-def Zset-def*)
apply (*rule conjI*, *simp add*:*Zr-def Zset-def*)
apply (*rule conjI*,
rule allI, *rule impI*, *simp add*:*Zr-def Zset-def*)
apply (*rule conjI*)
apply (*simp add*:*Zr-def Zset-def*)
apply (*rule conjI*,
(*rule allI*, *rule impI*)+, *simp add*:*Zr-def Zset-def*)
apply (*rule conjI*,
(*rule allI*, *rule impI*)+, *simp add*:*Zr-def Zset-def*)
apply (*rule conjI*)
apply (*simp add*:*Zr-def Zset-def*)
apply (*rule conjI*,
(*rule allI*, *rule impI*)+, *simp add*:*Zr-def Zset-def*)
apply (*simp add*: *distrib-left*)
apply (*rule allI*, *rule impI*)
apply (*simp add*:*Zr-def Zset-def*)
done

lemma *Zr-zero*:$\mathbf{0}_{Zr} = 0$
by (*simp add*:*Zr-def*)

lemma *Zr-one*:$1_{r\,Zr} = 1$
by (*simp add*:*Zr-def*)

lemma *Zr-minus*:$-_{a\,Zr}\ n = -\ n$
by (*simp add*:*Zr-def Zset-def*)

lemma *Zr-add*:$n \pm_{Zr} m = n + m$
by (*simp add*:*Zr-def Zset-def*)

lemma *Zr-times*:$n \cdot_{r\,Zr} m = n * m$
by (*simp add*:*Zr-def Zset-def*)

definition
lev :: *int set* ⇒ *int* **where**
lev I = *Zleast* {*n*. *n* ∈ *I* ∧ 0 < *n*}

lemma *Zr-gen-Zleast*:⟦*ideal Zr I*; *I* ≠ {0::*int*}⟧ ⟹
Rxa Zr (*lev I*) = *I*
apply (*cut-tac ring-of-integers*)
apply (*simp add*:*lev-def*)
apply (*subgoal-tac* {*n*. *n* ∈ *I* ∧ 0 < *n*} ≠ {})
apply (*subgoal-tac* {*n*. *n* ∈ *I* ∧ 0 < *n*} ⊆ *Zset*)
apply (*subgoal-tac LB* {*n*. *n* ∈ *I* ∧ 0 < *n*} 0)
apply (*frule-tac A* = {*n*. *n* ∈ *I* ∧ 0 < *n*} **and** *n* = 0 **in** *Zleast*, *assumption*+)
apply (*erule conjE*)+
apply (*fold lev-def*)
defer
apply (*simp add*:*LB-def*)
apply (*simp add*:*Zset-def*)
apply (*frule Ring.ideal-zero*[*of Zr I*], *assumption*+, *simp add*:*Zr-zero*)
apply (*frule singleton-sub*[*of 0 I*])
apply (*frule sets-not-eq*[*of I* {0}], *assumption*+, *erule bexE*, *simp*)
apply (*case-tac* 0 < *a*, *blast*)
apply (*frule Ring.ring-one*[*of Zr*])
apply (*frule Ring.ring-is-ag*[*of Zr*],
frule aGroup.ag-mOp-closed[*of Zr* $1_{r\,Zr}$], *assumption*)
apply (*frule-tac x* = *a* **in** *Ring.ideal-ring-multiple*[*of Zr I* - $-_{a\,Zr}\ 1_{r\,Zr}$],
assumption+)
apply (*simp add*:*Zr-one Zr-minus*,
thin-tac ideal Zr I, *thin-tac Ring Zr*, *thin-tac* 1 ∈ *carrier Zr*,
thin-tac −1 ∈ *carrier Zr*, *thin-tac aGroup Zr*)
apply (*simp add*:*Zr-def Zset-def*)
apply (*subgoal-tac* 0 < − *a*, *blast*)
apply *arith*
apply (*thin-tac* {*n* ∈ *I*. 0 < *n*} ≠ {}, *thin-tac* {*n* ∈ *I*. 0 < *n*} ⊆ *Zset*,
thin-tac LB {*n* ∈ *I*. 0 < *n*} 0)

apply *simp*
apply (*erule conjE*)
apply (*frule Ring.ideal-cont-Rxa*[*of Zr I lev I*], *assumption*+)
apply (*rule equalityI*, *assumption*,
thin-tac Rxa Zr (*lev I*) ⊆ *I*)

apply (*rule subsetI*)
apply (*simp add*:*Rxa-def*, *simp add*:*Zr-times*)
apply (*cut-tac* $a = x$ **and** $b = lev\ I$ **in** *zmod-zdiv-equality*)
apply (*subgoal-tac* $x = (x\ div\ lev\ I) * (lev\ I)$,
subgoal-tac $x\ div\ lev\ I \in carrier\ Zr$, *blast*)
apply (*simp add*:*Zr-def Zset-def*)
apply (*subgoal-tac* $x\ mod\ lev\ I = 0$, *simp*)
apply (*subst mult.commute*, *assumption*)
apply (*subgoal-tac* $x\ mod\ lev\ I \in I$)
apply (*thin-tac* $x = lev\ I * (x\ div\ lev\ I) + x\ mod\ lev\ I$)
apply (*frule-tac* $a = x$ **in** *pos-mod-conj*[*of lev I*])
apply (*rule contrapos-pp*, *simp*+)
apply (*erule conjE*)
apply (*frule-tac* $a = x\ mod\ (lev\ I)$ **in** *forall-spec*)
apply *simp* **apply** *arith*
apply (*frule-tac* $r = x\ div\ (lev\ I)$ **in**
Ring.ideal-ring-multiple1[*of Zr I lev I*], *assumption*+,
simp add:*Zr-def Zset-def*)
apply (*frule sym*, *thin-tac* $x = lev\ I * (x\ div\ lev\ I) + x\ mod\ lev\ I$)
apply (*rule-tac* $a = lev\ I * (x\ div\ lev\ I)$ **and** $b = x\ mod\ lev\ I$ **in**
Ring.ideal-ele-sumTr1[*of Zr I*], *assumption*+)
apply (*simp add*:*Zr-def Zset-def*)
apply (*simp add*:*Zr-def Zset-def*)
apply (*subst Zr-add*)
apply *simp*
apply (*simp add*:*Zr-times*)
done

lemma *Zr-pir*:*ideal Zr I* $\Longrightarrow \exists n.\ Rxa\ Zr\ n = I$
apply (*case-tac* $I = \{(0::int)\}$)
apply (*subgoal-tac Rxa Zr 0* $= I$) **apply** *blast*
apply (*rule equalityI*)
apply (*rule subsetI*) **apply** (*simp add*:*Rxa-def*)
apply (*simp add*:*Zr-def Zset-def*)
apply (*rule subsetI*)
apply (*simp add*:*Rxa-def Zr-def Zset-def*)
apply (*frule Zr-gen-Zleast* [*of I*], *assumption*+)
apply *blast*
done

4.4 Quotient rings

lemma (**in** *Ring*) *mem-set-ar-cos*:$[\![$*ideal R I*; $a \in carrier\ R]\!] \Longrightarrow$
$a \uplus_R I \in set\text{-}ar\text{-}cos\ R\ I$
by (*simp add*:*set-ar-cos-def*, *blast*)

lemma (**in** *Ring*) *I-in-set-ar-cos*:*ideal R I* $\Longrightarrow I \in set\text{-}ar\text{-}cos\ R\ I$
apply (*cut-tac ring-is-ag*,
frule ideal-asubg[*of I*],

rule aGroup.unit-in-set-ar-cos, *assumption*+)
done

lemma (**in** *Ring*) *ar-coset-same1*:⟦*ideal R I*; *a* ∈ *carrier R*; *b* ∈ *carrier R*;
b ± (−$_a$ *a*) ∈ *I* ⟧ ⟹ *a* ⊎$_R$ *I* = *b* ⊎$_R$ *I*
apply (*cut-tac ring-is-ag*)
apply (*frule aGroup.b-ag-group*[*of R*])
apply (*simp add*:*ideal-def asubGroup-def*) **apply** (*erule conjE*)
apply (*frule aGroup.ag-carrier-carrier*[*THEN sym*, *of R*])
apply *simp*
apply (*frule Group.rcs-eq*[*of b-ag R I a b*], *assumption*+)
apply (*frule aGroup.agop-gop* [*of R*])
apply (*frule aGroup.agiop-giop*[*of R*]) **apply** *simp*
apply (*simp add*:*ar-coset-def rcs-def*)
done

lemma (**in** *Ring*) *ar-coset-same2*:⟦*ideal R I*; *a* ∈ *carrier R*; *b* ∈ *carrier R*;
a ⊎$_R$ *I* = *b* ⊎$_R$ *I*⟧ ⟹ *b* ± (−$_a$ *a*) ∈ *I*
apply (*cut-tac ring-is-ag*)
apply (*simp add*:*ar-coset-def*)
apply (*frule aGroup.b-ag-group*[*of R*])
apply (*simp add*:*ideal-def asubGroup-def*, *frule conjunct1*, *fold asubGroup-def*,
fold ideal-def, *simp add*:*asubGroup-def*)
apply (*subgoal-tac a* ∈ *carrier* (*b-ag R*),
subgoal-tac b ∈ *carrier* (*b-ag R*))
apply (*simp add*:*Group.rcs-eq*[*THEN sym*, *of b-ag R I a b*])
apply (*frule aGroup.agop-gop* [*of R*])
apply (*frule aGroup.agiop-giop*[*of R*]) **apply** *simp*
apply (*simp add*:*b-ag-def*)+
done

lemma (**in** *Ring*) *ar-coset-same3*:⟦*ideal R I*; *a* ∈ *carrier R*; *a* ⊎$_R$ *I* = *I*⟧ ⟹
a∈*I*
apply (*cut-tac ring-is-ag*)
apply (*simp add*:*ar-coset-def*)
apply (*rule Group.rcs-fixed* [*of b-ag R I a*])
apply (*rule aGroup.b-ag-group*, *assumption*)
apply (*simp add*:*ideal-def asubGroup-def*)
apply (*simp add*:*b-ag-def*)
apply *assumption*
done

lemma (**in** *Ring*) *ar-coset-same3-1*:⟦*ideal R I*; *a* ∈ *carrier R*; *a* ∉ *I*⟧ ⟹
a ⊎$_R$ *I* ≠ *I*
apply (*rule contrapos-pp*, *simp*+)
apply (*simp add*:*ar-coset-same3*)
done

lemma (**in** *Ring*) *ar-coset-same4*:⟦*ideal R I*; *a* ∈ *I*⟧ ⟹

$$a \uplus_R I = I$$

apply (*cut-tac ring-is-ag*)
apply (*frule ideal-subset*[*of I a*], *assumption*+)
apply (*simp add*:*ar-coset-def*)
apply (*rule Group.rcs-Unit2* [*of b-ag R Ia*])
apply (*rule aGroup.b-ag-group*, *assumption*)
apply (*simp add*:*ideal-def asubGroup-def*)
apply *assumption*
done

lemma (**in** *Ring*) *ar-coset-same4-1*:⟦*ideal R I*; $a \uplus_R I \neq I$⟧ $\Longrightarrow a \notin I$
apply (*rule contrapos-pp*, *simp*+)
apply (*simp add*:*ar-coset-same4*)
done

lemma (**in** *Ring*) *belong-ar-coset1*:⟦*ideal R I*; $a \in$ *carrier R*; $x \in$ *carrier R*;
$x \pm (-_a\ a) \in I$⟧ $\Longrightarrow\ x \in a \uplus_R I$
apply (*frule ar-coset-same1* [*of I a x*], *assumption*+)
apply (*subgoal-tac* $x \in x \uplus_R I$)
apply *simp*
apply (*cut-tac ring-is-ag*)
apply (*subgoal-tac carrier R* = *carrier* (*b-ag R*))
apply (*frule aGroup.agop-gop*[*THEN sym*, *of R*])
apply (*frule aGroup.agiop-giop* [*THEN sym*, *of R*])
apply (*simp add*:*ar-coset-def*)
apply (*simp add*:*ideal-def asubGroup-def*)

apply (*rule Group.a-in-rcs* [*of b-ag R I x*])
apply (*simp add*: *aGroup.b-ag-group*)
apply *simp*
apply *simp*
apply (*simp add*:*b-ag-def*)
done

lemma (**in** *Ring*) *a-in-ar-coset*:⟦*ideal R I*; $a \in$ *carrier R*⟧ $\Longrightarrow a \in a \uplus_R I$
apply (*rule belong-ar-coset1*, *assumption*+)
apply (*cut-tac ring-is-ag*)
apply (*simp add*:*aGroup.ag-r-inv1*)
apply (*simp add*:*ideal-zero*)
done

lemma (**in** *Ring*) *ar-coset-subsetD*:⟦*ideal R I*; $a \in$ *carrier R*; $x \in a \uplus_R I$ ⟧ $\Longrightarrow$
$x \in$ *carrier R*
apply (*subgoal-tac carrier R* = *carrier* (*b-ag R*))
apply (*cut-tac ring-is-ag*)
apply (*frule aGroup.agop-gop* [*THEN sym*, *of R*])
apply (*frule aGroup.agiop-giop* [*THEN sym*, *of R*])
apply (*simp add*:*ar-coset-def*)
apply (*simp add*:*ideal-def asubGroup-def*)

apply (*rule Group.rcs-subset-elem*[*of b-ag R I a x*])
apply (*simp add*:*aGroup.b-ag-group*)
apply *simp*
apply *assumption*+
apply (*simp add*:*b-ag-def*)
done

lemma (**in** *Ring*) *ar-cos-mem*:⟦*ideal R I*; *a* ∈ *carrier R*⟧ ⟹
a ⊎$_R$ *I* ∈ *set-rcs* (*b-ag R*) *I*
apply (*cut-tac ring-is-ag*)
apply (*simp add*:*set-rcs-def ar-coset-def*)
apply (*frule aGroup.ag-carrier-carrier*[*THEN sym*, *of R*]) **apply** *simp*
apply *blast*
done

lemma (**in** *Ring*) *mem-ar-coset1*:⟦*ideal R I*; *a* ∈ *carrier R*; *x* ∈ *a* ⊎$_R$ *I*⟧ ⟹
∃ *h*∈*I*. *h* ± *a* = *x*
apply (*cut-tac ring-is-ag*)
apply (*frule aGroup.ag-carrier-carrier*[*THEN sym*, *of R*])
apply (*frule aGroup.agop-gop* [*THEN sym*, *of R*])
apply (*frule aGroup.agiop-giop* [*THEN sym*, *of R*])
apply (*simp add*:*ar-coset-def*)
apply (*simp add*:*ideal-def asubGroup-def*)
apply (*simp add*:*rcs-def*)
done

lemma (**in** *Ring*) *ar-coset-mem2*:⟦*ideal R I*; *a* ∈ *carrier R*; *x* ∈ *a* ⊎$_R$ *I*⟧ ⟹
∃ *h*∈*I*. *x* = *a* ± *h*
apply (*cut-tac ring-is-ag*)
apply (*frule mem-ar-coset1* [*of I a x*], *assumption*+)
apply (*erule bexE*,
frule-tac h = *h* **in** *ideal-subset*[*of I*], *assumption*+)
apply (*simp add*:*aGroup.ag-pOp-commute*[*of R* - *a*],
frule sym, *thin-tac a* ± *h* = *x*, *blast*)
done

lemma (**in** *Ring*) *belong-ar-coset2*:⟦*ideal R I*; *a* ∈ *carrier R*; *x* ∈ *a* ⊎$_R$ *I* ⟧
⟹ *x* ± (−$_a$ *a*) ∈ *I*
apply (*cut-tac ring-is-ag*)
apply (*frule mem-ar-coset1*, *assumption*+, *erule bexE*)
apply (*frule sym*, *thin-tac h* ± *a* = *x*, *simp*)
apply (*frule-tac h* = *h* **in** *ideal-subset*[*of I*], *assumption*)
apply (*frule aGroup.ag-mOp-closed*[*of R a*], *assumption*)
apply (*subst aGroup.ag-pOp-assoc*, *assumption*+,
simp add:*aGroup.ag-r-inv1*,
simp add:*aGroup.ag-r-zero*)
done

lemma (**in** *Ring*) *ar-c-top*: ⟦*ideal R I*; *a* ∈ *carrier R*; *b* ∈ *carrier R*⟧

$\Longrightarrow$ (*c-top* (*b-ag R*) *I* (*a* $\uplus_R$ *I*) (*b* $\uplus_R$ *I*)) = (*a* $\pm$ *b*) $\uplus_R$ *I*
apply (*cut-tac ring-is-ag*, *frule ideal-asubg*,
frule aGroup.asubg-nsubg[*of R I*], *assumption*,
frule aGroup.b-ag-group[*of R*])
apply (*simp add*:*ar-coset-def*)
apply (*subst Group.c-top-welldef*[*THEN sym*], *assumption*+)
apply (*simp add*:*aGroup.ag-carrier-carrier*)+
apply (*simp add*:*aGroup.agop-gop*)
done

Following lemma is not necessary to define a quotient ring. But it makes clear that the binary operation2 of the quotient ring is well defined.

lemma (**in** *Ring*) *quotient-ring-tr1*:⟦*ideal R I*; *a1* $\in$ *carrier R*; *a2* $\in$ *carrier R*;
b1 $\in$ *carrier R*; *b2* $\in$ *carrier R*;
a1 $\uplus_R$ *I* = *a2* $\uplus_R$ *I*; *b1* $\uplus_R$ *I* = *b2* $\uplus_R$ *I*⟧ $\Longrightarrow$
(*a1* $\cdot_r$ *b1*) $\uplus_R$ *I* = (*a2* $\cdot_r$ *b2*) $\uplus_R$ *I*
apply (*rule ar-coset-same1*, *assumption*+)
apply (*simp add*: *ring-tOp-closed*)+
apply (*frule ar-coset-same2* [*of I a1 a2*], *assumption*+)
apply (*frule ar-coset-same2* [*of I b1 b2*], *assumption*+)
apply (*frule ring-distrib4*[*of a2 b2 a1 b1*], *assumption*+)
apply *simp*
apply (*rule ideal-pOp-closed*[*of I*], *assumption*)
apply (*simp add*:*ideal-ring-multiple*, *simp add*:*ideal-ring-multiple1*)
done

definition
rcostOp :: [-, *'a set*] $\Rightarrow$ (['*a set*, *'a set*] $\Rightarrow$ *'a set*) **where**
rcostOp R I = ($\lambda X \in$(*set-rcs* (*b-ag R*) *I*). $\lambda Y \in$(*set-rcs* (*b-ag R*) *I*).
{*z*. $\exists$ *x* $\in$ *X*. $\exists$ *y* $\in$ *Y*. $\exists h \in I$. (*x* $\cdot_{r\,R}$ *y*) $\pm_R$ *h* = *z*})

lemma (**in** *Ring*) *rcostOp*:⟦*ideal R I*; *a* $\in$ *carrier R*; *b* $\in$ *carrier R*⟧ $\Longrightarrow$
rcostOp R I (*a* $\uplus_R$ *I*) (*b* $\uplus_R$ *I*) = (*a* $\cdot_r$ *b*) $\uplus_R$ *I*
apply (*cut-tac ring-is-ag*)
apply (*frule ar-cos-mem*[*of I a*], *assumption*+)
apply (*frule ar-cos-mem*[*of I b*], *assumption*+)
apply (*simp add*:*rcostOp-def*)
apply (*rule equalityI*)
apply (*rule subsetI*, *simp*) **apply** (*erule bexE*)+
apply (*rule belong-ar-coset1*, *assumption*+)
apply (*simp add*:*ring-tOp-closed*)
apply (*frule sym*, *thin-tac xa* $\cdot_r$ *y* $\pm$ *h* = *x*, *simp*)
apply (*rule aGroup.ag-pOp-closed*, *assumption*)
apply (*frule-tac x* = *xa* **in** *ar-coset-mem2*[*of I a*], *assumption*+,
frule-tac x = *y* **in** *ar-coset-mem2*[*of I b*], *assumption*+,
(*erule bexE*)+, *simp*)
apply (*rule ring-tOp-closed*, *rule aGroup.ag-pOp-closed*, *assumption*+,
simp add:*ideal-subset*)
apply (*rule aGroup.ag-pOp-closed*, *assumption*+, *simp add*:*ideal-subset*,

simp add:*ideal-subset*)
apply (*frule sym*, *thin-tac* $xa \cdot_r y \pm h = x$, *simp*)
apply (*frule-tac* $x = xa$ **in** *belong-ar-coset2*[*of I a*], *assumption*+,
frule-tac $x = y$ **in** *belong-ar-coset2*[*of I b*], *assumption*+)
apply (*frule-tac* $x = xa$ **in** *ar-coset-subsetD*[*of I a*], *assumption*+,
frule-tac $x = y$ **in** *ar-coset-subsetD*[*of I b*], *assumption*+)
apply (*subst aGroup.ag-pOp-commute*, *assumption*,
simp add:*ring-tOp-closed*, *simp add*:*ideal-subset*)
apply (*subst aGroup.ag-pOp-assoc*, *assumption*,
simp add:*ideal-subset*, *simp add*:*ring-tOp-closed*,
rule aGroup.ag-mOp-closed, *simp add*:*ring-tOp-closed*,
simp add:*ring-tOp-closed*)
apply (*rule ideal-pOp-closed*, *assumption*+)
apply (*rule-tac* $a = xa$ **and** $a' = y$ **and** $b = a$ **and** $b' = b$ **in** *times-modTr*,
assumption+)

apply (*rule subsetI*, *simp*)
apply (*frule-tac* $x = x$ **in** *ar-coset-mem2*[*of I a* $\cdot_r$ *b*],
simp add:*ring-tOp-closed*, *assumption*)
apply (*erule bexE*) **apply** *simp*
apply (*frule a-in-ar-coset*[*of I a*], *assumption*+,
frule a-in-ar-coset[*of I b*], *assumption*+)
apply *blast*
done

definition
qring :: [('*a*, '*m*) *Ring-scheme*, '*a set*] ⇒ ⦇ *carrier* :: '*a set set*,
pop :: ['*a set*, '*a set*] ⇒ '*a set*, *mop* :: '*a set* ⇒ '*a set*,
zero :: '*a set*, *tp* :: ['*a set*, '*a set*] ⇒ '*a set*, *un* :: '*a set* ⦈ **where**
qring R I = ⦇ *carrier* = *set-rcs* (*b-ag R*) *I*,
pop = *c-top* (*b-ag R*) *I*,
mop = *c-iop* (*b-ag R*) *I*,
zero = *I*,
tp = *rcostOp R I*,
un = $1_{rR} \uplus_R I$⦈

abbreviation
QRING (**infixl** '/$_r$ *200*) **where**
R /$_r$ *I* == *qring R I*

lemma (**in** *Ring*) *carrier-qring*:*ideal R I* ⟹
carrier (*qring R I*) = *set-rcs* (*b-ag R*) *I*
by (*simp add*:*qring-def*)

lemma (**in** *Ring*) *carrier-qring1*:*ideal R I* ⟹
carrier (*qring R I*) = *set-ar-cos R I*
apply (*cut-tac ring-is-ag*)
apply (*simp add*:*carrier-qring set-rcs-def set-ar-cos-def*)
apply (*simp add*:*ar-coset-def aGroup.ag-carrier-carrier*)

done

lemma (**in** *Ring*) *qring-ring*:*ideal R I* $\Longrightarrow$ *Ring* (*qring R I*)
apply (*cut-tac ring-is-ag*)
apply (*frule ideal-asubg*[*of I*],
frule aGroup.asubg-nsubg[*of R I*], *assumption*,
frule aGroup.b-ag-group[*of R*])
apply (*subst Ring-def*, *simp*)
apply (*rule conjI*)
apply (*rule Pi-I*)+
apply (*simp add*:*carrier-qring*, *simp add*:*set-rcs-def*, (*erule bexE*)+)
apply (*subst qring-def*, *simp*)
apply (*subst Group.c-top-welldef*[*THEN sym*, *of b-ag R I*], *assumption*+)
apply (*blast dest*: *Group.mult-closed*[*of b-ag R*])
apply (*rule conjI*)
apply (*rule allI*, *rule impI*)+
apply (*simp add*:*qring-def*)
apply (*simp add*:*Group.Qg-tassoc*[*of b-ag R I*])
apply (*rule conjI*)
apply (*rule allI*, *rule impI*)+
apply (*simp add*:*qring-def*)
apply (*simp add*:*set-rcs-def*, (*erule bexE*)+, *simp*)
apply (*subst Group.c-top-welldef*[*THEN sym*, *of b-ag R I*], *assumption*+)+
apply (*simp add*:*aGroup.agop-gop*)
apply (*simp add*:*aGroup.ag-carrier-carrier*)
apply (*simp add*:*aGroup.ag-pOp-commute*)
apply (*rule conjI*)
apply (*simp add*:*qring-def Group.Qg-iop-closed*)
apply (*rule conjI*)
apply (*rule allI*, *rule impI*)
apply (*simp add*:*qring-def*)
apply (*simp add*:*Group.Qg-i*[*of b-ag R I*])
apply (*rule conjI*)
apply (*simp add*:*qring-def*)
apply (*frule Group.nsg-sg*[*of b-ag R I*], *assumption*)
apply (*simp add*:*Group.unit-rcs-in-set-rcs*)
apply (*rule conjI*)
apply (*rule allI*, *rule impI*)
apply (*simp add*:*qring-def*)
apply (*simp add*:*Group.Qg-unit*[*of b-ag R I*])
apply (*rule conjI*)
apply(*rule Pi-I*)+
apply (*simp add*:*qring-def aGroup.aqgrp-carrier*)
apply (*simp add*:*set-ar-cos-def*, (*erule bexE*)+, *simp add*:*rcostOp*,
blast dest: *ring-tOp-closed*)
apply (*rule conjI*)
apply (*rule allI*, *rule impI*)+
apply (*simp add*:*qring-def aGroup.aqgrp-carrier*)
apply (*simp add*:*set-ar-cos-def*, (*erule bexE*)+, *simp add*:*rcostOp*)

apply (*frule-tac* $x = aa$ **and** $y = ab$ **in** *ring-tOp-closed*, *assumption*+,
frule-tac $x = ab$ **and** $y = ac$ **in** *ring-tOp-closed*, *assumption*+,
simp add:*rcostOp*, *simp add*:*ring-tOp-assoc*)
apply (*rule conjI*)
apply (*rule allI*, *rule impI*)+
apply (*simp add*:*qring-def aGroup.aqgrp-carrier*)
apply (*simp add*:*set-ar-cos-def*, (*erule bexE*)+, *simp add*:*rcostOp*,
simp add:*ring-tOp-commute*)
apply (*rule conjI*)
apply (*simp add*:*qring-def aGroup.aqgrp-carrier*)
apply (*cut-tac ring-one*, *simp add*:*set-ar-cos-def*, *blast*)
apply (*rule conjI*)
apply (*rule allI*, *rule impI*)+
apply (*simp add*:*qring-def aGroup.aqgrp-carrier*)
apply (*simp add*:*set-ar-cos-def*, (*erule bexE*)+, *simp*)
apply (*simp add*:*ar-c-top rcostOp*)
apply (*frule-tac* $x = ab$ **and** $y = ac$ **in** *aGroup.ag-pOp-closed*,
assumption+,
frule-tac $x = aa$ **and** $y = ab$ **in** *ring-tOp-closed*, *assumption*+ ,
frule-tac $x = aa$ **and** $y = ac$ **in** *ring-tOp-closed*, *assumption*+)
apply (*simp add*:*ar-c-top rcostOp*, *simp add*:*ring-distrib1*)
apply (*rule allI*, *rule impI*)
apply (*simp add*:*qring-def aGroup.aqgrp-carrier*)
apply (*simp add*:*set-ar-cos-def*, *erule bexE*, *simp*)
apply (*cut-tac ring-one*)
apply (*simp add*:*rcostOp*, *simp add*:*ring-l-one*)
done

lemma (**in** *Ring*) *qring-carrier*:*ideal R I* $\Longrightarrow$
carrier (*qring R I*) $= \{X.\ \exists\, a \in$ *carrier R*. $a \uplus_R I = X\}$
apply (*simp add*:*carrier-qring1 set-ar-cos-def*)
apply (*rule equalityI*)
apply (*rule subsetI*, *simp*, *erule bexE*, *frule sym*, *thin-tac* $x = a \uplus_R I$,
blast)
apply (*rule subsetI*, *simp*, *erule bexE*, *frule sym*, *thin-tac* $a \uplus_R I = x$,
blast)
done

lemma (**in** *Ring*) *qring-mem*:$[\![$*ideal R I*; $a \in$ *carrier R*$]\!] \Longrightarrow$
$a \uplus_R I \in$ *carrier* (*qring R I*)
apply (*simp add*:*qring-carrier*)
apply *blast*
done

lemma (**in** *Ring*) *qring-pOp*:$[\![$*ideal R I*; $a \in$ *carrier R*; $b \in$ *carrier R* $]\!]$
$\Longrightarrow$ *pop* (*qring R I*) ($a \uplus_R I$) ($b \uplus_R I$) $= (a \pm b) \uplus_R I$
by (*simp add*:*qring-def*, *simp add*:*ar-c-top*)

lemma (**in** *Ring*) *qring-zero*:*ideal R I* $\Longrightarrow$ *zero* (*qring R I*) $= I$

apply (*simp add*:*qring-def*)
done

lemma (**in** *Ring*) *qring-zero-1*:⟦*a* ∈ *carrier R*; *ideal R I*; *a* ⊎$_R$ *I* = *I*⟧ ⟹
a ∈ *I*
by (*frule a-in-ar-coset* [*of I a*], *assumption*+, *simp*)

lemma (**in** *Ring*) *Qring-fix1*:⟦*a* ∈ *carrier R*; *ideal R I*; *a* ∈ *I*⟧ ⟹ *a* ⊎$_R$ *I* = *I*
apply (*cut-tac ring-is-ag*, *frule aGroup.b-ag-group*)
apply (*simp add*:*ar-coset-def*)
apply (*frule ideal-asubg*[*of I*], *simp add*:*asubGroup-def*)
apply (*simp add*:*Group.rcs-fixed2*[*of b-ag R I*])
done

lemma (**in** *Ring*) *ar-cos-same*:⟦*a* ∈ *carrier R*; *ideal R I*; *x* ∈ *a* ⊎$_R$ *I*⟧ ⟹
x ⊎$_R$ *I* = *a* ⊎$_R$ *I*
apply (*cut-tac ring-is-ag*)
apply (*rule ar-coset-same1*[*of I x a*], *assumption*+)
apply (*rule ar-coset-subsetD*[*of I*], *assumption*+)
apply (*frule ar-coset-mem2*[*of I a x*], *assumption*+,
erule bexE)
apply (*frule-tac h* = *h* **in** *ideal-subset*[*of I*], *assumption*,
simp add:*aGroup.ag-p-inv*)
apply (*frule-tac x* = *a* **in** *aGroup.ag-mOp-closed*[*of R*], *assumption*+,
frule-tac x = *h* **in** *aGroup.ag-mOp-closed*[*of R*], *assumption*+)
apply (*simp add*:*aGroup.ag-pOp-assoc*[*THEN sym*],
simp add:*aGroup.ag-r-inv1 aGroup.ag-l-zero*)
apply (*simp add*:*ideal-inv1-closed*)
done

lemma (**in** *Ring*) *qring-tOp*:⟦*ideal R I*; *a* ∈ *carrier R*; *b* ∈ *carrier R*⟧ ⟹
tp (*qring R I*) (*a* ⊎$_R$ *I*) (*b* ⊎$_R$ *I*) = (*a* ·$_r$ *b*) ⊎$_R$ *I*
by (*simp add*:*qring-def*, *simp add*:*rcostOp*)

lemma *rind-hom-well-def*:⟦*Ring A*; *Ring R*; *f* ∈ *rHom A R*; *a* ∈ *carrier A* ⟧ ⟹
f a = (*f*°$_{A,R}$) (*a* ⊎$_A$ (*ker*$_{A,R}$ *f*))
apply (*frule ker-ideal*[*of A R f*], *assumption*+)
apply (*frule Ring.mem-set-ar-cos*[*of A ker*$_{A,R}$ *f a*], *assumption*+)
apply (*simp add*:*rind-hom-def*)
apply (*rule someI2-ex*)
apply (*frule Ring.a-in-ar-coset* [*of A ker*$_{A,R}$ *f a*], *assumption*+, *blast*)
apply (*frule-tac x* = *x* **in** *Ring.ar-coset-mem2*[*of A ker*$_{A,R}$ *f a*],
assumption+, *erule bexE*, *simp*,
frule-tac h = *h* **in** *Ring.ideal-subset*[*of A ker*$_{A,R}$ *f*], *assumption*+)
apply (*frule-tac Ring.ring-is-ag*[*of A*],
frule-tac Ring.ring-is-ag[*of R*],
simp add:*rHom-def*, *frule conjunct1*, *simp add*:*aHom-add*)
apply (*simp add*:*ker-def*)
apply (*frule aHom-mem*[*of A R f a*], *assumption*+,

simp add:*aGroup.ag-r-zero*)
done

lemma (**in** *Ring*) *set-r-ar-cos*:*ideal R I* $\Longrightarrow$
set-rcs (*b-ag R*) *I* = *set-ar-cos R I*
apply (*simp add*:*set-ar-cos-def set-rcs-def ar-coset-def*)
apply (*cut-tac ring-is-ag*)
apply (*simp add*:*aGroup.ag-carrier-carrier*)
done

lemma *set-r-ar-cos-ker*:⟦*Ring A*; *Ring R*; *f* ∈ *rHom A R* ⟧ $\Longrightarrow$
set-rcs (*b-ag A*) ($ker_{A,R}$ *f*) = *set-ar-cos A* ($ker_{A,R}$ *f*)
apply (*frule ker-ideal*[*of A R f*], *assumption*+)
apply (*simp add*:*Ring.carrier-qring*[*THEN sym*],
simp add:*Ring.carrier-qring1*[*THEN sym*])
done

lemma *ind-hom-rhom*:⟦*Ring A*; *Ring R*; *f* ∈ *rHom A R*⟧ $\Longrightarrow$
($f^\circ_{A,R}$) ∈ *rHom* (*qring A* ($ker_{A,R}$ *f*)) *R*
apply (*simp add*:*rHom-def* [*of qring A* ($ker_{A,R}$ *f*) *R*])
apply (*rule conjI*)
apply (*simp add*:*aHom-def*)
apply (*rule conjI*)
apply (*simp add*:*qring-def*)
apply (*simp add*:*rind-hom-def extensional-def*)
apply (*rule Pi-I*)
apply (*frule Ring.ring-is-ag* [*of A*], *frule Ring.ring-is-ag* [*of R*],
frule aGroup.b-ag-group [*of R*])
apply (*simp add*:*aGroup.ag-carrier-carrier* [*THEN sym*])
apply (*simp add*:*set-ar-cos-def*)
apply (*rule conjI*)
apply (*rule impI*)
apply (*erule bexE*, *simp*)
apply (*frule ker-ideal* [*of A R f*], *assumption*+)
apply (*frule-tac a* = *a* **in** *Ring.a-in-ar-coset* [*of A* $ker_{A,R}$ *f*],
assumption+)
apply (*rule someI2-ex*, *blast*)
apply (*frule-tac I* = $ker_{A,R}$ *f* **and** *a* = *a* **and** *x* = *xa* **in**
Ring.ar-coset-subsetD[*of A*], *assumption*+)
apply (*simp add*:*aGroup.ag-carrier-carrier*, *simp add*:*rHom-mem*)
apply (*simp add*:*set-r-ar-cos-ker*, *simp add*:*set-ar-cos-def*, *rule impI*, *blast*)
apply (*rule conjI*)
apply (*simp add*:*qring-def*)
apply (*simp add*:*set-r-ar-cos-ker*)
apply (*simp add*:*rind-hom-def extensional-def*)
apply (*rule ballI*)+
apply (*simp add*:*qring-def*)
apply (*simp add*:*set-r-ar-cos-ker*)
apply (*simp add*:*set-ar-cos-def*)

apply ((*erule bexE*)+, *simp*)
apply (*frule ker-ideal*[*of A R f*], *assumption*+)
apply (*simp add*:*Ring.ar-c-top*)
apply (*frule Ring.ring-is-ag*[*of A*],
frule Ring.ring-is-ag[*of R*],
frule-tac x = *aa* **and** *y* = *ab* **in** *aGroup.ag-pOp-closed*[*of A*],
assumption+)
apply (*simp add*:*rind-hom-well-def*[*THEN sym*])
apply (*simp add*:*rHom-def*, *frule conjunct1*, *simp add*:*aHom-add*)
apply (*rule conjI*)
apply (*rule ballI*)+
apply (*frule ker-ideal*[*of A R f*], *assumption*+,
simp add:*Ring.carrier-qring1*, *simp add*:*set-ar-cos-def*,
(*erule bexE*)+, *simp add*:*qring-def Ring.rcostOp*)
apply (*frule Ring.ring-is-ag*[*of A*],
frule-tac x = *a* **and** *y* = *aa* **in** *Ring.ring-tOp-closed*[*of A*],
assumption+)
apply (*simp add*:*rind-hom-well-def*[*THEN sym*], *simp add*:*rHom-tOp*)

apply (*simp add*:*qring-def*)
apply (*frule Ring.ring-one*[*of A*],
simp add:*rind-hom-well-def*[*THEN sym*],
simp add:*rHom-one*)
done

lemma *ind-hom-injec*:⟦*Ring A*; *Ring R*; *f* ∈ *rHom A R*⟧ ⟹
$injec_{(qring\ A\ (ker_{A,R}\ f)),R}\ (f^{\circ}{}_{A,R})$
apply (*simp add*:*injec-def*)
apply (*frule ind-hom-rhom* [*of A R f*], *assumption*+)
apply (*frule rHom-aHom*[*of* $f^{\circ}{}_{A,R}$ *A* $/_r$ ($ker_{A,R}\ f$) *R*], *simp*)
apply (*simp add*:*ker-def*[*of - -* $f^{\circ}{}_{A,R}$])
apply ((*subst qring-def*)+, *simp*)
apply (*simp add*:*set-r-ar-cos-ker*)

apply (*frule Ring.ring-is-ag*[*of A*],
frule Ring.ring-is-ag[*of R*],
frule ker-ideal[*of A R f*], *assumption*+)
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*simp*, *erule conjE*)
apply (*simp add*:*set-ar-cos-def*, *erule bexE*, *simp*)
apply (*simp add*:*rind-hom-well-def*[*THEN sym*, *of A R f*],
thin-tac x = *a* $\uplus_A$ $ker_{A,R}\ f$)
apply (*rule-tac a* = *a* **in** *Ring.Qring-fix1*[*of A -* $ker_{A,R}\ f$], *assumption*+)
apply (*simp add*:*ker-def*)

apply (*rule subsetI*, *simp*)
apply (*simp add*:*Ring.I-in-set-ar-cos*[*of A* $ker_{A,R}\ f$])
apply (*frule Ring.ideal-zero*[*of A* $ker_{A,R}\ f$], *assumption*+,

frule Ring.ring-zero[*of A*])

apply (*frule Ring.ar-coset-same4*[*of A* $ker_{A,R}$ *f* $\mathbf{0}_A$], *assumption*+)
apply (*frule rind-hom-well-def*[*THEN sym*, *of A R f* $\mathbf{0}_A$], *assumption*+)
apply *simp*

apply (*rule rHom-0-0*, *assumption*+)
done

lemma *rhom-to-rimg*:⟦*Ring A*; *Ring R*; *f* ∈ *rHom A R*⟧ ⟹
f ∈ *rHom A* (*rimg A R f*)
apply (*frule Ring.ring-is-ag*[*of A*], *frule Ring.ring-is-ag*[*of R*])
apply (*subst rHom-def*, *simp*)
apply (*rule conjI*)
apply (*subst aHom-def*, *simp*)
apply (*rule conjI*)
apply (*simp add*:*rimg-def*)
apply (*rule conjI*)
apply (*simp add*:*rHom-def aHom-def*)
apply ((*rule ballI*)+, *simp add*:*rimg-def*)
apply (*rule aHom-add*, *assumption*+)
apply (*simp add*:*rHom-aHom*, *assumption*+)

apply (*rule conjI*)
apply ((*rule ballI*)+, *simp add*:*rimg-def*, *simp add*:*rHom-tOp*)

apply (*simp add*:*rimg-def*, *simp add*:*rHom-one*)
done

lemma *ker-to-rimg*:⟦*Ring A*; *Ring R*; *f* ∈ *rHom A R* ⟧ ⟹
$ker_{A,R}$ *f* = $ker_{A,(rimg\ A\ R\ f)}$ *f*
apply (*frule rhom-to-rimg* [*of A R f*], *assumption*+)
apply (*simp add*:*ker-def*)
apply (*simp add*:*rimg-def*)
done

lemma *indhom-eq*:⟦*Ring A*; *Ring R*; *f* ∈ *rHom A R*⟧ ⟹ $f^\circ_{A,(rimg\ A\ R\ f)}$ = $f^\circ_{A,R}$
apply (*frule rimg-ring*[*of A R f*], *assumption*+)
apply (*frule rhom-to-rimg*[*of A R f*], *assumption*+,
frule ind-hom-rhom[*of A rimg A R f*], *assumption*+,
frule ind-hom-rhom[*of A R f*], *assumption*+)
apply (*rule funcset-eq*[*of* $f^\circ_{A,rimg\ A\ R\ f}$ *carrier* (*A* /$_r$ ($ker_{A,R}$ *f*)) $f^\circ_{A,R}$])
apply (*simp add*:*ker-to-rimg*[*THEN sym*],
simp add:*rHom-def*[*of - rimg A R f*] *aHom-def*)
apply (*simp add*:*rHom-def*[*of - R*] *aHom-def*)

apply (*simp add*:*ker-to-rimg*[*THEN sym*])
apply (*rule ballI*)

apply (*frule ker-ideal*[*of A R f*], *assumption*+,
simp add:*Ring.carrier-qring1*)
apply (*simp add*:*set-ar-cos-def*, *erule bexE*, *simp*)
apply (*simp add*:*rind-hom-well-def*[*THEN sym*])
apply (*frule rind-hom-well-def*[*THEN sym*, *of A rimg A R f f*],
assumption+, *simp add*:*ker-to-rimg*[*THEN sym*])
done

lemma *indhom-bijec2-rimg*:⟦*Ring A*; *Ring R*; $f \in$ *rHom A R*⟧ ⟹
$bijec_{(qring\ A\ (ker_{A,R}\ f)),(rimg\ A\ R\ f)}\ (f^\circ{}_{A,R})$
apply (*frule rimg-ring* [*of A R f*], *assumption*+)
apply (*frule rhom-to-rimg*[*of A R f*], *assumption*+)
apply (*frule ind-hom-rhom*[*of A rimg A R f f*], *assumption*+)
apply (*frule ker-to-rimg*[*THEN sym*, *of A R f*], *assumption*+)
apply (*frule indhom-eq*[*of A R f*], *assumption*+)
apply *simp*
apply (*simp add*:*bijec-def*)
apply (*rule conjI*)
apply (*simp add*:*injec-def*)
apply (*rule conjI*)
apply (*simp add*:*rHom-def*)
apply (*frule ind-hom-injec* [*of A R f*], *assumption*+)
apply (*simp add*:*injec-def*)
apply (*simp add*:*ker-def* [*of - - f*$^\circ{}_{A,R}$])
apply (*simp add*:*rimg-def*)

apply (*simp add*:*surjec-def*)
apply (*rule conjI*)
apply (*simp add*:*rHom-def*)
apply (*rule surj-to-test*)
apply (*simp add*:*rHom-def aHom-def*)
apply (*rule ballI*)
apply (*simp add*:*rimg-carrier*)
apply (*simp add*:*image-def*)
apply (*erule bexE*, *simp*)
apply (*frule-tac a1* = *x* **in** *rind-hom-well-def*[*THEN sym*, *of A R f*],
assumption+)
apply (*frule ker-ideal*[*of A R f*], *assumption*+,
simp add:*Ring.carrier-qring1*,
frule-tac a = *x* **in** *Ring.mem-set-ar-cos*[*of A ker*$_{A,R}$ *f*], *assumption*+)
apply *blast*
done

lemma *surjec-ind-bijec*:⟦*Ring A*; *Ring R*; $f \in$ *rHom A R*; *surjec*$_{A,R}$ *f*⟧ ⟹
$bijec_{(qring\ A\ (ker_{A,R}\ f)),R}\ (f^\circ{}_{A,R})$
apply (*frule ind-hom-rhom*[*of A R f*], *assumption*+)
apply (*simp add*:*surjec-def*)
apply (*simp add*:*bijec-def*)

apply (*simp add*:*ind-hom-injec*)

apply (*simp add*:*surjec-def*)
apply (*simp add*:*rHom-aHom*)
apply (*rule surj-to-test*)
apply (*simp add*:*rHom-def aHom-def*)
apply (*rule ballI*)
apply (*simp add*:*surj-to-def*, *frule sym*,
thin-tac f ‘ carrier A = carrier R, *simp*,
thin-tac carrier R = f ‘ carrier A)
apply (*simp add*:*image-def*, *erule bexE*)
apply (*frule-tac a1 = x* **in** *rind-hom-well-def*[*THEN sym*, *of A R f*],
assumption+)
apply (*frule ker-ideal*[*of A R f*], *assumption*+,
simp add:*Ring.carrier-qring1*,
frule-tac a = x **in** *Ring.mem-set-ar-cos*[*of A* $ker_{A,R}$ *f*], *assumption*+)
apply *blast*
done

lemma *ridmap-ind-bijec*:*Ring A* $\Longrightarrow$
$bijec_{(qring\ A\ (ker_{A,A}\ (ridmap\ A))),A}\ ((ridmap\ A)^{\circ}{}_{A,A})$
apply (*frule ridmap-surjec*[*of A*])
apply (*rule surjec-ind-bijec* [*of A A ridmap A*], *assumption*+)
apply (*simp add*:*rHom-def*, *simp add*:*surjec-def*)

apply (*rule conjI*)
apply (*rule ballI*)+
apply (*frule-tac x = x* **and** *y = y* **in** *Ring.ring-tOp-closed*[*of A*],
assumption+, *simp add*:*ridmap-def*)
apply (*simp add*:*ridmap-def Ring.ring-one*)

apply *assumption*
done

lemma *ker-of-idmap*:*Ring A* $\Longrightarrow$ $ker_{A,A}\ (ridmap\ A) = \{\mathbf{0}_A\}$
apply (*simp add*:*ker-def*)
apply (*simp add*:*ridmap-def*)
apply (*rule equalityI*)
apply (*rule subsetI*) **apply** (*simp add*:*CollectI*)
apply (*rule subsetI*) **apply** (*simp add*:*CollectI*)

apply (*simp add*:*Ring.ring-zero*)
done

lemma *ring-natural-isom*:*Ring A* $\Longrightarrow$
$bijec_{(qring\ A\ \{\mathbf{0}_A\}),A}\ ((ridmap\ A)^{\circ}{}_{A,A})$
apply (*frule ridmap-ind-bijec*)
apply (*simp add*: *ker-of-idmap*)
done

definition
pj :: [('*a*, '*m*) *Ring-scheme*, '*a set*] ⇒ ('*a* => '*a set*) **where**
pj R I = (λ*x*. *Pj* (*b-ag R*) *I x*)

lemma *pj-Hom*:⟦*Ring R*; *ideal R I*⟧ ⟹ (*pj R I*) ∈ *rHom R* (*qring R I*)
apply (*simp add*:*rHom-def*)
apply (*rule conjI*)
apply (*simp add*:*aHom-def*)
apply (*rule conjI*)
apply (*rule Pi-I*)
apply (*simp add*:*qring-def*)
apply (*frule Ring.ring-is-ag*)
apply (*simp add*:*aGroup.ag-carrier-carrier* [*THEN sym*])
apply (*simp add*:*pj-def Pj-def*)
apply (*simp add*:*set-rcs-def*) **apply** *blast*
apply (*rule conjI*)
apply (*simp add*:*pj-def Pj-def extensional-def*)
apply (*frule Ring.ring-is-ag*) **apply** (*simp add*:*aGroup.ag-carrier-carrier*)
apply (*rule ballI*)+
apply (*frule Ring.ring-is-ag*)
apply (*frule-tac x* = *a* **and** *y* = *b* **in** *aGroup.ag-pOp-closed*, *assumption*+)
apply (*simp add*:*aGroup.ag-carrier-carrier* [*THEN sym*])
apply (*simp add*:*pj-def Pj-def*)
apply (*simp add*:*qring-def*) **apply** (*frule aGroup.b-ag-group*)
apply (*simp add*:*aGroup.agop-gop* [*THEN sym*])
apply (*subst Group.c-top-welldef*[*of b-ag R I*], *assumption*+)
apply (*frule Ring.ideal-asubg*[*of R I*], *assumption*+)
apply (*simp add*:*aGroup.asubg-nsubg*)
apply *assumption*+
apply *simp*

apply (*rule conjI*)
apply (*rule ballI*)+
apply (*simp add*: *qring-def*)
apply (*frule-tac x* = *x* **and** *y* = *y* **in** *Ring.ring-tOp-closed*, *assumption*+)
apply (*frule Ring.ring-is-ag*)
apply (*simp add*:*aGroup.ag-carrier-carrier* [*THEN sym*])
apply (*simp add*:*pj-def Pj-def*)
apply (*simp add*:*aGroup.ag-carrier-carrier*)

apply (*frule-tac a1* = *x* **and** *b1* = *y* **in** *Ring.rcostOp* [*THEN sym*, *of R I*],
assumption+)
apply (*simp add*:*ar-coset-def*)
apply (*simp add*:*qring-def*)
apply (*frule Ring.ring-one*)
apply (*frule Ring.ring-is-ag*)

apply (*simp add*:*aGroup.ag-carrier-carrier* [*THEN sym*])
apply (*simp add*:*pj-def Pj-def*)
apply (*simp add*:*ar-coset-def*)
done

lemma *pj-mem*:⟦*Ring R*; *ideal R I*; $x \in$ *carrier R*⟧ ⟹ *pj R I x* = $x \uplus_R$ *I*
apply (*frule Ring.ring-is-ag*)
apply (*simp add*:*aGroup.ag-carrier-carrier* [*THEN sym*])
apply (*simp add*:*pj-def Pj-def*)
apply (*simp add*:*ar-coset-def*)
done

lemma *pj-zero*:⟦*Ring R*; *ideal R I*; $x \in$ *carrier R*⟧ ⟹
(*pj R I x* = $\mathbf{0}_{(R\ /_r\ I)}$) = ($x \in I$)
apply (*rule iffI*)
apply (*simp add*:*pj-mem Ring.qring-zero*,
simp add:*Ring.qring-zero-1*[*of R x I*])
apply (*simp add*:*pj-mem Ring.qring-zero*,
rule Ring.Qring-fix1, *assumption*+)
done

lemma *pj-surj-to*:⟦*Ring R*; *ideal R J*; *X* ∈ *carrier* (*R* /$_r$ *J*)⟧ ⟹
∃ *r*∈ *carrier R*. *pj R J r* = *X*
apply (*simp add*:*qring-def set-rcs-def*,
fold ar-coset-def, *simp add*:*b-ag-def*, *erule bexE*,
frule-tac x = *a* **in** *pj-mem*[*of R J*], *assumption*+, *simp*)
apply *blast*
done

lemma *invim-of-ideal*:⟦*Ring R*; *ideal R I*; *ideal* (*qring R I*) *J* ⟧ ⟹
ideal R (*rInvim R* (*qring R I*) (*pj R I*) *J*)
apply (*rule Ring.ideal-condition*, *assumption*)
apply (*simp add*:*rInvim-def*) **apply** (*rule subsetI*) **apply** (*simp add*:*CollectI*)
apply (*subgoal-tac* $\mathbf{0}_R \in$ *rInvim R* (*qring R I*) (*pj R I*) *J*)
apply (*simp add*:*nonempty*)
apply (*simp add*:*rInvim-def*)
apply (*simp add*: *Ring.ring-zero*)
apply (*frule Ring.ring-is-ag*)
apply (*frule pj-Hom* [*of R I*], *assumption*+)
apply (*frule Ring.qring-ring* [*of R I*], *assumption*+)
apply (*frule rHom-0-0* [*of R R* /$_r$ *I pj R I*], *assumption*+)
apply (*simp add*:*Ring.ideal-zero*)
apply (*rule ballI*)+
apply (*simp add*:*rInvim-def*) **apply** (*erule conjE*)+
apply (*rule conjI*)
apply (*frule Ring.ring-is-ag*)
apply (*rule aGroup.ag-pOp-closed*, *assumption*+)
apply (*rule aGroup.ag-mOp-closed*, *assumption*+)
apply (*frule pj-Hom* [*of R I*], *assumption*+)

apply (*frule Ring.ring-is-ag*)
apply (*frule-tac x = y* **in** *aGroup.ag-mOp-closed* [*of R*], *assumption*+)
apply (*simp add*:*rHom-def*) **apply** (*erule conjE*)+
apply (*subst aHom-add* [*of R R* /$_r$ *I pj R I*], *assumption*+)
apply (*simp add*:*Ring.qring-ring Ring.ring-is-ag*)
apply *assumption*+
apply (*frule Ring.qring-ring* [*of R I*], *assumption*+)
apply (*rule Ring.ideal-pOp-closed*, *assumption*+)
apply (*subst aHom-inv-inv*[*of R R* /$_r$ *I pj R I*], *assumption*+)
apply (*simp add*:*Ring.ring-is-ag*) **apply** *assumption*+
apply (*frule-tac x = pj R I y* **in** *Ring.ideal-inv1-closed* [*of R* /$_r$ *I J*],
assumption+)
apply (*rule ballI*)+
apply (*simp add*:*rInvim-def*) **apply** (*erule conjE*)
apply (*simp add*:*Ring.ring-tOp-closed*)
apply (*frule pj-Hom* [*of R I*], *assumption*+)
apply (*subst rHom-tOp* [*of R R* /$_r$ *I - - pj R I*], *assumption*+)
apply (*frule Ring.qring-ring*[*of R I*], *assumption*+)
apply (*rule Ring.ideal-ring-multiple* [*of R* /$_r$ *I J*])
apply (*simp add*:*Ring.qring-ring*) **apply** *assumption*+
apply (*simp add*:*rHom-mem*)
done

lemma *pj-invim-cont-I*:⟦*Ring R*; *ideal R I*; *ideal* (*qring R I*) *J*⟧ ⟹
I ⊆ (*rInvim R* (*qring R I*) (*pj R I*) *J*)
apply (*rule subsetI*)
apply (*simp add*:*rInvim-def*)
apply (*frule Ring.ideal-subset* [*of R I*], *assumption*+)
apply *simp*
apply (*frule pj-mem* [*of R I* -], *assumption*+)
apply (*simp add*:*Ring.ar-coset-same4*)
apply (*frule Ring.qring-ring*[*of R I*], *assumption*+)
apply (*frule Ring.ideal-zero* [*of qring R I J*], *assumption*+)

apply (*frule Ring.qring-zero*[*of R I*], *assumption*)
apply *simp*
done

lemma *pj-invim-mono1*:⟦*Ring R*; *ideal R I*; *ideal* (*qring R I*) *J1*;
ideal (*qring R I*) *J2*; *J1* ⊆ *J2* ⟧ ⟹
(*rInvim R* (*qring R I*) (*pj R I*) *J1*) ⊆ (*rInvim R* (*qring R I*) (*pj R I*) *J2*)
apply (*rule subsetI*)
apply (*simp add*:*rInvim-def*)
apply (*simp add*:*subsetD*)
done

lemma *pj-img-ideal*:⟦*Ring R*; *ideal R I*; *ideal R J*; *I* ⊆ *J*⟧ ⟹
ideal (*qring R I*) ((*pj R I*)‘*J*)
apply (*rule Ring.ideal-condition* [*of qring R I* (*pj R I*) ‘*J*])

apply (*simp add*:*Ring.qring-ring*)
apply (*rule subsetI*, *simp add*:*image-def*)
apply (*erule bexE*)
apply (*frule-tac h* = *xa* **in** *Ring.ideal-subset* [*of R J*], *assumption*+)
apply (*frule pj-Hom* [*of R I*], *assumption*+)
apply (*simp add*:*rHom-mem*)
apply (*frule Ring.ideal-zero* [*of R J*], *assumption*+)
apply (*simp add*:*image-def*) **apply** *blast*
apply (*rule ballI*)+
apply (*simp add*:*image-def*)
apply (*erule bexE*)+
apply (*frule pj-Hom* [*of R I*], *assumption*+)
apply (*rename-tac x y s t*)
apply (*frule-tac h* = *s* **in** *Ring.ideal-subset* [*of R J*], *assumption*+)
apply (*frule-tac h* = *t* **in** *Ring.ideal-subset* [*of R J*], *assumption*+)
apply (*simp add*:*rHom-def*) **apply** (*erule conjE*)+
apply (*frule Ring.ring-is-ag*)
apply (*frule Ring.qring-ring* [*of R I*], *assumption*+)
apply (*frule Ring.ring-is-ag* [*of R* /$_r$ *I*])
apply (*frule-tac x* = *t* **in** *aGroup.ag-mOp-closed* [*of R*], *assumption*+)
apply (*frule-tac a1* = *s* **and** *b1* = $-_{a\,R}$ *t* **in** *aHom-add* [*of R R* /$_r$ *I pj R I*, *THEN sym*], *assumption*+) **apply** (*simp add*:*aHom-inv-inv*)
apply (*frule-tac x* = *t* **in** *Ring.ideal-inv1-closed* [*of R J*], *assumption*+)
apply (*frule-tac x* = *s* **and** *y* = $-_{a\,R}$ *t* **in** *Ring.ideal-pOp-closed* [*of R J*], *assumption*+)
apply *blast*
apply (*rule ballI*)+
apply (*simp add*:*qring-def*)
apply (*simp add*:*Ring.set-r-ar-cos*)
apply (*simp add*:*set-ar-cos-def*, *erule bexE*)
apply *simp*
apply (*simp add*:*image-def*)
apply (*erule bexE*)
apply (*frule-tac x* = *xa* **in** *pj-mem* [*of R I*], *assumption*+)
apply (*simp add*:*Ring.ideal-subset*) **apply** *simp*
apply (*subst Ring.rcostOp*, *assumption*+)
apply (*simp add*:*Ring.ideal-subset*)
apply (*frule-tac x* = *xa* **and** *r* = *a* **in** *Ring.ideal-ring-multiple* [*of R J*], *assumption*+)
apply (*frule-tac h* = *a* $\cdot_{r\,R}$ *xa* **in** *Ring.ideal-subset* [*of R J*], *assumption*+)
apply (*frule-tac x1* = *a* $\cdot_{r\,R}$ *xa* **in** *pj-mem* [*THEN sym*, *of R I*], *assumption*+)
apply *simp*
apply *blast*
done

lemma *npQring*:⟦*Ring R*; *ideal R I*; *a* ∈ *carrier R*⟧ ⟹
npow (*qring R I*) (*a* ⊎$_R$ *I*) *n* = (*npow R a n*) ⊎$_R$ *I*

apply (*induct-tac n*)
apply (*simp add*:*qring-def*)

apply (*simp add*:*qring-def*)
apply (*rule Ring.rcostOp*, *assumption*+)
apply (*rule Ring.npClose*, *assumption*+)
done

4.5 Primary ideals, Prime ideals

definition
maximal-set :: [*'a set set*, *'a set*] ⇒ *bool* **where**
maximal-set S mx ⟷ *mx* ∈ *S* ∧ (∀ *s*∈*S*. *mx* ⊆ *s* ⟶ *s* = *mx*)

definition
nilpotent :: [-, *'a*] ⇒ *bool* **where**
nilpotent R a ⟷ (∃ (*n*::*nat*). *a* ^*R n* = **0**$_R$)

definition
zero-divisor :: [-, *'a*] ⇒ *bool* **where**
zero-divisor R a ⟷ (∃ *x*∈ *carrier R*. *x* ≠ **0**$_R$ ∧ *x* ·$_{r\,R}$ *a* = **0**$_R$)

definition
primary-ideal :: [-, *'a set*] ⇒ *bool* **where**
primary-ideal R q ⟷ *ideal R q* ∧ (*1*$_{r\,R}$) ∉ *q* ∧
(∀ *x*∈ *carrier R*. ∀ *y*∈ *carrier R*.
x ·$_{r\,R}$ *y* ∈ *q* ⟶ (∃ *n*. (*npow R x n*) ∈ *q* ∨ *y* ∈ *q*))

definition
prime-ideal :: [-, *'a set*] ⇒ *bool* **where**
prime-ideal R p ⟷ *ideal R p* ∧ (*1*$_{r\,R}$) ∉ *p* ∧ (∀ *x*∈ *carrier R*. ∀ *y*∈ *carrier R*.
(*x* ·$_{r\,R}$ *y* ∈ *p* ⟶ *x* ∈ *p* ∨ *y* ∈ *p*))

definition
maximal-ideal :: [-, *'a set*] ⇒ *bool* **where**
maximal-ideal R mx ⟷ *ideal R mx* ∧ *1*$_{r\,R}$ ∉ *mx* ∧
{*J*. (*ideal R J* ∧ *mx* ⊆ *J*)} = {*mx*, *carrier R*}

lemma (**in** *Ring*) *maximal-ideal-ideal*:⟦*maximal-ideal R mx*⟧ ⟹ *ideal R mx*
by (*simp add*:*maximal-ideal-def*)

lemma (**in** *Ring*) *maximal-ideal-proper*:*maximal-ideal R mx* ⟹ *1*$_r$ ∉ *mx*
by (*simp add*:*maximal-ideal-def*)

lemma (**in** *Ring*) *prime-ideal-ideal*:*prime-ideal R I* ⟹ *ideal R I*
by (*simp add*:*prime-ideal-def*)

lemma (**in** *Ring*) *prime-ideal-proper*:*prime-ideal R I* ⟹ *I* ≠ *carrier R*
apply (*simp add*:*prime-ideal-def*, (*erule conjE*)+)

apply (*simp add*:*proper-ideal*)
done

lemma (**in** *Ring*) *prime-ideal-proper1*:*prime-ideal R p* $\Longrightarrow$ $1_r \notin p$
by (*simp add*:*prime-ideal-def*)

lemma (**in** *Ring*) *primary-ideal-ideal*:*primary-ideal R q* $\Longrightarrow$ *ideal R q*
by (*simp add*:*primary-ideal-def*)

lemma (**in** *Ring*) *primary-ideal-proper1*:*primary-ideal R q* $\Longrightarrow$ $1_r \notin q$
by (*simp add*:*primary-ideal-def*)

lemma (**in** *Ring*) *prime-elems-mult-not*:$[\![$*prime-ideal R P*; $x \in$ *carrier R*;
$y \in$ *carrier R*; $x \notin P$; $y \notin P$ $]\!] \Longrightarrow x \cdot_r y \notin P$
apply (*simp add*:*prime-ideal-def*, (*erule conjE*)+)
apply (*rule contrapos-pp*, *simp*+)
apply (*frule-tac* $x = x$ **in** *bspec*, *assumption*,
thin-tac $\forall x \in$*carrier R*. $\forall y \in$*carrier R*. $x \cdot_r y \in P \longrightarrow x \in P \vee y \in P$,
frule-tac $x = y$ **in** *bspec*, *assumption*,
thin-tac $\forall y \in$*carrier R*. $x \cdot_r y \in P \longrightarrow x \in P \vee y \in P$, *simp*)
done

lemma (**in** *Ring*) *prime-is-primary*:*prime-ideal R p* $\Longrightarrow$ *primary-ideal R p*
apply (*unfold primary-ideal-def*)
apply (*rule conjI*, *simp add*:*prime-ideal-def*)
apply (*rule conjI*, *simp add*:*prime-ideal-def*)
apply ((*rule ballI*)+, *rule impI*)
apply (*simp add*:*prime-ideal-def*, (*erule conjE*)+)
apply (*frule-tac* $x = x$ **in** *bspec*, *assumption*,
thin-tac $\forall x \in$*carrier R*. $\forall y \in$*carrier R*. $x \cdot_r y \in p \longrightarrow x \in p \vee y \in p$,
frule-tac $x = y$ **in** *bspec*, *assumption*,
thin-tac $\forall y \in$*carrier R*. $x \cdot_r y \in p \longrightarrow x \in p \vee y \in p$, *simp*)
apply (*erule disjE*)
apply (*frule-tac* $t = x$ **in** *np-1*[*THEN sym*])
apply (*frule-tac* $a = x$ **and** $A = p$ **and** $b = x \hat{}_R$ (*Suc 0*) **in** *eq-elem-in*,
assumption)
apply *blast*
apply *simp*
done

lemma (**in** *Ring*) *maximal-prime-Tr0*:$[\![$*maximal-ideal R mx*; $x \in$ *carrier R*; $x \notin$ *mx*$]\!]$
$\Longrightarrow$ $mx \mp (Rxa\ R\ x) =$ *carrier R*
apply (*frule principal-ideal* [*of x*])
apply (*frule maximal-ideal-ideal*[*of mx*])
apply (*frule sum-ideals* [*of mx Rxa R x*], *assumption*)
apply (*frule sum-ideals-la1* [*of mx Rxa R x*], *assumption*)
apply (*simp add*:*maximal-ideal-def*)

apply (*erule conjE*)+
apply (*subgoal-tac* $mx \mp (Rxa\ R\ x) \in \{J.\ ideal\ R\ J \wedge mx \subseteq J\}$)
apply *simp*
apply (*frule sum-ideals-la2* [*of mx Rxa R x*], *assumption*+)
apply (*frule a-in-principal* [*of x*])
apply (*frule subsetD* [*of Rxa R x* $mx \mp (Rxa\ R\ x)$ *x*], *assumption*+)
apply (*thin-tac* $\{J.\ ideal\ R\ J \wedge mx \subseteq J\} = \{mx,\ carrier\ R\}$)
apply (*erule disjE*)
apply *simp* **apply** *simp*

apply (*thin-tac* $\{J.\ ideal\ R\ J \wedge mx \subseteq J\} = \{mx,\ carrier\ R\}$)
apply *simp*
done

lemma (**in** *Ring*) *maximal-is-prime*:*maximal-ideal R mx* $\Longrightarrow$ *prime-ideal R mx*
apply (*cut-tac ring-is-ag*)
apply (*simp add*:*prime-ideal-def*)
apply (*simp add*:*maximal-ideal-ideal*)
apply (*simp add*:*maximal-ideal-proper*)

apply ((*rule ballI*)+, *rule impI*)
apply (*rule contrapos-pp*, *simp*+, *erule conjE*)
apply (*frule-tac* $x = x$ **in** *maximal-prime-Tr0*[*of mx*], *assumption*+,
frule-tac $x = y$ **in** *maximal-prime-Tr0*[*of mx*], *assumption*+,
frule maximal-ideal-ideal[*of mx*],
frule ideal-subset1[*of mx*],
frule-tac $a = x$ **in** *principal-ideal*,
frule-tac $a = y$ **in** *principal-ideal*,
frule-tac $I = R \diamondsuit_p x$ **in** *ideal-subset1*,
frule-tac $I = R \diamondsuit_p y$ **in** *ideal-subset1*)
apply (*simp add*:*aGroup.set-sum*)
apply (*cut-tac ring-one*)
apply (*frule sym*,
thin-tac $\{xa.\ \exists h{\in}mx.\ \exists k{\in}R \diamondsuit_p x.\ xa = h \pm k\} = carrier\ R$,
frule sym,
thin-tac $\{x.\ \exists h{\in}mx.\ \exists k{\in}R \diamondsuit_p y.\ x = h \pm k\} = carrier\ R$)
apply (*frule-tac* $a = 1_r$ **and** $B = \{xa.\ \exists i{\in}mx.\ \exists j{\in}(Rxa\ R\ x).\ xa = i \pm j\}$ **in**
eq-set-inc[*of - carrier R*], *assumption*,
frule-tac $a = 1_r$ **and** $B = \{xa.\ \exists i{\in}mx.\ \exists j{\in}(Rxa\ R\ y).\ xa = i \pm j\}$ **in**
eq-set-inc[*of - carrier R*], *assumption*,
thin-tac carrier $R = \{xa.\ \exists i{\in}mx.\ \exists j{\in}(Rxa\ R\ x).\ xa = i \pm j\}$,
thin-tac carrier $R = \{x.\ \exists i{\in}mx.\ \exists j{\in}(Rxa\ R\ y).\ x = i \pm j\}$)
apply (*drule CollectD*, (*erule bexE*)+,
frule sym, *thin-tac* $1_r = i \pm j$)
apply (*drule CollectD*, (*erule bexE*)+, *rotate-tac* -1,
frule sym, *thin-tac* $1_r = ia \pm ja$)
apply (*frule-tac* $h = i$ **in** *ideal-subset*[*of mx*], *assumption*,
frule-tac $h = ia$ **in** *ideal-subset*[*of mx*], *assumption*,
frule-tac $h = j$ **in** *ideal-subset*, *assumption*+,

frule-tac $h = ja$ **in** *ideal-subset*, *assumption*+)
apply (*cut-tac ring-one*)
apply (*frule-tac* $x = i$ **and** $y = j$ **in** *aGroup.ag-pOp-closed*, *assumption*+)
apply (*frule-tac* $x = i \pm j$ **and** $y = ia$ **and** $z = ja$ **in** *ring-distrib1*,
assumption+)
apply (*frule-tac* $x = ia$ **and** $y = i$ **and** $z = j$ **in** *ring-distrib2*, *assumption*+,
frule-tac $x = ja$ **and** $y = i$ **and** $z = j$ **in** *ring-distrib2*, *assumption*+,
simp)
apply (*thin-tac* $1_r \cdot_r ia = i \cdot_r ia \pm j \cdot_r ia$,
thin-tac $1_r \cdot_r ja = i \cdot_r ja \pm j \cdot_r ja$,
simp add:*ring-l-one*[*of* 1_r])
apply (*frule-tac* $x = ia$ **and** $r = i$ **in** *ideal-ring-multiple*[*of mx*], *assumption*+,
frule-tac $x = i$ **and** $r = j$ **in** *ideal-ring-multiple1*[*of mx*], *assumption*+,
frule-tac $x = i$ **and** $r = ja$ **in** *ideal-ring-multiple1*[*of mx*], *assumption*+,
frule-tac $r = j$ **and** $x = ia$ **in** *ideal-ring-multiple*[*of mx*], *assumption*+)
apply (*subgoal-tac* $j \cdot_r ja \in mx$)
apply (*frule-tac* $x = i \cdot_r ia$ **and** $y = j \cdot_r ia$ **in** *ideal-pOp-closed*[*of mx*],
assumption+) **apply** (
frule-tac $x = i \cdot_r ja$ **and** $y = j \cdot_r ja$ **in** *ideal-pOp-closed*[*of mx*],
assumption+)
apply (*frule-tac* $x = i \cdot_r ia \pm j \cdot_r ia$ **and** $y = i \cdot_r ja \pm j \cdot_r ja$ **in**
ideal-pOp-closed[*of mx*], *assumption*+,
thin-tac $i \pm j = i \cdot_r ia \pm j \cdot_r ia \pm (i \cdot_r ja \pm j \cdot_r ja)$,
thin-tac $ia \pm ja = i \cdot_r ia \pm j \cdot_r ia \pm (i \cdot_r ja \pm j \cdot_r ja)$)
apply (*frule sym*, *thin-tac* $1_r = i \cdot_r ia \pm j \cdot_r ia \pm (i \cdot_r ja \pm j \cdot_r ja)$,
simp)
apply (*simp add*:*maximal-ideal-def*)

apply (*thin-tac* $i \pm j = i \cdot_r ia \pm j \cdot_r ia \pm (i \cdot_r ja \pm j \cdot_r ja)$,
thin-tac $ia \pm ja = i \cdot_r ia \pm j \cdot_r ia \pm (i \cdot_r ja \pm j \cdot_r ja)$,
thin-tac $i \cdot_r ia \pm j \cdot_r ia \pm (i \cdot_r ja \pm j \cdot_r ja) \in carrier\ R$,
thin-tac $1_r = i \cdot_r ia \pm j \cdot_r ia \pm (i \cdot_r ja \pm j \cdot_r ja)$,
thin-tac $i \cdot_r j \in mx$, *thin-tac* $i \cdot_r ja \in mx$,
thin-tac $R \diamondsuit_p y \subseteq carrier\ R$, *thin-tac* $R \diamondsuit_p x \subseteq carrier\ R$,
thin-tac ideal R $(R \diamondsuit_p y)$, *thin-tac ideal* R $(R \diamondsuit_p x)$)
apply (*simp add*:*Rxa-def*, (*erule bexE*)+, *simp*)
apply (*simp add*:*ring-tOp-assoc*)
apply (*simp add*:*ring-tOp-assoc*[*THEN sym*])
apply (*frule-tac* $x = x$ **and** $y = ra$ **in** *ring-tOp-commute*, *assumption*+, *simp*)
apply (*simp add*:*ring-tOp-assoc*,
frule-tac $x = x$ **and** $y = y$ **in** *ring-tOp-closed*, *assumption*+)
apply (*frule-tac x1* $= r$ **and** *y1* $= ra$ **and** *z1* $= x \cdot_r y$ **in**
ring-tOp-assoc[*THEN sym*], *assumption*+, *simp*)
apply (*frule-tac* $x = r$ **and** $y = ra$ **in** *ring-tOp-closed*, *assumption*+,
rule ideal-ring-multiple[*of mx*], *assumption*+)
done

lemma (**in** *Ring*) *chains-un*:$[\![c \in chains\ \{I.\ ideal\ R\ I \wedge I \subset carrier\ R\};\ c \neq \{\}]\!]$
$\Longrightarrow ideal\ R\ (\bigcup c)$

apply (*rule ideal-condition1*)
apply (*rule Union-least*[*of c carrier R*])
 apply (*simp add*:*chains-def*,
 erule conjE,
 frule-tac c = *X* **in** *subsetD*[*of c* {*I*. *ideal R I* ∧ *I* ⊂ *carrier R*}],
 assumption+, *simp add*:*psubset-imp-subset*)
 apply (*simp add*:*chains-def*,
 erule conjE)
 apply (*frule nonempty-ex*[*of c*], *erule exE*)
 apply (*frule-tac c* = *x* **in** *subsetD*[*of c* {*I*. *ideal R I* ∧ *I* ⊂ *carrier R*}],
 assumption+, *simp*, *erule conjE*)
 apply (*frule-tac I* = *x* **in** *ideal-zero*, *blast*)

apply (*rule ballI*)+
 apply *simp*
 apply (*erule bexE*)+
apply (*simp add*: *chains-def chain-subset-def*)
 apply (*frule conjunct1*) **apply** (*frule conjunct2*)
 apply (*thin-tac c* ⊆ {*I*. *ideal R I* ∧ *I* ⊂ *carrier R*} ∧ (∀ *x*∈*c*. ∀ *y*∈*c*. *x* ⊆ *y* ∨ *y* ⊆ *x*))
 apply (*frule-tac x* = *X* **in** *bspec*, *assumption*,
 thin-tac ∀ *x*∈*c*. ∀ *y*∈*c*. *x* ⊆ *y* ∨ *y* ⊆ *x*,
 frule-tac x = *Xa* **in** *bspec*, *assumption*,
 thin-tac ∀ *y*∈*c*. *X* ⊆ *y* ∨ *y* ⊆ *X*)
 apply (*frule-tac c* = *Xa* **in** *subsetD*[*of c* {*I*. *ideal R I* ∧ *I* ⊂ *carrier R*}],
 assumption+,
 frule-tac c = *X* **in** *subsetD*[*of c* {*I*. *ideal R I* ∧ *I* ⊂ *carrier R*}],
 assumption+, *simp*)
 apply (*erule conjE*)+
 apply (*erule disjE*,
 frule-tac c = *x* **and** *A* = *X* **and** *B* = *Xa* **in** *subsetD*, *assumption*+,
 frule-tac x = *x* **and** *y* = *y* **and** *I* = *Xa* **in** *ideal-pOp-closed*, *assumption*+,
 blast)
 apply (*frule-tac c* = *y* **and** *A* = *Xa* **and** *B* = *X* **in** *subsetD*, *assumption*+,
 frule-tac x = *x* **and** *y* = *y* **and** *I* = *X* **in** *ideal-pOp-closed*, *assumption*+,
 blast)

apply (*rule ballI*)+
 apply (*simp*, *erule bexE*)
 apply (*simp add*:*chains-def*, *erule conjE*)
 apply (*frule-tac c* = *X* **in** *subsetD*[*of c* {*I*. *ideal R I* ∧ *I* ⊂ *carrier R*}],
 assumption+, *simp*, *erule conjE*)
 apply (*frule-tac I* = *X* **and** *x* = *x* **and** *r* = *r* **in** *ideal-ring-multiple*,
 assumption+, *blast*)
done

lemma (**in** *Ring*) *zeroring-no-maximal*:*zeroring R* ⟹ ¬ (∃ *I*. *maximal-ideal R I*)
apply (*rule contrapos-pp*, *simp*+, *erule exE*,
 frule-tac mx = *x* **in** *maximal-ideal-ideal*)

apply (*frule-tac* $I = x$ **in** *ideal-zero*)
apply (*simp add*:*zeroring-def*, *erule conjE*,
cut-tac ring-one, *simp*, *thin-tac carrier* $R = \{\mathbf{0}\}$,
frule sym, *thin-tac* $1_r = \mathbf{0}$, *simp*, *thin-tac* $\mathbf{0} = 1_r$)
apply (*simp add*:*maximal-ideal-def*)
done

lemma (**in** *Ring*) *id-maximal-Exist*:$\neg$(*zeroring R*) $\Longrightarrow \exists I.$ *maximal-ideal R I*
apply (*cut-tac A*=$\{$ $I.$ *ideal R I* $\wedge$ $I \subset$ *carrier R* $\}$ **in** *Zorn-Lemma2*)
apply (*rule ballI*)

apply (*case-tac C*=$\{\}$, *simp*)
apply (*cut-tac zero-ideal*)
apply (*simp add*:*zeroring-def*)
apply (*cut-tac Ring*, *simp*,
frule not-sym, *thin-tac carrier* $R \neq \{\mathbf{0}\}$)
apply (*cut-tac ring-zero*,
frule singleton-sub[*of* $\mathbf{0}$ *carrier R*],
thin-tac $\mathbf{0} \in$ *carrier R*)
apply (*subst psubset-eq*)
apply *blast*
apply (*subgoal-tac* $\bigcup C \in \{I.$ *ideal R I* $\wedge$ $I \subset$ *carrier R*$\}$)
apply (*subgoal-tac* $\forall x \in C.\ x \subseteq (\bigcup C)$, *blast*)
apply (*rule ballI*, *rule Union-upper*, *assumption*)
apply (*simp add*:*chains-un*)
apply (*cut-tac* $A = C$ **in** *Union-least*[*of* - *carrier R*])
apply (*simp add*:*chains-def*, *erule conjE*,
frule-tac $c = X$ **and** $A = C$ **in**
subsetD[*of* - $\{I.$ *ideal R I* $\wedge$ $I \subset$ *carrier R*$\}$], *assumption*+,
simp add:*ideal-subset1*, *simp add*:*psubset-eq*)
apply (*rule contrapos-pp*, *simp*+,
cut-tac ring-one, *frule sym*, *thin-tac* $\bigcup C =$ *carrier R*)
apply (*frule-tac* $B = \bigcup C$ **in** *eq-set-inc*[*of* 1_r *carrier R*], *assumption*,
thin-tac carrier $R = \bigcup C$)
apply (*simp*, *erule bexE*)
apply (*simp add*:*chains-def*, *erule conjE*)
apply (*frule-tac* $c = X$ **and** $A = C$ **in**
subsetD[*of* - $\{I.$ *ideal R I* $\wedge$ $I \subseteq$ *carrier R* $\wedge$ $I \neq$ *carrier R*$\}$],
assumption+, *simp*, (*erule conjE*)+)
apply (*frule-tac* $I = X$ **in** *ideal-inc-one*, *assumption*+, *simp*)

apply (*erule bexE*, *simp*, *erule conjE*)
apply (*subgoal-tac maximal-ideal R M*, *blast*)
apply (*simp add*:*maximal-ideal-def*)

apply (*rule conjI*, *rule contrapos-pp*, *simp*+,
frule-tac $I = M$ **in** *ideal-inc-one*, *assumption*+, *simp*)

apply (*rule equalityI*)

apply (*rule subsetI*, *simp*)
apply (*erule conjE*)
apply (*frule-tac* $x = x$ **in** *spec*,
thin-tac $\forall x.\ ideal\ R\ x \wedge x \subset carrier\ R \longrightarrow M \subseteq x \longrightarrow x = M$, *simp*)
apply (*frule-tac* $I = x$ **in** *ideal-subset1*, *simp add*:*psubset-eq*)
apply (*case-tac* $x = carrier\ R$, *simp*)
apply *simp*

apply (*rule subsetI*, *simp*)
apply (*erule disjE*)
apply *simp*
apply (*simp add*:*whole-ideal*)
done

definition
ideal-Int :: [-, *'a set set*] ⇒ *'a set* **where**
ideal-Int R S == $\bigcap S$

lemma (**in** *Ring*) *ideal-Int-ideal*:⟦$S \subseteq \{I.\ ideal\ R\ I\}$; $S \neq \{\}$⟧ ⟹
ideal R ($\bigcap S$)
apply (*rule ideal-condition1*)
apply (*frule nonempty-ex*[*of S*], *erule exE*)
apply (*frule-tac* $c = x$ **in** *subsetD*[*of S* $\{I.\ ideal\ R\ I\}$], *assumption*+)
apply (*simp*, *frule-tac* $I = x$ **in** *ideal-subset1*)
apply (*frule-tac* $B = x$ **and** $A = S$ **in** *Inter-lower*)
apply (*rule-tac* $A = \bigcap S$ **and** $B = x$ **and** $C = carrier\ R$ **in** *subset-trans*,
assumption+)

apply (*cut-tac ideal-zero-forall*, *blast*)
apply (*simp*, *rule ballI*)

apply (*rule ballI*)+
apply *simp*
apply (*frule-tac* $x = X$ **in** *bspec*, *assumption*,
thin-tac $\forall X \in S.\ x \in X$,
frule-tac $x = X$ **in** *bspec*, *assumption*,
thin-tac $\forall X \in S.\ y \in X$)
apply (*frule-tac* $c = X$ **in** *subsetD*[*of S* $\{I.\ ideal\ R\ I\}$], *assumption*+,
simp, *rule-tac* $x = x$ **and** $y = y$ **in** *ideal-pOp-closed*, *assumption*+)

apply (*rule ballI*)+
apply (*simp*, *rule ballI*)
apply (*frule-tac* $x = X$ **in** *bspec*, *assumption*,
thin-tac $\forall X \in S.\ x \in X$,
frule-tac $c = X$ **in** *subsetD*[*of S* $\{I.\ ideal\ R\ I\}$], *assumption*+,
simp add:*ideal-ring-multiple*)
done

lemma (**in** *Ring*) *sum-prideals-Int*:⟦$\forall l \leq n.\ f\ l \in carrier\ R$;

$S = \{I.\ ideal\ R\ I \wedge f\ `\ \{i.\ i \leq n\} \subseteq I\}\rrbracket \Longrightarrow$
$(sum\text{-}pr\text{-}ideals\ R\ f\ n) = \bigcap\ S$
apply (*rule equalityI*)
apply (*subgoal-tac* $\forall X{\in}S.\ sum\text{-}pr\text{-}ideals\ R\ f\ n \subseteq X$)
apply *blast*
apply (*rule ballI*)
apply (*simp*, *erule conjE*)
apply (*rule-tac* $I = X$ **and** $n = n$ **and** $f = f$ **in** *sum-of-prideals4*, *assumption*+)
apply (*subgoal-tac* (*sum-pr-ideals R f n*) $\in S$)
apply *blast*
apply (*simp add*:*CollectI*)
apply (*simp add*: *sum-of-prideals2*)
apply (*simp add*: *sum-of-prideals*)
done

This proves that (*sum-pr-ideals R f n*) is the smallest ideal containing *f* ‘ (*Nset n*)

primrec *ideal-n-prod*::[('*a*, '*m*) *Ring-scheme*, *nat*, *nat* ⇒ '*a set*] ⇒ '*a set*
where
ideal-n-prod0: *ideal-n-prod R 0 J* = *J 0*
| *ideal-n-prodSn*: *ideal-n-prod R* (*Suc n*) *J* =
(*ideal-n-prod R n J*) $\diamondsuit_{r\,R}$ (*J* (*Suc n*))

abbreviation
IDNPROD ((*3i*Π_,_ -) [*98*,*98*,*99*]*98*) **where**
$i\Pi_{R,n}\ J == ideal\text{-}n\text{-}prod\ R\ n\ J$

primrec
ideal-pow :: ['*a set*, ('*a*, '*more*) *Ring-scheme*, *nat*] ⇒ '*a set*
((*3-*/ $\diamondsuit$_ _) [*120*,*120*,*121*]*120*)
where
ip0: *I* $\diamondsuit$*R 0* = *carrier R*
| *ipSuc*: *I* $\diamondsuit$*R* (*Suc n*) = *I* $\diamondsuit_{r\,R}$ (*I* $\diamondsuit$*R n*)

lemma (**in** *Ring*) *prod-mem-prod-ideals*:⟦*ideal R I*; *ideal R J*; $i \in I$; $j \in J$⟧ ⟹
$i \cdot_r j \in (I \diamondsuit_r J)$
apply (*simp add*:*ideal-prod-def*)
apply (*rule allI*, *rule impI*, *erule conjE*, *rename-tac X*)
apply (*rule-tac* $A = \{x.\ \exists i{\in}I.\ \exists j{\in}J.\ x = Ring.tp\ R\ i\ j\}$ **and** $B = X$ **and** $c = i \cdot_r j$ **in** *subsetD*, *assumption*)
apply *simp* **apply** *blast*
done

lemma (**in** *Ring*) *ideal-prod-ideal*:⟦*ideal R I*; *ideal R J* ⟧ ⟹
ideal R (*I* $\diamondsuit_r$ *J*)
apply (*rule ideal-condition1*)
apply (*simp add*:*ideal-prod-def*)
apply (*rule subsetI*, *simp*)
apply (*cut-tac whole-ideal*)

apply (*frule-tac* $x = \mathit{carrier}\ R$ **in** *spec*,
  thin-tac $\forall xa.\ \mathit{ideal}\ R\ xa \wedge \{x.\ \exists i{\in}I.\ \exists j{\in}J.\ x = i \cdot_r j\} \subseteq xa \longrightarrow x \in xa$)
apply (*subgoal-tac* $\{x.\ \exists i{\in}I.\ \exists j{\in}J.\ x = i \cdot_r j\} \subseteq \mathit{carrier}\ R$, *simp*)
  apply (*thin-tac* $\mathit{ideal}\ R\ (\mathit{carrier}\ R) \wedge \{x.\ \exists i{\in}I.\ \exists j{\in}J.\ x = i \cdot_r j\} \subseteq \mathit{carrier}\ R \longrightarrow x \in \mathit{carrier}\ R$)
apply (*rule subsetI*, *simp*, (*erule bexE*)+, *simp*)
apply (*frule-tac* $h = i$ **in** *ideal-subset*[*of I*], *assumption*+,
  frule-tac $h = j$ **in** *ideal-subset*[*of J*], *assumption*+)
apply (*rule-tac* $x = i$ **and** $y = j$ **in** *ring-tOp-closed*, *assumption*+)

apply (*frule ideal-zero*[*of I*],
  frule ideal-zero[*of J*],
  subgoal-tac $\mathbf{0} \in I \diamondsuit_{r\ R} J$, *blast*)
apply (*simp add*:*ideal-prod-def*)
apply (*rule allI*, *rule impI*, *erule conjE*)
apply (*rule ideal-zero*, *assumption*)

apply (*rule ballI*)+
apply (*simp add*:*ideal-prod-def*)
apply (*rule allI*, *rule impI*)
apply (*frule-tac* $x = xa$ **in** *spec*,
  thin-tac $\forall xa.\ \mathit{ideal}\ R\ xa \wedge \{x.\ \exists i{\in}I.\ \exists j{\in}J.\ x = i \cdot_r j\} \subseteq xa \longrightarrow x \in xa$,
  frule-tac $x = xa$ **in** *spec*,
  thin-tac $\forall x.\ \mathit{ideal}\ R\ x \wedge \{x.\ \exists i{\in}I.\ \exists j{\in}J.\ x = i \cdot_r j\} \subseteq x \longrightarrow y \in x$,
  erule conjE, *simp*,
  rule-tac $x = x$ **and** $y = y$ **in** *ideal-pOp-closed*, *assumption*+)
apply (*rule ballI*)+
  apply (*simp add*:*ideal-prod-def*)
  apply (*rule allI*, *rule impI*, *erule conjE*)
  apply (*frule-tac* $x = xa$ **in** *spec*,
    thin-tac $\forall xa.\ \mathit{ideal}\ R\ xa \wedge \{x.\ \exists i{\in}I.\ \exists j{\in}J.\ x = i \cdot_r j\} \subseteq xa \longrightarrow x \in xa$, *simp*)
apply (*simp add*:*ideal-ring-multiple*)
done

lemma (**in** *Ring*) *ideal-prod-commute*:$[\![\mathit{ideal}\ R\ I;\ \mathit{ideal}\ R\ J]\!] \Longrightarrow I \diamondsuit_r J = J \diamondsuit_r I$
apply (*simp add*:*ideal-prod-def*)
apply (*subgoal-tac* $\{K.\ \mathit{ideal}\ R\ K \wedge \{x.\ \exists i{\in}I.\ \exists j{\in}J.\ x = i \cdot_r j\} \subseteq K\} = \{K.\ \mathit{ideal}\ R\ K \wedge \{x.\ \exists i{\in}J.\ \exists j{\in}I.\ x = i \cdot_r j\} \subseteq K\}$)
apply *simp*
apply (*rule equalityI*)
apply (*rule subsetI*, *rename-tac X*, *simp*, *erule conjE*)
 apply (*rule subsetI*, *simp*)
 apply ((*erule bexE*)+)
 apply (*subgoal-tac* $x \in \{x.\ \exists i{\in}I.\ \exists j{\in}J.\ x = i \cdot_r j\}$,
  rule-tac $c = x$ **and** $A = \{x.\ \exists i{\in}I.\ \exists j{\in}J.\ x = i \cdot_r j\}$ **and** $B = X$ **in**

subsetD, *assumption*+,
frule-tac $h = i$ **in** *ideal-subset*[*of J*], *assumption*,
frule-tac $h = j$ **in** *ideal-subset*[*of I*], *assumption*,
frule-tac $x = i$ **and** $y = j$ **in** *ring-tOp-commute*, *assumption*+, *simp*,
blast)
apply (*rule subsetI*, *simp*, *erule conjE*,
rule subsetI, *simp*,
(*erule bexE*)+,
subgoal-tac $xa \in \{x.\ \exists\, i{\in}J.\ \exists\, j{\in}I.\ x = i \cdot_r j\}$,
rule-tac $c = xa$ **and** $A = \{x.\ \exists\, i{\in}J.\ \exists\, j{\in}I.\ x = i \cdot_r j\}$ **and** $B = x$ **in**
subsetD, *assumption*+,
frule-tac $h = i$ **in** *ideal-subset*[*of I*], *assumption*,
frule-tac $h = j$ **in** *ideal-subset*[*of J*], *assumption*,
frule-tac $x = i$ **and** $y = j$ **in** *ring-tOp-commute*, *assumption*+, *simp*,
blast)
done

lemma (**in** *Ring*) *ideal-prod-subTr*:$[\![$*ideal R I*; *ideal R J*; *ideal R C*;
$\forall\, i{\in}I.\ \forall\, j{\in}J.\ i \cdot_r j \in C]\!] \Longrightarrow I \diamondsuit_r J \subseteq C$
apply (*simp add*:*ideal-prod-def*)
apply (*rule-tac* $B = C$ **and**
$A = \{L.\ ideal\ R\ L \wedge \{x.\ \exists\, i{\in}I.\ \exists\, j{\in}J.\ x = i \cdot_r j\} \subseteq L\}$ **in**
Inter-lower)
apply *simp*
apply (*rule subsetI*, *simp*, (*erule bexE*)+, *simp*)
done

lemma (**in** *Ring*) *n-prod-idealTr*:
$(\forall\, k \leq n.\ ideal\ R\ (J\ k)) \longrightarrow ideal\ R\ (ideal\text{-}n\text{-}prod\ R\ n\ J)$
apply (*induct-tac n*)
apply (*rule impI*)
apply *simp*

apply (*rule impI*)
apply (*simp only*:*ideal-n-prodSn*)
apply (*cut-tac* $n = n$ **in** *Nsetn-sub-mem1*, *simp*)
apply (*rule ideal-prod-ideal*, *assumption*)
apply *simp*
done

lemma (**in** *Ring*) *n-prod-ideal*:$[\![\forall\, k \leq n.\ ideal\ R\ (J\ k)]\!]$
$\Longrightarrow\ ideal\ R\ (ideal\text{-}n\text{-}prod\ R\ n\ J)$
apply (*simp add*:*n-prod-idealTr*)
done

lemma (**in** *Ring*) *ideal-prod-la1*:$[\![ideal\ R\ I;\ ideal\ R\ J]\!] \Longrightarrow (I \diamondsuit_r J) \subseteq I$
apply (*simp add*:*ideal-prod-def*)
apply (*rule subsetI*)
apply (*simp add*:*CollectI*)

apply (*subgoal-tac* $\{x.\ \exists\, i{\in}I.\ \exists\, j{\in}J.\ x = \ i \cdot_r j\} \subseteq I$)
apply *blast*
apply (*thin-tac* $\forall\, xa.\ ideal\ R\ xa \wedge \{x.\ \exists\, i{\in}I.\ \exists\, j{\in}J.\ x = \ i \cdot_r j\} \subseteq xa \longrightarrow x \in xa$)
apply (*rule subsetI*, *simp add*:*CollectI*,
(*erule bexE*)+, *frule-tac* $h = j$ **in** *ideal-subset*[*of J*], *assumption*+)
apply (*simp add*:*ideal-ring-multiple1*)
done

lemma (**in** *Ring*) *ideal-prod-el1*:$[\![ideal\ R\ I;\ ideal\ R\ J;\ a \in (I \diamondsuit_r J)]\!] \Longrightarrow a \in I$
apply (*frule ideal-prod-la1* [*of I J*], *assumption*+)
apply (*rule subsetD*, *assumption*+)
done

lemma (**in** *Ring*) *ideal-prod-la2*:$[\![ideal\ R\ I;\ ideal\ R\ J\ ]\!] \Longrightarrow (I \diamondsuit_r J) \subseteq J$
apply (*subst ideal-prod-commute*, *assumption*+,
rule ideal-prod-la1[*of J I*], *assumption*+)
done

lemma (**in** *Ring*) *ideal-prod-sub-Int*:$[\![ideal\ R\ I;\ ideal\ R\ J\ ]\!] \Longrightarrow (I \diamondsuit_r J) \subseteq I \cap J$
by (*simp add*:*ideal-prod-la1 ideal-prod-la2*)

lemma (**in** *Ring*) *ideal-prod-el2*:$[\![ideal\ R\ I;\ ideal\ R\ J;\ a \in (I \diamondsuit_r J)]\!] \Longrightarrow a \in J$
by (*frule ideal-prod-la2* [*of I J*], *assumption*+,
rule subsetD, *assumption*+)

$i\Pi_{R,n}\ J$ is the product of ideals

lemma (**in** *Ring*) *ele-n-prodTr0*:$[\![\forall\, k \leq (Suc\ n).\ ideal\ R\ (J\ k);\ a \in i\Pi_{R,(Suc\ n)}\ J\ ]\!] \Longrightarrow a \in (i\Pi_{R,n}\ J) \wedge a \in (J\ (Suc\ n))$
apply (*simp add*:*Nset-Suc*[*of n*])
apply (*cut-tac n-prod-ideal*[*of n J*])
apply (*rule conjI*)
apply (*rule ideal-prod-el1* [*of* $i\Pi_{R,n}\ J\ J\ (Suc\ n)$], *assumption*, *simp*+)
apply (*rule ideal-prod-el2*[*of* $i\Pi_{R,n}\ J\ J\ (Suc\ n)$], *assumption*+, *simp*+)
done

lemma (**in** *Ring*) *ele-n-prodTr1*:
$(\forall\, k \leq n.\ ideal\ R\ (J\ k)) \wedge a \in ideal\text{-}n\text{-}prod\ R\ n\ J \longrightarrow (\forall\, k \leq n.\ a \in (J\ k))$
apply (*induct-tac n*)

apply *simp*

apply (*rule impI*)
apply (*rule allI*, *rule impI*)
apply (*cut-tac* $n = n$ **in** *Nsetn-sub-mem1*, *simp*)

apply (*erule conjE*)
apply (*frule-tac* $n = n$ **in** *ele-n-prodTr0*[*of* - J a])
apply *simp*

apply (*erule conjE*,
  thin-tac $\forall k{\leq}Suc\ n.$ *ideal* R $(J\ k)$)
apply *simp*
apply (*case-tac* $k = Suc\ n$, *simp*)
apply (*frule-tac* $m = k$ **and** $n = Suc\ n$ **in** *noteq-le-less*, *assumption*+,
  thin-tac $k \leq Suc\ n$)
apply (*frule-tac* $x = k$ **and** $n = Suc\ n$ **in** *less-le-diff*, *simp*)
done

lemma (**in** *Ring*) *ele-n-prod*:$[\![\forall k \leq n.$ *ideal* R $(J\ k)$;
  $a \in$ *ideal-n-prod* R n $J\ ]\!] \Longrightarrow \forall k \leq n.\ a \in (J\ k)$
by (*simp add*: *ele-n-prodTr1* [*of* n J a])

lemma (**in** *Ring*) *idealprod-whole-l*:*ideal* R $I \Longrightarrow$ (*carrier* R) $\diamondsuit_{r\,R}$ $I = I$
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*simp add*:*ideal-prod-def*)
apply (*subgoal-tac* $\{x.\ \exists i{\in}carrier\ R.\ \exists j{\in}I.\ x = i \cdot_r j\} \subseteq I$)
apply *blast*
apply (*thin-tac* $\forall xa.$ *ideal* R $xa \wedge \{x.\ \exists i{\in}carrier\ R.\ \exists j{\in}I.$
  $x = i \cdot_r j\} \subseteq xa \longrightarrow x \in xa$)
apply (*rule subsetI*)
apply *simp*
apply ((*erule bexE*)+, *simp*)
apply (*thin-tac* $xa = i \cdot_r j$, *simp add*:*ideal-ring-multiple*)
apply (*rule subsetI*)
apply (*simp add*:*ideal-prod-def*)
apply (*rule allI*, *rule impI*) **apply** (*erule conjE*)
apply (*rename-tac xa X*)
apply (*cut-tac ring-one*)
apply (*frule-tac* $h = xa$ **in** *ideal-subset*[*of I*], *assumption*,
  frule-tac $x = xa$ **in** *ring-l-one*)
apply (*subgoal-tac* $1_r \cdot_r xa \in \{x.\ \exists i{\in}carrier\ R.\ \exists j{\in}I.\ x = i \cdot_r j\}$)
apply (*rule-tac* $c = xa$ **and** $A = \{x.\ \exists i{\in}carrier\ R.\ \exists j{\in}I.\ x = i \cdot_r j\}$ **and**
  $B = X$ **in** *subsetD*, *assumption*+)
apply *simp*
apply *simp*
apply (*frule sym*, *thin-tac* $1_r \cdot_r xa = xa$, *blast*)
done

lemma (**in** *Ring*) *idealprod-whole-r*:*ideal* R $I \Longrightarrow I \diamondsuit_r$ (*carrier* R) $= I$
by (*cut-tac whole-ideal*,
  simp add:*ideal-prod-commute*[*of I carrier R*],
  simp add:*idealprod-whole-l*)

lemma (**in** *Ring*) *idealpow-1-self*:*ideal R I* $\Longrightarrow$ $I \diamondsuit_R (Suc\ 0) = I$
apply *simp*
apply (*simp add*:*idealprod-whole-r*)
done

lemma (**in** *Ring*) *ideal-pow-ideal*:*ideal R I* $\Longrightarrow$ *ideal R* $(I \diamondsuit_R n)$
apply (*induct-tac n*)
apply (*simp add*:*whole-ideal*)
apply *simp*
apply (*simp add*:*ideal-prod-ideal*)
done

lemma (**in** *Ring*) *ideal-prod-prime*:$[\![$*ideal R I*; *ideal R J*; *prime-ideal R P*;
$I \diamondsuit_r J \subseteq P\ ]\!] \Longrightarrow I \subseteq P \vee J \subseteq P$
apply (*rule contrapos-pp*, *simp+*)
apply (*erule conjE*, *simp add*:*subset-eq*, (*erule bexE*)+)
apply (*frule-tac i* = *x* **and** *j* = *xa* **in** *prod-mem-prod-ideals*[*of I J*],
assumption+)
apply (*frule-tac x* = $x \cdot_r xa$ **in** *bspec*, *assumption*,
thin-tac $\forall x \in I \diamondsuit_r J.\ x \in P$)
apply (*simp add*: *prime-ideal-def*, (*erule conjE*)+)
apply (*frule-tac h* = *x* **in** *ideal-subset*, *assumption*,
frule-tac x = *x* **in** *bspec*, *assumption*,
thin-tac $\forall x \in carrier\ R.\ \forall y \in carrier\ R.\ x \cdot_r y \in P \longrightarrow x \in P \vee y \in P$,
frule-tac h = *xa* **in** *ideal-subset*, *assumption*,
frule-tac x = *xa* **in** *bspec*, *assumption*,
thin-tac $\forall y \in carrier\ R.\ x \cdot_r y \in P \longrightarrow x \in P \vee y \in P$,
simp)
done

lemma (**in** *Ring*) *ideal-n-prod-primeTr*:*prime-ideal R P* $\Longrightarrow$
$(\forall k \leq n.\ ideal\ R\ (J\ k)) \longrightarrow (ideal\text{-}n\text{-}prod\ R\ n\ J \subseteq P) \longrightarrow$
$(\exists i \leq n.\ (J\ i) \subseteq P)$
apply (*induct-tac n*)
apply *simp*

apply (*rule impI*)
apply (*rule impI*, *simp*)
apply (*cut-tac I* = $i\Pi_{R,n}$ *J* **and** *J* = *J* (*Suc n*) **in**
ideal-prod-prime[*of* - - *P*],
rule-tac n = *n* **and** *J* = *J* **in** *n-prod-ideal*,
rule allI, *simp+*)
apply (*erule disjE*, *simp*)
apply (*cut-tac n* = *n* **in** *Nsetn-sub-mem1*,
blast)
apply *blast*
done

lemma (**in** *Ring*) *ideal-n-prod-prime*:$[\![$*prime-ideal R P*;

$\forall k \leq n.\ ideal\ R\ (J\ k);\ ideal\text{-}n\text{-}prod\ R\ n\ J \subseteq P\rrbracket \Longrightarrow$
$\exists i \leq n.\ (J\ i) \subseteq P$
apply (*simp add*:*ideal-n-prod-primeTr*)
done

definition
ppa::[-, *nat* ⇒ 'a *set*, 'a *set*, *nat*] ⇒ (*nat* ⇒ 'a) **where**
ppa R P A i l = (*SOME x*. $x \in A \wedge x \in (P\ (skip\ i\ l)) \wedge x \notin P\ i$)

lemma (**in** *Ring*) *prod-primeTr*:⟦*prime-ideal R P*; *ideal R A*; ¬ $A \subseteq P$;
ideal R B; ¬ $B \subseteq P$ ⟧ ⟹ ∃ *x*. $x \in A \wedge x \in B \wedge x \notin P$
apply (*simp add*:*subset-eq*)
apply (*erule bexE*)+
apply (*subgoal-tac x* ·$_r$ *xa* ∈ *A* ∧ *x* ·$_r$ *xa* ∈ *B* ∧ *x* ·$_r$ *xa* ∉ *P*)
apply *blast*
apply (*rule conjI*)
apply (*rule ideal-ring-multiple1*, *assumption*+)
apply (*simp add*:*ideal-subset*)
apply (*rule conjI*)
apply (*rule ideal-ring-multiple*, *assumption*+)
apply (*simp add*:*ideal-subset*)

apply (*rule contrapos-pp*, *simp*+)
apply (*simp add*:*prime-ideal-def*, (*erule conjE*)+)
apply (*frule-tac h* = *x* **in** *ideal-subset*[*of A*], *assumption*+,
frule-tac h = *xa* **in** *ideal-subset*[*of B*], *assumption*+,
frule-tac x = *x* **in** *bspec*, *assumption*,
thin-tac ∀ *x*∈*carrier R*. ∀ *y*∈*carrier R*. *x* ·$_r$ *y* ∈ *P* ⟶ *x* ∈ *P* ∨ *y* ∈ *P*,
frule-tac x = *xa* **in** *bspec*, *assumption*,
thin-tac ∀ *y*∈*carrier R*. *x* ·$_r$ *y* ∈ *P* ⟶ *x* ∈ *P* ∨ *y* ∈ *P*)
apply *simp*
done

lemma (**in** *Ring*) *prod-primeTr1*:⟦∀ *k* ≤ (*Suc n*). *prime-ideal R* (*P k*);
ideal R A; ∀ *l* ≤ (*Suc n*). ¬ (*A* ⊆ *P l*);
∀ *k* ≤ (*Suc n*). ∀ *l* ≤ (*Suc n*). *k* = *l* ∨ ¬ (*P k*) ⊆ (*P l*); *i* ≤ (*Suc n*)⟧ ⟹
∀ *l* ≤ *n*. *ppa R P A i l* ∈ *A* ∧
ppa R P A i l ∈ (*P* (*skip i l*)) ∧ *ppa R P A i l* ∉ (*P i*)
apply (*rule allI*, *rule impI*)
apply (*cut-tac i* = *i* **and** *l* = *l* **in** *skip-il-neq-i*)
apply (*rotate-tac 2*)
apply (*frule-tac x* = *i* **in** *spec*,
thin-tac ∀ *l* ≤ (*Suc n*). ¬ *A* ⊆ *P l*, *simp*)

apply (*cut-tac l* = *l* **in** *skip-mem*[*of* - *n i*], *simp*,
frule-tac x = *skip i l* **in** *spec*,
thin-tac ∀ *k* ≤ (*Suc n*). ∀ *l* ≤ (*Suc n*). *k* = *l* ∨ ¬ *P k* ⊆ *P l*,
simp)

```
    apply (rotate-tac −1,
          frule-tac x = i in spec,
          thin-tac ∀ la ≤ (Suc n). skip i l = la ∨ ¬ P (skip i l) ⊆ P la,
          simp)
apply (cut-tac P = P i and A = A and B = P (skip i l) in prod-primeTr,
      simp, assumption+)
 apply (frule-tac x = skip i l in spec,
       thin-tac ∀ k≤Suc n. prime-ideal R (P k), simp,
       rule prime-ideal-ideal, assumption+)
 apply (simp add:ppa-def)
 apply (rule someI2-ex, assumption+)
done

lemma (in Ring) ppa-mem:⟦∀ k ≤ (Suc n). prime-ideal R (P k); ideal R A;
     ∀ l ≤ (Suc n). ¬ (A ⊆ P l);
     ∀ k ≤ (Suc n). ∀ l ≤ (Suc n). k = l ∨ ¬ (P k) ⊆ (P l);
     i ≤ (Suc n); l ≤ n⟧ ⟹ ppa R P A i l ∈ carrier R
apply (frule-tac prod-primeTr1[of n P A], assumption+)
 apply (rotate-tac −1, frule-tac x = l in spec,
       thin-tac ∀ l≤n. ppa R P A i l ∈ A ∧
          ppa R P A i l ∈ P (skip i l) ∧ ppa R P A i l ∉ P i, simp)
 apply (simp add:ideal-subset)
done

lemma (in Ring) nsum-memrTr:(∀ i ≤ n. f i ∈ carrier R) ⟶
                          (∀ l ≤ n. nsum R f l ∈ carrier R)
apply (cut-tac ring-is-ag)
apply (induct-tac n)

 apply (rule impI, rule allI, rule impI)
 apply simp

apply (rule impI)
 apply (rule allI, rule impI)

 apply (rule aGroup.nsum-mem, assumption)
 apply (rule allI, simp)
done

lemma (in Ring) nsum-memr:∀ i ≤ n. f i ∈ carrier R ⟹
                        ∀ l ≤ n. nsum R f l ∈ carrier R
by (simp add:nsum-memrTr)

lemma (in Ring) nsum-ideal-incTr:ideal R A ⟹
             (∀ i ≤ n. f i ∈ A) ⟶ nsum R f n ∈ A
 apply (induct-tac n)
 apply (rule impI)
  apply simp
```

apply (*rule impI*)
apply *simp*
apply (*rule ideal-pOp-closed*, *assumption*+)
apply *simp*
done

lemma (**in** *Ring*) *nsum-ideal-inc*:⟦*ideal R A*; ∀ *i* ≤ *n*. *f i* ∈ *A*⟧ ⟹
nsum R f n ∈ *A*
by (*simp add*:*nsum-ideal-incTr*)

lemma (**in** *Ring*) *nsum-ideal-excTr*:*ideal R A* ⟹
(∀ *i* ≤ *n*. *f i* ∈ *carrier R*) ∧ (∃ *j* ≤ *n*. (∀ *l* ∈ {*i*. *i* ≤ *n*} −{*j*}. *f l* ∈ *A*)
∧ (*f j* ∉ *A*)) ⟶ *nsum R f n* ∉ *A*
apply (*induct-tac n*)

apply *simp*

apply (*rule impI*)
apply (*erule conjE*)+
apply (*erule exE*)
apply (*case-tac j* = *Suc n*, *simp*) **apply** (
thin-tac (∃ *j*≤*n*. *f j* ∉ *A*) ⟶ Σ$_e$ *R f n* ∉ *A*)
apply (*erule conjE*)
apply (*cut-tac n* = *n* **and** *f* = *f* **in** *nsum-ideal-inc*[*of A*], *assumption*,
rule allI, *simp*)
apply (*rule contrapos-pp*, *simp*+)
apply (*frule-tac a* = Σ$_e$ *R f n* **and** *b* = *f* (*Suc n*) **in**
ideal-ele-sumTr1[*of A*],
simp add:*ideal-subset*, *simp*, *assumption*+, *simp*)

apply (*erule conjE*,
frule-tac m = *j* **and** *n* = *Suc n* **in** *noteq-le-less*, *assumption*,
frule-tac x = *j* **and** *n* = *Suc n* **in** *less-le-diff*,
thin-tac j ≤ *Suc n*, *thin-tac j* < *Suc n*, *simp*,
cut-tac n = *n* **in** *Nsetn-sub-mem1*, *simp*)
apply (*erule conjE*,
frule-tac x = *Suc n* **in** *bspec*, *simp*)
apply (*rule contrapos-pp*, *simp*+)
apply (*frule-tac a* = Σ$_e$ *R f n* **and** *b* = *f* (*Suc n*) **in**
ideal-ele-sumTr2[*of A*])
apply (*cut-tac ring-is-ag*,
rule-tac n = *n* **in** *aGroup.nsum-mem*[*of R - f*], *assumption*+,
rule allI, *simp*, *simp*, *assumption*+, *simp*)
apply (*subgoal-tac* ∃ *j*≤*n*. (∀ *l*∈{*i*. *i* ≤ *n*} − {*j*}. *f l* ∈ *A*) ∧ *f j* ∉ *A*,
simp,
thin-tac (∃ *j*≤*n*. (∀ *l*∈{*i*. *i* ≤ *n*} − {*j*}. *f l* ∈ *A*) ∧ *f j* ∉ *A*)
⟶ Σ$_e$ *R f n* ∉ *A*)
apply (*subgoal-tac* ∀ *l*∈{*i*. *i* ≤ *n*} − {*j*}. *f l* ∈ *A*, *blast*,
thin-tac Σ$_e$ *R f n* ± *f* (*Suc n*) ∈ *A*,

thin-tac $\Sigma_e\ R\ f\ n \in A$)
apply (*rule ballI*)
apply (*frule-tac* $x = l$ **in** *bspec*, *simp*, *assumption*)
done

lemma (**in** *Ring*) *nsum-ideal-exc*:$[\![$*ideal R A*; $\forall\, i \leq n.\ f\ i \in$ *carrier R*;
$\exists j \leq n.\ (\forall\, l \in \{i.\ i \leq n\} - \{j\}.\ f\ l \in A) \wedge (f\ j \notin A)\ ]\!] \Longrightarrow$ *nsum R f n* $\notin A$
by (*simp add*:*nsum-ideal-excTr*)

lemma (**in** *Ring*) *nprod-memTr*:$(\forall\, i \leq n.\ f\ i \in$ *carrier R*$) \longrightarrow$
$(\forall\, l.\ l \leq n \longrightarrow$ *nprod R f l* $\in$ *carrier R*)
apply (*induct-tac n*)
apply (*rule impI*, *rule allI*, *rule impI*, *simp*)

apply (*rule impI*, *rule allI*, *rule impI*)
apply (*case-tac* $l \leq n$)
apply (*cut-tac* $n = n$ **in** *Nset-Suc*, *blast*)
apply (*cut-tac* $m = l$ **and** $n =$ *Suc n* **in** *Nat.le-antisym*, *assumption*)
apply (*simp add*: *not-less*)
apply *simp*
apply (*rule ring-tOp-closed*, *simp*)
apply (*cut-tac* $n = n$ **in** *Nset-Suc*, *blast*)
done

lemma (**in** *Ring*) *nprod-mem*:$[\![\forall\, i \leq n.\ f\ i \in$ *carrier R*; $l \leq n]\!] \Longrightarrow$
nprod R f l $\in$ *carrier R*
by (*simp add*:*nprod-memTr*)

lemma (**in** *Ring*) *ideal-nprod-incTr*:*ideal R A* $\Longrightarrow$
$(\forall\, i \leq n.\ f\ i \in$ *carrier R*$) \wedge$
$(\exists\, l \leq n.\ f\ l \in A) \longrightarrow$ *nprod R f n* $\in A$
apply (*induct-tac n*)

apply *simp*

apply (*rule impI*)
apply (*erule conjE*)+
apply *simp*
apply (*erule exE*)
apply (*case-tac* $l =$ *Suc n*, *simp*)
apply (*rule-tac* $x = f\ (Suc\ n)$ **and** $r =$ *nprod R f n* **in**
ideal-ring-multiple[*of A*], *assumption*+)
apply (*rule-tac* $n =$ *Suc n* **and** $f = f$ **and** $l = n$ **in** *nprod-mem*,
assumption+, *simp*)
apply (*erule conjE*)
apply (*frule-tac* $m = l$ **and** $n =$ *Suc n* **in** *noteq-le-less*, *assumption*,
frule-tac $x = l$ **and** $n =$ *Suc n* **in** *less-le-diff*,
thin-tac $l \leq$ *Suc n*, *thin-tac* $l <$ *Suc n*, *simp*)
apply (*rule-tac* $x =$ *nprod R f n* **and** $r = f\ (Suc\ n)$ **in**

ideal-ring-multiple1[*of A*], *assumption*+)
apply *blast*
apply *simp*
done

lemma (**in** *Ring*) *ideal-nprod-inc*:⟦*ideal R A*; ∀ *i* ≤ *n*. *f i* ∈ *carrier R*;
∃ *l* ≤ *n*. *f l* ∈ *A*⟧ ⟹ *nprod R f n* ∈ *A*
by (*simp add*:*ideal-nprod-incTr*)

lemma (**in** *Ring*) *nprod-excTr*:*prime-ideal R P* ⟹
(∀ *i* ≤ *n*. *f i* ∈ *carrier R*) ∧ (∀ *l* ≤ *n*. *f l* ∉ *P*) ⟶
nprod R f n ∉ *P*
apply (*induct-tac n*)

apply *simp*

apply (*rule impI*)
apply (*erule conjE*)+
apply *simp*
apply (*rule-tac y* = *f* (*Suc n*) **and** *x* = *nprod R f n* **in**
prime-elems-mult-not[*of P*], *assumption*,
rule-tac n = *n* **in** *nprod-mem*, *rule allI*, *simp*+)
done

lemma (**in** *Ring*) *prime-nprod-exc*:⟦*prime-ideal R P*; ∀ *i* ≤ *n*. *f i* ∈ *carrier R*;
∀ *l* ≤ *n*. *f l* ∉ *P*⟧ ⟹ *nprod R f n* ∉ *P*
by (*simp add*:*nprod-excTr*)

definition
nilrad :: - ⇒ *'a set* **where**
nilrad R = {*x*. *x* ∈ *carrier R* ∧ *nilpotent R x*}

lemma (**in** *Ring*) *id-nilrad-ideal*:*ideal R* (*nilrad R*)
apply (*cut-tac ring-is-ag*)
apply (*rule ideal-condition1*[*of nilrad R*])
apply (*rule subsetI*) **apply** (*simp add*:*nilrad-def CollectI*)
apply (*simp add*:*nilrad-def*)
apply (*cut-tac ring-zero*)
apply (*subgoal-tac nilpotent R* **0**)
apply *blast*
apply (*simp add*:*nilpotent-def*)
apply (*frule np-1*[*of* **0**], *blast*)

apply (*rule ballI*)+
apply (*simp add*:*nilrad-def nilpotent-def*, (*erule conjE*)+)
apply (*erule exE*)+
apply (*simp add*:*aGroup.ag-pOp-closed*[*of R*])
apply (*frule-tac x* = *x* **and** *y* = *y* **and** *m* = *n* **and** *n* = *na* **in** *npAdd*,
assumption+, *blast*)

```
 apply (rule ballI)+
 apply (simp add:nilrad-def nilpotent-def, erule conjE, erule exE)
 apply (simp add:ring-tOp-closed,
        frule-tac x = r and y = x and n = n in npMul, assumption+,
          simp,
        frule-tac x = r and n = n in npClose)
        apply (simp add:ring-times-x-0, blast)
done

definition
  rad-ideal :: [-, 'a set ] ⇒ 'a set where
  rad-ideal R I = {a. a ∈ carrier R ∧ nilpotent (qring R I) ((pj R I) a)}

lemma (in Ring) id-rad-invim:ideal R I ⟹
      rad-ideal R I = (rInvim R (qring R I) (pj R I ) (nilrad (qring R I)))
apply (cut-tac ring-is-ag)
apply (rule equalityI)
 apply (rule subsetI)
 apply (simp add:rad-ideal-def)
 apply (erule conjE)+
 apply (simp add:rInvim-def)
 apply (simp add:nilrad-def)
 apply (subst pj-mem, rule Ring-axioms)
 apply assumption+
 apply (simp add:qring-def ar-coset-def set-rcs-def)
 apply (simp add:aGroup.ag-carrier-carrier)
 apply blast

apply (rule subsetI)
 apply (simp add:rInvim-def nilrad-def)
apply (simp add: rad-ideal-def)
done

lemma (in Ring) id-rad-ideal:ideal R I ⟹ ideal R (rad-ideal R I)

apply (subst id-rad-invim [of I], assumption)
apply (rule invim-of-ideal, rule Ring-axioms, assumption)
apply (rule Ring.id-nilrad-ideal)
apply (simp add:qring-ring)
done

lemma (in Ring) id-rad-cont-I:ideal R I ⟹ I ⊆ (rad-ideal R I)
apply (simp add:rad-ideal-def)
apply (rule subsetI, simp,
      simp add:ideal-subset)
apply (simp add:nilpotent-def)
apply (subst pj-mem, rule Ring-axioms, assumption+,
      simp add:ideal-subset)
```

apply (*frule-tac* $h = x$ **in** *ideal-subset*[*of I*], *assumption*,
frule-tac $a = x$ **in** *npQring*[*OF Ring*, *of I - Suc 0*], *assumption*,
simp only:*np-1*, *simp only*:*Qring-fix1*,
subst qring-zero[*of I*], *assumption*)
apply *blast*
done

lemma (**in** *Ring*) *id-rad-set*:*ideal R I* $\Longrightarrow$
rad-ideal R I $= \{x.\ x \in carrier\ R \wedge (\exists n.\ npow\ R\ x\ n \in I)\}$
apply (*simp add*:*rad-ideal-def*)
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*simp add*:*nilpotent-def*, *erule conjE*, *erule exE*)
apply (*simp add*: *pj-mem*[*OF Ring*], *simp add*:*npQring*[*OF Ring*])
apply (*simp add*:*qring-zero*)
apply (*frule-tac* $x = x$ **and** $n = n$ **in** *npClose*)
apply (*frule-tac* $a = x\ \hat{}_R\ n$ **in** *ar-coset-same3*[*of I*], *assumption*+,
blast)
apply (*rule subsetI*, *simp*, *erule conjE*, *erule exE*)
apply (*simp add*:*nilpotent-def*)
apply (*simp add*: *pj-mem*[*OF Ring*], *simp add*:*npQring*[*OF Ring*],
simp add:*qring-zero*)
apply (*frule-tac* $a = x\ \hat{}_R\ n$ **in** *ar-coset-same4*[*of I*], *assumption*+)
apply *blast*
done

lemma (**in** *Ring*) *rad-primary-prime*:*primary-ideal R q* $\Longrightarrow$
prime-ideal R (*rad-ideal R q*)
apply (*simp add*:*prime-ideal-def*)
apply (*frule primary-ideal-ideal*[*of q*])
apply (*simp add*:*id-rad-ideal*)
apply (*rule conjI*)
apply (*rule contrapos-pp*, *simp*+)
apply (*simp add*:*id-rad-set*, *erule conjE*, *erule exE*)
apply (*simp add*:*npOne*)
apply (*simp add*:*primary-ideal-proper1*[*of q*])

apply ((*rule ballI*)+, *rule impI*)
apply (*rule contrapos-pp*, *simp*+, *erule conjE*)
apply (*simp add*:*id-rad-set*, *erule conjE*, *erule exE*)
apply (*simp add*:*npMul*)
apply (*simp add*:*primary-ideal-def*, (*erule conjE*)+)
apply (*frule-tac* $x = x$ **and** $n = n$ **in** *npClose*,
frule-tac $x = y$ **and** $n = n$ **in** *npClose*)
apply (*frule-tac* $x = x\ \hat{}_R\ n$ **in** *bspec*, *assumption*,
thin-tac $\forall x \in carrier\ R.\ \forall y \in carrier\ R.\ x \cdot_r y \in q \longrightarrow$
$(\exists n.\ x\ \hat{}_R\ n \in q) \vee y \in q$,
frule-tac $x = y\ \hat{}_R\ n$ **in** *bspec*, *assumption*,

thin-tac $\forall\, y{\in}carrier\ R.\ x\ \hat{}_R\ n\ \cdot_r\ y \in q \longrightarrow (\exists\, na.\ x\ \hat{}_R\ n\ \hat{}_R\ na \in q) \vee y \in q$, *simp*)
apply (*simp add*:*npMulExp*)
done

lemma (**in** *Ring*) *npow-notin-prime*:$[\![$*prime-ideal R P*; $x \in$ *carrier R*; $x \notin P]\!] \Longrightarrow \forall\, n.\ npow\ R\ x\ n \notin P$
apply (*rule allI*)
apply (*induct-tac n*)
apply *simp*
apply (*simp add*:*prime-ideal-proper1*)

apply *simp*
apply (*frule-tac x* = *x* **and** *n* = *na* **in** *npClose*)
apply (*simp add*:*prime-elems-mult-not*)
done

lemma (**in** *Ring*) *npow-in-prime*:$[\![$*prime-ideal R P*; $x \in$ *carrier R*; $\exists\, n.\ npow\ R\ x\ n \in P\ ]\!] \Longrightarrow x \in P$
apply (*rule contrapos-pp*, *simp*+)
apply (*frule npow-notin-prime*, *assumption*+)
apply *blast*
done

definition
mul-closed-set::[-, *'a set*] ⇒ *bool* **where**
mul-closed-set R S ⟷ $S \subseteq carrier\ R \wedge (\forall\, s{\in}S.\ \forall\, t{\in}S.\ s\ \cdot_{r\,R}\ t \in S)$

locale *Idomain* = *Ring* +
assumes *idom*:
$[\![a \in carrier\ R;\ b \in carrier\ R;\ a\ \cdot_r\ b = \mathbf{0}]\!] \Longrightarrow a = \mathbf{0} \vee b = \mathbf{0}$

locale *Corps* =
fixes *K* (**structure**)
assumes *f-is-ring*: *Ring K*
and *f-inv*: $\forall\, x{\in}carrier\ K - \{\mathbf{0}\}.\ \exists\, x' \in carrier\ K.\ x' \cdot_r\ x = 1_r$

lemma (**in** *Ring*) *mul-closed-set-sub*:*mul-closed-set R S* $\Longrightarrow S \subseteq carrier\ R$
by (*simp add*:*mul-closed-set-def*)

lemma (**in** *Ring*) *mul-closed-set-tOp-closed*:$[\![$*mul-closed-set R S*; $s \in S$; $t \in S]\!] \Longrightarrow s\ \cdot_r\ t \in S$
by (*simp add*:*mul-closed-set-def*)

lemma (**in** *Corps*) *f-inv-unique*:$[\![\ x \in carrier\ K - \{\mathbf{0}\};\ x' \in carrier\ K;\ x'' \in carrier\ K;\ x' \cdot_r\ x = 1_r;\ x'' \cdot_r\ x = 1_r\ ]\!] \Longrightarrow x' = x''$
apply (*cut-tac f-is-ring*)

apply (*cut-tac* $x = x'$ **and** $y = x$ **and** $z = x''$ **in** *Ring.ring-tOp-assoc*[*of* K],
assumption+, *simp*, *assumption*, *simp*)
apply (*simp add*:*Ring.ring-l-one*[*of* K],
simp add:*Ring.ring-tOp-commute*[*of* K x x''] *Ring.ring-r-one*[*of* K])
done

definition
invf :: [-, 'a] ⇒ 'a **where**
invf K x = (*THE* y. $y \in$ *carrier* $K \wedge y \cdot_{r\,K} x = 1_{r\,K}$)

lemma (**in** *Corps*) *invf-inv*:$x \in$ *carrier* $K - \{\mathbf{0}\} \Longrightarrow$
(*invf* K x) ∈ *carrier* $K \wedge$ (*invf* K x) $\cdot_r$ $x = 1_r$
apply (*simp add*:*invf-def*)
apply (*rule theI'*)
apply (*rule ex-ex1I*)
apply (*cut-tac f-inv*, *blast*)
apply (*rule-tac* $x' = xa$ **and** $x'' = y$ **in** *f-inv-unique*[*of* x])
apply *simp*+
done

definition
npowf :: - ⇒ 'a ⇒ *int* ⇒ 'a **where**
npowf K x n =
(*if* 0 ≤ n *then npow* K x (*nat* n) *else npow* K (*invf* K x) (*nat* (− n)))

abbreviation
NPOWF :: ['a, -, *int*] ⇒ 'a ((3-_^) [77,77,78]77) **where**
$a_K{}^n$ == *npowf* K a n

abbreviation
IOP :: ['a, -] ⇒ 'a ((-^ -) [87,88]87) **where**
a^{-K} == *invf* K a

lemma (**in** *Idomain*) *idom-is-ring*: *Ring* R **..**

lemma (**in** *Idomain*) *idom-tOp-nonzeros*:⟦$x \in$ *carrier* R;
$y \in$ *carrier* R; $x \neq \mathbf{0}$; $y \neq \mathbf{0}$⟧ ⟹ $x \cdot_r y \neq \mathbf{0}$
apply (*rule contrapos-pp*, *simp*+)
apply (*cut-tac idom*[*of* x y]) **apply** (*erule disjE*, *simp*+)
done

lemma (**in** *Idomain*) *idom-potent-nonzero*:
⟦$x \in$ *carrier* R; $x \neq \mathbf{0}$⟧ ⟹ *npow* R x $n \neq \mathbf{0}$
apply (*induct-tac* n)
apply *simp*
apply (*rule contrapos-pp*, *simp*+)
apply (*frule ring-l-one*[*of* x, *THEN sym*]) **apply** *simp*

apply (*simp add*:*ring-times-0-x*)

apply (*rule contrapos-pp*, *simp*+)
apply (*frule-tac n* = *n* **in** *npClose*[*of x*],
cut-tac a = *x* ^*R n* **and** *b* = *x* **in** *idom*, *assumption*+)
apply (*erule disjE*, *simp*+)
done

lemma (**in** *Idomain*) *idom-potent-unit*:⟦*a* ∈ *carrier R*; *0* < *n*⟧
⟹ (*Unit R a*) = (*Unit R* (*npow R a n*))
apply (*rule iffI*)
apply (*simp add*:*Unit-def*, *erule bexE*)
apply (*simp add*:*npClose*)
apply (*frule-tac x1* = *a* **and** *y1* = *b* **and** *n1* = *n* **in** *npMul*[*THEN sym*], *assumption*,
simp add:*npOne*)
apply (*frule-tac x* = *b* **and** *n* = *n* **in** *npClose*, *blast*)

apply (*case-tac n* = *Suc 0*, *simp only*: *np-1*)
apply (*simp add*:*Unit-def*, *erule conjE*, *erule bexE*)
apply (*cut-tac x* = *a* **and** *n* = *n* − *Suc 0* **in** *npow-suc*[*of R*], *simp del*:*npow-suc*,
thin-tac a ^*R n* = *a* ^*R* (*n* − *Suc 0*) ·*r a*,
frule-tac x = *a* **and** *n* = *n* − *Suc 0* **in** *npClose*,
frule-tac x = *a* ^*R* (*n* − *Suc 0*) **and** *y* = *a* **in** *ring-tOp-commute*, *assumption*+,
simp add:*ring-tOp-assoc*,
frule-tac x = *a* ^*R* (*n* − *Suc 0*) **and** *y* = *b* **in** *ring-tOp-closed*, *assumption*+)
apply *blast*
done

lemma (**in** *Idomain*) *idom-mult-cancel-r*:⟦*a* ∈ *carrier R*;
b ∈ *carrier R*; *c* ∈ *carrier R*; *c* ≠ **0**; *a* ·*r c* = *b* ·*r c*⟧ ⟹ *a* = *b*
apply (*cut-tac ring-is-ag*)
apply (*frule ring-tOp-closed*[*of a c*], *assumption*+,
frule ring-tOp-closed[*of b c*], *assumption*+)
apply (*simp add*:*aGroup.ag-eq-diffzero*[*of R a* ·*r c b* ·*r c*],
simp add:*ring-inv1-1*,
frule aGroup.ag-mOp-closed[*of R b*], *assumption*,
simp add:*ring-distrib2*[*THEN sym*, *of c a* −*a b*])
apply (*frule aGroup.ag-pOp-closed*[*of R a* −*a b*], *assumption*+)
apply (*subst aGroup.ag-eq-diffzero*[*of R a b*], *assumption*+)
apply (*rule contrapos-pp*, *simp*+)
apply (*frule idom-tOp-nonzeros*[*of a* ± −*a b c*], *assumption*+, *simp*)
done

lemma (**in** *Idomain*) *idom-mult-cancel-l*:⟦*a* ∈ *carrier R*;
b ∈ *carrier R*; *c* ∈ *carrier R*; *c* ≠ **0**; *c* ·*r a* = *c* ·*r b*⟧ ⟹ *a* = *b*
apply (*simp add*:*ring-tOp-commute*)
apply (*simp add*:*idom-mult-cancel-r*)

done

lemma (**in** *Corps*) *invf-closed1*:$x \in$ *carrier* $K - \{\mathbf{0}\} \Longrightarrow$
invf $K\ x \in$ (*carrier* K) $- \{\mathbf{0}\}$
apply (*frule invf-inv*[*of* x], *erule conjE*)
apply (*rule contrapos-pp*, *simp*+)
apply (*cut-tac f-is-ring*) **apply** (
simp add:*Ring.ring-times-0-x*[*of* K])
apply (*frule sym*, *thin-tac* $\mathbf{0} = 1_r$, *simp*, *erule conjE*)
apply (*frule Ring.ring-l-one*[*of* $K\ x$], *assumption*)
apply (*rotate-tac* -1, *frule sym*, *thin-tac* $1_r \cdot_r x = x$,
simp add:*Ring.ring-times-0-x*)
done

lemma (**in** *Corps*) *linvf*:$x \in$ *carrier* $K - \{\mathbf{0}\} \Longrightarrow$ (*invf* $K\ x$) $\cdot_r\ x = 1_r$
by (*simp add*:*invf-inv*)

lemma (**in** *Corps*) *field-is-ring*:*Ring* K
by (*simp add*:*f-is-ring*)

lemma (**in** *Corps*) *invf-one*:$1_r \neq \mathbf{0} \implies$ *invf* $K\ (1_r) = 1_r$
apply (*cut-tac field-is-ring*)
apply (*frule-tac Ring.ring-one*)
apply (*cut-tac invf-closed1* [*of* 1_r])
apply (*cut-tac linvf*[*of* 1_r])
apply (*simp add*:*Ring.ring-r-one*[*of* K])
apply *simp*+
done

lemma (**in** *Corps*) *field-tOp-assoc*:$[\![x \in$ *carrier* K; $y \in$ *carrier* K; $z \in$ *carrier* $K]\!]$
$\Longrightarrow x \cdot_r y \cdot_r z = x \cdot_r (y \cdot_r z)$
apply (*cut-tac field-is-ring*)
apply (*simp add*:*Ring.ring-tOp-assoc*)
done

lemma (**in** *Corps*) *field-tOp-commute*:$[\![x \in$ *carrier* K; $y \in$ *carrier* $K]\!]$
$\Longrightarrow x \cdot_r y = y \cdot_r x$
apply (*cut-tac field-is-ring*)
apply (*simp add*:*Ring.ring-tOp-commute*)
done

lemma (**in** *Corps*) *field-inv-inv*:$[\![x \in$ *carrier* K; $x \neq \mathbf{0}]\!] \Longrightarrow (x^{\text{-}K})^{\text{-}K} = x$
apply (*cut-tac invf-closed1*[*of* x])
apply (*cut-tac invf-inv*[*of* $x^{\text{-}K}$], *erule conjE*)
apply (*frule field-tOp-assoc*[*THEN sym*, *of* $x^{\text{-}}\ ^{K\text{-}}\ K\ x^{\text{-}}\ ^{K}\ x$],
simp, *assumption*, *simp*)
apply (*cut-tac field-is-ring*,
simp add:*Ring.ring-l-one Ring.ring-r-one*, *erule conjE*,
cut-tac invf-inv[*of* x], *erule conjE*, *simp add*:*Ring.ring-r-one*)

apply *simp+*
done

lemma (**in** *Corps*) *field-is-idom*:*Idomain K*
apply (*rule Idomain.intro*)
apply (*simp add*:*field-is-ring*)
apply (*cut-tac field-is-ring*)
apply (*rule Idomain-axioms.intro*)
apply (*rule contrapos-pp*, *simp+*, *erule conjE*)
apply (*cut-tac* $x = a$ **in** *invf-closed1*, *simp*, *simp*, *erule conjE*)
apply (*frule-tac* $x = a^{-}{}^{K}$ **and** $y = a$ **and** $z = b$ **in** *field-tOp-assoc*,
assumption+)
apply (*simp add*:*linvf Ring.ring-times-x-0 Ring.ring-l-one*)
done

lemma (**in** *Corps*) *field-potent-nonzero*:$[\![x \in carrier\ K;\ x \neq \mathbf{0}]\!] \Longrightarrow$
$x \hat{}^{K}\ n \neq \mathbf{0}$
apply (*cut-tac field-is-idom*)
apply (*cut-tac field-is-ring*,
simp add:*Idomain.idom-potent-nonzero*)
done

lemma (**in** *Corps*) *field-potent-nonzero1*:$[\![x \in carrier\ K;\ x \neq \mathbf{0}]\!] \Longrightarrow x\,{}_{K}{}^{n} \neq \mathbf{0}$
apply (*simp add*:*npowf-def*)
apply (*case-tac* $0 \leq n$)
apply (*simp add*:*field-potent-nonzero*)

apply *simp*
apply (*cut-tac invf-closed1*[*of x*], *simp+*, (*erule conjE*)+)
apply (*simp add*:*field-potent-nonzero*)
apply *simp*
done

lemma (**in** *Corps*) *field-nilp-zero*:$[\![x \in carrier\ K;\ x \hat{}^{K}\ n = \mathbf{0}]\!] \Longrightarrow x = \mathbf{0}$
by (*rule contrapos-pp*, *simp+*, *simp add*:*field-potent-nonzero*)

lemma (**in** *Corps*) *npowf-mem*:$[\![a \in carrier\ K;\ a \neq \mathbf{0}]\!] \Longrightarrow$
$npowf\ K\ a\ n \in carrier\ K$
apply (*simp add*:*npowf-def*)
apply (*cut-tac field-is-ring*)
apply (*case-tac* $0 \leq n$, *simp*,
simp add:*Ring.npClose*, *simp*)

apply (*cut-tac invf-closed1*[*of a*], *simp*, *erule conjE*,
simp add:*Ring.npClose*, *simp*)
done

lemma (**in** *Corps*) *field-npowf-exp-zero*:$[\![a \in carrier\ K;\ a \neq \mathbf{0}]\!] \Longrightarrow$
$npowf\ K\ a\ 0 = 1_{r}$

by (*cut-tac field-is-ring*, *simp add*:*npowf-def*)

lemma (**in** *Corps*) *npow-exp-minusTr1*:⟦*x* ∈ *carrier K*; *x* ≠ **0**; *0* ≤ *i*⟧ ⟹
0 ≤ *i* − (*int j*) ⟶ $x_K^{(i - (int\ j))} = x \hat{}_K\ (nat\ i)\ \cdot_r\ (x^{-K}) \hat{}_K\ j$
apply (*cut-tac field-is-ring*,
cut-tac invf-closed1[*of x*], *simp*,
simp add:*npowf-def*, *erule conjE*)
apply (*induct-tac j*, *simp*)
apply (*frule Ring.npClose*[*of K x nat i*], *assumption*+,
simp add:*Ring.ring-r-one*)
apply (*rule impI*, *simp*)
apply (*subst zdiff*)
apply (*simp add*:*add.commute*[*of 1*])
apply (*cut-tac z* = *i* **and** *w* = *int n* + *1* **in** *zdiff*,
simp only:*minus-add-distrib*,
thin-tac i − (*int n* + *1*) = *i* + (− *int n* + − *1*))
apply (*cut-tac z* = *i* + − *int n* **in** *nat-diff-distrib*[*of 1*],
simp, *simp*)
apply (*simp only*:*zdiff*[*of* - *1*], *simp*)

apply (*cut-tac field-is-idom*)
apply (*frule-tac n* = *nat i* **in** *Ring.npClose*[*of K x*], *assumption*+,
frule-tac n = *nat i* **in** *Ring.npClose*[*of K* $x^{-\ K}$], *assumption*+,
frule-tac n = *n* **in** *Ring.npClose*[*of K* $x^{-\ K}$], *assumption*+)
apply (*rule-tac a* = $x \hat{}_K\ (nat\ (i + (-\ int\ n - 1)))$ **and**
b = $x \hat{}_K\ (nat\ i)\ \cdot_r\ (x^{-\ K} \hat{}_K\ n\ \cdot_r\ x^{-\ K})$ **and** *c* = *x* **in**
Idomain.idom-mult-cancel-r[*of K*], *assumption*+)
apply (*simp add*:*Ring.npClose*, *rule Ring.ring-tOp-closed*, *assumption*+,
rule Ring.ring-tOp-closed, *assumption*+)
apply (*subgoal-tac 0* < *nat* (*i* − *int n*))
apply (*subst Ring.npMulElmR*, *assumption*+, *simp*,
simp add:*field-tOp-assoc*[*THEN sym*, *of* $x \hat{}_K\ (nat\ i)$ - $x^{-\ K}$])
apply (*subst field-tOp-assoc*[*of* - - *x*])
apply (*rule Ring.ring-tOp-closed*[*of K*], *assumption*+)
apply (*simp add*: *linvf*)
apply (*subst Ring.ring-r-one*[*of K*], *assumption*)
apply *auto*
apply (*metis Ring.npClose*)
apply (*simp only*: *uminus-add-conv-diff* [*symmetric*] *add.assoc* [*symmetric*])
apply (*simp add*: *algebra-simps nat-diff-distrib Suc-diff-Suc*)
apply (*smt Ring.npMulElmR Suc-nat-eq-nat-zadd1 nat-diff-distrib' nat-int of-nat-0-le-iff*)
done

lemma (**in** *Corps*) *npow-exp-minusTr2*:⟦*x* ∈ *carrier K*; *x* ≠ **0**; *0* ≤ *i*; *0* ≤ *j*;
0 ≤ *i* − *j*⟧ ⟹ $x_K^{(i - j)} = x \hat{}_K\ (nat\ i)\ \cdot_r\ (x^{-K}) \hat{}_K\ (nat\ j)$
apply (*frule npow-exp-minusTr1*[*of x i nat j*], *assumption*+)
apply *simp*
done

lemma (**in** *Corps*) *npowf-inv*:$[\![x \in \mathit{carrier}\ K;\ x \neq \mathbf{0};\ 0 \leq j]\!] \Longrightarrow x_K{}^{j} = (x^{\text{-}K})_K{}^{(-j)}$
apply (*simp add*:*npowf-def*)
 apply (*rule impI*, *simp add*:*zle*)
 apply (*simp add*:*field-inv-inv*)
done

lemma (**in** *Corps*) *npowf-inv1*:$[\![x \in \mathit{carrier}\ K;\ x \neq \mathbf{0};\ \neg\ 0 \leq j]\!] \Longrightarrow$
$x_K{}^{j} = (x^{\text{-}K})_K{}^{(-j)}$
apply (*simp add*:*npowf-def*)
done

lemma (**in** *Corps*) *npowf-inverse*:$[\![x \in \mathit{carrier}\ K;\ x \neq \mathbf{0}]\!] \Longrightarrow x_K{}^{j} = (x^{\text{-}K})_K{}^{(-j)}$
apply (*case-tac* $0 \leq j$)
apply (*simp add*:*npowf-inv*, *simp add*:*npowf-inv1*)
done

lemma (**in** *Corps*) *npowf-expTr1*:$[\![x \in \mathit{carrier}\ K;\ x \neq \mathbf{0};\ 0 \leq i;\ 0 \leq j;$
$0 \leq i - j]\!] \Longrightarrow x_K{}^{(i - j)} = x_K{}^{i} \cdot_r x_K{}^{(-\ j)}$
apply (*simp add*:*npow-exp-minusTr2*)
apply (*simp add*:*npowf-def*)
done

lemma (**in** *Corps*) *npowf-expTr2*:$[\![x \in \mathit{carrier}\ K;\ x \neq \mathbf{0};\ 0 \leq i + j]\!] \Longrightarrow$
$x_K{}^{(i + j)} = x_K{}^{i} \cdot_r x_K{}^{j}$
apply (*cut-tac field-is-ring*)
 apply (*case-tac* $0 \leq i$)
  apply (*case-tac* $0 \leq j$)
  apply (*simp add*:*npowf-def*, *simp add*:*nat-add-distrib*,
 rule Ring.npMulDistr[*THEN sym*], *assumption*+)
 apply (*subst zminus-minus*[*THEN sym*, *of i j*],
 subst npow-exp-minusTr2[*of x i* $-j$], *assumption*+)
  apply (*simp add*:*zle*, *simp add*:*zless-imp-zle*, *simp add*:*npowf-def*)
 apply (*simp add*:*add.commute*[*of i j*],
 subst zminus-minus[*THEN sym*, *of j i*],
 subst npow-exp-minusTr2[*of x j* $-i$], *assumption*+)
  apply (*simp add*:*zle*, *simp add*:*zless-imp-zle*, *simp*)
  apply (*frule npowf-mem*[*of x i*], *assumption*+,
 frule npowf-mem[*of x j*], *assumption*+,
 simp add:*field-tOp-commute*[*of* $x_K{}^{i}$ $x_K{}^{j}$])
  apply (*simp add*:*npowf-def*)
done

lemma (**in** *Corps*) *npowf-exp-add*:$[\![x \in \mathit{carrier}\ K;\ x \neq \mathbf{0}]\!] \Longrightarrow$
$x_K{}^{(i + j)} = x_K{}^{i} \cdot_r x_K{}^{j}$
apply (*case-tac* $0 \leq i + j$)
apply (*simp add*:*npowf-expTr2*)
apply (*simp add*:*npowf-inv1*[*of x i* $+ j$])
 apply (*simp add*:*zle*)

apply (*subgoal-tac* $0 < -i + -j$) **prefer** *2* **apply** *simp*
apply (*thin-tac* $i + j < 0$)
apply (*frule zless-imp-zle*[*of* 0 $-i + -j$])
apply (*thin-tac* $0 < -i + -j$)
apply (*cut-tac invf-closed1*[*of* x])
apply (*simp*, *erule conjE*,
  frule npowf-expTr2[*of* x ^K $-i$ $-j$], *assumption*+)
apply (*simp add*:*zdiff*[*THEN sym*])
apply (*simp add*:*npowf-inverse*, *simp*)
done

lemma (**in** *Corps*) *npowf-exp-1-add*:⟦$x \in$ *carrier* K; $x \neq \mathbf{0}$⟧ ⟹
  $x_K{}^{(1 + j)} = x \cdot_r x_K{}^j$
apply (*simp add*:*npowf-exp-add*[*of* x 1 j])
apply (*cut-tac field-is-ring*)
apply (*simp add*:*npowf-def*, *simp add*:*Ring.ring-l-one*)
done

lemma (**in** *Corps*) *npowf-minus*:⟦$x \in$ *carrier* K; $x \neq \mathbf{0}$⟧ ⟹ $(x_K{}^j)$^K $= x_K{}^{(-\ j)}$
apply (*frule npowf-exp-add*[*of* x j $-j$], *assumption*+)
apply (*simp add*:*field-npowf-exp-zero*)
apply (*cut-tac field-is-ring*)
apply (*frule npowf-mem*[*of* x j], *assumption*+)
apply (*frule field-potent-nonzero1*[*of* x j], *assumption*+)
apply (*cut-tac invf-closed1*[*of* $x_K{}^j$], *simp*, *erule conjE*,
  frule Ring.ring-r-one[*of* K $(x_K{}^j)$^K], *assumption*, *simp*,
  thin-tac $1_r = x_K{}^j \cdot_r x_K{}^{-\ j}$,
  frule npowf-mem[*of* x $-j$], *assumption*+)
apply (*simp add*:*field-tOp-assoc*[*THEN sym*], *simp add*:*linvf*,
  simp add:*Ring.ring-l-one*, *simp*)
done

lemma (**in** *Ring*) *residue-fieldTr*:⟦*maximal-ideal* R mx; $x \in$ *carrier*(*qring* R mx);
 $x \neq \mathbf{0}_{(qring\ R\ mx)}$⟧ ⟹∃ y∈*carrier* (*qring* R mx). $y \cdot_{r(qring\ R\ mx)} x = 1_{r(qring\ R\ mx)}$
apply (*frule maximal-ideal-ideal*[*of* mx])
apply (*simp add*:*qring-carrier*)
apply (*simp add*:*qring-zero*)
apply (*simp add*:*qring-def*)
apply (*erule bexE*)
apply (*frule sym*, *thin-tac* $a \uplus_R mx = x$, *simp*)
apply (*frule-tac* $a = a$ **in** *ar-coset-same4-1*[*of* mx], *assumption*+)
apply (*frule-tac* $x = a$ **in** *maximal-prime-Tr0*[*of* mx], *assumption*+)
apply (*cut-tac ring-one*)
apply (*rotate-tac* -2, *frule sym*, *thin-tac* $mx \mp R \diamondsuit_p a = carrier\ R$)
apply (*frule-tac* $B = mx \mp R \diamondsuit_p a$ **in** *eq-set-inc*[*of* 1_r *carrier* R],
    assumption+,
  thin-tac carrier $R = mx \mp R \diamondsuit_p a$)
apply (*frule ideal-subset1*[*of* mx])

apply (*frule-tac* $a = a$ **in** *principal-ideal*,
  frule-tac $I = R \diamondsuit_p a$ **in** *ideal-subset1*)
apply (*cut-tac ring-is-ag*,
  simp add:*aGroup.set-sum*, (*erule bexE*)+)
apply (*thin-tac ideal R* ($R \diamondsuit_p a$), *thin-tac* $R \diamondsuit_p a \subseteq$ *carrier R*,
  simp add:*Rxa-def*, (*erule bexE*)+, *simp*, *thin-tac* $k = r \cdot_r a$)
apply (*frule-tac* $a = r$ **and** $b = a$ **in** *rcostOp*[*of mx*], *assumption*+)
apply (*frule-tac* $x = r$ **and** $y = a$ **in** *ring-tOp-closed*, *assumption*+)
apply (*frule-tac* $a = r \cdot_r a$ **and** $x = h$ **and** $b = 1_r$ **in**
  aGroup.ag-eq-sol2[*of R*], *assumption*+)
 apply (*simp add*:*ideal-subset*) **apply** (*simp add*:*ring-one*, *simp*)
 apply (*frule-tac* $a = h$ **and** $b = 1_r \pm -_a (r \cdot_r a)$ **and** $A = mx$ **in**
    eq-elem-in, *assumption*+)
apply (*frule-tac* $a = r \cdot_r a$ **and** $b = 1_r$ **in** *ar-coset-same1*[*of mx*],
  rule ring-tOp-closed, *assumption*+, *rule ring-one*, *assumption*)
 apply (*frule-tac* $a1 = r \cdot_r a$ **and** $h1 = h$ **in** *aGroup.arcos-fixed*[*THEN sym*,
   of R mx], *unfold ideal-def*, *erule conjE*, *assumption*+,
   thin-tac $R +> mx \wedge (\forall r{\in}$*carrier R*$. \forall x{\in}mx.\ r \cdot_r x \in mx)$,
   thin-tac $x = a \uplus_R mx$,
   thin-tac $1_r = h \pm r \cdot_r a$,
   thin-tac $h = 1_r \pm -_a (r \cdot_r a)$, *thin-tac* $1_r \pm -_a (r \cdot_r a) \in mx$)
 apply (*rename-tac b h k r*) **apply** *simp*
 apply *blast*
done

lemma (**in** *Ring*) *residue-field-cd*:*maximal-ideal R mx* $\Longrightarrow$
  Corps (*qring R mx*)
apply (*rule Corps.intro*)
apply (*rule Ring.qring-ring*, *rule Ring-axioms*)
apply (*simp add*:*maximal-ideal-ideal*)
apply (*simp add*:*residue-fieldTr*[*of mx*])
done

lemma (**in** *Ring*) *maximal-set-idealTr*:
  maximal-set $\{I.$ *ideal R I* $\wedge\ S \cap I = \{\}\}$ *mx* $\Longrightarrow$ *ideal R mx*
by (*simp add*:*maximal-set-def*)

lemma (**in** *Ring*) *maximal-setTr*:$[\![$*maximal-set* $\{I.$ *ideal R I* $\wedge\ S \cap I = \{\}\}$ *mx*;
  ideal R J; *mx* $\subset$ *J* $]\!] \Longrightarrow S \cap J \neq \{\}$
by (*rule contrapos-pp*, *simp*+, *simp add*:*psubset-eq*, *erule conjE*,
  simp add:*maximal-set-def*, *blast*)

lemma (**in** *Ring*) *mulDisj*:⟦*mul-closed-set R S*; $1_r \in S$; $\mathbf{0} \notin S$;
$T = \{I.\ ideal\ R\ I \wedge S \cap I = \{\}\}$; *maximal-set T mx* ⟧ ⟹ *prime-ideal R mx*
apply (*simp add*:*prime-ideal-def*)
apply (*rule conjI*, *simp add*:*maximal-set-def*,
rule conjI, *simp add*:*maximal-set-def*)
apply (*rule contrapos-pp*, *simp*+)
apply ((*erule conjE*)+, *blast*)

apply ((*rule ballI*)+, *rule impI*)
apply (*rule contrapos-pp*, *simp*+, (*erule conjE*)+)
apply (*cut-tac a* = *x* **in** *id-ideal-psub-sum*[*of mx*],
simp add:*maximal-set-def*, *assumption*+,
cut-tac a = *y* **in** *id-ideal-psub-sum*[*of mx*],
simp add:*maximal-set-def*, *assumption*+)
apply (*frule-tac J* = *mx* ∓ *R* ◇$_p$ *x* **in** *maximal-setTr*[*of S mx*],
rule sum-ideals, *simp add*:*maximal-set-def*,
simp add:*principal-ideal*, *assumption*,
thin-tac mx ⊂ *mx* ∓ *R* ◇$_p$ *x*)
apply (*frule-tac J* = *mx* ∓ *R* ◇$_p$ *y* **in** *maximal-setTr*[*of S mx*],
rule sum-ideals, *simp add*:*maximal-set-def*,
simp add:*principal-ideal*, *assumption*,
thin-tac mx ⊂ *mx* ∓ *R* ◇$_p$ *y*)
apply (*frule-tac A* = *S* ∩ (*mx* ∓ *R* ◇$_p$ *x*) **in** *nonempty-ex*,
frule-tac A = *S* ∩ (*mx* ∓ *R* ◇$_p$ *y*) **in** *nonempty-ex*,
(*erule exE*)+, *simp*, (*erule conjE*)+)
apply (*rename-tac x y s1 s2*,
thin-tac S ∩ (*mx* ∓ *R* ◇$_p$ *x*) ≠ {},
thin-tac S ∩ (*mx* ∓ *R* ◇$_p$ *y*) ≠ {})
apply (*frule maximal-set-idealTr*,
frule-tac a = *x* **in** *principal-ideal*,
frule-tac a = *y* **in** *principal-ideal*,
frule ideal-subset1[*of mx*],
frule-tac I = *R* ◇$_p$ *x* **in** *ideal-subset1*,
frule-tac I = *R* ◇$_p$ *y* **in** *ideal-subset1*)
apply (*cut-tac ring-is-ag*,
simp add:*aGroup.set-sum*[*of R mx*],
erule bexE, *erule bexE*, *simp*)
apply (*frule-tac s* = *s1* **and** *t* = *s2* **in** *mul-closed-set-tOp-closed*, *simp*,
assumption, *simp*,
frule-tac c = *h* **in** *subsetD*[*of mx carrier R*], *assumption*+,
frule-tac c = *k* **and** *A* = *R* ◇$_p$ *x* **in** *subsetD*[*of - carrier R*],
assumption+)
apply (
cut-tac mul-closed-set-sub,
frule-tac c = *s2* **in** *subsetD*[*of S carrier R*], *assumption*+,
simp add:*ring-distrib2*)
apply ((*erule bexE*)+, *simp*,
frule-tac c = *ha* **in** *subsetD*[*of mx carrier R*], *assumption*+,

frule-tac $c = ka$ **and** $A = R \diamondsuit_p y$ **in** *subsetD*[*of* - *carrier R*],
assumption+,
simp add:*ring-distrib1*)
apply (*frule-tac* $x = h$ **and** $r = ha$ **in** *ideal-ring-multiple1*[*of mx*], *assumption*+)
apply (*frule-tac* $x = h$ **and** $r = ka$ **in** *ideal-ring-multiple1*[*of mx*], *assumption*+,
frule-tac $x = ha$ **and** $r = k$ **in** *ideal-ring-multiple*[*of mx*], *assumption*+)
apply (*frule-tac* $a = x$ **and** $b = y$ **and** $x = k$ **and** $y = ka$ **in**
mul-two-principal-idealsTr, *assumption*+,
erule bexE,
frule-tac $x = x \cdot_r y$ **and** $r = r$ **in** *ideal-ring-multiple*[*of mx*],
assumption+,
rotate-tac -2, *frule sym*, *thin-tac* $k \cdot_r ka = r \cdot_r (x \cdot_r y)$, *simp*)
apply (*frule-tac* $x = h \cdot_r ha \pm h \cdot_r ka$ **and** $y = k \cdot_r ha \pm k \cdot_r ka$ **in**
ideal-pOp-closed[*of mx*])
apply (*rule ideal-pOp-closed*, *assumption*+)+
apply (*simp add*:*maximal-set-def*)
apply *blast*
apply *assumption*
done

lemma (**in** *Ring*) *ex-mulDisj-maximal*:⟦*mul-closed-set R S*; $\mathbf{0} \notin S$; $1_r \in S$;
$T = \{I.\ ideal\ R\ I \wedge S \cap I = \{\}\}$⟧ $\Longrightarrow$ $\exists mx.\ maximal\text{-}set\ T\ mx$
apply (*cut-tac* A=$\{\ I.\ ideal\ R\ I \wedge S \cap I = \{\}\}$ **in** *Zorn-Lemma2*)
prefer *2*
apply (*simp add*:*maximal-set-def*)

apply (*rule ballI*)
apply (*case-tac* $C = \{\}$)
apply (*cut-tac zero-ideal*, *blast*)

apply (*subgoal-tac* $C \in chains\ \{I.\ ideal\ R\ I \wedge I \subset carrier\ R\}$)
apply (*frule chains-un*, *assumption*)
apply (*subgoal-tac* $S \cap (\bigcup\ C) = \{\}$)
apply (*subgoal-tac* $\forall x \in C.\ x \subseteq \bigcup\ C$, *blast*)
apply (*rule ballI*, *rule subsetI*, *simp add*:*CollectI*)
apply *blast*

apply (*rule contrapos-pp*, *simp*+)
apply (*frule-tac* $A = S$ **and** $B = \bigcup\ C$ **in** *nonempty-int*)
apply (*erule exE*)
apply (*simp*, *erule conjE*, *erule bexE*)
apply (*simp add*:*chains-def*, *erule conjE*)
apply (*frule-tac* $c = X$ **and** $A = C$ **and** $B = \{I.\ ideal\ R\ I \wedge S \cap I = \{\}\}$ **in**
subsetD, *assumption*+,
thin-tac $C \subseteq \{I.\ ideal\ R\ I \wedge I \subset carrier\ R\}$,
thin-tac $C \subseteq \{I.\ ideal\ R\ I \wedge S \cap I = \{\}\}$)
apply (*simp*, *blast*)

apply (*simp add*:*chains-def chain-subset-def*, *erule conjE*)

apply (*rule subsetI*)
apply (*frule-tac* $c = x$ **and** $A = C$ **and** $B = \{I.\ ideal\ R\ I \wedge S \cap I = \{\}\}$ **in**
subsetD, *assumption*+,
thin-tac $C \subseteq \{I.\ ideal\ R\ I \wedge S \cap I = \{\}\}$,
thin-tac $T = \{I.\ ideal\ R\ I \wedge S \cap I = \{\}\}$)
apply (*simp*, *thin-tac* $\forall x{\in}C.\ \forall y{\in}C.\ x \subseteq y \vee y \subseteq x$, *erule conjE*)
apply (*simp add*:*psubset-eq ideal-subset1*)
apply (*rule contrapos-pp*, *simp*+)
apply (*rotate-tac* -1, *frule sym*, *thin-tac* $x = carrier\ R$,
thin-tac carrier $R = x$)
apply (*cut-tac ring-one*, *blast*)
done

lemma (**in** *Ring*) *ex-mulDisj-prime*:$[\![$*mul-closed-set* $R\ S$; $\mathbf{0} \notin S$; $1_r \in S]\!] \Longrightarrow$
$\exists\, mx.\ prime\text{-}ideal\ R\ mx \wedge S \cap mx = \{\}$
apply (*frule ex-mulDisj-maximal*[*of* S $\{I.\ ideal\ R\ I \wedge S \cap I = \{\}\}$],
assumption+, *simp*, *erule exE*)
apply (*frule-tac* $mx = mx$ **in** *mulDisj* [*of* S $\{I.\ ideal\ R\ I \wedge S \cap I = \{\}\}$],
assumption+, *simp*, *assumption*)
apply (*simp add*:*maximal-set-def*, (*erule conjE*)+, *blast*)
done

lemma (**in** *Ring*) *nilradTr1*:$\neg$ *zeroring* $R \Longrightarrow nilrad\ R = \bigcap \{p.\ prime\text{-}ideal\ R\ p\}$
apply (*rule equalityI*)

apply (*rule subsetI*)
apply (*simp add*:*nilrad-def CollectI nilpotent-def*)
apply (*erule conjE*, *erule exE*)
apply (*rule allI*, *rule impI*)
apply (*frule-tac prime-ideal-ideal*)
apply (*frule sym*, *thin-tac* $x\hat{}^{R}\ n = \mathbf{0}$, *frule ideal-zero*, *simp*)
apply (*case-tac* $n = 0$, *simp*)
apply (*frule Zero-ring1*[*THEN not-sym*], *simp*)
apply (*rule-tac* $P = xa$ **and** $x = x$ **in** *npow-in-prime*,*assumption*+, *blast*)

apply (*rule subsetI*)
apply (*rule contrapos-pp*, *simp*+)
apply (*frule id-maximal-Exist*, *erule exE*,
frule maximal-is-prime)
apply (*frule-tac* $a = I$ **in** *forall-spec*, *assumption*,
frule-tac $I = I$ **in** *prime-ideal-ideal*,
frule-tac $h = x$ **and** $I = I$ **in** *ideal-subset*, *assumption*)
apply (*subgoal-tac* $\mathbf{0} \notin \{s.\ \exists n.\ s = npow\ R\ x\ n\} \wedge$
$1_r \in \{s.\ \exists n.\ s = npow\ R\ x\ n\}$)
apply (*subgoal-tac mul-closed-set* R $\{s.\ \exists n.\ s = npow\ R\ x\ n\}$)
apply (*erule conjE*)
apply (*frule-tac* $S = \{s.\ \exists n.\ s = npow\ R\ x\ n\}$ **in** *ex-mulDisj-prime*,
assumption+, *erule exE*, *erule conjE*)
apply (*subgoal-tac* $x \in \{s.\ \exists n.\ s = x\hat{}^{R}\ n\}$, *blast*)

apply *simp*
apply (*cut-tac* $t = x$ **in** *np-1*[*THEN sym*], *assumption*, *blast*)

apply (*thin-tac* $\mathbf{0} \notin \{s.\ \exists n.\ s = x\hat{}^{R}\ n\} \wedge 1_r \in \{s.\ \exists n.\ s = x\hat{}^{R}\ n\}$,
thin-tac $\forall xa.$ *prime-ideal R xa* $\longrightarrow x \in xa$)
apply (*subst mul-closed-set-def*)
apply (*rule conjI*)
apply (*rule subsetI*, *simp*, *erule exE*)
apply (*simp add*:*npClose*)
apply ((*rule ballI*)+, *simp*, (*erule exE*)+, *simp*)
apply (*simp add*:*npMulDistr*, *blast*)

apply (*rule conjI*)
apply *simp*
apply (*rule contrapos-pp*, *simp*+, *erule exE*)
apply (*frule sym*, *thin-tac* $\mathbf{0} = x\hat{}^{R}\ n$)
apply (*simp add*:*nilrad-def nilpotent-def*)

apply *simp*
apply (*cut-tac x1* $= x$ **in** *npow-0*[*THEN sym*, *of R*], *blast*)
done

lemma (**in** *Ring*) *nonilp-residue-nilrad*:$[\![\neg$ *zeroring R*; $x \in$ *carrier R*;
nilpotent (*qring R* (*nilrad R*)) ($x \uplus_R$ (*nilrad R*))$]\!] \Longrightarrow$
$x \uplus_R$ (*nilrad R*) $= \mathbf{0}_{(qring\ R\ (nilrad\ R))}$
apply (*simp add*:*nilpotent-def*)
apply (*erule exE*)
apply (*cut-tac id-nilrad-ideal*)
apply (*simp add*:*qring-zero*)
apply (*cut-tac Ring*)
apply (*simp add*:*npQring*)
apply (*frule-tac* $x = x$ **and** $n = n$ **in** *npClose*)
apply (*frule-tac* $I =$ *nilrad R* **and** $a = x\hat{}^{R}\ n$ **in** *ar-coset-same3*,
assumption+)
apply (*rule-tac* $I =$ *nilrad R* **and** $a = x$ **in** *ar-coset-same4*, *assumption*)
apply (*thin-tac* $x\hat{}^{R}\ n \uplus_R$ *nilrad R* = *nilrad R*,
simp add:*nilrad-def nilpotent-def*, *erule exE*)
apply (*simp add*:*npMulExp*, *blast*)
done

lemma (**in** *Ring*) *ex-contid-maximal*:$[\![$ $S = \{1_r\}$; $\mathbf{0} \notin S$; *ideal R I*; $I \cap S = \{\}$;
$T = \{J.$ *ideal R J* $\wedge$ $S \cap J = \{\} \wedge I \subseteq J\}]\!] \Longrightarrow \exists mx.$ *maximal-set T mx*
apply (*cut-tac A*=$\{J.$ *ideal R J* $\wedge$ $S \cap J = \{\} \wedge I \subseteq J\}$ **in** *Zorn-Lemma2*)
apply (*rule ballI*)
apply (*case-tac* $C = \{\}$)
apply *blast*

apply (*subgoal-tac* $\bigcup C \in \{J.$ *ideal R J* $\wedge$ $S \cap J = \{\} \wedge I \subseteq J\} \wedge$

$(\forall x{\in}C.\ x \subseteq \bigcup C))$

apply *blast*
apply (*rule conjI*,
simp add:*CollectI*)
apply (*subgoal-tac* $C \in$ *chains* $\{I.$ *ideal* $R\ I \wedge I \subset$ *carrier* $R\}$)
apply (*rule conjI*,
simp add:*chains-un*)
apply (*rule conjI*)
apply (*rule contrapos-pp*, *simp*+, *erule bexE*)
apply (*thin-tac* $C \in$ *chains* $\{I.$ *ideal* $R\ I \wedge I \subset$ *carrier* $R\}$)
apply (*simp add*:*chains-def*, *erule conjE*)
apply (*frule-tac* $c = x$ **and** $A = C$ **and** $B = \{J.$ *ideal* $R\ J \wedge 1_r \notin J \wedge I \subseteq J\}$
in *subsetD*, *assumption*+, *simp*,
thin-tac $C \in$ *chains* $\{I.$ *ideal* $R\ I \wedge I \subset$ *carrier* $R\}$)
apply (*frule-tac* $A = C$ **in** *nonempty-ex*, *erule exE*, *simp add*:*chains-def*,
erule conjE,
frule-tac $c = x$ **and** $A = C$ **and** $B = \{J.$ *ideal* $R\ J \wedge 1_r \notin J \wedge I \subseteq J\}$ **in**
subsetD, *assumption*+, *simp*, (*erule conjE*)+)
apply (*rule-tac* $A = I$ **and** $B = x$ **and** $C = \bigcup C$ **in** *subset-trans*, *assumption*,
rule-tac $B = x$ **and** $A = C$ **in** *Union-upper*, *assumption*+)
apply (*simp add*:*chains-def*, *erule conjE*)
apply (*rule subsetI*, *simp*)
apply (*frule-tac* $c = x$ **and** $A = C$ **and** $B = \{J.$ *ideal* $R\ J \wedge 1_r \notin J \wedge I \subseteq J\}$
in *subsetD*, *assumption*+, *simp*, (*erule conjE*)+)
apply (*subst psubset-eq*, *simp add*:*ideal-subset1*)
apply (*rule contrapos-pp*, *simp*+, *simp add*:*ring-one*)

apply (*rule ballI*)
apply (*rule Union-upper*, *assumption*)
apply (*erule bexE*)
apply (*simp add*:*maximal-set-def*)
apply *blast*
done

lemma (**in** *Ring*) *contid-maximal*:$[\![S = \{1_r\};\ \mathbf{0} \notin S;$ *ideal* $R\ I;\ I \cap S = \{\};$
$T = \{J.$ *ideal* $R\ J \wedge S \cap J = \{\} \wedge I \subseteq J\};$ *maximal-set* $T\ mx]\!] \Longrightarrow$
maximal-ideal $R\ mx$
apply (*simp add*:*maximal-set-def maximal-ideal-def*)
apply (*erule conjE*)+
apply (*rule equalityI*)

apply (*rule subsetI*, *simp add*:*CollectI*, *erule conjE*)
apply (*case-tac* $x = mx$, *simp*, *simp*)
apply (*subgoal-tac* $1_r \in x$)
apply (*rule-tac* $I = x$ **in** *ideal-inc-one*, *assumption*+)
apply (*rule contrapos-pp*, *simp*+)
apply (*drule spec*[*of* - *mx*])
apply (*simp add*:*whole-ideal*,
rule subsetI, *rule ideal-subset*[*of mx*], *assumption*+)

done

lemma (**in** *Ring*) *ideal-contained-maxid*:⟦¬(*zeroring R*); *ideal R I*; $1_r \notin I$⟧ ⟹
$\exists\, mx.$ *maximal-ideal R mx* ∧ $I \subseteq mx$
apply (*cut-tac ex-contid-maximal*[*of* $\{1_r\}$ *I*
$\{J.$ *ideal R J* ∧ $\{1_r\} \cap J = \{\} \wedge I \subseteq J\}$])
apply (*erule exE*,
cut-tac mx = *mx* **in** *contid-maximal*[*of* $\{1_r\}$ *I*
$\{J.$ *ideal R J* ∧ $\{1_r\} \cap J = \{\} \wedge I \subseteq J\}$])
apply *simp*
apply (*frule Zero-ring1*, *simp*,
assumption, *simp*, *simp*, *simp*,
simp add:*maximal-set-def*, (*erule conjE*)+, *blast*,
simp, *frule Zero-ring1*, *simp*)
apply (*assumption*, *simp*, *simp*)
done

lemma (**in** *Ring*) *nonunit-principal-id*:⟦*a* ∈ *carrier R*; ¬ (*Unit R a*)⟧ ⟹
(*R* $\diamondsuit_p$ *a*) ≠ (*carrier R*)
apply (*rule contrapos-pp*, *simp*+)
apply (*frule sym*, *thin-tac R* $\diamondsuit_p$ *a* = *carrier R*)
apply (*cut-tac ring-one*)
apply (*frule eq-set-inc*[*of* 1_r *carrier R R* $\diamondsuit_p$ *a*], *assumption*,
thin-tac carrier R = *R* $\diamondsuit_p$ *a*, *thin-tac* 1_r ∈ *carrier R*)
apply (*simp add*:*Rxa-def*, *erule bexE*, *simp add*:*ring-tOp-commute*[*of* - *a*],
frule sym, *thin-tac* 1_r = *a* ·$_r$ *r*)
apply (*simp add*:*Unit-def*)
done

lemma (**in** *Ring*) *nonunit-contained-maxid*:⟦¬(*zeroring R*); *a* ∈ *carrier R*;
¬ *Unit R a* ⟧ ⟹ $\exists\, mx.$ *maximal-ideal R mx* ∧ *a* ∈ *mx*
apply (*frule principal-ideal*[*of a*],
frule ideal-contained-maxid[*of R* $\diamondsuit_p$ *a*], *assumption*)
apply (*rule contrapos-pp*, *simp*+,
frule ideal-inc-one[*of R* $\diamondsuit_p$ *a*], *assumption*,
simp add:*nonunit-principal-id*)
apply (*erule exE*, *erule conjE*)
apply (*frule a-in-principal*[*of a*])
apply (*frule-tac B* = *mx* **in** *subsetD*[*of R* $\diamondsuit_p$ *a* - *a*], *assumption*, *blast*)
done

definition
local-ring :: - ⇒ *bool* **where**
local-ring R == *Ring R* ∧ ¬ *zeroring R* ∧ *card* {*mx. maximal-ideal R mx*} = *1*

lemma (**in** *Ring*) *local-ring-diff*:⟦¬ *zeroring R*; *ideal R mx*; *mx* ≠ *carrier R*;
∀ *a*∈ (*carrier R* − *mx*). *Unit R a* ⟧ ⟹ *local-ring R* ∧ *maximal-ideal R mx*
apply (*subgoal-tac* {*mx*} = {*m. maximal-ideal R m*})
apply (*cut-tac singletonI*[*of mx*], *simp*)

apply (*frule sym*, *thin-tac* $\{mx\} = \{m.\ maximal\text{-}ideal\ R\ m\}$)
apply (*simp add*:*local-ring-def*, *simp add*:*Ring*)
apply (*rule equalityI*)
apply (*rule subsetI*, *simp*)
apply (*simp add*:*maximal-ideal-def*)
apply (*simp add*:*ideal-inc-one1*[*of mx*, *THEN sym*])
apply (*thin-tac* $x = mx$, *simp*)
apply (*rule equalityI*)
apply (*rule subsetI*, *simp*, *erule conjE*)
apply (*case-tac* $x \neq mx$)
apply (*frule-tac* $A = x$ **and** $B = mx$ **in** *sets-not-eq*, *assumption*)
apply (*erule bexE*)
apply (*frule-tac* $h = a$ **and** $I = x$ **in** *ideal-subset*, *assumption*+)
apply (*frule-tac* $x = a$ **in** *bspec*, *simp*)
apply (*frule-tac* $I = x$ **and** $a = a$ **in** *ideal-inc-unit1*, *assumption*+,
simp)
apply *simp*

apply (*rule subsetI*, *simp*)
apply (*erule disjE*)
apply *simp*
apply (*simp add*:*whole-ideal ideal-subset1*)

apply (*rule subsetI*)
apply *simp*
apply (*subgoal-tac* $x \subseteq mx$,
thin-tac $\forall a{\in}carrier\ R - mx.\ Unit\ R\ a$,
simp add:*maximal-ideal-def*, (*erule conjE*)+)
apply (*subgoal-tac* $mx \in \{J.\ ideal\ R\ J \wedge x \subseteq J\}$, *simp*)
apply (*thin-tac* $\{J.\ ideal\ R\ J \wedge x \subseteq J\} = \{x, carrier\ R\}$)
apply *simp*

apply (*rule contrapos-pp*, *simp*+)
apply (*simp add*:*subset-eq*, *erule bexE*)
apply (*frule-tac* $mx = x$ **in** *maximal-ideal-ideal*,
frule-tac $x = xa$ **in** *bspec*,
thin-tac $\forall a{\in}carrier\ R - mx.\ Unit\ R\ a$, *simp*,
simp add:*ideal-subset*)
apply (*frule-tac* $I = x$ **and** $a = xa$ **in** *ideal-inc-unit*, *assumption*+,
simp add:*maximal-ideal-def*)
done

lemma (**in** *Ring*) *localring-unit*:$[\![\neg\ zeroring\ R;\ maximal\text{-}ideal\ R\ mx;$
$\forall x.\ x \in mx \longrightarrow Unit\ R\ (x \pm 1_r)\ ]\!] \Longrightarrow local\text{-}ring\ R$
apply (*frule maximal-ideal-ideal*[*of mx*])
apply (*frule local-ring-diff*[*of mx*], *assumption*)
apply (*simp add*:*maximal-ideal-def*, *erule conjE*)
apply (*simp add*:*ideal-inc-one1*[*THEN sym*, *of mx*])
apply (*rule ballI*, *simp*, *erule conjE*)

apply (*frule-tac* $x = a$ **in** *maximal-prime-Tr0*[*of* mx], *assumption*+)

apply (*frule sym*, *thin-tac* $mx \mp R \diamondsuit_p a = carrier\ R$,
cut-tac ring-one,
frule-tac $a = 1_r$ **and** $A = carrier\ R$ **and** $B = mx \mp R \diamondsuit_p a$ **in**
eq-set-inc, *assumption*+,
thin-tac carrier $R = mx \mp R \diamondsuit_p a$)
apply (*frule-tac* $a = a$ **in** *principal-ideal*,
frule ideal-subset1[*of* mx],
frule-tac $I = R \diamondsuit_p a$ **in** *ideal-subset1*)
apply (*cut-tac ring-is-ag*,
simp add:*aGroup.set-sum*, (*erule bexE*)+)
apply (*simp add*:*Rxa-def*, *erule bexE*, *simp*)
apply (*frule sym*, *thin-tac* $1_r = h \pm r \cdot_r a$,
frule-tac $x = r$ **and** $y = a$ **in** *ring-tOp-closed*, *assumption*+,
frule-tac $h = h$ **in** *ideal-subset*[*of* mx], *assumption*+)
apply (*frule-tac* $I = mx$ **and** $x = h$ **in** *ideal-inv1-closed*, *assumption*)
apply (*frule-tac* $a = -_a h$ **in** *forall-spec*, *assumption*,
thin-tac $\forall x.\ x \in mx \longrightarrow Unit\ R\ (x \pm (h \pm r \cdot_r a))$,
thin-tac $h \pm r \cdot_r a = 1_r$)
apply (*frule-tac* $h = -_a h$ **in** *ideal-subset*[*of* mx], *assumption*,
frule-tac $x1 = -_a h$ **and** $y1 = h$ **and** $z1 = r \cdot_r a$ **in**
aGroup.ag-pOp-assoc[*THEN sym*], *assumption*+,
simp add:*aGroup.ag-l-inv1 aGroup.ag-l-zero*,
thin-tac $k = r \cdot_r a$, *thin-tac* $h \pm r \cdot_r a \in carrier\ R$,
thin-tac $h \in carrier\ R$, *thin-tac* $-_a h \in mx$,
thin-tac $-_a h \pm (h \pm r \cdot_r a) = r \cdot_r a$)
apply (*simp add*:*ring-tOp-commute*, *simp add*:*Unit-def*, *erule bexE*,
simp add:*ring-tOp-assoc*,
frule-tac $x = r$ **and** $y = b$ **in** *ring-tOp-closed*, *assumption*+, *blast*)
apply *simp*
done

definition
J-rad ::- $\Rightarrow$ *'a set* **where**
J-rad R = (*if* (*zeroring* R) *then* (*carrier* R) *else*
$\bigcap$ {mx. *maximal-ideal* R mx})

lemma (**in** *Ring*) *zeroring-J-rad-empty*:*zeroring* $R \Longrightarrow$ *J-rad* R = *carrier* R
by (*simp add*:*J-rad-def*)

lemma (**in** *Ring*) *J-rad-mem*:$x \in$ *J-rad* $R \Longrightarrow x \in carrier\ R$
apply (*simp add*:*J-rad-def*)
apply (*case-tac zeroring* R, *simp*)
apply *simp*
apply (*frule id-maximal-Exist*, *erule exE*)
apply (*frule-tac* $a = I$ **in** *forall-spec*, *assumption*,

thin-tac $\forall$ *xa. maximal-ideal R xa* $\longrightarrow$ $x \in xa$)
apply (*frule maximal-ideal-ideal*,
simp add:*ideal-subset*)
done

lemma (**in** *Ring*) *J-rad-unit*:$[\![\neg$ *zeroring R*; $x \in$ *J-rad R*$]\!] \Longrightarrow$
$\forall y.$ ($y \in$ *carrier R* $\longrightarrow$ *Unit R* ($1_r \pm (-_a\ x) \cdot_r y$))
apply (*cut-tac ring-is-ag*,
rule allI, *rule impI*,
rule contrapos-pp, *simp*+)
apply (*frule J-rad-mem*[*of x*],
frule-tac x = *x* **and** *y* = *y* **in** *ring-tOp-closed*, *assumption*,
frule-tac x = $x \cdot_r y$ **in** *aGroup.ag-mOp-closed*, *assumption*+)
apply (*cut-tac ring-one*,
frule-tac x = 1_r **and** *y* = $-_a\ (x \cdot_r y)$ **in** *aGroup.ag-pOp-closed*,
assumption+)
apply (*frule-tac a* = $1_r \pm -_a\ (x \cdot_r y)$ **in** *nonunit-contained-maxid*,
assumption+, *simp add*:*ring-inv1-1*)
apply (*erule exE*, *erule conjE*)
apply (*simp add*:*J-rad-def*,
frule-tac a = *mx* **in** *forall-spec*, *assumption*,
thin-tac $\forall$ *xa. maximal-ideal R xa* $\longrightarrow$ $x \in xa$,
frule-tac mx = *mx* **in** *maximal-ideal-ideal*,
frule-tac I = *mx* **and** *x* = *x* **and** *r* = *y* **in** *ideal-ring-multiple1*,
assumption+)
apply (*frule-tac I* = *mx* **and** *x* = $x \cdot_r y$ **in** *ideal-inv1-closed*,
assumption+)

apply (*frule-tac I* = *mx* **and** *a* = 1_r **and** *b* = $-_a\ (x \cdot_r y)$ **in** *ideal-ele-sumTr2*,
assumption+)
apply (*simp add*:*maximal-ideal-def*)
done

end

theory *Algebra5* **imports** *Algebra4* **begin**

4.6 Operation of ideals

lemma (**in** *Ring*) *ideal-sumTr1*:$[\![$*ideal R A*; *ideal R B*$]\!] \Longrightarrow$
$A \mp B = \bigcap \{J.\ ideal\ R\ J \wedge (A \cup B) \subseteq J\}$
apply (*frule sum-ideals*[*of A B*], *assumption*,
frule sum-ideals-la1[*of A B*], *assumption*,
frule sum-ideals-la2[*of A B*], *assumption*)
apply (*rule equalityI*)

apply (*rule-tac A* = $\{J.\ ideal\ R\ J \wedge (A \cup B) \subseteq J\}$ **and** *C* = $A \mp B$ **in**

Inter-greatest)
apply (*simp*, (*erule conjE*)+)
apply (*rule-tac* $A = A$ **and** $B = B$ **and** $I = X$ **in** *sum-ideals-cont*,
simp add:*ideal-subset1*, *simp add*:*ideal-subset1*, *assumption*+)
apply (*rule-tac* $B = A \mp B$ **and** $A = \{J.\ ideal\ R\ J \wedge (A \cup B) \subseteq J\}$ **in**
Inter-lower)
apply *simp*
done

lemma (**in** *Ring*) *sum-ideals-commute*:$[\![ideal\ R\ A;\ ideal\ R\ B]\!] \Longrightarrow$
$A \mp B = B \mp A$
apply (*frule ideal-sumTr1* [*of B*], *assumption*+,
frule ideal-sumTr1 [*of A B*], *assumption*+)
apply (*simp add*:*Un-commute*[*of B A*])
done

lemma (**in** *Ring*) *ideal-prod-mono1*:$[\![ideal\ R\ A;\ ideal\ R\ B;\ ideal\ R\ C;$
$A \subseteq B\ ]\!] \Longrightarrow A \diamondsuit_r\ C \subseteq B \diamondsuit_r\ C$
apply (*frule ideal-prod-ideal*[*of B C*], *assumption*+)
apply (*rule ideal-prod-subTr*[*of A C B* $\diamondsuit_r$ *C*], *assumption*+)
apply (*rule ballI*)+
apply (*frule-tac* $c = i$ **in** *subsetD*[*of A B*], *assumption*+)
apply (*rule-tac* $i = i$ **and** $j = j$ **in** *prod-mem-prod-ideals*[*of B C*],
assumption+)
done

lemma (**in** *Ring*) *ideal-prod-mono2*:$[\![ideal\ R\ A;\ ideal\ R\ B;\ ideal\ R\ C;$
$A \subseteq B\ ]\!] \Longrightarrow C \diamondsuit_r\ A \subseteq C \diamondsuit_r\ B$
apply (*frule ideal-prod-mono1*[*of A B C*], *assumption*+)
apply (*simp add*:*ideal-prod-commute*)
done

lemma (**in** *Ring*) *cont-ideal-prod*:$[\![ideal\ R\ A;\ ideal\ R\ B;\ ideal\ R\ C;$
$A \subseteq C;\ B \subseteq C\ ]\!] \Longrightarrow A \diamondsuit_r\ B \subseteq C$
apply (*simp add*:*ideal-prod-def*)
apply (*rule subsetI*, *simp*)
apply (*frule ideal-prod-ideal*[*of A B*], *assumption*,
frule-tac $a = A \diamondsuit_r\ B$ **in** *forall-spec*,
thin-tac $\forall xa.\ ideal\ R\ xa \wedge \{x.\ \exists i{\in}A.\ \exists j{\in}B.\ x = i \cdot_r j\} \subseteq xa \longrightarrow x \in xa$,
simp)
apply (*rule subsetI*, *simp*, (*erule bexE*)+, *simp add*:*prod-mem-prod-ideals*)
apply (*frule ideal-prod-la1*[*of A B*], *assumption*,
frule-tac $c = x$ **in** *subsetD*[*of A* $\diamondsuit_r$ *B A*], *assumption*+,
simp add:*subsetD*)
done

lemma (**in** *Ring*) *ideal-distrib*:$[\![ideal\ R\ A;\ ideal\ R\ B;\ ideal\ R\ C]\!] \Longrightarrow$
$A \diamondsuit_r\ (B \mp C) = A \diamondsuit_r\ B \mp A \diamondsuit_r\ C$
apply (*frule sum-ideals*[*of B C*], *assumption*,

frule ideal-prod-ideal[*of* A B $\mp$ C], *assumption*+,
frule ideal-prod-ideal[*of* A B], *assumption*+,
frule ideal-prod-ideal[*of* A C], *assumption*+,
frule sum-ideals[*of* A $\diamondsuit_r$ B A $\diamondsuit_r$ C], *assumption*)
apply (*rule equalityI*)
apply (*rule ideal-prod-subTr*[*of* A B $\mp$ C A $\diamondsuit_r$ B $\mp$ A $\diamondsuit_r$ C], *assumption*+)
apply (*rule ballI*)+
apply (*frule-tac* $x = j$ **in** *ideals-set-sum*[*of* B C], *assumption*+,
(*erule bexE*)+, *simp*) **apply** (
thin-tac $j = h \pm k$,
frule-tac $h = i$ **in** *ideal-subset*[*of* A], *assumption*+,
frule-tac $h = h$ **in** *ideal-subset*[*of* B], *assumption*+,
frule-tac $h = k$ **in** *ideal-subset*[*of* C], *assumption*+)
apply (*simp add*:*ring-distrib1*)
apply (*frule-tac* $i = i$ **and** $j = h$ **in** *prod-mem-prod-ideals*[*of* A B],
assumption+,
frule-tac $i = i$ **and** $j = k$ **in** *prod-mem-prod-ideals*[*of* A C],
assumption+)
apply (*frule sum-ideals-la1*[*of* A $\diamondsuit_r$ B A $\diamondsuit_r$ C], *assumption*+,
frule sum-ideals-la2[*of* A $\diamondsuit_r$ B A $\diamondsuit_r$ C], *assumption*+)
apply (*frule-tac* $c = i \cdot_r h$ **in** *subsetD*[*of* A $\diamondsuit_r$ B A $\diamondsuit_r$ B $\mp$ A $\diamondsuit_r$ C],
assumption+,
frule-tac $c = i \cdot_r k$ **in** *subsetD*[*of* A $\diamondsuit_r$ C A $\diamondsuit_r$ B $\mp$ A $\diamondsuit_r$ C],
assumption+)
apply (*simp add*:*ideal-pOp-closed*)
apply (*rule sum-ideals-cont*[*of* A $\diamondsuit_r$ (B $\mp$ C) A $\diamondsuit_r$ B A $\diamondsuit_r$ C],
assumption+)
apply (*rule ideal-prod-subTr*[*of* A B A $\diamondsuit_r$ (B $\mp$ C)], *assumption*+)
apply (*rule ballI*)+
apply (*frule sum-ideals-la1*[*of* B C], *assumption*+,
frule-tac $c = j$ **in** *subsetD*[*of* B B $\mp$ C], *assumption*+,
rule-tac $i = i$ **and** $j = j$ **in** *prod-mem-prod-ideals*[*of* A B $\mp$ C],
assumption+)

apply (*rule ideal-prod-subTr*[*of* A C A $\diamondsuit_r$ (B $\mp$ C)], *assumption*+)
apply (*rule ballI*)+
apply (*frule sum-ideals-la2*[*of* B C], *assumption*+,
frule-tac $c = j$ **in** *subsetD*[*of* C B $\mp$ C], *assumption*+,
rule-tac $i = i$ **and** $j = j$ **in** *prod-mem-prod-ideals*[*of* A B $\mp$ C],
assumption+)
done

definition
coprime-ideals::[-, *'a set*, *'a set*] $\Rightarrow$ *bool* **where**
coprime-ideals R A B $\longleftrightarrow$ A $\mp_R$ B = *carrier* R

lemma (**in** *Ring*) *coprimeTr*:$[\![$*ideal* R A; *ideal* R $B$$]\!] \Longrightarrow$
coprime-ideals R A B = ($\exists\, a \in A.\ \exists\, b \in B.\ a \pm b = 1_r$)
apply (*rule iffI*)

apply (*simp add*:*coprime-ideals-def*)
apply (*cut-tac ring-one*, *frule sym*, *thin-tac* $A \mp B = carrier\ R$, *simp*,
thin-tac carrier $R = A \mp B$, *frule ideals-set-sum*[*of* $A\ B$],
assumption+, (*erule bexE*)+, *frule sym*, *blast*)

apply (*erule bexE*)+
apply (*frule ideal-subset1*[*of* A], *frule ideal-subset1*[*of* B])
apply (*frule-tac* $a = a$ **and** $b = b$ **in** *set-sum-mem*[*of* - A - B], *assumption*+)
apply (*simp add*:*coprime-ideals-def*)
apply (*frule sum-ideals*[*of* $A\ B$], *assumption*+,
frule ideal-inc-one[*of* $A \mp B$], *assumption*)
apply (*simp add*:*coprime-ideals-def*)
done

lemma (**in** *Ring*) *coprime-int-prod*:⟦*ideal* $R\ A$; *ideal* $R\ B$; *coprime-ideals* $R\ A\ B$⟧
$\Longrightarrow$ $A \cap B = A \diamondsuit_r B$
apply (*frule ideal-prod-la1*[*of* $A\ B$], *assumption*+,
frule ideal-prod-la2[*of* $A\ B$], *assumption*+)
apply (*rule equalityI*)
defer

apply *simp*
apply (*simp add*:*coprime-ideals-def*)
apply (*frule int-ideal*[*of* $A\ B$], *assumption*)
apply (*frule idealprod-whole-r*[*of* $A \cap B$])
apply (*frule sym*, *thin-tac* $A \mp B = carrier\ R$, *simp*)
apply (*simp add*:*ideal-distrib*)
apply (*simp add*:*ideal-prod-commute*[*of* $A \cap B\ A$])
apply (*cut-tac Int-lower1*[*of* $A\ B$], *cut-tac Int-lower2*[*of* $A\ B$])
apply (*frule ideal-prod-mono2*[*of* $A \cap B\ B\ A$], *assumption*+,
frule ideal-prod-mono1[*of* $A \cap B\ A\ B$], *assumption*+)
apply (*frule ideal-prod-ideal*[*of* $A \cap B\ B$], *assumption*+,
frule ideal-prod-ideal[*of* $A\ A \cap B$], *assumption*+,
frule ideal-subset1[*of* $A \diamondsuit_r (A \cap B)$],
frule ideal-subset1[*of* $(A \cap B) \diamondsuit_r B$])
apply (*frule ideal-prod-ideal*[*of* $A\ B$], *assumption*+,
frule sum-ideals-cont[*of* $A \diamondsuit_r B\ A \diamondsuit_r (A \cap B)\ (A \cap B) \diamondsuit_r B$],
assumption+)
apply *simp*
done

lemma (**in** *Ring*) *coprime-elems*:⟦*ideal* $R\ A$; *ideal* $R\ B$; *coprime-ideals* $R\ A\ B$⟧ $\Longrightarrow$
$\exists\, a{\in}A.\ \exists\, b{\in}B.\ a \pm b = 1_r$
by (*simp add*:*coprimeTr*)

lemma (**in** *Ring*) *coprime-elemsTr*:⟦*ideal* $R\ A$; *ideal* $R\ B$; $a{\in}A$; $b{\in}B$; $a \pm b = 1_r$⟧
$\Longrightarrow$ *pj* $R\ A\ b = 1_{r(qring\ R\ A)} \wedge$ *pj* $R\ B\ a = 1_{r(qring\ R\ B)}$
apply (*frule pj-Hom* [*OF Ring*, *of A*],

```
      frule pj-Hom [OF Ring, of B])
 apply (frule qring-ring[of A], frule qring-ring[of B])
 apply (cut-tac ring-is-ag,
       frule Ring.ring-is-ag[of R /r A],
        frule Ring.ring-is-ag[of R /r B])
 apply (frule-tac a = a and b = b in aHom-add[of R R /r A pj R A],
        assumption+,
       simp add:rHom-def, simp add:ideal-subset,
       simp add:ideal-subset, simp)
 apply (frule ideal-subset[of A a], assumption,
       frule ideal-subset[of B b], assumption+)
 apply (simp only:pj-zero[OF Ring, THEN sym, of A a],
       frule rHom-mem[of pj R A R qring R A b], assumption+,
       simp add:aGroup.l-zero[of R /r A])
  apply (simp add:rHom-one[OF Ring, of qring R A])

  apply (frule-tac a = a and b = b in aHom-add[of R R /r B pj R B],
        assumption+,
       simp add:rHom-def, simp add:ideal-subset,
       simp add:ideal-subset, simp)
  apply (simp only:pj-zero[OF Ring, THEN sym, of B b],
       frule rHom-mem[of pj R B R qring R B a], assumption+,
       simp add:aGroup.ag-r-zero[of R /r B])
  apply (simp add:rHom-one[OF Ring, of qring R B])
done

lemma (in Ring) partition-of-unity:⟦ideal R A; a ∈ A; b ∈ carrier R;
      a ± b = 1r; u ∈ carrier R; v ∈ carrier R⟧ ⟹
                                pj R A (a ·r v ± b ·r u ) = pj R A u
apply (frule pj-Hom [OF Ring, of A],
       frule ideal-subset [of A a], assumption+,
       frule ring-tOp-closed[of a v], assumption+,
       frule ring-tOp-closed[of b u], assumption+,
       frule qring-ring[of A])
 apply (simp add:ringhom1[OF Ring, of qring R A a ·r v b ·r u pj R A])
 apply (frule ideal-ring-multiple1[of A a v], assumption+,
        simp add:pj-zero[OF Ring, THEN sym, of A a ·r v],
        frule rHom-mem[of pj R A R qring R A b ·r u], assumption+,
        simp add:Ring.l-zero, simp add:rHom-tOp[OF Ring])
 apply (frule ringhom1[OF Ring, of qring R A a b pj R A], assumption+,
        simp,
        simp add:pj-zero[OF Ring, THEN sym, of A a],
        frule rHom-mem[of pj R A R qring R A b], assumption+,
        simp add:Ring.l-zero)
   apply (simp add:rHom-one[OF Ring, of qring R A pj R A],
          rotate-tac −2, frule sym, thin-tac 1r R /r A = pj R A b,
          simp,
          rule Ring.ring-l-one[of qring R A pj R A u], assumption)
  apply (simp add:rHom-mem)
```

done

lemma (**in** *Ring*) *coprimes-commute*:⟦*ideal R A*; *ideal R B*; *coprime-ideals R A B* ⟧
⟹ *coprime-ideals R B A*
apply (*simp add*:*coprime-ideals-def*)
apply (*simp add*:*sum-ideals-commute*)
done

lemma (**in** *Ring*) *coprime-surjTr*:⟦*ideal R A*; *ideal R B*; *coprime-ideals R A B*;
X ∈ *carrier* (*qring R A*); *Y* ∈ *carrier* (*qring R B*) ⟧ ⟹
∃ *r*∈*carrier R*. *pj R A r* = *X* ∧ *pj R B r* = *Y*
apply (*frule qring-ring* [*of A*],
frule qring-ring [*of B*],
frule coprime-elems [*of A B*], *assumption*+,
frule pj-Hom [*OF Ring*, *of A*],
frule pj-Hom [*OF Ring*, *of B*])
apply (*erule bexE*)+
apply (*simp add*:*qring-carrier*[*of A*],
simp add:*qring-carrier*[*of B*], (*erule bexE*)+,
rename-tac a b u v)
apply (*rotate-tac* −*1*, *frule sym*, *thin-tac v* ⊎$_R$ *B* = *Y*,
rotate-tac −*3*, *frule sym*, *thin-tac u* ⊎$_R$ *A* = *X*, *simp*)
apply (*frule-tac h* = *a* **in** *ideal-subset*[*of A*], *assumption*+,
frule-tac h = *b* **in** *ideal-subset*[*of B*], *assumption*+,
frule-tac x = *a* **and** *y* = *v* **in** *ring-tOp-closed*, *assumption*+,
frule-tac x = *b* **and** *y* = *u* **in** *ring-tOp-closed*, *assumption*+,
cut-tac ring-is-ag,
frule-tac x = *a* ·$_r$ *v* **and** *y* = *b* ·$_r$ *u* **in** *aGroup.ag-pOp-closed*[*of R*],
assumption+)
apply (*frule-tac a* = *a* **and** *b* = *b* **and** *u* = *u* **and** *v* = *v* **in**
partition-of-unity[*of A*], *assumption*+,
simp add:*pj-mem*[*OF Ring*, *of A*])
apply (*frule-tac a* = *b* **and** *b* = *a* **and** *u* = *v* **and** *v* = *u* **in**
partition-of-unity[*of B*], *assumption*+,
subst aGroup.ag-pOp-commute[*of R*], *assumption*+,
simp add:*pj-mem*[*OF Ring*, *of B*])
apply (*frule-tac x* = *b* ·$_r$ *u* **and** *y* = *a* ·$_r$ *v* **in**
aGroup.ag-pOp-commute[*of R*], *assumption*+, *simp*)
apply (*simp add*:*pj-mem*[*OF Ring*], *blast*)
done

lemma (**in** *Ring*) *coprime-n-idealsTr0*:⟦*ideal R A*; *ideal R B*; *ideal R C*;
coprime-ideals R A C; *coprime-ideals R B C* ⟧ ⟹
coprime-ideals R (*A* ◇$_r$ *B*) *C*
apply (*simp add*:*coprimeTr*[*of A C*],
simp add:*coprimeTr*[*of B C*], (*erule bexE*)+)
apply (*cut-tac ring-is-ag*,
frule-tac h = *a* **in** *ideal-subset*[*of A*], *assumption*+,

frule-tac $h = aa$ **in** *ideal-subset*[*of* B], *assumption*+,
frule-tac $h = b$ **in** *ideal-subset*[*of* C], *assumption*+,
frule-tac $h = ba$ **in** *ideal-subset*[*of* C], *assumption*+,
frule-tac $x = a$ **and** $y = b$ **in** *aGroup.ag-pOp-closed*, *assumption*+)
apply (*frule-tac* $x = a \pm b$ **and** $y = aa$ **and** $z = ba$ **in** *ring-distrib1*,
assumption+) **apply** (
rotate-tac 6, *frule sym*, *thin-tac* $a \pm b = 1_r$,
frule sym, *thin-tac* $aa \pm ba = 1_r$)
apply (*simp only*:*ring-distrib2*)
apply (*rotate-tac* -1, *frule sym*, *thin-tac* $1_r = a \pm b$, *simp*,
thin-tac $aa \pm ba = 1_r$)
apply (*simp add*:*ring-r-one*,
thin-tac $a \cdot_r aa \pm b \cdot_r aa \pm (a \cdot_r ba \pm b \cdot_r ba) \in$ *carrier* R,
thin-tac $a \pm b = a \cdot_r aa \pm b \cdot_r aa \pm (a \cdot_r ba \pm b \cdot_r ba)$)
apply (*frule-tac* $x = a$ **and** $y = aa$ **in** *ring-tOp-closed*, *assumption*+,
frule-tac $x = b$ **and** $y = aa$ **in** *ring-tOp-closed*, *assumption*+,
frule-tac $x = a$ **and** $y = ba$ **in** *ring-tOp-closed*, *assumption*+,
frule-tac $x = b$ **and** $y = ba$ **in** *ring-tOp-closed*, *assumption*+,
frule-tac $x = a \cdot_r ba$ **and** $y = b \cdot_r ba$ **in** *aGroup.ag-pOp-closed*,
assumption+)
apply (*simp add*:*aGroup.ag-pOp-assoc*,
frule sym, *thin-tac* $1_r = a \cdot_r aa \pm (b \cdot_r aa \pm (a \cdot_r ba \pm b \cdot_r ba))$)
apply (*frule-tac* $i = a$ **and** $j = aa$ **in** *prod-mem-prod-ideals*[*of* A B],
assumption+)
apply (*frule-tac* $x = ba$ **and** $r = a$ **in** *ideal-ring-multiple*[*of* C],
assumption+,
frule-tac $x = ba$ **and** $r = b$ **in** *ideal-ring-multiple*[*of* C],
assumption+,
frule-tac $x = b$ **and** $r = aa$ **in** *ideal-ring-multiple1*[*of* C],
assumption+)
apply (*frule-tac* $I = C$ **and** $x = a \cdot_r ba$ **and** $y = b \cdot_r ba$ **in**
ideal-pOp-closed, *assumption*+,
frule-tac $I = C$ **and** $x = b \cdot_r aa$ **and** $y = a \cdot_r ba \pm b \cdot_r ba$ **in**
ideal-pOp-closed, *assumption*+)
apply (*frule ideal-prod-ideal*[*of* A B], *assumption*)
apply (*subst coprimeTr*, *assumption*+, *blast*)
done

lemma (**in** *Ring*) *coprime-n-idealsTr1*:*ideal* R $C \Longrightarrow$
$(\forall k \leq n.$ *ideal* R $(J\ k)) \wedge (\forall i \leq n.$ *coprime-ideals* R $(J\ i)$ $C) \longrightarrow$
coprime-ideals R $(i\Pi_{R,n}\ J)$ C
apply (*induct-tac* n)
apply (*rule impI*)
apply (*erule conjE*)
apply *simp*

apply (*rule impI*)
apply (*erule conjE*)
apply (*cut-tac* $n = Suc\ n$ **in** *n-in-Nsetn*)

apply (*cut-tac* $n = n$ **in** *Nset-Suc*) **apply** *simp*
apply (*cut-tac* $n = n$ **and** $J = J$ **in** *n-prod-ideal*,
rule allI, *simp*)
apply (*rule-tac* $A = i\Pi_{R,n}\ J$ **and** $B = J\ (Suc\ n)$ **in**
coprime-n-idealsTr0[*of* - - *C*], *simp*+)
done

lemma (**in** *Ring*) *coprime-n-idealsTr2*:$[\![ideal\ R\ C;\ (\forall k \leq n.\ ideal\ R\ (J\ k));$
$(\forall i \leq n.\ coprime\text{-}ideals\ R\ (J\ i)\ C)\ ]\!] \Longrightarrow$
$coprime\text{-}ideals\ R\ (i\Pi_{R,n}\ J)\ C$
by (*simp add*:*coprime-n-idealsTr1*)

lemma (**in** *Ring*) *coprime-n-idealsTr3*:$(\forall k \leq (Suc\ n).\ ideal\ R\ (J\ k)) \wedge$
$(\forall i \leq (Suc\ n).\ \forall j \leq (Suc\ n).\ i \neq j \longrightarrow$
$coprime\text{-}ideals\ R\ (J\ i)\ (J\ j)) \longrightarrow\ coprime\text{-}ideals\ R\ (i\Pi_{R,n}\ J)\ (J\ (Suc\ n))$
apply (*rule impI*, *erule conjE*)
apply (*case-tac* $n = 0$, *simp*)
apply (*simp add*:*Nset-1*)
apply (*cut-tac nat-eq-le*[*of Suc n Suc n*],
frule-tac $a = Suc\ n$ **in** *forall-spec*, *assumption*)
apply (*rotate-tac 1*, *frule-tac* $a = Suc\ n$ **in** *forall-spec*, *assumption*,
thin-tac $\forall j{\leq}Suc\ n.\ Suc\ n \neq j \longrightarrow coprime\text{-}ideals\ R\ (J\ (Suc\ n))\ (J\ j))$
apply (*rule-tac* $C = J\ (Suc\ n)$ **and** $n = n$ **and** $J = J$ **in** *coprime-n-idealsTr2*,
assumption, *rule allI*, *simp*)
apply (*rule allI*, *rule impI*)
apply (*frule-tac* $a = i$ **in** *forall-spec*, *simp*,
thin-tac $\forall i{\leq}Suc\ n.\ \forall j{\leq}Suc\ n.\ i \neq j \longrightarrow coprime\text{-}ideals\ R\ (J\ i)\ (J\ j)$,
rotate-tac -1,
frule-tac $a = Suc\ n$ **in** *forall-spec*, *assumption*+)
apply *simp*+
done

lemma (**in** *Ring*) *coprime-n-idealsTr4*:$[\![(\forall k \leq (Suc\ n).\ ideal\ R\ (J\ k)) \wedge$
$(\forall i \leq (Suc\ n).\ \forall j \leq (Suc\ n).\ i \neq j \longrightarrow$
$coprime\text{-}ideals\ R\ (J\ i)\ (J\ j))]\!] \Longrightarrow coprime\text{-}ideals\ R\ (i\Pi_{R,n}\ J)\ (J\ (Suc\ n))$
apply (*simp add*:*coprime-n-idealsTr3*)
done

4.7 Direct product1, general case

definition
prod-tOp :: [$'i$ *set*, $'i \Rightarrow ('a, 'm)$ *Ring-scheme*] $\Rightarrow$
$('i \Rightarrow 'a) \Rightarrow ('i \Rightarrow 'a) \Rightarrow\ ('i \Rightarrow 'a)$ **where**
prod-tOp $I\ A = (\lambda f{\in}carr\text{-}prodag\ I\ A.\ \lambda g{\in}carr\text{-}prodag\ I\ A.$
$\lambda x{\in}I.\ (f\ x)\ \cdot_{r(A\ x)}\ (g\ x))$

definition
prod-one::[$'i$ *set*, $'i\ \Rightarrow ('a, 'm)$ *Ring-scheme*] $\Rightarrow ('i \Rightarrow 'a)$ **where**

prod-one I A == $\lambda x{\in}I.\ 1_{r(A\ x)}$

definition
prodrg :: [*'i set*, *'i* ⇒ (*'a*, *'more*) *Ring-scheme*] ⇒ (*'i* ⇒ *'a*) *Ring* **where**
prodrg I A = (|*carrier* = *carr-prodag I A*, *pop* = *prod-pOp I A*, *mop* =
prod-mOp I A, *zero* = *prod-zero I A*, *tp* = *prod-tOp I A*,
un = *prod-one I A* |)

abbreviation
PRODRING ((*rΠ*_/ -) [*72*,*73*]*72*) **where**
$r\Pi_I\ A$ == *prodrg I A*

definition
augm-func :: [*nat*, *nat* ⇒ *'a*,*'a set*, *nat*, *nat* ⇒ *'a*, *'a set*] ⇒ *nat* ⇒ *'a* **where**
augm-func n f A m g B = (λ*i*∈{*j*. *j* ≤ (*n* + *m*)}. *if i* ≤ *n then f i else*
if (*Suc n*) ≤ *i* ∧ *i* ≤ *n* + *m then g* ((*sliden* (*Suc n*)) *i*) *else undefined*)

definition
ag-setfunc :: [*nat*, *nat* ⇒ (*'a*, *'more*) *Ring-scheme*, *nat*,
nat ⇒ (*'a*, *'more*) *Ring-scheme*] ⇒ (*nat* ⇒ *'a*) *set* ⇒ (*nat* ⇒ *'a*) *set*
⇒ (*nat* ⇒ *'a*) *set* **where**
ag-setfunc n B1 m B2 X Y =
{*f*. ∃ *g*. ∃ *h*. (*g*∈*X*) ∧(*h*∈*Y*) ∧(*f* = (*augm-func n g* (*Un-carrier* {*j*. *j* ≤ *n*} *B1*)

m h (*Un-carrier* {*j*. *j* ≤ (*m* − *1*)} *B2*)))}

primrec
ac-fProd-Rg :: [*nat*, *nat* ⇒ (*'a*, *'more*) *Ring-scheme*] ⇒
(*nat* ⇒ *'a*) *set*
where
fprod-0: *ac-fProd-Rg 0 B* = *carr-prodag* {*0*::*nat*} *B*
| *frpod-n*: *ac-fProd-Rg* (*Suc n*) *B* = *ag-setfunc n B* (*Suc 0*) (*compose* {*0*::*nat*}
B (*slide* (*Suc n*))) (*carr-prodag* {*j*. *j* ≤ *n*} *B*) (*carr-prodag* {*0*} (*compose* {*0*} *B*
(*slide* (*Suc n*))))

definition
prodB1 :: [(*'a*, *'m*) *Ring-scheme*, (*'a*, *'m*) *Ring-scheme*] ⇒
(*nat* ⇒ (*'a*, *'m*) *Ring-scheme*) **where**
prodB1 R S = (λ*k*. *if k=0 then R else if k=Suc 0 then S else*
undefined)

definition
Prod2Rg :: [(*'a*, *'m*) *Ring-scheme*, (*'a*, *'m*) *Ring-scheme*]
⇒ (*nat* ⇒ *'a*) *Ring* (**infixl** $\bigoplus_r$ *80*) **where**
A1 $\bigoplus_r$ *A2* = *prodrg* {*0*, *Suc 0*} (*prodB1 A1 A2*)

Don't try $(Prod\text{-}ring\ (Nset\ n)\ B) \bigoplus_r (B\ (Suc\ n))$

lemma *carr-prodrg-mem-eq*:$[\![f \in carrier\ (r\Pi_I\ A);\ g \in carrier\ (r\Pi_I\ A);$
$\forall i{\in}I.\ f\ i = g\ i\ ]\!] \Longrightarrow f = g$
by (*simp add*:*prodrg-def carr-prodag-def*, (*erule conjE*)+,
rule funcset-eq[*of* - *I*], *assumption*+)

lemma *prod-tOp-mem*:$[\![\forall k{\in}I.\ Ring\ (A\ k);\ X \in carr\text{-}prodag\ I\ A;$
$Y \in carr\text{-}prodag\ I\ A]\!] \Longrightarrow prod\text{-}tOp\ I\ A\ X\ Y \in carr\text{-}prodag\ I\ A$
apply (*subst carr-prodag-def*, *simp*)
apply (*rule conjI*)
apply (*simp add*:*prod-tOp-def restrict-def extensional-def*)
apply (*rule conjI*)
apply (*rule Pi-I*)
apply (*simp add*:*Un-carrier-def prod-tOp-def*)
apply (*simp add*:*carr-prodag-def*, (*erule conjE*)+)
apply (*blast dest*: *Ring.ring-tOp-closed del*:*PiE*)

apply (*rule ballI*)
apply (*simp add*:*prod-tOp-def*)
apply (*simp add*:*carr-prodag-def*, (*erule conjE*)+)
apply (*simp add*:*Ring.ring-tOp-closed*)
done

lemma *prod-tOp-func*:$\forall k{\in}I.\ Ring\ (A\ k) \Longrightarrow$
$prod\text{-}tOp\ I\ A \in carr\text{-}prodag\ I\ A \to carr\text{-}prodag\ I\ A \to carr\text{-}prodag\ I\ A$
by (*simp add*:*prod-tOp-mem*)

lemma *prod-one-func*:$\forall k{\in}I.\ Ring\ (A\ k) \Longrightarrow$
$prod\text{-}one\ I\ A \in carr\text{-}prodag\ I\ A$
apply (*simp add*:*prod-one-def carr-prodag-def*)
apply (*rule conjI*)
apply (*rule Pi-I*)
apply (*simp add*:*Un-carrier-def*)
apply (*blast dest*: *Ring.ring-one*)
apply (*simp add*: *Ring.ring-one*)
done

lemma *prodrg-carrier*:$\forall k{\in}I.\ Ring\ (A\ k) \Longrightarrow$
$carrier\ (prodrg\ I\ A) = carrier\ (prodag\ I\ A)$
by (*simp add*:*prodrg-def prodag-def*)

lemma *prodrg-ring*:$\forall k{\in}I.\ Ring\ (A\ k) \Longrightarrow Ring\ (prodrg\ I\ A)$
apply (*rule Ring.intro*)
apply (*simp add*:*prodrg-def*)
apply (*rule prod-pOp-func*,
rule ballI, *simp add*:*Ring.ring-is-ag*)

apply (*simp add*:*prodrg-def*, *rule prod-pOp-assoc*,

simp add:*Ring.ring-is-ag*, *assumption*+)

apply (*simp add*:*prodrg-def*, *rule prod-pOp-commute*,
simp add:*Ring.ring-is-ag*, *assumption*+)

apply (*simp add*:*prodrg-def*, *rule prod-mOp-func*,
simp add:*Ring.ring-is-ag*)

apply (*simp add*:*prodrg-def*)
apply (*cut-tac* $X = a$ **in** *prod-mOp-mem*[*of* I A])
apply (*simp add*:*Ring.ring-is-ag*, *assumption*)
apply (*cut-tac* X = *prod-mOp* I A a **and** $Y = a$ **in** *prod-pOp-mem*[*of* I A],
simp add:*Ring.ring-is-ag*, *assumption*+,
cut-tac prod-zero-func[*of* I A])
apply (*rule carr-prodag-mem-eq*[*of* I A],
simp add:*Ring.ring-is-ag*, *assumption*+,
rule ballI, *simp add*:*prod-pOp-def*,
subst prod-mOp-mem-i, *simp add*:*Ring.ring-is-ag*, *assumption*+,
subst prod-zero-i, *simp add*:*Ring.ring-is-ag*, *assumption*+,
rule aGroup.l-m, *simp add*:*Ring.ring-is-ag*,
simp add:*prodag-comp-i*, *simp add*:*Ring.ring-is-ag*)
apply (*simp add*:*prodrg-def*,
rule prod-zero-func, *simp add*:*Ring.ring-is-ag*)
apply (*simp add*:*prodrg-def*,
cut-tac prod-zero-func[*of* I A],
cut-tac X = *prod-zero* I A **and** $Y = a$ **in** *prod-pOp-mem*[*of* I A],
simp add:*Ring.ring-is-ag*, *assumption*+,
rule carr-prodag-mem-eq[*of* I A],
simp add:*Ring.ring-is-ag*, *assumption*+)
apply (*rule ballI*)
apply (*simp add*:*prod-pOp-def prod-zero-def*)
apply (*rule aGroup.ag-l-zero*, *simp add*:*Ring.ring-is-ag*)
apply (*simp add*:*prodag-comp-i*, *simp add*:*Ring.ring-is-ag*)
apply (*simp add*:*prodrg-def*,
rule prod-tOp-func, *simp add*:*Ring.ring-is-ag*)
apply (*simp add*:*prodrg-def*)
apply (*frule-tac* $X = a$ **and** $Y = b$ **in** *prod-tOp-mem*[*of* I A], *assumption*+,
frule-tac X = *prod-tOp* I A a b **and** $Y = c$ **in**
prod-tOp-mem[*of* I A], *assumption*+,
frule-tac $X = b$ **and** $Y = c$ **in** *prod-tOp-mem*[*of* I A], *assumption*+,
frule-tac $X = a$ **and** Y = *prod-tOp* I A b c **in**
prod-tOp-mem[*of* I A], *assumption*+)
apply (*rule carr-prodag-mem-eq*[*of* I A],
simp add:*Ring.ring-is-ag*, *assumption*+, *rule ballI*)
apply (*simp add*:*prod-tOp-def*)
apply (*rule Ring.ring-tOp-assoc*, *simp*, (*simp add*:*prodag-comp-i*)+)
apply (*simp add*:*prodrg-def*)
apply (*frule-tac* $X = a$ **and** $Y = b$ **in** *prod-tOp-mem*[*of* I A], *assumption*+,
frule-tac $X = b$ **and** $Y = a$ **in** *prod-tOp-mem*[*of* I A], *assumption*+,

rule carr-prodag-mem-eq[*of I A*],
simp add:*Ring.ring-is-ag*, *assumption*+)
apply (*rule ballI*, *simp add*:*prod-tOp-def*)
apply (*rule Ring.ring-tOp-commute*, (*simp add*:*prodag-comp-i*)+)
apply (*simp add*:*prodrg-def*, *rule prod-one-func*, *assumption*)

apply (*simp add*:*prodrg-def*)
apply (*cut-tac* $X = b$ **and** $Y = c$ **in** *prod-pOp-mem*[*of I A*],
simp add:*Ring.ring-is-ag*, *assumption*+,
cut-tac $X = a$ **and** $Y = b$ **in** *prod-tOp-mem*[*of I A*], *assumption*+,
cut-tac $X = a$ **and** $Y = c$ **in** *prod-tOp-mem*[*of I A*], *assumption*+,
cut-tac $X =$ *prod-tOp I A a b* **and** $Y =$ *prod-tOp I A a c* **in**
prod-pOp-mem[*of I A*], *simp add*:*Ring.ring-is-ag*, *assumption*+)
apply (*rule carr-prodag-mem-eq*[*of I A*],
simp add:*Ring.ring-is-ag*, *rule prod-tOp-mem*[*of I A*], *assumption*+)
apply (*rule ballI*, *simp add*:*prod-tOp-def prod-pOp-def*)
apply (*rule Ring.ring-distrib1*, (*simp add*:*prodag-comp-i*)+)

apply (*simp add*:*prodrg-def*,
cut-tac prod-one-func[*of I A*],
cut-tac $X =$ *prod-one I A* **and** $Y = a$ **in** *prod-tOp-mem*[*of I A*],
assumption+)
apply (*rule carr-prodag-mem-eq*[*of I A*],
simp add:*Ring.ring-is-ag*, *assumption*+,
rule ballI, *simp add*:*prod-tOp-def prod-one-def*,
rule Ring.ring-l-one, *simp*, *simp add*:*prodag-comp-i*)
apply *simp*
done

lemma *prodrg-elem-extensional*:⟦∀ $k \in I$. *Ring* (*A k*); $f \in$ *carrier* (*prodrg I A*)⟧
$\Longrightarrow$ $f \in$ *extensional I*
apply (*simp add*:*prodrg-carrier*)
apply (*simp add*:*prodag-def carr-prodag-def*)
done

lemma *prodrg-pOp*:∀ $k \in I$. *Ring* (*A k*) $\Longrightarrow$
pop (*prodrg I A*) = *prod-pOp I A*
apply (*simp add*:*prodrg-def*)
done

lemma *prodrg-mOp*:∀ $k \in I$. *Ring* (*A k*) $\Longrightarrow$
mop (*prodrg I A*) = *prod-mOp I A*
apply (*simp add*:*prodrg-def*)
done

lemma *prodrg-zero*:∀ $k \in I$. *Ring* (*A k*) $\Longrightarrow$
zero (*prodrg I A*) = *prod-zero I A*
apply (*simp add*:*prodrg-def*)
done

lemma *prodrg-tOp*:$\forall\, k{\in}I.\ Ring\ (A\ k) \Longrightarrow$
$tp\ (prodrg\ I\ A) = prod\text{-}tOp\ I\ A$
apply (*simp add*:*prodrg-def*)
done

lemma *prodrg-one*:$\forall\, k{\in}I.\ Ring\ (A\ k) \Longrightarrow$
$un\ (prodrg\ I\ A) = prod\text{-}one\ I\ A$
apply (*simp add*:*prodrg-def*)
done

lemma *prodrg-sameTr5*:$[\![\forall\, k{\in}I.\ Ring\ (A\ k); \forall\, k{\in}I.\ A\ k = B\ k]\!]$
$\Longrightarrow prod\text{-}tOp\ I\ A = prod\text{-}tOp\ I\ B$
apply (*frule prod-tOp-func*)
apply (*subgoal-tac carr-prodag I A = carr-prodag I B*, *simp*)
apply (*frule prod-tOp-func* [*of I B*])
apply (*rule funcset-eq* [*of - carr-prodag I B -*])
apply (*simp add*:*prod-tOp-def extensional-def*)
apply (*simp add*:*prod-tOp-def extensional-def*)
apply (*rule ballI*)
apply (*frule-tac x = x* **in** *funcset-mem* [*of prod-tOp I A carr-prodag I B*
carr-prodag I B → carr-prodag I B], *assumption*+)
apply (*frule-tac x = x* **in** *funcset-mem* [*of prod-tOp I B carr-prodag I B*
carr-prodag I B → carr-prodag I B], *assumption*+)
apply (*thin-tac prod-tOp I A*
$\in$ *carr-prodag I B → carr-prodag I B → carr-prodag I B*)
apply (*thin-tac prod-tOp I B*
$\in$ *carr-prodag I B → carr-prodag I B → carr-prodag I B*)
apply (*rule funcset-eq* [*of - carr-prodag I B*])
apply (*simp add*:*prod-tOp-def extensional-def*)
apply (*simp add*:*prod-tOp-def extensional-def*)
apply (*rule ballI*)
apply (*frule-tac f = prod-tOp I A x* **and** *A = carr-prodag I B* **and**
x = xa **in** *funcset-mem*, *assumption*+)
apply (*frule-tac f = prod-tOp I B x* **and** *A = carr-prodag I B* **and**
x = xa **in** *funcset-mem*, *assumption*+)
apply (*subgoal-tac* $\forall\, k{\in}I.$ *aGroup* (*B k*))
apply (*rule carr-prodag-mem-eq*, *assumption*+)
apply (*rule ballI*, *simp add*:*prod-tOp-def*)
apply (*rule ballI*, *rule Ring.ring-is-ag*, *simp*)
apply (*subgoal-tac* $\forall\, k{\in}I.$ *aGroup* (*A k*))
apply (*simp add*:*prodag-sameTr1*)
apply (*rule ballI*, *rule Ring.ring-is-ag*, *simp*)
done

lemma *prodrg-sameTr6*:$[\![\forall\, k{\in}I.\ Ring\ (A\ k); \forall\, k{\in}I.\ A\ k = B\ k]\!]$
$\Longrightarrow prod\text{-}one\ I\ A = prod\text{-}one\ I\ B$
apply (*frule prod-one-func* [*of I A*])
apply (*cut-tac prodag-sameTr1* [*of I A B*])

apply (*rule carr-prodag-mem-eq*[*of I A prod-one I A prod-one I B*])
apply (*simp add*:*Ring.ring-is-ag*, *assumption*)
apply (*cut-tac prod-one-func* [*of I B*], *simp*)
apply *simp*
apply (*rule ballI*, *simp add*:*prod-one-def*)
apply (*simp add*:*Ring.ring-is-ag*, *simp*)
done

lemma *prodrg-same*:⟦∀ *k*∈*I*. *Ring* (*A k*); ∀ *k*∈*I*. *A k* = *B k*⟧
⟹ *prodrg I A* = *prodrg I B*
apply (*subgoal-tac* ∀ *k*∈*I*. *aGroup* (*A k*))
apply (*frule prodag-sameTr1*, *assumption*+)
apply (*frule prodag-sameTr2*, *assumption*+)
apply (*frule prodag-sameTr3*, *assumption*+)
apply (*frule prodag-sameTr4*, *assumption*+)
apply (*frule prodrg-sameTr5*, *assumption*+)
apply (*frule prodrg-sameTr6*, *assumption*+)
apply (*simp add*:*prodrg-def*)
apply (*rule ballI*, *rule Ring.ring-is-ag*, *simp*)
done

lemma *prodrg-component*:⟦*f* ∈ *carrier* (*prodrg I A*); *i* ∈ *I*⟧ ⟹
f i ∈ *carrier* (*A i*)
by (*simp add*:*prodrg-def carr-prodag-def*)

lemma *project-rhom*:⟦∀ *k*∈*I*. *Ring* (*A k*); *j* ∈ *I*⟧ ⟹
PRoject I A j ∈ *rHom* (*prodrg I A*) (*A j*)
apply (*simp add*:*rHom-def*)
apply (*rule conjI*)
apply (*simp add*:*aHom-def*)
apply (*rule conjI*)
apply (*rule Pi-I*)
apply (*simp add*:*prodrg-carrier*)
apply (*cut-tac prodag-carrier*[*of I A*], *simp*)

apply (*simp add*:*PRoject-def*)
apply (*cut-tac prodag-carrier*[*of I A*], *simp*,
thin-tac carrier (*a*Π$_I$ *A*) = *carr-prodag I A*)
apply (*simp add*:*prodag-comp-i*)
apply (*simp add*:*Ring.ring-is-ag*)
apply (*simp add*:*Ring.ring-is-ag*)

apply (*subgoal-tac* ∀ *k*∈*I*. *aGroup* (*A k*))
apply (*frule project-aHom* [*of I A j*], *assumption*+)
apply (*rule conjI*)
apply (*simp add*:*prodrg-carrier aHom-def*)
apply (*simp add*:*prodrg-carrier*)
apply (*simp add*:*prodrg-pOp*)
apply (*simp add*:*prodag-pOp*[*THEN sym*])

apply (*simp add*:*aHom-def*)

apply (*rule ballI*, *simp add*:*Ring.ring-is-ag*)

apply (*rule conjI*)
apply (*rule ballI*)+
apply (*simp add*:*prodrg-carrier*)
apply (*cut-tac prodag-carrier*[*of I A*], *simp*)
apply (*frule-tac* $X = x$ **and** $Y = y$ **in** *prod-tOp-mem*[*of I A*], *assumption*+)
apply (*simp add*:*prodrg-tOp*)
apply (*simp add*:*PRoject-def*)
apply (*simp add*:*prod-tOp-def*)

apply (*rule ballI*)
apply (*simp add*:*Ring.ring-is-ag*)

apply (*frule prodrg-ring* [*of I A*])
apply (*frule Ring.ring-one*[*of* $r\Pi_I$ *A*])
apply (*simp add*:*prodrg-carrier*)
apply (*cut-tac prodag-carrier*[*of I A*], *simp*)
apply (*simp add*:*PRoject-def*) **apply** (*simp add*:*prodrg-def*)
apply (*fold prodrg-def*) **apply** (*simp add*:*prod-one-def*)

apply (*rule ballI*)
apply (*simp add*:*Ring.ring-is-ag*)
done

lemma *augm-funcTr*:$[\![\forall k \leq (Suc\ n).\ Ring\ (B\ k);$
$f \in carr\text{-}prodag\ \{i.\ i \leq (Suc\ n)\}\ B]\!] \Longrightarrow$
$f = augm\text{-}func\ n\ (restrict\ f\ \{i.\ i \leq n\})\ (Un\text{-}carrier\ \{i.\ i \leq n\}\ B)\ (Suc\ 0)$
$(\lambda x{\in}\{0{::}nat\}.\ f\ (x + Suc\ n))$
$(Un\text{-}carrier\ \{0\}\ (compose\ \{0\}\ B\ (slide\ (Suc\ n))))$
apply (*rule carr-prodag-mem-eq* [*of* $\{i.\ i \leq (Suc\ n)\}$ *B f*
augm-func n (*restrict f* $\{i.\ i \leq n\}$) (*Un-carrier* $\{i.\ i \leq n\}$ *B*) (*Suc 0*)
$(\lambda x{\in}\{0\}.\ f\ (x + Suc\ n))$ (*Un-carrier* $\{0\}$ (*compose* $\{0\}$ *B* (*slide* (*Suc n*))))])
apply (*rule ballI*, *simp add*:*Ring.ring-is-ag*)
apply (*simp add*:*augm-func-def*)
apply (*simp add*:*carr-prodag-def*)
apply (*rule conjI*)
apply (*simp add*:*augm-func-def*)
apply (*rule conjI*)
apply (*rule Pi-I*)
apply (*simp add*:*augm-func-def sliden-def*)
apply (*erule conjE*)+
apply (*frule-tac* $x = x$ **in** *funcset-mem*[*of f* $\{i.\ i \leq Suc\ n\}$
Un-carrier $\{i.\ i \leq Suc\ n\}$ *B*]) **apply** *simp*
apply *simp*
apply (*rule impI*)
apply (*rule-tac* $x = Suc\ n$ **in** *funcset-mem*[*of f* $\{i.\ i \leq Suc\ n\}$

Un-carrier $\{i.\ i \leq Suc\ n\}\ B]$, *assumption*) **apply** *simp*
apply (*rule allI*, (*erule conjE*)+)
apply (*simp add*:*augm-func-def*)
apply (*case-tac* $i \leq n$, *simp add*:*sliden-def*)
apply (*simp add*:*sliden-def*, *rule impI*)
apply (*simp add*:*nat-not-le-less*,
frule-tac $m = n$ **and** $n = i$ **in** *Suc-leI*,
frule-tac $m = i$ **and** $n = Suc\ n$ **in** *Nat.le-antisym*, *assumption*+,
simp)

apply (*rule ballI*, *simp*)
apply (*simp add*:*augm-func-def sliden-def*)
apply (*rule impI*, *simp add*:*nat-not-le-less*)
apply (*frule-tac* $n = l$ **in** *Suc-leI*[*of n* -])
apply (*frule-tac* $m = l$ **and** $n = Suc\ n$ **in** *Nat.le-antisym*, *assumption*+,
simp)
done

lemma *A-to-prodag-mem*:⟦*Ring A*; ∀ *k*∈*I*. *Ring* (*B k*); ∀ *k*∈*I*. (*S k*) ∈
rHom A (*B k*); *x* ∈ *carrier A* ⟧ ⟹ *A-to-prodag A I S B x* ∈ *carr-prodag I B*
apply (*simp add*:*carr-prodag-def*)
apply (*rule conjI*)
apply (*simp add*:*A-to-prodag-def extensional-def*)
apply (*rule conjI*)
apply (*rule Pi-I*)
apply (*simp add*: *A-to-prodag-def*)
apply (*subgoal-tac* (*S xa*) ∈ *rHom A* (*B xa*)) **prefer** *2* **apply** *simp*
apply (*thin-tac* ∀ *k*∈*I*. *S k* ∈ *rHom A* (*B k*))
apply (*frule-tac f* = *S xa* **and** *A* = *A* **and** *R* = *B xa* **in** *rHom-mem*, *assumption*+)
apply (*simp add*:*Un-carrier-def*) **apply** *blast*
apply (*rule ballI*)

apply (*simp add*:*A-to-prodag-def*)
apply (*rule-tac f* = *S i* **and** *A* = *A* **and** *R* = *B i* **and** *a* = *x* **in** *rHom-mem*)
apply *simp*
apply *assumption*
done

lemma *A-to-prodag-rHom*:⟦*Ring A*; ∀ *k*∈*I*. *Ring* (*B k*); ∀ *k*∈*I*. (*S k*) ∈
rHom A (*B k*) ⟧ ⟹ *A-to-prodag A I S B* ∈ *rHom A* ($r\Pi_I$ *B*)
apply (*simp add*:*rHom-def* [*of A* $r\Pi_I$ *B*])
apply (*rule conjI*)
apply (*cut-tac A-to-prodag-aHom*[*of A I B S*])
apply (*subst aHom-def*, *simp*)
apply (*simp add*:*prodrg-carrier*)
apply (*simp add*:*aHom-def*)
apply (*simp add*:*prodrg-def*)
apply (*cut-tac prodag-pOp*[*of I B*], *simp*)

apply (*rule ballI*, *simp add*:*Ring.ring-is-ag*,
simp add:*Ring.ring-is-ag*,
rule ballI, *simp add*:*Ring.ring-is-ag*)

apply (*rule ballI*)
apply (*simp add*:*rHom-def*)

apply (*rule conjI*)
apply (*rule ballI*)+
apply (*frule-tac* $x = x$ **and** $y = y$ **in** *Ring.ring-tOp-closed*[*of A*], *assumption*+)
apply (*frule-tac* $x = x \cdot_{r\,A} y$ **in** *A-to-prodag-mem*[*of A I B S*], *assumption*+,
frule-tac $x = x$ **in** *A-to-prodag-mem*[*of A I B S*], *assumption*+,
frule-tac $x = y$ **in** *A-to-prodag-mem*[*of A I B S*], *assumption*+)
apply (*simp add*:*prodrg-tOp*[*of I B*])
apply (*frule-tac* X = *A-to-prodag A I S B x* **and** Y = *A-to-prodag A I S B y*
in *prod-tOp-mem*[*of I B*], *assumption*+)
apply (*cut-tac* X = *A-to-prodag A I S B* ($x \cdot_{r\,A} y$) **and** Y = *prod-tOp I B*
(*A-to-prodag A I S B x*) (*A-to-prodag A I S B y*) **in** *carr-prodag-mem-eq*[*of I B*])
apply (*rule ballI*, *simp add*:*Ring.ring-is-ag*) **apply** *assumption*+
apply (*rule ballI*, *simp add*:*prod-tOp-def A-to-prodag-def*)
apply (*frule-tac* $x = l$ **in** *bspec*, *assumption*,
thin-tac $\forall k{\in}I.$ *Ring* (*B k*),
frule-tac $x = l$ **in** *bspec*, *assumption*,
thin-tac $\forall k{\in}I.$ $S\ k \in$ *rHom A* (*B k*))
apply (*simp add*:*rHom-tOp*) **apply** *simp*

apply (*simp add*:*prodrg-one*[*of I B*])
apply (*frule prod-one-func*[*of I B*])
apply (*frule Ring.ring-one*[*of A*],
frule-tac $x = 1_{r\,A}$ **in** *A-to-prodag-mem*[*of A I B S*], *assumption*+)
apply (*cut-tac* X = *A-to-prodag A I S B* $1_{r\,A}$ **and** Y = *prod-one I B* **in**
carr-prodag-mem-eq[*of I B*])
apply (*rule ballI*, *simp add*:*Ring.ring-is-ag*)
apply *assumption*+
apply (*rule ballI*)
apply (*subst A-to-prodag-def*, *simp add*:*prod-one-def*)
apply (*frule-tac* $x = l$ **in** *bspec*, *assumption*,
thin-tac $\forall k{\in}I.$ *Ring* (*B k*),
frule-tac $x = l$ **in** *bspec*, *assumption*,
thin-tac $\forall k{\in}I.$ $S\ k \in$ *rHom A* (*B k*))
apply (*simp add*:*rHom-one*)
apply *assumption*
done

lemma *ac-fProd-ProdTr1*:$\forall k \leq$ (*Suc n*). *Ring* (*B k*) $\Longrightarrow$
ag-setfunc n B (*Suc 0*) (*compose* {*0*::*nat*} *B* (*slide* (*Suc n*)))
(*carr-prodag* {*i*. $i \leq n$} *B*) (*carr-prodag* {*0*}
(*compose* {*0*} *B* (*slide* (*Suc n*)))) $\subseteq$ *carr-prodag* {*i*. $i \leq$ (*Suc n*)} *B*
apply (*rule subsetI*)

apply (*simp add*:*ag-setfunc-def*)
apply (*erule exE*, *erule conjE*, *erule exE*, *erule conjE*)
apply (*simp*,
  thin-tac x =
   augm-func n g (*Un-carrier* {*j*. *j* ≤ *n*} *B*) (*Suc 0*) *h*
   (*Un-carrier* {*0*} (*compose* {*0*} *B* (*slide* (*Suc n*)))))
apply (*simp add*:*carr-prodag-def* [*of* {*j*. *j* ≤ (*Suc n*)} *B*])
apply (*rule conjI*)
 apply (*simp add*:*augm-func-def*)
apply (*rule conjI*)
 apply (*simp add*:*Pi-def*) **apply** (*rule allI*) **apply** (*rule impI*)
 apply (*simp add*:*augm-func-def*)
 apply (*case-tac x* ≤ *n*)
 apply *simp* **apply** (*simp add*:*carr-prodag-def*)
 apply (*erule conjE*)+ **apply** (*frule-tac x* = *x* **in** *mem-of-Nset* [*of* - *n*])
 apply (*frule-tac f* = *g* **and** *x* = *x* **in** *funcset-mem*[*of* - {*j*. *j* ≤ *n*}
                Un-carrier {*j*. *j* ≤ *n*} *B*], *assumption*+)
 apply (*simp add*:*Un-carrier-def*,
   erule exE, *erule conjE*, *erule exE*, *simp*, *erule conjE*,
   frule-tac x = *i* **and** *y* = *n* **and** *z* = *Suc n* **in** *le-less-trans*,
   simp,
   frule-tac x = *i* **and** *y* = *Suc n* **in** *less-imp-le*, *blast*)
 apply (*simp add*:*sliden-def*)
 apply (*simp add*:*carr-prodag-def Un-carrier-def*, (*erule conjE*)+)
 apply (*simp add*:*compose-def slide-def*)
 apply (*cut-tac n-in-Nsetn*[*of Suc n*], *blast*)
 apply (*rule allI*, *rule impI*)
 apply (*simp add*:*augm-func-def*)
 apply (*case-tac i* ≤ *n*, *simp*)
 apply (*simp add*:*carr-prodag-def* [*of* {*i*. *i* ≤ *n*} -])
 apply *simp* **apply** (*thin-tac g* ∈ *carr-prodag* {*i*. *i* ≤ *n*} *B*)
 apply (*simp add*: *not-less* [*symmetric*, *of* - *n*],
   frule-tac n = *i* **in** *Suc-leI*[*of n*],
   frule-tac m = *i* **and** *n* = *Suc n* **in** *le-antisym*, *assumption*+, *simp*)
 apply (*simp add*:*carr-prodag-def compose-def slide-def sliden-def*)
done

lemma *ac-fProd-Prod*:∀ *k* ≤ *n*. *Ring* (*B k*) ⟹
          ac-fProd-Rg n B = *carr-prodag* {*j*. *j* ≤ *n*} *B*
apply (*case-tac n* = *0*)
 apply *simp*
 apply (*subgoal-tac* ∃ *m*. *n* = *Suc m*)
 apply (*subgoal-tac* ∀ *m*. *n* = *Suc m* ⟶
          ac-fProd-Rg n B = *carr-prodag* {*j*. *j* ≤ *n*} *B*)
 apply *blast* **apply** (*thin-tac* ∃ *m*. *n* = *Suc m*)
 apply (*rule allI*, *rule impI*) **apply** (*simp*, *thin-tac n* = *Suc m*)
 apply (*rule equalityI*)
 apply (*simp add*:*ac-fProd-ProdTr1*)
 apply (*rule subsetI*)

apply (*rename-tac m f*)
apply (*frule augm-funcTr*, *assumption*+)
apply (*simp add*:*ag-setfunc-def*)
apply (*subgoal-tac* (*restrict f* {*j*. *j* ≤ *m*}) ∈ *carr-prodag* {*j*. *j* ≤ *m*} *B*)
apply (*subgoal-tac* (λ*x*∈{*0*::*nat*}. *f* (*Suc* (*x* + *m*))) ∈ *carr-prodag* {*0*}
(*compose* {*0*} *B* (*slide* (*Suc m*))))

apply *blast*
apply (*thin-tac f* =
augm-func m (*restrict f* {*i*. *i* ≤ *m*}) (*Un-carrier* {*i*. *i* ≤ *m*} *B*)
(*Suc 0*) (λ*x*∈{*0*}. *f* (*Suc* (*x* + *m*)))
(*Un-carrier* {*0*} (*compose* {*0*} *B* (*slide* (*Suc m*)))))
apply (*simp add*:*carr-prodag-def*)
apply (*rule conjI*)
apply (*simp add*:*Pi-def restrict-def*)
apply (*simp add*:*Un-carrier-def compose-def slide-def*)
apply (*simp add*:*compose-def slide-def*)

apply (*thin-tac f* =
augm-func m (*restrict f* {*i*. *i* ≤ *m*}) (*Un-carrier* {*i*. *i* ≤ *m*} *B*)
(*Suc 0*) (λ*x*∈{*0*}. *f* (*Suc* (*x* + *m*)))
(*Un-carrier* {*0*} (*compose* {*0*} *B* (*slide* (*Suc m*)))))
apply (*simp add*:*carr-prodag-def*)
apply (*simp add*:*Un-carrier-def*)
apply (*simp add*:*Pi-def*)
apply (*rule allI*) **apply** (*rule impI*)
apply (*erule conjE*)+
apply (*rotate-tac 1*)
apply (*frule-tac a* = *x* **in** *forall-spec*, *simp*)
apply (*erule exE*,
thin-tac ∀ *x*≤*Suc m*. ∃ *xa*. (∃ *i*≤*Suc m*. *xa* = *carrier* (*B i*)) ∧ *f x* ∈ *xa*)
apply (*frule-tac a* = *x* **in** *forall-spec*, *simp*)
apply *blast*
apply (*cut-tac t* = *n* **in** *Suc-pred*[*THEN sym*], *simp*)
apply *blast*
done

A direct product of a finite number of rings defined with *ac-fProd-Rg* is equal to that defined by using *carr-prodag*.

definition
fprodrg :: [*nat*, *nat* ⇒ ('*a*, '*more*) *Ring-scheme*] ⇒
(|*carrier*:: (*nat* ⇒ '*a*) *set*, *pop*::[(*nat* ⇒ '*a*), (*nat* ⇒ '*a*)]
⇒ (*nat* ⇒ '*a*), *mop*:: (*nat* ⇒ '*a*) ⇒ (*nat* ⇒ '*a*), *zero*::(*nat* ⇒ '*a*),
tp :: [(*nat* ⇒ '*a*), (*nat* ⇒ '*a*)] ⇒ (*nat* ⇒ '*a*), *un* :: (*nat* ⇒ '*a*) |) **where**

fprodrg n B = (| *carrier* = *ac-fProd-Rg n B*,
pop = λ*f*. λ*g*. *prod-pOp* {*i*. *i* ≤ *n*} *B f g*, *mop* = λ*f*. *prod-mOp* {*i*. *i* ≤ *n*} *B*
f,
zero = *prod-zero* {*i*. *i* ≤ *n*} *B*, *tp* = λ*f*. λ*g*. *prod-tOp* {*i*. *i* ≤ *n*} *B f g*,

$un = prod\text{-}one\ \{i.\ i \leq n\}\ B\ ⟧$

definition
fPRoject ::[*nat*, *nat* ⇒ ('*a*, '*more*) *Ring-scheme*, *nat*]
⇒ (*nat* ⇒ '*a*) ⇒ '*a* **where**
fPRoject n B x = (λ*f*∈*ac-fProd-Rg n B*. *f x*)

lemma *fprodrg-ring*:∀ *k* ≤ *n*. *Ring* (*B k*) ⟹ *Ring* (*fprodrg n B*)
apply (*simp add*:*fprodrg-def*)
apply (*frule ac-fProd-Prod*)
apply *simp*
apply (*fold prodrg-def*)
apply (*simp add*:*prodrg-ring*)
done

4.8 Chinese remainder theorem

lemma *Chinese-remTr1*:⟦*Ring A*; ∀ *k* ≤ (*n*::*nat*). *ideal A* (*J k*);
∀ *k* ≤ *n*. *B k* = *qring A* (*J k*); ∀ *k* ≤ *n*. *S k* = *pj A* (*J k*) ⟧ ⟹
$ker_{A,(r\Pi_{\{j.\ j \leq n\}}\ B)}$ (*A-to-prodag A* {*j*. *j* ≤ *n*} *S B*) =
⋂ {*I*. ∃*k*∈{*j*. *j* ≤ *n*}. *I* = (*J k*)}
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*simp add*:*ker-def*)
apply (*rule allI*, *rule impI*)
apply (*rename-tac a K*, *erule conjE*)
apply (*simp add*:*prodrg-def*, *simp add*:*A-to-prodag-def prod-one-def*)
apply (*erule exE*, *erule conjE*)

apply (*subgoal-tac* (λ*k*∈{*j*. *j* ≤ *n*}. *S k a*) *k* = (λ*x*∈{*j*. *j* ≤ *n*}. $\mathbf{0}_{B\ x}$) *k*)
apply (*thin-tac* (λ*k*∈{*j*. *j* ≤ *n*}. *S k a*) = *prod-zero* {*j*. *j* ≤ *n*} *B*)
apply *simp* **apply** (*frule-tac I* = *J k* **in** *Ring.qring-zero* [*of A*])
apply *simp*
apply (*frule-tac I* = *J k* **and** *x* = *a* **in** *pj-mem* [*of A*]) **apply** *simp*
apply *assumption* **apply** *simp*
apply (*frule-tac I* = *J k* **and** *a* = *a* **in** *Ring.a-in-ar-coset* [*of A*])
apply *simp* **apply** *assumption* **apply** *simp*
apply (*simp add*:*prod-zero-def*)
apply (*rule subsetI*)
apply (*simp add*:*CollectI ker-def*)

apply (*cut-tac Nset-inc-0*[*of n*])
apply (*frule-tac a* = *J 0* **in** *forall-spec*, *blast*)
apply (*frule-tac x* = *0* **in** *spec*, *simp*)
apply (*frule-tac h* = *x* **in** *Ring.ideal-subset* [*of A J 0*], *simp*+)
apply (*thin-tac x* ∈ *J 0*)
apply (*simp add*:*A-to-prodag-def prodrg-def*)
apply (*simp add*:*prod-zero-def*)

apply (*rule funcset-eq* [*of* - {$j.\ j \leq n$}])
apply (*simp add*:*extensional-def restrict-def*)+
apply (*rule allI*, *rule impI*)
apply (*simp add*:*Ring.qring-zero*)
apply (*frule-tac* $a = xa$ **in** *forall-spec*, *assumption*,
thin-tac $\forall k \leq n.$ *ideal* A $(J\ k)$)
apply (*subst pj-mem* [*of* A], *assumption*+)
apply (*frule-tac* $I = J\ xa$ **and** $a = x$ **in** *Ring.a-in-ar-coset* [*of* A],
assumption+)
apply (*rule-tac* $a = x$ **and** $I = J\ xa$ **in** *Ring.Qring-fix1* [*of* A], *assumption*+)
apply *blast*
done

lemma (**in** *Ring*) *coprime-prod-int2Tr*:
$((\forall k \leq (Suc\ n).\ ideal\ R\ (J\ k)) \wedge$
$(\forall i \leq (Suc\ n).\ \forall j \leq (Suc\ n).\ (i \neq j \longrightarrow coprime\text{-}ideals\ R\ (J\ i)\ (J\ j))))$
$\longrightarrow (\bigcap \{I.\ \exists k \leq (Suc\ n).\ I = (J\ k)\} = ideal\text{-}n\text{-}prod\ R\ (Suc\ n)\ J)$
apply (*induct-tac* n)
apply (*rule impI*)
apply (*erule conjE*)
apply (*simp add*:*Nset-1*)
apply (*subst coprime-int-prod* [*THEN sym*, *of J 0 J* (*Suc 0*)], *simp*+)
apply (*rule equalityI*, *rule subsetI*)
apply (*simp*, *blast*)
apply (*rule subsetI*, *simp*, *rule allI*, *rule impI*, *erule exE*, (*erule conjE*)+)
apply *simp*
apply (*simp add*:*Nset-1-1*, *erule disjE*, (*simp del*:*ideal-n-prodSn*)+)

apply (*rule impI*)
apply (*subgoal-tac* $\bigcap \{I.\ \exists k \leq (Suc\ (Suc\ n)).\ I = J\ k\} =$
$(\bigcap \{I.\ \exists k \leq (Suc\ n).\ I = J\ k\}) \cap (J\ (Suc\ (Suc\ n)))$)
apply (*subgoal-tac* $\bigcap \{I.\ \exists k \leq (Suc\ n).\ I = J\ k\} = (i\Pi_{R,(Suc\ n)}\ J)$)

apply (*frule-tac* $n = Suc\ n$ **and** $J = J$ **in** *coprime-n-idealsTr4*)
apply (*simp del*:*ideal-n-prodSn*)
apply (*subst coprime-int-prod*)
apply (*rule n-prod-ideal*)
apply (*rule allI*, *simp*, *simp*, *assumption*)
apply *simp* **apply** (*cut-tac* $n = Suc\ n$ **in** *Nsetn-sub-mem1*)
apply *simp*
apply (*thin-tac* $(\forall k \leq Suc\ n.\ ideal\ R\ (J\ k)) \wedge$
$(\forall i \leq Suc\ n.\ \forall j \leq Suc\ n.\ i \neq j \longrightarrow coprime\text{-}ideals\ R\ (J\ i)\ (J\ j)) \longrightarrow$
$\bigcap \{I.\ \exists k \leq Suc\ n.\ I = J\ k\} = i\Pi_{R,Suc\ n}\ J$,
thin-tac $(\forall k \leq Suc\ (Suc\ n).\ ideal\ R\ (J\ k)) \wedge$
$(\forall i \leq Suc\ (Suc\ n).$
$\forall j \leq Suc\ (Suc\ n).\ i \neq j \longrightarrow coprime\text{-}ideals\ R\ (J\ i)\ (J\ j))$)
apply (*rule equalityI*, *rule subsetI*, *simp*)
apply (*rule conjI*,

rule allI, *rule impI*, *erule exE*, *erule conjE*, *simp*,
frule-tac a = xa **in** *forall-spec*,
frule-tac x = k **and** *y = Suc n* **and** *z = Suc (Suc n)* **in**
le-less-trans, *simp*,
frule-tac x = k **and** *y = Suc (Suc n)* **in** *less-imp-le*, *blast*)
apply *simp*
apply (*frule-tac a = J (Suc (Suc n))* **in** *forall-spec*,
cut-tac n = Suc (Suc n) **in** *le-refl*, *blast*, *simp*)

apply (*rule subsetI*, *simp*, *rule allI*, *rule impI*)
apply (*erule exE*, *erule conjE*)
apply (*erule conjE*,
case-tac k = Suc (Suc n), *simp*)
apply (*frule-tac m = k* **and** *n = Suc (Suc n)* **in** *noteq-le-less*, *assumption*,
thin-tac k ≤ Suc (Suc n))
apply (*frule-tac x = k* **and** *n = Suc n* **in** *Suc-less-le*)
apply (*frule-tac a = xa* **in** *forall-spec*,
blast,
thin-tac ∀ *xa*. (∃ *k*≤*Suc n*. *xa = J k*) ⟶ *x* ∈ *xa*,
simp)
done

lemma (**in** *Ring*) *coprime-prod-int2*:⟦ ∀ *k* ≤ (*Suc n*). *ideal R* (*J k*);
∀ *i* ≤ (*Suc n*). ∀ *j* ≤ (*Suc n*). (*i* ≠*j* ⟶ *coprime-ideals R* (*J i*) (*J j*))⟧
⟹ (⋂ {*I*. ∃ *k* ≤ (*Suc n*). *I* = (*J k*)} = *ideal-n-prod R* (*Suc n*) *J*)
apply (*simp add*:*coprime-prod-int2Tr*)
done

lemma (**in** *Ring*) *coprime-2-n*:⟦*ideal R A*; *ideal R B*⟧ ⟹
(*qring R A*) ⨁$_r$ (*qring R B*) = *r*Π$_{\{j.\ j \le (Suc\ 0)\}}$ (*prodB1* (*qring R A*) (*qring R B*))
apply (*simp add*:*Prod2Rg-def Nset-1*)
done

In this and following lemmata, ideals A and B are of type ('*a*, '*more*) *RingType-scheme*. Don't try (*r*Π$_($*Nset n*) *B*) ⨁$_r$ *B* (*Suc n*)

lemma (**in** *Ring*) *A-to-prodag2-hom*:⟦*ideal R A*; *ideal R B*; *S 0 = pj R A*;
S (*Suc 0*) = *pj R B*⟧ ⟹
A-to-prodag R {*j*. *j* ≤ (*Suc 0*)} *S* (*prodB1* (*qring R A*) (*qring R B*)) ∈
rHom R (*qring R A* ⨁$_r$ *qring R B*)
apply (*subst coprime-2-n* [*of A B*], *assumption*+)
apply (*rule A-to-prodag-rHom*, *rule Ring-axioms*)
apply (*rule ballI*)
apply (*case-tac k = 0*)
apply (*simp add*:*prodB1-def*)
apply (*simp add*:*qring-ring*)
apply (*simp*)
apply (*frule-tac n = k* **in** *Suc-leI*[*of 0*], *thin-tac 0 < k*)
apply (*frule-tac m = k* **and** *n = Suc 0* **in** *le-antisym*, *assumption*)

apply (*simp*, *simp add*:*prodB1-def*, *simp add*:*qring-ring*)

apply (*rule ballI*)
apply (*simp add*:*Nset-1*)
apply (*erule disjE*)
apply (*simp add*:*prodB1-def*, *rule pj-Hom*, *rule Ring-axioms*, *assumption*)
apply (*simp*, *simp add*:*prodB1-def*)
apply (*rule pj-Hom*, *rule Ring-axioms*, *assumption*+)
done

lemma (**in** *Ring*) *A2coprime-rsurjecTr*:⟦*ideal R A*; *ideal R B*; *S 0* = *pj R A*;
S (*Suc 0*) = *pj R B*⟧ ⟹
(*carrier* (*qring R A* ⨁$_r$ *qring R B*)) =
carr-prodag {*j*. *j* ≤ (*Suc 0*)} (*prodB1* (*qring R A*) (*qring R B*))
apply (*simp add*:*Prod2Rg-def prodrg-def Nset-1*)
done

lemma (**in** *Ring*) *A2coprime-rsurjec*:⟦*ideal R A*; *ideal R B*; *S 0* = *pj R A*;
S (*Suc 0*) = *pj R B*; *coprime-ideals R A B*⟧ ⟹
surjec$_{R,((qring\ R\ A) \bigoplus_r (qring\ R\ B))}$
(*A-to-prodag R* {*j*. *j*≤(*Suc 0*)} *S* (*prodB1* (*qring R A*) (*qring R B*)))
apply (*frule A-to-prodag2-hom* [*of A B S*], *assumption*+)
apply (*simp add*:*surjec-def*)
apply (*rule conjI*,
simp add:*rHom-def*)
apply (*rule surj-to-test*[*of A-to-prodag R* {*j*. *j* ≤ (*Suc 0*)} *S*
(*prodB1* (*qring R A*) (*qring R B*)) *carrier R*
carrier (*qring R A* ⨁$_r$ *qring R B*)])
apply (*simp add*:*rHom-def aHom-def*)

apply (*rule ballI*)
apply (*frule rHom-func*[*of A-to-prodag R* {*j*. *j* ≤ (*Suc 0*)} *S*
(*prodB1* (*R* /$_r$ *A*) (*R* /$_r$ *B*)) *R*],
thin-tac A-to-prodag R {*j*. *j* ≤ (*Suc 0*)} *S* (*prodB1* (*R* /$_r$ *A*) (*R* /$_r$ *B*))
∈ *rHom R* (*R* /$_r$ *A* ⨁$_r$ *R* /$_r$ *B*))
apply (*simp add*:*A2coprime-rsurjecTr*[*of A B S*])
apply (*simp add*:*Nset-1*)
apply (*frule-tac X* = *b 0* **and** *Y* = *b* (*Suc 0*) **in**
coprime-surjTr[*of A B*], *assumption*+)
apply (*simp add*:*carr-prodag-def prodB1-def*,
simp add:*carr-prodag-def prodB1-def*)

apply (*erule bexE*)
apply (*frule-tac x* = *r* **in** *funcset-mem*[*of A-to-prodag R* {*0*, *Suc 0*} *S*
(*prodB1* (*R* /$_r$ *A*) (*R* /$_r$ *B*))
carrier R carr-prodag {*0*, *Suc 0*} (*prodB1* (*R* /$_r$ *A*) (*R* /$_r$ *B*))],
assumption+)
apply (*cut-tac X* = *A-to-prodag R* {*0*, *Suc 0*} *S* (*prodB1* (*R* /$_r$ *A*) (*R* /$_r$ *B*)) *r*
and *Y* = *b* **in**

carr-prodag-mem-eq[*of* {0, *Suc* 0} *prodB1* (*R* /$_r$ *A*) (*R* /$_r$ *B*)])
apply (*rule ballI*)
apply (*simp*, *erule disjE*)
apply (*simp add*:*prodB1-def*, *fold prodB1-def*,
simp add:*qring-ring Ring.ring-is-ag*)
apply (*simp add*:*prodB1-def*, *fold prodB1-def*,
simp add:*qring-ring Ring.ring-is-ag*)
apply *assumption*+
apply (*rule ballI*, *simp*, *erule disjE*, *simp*)
apply (*subst A-to-prodag-def*, *simp*)
apply (*subst A-to-prodag-def*, *simp*)
apply *blast*
done

lemma (**in** *Ring*) *prod2-n-Tr1*:⟦∀ *k* ≤ (*Suc* 0). *ideal R* (*J k*);
∀ *k* ≤ (*Suc* 0). *B k* = *qring R* (*J k*);
∀ *k* ≤ (*Suc* 0). *S k* = *pj R* (*J k*) ⟧ ⟹
A-to-prodag R {*j*. *j* ≤ (*Suc* 0)} *S*
(*prodB1* (*qring R* (*J* 0)) (*qring R* (*J* (*Suc* 0)))) =
A-to-prodag R {*j*. *j* ≤ (*Suc* 0)} *S B*
apply (*subgoal-tac* ∀ *k* ≤ (*Suc* 0). (*prodB1* (*qring R* (*J* 0)) (*qring R* (*J* (*Suc* 0))))
k = *B k*)
apply (*simp add*:*A-to-prodag-def*)
apply (*rule allI*, *rule impI*)
apply (*case-tac k* = 0, *simp add*:*Nset-1-1*)
apply (*simp add*:*prodB1-def*)
apply (*simp add*:*Nset-1-1*)
apply (*simp add*:*prodB1-def*)
done

lemma (**in** *aGroup*) *restrict-prod-Suc*:⟦∀ *k* ≤ (*Suc* (*Suc n*)). *ideal R* (*J k*);
∀ *k* ≤ (*Suc* (*Suc n*)). *B k* = *R* /$_r$ *J k*;
∀ *k* ≤ (*Suc* (*Suc n*)). *S k* = *pj R* (*J k*);
f ∈ *carrier* (*r*Π$_{\{j.\ j \le (Suc\ (Suc\ n))\}}$ *B*)⟧ ⟹
restrict f {*j*. *j* ≤ (*Suc n*)} ∈ *carrier* (*r*Π$_{\{j.\ j \le (Suc\ n)\}}$ *B*)
apply (*cut-tac Nsetn-sub-mem1*[*of Suc n*])
apply (*simp add*:*prodrg-def*)
apply (*simp add*:*carr-prodag-def*, (*erule conjE*)+)
apply (*simp add*:*Un-carrier-def*)
apply (*rule Pi-I*)
apply *simp*
apply (*frule-tac x* = *x* **in** *funcset-mem*[*of f* {*j*. *j* ≤ (*Suc* (*Suc n*))}
⋃{*X*. ∃ *i* ≤ (*Suc* (*Suc n*)). *X* = *carrier* (*R* /$_r$ *J i*)}],
simp)
apply *simp*
apply (*erule exE*, *erule conjE*, *erule exE*, *erule conjE*, *simp*)

apply (*rotate-tac* −5)
apply (*frule-tac a* = *x* **in** *forall-spec*) **apply** *simp*

apply *blast*
done

lemma (**in** *Ring*) *Chinese-remTr2*:($\forall\, k \leq (Suc\ n).\ ideal\ R\ (J\ k)$) $\wedge$
$(\forall\, k{\leq}(Suc\ n).\ B\ k = qring\ R\ (J\ k)) \wedge$
$(\forall\, k{\leq}(Suc\ n).\ S\ k = pj\ R\ (J\ k)) \wedge$
$(\forall\, i{\leq}(Suc\ n).\ \forall j{\leq}\ (Suc\ n).\ (i \neq j \longrightarrow$
coprime-ideals $R\ (J\ i)\ (J\ j))) \longrightarrow$
$surjec_{R,(r\Pi_{\{j.\ j\leq\ (Suc\ n)\}}\ B)}$
$(A\text{-}to\text{-}prodag\ R\ \{j.\ j{\leq}(Suc\ n)\}\ S\ B)$
apply (*cut-tac Ring*)
apply (*induct-tac n*)

apply (*rule impI*) **apply** (*erule conjE*)+
apply (*frule A-to-prodag-rHom* [*of R* $\{j.\ j \leq Suc\ 0\}$ *B S*])
apply (*rule ballI*, *simp add*:*Ring.qring-ring*)
apply (*rule ballI*, *simp add*:*pj-Hom*)
apply (*frule rHom-func*[*of A-to-prodag R* $\{j.\ j \leq (Suc\ 0)\}$ *S B R*
$r\Pi_{\{j.\ j\ \leq\ (Suc\ 0)\}}\ B$])
apply (*simp add*:*surjec-def*)
apply (*rule conjI*)
apply (*simp add*:*rHom-def*)
apply (*rule surj-to-test*, *assumption*+)
apply (*rule ballI*) **apply** (*simp add*:*Nset-1*)
apply (*cut-tac coprime-elems* [*of J 0 J* (*Suc 0*)])
apply (*rename-tac f*)
apply (*erule bexE*, *erule bexE*)
apply (*simp add*:*prodrg-def*) **apply** (*fold prodrg-def*)
apply (*cut-tac* $X = f\ 0$ **and** $Y = f\ (Suc\ 0)$ **in**
coprime-surjTr[*of J 0 J* (*Suc 0*)], *simp*+)
apply (*simp add*:*carr-prodag-def*, *simp add*:*carr-prodag-def*)
apply (*erule bexE*, (*erule conjE*)+)
apply (*frule-tac* $x = r$ **in** *funcset-mem*[*of A-to-prodag R* $\{0,\ Suc\ 0\}$ *S B*
carrier R carr-prodag $\{0,\ Suc\ 0\}$ *B*], *assumption*+)
apply (*cut-tac* $X =$ *A-to-prodag R* $\{0,\ Suc\ 0\}$ *S B r* **and** $Y = f$ **in**
carr-prodag-mem-eq[*of* $\{0,\ Suc\ 0\}$ *B*])
apply (*rule ballI*, *simp*, *erule disjE*, *simp add*:*qring-ring*
Ring.ring-is-ag,
simp add:*Ring.qring-ring Ring.ring-is-ag*)
apply *assumption*+
apply (*rule ballI*, *simp*, *erule disjE*, *simp*)
apply (*simp add*:*A-to-prodag-def*, *simp add*:*A-to-prodag-def*)
apply *blast* **apply** *simp*+

apply (*rule impI*, (*erule conjE*)+)

apply (*cut-tac* $n = Suc\ n$ **in** *Nsetn-sub-mem1*)
apply (*subgoal-tac* $\forall\, k{\in}\{i.\ i \leq Suc\ (Suc\ n)\}.\ Ring\ (B\ k)$)

apply (*frule-tac* $I = \{i.\ i \leq Suc\ (Suc\ n)\}$ **in** *A-to-prodag-rHom* [*of R - B S*])
apply *assumption*
apply (*rule ballI*)
apply (*simp add*:*pj-Hom*)
apply *simp*
apply (*subst surjec-def*, *rule conjI*,
simp add:*rHom-def*)
apply (*cut-tac* $n = Suc\ n$ **in** *coprime-n-idealsTr4*[*of - J*])
apply *blast*
apply (*frule-tac* $f = A\text{-}to\text{-}prodag\ R\ \{j.\ j \leq (Suc\ (Suc\ n))\}\ S\ B$ **and**
$A = R$ **in** *rHom-func*)
apply (*rule-tac* $f = A\text{-}to\text{-}prodag\ R\ \{j.\ j \leq (Suc\ (Suc\ n))\}\ S\ B$ **and**
$A = carrier\ R$ **and** $B = carrier\ (r\Pi_{\{j.\ j \leq (Suc\ (Suc\ n))\}}\ B)$ **in**
surj-to-test, *assumption*+)
apply (*rule ballI*)
apply (*cut-tac* $n = Suc\ n$ **in** *n-prod-ideal*[*of - J*])
apply (*rule allI*, *simp*)
apply (*frule-tac* $A = i\Pi_{R,(Suc\ n)}\ J$ **and** $B = J\ (Suc\ (Suc\ n))$ **in**
coprime-elems,
cut-tac $n = Suc\ (Suc\ n)$ **in** *n-in-Nsetn*,
blast, *assumption*)
apply (*erule bexE*, *erule bexE*) **apply** (*rename-tac n f a b*)
apply (*thin-tac coprime-ideals R* $(i\Pi_{R,(Suc\ n)}\ J)$ $(J\ (Suc\ (Suc\ n))))$
apply (*cut-tac* $n = Suc\ n$ **and** $a = a$ **and** $J = J$ **in** *ele-n-prod*,
rule allI, *simp*, *assumption*)

apply (*cut-tac ring-is-ag*)
apply (*frule-tac* $n = n$ **and** $f = f$ **in** *aGroup.restrict-prod-Suc*[*of R - R J B S*],
assumption+)
apply (*frule-tac* $S = r\Pi_{\{j.\ j \leq (Suc\ n)\}}\ B$ **and**
$f = A\text{-}to\text{-}prodag\ R\ \{j.\ j \leq (Suc\ n)\}\ S\ B$ **in** *surjec-surj-to*[*of R*])
apply (*frule-tac* $f = A\text{-}to\text{-}prodag\ R\ \{j.\ j \leq (Suc\ n)\}\ S\ B$ **and** $A = carrier\ R$
and $B = carrier\ (r\Pi_{\{j.\ j \leq\ (Suc\ n)\}}\ B)$ **and**
$b = restrict\ f\ \{j.\ j \leq (Suc\ n)\}$ **in** *surj-to-el2*, *assumption*)
apply (*erule bexE*, *rename-tac n f a b u*)
apply (*cut-tac* $n = Suc\ (Suc\ n)$ **in** *n-in-Nsetn*,
frule-tac $f = f$ **and** $I = \{j.\ j \leq (Suc\ (Suc\ n))\}$ **and** $A = B$ **and**
$i = Suc\ (Suc\ n)$ **in** *prodrg-component*, *assumption*)
apply *simp*
apply (*frule-tac* $J = J\ (Suc\ (Suc\ n))$ **and** $X = f\ (Suc\ (Suc\ n))$ **in**
pj-surj-to[*of R*], *simp*, *assumption*)
apply (*erule bexE*, *rename-tac n f a b u v*)
apply (*frule-tac* $a = Suc\ (Suc\ n)$ **in** *forall-spec*, *simp*,
frule-tac $I = J\ (Suc\ (Suc\ n))$ **and** $h = b$ **in** *Ring.ideal-subset*[*of R*],
assumption+,
cut-tac $I = i\Pi_{R,n}\ J \diamondsuit_{r\,R}\ J\ (Suc\ n)$ **and** $h = a$ **in**
Ring.ideal-subset[*of R*], *assumption*+)
apply (*frule-tac* $x = b$ **and** $y = u$ **in** *Ring.ring-tOp-closed*[*of R*], *assumption*+,

frule-tac $x = a$ **and** $y = v$ **in** *Ring.ring-tOp-closed*[*of R*], *assumption*+,
frule Ring.ring-is-ag[*of R*],
frule-tac $x = b \cdot_{r\,R} u$ **and** $y = a \cdot_{r\,R} v$ **in** *aGroup.ag-pOp-closed*[*of R*],
assumption+)
apply (*frule-tac* $f = A$-*to-prodag R* $\{j.\ j \leq (Suc\ (Suc\ n))\}$ *S B* **and**
$A = carrier\ R$ **and** $B = carrier\ (r\Pi_{\{j.\ j \leq (Suc\ (Suc\ n))\}}\ B)$ **and**
$x = b \cdot_{r\,R} u \pm_R a \cdot_{r\,R} v$ **in** *funcset-mem*, *assumption*+)
apply (*frule-tac* $f = A$-*to-prodag R* $\{j.\ j \leq (Suc\ (Suc\ n))\}$ *S B*
$(b \cdot_{r\,R} u \pm_R a \cdot_{r\,R} v)$ **and** $I = \{j.\ j \leq (Suc\ (Suc\ n))\}$
and $g = f$ **in** *carr-prodrg-mem-eq*, *simp*)
apply (*rule ballI*)
apply (*subst A-to-prodag-def*, *simp*)
apply (*frule-tac* $I = J\ i$ **in** *pj-Hom*[*of R*], *simp*)
apply (*simp add*: *rHom-add*)
apply (*frule-tac* $a = i$ **in** *forall-spec*, *assumption*,
thin-tac $\forall k \leq (Suc\ (Suc\ n)).\ ideal\ R\ (J\ k)$,
frule-tac $I = J\ i$ **in** *Ring.qring-ring*[*of R*], *assumption*,
thin-tac $\forall k \leq (Suc\ (Suc\ n)).\ Ring\ (R\ /_r\ J\ k)$,
frule-tac $R = R\ /_r\ (J\ i)$ **and** $x = b$ **and** $y = u$ **and** $f = pj\ R\ (J\ i)$ **in**
rHom-tOp[*of R*], *assumption*+, *simp*,
thin-tac $pj\ R\ (J\ i)\ (b \cdot_{r\,R} u) = pj\ R\ (J\ i)\ b \cdot_{r\,R\ /_r\ J\ i} pj\ R\ (J\ i)\ u$,
frule-tac $R = R\ /_r\ (J\ i)$ **and** $x = a$ **and** $y = v$ **and** $f = pj\ R\ (J\ i)$ **in**
rHom-tOp[*of R*], *simp add*:*Ring.qring-ring*, *assumption*+)
apply (*frule-tac* $f = pj\ R\ (J\ i)$ **and** $R = R\ /_r\ J\ i$ **and** $a = v$ **in**
rHom-mem[*of* - *R*], *assumption*+,
frule-tac $f = pj\ R\ (J\ i)$ **and** $R = R\ /_r\ J\ i$ **and** $a = u$ **in**
rHom-mem[*of* - *R*], *assumption*+,
frule-tac $f = pj\ R\ (J\ i)$ **and** $R = R\ /_r\ J\ i$ **and** $a = b$ **in**
rHom-mem[*of* - *R*], *assumption*+,
frule-tac $f = pj\ R\ (J\ i)$ **and** $R = R\ /_r\ J\ i$ **and** $a = a$ **in**
rHom-mem[*of* - *R*], *assumption*+)
apply (*frule-tac* $R = R\ /_r\ J\ i$ **in** *Ring.ring-is-ag*)
apply (*case-tac* $i \leq (Suc\ n)$)
apply (*frule-tac* $I1 = J\ i$ **and** $x1 = a$ **in** *pj-zero*[*THEN sym*, *of R*],
assumption+, *simp*,
thin-tac $pj\ R\ (J\ i)\ (a \cdot_{r\,R} v) = \mathbf{0}_{R\ /_r\ J\ i} \cdot_{r\,R\ /_r\ J\ i} pj\ R\ (J\ i)\ v$)
apply (*simp add*:*Ring.ring-times-0-x*)
apply (*frule-tac* $f = pj\ R\ (J\ i)$ **and** $A = R$ **and** $R = R\ /_r\ J\ i$ **and**
$x = a$ **and** $y = b$ **in** *rHom-add*, *assumption*+, *simp*,
thin-tac A-*to-prodag R* $\{j.\ j \leq Suc\ (Suc\ n)\}$ *S B*
$(b \cdot_r u) \pm_{r\Pi_{\{i.\ i \leq Suc\ (Suc\ n)\}}} B$
A-to-prodag R $\{j.\ j \leq Suc\ (Suc\ n)\}$ *S B* $(a \cdot_r v)$
$\in carrier\ (r\Pi_{\{j.\ j \leq Suc\ (Suc\ n)\}}\ B)$)

apply (*simp add*:*aGroup.ag-l-zero*)
apply (*rotate-tac* -1, *frule sym*, *thin-tac* $pj\ R\ (J\ i)\ 1_{r\,R} = pj\ R\ (J\ i)\ b$,
simp add:*rHom-one*) **apply** (*simp add*:*Ring.ring-l-one*)
apply (*simp add*:*aGroup.ag-r-zero*)

apply (*frule-tac* $f = A\text{-}to\text{-}prodag\ R\ \{j.\ j \leq (Suc\ n)\}\ S\ B\ u$ **and**
$g = restrict\ f\ \{j.\ j \leq (Suc\ n)\}$ **and** $x = i$ **in** *eq-fun-eq-val*,
thin-tac $A\text{-}to\text{-}prodag\ R\ \{j.\ j \leq (Suc\ n)\}\ S\ B\ u = restrict\ f\ \{j.\ j \leq (Suc\ n)\}$)
apply (*simp add*:*A-to-prodag-def*)
apply *simp*
apply (*frule-tac* $y = i$ **and** $x = Suc\ n$ **in** *not-le-imp-less*,
frule-tac $m = Suc\ n$ **and** $n = i$ **in** *Suc-leI*,
frule-tac $m = i$ **and** $n = Suc\ (Suc\ n)$ **in** *Nat.le-antisym*, *assumption*+,
simp)
apply (*frule-tac* $I1 = J\ (Suc\ (Suc\ n))$ **and** $x1 = b$ **in** *pj-zero*[*THEN sym*, *of*
R], *assumption*+, *simp add*:*Ring.ring-times-0-x*)
apply (*frule-tac* $f = pj\ R\ (J\ (Suc\ (Suc\ n)))$ **and** $A = R$ **and**
$R = R\ /_r\ J\ (Suc\ (Suc\ n))$ **and**
$x = a$ **and** $y = b$ **in** *rHom-add*, *assumption*+, *simp*)
apply (*simp add*:*aGroup.ag-r-zero*)
apply (*rotate-tac* -1, *frule sym*,
thin-tac $pj\ R\ (J\ (Suc\ (Suc\ n)))\ 1_{r\,R} = pj\ R\ (J\ (Suc\ (Suc\ n)))\ a$,
simp add:*rHom-one*,
simp add:*Ring.ring-l-one*)
apply (*simp add*:*aGroup.ag-l-zero*)

apply *blast*
apply (*rule ballI*, *simp add*:*Ring.qring-ring*)
done

lemma (**in** *Ring*) *Chinese-remTr3*:$[\![\forall k \leq (Suc\ n).\ ideal\ R\ (J\ k);$
$\forall k \leq (Suc\ n).\ B\ k = qring\ R\ (J\ k);\ \forall k \leq (Suc\ n).\ S\ k = pj\ R\ (J\ k);$
$\forall i \leq (Suc\ n).\ \forall j \leq (Suc\ n).\ (i \neq j \longrightarrow coprime\text{-}ideals\ R\ (J\ i)\ (J\ j))]\!] \Longrightarrow$
$surjec_{R,(r\Pi_{\{j.\ j \leq (Suc\ n)\}}\ B)}$
$(A\text{-}to\text{-}prodag\ R\ \{j.\ j \leq (Suc\ n)\}\ S\ B)$
apply (*simp add*:*Chinese-remTr2* [*of* $n\ J\ B\ S$])
done

lemma (**in** *Ring*) *imset*:$[\![\forall k \leq (Suc\ n).\ ideal\ R\ (J\ k)]\!]$
$\Longrightarrow \{I.\ \exists k \leq (Suc\ n).\ I = J\ k\} = \{J\ k|\ k.\ k \in \{j.\ j \leq (Suc\ n)\}\}$
apply *blast*
done

theorem (**in** *Ring*) *Chinese-remThm*:$[\![(\forall k \leq (Suc\ n).\ ideal\ R\ (J\ k));$
$\forall k \leq (Suc\ n).\ B\ k = qring\ R\ (J\ k);\ \forall k \leq (Suc\ n).\ S\ k = pj\ R\ (J\ k);$
$\forall i \leq (Suc\ n).\ \forall j \leq (Suc\ n).\ (i \neq j \longrightarrow coprime\text{-}ideals\ R\ (J\ i)\ (J\ j))]\!]$
$\Longrightarrow bijec_{(qring\ R\ (\bigcap\ \{J\ k\ |\ k.\ k \in \{j.\ j \leq (Suc\ n)\}\}))},{}_{(r\Pi_{\{j.\ j \leq (Suc\ n)\}}\ B)}$
$((A\text{-}to\text{-}prodag\ R\ \{j.\ j \leq (Suc\ n)\}\ S\ B)^{\circ}{}_{R,(prodrg\ \{j.\ j \leq (Suc\ n)\}\ B)})$
apply (*frule Chinese-remTr3*, *assumption*+)
apply (*cut-tac* $I = \{j.\ j \leq (Suc\ n)\}$ **and** $A = B$ **in** *prodrg-ring*)
apply (*rule ballI*, *simp add*:*qring-ring*)
apply (*cut-tac surjec-ind-bijec* [*of* $R\ r\Pi_{\{j.\ j \leq (Suc\ n)\}}\ B$
$A\text{-}to\text{-}prodag\ R\ \{j.\ j \leq (Suc\ n)\}\ S\ B$])

apply (*cut-tac Ring*,
  simp add:*Chinese-remTr1* [*of R Suc n J B S*])
apply (*simp add*:*imset*, *rule Ring-axioms*, *assumption*+)
apply (*rule A-to-prodag-rHom*, *rule Ring-axioms*)
apply (*rule ballI*)
apply (*simp add*:*qring-ring*)
apply (*rule ballI*, *simp*, *rule pj-Hom*, *rule Ring-axioms*, *simp*)
apply *assumption*
done

lemma (**in** *Ring*) *prod-prime*:⟦*ideal R A*; ∀ *k*≤(*Suc n*). *prime-ideal R* (*P k*);
  ∀ *l*≤(*Suc n*). ¬ (*A* ⊆ *P l*);
  ∀ *k*≤ (*Suc n*). ∀ *l*≤ (*Suc n*). *k* = *l* ∨ ¬ (*P k*) ⊆ (*P l*)⟧ ⟹
  ∀ *i* ≤ (*Suc n*). (*nprod R* (*ppa R P A i*) *n* ∈ *A* ∧
    (∀ *l* ∈ {*j*. *j*≤(*Suc n*)} − {*i*}. *nprod R* (*ppa R P A i*) *n* ∈ *P l*) ∧
    (*nprod R* (*ppa R P A i*) *n* ∉ *P i*))
apply (*rule allI*, *rule impI*)
apply (*rule conjI*)
apply (*frule-tac i* = *i* **in** *prod-primeTr1*[*of n P A*], *assumption*+)
apply (*frule-tac n* = *n* **and** *f* = *ppa R P A i* **in** *ideal-nprod-inc*[*of A*])
 apply (*rule allI*, *rule impI*)
 apply (*rotate-tac* −*2*,
   frule-tac a = *ia* **in** *forall-spec*, *assumption*,
   thin-tac ∀ *l* ≤ *n*.
     ppa R P A i l ∈ *A* ∧
     ppa R P A i l ∈ *P* (*skip i l*) ∧ *ppa R P A i l* ∉ *P i*,
   simp add:*ideal-subset*)
 apply (*rotate-tac* −*1*,
   frule-tac a = *n* **in** *forall-spec*, *simp*,
   thin-tac ∀ *l*≤ *n*.
     ppa R P A i l ∈ *A* ∧
     ppa R P A i l ∈ *P* (*skip i l*) ∧ *ppa R P A i l* ∉ *P i*,
   (*erule conjE*)+,
   blast, *assumption*)
apply (*frule-tac i* = *i* **in** *prod-primeTr1*[*of n P A*], *assumption*+)
apply (*rule conjI*)
apply (*rule ballI*)
apply (*frule-tac a* = *l* **in** *forall-spec*, *simp*,
  frule-tac I = *P l* **in** *prime-ideal-ideal*)
apply (*frule-tac n* = *n* **and** *f* = *ppa R P A i* **and** *A* = *P l* **in** *ideal-nprod-inc*)
apply (*rule allI*) **apply** (*simp add*:*ppa-mem*, *simp*)
apply (*case-tac l* < *i*,
  thin-tac ∀ *l*≤ (*Suc n*). ¬ *A* ⊆ *P l*,
  thin-tac ∀ *k*≤ (*Suc n*). ∀ *l* ≤ (*Suc n*). *k* = *l* ∨ ¬ *P k* ⊆ *P l*)
 apply (*erule conjE*,
   frule-tac x = *l* **and** *y* = *i* **and** *z* = *Suc n* **in** *less-le-trans*,
   assumption,
   frule-tac x = *l* **and** *n* = *n* **in** *Suc-less-le*)
 apply (*rotate-tac 2*,

frule-tac $a = l$ **in** *forall-spec*, *assumption*,
thin-tac $\forall\, l{\leq}n.$ *ppa* $R\ P\ A\ i\ l \in A \wedge$
$\quad$ *ppa* $R\ P\ A\ i\ l \in P\ (skip\ i\ l) \wedge$ *ppa* $R\ P\ A\ i\ l \notin P\ i$,
thin-tac $l < Suc\ n$)
apply (*simp only*:*skip-im-Tr1-2*, *blast*)
apply (*frule-tac* $x = l$ **and** $y = i$ **in** *leI*,
thin-tac $\neg\ l < i$,
cut-tac $x = l$ **and** $A = \{j.\ j \leq (Suc\ n)\}$ **and** $a = i$ **in** *in-diff1*)
apply *simp*
apply (*erule conjE*,
frule not-sym, *thin-tac* $l \neq i$,
frule-tac $x = i$ **and** $y = l$ **in** *le-imp-less-or-eq*,
erule disjE, *thin-tac* $i \leq l$,
frule-tac $x = i$ **and** $n = l$ **in** *less-le-diff*)
apply (*cut-tac* $i = i$ **and** $n = n$ **and** $x = l - Suc\ 0$ **in** *skip-im-Tr2-1*,
simp, *assumption*+, *simp*,
frule-tac $x = l$ **and** $n = n$ **in** *le-Suc-diff-le*)
apply (*rotate-tac* -7,
frule-tac $a = l - Suc\ 0$ **in** *forall-spec*, *assumption*,
thin-tac $\forall\, l{\leq}n.$ *ppa* $R\ P\ A\ i\ l \in A \wedge$
$\quad$ *ppa* $R\ P\ A\ i\ l \in P\ (skip\ i\ l) \wedge$ *ppa* $R\ P\ A\ i\ l \notin P\ i$,
simp, (*erule conjE*)+)
apply *blast*
apply *simp*
apply *assumption*
apply (*frule-tac* $a = i$ **in** *forall-spec*, *assumption*,
thin-tac $\forall\, k{\leq}\ (Suc\ n).$ *prime-ideal* $R\ (P\ k)$)
apply (*rule-tac* $P = P\ i$ **and** $n = n$ **and** $f = ppa\ R\ P\ A\ i$ **in**
prime-nprod-exc, *assumption*+)
apply (*rule allI*, *rule impI*)
apply (*rotate-tac* -3,
frule-tac $a = ia$ **in** *forall-spec*, *assumption*,
thin-tac $\forall\, l \leq n.$
ppa $R\ P\ A\ i\ l \in A \wedge$
ppa $R\ P\ A\ i\ l \in P\ (skip\ i\ l) \wedge$ *ppa* $R\ P\ A\ i\ l \notin P\ i$,
simp add:*ideal-subset*)
apply (*rule allI*, *rule impI*) **apply** (
rotate-tac 4,
frule-tac $a = l$ **in** *forall-spec*, *assumption*,
thin-tac $\forall\, l{\leq}\ n.$
ppa $R\ P\ A\ i\ l \in A \wedge$
ppa $R\ P\ A\ i\ l \in P\ (skip\ i\ l) \wedge$ *ppa* $R\ P\ A\ i\ l \notin P\ i$,
simp)
done

lemma *skip-im1*:$[\![i \leq (Suc\ n);\ P \in \{j.\ j \leq (Suc\ n)\} \to Collect\ (prime\text{-}ideal\ R)]\!]$
$\Longrightarrow$
compose $\{j.\ j \leq n\}\ P\ (skip\ i)$ ‘ $\{j.\ j \leq n\} = P$ ‘ $(\{j.\ j \leq (Suc\ n)\} - \{i\})$
apply (*cut-tac skip-fun*[*of i n*])

apply (*subst setim-cmpfn*[*of* - - - - $\{X.$ *prime-ideal R X*$\}$], *assumption*+)
apply *simp*
apply (*simp add*:*skip-fun-im*)
done

lemma (**in** *Ring*) *mutch-aux1*:$[\![$*ideal R A*; $i \leq$ (*Suc n*);
$P \in \{j.\ j \leq$ (*Suc n*)$\} \to$ *Collect* (*prime-ideal R*)$]\!] \Longrightarrow$
compose $\{j.\ j \leq n\}$ *P* (*skip i*) $\in \{j.\ j \leq n\} \to$ *Collect* (*prime-ideal R*)
apply (*cut-tac skip-fun*[*of i n*])
apply (*simp add*:*composition*[*of skip i* $\{j.\ j \leq n\}$ $\{j.\ j \leq$ (*Suc n*)$\}$ *P*
Collect (*prime-ideal R*)])
done

lemma (**in** *Ring*) *prime-ideal-cont1Tr*:*ideal R A* $\Longrightarrow$
$\forall P.\ ((P \in \{j.\ j \leq (n::nat)\} \to \{X.$ *prime-ideal R X*$\}) \wedge$
$(A \subseteq \bigcup (P\ `\ \{j.\ j \leq n\})))) \longrightarrow (\exists i \leq n.\ A \subseteq (P\ i))$
apply (*induct-tac n*)
apply (*rule allI*, *rule impI*)
apply (*erule conjE*)
apply *simp*

apply (*rule allI*, *rule impI*)
apply (*erule conjE*)+
apply (*case-tac* $\exists i \leq$ (*Suc n*). $\exists j \leq$ (*Suc n*). $(i \neq j \wedge P\ i \subseteq P\ j)$)
apply ((*erule exE*, *erule conjE*)+, *erule conjE*)
apply (*frule-tac f* = *P* **and** *n* = *n* **and** *X* = $\{X.$ *prime-ideal R X*$\}$ **and**
A = *A* **and** *i* = *i* **and** *j* = *j* **in** *Un-less-Un*, *assumption*+, *simp*+)
apply (*frule mutch-aux1*, *assumption*+)
apply (*frule-tac a* = *compose* $\{j.\ j \leq n\}$ *P* (*skip i*) **in** *forall-spec*,
simp, *erule exE*)
apply (*cut-tac i* = *i* **and** *n* = *n* **and** *x* = *ia* **in** *skip-fun-im1*,
simp+, *erule conjE*, *simp add*:*compose-def*,*blast*)

apply (*thin-tac* $\forall P.\ P \in \{j.\ j \leq n\} \to \{X.$ *prime-ideal R X*$\} \wedge$
$A \subseteq \bigcup (P\ `\ \{j.\ j \leq n\}) \longrightarrow$
$(\exists i \leq n.\ A \subseteq P\ i)$,
rule contrapos-pp, *simp*+)
apply (*cut-tac n* = *n* **and** *P* = *P* **in** *prod-prime* [*of A*], *assumption*)
apply (*rule allI*, *rule impI*,
frule-tac f = *P* **and** *A* = $\{j.\ j \leq$ (*Suc n*)$\}$ **and** *B* = $\{X.$ *prime-ideal R X*$\}$
and *x* = *k* **in** *funcset-mem*, *simp*, *simp*, *assumption*+)
apply (*frule-tac n* = *Suc n* **and**
f = $\lambda i \in \{j.\ j \leq$ (*Suc n*)$\}$. (*nprod R* (*ppa R P A i*) *n*) **in**
nsum-ideal-inc[*of A*], *rule allI*, *rule impI*, *simp*)
apply (*subgoal-tac* (*nsum R* $(\lambda i \in \{j.\ j \leq$ (*Suc n*)$\}$. *nprod R* (*ppa R P A i*) *n*)
(*Suc n*)) $\notin (\bigcup x \in \{j.\ j \leq$ (*Suc n*)$\}$. *P x*))
apply *blast*
apply (*simp del*:*nsum-suc*)
apply (*rule allI*, *rule impI*) **apply** (*rename-tac n P l*)

apply (*frule-tac* $f = P$ **and** $A = \{j.\ j \leq (Suc\ n)\}$ **and**
$B = \{X.\ prime\text{-}ideal\ R\ X\}$
and $x = l$ **in** *funcset-mem*, *simp*, *simp del*:*nsum-suc*,
frule-tac $I = P\ l$ **in** *prime-ideal-ideal*)
apply (*rule-tac* $A = P\ l$ **and** $n = Suc\ n$ **and**
$f = \lambda i{\in}\{j.\ j \leq (Suc\ n)\}.\ (nprod\ R\ (ppa\ R\ P\ A\ i)\ n)$ **in**
nsum-ideal-exc, *simp*+, *rule allI*, *simp add*:*ideal-subset*)
apply (*rule contrapos-pp*, *simp*+)
apply (*rotate-tac* -1,
frule-tac $a = l$ **in** *forall-spec*, *simp*,
thin-tac $\forall j{\leq}Suc\ n.$
$(\exists\, la{\in}\{i.\ i \leq Suc\ n\} - \{j\}.\ e\Pi_{R,n}\ ppa\ R\ P\ A\ la \notin P\ l) \vee$
$e\Pi_{R,n}\ ppa\ R\ P\ A\ j \in P\ l$,
thin-tac $\forall i{\leq}Suc\ n.\ \forall j{\leq}Suc\ n.\ i = j \vee \neg\ P\ i \subseteq P\ j$,
thin-tac $\forall i{\leq}Suc\ n.\ \neg\ A \subseteq P\ i$)
apply (*erule disjE*, *erule bexE*)
apply (*frule-tac* $a = la$ **in** *forall-spec*, *simp*,
thin-tac $\forall i{\leq}Suc\ n.$
$e\Pi_{R,n}\ ppa\ R\ P\ A\ i \in A \wedge$
$(\forall\, l{\in}\{j.\ j \leq Suc\ n\} - \{i\}.\ e\Pi_{R,n}\ ppa\ R\ P\ A\ i \in P\ l) \wedge$
$e\Pi_{R,n}\ ppa\ R\ P\ A\ i \notin P\ i$,
(*erule conjE*)+)
apply *blast*
apply *blast*
done

lemma (**in** *Ring*) *prime-ideal-cont1*:$[\![ideal\ R\ A;\ \forall\, i \leq (n{::}nat).$
$prime\text{-}ideal\ R\ (P\ i);\ A \subseteq \bigcup\ \{X.\ (\exists\, i \leq n.\ X = (P\ i))\}\ ]\!] \Longrightarrow$
$\exists\, i{\leq}\ n.\ A{\subseteq}(P\ i)$
apply (*frule prime-ideal-cont1Tr*[*of A n*])
apply (*frule-tac* $a = P$ **in** *forall-spec*,
thin-tac $\forall P.\ P \in \{j.\ j \leq n\} \to \{X.\ prime\text{-}ideal\ R\ X\} \wedge$
$A \subseteq \bigcup(P\ `\ \{j.\ j \leq n\}) \longrightarrow (\exists\, i{\leq}n.\ A \subseteq P\ i)$)
apply (*rule conjI*, *simp*,
rule subsetI, *simp*,
frule-tac $c = x$ **in** *subsetD*[*of A* $\bigcup\{X.\ \exists\, i{\leq}n.\ X = P\ i\}$], *assumption*+,
simp, *blast*)
apply *assumption*
done

lemma (**in** *Ring*) *prod-n-ideal-contTr0*:$(\forall\, l{\leq}\ n.\ ideal\ R\ (J\ l)) \longrightarrow$
$i\Pi_{R,n}\ J\ \subseteq\ \bigcap\{X.\ (\exists\, k{\leq}n.\ X = (J\ k))\}$
apply (*induct-tac n*)
apply *simp*

apply (*rule impI*)
apply (*cut-tac* $n = n$ **in** *Nsetn-sub-mem1*,
simp)
apply (*cut-tac* $n = n$ **in** *n-prod-ideal*[*of - J*], *simp*)

apply (*cut-tac* $I = i\Pi_{R,n}\ J$ **and** $J = J\ (Suc\ n)$ **in**
ideal-prod-sub-Int) **apply** *assumption* **apply** *simp*
apply (*frule-tac* $A = i\Pi_{R,n}\ J \diamondsuit_r R\ J\ (Suc\ n)$ **and**
$B = i\Pi_{R,n}\ J \cap J\ (Suc\ n)$ **and**
$C = \bigcap\{X.\ \exists k{\leq}\ n.\ X = J\ k\} \cap J\ (Suc\ n)$ **in** *subset-trans*)
apply (*rule-tac* $A = i\Pi_{R,n}\ J$ **and** $B = \bigcap\{X.\ \exists k{\leq}n.\ X = J\ k\}$ **and**
$C = J\ (Suc\ n)$ **in** *inter-mono*, *assumption*)
apply (*rule-tac* $A = i\Pi_{R,n}\ J \diamondsuit_r J\ (Suc\ n)$ **and**
$B = \bigcap\{X.\ \exists k{\leq}\ n.\ X = J\ k\} \cap J\ (Suc\ n)$ **and**
$C = \bigcap\{X.\ \exists k{\leq}\ (Suc\ n).\ X = J\ k\}$ **in** *subset-trans*,
assumption)
apply (*rule subsetI*)
apply *simp*
apply (*rule allI*, *rule impI*)
apply (*erule exE*, (*erule conjE*)+)
apply (*case-tac* $k = Suc\ n$, *simp*)
apply (*frule-tac* $m = k$ **and** $n = Suc\ n$ **in** *noteq-le-less*, *assumption*)
apply (*thin-tac* $k \leq Suc\ n$)
apply (*frule-tac* $x = k$ **and** $n = Suc\ n$ **in** *less-le-diff*,
thin-tac $k < Suc\ n$, *simp*, *thin-tac* $\forall l{\leq}Suc\ n.\ ideal\ R\ (J\ l)$)
apply (*frule-tac* $a = xa$ **in** *forall-spec*, *blast*,
thin-tac $\forall xa.\ (\exists k{\leq}n.\ xa = J\ k) \longrightarrow x \in xa$,
simp)
done

lemma (**in** *Ring*) *prod-n-ideal-contTr*:$[\![\forall l{\leq}\ n.\ ideal\ R\ (J\ l)]\!] \Longrightarrow$
$i\Pi_{R,n}\ J\ \subseteq\ \bigcap\{X.\ (\exists k \leq n.\ X = (J\ k))\}$
apply (*simp add*:*prod-n-ideal-contTr0*)
done

lemma (**in** *Ring*) *prod-n-ideal-cont2*:$[\![\forall l{\leq}\ (n{::}nat).\ ideal\ R\ (J\ l)$;
prime-ideal $R\ P$; $\bigcap\{X.\ (\exists k{\leq}\ n.\ X = (J\ k))\} \subseteq P]\!] \Longrightarrow$
$\exists l{\leq}\ n.\ (J\ l) \subseteq P$
apply (*frule prod-n-ideal-contTr*[*of n J*])
apply (*frule-tac* $A = i\Pi_{R,n}\ J$ **and** $B = \bigcap\{X.\ \exists k{\leq}\ n.\ X = J\ k\}$ **and** $C = P$
in *subset-trans*, *assumption*+)
apply (*thin-tac* $\bigcap\{X.\ \exists k{\leq}\ n.\ X = J\ k\} \subseteq P$,
thin-tac $i\Pi_{R,n}\ J \subseteq \bigcap\{X.\ \exists k{\leq}\ n.\ X = J\ k\}$)
apply (*simp add*:*ideal-n-prod-prime*)
done

lemma (**in** *Ring*) *prod-n-ideal-cont3*:$[\![\forall l{\leq}\ (n{::}nat).\ ideal\ R\ (J\ l)$;
prime-ideal $R\ P$; $\bigcap\{X.\ (\exists k{\leq}\ n.\ X = (J\ k))\} = P]\!] \Longrightarrow$
$\exists l{\leq}\ n.\ (J\ l) = P$
apply (*frule prod-n-ideal-cont2*[*of n J P*], *assumption*+)
apply *simp*
apply (*erule exE*)
apply (*subgoal-tac* $J\ l = P$)
apply *blast*

apply (*rule equalityI*, *simp*)
apply (*rule subsetI*)
apply (*rotate-tac* -4, *frule sym*, *thin-tac* $\bigcap\{X.\ \exists k{\leq}\ n.\ X = J\ k\} = P$)
apply *simp*
apply *blast*
done

definition
ideal-quotient :: [- , *'a set*, *'a set*] ⇒ *'a set* **where**
ideal-quotient R A B = $\{x|\ x.\ x \in$ *carrier R* $\wedge\ (\forall\, b{\in}B.\ x\ \cdot_{r\,R}\ b \in A)\}$

abbreviation
IDEALQT ((*3-*/ †-/ -) [*82*,*82*,*83*]*82*) **where**
$A\ \dagger_R\ B$ == *ideal-quotient R A B*

lemma (**in** *Ring*) *ideal-quotient-is-ideal*:
⟦*ideal R A*; *ideal R B*⟧ ⟹ *ideal R* (*ideal-quotient R A B*)
apply (*rule ideal-condition*)
apply (*rule subsetI*)
apply (*simp add*:*ideal-quotient-def CollectI*)
apply (*simp add*:*ideal-quotient-def*)
apply (*cut-tac ring-zero*)
apply (*subgoal-tac* $\forall\, b{\in}B.\ \mathbf{0}\ \cdot_r\ b \in A$)
apply *blast*
apply (*rule ballI*)
apply (*frule-tac h* = *b* **in** *ideal-subset*[*of B*], *assumption*)
apply (*frule-tac x* = *b* **in** *ring-times-0-x*)
apply (*simp add*:*ideal-zero*)
apply (*rule ballI*)+
apply (*simp add*:*ideal-quotient-def*, (*erule conjE*)+,
rule conjI)
apply (*rule ideal-pOp-closed*)
apply (*simp add*:*whole-ideal*, *assumption*+)
apply (*cut-tac ring-is-ag*)
apply (*simp add*:*aGroup.ag-mOp-closed*)
apply (*rule ballI*)
apply (*subst ring-distrib2*)
apply (*simp add*:*ideal-subset*, *assumption*)
apply (*cut-tac ring-is-ag*, *simp add*: *aGroup.ag-mOp-closed*)
apply (*frule-tac a1* = *y* **and** *b1* = *b* **in** *ring-inv1-1* [*THEN sym*])
apply (*simp add*:*ideal-subset*, *simp*)
apply (*rule ideal-pOp-closed*, *assumption*+, *simp*)
apply (*rule ideal-inv1-closed*, *assumption*+, *simp*)
apply (*rule ballI*)+
apply (*simp add*:*ideal-quotient-def*)
apply (*rule conjI*)
apply (*erule conjE*)
apply (*simp add*:*ring-tOp-closed*)

apply (*erule conjE*)
apply (*rule ballI*)
apply (*subst ring-tOp-assoc*, *assumption*+, *simp add*:*ideal-subset*)
apply (*simp add*:*ideal-ring-multiple* [*of A*])
done

4.9 Addition of finite elements of a ring and *ideal-multiplication*

We consider sum in an abelian group

lemma (**in** *aGroup*) *nsum-mem1Tr*: $A +> J \Longrightarrow (\forall j \leq n.\ f\ j \in J) \longrightarrow nsum\ A\ f\ n \in J$
apply (*induct-tac n*)
apply (*rule impI*)
apply *simp*
apply (*rule impI*)
apply *simp*
apply (*rule asubg-pOp-closed*, *assumption*+)
apply *simp*
done

lemma (**in** *aGroup*) *fSum-mem*:$[\![\forall j \in nset\ (Suc\ n)\ m.\ f\ j \in carrier\ A;\ n < m]\!] \Longrightarrow fSum\ A\ f\ (Suc\ n)\ m \in carrier\ A$
apply (*simp add*:*fSum-def*)
apply (*rule nsum-mem*)
apply (*rule allI*, *simp add*:*cmp-def slide-def*)
apply (*rule impI*)
apply (*frule-tac x* = *Suc* (*n* + *j*) **in** *bspec*)
apply (*simp add*:*nset-def*, *arith*)
done

lemma (**in** *aGroup*) *nsum-mem1*:$[\![A +> J; \forall j \leq n.\ f\ j \in J]\!] \Longrightarrow nsum\ A\ f\ n \in J$
apply (*simp add*:*nsum-mem1Tr*)
done

lemma (**in** *aGroup*) *nsum-eq-i*:$[\![\forall j{\leq}n.\ f\ j \in carrier\ A; \forall j{\leq}n.\ g\ j \in carrier\ A; i \leq n; \forall l \leq i.\ f\ l = g\ l]\!] \Longrightarrow nsum\ A\ f\ i = nsum\ A\ g\ i$
apply (*rule nsum-eq*)
apply (*rule allI*, *rule impI*, *simp*)+
done

lemma (**in** *aGroup*) *nsum-cmp-eq*:$[\![f \in \{j.\ j{\leq}(n::nat)\} \to carrier\ A; h1 \in \{j.\ j \leq n\} \to \{j.\ j \leq n\};\ h2 \in \{j.\ j \leq n\} \to \{j.\ j \leq n\}; i \leq n]\!] \Longrightarrow nsum\ A\ (cmp\ f\ (cmp\ h2\ h1))\ i = nsum\ A\ (cmp\ (cmp\ f\ h2)\ h1)\ i$
apply (*rule nsum-eq-i* [*of n cmp f* (*cmp h2 h1*) *cmp* (*cmp f h2*) *h1 i*])
apply (*rule allI*, *rule impI*, *simp add*:*cmp-def*)
apply ((*rule funcset-mem*, *assumption*)+, *simp*)
apply (*rule allI*, *rule impI*, *simp add*:*cmp-def*,

```
      (rule funcset-mem, assumption)+, simp+)
apply (rule allI, rule impI, simp add:cmp-def)
done

lemma (in aGroup) nsum-cmp-eq-transpos:⟦ ∀j≤(Suc n). f j ∈ carrier A;
      i ≤ n;i ≠ n ⟧ ⟹
 nsum A (cmp f (cmp (transpos i n) (cmp (transpos n (Suc n)) (transpos i n))))
 (Suc n) = nsum A (cmp f (transpos i (Suc n))) (Suc n)
apply (rule nsum-eq [of Suc n cmp f (cmp (transpos i n)
                          (cmp (transpos n (Suc n)) (transpos i n)))
      cmp f (transpos i (Suc n))])
apply (rule allI, rule impI)
apply (simp add:cmp-def)
apply (cut-tac i = i and n = Suc n and j = n and l = j in transpos-mem,
      simp+)
apply (cut-tac i = n and n = Suc n and j = Suc n and l = transpos i n j
       in transpos-mem, simp+)
apply (cut-tac i = i and n = Suc n and j = n and
       l = transpos n (Suc n) (transpos i n j) in transpos-mem,
      simp+)
apply (rule allI, rule impI, simp add:cmp-def)
apply (cut-tac i = i and n = Suc n and j = Suc n and l = j in transpos-mem,
      simp+)
apply (rule allI, rule impI)
 apply (simp add:cmp-def)
 apply (thin-tac ∀j≤Suc n. f j ∈ carrier A,
       rule eq-elems-eq-val[of - - f])
 apply (simp add:transpos-def)
done

lemma transpos-Tr-n1:Suc (Suc 0) ≤ n ⟹
                        transpos (n − Suc 0) n n = n − Suc 0
apply (simp add:transpos-def)
done

lemma transpos-Tr-n2:Suc (Suc 0) ≤ n ⟹
             transpos (n − (Suc 0)) n (n − (Suc 0)) = n
apply (simp add:transpos-def)
done

lemma (in aGroup) additionTr0:⟦0 < n; ∀j ≤ n. f j ∈ carrier A⟧
 ⟹ nsum A (cmp f (transpos (n − 1) n)) n = nsum A f n
apply (case-tac n ≤ 1)
 apply simp
 apply (frule Suc-leI [of 0 n])
 apply (frule le-antisym [of n Suc 0], assumption+, simp)
 apply (simp add:cmp-def)
 apply (subst transpos-ij-1[of 0 Suc 0], simp+)
 apply (subst transpos-ij-2[of 0 Suc 0], simp+)
```

apply (*rule ag-pOp-commute*, *simp+*)
apply (*frule not-le-imp-less*[*of n Suc 0*])
apply (*frule-tac Suc-leI* [*of Suc 0 n*],
thin-tac $\neg$ $n \leq Suc\ 0$)
apply (*cut-tac nsum-suc*[*of A f n − Suc 0*], *simp*)
apply (*cut-tac nsum-suc*[*of A cmp f* (*transpos* (*n − Suc 0*) *n*) *n − Suc 0*],
simp,
thin-tac $\Sigma_e\ A\ f\ n = \Sigma_e\ A\ f\ (n - Suc\ 0) \pm f\ n$,
thin-tac $\Sigma_e\ A$ (*cmp f* (*transpos* (*n − Suc 0*) *n*)) *n* =
$\Sigma_e\ A$ (*cmp f* (*transpos* (*n − Suc 0*) *n*)) (*n − Suc 0*) ±
(*cmp f* (*transpos* (*n − Suc 0*) *n*) *n*))
apply (*case-tac n* = *Suc* (*Suc 0*), *simp*)
apply (*cut-tac transpos-id-1*[*of Suc 0 Suc* (*Suc 0*) *Suc* (*Suc 0*) *0*],
cut-tac transpos-ij-1[*of Suc 0 Suc* (*Suc 0*) *Suc* (*Suc 0*)],
cut-tac transpos-ij-2[*of Suc 0 Suc* (*Suc 0*) *Suc* (*Suc 0*)],
simp add:*cmp-def*,
thin-tac n = *Suc* (*Suc 0*),
thin-tac transpos (*Suc 0*) (*Suc* (*Suc 0*)) *0* = *0*,
thin-tac transpos (*Suc 0*) (*Suc* (*Suc 0*)) (*Suc 0*) = *Suc* (*Suc 0*),
thin-tac transpos (*Suc 0*) (*Suc* (*Suc 0*)) (*Suc* (*Suc 0*)) = *Suc 0*)
apply (*subst ag-pOp-assoc*, *simp+*)
apply (*subst ag-pOp-commute*[*of f* (*Suc* (*Suc 0*)) *f* (*Suc 0*)], *simp+*)
apply (*subst ag-pOp-assoc*[*THEN sym*], *simp+*)

apply (*frule not-sym*)
apply (*frule noteq-le-less*[*of Suc* (*Suc 0*) *n*], *assumption*,
thin-tac Suc (*Suc 0*) ≤ *n*)
apply (*cut-tac nsum-suc*[*of A f n − Suc 0 − Suc 0*])
apply (*cut-tac Suc-pred*[*of n − Suc 0*], *simp del*:*nsum-suc*)
apply (*cut-tac nsum-suc*[*of A cmp f* (*transpos* (*n − Suc 0*) *n*)
n − Suc (*Suc 0*)], *simp del*:*nsum-suc*,
thin-tac $\Sigma_e\ A\ f\ (n - Suc\ 0) = \Sigma_e\ A\ f\ (n - Suc\ (Suc\ 0)) \pm f\ (n - Suc\ 0)$,
thin-tac Suc (*n − Suc* (*Suc 0*)) = *n − Suc 0*,
thin-tac $\Sigma_e\ A$ (*cmp f* (*transpos* (*n − Suc 0*) *n*)) (*n − Suc 0*) =
$\Sigma_e\ A$ (*cmp f* (*transpos* (*n − Suc 0*) *n*)) (*n − Suc* (*Suc 0*)) ±
(*cmp f* (*transpos* (*n − Suc 0*) *n*)) (*n − Suc 0*))
apply (*cut-tac nsum-eq-i*[*of n cmp f* (*transpos* (*n − Suc 0*) *n*) *f*
n − Suc (*Suc 0*)], *simp*,
thin-tac $\Sigma_e\ A$ (*cmp f* (*transpos* (*n − Suc 0*) *n*)) (*n − Suc* (*Suc 0*)) =
$\Sigma_e\ A\ f\ (n - Suc\ (Suc\ 0))$)
apply (*simp add*:*cmp-def*)
apply (*cut-tac transpos-ij-1*[*of n − Suc 0 n n*], *simp*)
apply (*cut-tac transpos-ij-2*[*of n − Suc 0 n n*], *simp*)
apply (*subst ag-pOp-assoc*,
rule nsum-mem, *rule allI*, *rule impI*)
apply (*frule-tac x* = *j* **and** *y* = *n − Suc* (*Suc 0*) **and** *z* = *n* **in**
le-less-trans, *simp*, *frule-tac x* = *j* **and** *y* = *n* **in** *less-imp-le*)
apply *simp+*
apply (*subst ag-pOp-commute*[*of f n*], *simp*, *simp*)

apply (*subst ag-pOp-assoc*[*THEN sym*],
rule nsum-mem, *rule allI*, *rule impI*,
frule-tac $x = j$ **and** $y = n - Suc\ (Suc\ 0)$ **and** $z = n$ **in**
le-less-trans, *simp*, *frule-tac* $x = j$ **and** $y = n$ **in** *less-imp-le*)
apply *simp*+

apply (*rule allI*, *rule impI*, *simp add*:*cmp-def*)
apply (*cut-tac* $i = n - Suc\ 0$ **and** $n = n$ **and** $j = n$ **and** $l = j$ **in** *transpos-mem*,
simp+)

apply (*rule allI*, *rule impI*)
apply (*simp add*:*cmp-def*)
apply (*cut-tac* $i = n - Suc\ 0$ **and** $n = n$ **and** $j = n$ **and** $x = l$ **in** *transpos-id*,
simp+)

apply (*cut-tac* $x = l$ **and** $y = n - Suc\ (Suc\ 0)$ **and** $z = n$ **in** *le-less-trans*,
assumption) **apply** *simp*
apply *arith*
apply *simp*
apply *arith*
done

lemma (**in** *aGroup*) *additionTr1*:$[\![\forall f.\ \forall h.\ f \in \{j.\ j \leq (Suc\ n)\} \to carrier\ A \wedge$
$h \in \{j.\ j \leq (Suc\ n)\} \to \{j.\ j \leq (Suc\ n)\} \wedge inj\text{-}on\ h\ \{j.\ j \leq (Suc\ n)\} \longrightarrow$
$nsum\ A\ (cmp\ f\ h)\ (Suc\ n) = nsum\ A\ f\ (Suc\ n);$
$f \in \{j.\ j \leq (Suc\ (Suc\ n))\} \to carrier\ A;$
$h \in \{j.\ j \leq (Suc\ (Suc\ n))\} \to \{j.\ j \leq (Suc\ (Suc\ n))\};$
$inj\text{-}on\ h\ \{j.\ j \leq (Suc\ (Suc\ n))\};\ h\ (Suc\ (Suc\ n)) = Suc\ (Suc\ n)]\!]$
$\Longrightarrow nsum\ A\ (cmp\ f\ h)\ (Suc\ (Suc\ n)) = nsum\ A\ f\ (Suc\ (Suc\ n))$
apply (*subgoal-tac* $f \in \{j.\ j \leq (Suc\ n)\} \to carrier\ A$)
apply (*subgoal-tac* $h \in \{j.\ j \leq (Suc\ n)\} \to \{j.\ j \leq (Suc\ n)\}$)
apply (*subgoal-tac inj-on h* $\{j.\ j \leq (Suc\ n)\}$)
apply (*subgoal-tac nsum A* $(cmp\ f\ h)\ (Suc\ n) = nsum\ A\ f\ (Suc\ n)$)
apply (*thin-tac* $\forall f.\ \forall h.\ f \in \{j.\ j \leq (Suc\ n)\} \to carrier\ A \wedge$
$h \in \{j.\ j \leq (Suc\ n)\} \to \{j.\ j \leq (Suc\ n)\} \wedge inj\text{-}on\ h\ \{j.\ j \leq (Suc\ n)\} \longrightarrow$
$nsum\ A\ (cmp\ f\ h)\ (Suc\ n) = nsum\ A\ f\ (Suc\ n)$)
prefer *2* **apply** *simp*
apply *simp*
apply (*thin-tac nsum A* $(cmp\ f\ h)\ n \pm (cmp\ f\ h\ (Suc\ n)) =$ *nsum A f n* $\pm$ (f
$(Suc\ n)$))
apply (*simp add*:*cmp-def*)
apply (*thin-tac* $\forall f\ h.\ (f \in \{j.\ j \leq Suc\ n\} \to carrier\ A) \wedge$
$(h \in \{j.\ j \leq Suc\ n\} \to \{j.\ j \leq Suc\ n\}) \wedge (inj\text{-}on\ h\ \{j.\ j \leq Suc\ n\}) \longrightarrow$
$\Sigma_e\ A\ (cmp\ f\ h)\ (Suc\ n) = \Sigma_e\ A\ f\ (Suc\ n)$)
apply (*frule Nset-injTr0* [*of h Suc n*], *assumption*+, *simp*)
apply (*frule Nset-injTr0* [*of h Suc n*], *assumption*+, *simp*)
apply (*simp add*:*Pi-def*)
done

lemma (**in** *aGroup*) *additionTr1-1*:⟦∀ *f*. ∀ *h*. *f* ∈ {*j*. *j*≤*Suc n*} → *carrier A* ∧
h ∈ {*j*. *j*≤*Suc n*} → {*j*. *j*≤*Suc n*} ∧ *inj-on h* {*j*. *j*≤*Suc n*} ⟶
nsum A (*cmp f h*) (*Suc n*) = *nsum A f* (*Suc n*);
f ∈ {*j*. *j*≤*Suc* (*Suc n*)} → *carrier A*; *i* ≤ *n*⟧ ⟹
nsum A (*cmp f* (*transpos i* (*Suc n*))) (*Suc* (*Suc n*)) = *nsum A f* (*Suc* (*Suc n*))
apply (*rule additionTr1* [*of n f transpos i* (*Suc n*)], *assumption*+)
apply (*rule transpos-hom* [*of i Suc* (*Suc n*) *Suc n*])
apply *simp*+
apply (*rule transpos-inj* [*of i Suc* (*Suc n*) *Suc n*])
apply *simp*+
apply (*subst transpos-id*[*of i Suc* (*Suc n*) *Suc n Suc* (*Suc n*)])
apply *simp*+
done

lemma (**in** *aGroup*) *additionTr1-2*:⟦∀ *f*. ∀ *h*. *f* ∈ {*j*. *j*≤*Suc n*} → *carrier A* ∧
h ∈ {*j*. *j*≤*Suc n*} → {*j*. *j*≤*Suc n*} ∧
inj-on h {*j*. *j*≤*Suc n*} ⟶
nsum A (*cmp f h*) (*Suc n*) = *nsum A f* (*Suc n*);
f ∈ {*j*. *j*≤ *Suc* (*Suc n*)} → *carrier A*; *i* ≤ (*Suc n*)⟧ ⟹
nsum A (*cmp f* (*transpos i* (*Suc* (*Suc n*)))) (*Suc* (*Suc n*)) =
nsum A f (*Suc* (*Suc n*))
apply (*case-tac i* = *Suc n*)
apply (*simp del*:*nsum-suc*)
apply (*cut-tac additionTr0* [*of Suc* (*Suc n*) *f*], *simp*, *simp*,
rule allI, *rule impI*, *rule funcset-mem*[*of f* {*j*. *j* ≤ *Suc* (*Suc n*)}
carrier A], (*simp del*:*nsum-suc*)+)
apply (*subst nsum-cmp-eq-transpos* [*THEN sym*, *of Suc n f i*],
rule allI, *rule impI*, *rule funcset-mem*[*of f* {*j*. *j* ≤ *Suc* (*Suc n*)}
carrier A], *assumption*+,
simp, *assumption*+)
apply (*subst nsum-cmp-eq* [*of f Suc* (*Suc n*)
cmp (*transpos* (*Suc n*) (*Suc*(*Suc n*))) (*transpos i* (*Suc n*))
transpos i (*Suc n*) *Suc* (*Suc n*)], *assumption*+,
rule Pi-I, *simp add*:*cmp-def*,
rule transpos-mem, (*simp del*:*nsum-suc*)+,
rule transpos-mem, (*simp del*:*nsum-suc*)+,
rule Pi-I, *simp*,
rule transpos-mem, (*simp del*:*nsum-suc*)+)
apply (*subst nsum-cmp-eq* [*of cmp f* (*transpos i* (*Suc n*)) *Suc* (*Suc n*)
(*transpos i* (*Suc n*)) *transpos* (*Suc n*) (*Suc* (*Suc n*)) *Suc* (*Suc n*)],
rule Pi-I, *simp add*:*cmp-def*,
rule funcset-mem[*of f* {*j*. *j* ≤ *Suc* (*Suc n*)} *carrier A*], *assumption*,
simp,
rule transpos-mem, (*simp del*:*nsum-suc*)+,
(*rule Pi-I*, *simp*,
rule transpos-mem, (*simp del*:*nsum-suc*)+)+)

apply (*subst additionTr1-1* [*of n cmp* (*cmp f* (*transpos i* (*Suc n*)))
(*transpos* (*Suc n*) (*Suc* (*Suc n*))) *i*], *assumption*+,

rule cmp-fun [*of* - {*j*. *j* ≤ (*Suc* (*Suc n*))}
{*j*. *j* ≤ (*Suc* (*Suc n*))} - *carrier A*],
rule transpos-hom, *simp*, *simp*, *simp*,
rule cmp-fun [*of* - {*j*. *j* ≤ (*Suc* (*Suc n*))}
{*j*. *j* ≤ (*Suc* (*Suc n*))} *f carrier A*],
rule transpos-hom, *simp*, *simp*, *assumption*+, *arith*)
apply (*cut-tac additionTr0* [*of* *Suc* (*Suc n*) *cmp f* (*transpos i* (*Suc n*))],
simp del:*nsum-suc*,
thin-tac nsum A (*cmp*
(*cmp f* (*transpos i* (*Suc n*))) (*transpos* (*Suc n*) (*Suc* (*Suc n*)))) (*Suc* (*Suc n*))
= *nsum A* (*cmp f* (*transpos i* (*Suc n*))) (*Suc* (*Suc n*)))
apply (*rule additionTr1-1*, *assumption*+, *arith*, *simp*,
rule allI, *rule impI*, *simp add*:*cmp-def*,
rule funcset-mem[*of f* {*j*. *j* ≤ *Suc* (*Suc n*)} *carrier A*],
assumption)
apply (*simp add*:*transpos-mem*)
done

lemma (**in** *aGroup*) *additionTr2*: ∀ *f*. ∀ *h*. *f* ∈ {*j*. *j* ≤ (*Suc n*)} → *carrier A* ∧
h ∈ {*j*. *j* ≤ (*Suc n*)} → {*j*. *j* ≤ (*Suc n*)} ∧
inj-on h {*j*. *j* ≤ (*Suc n*)} ⟶
nsum A (*cmp f h*) (*Suc n*) = *nsum A f* (*Suc n*)
apply (*induct-tac n*)
apply (*rule allI*)+
apply (*rule impI*, (*erule conjE*)+)
apply (*simp add*:*cmp-def*)
apply (*case-tac h 0 = 0*)
apply (*simp add*:*Nset-1*)
apply (*simp add*:*Nset-1 ag-pOp-commute*)

apply (*rule allI*)+
apply (*rule impI*, (*erule conjE*)+)
apply (*case-tac h* (*Suc* (*Suc n*)) = *Suc* (*Suc n*))
apply (*rule additionTr1*, *assumption*+)
apply (*frule-tac f* = *h* **and** *n* = *Suc* (*Suc n*) **in** *inj-surj*, *assumption*+)
apply (*frule sym*, *thin-tac h* ‘ {*i*. *i* ≤ *Suc* (*Suc n*)} = {*i*. *i* ≤ *Suc* (*Suc n*)})
apply (*cut-tac n* = *Suc* (*Suc n*) **in** *n-in-Nsetn*)
apply (*frule-tac a* = *Suc* (*Suc n*) **and** *A* = {*i*. *i* ≤ *Suc* (*Suc n*)} **and**
B = *h* ‘ {*i*. *i* ≤ *Suc* (*Suc n*)} **in** *eq-set-inc*, *assumption*+)
apply (*thin-tac* {*i*. *i* ≤ *Suc* (*Suc n*)} = *h* ‘ {*i*. *i* ≤ *Suc* (*Suc n*)})
apply (*simp del*:*nsum-suc add*:*image-def*)
apply (*erule exE*, *erule conjE*)
apply (*frule sym*, *thin-tac Suc* (*Suc n*) = *h x*)
apply (*frule-tac i* = *x* **and** *n* = *Suc* (*Suc n*) **and** *j* = *Suc* (*Suc n*) **in**
transpos-ij-2, *simp del*:*nsum-suc add*:*n-in-Nsetn*)
apply (*rule contrapos-pp*, (*simp del*:*nsum-suc*)+)
apply (*frule-tac x* = *transpos x* (*Suc* (*Suc n*)) (*Suc* (*Suc n*)) **and** *y* = *x* **and**
f = *h* **in** *eq-elems-eq-val*,

thin-tac transpos x (Suc (Suc n)) (Suc (Suc n)) = x,
simp del:nsum-suc)
apply (*frule-tac f = h* **and** *A = {i. i ≤ Suc (Suc n)}* **and** *x = x* **and**
y = Suc (Suc n) **in** *inj-onTr2*, *simp*, *simp*,
frule not-sym, *simp*)
apply (*cut-tac f1 = cmp f h* **and** *n1 = n* **and** *i1 = x* **in**
additionTr1-2[*THEN sym*], *assumption*)
apply (*rule cmp-fun*, *simp*, *assumption*, *arith*)
apply (*simp del:nsum-suc*,
thin-tac Σ_e *A (cmp f h) (Suc (Suc n)) =*
Σ_e *A (cmp (cmp f h) (transpos x (Suc (Suc n)))) (Suc (Suc n))*)
apply (*frule-tac f = f* **and** *n = Suc n* **and** *A = carrier A* **in** *func-pre*)
apply (*cut-tac f = cmp h (transpos x (Suc (Suc n)))* **and** *A = {j. j ≤ (Suc (*
Suc n))} **and** *?A1.0 = {j. j ≤ (Suc n)}* **in** *restrict-inj*)
apply (*rule-tac f = transpos x (Suc (Suc n))* **and** *A = {j. j ≤ Suc (Suc n)}*
and *B = {j. j ≤ Suc (Suc n)}* **and** *g = h* **and** *C = {j. j ≤ Suc (Suc n)}* **in**
cmp-inj, *simp*,
rule transpos-hom, *simp*, *simp*, *assumption+*,
rule transpos-inj, *simp*, *simp*, *assumption+*,
rule subsetI, *simp*)
apply (*subst nsum-cmp-assoc*,
rule allI, *rule impI*, *simp add:Pi-def*,
rule transpos-hom, *assumption*, *simp*, *assumption+*)
apply (*cut-tac f = cmp f (cmp h (transpos x (Suc (Suc n))))* **and** *n = Suc n*
in *nsum-suc*[*of A*], *simp del:nsum-suc*,
thin-tac Σ_e *A (cmp f (cmp h (transpos x (Suc (Suc n))))) (Suc (Suc n)) =*
Σ_e *A (cmp f (cmp h (transpos x (Suc (Suc n))))) (Suc n)* $\pm$
cmp f (cmp h (transpos x (Suc (Suc n)))) (Suc (Suc n)))
apply (*frule-tac x = f* **in** *spec*,
thin-tac $\forall f\, h.\ f \in \{j.\ j \le Suc\ n\} \to carrier\ A \wedge$
$h \in \{j.\ j \le Suc\ n\} \to \{j.\ j \le Suc\ n\} \wedge$
inj-on h $\{j.\ j \le Suc\ n\} \longrightarrow$
Σ_e *A (cmp f h) (Suc n)* $= \Sigma_e$ *A f (Suc n)*)
apply (*frule-tac a = cmp h (transpos x (Suc (Suc n)))* **in** *forall-spec*,
thin-tac $\forall h.\ f \in \{j.\ j \le Suc\ n\} \to carrier\ A \wedge$
$h \in \{j.\ j \le Suc\ n\} \to \{j.\ j \le Suc\ n\} \wedge$ *inj-on h* $\{j.\ j \le Suc\ n\} \longrightarrow$
Σ_e *A (cmp f h) (Suc n)* $= \Sigma_e$ *A f (Suc n)*)
apply *simp*
apply (*rule Pi-I*)
apply (*simp add:cmp-def*)
apply (*case-tac xa = x*, *simp*)
apply (*cut-tac i = x* **and** *n = Suc (Suc n)* **and** *j = Suc (Suc n)* **in**
transpos-ij-1, *simp*, *simp*, *simp*, *simp*,
frule-tac x = Suc (Suc n) **and** *f = h* **and** *A = {j. j ≤ Suc (Suc n)}*
and *B = {j. j ≤ Suc (Suc n)}* **in** *funcset-mem*, *simp*,
thin-tac h ∈ {j. j ≤ Suc (Suc n)} → {j. j ≤ Suc (Suc n)})
apply (*cut-tac m = h (Suc (Suc n))* **and** *n = Suc (Suc n)* **in** *noteq-le-less*,
simp, *simp*,
rule-tac x = h (Suc (Suc n)) **and** *n = Suc n* **in** *Suc-less-le*,

assumption)
apply (*subst transpos-id*, *simp*, *simp*, *simp*, *simp*,
frule-tac $x = xa$ **and** $f = h$ **and** $A = \{j.\ j \leq Suc\ (Suc\ n)\}$ **and**
$B = \{j.\ j \leq Suc\ (Suc\ n)\}$ **in** *funcset-mem*, *simp*)
apply (*frule-tac* $f = h$ **and** $A = \{j.\ j \leq Suc\ (Suc\ n)\}$ **and** $x = xa$ **and** $y = x$
in *injective*, *simp*, *simp*, *assumption*)
apply (*cut-tac* $m = h\ xa$ **and** $n = Suc\ (Suc\ n)$ **in** *noteq-le-less*, *simp*,
simp)
apply (*rule Suc-less-le*, *assumption*,
thin-tac $\forall h.\ f \in \{j.\ j \leq Suc\ n\} \to carrier\ A \wedge$
$h \in \{j.\ j \leq Suc\ n\} \to \{j.\ j \leq Suc\ n\} \wedge inj\text{-}on\ h\ \{j.\ j \leq Suc\ n\} \longrightarrow$
$\Sigma_e\ A\ (cmp\ f\ h)\ (Suc\ n) = \Sigma_e\ A\ f\ (Suc\ n)$)
apply (*simp del*:*nsum-suc add*:*cmp-def*)
apply *simp*
done

lemma (**in** *aGroup*) *addition2*:$[\![f \in \{j.\ j \leq (Suc\ n)\} \to carrier\ A;$
$h \in \{j.\ j \leq (Suc\ n)\} \to \{j.\ j \leq (Suc\ n)\};\ inj\text{-}on\ h\ \{j.\ j \leq (Suc\ n)\}]\!] \Longrightarrow$
$nsum\ A\ (cmp\ f\ h)\ (Suc\ n) = nsum\ A\ f\ (Suc\ n)$
apply (*simp del*:*nsum-suc add*:*additionTr2*)
done

lemma (**in** *aGroup*) *addition21*:$[\![f \in \{j.\ j \leq n\} \to carrier\ A;$
$h \in \{j.\ j \leq n\} \to \{j.\ j \leq n\};\ inj\text{-}on\ h\ \{j.\ j \leq n\}]\!] \Longrightarrow$
$nsum\ A\ (cmp\ f\ h)\ n = nsum\ A\ f\ n$
apply (*case-tac* $n = 0$)
apply (*simp add*: *cmp-def*)
apply (*cut-tac* $f = f$ **and** $n = n - Suc\ 0$ **and** $h = h$ **in** *addition2*)
apply *simp+*
done

lemma (**in** *aGroup*) *addition3*:$[\![\forall j \leq (Suc\ n).\ f\ j \in carrier\ A;\ j \leq (Suc\ n);$
$j \neq Suc\ n]\!] \Longrightarrow nsum\ A\ f\ (Suc\ n) = nsum\ A\ (cmp\ f\ (transpos\ j\ (Suc\ n)))\ (Suc$
$n)$
apply (*rule addition2* [*THEN sym*,*of f n transpos j* (*Suc n*)])
apply (*simp*)
apply (*rule transpos-hom*, *assumption+*, *simp*, *assumption*)
apply (*rule transpos-inj*, *simp+*)
done

lemma (**in** *aGroup*) *nsum-splitTr*:$(\forall j \leq (Suc\ (n + m)).\ f\ j \in carrier\ A) \longrightarrow$
$nsum\ A\ f\ (Suc\ (n + m)) = nsum\ A\ f\ n \pm (nsum\ A\ (cmp\ f\ (slide\ (Suc\ n)))\ m)$
apply (*induct-tac m*)
apply (*rule impI*) **apply** (*simp add*:*slide-def cmp-def*)
apply (*rule impI*, *simp del*:*nsum-suc*)

apply (*cut-tac* $n = Suc\ (n + na)$ **in** *nsum-suc*[*of A f*],
simp del:*nsum-suc*,
thin-tac $\Sigma_e\ A\ f\ (Suc\ (Suc\ (n + na))) =$

$\Sigma_e\ A\ f\ n \pm \Sigma_e\ A\ (cmp\ f\ (slide\ (Suc\ n)))\ na \pm f\ (Suc\ (Suc\ (n + na))))$
apply (*cut-tac* $f = cmp\ f\ (slide\ (Suc\ n))$ **and** $n = na$ **in** *nsum-suc*[*of A*],
simp del:*nsum-suc*)
apply (*subst ag-pOp-assoc*)
apply (*rule nsum-mem*, *rule allI*, *simp*)
apply (*rule-tac* $n = na$ **in** *nsum-mem*,
thin-tac $\Sigma_e\ A\ (cmp\ f\ (slide\ (Suc\ n)))\ (Suc\ na) =$
$\Sigma_e\ A\ (cmp\ f\ (slide\ (Suc\ n)))\ na \pm (cmp\ f\ (slide\ (Suc\ n))\ (Suc\ na))$)
apply (*rule allI*, *rule impI*,
simp add:*cmp-def slide-def*, *simp*)
apply (*simp add*:*cmp-def slide-def*)
done

lemma (**in** *aGroup*) *nsum-split*:$\forall j \leq (Suc\ (n + m)).\ f\ j \in carrier\ A \Longrightarrow$
$nsum\ A\ f\ (Suc\ (n + m)) = nsum\ A\ f\ n \pm (nsum\ A\ (cmp\ f\ (slide\ (Suc\ n)))\ m)$

by (*simp del*:*nsum-suc add*:*nsum-splitTr*)

lemma (**in** *aGroup*) *nsum-split1*:$[\![\forall j \leq m.\ f\ j \in carrier\ A;\ n < m]\!] \Longrightarrow$
$nsum\ A\ f\ m = nsum\ A\ f\ n \pm (fSum\ A\ f\ (Suc\ n)\ m)$
apply (*cut-tac nsum-split*[*of n m − n − Suc 0 f*])
apply *simp*
apply (*simp add*:*fSum-def*)
apply *simp*
done

lemma (**in** *aGroup*) *nsum-minusTr*: $(\forall j \leq n.\ f\ j \in carrier\ A) \longrightarrow$
$-_a\ (nsum\ A\ f\ n) = nsum\ A\ (\lambda x \in \{j.\ j \leq n\}.\ -_a\ (f\ x))\ n$
apply (*induct-tac n*)
apply (*rule impI*, *simp*)
apply (*rule impI*)
apply (*subst nsum-suc*, *subst nsum-suc*)
apply (*subst ag-p-inv*)
apply (*rule-tac* $n = n$ **in** *nsum-mem* [*of - f*],
rule allI, *simp*, *simp*)
apply (*subgoal-tac* $\forall j \leq n.\ f\ j \in carrier\ A$, *simp*)
apply (*rule-tac* $a = \Sigma_e\ A\ (\lambda u.\ if\ u \leq n\ then\ -_a\ (f\ u)\ else\ undefined)\ n$
and $b = \Sigma_e\ A\ (\lambda x \in \{j.\ j \leq (Suc\ n)\}.\ -_a\ (f\ x))\ n$ **and** $c = -_a\ (f\ (Suc\ n))$
in *ag-pOp-add-r*,
rule-tac $n = n$ **in** *nsum-mem*,
rule allI, *rule impI*, *simp*,
rule ag-mOp-closed, *simp*)
apply (*rule-tac* $n = n$ **in** *nsum-mem*,
rule allI, *rule impI*, *simp*,
rule ag-mOp-closed, *simp*,
rule ag-mOp-closed, *simp*)
apply (*rule-tac* $f = \lambda u.\ if\ u \leq n\ then\ -_a\ (f\ u)\ else\ undefined$ **and**
$n = n$ **and** $g = \lambda x \in \{j.\ j \leq (Suc\ n)\}.\ -_a\ (f\ x)$ **in** *nsum-eq*,
rule allI, *rule impI*,

simp, *rule ag-mOp-closed*, *simp*,
rule allI, *simp*, *rule impI*, *rule ag-mOp-closed*, *simp*)
apply (*rule allI*, *simp*)
apply (*rule allI*, *simp*)
done

lemma (**in** *aGroup*) *nsum-minus*:$\forall j \leq n.\ f\,j \in carrier\ A \Longrightarrow$
$-_a\ (nsum\ A\ f\ n) = nsum\ A\ (\lambda x{\in}\{j.\ j \leq n\}.\ -_a\ (f\,x))\ n$
apply (*simp add*:*nsum-minusTr*)
done

lemma (**in** *aGroup*) *ring-nsum-zeroTr*:$(\forall j \leq (n{::}nat).\ f\,j \in carrier\ A) \wedge$
$(\forall j \leq n.\ f\,j = \mathbf{0}) \longrightarrow nsum\ A\ f\ n = \mathbf{0}$
apply (*induct-tac n*)
apply (*rule impI*) **apply** (*erule conjE*)+ **apply** *simp*

apply (*rule impI*, (*erule conjE*)+)
apply (*cut-tac n* = *n* **in** *Nsetn-sub-mem1*, *simp*)
apply (*simp add*:*ag-inc-zero*)
apply (*cut-tac ag-inc-zero*,
simp add:*ag-r-zero*)
done

lemma (**in** *aGroup*) *ring-nsum-zero*:$\forall j \leq (n{::}nat).\ f\,j = \mathbf{0} \implies \Sigma_e\ A\ f\ n = \mathbf{0}$
apply (*cut-tac ring-nsum-zeroTr*[*of n f*])
apply (*simp add*:*ag-inc-zero*)
done

lemma (**in** *aGroup*) *ag-nsum-1-nonzeroTr*:
$\forall f.\ (\forall j \leq n.\ f\,j \in carrier\ A) \wedge$
$(l \leq n \wedge (\forall j \in \{j.\ j \leq n\} - \{l\}.\ f\,j = \mathbf{0}))$
$\longrightarrow nsum\ A\ f\ n = f\,l$
apply (*induct-tac n*)
apply *simp*
apply (*rule allI*,
rule impI, (*erule conjE*)+)
apply (*case-tac l* = *Suc n*)
apply *simp*
apply (*subgoal-tac* $\{j.\ j \leq Suc\ n\} - \{Suc\ n\} = \{j.\ j \leq n\}$, *simp*,
frule ring-nsum-zero, *simp*)
apply (*rule ag-l-zero*, *simp*)
apply (*rule equalityI*, *rule subsetI*, *simp*,
rule subsetI, *simp*)
apply (*frule-tac m* = *l* **and** *n* = *Suc n* **in** *noteq-le-less*, *assumption*,
thin-tac l ≤ *Suc n*,
frule-tac x = *l* **and** *n* = *n* **in** *Suc-less-le*)
apply (*cut-tac n* = *n* **in** *Nsetn-sub-mem1*, *simp*)
apply (*thin-tac* $\forall f.\ (\forall j{\leq}n.\ f\,j \in carrier\ A) \wedge$
$(\forall j{\in}\{j.\ j \leq n\} - \{l\}.\ f\,j = \mathbf{0}) \longrightarrow$

```
            Σe A f n = f l,
        frule-tac a = l in forall-spec, simp)
 apply (simp add:ag-r-zero)
done

lemma (in aGroup) ag-nsum-1-nonzero:⟦∀ j ≤ n. f j ∈ carrier A; l ≤ n;
      ∀ j∈({j. j ≤ n} − {l}). f j = 0 ⟧ ⟹ nsum A f n = f l
apply (simp add:ag-nsum-1-nonzeroTr[of n l])
done

definition
  set-mult :: [- , 'a set, 'a set] ⇒ 'a set where
  set-mult R A B = {z. ∃ x∈A. ∃ y∈B.  x ·r R y = z}

definition
  sum-mult :: [- , 'a set, 'a set] ⇒ 'a set where
  sum-mult R A B = {x. ∃ n. ∃ f ∈ {j. j ≤ (n::nat)}
                        → set-mult R A B. nsum R f n = x}


lemma (in Ring) set-mult-sub:⟦A ⊆ carrier R; B ⊆ carrier R⟧ ⟹
                                  set-mult R A B ⊆ carrier R
apply (rule subsetI, simp add:set-mult-def, (erule bexE)+,
      frule sym, thin-tac xa ·r y = x, simp)
apply (rule ring-tOp-closed, (simp add:subsetD)+)
done

lemma (in Ring) set-mult-mono:⟦A1 ⊆ carrier R; A2 ⊆ carrier R; A1 ⊆ A2;
      B ⊆ carrier R⟧ ⟹ set-mult R A1 B ⊆ set-mult R A2 B
apply (rule subsetI)
 apply (simp add:set-mult-def, (erule bexE)+)
 apply (frule-tac c = xa in subsetD[of A1 A2], assumption+)
 apply blast
done

lemma (in Ring) sum-mult-Tr1:⟦A ⊆ carrier R; B ⊆ carrier R⟧ ⟹
            (∀ j ≤ n. f j ∈ set-mult R A B) ⟶ nsum R f n ∈ carrier R
apply (cut-tac ring-is-ag)
apply (induct-tac n)
 apply (rule impI, simp)
 apply (frule set-mult-sub[of A B], assumption, simp add:subsetD)
apply (rule impI)
 apply (cut-tac n = n in Nsetn-sub-mem1, simp)
 apply (frule set-mult-sub[of A B], assumption)
 apply (frule-tac a = Suc n in forall-spec, simp,
       frule-tac c = f (Suc n) in subsetD[of set-mult R A B carrier R],
       assumption)
 apply (rule aGroup.ag-pOp-closed, assumption+)
done
```

lemma (**in** *Ring*) *sum-mult-mem*:⟦$A \subseteq$ *carrier* R; $B \subseteq$ *carrier* R;
  $\forall j \leq n.\ f\ j \in$ *set-mult* $R\ A\ B$⟧ $\Longrightarrow$ *nsum* $R\ f\ n \in$ *carrier* R
apply (*cut-tac ring-is-ag*)
apply (*simp add*:*sum-mult-Tr1*)
done

lemma (**in** *Ring*) *sum-mult-mem1*:⟦$A \subseteq$ *carrier* R; $B \subseteq$ *carrier* R;
    $x \in$ *sum-mult* $R\ A\ B$⟧ $\Longrightarrow$
    $\exists n.\ \exists f \in \{j.\ j \leq (n::nat)\} \to$ *set-mult* $R\ A\ B.$ *nsum* $R\ f\ n = x$
by (*simp add*:*sum-mult-def*)

lemma (**in** *Ring*) *sum-mult-subR*:⟦$A \subseteq$ *carrier* R; $B \subseteq$ *carrier* R⟧ $\Longrightarrow$
                sum-mult $R\ A\ B \subseteq$ *carrier* R
apply (*rule subsetI*)
apply (*frule-tac* $x = x$ **in** *sum-mult-mem1*[*of A B*], *assumption*+)
apply (*erule exE*, *erule bexE*, *frule sym*, *thin-tac* $\Sigma_e\ R\ f\ n = x$, *simp*)
apply (*cut-tac ring-is-ag*)
apply (*rule aGroup.nsum-mem*[*of R*], *assumption*)
apply (*rule allI*, *rule impI*)
apply (*frule-tac* $f = f$ **and** $A = \{j.\ j \leq n\}$ **and** $B =$ *set-mult* $R\ A\ B$ **and**
    $x = j$ **in** *funcset-mem*, *simp*)
apply (*frule set-mult-sub*[*of A B*], *assumption*)
apply (*simp add*:*subsetD*)
done

lemma (**in** *Ring*) *times-mem-sum-mult*:⟦$A \subseteq$ *carrier* R; $B \subseteq$ *carrier* R;
    $a \in A$; $b \in B$ ⟧ $\Longrightarrow$ $a \cdot_r b \in$ *sum-mult* $R\ A\ B$
apply (*simp add*:*sum-mult-def*)
apply (*subgoal-tac* $(\lambda i \in \{j.\ j \leq (0::nat)\}.\ a \cdot_r b) \in \{j.\ j \leq 0\} \to$ *set-mult* $R\ A$
B)
apply (*subgoal-tac nsum* $R\ (\lambda i \in \{j.\ j \leq (0::nat)\}.\ a \cdot_r b)\ 0 = a \cdot_r b$)
apply *blast*
 apply *simp*
 apply (*rule Pi-I*, *simp add*:*set-mult-def*, *blast*)
done

lemma (**in** *Ring*) *mem-minus-sum-multTr2*:⟦$A \subseteq$ *carrier* R; $B \subseteq$ *carrier* R;
    $\forall j \leq n.\ f\ j \in$ *set-mult* $R\ A\ B$; $i \leq n$⟧ $\Longrightarrow f\ i \in$ *carrier* R
apply (*frule-tac* $a = i$ **in** *forall-spec*, *simp*)
apply (*frule set-mult-sub*[*of A B*], *assumption*, *simp add*:*subsetD*)
done

lemma (**in** *aGroup*) *nsum-jointfun*:⟦$\forall j \leq n.\ f\ j \in$ *carrier* A;
    $\forall j \leq m.\ g\ j \in$ *carrier* A⟧ $\Longrightarrow$
    $\Sigma_e\ A$ (*jointfun* $n\ f\ m\ g$) (*Suc* $(n + m)) = \Sigma_e\ A\ f\ n \pm (\Sigma_e\ A\ g\ m)$
 apply (*subst nsum-split*)
 apply (*rule allI*, *rule impI*)
 apply (*frule-tac* $f = f$ **and** $n = n$ **and** $A =$ *carrier* A **and** $g = g$ **and** $m = m$

and $B = carrier\ A$ **in** *jointfun-mem*, *assumption*+, *simp*)
apply (*subgoal-tac nsum A* (*jointfun n f m g*) $n = nsum\ A\ f\ n$)
apply (*subgoal-tac nsum A* (*cmp* (*jointfun n f m g*) (*slide* (*Suc n*))) $m =$
$nsum\ A\ g\ m$)
apply *simp*
apply (*thin-tac nsum A* (*jointfun n f m g*) $n = nsum\ A\ f\ n$)
apply (*rule nsum-eq*)
apply (*rule allI*, *rule impI*,
simp add:*cmp-def jointfun-def slide-def sliden-def*,
assumption)
apply (*rule allI*, *simp add*:*cmp-def jointfun-def slide-def sliden-def*)

apply (*rule nsum-eq*)
apply (*rule allI*, *rule impI*,
simp add:*jointfun-def*, *assumption*)
apply (*rule allI*, *rule impI*)
apply (*simp add*:*jointfun-def*)
done

lemma (**in** *Ring*) *sum-mult-pOp-closed*:⟦$A \subseteq carrier\ R$; $B \subseteq carrier\ R$;
$a \in sum\text{-}mult\ R\ A\ B$; $b \in sum\text{-}mult\ R\ A\ B$ ⟧ ⟹ $a \pm_R b \in sum\text{-}mult\ R\ A\ B$
apply (*cut-tac ring-is-ag*)
apply (*simp add*:*sum-mult-def*)
apply ((*erule exE*)+, (*erule bexE*)+)
apply (*rename-tac n m f g*)
apply (*frule sym*, *thin-tac* $\Sigma_e\ R\ f\ n = a$, *frule sym*,
thin-tac $\Sigma_e\ R\ g\ m = b$, *simp*)
apply (*frule set-mult-sub*[*of A B*], *assumption*)
apply (*subst aGroup.nsum-jointfun*[*THEN sym*, *of R*], *assumption*)
apply (*rule allI*, *rule impI*,
frule-tac $f = f$ **and** $A = \{j.\ j \leq n\}$ **and** $B = set\text{-}mult\ R\ A\ B$ **and**
$x = j$ **in** *funcset-mem*, *simp*, *simp add*:*subsetD*)
apply (*rule allI*, *rule impI*,
frule-tac $f = g$ **and** $A = \{j.\ j \leq m\}$ **and** $B = set\text{-}mult\ R\ A\ B$ **and**
$x = j$ **in** *funcset-mem*, *simp*, *simp add*:*subsetD*)
apply (*frule-tac* $f = f$ **and** $n = n$ **and** $A = set\text{-}mult\ R\ A\ B$ **and** $g = g$ **and** $m = m$
and $B = set\text{-}mult\ R\ A\ B$ **in** *jointfun-hom*, *assumption*+)
apply (*simp del*:*nsum-suc*)
apply *blast*
done

lemma (**in** *Ring*) *set-mult-mOp-closed*:⟦$A \subseteq carrier\ R$; *ideal R B*;
$x \in set\text{-}mult\ R\ A\ B$⟧ ⟹ $-_a\ x \in set\text{-}mult\ R\ A\ B$
apply (*cut-tac ring-is-ag*,
simp add:*set-mult-def*,
(*erule bexE*)+, *frule sym*, *thin-tac* $xa \cdot_r y = x$, *simp*,
frule-tac $c = xa$ **in** *subsetD*[*of A carrier R*], *assumption*+,

frule ideal-subset1[*of B*],
frule-tac c = y **in** *subsetD*[*of B carrier R*], *assumption*+,
simp add:*ring-inv1-2*,
frule-tac I = B **and** *x = y* **in** *ideal-inv1-closed*,
assumption+)
apply *blast*
done

lemma (**in** *Ring*) *set-mult-ring-times-closed*:⟦*A* ⊆ *carrier R*; *ideal R B*;
x ∈ *set-mult R A B*; *r* ∈ *carrier R*⟧ ⟹ *r* ·$_r$ *x* ∈ *set-mult R A B*
apply (*cut-tac ring-is-ag*,
simp add:*set-mult-def*,
(*erule bexE*)+, *frule sym*, *thin-tac xa* ·$_r$ *y = x*, *simp*,
frule-tac c = xa **in** *subsetD*[*of A carrier R*], *assumption*+,
frule ideal-subset1[*of B*],
frule-tac c = y **in** *subsetD*[*of B carrier R*], *assumption*,
frule-tac x = r **and** *y = xa* ·$_r$ *y* **in** *ring-tOp-commute*,
simp add:*ring-tOp-closed*, *simp*,
subst ring-tOp-assoc, *assumption*+)
apply (*frule-tac x = y* **and** *y = r* **in** *ring-tOp-commute*, *assumption*+,
simp,
frule-tac x = y **and** *r = r* **in** *ideal-ring-multiple* [*of B*], *assumption*+)
apply *blast*
done

lemma (**in** *Ring*) *set-mult-sub-sum-mult*:⟦*A* ⊆ *carrier R*; *ideal R B*⟧ ⟹
set-mult R A B ⊆ *sum-mult R A B*
apply (*rule subsetI*)
apply (*simp add*:*sum-mult-def*)
apply (*cut-tac f* = (λ*i*∈{*j*. *j* ≤ (*0*::*nat*)}. *x*) **in** *nsum-0*[*of R*])
apply (*cut-tac n-in-Nsetn*[*of 0*],
simp del:*nsum-0*)
apply (*cut-tac f* = λ*i*∈{*j*. *j* ≤ (*0*::*nat*)}. *x* **and** *B* = %-. *set-mult R A B* **in**
Pi-I[*of* {*j*. *j* ≤ *0*}],
simp)
apply (*subgoal-tac* Σ$_e$ *R* (λ*i*∈{*j*. *j* ≤ *0*}. *x*) *0 = x*)
apply *blast*
apply *simp*
done

lemma (**in** *Ring*) *sum-mult-pOp-closedn*:⟦*A* ⊆ *carrier R*; *ideal R B*⟧ ⟹
(∀ *j* ≤ *n*. *f j* ∈ *set-mult R A B*) ⟶ Σ$_e$ *R f n* ∈ *sum-mult R A B*
apply (*induct-tac n*)
apply (*rule impI*, *simp*)
apply (*frule set-mult-sub-sum-mult*[*of A B*], *assumption*+, *simp add*:*subsetD*)

apply (*rule impI*)
apply *simp*
apply (*frule-tac a = Suc n* **in** *forall-spec*, *simp*)

apply (*frule set-mult-sub-sum-mult*[*of A B*], *assumption*+,
frule-tac c= *f* (*Suc n*) **in**
subsetD[*of set-mult R A B sum-mult R A B*], *assumption*+)
apply (*rule sum-mult-pOp-closed*, *assumption*,
simp add:*ideal-subset1*, *assumption*+)
done

lemma (**in** *Ring*) *mem-minus-sum-multTr4*:⟦*A* ⊆ *carrier R*; *ideal R B*⟧ ⟹
(∀ *j* ≤ *n*. *f j* ∈ *set-mult R A B*) ⟶ $-_a$ (*nsum R f n*) ∈ *sum-mult R A B*
apply (*cut-tac ring-is-ag*)
apply (*induct-tac n*)
apply (*rule impI*)
apply (*cut-tac n-in-Nsetn*[*of 0*])
apply (*frule-tac x* = *f 0* **in** *set-mult-mOp-closed*[*of A B*], *assumption*+)
apply *simp*
apply (*frule set-mult-sub-sum-mult*[*of A B*], *assumption*+,
simp add:*subsetD*)
apply (*rule impI*)
apply (*cut-tac n* = *n* **in** *Nsetn-sub-mem1*, *simp*)
apply (*frule sum-mult-subR*[*of A B*], *simp add*:*ideal-subset1*)
apply (*frule-tac n* = *n* **and** *f* = *f* **in** *sum-mult-pOp-closedn*[*of A B*], *assumption*,
cut-tac n = *n* **in** *Nsetn-sub-mem1*, *simp*)
apply (*frule-tac c* = Σ_e *R f n* **in** *subsetD*[*of sum-mult R A B carrier R*],
assumption+,
frule-tac a = *Suc n* **in** *forall-spec*, *simp*,
thin-tac ∀ *j*≤*Suc n*. *f j* ∈ *set-mult R A B*,
frule set-mult-sub[*of A B*], *simp add*:*ideal-subset1*,
frule-tac c = *f* (*Suc n*) **in** *subsetD*[*of set-mult R A B carrier R*],
assumption+)
apply (*frule-tac x* = Σ_e *R f n* **and** *y* = *f* (*Suc n*) **in** *aGroup.ag-p-inv*[*of R*],
assumption+, *simp*)
apply (*rule-tac a* = $-_a$ (Σ_e *R f n*) **and** *b* = $-_a$ (*f* (*Suc n*)) **in**
sum-mult-pOp-closed[*of A B*], *assumption*+,
simp add:*ideal-subset1*, *assumption*)
apply (*frule-tac x* = *f* (*Suc n*) **in** *set-mult-mOp-closed*[*of A B*], *assumption*+,
frule set-mult-sub-sum-mult[*of A B*], *assumption*+)
apply (*simp add*:*subsetD*)
done

lemma (**in** *Ring*) *sum-mult-iOp-closed*:⟦*A* ⊆ *carrier R*; *ideal R B*;
x ∈ *sum-mult R A B* ⟧ ⟹ $-_a$ *x* ∈ *sum-mult R A B*
apply (*frule sum-mult-mem1* [*of A B x*],
simp add:*ideal-subset1*, *assumption*)
apply (*erule exE*, *erule bexE*, *frule sym*, *thin-tac* Σ_e *R f n* = *x*)
apply *simp*
apply (*frule-tac n* = *n* **and** *f* = *f* **in** *mem-minus-sum-multTr4*[*of A B*],
assumption+)
apply (*simp add*:*Pi-def*)
done

lemma (**in** *Ring*) *sum-mult-ring-multiplicationTr*:
  $[\![A \subseteq carrier\ R;\ ideal\ R\ B;\ r \in carrier\ R]\!] \Longrightarrow$
  $(\forall j \leq n.\ f\ j \in set\text{-}mult\ R\ A\ B) \longrightarrow r \cdot_r (nsum\ R\ f\ n) \in sum\text{-}mult\ R\ A\ B$
apply (*cut-tac ring-is-ag*)
apply (*induct-tac n*)
 apply (*rule impI*, *simp*)
 apply (*simp add*:*set-mult-def*)
 apply ((*erule bexE*)+, *frule sym*, *thin-tac* $x \cdot_r y = f\ 0$, *simp*)
 apply (*frule-tac c* = *x* **in** *subsetD*[*of A carrier R*], *assumption*+)
 apply (*frule ideal-subset1*[*of B*],
   frule-tac c = *y* **in** *subsetD*[*of B carrier R*], *assumption*,
   frule-tac x = *r* **and** *y* = *x* $\cdot_r$ *y* **in** *ring-tOp-commute*,
   simp add:*ring-tOp-closed*, *simp*,
   subst ring-tOp-assoc, *assumption*+)
 apply (*frule-tac x* = *y* **and** *y* = *r* **in** *ring-tOp-commute*, *assumption*+,
   simp,
   frule-tac x = *y* **and** *r* = *r* **in** *ideal-ring-multiple* [*of B*], *assumption*+)
 apply (*rule times-mem-sum-mult*, *assumption*+)

apply (*rule impI*)
 apply (*cut-tac n* = *n* **in** *Nsetn-sub-mem1*, *simp*)
 apply (*frule-tac f* = *f* **and** *n* = *n* **in** *aGroup.nsum-mem*,
   frule set-mult-sub [*of A B*], *simp add*:*ideal-subset1*,
   rule allI, *rule impI*, *cut-tac n* = *n* **in** *Nsetn-sub-mem1*,
    simp add: *subsetD*,
   frule-tac a = *Suc n* **in** *forall-spec*, *simp*)
 apply (*frule set-mult-sub*[*of A B*], *simp add*:*ideal-subset1*,
   frule-tac c = *f* (*Suc n*) **in** *subsetD*[*of set-mult R A B carrier R*],
   assumption)
 apply (*simp add*: *ring-distrib1*)
 apply (*rule sum-mult-pOp-closed*[*of A B*], *assumption*+,
   simp add:*ideal-subset1*, *assumption*)
 apply (*frule-tac x* = *f* (*Suc n*) **in** *set-mult-ring-times-closed* [*of A B - r*],
   assumption+, *simp*, *assumption*,
   frule set-mult-sub-sum-mult[*of A B*], *assumption*+,
   simp add:*subsetD*)
done

lemma (**in** *Ring*) *sum-mult-ring-multiplication*:$[\![A \subseteq carrier\ R;\ ideal\ R\ B;$
 $r \in carrier\ R;\ a \in sum\text{-}mult\ R\ A\ B]\!] \Longrightarrow r \cdot_r a \in sum\text{-}mult\ R\ A\ B$
apply (*cut-tac ring-is-ag*)
apply (*frule sum-mult-mem1*[*of A B a*],
   simp add:*ideal-subset1*, *assumption*)
apply (*erule exE*, *erule bexE*, *frule sym*, *thin-tac* $\Sigma_e\ R\ f\ n = a$, *simp*)
apply (*subgoal-tac* $\forall j \leq n.\ f\ j \in set\text{-}mult\ R\ A\ B$)
apply (*simp add*:*sum-mult-ring-multiplicationTr*)
apply (*simp add*:*Pi-def*)
done

lemma (**in** *Ring*) *ideal-sum-mult*:⟦$A \subseteq$ *carrier R*; $A \neq \{\}$; *ideal R B*⟧ ⟹
ideal R (*sum-mult R A B*)
apply (*simp add*:*ideal-def* [*of* - *sum-mult R A B*])
apply (*cut-tac ring-is-ag*)
apply (*rule conjI*)
apply (*rule aGroup.asubg-test*, *assumption*+)
apply (*rule subsetI*)
apply (*frule-tac x* = *x* **in** *sum-mult-mem1*[*of A B*],
simp add:*ideal-subset1*, *assumption*,
erule exE, *erule bexE*, *frule sym*, *thin-tac* Σ_e *R f n* = *x*, *simp*)
apply (*rule-tac f* = *f* **and** *n* = *n* **in** *sum-mult-mem*[*of A B* - -], *assumption*+)
apply (*simp add*:*ideal-subset1*)
apply (*simp add*:*Pi-def*)
apply (*frule nonempty-ex*[*of A*], *erule exE*)
apply (*frule ideal-zero*[*of B*])
apply (*frule-tac a* = *x* **and** *b* = **0** **in** *times-mem-sum-mult*[*of A B*],
simp add:*ideal-subset1*, *assumption*+) **apply** *blast*

apply (*rule ballI*)+
apply (*rule-tac a* = *a* **and** *b* = $-_a$ *b* **in** *sum-mult-pOp-closed*[*of A B*],
assumption, *simp add*:*ideal-subset1*, *assumption*+,
rule-tac x = *b* **in** *sum-mult-iOp-closed*[*of A B*], *assumption*+)
apply (*rule ballI*)+
apply (*rule sum-mult-ring-multiplication*, *assumption*+)
done

lemma (**in** *Ring*) *ideal-inc-set-multTr*:⟦$A \subseteq$ *carrier R*; *ideal R B*; *ideal R C*;
set-mult R A B $\subseteq$ *C* ⟧ ⟹
$\forall f \in \{j.\ j \leq (n::nat)\} \rightarrow$ *set-mult R A B*. Σ_e *R f n* $\in$ *C*
apply (*induct-tac n*)
apply (*simp add*:*subsetD*)

apply (*rule ballI*)
apply (
frule-tac f = *f* **and** *A* = $\{j.\ j \leq$ *Suc n*$\}$ **and** *x* = *Suc n* **and**
B = *set-mult R A B***in** *funcset-mem*, *simp*,
frule-tac c = *f* (*Suc n*) **in** *subsetD*[*of set-mult R A B C*],
assumption+, *simp*)
apply (*rule ideal-pOp-closed*[*of C*], *assumption*+,
cut-tac n = *n* **in** *Nsetn-sub-mem1*,
frule-tac x = *f* **in** *bspec*, *simp*)
apply (*simp add*:*Pi-def*, *assumption*+)
done

lemma (**in** *Ring*) *ideal-inc-set-mult*:⟦$A \subseteq$ *carrier R*; *ideal R B*; *ideal R C*;
set-mult R A B $\subseteq$ *C* ⟧ ⟹ *sum-mult R A B* $\subseteq$ *C*
apply (*rule subsetI*)
apply (*frule-tac x* = *x* **in** *sum-mult-mem1*[*of A B*],

simp add:*ideal-subset1*, *assumption*+)
apply (*erule exE*, *erule bexE*, *frule sym*, *thin-tac* $\Sigma_e\ R\ f\ n = x$, *simp*,
thin-tac $x = \Sigma_e\ R\ f\ n$, *simp add*:*subsetD*)
apply (*simp add*:*ideal-inc-set-multTr*)
done

lemma (**in** *Ring*) *AB-inc-sum-mult*:⟦*ideal R A*; *ideal R B*⟧ ⟹
sum-mult R A B ⊆ *A* ∩ *B*
apply (*frule ideal-subset1*[*of A*], *frule ideal-subset1*[*of B*])
apply (*frule ideal-inc-set-mult* [*of A B A*], *assumption*+)
apply (*rule subsetI*,
simp add:*set-mult-def*, (*erule bexE*)+, *frule sym*, *thin-tac xa* ·$_r$ *y* = *x*,
simp,
frule-tac c = *xa* **in** *subsetD*[*of A carrier R*], *assumption*+,
frule-tac c = *y* **in** *subsetD*[*of B carrier R*], *assumption*+,
subst ring-tOp-commute, *assumption*+,
simp add:*ideal-ring-multiple*)
apply (*frule ideal-inc-set-mult* [*of A B B*], *assumption*+)
apply (*rule subsetI*,
simp add:*set-mult-def*, (*erule bexE*)+, *frule sym*, *thin-tac xa* ·$_r$ *y* = *x*,
simp,
frule-tac c = *xa* **in** *subsetD*[*of A carrier R*], *assumption*+,
simp add:*ideal-ring-multiple*)
apply *simp*
done

lemma (**in** *Ring*) *sum-mult-is-ideal-prod*:⟦*ideal R A*; *ideal R B*⟧ ⟹
sum-mult R A B = *A* ◇$_r$ *B*
apply (*rule equalityI*)
apply (*frule ideal-prod-ideal* [*of A B*], *assumption*+)
apply (*rule ideal-inc-set-mult*)
apply (*simp add*:*ideal-subset1*)+
apply (*rule subsetI*)
apply (*simp add*:*set-mult-def ideal-prod-def*)
apply (*auto del*:*subsetI*)
apply (*rule subsetI*)
apply (*simp add*:*ideal-prod-def*)
apply (*frule ideal-subset1*[*of A*],
frule ideal-sum-mult[*of A B*],
frule ideal-zero[*of A*], *blast*, *assumption*)
apply (*frule-tac a* = *sum-mult R A B* **in** *forall-spec*, *simp*)
apply (*rule subsetI*, *simp*,
thin-tac ∀ *xa*. *ideal R xa* ∧ {*x*. ∃ *i*∈*A*. ∃ *j*∈*B*. *x* = *i* ·$_r$ *j*} ⊆ *xa* ⟶ *x* ∈ *xa*,
(*erule bexE*)+, *simp*)
apply (*rule times-mem-sum-mult*, *assumption*,
simp add:*ideal-subset1*, *assumption*+)
done

lemma (**in** *Ring*) *ideal-prod-assocTr0*:⟦*ideal R A*; *ideal R B*; *ideal R C*; *y* ∈ *C*;

$z \in$ *set-mult R A B*$\rrbracket \Longrightarrow z \cdot_r y \in$ *sum-mult R A* $(B \diamondsuit_r C)$
apply (*simp add*:*set-mult-def*, (*erule bexE*)+,
frule sym, *thin-tac* $x \cdot_r ya = z$, *simp*)
apply (*frule-tac* $h = x$ **in** *ideal-subset*[*of A*], *assumption*,
frule-tac $h = ya$ **in** *ideal-subset*[*of B*], *assumption*,
frule-tac $h = y$ **in** *ideal-subset*[*of C*], *assumption*,
subst ring-tOp-assoc, *assumption*+)
apply (*frule ideal-subset1*[*of A*],
frule ideal-subset1[*of B*],
frule ideal-subset1[*of C*],
frule ideal-prod-ideal[*of B C*], *assumption*,
frule ideal-subset1[*of* $B \diamondsuit_r C$])
apply (*rule times-mem-sum-mult*[*of A B* $\diamondsuit_r$ *C*], *assumption*+,
subst sum-mult-is-ideal-prod[*of B C*, *THEN sym*], *assumption*+,
rule times-mem-sum-mult[*of B C*], *assumption*+)
done

lemma (**in** *Ring*) *ideal-prod-assocTr1*:$\llbracket$*ideal R A*; *ideal R B*; *ideal R C*; $y \in C\rrbracket$
$\Longrightarrow \forall f \in \{j.\ j \leq (n::nat)\} \rightarrow$ *set-mult R A B*. $(\Sigma_e\ R\ f\ n) \cdot_r y \in A \diamondsuit_r (B \diamondsuit_r C)$
apply (*cut-tac ring-is-ag*)
apply (*frule ideal-prod-ideal*[*of B C*], *assumption*+,
subst sum-mult-is-ideal-prod[*of A B* $\diamondsuit_r$ *C*, *THEN sym*], *assumption*+)
apply (*induct-tac n*)
apply *simp*
apply (*simp add*:*ideal-prod-assocTr0*)
apply (*rule ballI*,
frule-tac $x = f$ **in** *bspec*,
thin-tac $\forall f \in \{j.\ j \leq n\} \rightarrow$ *set-mult R A B*.
$\Sigma_e\ R\ f\ n \cdot_r y \in$ *sum-mult R A* $(B \diamondsuit_r C)$,
rule Pi-I, *simp*,
frule-tac $f = f$ **and** $A = \{j.\ j \leq Suc\ n\}$ **and** $B =$ *set-mult R A B* **and**
$x = x$ **in** *funcset-mem*, *simp*, *assumption*)
apply *simp*

apply (*frule ideal-subset1*[*of A*], *frule ideal-subset1*[*of B*],
frule set-mult-sub[*of A B*], *assumption*,
frule-tac $f = f$ **and** $A = \{j.\ j \leq (Suc\ n)\}$ **in** *extend-fun*[*of - -*
set-mult R A B
carrier R], *assumption*,
subst ring-distrib2,
simp add:*ideal-subset*)
apply (*rule aGroup.nsum-mem*, *assumption*)
apply (*simp add*:*Pi-def*)
apply (*simp add*:*funcset-mem del*:*Pi-I'*,
frule-tac $f = f$ **and** $A = \{j.\ j \leq Suc\ n\}$ **and** $B =$ *set-mult R A B* **and**
$x =$ *Suc n* **in** *funcset-mem*, *simp*)
apply (*frule ideal-subset1*[*of A*],
frule ideal-zero[*of A*],
frule ideal-sum-mult[*of A B* $\diamondsuit_r$ *C*], *blast*, *assumption*)

apply (*rule ideal-pOp-closed*, *assumption*+)
apply (*simp add*:*ideal-prod-assocTr0*)
done

lemma (**in** *Ring*) *ideal-quotient-idealTr*:⟦*ideal R A*; *ideal R B*; *ideal R C*;
x ∈ *carrier R*;∀ *c*∈*C*. *x* ·$_r$ *c* ∈ *ideal-quotient R A B*⟧ ⟹
f∈{*j*. *j* ≤ *n*} → *set-mult R B C* ⟶ *x* ·$_r$ (*nsum R f n*) ∈ *A*

apply (*frule ideal-subset1* [*of A*],
frule ideal-subset1 [*of B*])
apply (*induct-tac n*)
apply (*rule impI*)
apply (*cut-tac n-in-Nsetn*[*of 0*])
apply (*frule funcset-mem*, *assumption*+)
apply (*thin-tac f* ∈ {*j*. *j* ≤ *0*} → *set-mult R B C*)
apply (*simp add*:*set-mult-def*)
apply (*erule bexE*)+
apply (*frule sym*, *thin-tac xa* ·$_r$ *y* = *f 0*, *simp*)
apply (*frule-tac h* = *xa* **in** *ideal-subset*[*of B*], *assumption*,
frule-tac h = *y* **in** *ideal-subset*[*of C*], *assumption*)
apply (*frule-tac x* = *xa* **and** *y* = *y* **in** *ring-tOp-commute*, *assumption*+,
simp)
apply (*subst ring-tOp-assoc*[*THEN sym*], *assumption*+)
apply (*frule-tac x* = *y* **in** *bspec*, *assumption*,
thin-tac ∀ *c*∈*C*. *x* ·$_r$ *c* ∈ *A* †$_R$ *B*)
apply (*simp add*:*ideal-quotient-def*)

apply (*rule impI*)
apply (*frule func-pre*) **apply** *simp*
apply (*cut-tac ring-is-ag*)
apply (*frule ideal-subset1*[*of B*], *frule ideal-subset1*[*of C*],
frule set-mult-sub[*of B C*], *assumption*+)
apply (*cut-tac n* = *n* **in** *nsum-memr* [*of - f*],
rule allI, *rule impI*,
frule-tac x = *i* **in** *funcset-mem*, *simp*, *simp add*:*subsetD*)
apply (*frule-tac a* = *n* **in** *forall-spec*, *simp*)
apply (*thin-tac* ∀ *l*≤*n*. Σ$_e$ *R f l* ∈ *carrier R*,
frule-tac f = *f* **and** *A* = {*j*. *j* ≤ *Suc n*} **and** *B* = *set-mult R B C*
and *x* = *Suc n* **in** *funcset-mem*, *simp*,
frule-tac c = *f* (*Suc n*) **in** *subsetD*[*of set-mult R B C carrier R*],
assumption+)
apply (*subst ring-distrib1*, *assumption*+)
apply (*rule ideal-pOp-closed*[*of A*], *assumption*+)
apply (*simp add*: *set-mult-def*, (*erule bexE*)+,
fold set-mult-def,
frule sym, *thin-tac xa* ·$_r$ *y* = *f* (*Suc n*), *simp*)
apply (*frule-tac c* = *xa* **in** *subsetD*[*of B carrier R*], *assumption*+,
frule-tac c = *y* **in** *subsetD*[*of C carrier R*], *assumption*+,
frule-tac x = *xa* **and** *y* = *y* **in** *ring-tOp-commute*, *assumption*, *simp*,

subst ring-tOp-assoc[*THEN sym*], *assumption*+)
apply (*simp add*:*ideal-quotient-def*)
done

lemma (**in** *Ring*) *ideal-quotient-ideal*:⟦*ideal R A*; *ideal R B*; *ideal R C*⟧ ⟹
A †$_R$ *B* †$_R$ *C* = *A* †$_R$ *B* ◇$_r$ *C*
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*simp add*:*ideal-quotient-def* [*of* - - *C*])
apply (*erule conjE*)
apply (*simp add*:*ideal-quotient-def* [*of* - - *B* ◇$_r$ *C*])
apply (*rule ballI*)
apply (*simp add*:*sum-mult-is-ideal-prod* [*THEN sym*])
apply (*simp add*:*sum-mult-def*)
apply (*erule exE*, *erule bexE*)
apply (*rename-tac x c n f*)
apply (*frule sym*) **apply** *simp*
apply (*simp add*:*ideal-quotient-idealTr*)
apply (*rule subsetI*)
apply (*simp add*:*sum-mult-is-ideal-prod* [*THEN sym*])
apply (*simp add*:*ideal-quotient-def*)
apply (*erule conjE*)
apply (*rule ballI*)
apply (*rename-tac x c*)
apply (*frule ideal-subset* [*of C*], *assumption*+)
apply (*simp add*:*ring-tOp-closed*)
apply (*rule ballI*)
apply (*rename-tac x v u*)
apply (*frule ideal-subset* [*of B*], *assumption*+)
apply (*subst ring-tOp-assoc*, *assumption*+)
apply (*frule ideal-subset1*[*of B*],
frule ideal-subset1[*of C*],
frule-tac a = *u* **and** *b* = *v* **in** *times-mem-sum-mult*[*of B C*], *assumption*+)
apply (*frule-tac x* = *u* **and** *y* = *v* **in** *ring-tOp-commute*, *assumption*,
simp)
done

lemma (**in** *Ring*) *ideal-prod-assocTr*:⟦*ideal R A*; *ideal R B*; *ideal R C*⟧ ⟹
∀*f*. (*f* ∈ {*j*. *j* ≤ (*n*::*nat*)} → *set-mult R* (*A* ◇$_r$ *B*) *C* ⟶
(Σ$_e$ *R f n*) ∈ *A* ◇$_r$ (*B* ◇$_r$ *C*))
apply (*subgoal-tac* ∀ *x*∈(*A* ◇$_r$ *B*). ∀ *y*∈*C*. *x* ·$_r$ *y* ∈ *A* ◇$_r$ (*B* ◇$_r$ *C*))
apply (*induct-tac n*)
apply (*rule allI*) **apply** (*rule impI*)
apply (*frule-tac f* = *f* **and** *A* = {*j*. *j* ≤ *0*} **and** *B* = *set-mult R* (*A* ◇$_r$ *B*) *C*
and *x* = *0* **in** *funcset-mem*, *simp*, *simp*)
apply (*simp add*:*set-mult-def*)
apply ((*erule bexE*)+, *frule sym*, *thin-tac x* ·$_r$ *y* = *f 0*, *simp*)
apply (*rule allI*, *rule impI*)
apply (*frule func-pre*)

apply (*frule-tac* $a = f$ **in** *forall-spec*, *simp*,
thin-tac $\forall f.\ f \in \{j.\ j \leq n\} \to$ *set-mult* $R\ (A \diamondsuit_r B)\ C \longrightarrow$
$\Sigma_e\ R\ f\ n \in A \diamondsuit_r (B \diamondsuit_r C)$,
frule ideal-prod-ideal[*of* $B\ C$], *assumption*+,
frule ideal-prod-ideal[*of* $A\ B \diamondsuit_r C$], *assumption*+, *simp*)
apply (*rule ideal-pOp-closed*[*of* $A \diamondsuit_r (B \diamondsuit_r C)$], *assumption*+)
apply (*cut-tac* $n = Suc\ n$ **in** *n-in-Nsetn*,
frule-tac $f = f$ **and** $A = \{j.\ j \leq Suc\ n\}$ **and**
$B =$ *set-mult* $R\ (A \diamondsuit_r B)\ C$ **and** $x = Suc\ n$ **in** *funcset-mem*, *assumption*)
apply (*thin-tac* $f \in \{j.\ j \leq n\} \to$ *set-mult* $R\ (A \diamondsuit_r B)\ C$,
thin-tac $f \in \{j.\ j \leq Suc\ n\} \to$ *set-mult* $R\ (A \diamondsuit_r B)\ C$)
apply (*simp add*:*set-mult-def*)
apply ((*erule bexE*)+,
frule sym, *thin-tac* $x \cdot_r y = f\ (Suc\ n)$, *simp*)

apply (*rule ballI*)+
apply (*simp add*:*sum-mult-is-ideal-prod*[*of* $A\ B$, *THEN sym*])
apply (*frule ideal-subset1*[*of* A], *frule ideal-subset1*[*of* B],
frule-tac $x = x$ **in** *sum-mult-mem1*[*of* $A\ B$], *assumption*+)
apply (*erule exE*, *erule bexE*, *frule sym*, *thin-tac* $\Sigma_e\ R\ f\ n = x$,
simp)
apply (*simp add*:*ideal-prod-assocTr1*)
done

lemma (**in** *Ring*) *ideal-prod-assoc*:⟦*ideal* $R\ A$; *ideal* $R\ B$; *ideal* $R\ C$⟧ ⟹
$(A \diamondsuit_r B) \diamondsuit_r C = A \diamondsuit_r (B \diamondsuit_r C)$
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*frule ideal-prod-ideal*[*of* $A\ B$], *assumption*+)
apply (*frule sum-mult-is-ideal-prod*[*of* $A \diamondsuit_r B\ C$], *assumption*+)
apply (*frule sym*) **apply** (*thin-tac sum-mult* $R\ (A \diamondsuit_r B)\ C = (A \diamondsuit_r B) \diamondsuit_r C$)
apply *simp* **apply** (*thin-tac* $(A \diamondsuit_r B) \diamondsuit_r C =$ *sum-mult* $R\ (A \diamondsuit_r B)\ C$)
apply (*thin-tac ideal* $R\ (A \diamondsuit_r B)$)
apply (*frule ideal-prod-ideal*[*of* $B\ C$], *assumption*+)
apply (*simp add*:*sum-mult-def*)
apply (*erule exE*, *erule bexE*)
apply (*frule sym*, *thin-tac* $\Sigma_e\ R\ f\ n = x$, *simp*)
apply (*simp add*:*ideal-prod-assocTr*)
apply (*rule subsetI*)
apply (*frule ideal-prod-ideal*[*of* $B\ C$], *assumption*+)
apply (*simp add*:*ideal-prod-commute* [*of* $A\ B \diamondsuit_r C$])
apply (*frule ideal-prod-ideal*[*of* $A\ B$], *assumption*+)
apply (*simp add*:*ideal-prod-commute*[*of* $A \diamondsuit_r B\ C$])
apply (*simp add*:*ideal-prod-commute*[*of* $A\ B$])
apply (*simp add*:*ideal-prod-commute*[*of* $B\ C$])
apply (*frule ideal-prod-ideal*[*of* $C\ B$], *assumption*+)
apply (*frule sum-mult-is-ideal-prod*[*of* $C \diamondsuit_r B\ A$], *assumption*+)
apply (*frule sym*) **apply** (*thin-tac sum-mult* $R\ (C \diamondsuit_r B)\ A = (C \diamondsuit_r B) \diamondsuit_r A$)
apply *simp* **apply** (*thin-tac* $(C \diamondsuit_r B) \diamondsuit_r A =$ *sum-mult* $R\ (C \diamondsuit_r B)\ A$)

apply (*thin-tac ideal R* ($C \diamondsuit_r B$))
apply (*frule ideal-prod-ideal*[*of B A*], *assumption*+)
apply (*simp add*:*sum-mult-def*)
apply (*erule exE*, *erule bexE*)
apply (*frule sym*, *thin-tac* $\Sigma_e\ R\ f\ n = x$, *simp*)
apply (*simp add*:*ideal-prod-assocTr*)
done

lemma (**in** *Ring*) *prod-principal-idealTr0*: $[\![a \in carrier\ R;\ b \in carrier\ R;$
$z \in set\text{-}mult\ R\ (R \diamondsuit_p a)\ (R \diamondsuit_p b)]\!] \Longrightarrow\ z \in R \diamondsuit_p (a \cdot_r b)$
apply (*simp add*:*set-mult-def*, (*erule bexE*)+,
simp add:*Rxa-def*, (*erule bexE*)+, *simp*)
apply (*frule-tac* $x = r$ **and** $y = a$ **and** $z = ra \cdot_r b$ **in** *ring-tOp-assoc*,
assumption+, *simp add*:*ring-tOp-closed*, *simp*)
apply (*simp add*:*ring-tOp-assoc*[*THEN sym*, *of a - b*])
apply (*frule-tac* $x = a$ **and** $y = ra$ **in** *ring-tOp-commute*, *assumption*+, *simp*)
apply (*simp add*:*ring-tOp-assoc*[*of - a b*],
frule-tac $x = a$ **and** $y = b$ **in** *ring-tOp-closed*, *assumption*)
apply (*simp add*:*ring-tOp-assoc*[*THEN sym*, *of - - a* $\cdot_r$ *b*],
frule sym, *thin-tac* $r \cdot_r ra \cdot_r (a \cdot_r b) = z$, *simp*,
frule-tac $x = r$ **and** $y = ra$ **in** *ring-tOp-closed*, *assumption*+)
apply *blast*
done

lemma (**in** *Ring*) *prod-principal-idealTr1*: $[\![a \in carrier\ R;\ b \in carrier\ R]\!] \Longrightarrow$
$\forall f \in \{j.\ j \leq (n::nat)\} \rightarrow set\text{-}mult\ R\ (R \diamondsuit_p a)\ (R \diamondsuit_p b).$
$\Sigma_e\ R\ f\ n \in R \diamondsuit_p (a \cdot_r b)$
apply (*induct-tac n*)
apply (*rule ballI*,
frule-tac $f = f$ **in** *funcset-mem*[*of* - $\{j.\ j \leq 0\}$
set-mult R ($R \diamondsuit_p a$) ($R \diamondsuit_p b$)], *simp*)
apply (*simp add*:*prod-principal-idealTr0*)
apply (*rule ballI*,
frule func-pre,
frule-tac $x = f$ **in** *bspec*, *assumption*,
thin-tac $\forall f \in \{j.\ j \leq n\} \rightarrow set\text{-}mult\ R\ (R \diamondsuit_p a)\ (R \diamondsuit_p b).$
$\Sigma_e\ R\ f\ n \in R \diamondsuit_p (a \cdot_r b)$)
apply (*frule ring-tOp-closed*[*of a b*], *assumption*)
apply (*frule principal-ideal*[*of a* $\cdot_r$ *b*], *simp*,
rule ideal-pOp-closed, *assumption*+)
apply (*cut-tac* $n = Suc\ n$ **in** *n-in-Nsetn*,
frule-tac $f = f$ **and** $A = \{j.\ j \leq Suc\ n\}$ **and**
$B = set\text{-}mult\ R\ (R \diamondsuit_p a)\ (R \diamondsuit_p b)$ **in** *funcset-mem*, *assumption*)
apply (*simp add*:*prod-principal-idealTr0*)
done

lemma (**in** *Ring*) *prod-principal-ideal*:$[\![a \in carrier\ R;\ b \in carrier\ R]\!] \Longrightarrow$
$(Rxa\ R\ a) \diamondsuit_r (Rxa\ R\ b) = Rxa\ R\ (a \cdot_r b)$

apply (*frule principal-ideal*[*of a*],
frule principal-ideal[*of b*])
apply (*subst sum-mult-is-ideal-prod*[*THEN sym*, *of Rxa R a Rxa R b*],
assumption+)
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*simp add*:*sum-mult-def*)
apply (*erule exE*, *erule bexE*)
apply (*frule sym*, *thin-tac* $\Sigma_e\ R\ f\ n = x$, *simp*, *thin-tac* $x = \Sigma_e\ R\ f\ n$)
apply (*simp add*:*prod-principal-idealTr1*)

apply (*rule subsetI*)
apply (*simp add*:*Rxa-def*, *fold Rxa-def*)
apply (*erule bexE*)
apply (*simp add*:*ring-tOp-assoc*[*THEN sym*])
apply (*frule ideal-subset1*[*of R* $\diamondsuit_p$ *a*],
frule ideal-subset1[*of R* $\diamondsuit_p$ *b*])
apply (*rule-tac a* = *r* $\cdot_r$ *a* **and** *b* = *b* **in** *times-mem-sum-mult*[*of R* $\diamondsuit_p$ *a*
R $\diamondsuit_p$ *b*], *assumption*+)
apply (*simp add*:*Rxa-def*, *blast*)
apply (*simp add*:*a-in-principal*)
done

lemma (**in** *Ring*) *principal-ideal-n-pow1*:$a \in$ *carrier R* $\Longrightarrow$
$(Rxa\ R\ a)^{\diamondsuit R\ n} = Rxa\ R\ (a\hat{}^{R\ n})$
apply (*cut-tac ring-one*)
apply (*induct-tac n*)
apply *simp*
apply (*cut-tac a-in-principal*[*of* 1_r])
apply (*frule principal-ideal*[*of* 1_r])
apply (*frule ideal-inc-one*, *assumption*, *simp*)
apply (*simp add*:*ring-one*)
apply *simp*
apply (*frule-tac n* = *n* **in** *npClose*[*of a*],
subst prod-principal-ideal, *assumption*+)
apply (*simp add*:*ring-tOp-commute*)
done

lemma (**in** *Ring*) *principal-ideal-n-pow*:$[\![a \in$ *carrier R*; $I = Rxa\ R\ a]\!] \Longrightarrow$
$I\ ^{\diamondsuit R\ n} = Rxa\ R\ (a\hat{}^{R\ n})$
apply *simp*
apply (*rule principal-ideal-n-pow1*[*of a n*], *assumption*+)
done

more about *ideal-n-prod*

lemma (**in** *Ring*) *nprod-eqTr*: $f \in \{j.\ j \leq (n::nat)\} \to$ *carrier R* $\wedge$
$g \in \{j.\ j \leq n\} \to$ *carrier R* $\wedge\ (\forall j \leq n.\ f\ j = g\ j) \longrightarrow$
nprod R f n = *nprod R g n*
apply (*induct-tac n*)

apply *simp*
apply (*rule impI*, (*erule conjE*)+)
apply (*frule func-pre*[*of f*], *frule func-pre*[*of g*],
cut-tac n = n **in** *Nsetn-sub-mem1*, *simp*)
done

lemma (**in** *Ring*) *nprod-eq*:⟦∀ *j* ≤ *n*. *f j* ∈ *carrier R*; ∀ *j* ≤ *n*. *g j* ∈ *carrier R*;
(∀ *j* ≤ (*n*::*nat*). *f j* = *g j*)⟧ ⟹ *nprod R f n* = *nprod R g n*
apply (*cut-tac nprod-eqTr*[*of f n g*])
apply *simp*
done

definition
mprod-expR :: [('*b*, '*m*) *Ring-scheme*, *nat* ⇒ *nat*, *nat* ⇒ '*b*, *nat*] ⇒ '*b* **where**
mprod-expR R e f n = *nprod R* (λ*j*. ((*f j*) ^*R* (*e j*))) *n*

lemma (**in** *Ring*) *mprodR-Suc*:⟦*e* ∈ {*j*. *j* ≤ (*Suc n*)} → {*j*. (*0*::*nat*) ≤ *j*};
f ∈ {*j*. *j* ≤ (*Suc n*)} → *carrier R*⟧ ⟹
mprod-expR R e f (*Suc n*) =
(*mprod-expR R e f n*) ·*r* ((*f* (*Suc n*)) ^*R* (*e* (*Suc n*)))
apply (*simp add*:*mprod-expR-def*)
done

lemma (**in** *Ring*) *mprod-expR-memTr*:*e* ∈ {*j*. *j* ≤ *n*} → {*j*. (*0*::*nat*) ≤ *j*} ∧
f ∈ {*j*. *j* ≤ *n*} → *carrier R* ⟶ *mprod-expR R e f n* ∈ *carrier R*
apply (*induct-tac n*)
apply (*rule impI*, (*erule conjE*)+)
apply (*cut-tac n-in-Nsetn*[*of 0*],
simp add: *mprod-expR-def*)
apply (*rule npClose*,
simp add:*Pi-def*)

apply (*rule impI*, (*erule conjE*)+)
apply (*frule func-pre*[*of e*], *frule func-pre*[*of f*])
apply *simp*
apply (*simp add*:*mprodR-Suc*)
apply (*rule ring-tOp-closed*, *assumption*+)
apply (*rule npClose*, *cut-tac n* = *Suc n* **in** *n-in-Nsetn*)
apply (*simp add*:*Pi-def*)
done

lemma (**in** *Ring*) *mprod-expR-mem*:⟦ *e* ∈ {*j*. *j* ≤ *n*} → {*j*. (*0*::*nat*) ≤ *j*};
f ∈ {*j*. *j* ≤ *n*} → *carrier R*⟧ ⟹ *mprod-expR R e f n* ∈ *carrier R*
apply (*simp add*:*mprod-expR-memTr*)
done

lemma (**in** *Ring*) *prod-n-principal-idealTr*:*e* ∈ {*j*. *j*≤*n*} → {*j*. (*0*::*nat*)≤*j*} ∧

$f \in \{j.\ j \leq n\} \to carrier\ R \wedge (\forall k \leq n.\ J\ k = (Rxa\ R\ (f\ k))^{\diamondsuit R\ (e\ k)}) \longrightarrow$
$ideal\text{-}n\text{-}prod\ R\ n\ J = Rxa\ R\ (mprod\text{-}expR\ R\ e\ f\ n)$
apply (*induct-tac n*)
apply (*rule impI*) **apply** (*erule conjE*)+
apply (*simp add*:*mprod-expR-def*)
apply (*subgoal-tac* $J\ 0 = R \diamondsuit_p\ (f\ 0)\ ^{\diamondsuit R\ (e\ 0)}$)
apply *simp*
apply (*rule principal-ideal-n-pow*[*of f 0 R* $\diamondsuit_p\ (f\ 0)$])
apply (*cut-tac n-in-Nsetn*[*of 0*], *simp add*:*Pi-def*) **apply** *simp*
apply (*cut-tac n-in-Nsetn*[*of 0*], *simp*)

apply (*rule impI*, (*erule conjE*)+)
apply (*frule func-pre*[*of e*], *frule func-pre*[*of f*])
apply (*cut-tac* $n = n$ **in** *Nsetn-sub-mem1*,
simp add:*mprodR-Suc*)
apply (*cut-tac* $n = Suc\ n$ **in** *n-in-Nsetn*, *simp*)
apply (*frule-tac* $A = \{j.\ j \leq Suc\ n\}$ **and** $x = Suc\ n$ **in** *funcset-mem*[*of f - carrier R*], *simp*)
apply (*frule-tac* $a = f\ (Suc\ n)$ **and** $I = R \diamondsuit_p\ (f\ (Suc\ n))$ **and** $n = e\ (Suc\ n)$ **in** *principal-ideal-n-pow*) **apply** *simp*
apply (*subst prod-principal-ideal*[*THEN sym*])
apply (*simp add*:*mprod-expR-mem*)
apply (*rule npClose*, *assumption*+) **apply** *simp*
done

lemma (**in** *Ring*) *prod-n-principal-ideal*:$[\![e \in \{j.\ j \leq n\} \to \{j.\ (0::nat) \leq j\};$
$f \in \{j.\ j \leq n\} \to carrier\ R; \forall k \leq n.\ J\ k = (Rxa\ R\ (f\ k))^{\diamondsuit R\ (e\ k)}]\!] \Longrightarrow$
$ideal\text{-}n\text{-}prod\ R\ n\ J = Rxa\ R\ (mprod\text{-}expR\ R\ e\ f\ n)$
apply (*simp add*:*prod-n-principal-idealTr*[*of e n f J*])
done

lemma (**in** *Idomain*) *a-notin-n-pow1*:$[\![a \in carrier\ R; \neg\ Unit\ R\ a;\ a \neq \mathbf{0};\ 0 < n]\!]$
$\Longrightarrow\ a \notin (Rxa\ R\ a)\ ^{\diamondsuit R\ \ (Suc\ n)}$
apply (*rule contrapos-pp*)
apply (*simp del*:*ipSuc*) **apply** (*simp del*:*ipSuc*)
apply (*frule principal-ideal*[*of a*])
apply (*frule principal-ideal-n-pow*[*of a R* $\diamondsuit_p$ *a Suc n*])
apply *simp* **apply** (*simp del*:*ipSuc*)
apply (*thin-tac* $R \diamondsuit_p\ a\ ^{\diamondsuit R\ (Suc\ n)} = R \diamondsuit_p\ (a\ \hat{}^{R\ n} \cdot_r\ a)$)
apply (*thin-tac ideal R* $(R \diamondsuit_p\ a)$)
apply (*simp add*:*Rxa-def*)
apply (*erule bexE*)
apply (*frule npClose*[*of a n*])
apply (*simp add*:*ring-tOp-assoc*[*THEN sym*])
apply (*frule ring-l-one*[*THEN sym*, *of a*])
apply (*subgoal-tac* $1_r \cdot_r\ a = r \cdot_r\ a\ \hat{}^{R\ n} \cdot_r\ a$)

apply (*cut-tac* $b = r \cdot_r (a \hat{}_R n)$ **in** *idom-mult-cancel-r*[*of* 1_r - a])
apply (*simp add*:*ring-one*) **apply** (*simp add*:*ring-tOp-closed*)
apply *assumption*+
apply (*thin-tac* $1_r \cdot_r a = r \cdot_r a \hat{}_R n \cdot_r a$,
thin-tac $a = 1_r \cdot_r a$,
thin-tac $a = r \cdot_r a \hat{}_R n \cdot_r a$)
apply (*subgoal-tac* $1_r = r \cdot_r (a \hat{}_R (Suc\ (n - Suc\ 0))))$ **prefer** *2*
apply (*simp del*:*ipSuc*)
apply (*thin-tac* $1_r = r \cdot_r a \hat{}_R n$)
apply (*simp del*:*Suc-pred*)
apply (*frule npClose*[*of* a $n - Suc\ 0$])
apply (*simp add*:*ring-tOp-assoc*[*THEN sym*])
apply (*frule-tac* $x = r$ **and** $y = a \hat{}_R (n - Suc\ 0)$ **in** *ring-tOp-closed*, *assumption*)
apply (*simp add*:*ring-tOp-commute*[*of* - a])
apply (*simp add*:*Unit-def*) **apply** *blast*
apply *simp*
done

lemma (**in** *Idomain*) *a-notin-n-pow2*:⟦$a \in carrier\ R$; $\neg\ Unit\ R\ a$; $a \neq \mathbf{0}$;
$0 < n$⟧ $\Longrightarrow a \hat{}_R n \notin (Rxa\ R\ a) \diamondsuit_R (Suc\ n)$
apply (*rule contrapos-pp*)
apply (*simp del*:*ipSuc*, *simp del*:*ipSuc*)
apply (*frule principal-ideal*[*of* a])
apply (*frule principal-ideal-n-pow*[*of* a R $\diamondsuit_p$ a *Suc* n])
apply (*simp*, *simp del*:*ipSuc*)
apply (*thin-tac* $R \diamondsuit_p a \diamondsuit_R (Suc\ n) = R \diamondsuit_p (a \hat{}_R n \cdot_r a)$)
apply (*thin-tac ideal* R $(R \diamondsuit_p a)$)
apply (*simp add*:*Rxa-def*)
apply (*erule bexE*)
apply (*frule idom-potent-nonzero*[*of* a n], *assumption*+)
apply (*frule npClose*[*of* a n])
apply (*frule ring-l-one*[*THEN sym*, *of* $a \hat{}_R n$])
apply (*subgoal-tac* $1_r \cdot_r (a \hat{}_R n) = r \cdot_r ((a \hat{}_R n) \cdot_r a)$)
prefer *2* **apply** *simp*
apply (*thin-tac* $a \hat{}_R n = 1_r \cdot_r a \hat{}_R n$,
thin-tac $a \hat{}_R n = r \cdot_r (a \hat{}_R n \cdot_r a)$)
apply (*simp add*:*ring-tOp-commute*[*of* $a \hat{}_R n$ a])
apply (*simp add*:*ring-tOp-assoc*[*THEN sym*])
apply (*cut-tac ring-one*,
frule-tac $b = r \cdot_r a$ **in** *idom-mult-cancel-r*[*of* 1_r - $a \hat{}_R n$],
simp add:*ring-tOp-closed*,
assumption+)
apply (*simp add*:*ring-tOp-commute*[*of* - a])
apply (*simp add*:*Unit-def*, *blast*)
done

lemma (**in** *Idomain*) *n-pow-not-prime*:⟦$a \in carrier\ R$; $a \neq \mathbf{0}$; $0 < n$⟧
$\Longrightarrow$ $\neg$ *prime-ideal* R $((Rxa\ R\ a) \diamondsuit_R (Suc\ n))$

apply (*case-tac n = 0*)
 apply *simp*
apply (*case-tac Unit R a*)
 apply (*simp del*:*ipSuc add*:*prime-ideal-def*, *rule impI*)
 apply (*frule principal-ideal*[*of a*])
 apply (*frule principal-ideal-n-pow*[*of a R* $\diamondsuit_p$ *a Suc n*])
 apply *simp* **apply** (*simp del*:*npow-suc*)
 apply (*simp del*:*npow-suc add*:*idom-potent-unit* [*of a Suc n*])
 apply (*thin-tac R* $\diamondsuit_p$ *a* $\diamondsuit_r$ *R* $\diamondsuit_p$ *a* $^{\diamondsuit R\ n} = R \diamondsuit_p (a^{\hat{}R\ (Suc\ n)})$)
 apply (*frule npClose*[*of a Suc n*])
 apply (*frule a-in-principal*[*of* $a^{\hat{}R\ (Suc\ n)}$])
 apply (*simp add*: *ideal-inc-unit*)
 apply (*frule a-notin-n-pow1*[*of a n*], *assumption*+)
 apply (*frule a-notin-n-pow2*[*of a n*], *assumption*+)

 apply (*frule npClose*[*of a n*])
 apply (*frule principal-ideal*[*of a*])
 apply (*frule principal-ideal-n-pow*[*of a R* $\diamondsuit_p$ *a Suc n*])
 apply *simp* **apply** (*simp del*:*ipSuc npow-suc*)
 apply (*thin-tac R* $\diamondsuit_p$ *a* $^{\diamondsuit R\ (Suc\ n)} = R \diamondsuit_p (a^{\hat{}R\ (Suc\ n)})$)
 apply (*subst prime-ideal-def*)
 apply (*simp del*:*npow-suc*) **apply** (*rule impI*)
 apply (*subgoal-tac* $(a^{\hat{}R\ n}) \cdot_r a \in R \diamondsuit_p (a^{\hat{}R\ (Suc\ n)})$)
 apply *blast*

 apply (*simp add*:*Rxa-def*)
  apply (*frule ring-tOp-closed*[*of a* $a^{\hat{}R\ n}$], *assumption*+)
 apply (*frule ring-l-one*[*THEN sym*, *of a* $\cdot_r$ $(a^{\hat{}R\ n})$])
 apply (*cut-tac ring-one*)
 apply (*simp add*:*ring-tOp-commute*[*of* - *a*], *blast*)
done

lemma (**in** *Idomain*) *principal-pow-prime-condTr*:
  $[\![a \in carrier\ R;\ a \neq \mathbf{0};\ prime\text{-}ideal\ R\ ((Rxa\ R\ a)\ ^{\diamondsuit R\ (Suc\ n)})]\!] \Longrightarrow n = 0$
apply (*rule contrapos-pp*, (*simp del*:*ipSuc*)+)
apply (*frule n-pow-not-prime*[*of a n*], *assumption*+)
apply (*simp del*:*ipSuc*)
done

lemma (**in** *Idomain*) *principal-pow-prime-cond*:
  $[\![a \in carrier\ R;\ a \neq \mathbf{0};\ \ prime\text{-}ideal\ R\ ((Rxa\ R\ a)\ ^{\diamondsuit R\ n})]\!] \Longrightarrow n = Suc\ 0$
apply (*case-tac n = 0*)
 apply *simp*
 apply (*simp add*:*prime-ideal-def*) **apply** (*erule conjE*)
 apply (*cut-tac ring-one*, *simp*)
apply (*subgoal-tac prime-ideal R* $(R \diamondsuit_p a\ ^{\diamondsuit R\ (Suc\ (n - Suc\ 0))})$)
apply (*frule principal-pow-prime-condTr*[*of a n* − *Suc 0*], *assumption*+)
apply *simp* **apply** *simp*

done

4.10 Extension and contraction

locale *TwoRings* = *Ring* +
  fixes R' (**structure**)
  assumes *secondR*: *Ring* R'

definition
  i-contract :: [$'a \Rightarrow 'b$, ($'a$, $'m1$) *Ring-scheme*, ($'b$, $'m2$) *Ring-scheme*,
    $'b$ *set*] $\Rightarrow$ $'a$ *set* **where**
  i-contract f R R' J = *invim f* (*carrier R*) *J*

definition
  i-extension :: [$'a \Rightarrow 'b$, ($'a$, $'m1$) *Ring-scheme*, ($'b$, $'m2$) *Ring-scheme*,
      $'a$ *set*] $\Rightarrow$ $'b$ *set* **where**
  i-extension f R R' I = *sum-mult R'* (*f* ' *I*) (*carrier R'*)

lemma (**in** *TwoRings*) *i-contract-sub*:$[\![f \in rHom\ R\ R'; ideal\ R'\ J\ ]\!] \Longrightarrow$
      (*i-contract f R R' J*) $\subseteq$ *carrier R*
apply (*simp add*:*i-contract-def invim-def*)
apply *blast*
done

lemma (**in** *TwoRings*) *i-contract-ideal*:$[\![f \in rHom\ R\ R'; ideal\ R'\ J\ ]\!] \Longrightarrow$
      ideal R (*i-contract f R R' J*)
 apply (*cut-tac Ring*,
    cut-tac secondR)
apply (*rule ideal-condition*)
apply (*simp add*:*i-contract-sub*)
apply (*simp add*:*i-contract-def invim-def*)
 apply (*cut-tac ring-zero*)
 apply (*cut-tac Ring*)
 apply (*frule rHom-0-0*[*of R R' f*], *assumption*+,
    cut-tac Ring.ideal-zero[*of R' J*])
 apply (*frule sym*, *thin-tac f* $\mathbf{0} = \mathbf{0}_{R'}$, *simp*, *blast*,
    assumption+)
apply (*rule ballI*)+
 apply (*simp add*:*i-contract-def invim-def*, (*erule conjE*)+)
 apply (*cut-tac ring-is-ag*,
    frule-tac x = *y* **in** *aGroup.ag-mOp-closed*[*of R*], *assumption*)
 apply (*simp add*:*aGroup.ag-pOp-closed*)
 apply (*simp add*:*rHom-add*)
 apply (*frule-tac x* = *y* **in** *rHom-inv-inv*[*of R R'* - *f*], *assumption*+, *simp*,
    thin-tac f ($-_a$ *y*) = $-_{a\,R'}$ (*f y*),
    frule-tac x = *f y* **in** *Ring.ideal-inv1-closed*[*of R' J*], *assumption*+,
    rule Ring.ideal-pOp-closed[*of R'*], *assumption*+)
 apply ((*rule ballI*)+,

simp add:*i-contract-def invim-def*, *erule conjE*,
simp add:*ring-tOp-closed*,
simp add:*rHom-tOp*)
apply (*frule-tac a* = *r* **in** *rHom-mem*[*of f R R'*], *assumption*,
simp add:*Ring.ideal-ring-multiple*[*of R' J*])
done

lemma (**in** *TwoRings*) *i-contract-mono*:⟦*f* ∈ *rHom R R'*; *ideal R' J1*; *ideal R' J2*;
J1 ⊆ *J2* ⟧ ⟹ *i-contract f R R' J1* ⊆ *i-contract f R R' J2*
apply (*rule subsetI*)
apply (*simp add*:*i-contract-def invim-def*) **apply** (*erule conjE*)
apply (*rule subsetD*, *assumption*+)
done

lemma (**in** *TwoRings*) *i-contract-prime*:⟦*f* ∈ *rHom R R'*; *prime-ideal R' P*⟧ ⟹
prime-ideal R (*i-contract f R R' P*)
apply (*cut-tac Ring*,
cut-tac secondR)
apply (*simp add*:*prime-ideal-def*, (*erule conjE*)+)
apply (*simp add*:*i-contract-ideal*)
apply (*rule conjI*)
apply (*rule contrapos-pp*, *simp*+)
apply (*simp add*:*i-contract-def invim-def*, *erule conjE*)
apply (*simp add*:*rHom-one*)
apply (*rule ballI*)+
apply (*frule-tac a* = *x* **in** *rHom-mem*[*of f R R'*], *assumption*+,
frule-tac a = *y* **in** *rHom-mem*[*of f R R'*], *assumption*+)
apply (*rule impI*)
apply (*simp add*:*i-contract-def invim-def*, *erule conjE*)
apply (*simp add*:*rHom-tOp*)
done

lemma (**in** *TwoRings*) *i-extension-ideal*:⟦*f* ∈ *rHom R R'*; *ideal R I* ⟧ ⟹
ideal R' (*i-extension f R R' I*)
apply (*cut-tac Ring*, *cut-tac secondR*)
apply (*simp add*:*i-extension-def*)
apply (*rule Ring.ideal-sum-mult* [*of R' f* ‘ *I carrier R'*], *assumption*+)
apply (*rule subsetI*)
apply (*simp add*:*image-def*)
apply (*erule bexE*, *frule-tac a* = *xa* **in** *rHom-mem*[*of f R R'*],
rule ideal-subset, *assumption*+, *simp*)
apply (*frule ideal-zero*, *simp*, *blast*)
apply (*simp add*:*Ring.whole-ideal*[*of R'*])
done

lemma (**in** *TwoRings*) *i-extension-mono*:⟦*f* ∈ *rHom R R'*; *ideal R I1*; *ideal R I2*;
I1 ⊆ *I2* ⟧ ⟹ (*i-extension f R R' I1*) ⊆ (*i-extension f R R' I2*)
apply (*rule subsetI*)
apply (*simp add*:*i-extension-def*)

apply (*simp add*:*sum-mult-def*)
apply (*erule exE*, *erule bexE*)
apply (*cut-tac Ring.set-mult-mono*[*of R′ f ‘ I1 f ‘ I2 carrier R′*])
apply (*frule-tac f = fa* **and** *A* = {*j*. *j* ≤ *n*} **in** *extend-fun*[*of - -
set-mult R′* (*f ‘ I1*) (*carrier R′*) *set-mult R′* (*f ‘ I2*) (*carrier R′*)],
assumption+) **apply** *blast*
apply (*simp add*:*secondR*)
apply (*simp add*:*image-def*, *rule subsetI*, *simp*, *erule bexE*,
frule-tac h = xb **in** *ideal-subset*[*of I1*], *assumption*, *simp add*:*rHom-mem*)
apply (*simp add*:*image-def*, *rule subsetI*, *simp*, *erule bexE*,
frule-tac h = xb **in** *ideal-subset*[*of I2*], *assumption*, *simp add*:*rHom-mem*)
apply (*rule subsetI*,
simp add:*image-def*, *erule bexE*,
frule-tac c = xb **in** *subsetD*[*of I1 I2*], *assumption*+, *blast*)
apply *simp*
done

lemma (**in** *TwoRings*) *e-c-inc-self*:⟦*f* ∈ *rHom R R′*; *ideal R I*⟧ ⟹
I ⊆ *i-contract f R R′* (*i-extension f R R′ I*)
apply (*rule subsetI*)
apply (*simp add*:*i-contract-def i-extension-def invim-def*)
apply (*simp add*:*ideal-subset*)
apply (*cut-tac secondR*,
frule Ring.ring-one [*of R′*])
apply (*frule-tac h = x* **in** *ideal-subset*[*of I*], *assumption*,
frule-tac f = f **and** *A = R* **and** *R = R′* **and** *a = x* **in** *rHom-mem*,
assumption)
apply (*frule-tac t = f x* **in** *Ring.ring-r-one*[*THEN sym*, *of R′*], *assumption*)
apply (*frule-tac a = f x* **and** *b* = $1_{r\,R'}$ **in** *Ring.times-mem-sum-mult*[*of R′
f ‘ I carrier R′*],
rule subsetI,
simp add:*image-def*, *erule bexE*,
frule-tac h = xb **in** *ideal-subset*[*of I*], *assumption*,
simp add:*rHom-mem*, *simp*,
simp add:*image-def*, *blast*, *assumption*+)
apply *simp*
done

lemma (**in** *TwoRings*) *c-e-incd-self*:⟦*f* ∈ *rHom R R′*; *ideal R′ J* ⟧ ⟹
i-extension f R R′ (*i-contract f R R′ J*) ⊆ *J*
apply (*rule subsetI*)
apply (*simp add*:*i-extension-def*)
apply (*simp add*:*sum-mult-def*)
apply (*erule exE*, *erule bexE*)
apply (*cut-tac secondR*,
frule-tac n = n **and** *f = fa* **in** *Ring.ideal-nsum-closed*[*of R′ J*],
assumption)
apply (*rule allI*, *rule impI*) **apply** (
frule-tac f = fa **and** *A* = {*j*. *j* ≤ *n*} **and**

```
      B = set-mult R' (f ‘ i-contract f R R' J) (carrier R') and x = j in
      funcset-mem, simp) apply (
  thin-tac fa ∈ {j. j ≤ n} → set-mult R' (f ‘ i-contract f R R' J) (carrier R'))
  apply (simp add:set-mult-def, (erule bexE)+,
       simp add:i-contract-def invim-def, erule conjE)
  apply (frule-tac x = f xa and r = y in Ring.ideal-ring-multiple1[of R' J],
       assumption+, simp)

  apply simp
done

lemma (in TwoRings) c-e-c-eq-c:⟦f ∈ rHom R R'; ideal R' J ⟧ ⟹
  i-contract f R R' (i-extension f R R' (i-contract f R R' J))
                                    = i-contract f R R' J
apply (frule i-contract-ideal [of f J], assumption)
apply (frule e-c-inc-self [of f i-contract f R R' J], assumption+)
apply (frule c-e-incd-self [of f J], assumption+)
apply (frule i-contract-mono [of f
       i-extension f R R' (i-contract f R R' J) J])
apply (rule i-extension-ideal, assumption+)
apply (rule equalityI, assumption+)
done

lemma (in TwoRings) e-c-e-eq-e:⟦f ∈ rHom R R'; ideal R I ⟧ ⟹
  i-extension f R R' (i-contract f R R' (i-extension f R R' I))
                                    = i-extension f R R' I
apply (frule i-extension-ideal [of f I], assumption+)
apply (frule c-e-incd-self [of f i-extension f R R' I], assumption+)
apply (rule equalityI, assumption+)
 apply (thin-tac i-extension f R R' (i-contract f R R' (i-extension f R R' I))
      ⊆ i-extension f R R' I)
apply (frule e-c-inc-self [of f I], assumption+)
apply (rule i-extension-mono [of f I
            i-contract f R R' (i-extension f R R' I)], assumption+)
apply (rule i-contract-ideal, assumption+)
done
```

4.11 Complete system of representatives

```
definition
  csrp-fn :: [-, 'a set] ⇒ 'a set ⇒ 'a where
  csrp-fn R I = (λx∈carrier (R /r I). (if x = I then 0R else SOME y. y ∈ x))

definition
  csrp :: [- , 'a set] ⇒ 'a set where
  csrp R I == (csrp-fn R I) ‘ (carrier (R /r I))
```

lemma (**in** *Ring*) *csrp-mem*:⟦*ideal R I*; *a* ∈ *carrier R*⟧ ⟹
csrp-fn R I (*a* ⊎$_R$ *I*) ∈ *a* ⊎$_R$ *I*
apply (*simp add*:*csrp-fn-def qring-carrier*)
apply (*case-tac a* ⊎$_R$ *I* = *I*) **apply** *simp*
apply (*rule conjI*, *rule impI*)
apply (*simp add*:*ideal-zero*)
apply (*rule impI*)
apply (*cut-tac ring-zero*)
apply (*frule-tac x* = **0** **in** *bspec*, *assumption*+)
apply (*thin-tac* ∀ *a*∈*carrier R*. *a* ⊎$_R$ *I* ≠ *I*)
apply (*frule ideal-zero*[*of I*])
apply (*frule ar-coset-same4*[*of I* **0**], *assumption*+, *simp*)
apply *simp*
apply (*rule conjI*)
apply (*rule impI*, *rule someI2-ex*)
apply (*frule a-in-ar-coset*[*of I a*], *assumption*+, *blast*, *assumption*+)
apply (*rule impI*)
apply (*frule-tac x* = *a* **in** *bspec*, *assumption*+,
thin-tac ∀ *aa*∈*carrier R*. *aa* ⊎$_R$ *I* ≠ *a* ⊎$_R$ *I*, *simp*)
done

lemma (**in** *Ring*) *csrp-same*:⟦*ideal R I*; *a* ∈ *carrier R*⟧ ⟹
csrp-fn R I (*a* ⊎$_R$ *I*) ⊎$_R$ *I* = *a* ⊎$_R$ *I*
apply (*frule csrp-mem*[*of I a*], *assumption*+)
apply (*rule ar-cos-same*[*of a I csrp-fn R I* (*a* ⊎$_R$ *I*)], *assumption*+)
done

lemma (**in** *Ring*) *csrp-mem1*:⟦*ideal R I*; *x* ∈ *carrier* (*R* /$_r$ *I*)⟧ ⟹
csrp-fn R I x ∈ *x*
apply (*simp add*:*qring-carrier*, *erule bexE*, *frule sym*,
thin-tac a ⊎$_R$ *I* = *x*, *simp*)
apply (*simp add*:*csrp-mem*)
done

lemma (**in** *Ring*) *csrp-fn-mem*:⟦*ideal R I*; *x* ∈ *carrier* (*R* /$_r$ *I*)⟧ ⟹
(*csrp-fn R I x*) ∈ *carrier R*
apply (*simp add*:*qring-carrier*, *erule bexE*, *frule sym*,
thin-tac a ⊎$_R$ *I* = *x*, *simp*,
frule-tac a = *a* **in** *csrp-mem*[*of I*], *assumption*+)
apply (*rule-tac a* = *a* **and** *x* = *csrp-fn R I* (*a* ⊎$_R$ *I*) **in**
ar-coset-subsetD[*of I*], *assumption*+)
done

lemma (**in** *Ring*) *csrp-eq-coset*:⟦*ideal R I*; *x* ∈ *carrier* (*R* /$_r$ *I*)⟧ ⟹
(*csrp-fn R I x*) ⊎$_R$ *I* = *x*
apply (*simp add*:*qring-carrier*, *erule bexE*)
apply (*frule sym*, *thin-tac a* ⊎$_R$ *I* = *x*, *simp*)
apply (*frule-tac a* = *a* **in** *csrp-mem*[*of I*], *assumption*+)
apply (*rule ar-cos-same*, *assumption*+)

done

lemma (**in** *Ring*) *csrp-nz-nz*:⟦*ideal R I*; *x* ∈ *carrier* (*R* /$_r$ *I*);
x ≠ **0**$_{(R\ /_r\ I)}$⟧ ⟹ (*csrp-fn R I x*) ≠ **0**
apply (*rule contrapos-pp*, *simp*+)
apply (*frule csrp-eq-coset*[*of I x*], *assumption*+, *simp*)
apply (*simp add*:*qring-zero*[*of I*])
apply (*frule ideal-zero*[*of I*]) **apply** (
cut-tac ring-zero)
apply (*simp add*:*Qring-fix1* [*of* **0** *I*])
done

lemma (**in** *Ring*) *csrp-diff-in-vpr*:⟦*ideal R I*; *x* ∈ *carrier R*⟧ ⟹
x ± (−$_a$ (*csrp-fn R I* (*pj R I x*))) ∈ *I*
apply (*frule csrp-mem*[*of I x*],
frule csrp-same[*of I x*],
simp add:*pj-mem*, *assumption*,
frule ar-coset-subsetD[*of I x csrp-fn R I* (*x* ⊎$_R$ *I*)],
assumption+)
apply (*frule belong-ar-coset2*[*of I x csrp-fn R I* (*x* ⊎$_R$ *I*)], *assumption*+,
frule ideal-inv1-closed[*of I csrp-fn R I* (*x* ⊎$_R$ *I*) ± −$_a$ *x*], *assumption*+,
cut-tac ring-is-ag,
frule aGroup.ag-mOp-closed[*of R x*], *assumption*,
simp add:*aGroup.ag-pOp-commute*[*of R csrp-fn R I* (*x* ⊎$_R$ *I*) −$_a$ *x*])
apply (*simp add*:*aGroup.ag-p-inv*[*of R* −$_a$ *x csrp-fn R I* (*x* ⊎$_R$ *I*)],
simp add:*aGroup.ag-inv-inv*,
cut-tac Ring, *simp add*:*pj-mem*[*of R I x*])
done

lemma (**in** *Ring*) *csrp-pj*:⟦*ideal R I*; *x* ∈ *carrier* (*R* /$_r$ *I*)⟧ ⟹
(*pj R I*) (*csrp-fn R I x*) = *x*
apply(*cut-tac Ring*,
frule csrp-fn-mem[*of I x*], *assumption*+,
simp add:*pj-mem*[*of R I csrp-fn R I x*],
simp add:*csrp-eq-coset*)
done

4.12 Polynomial ring

In this section, we treat a ring of polynomials over a ring S. Numbers are of type ant

definition
pol-coeff :: [('*a*, '*more*) *Ring-scheme*, (*nat* × (*nat* ⇒ '*a*))] ⇒ *bool* **where**
pol-coeff S c ⟷ (∀ *j* ≤ (*fst c*). (*snd c*) *j* ∈ *carrier S*)

definition
c-max :: [('*a*, '*more*) *Ring-scheme*, *nat* × (*nat* ⇒ '*a*)] ⇒ *nat* **where**

c-max S c = (*if* {*j*. *j* ≤ (*fst c*) ∧ (*snd c*) *j* ≠ $\mathbf{0}_S$} = {} *then 0 else*
n-max {*j*. *j* ≤ (*fst c*) ∧ (*snd c*) *j* ≠ $\mathbf{0}_S$})

definition
polyn-expr :: [('*a*, '*more*) *Ring-scheme*, '*a*, *nat*, *nat* × (*nat* ⇒ '*a*)] ⇒ '*a* **where**
polyn-expr R X k c == *nsum R* (λ*j*. ((*snd c*) *j*) ·$_{rR}$ (*X* ^$_R$ *j*)) *k*

definition
algfree-cond :: [('*a*, '*m*) *Ring-scheme*, ('*a*, '*m1*) *Ring-scheme*,
'*a*] ⇒ *bool* **where**
algfree-cond R S X ⟷ (∀ *c*. *pol-coeff S c* ∧ (∀ *k* ≤ (*fst c*).
(*nsum R* (λ*j*. ((*snd c*) *j*) ·$_{rR}$ (*X* ^$_R$ *j*)) *k* = $\mathbf{0}_R$ ⟶
(∀ *j* ≤ *k*. (*snd c*) *j* = $\mathbf{0}_S$))))

locale *PolynRg* = *Ring* +
fixes *S* (**structure**)
fixes *X* (**structure**)
assumes *X-mem-R*:*X* ∈ *carrier R*
and *not-zeroring*:¬ *Zero-ring S*
and *subring*: *Subring R S*
and *algfree*: *algfree-cond R S X*
and *S-X-generate*:*x* ∈ *carrier R* ⟹
∃ *f*. *pol-coeff S f* ∧ *x* = *polyn-expr R X* (*fst f*) *f*

4.13 Addition and multiplication of *polyn-exprs*

4.13.1 Simple properties of a *polyn-ring*

lemma *Subring-subset*:*Subring R S* ⟹ *carrier S* ⊆ *carrier R*
by (*simp add*:*Subring-def*)

lemma (**in** *Ring*) *subring-Ring*:*Subring R S* ⟹ *Ring S*
by (*simp add*:*Subring-def*)

lemma (**in** *Ring*) *mem-subring-mem-ring*:⟦*Subring R S*; *x* ∈ *carrier S*⟧ ⟹
x ∈ *carrier R*
by (*simp add*:*Subring-def*, (*erule conjE*)+, *simp add*: *subsetD*)

lemma (**in** *Ring*) *Subring-pOp-ring-pOp*:⟦*Subring R S*; *a* ∈ *carrier S*;
b ∈ *carrier S* ⟧ ⟹ *a* ±$_S$ *b* = *a* ± *b*
apply (*simp add*:*Subring-def*, (*erule conjE*)+)
apply (*frule rHom-add*[*of ridmap S S R a b*], *assumption*+)
apply (*cut-tac Ring.ring-is-ag*[*of S*],
frule aGroup.ag-pOp-closed[*of S a b*], *assumption*+,
simp add:*ridmap-def*, *assumption*)
done

lemma (**in** *Ring*) *Subring-tOp-ring-tOp*:⟦*Subring R S*; *a* ∈ *carrier S*;
b ∈ *carrier S* ⟧ ⟹ *a* ·$_{rS}$ *b* = *a* ·$_r$ *b*

apply (*simp add*:*Subring-def*, (*erule conjE*)+)
apply (*frule rHom-tOp*[*of S R a b ridmap S*], *rule Ring-axioms*, *assumption*+)
apply (*frule Ring.ring-tOp-closed*[*of S a b*], *assumption*+,
    simp add:*ridmap-def*)
done

lemma (**in** *Ring*) *Subring-one-ring-one*:*Subring R S* $\Longrightarrow$ $1_{rS} = 1_r$
apply (*simp add*:*Subring-def*, (*erule conjE*)+)
apply (*frule rHom-one*[*of S R ridmap S*], *rule Ring-axioms*, *assumption*+)
apply (*simp add*:*ridmap-def*, *simp add*:*Ring.ring-one*[*of S*])
done

lemma (**in** *Ring*) *Subring-zero-ring-zero*:*Subring R S* $\Longrightarrow$ $\mathbf{0}_S = \mathbf{0}$
apply (*simp add*:*Subring-def*, (*erule conjE*)+,
    frule rHom-0-0[*of S R ridmap S*], *rule Ring-axioms*, *assumption*+,
    simp add:*ridmap-def*, *simp add*:*Ring.ring-zero*[*of S*])
done

lemma (**in** *Ring*) *Subring-minus-ring-minus*:⟦*Subring R S*; $x \in$ *carrier S*⟧
    $\Longrightarrow$ $-_{aS}\ x = -_a\ x$
apply (*simp add*:*Subring-def*, (*erule conjE*)+, *simp add*:*rHom-def*, (*erule conjE*)+)
apply (*cut-tac ring-is-ag*, *frule Ring.ring-is-ag*[*of S*])
apply (*frule aHom-inv-inv*[*of S R ridmap S x*], *assumption*+,
    frule aGroup.ag-mOp-closed[*of S x*], *assumption*+)
apply (*simp add*:*ridmap-def*)
done

lemma (**in** *PolynRg*) *Subring-pow-ring-pow*:$x \in$ *carrier S* $\Longrightarrow$
            $x\hat{}^{S}\ n = x\hat{}^{R}\ n$
apply (*cut-tac subring*, *frule subring-Ring*)
apply (*induct-tac n*)
apply (*simp*, *simp add*:*Subring-one-ring-one*)
apply (*frule-tac n* = *n* **in** *Ring.npClose*[*of S x*], *assumption*+)
apply (*simp add*:*Subring-tOp-ring-tOp*)
done

lemma (**in** *PolynRg*) *is-Ring*: *Ring R* **..**

lemma (**in** *PolynRg*) *polyn-ring-nonzero*:$1_r \neq \mathbf{0}$
apply (*cut-tac Ring*, *cut-tac subring*)
apply (*simp add*:*Subring-zero-ring-zero*[*THEN sym*])
apply (*simp add*:*Subring-one-ring-one*[*THEN sym*])
apply (*simp add*:*not-zeroring*)
done

lemma (**in** *PolynRg*) *polyn-ring-S-nonzero*:$1_{rS} \neq \mathbf{0}_S$
apply (*cut-tac subring*)
apply (*simp add*:*Subring-zero-ring-zero*)
apply (*simp add*:*Subring-one-ring-one*)

apply (*simp add*:*polyn-ring-nonzero*)
done

lemma (**in** *PolynRg*) *polyn-ring-X-nonzero*:$X \neq \mathbf{0}$
apply (*cut-tac algfree*,
cut-tac subring)
apply (*simp add*:*algfree-cond-def*)
apply (*rule contrapos-pp*, *simp*+)
apply (*drule-tac x* = *Suc 0* **in** *spec*)
apply (*subgoal-tac pol-coeff S* ((*Suc 0*),
($\lambda j{\in}\{l.\ l \leq (Suc\ 0)\}$. *if j* = *0 then* $\mathbf{0}_S$ *else* $1_{r\,S}$)))
apply (*drule-tac x* = $\lambda j{\in}\{l.\ l \leq (Suc\ 0)\}$. *if j* = *0 then* $\mathbf{0}_S$ *else* $1_{r\,S}$ **in**
spec)
apply (*erule conjE*, *simp*)
apply (*simp only*:*Nset-1*)
apply (*drule-tac a* = *Suc 0* **in** *forall-spec*, *simp*)
apply *simp*
apply (*cut-tac subring*, *simp add*:*Subring-zero-ring-zero*,
simp add:*Subring-one-ring-one*, *cut-tac ring-zero*, *cut-tac ring-one*,
simp add:*ring-r-one*, *simp add*:*ring-times-x-0*, *cut-tac ring-is-ag*,
simp add:*aGroup.ag-r-zero*,
drule-tac a = *Suc 0* **in** *forall-spec*, *simp*, *simp*)
apply (*cut-tac polyn-ring-S-nonzero*, *simp add*:*Subring-zero-ring-zero*)

apply (*thin-tac* $\forall\, b.$ *pol-coeff S* (*Suc 0*, *b*) $\wedge$
$(\forall\, k{\leq}Suc\ 0.\ \Sigma_e\ R\ (\lambda j.\ b\ j \cdot_r \mathbf{0}\,\hat{}^R\, j)\ k = \mathbf{0} \longrightarrow (\forall\, j{\leq}k.\ b\ j = \mathbf{0}_S))$,
simp add:*pol-coeff-def*,
rule allI,
simp add:*Subring-def*, *simp add*:*Ring.ring-zero*,
(*rule impI*)+,
simp add:*Ring.ring-one*)
done

4.13.2 Coefficients of a polynomial

lemma (**in** *PolynRg*) *pol-coeff-split*:*pol-coeff S f* = *pol-coeff S* (*fst f*, *snd f*)
by *simp*

lemma (**in** *PolynRg*) *pol-coeff-cartesian*:*pol-coeff S c* $\Longrightarrow$
(*fst c*, *snd c*) = *c*
by *simp*

lemma (**in** *PolynRg*) *split-pol-coeff*:$[\![$*pol-coeff S c*; *k* $\leq$ (*fst c*)$]\!] \Longrightarrow$
pol-coeff S (*k*, *snd c*)
by (*simp add*:*pol-coeff-def*)

lemma (**in** *PolynRg*) *pol-coeff-pre*:*pol-coeff S* ((*Suc n*), *f*) $\Longrightarrow$
pol-coeff S (*n*, *f*)
apply (*simp add*:*pol-coeff-def*)

done

lemma (**in** *PolynRg*) *pol-coeff-le*:⟦*pol-coeff S c*; $n \leq$ (*fst c*)⟧ ⟹
pol-coeff S (n, (*snd c*))
apply (*simp add*:*pol-coeff-def*)
done

lemma (**in** *PolynRg*) *pol-coeff-mem*:⟦*pol-coeff S c*; $j \leq$ (*fst c*)⟧ ⟹
((*snd c*) j) ∈ *carrier S*
by (*simp add*:*pol-coeff-def*)

lemma (**in** *PolynRg*) *pol-coeff-mem-R*:⟦*pol-coeff S c*; $j \leq$ (*fst c*)⟧
⟹ ((*snd c*) j) ∈ *carrier R*
apply (*cut-tac subring*, *frule subring-Ring*)
apply (*frule pol-coeff-mem*[*of c j*], *assumption*+,
simp add:*mem-subring-mem-ring*)
done

lemma (**in** *PolynRg*) *Slide-pol-coeff*:⟦*pol-coeff S c*; $n <$ (*fst c*)⟧ ⟹
pol-coeff S (((*fst c*) − *Suc n*), (λ*x*. (*snd c*) (*Suc* (n + x))))
apply (*simp add*: *pol-coeff-def*)
done

4.13.3 Addition of *polyn-exprs*

lemma (**in** *PolynRg*) *monomial-mem*:*pol-coeff S c* ⟹
∀ $j \leq$ (*fst c*). (*snd c*) j ·$_r$ X ^$_R$ j ∈ *carrier R*
apply (*rule allI*, *rule impI*)
apply (*rule ring-tOp-closed*)
apply (*simp add*:*pol-coeff-mem-R*[*of c*],
cut-tac X-mem-R, *simp add*:*npClose*)
done

lemma (**in** *PolynRg*) *polyn-mem*:⟦*pol-coeff S c*; $k \leq$ (*fst c*)⟧ ⟹
polyn-expr R X k c ∈ *carrier R*
apply (*simp add*:*polyn-expr-def*,
cut-tac ring-is-ag)
apply (*rule aGroup.nsum-mem*[*of R k* λ*j*. (*snd c*) j ·$_r$ X ^$_R$ j], *assumption*+)
apply (*simp add*:*monomial-mem*)
done

lemma (**in** *PolynRg*) *polyn-exprs-eq*:⟦*pol-coeff S c*; *pol-coeff S d*;
$k \leq$ (*min* (*fst c*) (*fst d*)); ∀ $j \leq k$. (*snd c*) j = (*snd d*) j⟧ ⟹
polyn-expr R X k c = *polyn-expr R X k d*
apply (*cut-tac ring-is-ag*,
simp add:*polyn-expr-def*,
cut-tac subring,
cut-tac X-mem-R)
apply (*rule aGroup.nsum-eq*[*of R k* λ*j*. (*snd c*) j ·$_r$ X ^$_R$ j

$\lambda j.\ (snd\ d)\ j\ \cdot_r\ X\hat{}^R\ j]$, *assumption*)
apply (*simp add*:*monomial-mem*)+
done

lemma (**in** *PolynRg*) *polyn-expr-restrict*:*pol-coeff S* (*Suc n*, *f*) $\Longrightarrow$
polyn-expr R X n (*Suc n*, *f*) = *polyn-expr R X n* (*n*, *f*)
apply (*cut-tac subring*, *frule subring-Ring*,
cut-tac pol-coeff-le[*of* (*Suc n*, *f*) *n*])
apply (*cut-tac polyn-exprs-eq*[*of* (*Suc n*, *f*) (*n*, *f*) *n*],
(*simp add*:*pol-coeff-split*[*THEN sym*])+)
done

lemma (**in** *PolynRg*) *polyn-expr-short*:⟦*pol-coeff S c*; *k* ≤ (*fst c*)⟧ $\Longrightarrow$
polyn-expr R X k c = *polyn-expr R X k* (*k*, *snd c*)
apply (*rule polyn-exprs-eq*[*of c* (*k*, *snd c*) *k*], *assumption*+)
apply (*simp add*:*pol-coeff-def*)
apply (*simp*)
apply *simp*
done

lemma (**in** *PolynRg*) *polyn-expr0*:*pol-coeff S c* $\Longrightarrow$
polyn-expr R X 0 c = (*snd c*) *0*
apply (*simp add*:*polyn-expr-def*)
apply (*cut-tac subring*,
cut-tac subring-Ring[*of S*])
apply (*frule pol-coeff-mem*[*of c 0*], *simp*)
apply (*frule mem-subring-mem-ring* [*of S* (*snd c*) *0*], *assumption*)
apply (*simp add*:*ring-r-one*, *assumption*)
done

lemma (**in** *PolynRg*) *polyn-expr-split*:
polyn-expr R X k f = *polyn-expr R X k* (*fst f*, *snd f*)
by *simp*

lemma (**in** *PolynRg*) *polyn-Suc*:*Suc n* ≤ (*fst c*) $\Longrightarrow$
polyn-expr R X (*Suc n*) ((*Suc n*), (*snd c*)) =
polyn-expr R X n c ± ((*snd c*) (*Suc n*)) $\cdot_r$ ($X\hat{}^R$ (*Suc n*))
by (*simp add*:*polyn-expr-def*)

lemma (**in** *PolynRg*) *polyn-Suc-split*:*pol-coeff S* (*Suc n*, *f*) $\Longrightarrow$
polyn-expr R X (*Suc n*) ((*Suc n*), *f*) =
polyn-expr R X n (*n*, *f*) ± (*f* (*Suc n*)) $\cdot_r$ ($X\hat{}^R$ (*Suc n*))
apply (*cut-tac polyn-Suc*[*of n* (*Suc n*, *f*)])
apply (*simp del*:*npow-suc*)
apply (*subst polyn-expr-short*[*of* (*Suc n*, *f*) *n*], *assumption*+, *simp*)
apply (*simp del*:*npow-suc*)
apply *simp*
done

lemma (**in** *PolynRg*) *polyn-n-m*:⟦*pol-coeff S c*; $n < m$; $m \leq (fst\ c)$⟧ ⟹
$polyn\text{-}expr\ R\ X\ m\ (m, (snd\ c)) = polyn\text{-}expr\ R\ X\ n\ (n, (snd\ c)) \pm$
$(fSum\ R\ (\lambda j.\ ((snd\ c)\ j) \cdot_r (X\hat{}^R\ j))\ (Suc\ n)\ m)$
apply (*simp add*:*polyn-expr-def*, *cut-tac ring-is-ag*)
apply (*rule aGroup.nsum-split1*[*of R m* $\lambda j.\ ((snd\ c)\ j) \cdot_r (X\hat{}^R\ j)\ n$],
assumption+)
apply (*rule allI*, *rule impI*)
apply (*frule-tac monomial-mem*[*of c*],
frule-tac $i = j$ **and** $j = m$ **and** $k = (fst\ c)$ **in** *le-trans*, *assumption*+,
simp+)
done

lemma (**in** *PolynRg*) *polyn-n-m1*:⟦*pol-coeff S c*; $n < m$; $m \leq (fst\ c)$⟧ ⟹
$polyn\text{-}expr\ R\ X\ m\ c = polyn\text{-}expr\ R\ X\ n\ c \pm$
$(fSum\ R\ (\lambda j.\ ((snd\ c)\ j) \cdot_r (X\hat{}^R\ j))\ (Suc\ n)\ m)$
apply (*subst polyn-expr-short*[*of c n*], *assumption*)
apply (*frule-tac* $x = n$ **and** $y = m$ **and** $z = fst\ c$ **in** *less-le-trans*, *assumption*,
simp add:*less-imp-le*)
apply (*subst polyn-expr-short*[*of c m*], *assumption*+)
apply (*simp add*:*polyn-n-m*)
done

lemma (**in** *PolynRg*) *polyn-n-m-mem*:⟦*pol-coeff S c*; $n < m$; $m \leq (fst\ c)$⟧ ⟹
$(fSum\ R\ (\lambda j.\ ((snd\ c)\ j) \cdot_r (X\hat{}^R\ j))\ (Suc\ n)\ m) \in carrier\ R$
apply (*simp add*:*fSum-def*)
apply (*cut-tac ring-is-ag*,
rule-tac $n = m - Suc\ n$ **in** *aGroup.nsum-mem*, *assumption*+)
apply (*rule allI*, *rule impI*,
simp del:*npow-suc add*:*cmp-def slide-def*)
apply (*rule ring-tOp-closed*)
apply (*simp add*:*pol-coeff-def*)
apply (*frule-tac* $a = Suc\ (n + j)$ **in** *forall-spec*, *arith*)
apply (*cut-tac subring*)
apply (*simp add*:*mem-subring-mem-ring*)
apply (*rule npClose*)
apply (*cut-tac X-mem-R*,
simp del:*npow-suc add*:*npClose*)
done

lemma (**in** *PolynRg*) *polyn-n-ms-eq*:⟦*pol-coeff S c*; *pol-coeff S d*;
$m \leq min\ (fst\ c)\ (fst\ d)$; $n < m$;
$\forall j \in nset\ (Suc\ n)\ m.\ (snd\ c)\ j = (snd\ d)\ j$⟧ ⟹
$(fSum\ R\ (\lambda j.\ ((snd\ c)\ j) \cdot_r (X\hat{}^R\ j))\ (Suc\ n)\ m) =$
$(fSum\ R\ (\lambda j.\ ((snd\ d)\ j) \cdot_r (X\hat{}^R\ j))\ (Suc\ n)\ m)$
apply (*cut-tac ring-is-ag*)
apply (*cut-tac aGroup.fSum-eq1*[*of R Suc n m* $\lambda j.\ (snd\ c)\ j \cdot_r X\hat{}^R\ j$
$\lambda j.\ (snd\ d)\ j \cdot_r X\hat{}^R\ j$],
assumption+)

```
  apply (rule Suc-leI, assumption,
        simp add:nset-def, simp add:monomial-mem)
  apply (frule Suc-leI,
           rule ballI, simp add:nset-def)

  apply (simp add:monomial-mem)
 apply simp
done


lemma (in PolynRg) polyn-addTr:
 (pol-coeff S (n, f)) ∧ (pol-coeff S (n, g)) ⟶
    (polyn-expr R X n (n, f)) ± (polyn-expr R X n (n, g)) =
                nsum R (λj. ((f j) ±S (g j)) ·r (X ^R j)) n
apply (cut-tac subring,
       frule subring-Ring[of S])
apply (induct-tac n)
 apply (rule impI, simp, erule conjE)
 apply (simp add:polyn-expr0)
 apply (cut-tac pol-coeff-mem[of (0, f) 0], simp,
        cut-tac pol-coeff-mem[of (0, g) 0], simp,
       frule  mem-subring-mem-ring[of S f 0], assumption+,
       frule  mem-subring-mem-ring[of S g 0], assumption+,
       frule Ring.ring-is-ag[of S],
       frule aGroup.ag-pOp-closed[of S f 0 g 0], assumption+,
       frule mem-subring-mem-ring[of S f 0 ±S g 0], assumption+)
apply (simp add:ring-r-one)
 apply (simp add:Subring-pOp-ring-pOp[of S f 0 g 0])
 apply (simp del:npow-suc)+
apply (rule impI, erule conjE)
 apply (frule-tac n = n in  pol-coeff-pre[of - f],
        frule-tac n = n in  pol-coeff-pre[of - g], simp del:npow-suc)
 apply (cut-tac n = n and c = (Suc n, f) in polyn-Suc, simp del:npow-suc,
        simp del:npow-suc,
        thin-tac polyn-expr R X (Suc n) (Suc n, f) =
         polyn-expr R X n (Suc n, f) ± f (Suc n) ·r X ^R (Suc n))
 apply (cut-tac n = n and c = (Suc n, g) in polyn-Suc, simp del:npow-suc,
        simp del:npow-suc,
        thin-tac polyn-expr R X (Suc n) (Suc n, g) =
         polyn-expr R X n (Suc n, g) ± g (Suc n) ·r X ^R (Suc n))
 apply (cut-tac c = (Suc n, f) and k = n in polyn-mem, assumption,
              simp del:npow-suc,
        cut-tac k = n and c = (Suc n, g) in polyn-mem, assumption,
              simp del:npow-suc)
 apply (frule-tac j = Suc n and c = (Suc n, f) in pol-coeff-mem-R, simp,
        frule-tac j = Suc n and c = (Suc n, g) in pol-coeff-mem-R, simp,
        cut-tac  X-mem-R,
        frule-tac n = Suc n in npClose[of X], simp del:npow-suc)
 apply (frule-tac x = f (Suc n) and y = X ^R (Suc n) in ring-tOp-closed,
```

assumption+,
frule-tac $x = g\ (Suc\ n)$ **and** $y = X\hat{}^R\ (Suc\ n)$ **in** *ring-tOp-closed*,
assumption+)
apply (*cut-tac ring-is-ag*,
subst aGroup.pOp-assocTr43, *assumption*+)
apply (*frule-tac* $x = f\ (Suc\ n) \cdot_r X\hat{}^R\ (Suc\ n)$ **and**
$y =$ *polyn-expr R X n* $(Suc\ n, g)$ **in** *aGroup.ag-pOp-commute*[*of R*],
assumption+, *simp del*:*npow-suc*,
thin-tac $f\ (Suc\ n) \cdot_r X\hat{}^R\ (Suc\ n) \pm$ *polyn-expr R X n* $(Suc\ n, g) =$
polyn-expr R X n $(Suc\ n, g) \pm f\ (Suc\ n) \cdot_r X\hat{}^R\ (Suc\ n)$)
apply (*subst aGroup.pOp-assocTr43*[*THEN sym*], *assumption*+,
simp del:*npow-suc add*:*polyn-expr-restrict*)

apply (*frule-tac* $c = (Suc\ n, f)$ **and** $j = Suc\ n$ **in** *pol-coeff-mem*, *simp*,
frule-tac $c = (Suc\ n, g)$ **and** $j = Suc\ n$ **in** *pol-coeff-mem*, *simp*)
apply (*subst ring-distrib2*[*THEN sym*], *assumption*+)
apply (*frule-tac* $c = (Suc\ n, f)$ **and** $j = Suc\ n$ **in** *pol-coeff-mem*, *simp*,
frule-tac $c = (Suc\ n, g)$ **and** $j = Suc\ n$ **in** *pol-coeff-mem*, *simp*)
apply (*frule-tac* $a = f\ (Suc\ n)$ **and** $b = g\ (Suc\ n)$ **in**
Subring-pOp-ring-pOp[*of S*], *simp*, *simp*)
apply *simp*
done

lemma (**in** *PolynRg*) *polyn-add-n*:⟦*pol-coeff S* (n, f); *pol-coeff S* (n, g)⟧ ⟹
(*polyn-expr R X n* (n, f)) $\pm$ (*polyn-expr R X n* (n, g)) =
nsum R $(\lambda j.\ ((f\ j) \pm_S (g\ j)) \cdot_r (X\hat{}^R\ j))\ n$
by (*simp add*:*polyn-addTr*)

definition
add-cf :: [('*a*, '*m*) *Ring-scheme*, *nat* × (*nat* ⇒ '*a*), *nat* × (*nat* ⇒ '*a*)] ⇒
nat × (*nat* ⇒ '*a*) **where**
add-cf S c d =
(*if* (*fst c*) < (*fst d*) *then* ((*fst d*), $\lambda j.$ (*if* $j \leq$ (*fst c*)
then (((*snd c*) j) $\pm_S$ ((*snd d*) j)) *else* ((*snd d*) j)))
else if (*fst c*) = (*fst d*) *then* ((*fst c*), $\lambda j.$ ((*snd c*) $j \pm_S$ (*snd d*) j))
else ((*fst c*), $\lambda j.$ (*if* $j \leq$ (*fst d*) *then*
((*snd c*) $j \pm_S$ (*snd d*) j) *else* ((*snd c*) j))))

lemma (**in** *PolynRg*) *add-cf-pol-coeff*:⟦*pol-coeff S c*; *pol-coeff S d*⟧
⟹ *pol-coeff S* (*add-cf S c d*)
apply (*cut-tac subring*,
frule subring-Ring[*of S*], *frule Ring.ring-is-ag*[*of S*])
apply (*simp add*:*pol-coeff-def*)
apply (*rule allI*, *rule impI*)

apply (*case-tac* (*fst c*) < (*fst d*), *simp add*:*add-cf-def*)
apply (*rule impI*, *rule aGroup.ag-pOp-closed*, *assumption*+, *simp*+)
apply (*drule leI*[*of fst c fst d*],

```
        drule le-imp-less-or-eq[of fst d fst c])
apply (erule disjE)
 apply (simp add:add-cf-def, rule impI)
 apply (frule Ring.ring-is-ag[of S], rule aGroup.ag-pOp-closed, assumption,
      simp+)

apply (simp add:add-cf-def)
apply (frule Ring.ring-is-ag[of S], rule aGroup.ag-pOp-closed, assumption,
      simp+)
done

lemma (in PolynRg) add-cf-len:⟦pol-coeff S c; pol-coeff S d⟧
     ⟹ fst (add-cf S c d) = (max (fst c) (fst d))
by (simp add: add-cf-def max.absorb1 max.absorb2)

lemma (in PolynRg) polyn-expr-restrict1:⟦pol-coeff S (n, f);
    pol-coeff S (Suc (m + n), g)⟧ ⟹
    polyn-expr R X (m + n) (add-cf S (n, f) (m + n, g)) =
    polyn-expr R X (m + n) (m + n, snd (add-cf S (n, f) (Suc (m + n), g)))
apply (frule pol-coeff-pre[of m+n g])
apply (frule add-cf-pol-coeff[of (n, f) (Suc (m + n), g)], assumption+,
      frule add-cf-pol-coeff[of (n, f) (m + n, g)], assumption+)
apply (rule polyn-exprs-eq[of add-cf S (n, f) (m + n, g)
      (m + n, snd (add-cf S (n, f) (Suc (m + n), g))) m + n], assumption+)
 apply (rule split-pol-coeff[of add-cf S (n, f) (Suc (m + n), g) m + n],
       assumption, simp add:add-cf-len)
 apply (simp add:add-cf-len)

apply (rule allI, rule impI)
 apply (simp add:add-cf-def)
done

lemma (in PolynRg) polyn-add-n1:⟦pol-coeff S (n, f); pol-coeff S (n, g)⟧ ⟹
     (polyn-expr R X n (n, f)) ± (polyn-expr R X n (n, g)) =
                          polyn-expr R X n (add-cf S (n, f) (n, g))
apply (subst polyn-add-n, assumption+)
 apply (simp add:polyn-expr-def add-cf-def)
done

lemma (in PolynRg) add-cf-val-hi:(fst c) < (fst d) ⟹
                   snd (add-cf S c d) (fst d) = (snd d) (fst d)
by (simp add:add-cf-def)

lemma (in PolynRg) add-cf-commute:⟦pol-coeff S c; pol-coeff S d⟧
  ⟹ ∀ j ≤ (max (fst c) (fst d)). snd (add-cf S c d) j =
                       snd (add-cf S d c) j
apply (cut-tac subring, frule subring-Ring,
      frule Ring.ring-is-ag[of S])
apply (simp add: add-cf-def max.absorb1 max.absorb2)
```

apply (*case-tac* (*fst c*) = (*fst d*), *simp add*: *pol-coeff-def*)
 apply (*rule allI*, *rule impI*,
  rule aGroup.ag-pOp-commute[*of S*], *simp*+)

apply (*case-tac* (*fst d*) < (*fst c*), *simp*,
  rule allI, *rule impI*,
  rule aGroup.ag-pOp-commute, *assumption*+)
apply (*frule-tac x* = *j* **and** *y* = *fst d* **and** *z* = *fst c* **in** *le-less-trans*,
   assumption+, *frule-tac x* = *j* **and** *y* = *fst c* **in** *less-imp-le*,
   thin-tac j < *fst c*, *simp add*:*pol-coeff-mem*, *simp add*:*pol-coeff-mem*)

apply *simp*
apply (*frule leI*[*of fst d fst c*],
  frule noteq-le-less[*of fst c fst d*], *assumption*,
  rule allI, *rule impI*,
  simp)

apply (*rule aGroup.ag-pOp-commute*, *assumption*+,
  simp add:*pol-coeff-mem*,
  frule-tac x = *j* **and** *y* = *fst c* **and** *z* = *fst d* **in** *le-less-trans*,
   assumption+, *frule-tac x* = *j* **and** *y* = *fst d* **in** *less-imp-le*,
   thin-tac j < *fst d*, *simp add*:*pol-coeff-mem*)
done

lemma (**in** *PolynRg*) *polyn-addTr1*:*pol-coeff S* (*n*, *f*) ⟹
 ∀ *g*. *pol-coeff S* (*n* + *m*, *g*) ⟶
  (*polyn-expr R X n* (*n*, *f*) ± (*polyn-expr R X* (*n* + *m*) ((*n* + *m*), *g*))
    = *polyn-expr R X* (*n* + *m*) (*add-cf S* (*n*, *f*) ((*n* + *m*), *g*)))
apply (*cut-tac subring*, *frule subring-Ring*)
apply (*induct-tac m*)
 apply (*rule allI*, *rule impI*, *simp*)
 apply (*simp add*:*polyn-add-n1*)

apply (*simp add*:*add.commute*[*of n*])
 apply (*rule allI*, *rule impI*)
  apply (*frule-tac n* = *na* + *n* **and** *f* = *g* **in** *pol-coeff-pre*)
  apply (*drule-tac a* = *g* **in** *forall-spec*, *assumption*)
  apply (*cut-tac n* = *na* + *n* **and** *c* = (*Suc* (*na* + *n*), *g*) **in** *polyn-Suc*,
   simp, *simp del*:*npow-suc*,
   thin-tac polyn-expr R X (*Suc* (*na* + *n*)) (*Suc* (*na* + *n*), *g*) =
   polyn-expr R X (*na* + *n*) (*Suc* (*na* + *n*), *g*) ±
   g (*Suc* (*na* + *n*)) ·$_r$ *X*^*R* (*Suc* (*na* + *n*)))
  apply (*frule-tac c* = (*n*, *f*) **and** *k* = *n* **in** *polyn-mem*, *simp*,
   frule-tac c = (*Suc* (*na* + *n*), *g*) **and** *k* = *na* + *n* **in** *polyn-mem*, *simp*,
   frule-tac c = (*Suc* (*na* + *n*), *g*) **in** *monomial-mem*)
  apply (*drule-tac a* = *Suc* (*na* + *n*) **in** *forall-spec*, *simp del*:*npow-suc*,
   cut-tac ring-is-ag,
   subst aGroup.ag-pOp-assoc[*THEN sym*], *assumption*+, *simp del*:*npow-suc*)
  apply (*simp del*:*npow-suc add*:*polyn-expr-restrict*)

apply (*frule-tac* $c = (n, f)$ **and** $d = (Suc\ (na + n), g)$ **in**
add-cf-pol-coeff, *assumption*+,
frule-tac $c = (n, f)$ **and** $d = (na + n, g)$ **in**
add-cf-pol-coeff, *assumption*+)
apply (*frule-tac* $c =$ *add-cf* $S\ (n, f)\ (Suc\ (na + n), g)$ **and**
$n = na + n$ **and** $m = Suc\ (na + n)$ **in** *polyn-n-m*, *simp*,
subst add-cf-len, *assumption*+, *simp*)
apply (*cut-tac* $k = Suc\ (na + n)$ **and** $f =$ *add-cf* $S\ (n, f)\ (Suc\ (na + n), g)$
in *polyn-expr-split*)
apply (*frule-tac* $c = (n, f)$ **and** $d = (Suc\ (na + n), g)$ **in**
add-cf-len, *assumption*+, *simp del*: *npow-suc add*: *max.absorb1 max.absorb2*)
apply (*thin-tac polyn-expr* $R\ X\ (Suc\ (na + n))$
$(Suc\ (na + n),$ *snd* $($*add-cf* $S\ (n, f)\ (Suc\ (na + n), g)))) =$
polyn-expr $R\ X\ (na + n)$
$(na + n,$ *snd* $($*add-cf* $S\ (n, f)\ (Suc\ (na + n), g))) \pm$
$\Sigma_f\ R\ (\lambda j.$ *snd* $($*add-cf* $S\ (n, f)\ (Suc\ (na + n), g))\ j \cdot_r$
$X\hat{}^R\ j)\ (Suc\ (na + n))\ (Suc\ (na + n)),$
thin-tac polyn-expr $R\ X\ (Suc\ (na + n))\ ($*add-cf* $S\ (n, f)\ (Suc\ (na + n),$
$g)) =$
polyn-expr $R\ X\ (na + n)$
$(na + n,$ *snd* $($*add-cf* $S\ (n, f)\ (Suc\ (na + n), g))) \pm$
$\Sigma_f\ R\ (\lambda j.$ *snd* $($*add-cf* $S\ (n, f)\ (Suc\ (na + n), g))\ j \cdot_r$
$X\hat{}^R\ j)\ (Suc\ (na + n))\ (Suc\ (na + n)))$
apply (*simp del*:*npow-suc add*:*fSum-def cmp-def slide-def*)
apply (*cut-tac* $d = (Suc\ (na + n), g)$ **in** *add-cf-val-hi*[*of* (n, f)],
simp, *simp del*:*npow-suc*,
thin-tac snd $($*add-cf* $S\ (n, f)\ (Suc\ (na + n), g))\ (Suc\ (na + n)) =$
$g\ (Suc\ (na + n)))$
apply (*frule-tac* $c =$ *add-cf* $S\ (n, f)\ (Suc\ (na + n), g)$ **and** $k = na + n$ **in**
polyn-mem, *simp*,
frule-tac $c =$ *add-cf* $S\ (n, f)\ (na + n, g)$ **and** $k = na + n$ **in**
polyn-mem, *simp*)
apply (*subst add-cf-len*, *assumption*+, *simp del*:*npow-suc*)
apply (*frule-tac* $a =$ *polyn-expr* $R\ X\ (na + n)\ ($*add-cf* $S\ (n, f)\ (na + n, g))$
and $b =$ *polyn-expr* $R\ X\ (na + n)\ ($*add-cf* $S\ (n, f)\ (Suc\ (na + n), g))$
and $c = g\ (Suc\ (na + n)) \cdot_r\ \ X\hat{}^R\ (Suc\ (na + n))$ **in**
aGroup.ag-pOp-add-r[*of R*], *assumption*+)
apply (*rule-tac* $c =$ *add-cf* $S\ (n, f)\ (na + n, g)$ **and**
$d =$ *add-cf* $S\ (n, f)\ (Suc\ (na + n), g)$ **and** $k = na + n$ **in**
polyn-exprs-eq, *assumption*+, *simp*,
subst add-cf-len, *assumption*+)
apply (*simp*)

apply (*rule allI*, *rule impI*,
(*subst add-cf-def*)+, *simp*,
frule-tac $m = na$ **and** $g = g$ **in** *polyn-expr-restrict1*[*of n f*], *assumption*,
simp del:*npow-suc*)
done

lemma (**in** *PolynRg*) *polyn-add*:⟦*pol-coeff S* (*n*, *f*); *pol-coeff S* (*m*, *g*)⟧
⟹ *polyn-expr R X n* (*n*, *f*) ± (*polyn-expr R X m* (*m*, *g*))
= *polyn-expr R X* (*max n m*) (*add-cf S* (*n*, *f*) (*m*, *g*))
apply (*cut-tac less-linear*[*of n m*])
apply (*erule disjE*,
frule polyn-addTr1[*of n f m* − *n*],
drule-tac a = *g* **in** *forall-spec*, *simp*, *simp add*: *max.absorb1 max.absorb2*)

apply (*erule disjE*,
simp add:*polyn-add-n1*)
apply (*frule polyn-mem*[*of* (*n*, *f*) *n*], *simp*,
frule polyn-mem[*of* (*m*, *g*) *m*], *simp*)
apply (*cut-tac ring-is-ag*, *simp add*:*aGroup.ag-pOp-commute*)

apply (*frule polyn-addTr1*[*of m g n* − *m*],
drule-tac a = *f* **in** *forall-spec*, *simp*, *simp*,
frule add-cf-commute[*of* (*m*, *g*) (*n*, *f*)], *assumption*+,
simp add:*max-def*,
frule add-cf-pol-coeff[*of* (*n*, *f*) (*m*, *g*)], *assumption*+,
frule add-cf-pol-coeff[*of* (*m*, *g*) (*n*, *f*)], *assumption*+)
apply (*rule polyn-exprs-eq*[*of add-cf S* (*m*, *g*) (*n*, *f*)
add-cf S (*n*, *f*) (*m*, *g*) *n*], *assumption*+)
apply (*simp add*:*add-cf-len*, *simp*)
done

lemma (**in** *PolynRg*) *polyn-add1*:⟦*pol-coeff S c*; *pol-coeff S d*⟧
⟹ *polyn-expr R X* (*fst c*) *c* ± (*polyn-expr R X* (*fst d*) *d*)
= *polyn-expr R X* (*max* (*fst c*) (*fst d*)) (*add-cf S c d*)
apply (*cases c*)
apply (*cases d*)
apply (*simp add*: *polyn-add*)
done

lemma (**in** *PolynRg*) *polyn-minus-nsum*:⟦*pol-coeff S c*; *k* ≤ (*fst c*)⟧ ⟹
$-_a$ (*polyn-expr R X k c*) = *nsum R* (λ*j*. (($-_{a\,S}$ ((*snd c*) *j*)) ·$_r$ (*X* ^^{R} *j*))) *k*
apply (*cut-tac subring*,
frule subring-Ring[*of S*],
frule Ring.ring-is-ag[*of S*],
cut-tac ring-is-ag,
cut-tac X-mem-R)
apply (*simp add*:*polyn-expr-def*,
subst aGroup.nsum-minus[*of R*], *assumption*)
apply (*frule monomial-mem*[*of c*], *rule allI*, *rule impI*,
frule-tac i = *j* **and** *j* = *k* **and** *k* = *fst c* **in** *le-trans*, *assumption*+,
simp)
apply (*rule aGroup.nsum-eq*, *assumption*,
rule allI, *rule impI*, *simp*,
rule aGroup.ag-mOp-closed, *assumption*)
apply (*frule monomial-mem*[*of c*],

frule-tac $i = j$ **and** $j = k$ **and** $k = fst\ c$ **in** *le-trans*, *assumption*+,
simp)
apply (*rule allI*, *rule impI*,
rule ring-tOp-closed)
apply (*frule-tac* $j = j$ **in** *pol-coeff-mem*[*of c*])
apply (*frule-tac* $i = j$ **and** $j = k$ **and** $k = fst\ c$ **in** *le-trans*, *assumption*+,
simp add:*Subring-minus-ring-minus*,
frule-tac $x = (snd\ c)\ j$ **in** *mem-subring-mem-ring*[*of S*], *assumption*,
simp add:*aGroup.ag-mOp-closed*,
simp add:*npClose*)
apply (*rule allI*, *rule impI*, *simp*,
cut-tac $j = j$ **in** *pol-coeff-mem*[*of c*], *assumption*,
rule-tac $i = j$ **and** $j = k$ **and** $k = fst\ c$ **in** *le-trans*, *assumption*+)
apply (*simp add*:*Subring-minus-ring-minus*,
frule-tac $x = (snd\ c)\ j$ **in** *mem-subring-mem-ring*[*of S*], *assumption*)
apply (*subst ring-inv1-1*, *assumption*+)
apply (*simp add*:*npClose*, *simp*)
done

lemma (**in** *PolynRg*) *minus-pol-coeff*:*pol-coeff S c* $\Longrightarrow$
pol-coeff S ((*fst c*), ($\lambda j.\ (-_{a\,S}\ ((snd\ c)\ j))$))
apply (*simp add*:*pol-coeff-def*)
apply (*rule allI*, *rule impI*)
apply (*cut-tac subring*, *frule subring-Ring*)
apply (*frule Ring.ring-is-ag*[*of S*])
apply (*rule aGroup.ag-mOp-closed*, *assumption*)
apply *simp*
done

lemma (**in** *PolynRg*) *polyn-minus*:$[\![$*pol-coeff S c*; $k \leq (fst\ c)]\!] \Longrightarrow$
$-_a$ (*polyn-expr R X k c*) =
polyn-expr R X k (*fst c*, ($\lambda j.\ (-_{a\,S}\ ((snd\ c)\ j))$))
apply (*cases c*)
apply (*subst polyn-minus-nsum*)
apply (*simp-all add*: *polyn-expr-def*)
done

definition
m-cf :: [('*a*, '*m*) *Ring-scheme*, *nat* $\times$ (*nat* $\Rightarrow$ '*a*)] $\Rightarrow$ *nat* $\times$ (*nat* $\Rightarrow$ '*a*) **where**
m-cf S c = (*fst c*, ($\lambda j.\ (-_{a\,S}\ ((snd\ c)\ j))$))

lemma (**in** *PolynRg*) *m-cf-pol-coeff*:*pol-coeff S c* $\Longrightarrow$
pol-coeff S (*m-cf S c*)
by (*simp add*:*m-cf-def*, *simp add*:*minus-pol-coeff*)

lemma (**in** *PolynRg*) *m-cf-len*:*pol-coeff S c* $\Longrightarrow$
fst (*m-cf S c*) = *fst c*
by (*simp add*:*m-cf-def*)

lemma (**in** *PolynRg*) *polyn-minus-m-cf*:⟦*pol-coeff S c*; $k \leq$ (*fst c*)⟧ ⟹
$-_a$ (*polyn-expr R X k c*) =
polyn-expr R X k (*m-cf S c*)
by (*simp add*:*m-cf-def polyn-minus*)

lemma (**in** *PolynRg*) *polyn-zero-minus-zero*:⟦*pol-coeff S c*; $k \leq$ (*fst c*)⟧ ⟹
(*polyn-expr R X k c* = **0**) = (*polyn-expr R X k* (*m-cf S c*) = **0**)
apply (*cut-tac ring-is-ag*)
apply (*simp add*:*polyn-minus-m-cf*[*THEN sym*])
apply (*rule iffI*, *simp*)
apply (*simp add*:*aGroup.ag-inv-zero*)
apply (*frule polyn-mem*[*of c k*], *assumption*)
apply (*frule aGroup.ag-inv-inv*[*of R polyn-expr R X k c*], *assumption*)
apply (*simp add*:*aGroup.ag-inv-zero*)
done

lemma (**in** *PolynRg*) *coeff-0-pol-0*:⟦*pol-coeff S c*; $k \leq$ *fst c*⟧ ⟹
(∀ $j \leq k$. (*snd c*) $j = \mathbf{0}_S$) = (*polyn-expr R X k c* = **0**)
apply (*rule iffI*)
apply (*cut-tac ring-is-ag*, *cut-tac subring*,
frule subring-Ring)
apply (*simp add*:*Subring-zero-ring-zero*)
apply (*simp add*:*polyn-expr-def*,
rule aGroup.nsum-zeroA[*of R*], *assumption*)
apply (*rule allI*, *rule impI*,
cut-tac X-mem-R)
apply (*drule-tac a* = *j* **in** *forall-spec*, *simp*,
frule-tac n = *j* **in** *npClose*[*of X*], *simp*)
apply (*simp add*:*ring-times-0-x*)
apply (*cases c*)
using *algfree* [*simplified algfree-cond-def*] **by** (*auto simp add*: *polyn-expr-def*)

4.13.4 Multiplication of *pol-exprs*

4.13.5 Multiplication

definition
ext-cf :: [('*a*, '*m*) *Ring-scheme*, *nat*, *nat* × (*nat* ⇒ '*a*)] ⇒
nat × (*nat* ⇒ '*a*) **where**
ext-cf S n c = (*n* + *fst c*, λ*i*. *if n* ≤ *i then* (*snd c*) (*sliden n i*) *else* $\mathbf{0}_S$)

definition
sp-cf :: [('*a*, '*m*) *Ring-scheme*, '*a*, *nat* × (*nat* ⇒ '*a*)] ⇒ *nat* × (*nat* ⇒ '*a*) **where**
sp-cf S a c = (*fst c*, λ*j*. *a* $\cdot_{r\,S}$ ((*snd c*) *j*))

definition
special-cf :: ('*a*, '*m*) *Ring-scheme* ⇒ *nat* × (*nat* ⇒ '*a*) (C_0) **where**
C_0 *S* = (*0*, λ*j*. $1_{r\,S}$)

lemma (**in** *PolynRg*) *special-cf-pol-coeff*:*pol-coeff S* (C_0 *S*)
apply (*cut-tac subring*, *frule subring-Ring*)
apply (*simp add*:*pol-coeff-def special-cf-def*)
apply (*simp add*:*Ring.ring-one*)
done

lemma (**in** *PolynRg*) *special-cf-len*:*fst* (C_0 *S*) = *0*
apply (*simp add*:*special-cf-def*)
done

lemma (**in** *PolynRg*) *ext-cf-pol-coeff*:*pol-coeff S c* ⟹
pol-coeff S (*ext-cf S n c*)
apply (*simp add*: *pol-coeff-def ext-cf-def sliden-def*)
apply (*rule impI*)
apply (*rule Ring.ring-zero*)
apply (*rule subring-Ring*)
apply (*rule subring*)
done

lemma (**in** *PolynRg*) *ext-cf-len*:*pol-coeff S c* ⟹
fst (*ext-cf S m c*) = *m* + *fst c*
by (*simp add*:*ext-cf-def*)

lemma (**in** *PolynRg*) *ext-special-cf-len*:*fst* (*ext-cf S m* (C_0 *S*)) = *m*
apply (*cut-tac special-cf-pol-coeff*)
apply (*simp add*:*ext-cf-len special-cf-def*)
done

lemma (**in** *PolynRg*) *ext-cf-self*:*pol-coeff S c* ⟹
∀ *j* ≤ (*fst c*). *snd* (*ext-cf S 0 c*) *j* = (*snd c*) *j*
apply (*rule allI*, *rule impI*, *simp add*:*ext-cf-def sliden-def*)
done

lemma (**in** *PolynRg*) *ext-cf-hi*:*pol-coeff S c* ⟹
(*snd c*) (*fst c*) =
snd (*ext-cf S n c*) (*n* + (*fst c*))
apply (*subst ext-cf-def*)
apply (*simp add*:*sliden-def*)
done

lemma (**in** *PolynRg*) *ext-special-cf-hi*:*snd* (*ext-cf S n* (C_0 *S*)) *n* = $1_{r\,S}$
apply (*cut-tac special-cf-pol-coeff*)
apply (*cut-tac ext-cf-hi*[*of* C_0 *S n*, *THEN sym*])
apply (*simp add*:*special-cf-def*, *assumption*)
done

lemma (**in** *PolynRg*) *ext-cf-lo-zero*:⟦*pol-coeff S c*; *0* < *n*; *x* ≤ (*n* − *Suc 0*)⟧

$\Longrightarrow$ *snd* (*ext-cf S n c*) *x* = $\mathbf{0}_S$
apply (*cut-tac Suc-le-mono*[*THEN sym*, *of x n* − *Suc 0*], *simp*,
cut-tac x = *x* **and** *y* = *Suc x* **and** *z* = *n* **in** *less-le-trans*,
simp, *assumption*,
simp add:*nat-not-le-less*[*THEN sym*, *of x n*],
thin-tac x ≤ *n* − *Suc 0*)
apply (*simp add*:*ext-cf-def*)
done

lemma (**in** *PolynRg*) *ext-special-cf-lo-zero*:⟦*0* < *n*; *x* ≤ (*n* − *Suc 0*)⟧
$\Longrightarrow$ *snd* (*ext-cf S n* (*C*$_0$ *S*)) *x* = $\mathbf{0}_S$
by (*cut-tac special-cf-pol-coeff*,
frule ext-cf-lo-zero[*of C*$_0$ *S n*], *assumption*+)

lemma (**in** *PolynRg*) *sp-cf-pol-coeff*:⟦*pol-coeff S c*; *a* ∈ *carrier S*⟧ $\Longrightarrow$
pol-coeff S (*sp-cf S a c*)
apply (*cut-tac subring*, *frule subring-Ring*)
apply (*simp add*:*pol-coeff-def sp-cf-def*,
rule allI, *rule impI*,
rule Ring.ring-tOp-closed, *assumption*+)
apply *simp*
done

lemma (**in** *PolynRg*) *sp-cf-len*:⟦*pol-coeff S c*; *a* ∈ *carrier S*⟧ $\Longrightarrow$
fst (*sp-cf S a c*) = *fst c*
by (*simp add*:*sp-cf-def*)

lemma (**in** *PolynRg*) *sp-cf-val*:⟦*pol-coeff S c*; *j* ≤ (*fst c*); *a* ∈ *carrier S*⟧ $\Longrightarrow$
snd (*sp-cf S a c*) *j* = *a* ·$_{rS}$ ((*snd c*) *j*)
by (*simp add*:*sp-cf-def*)

lemma (**in** *PolynRg*) *polyn-ext-cf-lo-zero*:⟦*pol-coeff S c*; *0* < *j*⟧ $\Longrightarrow$
polyn-expr R X (*j* − *Suc 0*) (*ext-cf S j c*) = $\mathbf{0}$
apply (*simp add*:*polyn-expr-def*, *cut-tac ring-is-ag*,
rule aGroup.nsum-zeroA, *assumption*)
apply (*rule allI*, *rule impI*)
apply (*frule-tac x* = *ja* **in** *ext-cf-lo-zero* [*of c j*], *assumption*+)
apply (*cut-tac X-mem-R*, *frule-tac n* = *ja* **in** *npClose*[*of X*])
apply (*cut-tac subring*,
simp add:*Subring-zero-ring-zero*,
simp add:*ring-times-0-x*)
done

lemma (**in** *PolynRg*) *monomial-d*:*pol-coeff S c* $\Longrightarrow$
polyn-expr R X d (*ext-cf S d c*) = ((*snd c*) *0*) ·$_r$ *X*^$_R$ *d*
apply (*cut-tac ring-is-ag*,
cut-tac subring,
cut-tac X-mem-R,
frule subring-Ring[*of S*])

apply (*frule pol-coeff-mem* [*of c 0*], *simp*)
apply (*case-tac d = 0*)
apply *simp*
apply (*simp add*:*polyn-expr-def ext-cf-def sliden-def*)
apply (*frule ext-cf-pol-coeff*[*of c d*])
apply (*cut-tac polyn-Suc*[*of d − Suc 0 ext-cf S d c*])
apply (*simp*)
apply (*cut-tac polyn-ext-cf-lo-zero*[*of c d*], *simp*,
thin-tac polyn-expr R X (*d − Suc 0*) (*ext-cf S d c*) = **0**)
apply (*frule monomial-mem*[*of ext-cf S d c*], *simp add*:*ext-cf-len*,
drule-tac a = d **in** *forall-spec*, *simp*, *simp add*:*aGroup.ag-l-zero*)
apply (*subst polyn-expr-short*[*of ext-cf S d c d*], *assumption*,
simp add:*ext-cf-len*)
apply (*simp*,
subst ext-cf-def , *simp add*:*sliden-def* , *assumption*+,
simp add:*ext-cf-len*)
done

lemma (**in** *PolynRg*) *X-to-d*: $X\hat{}^{R\ d} =$ *polyn-expr R X d* (*ext-cf S d* (C_0 *S*))
apply (*cut-tac special-cf-pol-coeff*)
apply (*subst monomial-d*[*of* C_0 *S d*], *assumption*+)
apply (*subst special-cf-def*, *simp*)
apply (*cut-tac subring*, *frule subring-Ring*)
apply (*simp add*:*Subring-one-ring-one*)
apply (*cut-tac X-mem-R*, *frule-tac n = d* **in** *npClose*[*of X*])
apply (*simp add*:*ring-l-one*)
done

lemma (**in** *PolynRg*) *c-max-ext-special-cf*:*c-max S* (*ext-cf S n* (C_0 *S*)) = *n*
apply (*cut-tac polyn-ring-S-nonzero*,
cut-tac subring, *frule subring-Ring*)
apply (*simp add*:*c-max-def special-cf-def ext-cf-def*)
apply (*cut-tac n-max*[*of* {*j*. (*n* ≤ *j* ⟶ *j* = *n*) ∧ *n* ≤ *j*} *n*])
apply (*erule conjE*)+ **apply** *simp*
apply (*rule subsetI*, *simp*, *erule conjE*, *simp*)
apply (*cut-tac le-refl*[*of n*], *blast*)
done

lemma (**in** *PolynRg*) *scalar-times-polynTr*:*a* ∈ *carrier S* ⟹
∀*f*. *pol-coeff S* (*n*, *f*) ⟶
a ·$_r$ (*polyn-expr R X n* (*n*, *f*)) = *polyn-expr R X n* (*sp-cf S a* (*n*, *f*))
apply (*cut-tac subring*,
cut-tac X-mem-R,
frule-tac x = a **in** *mem-subring-mem-ring*, *assumption*)
apply (*induct-tac n*,
rule allI, *rule impI*, *simp add*:*polyn-expr-def sp-cf-def*,
cut-tac n-in-Nsetn[*of 0*])
apply (*cut-tac subring-Ring*,
frule-tac c = (*0*, *f*) **in** *pol-coeff-mem*[*of - 0*], *simp*)

apply (*simp*,
frule-tac $x = f\ 0$ **in** *mem-subring-mem-ring*, *assumption*)
apply (*simp add*:*Subring-tOp-ring-tOp*,
frule-tac $y = f\ 0$ **in** *ring-tOp-closed*[*of a*], *assumption*+,
cut-tac ring-one, *simp add*:*ring-tOp-assoc*, *assumption*)

apply (*rule allI*, *rule impI*,
frule subring-Ring,
frule-tac $n = n$ **and** $f = f$ **in** *pol-coeff-pre*,
drule-tac $x = f$ **in** *spec*, *simp*)
apply (*cut-tac* $n = n$ **and** $c = (Suc\ n, f)$ **in** *polyn-Suc*, *simp*,
simp del:*npow-suc*,
thin-tac polyn-expr R X (*Suc n*) (*Suc n*, *f*) =
polyn-expr R X n (*Suc n*, *f*) $\pm$ *f* (*Suc n*) $\cdot_r$ $X\hat{}R$ (*Suc n*))
apply (*cut-tac* $n = n$ **and** $c = sp\text{-}cf\ S\ a\ (Suc\ n, f)$ **in** *polyn-Suc*,
simp add:*sp-cf-len*)
apply (*frule-tac* $c = (Suc\ n, f)$ **and** $a = a$ **in** *sp-cf-len*, *assumption*+,
simp only:*fst-conv*)
apply (*cut-tac* $k = Suc\ n$ **and** $f = sp\text{-}cf\ S\ a\ (Suc\ n, f)$ **in**
polyn-expr-split, *simp del*:*npow-suc*,
thin-tac polyn-expr R X (*Suc n*) (*Suc n*, *snd* (*sp-cf S a* (*Suc n*, *f*))) =
polyn-expr R X n (*sp-cf S a* (*Suc n*, *f*)) $\pm$
snd (*sp-cf S a* (*Suc n*, *f*)) (*Suc n*) $\cdot_r$ $X\hat{}R$ (*Suc n*),
thin-tac polyn-expr R X (*Suc n*) (*sp-cf S a* (*Suc n*, *f*)) =
polyn-expr R X n (*sp-cf S a* (*Suc n*, *f*)) $\pm$
snd (*sp-cf S a* (*Suc n*, *f*)) (*Suc n*) $\cdot_r$ $X\hat{}R$ (*Suc n*))
apply (*frule-tac* $c = (Suc\ n, f)$ **and** $a = a$ **in** *sp-cf-pol-coeff*, *assumption*)
apply (*frule-tac* $c = (Suc\ n, f)$ **and** $k = n$ **in** *polyn-mem*,
simp,
frule-tac $c = (Suc\ n, f)$ **in** *monomial-mem*,
drule-tac $a = Suc\ n$ **in** *forall-spec*, *simp*,
simp only:*snd-conv*)
apply (*subst ring-distrib1*, *assumption*+,
subst polyn-expr-restrict, *assumption*+, *simp del*:*npow-suc*,
subst sp-cf-val, *assumption*, *simp*, *assumption*,
simp only:*snd-conv*,
frule-tac $c = (Suc\ n, f)$ **and** $j = Suc\ n$ **in** *pol-coeff-mem*,
simp, *simp only*:*snd-conv*,
simp del:*npow-suc add*:*Subring-tOp-ring-tOp*,
subst ring-tOp-assoc[*THEN sym*, *of a*], *assumption*+,
simp add:*mem-subring-mem-ring*, *rule npClose*, *assumption*)
apply (*cut-tac ring-is-ag*,
rule aGroup.ag-pOp-add-r, *assumption*+,
rule polyn-mem, *rule sp-cf-pol-coeff*, *assumption*+,
simp add:*sp-cf-len*,
rule polyn-mem, *assumption*, *simp add*:*sp-cf-len*,
frule-tac $c = (Suc\ n, f)$ **and** $j = Suc\ n$ **in** *pol-coeff-mem-R*,
simp, *simp only*:*snd-conv*,
(*rule ring-tOp-closed*)+, *assumption*+, *rule npClose*, *assumption*)

apply (*rule-tac* $c = sp\text{-}cf\ S\ a\ (n, f)$ **and** $d = sp\text{-}cf\ S\ a\ (Suc\ n, f)$ **and**
$k = n$ **in** *polyn-exprs-eq*, *rule sp-cf-pol-coeff*, *assumption*+,
simp add:*sp-cf-len*)

apply (*rule allI*, *rule impI*,
(*subst sp-cf-def*)+, *simp*)
done

lemma (**in** *PolynRg*) *scalar-times-pol-expr*:⟦$a \in carrier\ S$; *pol-coeff S c*;
$n \leq fst\ c$⟧ ⟹
$a \cdot_r (polyn\text{-}expr\ R\ X\ n\ c) = polyn\text{-}expr\ R\ X\ n\ (sp\text{-}cf\ S\ a\ c)$
apply (*cases c*) **apply** (*simp only*:)
apply (*rename-tac m g*)
apply (*thin-tac* $c = (m, g)$)
apply (*frule-tac* $c = (m, g)$ **and** $k = n$ **in** *polyn-expr-short*, *simp*,
simp)
apply (*frule scalar-times-polynTr*[*of a n*],
drule-tac $x = g$ **in** *spec*)
apply (*frule-tac* $c = (m, g)$ **and** $n = n$ **in** *pol-coeff-le*, *simp*, *simp*,
thin-tac polyn-expr R X n $(m, g) = polyn\text{-}expr\ R\ X\ n\ (n, g)$,
thin-tac a $\cdot_r$ *polyn-expr R X n* $(n, g) =$
polyn-expr R X n $(sp\text{-}cf\ S\ a\ (n, g))$)
apply (*frule-tac* $c = (m, g)$ **and** $n = n$ **in** *pol-coeff-le*, *simp*, *simp*,
frule-tac $c = (n, g)$ **and** $a = a$ **in** *sp-cf-pol-coeff*, *assumption*,
frule-tac $c = (m, g)$ **and** $a = a$ **in** *sp-cf-pol-coeff*, *assumption*)
apply (*rule-tac* $c = sp\text{-}cf\ S\ a\ (n, g)$ **and** $d = sp\text{-}cf\ S\ a\ (m, g)$ **and**
$k = n$ **in** *polyn-exprs-eq*, *assumption*+)
apply (*simp add*:*sp-cf-len*)
apply (*rule allI*, (*subst sp-cf-def*)+, *simp*)
done

lemma (**in** *PolynRg*) *sp-coeff-nonzero*:⟦*Idomain S*; $a \in carrier\ S$; $a \neq \mathbf{0}_S$;
pol-coeff S c; $(snd\ c)\ j \neq \mathbf{0}_S$; $j \leq (fst\ c)$⟧ ⟹
$snd\ (sp\text{-}cf\ S\ a\ c)\ j \neq \mathbf{0}_S$
apply (*simp add*:*sp-cf-def*)
apply (*frule-tac* $y = (snd\ c)\ j$ **in** *Idomain.idom-tOp-nonzeros*[*of S a*],
assumption+,
simp add:*pol-coeff-def*, *simp add*:*Pi-def*, *assumption*+)
done

lemma (**in** *PolynRg*) *ext-cf-inductTl*:*pol-coeff S* $(Suc\ n, f)$ ⟹
$polyn\text{-}expr\ R\ X\ (n + j)\ (ext\text{-}cf\ S\ j\ (Suc\ n, f)) =$
$polyn\text{-}expr\ R\ X\ (n + j)\ (ext\text{-}cf\ S\ j\ (n, f))$
apply (*frule pol-coeff-pre*[*of n f*],
frule ext-cf-pol-coeff[*of* $(Suc\ n, f)\ j$],
frule ext-cf-pol-coeff[*of* $(n, f)\ j$],
rule polyn-exprs-eq[*of ext-cf S j* $(Suc\ n, f)$ *ext-cf S j* (n, f)
$n + j$], *assumption*+)
apply (*simp add*:*ext-cf-len*)

apply (*rule allI*, (*subst ext-cf-def*)+, *simp add*:*sliden-def*)
done

lemma (**in** *PolynRg*) *low-deg-terms-zeroTr*:
  pol-coeff S (*n*, *f*) ⟶
  polyn-expr R X (*n* + *j*) (*ext-cf S j* (*n*, *f*)) =
    (*X* ^*R j*) ·*r* (*polyn-expr R X n* (*n*, *f*))
apply (*cut-tac ring-is-ag*,
    cut-tac X-mem-R, *frule npClose*[*of X j*])
apply (*induct-tac n*)
 apply (*rule impI*, *simp*)
 apply (*case-tac j* = *0*, *simp add*:*ext-cf-def sliden-def polyn-expr-def*)
 apply (*frule-tac c* = (*0*, *f*) **and** *j* = *0* **in** *pol-coeff-mem-R*, *simp*, *simp*)
 apply (*simp add*:*ring-r-one ring-l-one*)
 apply (*cut-tac polyn-Suc*[*of j* − *Suc 0 ext-cf S j* (*0*, *f*)],
    simp del:*npow-suc*)
 apply (*frule ext-cf-len*[*of* (*0*, *f*) *j*],
    cut-tac polyn-expr-split[*of j ext-cf S j* (*0*, *f*)], *simp*,
    thin-tac polyn-expr R X j (*ext-cf S j* (*0*, *f*)) =
    polyn-expr R X (*j* − *Suc 0*) (*ext-cf S j* (*0*, *f*)) ±
    snd (*ext-cf S j* (*0*, *f*)) *j* ·*r* *X* ^*R j*)
 apply (*simp add*:*polyn-ext-cf-lo-zero*[*of* (*0*, *f*) *j*],
    thin-tac polyn-expr R X j (*j*, *snd* (*ext-cf S j* (*0*, *f*))) =
    0 ± *snd* (*ext-cf S j* (*0*, *f*)) *j* ·*r* *X* ^*R j*,
    frule ext-cf-hi[*THEN sym*, *of* (*0*, *f*) *j*], *simp add*:*polyn-expr-def*)
  apply (*frule-tac c* = (*0*, *f*) **and** *j* = *0* **in** *pol-coeff-mem-R*, *simp*, *simp*)
  apply (*subst aGroup.ag-l-zero*, *assumption*, *simp add*:*ring-tOp-closed*,
    simp add:*ring-r-one*, *subst ring-tOp-commute*, *assumption*+, *simp*)

 apply (*simp add*:*ext-cf-len*)
apply (*rule impI*,
    cut-tac subring,
    cut-tac subring-Ring[*of S*],
    frule-tac n = *n* **in** *pol-coeff-pre*[*of* - *f*])
    apply *simp*
 apply (*subst polyn-expr-split*)
 apply (*cut-tac n* = *n* + *j* **and** *c* = *ext-cf S j* (*Suc n*, *f*) **in** *polyn-Suc*,
    simp add:*ext-cf-len*)
 apply (*subst ext-cf-len*, *assumption*+, *simp del*:*npow-suc add*:*add.commute*[*of j*],
    thin-tac polyn-expr R X (*Suc* (*n* + *j*))
      (*Suc* (*n* + *j*), *snd* (*ext-cf S j* (*Suc n*, *f*))) =
     polyn-expr R X (*n* + *j*) (*ext-cf S j* (*Suc n*, *f*)) ±
     snd (*ext-cf S j* (*Suc n*, *f*)) (*Suc* (*n* + *j*)) ·*r* *X* ^*R* (*Suc* (*n* + *j*)),
    subst ext-cf-inductTl, *assumption*+, *simp del*:*npow-suc*,
    thin-tac polyn-expr R X (*n* + *j*) (*ext-cf S j* (*n*, *f*)) =
     X ^*R j* ·*r* *polyn-expr R X n* (*n*, *f*))
 apply (*cut-tac c1* = (*Suc n*, *f*) **and** *n1* = *j* **in** *ext-cf-hi*[*THEN sym*],
    assumption+,
    simp del:*npow-suc add*:*add.commute*[*of j*])

apply (*cut-tac* $n = n$ **and** $c = (Suc\ n, f)$ **in** *polyn-Suc*, *simp*,
simp del:*npow-suc*)
apply (*frule-tac* $c = (Suc\ n, f)$ **and** $k = n$ **in** *polyn-mem*, *simp*,
frule-tac $c = (Suc\ n, f)$ **and** $j = Suc\ n$ **in** *pol-coeff-mem-R*, *simp*,
simp del:*npow-suc*,
frule-tac $x = f\ (Suc\ n)$ **and** $y = X\hat{}^R\ (Suc\ n)$ **in** *ring-tOp-closed*,
rule npClose, *assumption*,
subst ring-distrib1, *assumption*+)
apply (*subst polyn-expr-restrict*, *assumption*+)
apply (*rule-tac* $a = f\ (Suc\ n) \cdot_r X\hat{}^R\ (Suc\ (n + j))$ **and**
$b = X\hat{}^R\ j \cdot_r (f\ (Suc\ n) \cdot_r X\hat{}^R\ (Suc\ n))$ **and**
$c = X\hat{}^R\ j \cdot_r$ *polyn-expr R X n* (n, f) **in** *aGroup.ag-pOp-add-l*,
assumption+,
rule ring-tOp-closed, *assumption*+, *rule npClose*, *assumption*,
(*rule ring-tOp-closed*, *assumption*+)+,
simp add:*polyn-mem*,
frule-tac $n = Suc\ n$ **in** *npClose*[*of X*],
subst ring-tOp-assoc[*THEN sym*], *assumption*+,
subst ring-tOp-commute[*of* $X\hat{}^R\ j$], *assumption*,
simp add:*pol-coeff-mem*,
subst ring-tOp-assoc, *assumption*+,
subst npMulDistr[*of X*], *assumption*, *simp add*:*add.commute*[*of j*])
apply *simp*
done

lemma (**in** *PolynRg*) *low-deg-terms-zero*:*pol-coeff S* $(n, f) \Longrightarrow$
polyn-expr R X $(n + j)$ (*ext-cf S j* (n, f)) =
$(X\hat{}^R\ j) \cdot_r$ (*polyn-expr R X n* (n, f))
by (*simp add*:*low-deg-terms-zeroTr*)

lemma (**in** *PolynRg*) *low-deg-terms-zero1*:*pol-coeff S c* $\Longrightarrow$
polyn-expr R X ((*fst c*) + j) (*ext-cf S j c*) =
$(X\hat{}^R\ j) \cdot_r$ (*polyn-expr R X* (*fst c*) *c*)
by (*cases c*) (*simp add*: *low-deg-terms-zeroTr*)

lemma (**in** *PolynRg*) *polyn-expr-tOpTr*:*pol-coeff S* $(n, f) \Longrightarrow$
$\forall g.$ (*pol-coeff S* $(m, g) \longrightarrow (\exists h.$ *pol-coeff S* $((n + m), h) \wedge$
$h\ (n + m) = (f\ n) \cdot_{r\,S} (g\ m) \wedge$
(*polyn-expr R X* $(n + m)$ $(n + m, h) =$
(*polyn-expr R X n* (n, f)) $\cdot_r$ (*polyn-expr R X m* (m, g))))))
apply (*cut-tac subring*,
cut-tac X-mem-R,
frule subring-Ring[*of S*])
apply (*induct-tac m*)
apply (*rule allI*, *rule impI*, *simp*)
apply (*simp add*:*polyn-expr-def* [*of R X 0*])
apply (*frule-tac* $c = (0,g)$ **in** *pol-coeff-mem*[*of - 0*], *simp*, *simp*,

frule-tac $c = (0,g)$ **in** *pol-coeff-mem-R*[*of* - *0*], *simp*, *simp*)
apply (*simp add*:*ring-r-one*,
frule-tac $c = (n, f)$ **and** $k = n$ **in** *polyn-mem*, *simp*,
simp only:*ring-tOp-commute*[*of polyn-expr R X n* (n, f)],
subst scalar-times-pol-expr, *assumption*+, *simp*)
apply (*cut-tac* f = *sp-cf S* $(g\ 0)$ (n, f) **in** *pol-coeff-split*)
apply (*simp add*:*sp-cf-len*)
apply (*cut-tac* f = *sp-cf S* $(g\ 0)$ (n, f) **in** *polyn-expr-split*[*of n*],
simp only:*sp-cf-len*, *simp only*:*fst-conv*,
frule-tac $a = g\ 0$ **in** *sp-cf-pol-coeff*[*of* (n, f)], *assumption*+,
simp,
subgoal-tac snd (*sp-cf S* $(g\ 0)$ (n, f)) $n = (f\ n) \cdot_{r\,S} (g\ 0)$, *blast*)
apply (*thin-tac pol-coeff S* $(n,$ *snd* (*sp-cf S* $(g\ 0)$ (n, f))),
thin-tac polyn-expr R X n (*sp-cf S* $(g\ 0)$ (n, f)) =
polyn-expr R X n $(n,$ *snd* (*sp-cf S* $(g\ 0)$ (n, f))),
thin-tac pol-coeff S (*sp-cf S* $(g\ 0)$ (n, f)))
apply (*subst sp-cf-val*[*of* (n, f) *n*], *assumption*+, *simp*, *assumption*, *simp*,
frule-tac $c = (n,f)$ **in** *pol-coeff-mem*[*of* - *n*], *simp*, *simp*,
simp add:*Ring.ring-tOp-commute*)

apply (*rule allI*, *rule impI*)
apply (*frule-tac* $n = na$ **and** $f = g$ **in** *pol-coeff-pre*,
drule-tac $a = g$ **in** *forall-spec*, *assumption*+)
apply (*erule exE*, (*erule conjE*)+)
apply (*cut-tac* $n = na$ **and** $c = (Suc\ na,\ g)$ **in** *polyn-Suc*, (*simp del*:*npow-suc*)+,
thin-tac polyn-expr R X (*Suc na*) (*Suc na*, *g*) =
polyn-expr R X na (*Suc na*, *g*) $\pm$ *g* (*Suc na*) $\cdot_r$ $X\hat{}R$ (*Suc na*),
subst polyn-expr-restrict, *assumption*)
apply (*frule-tac* $c = (n, f)$ **and** $k = n$ **in** *polyn-mem*,*simp del*:*npow-suc*,
frule-tac $c = (na, g)$ **and** $k = na$ **in** *polyn-mem*, *simp del*:*npow-suc*,
frule-tac $c = (Suc\ na, g)$ **in** *monomial-mem*, *simp del*:*npow-suc*,
drule-tac $a = Suc\ na$ **in** *forall-spec*, *simp del*:*npow-suc*)
apply (*subst ring-distrib1*, *assumption*+)
apply (*rotate-tac 8*, *drule sym*,
simp del:*npow-suc*)
apply (*thin-tac polyn-expr R X n* $(n, f) \cdot_r$ *polyn-expr R X na* (na, g) =
polyn-expr R X $(n + na)$ $(n + na, h)$)
apply (*frule-tac* $c = (Suc\ na, g)$ **and** j =*Suc na* **in** *pol-coeff-mem-R*, *simp*,
simp del:*npow-suc*,
frule-tac $c = (Suc\ na, g)$ **and** j =*Suc na* **in** *pol-coeff-mem*, *simp*,
simp del:*npow-suc*)
apply (*subst ring-tOp-commute*, *assumption*+,
subst ring-tOp-assoc, *assumption*+, *rule npClose*, *assumption*+,
subst low-deg-terms-zero[*THEN sym*], *assumption*+)
apply (*frule-tac* $c = (n, f)$ **and** $n = Suc\ na$ **in** *ext-cf-pol-coeff*)
apply (*frule-tac* c = *ext-cf S* $(Suc\ na)$ (n, f) **and** $a = g\ (Suc\ na)$ **in**
sp-cf-pol-coeff, *assumption*)
apply (*subst scalar-times-pol-expr*, *assumption*+,
simp add:*ext-cf-len*,

cut-tac $k = n + Suc\ na$ **and**
$f = sp\text{-}cf\ S\ (g\ (Suc\ na))\ (ext\text{-}cf\ S\ (Suc\ na)\ (n, f))$ **in**
polyn-expr-split,
simp only:*sp-cf-len*,
thin-tac polyn-expr R X $(n + Suc\ na)$
$(sp\text{-}cf\ S\ (g\ (Suc\ na))\ (ext\text{-}cf\ S\ (Suc\ na)\ (n, f))) =$
polyn-expr R X $(n + Suc\ na)$
$(fst\ (ext\text{-}cf\ S\ (Suc\ na)\ (n, f)),$
$snd\ (sp\text{-}cf\ S\ (g\ (Suc\ na))\ (ext\text{-}cf\ S\ (Suc\ na)\ (n, f))))$,
simp only:*ext-cf-len*, *simp only*:*fst-conv*,
simp add:*add.commute*[*of* - n])
apply (*subst polyn-add*, *assumption*+,
cut-tac $f = sp\text{-}cf\ S\ (g\ (Suc\ na))\ (ext\text{-}cf\ S\ (Suc\ na)\ (n, f))$ **in**
pol-coeff-split, *simp only*:*sp-cf-len*, *simp only*:*ext-cf-len*,
simp add:*add.commute*[*of* - n], *simp add*: *max-def*,
frule-tac $c = sp\text{-}cf\ S\ (g\ (Suc\ na))\ (ext\text{-}cf\ S\ (Suc\ na)\ (n, f))$
in *pol-coeff-cartesian*,
simp only:*sp-cf-len*, *simp only*:*ext-cf-len*,
simp add:*add.commute*[*of* - n]],
thin-tac $(Suc\ (n + na)$,
$snd\ (sp\text{-}cf\ S\ (g\ (Suc\ na))\ (ext\text{-}cf\ S\ (Suc\ na)\ (n, f)))) =$
$sp\text{-}cf\ S\ (g\ (Suc\ na))\ (ext\text{-}cf\ S\ (Suc\ na)\ (n, f))$,
frule-tac $c = (n + na, h)$ **and**
$d = sp\text{-}cf\ S\ (g\ (Suc\ na))\ (ext\text{-}cf\ S\ (Suc\ na)\ (n, f))$ **in**
add-cf-pol-coeff, *assumption*)
apply (*cut-tac* $k = Suc\ (n + na)$ **and** $f = add\text{-}cf\ S\ (n + na, h)$
$(sp\text{-}cf\ S\ (g\ (Suc\ na))\ (ext\text{-}cf\ S\ (Suc\ na)\ (n, f)))$ **in** *polyn-expr-split*,
simp only:*mp*,
thin-tac polyn-expr R X $(Suc\ (n + na))$
$(add\text{-}cf\ S\ (n + na, h)$
$(sp\text{-}cf\ S\ (g\ (Suc\ na))\ (ext\text{-}cf\ S\ (Suc\ na)\ (n, f)))) =$
polyn-expr R X $(Suc\ (n + na))$
$(fst\ (add\text{-}cf\ S\ (n + na, h)$
$(sp\text{-}cf\ S\ (g\ (Suc\ na))\ (ext\text{-}cf\ S\ (Suc\ na)\ (n, f)))),$
$snd\ (add\text{-}cf\ S\ (n + na, h)$
$(sp\text{-}cf\ S\ (g\ (Suc\ na))\ (ext\text{-}cf\ S\ (Suc\ na)\ (n, f)))))$,
subst add-cf-len, *assumption*+,
simp add:*sp-cf-len*, *simp add*:*ext-cf-len max-def*,
cut-tac $f = add\text{-}cf\ S\ (n + na, h)$
$(sp\text{-}cf\ S\ (g\ (Suc\ na))\ (ext\text{-}cf\ S\ (Suc\ na)\ (n, f)))$ **in**
pol-coeff-split,
simp only:*add-cf-len*,
simp only:*sp-cf-len*, *simp add*:*ext-cf-len*, *simp add*:*max-def*,
thin-tac pol-coeff S
$(add\text{-}cf\ S\ (n + na, h)$
$(sp\text{-}cf\ S\ (g\ (Suc\ na))\ (ext\text{-}cf\ S\ (Suc\ na)\ (n, f)))),$
subgoal-tac snd $(add\text{-}cf\ S\ (n + na, h)\ (sp\text{-}cf\ S\ (g\ (Suc\ na))$
$(ext\text{-}cf\ S\ (Suc\ na)\ (n, f))))\ (Suc\ (n + na)) = f\ n\ \cdot_{r\,S}\ g\ (Suc\ na)$,
simp add:*add.commute*[*of* - n], *blast*)

apply (*subst add-cf-def*, *simp add*:*sp-cf-len ext-cf-len*,
subst sp-cf-def, *simp add*:*ext-cf-len*,
subst ext-cf-def, *simp add*:*sliden-def*,
frule pol-coeff-mem[*of* (*n*, *f*) *n*], *simp*,
simp add:*Ring.ring-tOp-commute*)
done

lemma (**in** *PolynRg*) *polyn-expr-tOp*:⟦
pol-coeff S (*n*, *f*); *pol-coeff S* (*m*, *g*)⟧ ⟹ ∃ *e*. *pol-coeff S* ((*n* + *m*), *e*) ∧
e (*n* + *m*) = (*f n*) ·$_{r\,S}$ (*g m*) ∧
polyn-expr R X (*n* + *m*)(*n* + *m*, *e*) =
(*polyn-expr R X n* (*n*, *f*)) ·$_r$ (*polyn-expr R X m* (*m*, *g*))
by (*simp add*:*polyn-expr-tOpTr*)

lemma (**in** *PolynRg*) *polyn-expr-tOp-c*:⟦*pol-coeff S c*; *pol-coeff S d*⟧ ⟹
∃ *e*. *pol-coeff S e* ∧ (*fst e* = *fst c* + *fst d*) ∧
(*snd e*) (*fst e*) = (*snd c* (*fst c*)) ·$_{r\,S}$ (*snd d*) (*fst d*) ∧
polyn-expr R X (*fst e*) *e* =
(*polyn-expr R X* (*fst c*) *c*) ·$_r$ (*polyn-expr R X* (*fst d*) *d*)
by (*cases c*, *cases d*) (*simp add*: *polyn-expr-tOpTr*)

4.14 The degree of a polynomial

lemma (**in** *PolynRg*) *polyn-degreeTr*:⟦*pol-coeff S c*; *k* ≤ (*fst c*)⟧ ⟹
(*polyn-expr R X k c* = **0**) = ({*j*. *j* ≤ *k* ∧ (*snd c*) *j* ≠ **0**$_S$} = {})
apply (*subst coeff-0-pol-0*[*THEN sym*, *of c k*], *assumption*+)
apply *blast*
done

lemma (**in** *PolynRg*) *higher-part-zero*:⟦*pol-coeff S c*; *k* < *fst c*;
∀ *j*∈*nset* (*Suc k*) (*fst c*). *snd c j* = **0**$_S$⟧ ⟹
Σ$_f$ *R* (λ*j*. *snd c j* ·$_r$ *X*^$^{R\ j}$) (*Suc k*) (*fst c*) = **0**
apply (*cut-tac ring-is-ag*,
rule aGroup.fSum-zero1[*of R k fst c* λ*j*. *snd c j* ·$_r$ *X*^$^{R\ j}$],
assumption+)
apply (*rule ballI*,
drule-tac x = *j* **in** *bspec*, *assumption*, *simp*)
apply (*cut-tac subring*,
simp add:*Subring-zero-ring-zero*,
cut-tac X-mem-R,
frule-tac n = *j* **in** *npClose*[*of X*],
simp add:*ring-times-0-x*)
done

lemma (**in** *PolynRg*) *coeff-nonzero-polyn-nonzero*:⟦*pol-coeff S c*; *k* ≤ (*fst c*)⟧
⟹ (*polyn-expr R X k c* ≠ **0**) = (∃ *j*≤*k*. (*snd c*) *j* ≠ **0**$_S$)
by (*simp add*:*coeff-0-pol-0*[*THEN sym*, *of c k*])

lemma (**in** *PolynRg*) *pol-expr-unique*:⟦$p \in$ *carrier* R; $p \neq \mathbf{0}$;
  pol-coeff S c; p = *polyn-expr* R X (*fst* c) c; (*snd* c) (*fst* c) $\neq \mathbf{0}_S$;
  pol-coeff S d; p = *polyn-expr* R X (*fst* d) d; (*snd* d) (*fst* d) $\neq \mathbf{0}_S$⟧ $\Longrightarrow$
  (*fst* c) = (*fst* d) $\wedge$ ($\forall j \leq$ (*fst* c). (*snd* c) j = (*snd* d) j)
apply (*cut-tac ring-is-ag*,
  cut-tac subring, *frule subring-Ring*,
  frule Ring.ring-is-ag[*of* S])
apply (*frule m-cf-pol-coeff*[*of* d])
apply (*frule polyn-minus-m-cf*[*of* d *fst* d], *simp*)
 apply (*drule sym*, *drule sym*, *simp*)
 apply (*rotate-tac* -2, *drule sym*, *drule sym*)
 apply (*frule-tac* $x = p$ **in** *aGroup.ag-r-inv1*[*of* R], *assumption*, *simp*,
   thin-tac p = *polyn-expr* R X (*fst* c) c,
   thin-tac polyn-expr R X (*fst* d) d = *polyn-expr* R X (*fst* c) c,
   thin-tac $-_a$ (*polyn-expr* R X (*fst* c) c) =
                polyn-expr R X (*fst* d) (*m-cf* S d))
 apply (*frule polyn-add1*[*of* c *m-cf* S d], *assumption*+, *simp add*:*m-cf-len*,
   thin-tac polyn-expr R X (*fst* c) $c \pm$ *polyn-expr* R X (*fst* d) (*m-cf* S d)
     = *polyn-expr* R X (*max* (*fst* c) (*fst* d)) (*add-cf* S c (*m-cf* S d)),
   thin-tac polyn-expr R X (*fst* c) $c \neq$
     polyn-expr R X (*max* (*fst* c) (*fst* d)) (*add-cf* S c (*m-cf* S d)))
 apply (*frule-tac add-cf-pol-coeff*[*of* c *m-cf* S d], *simp add*:*m-cf-len*)
 apply (*cut-tac coeff-0-pol-0*[*THEN sym*, *of add-cf* S c (*m-cf* S d)
          max (*fst* c) (*fst* d)],
   drule sym, *simp*,
   thin-tac polyn-expr R X (*max* (*fst* c) (*fst* d))
                (*add-cf* S c (*m-cf* S d)) = $\mathbf{0}$,
   thin-tac pol-coeff S (*add-cf* S c (*m-cf* S d)),
   thin-tac pol-coeff S (*m-cf* S d))

apply (*case-tac fst* c = *fst* d, *simp*)
  apply (*rule allI*, *rule impI*,
    drule-tac $a = j$ **in** *forall-spec*, *assumption*)
   apply (*simp add*:*add-cf-def m-cf-def m-cf-len*)
  apply (*frule-tac* $j = j$ **in** *pol-coeff-mem*[*of* c], *simp*,
    frule-tac $j = j$ **in** *pol-coeff-mem*[*of* d], *simp*)
 apply (*subst aGroup.ag-eq-diffzero*[*of* S], *assumption*+)

 apply (*simp add*:*add-cf-def*)
 apply (*case-tac* $\neg$ (*fst* c) $\leq$ (*fst* d), *simp*)
  apply (*simp add*:*m-cf-len*)
 apply (*drule-tac* a = *fst* c **in** *forall-spec*, *simp*, *simp*)

 apply *simp*
 apply (*drule-tac* a = *fst* d **in** *forall-spec*, *simp*, *simp add*:*m-cf-len*)
 apply (*case-tac fst* $c \neq$ *fst* d,
   frule noteq-le-less[*of fst* c *fst* d], *assumption*, *simp*)

```
      apply (simp add:m-cf-def)
      apply (frule pol-coeff-mem[of d fst d], simp)
      apply (frule Ring.ring-is-ag[of S],
            frule aGroup.ag-inv-inv[of S snd d (fst d)], assumption)
            apply (simp add:aGroup.ag-inv-zero)
 apply simp
 apply simp
 apply (simp add:add-cf-len m-cf-len)
done

lemma (in PolynRg) pol-expr-unique2:⟦pol-coeff S c; pol-coeff S d;
     fst c = fst d⟧ ⟹
  (polyn-expr R X (fst c) c = polyn-expr R X (fst d) d ) =
     (∀ j ≤ (fst c). (snd c) j = (snd d) j)
apply (cut-tac ring-is-ag,
       cut-tac subring, frule subring-Ring,
       frule Ring.ring-is-ag[of S])
apply (rule iffI)
apply (frule m-cf-pol-coeff[of d])
 apply (frule polyn-mem[of c fst c], simp,
        frule polyn-mem[of d fst d], simp)
 apply (frule aGroup.ag-eq-diffzero[of R polyn-expr R X (fst c) c
                  polyn-expr R X (fst d) d], assumption+,
        simp,
        simp only:polyn-minus-m-cf[of d fst d],
        drule sym, simp)
 apply (frule polyn-add1[of c m-cf S d], assumption+, simp add:m-cf-len)
 apply (thin-tac polyn-expr R X (fst c) d ± polyn-expr R X (fst c)
         (m-cf S d) =
         polyn-expr R X (fst c) (add-cf S c (m-cf S d)),
        thin-tac polyn-expr R X (fst c) c = polyn-expr R X (fst c) d,
        thin-tac polyn-expr R X (fst c) d ∈ carrier R,
        drule sym)
 apply (frule-tac add-cf-pol-coeff[of c m-cf S d], simp add:m-cf-len)
 apply (frule coeff-0-pol-0[THEN sym, of add-cf S c (m-cf S d)
               fst c],
        simp add:add-cf-len, simp add:m-cf-len,
        thin-tac 0 = polyn-expr R X (fst d) (add-cf S c (m-cf S d)),
        thin-tac pol-coeff S (add-cf S c (m-cf S d)))
 apply (simp add:add-cf-def m-cf-def)
  apply (rule allI, rule impI)
  apply (drule-tac a = j in forall-spec, assumption)
  apply (frule-tac j = j in pol-coeff-mem[of c], simp,
         frule-tac j = j in pol-coeff-mem[of d], simp)
  apply (simp add:aGroup.ag-eq-diffzero[THEN sym])

 apply simp
 apply (rule polyn-exprs-eq[of c d fst d], assumption+)
       apply (simp, assumption+)
```

done

lemma (**in** *PolynRg*) *pol-expr-unique3*:⟦*pol-coeff S c*; *pol-coeff S d*;
fst c < *fst d*⟧ ⟹
(*polyn-expr R X* (*fst c*) *c* = *polyn-expr R X* (*fst d*) *d*) =
((∀ *j* ≤ (*fst c*). (*snd c*) *j* = (*snd d*) *j*) ∧
(∀ *j*∈*nset* (*Suc* (*fst c*)) (*fst d*). (*snd d*) *j* = $\mathbf{0}_S$))
apply (*rule iffI*)
apply (*cut-tac ring-is-ag*,
cut-tac subring, *frule subring-Ring*,
frule Ring.ring-is-ag[*of S*])
apply (*frule m-cf-pol-coeff*[*of d*])
apply (*frule polyn-mem*[*of c fst c*], *simp*,
frule polyn-mem[*of d fst d*], *simp*)
apply (*frule aGroup.ag-eq-diffzero*[*of R polyn-expr R X* (*fst c*) *c*
polyn-expr R X (*fst d*) *d*], *assumption*+,
simp,
simp only:*polyn-minus-m-cf*[*of d fst d*],
drule sym, *simp*)
apply (*frule polyn-add1*[*of c m-cf S d*], *assumption*+, *simp add*:*m-cf-len*,
thin-tac polyn-expr R X (*fst c*) *c* ± *polyn-expr R X* (*fst d*)
(*m-cf S d*) =
polyn-expr R X (*max* (*fst c*) (*fst d*)) (*add-cf S c* (*m-cf S d*)),
thin-tac polyn-expr R X (*fst d*) *d* = *polyn-expr R X* (*fst c*) *c*,
thin-tac polyn-expr R X (*fst c*) *c* ∈ *carrier R*, *drule sym*)
apply (*frule-tac add-cf-pol-coeff*[*of c m-cf S d*], *simp add*:*m-cf-len*)
apply (*frule coeff-0-pol-0*[*THEN sym*, *of add-cf S c* (*m-cf S d*)
max (*fst c*) (*fst d*)],
simp add:*add-cf-len m-cf-len*, *simp*,
thin-tac polyn-expr R X (*max* (*fst c*) (*fst d*))
(*add-cf S c* (*m-cf S d*)) = **0**,
thin-tac pol-coeff S (*add-cf S c* (*m-cf S d*)))
apply (*simp add*:*add-cf-def m-cf-def max-def*)
apply (*rule conjI*)
apply (*rule allI*, *rule impI*,
frule-tac x = *j* **and** *y* = *fst c* **and** *z* = *fst d* **in** *le-less-trans*,
assumption+,
frule-tac x = *j* **and** *y* = *fst d* **in** *less-imp-le*)
apply (*drule-tac a* = *j* **in** *forall-spec*, *simp*, *simp*)
apply (*frule-tac j* = *j* **in** *pol-coeff-mem*[*of c*], *simp*,
frule-tac j = *j* **in** *pol-coeff-mem*[*of d*], *simp*)
apply (*simp add*:*aGroup.ag-eq-diffzero*[*THEN sym*])

apply (*rule ballI*, *simp add*:*nset-def*, *erule conjE*)
apply (*cut-tac x* = *fst c* **and** *y* = *Suc* (*fst c*) **and** *z* = *j* **in**
less-le-trans, *simp*, *assumption*)
apply (*cut-tac m1* = *fst c* **and** *n1* = *j* **in** *nat-not-le-less*[*THEN sym*], *simp*)
apply (*drule-tac a* = *j* **in** *forall-spec*, *assumption*, *simp*,
frule-tac j = *j* **in** *pol-coeff-mem*[*of d*], *simp*)

apply (*frule-tac x = snd d j* **in** *aGroup.ag-inv-inv*[*of S*], *assumption*,
simp add:*aGroup.ag-inv-inv aGroup.ag-inv-zero*)
apply (*cut-tac polyn-n-m*[*of d fst c fst d*])
apply (*subst polyn-expr-split*[*of fst d d*], *simp*,
thin-tac polyn-expr R X (*fst d*) *d* =
polyn-expr R X (*fst c*) (*fst c*, *snd d*) ±
Σ_f *R* (λ*j*. *snd d j* ·$_r$ *X*^$^{R\ j}$) (*Suc* (*fst c*)) (*fst d*), *erule conjE*)
apply (*subst higher-part-zero*[*of d fst c*], *assumption*+)
apply (*frule pol-coeff-le*[*of d fst c*], *simp add*:*less-imp-le*,
frule polyn-mem[*of* (*fst c*, *snd d*) *fst c*], *simp*,
cut-tac ring-is-ag,
simp add:*aGroup.ag-r-zero*,
subst polyn-expr-short[*THEN sym*, *of d fst c*], *assumption*+,
simp add:*less-imp-le*)
apply (*rule polyn-exprs-eq*[*of c d fst c*], *assumption*+)
apply (*simp*, *assumption*+)
apply (*simp add*:*less-imp-le*)
done

lemma (**in** *PolynRg*) *polyn-degree-unique*:⟦*pol-coeff S c*; *pol-coeff S d*;
polyn-expr R X (*fst c*) *c* = *polyn-expr R X* (*fst d*) *d*⟧ ⟹
c-max S c = *c-max S d*
apply (*cut-tac ring-is-ag*,
cut-tac subring,
frule subring-Ring,
frule Ring.ring-is-ag[*of S*])

apply (*case-tac polyn-expr R X* (*fst d*) *d* = $\mathbf{0}_R$)
apply (*cut-tac coeff-0-pol-0*[*THEN sym*, *of d fst d*], *simp*,
cut-tac coeff-0-pol-0[*THEN sym*, *of c fst c*], *simp*)
apply (*simp add*:*c-max-def*, *assumption*, *simp*, *assumption*, *simp*)

apply (*frule polyn-mem*[*of c fst c*], *simp*, *frule polyn-mem*[*of d fst d*],
simp)
apply (*frule aGroup.ag-eq-diffzero*[*of R polyn-expr R X* (*fst c*) *c*
polyn-expr R X (*fst d*) *d*], *assumption*+)
apply (*simp only*:*polyn-minus-m-cf*[*of d fst d*],
frule m-cf-pol-coeff [*of d*])
apply (*frule polyn-add1*[*of c m-cf S d*], *assumption*+,
simp only:*m-cf-len*)
apply (*rotate-tac* −*1*, *drule sym*, *simp*,
thin-tac polyn-expr R X (*fst d*) *d* ±
polyn-expr R X (*fst d*) (*m-cf S d*) = **0**,
frule add-cf-pol-coeff[*of c m-cf S d*], *assumption*+)
apply (*cut-tac coeff-0-pol-0*[*THEN sym*, *of add-cf S c* (*m-cf S d*)
fst (*add-cf S c* (*m-cf S d*))],
simp add:*add-cf-len m-cf-len*,
thin-tac polyn-expr R X (*max* (*fst c*) (*fst d*))
(*add-cf S c* (*m-cf S d*)) = **0**,

thin-tac pol-coeff S (add-cf S c (m-cf S d)),
thin-tac pol-coeff S (m-cf S d))
apply (*frule coeff-nonzero-polyn-nonzero*[*of d fst d*], *simp*, *simp*)
apply (*drule sym*, *simp*)
apply (*frule coeff-nonzero-polyn-nonzero*[*of c fst c*], *simp*, *simp*)
apply (*simp add*:*c-max-def*, *rule conjI*, *rule impI*, *blast*,
rule conjI, *rule impI*, *blast*)
apply (*rule n-max-eq-sets*)
apply (*rule equalityI*)
apply (*rule subsetI*, *simp*)
apply (*erule conjE*)

apply (*case-tac fst c* ≤ *fst d*)
apply (*frule-tac i* = *x* **in** *le-trans*[*of - fst c fst d*], *assumption*+, *simp*,
rule contrapos-pp, *simp*+, *simp add*:*max-def*,
frule-tac i = *x* **in** *le-trans*[*of - fst c fst d*], *assumption*+,
drule-tac a = *x* **in** *forall-spec*, *assumption*,
drule le-imp-less-or-eq[*of fst c fst d*],
erule disjE, *simp add*:*add-cf-def m-cf-len m-cf-def*,
frule-tac j = *x* **in** *pol-coeff-mem*[*of c*], *assumption*+,
simp add:*aGroup.ag-inv-zero aGroup.ag-r-zero*[*of S*])

apply (*simp add*:*add-cf-def m-cf-len m-cf-def*,
rotate-tac −*1*, *drule sym*, *simp*,
frule-tac j = *x* **in** *pol-coeff-mem*[*of c*], *simp*,
simp add:*aGroup.ag-inv-zero aGroup.ag-r-zero*[*of S*])

apply (*simp add*:*nat-not-le-less*)
apply (*case-tac* ¬ *x* ≤ (*fst d*), *simp*,
simp add:*nat-not-le-less*,
frule-tac x = *fst d* **and** *y* = *x* **and** *z* = *fst c* **in**
less-le-trans, *assumption*+,
drule-tac x = *x* **in** *spec*, *simp add*:*max-def*,
simp add:*add-cf-def m-cf-len m-cf-def*)

apply (*simp*,
drule-tac x = *x* **in** *spec*, *simp add*:*max-def*,
rule contrapos-pp, *simp*+,
simp add:*add-cf-def m-cf-len m-cf-def*,
frule-tac j = *x* **in** *pol-coeff-mem*[*of c*],
frule-tac x = *x* **and** *y* = *fst d* **and** *z* = *fst c* **in**
le-less-trans, *assumption*+, *simp add*:*less-imp-le*,
simp add:*aGroup.ag-inv-zero aGroup.ag-r-zero*[*of S*])

apply (*rule subsetI*, *simp*, *erule conjE*,
case-tac fst d ≤ *fst c*,
frule-tac i = *x* **and** *j* = *fst d* **and** *k* = *fst c* **in** *le-trans*,
assumption+, *simp*,
drule-tac x = *x* **in** *spec*, *simp add*:*max-def*,

rule contrapos-pp, *simp*+,
simp add:*add-cf-def m-cf-len m-cf-def*)
apply (*case-tac fst d* = *fst c*, *simp*, *rotate-tac* −*1*, *drule sym*, *simp*,
frule-tac j = *x* **in** *pol-coeff-mem*[*of d*], *assumption*,
frule-tac x = *snd d x* **in** *aGroup.ag-mOp-closed*, *assumption*+,
simp add:*aGroup.ag-l-zero*,
frule-tac x = *snd d x* **in** *aGroup.ag-inv-inv*[*of S*],
assumption, *simp add*:*aGroup.ag-inv-zero*)

apply (*drule noteq-le-less*[*of fst d fst c*], *assumption*,
simp,
frule-tac j = *x* **in** *pol-coeff-mem*[*of d*], *assumption*,
frule-tac x = *snd d x* **in** *aGroup.ag-mOp-closed*, *assumption*+,
simp add:*aGroup.ag-l-zero*,
frule-tac x = *snd d x* **in** *aGroup.ag-inv-inv*[*of S*],
assumption, *simp add*:*aGroup.ag-inv-zero*)

apply (*simp add*:*nat-not-le-less*,
case-tac ¬ *x* ≤ *fst c*, *simp*,
simp add:*nat-not-le-less*,
drule-tac x = *x* **in** *spec*, *simp add*:*max-def*,
simp add:*add-cf-def m-cf-len m-cf-def*,
frule-tac j = *x* **in** *pol-coeff-mem*[*of d*], *assumption*,
frule-tac x = *snd d x* **in** *aGroup.ag-mOp-closed*, *assumption*+,
simp add:*aGroup.ag-l-zero*,
frule-tac x = *snd d x* **in** *aGroup.ag-inv-inv*[*of S*],
assumption, *simp add*:*aGroup.ag-inv-zero*)

apply (*simp*,
drule-tac x = *x* **in** *spec*, *simp add*:*max-def*,
rule contrapos-pp, *simp*+,
simp add:*add-cf-def m-cf-len m-cf-def*,
frule-tac x = *x* **and** *y* = *fst c* **and** *z* = *fst d* **in** *le-less-trans*,
assumption+, *frule-tac x* = *x* **and** *y* = *fst d* **in** *less-imp-le*,
frule-tac j = *x* **in** *pol-coeff-mem*[*of d*], *assumption*,
frule-tac x = *snd d x* **in** *aGroup.ag-mOp-closed*, *assumption*+,
simp add:*aGroup.ag-l-zero*,
frule-tac x = *snd d x* **in** *aGroup.ag-inv-inv*[*of S*],
assumption, *simp add*:*aGroup.ag-inv-zero*)

apply (*thin-tac* ∀ *j*≤*max* (*fst c*) (*fst d*). *snd* (*add-cf S c* (*m-cf S d*)) *j* = $\mathbf{0}_S$)
apply (*rotate-tac* −*1*, *drule sym*, *simp*)
apply (*simp add*:*coeff-0-pol-0*[*THEN sym*, *of c fst c*])
apply *blast*
apply *simp*+
done

lemma (**in** *PolynRg*) *ex-polyn-expr*:*p* ∈ *carrier R* ⟹
∃ *c*. *pol-coeff S c* ∧ *p* = *polyn-expr R X* (*fst c*) *c*

apply (*cut-tac S-X-generate*[*of p*], *blast*)
apply *assumption*
done

lemma (**in** *PolynRg*) *c-max-eqTr0*:⟦*pol-coeff S c*; $k \leq$ (*fst c*);
polyn-expr R X k c = *polyn-expr R X* (*fst c*) *c*; $\exists j \leq k$. (*snd c*) $j \neq \mathbf{0}_S$⟧ ⟹
c-max S (*k*, *snd c*) = *c-max S c*
apply (*simp add*:*polyn-expr-short*[*of c k*],
frule pol-coeff-le[*of c k*], *assumption*+,
rule polyn-degree-unique[*of* (*k*, *snd c*) *c*], *assumption*+,
simp)
done

definition
cf-sol :: [('*a*, '*b*) *Ring-scheme*, ('*a*, '*b1*) *Ring-scheme*, '*a*, '*a*,
nat × (*nat* ⇒ '*a*)] ⇒ *bool* **where**
cf-sol R S X p c ⟷ *pol-coeff S c* ∧ (*p* = *polyn-expr R X* (*fst c*) *c*)

definition
deg-n ::[('*a*, '*b*) *Ring-scheme*, ('*a*, '*b1*) *Ring-scheme*, '*a*, '*a*] ⇒ *nat* **where**
deg-n R S X p = *c-max S* (*SOME c*. *cf-sol R S X p c*)

definition
deg ::[('*a*, '*b*) *Ring-scheme*, ('*a*, '*b1*) *Ring-scheme*, '*a*, '*a*] ⇒ *ant* **where**
deg R S X p = (*if p* = $\mathbf{0}_R$ *then* $-\infty$ *else* (*an* (*deg-n R S X p*)))

lemma (**in** *PolynRg*) *ex-cf-sol*:*p* ∈ *carrier R* ⟹
∃ *c*. *cf-sol R S X p c*
apply (*unfold cf-sol-def*)
apply (*frule ex-polyn-expr*[*of p*], (*erule exE*)+)
apply (*cut-tac n* = *fst c* **in** *le-refl*, *blast*)
done

lemma (**in** *PolynRg*) *deg-in-aug-minf*:*p* ∈ *carrier R* ⟹
deg R S X p ∈ $Z_{-\infty}$
apply (*simp add*:*aug-minf-def deg-def an-def*)
done

lemma (**in** *PolynRg*) *deg-noninf*:*p* ∈ *carrier R* ⟹
deg R S X p ≠ ∞
apply (*cut-tac deg-in-aug-minf*[*of p*], *simp add*:*deg-def*,
simp add:*aug-minf-def*)
apply (*case-tac p* = $\mathbf{0}_R$, *simp*+)
done

lemma (**in** *PolynRg*) *deg-ant-int*:⟦*p* ∈ *carrier R*; *p* ≠ **0**⟧
⟹ *deg R S X p* = *ant* (*int* (*deg-n R S X p*))
by (*simp add*:*deg-def an-def*)

lemma (**in** *PolynRg*) *deg-an*:⟦$p \in$ *carrier* R; $p \neq \mathbf{0}$⟧
$\Longrightarrow$ *deg* R S X p = *an* (*deg-n* R S X p)
by (*simp add*:*deg-def*)

lemma (**in** *PolynRg*) *pol-SOME-1*:$p \in$ *carrier* R $\Longrightarrow$
cf-sol R S X p (*SOME* f. *cf-sol* R S X p f)
apply (*frule ex-cf-sol*[*of p*])
apply (*rule-tac* P = *cf-sol* R S X p **in** *someI-ex*, *assumption*)
done

lemma (**in** *PolynRg*) *pol-SOME-2*:$p \in$ *carrier* R $\Longrightarrow$
pol-coeff S (*SOME* c. *cf-sol* R S X p c) $\wedge$
p = *polyn-expr* R X (*fst* (*SOME* c. *cf-sol* R S X p c))
(*SOME* c. *cf-sol* R S X p c)
apply (*frule pol-SOME-1*[*of p*])
apply (*simp add*:*cf-sol-def*)
done

lemma (**in** *PolynRg*) *coeff-max-zeroTr*:*pol-coeff* S c $\Longrightarrow$
$\forall j.\ j \leq$ (*fst* c) $\wedge$ (*c-max* S c) $< j \longrightarrow$ (*snd* c) $j = \mathbf{0}_S$
apply (*case-tac* $\forall j \leq$ (*fst* c). (*snd* c) $j = \mathbf{0}_S$, *rule allI*, *rule impI*,
erule conjE, *simp*)
apply *simp*
apply (*frule coeff-nonzero-polyn-nonzero*[*THEN sym*, *of c fst c*], *simp*,
simp)
apply (*rule allI*, *rule impI*, *erule conjE*,
simp add:*c-max-def*,
simp add:*polyn-degreeTr*[*of c fst c*])
apply (*subgoal-tac* $\{j.\ j \leq$ (*fst* c) $\wedge$ (*snd* c) $j \neq \mathbf{0}_S\} \subseteq \{j.\ j \leq$ (*fst* c)$\}$,
frule n-max[*of* $\{j.\ j \leq$ (*fst* c) $\wedge$ (*snd* c) $j \neq \mathbf{0}_S\}$ *fst c*], *blast*)
apply (*case-tac* $\forall x \leq$*fst c*. *snd c* $x = \mathbf{0}_S$, *blast*, *simp*)
apply (*erule conjE*)
apply (*rule contrapos-pp*, *simp*+,
thin-tac $\exists x \leq$*fst c*. *snd c* $x \neq \mathbf{0}_S$,
thin-tac $\{j.\ j \leq$ *fst c* $\wedge$ *snd c* $j \neq \mathbf{0}_S\} \subseteq \{j.\ j \leq$ *fst c*$\}$,
thin-tac snd c (*n-max* $\{j.\ j \leq$ *fst c* $\wedge$ *snd c* $j \neq \mathbf{0}_S\}$) $\neq \mathbf{0}_S$,
drule-tac $a = j$ **in** *forall-spec*, *simp*)
apply *simp*
apply (*rule subsetI*, *simp*)
done

lemma (**in** *PolynRg*) *coeff-max-nonzeroTr*:⟦*pol-coeff* S c;
$\exists j \leq$ (*fst* c). (*snd* c) $j \neq \mathbf{0}_S$⟧ $\Longrightarrow$ (*snd* c) (*c-max* S c) $\neq \mathbf{0}_S$
apply (*simp add*:*c-max-def*)
apply (*subgoal-tac* $\{j.\ j \leq$ (*fst* c) $\wedge$ (*snd* c) $j \neq \mathbf{0}_S\} \subseteq \{j.\ j \leq$ (*fst* c)$\}$,
frule n-max[*of* $\{j.\ j \leq$ (*fst* c) $\wedge$ (*snd* c) $j \neq \mathbf{0}_S\}$ *fst c*], *blast*)
apply (*erule conjE*, *simp*)

apply (*rule subsetI*, *simp*)

done

lemma (**in** *PolynRg*) *coeff-max-bddTr*:*pol-coeff S c* $\Longrightarrow$ *c-max S c* $\leq$ (*fst c*)
apply (*case-tac* $\forall j{\leq}$(*fst c*). (*snd c*) *j* = $\mathbf{0}_S$, *simp add*:*c-max-def*)
apply (*simp add*:*c-max-def*,
  frule polyn-degreeTr[*of c fst c*], *simp*, *simp*,
  subgoal-tac {*j*. *j* $\leq$ (*fst c*) $\wedge$ (*snd c*) *j* $\neq$ $\mathbf{0}_S$} $\subseteq$ {*j*. *j* $\leq$ (*fst c*)},
  frule n-max[*of* {*j*. *j* $\leq$ (*fst c*) $\wedge$ (*snd c*) *j* $\neq$ $\mathbf{0}_S$} *fst c*],
  blast, *erule conjE*, *simp*)
apply (*rule subsetI*, *simp*)
done

lemma (**in** *PolynRg*) *pol-coeff-max*:*pol-coeff S c* $\Longrightarrow$
  pol-coeff S ((*c-max S c*), *snd c*)
apply (*rule pol-coeff-le*[*of c c-max S c*], *assumption*)
apply (*simp add*:*coeff-max-bddTr*)
done

lemma (**in** *PolynRg*) *polyn-c-max*:*pol-coeff S c* $\Longrightarrow$
  polyn-expr R X (*fst c*) *c* = *polyn-expr R X* (*c-max S c*) *c*
apply (*case-tac* (*c-max S c*) = (*fst c*), *simp*)
apply (*frule coeff-max-bddTr*[*of c*],
  frule noteq-le-less[*of c-max S c fst c*], *assumption*)
apply (*subst polyn-n-m1*[*of c c-max S c fst c*], *assumption*+, *simp*)

apply (*frule-tac polyn-mem*[*of c c-max S c*], *assumption*+)
apply (*subst higher-part-zero*[*of c c-max S c*], *assumption*+)
apply (*frule coeff-max-zeroTr*[*of c*],
  rule ballI, *simp add*:*nset-def*)

apply (*cut-tac ring-is-ag*, *simp add*:*aGroup.ag-r-zero*)
done

lemma (**in** *PolynRg*) *pol-deg-eq-c-max*:⟦*p* $\in$ *carrier R*;
  pol-coeff S c; *p* = *polyn-expr R X* (*fst c*) *c*⟧ $\Longrightarrow$
    deg-n R S X p = *c-max S c*
apply (*cut-tac subring*, *frule subring-Ring*)
apply (*frule polyn-c-max*[*of c*])
apply (*frule pol-SOME-2*[*of p*])
apply (*subst deg-n-def*, *erule conjE*)
apply (*rule polyn-degree-unique*[*of Eps* (*cf-sol R S X p*) *c*], *simp*,
  assumption)
apply *simp*
done

lemma (**in** *PolynRg*) *pol-deg-le-n*:⟦*p* $\in$ *carrier R*; *pol-coeff S c*;
  p = *polyn-expr R X* (*fst c*) *c*⟧ $\Longrightarrow$ *deg-n R S X p* $\leq$ (*fst c*)
apply (*frule pol-deg-eq-c-max*[*of p c*], *assumption*+,
  frule coeff-max-bddTr[*of c*])

apply *simp*
done

lemma (**in** *PolynRg*) *pol-deg-le-n1*:⟦$p \in$ *carrier R*; *pol-coeff S c*; $k \leq$ (*fst c*);
p = *polyn-expr R X k c*⟧ $\Longrightarrow$ *deg-n R S X p* $\leq k$
apply (*simp add*:*deg-n-def*, *drule sym*, *simp*)
apply (*frule pol-SOME-2*[*of p*], *erule conjE*)
apply (*frule pol-coeff-le*[*of c k*], *assumption*)
apply (*simp only*:*polyn-expr-short*[*of c k*])
apply (*drule sym*)
apply (*subst polyn-degree-unique*[*of SOME c. cf-sol R S X p c* (*k*, *snd c*)],
assumption+, *simp*)
apply (*frule coeff-max-bddTr*[*of* (*k*, *snd c*)], *simp*)
done

lemma (**in** *PolynRg*) *pol-len-gt-deg*:⟦$p \in$ *carrier R*; *pol-coeff S c*;
p = *polyn-expr R X* (*fst c*) *c*; *deg R S X p* $<$ (*an j*); $j \leq$ (*fst c*)⟧
$\Longrightarrow$ (*snd c*) $j = \mathbf{0}_S$
apply (*case-tac* $p = \mathbf{0}_R$, *simp*, *drule sym*)
apply (*simp add*:*coeff-0-pol-0*[*THEN sym*, *of c fst c*])
apply (*simp add*:*deg-def*, *simp add*:*aless-natless*)
apply (*drule sym*, *simp*)
apply (*frule coeff-max-zeroTr*[*of c*])
apply (*simp add*:*pol-deg-eq-c-max*)
done

lemma (**in** *PolynRg*) *pol-diff-deg-less*:⟦$p \in$ *carrier R*; *pol-coeff S c*;
p = *polyn-expr R X* (*fst c*) *c*; *pol-coeff S d*;
fst c = *fst d*; (*snd c*) (*fst c*) = (*snd d*) (*fst d*)⟧ $\Longrightarrow$
$p \pm (-_a$ (*polyn-expr R X* (*fst d*) *d*)) $= \mathbf{0} \vee$
deg-n R S X ($p \pm (-_a$ (*polyn-expr R X* (*fst d*) *d*))) $<$ (*fst c*)
apply (*cut-tac ring-is-ag*,
cut-tac subring, *frule subring-Ring*)
apply (*case-tac* $p \pm_R (-_a{}_R$ (*polyn-expr R X* (*fst d*) *d*)) $= \mathbf{0}_R$, *simp*)
apply *simp*
apply (*simp add*:*polyn-minus-m-cf*[*of d fst d*],
frule m-cf-pol-coeff[*of d*])
apply (*cut-tac polyn-add1*[*of c m-cf S d*], *simp add*:*m-cf-len*,
thin-tac polyn-expr R X (*fst d*) *c* $\pm$ *polyn-expr R X* (*fst d*) (*m-cf S d*)
= *polyn-expr R X* (*fst d*) (*add-cf S c* (*m-cf S d*)))
apply (*frule add-cf-pol-coeff*[*of c m-cf S d*], *assumption*+)
apply (*cut-tac polyn-mem*[*of add-cf S c* (*m-cf S d*) *fst d*],
frule pol-deg-le-n[*of polyn-expr R X* (*fst d*) (*add-cf S c* (*m-cf S d*))
add-cf S c (*m-cf S d*)], *assumption*+,
simp add:*add-cf-len m-cf-len*,
simp add:*add-cf-len m-cf-len*)
apply (*rule noteq-le-less*[*of deg-n R S X* (*polyn-expr R X* (*fst d*)
(*add-cf S c* (*m-cf S d*))) *fst d*], *assumption*)
apply (*rule contrapos-pp*, *simp*+)

apply (*cut-tac pol-deg-eq-c-max*[*of polyn-expr R X* (*fst d*)
(*add-cf S c* (*m-cf S d*)) *add-cf S c* (*m-cf S d*)],
simp,
thin-tac deg-n R S X (*polyn-expr R X* (*fst d*) (*add-cf S c* (*m-cf S d*)))
= *fst d*)
apply (*frule coeff-nonzero-polyn-nonzero*[*of add-cf S c* (*m-cf S d*) *fst d*],
simp add:*add-cf-len m-cf-len*, *simp*,
thin-tac polyn-expr R X (*fst d*) (*add-cf S c* (*m-cf S d*)) ≠ **0**,
frule coeff-max-nonzeroTr[*of add-cf S c* (*m-cf S d*)],
simp add:*add-cf-len m-cf-len*,
thin-tac ∃ *j*≤*fst d*. *snd* (*add-cf S c* (*m-cf S d*)) *j* ≠ **0**$_S$,
thin-tac pol-coeff S (*m-cf S d*),
thin-tac pol-coeff S (*add-cf S c* (*m-cf S d*)),
thin-tac polyn-expr R X (*fst d*) (*add-cf S c* (*m-cf S d*)) ∈
carrier R, *simp*,
thin-tac c-max S (*add-cf S c* (*m-cf S d*)) = *fst d*)
apply (*simp add*:*add-cf-def m-cf-def*,
frule pol-coeff-mem[*of d fst d*], *simp*,
frule Ring.ring-is-ag[*of S*],
simp add:*aGroup.ag-r-inv1*, *assumption*+,
simp add:*add-cf-len m-cf-len*, *assumption*,
simp add:*add-cf-len m-cf-len*, *assumption*+)
done

lemma (**in** *PolynRg*) *pol-pre-lt-deg*:⟦*p* ∈ *carrier R*; *pol-coeff S c*;
deg-n R S X p ≤ (*fst c*); (*deg-n R S X p*) ≠ *0*;
p = *polyn-expr R X* (*deg-n R S X p*) *c* ⟧ ⟹
(*deg-n R S X* (*polyn-expr R X* ((*deg-n R S X p*) − *Suc 0*) *c*)) < (*deg-n R S X p*)
apply (*frule polyn-expr-short*[*of c deg-n R S X p*], *assumption*)
apply (*cut-tac pol-deg-le-n*[*of polyn-expr R X* (*deg-n R S X p* − *Suc 0*) *c*
(*deg-n R S X p* − *Suc 0*, *snd c*)], *simp*)
apply (*rule polyn-mem*[*of c deg-n R S X p* − *Suc 0*], *assumption*+,
arith,
rule pol-coeff-le[*of c deg-n R S X p* − *Suc 0*], *assumption*,
arith, *simp*)
apply (*subst polyn-expr-short*[*of c deg-n R S X p* − *Suc 0*],
assumption+, *arith*, *simp*)
done

lemma (**in** *PolynRg*) *pol-deg-n*:⟦*p* ∈ *carrier R*; *pol-coeff S c*;
n ≤ *fst c*; *p* = *polyn-expr R X n c*; (*snd c*) *n* ≠ **0**$_S$⟧ ⟹
deg-n R S X p = *n*
apply (*simp add*:*polyn-expr-short*[*of c n*])
apply (*frule pol-coeff-le*[*of c n*], *assumption*+,
cut-tac pol-deg-eq-c-max[*of p* (*n*, *snd c*)],
drule sym, *simp*, *simp add*:*c-max-def*)
apply (*rule conjI*, *rule impI*, *cut-tac le-refl*[*of n*],
thin-tac deg-n R S X p =
(*if* ∀ *x*≤*n*. *snd c x* = **0**$_S$ *then 0*

else n-max $\{j.\ j \leq fst\ (n,\ snd\ c) \wedge snd\ (n,\ snd\ c)\ j \neq \mathbf{0}_S\}$),
drule-tac $a = n$ **in** *forall-spec*, *assumption*, *simp*)
apply (*rule impI*)
apply (*cut-tac n-max*[*of* $\{j.\ j \leq n \wedge snd\ c\ j \neq \mathbf{0}_S\}$ n], *erule conjE*,
drule-tac $x = n$ **in** *bspec*, *simp*, *simp*)
apply (*rule subsetI*, *simp*, *blast*,
drule sym, *simp*, *assumption*)
apply *simp*
done

lemma (**in** *PolynRg*) *pol-expr-deg*:$[\![p \in carrier\ R;\ p \neq \mathbf{0}]\!]$
$\Longrightarrow \exists c.\ pol\text{-}coeff\ S\ c \wedge deg\text{-}n\ R\ S\ X\ p \leq (fst\ c) \wedge$
$p = polyn\text{-}expr\ R\ X\ (deg\text{-}n\ R\ S\ X\ p)\ c \wedge$
$(snd\ c)\ (deg\text{-}n\ R\ S\ X\ p) \neq \mathbf{0}_S$
apply (*cut-tac subring*,
frule subring-Ring)
apply (*frule ex-polyn-expr*[*of p*], *erule exE*, *erule conjE*)
apply (*frule-tac* $c = c$ **in** *polyn-c-max*)
apply (*frule-tac* $c = c$ **in** *pol-deg-le-n*[*of p*], *assumption*+)
apply (*frule-tac* $c1 = c$ **and** $k1 = fst\ c$ **in** *coeff-0-pol-0*[*THEN sym*], *simp*)
apply (*subgoal-tac* $p = polyn\text{-}expr\ R\ X\ (deg\text{-}n\ R\ S\ X\ p)\ c \wedge$
$snd\ c\ (deg\text{-}n\ R\ S\ X\ p) \neq \mathbf{0}_S$, *blast*)
apply (*subst pol-deg-eq-c-max*, *assumption*+)+
apply *simp*
apply (*cut-tac* $c = c$ **in** *coeff-max-nonzeroTr*, *simp*+)
done

lemma (**in** *PolynRg*) *deg-n-pos*:$p \in carrier\ R \Longrightarrow 0 \leq deg\text{-}n\ R\ S\ X\ p$
by *simp*

lemma (**in** *PolynRg*) *pol-expr-deg1*:$[\![p \in carrier\ R;\ d = na\ (deg\ R\ S\ X\ p)]\!] \Longrightarrow$
$\exists c.\ (pol\text{-}coeff\ S\ c \wedge p = polyn\text{-}expr\ R\ X\ d\ c)$
apply (*cut-tac subring*, *frule subring-Ring*)
apply (*case-tac* $p = \mathbf{0}_R$,
simp add:*deg-def na-minf*,
subgoal-tac pol-coeff S $(0,\ (\lambda j.\ \mathbf{0}_S))$,
subgoal-tac $\mathbf{0} = polyn\text{-}expr\ R\ X\ d\ (0,\ (\lambda j.\ \mathbf{0}_S))$, *blast*,
cut-tac coeff-0-pol-0[*of* $(d,\ \lambda j.\ \mathbf{0}_S)$ d], *simp*+,
simp add:*pol-coeff-def*,
simp add:*Ring.ring-zero*)

apply (*simp add*:*deg-def na-an*,
frule pol-expr-deg[*of p*], *assumption*,
erule exE, (*erule conjE*)+,
unfold split-paired-all, *simp*, *blast*)
done

end

theory *Algebra6* **imports** *Algebra5* **begin**

definition
s-cf :: [('*a*, '*m*) *Ring-scheme*, ('*a*, '*m1*) *Ring-scheme*, '*a*, '*a*]
⇒ *nat* × (*nat* ⇒ '*a*) **where**
s-cf R S X p = (*if p* = $\mathbf{0}_R$ *then* (*0*, λ*j*. $\mathbf{0}_S$) *else*
SOME c. (*pol-coeff S c* ∧ *p* = *polyn-expr R X* (*fst c*) *c* ∧
(*snd c*) (*fst c*) ≠ $\mathbf{0}_S$))

definition
lcf :: [('*a*, '*m*) *Ring-scheme*, ('*a*, '*m1*) *Ring-scheme*, '*a*, '*a*] ⇒ '*a* **where**
lcf R S X p = (*snd* (*s-cf R S X p*)) (*fst* (*s-cf R S X p*))

lemma (**in** *PolynRg*) *lcf-val-0*:*lcf R S X* **0** = $\mathbf{0}_S$
by (*simp add*:*lcf-def s-cf-def*)

lemma (**in** *PolynRg*) *lcf-val*:⟦*p* ∈ *carrier R*; *p* ≠ **0** ⟧ ⟹
lcf R S X p = (*snd* (*s-cf R S X p*)) (*fst* (*s-cf R S X p*))
by (*simp add*:*lcf-def*)

lemma (**in** *PolynRg*) *s-cf-pol-coeff*:*p* ∈ *carrier R* ⟹
pol-coeff S (*s-cf R S X p*)
apply (*simp add*:*s-cf-def*)
apply (*case-tac p* = $\mathbf{0}_R$, *simp*)
apply (*cut-tac subring*, *frule subring-Ring*,
simp add:*pol-coeff-def Ring.ring-zero*)
apply *simp*
apply (*rule someI2-ex*)
apply (*frule ex-polyn-expr*[*of p*], *erule exE*, *erule conjE*)
apply (*frule-tac c* = *c* **in** *coeff-max-bddTr*)
apply (*frule-tac c* = *c* **and** *n* = *c-max S c* **in** *pol-coeff-le*, *assumption*)
apply (*subgoal-tac p* = *polyn-expr R X* (*fst* (*c-max S c*, *snd c*))
(*c-max S c*, *snd c*) ∧
snd (*c-max S c*, *snd c*) (*fst* (*c-max S c*, *snd c*)) ≠ $\mathbf{0}_S$,
blast)
apply (*rule conjI*, *simp*)
apply (*subst polyn-expr-short*[*THEN sym*], *assumption*+)
apply (*simp add*:*polyn-c-max*)

apply *simp*
apply (*rule coeff-max-nonzeroTr*, *assumption*)
apply (*simp add*:*coeff-0-pol-0*[*THEN sym*])

apply *simp*
done

lemma (**in** *PolynRg*) *lcf-mem*:$p \in$ *carrier* $R \Longrightarrow$ (*lcf* R S X p) $\in$ *carrier* S
apply (*cut-tac subring*, *frule subring-Ring*)
apply (*simp add*:*lcf-def*)
 apply (*cut-tac pol-coeff-mem*[*of s-cf R S X p fst* (*s-cf R S X p*)],
 assumption,
 rule s-cf-pol-coeff, *assumption*, *simp*)
done

lemma (**in** *PolynRg*) *s-cf-expr0*:$p \in$ *carrier* $R \Longrightarrow$
 pol-coeff S (*s-cf* R S X p) $\wedge$
 p = *polyn-expr* R X (*fst* (*s-cf* R S X p)) (*s-cf* R S X p)
apply (*cut-tac subring*, *frule subring-Ring*)
apply (*simp add*:*s-cf-def*)
 apply (*case-tac* $p = \mathbf{0}_R$, *simp*)
 apply (*rule conjI*, *simp add*:*pol-coeff-def*, *simp add*:*Ring.ring-zero*)
 apply (*simp add*:*polyn-expr-def*,
 simp add:*Subring-zero-ring-zero*, *simp add*:*ring-r-one*)

apply *simp*
 apply (*rule someI2-ex*)
 apply (*frule ex-polyn-expr*[*of p*], *erule exE*, *erule conjE*)
 apply (*frule-tac* $c = c$ **in** *coeff-max-bddTr*)
 apply (*frule-tac* $c = c$ **and** n = *c-max* S c **in** *pol-coeff-le*, *assumption*)
 apply (*subgoal-tac* p = *polyn-expr* R X (*fst* (*c-max* S c, *snd* c))
 (*c-max* S c, *snd* c) $\wedge$
 snd (*c-max* S c, *snd* c) (*fst* (*c-max* S c, *snd* c)) $\neq \mathbf{0}_S$,
 blast)
 apply (*rule conjI*, *simp*)
 apply (*subst polyn-expr-short*[*THEN sym*], *assumption*+)
 apply (*simp add*:*polyn-c-max*)

 apply *simp*
 apply (*rule coeff-max-nonzeroTr*, *assumption*)
 apply (*simp add*:*coeff-0-pol-0*[*THEN sym*])
 apply *simp*
done

lemma (**in** *PolynRg*) *pos-deg-nonzero*:$[\![p \in$ *carrier* R; $0 <$ *deg-n* R S X $p]\!] \Longrightarrow$
 $p \neq \mathbf{0}$
apply (*cut-tac s-cf-expr0*[*of p*], (*erule conjE*)+)
 apply (*frule pol-deg-eq-c-max*[*of p s-cf R S X p*], *assumption*+)
 apply (*simp*, *thin-tac deg-n R S X p* = *c-max S* (*s-cf R S X p*))
 apply (*simp add*:*c-max-def*)
 apply (*case-tac* $\forall x \leq$*fst* (*s-cf R S X p*). *snd* (*s-cf R S X p*) $x = \mathbf{0}_S$, *simp*)
 apply (*thin-tac* $0 <$ (*if* $\forall x \leq$*fst* (*s-cf R S X p*). *snd* (*s-cf R S X p*) $x = \mathbf{0}_S$
 then 0 else n-max
 $\{j.\ j \leq$ *fst* (*s-cf R S X p*) $\wedge$ *snd* (*s-cf R S X p*) $j \neq \mathbf{0}_S\}$))
 apply (*simp add*:*coeff-0-pol-0*[*of s-cf R S X p fst* (*s-cf R S X p*)])
 apply *assumption*

done

lemma (**in** *PolynRg*) *s-cf-expr*:⟦$p \in$ *carrier* R; $p \neq \mathbf{0}$⟧ ⟹
$\quad$ *pol-coeff* S (*s-cf* R S X p) ∧
$\quad$ p = *polyn-expr* R X (*fst* (*s-cf* R S X p)) (*s-cf* R S X p) ∧
$\quad$ (*snd* (*s-cf* R S X p)) (*fst* (*s-cf* R S X p)) ≠ $\mathbf{0}_S$
apply (*simp add*:*s-cf-def*)
apply (*rule someI2-ex*)

apply (*frule ex-polyn-expr*[*of p*], *erule exE*, *erule conjE*)
apply (*frule-tac* c = c **in** *coeff-max-bddTr*)
apply (*frule-tac* c = c **and** n = *c-max* S c **in** *pol-coeff-le*, *assumption*)
apply (*subgoal-tac* p = *polyn-expr* R X (*fst* (*c-max* S c, *snd* c))
$\quad$ (*c-max* S c, *snd* c) ∧
$\quad$ *snd* (*c-max* S c, *snd* c) (*fst* (*c-max* S c, *snd* c)) ≠ $\mathbf{0}_S$,
$\quad$ *blast*)
apply (*rule conjI*, *simp*)
apply (*subst polyn-expr-short*[*THEN sym*], *assumption*+)
apply (*simp add*:*polyn-c-max*)

apply *simp*
apply (*rule coeff-max-nonzeroTr*, *assumption*)
apply (*simp add*:*coeff-0-pol-0*[*THEN sym*])
apply *simp*
done

lemma (**in** *PolynRg*) *lcf-nonzero*:⟦$p \in$ *carrier* R; $p \neq \mathbf{0}$ ⟧ ⟹
$\quad$ *lcf* R S X p ≠ $\mathbf{0}_S$
apply (*frule s-cf-expr*[*of p*], *assumption*)
apply (*simp add*:*lcf-def*)
done

lemma (**in** *PolynRg*) *s-cf-deg*:⟦$p \in$ *carrier* R; $p \neq \mathbf{0}$⟧ ⟹
$\quad$ *deg-n* R S X p = *fst* (*s-cf* R S X p)
apply (*frule s-cf-expr*[*of p*], *assumption*, (*erule conjE*)+)
apply (*simp add*:*pol-deg-n*[*of p s-cf R S X p fst* (*s-cf R S X p*)])
done

lemma (**in** *PolynRg*) *pol-expr-edeg*:⟦$p \in$ *carrier* R; *deg* R S X $p \leq$ (*an* d)⟧ ⟹
$\quad$ ∃f. (*pol-coeff* S f ∧ *fst* f = d ∧ p = *polyn-expr* R X d f)
apply (*case-tac* p = $\mathbf{0}_R$)
apply (*subgoal-tac pol-coeff* S (d, λj. $\mathbf{0}_S$) ∧ *fst* (d, λj. $\mathbf{0}_S$) = d ∧
$\quad$ p = *polyn-expr* R X d (d, λj. $\mathbf{0}_S$), *blast*)
apply (*rule conjI*)
apply (*simp add*:*pol-coeff-def*, *cut-tac Ring.ring-zero*[*of S*], *simp*,
$\quad$ *cut-tac subring*, *simp add*:*subring-Ring*)
apply (*cut-tac coeff-0-pol-0*[*of* (d, λj. $\mathbf{0}_S$) d], *simp*)
apply (*simp add*:*pol-coeff-def*, *cut-tac Ring.ring-zero*[*of S*], *simp*,
$\quad$ *cut-tac subring*, *simp add*:*subring-Ring*)

apply *simp*
apply (*frule s-cf-expr*[*of p*], *assumption*+, (*erule conjE*)+)
apply (*simp add*:*deg-def na-an*, *simp add*:*ale-natle*)
apply (*simp add*:*s-cf-deg*)
apply (*subgoal-tac pol-coeff S* (*d*, λj. (*if j* $\leq$ (*fst* (*s-cf R S X p*)) *then*
(*snd* (*s-cf R S X p*) *j*) *else* $\mathbf{0}_S$)) $\wedge$
p = *polyn-expr R X d* (*d*, λj. (*if j* $\leq$ (*fst* (*s-cf R S X p*)) *then*
(*snd* (*s-cf R S X p*) *j*) *else* $\mathbf{0}_S$)), *blast*)
apply (*rule conjI*)
apply (*simp add*:*pol-coeff-def*, *rule allI*, *rule impI*, *rule impI*)
apply (*cut-tac subring*, *simp add*:*subring-Ring*, *simp add*:*subring-Ring*[*of S*]
Ring.ring-zero)
apply (*case-tac fst* (*s-cf R S X p*) = *d*, *simp*)
apply (*subst polyn-exprs-eq*[*of* (*d*, λj. *if j* $\leq$ *d then snd* (*s-cf R S X p*) *j*
else $\mathbf{0}_S$) *s-cf R S X p d*])
apply (*simp add*:*pol-coeff-def*, *cut-tac Ring.ring-zero*[*of S*], *simp*,
cut-tac subring, *simp add*:*subring-Ring*, *simp*)
apply (*rule allI*, *rule impI*, *simp*, *assumption*+)
apply (*drule noteq-le-less*[*of fst* (*s-cf R S X p*) *d*], *assumption*+)
apply (*cut-tac polyn-n-m1*[*of* (*d*, λj. *if j* $\leq$ *fst* (*s-cf R S X p*) *then*
snd (*s-cf R S X p*) *j else* $\mathbf{0}_S$) *fst* (*s-cf R S X p*) *d*], *simp*)
apply (*cut-tac higher-part-zero*[*of* (*d*, λj. *if j* $\leq$ *fst* (*s-cf R S X p*) *then*
snd (*s-cf R S X p*) *j else* $\mathbf{0}_S$) *fst* (*s-cf R S X p*)], *simp*,
thin-tac polyn-expr R X d
(*d*, λj. *if j* $\leq$ *fst* (*s-cf R S X p*) *then snd* (*s-cf R S X p*) *j else* $\mathbf{0}_S$) =
polyn-expr R X (*fst* (*s-cf R S X p*)) (*d*, λj. *if j* $\leq$ *fst* (*s-cf R S X p*)
then snd (*s-cf R S X p*) *j else* $\mathbf{0}_S$) $\pm$ $\mathbf{0}$,
thin-tac Σ_f *R* (λj. (*if j* $\leq$ *fst* (*s-cf R S X p*) *then snd* (*s-cf R S X p*) *j*
else $\mathbf{0}_S$) $\cdot_r$ $X \hat{}^R j$) (*Suc* (*fst* (*s-cf R S X p*))) *d* = $\mathbf{0}$)
apply (*subst polyn-exprs-eq*[*of* (*d*, λj. *if j* $\leq$ *fst* (*s-cf R S X p*) *then*
snd (*s-cf R S X p*) *j else* $\mathbf{0}_S$) *s-cf R S X p fst* (*s-cf R S X p*)])
apply (*simp add*:*pol-coeff-def*, *rule allI*, *rule impI*, *rule impI*,
cut-tac subring, *simp add*:*subring-Ring*, *simp add*:*subring-Ring*[*of S*]
Ring.ring-zero, *assumption*+) **apply** (*simp*)
apply (*rule allI*, *rule impI*, *simp*)
apply (*frule polyn-mem*[*of s-cf R S X p fst* (*s-cf R S X p*)], *simp*+)
apply (*cut-tac ring-is-ag*, *simp add*:*aGroup.ag-r-zero*)
apply (*simp add*:*pol-coeff-def*, *rule allI*, *rule impI*, *rule impI*,
cut-tac subring, *simp add*:*subring-Ring*, *simp add*:*subring-Ring*[*of S*]
Ring.ring-zero)
apply *simp*
apply (*rule ballI*, *simp add*:*nset-def*)
apply (*simp add*:*pol-coeff-def*, *rule allI*, *rule impI*, *rule impI*,
cut-tac subring, *simp add*:*subring-Ring*, *simp add*:*subring-Ring*[*of S*]
Ring.ring-zero)
apply *assumption* **apply** *simp*
done

lemma (**in** *PolynRg*) *cf-scf*:⟦*pol-coeff S c*; *k* $\leq$ *fst c*; *polyn-expr R X k c* $\neq$ $\mathbf{0}$⟧

$\Longrightarrow\ \forall j \leq fst\ (s\text{-}cf\ R\ S\ X\ (polyn\text{-}expr\ R\ X\ k\ c)).$
$snd\ (s\text{-}cf\ R\ S\ X\ (polyn\text{-}expr\ R\ X\ k\ c))\ j = snd\ c\ j$
apply (*frule polyn-mem*[*of c k*], *assumption*+)
apply (*simp add*:*polyn-expr-short*[*of c k*],
rule allI, *rule impI*)
apply (*cut-tac pol-deg-le-n1*[*of polyn-expr R X k c c k*],
frule s-cf-expr0[*of polyn-expr R X k* (*k*, *snd c*)], *erule conjE*)
apply (*rotate-tac* −*1*, *drule sym*)
apply (*case-tac fst* (*s-cf R S X* (*polyn-expr R X k* (*k*, *snd c*))) = *k*,
simp,
cut-tac c = *s-cf R S X* (*polyn-expr R X k* (*k*, *snd c*)) **and**
d = (*k*, *snd c*) **in** *pol-expr-unique2*,
simp add:*s-cf-pol-coeff*, *simp add*:*split-pol-coeff*, *simp*,
simp, *simp add*:*polyn-expr-short*[*THEN sym*, *of c k*])

apply (*simp add*:*s-cf-deg*[*of polyn-expr R X k c*],
drule noteq-le-less[*of fst* (*s-cf R S X* (*polyn-expr R X k c*)) *k*],
assumption)
apply (*frule pol-expr-unique3*[*of s-cf R S X* (*polyn-expr R X k c*)
(*k*, *snd c*)], *simp add*:*split-pol-coeff*, *simp*,
simp add:*polyn-expr-short*[*THEN sym*, *of c k*])
apply (*simp add*:*polyn-expr-short*[*THEN sym*, *of c k*], *assumption*+, *simp*)
done

definition
scf-cond :: [('*a*, '*m*) *Ring-scheme*, ('*a*, '*m1*) *Ring-scheme*, '*a*, '*a*,
nat, *nat* × (*nat* ⇒ '*a*)] ⇒ *bool* **where**
scf-cond R S X p d c ⟷ *pol-coeff S c* ∧ *fst c* = *d* ∧ *p* = *polyn-expr R X d c*

definition
scf-d :: [('*a*, '*m*) *Ring-scheme*, ('*a*, '*m1*) *Ring-scheme*, '*a*, '*a*, *nat*]
⇒ *nat* × (*nat* ⇒ '*a*) **where**
scf-d R S X p d = (*SOME f*. *scf-cond R S X p d f*)

lemma (**in** *PolynRg*) *scf-d-polTr*:⟦*p* ∈ *carrier R*; *deg R S X p* ≤ *an d*⟧ ⟹
scf-cond R S X p d (*scf-d R S X p d*)
apply (*simp add*:*scf-d-def*)
apply (*rule-tac P* = *scf-cond R S X p d* **in** *someI2-ex*)
apply (*frule pol-expr-edeg*[*of p d*], *assumption*+)
apply (*simp add*:*scf-cond-def*, *assumption*)
done

lemma (**in** *PolynRg*) *scf-d-pol*:⟦*p* ∈ *carrier R*; *deg R S X p* ≤ *an d*⟧ ⟹
pol-coeff S (*scf-d R S X p d*) ∧ *fst* (*scf-d R S X p d*) = *d* ∧
p = *polyn-expr R X d* (*scf-d R S X p d*)
apply (*frule scf-d-polTr*[*of p d*], *assumption*+)
apply (*simp add*:*scf-cond-def*)

done

lemma (**in** *PolynRg*) *pol-expr-of-X*:
  X = *polyn-expr R X* (*Suc 0*) (*ext-cf S* (*Suc 0*) (C_0 *S*))
apply (*cut-tac X-mem-R*, *cut-tac subring*)
apply (*cut-tac X-to-d*[*of Suc 0*])
apply (*simp add*:*ring-l-one*)
done

lemma (**in** *PolynRg*) *deg-n-of-X*:*deg-n R S X X* = *Suc 0*
apply (*cut-tac X-mem-R*, *cut-tac polyn-ring-S-nonzero*,
  cut-tac subring)
apply (*cut-tac pol-expr-of-X*)
apply (*cut-tac special-cf-pol-coeff*)
apply (*frule ext-cf-pol-coeff*[*of* C_0 *S Suc 0*])
apply (*frule pol-deg-eq-c-max*[*of X ext-cf S* (*Suc 0*) (C_0 *S*)], *assumption*)
  apply (*simp add*:*ext-cf-len special-cf-len*)
apply (*simp add*:*c-max-ext-special-cf*)
done

lemma (**in** *PolynRg*) *pol-X*:*cf-sol R S X X c* $\Longrightarrow$
    snd c 0 = $\mathbf{0}_S$ $\wedge$ *snd c* (*Suc 0*) = $1_{r\,S}$

apply (*simp add*:*cf-sol-def*, *erule conjE*)
apply (*cut-tac pol-expr-of-X*)
apply (*cut-tac special-cf-pol-coeff*,
    frule ext-cf-pol-coeff[*of* C_0 *S Suc 0*])
apply (*cut-tac X-mem-R*, *cut-tac polyn-ring-X-nonzero*,
  cut-tac subring)
apply (*frule pol-deg-le-n*[*of X c*], *assumption*+, *simp add*:*deg-n-of-X*)
apply (*case-tac fst c* = *Suc 0*)
apply (*frule box-equation*[*of X polyn-expr R X* (*Suc 0*)
  (*ext-cf S* (*Suc 0*) (C_0 *S*)) *polyn-expr R X* (*fst c*) *c*], *assumption*+,
  thin-tac X = *polyn-expr R X* (*Suc 0*) (*ext-cf S* (*Suc 0*) (C_0 *S*)),
  thin-tac X = *polyn-expr R X* (*fst c*) *c*)
apply (*cut-tac pol-expr-unique2*[*of ext-cf S* (*Suc 0*) (C_0 *S*) *c*],
  simp, *simp add*:*ext-cf-len special-cf-len*, *rule conjI*,
  drule-tac a = *0* **in** *forall-spec*, *simp*,
  simp add:*ext-special-cf-lo-zero*)
apply(*drule-tac a* = *Suc 0* **in** *forall-spec*, *simp*,
  simp add:*ext-special-cf-hi*, *assumption*+,
  simp add:*ext-cf-len special-cf-len*)

apply (*frule noteq-le-less*[*of Suc 0 fst c*],*rule not-sym*, *assumption*,
  cut-tac pol-expr-unique3[*of ext-cf S* (*Suc 0*) (C_0 *S*) *c*],
  simp add:*ext-cf-len special-cf-len*,
  erule conjE,
  thin-tac $\forall j{\in}$*nset* (*Suc* (*Suc 0*)) (*fst c*). *snd c j* = $\mathbf{0}_S$)
apply (*rule conjI*,

```
        drule-tac a = 0 in forall-spec, simp,
        simp add:ext-special-cf-lo-zero,
        drule-tac a = Suc 0 in forall-spec, simp,
        simp add:ext-special-cf-hi,
        assumption+)
 apply (simp add:ext-cf-len special-cf-len)
done

lemma (in PolynRg) pol-of-deg0:⟦p ∈ carrier R; p ≠ 𝟬⟧
      ⟹ (deg-n R S X p = 0) = (p ∈ carrier S)
apply (cut-tac subring,
       frule subring-Ring,
       cut-tac ring-is-ag,
       frule Ring.ring-is-ag[of S])
apply (rule iffI)
 apply (frule s-cf-expr[of p], assumption)
 apply (simp add:s-cf-deg, (erule conjE)+, simp add:polyn-expr-def)
 apply (frule pol-coeff-mem[of s-cf R S X p 0], simp)
 apply (cut-tac mem-subring-mem-ring[of S snd (s-cf R S X p) 0],
        simp add:ring-r-one, assumption+)

apply (frule s-cf-expr[of p], assumption+, (erule conjE)+,
       simp add:s-cf-deg)
apply (rule contrapos-pp, simp+)
apply (subgoal-tac pol-coeff S (0, (λj. p)) ∧
                   p = polyn-expr R X 0 (0, (λj. p)), erule conjE)
apply (cut-tac a = polyn-expr R X 0 (0, λj. p) and
               b = polyn-expr R X (fst (s-cf R S X p)) (s-cf R S X p) in
               aGroup.ag-eq-diffzero[of R],  assumption+, simp, simp)

 apply (frule-tac c = polyn-expr R X (fst (s-cf R S X p)) (s-cf R S X p) in
         box-equation[of p polyn-expr R X 0 (0, λj. p)], assumption,
        thin-tac p = polyn-expr R X (fst (s-cf R S X p)) (s-cf R S X p),
        thin-tac p = polyn-expr R X 0 (0, λj. p), simp)

 apply (simp only:polyn-minus-m-cf)
  apply (rotate-tac −2, drule sym, simp,
        thin-tac polyn-expr R X (fst (s-cf R S X p)) (s-cf R S X p) =
                 polyn-expr R X 0 (0, λj. p))
 apply (frule-tac c = s-cf R S X p in m-cf-pol-coeff)

 apply (frule-tac d = m-cf S (s-cf R S X p) in polyn-add1[of (0, λj. p)],
        assumption,
        simp add:m-cf-len,
        thin-tac polyn-expr R X 0 (0, λj. p) ±
     polyn-expr R X (fst (s-cf R S X p)) (m-cf S (s-cf R S X p)) =
     polyn-expr R X (fst (s-cf R S X p))
      (add-cf S (0, λj. p) (m-cf S (s-cf R S X p))))
     apply (rotate-tac −1, drule sym, simp)
```

apply (*frule-tac* d = *m-cf* S (*s-cf* R S X p) **in**
add-cf-pol-coeff[*of* (0, $\lambda j.$ p)], *assumption*)
apply (*frule-tac* *c1* = *add-cf* S (0, $\lambda j.$ p) (*m-cf* S (*s-cf* R S X p)) **and**
k1 = *fst* (*s-cf* R S X p) **in**
coeff-0-pol-0[*THEN sym*],
simp add:*add-cf-len m-cf-len*, *simp*,
thin-tac pol-coeff S (*add-cf* S (0, $\lambda j.$ p) (*m-cf* S (*s-cf* R S X p))),
thin-tac polyn-expr R X (*fst* (*s-cf* R S X p))
(*add-cf* S (0, $\lambda j.$ p) (*m-cf* S (*s-cf* R S X p))) = **0**)
apply (*drule-tac* a = *fst* (*s-cf* R S X p) **in** *forall-spec*, *simp*,
simp add:*add-cf-def m-cf-len m-cf-def*)
apply (*frule-tac* c = *s-cf* R S X p **and** j = *fst* (*s-cf* R S X p) **in**
pol-coeff-mem, *simp*,
frule-tac x = *snd* (*s-cf* R S X p) (*fst* (*s-cf* R S X p)) **in**
aGroup.ag-inv-inv[*of* S],
assumption, *simp add*:*aGroup.ag-inv-zero*)
apply (*subst polyn-expr-def*, *simp add*:*ring-r-one*)
apply (*simp add*:*pol-coeff-def*)
done

lemma (**in** *PolynRg*) *pols-const*:⟦$p \in$ *carrier* R; (*deg* R S X p) $\leq$ 0⟧ $\Longrightarrow$
$p \in$ *carrier* S
apply (*case-tac* $p = \mathbf{0}_R$)
apply (*cut-tac subring*)
apply (*frule Subring-zero-ring-zero*[*THEN sym*, *of* S], *simp*,
cut-tac subring,
frule subring-Ring[*of* S],
rule Ring.ring-zero[*of* S], *assumption*)
apply (*subst pol-of-deg0*[*THEN sym*], *assumption*+,
simp add:*deg-def*,
simp only:*an-0*[*THEN sym*],
simp add:*an-inj*)
done

lemma (**in** *PolynRg*) *less-deg-add-nonzero*:⟦$p \in$ *carrier* R; $p \neq \mathbf{0}$;
$q \in$ *carrier* R; $q \neq \mathbf{0}$;
(*deg-n* R S X p) < (*deg-n* R S X q)⟧ $\Longrightarrow$ $p \pm q \neq \mathbf{0}$
apply (*frule ex-polyn-expr*[*of* p], *erule exE*, *erule conjE*,
frule ex-polyn-expr[*of* q], *erule exE*, *erule conjE*,
rename-tac c d)
apply (*frule-tac* c = c **in** *pol-deg-eq-c-max*[*of* p], *assumption*+,
frule-tac c = d **in** *pol-deg-eq-c-max*[*of* q], *assumption*+,
frule-tac c = c **in** *coeff-max-bddTr*,
frule-tac c = d **in** *coeff-max-bddTr*)
apply (*frule-tac* c = c **and** n = *c-max* S c **in** *pol-coeff-le*, *assumption*,
frule-tac c = d **and** n = *c-max* S d **in** *pol-coeff-le*, *assumption*+)
apply *simp*

apply (*subst polyn-c-max*, *assumption*,
subst polyn-c-max, *assumption*,
subst polyn-expr-short, *assumption*+)
apply (*frule-tac* $c = d$ **and** $k =$ *c-max S d* **in** *polyn-expr-short*, *assumption*+,
simp,
thin-tac polyn-expr R X (*c-max S d*) *d* =
polyn-expr R X (*c-max S d*) (*c-max S d*, *snd d*),
thin-tac deg-n R S X (*polyn-expr R X* (*fst c*) *c*) = *c-max S c*,
thin-tac deg-n R S X (*polyn-expr R X* (*fst d*) *d*) = *c-max S d*)

apply (*subst polyn-add*, *assumption*+, *simp add*: *max.absorb1 max.absorb2*)

apply (*rule contrapos-pp*, *simp*+,
frule-tac $c =$ (*c-max S c*, *snd c*) **and** $d =$ (*c-max S d*, *snd d*) **in**
add-cf-pol-coeff, *assumption*+,
frule-tac c1 = *add-cf S* (*c-max S c*, *snd c*) (*c-max S d*, *snd d*) **and**
k1 = *c-max S d* **in** *coeff-0-pol-0*[*THEN sym*],
simp add:*add-cf-len*, *simp*,
thin-tac pol-coeff S (*add-cf S* (*c-max S c*, *snd c*) (*c-max S d*, *snd d*)),
thin-tac polyn-expr R X (*c-max S d*)
(*add-cf S* (*c-max S c*, *snd c*) (*c-max S d*, *snd d*)) = **0**)
apply (*drule-tac a* = *c-max S d* **in** *forall-spec*, *simp*,
simp add:*add-cf-def*,
frule-tac $c = d$ **and** $k =$ *fst d* **in** *coeff-nonzero-polyn-nonzero*,
simp, *simp*,
frule-tac $c = d$ **in** *coeff-max-nonzeroTr*, *assumption*+, *simp*)
done

lemma (**in** *PolynRg*) *polyn-deg-add1*:⟦$p \in$ *carrier R*; $p \neq$ **0**; $q \in$ *carrier R*;
$q \neq$ **0**; (*deg-n R S X p*) < (*deg-n R S X q*)⟧ ⟹
deg-n R S X ($p \pm q$) = (*deg-n R S X q*)
apply (*cut-tac subring*)
apply (*frule less-deg-add-nonzero*[*of p q*], *assumption*+)
apply (*frule ex-polyn-expr*[*of p*], *erule exE*, *erule conjE*,
frule ex-polyn-expr[*of q*], *erule exE*, *erule conjE*,
rename-tac c d)
apply (*simp only*:*pol-deg-eq-c-max*)
apply (*frule-tac* $c = c$ **in** *coeff-max-bddTr*,
frule-tac $c = d$ **in** *coeff-max-bddTr*)
apply (*frule-tac* $c = c$ **and** $n =$ *c-max S c* **in** *pol-coeff-le*, *assumption*,
frule-tac $c = d$ **and** $n =$ *c-max S d* **in** *pol-coeff-le*, *assumption*)
apply (*simp add*:*polyn-c-max*)

apply (*frule-tac* $c = c$ **and** $k =$ *c-max S c* **in** *polyn-expr-short*, *simp*,
frule-tac $c = d$ **and** $k =$ *c-max S d* **in** *polyn-expr-short*, *simp*,
simp)
apply (*frule-tac* $c =$ (*c-max S c*, *snd c*) **and** $d =$ (*c-max S d*, *snd d*) **in**
polyn-add1, *assumption*+, *simp*,
thin-tac polyn-expr R X (*c-max S c*) *c* =

```
        polyn-expr R X (c-max S c) (c-max S c, snd c),
      thin-tac polyn-expr R X (c-max S d) d =
        polyn-expr R X (c-max S d) (c-max S d, snd d))
 apply (simp add: max.absorb1 max.absorb2)
 apply (frule-tac c = (c-max S c, snd c) and d = (c-max S d, snd d) in
            add-cf-pol-coeff, assumption+,
       rule-tac p = polyn-expr R X (c-max S d)
         (add-cf S (c-max S c, snd c) (c-max S d, snd d)) and
        c = add-cf S (c-max S c, snd c) (c-max S d, snd d) and
        n = c-max S d in pol-deg-n)
 apply (rule-tac polyn-mem, simp, simp add:add-cf-len,
       assumption,
       simp add:add-cf-len, simp,
       subst add-cf-def, simp)

 apply (frule-tac c1 = (c-max S d, snd d) and k1 = c-max S d in
       coeff-0-pol-0[THEN sym], simp, simp,
       rule coeff-max-nonzeroTr, assumption+,
       erule exE, erule conjE,
       frule-tac i = j and j = c-max S d and k = fst d in
            le-trans, assumption+, blast)
done

lemma (in PolynRg) polyn-deg-add2:⟦p ∈ carrier R; p ≠ 0; q ∈ carrier R;
     q ≠ 0; p ± q ≠ 0; (deg-n R S X p) = (deg-n R S X q)⟧ ⟹
        deg-n R S X (p ± q) ≤ (deg-n R S X q)
apply (cut-tac subring)
apply (frule ex-polyn-expr[of p], erule exE, erule conjE,
      frule ex-polyn-expr[of q], erule exE, erule conjE,
      rename-tac c d)
     apply (simp only:pol-deg-eq-c-max)
apply (frule-tac c = c in coeff-max-bddTr,
      frule-tac c = d in coeff-max-bddTr)
apply (frule-tac c = c and n = c-max S c in pol-coeff-le, assumption,
      frule-tac c = d and n = c-max S d in pol-coeff-le, assumption)
apply (simp add:polyn-c-max)

apply (frule-tac c = c and k = c-max S c in polyn-expr-short, simp,
      frule-tac c = d and k = c-max S d in polyn-expr-short, simp,
      simp,
      frule-tac c = (c-max S d, snd c) and d = (c-max S d, snd d) in
        polyn-add1, simp)
 apply (thin-tac polyn-expr R X (c-max S d) d =
         polyn-expr R X (c-max S d) (c-max S d, snd d),
       thin-tac polyn-expr R X (c-max S d) c =
         polyn-expr R X (c-max S d) (c-max S d, snd c), simp)
 apply (thin-tac polyn-expr R X (c-max S d) (c-max S d, snd c) ±
         polyn-expr R X (c-max S d) (c-max S d, snd d) =
         polyn-expr R X (c-max S d)
```

```
          (add-cf S (c-max S d, snd c) (c-max S d, snd d)))
  apply (frule-tac c = (c-max S d, snd c) and d = (c-max S d, snd d) in
              add-cf-pol-coeff, assumption+)
 apply (cut-tac p = polyn-expr R X (c-max S d)
             (add-cf S (c-max S d, snd c) (c-max S d, snd d))
       and c = add-cf S (c-max S d, snd c) (c-max S d, snd d) in
       pol-deg-eq-c-max)
   apply (rule-tac polyn-mem, simp)
     apply (simp add:add-cf-len,
            assumption,
           simp add:add-cf-len, simp)
     apply (thin-tac deg-n R S X
          (polyn-expr R X (c-max S d)
            (add-cf S (c-max S d, snd c) (c-max S d, snd d))) =
         c-max S (add-cf S (c-max S d, snd c) (c-max S d, snd d)),
           thin-tac polyn-expr R X (c-max S d)
          (add-cf S (c-max S d, snd c) (c-max S d, snd d)) ≠
         0)

  apply (frule-tac c = add-cf S (c-max S d, snd c) (c-max S d, snd d) in
        coeff-max-bddTr)
  apply (simp add:add-cf-len)
done

lemma (in PolynRg) polyn-deg-add3:⟦p ∈ carrier R; p ≠ 0; q ∈ carrier R;
      q ≠ 0; p ± q ≠ 0; (deg-n R S X p) ≤ n; (deg-n R S X q) ≤ n⟧ ⟹
         deg-n R S X (p ± q) ≤ n
apply (case-tac (deg-n R S X p) = (deg-n R S X q),
       frule polyn-deg-add2[of p q], assumption+,
       simp)
apply (cut-tac less-linear[of deg-n R S X p deg-n R S X q],
       simp, thin-tac deg-n R S X p ≠ deg-n R S X q,
       erule disjE, simp add:polyn-deg-add1,
       cut-tac ring-is-ag, simp add:aGroup.ag-pOp-commute[of R p q],
       simp add:polyn-deg-add1)
done

lemma (in PolynRg) polyn-deg-add4:⟦p ∈ carrier R; q ∈ carrier R;
     (deg R S X p) ≤ (an n); (deg R S X q) ≤ (an n)⟧ ⟹
                deg R S X (p ± q) ≤ (an n)
apply (cut-tac ring-is-ag)
apply (case-tac p = 0_R, simp add:aGroup.ag-l-zero)
apply (case-tac q =  0_R, simp add:aGroup.ag-r-zero)
apply (case-tac p ±_R q = 0_R, simp add:deg-def)
apply (frule aGroup.ag-pOp-closed[of R p q], assumption+)
apply (simp add:deg-an)
apply (simp only:ale-natle)
apply (simp add:polyn-deg-add3)
done
```

lemma (**in** *PolynRg*) *polyn-deg-add5*:⟦$p \in$ *carrier* R; $q \in$ *carrier* R;
$(deg\ R\ S\ X\ p) \leq a$; $(deg\ R\ S\ X\ q) \leq a$⟧ $\Longrightarrow$
$deg\ R\ S\ X\ (p \pm q) \leq a$
apply (*case-tac* $a = \infty$, *simp*)
apply (*cut-tac ring-is-ag*,
case-tac $p = \mathbf{0}_R$, *simp add*:*aGroup.ag-l-zero*[*of R*],
case-tac $q = \mathbf{0}_R$, *simp add*:*aGroup.ag-r-zero*,
simp add:*deg-an*[*of p*])
apply (*cut-tac an-nat-pos*[*of deg-n R S X p*],
frule ale-trans[*of 0 an* $(deg\text{-}n\ R\ S\ X\ p)$ *a*], *assumption*+,
subgoal-tac an $(deg\text{-}n\ R\ S\ X\ p) \leq an\ (na\ a)$,
simp only:*ale-natle*[*of deg-n R S X p na a*])

apply (*simp add*:*deg-an*[*of q*])
apply (*cut-tac an-nat-pos*[*of deg-n R S X q*],
frule ale-trans[*of 0 an* $(deg\text{-}n\ R\ S\ X\ q)$ *a*], *assumption*+,
subgoal-tac an $(deg\text{-}n\ R\ S\ X\ q) \leq an\ (na\ a)$,
simp only:*ale-natle*[*of deg-n R S X q na a*])
apply (*frule polyn-deg-add4*[*of p q na a*], *assumption*+,
simp add:*an-na*, *simp add*:*deg-an*,
simp add:*deg-an an-na*, *simp add*:*an-na*)
apply (*simp add*:*deg-an an-na*, *simp add*:*deg-an an-na*)
done

lemma (**in** *PolynRg*) *lower-deg-part*:⟦$p \in$ *carrier* R; $p \neq \mathbf{0}$; $0 < deg\text{-}n\ R\ S\ X\ p$⟧
$\Longrightarrow$
$deg\ R\ S\ X\ (polyn\text{-}expr\ R\ X\ (deg\text{-}n\ R\ S\ X\ p - Suc\ 0)(SOME\ f.\ cf\text{-}sol\ R\ S\ X\ p\ f))$
$< deg\ R\ S\ X\ p$
apply (*case-tac polyn-expr R X* $(deg\text{-}n\ R\ S\ X\ p - Suc\ 0)$
$(SOME\ f.\ cf\text{-}sol\ R\ S\ X\ p\ f) = \mathbf{0}_R$)
apply (*simp add*:*deg-def*, *cut-tac minf-le-any*[*of an* $(deg\text{-}n\ R\ S\ X\ p)$])
apply (*subst less-le*, *simp*, *simp add*:*an-def*)
apply (*rule not-sym*, *rule contrapos-pp*, *simp*+)

apply (*simp add*:*deg-def*, *simp add*:*aless-natless*)
apply (*frule pol-SOME-2*[*of p*], *erule conjE*)
apply (*simp add*:*pol-deg-eq-c-max*[*of p SOME c. cf-sol R S X p c*])
apply (*frule-tac c = SOME c. cf-sol R S X p c* **in** *coeff-max-bddTr*)

apply (*cut-tac*
$p = polyn\text{-}expr\ R\ X\ (c\text{-}max\ S\ (SOME\ c.\ cf\text{-}sol\ R\ S\ X\ p\ c) - Suc\ 0)$
$(Eps\ (cf\text{-}sol\ R\ S\ X\ p))$ **and** $c = (c\text{-}max\ S\ (SOME\ c.\ cf\text{-}sol\ R\ S\ X\ p\ c) - Suc\ 0, snd\ (SOME\ c.\ cf\text{-}sol\ R\ S\ X\ p\ c))$ **in** *pol-deg-eq-c-max*)

apply (*rule polyn-mem*, *simp*, *arith*)
apply (*rule pol-coeff-le*, *assumption*, *arith*)
apply (*subst polyn-expr-short*, *arith*, *arith*, *simp*)

apply *simp*

apply (*cut-tac c* = (*c-max S* (*SOME c. cf-sol R S X p c*) − *Suc 0*,
snd (*SOME c. cf-sol R S X p c*)) **in** *coeff-max-bddTr*,
rule pol-coeff-le, *assumption*, *arith*, *simp*)
done

definition
ldeg-p :: [('*a*, '*m*) *Ring-scheme*, ('*a*, '*m1*) *Ring-scheme*, '*a*, *nat*, '*a*]
⇒ '*a* **where**
ldeg-p R S X d p = *polyn-expr R X d* (*scf-d R S X p* (*Suc d*))

definition
hdeg-p :: [('*a*, '*m*) *Ring-scheme*, ('*a*, '*m1*) *Ring-scheme*, '*a*, *nat*, '*a*]
⇒ '*a* **where**
hdeg-p R S X d p = (*snd* (*scf-d R S X p d*) *d*) $\cdot_{r\,R}$ ($X^{\hat{}R}$ *d*)

lemma (**in** *PolynRg*) *ldeg-p-mem*:⟦*p* ∈ *carrier R*; *deg R S X p* ≤ *an* (*Suc d*) ⟧ ⟹
ldeg-p R S X d p ∈ *carrier R*
apply (*frule scf-d-pol*[*of p Suc d*], *assumption*+,
erule conjE)
apply (*simp add*:*ldeg-p-def*)
apply (*rule polyn-mem*[*of scf-d R S X p* (*Suc d*) *d*],
assumption+)
apply *simp*
done

lemma (**in** *PolynRg*) *ldeg-p-zero*:*p* = $\mathbf{0}_R$ ⟹ *ldeg-p R S X d p* = $\mathbf{0}_R$
apply (*subgoal-tac deg R S X p* ≤ *an* (*Suc d*),
subgoal-tac p ∈ *carrier R*)
apply (*frule scf-d-pol*[*of p Suc d*], *assumption*+,
erule conjE)
apply *simp*
apply (*frule coeff-0-pol-0*[*of scf-d R S X* **0** (*Suc d*) *Suc d*], *simp*)
apply (*simp add*:*ldeg-p-def*)
apply (*subst coeff-0-pol-0*[*THEN sym*, *of scf-d R S X* **0** (*Suc d*)],
assumption+, *simp*)
apply (*rule allI*, *rule impI*, *simp*)
apply (*simp*, *simp add*:*ring-zero*)
apply (*simp add*:*deg-def*)
done

lemma (**in** *PolynRg*) *hdeg-p-mem*:⟦*p* ∈ *carrier R*; *deg R S X p* ≤ *an* (*Suc d*)⟧ ⟹
hdeg-p R S X (*Suc d*) *p* ∈ *carrier R*
apply (*frule scf-d-pol*[*of p Suc d*], *assumption*+, *erule conjE*)
apply (*simp only*:*hdeg-p-def*, (*erule conjE*)+)
apply (*cut-tac Ring*)

apply (*rule Ring.ring-tOp-closed*[*of R*], *assumption*)
apply (*frule pol-coeff-mem*[*of scf-d R S X p* (*Suc d*) *Suc d*], *simp*)
apply (*cut-tac subring*)
apply (*simp add*:*Ring.mem-subring-mem-ring*)
apply (*rule Ring.npClose*[*of R*], *assumption*+)
apply (*rule X-mem-R*)
done

lemma (**in** *PolynRg*) *hdeg-p-zero*:$p = \mathbf{0} \Longrightarrow$ *hdeg-p R S X* (*Suc d*) $p = \mathbf{0}$
apply (*cut-tac X-mem-R*)
apply (*subgoal-tac deg R S X p* $\leq$ *an* (*Suc d*),
  subgoal-tac p $\in$ *carrier R*)
apply (*frule scf-d-pol*[*of p Suc d*], *assumption*+, *erule conjE*)
apply *simp*
apply (*frule coeff-0-pol-0*[*of scf-d R S X* **0** (*Suc d*) *Suc d*],
  (*erule conjE*)+, *simp*)
apply (*simp only*:*hdeg-p-def*)
apply (*rotate-tac* -1, *drule sym*, *simp del*:*npow-suc*)
apply (*cut-tac subring*,
  simp del:*npow-suc add*:*Subring-zero-ring-zero*,
  rule ring-times-0-x, *rule npClose*, *assumption*)
apply (*simp add*:*ring-zero*)
apply (*simp add*:*deg-def*)
done

lemma (**in** *PolynRg*) *decompos-p*:$[\![p \in$ *carrier R*; *deg R S X p* $\leq$ *an* (*Suc d*)$]\!] \Longrightarrow$
    $p =$ (*ldeg-p R S X d p*) $\pm$ (*hdeg-p R S X* (*Suc d*) *p*)
apply (*frule scf-d-pol*[*of p Suc d*], *assumption*+, *erule conjE*)
apply (*cut-tac subring*, (*erule conjE*)+)
apply (*cut-tac polyn-Suc*[*of d scf-d R S X p* (*Suc d*)])
apply (*simp only*:*ldeg-p-def hdeg-p-def*)
apply (*rotate-tac* -1, *drule sym*, *simp del*:*npow-suc*)
apply (*thin-tac polyn-expr R X d* (*scf-d R S X p* (*Suc d*)) $\pm$
  snd (*scf-d R S X p* (*Suc d*)) (*Suc d*) $\cdot_r$ $X \hat{}_R$ (*Suc d*) $=$
  polyn-expr R X (*Suc d*) (*Suc d*, *snd* (*scf-d R S X p* (*Suc d*))))
apply (*simp add*:*polyn-expr-split*[*of Suc d scf-d R S X p* (*Suc d*)],
  simp)
done

lemma (**in** *PolynRg*) *deg-ldeg-p*:$[\![p \in$ *carrier R*; *deg R S X p* $\leq$ *an* (*Suc d*)$]\!] \Longrightarrow$
    deg R S X (*ldeg-p R S X d p*) $\leq$ *an d*

apply (*cut-tac subring*,
frule subring-Ring)
apply (*case-tac* $p = \mathbf{0}_R$)
apply (*simp add*:*ldeg-p-zero*, *simp add*:*deg-def*)
apply (*frule scf-d-pol*[*of p Suc d*], *assumption*+, (*erule conjE*)+)
apply (*simp only*:*ldeg-p-def*)
apply (*case-tac polyn-expr R X d* (*scf-d R S X p* (*Suc d*)) = $\mathbf{0}_R$)
apply (*simp add*:*deg-def*)

apply (*simp add*:*deg-an*)
apply (*simp add*:*ale-natle*)
apply (*cut-tac pol-deg-le-n1*[*of polyn-expr R X d* (*scf-d R S X p* (*Suc d*))
scf-d R S X p (*Suc d*) *d*], *simp add*:*deg-def ale-natle*)
apply (*rule polyn-mem*, *assumption*+, *simp*+)
done

lemma (**in** *PolynRg*) *deg-minus-eq*:⟦$p \in$ *carrier R*; $p \neq \mathbf{0}$⟧ ⟹
deg-n R S X ($-_a$ *p*) = *deg-n R S X p*
apply (*cut-tac subring*,
cut-tac ring-is-ag,
frule subring-Ring)
apply (*cut-tac ring-is-ag*)
apply (*frule s-cf-expr*[*of p*], *assumption*, (*erule conjE*)+,
frule polyn-minus-m-cf[*of s-cf R S X p fst* (*s-cf R S X p*)], *simp*,
drule sym, *simp*)
apply (*frule-tac x* = *p* **in** *aGroup.ag-mOp-closed*, *assumption*+,
frule m-cf-pol-coeff [*of s-cf R S X p*],
frule pol-deg-n[*of* $-_a$ *p m-cf S* (*s-cf R S X p*)
fst (*s-cf R S X p*)], *assumption*,
simp add:*m-cf-len*, *assumption*+)
apply (*simp add*:*m-cf-def*,
frule pol-coeff-mem[*of s-cf R S X p fst* (*s-cf R S X p*)], *simp*,
frule Ring.ring-is-ag[*of S*])
apply (*rule contrapos-pp*, *simp*+)
apply (*frule aGroup.ag-inv-inv*[*THEN sym*,
of S snd (*s-cf R S X p*) (*fst* (*s-cf R S X p*))], *assumption*,
simp add:*aGroup.ag-inv-zero*)
apply (*drule sym*, *simp*, *simp add*:*s-cf-deg*)
done

lemma (**in** *PolynRg*) *deg-minus-eq1*:*p* ∈ *carrier R* ⟹
deg R S X ($-_a$ *p*) = *deg R S X p*
apply (*cut-tac ring-is-ag*)
apply (*case-tac* $p = \mathbf{0}_R$)
apply (*simp add*:*aGroup.ag-inv-zero*)
apply (*frule deg-minus-eq*[*of p*], *assumption*+,
frule aGroup.ag-inv-inj[*of R p* **0**], *assumption*,
simp add:*ring-zero*, *assumption*, *simp add*:*aGroup.ag-inv-zero*)
apply (*frule aGroup.ag-mOp-closed*[*of R p*], *assumption*,

simp add:*deg-an*)
done

lemma (**in** *PolynRg*) *ldeg-p-pOp*:⟦$p \in$ *carrier R*; $q \in$ *carrier R*;
deg R S X p $\leq$ *an* (*Suc d*); *deg R S X q* $\leq$ *an* (*Suc d*)⟧ $\Longrightarrow$
(*ldeg-p R S X d p*) $\pm_R$ (*ldeg-p R S X d q*) =
ldeg-p R S X d (*p* $\pm_R$ *q*)
apply (*simp add*:*ldeg-p-def*,
cut-tac ring-is-ag, *cut-tac subring*, *frule subring-Ring*[*of S*],
frule scf-d-pol[*of p Suc d*], *assumption*+,
frule scf-d-pol[*of q Suc d*], *assumption*+, (*erule conjE*)+)
apply (*frule polyn-add1*[*of scf-d R S X p* (*Suc d*) *scf-d R S X q* (*Suc d*)],
assumption+,
rotate-tac -2, *drule sym*,
frule aGroup.ag-pOp-closed[*of R p q*], *assumption*+,
frule polyn-deg-add4 [*of p q Suc d*], *assumption*+,
rotate-tac -5, *drule sym*)
apply *simp*
apply (*rotate-tac 4*, *drule sym*, *simp*)

apply (*rotate-tac* -1, *drule sym*,
frule scf-d-pol[*of p* $\pm$ *q Suc d*], *assumption*+, (*erule conjE*)+,
frule box-equation[*of p* $\pm$ *q polyn-expr R X* (*Suc d*)
(*add-cf S* (*scf-d R S X p* (*Suc d*)) (*scf-d R S X q* (*Suc d*)))
polyn-expr R X (*Suc d*) (*scf-d R S X* (*p* $\pm$ *q*) (*Suc d*))], *assumption*+,
thin-tac p $\pm$ *q* =
polyn-expr R X (*Suc d*) (*scf-d R S X* (*p* $\pm$ *q*) (*Suc d*)))
apply (*frule add-cf-pol-coeff*[*of scf-d R S X p* (*Suc d*)
scf-d R S X q (*Suc d*)], *assumption*+)
apply (*frule pol-expr-unique2*[*of add-cf S* (*scf-d R S X p* (*Suc d*))
(*scf-d R S X q* (*Suc d*)) *scf-d R S X* (*p* $\pm$ *q*) (*Suc d*)], *assumption*+)
apply (*subst add-cf-len*[*of scf-d R S X p* (*Suc d*) *scf-d R S X q* (*Suc d*)],
assumption+)
apply (*thin-tac polyn-expr R X* (*Suc d*) (*scf-d R S X p* (*Suc d*)) = *p*,
thin-tac polyn-expr R X (*Suc d*) (*scf-d R S X q* (*Suc d*)) = *q*,
thin-tac p $\pm$ *q* = *polyn-expr R X* (*Suc d*)
(*add-cf S* (*scf-d R S X p* (*Suc d*)) (*scf-d R S X q* (*Suc d*))),
thin-tac polyn-expr R X (*Suc d*)
(*add-cf S* (*scf-d R S X p* (*Suc d*)) (*scf-d R S X q* (*Suc d*))) =
polyn-expr R X (*Suc d*) (*scf-d R S X* (*p* $\pm$ *q*) (*Suc d*)))
apply *simp*
apply (*thin-tac polyn-expr R X* (*Suc d*) (*scf-d R S X p* (*Suc d*)) = *p*,
thin-tac polyn-expr R X (*Suc d*) (*scf-d R S X q* (*Suc d*)) = *q*,
thin-tac p $\pm$ *q* = *polyn-expr R X* (*Suc d*)
(*add-cf S* (*scf-d R S X p* (*Suc d*)) (*scf-d R S X q* (*Suc d*))))

apply (*simp add*:*add-cf-len*,
thin-tac polyn-expr R X (*Suc d*)
(*add-cf S* (*scf-d R S X p* (*Suc d*)) (*scf-d R S X q* (*Suc d*))) =

```
    polyn-expr R X (Suc d) (scf-d R S X (p ± q) (Suc d)))
 apply (subst  polyn-expr-short[of scf-d R S X p (Suc d) d], assumption,
       simp)
 apply (subst  polyn-expr-short[of scf-d R S X q (Suc d) d], assumption,
       simp)
 apply (subst polyn-add-n[of d snd (scf-d R S X p (Suc d))
             snd (scf-d R S X q (Suc d))])
 apply (simp add:split-pol-coeff, simp add:split-pol-coeff,
       subst polyn-expr-def)
 apply (rule aGroup.nsum-eq, assumption+,
       rule allI, rule impI,
       frule-tac j = j in pol-coeff-mem[of scf-d R S X p (Suc d)],
             simp,
       frule-tac j = j in pol-coeff-mem[of scf-d R S X q (Suc d)],
             simp,
       cut-tac Ring, rule Ring.ring-tOp-closed, assumption+,
       rule Ring.mem-subring-mem-ring[of R S], assumption+,
       frule Ring.ring-is-ag[of S], rule aGroup.ag-pOp-closed[of S],
             assumption+,
       rule Ring.npClose, assumption, simp add:X-mem-R)
 apply (rule allI, rule impI,
       frule-tac j = j in pol-coeff-mem[of scf-d R S X (p ± q) (Suc d)],
       simp, cut-tac Ring,
       subst Ring.ring-tOp-closed, assumption,
       rule Ring.mem-subring-mem-ring[of R S], assumption+,
       rule Ring.npClose, assumption, simp add:X-mem-R,
       simp)
 apply (rule allI, rule impI,
       drule-tac a = j in forall-spec, simp,
       thin-tac  pol-coeff S
         (add-cf S (scf-d R S X p (Suc d)) (scf-d R S X q (Suc d))))
 apply (simp add:add-cf-def)
done

lemma (in PolynRg) hdeg-p-pOp:⟦p ∈ carrier R; q ∈ carrier R;
      deg R S X p ≤ an (Suc d); deg R S X q ≤ an (Suc d)⟧ ⟹
      (hdeg-p R S X (Suc d) p) ± (hdeg-p R S X (Suc d) q) =
                          hdeg-p R S X (Suc d) (p ± q)
apply (cut-tac Ring, frule Ring.ring-is-ag[of R])
apply (cut-tac subring, frule subring-Ring[of S])
apply (frule scf-d-pol[of p Suc d], assumption+,
       frule scf-d-pol[of q Suc d], assumption+,
        (erule conjE)+)
apply (frule polyn-add1[of scf-d R S X p (Suc d) scf-d R S X q (Suc d)],
       assumption+,
       rotate-tac −2, drule sym,
       frule aGroup.ag-pOp-closed[of R p q], assumption+,
       frule polyn-deg-add4 [of p q Suc d], assumption+,
       rotate-tac −5, drule sym)
```

apply *simp*
apply (*rotate-tac −13*, *drule sym*, *simp*)
apply (*rotate-tac −1*, *drule sym*)
apply (*frule scf-d-pol*[*of p ± q Suc d*], *assumption*+, (*erule conjE*)+)
apply (*drule box-equation*[*of p ± q polyn-expr R X* (*Suc d*)
(*add-cf S* (*scf-d R S X p* (*Suc d*)) (*scf-d R S X q* (*Suc d*)))
polyn-expr R X (*Suc d*) (*scf-d R S X* (*p ± q*) (*Suc d*))],
assumption+) **apply** (
thin-tac p ± q = polyn-expr R X (*Suc d*) (*scf-d R S X* (*p ± q*) (*Suc d*)))
apply (*frule add-cf-pol-coeff*[*of scf-d R S X p* (*Suc d*)
scf-d R S X q (*Suc d*)], *assumption*+)
apply (*cut-tac pol-expr-unique2*[*of add-cf S* (*scf-d R S X p* (*Suc d*))
(*scf-d R S X q* (*Suc d*)) *scf-d R S X* (*p ± q*) (*Suc d*)],
simp add:*add-cf-len*)
apply (*drule-tac a = Suc d* **in** *forall-spec*, *simp*)
apply (*simp del*:*npow-suc add*:*hdeg-p-def*)
apply (*rotate-tac −1*, *drule sym*, *simp del*:*npow-suc*)
apply (*subst add-cf-def*, *simp del*:*npow-suc*)
apply (*thin-tac polyn-expr R X* (*Suc d*) (*scf-d R S X p* (*Suc d*)) = *p*,
thin-tac polyn-expr R X (*Suc d*) (*scf-d R S X q* (*Suc d*)) = *q*,
thin-tac polyn-expr R X (*Suc d*) (*add-cf S* (*scf-d R S X p* (*Suc d*))
(*scf-d R S X q* (*Suc d*))) =
polyn-expr R X (*Suc d*) (*scf-d R S X* (*p ± q*) (*Suc d*)),
thin-tac pol-coeff S (*add-cf S* (*scf-d R S X p* (*Suc d*))
(*scf-d R S X q* (*Suc d*))),
thin-tac snd (*scf-d R S X* (*p ± q*) (*Suc d*)) (*Suc d*) =
snd (*add-cf S* (*scf-d R S X p* (*Suc d*)) (*scf-d R S X q* (*Suc d*))) (*Suc d*))
apply (*frule pol-coeff-mem*[*of scf-d R S X p* (*Suc d*) *Suc d*],
simp del:*npow-suc*,
frule pol-coeff-mem[*of scf-d R S X q* (*Suc d*) *Suc d*],
simp del:*npow-suc*)
apply (*simp del*:*npow-suc add*:*Subring-pOp-ring-pOp*)
apply (*frule mem-subring-mem-ring*[*of S snd* (*scf-d R S X p* (*Suc d*)) (*Suc d*)],
assumption,
frule mem-subring-mem-ring[*of S snd* (*scf-d R S X q* (*Suc d*)) (*Suc d*)],
assumption,
cut-tac X-mem-R, *frule Ring.npClose*[*of R X Suc d*], *assumption*+)
apply (*subst Ring.ring-distrib2*[*THEN sym*], *assumption*+, *simp*)

apply (*simp add*:*add-cf-pol-coeff*, *simp*)
apply (*simp add*:*add-cf-len*)
done

lemma (**in** *PolynRg*) *ldeg-p-mOp*:⟦*p ∈ carrier R*; *deg R S X p ≤ an* (*Suc d*)⟧ ⟹

$-_a$ (*ldeg-p R S X d p*) = *ldeg-p R S X d* ($-_a$ *p*)
apply (*cut-tac Ring*, *frule Ring.ring-is-ag*[*of R*],
cut-tac subring, *frule subring-Ring*[*of S*],
frule scf-d-pol[*of p Suc d*], *assumption*+, (*erule conjE*)+,

frule aGroup.ag-mOp-closed[*of R p*], *assumption*,
frule scf-d-pol[*of* $-_a$ *p Suc d*])
apply (*simp add*:*deg-minus-eq1*, (*erule conjE*)+)
apply (*frule polyn-minus*[*of scf-d R S X p* (*Suc d*) *Suc d*], *simp*)
apply (*drule box-equation*[*of* $-_a$ *p polyn-expr R X* (*Suc d*)
(*scf-d R S X* ($-_a$ *p*) (*Suc d*))
polyn-expr R X (*Suc d*) (*fst* (*scf-d R S X p* (*Suc d*)),
$\lambda j.$ $-_{a\,S}$ *snd* (*scf-d R S X p* (*Suc d*)) *j*)])
apply (*rotate-tac 8*, *drule sym*, *simp*,
thin-tac $-_a$ *polyn-expr R X* (*Suc d*) (*scf-d R S X p* (*Suc d*)) =
polyn-expr R X (*Suc d*)
(*fst* (*scf-d R S X p* (*Suc d*)), $\lambda j.$ $-_{a\,S}$ *snd* (*scf-d R S X p* (*Suc d*)) *j*))
apply *simp*
apply (*frule pol-expr-unique2*[*of scf-d R S X* ($-_a$ *p*) (*Suc d*)
(*Suc d*, $\lambda j.$ $-_{a\,S}$ *snd* (*scf-d R S X p* (*Suc d*)) *j*)])
apply (*subst pol-coeff-def*, *rule allI*, *rule impI*, *simp*)
apply (*frule-tac j* = *j* **in** *pol-coeff-mem*[*of scf-d R S X p* (*Suc d*)],
simp,
frule Ring.ring-is-ag[*of S*],
rule aGroup.ag-mOp-closed[*of S*], *assumption*+, *simp*)
apply *simp*

apply (*simp add*:*ldeg-p-def*)
apply (*subst polyn-minus*[*of scf-d R S X p* (*Suc d*) *d*], *assumption*, *simp*,
simp)
apply (*subst polyn-expr-short*[*of* (*Suc d*,
$\lambda j.$ $-_{a\,S}$ *snd* (*scf-d R S X p* (*Suc d*)) *j*) *d*])
apply (*subst pol-coeff-def*, *rule allI*, *rule impI*, *simp*,
frule-tac j = *j* **in** *pol-coeff-mem*[*of scf-d R S X p* (*Suc d*)],
simp,
frule Ring.ring-is-ag[*of S*],
rule aGroup.ag-mOp-closed[*of S*], *assumption*+, *simp*)
apply (*subst polyn-expr-short*[*of scf-d R S X* ($-_a$ *p*) (*Suc d*) *d*],
assumption, *simp*)
apply (*cut-tac pol-expr-unique2*[*of* (*d*, *snd* (*Suc d*,
$\lambda j.$ $-_{a\,S}$ *snd* (*scf-d R S X p* (*Suc d*)) *j*))
(*d*, *snd* (*scf-d R S X* ($-_a$ *p*) (*Suc d*)))])
apply (*thin-tac p* = *polyn-expr R X* (*Suc d*) (*scf-d R S X p* (*Suc d*)),
thin-tac polyn-expr R X (*Suc d*) (*scf-d R S X* ($-_a$ *p*) (*Suc d*)) =
polyn-expr R X (*Suc d*) (*Suc d*, $\lambda j.$ $-_{a\,S}$ *snd* (*scf-d R S X p* (*Suc d*)) *j*),
simp)
apply (*subst pol-coeff-def*, *rule allI*, *rule impI*, *simp*,
frule Ring.ring-is-ag[*of S*], *rule aGroup.ag-mOp-closed*, *assumption*,
simp add:*pol-coeff-mem*)
apply (*subst pol-coeff-def*, *rule allI*, *rule impI*, *simp*,
frule Ring.ring-is-ag[*of S*], *rule aGroup.ag-mOp-closed*, *assumption*,
simp add:*pol-coeff-mem*)
apply *simp*
done

lemma (**in** *PolynRg*) *hdeg-p-mOp*:⟦$p \in$ *carrier* R;*deg* $R\ S\ X\ p \leq$ *an* (*Suc* d)⟧
$\Longrightarrow -_a$ (*hdeg-p* $R\ S\ X$ (*Suc* d) p) = *hdeg-p* $R\ S\ X$ (*Suc* d) ($-_a\ p$)
apply (*cut-tac Ring*, *frule Ring.ring-is-ag*[*of R*],
cut-tac subring, *frule subring-Ring*[*of S*],
frule scf-d-pol[*of p Suc d*], *assumption*+, (*erule conjE*)+,
frule aGroup.ag-mOp-closed[*of R p*], *assumption*) **apply** (
frule scf-d-pol[*of* $-_a$ *p Suc d*])
apply (*simp add*:*deg-minus-eq1*, (*erule conjE*)+)
apply (*frule polyn-minus*[*of scf-d R S X p* (*Suc d*) *Suc d*], *simp*)
apply (*drule box-equation*[*of* $-_a$ *p polyn-expr R X* (*Suc d*)
(*scf-d R S X* ($-_a$ *p*) (*Suc d*))
polyn-expr R X (*Suc d*) (*fst* (*scf-d R S X p* (*Suc d*)),
$\lambda j.\ -_{a\,S}$ *snd* (*scf-d R S X p* (*Suc d*)) *j*)])
apply (*rotate-tac 8*, *drule sym*, *simp*,
thin-tac $-_a$ *polyn-expr R X* (*Suc d*) (*scf-d R S X p* (*Suc d*)) =
polyn-expr R X (*Suc d*)
(*fst* (*scf-d R S X p* (*Suc d*)), $\lambda j.\ -_{a\,S}$ *snd* (*scf-d R S X p* (*Suc d*)) *j*))
apply *simp*

apply (*frule pol-expr-unique2*[*of scf-d R S X* ($-_a$ *p*) (*Suc d*)
(*Suc d*, $\lambda j.\ -_{a\,S}$ *snd* (*scf-d R S X p* (*Suc d*)) *j*)])
apply (*subst pol-coeff-def*, *rule allI*, *rule impI*, *simp*)
apply (*frule-tac j* = *j* **in** *pol-coeff-mem*[*of scf-d R S X p* (*Suc d*)],
simp,
frule Ring.ring-is-ag[*of S*],
rule aGroup.ag-mOp-closed[*of S*], *assumption*+, *simp*)
apply *simp*
apply (*drule-tac a* = *Suc d* **in** *forall-spec*, *simp*)
apply (*simp del*:*npow-suc add*:*hdeg-p-def*)
apply (*thin-tac p* = *polyn-expr R X* (*Suc d*) (*scf-d R S X p* (*Suc d*)),
thin-tac polyn-expr R X (*Suc d*) (*scf-d R S X* ($-_a$ *p*) (*Suc d*)) =
polyn-expr R X (*Suc d*) (*Suc d*, $\lambda j.\ -_{a\,S}$ *snd* (*scf-d R S X p* (*Suc d*)) *j*),
thin-tac snd (*scf-d R S X* ($-_a$ *p*) (*Suc d*)) (*Suc d*) =
$-_{a\,S}$ *snd* (*scf-d R S X p* (*Suc d*)) (*Suc d*))
apply (*frule pol-coeff-mem*[*of scf-d R S X p* (*Suc d*) *Suc d*], *simp*,
frule mem-subring-mem-ring[*of S snd* (*scf-d R S X p* (*Suc d*)) (*Suc d*)],
assumption+,
frule Ring.npClose[*of R X Suc d*], *simp add*:*X-mem-R*)
apply (*subst Ring.ring-inv1-1*[*of R*], *assumption*+)
apply (*simp del*:*npow-suc add*:*Subring-minus-ring-minus*)
done

4.14.1 Multiplication of polynomials

lemma (**in** *PolynRg*) *deg-mult-pols*:⟦*Idomain S*;
$p \in$ *carrier* R; $p \neq \mathbf{0}$; $q \in$ *carrier* R; $q \neq \mathbf{0}$ ⟧ $\Longrightarrow$
$p \cdot_r q \neq \mathbf{0} \wedge$
deg-n $R\ S\ X$ ($p \cdot_r q$) = *deg-n* $R\ S\ X\ p$ + *deg-n* $R\ S\ X\ q$

apply (*frule Idomain.idom-is-ring*[*of S*],
frule-tac x = p **and** *y = q* **in** *ring-tOp-closed*, *assumption*+)
apply (*frule s-cf-expr*[*of p*], *assumption*,
frule s-cf-expr[*of q*], *assumption*, (*erule conjE*)+)
apply (*frule-tac c = s-cf R S X p* **and** *d = s-cf R S X q* **in**
polyn-expr-tOp-c, *assumption*, *erule exE*, (*erule conjE*)+)
apply (*drule sym*, *drule sym*, *simp*,
thin-tac polyn-expr R X (*fst* (*s-cf R S X q*)) (*s-cf R S X q*) = *q*,
thin-tac polyn-expr R X (*fst* (*s-cf R S X p*)) (*s-cf R S X p*) = *p*)
apply (*rotate-tac −1*, *drule sym*,
frule ring-tOp-closed[*of p q*], *assumption*+,
frule pol-coeff-mem[*of s-cf R S X p fst* (*s-cf R S X p*)], *simp*,
frule pol-coeff-mem[*of s-cf R S X q fst* (*s-cf R S X q*)], *simp*,
frule-tac x = snd (*s-cf R S X p*) (*fst* (*s-cf R S X p*)) **and**
y = snd (*s-cf R S X q*) (*fst* (*s-cf R S X q*)) **in**
Idomain.idom-tOp-nonzeros[*of S*], *assumption*+,
frule-tac c = e **in** *coeff-nonzero-polyn-nonzero*[*of* -
deg-n R S X p + deg-n R S X q], *simp*)
apply (*simp add*:*s-cf-deg*, *simp add*:*s-cf-deg*)
apply (*cut-tac n = fst* (*s-cf R S X p*) + *fst* (*s-cf R S X q*) **in** *le-refl*)
apply (*subgoal-tac* ∃*j*≤*fst* (*s-cf R S X p*) +
fst (*s-cf R S X q*). *snd e j* ≠ $\mathbf{0}_S$, *simp*)
apply (*cut-tac c = e* **in** *pol-deg-n*[*of p* ·$_r$ *q* -
fst (*s-cf R S X p*) + *fst* (*s-cf R S X q*)], *simp*+)
apply (*thin-tac* (*polyn-expr R X* (*fst* (*s-cf R S X p*) +
fst (*s-cf R S X q*)) *e* ≠ $\mathbf{0}$) =
(∃*j*≤*fst* (*s-cf R S X p*) + *fst* (*s-cf R S X q*). *snd e j* ≠ $\mathbf{0}_S$),
thin-tac p ·$_r$ *q* =
polyn-expr R X (*fst* (*s-cf R S X p*) + *fst* (*s-cf R S X q*)) *e*,
thin-tac polyn-expr R X (*fst* (*s-cf R S X p*) + *fst* (*s-cf R S X q*)) *e*
∈ *carrier R*,
thin-tac snd (*s-cf R S X p*) (*fst* (*s-cf R S X p*)) ∈ *carrier S*,
thin-tac snd (*s-cf R S X q*) (*fst* (*s-cf R S X q*)) ∈ *carrier S*)
apply (*drule sym*, *drule sym*, *simp*)
apply (*cut-tac n = fst e* **in** *le-refl*, *blast*)
done

lemma (**in** *PolynRg*) *deg-mult-pols1*:⟦*Idomain S*; *p* ∈ *carrier R*; *q* ∈ *carrier R*⟧
⟹
deg R S X (*p* ·$_r$ *q*) = *deg R S X p* + *deg R S X q*
apply (*case-tac p* = $\mathbf{0}_R$, *simp add*:*ring-times-0-x*, *simp add*:*deg-def*,
rule impI)
apply (*simp add*:*an-def*)
apply (*case-tac q* = $\mathbf{0}_R$, *simp add*:*ring-times-x-0*, *simp add*:*deg-def*)
apply (*simp add*:*an-def*)

apply (*frule deg-mult-pols*[*of p q*], *assumption*+, *erule conjE*)
apply (*frule Idomain.idom-is-ring*[*of S*])
apply (*frule ring-tOp-closed*[*of p q*], *assumption*+)

apply (*simp add*:*deg-an an-def a-zpz*)
done

lemma (**in** *PolynRg*) *const-times-polyn*:⟦*Idomain S*; *c* ∈ *carrier S*; *c* ≠ $\mathbf{0}_S$;
p ∈ *carrier R*; *p* ≠ **0**⟧ ⟹ (*c* ·$_r$ *p*) ≠ **0** ∧
deg-n R S X (*c* ·$_r$ *p*) = *deg-n R S X p*
apply (*frule Idomain.idom-is-ring*[*of S*],
cut-tac subring,
frule mem-subring-mem-ring[*of S c*], *assumption*+,
simp add:*Subring-zero-ring-zero*)
apply (*frule deg-mult-pols*[*of c p*], *assumption*+,
erule conjE, *simp*,
simp add:*pol-of-deg0*[*THEN sym*, *of c*])
done

lemma (**in** *PolynRg*) *p-times-monomial-nonzero*:⟦*p* ∈ *carrier R*; *p* ≠ **0**⟧ ⟹
(*X*^$^{R\ j}$) ·$_r$ *p* ≠ **0**
apply (*cut-tac subring*, *frule subring-Ring*)
apply (*frule s-cf-expr*[*of p*], *assumption*+, (*erule conjE*)+)
apply (*frule low-deg-terms-zero1*[*THEN sym*, *of s-cf R S X p j*])
apply (*drule sym*, *simp*,
thin-tac X^$^{R\ j}$ ·$_r$ *p* =
polyn-expr R X (*fst* (*s-cf R S X p*) + *j*) (*ext-cf S j* (*s-cf R S X p*)),
thin-tac polyn-expr R X (*fst* (*s-cf R S X p*)) (*s-cf R S X p*) = *p*)
apply (*frule ext-cf-pol-coeff*[*of s-cf R S X p j*])
apply(*frule coeff-nonzero-polyn-nonzero*[*of ext-cf S j* (*s-cf R S X p*)
fst (*ext-cf S j* (*s-cf R S X p*))],
simp)
apply (*simp add*:*ext-cf-len add.commute*[*of j*],
thin-tac (*polyn-expr R X* (*fst* (*s-cf R S X p*) + *j*)
(*ext-cf S j* (*s-cf R S X p*)) ≠ **0**) =
(∃*ja*≤*fst* (*s-cf R S X p*) + *j*. *snd* (*ext-cf S j* (*s-cf R S X p*)) *ja* ≠ $\mathbf{0}_S$))
apply (*cut-tac ext-cf-hi*[*of s-cf R S X p j*], *simp*,
thin-tac snd (*s-cf R S X p*) (*fst* (*s-cf R S X p*)) =
snd (*ext-cf S j* (*s-cf R S X p*)) (*j* + *fst* (*s-cf R S X p*)),
simp add:*add.commute*[*of* - *j*])
apply (*cut-tac n* = *j* + *fst* (*s-cf R S X p*) **in** *le-refl*, *blast*)
apply *assumption*
done

lemma (**in** *PolynRg*) *p-times-monomial-nonzero1*:⟦*Idomain S*; *p* ∈ *carrier R*;
p ≠ **0**; *c* ∈ *carrier S*; *c* ≠ $\mathbf{0}_S$⟧ ⟹(*c* ·$_r$ (*X*^$^{R\ j}$)) ·$_r$ *p* ≠ **0**
apply (*frule Idomain.idom-is-ring*[*of S*],
cut-tac subring,
cut-tac X-mem-R ,
frule mem-subring-mem-ring[*of S c*], *assumption*+,
frule npClose[*of X j*])

apply (*simp add*:*ring-tOp-commute*[*of c*],
simp add:*ring-tOp-assoc*,
frule const-times-polyn[*of c p*], *assumption*+,
erule conjE,
frule ring-tOp-closed[*of c p*], *assumption*+,
simp add:*p-times-monomial-nonzero*[*of c* $\cdot_r$ *p*])
done

lemma (**in** *PolynRg*) *polyn-ring-integral*:*Idomain S* = *Idomain R*
apply (*cut-tac subring*, *frule subring-Ring*)
apply (*rule iffI*)
apply (*subst Idomain-def*)
apply (*cut-tac Ring*, *simp*)

apply (*rule Idomain-axioms.intro*,
rule contrapos-pp, *simp*+, *erule conjE*,
frule-tac p = *a* **and** *q* = *b* **in** *deg-mult-pols*,
assumption+, *erule conjE*, *simp*)

apply (*subst Idomain-def*)
apply (*cut-tac Ring*, *simp*)
apply (*rule Idomain-axioms.intro*,
rule contrapos-pp, *simp*+, *erule conjE*)
apply (*simp add*:*Subring-tOp-ring-tOp*)
apply (*frule-tac x* = *a* **in** *mem-subring-mem-ring*[*of S*], *assumption*,
frule-tac x = *b* **in** *mem-subring-mem-ring*[*of S*], *assumption*)
apply (*frule-tac a* = *a* **and** *b* = *b* **in** *Idomain.idom*[*of R*], *assumption*+)
apply (*simp add*:*Subring-zero-ring-zero*)
apply (*erule disjE*, *simp add*:*Subring-zero-ring-zero*)
apply (*simp add*:*Subring-zero-ring-zero*)
done

lemma (**in** *PolynRg*) *deg-to-X-d*:*Idomain S* $\Longrightarrow$ *deg-n R S X* (*X* ^*R d*) = *d*
apply (*cut-tac subring*, *frule subring-Ring*,
cut-tac polyn-ring-S-nonzero,
cut-tac X-mem-R,
cut-tac polyn-ring-X-nonzero,
cut-tac polyn-ring-integral)
apply (*induct-tac d*)
apply (*cut-tac ring-one*,
simp add:*Subring-one-ring-one*[*THEN sym*],
simp add:*Subring-zero-ring-zero*)
apply (*subst pol-of-deg0*[*of* $1_{r\,S}$], *assumption*+, *simp add*:*Ring.ring-one*[*of S*])
apply *simp*
apply (*subst deg-mult-pols*, *assumption*+,
simp add:*npClose*,
rule Idomain.idom-potent-nonzero, *assumption*+)
apply (*simp add*:*deg-n-of-X*)
done

4.14.2 Degree with value in *aug-minf*

lemma (**in** *PolynRg*) *nonzero-deg-pos*:⟦$p \in$ *carrier* R; $p \neq \mathbf{0}$⟧ $\Longrightarrow$
$0 \leq$ *deg* R S X p
by (*simp add*:*deg-def*)

lemma (**in** *PolynRg*) *deg-minf-pol-0*:$p \in$ *carrier* $R \Longrightarrow$
(*deg* R S X $p = -\infty$) = ($p = \mathbf{0}$)
apply (*rule iffI*)
apply (*rule contrapos-pp*, *simp*+,
frule nonzero-deg-pos[*of p*], *assumption*+,
simp add:*deg-def an-def*)
apply (*simp add*:*deg-def*)
done

lemma (**in** *PolynRg*) *pol-nonzero*:$p \in$ *carrier* $R \Longrightarrow$
(0 $\leq$ *deg* R S X p) = ($p \neq \mathbf{0}$)
apply (*rule iffI*)
apply (*rule contrapos-pp*, *simp*+, *simp add*:*deg-def*,
cut-tac minf-le-any[*of 0*], *frule ale-antisym*[*of 0* $-\infty$],
assumption+,
simp only:*an-0*[*THEN sym*], *simp only*:*an-def*, *simp del*:*of-nat-0*)
apply (*simp add*:*deg-def*)
done

lemma (**in** *PolynRg*) *minus-deg-in-aug-minf*:⟦$p \in$ *carrier* R; $p \neq \mathbf{0}$⟧ $\Longrightarrow$
$-$ (*deg* R S X p) $\in Z_{-\infty}$
apply (*frule deg-in-aug-minf*[*of p*],
frule pol-nonzero[*THEN sym*, *of p*],
simp add:*aug-minf-def*,
rule contrapos-pp, *simp*+,
cut-tac a-minus-minus[*of deg R S X p*], *simp*)

apply (*thin-tac* $-$ *deg* R S X $p = \infty$, *frule sym*,
thin-tac $-\infty =$ *deg* R S X p,
frule deg-minf-pol-0[*of p*], *simp*)
done

lemma (**in** *PolynRg*) *deg-of-X*:*deg* R S X $X = 1$
apply (*cut-tac X-mem-R*,
cut-tac polyn-ring-X-nonzero,
cut-tac subring)
apply (*simp add*:*deg-def*, *simp only*:*an-1*[*THEN sym*],
rule nat-eq-an-eq, *simp add*:*deg-n-of-X*)
done

lemma (**in** *PolynRg*) *pol-deg-0*:⟦$p \in$ *carrier* R; $p \neq \mathbf{0}$⟧
$\Longrightarrow$ (*deg* R S X $p = 0$) = ($p \in$ *carrier* S)
apply (*simp add*:*deg-def*, *simp only*:*an-0*[*THEN sym*],
rule iffI,

frule an-inj[*of deg-n R S X p 0*], *simp*,
simp add:*pol-of-deg0*,
rule nat-eq-an-eq, *simp add*:*pol-of-deg0*[*of p*])
done

lemma (**in** *PolynRg*) *deg-of-X2n*:*Idomain S* $\Longrightarrow$ *deg R S X* $(X \hat{}^{R}\ n) = an\ n$
apply (*frule Idomain.idom-is-ring*[*of S*])
apply (*cut-tac subring*,
cut-tac X-mem-R,
cut-tac polyn-ring-X-nonzero,
cut-tac polyn-ring-integral, *simp*)
apply (*induct-tac n*)
apply *simp*
apply (*simp add*:*Subring-one-ring-one*[*THEN sym*, *of S*])
apply (*subst pol-deg-0*[*of* $1_{r\,S}$])
apply (*simp add*:*Subring-one-ring-one*, *simp add*:*ring-one*)
apply (*simp add*:*Subring-one-ring-one*[*of S*] *polyn-ring-nonzero*)
apply (*simp add*:*Ring.ring-one*[*of S*])

apply (*frule-tac n* = *n* **in** *npClose*[*of X*])
apply (*simp add*:*deg-def*)
apply (*simp add*:*Idomain.idom-potent-nonzero*,
frule-tac n = *Suc n* **in** *Idomain.idom-potent-nonzero*[*of R X*],
assumption+, *simp*)
apply (*rule nat-eq-an-eq*)
apply (*frule-tac n* = *n* **in** *Idomain.idom-potent-nonzero*[*of R X*], *assumption*+)
apply (*frule-tac n* = *deg-n R S X* $(X \hat{}^{R}\ n)$ **and** *m* = *n* **in** *an-inj*,
thin-tac an (*deg-n R S X* $(X \hat{}^{R}\ n)$) = *an n*)
apply (*cut-tac deg-of-X*)
apply (*simp add*:*deg-def*, *simp only*:*an-1*[*THEN sym*])
apply (*frule-tac n* = *deg-n R S X X* **and** *m* = *1* **in** *an-inj*)
apply (*simp add*:*deg-mult-pols*)
done

lemma (**in** *PolynRg*) *add-pols-nonzero*:⟦*p* ∈ *carrier R*; *q* ∈ *carrier R*;
(*deg R S X p*) ≠ (*deg R S X q*)⟧ $\Longrightarrow$ *p* ± *q* ≠ **0**
apply (*cut-tac ring-is-ag*,
cut-tac subring,
frule subring-Ring)
apply (*case-tac p* = $\mathbf{0}_R$, *simp add*:*deg-minf-pol-0*[*THEN sym*],
simp add:*aGroup.ag-l-zero*, *rule contrapos-pp*, *simp*+,
case-tac q = $\mathbf{0}_R$, *simp add*:*aGroup.ag-r-zero*)
apply (*simp add*:*deg-def*,
simp only:*aneq-natneq*[*of deg-n R S X p deg-n R S X q*],
cut-tac less-linear[*of deg-n R S X p deg-n R S X q*], *simp*,
erule disjE,
rule less-deg-add-nonzero[*of p q*],
assumption+,
frule less-deg-add-nonzero[*of q p*], *assumption*+,

simp add:*aGroup.ag-pOp-commute*)
done

lemma (**in** *PolynRg*) *deg-pols-add1*:⟦$p \in$ *carrier* R; $q \in$ *carrier* R;
(*deg* R S X p) < (*deg* R S X q)⟧ ⟹
deg R S X ($p \pm q$) = *deg* R S X q
apply (*cut-tac ring-is-ag*,
case-tac $p = \mathbf{0}_R$, *simp add*:*deg-def aGroup.ag-l-zero*,
case-tac $q = \mathbf{0}_R$, *simp add*:*deg-def*)
apply (*rule impI*) **apply** (*simp add*:*an-def z-neq-minf*)
apply (*fold an-def*,
frule aless-imp-le[*of an* (*deg-n* R S X p) $-\infty$],
cut-tac minf-le-any[*of an* (*deg-n* R S X p)],
frule ale-antisym[*of an* (*deg-n* R S X p) $-\infty$], *assumption*+,
simp add:*an-neq-minf*)
apply (*simp add*:*deg-def*, *simp add*:*aless-nat-less*,
frule less-deg-add-nonzero[*of p q*], *assumption*+,
simp, *frule polyn-deg-add1*[*of p q*], *assumption*+, *simp*)
done

lemma (**in** *PolynRg*) *deg-pols-add2*:⟦$p \in$ *carrier* R; $q \in$ *carrier* R;
(*deg* R S X p) = (*deg* R S X q)⟧ ⟹
deg R S X ($p \pm q$) ≤ (*deg* R S X q)
apply (*cut-tac ring-is-ag*,
cut-tac subring, *frule subring-Ring*)
apply (*case-tac* $p = \mathbf{0}_R$, *simp add*:*aGroup.ag-l-zero*)
apply (*case-tac* $q = \mathbf{0}_R$, *simp add*:*aGroup.ag-r-zero*)
apply (*simp add*:*deg-def*,
frule an-inj[*of deg-n R S X p deg-n R S X q*], *simp*,
rule impI, *subst ale-natle*, *simp add*:*polyn-deg-add2*)
done

lemma (**in** *PolynRg*) *deg-pols-add3*:⟦$p \in$ *carrier* R; $q \in$ *carrier* R;
(*deg* R S X p) ≤ *an* n; (*deg* R S X q) ≤ *an* n⟧ ⟹
deg R S X ($p \pm q$) ≤ *an* n
apply (*case-tac* (*deg* R S X p) = (*deg* R S X q),
frule deg-pols-add2[*of p q*], *assumption*+,
simp)
apply (*cut-tac aless-linear*[*of deg R S X p deg R S X q*],
simp, *thin-tac deg R S X p* ≠ *deg R S X q*,
erule disjE, *simp add*:*deg-pols-add1*,
cut-tac ring-is-ag, *simp add*:*aGroup.ag-pOp-commute*[*of R p q*],
simp add:*deg-pols-add1*)
done

lemma (**in** *PolynRg*) *const-times-polyn1*:⟦*Idomain* S; $p \in$ *carrier* R; $c \in$ *carrier* S;
$c \neq \mathbf{0}_S$⟧ ⟹ *deg* R S X ($c \cdot_r p$) = *deg* R S X p
apply (*frule Idomain.idom-is-ring*,
cut-tac subring,

frule mem-subring-mem-ring[*of S c*], *assumption*,
simp add:*Subring-zero-ring-zero*)
apply (*subst deg-mult-pols1*[*of c p*], *assumption*+,
simp add: *pol-deg-0*[*THEN sym*, *of c*],
simp add:*aadd-0-l*)
done

4.15 Homomorphism of polynomial rings

definition
cf-h :: $('a \Rightarrow 'b) \Rightarrow nat \times (nat \Rightarrow 'a) \Rightarrow nat \times (nat \Rightarrow 'b)$ **where**
cf-h f = $(\lambda c.\ (fst\ c,\ cmp\ f\ (snd\ c)))$

definition
polyn-Hom :: [$('a, 'm)$ *Ring-scheme*, $('a, 'm1)$ *Ring-scheme*, $'a$,
$('b, 'n)$ *Ring-scheme*, $('b, 'n1)$ *Ring-scheme*, $'b$] $\Rightarrow$
$('a \Rightarrow 'b)$ *set*
((*pHom* - - -, - - -) [67,67,67,67,67,68]67) **where**
pHom R S X, *A B Y* = $\{f.\ f \in rHom\ R\ A \wedge f`(carrier\ S) \subseteq carrier\ B \wedge$
$f\ X = Y\}$

definition
erh :: [$('a, 'm)$ *Ring-scheme*, $('a, 'm1)$ *Ring-scheme*, $'a$,
$('b, 'n)$ *Ring-scheme*, $('b, 'n1)$ *Ring-scheme*, $'b$, $'a \Rightarrow 'b$,
nat, $nat \times (nat \Rightarrow 'a)$] $\Rightarrow$ $'b$ **where**
erh R S X A B Y f n c = *polyn-expr A Y n* (*cf-h f c*)

lemma (**in** *PolynRg*) *cf-h-len*:$[\![$*PolynRg A B Y*; *f* $\in$ *rHom S B*;
pol-coeff S c$]\!] \Longrightarrow$ *fst* (*cf-h f c*) = *fst c*
by (*simp add*:*cf-h-def*)

lemma (**in** *PolynRg*) *cf-h-coeff*:$[\![$*PolynRg A B Y*; *f* $\in$ *rHom S B*;
pol-coeff S c$]\!] \Longrightarrow$ *pol-coeff B* (*cf-h f c*)
apply (*subst pol-coeff-def*)
apply (*rule allI*, *rule impI*, *simp add*:*cf-h-len cf-h-def*)
apply (*frule-tac j* = *j* **in** *pol-coeff-mem*[*of c*], *assumption*)
apply (*simp add*:*cmp-def rHom-mem*)
done

lemma (**in** *PolynRg*) *cf-h-cmp*:$[\![$*PolynRg A B Y*; *pol-coeff S* (n, f); *h* $\in$ *rHom S B*;
j $\leq$ *n*$]\!] \Longrightarrow$
(*snd* (*cf-h h* (n, f))) *j* = (*cmp h f*) *j*
by (*simp add*:*cf-h-def*)

lemma (**in** *PolynRg*) *cf-h-special-cf*:$[\![$*PolynRg A B Y*; *h* $\in$ *rHom S B*$]\!] \Longrightarrow$
polyn-expr A Y (*Suc 0*) (*cf-h h* (*ext-cf S* (*Suc 0*) (C_0 *S*))) =
polyn-expr A Y (*Suc 0*) (*ext-cf B* (*Suc 0*) (C_0 *B*))

apply (*cut-tac subring*, *frule subring-Ring*,
frule PolynRg.is-Ring[*of A B Y*],
frule PolynRg.subring[*of A B Y*],
frule Ring.subring-Ring[*of A B*], *assumption*)
apply (*cut-tac special-cf-pol-coeff*,
frule ext-cf-pol-coeff[*of* C_0 *S Suc 0*],
frule cf-h-coeff[*of A B Y h ext-cf S* (*Suc 0*) (C_0 *S*)], *assumption*+)
apply (*frule PolynRg.special-cf-pol-coeff*,
frule PolynRg.ext-cf-pol-coeff[*of A B Y* C_0 *B Suc 0*], *assumption*)
apply (*frule PolynRg.pol-expr-unique2*[*of A B Y*
cf-h h (*ext-cf S* (*Suc 0*) (C_0 *S*)) *ext-cf B* (*Suc 0*) (C_0 *B*)],
assumption+,
simp add:*cf-h-len PolynRg.ext-cf-len*,
simp add:*ext-cf-len special-cf-len PolynRg.special-cf-len*,
simp add:*cf-h-len PolynRg.ext-cf-len*,
simp add:*ext-cf-len special-cf-len PolynRg.special-cf-len*,
thin-tac (*polyn-expr A Y* (*Suc 0*) (*cf-h h* (*ext-cf S* (*Suc 0*) (C_0 *S*))) =
polyn-expr A Y (*Suc 0*) (*ext-cf B* (*Suc 0*) (C_0 *B*))) =
(∀ *j*≤*Suc 0*.
snd (*cf-h h* (*ext-cf S* (*Suc 0*) (C_0 *S*))) *j* =
snd (*ext-cf B* (*Suc 0*) (C_0 *B*)) *j*),
thin-tac pol-coeff S (C_0 *S*),
thin-tac pol-coeff S (*ext-cf S* (*Suc 0*) (C_0 *S*)),
thin-tac pol-coeff B (*cf-h h* (*ext-cf S* (*Suc 0*) (C_0 *S*))))
apply (*rule allI*, *rule impI*)
apply (*case-tac j* = *0*, *simp add*:*cf-h-def cmp-def ext-cf-def sliden-def*)
apply (*simp add*:*rHom-0-0*)
apply (*simp*)
apply (*frule-tac n* = *j* **in** *Suc-leI*[*of 0*],
frule-tac m = *j* **and** *n* = *Suc 0* **in** *le-antisym*, *assumption*+,
thin-tac j ≤ *Suc 0*, *thin-tac Suc 0* ≤ *j*,
simp add:*cf-h-def cmp-def ext-cf-def sliden-def special-cf-def*,
simp add:*rHom-one*)
done

lemma (**in** *PolynRg*) *polyn-Hom-coeff-to-coeff*:
⟦*PolynRg A B Y*; *f* ∈ *pHom R S X*, *A B Y*; *pol-coeff S c*⟧
⟹ *pol-coeff B* (*cf-h f c*)
apply (*subst pol-coeff-def*)
apply (*rule allI*, *rule impI*, *simp add*:*cf-h-len cf-h-def*)
apply (*frule-tac j* = *j* **in** *pol-coeff-mem*[*of c*], *assumption*)
apply (*simp add*:*cmp-def polyn-Hom-def*, (*erule conjE*)+)
apply (*simp add*:*image-def*)
apply (*rule subsetD*[*of* {*y*. ∃ *x*∈*carrier S*. *y* = *f x*} *carrier B*], *assumption*,
simp)
apply *blast*
done

lemma (**in** *PolynRg*) *cf-h-len1*:⟦*PolynRg A B Y*; *h* ∈ *rHom S B*;

$f \in pHom\ R\ S\ X,\ A\ B\ Y;\ \forall x{\in}carrier\ S.\ f\ x = h\ x;\ pol\text{-}coeff\ S\ c\rrbracket \Longrightarrow$
$fst\ (cf\text{-}h\ f\ c) = fst\ (cf\text{-}h\ h\ c)$
by (*simp add*:*cf-h-def*)

lemma (**in** *PolynRg*) *cf-h-len2*:⟦*PolynRg A B Y*; $f \in pHom\ R\ S\ X,\ A\ B\ Y$;
$pol\text{-}coeff\ S\ c\rrbracket \Longrightarrow fst\ (cf\text{-}h\ f\ c) = fst\ c$
by (*simp add*:*cf-h-def*)

lemma (**in** *PolynRg*) *cmp-pol-coeff*:⟦$f \in rHom\ S\ B$;
$pol\text{-}coeff\ S\ (n,\ c)\rrbracket \Longrightarrow pol\text{-}coeff\ B\ (n,(cmp\ f\ c))$
apply (*simp add*:*pol-coeff-def*,
rule allI, *rule impI*, *simp add*:*cmp-def*,
frule-tac $a = j$ **in** *forall-spec*, *simp*,
thin-tac $\forall j{\leq}n.\ c\ j \in carrier\ S$)
apply (*simp add*:*rHom-mem*)
done

lemma (**in** *PolynRg*) *cmp-pol-coeff-e*:⟦*PolynRg A B Y*; $f \in pHom\ R\ S\ X,\ A\ B\ Y$;

$pol\text{-}coeff\ S\ (n,\ c)\rrbracket \Longrightarrow pol\text{-}coeff\ B\ (n,\ (cmp\ f\ c))$
apply (*subst pol-coeff-def*)
apply (*rule allI*, *rule impI*, *simp*)
apply (*frule-tac* $j = j$ **in** *pol-coeff-mem*[*of* $(n,\ c)$], *simp*)
apply (*simp add*:*polyn-Hom-def cmp-def*, (*erule conjE*)+)
apply (*simp add*:*image-def*)
apply (*rule-tac* $c = f\ (c\ j)$ **in** *subsetD*[*of* $\{y.\ \exists x{\in}carrier\ S.\ y = f\ x\}$
carrier B], *assumption*+)
apply (*simp*, *blast*)
done

lemma (**in** *PolynRg*) *cf-h-pol-coeff*:⟦*PolynRg A B Y*; $h \in rHom\ S\ B$;
$pol\text{-}coeff\ S\ (n,\ f)\rrbracket \Longrightarrow cf\text{-}h\ h\ (n,\ f) = (n,\ cmp\ h\ f)$
by (*simp add*:*cf-h-def*)

lemma (**in** *PolynRg*) *cf-h-polyn*:⟦*PolynRg A B Y*; $h \in rHom\ S\ B$;
$pol\text{-}coeff\ S\ (n,\ f)\rrbracket \Longrightarrow$
$polyn\text{-}expr\ A\ Y\ n\ (cf\text{-}h\ h\ (n,\ f)) = polyn\text{-}expr\ A\ Y\ n\ (n,\ (cmp\ h\ f))$
apply (*frule cf-h-coeff*[*of A B Y h* $(n,\ f)$], *assumption*+,
frule cmp-pol-coeff[*of h B n f*], *assumption*+)
apply (*rule PolynRg.polyn-exprs-eq*[*of A B Y cf-h h* $(n,\ f)$ $(n,\ cmp\ h\ f)$ n],
assumption+,
simp add:*cf-h-len*,
rule allI, *rule impI*,
simp add:*cf-h-def*)
done

lemma (**in** *PolynRg*) *pHom-rHom*:⟦*PolynRg A B Y*; $f \in pHom\ R\ S\ X,\ A\ B\ Y\rrbracket$
$\Longrightarrow$
$f \in rHom\ R\ A$

by (*simp add*:*polyn-Hom-def*)

lemma (**in** *PolynRg*) *pHom-X-Y*:⟦*PolynRg A B Y*; *f* ∈ *pHom R S X*, *A B Y*⟧
⟹
$$f\ X = Y$$
by (*simp add*:*polyn-Hom-def*)

lemma (**in** *PolynRg*) *pHom-memTr*:⟦*PolynRg A B Y*;
f ∈ *pHom R S X*, *A B Y*⟧ ⟹
∀ *c*. *pol-coeff S* (*n*, *c*) ⟶
f (*polyn-expr R X n* (*n*, *c*)) = *polyn-expr A Y n* (*n*, *cmp f c*)
apply (*cut-tac subring*, *frule subring-Ring*,
frule PolynRg.is-Ring[*of A B Y*],
frule PolynRg.subring[*of A B Y*],
frule Ring.subring-Ring[*of A B*], *assumption*)
apply (*induct-tac n*)
apply (*rule allI*, *rule impI*)
apply (*simp add*:*polyn-expr-def cmp-def*)
apply (*frule-tac c* = (*0*, *c*) **and** *j* = *0* **in** *pol-coeff-mem*,
simp, *simp*,
frule-tac x = *c 0* **in** *mem-subring-mem-ring*, *assumption*+,
simp add:*polyn-Hom-def*, (*erule conjE*)+,
frule rHom-func[*of f R A*],
frule-tac x = *c 0* **in** *funcset-mem*[*of f carrier R carrier A*],
assumption+,
simp add:*ring-r-one*, *simp add*:*Ring.ring-r-one*)

apply (*rule allI*, *rule impI*)
apply (*cut-tac n* = *n* **and** *f* = *c* **in** *pol-coeff-pre*, *assumption*)
apply (
drule-tac a = *c* **in** *forall-spec*, *assumption*)
apply (*cut-tac n* = *n* **and** *c* = (*Suc n*, *c*) **in** *polyn-Suc*, *simp*,
simp del:*npow-suc*,
thin-tac polyn-expr R X (*Suc n*) (*Suc n*, *c*) =
polyn-expr R X n (*Suc n*, *c*) ± *c* (*Suc n*) ·$_r$ *X*^*R* (*Suc n*))
apply (*frule-tac c* = (*Suc n*, *c*) **and** *k* = *n* **in** *polyn-expr-short*, *simp*)
apply (*simp del*:*npow-suc*,
thin-tac polyn-expr R X n (*Suc n*, *c*) = *polyn-expr R X n* (*n*, *c*))
apply (*frule-tac c* = (*Suc n*, *c*) **and** *f* = *c* **in** *polyn-Hom-coeff-to-coeff*[*of A B Y f*],
assumption+, *simp del*:*npow-suc add*:*cf-h-def*)
apply (*frule-tac c* = (*Suc n*, *cmp f c*) **and** *n* = *n* **in**
PolynRg.polyn-Suc[*of A B Y*], *simp*, *simp del*:*npow-suc*,
thin-tac polyn-expr A Y (*Suc n*) (*Suc n*, *cmp f c*) =
polyn-expr A Y n (*Suc n*, *cmp f c*) ±$_A$ *cmp f c* (*Suc n*) ·$_{r\,A}$ *Y*^*A* (*Suc n*))
apply (*frule-tac k* = *n* **and** *c* = (*n*, *c*) **in** *polyn-mem*, *simp*)
apply (*frule-tac c* = (*Suc n*, *c*) **in** *monomial-mem*,
drule-tac a = *Suc n* **in** *forall-spec*, *simp*, *simp del*:*npow-suc*)

apply (*frule pHom-rHom*[*of A B Y f*], *assumption*+,

```
                                simp del:npow-suc add:rHom-add)
apply (frule-tac c = (Suc n, c) and j = Suc n in pol-coeff-mem-R, simp,
        simp del:npow-suc)
apply (cut-tac X-mem-R,
       frule-tac n = Suc n in npClose[of X],
       cut-tac Ring,
       subst rHom-tOp[of R A - - f], assumption+)
 apply (frule-tac c = (Suc n, cmp f c) and k = n in
        PolynRg.polyn-expr-short[of A B Y], assumption+, simp,
        simp del:npow-suc)
 apply (cut-tac c = (Suc n, cmp f c) and n = n in
        PolynRg.pol-coeff-le[of A B Y], assumption+, simp,
        simp del:npow-suc add:PolynRg.pol-coeff-le[of A B Y])
apply (subst rHom-npow[of R A X f], assumption+,
       simp del:npow-suc add:pHom-X-Y cmp-def)
done

lemma (in PolynRg) pHom-mem:⟦PolynRg A B Y;
      f ∈ pHom R S X, A B Y; pol-coeff S (n, c)⟧ ⟹
      f (polyn-expr R X n (n, c)) = polyn-expr A Y n (n, cmp f c)
apply (simp add:pHom-memTr)
done

lemma (in PolynRg) pHom-memc:⟦PolynRg A B Y; f ∈ pHom R S X, A B Y;
      pol-coeff S c⟧ ⟹
      f (polyn-expr R X (fst c) c) = polyn-expr A Y (fst c) (cf-h f c)
by (cases c) (simp add: cf-h-def pHom-mem)

lemma (in PolynRg) pHom-mem1:⟦PolynRg A B Y; f ∈ pHom R S X, A B Y;
       p ∈ carrier R⟧ ⟹ f p ∈ carrier A
apply (simp add:polyn-Hom-def, (erule conjE)+)
apply (simp add:rHom-mem)
done

lemma (in PolynRg) pHom-pol-mem:⟦PolynRg A B Y; f ∈ pHom R S X, A B
Y;
      p ∈ carrier R; p ≠ 0⟧ ⟹
      f p = polyn-expr A Y (deg-n R S X p) (cf-h f (s-cf R S X p))
apply (frule s-cf-expr[of p], assumption, (erule conjE)+)
apply (subst s-cf-deg[of p], assumption+)
apply (subst pHom-memc[THEN sym, of A B Y f], assumption+)
apply (drule sym, simp)
done

lemma (in PolynRg) erh-rHom-coeff:⟦PolynRg A B Y; h ∈ rHom S B;
       pol-coeff S c⟧ ⟹ erh R S X A B Y h 0 c = (cmp h (snd c)) 0
apply (cut-tac subring,
       frule subring-Ring,
       frule PolynRg.is-Ring[of A B Y],
```

frule PolynRg.subring[*of A B Y*],
frule Ring.subring-Ring[*of A B*], *assumption*)
apply (*simp add*:*erh-def polyn-expr-def cf-h-def*)
apply (*frule pol-coeff-mem* [*of c 0*], *simp*)
apply (*simp add*:*cmp-def*, *frule rHom-mem*[*of h S B snd c 0*], *assumption*)
apply (*frule Ring.mem-subring-mem-ring*[*of A B h* (*snd c 0*)], *assumption*+,
simp add:*Ring.ring-r-one*)
done

lemma (**in** *PolynRg*) *erh-polyn-exprs*:⟦*PolynRg A B Y*; *h* ∈ *rHom S B*;
pol-coeff S c; *pol-coeff S d*;
polyn-expr R X (*fst c*) *c* = *polyn-expr R X* (*fst d*) *d*⟧ ⟹
erh R S X A B Y h (*fst c*) *c* = *erh R S X A B Y h* (*fst d*) *d*
apply (*cut-tac subring*,
frule subring-Ring,
frule PolynRg.is-Ring[*of A B Y*],
frule PolynRg.subring[*of A B Y*],
frule Ring.subring-Ring[*of A B*], *assumption*+)
apply (*simp add*:*erh-def*)
apply (*cut-tac less-linear*[*of fst c fst d*])
apply (*erule disjE*,
frule pol-expr-unique3[*of c d*], *assumption*+, *simp*,
thin-tac polyn-expr R X (*fst c*) *c* = *polyn-expr R X* (*fst d*) *d*,
frule cf-h-coeff[*of A B Y h c*], *assumption*+,
frule cf-h-coeff[*of A B Y h d*], *assumption*+)
apply (*frule PolynRg.pol-expr-unique3*[*of A B Y cf-h h c cf-h h d*],
assumption+, *simp add*:*cf-h-len*, *simp add*:*cf-h-len*,
thin-tac (*polyn-expr A Y* (*fst c*) (*cf-h h c*) =
polyn-expr A Y (*fst d*) (*cf-h h d*)) =
((∀ *j*≤*fst c*. *snd* (*cf-h h c*) *j* = *snd* (*cf-h h d*) *j*) ∧
(∀ *j*∈*nset* (*Suc* (*fst c*)) (*fst d*). *snd* (*cf-h h d*) *j* = **0**$_B$)),
simp add:*cf-h-def cmp-def*, *simp add*:*rHom-0-0*)

apply (*erule disjE*,
frule pol-expr-unique2[*of c d*], *assumption*+, *simp*,
thin-tac polyn-expr R X (*fst d*) *c* = *polyn-expr R X* (*fst d*) *d*,
frule cf-h-coeff[*of A B Y h c*], *assumption*+,
frule cf-h-coeff[*of A B Y h d*], *assumption*+)
apply (*frule PolynRg.pol-expr-unique2*[*of A B Y cf-h h c cf-h h d*],
assumption+, *simp add*:*cf-h-len*, *simp add*:*cf-h-len*,
thin-tac (*polyn-expr A Y* (*fst d*) (*cf-h h c*) =
polyn-expr A Y (*fst d*) (*cf-h h d*)) =
(∀ *j*≤*fst d*. *snd* (*cf-h h c*) *j* = *snd* (*cf-h h d*) *j*),
simp add:*cf-h-def cmp-def*)

apply (*drule sym*, *rule sym*,
frule pol-expr-unique3[*of d c*], *assumption*+, *simp*,
thin-tac polyn-expr R X (*fst d*) *d* = *polyn-expr R X* (*fst c*) *c*,
frule cf-h-coeff[*of A B Y h c*], *assumption*+,

frule cf-h-coeff[*of A B Y h d*], *assumption*+)
apply (*frule PolynRg.pol-expr-unique3*[*of A B Y cf-h h d cf-h h c*],
assumption+, *simp add*:*cf-h-len*, *simp add*:*cf-h-len*,
thin-tac (*polyn-expr A Y* (*fst d*) (*cf-h h d*) =
polyn-expr A Y (*fst c*) (*cf-h h c*)) =
((∀ *j*≤*fst d*. *snd* (*cf-h h d*) *j* = *snd* (*cf-h h c*) *j*) ∧
(∀ *j*∈*nset* (*Suc* (*fst d*)) (*fst c*). *snd* (*cf-h h c*) *j* = $\mathbf{0}_B$)),
simp add:*cf-h-def cmp-def*, *simp add*:*rHom-0-0*)
done

definition
erH :: [('*a*, '*m*) *Ring-scheme*, ('*a*, '*m1*) *Ring-scheme*, '*a*,
('*b*, '*n*) *Ring-scheme*, ('*b*, '*n1*) *Ring-scheme*, '*b*, '*a* ⇒ '*b*] ⇒
'*a* ⇒ '*b* **where**
erH R S X A B Y h = (λ*x*∈*carrier R*. *erh R S X A B Y h*
(*fst* (*s-cf R S X x*)) (*s-cf R S X x*))

lemma (**in** *PolynRg*) *erH-rHom-0*:⟦*PolynRg A B Y*; *h* ∈ *rHom S B*⟧ ⟹
(*erH R S X A B Y h*) $\mathbf{0} = \mathbf{0}_A$
apply (*cut-tac subring*, *frule subring-Ring*,
cut-tac PolynRg.is-Ring[*of A B Y*],
cut-tac PolynRg.subring[*of A B Y*],
cut-tac Ring.subring-Ring[*of A B*])
apply (*simp add*:*erH-def erh-def s-cf-def polyn-expr-def*)
apply (*cut-tac ring-zero*,
simp add:*cf-h-def cmp-def rHom-0-0*,
simp add:*Ring.Subring-zero-ring-zero*,
frule Ring.ring-zero[*of A*], *simp add*:*Ring.ring-r-one*, *assumption*+)
done

lemma (**in** *PolynRg*) *erH-mem*:⟦*PolynRg A B Y*; *h* ∈ *rHom S B*; *p* ∈ *carrier R*⟧
⟹
erH R S X A B Y h p ∈ *carrier A*
apply (*cut-tac subring*, *frule subring-Ring*,
cut-tac PolynRg.is-Ring[*of A B Y*],
cut-tac PolynRg.subring[*of A B Y*],
cut-tac Ring.subring-Ring[*of A B*])
apply (*case-tac p* = $\mathbf{0}_R$)
apply (*simp add*:*erH-rHom-0*, *simp add*:*Ring.ring-zero*)

apply (*simp add*:*erH-def erh-def*)
apply (*frule s-cf-expr*[*of p*], *assumption*, (*erule conjE*)+)
apply (*rule PolynRg.polyn-mem*[*of A B Y cf-h h* (*s-cf R S X p*)], *assumption*+)
apply (*simp add*:*cf-h-coeff*)
apply (*simp add*:*cf-h-len*, *assumption*+)
done

lemma (**in** *PolynRg*) *erH-rHom-nonzero*:⟦*PolynRg A B Y*; $f \in$ *rHom S B*;
$p \in$ *carrier R*; (*erH R S X A B Y f*) $p \neq \mathbf{0}_A$⟧ $\Longrightarrow p \neq \mathbf{0}$
apply (*rule contrapos-pp*, *simp*+)
apply (*simp add*:*erH-rHom-0*)
done

lemma (**in** *PolynRg*) *erH-rHomTr2*:⟦*PolynRg A B Y*; $h \in$ *rHom S B*⟧ $\Longrightarrow$
(*erH R S X A B Y h*) (1_r) = ($1_{r\,A}$)
apply (*cut-tac subring*, *frule subring-Ring*,
frule PolynRg.is-Ring[*of A B Y*],
frule PolynRg.subring[*of A B Y*],
frule Ring.subring-Ring[*of A B*], *assumption*,
cut-tac polyn-ring-nonzero)
apply (*cut-tac Subring-one-ring-one*[*THEN sym*, *of S*],
frule Ring.ring-one[*of S*],
cut-tac ring-one)
apply (*frule s-cf-expr*[*of* 1_r], *assumption*+, (*erule conjE*)+)
apply (*frule s-cf-deg*[*THEN sym*, *of* 1_r], *assumption*, *simp*)
apply (*cut-tac pol-of-deg0*[*THEN sym*, *of* 1_r], *simp*,
simp add:*erH-def erh-def cf-h-def polyn-expr-def*,
frule pol-coeff-mem[*of s-cf R S X* $1_{r\,S}$ *0*], *simp*)
apply (*simp add*:*Subring-tOp-ring-tOp*[*THEN sym*],
simp add:*Ring.ring-r-one cmp-def*)
apply (*simp add*:*rHom-one*,
simp add:*Ring.Subring-one-ring-one*[*of A B*],
frule Ring.ring-one[*of A*], *simp add*:*Ring.ring-r-one*)
apply (*simp add*:*ring-one*)
apply *simp* **apply** *assumption*
done

declare *max.absorb1* [*simp*] *max.absorb2* [*simp*]

lemma (**in** *PolynRg*) *erH-multTr*:⟦*PolynRg A B Y*; $h \in$ *rHom S B*;
pol-coeff S c⟧ $\Longrightarrow$
$\forall f\ g.$ *pol-coeff S* (*m*, *f*) $\wedge$ *pol-coeff S* (((*fst c*) + *m*), *g*) $\wedge$
(*polyn-expr R X* (*fst c*) *c*) $\cdot_r$ (*polyn-expr R X m* (*m*, *f*)) =
(*polyn-expr R X* ((*fst c*) + *m*) ((*fst c*) + *m*, *g*)) $\longrightarrow$
(*polyn-expr A Y* (*fst c*) (*cf-h h c*)) $\cdot_{r\,A}$ (*polyn-expr A Y m* (*cf-h h* (*m*, *f*))) =
(*polyn-expr A Y* ((*fst c*) + *m*) (*cf-h h* ((*fst c*)+*m*, *g*)))
apply (*cut-tac subring*, *frule subring-Ring*,
frule PolynRg.is-Ring[*of A B Y*],
frule PolynRg.subring[*of A B Y*],
frule Ring.subring-Ring[*of A B*], *assumption*)
apply (*cases c*)
apply (*simp only*:)
apply (*rename-tac l u*)
apply (*thin-tac c* = (*l*, *u*))
apply (*induct-tac m*)
apply ((*rule allI*)+, *rule impI*, (*erule conjE*)+, *simp*)

apply (*simp add*:*cf-h-polyn*[*of A B Y h*])
apply (*simp add*:*polyn-expr-def*[*of - - 0*])
apply (*frule-tac c* = (*0*, *f*) **and** *j* = *0* **in** *pol-coeff-mem*, *simp*, *simp*,
frule-tac c = (*0*, *f*) **and** *j* = *0* **in** *pol-coeff-mem-R*, *simp*, *simp*,
frule-tac c = (*l*, *u*) **and** *k* = *l* **in** *polyn-mem*, *simp*,
simp add:*ring-r-one*,
simp add:*ring-tOp-commute*,
simp add:*scalar-times-pol-expr*)
apply (*frule-tac c* = (*0*, *f*) **in** *cf-h-coeff*[*of A B Y h*], *assumption*+,
frule-tac c = (*l*, *u*) **in** *cf-h-coeff*[*of A B Y h*], *assumption*+)
apply (*frule-tac c* = *cf-h h* (*0*, *f*) **in** *PolynRg.pol-coeff-mem*[*of A B Y - 0*],
assumption+, *simp*, *simp add*:*cf-h-cmp*,
frule-tac c = *cf-h h* (*0*, *f*) **in** *PolynRg.pol-coeff-mem-R*[*of A B Y - 0*],
assumption+, *simp*, *simp add*:*cf-h-cmp*,
frule-tac c = *cf-h h* (*l*, *u*) **and** *k* = *l* **in** *PolynRg.polyn-mem*, *simp*,
simp add:*cf-h-len*, *simp add*:*cf-h-polyn*,
simp add:*Ring.ring-r-one*, *simp add*:*Ring.ring-tOp-commute*[*of A*],
frule-tac n = *l* **and** *f* = *u* **in** *cf-h-pol-coeff*[*of A B Y h*],
assumption+, *simp*)
apply (*simp add*:*PolynRg.scalar-times-pol-expr*,
frule-tac c = (*l*, *u*) **and** *a* = *f 0* **in** *sp-cf-pol-coeff*, *assumption*+,
frule-tac c = (*l*, *cmp h u*) **and** *a* = (*cmp h f*) *0* **in**
PolynRg.sp-cf-pol-coeff, *assumption*+,
frule-tac c = (*l*, *g*) **in** *cf-h-coeff*[*of A B Y h*], *assumption*+,
simp add:*cf-h-pol-coeff*)
apply (*rule-tac c* = *sp-cf B* (*cmp h f 0*) (*l*, *cmp h u*) **and**
d = (*l*, *cmp h g*) **and** *k* = *l* **in** *PolynRg.polyn-exprs-eq*[*of A B Y*],
assumption+, *simp add*:*sp-cf-len*,
simp add:*PolynRg.sp-cf-len*)
apply (*rule allI*, *rule impI*)
apply (*frule-tac c* = *sp-cf S* (*f 0*) (*l*, *u*) **and** *d* = (*l*, *g*) **in**
pol-expr-unique2, *assumption*+,
simp add:*sp-cf-len*, *simp add*:*sp-cf-len*)
apply (*drule-tac a* = *j* **in** *forall-spec*, *simp*)
apply (*simp add*:*sp-cf-def*)
apply (*rotate-tac* −*1*, *drule sym*, *simp add*:*cmp-def*,
thin-tac pol-coeff B (*l*, λ*x*. *h* (*g x*)),
thin-tac pol-coeff B (*l*, λ*j*. *h* (*f 0*) $\cdot_{r\,B}$ *h* (*u j*)),
thin-tac pol-coeff S (*l*, λ*j*. *f 0* $\cdot_{r\,S}$ *u j*),
thin-tac polyn-expr A Y l (*l*, λ*x*. *h* (*u x*)) ∈ *carrier A*,
thin-tac pol-coeff B (*l*, λ*x*. *h* (*u x*)),
thin-tac polyn-expr R X l (*l*, λ*j*. *f 0* $\cdot_{r\,S}$ *u j*) =
polyn-expr R X l (*l*, *g*))
apply (*frule-tac c* = (*l*, *u*) **and** *j* = *j* **in** *pol-coeff-mem*, *simp*, *simp*)
apply (*simp add*:*rHom-tOp*)

apply ((*rule allI*)+, (*rule impI*), (*erule conjE*)+)
apply (*simp add*:*cf-h-polyn*,
frule-tac n = *n* **and** *f* = *f* **in** *pol-coeff-pre*,

frule-tac $n = l + n$ **and** $f = g$ **in** *pol-coeff-pre*,
frule-tac $n = l$ **and** $f = u$ **and** $m = n$ **and** $g = f$ **in** *polyn-expr-tOp*,
assumption+, *erule exE*, (*erule conjE*)+,
rotate-tac -1, *drule sym*)

apply (*drule-tac* $x = f$ **in** *spec*,
drule-tac $x = e$ **in** *spec*, *simp*,
frule-tac $n = n$ **and** $f = f$ **in** *polyn-Suc-split*,
simp del:*npow-suc*,
thin-tac polyn-expr R X (*Suc n*) (*Suc n*, *f*) =
polyn-expr R X n (*n*, *f*) $\pm$ *f* (*Suc n*) $\cdot_r$ $X\hat{}R$ (*Suc n*))

apply (*frule-tac* $c = (Suc\ n, f)$ **in** *cf-h-coeff*[*of A B Y h*], *assumption*+,
simp del:*npow-suc add*:*cf-h-pol-coeff*)
apply (*frule-tac* $n = n$ **and** $f = cmp\ h\ f$ **in** *PolynRg.polyn-Suc-split*[*of A B Y*],
simp add:*cf-h-pol-coeff*, *simp del*:*npow-suc*,
thin-tac polyn-expr A Y (*Suc n*) (*Suc n*, *cmp h f*) =
polyn-expr A Y n (*n*, *cmp h f*) $\pm_A$ *cmp h f* (*Suc n*) $\cdot_{r\,A}$ $Y\hat{}A$ (*Suc n*))

apply (*frule-tac* $c = (l, u)$ **and** $k = l$ **in** *polyn-mem*, *simp*,
frule-tac $n = n$ **and** $f = f$ **in** *pol-coeff-pre*,
frule-tac $c = (n, f)$ **and** $k = n$ **in** *polyn-mem*, *simp*,
frule-tac $c = (Suc\ n, f)$ **and** $j = Suc\ n$ **in** *pol-coeff-mem-R*, *simp*,
cut-tac X-mem-R, *simp del*:*npow-suc*,
cut-tac $n = Suc\ n$ **in** *npClose*[*of X*], *assumption*,
frule-tac $x = f\ (Suc\ n)$ **and** $y =\ X\hat{}R\ (Suc\ n)$ **in** *ring-tOp-closed*,
assumption+,
simp del:*npow-suc add*:*ring-distrib1*)
apply (*frule-tac* $c = (l, u)$ **in** *cf-h-coeff*[*of A B Y h*], *assumption*+,
frule-tac $c = (Suc\ n, f)$ **in** *cf-h-coeff*[*of A B Y h*], *assumption*+,
frule-tac $n = n$ **and** $f = f$ **in** *pol-coeff-pre*,
frule-tac $c = (n, f)$ **in** *cf-h-coeff*[*of A B Y h*], *assumption*+,
simp del:*npow-suc add*:*cf-h-pol-coeff*)
apply (*frule-tac* $c = (l, cmp\ h\ u)$ **and** $k = l$ **in** *PolynRg.polyn-mem*, *simp*, *simp*,
frule-tac $c = (n, cmp\ h\ f)$ **and** $k = n$ **in** *PolynRg.polyn-mem*, *simp*, *simp*)

apply (*frule-tac* $c = (Suc\ n, f)$ **and** $j = Suc\ n$ **in** *pol-coeff-mem-R*, *simp*,
cut-tac X-mem-R, *simp del*:*npow-suc*,
cut-tac $n = Suc\ n$ **in** *npClose*[*of X*], *assumption*,
frule-tac $x = f\ (Suc\ n)$ **and** $y =\ X\hat{}R\ (Suc\ n)$ **in** *ring-tOp-closed*,
assumption+,
simp del:*npow-suc add*:*ring-distrib1*)

apply (*frule-tac* $c = (Suc\ n, cmp\ h\ f)$ **and** $j = Suc\ n$ **in**
PolynRg.pol-coeff-mem-R[*of A B Y*], *simp del*:*npow-suc*,
simp del:*npow-suc*,
frule-tac PolynRg.X-mem-R[*of A B Y*], *simp del*:*npow-suc*,

frule-tac $n = Suc\ n$ **in** *Ring.npClose*[*of* $A\ Y$], *assumption*,
frule-tac $x = cmp\ h\ f\ (Suc\ n)$ **and** $y = \ Y\hat{}A\ (Suc\ n)$ **in**
Ring.ring-tOp-closed[*of* A], *assumption*+,
simp del:*npow-suc add*:*Ring.ring-distrib1*)

apply (*frule-tac* $x1 = polyn\text{-}expr\ R\ X\ l\ (l,\ u)$ **and** $y1 = f\ (Suc\ n)$ **and**
$z1 = \ X\hat{}R\ (Suc\ n)$ **in** *ring-tOp-assoc*[*THEN sym*], *assumption*+,
simp del:*npow-suc*,
thin-tac $polyn\text{-}expr\ R\ X\ l\ (l,\ u) \cdot_r polyn\text{-}expr\ R\ X\ n\ (n,\ f) =$
$polyn\text{-}expr\ R\ X\ (l + n)\ (l + n,\ e)$,
thin-tac $polyn\text{-}expr\ R\ X\ l\ (l,\ u) \cdot_r (f\ (Suc\ n) \cdot_r X\hat{}R\ (Suc\ n)) =$
$polyn\text{-}expr\ R\ X\ l\ (l,\ u) \cdot_r f\ (Suc\ n) \cdot_r X\hat{}R\ (Suc\ n)$,
simp only:*ring-tOp-commute*,
frule-tac $c = (Suc\ n,\ f)$ **and** $j = Suc\ n$ **in** *pol-coeff-mem*, *simp*,
simp del:*npow-suc*,
simp del:*npow-suc add*:*scalar-times-pol-expr*)

apply (*frule-tac* $c = (l,\ u)$ **and** $a = f\ (Suc\ n)$ **in** *sp-cf-pol-coeff*,
assumption+,
frule-tac $c = sp\text{-}cf\ S\ (f\ (Suc\ n))\ (l,\ u)$ **and** $k = l$ **in** *polyn-mem*,
simp add:*sp-cf-len*, *simp only*:*ring-tOp-commute*,
frule-tac $c1 = sp\text{-}cf\ S\ (f\ (Suc\ n))\ (l,\ u)$ **and** $j1 = Suc\ n$ **in**
low-deg-terms-zero1[*THEN sym*],
simp only:*sp-cf-len*, *simp del*: *npow-suc*)
apply (*frule-tac* $c = sp\text{-}cf\ S\ (f\ (Suc\ n))\ (l,\ u)$ **and** $n = Suc\ n$ **in**
ext-cf-pol-coeff,
frule-tac $c = (l + n,\ e)$ **and** $d = ext\text{-}cf\ S\ (Suc\ n)\ (sp\text{-}cf\ S\ ($
$f\ (Suc\ n))\ (l,\ u))$ **in** *polyn-add1*, *assumption*+,
simp del:*npow-suc add*:*ext-cf-len sp-cf-len*,
cut-tac $a = l$ **and** $b = n$ **in** *add.commute*,
simp del:*npow-suc*,
thin-tac $polyn\text{-}expr\ R\ X\ (n + l)\ (n + l,\ e) \pm$
$polyn\text{-}expr\ R\ X\ (Suc\ (n + l))$
$(ext\text{-}cf\ S\ (Suc\ n)\ (sp\text{-}cf\ S\ (f\ (Suc\ n))\ (l,\ u))) =$
$polyn\text{-}expr\ R\ X\ (Suc\ (n + l))$
$(add\text{-}cf\ S\ (n + l,\ e)$
$(ext\text{-}cf\ S\ (Suc\ n)\ (sp\text{-}cf\ S\ (f\ (Suc\ n))\ (l,\ u)))))$,
thin-tac $polyn\text{-}expr\ R\ X\ l\ (l,\ u) \in carrier\ R$,
thin-tac $polyn\text{-}expr\ R\ X\ n\ (n,\ f) \in carrier\ R$,
thin-tac $f\ (Suc\ n) \in carrier\ R$,
thin-tac $X \in carrier\ R$,
thin-tac $X\hat{}R\ (Suc\ n) \in carrier\ R$,
thin-tac $f\ (Suc\ n) \cdot_r X\hat{}R\ (Suc\ n) \in carrier\ R$,
thin-tac $polyn\text{-}expr\ R\ X\ l\ (sp\text{-}cf\ S\ (f\ (Suc\ n))\ (l,\ u)) \in carrier\ R$,
thin-tac $X\hat{}R\ (Suc\ n) \cdot_r polyn\text{-}expr\ R\ X\ l\ (sp\text{-}cf\ S\ (f\ (Suc\ n))\ (l,\ u)) =$
$polyn\text{-}expr\ R\ X\ (Suc\ (n + l))$
$(ext\text{-}cf\ S\ (Suc\ n)\ (sp\text{-}cf\ S\ (f\ (Suc\ n))\ (l,\ u))))$

apply (*subst Ring.ring-tOp-assoc*[*THEN sym*], *assumption*+,
subst Ring.ring-tOp-commute, *assumption*+)
apply (*frule-tac c* = (*Suc n*, *f*) **in** *cf-h-coeff*[*of A B Y h*],
assumption+,
simp only:*cf-h-pol-coeff*)
apply (*frule-tac c* = (*Suc n*, *cmp h f*) **and** *j* = *Suc n* **in**
PolynRg.pol-coeff-mem[*of A B Y*], *assumption*+, *simp*,
simp del:*npow-suc*)
apply (*subst PolynRg.scalar-times-pol-expr*[*of A B Y*], *assumption*+,
simp)
apply (*frule-tac a* = *cmp h f* (*Suc n*) **and** *c* = (*l*, *cmp h u*) **in**
PolynRg.sp-cf-pol-coeff[*of A B Y*], *assumption*+)
apply (*frule-tac c* = *sp-cf B* (*cmp h f* (*Suc n*)) (*l*, *cmp h u*) **and** *k* = *l*
in *PolynRg.polyn-mem*[*of A B Y*], *assumption*, *simp*)
apply (*simp add*:*PolynRg.sp-cf-len*)
apply (*subst Ring.ring-tOp-commute*[*of A*], *assumption*+)
apply (*frule-tac c1* = *sp-cf B* (*cmp h f* (*Suc n*)) (*l*, *cmp h u*) **and**
j1 = *Suc n* **in** *PolynRg.low-deg-terms-zero1*[*of A B Y*, *THEN sym*],
assumption+)
apply (*simp del*:*npow-suc add*:*sp-cf-len PolynRg.sp-cf-len*,
thin-tac $Y^{\wedge}A$ (*Suc n*) $\cdot_{rA}$
polyn-expr A Y l (*sp-cf B* (*cmp h f* (*Suc n*)) (*l*, *cmp h u*)) =
polyn-expr A Y (*Suc* (*l* + *n*))
(*ext-cf B* (*Suc n*) (*sp-cf B* (*cmp h f* (*Suc n*)) (*l*, *cmp h u*))),
thin-tac polyn-expr A Y l (*sp-cf B* (*cmp h f* (*Suc n*)) (*l*, *cmp h u*))
∈ *carrier A*)
apply (*frule-tac c* = *sp-cf B* (*cmp h f* (*Suc n*)) (*l*, *cmp h u*) **and**
n = *Suc n* **in** *PolynRg.ext-cf-pol-coeff*[*of A B Y*], *assumption*+)
apply (*frule-tac c* = (*l* + *n*, *cmp h e*) **and**
d = *ext-cf B* (*Suc n*) (*sp-cf B* (*cmp h f* (*Suc n*)) (*l*, *cmp h u*)) **in**
PolynRg.polyn-add1[*of A B Y*])
apply (*frule-tac c* = (*n* + *l*, *e*) **in** *cf-h-coeff*[*of A B Y h*], *assumption*+)
apply (*simp add*:*cf-h-pol-coeff*[*of A B Y h*] *add.commute*, *assumption*)
apply (*simp add*:*PolynRg.ext-cf-len PolynRg.sp-cf-len*)
apply (*cut-tac a* = *n* **and** *b* = *l* **in** *add.commute*, *simp*)

apply (*frule-tac c* = (*l* + *n*, *e*) **and**
d = *ext-cf S* (*Suc n*) (*sp-cf S* (*f* (*Suc n*)) (*l*, *u*)) **in**
add-cf-pol-coeff, *assumption*+)
apply (*frule-tac c* = (*l* + *n*, *cmp h e*) **and**
d = *ext-cf B* (*Suc n*) (*sp-cf B* (*cmp h f* (*Suc n*)) (*l*, *cmp h u*)) **in**
PolynRg.add-cf-pol-coeff[*of A B Y*])
apply (*frule-tac c* = (*l* + *n*, *e*) **in** *cf-h-coeff*[*of A B Y h*], *assumption*+)
apply (*simp add*:*cf-h-pol-coeff*) **apply** *simp*
apply (*thin-tac polyn-expr A Y l* (*l*, *cmp h u*) $\cdot_{rA}$ *polyn-expr A Y n*
(*n*, *cmp h f*) = *polyn-expr A Y* (*l* + *n*) (*l* + *n*, *cmp h e*),
thin-tac polyn-expr A Y l (*l*, *cmp h u*) ∈ *carrier A*,

thin-tac polyn-expr A Y n (n, cmp h f) ∈ carrier A,
thin-tac Y ∈ carrier A,
thin-tac Y ^A n ·rA Y ∈ carrier A,
thin-tac cmp h f (Suc n) ·rA (Y ^A n ·rA Y) ∈ carrier A,
thin-tac polyn-expr A Y (l + n) (l + n, cmp h e) ±A
polyn-expr A Y (Suc (l + n))
(ext-cf B (Suc n) (sp-cf B (cmp h f (Suc n)) (l, cmp h u))) =
polyn-expr A Y (Suc (l + n))
(add-cf B (l + n, cmp h e)
(ext-cf B (Suc n) (sp-cf B (cmp h f (Suc n)) (l, cmp h u)))))
apply *(frule-tac c = (Suc (l + n), g)* **in** *cf-h-coeff[of A B Y h],*
assumption+)
apply *(simp add:cf-h-pol-coeff)*
apply *(frule-tac c = (Suc (l + n), g)* **and** *d = add-cf S (l + n, e)*
(ext-cf S (Suc n) (sp-cf S (f (Suc n)) (l, u))) **in** *pol-expr-unique2)*
apply *assumption* **apply** *(simp add:add-cf-len)*
apply *(simp add:ext-cf-len sp-cf-len)*
apply *(simp add:add-cf-len ext-cf-len sp-cf-len)*
apply *(cut-tac a = n* **and** *b = l* **in** *add.commute, simp,*
thin-tac pol-coeff B
(add-cf B (l + n, cmp h e)
(ext-cf B (Suc n) (sp-cf B (cmp h f (Suc n)) (l, cmp h u)))),
thin-tac pol-coeff B (Suc (l + n), cmp h g),
thin-tac pol-coeff S
(add-cf S (l + n, e)
(ext-cf S (Suc n) (sp-cf S (f (Suc n)) (l, u)))),
thin-tac pol-coeff B
(ext-cf B (Suc n) (sp-cf B (cmp h f (Suc n)) (l, cmp h u))),
thin-tac pol-coeff B (sp-cf B (cmp h f (Suc n)) (l, cmp h u)),
thin-tac polyn-expr R X (Suc (l + n)) (Suc (l + n), g) =
polyn-expr R X (Suc (l + n))
(add-cf S (l + n, e)
(ext-cf S (Suc n) (sp-cf S (f (Suc n)) (l, u)))))
apply *(rule sym)*
apply *(frule-tac c = (Suc (l + n), g)* **in** *cf-h-coeff[of A B Y h],*
assumption+,
frule-tac c = (l + n, e) **in** *cf-h-coeff[of A B Y h],*
assumption+)
apply *(frule-tac c = (l, cmp h u)* **and** *a = cmp h f (Suc n)* **in**
PolynRg.sp-cf-pol-coeff[of A B Y], assumption+)
apply *(cut-tac c = (l + n, cmp h e)* **and**
d = ext-cf B (Suc n) (sp-cf B (cmp h f (Suc n)) (l, cmp h u)) **in**
PolynRg.add-cf-pol-coeff, assumption+)
apply *(simp add:cf-h-pol-coeff)*
apply *(rule PolynRg.ext-cf-pol-coeff, assumption+)*
apply *(frule-tac c = (Suc (l + n), cmp h g)* **and**
d = add-cf B (l + n, cmp h e)
(ext-cf B (Suc n) (sp-cf B (cmp h f (Suc n)) (l, cmp h u))) **in**
PolynRg.pol-expr-unique2[of A B Y])

```
  apply (simp add:cf-h-pol-coeff) apply assumption
  apply (simp add:add-cf-len)
  apply (frule-tac n = l + n and f = e in  cf-h-pol-coeff[of A B Y h],
         assumption+)
  apply (frule-tac c = sp-cf B (cmp h f (Suc n)) (l, cmp h u) and
         n = Suc n in
         PolynRg.ext-cf-pol-coeff[of A B Y], assumption+)
  apply (simp add:PolynRg.add-cf-len)
        apply (simp add:PolynRg.ext-cf-len)
        apply (simp add:PolynRg.sp-cf-len)
  apply (simp add:PolynRg.add-cf-len)
  apply (frule-tac c = sp-cf B (cmp h f (Suc n)) (l, cmp h u) and
         n = Suc n in PolynRg.ext-cf-pol-coeff[of A B Y],
          assumption+)
  apply (frule-tac c = (l + n, cmp h e) and
         d = ext-cf B (Suc n)
               (sp-cf B (cmp h f (Suc n)) (l, cmp h u)) in
         PolynRg.add-cf-pol-coeff[of A B Y])
  apply (simp add:cf-h-pol-coeff, assumption)
apply (simp add:cf-h-pol-coeff)
  apply (simp add:PolynRg.add-cf-len)
  apply (simp add:PolynRg.ext-cf-len)
  apply (simp add:PolynRg.sp-cf-len)
  apply (cut-tac a = n and b = l in add.commute, simp)


  apply (thin-tac (polyn-expr A Y (Suc (l + n)) (Suc (l + n), cmp h g) =
       polyn-expr A Y (Suc (l + n))
        (add-cf B (l + n, cmp h e)
          (ext-cf B (Suc n) (sp-cf B (cmp h f (Suc n)) (l, cmp h u))))) =
      (∀j≤Suc (l + n).
         cmp h g j =
         snd (add-cf B (l + n, cmp h e)
               (ext-cf B (Suc n)
                 (sp-cf B (cmp h f (Suc n)) (l, cmp h u))))
          j))

 apply (rule allI, rule impI)
 apply (subst cmp-def)+
 apply (drule-tac a = j in forall-spec, simp, simp,
        thin-tac g j = snd (add-cf S (l + n, e)
                 (ext-cf S (Suc n) (sp-cf S (f (Suc n)) (l, u)))) j)
 apply (case-tac j = Suc (l+n), simp)
   apply ((subst add-cf-def)+,
          simp add:ext-cf-len, simp add:sp-cf-len,
          simp add:cmp-def PolynRg.ext-cf-len,
          simp add:PolynRg.sp-cf-len,
          (subst ext-cf-def)+, simp add:sp-cf-len sliden-def,
          (subst sp-cf-def)+, simp,
```

frule-tac $c = (l, u)$ **and** $j = l$ **in** *pol-coeff-mem*, *simp*, *simp*,
simp add:*rHom-tOp*)

apply (*frule-tac* $m = j$ **and** $n = Suc\ (l + n)$ **in** *noteq-le-less*, *assumption*,
thin-tac $j \leq Suc\ (l + n)$, *thin-tac* $j \neq Suc\ (l + n)$,
(*subst add-cf-def*)+,
simp add:*ext-cf-len sp-cf-len*, *simp add*:*cmp-def*,
simp add:*PolynRg.ext-cf-len PolynRg.sp-cf-len*,
(*subst ext-cf-def*)+, *simp add*:*sp-cf-len*,
(*subst sp-cf-def*)+, *simp add*:*sliden-def*,
frule-tac $x = j$ **and** $n = l + n$ **in** *Suc-less-le*)

apply (*rule conjI*)
apply (*rule impI*,
frule-tac $x = j$ **and** $y = Suc\ (l + n)$ **in** *less-imp-le*,
frule-tac $m = j$ **and** $n = Suc\ (l + n)$ **and** $l = Suc\ n$ **in** *diff-le-mono*,
simp,
frule-tac $c = (l, u)$ **and** $j = j - Suc\ n$ **in** *pol-coeff-mem*, *simp*,
frule-tac $c = (l + n, e)$ **and** $j = j$ **in** *pol-coeff-mem*, *simp*, *simp*,
frule-tac $x = f\ (Suc\ n)$ **and** $y = u\ (j - Suc\ n)$ **in**
Ring.ring-tOp-closed[*of S*], *assumption*+,
simp add:*rHom-add rHom-tOp*)
apply (*rule impI*)
apply (*frule-tac* $c = (l + n, e)$ **and** $j = j$ **in** *pol-coeff-mem*, *simp*, *simp*,
frule-tac Ring.ring-zero[*of S*],
simp add:*rHom-add rHom-0-0*)
done

lemma (**in** *PolynRg*) *erH-multTr1*:⟦*PolynRg A B Y*; $h \in rHom\ S\ B$;
pol-coeff S c; *pol-coeff S d*; *pol-coeff S e*; $fst\ e = fst\ c + fst\ d$;
$(polyn\text{-}expr\ R\ X\ (fst\ c)\ c) \cdot_r (polyn\text{-}expr\ R\ X\ (fst\ d)\ d) =$
$polyn\text{-}expr\ R\ X\ ((fst\ c) + (fst\ d))\ e$ ⟧ ⟹
$(polyn\text{-}expr\ A\ Y\ (fst\ c)\ (cf\text{-}h\ h\ c)) \cdot_{rA} (polyn\text{-}expr\ A\ Y\ (fst\ d)\ (cf\text{-}h\ h\ d))$
$= (polyn\text{-}expr\ A\ Y\ (fst\ e)\ (cf\text{-}h\ h\ e))$
by (*cases d*, *cases e*) (*simp add*: *erH-multTr*)

lemma (**in** *PolynRg*) *erHomTr0*:⟦*PolynRg A B Y*; $h \in rHom\ S\ B$; $x \in carrier\ R$⟧
⟹ $erH\ R\ S\ X\ A\ B\ Y\ h\ (-_a\ x) = -_{aA}\ (erH\ R\ S\ X\ A\ B\ Y\ h\ x)$
apply (*cut-tac ring-is-ag*,
cut-tac subring, *frule subring-Ring*,
frule PolynRg.is-Ring[*of A B Y*],
frule Ring.ring-is-ag[*of A*],
frule PolynRg.subring[*of A B Y*],
frule Ring.subring-Ring[*of A B*], *assumption*+)
apply (*case-tac* $x = \mathbf{0}_R$, *simp add*:*aGroup.ag-inv-zero*,
simp add:*erH-rHom-0*[*of A B Y h*],
frule Ring.ring-is-ag[*of A*], *simp add*:*aGroup.ag-inv-zero*)
apply (*simp add*:*erH-def erh-def*)

apply (*simp add*:*aGroup.ag-mOp-closed*)
apply (*frule-tac* $p = x$ **in** *s-cf-expr*, *assumption*+, (*erule conjE*)+)
apply (*frule-tac* $x = x$ **in** *aGroup.ag-mOp-closed*, *assumption*+,
frule-tac $p = -_a\ x$ **in** *s-cf-expr*,
thin-tac $x =$ *polyn-expr R X* (*fst* (*s-cf R S X x*)) (*s-cf R S X x*),
rule contrapos-pp, *simp*+,
frule-tac $x = x$ **in** *aGroup.ag-inv-inv*, *simp*, *simp add*:*aGroup.ag-inv-zero*,
(*erule conjE*)+)

apply (*frule-tac* $c =$ *s-cf R S X* $(-_a\ x)$ **in** *cf-h-coeff*[*of A B Y h*],
assumption+,
frule-tac $c =$ *s-cf R S X x* **in** *cf-h-coeff*[*of A B Y h*],
assumption+)
apply (*frule polyn-minus-m-cf*[*of s-cf R S X x fst* (*s-cf R S X x*)],
simp)
apply (*cut-tac* $a = -_a\ x$ **and**
$b =$ *polyn-expr R X* (*fst* (*s-cf R S X* $(-_a\ x)$)) (*s-cf R S X* $(-_a\ x)$)
and $c =$ *polyn-expr R X* (*fst* (*s-cf R S X x*)) (*m-cf S* (*s-cf R S X x*))
in *box-equation*, *assumption*, *simp*,
thin-tac $x =$ *polyn-expr R X* (*fst* (*s-cf R S X x*)) (*s-cf R S X x*),
thin-tac $-_a\ x =$
polyn-expr R X (*fst* (*s-cf R S X* $(-_a\ x)$)) (*s-cf R S X* $(-_a\ x)$),
thin-tac $-_a$ (*polyn-expr R X* (*fst* (*s-cf R S X x*)) (*s-cf R S X x*)) =
polyn-expr R X (*fst* (*s-cf R S X x*)) (*m-cf S* (*s-cf R S X x*)),
frule m-cf-pol-coeff[*of s-cf R S X x*])

apply (*subst PolynRg.polyn-minus-m-cf*[*of A B Y*], *assumption*+,
simp add:*cf-h-len*)
apply (*frule-tac* $c =$ *cf-h h* (*s-cf R S X x*) **in**
PolynRg.m-cf-pol-coeff[*of A B Y*], *assumption*,
frule PolynRg.pol-expr-unique2[*of A B Y cf-h h* (*s-cf R S X* $(-_a\ x)$)
m-cf B (*cf-h h* (*s-cf R S X x*))], *assumption*+)
apply (*simp add*:*cf-h-len*)
apply (*simp add*:*PolynRg.m-cf-len cf-h-len*)
apply (*simp add*:*s-cf-deg*[*THEN sym*, *of x*],
cut-tac ring-zero,
frule aGroup.ag-inv-inj[*of R x* **0**], *assumption*+,
simp only:*aGroup.ag-inv-zero*,
subst s-cf-deg[*THEN sym*, *of* $-_a\ x$], *assumption*+,
simp add:*deg-minus-eq*)
apply (*simp add*:*cf-h-len PolynRg.m-cf-len*,
thin-tac (*polyn-expr A Y* (*fst* (*s-cf R S X* $(-_a\ x)$)))
(*cf-h h* (*s-cf R S X* $(-_a\ x)$)) = *polyn-expr A Y* (*fst* (*s-cf R S X x*))
(*m-cf B* (*cf-h h* (*s-cf R S X x*)))) =
($\forall j \leq$*fst* (*s-cf R S X* $(-_a\ x)$). *snd* (*cf-h h* (*s-cf R S X* $(-_a\ x)$)) j =
snd (*m-cf B* (*cf-h h* (*s-cf R S X x*))) j))
apply (*rule allI*, *rule impI*,
subst m-cf-def)
apply ((*subst cf-h-def*)+, *simp add*:*cmp-def*)

apply (*thin-tac snd* (*s-cf R S X* ($-_a$ *x*)) (*fst* (*s-cf R S X* ($-_a$ *x*))) $\neq$ $\mathbf{0}_S$,
thin-tac pol-coeff B (*cf-h h* (*s-cf R S X* ($-_a$ *x*))),
thin-tac pol-coeff B (*cf-h h* (*s-cf R S X x*)),
thin-tac pol-coeff B (*m-cf B* (*cf-h h* (*s-cf R S X x*))))
apply (*frule m-cf-pol-coeff*[*of s-cf R S X x*])
apply (*frule pol-expr-unique2*[*of s-cf R S X* ($-_a$ *x*)
m-cf S (*s-cf R S X x*)], *assumption*+,
simp add:*m-cf-len cf-h-len*)
apply (*simp add*:*s-cf-deg*[*THEN sym*, *of x*],
cut-tac ring-zero,
frule aGroup.ag-inv-inj[*of R x* $\mathbf{0}$], *assumption*+,
simp only:*aGroup.ag-inv-zero*,
subst s-cf-deg[*THEN sym*, *of* $-_a$ *x*], *assumption*+,
simp add:*deg-minus-eq*, *simp add*:*m-cf-len*)
apply (*drule-tac a* = *j* **in** *forall-spec*, *assumption*,
thin-tac snd (*s-cf R S X* ($-_a$ *x*)) *j* = *snd* (*m-cf S* (*s-cf R S X x*)) *j*,
thin-tac polyn-expr R X (*fst* (*s-cf R S X* ($-_a$ *x*)))
(*s-cf R S X* ($-_a$ *x*)) =
polyn-expr R X (*fst* (*s-cf R S X x*)) (*m-cf S* (*s-cf R S X x*)))
apply (*cut-tac ring-zero*,
frule aGroup.ag-inv-inj[*of R x* $\mathbf{0}$], *assumption*+,
simp only:*aGroup.ag-inv-zero*,
(*simp add*:*s-cf-deg*[*THEN sym*, *of* $-_a$ *x*] *deg-minus-eq*,
simp add:*s-cf-deg*[*of x*]))
apply (*frule-tac j* = *j* **in** *pol-coeff-mem*[*of s-cf R S X x*],
assumption+)
apply (*subst m-cf-def*, *simp*)
apply (*simp add*:*rHom-inv-inv*)
done

lemma (**in** *PolynRg*) *erHomTr1*:⟦*PolynRg A B Y*; *h* ∈ *rHom S B*;
a ∈ *carrier R*; *b* ∈ *carrier R*; *a* ≠ $\mathbf{0}$; *b* ≠ $\mathbf{0}$; *a* ± *b* ≠ $\mathbf{0}$;
deg-n R S X a = *deg-n R S X b*⟧ ⟹
erH R S X A B Y h (*a* ± *b*) = *erH R S X A B Y h a* $\pm_A$
(*erH R S X A B Y h b*)
apply (*cut-tac ring-is-ag*,
cut-tac subring, *frule subring-Ring*,
frule PolynRg.subring[*of A B Y*],
frule-tac x = *a* **and** *y* = *b* **in** *aGroup.ag-pOp-closed*[*of R*],
assumption+,
simp add:*erH-def erh-def*)
apply (*frule s-cf-expr*[*of a*], *assumption*,
frule s-cf-expr[*of b*], *assumption*,
frule s-cf-expr[*of a* ± *b*], *assumption*,
(*erule conjE*)+,
simp add:*s-cf-deg*)
apply (*frule polyn-add1*[*of s-cf R S X a s-cf R S X b*], *assumption*+)
apply (*cut-tac a* = *a* ± *b* **and**
b = *polyn-expr R X* (*fst* (*s-cf R S X* (*a* ± *b*))) (*s-cf R S X* (*a* ± *b*)) **and**

```
    c = polyn-expr R X (max (fst (s-cf R S X a)) (fst (s-cf R S X b)))
    (add-cf S (s-cf R S X a) (s-cf R S X b)) in box-equation,
    drule sym, drule sym, drule sym, simp,
    drule sym, drule sym, drule sym, simp,
    thin-tac polyn-expr R X (fst (s-cf R S X a)) (s-cf R S X a) ±
     polyn-expr R X (fst (s-cf R S X b)) (s-cf R S X b) =
     polyn-expr R X (max (fst (s-cf R S X a)) (fst (s-cf R S X b)))
     (add-cf S (s-cf R S X a) (s-cf R S X b)),
    thin-tac a = polyn-expr R X (fst (s-cf R S X b)) (s-cf R S X a),
    thin-tac b = polyn-expr R X (fst (s-cf R S X b)) (s-cf R S X b),
    thin-tac a ± b =
      polyn-expr R X (fst (s-cf R S X (a ± b))) (s-cf R S X (a ± b)))
    apply simp

apply (frule cf-h-coeff[of A B Y h s-cf R S X a], assumption+,
    frule cf-h-coeff[of A B Y h s-cf R S X b], assumption+,
    frule cf-h-coeff[of A B Y h s-cf R S X (a ± b)], assumption+)
apply (frule PolynRg.polyn-add1[of A B Y cf-h h (s-cf R S X a)
                                cf-h h (s-cf R S X b)], assumption+,
    simp add:cf-h-len,
    thin-tac polyn-expr A Y (fst (s-cf R S X b)) (cf-h h (s-cf R S X a))
     ±_A polyn-expr A Y (fst (s-cf R S X b)) (cf-h h (s-cf R S X b)) =
         polyn-expr A Y (fst (s-cf R S X b))
           (add-cf B (cf-h h (s-cf R S X a)) (cf-h h (s-cf R S X b))),
    frule PolynRg.add-cf-pol-coeff[of A B Y cf-h h (s-cf R S X a)
           cf-h h (s-cf R S X b)], assumption+)
apply (case-tac fst (s-cf R S X (a ±_R b)) = fst (s-cf R S X b))
apply (frule PolynRg.pol-expr-unique2[of A B Y cf-h h (s-cf R S X (a ± b))
           add-cf B (cf-h h (s-cf R S X a)) (cf-h h (s-cf R S X b))],
    assumption+)
    apply (simp add:cf-h-len add-cf-len,
          simp add:PolynRg.add-cf-len cf-h-len)
    apply (simp add:PolynRg.add-cf-len cf-h-len,
    thin-tac (polyn-expr A Y (fst (s-cf R S X b)) (cf-h h (s-cf R S X
     (a ± b))) = polyn-expr A Y (fst (s-cf R S X b))
      (add-cf B (cf-h h (s-cf R S X a)) (cf-h h (s-cf R S X b)))) =
      (∀ j≤fst (s-cf R S X b).
       snd (cf-h h (s-cf R S X (a ± b))) j =
       snd (add-cf B (cf-h h (s-cf R S X a)) (cf-h h (s-cf R S X b))) j))

apply (frule pol-expr-unique2[of s-cf R S X (a ± b)
           add-cf S (s-cf R S X a) (s-cf R S X b)])
    apply (simp add:add-cf-pol-coeff)
    apply (simp add:add-cf-len, simp add:add-cf-len,
    thin-tac polyn-expr R X (fst (s-cf R S X b)) (s-cf R S X (a ± b)) =
             polyn-expr R X (fst (s-cf R S X b))
             (add-cf S (s-cf R S X a) (s-cf R S X b)))
apply (rule allI, rule impI,
    drule-tac a = j in forall-spec, assumption,
```

subst add-cf-def, *simp add*:*cf-h-len*,
(*subst cf-h-def*)+, (*subst cmp-def*)+, *simp*,
thin-tac snd (*s-cf R S X* (*a* ± *b*)) *j* =
snd (*add-cf S* (*s-cf R S X a*) (*s-cf R S X b*)) *j*)
apply (*subst add-cf-def*, *simp*)
apply (*frule-tac j* = *j* **in** *pol-coeff-mem*[*of s-cf R S X a*], *simp*,
frule-tac j = *j* **in** *pol-coeff-mem*[*of s-cf R S X b*], *simp*,
simp add:*rHom-add*)

apply (*frule s-cf-deg*[*of a*], *assumption*,
frule s-cf-deg[*of b*], *assumption*,
frule s-cf-deg[*of a* ± *b*], *assumption*,
frule deg-pols-add2[*of a b*], *assumption*+,
simp add:*deg-def*, *simp add*:*deg-def ale-natle*,
frule-tac m = *fst* (*s-cf R S X* (*a* ± *b*)) **and** *n* = *fst* (*s-cf R S X b*)
in *noteq-le-less*, *assumption*+)
apply (*frule pol-expr-unique3*[*of s-cf R S X* (*a* ± *b*)
add-cf S (*s-cf R S X a*) (*s-cf R S X b*)],
simp add:*add-cf-pol-coeff*,
simp add:*add-cf-len*,
simp add:*add-cf-len*,
thin-tac polyn-expr R X (*fst* (*s-cf R S X* (*a* ± *b*))) (*s-cf R S X* (*a* ± *b*))
= *polyn-expr R X* (*fst* (*s-cf R S X b*))
(*add-cf S* (*s-cf R S X a*) (*s-cf R S X b*)))
apply (*frule PolynRg.pol-expr-unique3*[*of A B Y cf-h h* (*s-cf R S X* (*a* ± *b*))
add-cf B (*cf-h h* (*s-cf R S X a*)) (*cf-h h* (*s-cf R S X b*))],
assumption+,
simp add:*cf-h-len PolynRg.add-cf-len*,
simp add:*PolynRg.add-cf-len cf-h-len*,
thin-tac (*polyn-expr A Y* (*fst* (*s-cf R S X* (*a* ± *b*)))
(*cf-h h* (*s-cf R S X* (*a* ± *b*))) = *polyn-expr A Y* (*fst* (*s-cf R S X b*))
(*add-cf B* (*cf-h h* (*s-cf R S X a*)) (*cf-h h* (*s-cf R S X b*)))) =
((∀ *j*≤*fst* (*s-cf R S X* (*a* ± *b*)).
snd (*cf-h h* (*s-cf R S X* (*a* ± *b*))) *j* =
snd (*add-cf B* (*cf-h h* (*s-cf R S X a*)) (*cf-h h* (*s-cf R S X b*))) *j*) ∧
(∀ *j*∈*nset* (*Suc* (*fst* (*s-cf R S X* (*a* ± *b*)))) (*fst* (*s-cf R S X b*)).
snd (*add-cf B* (*cf-h h* (*s-cf R S X a*)) (*cf-h h* (*s-cf R S X b*))) *j* =
$\mathbf{0}_B$)),
thin-tac pol-coeff B (*cf-h h* (*s-cf R S X a*)),
thin-tac pol-coeff B (*cf-h h* (*s-cf R S X b*)),
thin-tac pol-coeff B (*cf-h h* (*s-cf R S X* (*a* ± *b*))),
thin-tac pol-coeff B (*add-cf B* (*cf-h h* (*s-cf R S X a*))
(*cf-h h* (*s-cf R S X b*))),
thin-tac deg-n R S X a = *fst* (*s-cf R S X b*),
thin-tac deg-n R S X b = *fst* (*s-cf R S X b*),
thin-tac deg-n R S X (*a* ± *b*) = *fst* (*s-cf R S X* (*a* ± *b*)))
apply (*rule conjI*, *erule conjE*,
thin-tac ∀ *j*∈*nset* (*Suc* (*fst* (*s-cf R S X* (*a* ± *b*)))) (*fst* (*s-cf R S X b*)).
snd (*add-cf S* (*s-cf R S X a*) (*s-cf R S X b*)) *j* = $\mathbf{0}_S$)

apply (*rule allI*, *rule impI*,
drule-tac $a = j$ **in** *forall-spec*, *assumption*,
(*subst cf-h-def*)+, (*subst cmp-def*)+, *simp*,
(*subst add-cf-def*)+, *simp*,
frule-tac $j = j$ **in** *pol-coeff-mem*[*of s-cf R S X a*], *simp*,
frule-tac $j = j$ **in** *pol-coeff-mem*[*of s-cf R S X b*], *simp*,
simp add:*rHom-add*)
apply (*erule conjE*,
thin-tac $\forall j{\leq}fst\ (s\text{-}cf\ R\ S\ X\ (a \pm b)).\ snd\ (s\text{-}cf\ R\ S\ X\ (a \pm b))\ j =$
snd (*add-cf S* (*s-cf R S X a*) (*s-cf R S X b*)) *j*)
apply (*rule ballI*,
drule-tac $x = j$ **in** *bspec*, *assumption*,
simp add:*add-cf-def cf-h-len*,
simp add:*cf-h-def cmp-def*,
frule-tac $j = j$ **in** *pol-coeff-mem*[*of s-cf R S X a*],
simp add:*nset-def*,
frule-tac $j = j$ **in** *pol-coeff-mem*[*of s-cf R S X b*],
simp add:*nset-def*)
apply (*subst rHom-add*[*THEN sym*, *of h S B*], *assumption*+, *simp*,
(*frule PolynRg.is-Ring*[*of A B Y*],
frule Ring.subring-Ring[*of A B*], *assumption*),
simp add:*rHom-0-0*[*of S B*])
done

lemma (**in** *PolynRg*) *erHomTr2*:⟦*PolynRg A B Y*; $h \in rHom\ S\ B$;
$a \in carrier\ R$; $b \in carrier\ R$; $a \neq \mathbf{0}$; $b \neq \mathbf{0}$; $a \pm b \neq \mathbf{0}$;
deg-n R S X a < *deg-n R S X b*⟧ ⟹
erH R S X A B Y h $(a \pm b) =$ *erH R S X A B Y h a* $\pm_A$
(*erH R S X A B Y h b*)
apply (*cut-tac ring-is-ag*,
cut-tac subring, *frule subring-Ring*,
frule PolynRg.subring[*of A B Y*],
frule-tac $x = a$ **and** $y = b$ **in** *aGroup.ag-pOp-closed*[*of R*],
assumption+,
simp add:*erH-def erh-def*)
apply (*frule s-cf-expr*[*of a*], *assumption*,
frule s-cf-expr[*of b*], *assumption*,
frule s-cf-expr[*of a* $\pm$ *b*], *assumption*,
(*erule conjE*)+,
frule polyn-deg-add1[*of a b*], *assumption*+,
simp add:*s-cf-deg*)
apply (*frule polyn-add1*[*of s-cf R S X a s-cf R S X b*], *assumption*+)
apply (*cut-tac* $a = a \pm b$ **and**
$b = polyn\text{-}expr\ R\ X\ (fst\ (s\text{-}cf\ R\ S\ X\ b))\ (s\text{-}cf\ R\ S\ X\ (a \pm b))$ **and**
$c = polyn\text{-}expr\ R\ X\ (max\ (fst\ (s\text{-}cf\ R\ S\ X\ a))\ (fst\ (s\text{-}cf\ R\ S\ X\ b)))$
(*add-cf S* (*s-cf R S X a*) (*s-cf R S X b*)) **in** *box-equation*,
drule sym, *drule sym*, *drule sym*, *simp*,
drule sym, *drule sym*, *drule sym*, *simp*,
thin-tac polyn-expr R X (*fst* (*s-cf R S X a*)) (*s-cf R S X a*) $\pm$

polyn-expr R X (fst (s-cf R S X b)) (s-cf R S X b) =
polyn-expr R X (max (fst (s-cf R S X a)) (fst (s-cf R S X b)))
(add-cf S (s-cf R S X a) (s-cf R S X b)),
thin-tac a = polyn-expr R X (fst (s-cf R S X a)) (s-cf R S X a),
thin-tac b = polyn-expr R X (fst (s-cf R S X b)) (s-cf R S X b),
thin-tac a ± b = polyn-expr R X (fst (s-cf R S X b))
(s-cf R S X (a ± b)),
simp)

apply (*frule cf-h-coeff*[*of A B Y h s-cf R S X a*], *assumption*+,
frule cf-h-coeff[*of A B Y h s-cf R S X b*], *assumption*+,
frule cf-h-coeff[*of A B Y h s-cf R S X (a ± b)*], *assumption*+)
apply (*frule PolynRg.polyn-add1*[*of A B Y cf-h h (s-cf R S X a)*
cf-h h (s-cf R S X b)], *assumption*+,
simp add:*cf-h-len*)
apply (*thin-tac polyn-expr A Y (fst (s-cf R S X a)) (cf-h h (s-cf R S X a))*
$\pm_A$ *polyn-expr A Y (fst (s-cf R S X b)) (cf-h h (s-cf R S X b)) =*
polyn-expr A Y (fst (s-cf R S X b))
(add-cf B (cf-h h (s-cf R S X a)) (cf-h h (s-cf R S X b))),
frule PolynRg.add-cf-pol-coeff[*of A B Y cf-h h (s-cf R S X a)*
cf-h h (s-cf R S X b)], *assumption*+)
apply (*frule PolynRg.pol-expr-unique2*[*of A B Y cf-h h (s-cf R S X (a ± b))*
add-cf B (cf-h h (s-cf R S X a)) (cf-h h (s-cf R S X b))],
assumption+,
simp add:*cf-h-len add-cf-len*,
simp add:*PolynRg.add-cf-len cf-h-len*,
simp add:*PolynRg.add-cf-len cf-h-len*,
thin-tac (*polyn-expr A Y (fst (s-cf R S X b))*
(cf-h h (s-cf R S X (a ± b))) = polyn-expr A Y (fst (s-cf R S X b))
(add-cf B (cf-h h (s-cf R S X a)) (cf-h h (s-cf R S X b)))) =
($\forall$ *j*$\leq$*fst (s-cf R S X b).*
snd (cf-h h (s-cf R S X (a ± b))) j =
snd (add-cf B (cf-h h (s-cf R S X a)) (cf-h h (s-cf R S X b))) j))

apply (*frule pol-expr-unique2*[*of s-cf R S X (a ± b)*
add-cf S (s-cf R S X a) (s-cf R S X b)],
simp add:*add-cf-pol-coeff*,
simp add:*add-cf-len*, *simp add*:*add-cf-len*)
apply (*thin-tac polyn-expr R X (fst (s-cf R S X b)) (s-cf R S X (a ± b)) =*
polyn-expr R X (fst (s-cf R S X b))
(add-cf S (s-cf R S X a) (s-cf R S X b)))
apply (*rule allI*, *rule impI*,
drule-tac a = j **in** *forall-spec*, *assumption*,
subst add-cf-def, *simp add*:*cf-h-len*,
(*subst cf-h-def*)+, (*subst cmp-def*)+, *simp*,
subst add-cf-def, *simp*)
apply (*case-tac j* $\leq$ *fst (s-cf R S X a)*, *simp*)
apply (*frule-tac j = j* **in** *pol-coeff-mem*[*of s-cf R S X a*], *simp*,
frule-tac j = j **in** *pol-coeff-mem*[*of s-cf R S X b*], *simp*,

```
      simp add:rHom-add)
   apply simp
  apply (subst add-cf-def, simp)
done

lemma (in PolynRg) erH-rHom:⟦Idomain S; PolynRg A B Y; h ∈ rHom S B⟧
  ⟹ erH R S X A B Y h ∈ pHom R S X, A B Y
apply (frule Idomain.idom-is-ring[of S],
       cut-tac subring,
       cut-tac polyn-ring-integral, simp,
       frule PolynRg.subring[of A B Y],
       frule PolynRg.is-Ring[of A B Y],
       frule Ring.subring-Ring[of A B], assumption)
apply (simp add:polyn-Hom-def,
       rule conjI)
 prefer 2
apply (cut-tac polyn-ring-X-nonzero,
       cut-tac X-mem-R, rule conjI,
       rule subsetI, simp add:image-def,
       erule bexE)

apply (case-tac xa = 0_S,
       simp add:Subring-zero-ring-zero,
       simp add:erH-rHom-0,
       simp add:Ring.Subring-zero-ring-zero[THEN sym, of A B],
       simp add:Ring.ring-zero[of B])
apply (simp add:Subring-zero-ring-zero,
       frule-tac x = xa in mem-subring-mem-ring, assumption,
       frule-tac p = xa in s-cf-expr, assumption+, (erule conjE)+,
       frule-tac p1 = xa in s-cf-deg[THEN sym], assumption+,
       frule-tac p1 = xa in pol-of-deg0[THEN sym], assumption+, simp)
apply (simp add:erH-def erh-def, subst polyn-expr-def, simp,
       frule-tac c = s-cf R S X xa in cf-h-coeff[of A B Y h], assumption+,
       frule-tac c = cf-h h (s-cf R S X xa) and j = 0 in
                     PolynRg.pol-coeff-mem[of A B Y], assumption, simp,
       frule-tac c = cf-h h (s-cf R S X xa) and j = 0 in
                     PolynRg.pol-coeff-mem-R[of A B Y], assumption, simp,
       simp add:Ring.ring-r-one[of A])

apply (cut-tac pol-expr-of-X,
       cut-tac special-cf-pol-coeff,
       frule ext-cf-pol-coeff[of C_0 S Suc 0])
apply (simp add:erH-def erh-def)
apply (cut-tac deg-n-of-X)
 apply (frule s-cf-expr[of X], assumption+, (erule conjE)+,
        frule-tac a = X and
        b = polyn-expr R X (Suc 0) (ext-cf S (Suc 0) (C_0 S)) and
        c = polyn-expr R X (fst (s-cf R S X X)) (s-cf R S X X) in
        box-equation, assumption+,
```

thin-tac X = *polyn-expr* R X (*Suc* 0) (*ext-cf* S (*Suc* 0) (C_0 S)),
thin-tac X = *polyn-expr* R X (*fst* (*s-cf* R S X X)) (*s-cf* R S X X))
apply (*rule sym*, *subst PolynRg.pol-expr-of-X*[*of* A B Y], *assumption*+,
frule s-cf-deg[*of* X], *assumption*+, *simp*)
apply (*frule pol-expr-unique2*[*of ext-cf* S (*Suc* 0) (C_0 S) *s-cf* R S X X],
assumption+, *simp add*:*ext-cf-len special-cf-len*,
simp add:*ext-cf-len special-cf-len*)
apply (*frule cf-h-coeff*[*of* A B Y h *ext-cf* S (*Suc* 0) (C_0 S)], *assumption*+,
frule cf-h-coeff[*of* A B Y h *s-cf* R S X X], *assumption*+)
apply (*frule PolynRg.pol-expr-unique2*[*of* A B Y
cf-h h (*ext-cf* S (*Suc* 0) (C_0 S)) *cf-h* h (*s-cf* R S X X)],
assumption+,
simp add:*cf-h-len ext-cf-len special-cf-len*,
simp add:*cf-h-len ext-cf-len special-cf-len*)

apply (*simp add*:*cf-h-special-cf*)
apply (*thin-tac* (*polyn-expr* A Y (*Suc* 0) (*ext-cf* B (*Suc* 0) (C_0 B)) =
polyn-expr A Y (*Suc* 0) (*cf-h* h (*s-cf* R S X X))) =
($\forall j \leq$*Suc* 0.
snd (*cf-h* h (*ext-cf* S (*Suc* 0) (C_0 S))) j =
snd (*cf-h* h (*s-cf* R S X X)) j))
apply (*rule allI*, *rule impI*,
drule-tac a = j **in** *forall-spec*, *assumption*,
(*subst cf-h-def*)+, (*subst cmp-def*)+, *simp*)

apply (*subst rHom-def*, *simp*,
cut-tac ring-is-ag,
frule Ring.ring-is-ag[*of* A])
apply (*rule conjI*)
apply (*subst aHom-def*, *simp*)
apply (*rule conjI*)
apply (*simp add*:*erH-mem*)
apply (*rule conjI*, *simp add*:*erH-def erh-def extensional-def*)
apply (*rule ballI*)+

apply (*case-tac* a = $\mathbf{0}_R$,
case-tac b = $\mathbf{0}_R$, *simp*)
apply (*simp add*:*erH-rHom-0*,
frule Ring.ring-is-ag[*of* A], *frule Ring.ring-zero*[*of* A],
simp add:*aGroup.ag-r-zero*)
apply (*simp add*:*erH-rHom-0*, *simp add*:*erH-rHom-0*)
apply (*frule-tac* p = b **in** *erH-mem*[*of* A B Y h], *assumption*+)
apply (*simp add*:*aGroup.ag-l-zero*)

apply (*case-tac* b = $\mathbf{0}_R$, *simp*)
apply (*simp add*:*erH-rHom-0*,
frule-tac p = a **in** *erH-mem*[*of* A B Y h], *assumption*+,
simp add:*aGroup.ag-r-zero*)

apply (*case-tac* $a \pm_R b = \mathbf{0}_R$, *simp add*:*erH-rHom-0*)
apply (*frule-tac* $x = a$ **and** $y = b$ **in** *aGroup.ag-inv-unique*[*of R*],
assumption+, *simp*,
thin-tac $b = -_a a$)
apply (*subst erHomTr0*[*of A B Y h*], *assumption*+,
frule-tac $p = a$ **in** *erH-mem*[*of A B Y h*], *assumption*+,
simp add:*aGroup.ag-r-inv1*)

apply (*case-tac deg-n R S X a* = *deg-n R S X b*,
simp add:*erHomTr1*[*of A B Y h*])
apply (*cut-tac y* = *deg-n R S X a* **and** *x* = *deg-n R S X b* **in** *less-linear*,
simp)
apply (*erule disjE*)
apply (*subst aGroup.ag-pOp-commute*, *assumption*+,
frule-tac $p = a$ **in** *erH-mem*[*of A B Y h*], *assumption*+,
frule-tac $p = b$ **in** *erH-mem*[*of A B Y h*], *assumption*+,
subst aGroup.ag-pOp-commute[*of A*], *assumption*+,
rule erHomTr2[*of A B Y h*], *assumption*+,
simp add:*aGroup.ag-pOp-commute*, *assumption*)

apply(*rule erHomTr2*[*of A B Y h*], *assumption*+)

apply (*simp add*:*erH-rHomTr2*)
apply (*rule ballI*)+
apply (*case-tac* $x = \mathbf{0}_R$, *simp add*:*ring-times-0-x erH-rHom-0*,
frule-tac $p = y$ **in** *erH-mem*[*of A B Y h*], *assumption*+,
simp add:*Ring.ring-times-0-x*[*of A*])
apply (*case-tac* $y = \mathbf{0}_R$, *simp add*:*ring-times-x-0 erH-rHom-0*,
frule-tac $p = x$ **in** *erH-mem*[*of A B Y h*], *assumption*+,
simp add:*Ring.ring-times-x-0*[*of A*])

apply (*frule-tac* $p = x$ **in** *s-cf-expr*, *assumption*+,
frule-tac $p = y$ **in** *s-cf-expr*, *assumption*+,
frule-tac $x = x$ **and** $y = y$ **in** *ring-tOp-closed*, *assumption*+,
frule-tac $p = x \cdot_r y$ **in** *s-cf-expr*,
simp add:*Idomain.idom-tOp-nonzeros*, (*erule conjE*)+)

apply (*frule-tac c* = *s-cf R S X x* **and** *d* = *s-cf R S X y* **in**
polyn-expr-tOp-c, *assumption*+, *erule exE*, (*erule conjE*)+,
cut-tac $a = x \cdot_r y$ **and**
b = *polyn-expr R X* (*fst* (*s-cf R S X* ($x \cdot_r y$))) (*s-cf R S X* ($x \cdot_r y$))
and *c* = *polyn-expr R X* (*fst e*) *e* **in** *box-equation*)
apply *assumption*
apply (*thin-tac* $x \cdot_r y =$
polyn-expr R X (*fst* (*s-cf R S X* ($x \cdot_r y$))) (*s-cf R S X* ($x \cdot_r y$)))
apply (*drule sym*, *drule sym*, *simp*)

apply (*thin-tac x* = *polyn-expr R X* (*fst* (*s-cf R S X x*)) (*s-cf R S X x*),
thin-tac y = *polyn-expr R X* (*fst* (*s-cf R S X y*)) (*s-cf R S X y*),

thin-tac $x \cdot_r y =$
polyn-expr $R\ X$ (*fst* (*s-cf* $R\ S\ X$ ($x \cdot_r y$))) (*s-cf* $R\ S\ X$ ($x \cdot_r y$)))

apply ((*subst erH-def*)+, (*subst erh-def*)+, *simp*)

apply (*frule-tac* $c = $ *s-cf* $R\ S\ X\ x$ **and** $d = $ *s-cf* $R\ S\ X\ y$ **and** $e = e$ **in**
erH-multTr1[*of* $A\ B\ Y\ h$], *assumption*+, *simp*, *simp*,
thin-tac polyn-expr $A\ Y$ (*fst* (*s-cf* $R\ S\ X\ x$)) (*cf-h* h (*s-cf* $R\ S\ X\ x$)) $\cdot_r A$
polyn-expr $A\ Y$ (*fst* (*s-cf* $R\ S\ X\ y$)) (*cf-h* h (*s-cf* $R\ S\ X\ y$)) =
polyn-expr $A\ Y$ (*fst* (*s-cf* $R\ S\ X\ x$) + *fst* (*s-cf* $R\ S\ X\ y$)) (*cf-h* $h\ e$))
apply (*rotate-tac* -1, *drule sym*, *simp*,
thin-tac polyn-expr $R\ X$ (*fst* (*s-cf* $R\ S\ X\ x$)) (*s-cf* $R\ S\ X\ x$) $\cdot_r$
polyn-expr $R\ X$ (*fst* (*s-cf* $R\ S\ X\ y$)) (*s-cf* $R\ S\ X\ y$) =
polyn-expr $R\ X$ (*fst* (*s-cf* $R\ S\ X$ ($x \cdot_r y$))) (*s-cf* $R\ S\ X$ ($x \cdot_r y$)),
rotate-tac -1, *drule sym*)
apply (*frule-tac* $p = x$ **in** *s-cf-deg*, *assumption*,
frule-tac $p = y$ **in** *s-cf-deg*, *assumption*,
frule-tac $x = x$ **and** $y = y$ **in** *Idomain.idom-tOp-nonzeros*[*of* R],
assumption+,
frule-tac $p = x \cdot_r y$ **in** *s-cf-deg*, *assumption*+)
apply (*frule-tac* $p = x$ **and** $q = y$ **in** *deg-mult-pols*, *assumption*+,
(*erule conjE*)+, *simp*,
thin-tac deg-n $R\ S\ X\ x =$ *fst* (*s-cf* $R\ S\ X\ x$),
thin-tac deg-n $R\ S\ X\ y =$ *fst* (*s-cf* $R\ S\ X\ y$),
thin-tac deg-n $R\ S\ X$ ($x \cdot_r y$) = *fst* (*s-cf* $R\ S\ X$ ($x \cdot_r y$)),
rotate-tac -2, *drule sym*)
apply (*simp add*:*cf-h-len*)
apply (*frule-tac* $c = $ *s-cf* $R\ S\ X$ ($x \cdot_r y$) **in** *cf-h-coeff*[*of* $A\ B\ Y\ h$],
assumption+,
frule-tac $c = e$ **in** *cf-h-coeff*[*of* $A\ B\ Y\ h$], *assumption*+)
apply (*frule-tac* $c = $ *cf-h* h (*s-cf* $R\ S\ X$ ($x \cdot_r y$)) **and** $d = $ *cf-h* $h\ e$ **in**
PolynRg.pol-expr-unique2[*of* $A\ B\ Y$], *assumption*+,
simp add:*cf-h-len*, *simp add*:*cf-h-len*,
thin-tac (*polyn-expr* $A\ Y$ (*fst* (*s-cf* $R\ S\ X\ x$) + *fst* (*s-cf* $R\ S\ X\ y$))
(*cf-h* h (*s-cf* $R\ S\ X$ ($x \cdot_r y$))) =
polyn-expr $A\ Y$ (*fst* (*s-cf* $R\ S\ X\ x$) + *fst* (*s-cf* $R\ S\ X\ y$))
(*cf-h* $h\ e$)) =
($\forall j \leq$ *fst* (*s-cf* $R\ S\ X\ x$) + *fst* (*s-cf* $R\ S\ X\ y$).
snd (*cf-h* h (*s-cf* $R\ S\ X$ ($x \cdot_r y$))) j = *snd* (*cf-h* $h\ e$) j))
apply (*frule-tac* $c = $ *s-cf* $R\ S\ X$ ($x \cdot_r y$) **and** $d = e$ **in**
pol-expr-unique2, *assumption*+, *simp*, *simp*,
thin-tac polyn-expr $R\ X$ (*fst* (*s-cf* $R\ S\ X\ x$) + *fst* (*s-cf* $R\ S\ X\ y$))
(*s-cf* $R\ S\ X$ ($x \cdot_r y$)) =
polyn-expr $R\ X$ (*fst* (*s-cf* $R\ S\ X\ x$) + *fst* (*s-cf* $R\ S\ X\ y$)) e)
apply (*rule allI*, *rule impI*, *drule-tac* $a = j$ **in** *forall-spec*, *assumption*,
subst cf-h-def, *subst cmp-def*, *simp*,
subst cf-h-def, *subst cmp-def*, *simp*)
done

lemma (**in** *PolynRg*) *erH-q-rHom*:⟦*Idomain S*; *maximal-ideal S P*;
PolynRg R′ (S /r P) Y⟧ ⟹
erH R S X R′ (S /r P) Y (pj S P) ∈ pHom R S X, *R′ (S /r P) Y*
apply (*frule Idomain.idom-is-ring*[*of S*],
frule Ring.qring-ring[*of S P*], *simp add*:*Ring.maximal-ideal-ideal*,
rule erH-rHom[*of R′ S /r P Y pj S P*], *assumption*+)
apply (*rule pj-Hom*[*of S P*], *assumption*+,
simp add:*Ring.maximal-ideal-ideal*)
done

lemma (**in** *PolynRg*) *erH-add*:⟦*Idomain S*; *PolynRg A B Y*; *h ∈ rHom S B*;
p ∈ carrier R; *q ∈ carrier R*⟧ ⟹
erH R S X A B Y h (p ± q) =
(erH R S X A B Y h p) ±A (erH R S X A B Y h q)
apply (*frule erH-rHom*[*of A B Y h*], *assumption*+)
apply (*simp add*:*polyn-Hom-def*, (*erule conjE*)+)
apply (*simp add*:*rHom-add*)
done

lemma (**in** *PolynRg*) *erH-minus*:⟦*Idomain S*; *PolynRg A B Y*;
h ∈ rHom S B; *p ∈ carrier R*⟧ ⟹
erH R S X A B Y h (−a p) = −a A (erH R S X A B Y h p)
apply (*frule erH-rHom*[*of A B Y h*], *assumption*+,
simp add:*polyn-Hom-def*, (*erule conjE*)+)
apply (*frule PolynRg.is-Ring*[*of A B Y*])
apply (*rule rHom-inv-inv*[*of R A p erH R S X A B Y h*])
apply (*cut-tac is-Ring*, *assumption*+)
done

lemma (**in** *PolynRg*) *erH-mult*:⟦*Idomain S*; *PolynRg A B Y*; *h ∈ rHom S B*;
p ∈ carrier R; *q ∈ carrier R*⟧ ⟹
erH R S X A B Y h (p ·r q) =
(erH R S X A B Y h p) ·r A (erH R S X A B Y h q)
apply (*frule erH-rHom*[*of A B Y h*], *assumption*+,
simp add:*polyn-Hom-def*, (*erule conjE*)+,
frule PolynRg.is-Ring[*of A B Y*],
cut-tac is-Ring,
rule rHom-tOp[*of R A p q erH R S X A B Y h*], *assumption*+)
done

lemma (**in** *PolynRg*) *erH-rHom-cf*:⟦*Idomain S*; *PolynRg A B Y*; *h ∈ rHom S B*;

s ∈ carrier S⟧ ⟹ *erH R S X A B Y h s = h s*
apply (*cut-tac subring*, *frule subring-Ring*,
frule PolynRg.subring[*of A B Y*],
frule PolynRg.is-Ring[*of A B Y*],
frule Ring.subring-Ring[*of A B*], *assumption*,
frule mem-subring-mem-ring[*of S s*], *assumption*+)
apply (*case-tac s = 0S*, *simp add*:*Subring-zero-ring-zero*,

simp add:*erH-rHom-0*,
simp add:*Subring-zero-ring-zero*[*THEN sym*],
simp add:*rHom-0-0*, *simp add*:*Ring.Subring-zero-ring-zero*)
apply (*frule s-cf-expr*[*of s*],*simp add*:*Subring-zero-ring-zero*,
(*erule conjE*)+,
simp add:*Subring-zero-ring-zero*)
apply (*frule s-cf-deg*[*of s*], *assumption*+,
frule pol-of-deg0[*of s*], *assumption*+, *simp*)
apply (*subst erH-def*, *simp*,
subst erh-rHom-coeff[*of A B Y h s-cf R S X s*], *assumption*+,
simp add:*cmp-def polyn-expr-def*,
frule-tac c = *s-cf R S X s* **and** *j* = *0* **in** *pol-coeff-mem*, *simp*,
frule mem-subring-mem-ring[*of S snd* (*s-cf R S X s*) *0*], *assumption*+,
simp add:*ring-r-one*)
done

lemma (**in** *PolynRg*) *erH-rHom-coeff*:⟦*Idomain S*; *PolynRg A B Y*; *h* ∈ *rHom S B*;
pol-coeff S (*n*, *f*)⟧ ⟹ *pol-coeff B* (*n*, *cmp h f*)
apply (*simp add*:*pol-coeff-def*)
apply (*rule allI*, *rule impI*, *drule-tac a* = *j* **in** *forall-spec*, *assumption*)
apply (*simp add*:*cmp-def rHom-mem*)
done

lemma (**in** *PolynRg*) *erH-rHom-unique*:⟦*Idomain S*; *PolynRg A B Y*; *h* ∈ *rHom S B*⟧
⟹ ∃!*g*. *g* ∈ *pHom R S X*, *A B Y* ∧ (∀ *x*∈*carrier S*. *h x* = *g x*)
apply (*cut-tac subring*, *frule subring-Ring*,
cut-tac is-Ring,
frule PolynRg.subring[*of A B Y*],
frule PolynRg.is-Ring[*of A B Y*],
frule Ring.subring-Ring[*of A B*], *assumption*,
frule Idomain.idom-is-ring[*of S*])

apply (*rule ex-ex1I*)
apply (*frule erH-rHom*[*of A B Y h*], *assumption*+)
apply (*subgoal-tac* ∀ *x*∈*carrier S*. *h x* = (*erH R S X A B Y h*) *x*, *blast*)
apply (*rule ballI*, *simp add*:*erH-rHom-cf*, (*erule conjE*)+)
apply (*frule pHom-rHom*[*of A B Y*], *assumption*+,
frule pHom-rHom[*of A B Y*], *assumption*+,
frule-tac f = *g* **in** *rHom-func*[*of - R A*],
frule-tac f = *y* **in** *rHom-func*[*of - R A*])
apply (*rule-tac f* = *g* **and** *g* = *y* **in** *funcset-eq*[*of - carrier R*],
simp add:*rHom-def aHom-def*, *simp add*:*rHom-def aHom-def*)
apply (*rule ballI*,
thin-tac g ∈ *carrier R* → *carrier A*,
thin-tac y ∈ *carrier R* → *carrier A*)

apply (*case-tac x* = $\mathbf{0}_R$, *simp*,

subst rHom-0-0[*of R A*], *assumption*+, *rule sym*,
subst rHom-0-0[*of R A*], *assumption*+, *simp*)
apply (*subst pHom-pol-mem*[*of A B Y*], *assumption*+)
apply (*frule-tac f* = *y* **and** *p* = *x* **in** *pHom-pol-mem*[*of A B Y*], *assumption*+,
simp,
frule-tac f = *g* **and** *c* = *s-cf R S X x* **in** *polyn-Hom-coeff-to-coeff*,
assumption+, *simp add*:*s-cf-pol-coeff*,
frule-tac f = *y* **and** *c* = *s-cf R S X x* **in** *polyn-Hom-coeff-to-coeff*,
assumption+, *simp add*:*s-cf-pol-coeff*)
apply (*simp add*:*s-cf-deg*)
apply (*frule-tac f* = *g* **and** *c* = *s-cf R S X x* **in** *cf-h-len1*[*of A B Y h*],
assumption+, *rule ballI*, *rule sym*, *simp*, *rule s-cf-pol-coeff*,
assumption+)
apply (*frule-tac f* = *y* **and** *c* = *s-cf R S X x* **in** *cf-h-len1*[*of A B Y h*],
assumption+, *rule ballI*, *rule sym*, *simp*, *rule s-cf-pol-coeff*,
assumption+)
apply (*frule-tac c* = *cf-h g* (*s-cf R S X x*) **and** *d* = *cf-h y* (*s-cf R S X x*)
in *PolynRg.pol-expr-unique2*[*of A B Y*], *assumption*+, *simp*)
apply (*frule-tac p* = *x* **in** *s-cf-pol-coeff*, *simp add*:*cf-h-len*,
thin-tac (*polyn-expr A Y* (*fst* (*s-cf R S X x*)) (*cf-h g* (*s-cf R S X x*))
= *polyn-expr A Y* (*fst* (*s-cf R S X x*)) (*cf-h y* (*s-cf R S X x*))) =
(∀ *j*≤*fst* (*s-cf R S X x*).
snd (*cf-h g* (*s-cf R S X x*)) *j* = *snd* (*cf-h y* (*s-cf R S X x*)) *j*))
apply (*rule allI*, *rule impI*,
(*subst cf-h-def*)+, (*subst cmp-def*)+, *simp*,
frule-tac c = *s-cf R S X x* **and** *j* = *j* **in** *pol-coeff-mem*, *assumption*+)
apply *simp*
done

lemma (**in** *PolynRg*) *erH-rHom-unique1*:⟦*Idomain S*; *PolynRg A B Y*; *h* ∈ *rHom S B*;
f ∈ *pHom R S X*, *A B Y*; ∀ *x* ∈ *carrier S*. *f x* = *h x*⟧ ⟹
f = (*erH R S X A B Y h*)
apply (*frule erH-rHom-unique*[*of A B Y h*], *assumption*+,
erule ex1E,
frule-tac x = *f* **in** *spec*,
drule-tac x = *erH R S X A B Y h* **in** *spec*)
apply (*frule erH-rHom*[*of A B Y h*], *assumption*+,
simp add:*erH-rHom-cf*[*THEN sym*])
done

lemma (**in** *PolynRg*) *pHom-dec-deg*:⟦*PolynRg A B Y*; *f* ∈ *pHom R S X*, *A B Y*;
p ∈ *carrier R*⟧ ⟹
deg A B Y (*f p*) ≤ *deg R S X p*
apply (*cut-tac subring*, *frule subring-Ring*,
frule PolynRg.subring[*of A B Y*],
cut-tac is-Ring)
apply (*frule PolynRg.is-Ring*[*of A B Y*],
frule Ring.subring-Ring[*of A B*], *assumption*)

apply (*case-tac f p* = $\mathbf{0}_A$,
case-tac p = $\mathbf{0}_R$,
simp add:*deg-def*, *simp add*:*deg-def an-def*,
simp add:*deg-def*, *subst ale-natle*)
apply (*case-tac p* = $\mathbf{0}_R$,
frule pHom-rHom[*of A B Y f*], *assumption*+,
rule conjI, *rule impI*, *frule rHom-0-0*[*of R A f*], *assumption*+,
simp, *rule impI*, *simp*)

apply *simp*
apply (*frule pHom-pol-mem*[*of A B Y f p*], *assumption*+)
apply (*cut-tac polyn-Hom-coeff-to-coeff*[*of A B Y f s-cf R S X p*])
apply (*frule PolynRg.pol-deg-le-n*[*of A B Y f p cf-h f* (*s-cf R S X p*)],
frule pHom-rHom[*of A B Y f*], *assumption*+,
rule rHom-mem[*of f R A p*], *assumption*+,
frule s-cf-pol-coeff[*of p*],
subst cf-h-len2[*of A B Y f s-cf R S X p*], *assumption*+,
simp add:*s-cf-deg*,
thin-tac f p = *polyn-expr A Y* (*deg-n R S X p*) (*cf-h f* (*s-cf R S X p*)))
apply (*frule s-cf-pol-coeff*[*of p*], *simp add*:*cf-h-len2*,
simp add:*s-cf-deg*[*THEN sym*],
assumption+,
simp add:*s-cf-pol-coeff*)
done

lemma (**in** *PolynRg*) *erH-map*:⟦*Idomain S*; *PolynRg A B Y*; *h* ∈ *rHom S B*;
pol-coeff S (*n*, *c*)⟧ ⟹
(*erH R S X A B Y h*) (*polyn-expr R X n* (*n*, *c*)) =
polyn-expr A Y n (*n*, (*cmp h c*))
apply (*cut-tac subring*, *frule subring-Ring*,
frule PolynRg.subring[*of A B Y*],
cut-tac is-Ring,
frule PolynRg.is-Ring[*of A B Y*],
frule Ring.subring-Ring[*of A B*], *assumption*)
apply (*case-tac polyn-expr R X n* (*n*, *c*) = $\mathbf{0}_R$, *simp add*:*erH-rHom-0*)
apply (*frule coeff-0-pol-0*[*THEN sym*, *of* (*n*, *c*) *n*], *simp*, *simp*,
thin-tac polyn-expr R X n (*n*, *c*) = **0**)
apply (*frule cf-h-coeff*[*of A B Y h* (*n*, *c*)], *assumption*+,
simp add:*cf-h-pol-coeff*)
apply (*rule sym*,
frule-tac PolynRg.coeff-0-pol-0[*THEN sym*, *of A B Y* (*n*, *cmp h c*) *n*],
simp+)
apply (*rule allI*, *rule impI*, *simp add*:*cmp-def*, *simp add*:*rHom-0-0*)
apply (*frule erH-rHom*[*of A B Y h*], *assumption*+)
apply (*subst pHom-mem*[*of A B Y erH R S X A B Y h n c*], *assumption*+)
apply (*frule PolynRg.pol-expr-unique2*[*of A B Y*
(*n*, *cmp* (*erH R S X A B Y h*) *c*) (*n*, *cmp h c*)],
simp add:*cmp-pol-coeff-e*, *simp add*:*cmp-pol-coeff*)
apply (*simp*, *simp*,

thin-tac (*polyn-expr A Y n* (*n*, *cmp* (*erH R S X A B Y h*) *c*) =
polyn-expr A Y n (*n*, *cmp h c*)) =
(∀ *j*≤*n*. *cmp* (*erH R S X A B Y h*) *c j* = *cmp h c j*))
apply (*rule allI*, *rule impI*,
frule-tac j = *j* **in** *pol-coeff-mem*[*of* (*n*, *c*)], *simp*,
simp add:*cmp-def*)
apply (*simp add*:*erH-rHom-cf*)
done

4.16 Relatively prime polynomials

definition
rel-prime-pols :: [('*a*, '*m*) *Ring-scheme*, ('*a*, '*m1*) *Ring-scheme*, '*a*,
'*a*, '*a*] ⇒ *bool* **where**
rel-prime-pols R S X p q ⟷ (1$_{r\,R}$) ∈ ((*Rxa R p*) ∓$_R$ (*Rxa R q*))

definition
div-condn :: [('*a*, '*m*) *Ring-scheme*, ('*a*, '*m1*) *Ring-scheme*, '*a*, *nat*,
'*a*, '*a*] ⇒ *bool* **where**
div-condn R S X n g f ⟷ *f* ∈ *carrier R* ∧ *n* = *deg-n R S X f* ⟶
(∃ *q*. *q* ∈ *carrier R* ∧ ((*f* ±$_R$ (−$_{a\,R}$ (*q* ·$_{r\,R}$ *g*)) = **0**$_R$) ∨ (*deg-n R S X*
(*f* ±$_R$ (−$_{a\,R}$ (*q* ·$_{r\,R}$ *g*))) < *deg-n R S X g*)))

lemma (**in** *PolynRg*) *divisionTr0*:⟦*Idomain S*; *p* ∈ *carrier R*;
c ∈ *carrier S*; *c* ≠ **0**$_S$⟧ ⟹
lcf R S X (*c* ·$_r$ *X*^$^{R\ n}$ ·$_r$ *p*) = *c* ·$_{r\,S}$ (*lcf R S X p*)
apply (*cut-tac polyn-ring-integral*, *simp*,
cut-tac subring, *frule subring-Ring*,
cut-tac polyn-ring-X-nonzero,
cut-tac X-mem-R)
apply (*frule mem-subring-mem-ring*[*of S c*], *assumption*+,
frule npClose[*of X n*])
apply (*case-tac p* = **0**$_R$, *simp*,
frule ring-tOp-closed[*of c X*^$^{R\ n}$], *assumption*+,
simp add:*ring-times-x-0 lcf-val-0*,
simp add:*Ring.ring-times-x-0*[*of S*])
apply (*frule-tac x* = *c* **and** *y* = *X*^$^{R\ n}$ **in** *Idomain.idom-tOp-nonzeros*[*of R*],
assumption+,
simp add:*Subring-zero-ring-zero*,
frule Idomain.idom-potent-nonzero[*of R X n*], *assumption*+,
frule-tac x = *c* **and** *y* = *X*^$^{R\ n}$ **in** *ring-tOp-closed*, *assumption*+,
frule-tac x = *c* ·$_r$ *X*^$^{R\ n}$ **and** *y* = *p* **in** *Idomain.idom-tOp-nonzeros*[*of R*],
assumption+,
frule-tac x = *c* ·$_r$ *X*^$^{R\ n}$ **and** *y* = *p* **in** *ring-tOp-closed*, *assumption*+)
apply (*simp add*:*lcf-val*)

apply (*frule s-cf-expr*[*of p*], *assumption*, (*erule conjE*)+,
simp add:*ring-tOp-assoc*[*of c - p*],

```
      frule low-deg-terms-zero1[THEN sym, of s-cf R S X p n])
apply (cut-tac a = X^R n ·_r
                  polyn-expr R X (fst (s-cf R S X p)) (s-cf R S X p) and
              b = X^R n ·_r p and
              c = polyn-expr R X (fst (s-cf R S X p) + n)
                   (ext-cf S n (s-cf R S X p)) in box-equation,
      simp, assumption,
   thin-tac p = polyn-expr R X (fst (s-cf R S X p)) (s-cf R S X p),
   thin-tac X^R n ·_r polyn-expr R X (fst (s-cf R S X p)) (s-cf R S X p) =
     polyn-expr R X (fst (s-cf R S X p) + n) (ext-cf S n (s-cf R S X p)))
apply (frule ext-cf-pol-coeff[of s-cf R S X p n],
      frule scalar-times-pol-expr[of c ext-cf S n (s-cf R S X p)
          fst (s-cf R S X p) + n], assumption+,
      simp add:ext-cf-len)
   apply (frule sp-cf-pol-coeff[of ext-cf S n (s-cf R S X p) c],
         assumption+,
     cut-tac a = c ·_r
     polyn-expr R X (fst (s-cf R S X p) + n) (ext-cf S n (s-cf R S X p))
     and b = c ·_r (X^R n ·_r p) and
         c = polyn-expr R X (fst (s-cf R S X p) + n)
             (sp-cf S c (ext-cf S n (s-cf R S X p))) in box-equation,
     simp, simp,
     thin-tac X^R n ·_r p =
       polyn-expr R X (fst (s-cf R S X p) + n) (ext-cf S n (s-cf R S X p)),
     thin-tac c ·_r
       polyn-expr R X (fst (s-cf R S X p) + n) (ext-cf S n (s-cf R S X p)) =
       polyn-expr R X (fst (s-cf R S X p) + n)
       (sp-cf S c (ext-cf S n (s-cf R S X p))))
  apply (frule s-cf-expr[of c ·_r (X^R n ·_r p)], assumption+,
        (erule conjE)+,
        drule-tac a = c ·_r (X^R n ·_r p) and
        b = polyn-expr R X (fst (s-cf R S X (c ·_r (X^R n ·_r p))))
                  (s-cf R S X (c ·_r (X^R n ·_r p))) and
        c = polyn-expr R X (fst (s-cf R S X p) + n)
            (sp-cf S c (ext-cf S n (s-cf R S X p))) in box-equation,
       assumption,
       thin-tac c ·_r (X^R n ·_r p) =
         polyn-expr R X (fst (s-cf R S X p) + n)
         (sp-cf S c (ext-cf S n (s-cf R S X p))),
       frule pol-expr-unique2[of s-cf R S X (c ·_r (X^R n ·_r p))
             sp-cf S c (ext-cf S n (s-cf R S X p))], assumption+,
       subst s-cf-deg[THEN sym], assumption+,
        frule-tac Idomain.idom-potent-nonzero[of R X n], assumption+,
        frule-tac x = X^R n and y = p in Idomain.idom-tOp-nonzeros[of R],
        assumption+)
 apply (subst deg-mult-pols, assumption+, simp add:Subring-zero-ring-zero,
        simp add:ring-tOp-closed, assumption+,
        subst deg-mult-pols, assumption+,
```

simp add:*deg-to-X-d*,
cut-tac pol-of-deg0[*THEN sym*, *of c*], *simp*,
simp add:*sp-cf-len ext-cf-len s-cf-deg*, *assumption*+,
simp add:*Subring-zero-ring-zero*,
simp add:*sp-cf-len ext-cf-len*)

apply (*subst s-cf-deg*[*THEN sym*], *assumption*+,
frule-tac Idomain.idom-potent-nonzero[*of R X n*], *assumption*+,
frule-tac $x = X \hat{}^{R\ n}$ **and** $y = p$ **in** *Idomain.idom-tOp-nonzeros*[*of R*],
assumption+,
simp add:*s-cf-deg*[*THEN sym*],
frule-tac $x = X \hat{}^{R\ n}$ **and** $y = p$ **in** *ring-tOp-closed*, *assumption*+)
apply (*frule deg-mult-pols*[*of c* $X \hat{}^{R\ n}$ $\cdot_r$ *p*], *assumption*+,
simp add:*Subring-zero-ring-zero*, *assumption*+, (*erule conjE*)+, *simp*,
thin-tac deg-n R S X $(c \cdot_r (X \hat{}^{R\ n} \cdot_r p)) =$
deg-n R S X c + *deg-n R S X* $(X \hat{}^{R\ n} \cdot_r p)$,
cut-tac pol-of-deg0[*THEN sym*, *of c*], *simp*,
frule deg-mult-pols[*of* $X \hat{}^{R\ n}$ *p*], *assumption*+, (*erule conjE*)+,
simp,
thin-tac deg-n R S X $(X \hat{}^{R\ n} \cdot_r p) =$ *deg-n R S X* $(X \hat{}^{R\ n})$ + *deg-n R S X p*,
simp add:*deg-to-X-d*, *simp add*:*add.commute*[*of n*],
thin-tac polyn-expr R X (*deg-n R S X p* + *n*)
(*s-cf R S X* $(c \cdot_r (X \hat{}^{R\ n} \cdot_r p))) =$ *polyn-expr R X* (*deg-n R S X p* + *n*)
(*sp-cf S c* (*ext-cf S n* (*s-cf R S X p*))))
apply (*subst sp-cf-def*, *simp*)
apply (*subst ext-cf-def*, *simp add*:*sliden-def*, *assumption*)
apply (*simp add*:*Subring-zero-ring-zero*)
done

lemma (**in** *PolynRg*) *divisionTr1*:⟦*Corps S*; $g \in$ *carrier R*; $g \neq \mathbf{0}$;
$0 <$ *deg-n R S X g*; $f \in$ *carrier R*; $f \neq \mathbf{0}$; *deg-n R S X g* $\leq$ *deg-n R S X f*⟧
$\Longrightarrow$
$f \pm -_a ((lcf\ R\ S\ X\ f) \cdot_{r S} ((lcf\ R\ S\ X\ g)^{-\ S}) \cdot_r$
$(X \hat{}^R ((deg\text{-}n\ R\ S\ X\ f) - (deg\text{-}n\ R\ S\ X\ g))) \cdot_r g) = \mathbf{0} \vee$
deg-n R S X $(f \pm -_a ((lcf\ R\ S\ X\ f) \cdot_{r S} ((lcf\ R\ S\ X\ g)^{-\ S}) \cdot_r$
$(X \hat{}^R ((deg\text{-}n\ R\ S\ X\ f) - (deg\text{-}n\ R\ S\ X\ g))) \cdot_r g)) <$ *deg-n R S X f*
apply (*cut-tac ring-is-ag*,
cut-tac subring,
frule Corps.field-is-idom[*of S*],
frule subring-Ring,
cut-tac subring,
cut-tac polyn-ring-X-nonzero,
cut-tac X-mem-R,
cut-tac polyn-ring-integral, *simp*)
apply (*frule npClose*[*of X deg-n R S X f* − *deg-n R S X g*],
frule-tac Idomain.idom-potent-nonzero[*of R X*
fst (*s-cf R S X f*) − *fst* (*s-cf R S X g*)], *assumption*+,
frule s-cf-expr[*of f*], *assumption*+, (*erule conjE*)+,

frule s-cf-expr[*of g*], *assumption*+, (*erule conjE*)+,
simp add:*s-cf-deg*, *simp add*:*lcf-val*[*THEN sym*],
frule Corps.invf-closed1[*of S lcf R S X g*], *simp*, *simp add*:*lcf-mem*,
frule lcf-mem[*of f*], *simp*,
frule subring-Ring, *frule Ring.ring-is-ag*[*of S*], (*erule conjE*)+,
frule-tac x = *lcf R S X f* **and** *y* = *lcf R S X g*$^{-}{}^{S}$ **in**
Ring.ring-tOp-closed[*of S*], *assumption*+,
frule mem-subring-mem-ring[*of S lcf R S X f* $\cdot_{r\,S}$ *lcf R S X g*$^{-}{}^{S}$],
assumption+,
frule-tac x = *lcf R S X f* **and** *y* = *lcf R S X g*$^{-}{}^{S}$ **in**
Idomain.idom-tOp-nonzeros[*of S*], *assumption*+,
simp add:*Subring-zero-ring-zero*)
apply(*frule-tac x* = *lcf R S X f* $\cdot_{r\,S}$ *lcf R S X g*$^{-}{}^{S}$ **and**
y = *X*$^\wedge$*R* (*fst* (*s-cf R S X f*) − *fst* (*s-cf R S X g*)) **in**
Idomain.idom-tOp-nonzeros[*of R*], *assumption*+,
frule-tac x = *lcf R S X f* $\cdot_{r\,S}$ *lcf R S X g*$^{-}{}^{S}$ **and**
y = *X*$^\wedge$*R* (*fst* (*s-cf R S X f*) − *fst* (*s-cf R S X g*)) **in** *ring-tOp-closed*,
assumption+,
frule-tac x = *lcf R S X f* $\cdot_{r\,S}$ *lcf R S X g*$^{-}{}^{S}$ $\cdot_r$
X$^\wedge$*R* (*fst* (*s-cf R S X f*) − *fst* (*s-cf R S X g*)) **and** *y* = *g* **in**
Idomain.idom-tOp-nonzeros[*of R*], *assumption*+,
frule-tac x = *lcf R S X f* $\cdot_{r\,S}$ *lcf R S X g*$^{-}{}^{S}$ $\cdot_r$
X$^\wedge$*R* (*fst* (*s-cf R S X f*) − *fst* (*s-cf R S X g*)) **and** *y* = *g* **in** *ring-tOp-closed*,
assumption+)

apply (*frule pol-diff-deg-less*[*of f s-cf R S X f*
s-cf R S X (*lcf R S X f* $\cdot_{r\,S}$ *lcf R S X g*$^{-}{}^{S}$ $\cdot_r$
X$^\wedge$*R* (*fst* (*s-cf R S X f*) − *fst* (*s-cf R S X g*)) $\cdot_r$ *g*)], *assumption*+,
simp add:*s-cf-pol-coeff*)
apply (*simp add*:*s-cf-deg*[*THEN sym*])
apply (*frule deg-mult-pols*[*of lcf R S X f* $\cdot_{r\,S}$ (*lcf R S X g*)$^{-}{}^{S}$
(*X*$^\wedge$*R* (*deg-n R S X f* − *deg-n R S X g*))$\cdot_r$ *g*], *assumption*+,
rule ring-tOp-closed, *assumption*+,
rule Idomain.idom-tOp-nonzeros[*of R*], *assumption*+,
(*erule conjE*)+,
subst ring-tOp-assoc[*of lcf R S X f* $\cdot_{r\,S}$ *lcf R S X g*$^{-}{}^{S}$ - *g*],
assumption+, *simp*,
cut-tac pol-of-deg0[*THEN sym*, *of lcf R S X f* $\cdot_{r\,S}$ *lcf R S X g*$^{-}{}^{S}$],
simp,
thin-tac deg-n R S X (*lcf R S X f* $\cdot_{r\,S}$ *lcf R S X g*$^{-}{}^{S}$ $\cdot_r$
(*X*$^\wedge$*R* (*deg-n R S X f* − *deg-n R S X g*) $\cdot_r$ *g*)) =
deg-n R S X (*X*$^\wedge$*R* (*deg-n R S X f* − *deg-n R S X g*) $\cdot_r$ *g*))
apply (*frule deg-mult-pols*[*of* (*X*$^\wedge$*R* (*deg-n R S X f* − *deg-n R S X g*)) *g*],
assumption+, *simp*,
simp add:*deg-to-X-d*, *assumption*+,
fold lcf-def,
subst divisionTr0[*of g lcf R S X f* $\cdot_{r\,S}$ *lcf R S X g*$^{-}{}^{S}$

```
                fst (s-cf R S X f) − fst (s-cf R S X g)],
        assumption+, simp add:Subring-zero-ring-zero)
  apply (subst Ring.ring-tOp-assoc, assumption+, simp add:lcf-mem,
        frule Corps.invf-inv[of S lcf R S X g], simp add:lcf-mem,
        simp add:Subring-zero-ring-zero, simp add:Ring.ring-r-one,
        frule s-cf-expr[of lcf R S X f ·_r_S lcf R S X g⁻ S ·_r
            X^R (fst (s-cf R S X f) − fst (s-cf R S X g)) ·_r g],
        rule Idomain.idom-tOp-nonzeros[of R], assumption+)
 apply ((erule conjE)+,
       thin-tac snd (s-cf R S X
         (lcf R S X f ·_r_S lcf R S X g⁻ S ·_r
          X^R (fst (s-cf R S X f) − fst (s-cf R S X g)) ·_r g))
         (fst (s-cf R S X (lcf R S X f ·_r_S lcf R S X g⁻ S ·_r
           X^R (fst (s-cf R S X f) − fst (s-cf R S X g)) ·_r g))) ≠ 0_S)
  apply (rotate-tac −1, drule sym, simp)
done

lemma (in PolynRg) divisionTr2:⟦Corps S; g ∈ carrier R; g ≠ 0;
                0 < deg-n R S X g⟧ ⟹ ∀f. div-condn R S X n g f
apply (cut-tac ring-is-ag,
      frule Corps.field-is-idom[of S],
      cut-tac subring, frule subring-Ring,
      cut-tac polyn-ring-integral, simp,
      cut-tac X-mem-R)

apply (rule nat-less-induct)
apply (rule allI)
apply (subst div-condn-def, rule impI, (erule conjE)+)
apply (case-tac f = 0_R,
      cut-tac ring-zero,
      subgoal-tac f ± −_a (0 ·_r g) = 0,
      blast,
      simp add:ring-times-0-x, simp add:aGroup.ag-inv-zero[of R],
      simp add:aGroup.ag-r-zero)
apply (case-tac n < deg-n R S X g)
 apply (cut-tac ring-zero,
       subgoal-tac deg-n R S X (f ± −_a (0 ·_r g)) < deg-n R S X g,
       blast) apply (
      simp add:ring-times-0-x, simp add:aGroup.ag-inv-zero,
      simp add:aGroup.ag-r-zero)
apply (frule-tac x = n and y = deg-n R S X g in leI,
      thin-tac ¬ n < deg-n R S X g)

apply (frule-tac f = f in divisionTr1[of g], assumption+, simp)
apply (frule-tac p = f in lcf-mem,
      frule lcf-mem[of g],
      frule lcf-nonzero[of g], assumption+,
      frule Corps.invf-closed1[of S lcf R S X g], simp)
```

apply (*frule-tac* $x = lcf\ R\ S\ X\ f$ **and** $y = lcf\ R\ S\ X\ g^{-\ S}$ **in**
Ring.ring-tOp-closed[*of S*], *assumption*+, *simp*)
apply (*frule-tac* $x = lcf\ R\ S\ X\ f\ \cdot_{r\ S}\ lcf\ R\ S\ X\ g^{-\ S}$ **in** *mem-subring-mem-ring*,
assumption)
apply (*frule-tac* $n = deg\text{-}n\ R\ S\ X\ f\ -\ deg\text{-}n\ R\ S\ X\ g$ **in** *npClose*[*of X*],
frule-tac $x = lcf\ R\ S\ X\ f\ \cdot_{r\ S}\ lcf\ R\ S\ X\ g^{-\ S}$ **and**
$y = X\hat{}R\ (deg\text{-}n\ R\ S\ X\ f\ -\ deg\text{-}n\ R\ S\ X\ g)$ **in**
ring-tOp-closed, *assumption*+,
frule-tac $x = lcf\ R\ S\ X\ f\ \cdot_{r\ S}\ lcf\ R\ S\ X\ g^{-\ S}\ \cdot_r$
$X\hat{}R\ (deg\text{-}n\ R\ S\ X\ f\ -\ deg\text{-}n\ R\ S\ X\ g)$ **and**
$y = g$ **in** *ring-tOp-closed*, *assumption*+)
apply (*erule disjE*, *blast*)
apply (*drule-tac* $a = deg\text{-}n\ R\ S\ X\ (f \pm -_a\ (lcf\ R\ S\ X\ f\ \cdot_{r\ S}$
$lcf\ R\ S\ X\ g^{-\ S}\ \cdot_r\ X\hat{}R\ (deg\text{-}n\ R\ S\ X\ f\ -\ deg\text{-}n\ R\ S\ X\ g)\ \cdot_r\ g))$ **in**
forall-spec, *simp*)
apply (*simp add*:*div-condn-def*)
apply (*drule-tac* $x = f \pm$
$-_a\ (lcf\ R\ S\ X\ f\ \cdot_{r\ S}\ lcf\ R\ S\ X\ g^{-\ S}\ \cdot_r$
$X\hat{}R\ (deg\text{-}n\ R\ S\ X\ f\ -\ deg\text{-}n\ R\ S\ X\ g)\ \cdot_r$
$g)$ **in** *spec*)
apply (*frule-tac* $x = lcf\ R\ S\ X\ f\ \cdot_{r\ S}\ lcf\ R\ S\ X\ g^{-\ S}\ \cdot_r$
$X\hat{}R\ (deg\text{-}n\ R\ S\ X\ f\ -\ deg\text{-}n\ R\ S\ X\ g)\ \cdot_r\ g$ **in** *aGroup.ag-mOp-closed*,
assumption)
apply (*frule-tac* $x = f$ **and** $y = -_a\ (lcf\ R\ S\ X\ f\ \cdot_{r\ S}\ lcf\ R\ S\ X\ g^{-\ S}\ \cdot_r$
$X\hat{}R\ (deg\text{-}n\ R\ S\ X\ f\ -\ deg\text{-}n\ R\ S\ X\ g)\ \cdot_r\ g)$ **in**
aGroup.ag-pOp-closed, *assumption*+, *simp*,
thin-tac $deg\text{-}n\ R\ S\ X\ (f \pm\ -_a\ (lcf\ R\ S\ X\ f\ \cdot_{r\ S}\ lcf\ R\ S\ X\ g^{-\ S}\ \cdot_r$
$X\hat{}R\ (deg\text{-}n\ R\ S\ X\ f\ -\ deg\text{-}n\ R\ S\ X\ g)\ \cdot_r\ g)) < deg\text{-}n\ R\ S\ X\ f)$
apply (*erule exE*,
thin-tac $f \pm -_a\ (lcf\ R\ S\ X\ f\ \cdot_{r\ S}\ lcf\ R\ S\ X\ g^{-\ S}\ \cdot_r$
$X\hat{}R\ (deg\text{-}n\ R\ S\ X\ f\ -\ deg\text{-}n\ R\ S\ X\ g)\ \cdot_r\ g) \in carrier\ R$)
apply ((*erule conjE*)+,
frule-tac $x = q$ **and** $y = g$ **in** *ring-tOp-closed*, *assumption*+,
frule-tac $x = q\ \cdot_r\ g$ **in** *aGroup.ag-mOp-closed*, *assumption*+,
simp add:*aGroup.ag-pOp-assoc*,
simp add:*aGroup.ag-p-inv*[*THEN sym*],
simp add:*ring-distrib2*[*THEN sym*])
apply (*frule-tac* $x = lcf\ R\ S\ X\ f\ \cdot_{r\ S}\ lcf\ R\ S\ X\ g^{-\ S}\ \cdot_r$
$X\hat{}R\ (deg\text{-}n\ R\ S\ X\ f\ -\ deg\text{-}n\ R\ S\ X\ g)$ **and** $y = q$ **in**
aGroup.ag-pOp-closed, *assumption*+)
apply (*erule disjE*)
apply *blast*
apply *blast*
done

lemma (**in** *PolynRg*) *divisionTr3*:$[\![Corps\ S;\ g \in carrier\ R;\ g \neq \mathbf{0};$
$0 < deg\text{-}n\ R\ S\ X\ g;\ f \in carrier\ R]\!] \Longrightarrow$

$\exists\, q{\in}carrier\ R.\ (f \pm -_a\ (q \cdot_r\ \ g) = \mathbf{0}) \vee (\ f \pm -_a\ (q \cdot_r\ g) \neq \mathbf{0} \wedge$
$deg\text{-}n\ R\ S\ X\ (f \pm -_a\ (q \cdot_r\ g)) < (deg\text{-}n\ R\ S\ X\ g))$
apply (*frule divisionTr2*[*of g deg-n R S X f*], *assumption*+)
apply (*drule-tac x* = *f* **in** *spec*)
apply (*simp add*:*div-condn-def*, *blast*)
done

lemma (**in** *PolynRg*) *divisionTr4*:⟦*Corps S*; *g* ∈ *carrier R*; *g* ≠ **0**;
0 < *deg-n R S X g*; *f* ∈ *carrier R*⟧ ⟹
$\exists\, q{\in}carrier\ R.\ (f = q \cdot_r\ g) \vee (\exists\, r{\in}carrier\ R.\ r \neq \mathbf{0} \wedge (f = (q \cdot_r\ g) \pm r)$
$\wedge\ (deg\text{-}n\ R\ S\ X\ r) < (deg\text{-}n\ R\ S\ X\ g))$
apply (*cut-tac is-Ring*,
cut-tac ring-is-ag)
apply (*frule divisionTr3*[*of g f*], *assumption*+,
erule bexE,
frule-tac x = *q* **in** *ring-tOp-closed*[*of - g*], *assumption*+,
erule disjE)
apply (*simp add*:*aGroup.ag-eq-diffzero*[*THEN sym*, *of R f*], *blast*)
apply (*subgoal-tac f* = *q* ·$_r$ *g* ± (*f* ± −$_a$ (*q* ·$_r$ *g*)),
subgoal-tac (*f* ± −$_a$ (*q* ·$_r$ *g*)) ∈ *carrier R*, *blast*,
rule aGroup.ag-pOp-closed, *assumption*+,
rule aGroup.ag-mOp-closed, *assumption*+)
apply (*frule-tac x* = *q* ·$_r$ *g* **in** *aGroup.ag-mOp-closed*, *assumption*+,
subst aGroup.ag-pOp-assoc[*THEN sym*], *assumption*+,
subst aGroup.ag-pOp-commute[*of R - f*], *assumption*+,
subst aGroup.ag-pOp-assoc, *assumption*+,
simp add:*aGroup.ag-r-inv1*, *simp add*:*aGroup.ag-r-zero*)
done

lemma (**in** *PolynRg*) *divisionTr*:⟦*Corps S*; *g* ∈ *carrier R*; *0* < *deg R S X g*;
f ∈ *carrier R*⟧ ⟹
$\exists\, q{\in}carrier\ R.\ (\exists\, r{\in}carrier\ R.\ (f = (q \cdot_r\ g) \pm r) \wedge$
$(deg\ R\ S\ X\ r) < (deg\ R\ S\ X\ g))$
apply (*subgoal-tac g* ≠ **0**,
frule divisionTr4[*of g f*], *assumption*+,
simp add:*deg-def*, *simp only*:*an-0*[*THEN sym*],
cut-tac aless-nat-less[*of 0 deg-n R S X g*], *simp*, *assumption*)
apply (*erule bexE*, *erule disjE*,
cut-tac ring-is-ag, *frule aGroup.ag-r-zero*[*of R f*], *simp*, *simp*,
rotate-tac −*1*, *frule sym*,
cut-tac ring-zero,
subgoal-tac deg R S X **0** < *deg R S X g*, *blast*,
simp add:*deg-def an-def*)
apply (*erule bexE*, (*erule conjE*)+,
cut-tac n1 = *deg-n R S X r* **and** *m1* = *deg-n R S X g* **in**
aless-natless[*THEN sym*], *simp add*:*deg-def*,
drule sym, *simp*, *rotate-tac* −*1*, *drule sym*, *blast*)
apply (*rule contrapos-pp*, *simp*+,
simp add:*deg-def*, *frule aless-imp-le*[*of 0* −∞],

cut-tac minf-le-any[*of 0*])
apply (*frule ale-antisym*[*of 0* $-\infty$], *assumption*)
apply *simp*
done

lemma (**in** *PolynRg*) *rel-prime-equation*:⟦*Corps S*; $f \in$ *carrier R*; $g \in$ *carrier R*;
$0 <$ *deg R S X f*; $0 <$ *deg R S X g*; *rel-prime-pols R S X f g*;
$h \in$ *carrier R*⟧ $\Longrightarrow$
$\exists u \in$ *carrier R*. $\exists v \in$ *carrier R*.
(*deg R S X u* $\leq$ *amax* ((*deg R S X h*) $-$ (*deg R S X f*)) (*deg R S X g*)) $\wedge$
(*deg R S X v* $\leq$ (*deg R S X f*)) $\wedge$ ($u \cdot_r f \pm (v \cdot_r g) = h$)
apply (*cut-tac ring-is-ag*,
cut-tac ring-zero, *cut-tac subring*, *frule subring-Ring*,
frule aless-imp-le [*of 0 deg R S X f*],
frule pol-nonzero[*of f*], *simp*,
frule aless-imp-le [*of 0 deg R S X g*],
frule pol-nonzero[*of g*], *simp*,
frule Corps.field-is-idom[*of S*],
cut-tac polyn-ring-integral, *simp*,
frule Idomain.idom-tOp-nonzeros[*of R f g*], *assumption*+)
apply (*case-tac* $h = \mathbf{0}_R$)
apply (*cut-tac ring-is-ag*,
cut-tac ring-zero,
subgoal-tac deg R S X $\mathbf{0} \leq$
amax (*deg R S X h* $-$ *deg R S X f*) (*deg R S X g*),
subgoal-tac deg R S X $\mathbf{0} \leq$ *deg R S X f* $\wedge$
$\mathbf{0} \cdot_r f \pm \mathbf{0} \cdot_r g = h$, *blast*)
apply (*simp add*:*ring-times-0-x*, *simp add*:*aGroup.ag-r-zero*,
simp add:*deg-def*)
apply (*simp add*:*deg-def amax-def*)

apply (*simp add*:*rel-prime-pols-def*,
frule principal-ideal[*of f*], *frule principal-ideal*[*of g*],
frule ideals-set-sum[*of R* $\diamondsuit_p$ *f R* $\diamondsuit_p$ *g* 1_r], *assumption*+,
thin-tac $1_r \in R \diamondsuit_p f \mp R \diamondsuit_p g$,
(*erule bexE*)+,
thin-tac ideal R (*R* $\diamondsuit_p$ *f*), *thin-tac ideal R* (*R* $\diamondsuit_p$ *g*),
simp add:*Rxa-def*, (*erule bexE*)+, *simp*,
thin-tac ha $= r \cdot_r f$, *thin-tac k* $= ra \cdot_r g$)
apply (*frule-tac* $x = r$ **in** *ring-tOp-closed*[*of - f*], *assumption*+,
frule-tac $x = ra$ **in** *ring-tOp-closed*[*of - g*], *assumption*+,
frule-tac $y1 = r \cdot_r f$ **and** $z1 = ra \cdot_r g$ **in** *ring-distrib1*[*THEN sym*,
of h], *assumption*+, *simp add*:*ring-r-one*, *drule sym*,
simp,
thin-tac $r \cdot_r f \pm ra \cdot_r g = 1_r$,
simp add:*ring-tOp-assoc*[*THEN sym*], *simp add*:*ring-r-one*)
apply (*frule-tac* $f = h \cdot_r r$ **in** *divisionTr*[*of g*], *assumption*+,
simp add:*ring-tOp-closed*,
frule-tac $f = h \cdot_r ra$ **in** *divisionTr*[*of f*], *assumption*+,

simp add:*ring-tOp-closed*,
(*erule bexE*)+, (*erule conjE*)+)

apply (*thin-tac* $r \in carrier\ R$,
thin-tac $ra \in carrier\ R$,
thin-tac $r \cdot_r f \in carrier\ R$,
thin-tac $ra \cdot_r g \in carrier\ R$)
apply (*frule-tac* $x = q$ **in** *ring-tOp-closed*[*of* - g], *assumption*+,
frule-tac $x = qa$ **in** *ring-tOp-closed*[*of* - f], *assumption*+,
frule-tac $x = q \cdot_r g$ **and** $y = rb$ **in** *aGroup.ag-pOp-commute*, *assumption*+,
simp,
thin-tac $q \cdot_r g \pm rb = rb \pm q \cdot_r g$,
thin-tac $h \cdot_r r = rb \pm q \cdot_r g$,
thin-tac $h \cdot_r ra = qa \cdot_r f \pm rc$)
apply (*simp add*:*ring-distrib2*[*of* g],
frule-tac $x = rb$ **and** $y = q \cdot_r g$ **in** *aGroup.ag-pOp-closed*[*of* R],
assumption+,
frule-tac $x = rb \pm q \cdot_r g$ **and** $y = f$ **in** *ring-tOp-closed*,
assumption+,
frule-tac $x = qa \cdot_r f$ **and** $y = g$ **in** *ring-tOp-closed*, *assumption*+,
frule-tac $x = rc$ **and** $y = g$ **in** *ring-tOp-closed*, *assumption*+,
simp add:*aGroup.ag-pOp-assoc*[*THEN sym*, *of* R],
simp add:*ring-tOp-assoc*[*of* - f g],
simp add:*ring-tOp-commute*[*of* f g],
simp add:*ring-tOp-assoc*[*THEN sym*, *of* - g f],
frule-tac $x = qa$ **and** $y = g$ **in** *ring-tOp-closed*, *assumption*+,
simp add:*ring-distrib2*[*THEN sym*],
simp add:*aGroup.ag-pOp-assoc*,
simp add:*ring-distrib2*[*THEN sym*],
case-tac $q \pm_R qa = \mathbf{0}_R$, *simp add*:*ring-times-0-x*,
simp add:*aGroup.ag-r-zero*,
subgoal-tac deg R S X rb $\leq$
amax (*deg R S X h* $-$ *deg R S X f*) (*deg R S X g*),
subgoal-tac deg R S X rc $\leq$ *deg R S X f*,
blast,
simp add:*aless-imp-le*,
frule-tac $x =$ *deg R S X rb* **and** $y =$ *deg R S X g* **in**
aless-imp-le,
rule-tac $i =$ *deg R S X rb* **in** *ale-trans*[*of* - *deg R S X g*
amax (*deg R S X h* $-$ *deg R S X f*) (*deg R S X g*)], *assumption*,
simp add:*amax-def*, *simp add*:*aneg-le aless-imp-le*)
apply (*subgoal-tac* $rb \pm (q \pm qa) \cdot_r g \in carrier\ R$,
subgoal-tac deg R S X $(rb \pm (q \pm qa) \cdot_r g) \leq$
amax (*deg R S X h* $-$ *deg R S X f*) (*deg R S X g*),
subgoal-tac deg R S X rc $\leq$ *deg R S X f*, *blast*,
simp add:*aless-imp-le*,
frule-tac $x = q$ **and** $y = qa$ **in** *aGroup.ag-pOp-closed*[*of* R], *assumption*+,
frule-tac $p = rb$ **and** $q = (q \pm qa) \cdot_r g$ **in** *deg-pols-add1*,
rule ring-tOp-closed, *assumption*+, *simp add*:*deg-mult-pols1*,

frule-tac p1 = q ± qa **in** *pol-nonzero*[*THEN sym*], *simp*)

apply (*frule-tac y = deg R S X (q ± qa)* **and** *z = deg R S X rb* **in**
aadd-le-mono[*of 0*], *simp add*:*aadd-0-l*)
apply (*frule-tac p = q ± qa* **in** *deg-ant-int*, *assumption*+,
frule-tac x = deg R S X rb **and** *y = deg R S X g* **and**
z = int (*deg-n R S X* (*q ± qa*)) **in** *aadd-less-mono-z*,
simp add:*aadd-commute*)

apply (*simp add*:*deg-mult-pols1*,
frule-tac p = rb ± (q ± qa) ·r g **and** *q = f* **in**
deg-mult-pols1, *assumption*+, *simp*,
thin-tac deg R S X (rb ± (q ± qa) ·r g) =
deg R S X (q ± qa) + deg R S X g,
frule-tac x = rb ± (q ± qa) ·r g **and** *y = f* **in**
ring-tOp-closed, *assumption*+, *simp only*:*aGroup.ag-pOp-commute*,
frule-tac p = rc ·r g **and** *q = (rb ± (q ± qa) ·r g) ·r f* **in**
deg-pols-add1, *assumption*+, *simp*,
thin-tac deg R S X ((rb ± (q ± qa) ·r g) ·r f) =
deg R S X (q ± qa) + deg R S X g + deg R S X f)
apply (*simp add*:*deg-mult-pols1*,
frule-tac p1 = q ± qa **in** *pol-nonzero*[*THEN sym*], *simp*,
simp add:*deg-ant-int*[*of g*])
apply (*frule-tac x = deg R S X rc* **and** *y = deg R S X f* **and**
z = int (*deg-n R S X g*) **in** *aadd-less-mono-z*,
frule-tac a = deg R S X (*q ± qa*) **in** *aadd-pos-le*[*of* -
deg R S X f + ant (*int* (*deg-n R S X g*))],
frule-tac x = deg R S X rc + ant (*int* (*deg-n R S X g*)) **and**
y = deg R S X f + ant (*int* (*deg-n R S X g*)) **and**
z = deg R S X (*q ± qa*) + (*deg R S X f + ant* (*int* (*deg-n R S X g*)))
in *aless-le-trans*, *assumption*+,
thin-tac deg R S X rc + ant (*int* (*deg-n R S X g*))
< deg R S X f + ant (*int* (*deg-n R S X g*)),
thin-tac deg R S X f + ant (*int* (*deg-n R S X g*))
≤ deg R S X (*q ± qa*) + (*deg R S X f + ant* (*int* (*deg-n R S X g*))))
apply (*simp add*:*deg-ant-int*[*THEN sym*])
apply (*frule-tac p = q ± qa* **in** *deg-in-aug-minf*,
frule-tac p = g **in** *deg-in-aug-minf*,
frule-tac p = f **in** *deg-in-aug-minf*,
simp add:*aadd-commute*[*of deg R S X f deg R S X g*],
simp only:*aadd-assoc-m*[*THEN sym*], *simp*)
apply (*frule-tac p = g* **in** *deg-in-aug-minf*,
frule-tac p = f **in** *deg-in-aug-minf*,
frule-tac p = q ± qa **in** *deg-in-aug-minf*,
simp add:*diff-ant-def*,
subgoal-tac −(*deg R S X f*) ∈ $Z_{-\infty}$)
apply (*subst aadd-assoc-m*[*of* - *deg R S X f − deg R S X f*],
simp add:*Zminf-pOp-closed*, *assumption*+,
(*simp add*:*aadd-minus-r*, *simp add*:*aadd-0-r*), *simp add*:*amax-ge-l*,

simp add:*deg-ant-int*, *simp add*:*aminus*, *simp add*:*z-in-aug-minf*)
apply (*rule aGroup.ag-pOp-closed*, *assumption*+,
rule ring-tOp-closed,
rule aGroup.ag-pOp-closed, *assumption*+)
done

4.16.1 Polynomial, coeff mod P

definition
P-mod :: [('*a*, '*m*) *Ring-scheme*, ('*a*, '*m1*) *Ring-scheme*, '*a*, '*a set*, '*a*] ⇒ *bool* **where**
P-mod R S X P p ⟷ *p* = $\mathbf{0}_R$ ∨
(∀ *j* ≤ (*fst* (*s-cf R S X p*)). (*snd* (*s-cf R S X p*) *j*) ∈ *P*)

lemma (**in** *PolynRg*) *P-mod-whole*:*p* ∈ *carrier R* ⟹
P-mod R S X (*carrier S*) *p*
apply (*case-tac p* = $\mathbf{0}_R$, *simp add*:*P-mod-def*)
apply (*simp add*:*P-mod-def*,
rule allI, *rule impI*,
rule pol-coeff-mem,
simp add:*s-cf-pol-coeff*,
assumption)
done

lemma (**in** *PolynRg*) *zero-P-mod*:*ideal S I* ⟹ *P-mod R S X I* **0**
by (*simp add*:*P-mod-def*)

lemma (**in** *PolynRg*) *P-mod-mod*:⟦*ideal S I*; *p* ∈ *carrier R*; *pol-coeff S c*;
p = *polyn-expr R X* (*fst c*) *c*⟧ ⟹
(∀ *j* ≤ (*fst c*). (*snd c*) *j* ∈ *I*) = (*P-mod R S X I p*)
apply (*cut-tac subring*, *frule subring-Ring*)
apply (*case-tac p* = $\mathbf{0}_R$)
apply (*simp add*:*P-mod-def*,
drule sym,
frule coeff-0-pol-0[*THEN sym*, *of c fst c*], *simp*, *simp*)
apply (*rule impI*,
simp add:*Ring.ideal-zero*)

apply (*frule s-cf-expr*[*of p*],
simp add:*P-mod-def*, (*erule conjE*)+)
apply (*frule polyn-c-max*[*of c*])
apply (*frule coeff-nonzero-polyn-nonzero*[*of c fst c*], *simp*)
apply (*frule coeff-max-nonzeroTr*[*of c*], *simp*)
apply (*thin-tac* (*polyn-expr R X* (*fst c*) *c* ≠ **0**) = (∃ *j*≤*fst c*. *snd c j* ≠ $\mathbf{0}_S$))
apply (*frule coeff-max-bddTr*[*of c*])
apply (*frule polyn-expr-short*[*of c c-max S c*], *assumption*+)
apply (*frule pol-expr-unique*[*of p* (*c-max S c*, *snd c*) *s-cf R S X p*],
assumption+, *rule split-pol-coeff*[*of c*], *assumption*+,
simp, *simp*, *assumption*+)

apply (*thin-tac* $p = $ *polyn-expr* $R\ X$ (*fst* c) c,
  thin-tac $p = $ *polyn-expr* $R\ X$ (*fst* (*s-cf* $R\ S\ X\ p$)) (*s-cf* $R\ S\ X\ p$),
  thin-tac polyn-expr $R\ X$ (*fst* c) $c =$ *polyn-expr* $R\ X$ (*c-max* $S\ c$) c,
  thin-tac polyn-expr $R\ X$ (*c-max* $S\ c$) $c =$
    polyn-expr $R\ X$ (*c-max* $S\ c$) (*c-max* $S\ c$, *snd* c))
apply (*frule coeff-max-zeroTr*[*of* c], (*erule conjE*)+)
apply (*subst P-mod-def*, *simp*)
apply (*rule iffI*, *rule allI*, *rule impI*)
apply (*rotate-tac 10*,
  drule-tac $a = j$ **in** *forall-spec*,
  drule-tac $x = j$ **in** *spec*, *assumption*,
  frule-tac $i = j$ **and** $j =$ *fst* (*s-cf* $R\ S\ X\ p$) **and** $k =$ *fst* c **in**
    le-trans, *assumption*+,
  drule-tac $a = j$ **in** *forall-spec*, *assumption*, *simp*)

apply (*rule allI*, *rule impI*)
apply (*case-tac fst* (*s-cf* $R\ S\ X\ p$) $< j$,
  drule-tac $a = j$ **in** *forall-spec*, *simp*,
  simp add:*Ring.ideal-zero*,
  frule-tac $x =$ *fst* (*s-cf* $R\ S\ X\ p$) **and** $y = j$ **in** *leI*,
  thin-tac $\forall j.\ j \leq$ *fst* $c \wedge$ *fst* (*s-cf* $R\ S\ X\ p$) $< j \longrightarrow$ *snd* $c\ j = \mathbf{0}_S$,
  drule-tac $a = j$ **in** *forall-spec*, *assumption*,
  drule-tac $a = j$ **in** *forall-spec*, *assumption*, *simp*)
done

lemma (**in** *PolynRg*) *monomial-P-mod-mod*:⟦*ideal* $S\ I$; $c \in$ *carrier* S;
  $p = c \cdot_r (X \hat{}^R\ d)$⟧ $\Longrightarrow$ $(c \in I) =$ (*P-mod* $R\ S\ X\ I\ p$)
apply (*cut-tac subring*, *frule subring-Ring*)
apply (*cut-tac monomial-d*[*THEN sym*, *of* (*0*, $\lambda j.\ c$) d], *simp*)
apply (*drule sym*, *simp*)
apply (*subst P-mod-mod*[*THEN sym*, *of I p ext-cf S d* (*0*, $\lambda j.\ c$)],
  assumption+)
  apply (*frule mem-subring-mem-ring*[*of S c*], *assumption*,
    cut-tac X-mem-R,
    frule npClose[*of X d*], *drule sym*, *simp add*:*ring-tOp-closed*)
  apply (*simp add*:*pol-coeff-def*, *rule allI*, *rule impI*,
    simp add:*ext-cf-def sliden-def*, *rule impI*, *simp add*:*Ring.ring-zero*,
    subst ext-cf-len, *simp add*:*pol-coeff-def*,
    simp)
  apply (*subst ext-cf-len*, *simp add*:*pol-coeff-def*,
    simp add:*ext-cf-def*)
apply (*rule iffI*)
  apply (*simp add*:*Ring.ideal-zero*,
    drule-tac $x = d$ **in** *spec*,
    simp, *simp add*:*pol-coeff-def*)
done

lemma (**in** *PolynRg*) *P-mod-add*:⟦*ideal* $S\ I$; $p \in$ *carrier* R;

$q \in$ *carrier* R; *P-mod* $R\ S\ X\ I\ p$; *P-mod* $R\ S\ X\ I\ q]\!] \Longrightarrow$
P-mod $R\ S\ X\ I\ (p \pm q)$
apply (*cut-tac subring*,
frule subring-Ring,
cut-tac ring-is-ag)

apply (*case-tac* $p = \mathbf{0}_R$, *simp add*:*aGroup.ag-l-zero*,
case-tac $q = \mathbf{0}_R$, *simp add*:*aGroup.ag-r-zero*)
apply (*case-tac* $p \pm_R q = \mathbf{0}_R$, *simp add*:*P-mod-def*)

apply (*frule s-cf-expr*[*of p*], *assumption*,
frule s-cf-expr[*of q*], *assumption*, (*erule conjE*)+)
apply (*frule polyn-add1*[*of s-cf R S X p s-cf R S X q*], *assumption*+,
drule sym, *drule sym*, *simp*, *drule sym*, *simp*,
rotate-tac −*1*, *drule sym*)
apply (*frule P-mod-mod*[*THEN sym*, *of I p s-cf R S X p*], *assumption*+, *simp*,
thin-tac polyn-expr R X (*fst* (*s-cf R S X p*)) (*s-cf R S X p*) = *p*,
frule P-mod-mod[*THEN sym*, *of I q s-cf R S X q*], *assumption*+, *simp*,
thin-tac polyn-expr R X (*fst* (*s-cf R S X q*)) (*s-cf R S X q*) = *q*)
apply (*frule aGroup.ag-pOp-closed*[*of R p q*], *assumption*+)
apply (*subst P-mod-mod*[*THEN sym*, *of I p* ± *q*
add-cf S (*s-cf R S X p*) (*s-cf R S X q*)], *assumption*+,
simp add:*add-cf-pol-coeff*, *simp*, *simp add*:*add-cf-len*,
thin-tac p ± *q* =
polyn-expr R X (*max* (*fst* (*s-cf R S X p*)) (*fst* (*s-cf R S X q*)))
(*add-cf S* (*s-cf R S X p*) (*s-cf R S X q*)))
apply *simp*
apply (*subst add-cf-len*, *assumption*+)
apply (*rule allI*, *rule impI*)

apply (*cut-tac x* = *fst* (*s-cf R S X p*) **and** *y* = *fst* (*s-cf R S X q*) **in**
less-linear)
apply (*erule disjE*)
apply (*simp*, *subst add-cf-def*, *simp*,
(*rule impI*,
drule-tac a = *j* **in** *forall-spec*, *assumption*,
drule-tac x = *j* **in** *spec*,
frule-tac x = *j* **and** *y* = *fst* (*s-cf R S X p*) **and**
z = *fst* (*s-cf R S X q*) **in** *le-less-trans*, *assumption*+,
frule-tac x = *j* **and** *y* = *fst* (*s-cf R S X q*) **in** *less-imp-le*, *simp*))
apply (*rule Ring.ideal-pOp-closed*[*of S I*], *assumption*+)
apply (*erule disjE*)
apply (*simp*, *subst add-cf-def*, *simp*,
drule-tac a = *j* **in** *forall-spec*, *assumption*,
drule-tac a = *j* **in** *forall-spec*, *assumption*)
apply (*rule Ring.ideal-pOp-closed*[*of S I*], *assumption*+)

apply (*simp add*: *max.absorb1 max.absorb2*, *subst add-cf-def*, *simp*, *rule impI*,
drule-tac x = *j* **in** *spec*,

drule-tac $a = j$ **in** *forall-spec*, *assumption*,
frule-tac $x = j$ **and** $y = fst\ (s\text{-}cf\ R\ S\ X\ q)$ **and**
$z = fst\ (s\text{-}cf\ R\ S\ X\ p)$ **in** *le-less-trans*, *assumption*+,
frule-tac $x = j$ **and** $y = fst\ (s\text{-}cf\ R\ S\ X\ p)$ **in** *less-imp-le*, *simp*)
apply (*rule Ring.ideal-pOp-closed*[*of S I*], *assumption*+)
done

lemma (**in** *PolynRg*) *P-mod-minus*:⟦*ideal S I*; $p \in$ *carrier R*; *P-mod R S X I p*⟧
⟹
P-mod R S X I $(-_a\ p)$
apply (*cut-tac ring-is-ag*,
cut-tac subring,
frule subring-Ring)
apply (*case-tac* $p = \mathbf{0}_R$, *simp add*:*aGroup.ag-inv-zero*)

apply (*frule s-cf-expr*[*of p*], *assumption*+, (*erule conjE*)+,
frule polyn-minus-m-cf[*of s-cf R S X p fst* (*s-cf R S X p*)],
simp,
frule aGroup.ag-inv-inj[*of R p* **0**], *assumption*,
simp add:*ring-zero*, *assumption*, *simp add*:*aGroup.ag-inv-zero*,
frule m-cf-pol-coeff[*of s-cf R S X p*],
drule sym, *drule sym*, *simp*)
apply (*subst P-mod-mod*[*THEN sym*, *of I* $-_a\ p$ *m-cf S* (*s-cf R S X p*)],
assumption+,
rule aGroup.ag-mOp-closed[*of R p*], *assumption*+,
simp add:*m-cf-len*,
thin-tac polyn-expr R X (*fst* (*s-cf R S X p*))
(*m-cf S* (*s-cf R S X p*)) $= -_a\ p$)
apply (*frule P-mod-mod*[*THEN sym*, *of I p s-cf R S X p*], *assumption*+, *simp*,
thin-tac polyn-expr R X (*fst* (*s-cf R S X p*)) (*s-cf R S X p*) $= p$,
simp)
apply (*rule allI*, *rule impI*,
drule-tac $a = j$ **in** *forall-spec*, *simp add*:*m-cf-len*,
subst m-cf-def, *simp*,
rule Ring.ideal-inv1-closed[*of S I*], *assumption*+)
done

lemma (**in** *PolynRg*) *P-mod-pre*:⟦*ideal S I*; *pol-coeff S* ((*Suc n*), *f*);
P-mod R S X I (*polyn-expr R X* (*Suc n*) (*Suc n*, *f*))⟧ ⟹
P-mod R S X I (*polyn-expr R X n* (*n*, *f*))
apply (*frule pol-coeff-pre*[*of n f*],
frule polyn-mem[*of* (*n*, *f*) *n*],*simp*,
frule polyn-mem[*of* (*Suc n*, *f*) *Suc n*], *simp*)
apply (*case-tac polyn-expr R X n* (*n*, *f*) $= \mathbf{0}_R$, *simp add*:*P-mod-def*)
apply (*subst P-mod-mod*[*THEN sym*, *of I*
polyn-expr R X n (*n*, *f*) (*n*, *f*)], *assumption*+, *simp*,
frule P-mod-mod[*THEN sym*, *of I polyn-expr R X* (*Suc n*) (*Suc n*, *f*)
(*Suc n*, *f*)], *assumption*+, *simp*, *simp*)
done

lemma (**in** *PolynRg*) *P-mod-pre1*:⟦*ideal S I*; *pol-coeff S* ((*Suc n*), *f*);
P-mod R S X I (*polyn-expr R X* (*Suc n*) (*Suc n*, *f*))⟧ ⟹
P-mod R S X I (*polyn-expr R X n* (*Suc n*, *f*))
by (*simp add*:*polyn-expr-restrict*[*of n f*], *simp add*:*P-mod-pre*)

lemma (**in** *PolynRg*) *P-mod-coeffTr*:⟦*ideal S I*; *d* ∈ *carrier S*⟧ ⟹
(*P-mod R S X I d*) = (*d* ∈ *I*)
apply (*cut-tac subring*, *frule subring-Ring*,
subst monomial-P-mod-mod[*of I d d* ·$_r$ *X*^$_R$ *0 0*], *assumption*+,
simp, *simp*,
frule mem-subring-mem-ring[*of - d*], *assumption*+,
simp add:*ring-r-one*)
done

lemma (**in** *PolynRg*) *P-mod-mult-const*:⟦*ideal S I*; *ideal S J*;
pol-coeff S (*n*, *f*); *P-mod R S X I* (*polyn-expr R X n* (*n*, *f*));
pol-coeff S (*0*, *g*); *P-mod R S X J* (*polyn-expr R X 0* (*0*, *g*))⟧ ⟹
P-mod R S X (*I* ◇$_{r\,S}$ *J*) ((*polyn-expr R X n* (*n*, *f*)) ·$_r$
(*polyn-expr R X 0* (*0*, *g*)))
apply (*cut-tac subring*, *frule subring-Ring*)
apply (*frule-tac c* = (*n*, *f*) **in** *polyn-mem*[*of - n*], *simp*)
apply (*frule Ring.ideal-prod-ideal*[*of S I J*], *assumption*+)
apply (*case-tac polyn-expr R X n* (*n*, *f*) = **0**$_R$, *simp*)
apply (*frule-tac c* = (*0*, *g*) **in** *polyn-mem*[*of - 0*], *simp*,
simp add:*ring-times-0-x*, *simp add*:*P-mod-def*)
apply (*simp add*:*polyn-expr-def* [*of - - 0*])
apply (*frule pol-coeff-mem*[*of* (*0*, *g*) *0*], *simp*, *simp*,
frule mem-subring-mem-ring[*of S g 0*], *assumption*,
simp add:*ring-r-one*,
simp add:*ring-tOp-commute*[*of - g 0*])
apply (*frule sp-cf-pol-coeff*[*of* (*n*, *f*) *g 0*], *assumption*+)
apply (*subst scalar-times-pol-expr*[*of g 0* (*n*, *f*) *n*], *assumption*+,
simp)
apply (*subst P-mod-mod*[*THEN sym*, *of I* ◇$_{r\,S}$ *J*
polyn-expr R X n (*sp-cf S* (*g 0*) (*n*, *f*)) *sp-cf S* (*g 0*) (*n*, *f*)],
assumption+,
simp add:*polyn-mem*, *simp add*:*sp-cf-pol-coeff*,
rule polyn-mem, *simp add*:*sp-cf-pol-coeff*,
simp add:*sp-cf-len*, *simp*,
simp add:*sp-cf-len*)
apply (*frule P-mod-mod*[*THEN sym*, *of I polyn-expr R X n* (*n*, *f*)
(*n*, *f*)], *assumption*+, *simp*, *simp*,
simp add:*sp-cf-len*, *subst sp-cf-def*, *simp*,
simp add:*P-mod-coeffTr*[*of J g 0*])
apply (*rule allI*, *rule impI*,
drule-tac a = *j* **in** *forall-spec*, *assumption*,
frule-tac h = *f j* **in** *Ring.ideal-subset*[*of S I*], *assumption*+,
simp add:*Ring.ring-tOp-commute*[*of S g 0*])

apply (*simp add*:*Ring.prod-mem-prod-ideals*[*of S I J*])
done

lemma (**in** *PolynRg*) *P-mod-mult-const1*:⟦*ideal S I*; *ideal S J*;
pol-coeff S (*n*, *f*); *P-mod R S X I* (*polyn-expr R X n* (*n*, *f*));
d ∈ *J*⟧ ⟹
P-mod R S X (*I* ◇$_{r\,S}$ *J*) ((*polyn-expr R X n* (*n*, *f*)) ·$_r$ *d*)
apply (*cut-tac subring*, *frule subring-Ring*)
apply (*frule P-mod-coeffTr*[*THEN sym*, *of J d*],
simp add:*Ring.ideal-subset*, *simp*)
apply (*frule P-mod-mult-const*[*of I J n f* λ*j*. *d*], *assumption*+,
simp add:*pol-coeff-def*, *simp add*:*Ring.ideal-subset*)
apply (*subst polyn-expr-def*, *simp*,
frule Ring.ideal-subset[*of S J d*], *assumption*+,
frule mem-subring-mem-ring[*of S d*], *assumption*,
simp add:*ring-r-one*)
apply (*simp add*:*polyn-expr-def*[*of - - 0*],
frule Ring.ideal-subset[*of S J d*], *assumption*+,
frule mem-subring-mem-ring[*of S d*], *assumption*,
simp add:*ring-r-one*)
done

lemma (**in** *PolynRg*) *P-mod-mult-monomial*:⟦*ideal S I*; *p* ∈ *carrier R*⟧ ⟹
(*P-mod R S X I p*) = (*P-mod R S X I* (*p* ·$_r$ *X*^^{R} *m*))
apply (*cut-tac X-mem-R*,
cut-tac subring, *frule subring-Ring*)
apply (*frule npClose*[*of X m*],
simp add:*ring-tOp-commute*[*of p*])
apply (*case-tac p* = **0**$_R$, *simp add*:*ring-times-x-0*)
apply (*rule iffI*)
apply (*frule s-cf-expr*[*of p*], *assumption*+, (*erule conjE*)+,
cut-tac low-deg-terms-zero[*THEN sym*, *of fst* (*s-cf R S X p*)
snd (*s-cf R S X p*) *m*],
simp add:*polyn-expr-split*[*THEN sym*],
thin-tac X^^{R} *m* ·$_r$ *p* =
polyn-expr R X (*fst* (*s-cf R S X p*) + *m*) (*ext-cf S m* (*s-cf R S X p*)),
frule ext-cf-pol-coeff[*of s-cf R S X p m*])
apply (*frule P-mod-mod*[*THEN sym*, *of I p s-cf R S X p*], *assumption*+,
thin-tac p = *polyn-expr R X* (*fst* (*s-cf R S X p*)) (*s-cf R S X p*),
simp)
apply (*frule P-mod-mod*[*THEN sym*, *of I polyn-expr R X* (*fst* (*s-cf R S X p*) +
m) (*ext-cf S m* (*s-cf R S X p*)) *ext-cf S m* (*s-cf R S X p*)],
rule polyn-mem, *assumption*, *simp add*:*ext-cf-def*,
assumption, *simp add*:*ext-cf-len add.commute*, *simp*,
thin-tac P-mod R S X I (*polyn-expr R X* (*fst* (*s-cf R S X p*) + *m*)
(*ext-cf S m* (*s-cf R S X p*))) = (∀ *j*≤*fst* (*ext-cf S m* (*s-cf R S X p*)).
snd (*ext-cf S m* (*s-cf R S X p*)) *j* ∈ *I*),
thin-tac snd (*s-cf R S X p*) (*fst* (*s-cf R S X p*)) ≠ **0**$_S$)
apply (*rule allI*, *rule impI*, *simp add*:*ext-cf-len*) **apply** (

subst ext-cf-def, *simp add*:*sliden-def*) **apply** (*rule impI*,
simp add:*Ring.ideal-zero*[*of S*])
apply (*simp add*:*pol-coeff-split*[*THEN sym*])

apply (*frule s-cf-expr*[*of p*], *assumption*+, (*erule conjE*)+,
cut-tac low-deg-terms-zero[*THEN sym*, *of fst* (*s-cf R S X p*)
snd (*s-cf R S X p*) *m*],
simp add:*polyn-expr-split*[*THEN sym*],
thin-tac $X \hat{}_R\ m \cdot_r p =$
polyn-expr R X (*fst* (*s-cf R S X p*) + *m*) (*ext-cf S m* (*s-cf R S X p*)),
frule ext-cf-pol-coeff[*of s-cf R S X p m*])
apply (*frule P-mod-mod*[*THEN sym*, *of I polyn-expr R X* (*fst* (*s-cf R S X p*) +
m) (*ext-cf S m* (*s-cf R S X p*)) *ext-cf S m* (*s-cf R S X p*)],
rule polyn-mem, *assumption*, *simp add*:*ext-cf-def*,
assumption, *simp add*:*ext-cf-len add.commute*, *simp*,
thin-tac p = *polyn-expr R X* (*fst* (*s-cf R S X p*)) (*s-cf R S X p*),
thin-tac P-mod R S X I
(*polyn-expr R X* (*fst* (*s-cf R S X p*) + *m*) (*ext-cf S m* (*s-cf R S X p*))),
thin-tac snd (*s-cf R S X p*) (*fst* (*s-cf R S X p*)) $\neq \mathbf{0}_S$)
apply (*subst P-mod-mod*[*THEN sym*, *of I p s-cf R S X p*], *assumption*+,
cut-tac s-cf-expr[*of p*], *simp*, *assumption*+,
rule allI, *rule impI*,
thin-tac pol-coeff S (*ext-cf S m* (*s-cf R S X p*)),
simp add:*ext-cf-len*, *simp add*:*ext-cf-def*)
apply (*drule-tac x* = *m* + *j* **in** *spec*,
frule-tac i = *j* **and** *j* = *fst* (*s-cf R S X p*) **and** *k* = *m* **and** *l* = *m* **in**
add-le-mono, *simp*, *simp only*:*add.commute*[*of - m*],
thin-tac j ≤ *fst* (*s-cf R S X p*), *simp*,
simp add:*sliden-def*)
apply *simp*
done

lemma (**in** *PolynRg*) *P-mod-multTr*:⟦*ideal S I*; *ideal S J*; *pol-coeff S* (*n*, *f*);
P-mod R S X I (*polyn-expr R X n* (*n*, *f*))⟧ ⟹ ∀ *g*. ((*pol-coeff S* (*m*, *g*)
∧ (*P-mod R S X J* (*polyn-expr R X m* (*m*, *g*)))) ⟶
P-mod R S X (*I* $\diamondsuit_{r\,S}$ *J*)
((*polyn-expr R X n* (*n*, *f*)) $\cdot_r$ (*polyn-expr R X m* (*m*, *g*))))
apply (*cut-tac subring*, *frule subring-Ring*,
cut-tac ring-is-ag, *cut-tac X-mem-R*)
apply (*frule polyn-mem*[*of* (*n*, *f*) *n*], *simp*)
apply (*frule Ring.ideal-prod-ideal*[*of S I J*], *assumption*+)
apply (*case-tac polyn-expr R X n* (*n*, *f*) = $\mathbf{0}_R$, *simp*)
apply (*rule allI*, *rule impI*, *erule conjE*)
apply (*frule-tac c* = (*m*, *g*) **in** *polyn-mem*[*of - m*], *simp*,
simp add:*ring-times-0-x*, *simp add*:*P-mod-def*)
apply (*induct-tac m*)
apply (*rule allI*, *rule impI*, *erule conjE*,
rule-tac g = *g* **in** *P-mod-mult-const*[*of I J n f*], *assumption*+)

apply (*rule allI*, *rule impI*, *erule conjE*)
apply (*frule-tac* $n = na$ **and** $f = g$ **in** *pol-coeff-pre*,
frule-tac $n = na$ **and** $f = g$ **in** *P-mod-pre*[*of J*], *assumption*+)
apply (*drule-tac* $a = g$ **in** *forall-spec*, *simp*)
apply (*frule-tac* $n = na$ **and** $f = g$ **in** *polyn-Suc-split*, *simp del*:*npow-suc*)
apply (*thin-tac polyn-expr R X* (*Suc na*) (*Suc na*, *g*) =
polyn-expr R X na (*na*, *g*) ± *g* (*Suc na*) $\cdot_r$ $X^{\hat{}R}$ (*Suc na*))
apply (*frule-tac* $c = (na, g)$ **and** $k = na$ **in** *polyn-mem*, *simp*,
subgoal-tac (*g* (*Suc na*)) $\cdot_r$ ($X^{\hat{}R}$ (*Suc na*)) ∈ *carrier R*,
subst ring-distrib1, *assumption*+)
apply (*frule-tac* $p = ($*polyn-expr R X n* $(n, f)) \cdot_r ($*polyn-expr R X na* $(na, g))$
and $q = ($*polyn-expr R X n* $(n, f)) \cdot_r$
$((g\ (Suc\ na)) \cdot_r (X^{\hat{}R}\ (Suc\ na)))$ **in**
P-mod-add[*of I* $\diamondsuit_{r\,S}$ *J*])
apply (*simp add*:*ring-tOp-closed*, *rule ring-tOp-closed*, *assumption*+)
apply (*frule-tac* $c = (Suc\ na, g)$ **and** $j = Suc\ na$ **in** *pol-coeff-mem-R*,
simp)
apply (*subst ring-tOp-assoc*[*THEN sym*], *assumption*+, *simp*,
rule npClose, *assumption*+)
apply (*subst P-mod-mult-monomial*[*THEN sym*, *of I* $\diamondsuit_{r\,S}$ *J*], *assumption*,
rule ring-tOp-closed, *assumption*+, *simp add*:*pol-coeff-mem-R*)
apply (*rule P-mod-mult-const1*, *assumption*+,
thin-tac P-mod R S X (*I* $\diamondsuit_{r\,S}$ *J*)
(*polyn-expr R X n* $(n, f) \cdot_r$ *polyn-expr R X na* (na, g)))
apply (*cut-tac n1* = *na* **and** *c1* = (*Suc na*, *g*) **in** *polyn-Suc*[*THEN sym*], *simp*,
simp,
frule-tac $c = (Suc\ na, g)$ **and** $k = na$ **in** *polyn-expr-short*,
simp, *simp*,
thin-tac P-mod R S X J (*polyn-expr R X na* (*na*, *g*)),
thin-tac polyn-expr R X na (*na*, *g*) ∈ *carrier R*,
thin-tac polyn-expr R X na (*na*, *g*) ± *g* (*Suc na*) $\cdot_r$ ($X^{\hat{}R}$ *na* $\cdot_r$ *X*)
= *polyn-expr R X* (*Suc na*) (*Suc na*, *g*),
thin-tac polyn-expr R X na (*Suc na*, *g*) =
polyn-expr R X na (*na*, *g*))
apply (*frule-tac p1* = *polyn-expr R X* (*Suc na*) (*Suc na*, *g*) **and**
c1 = (*Suc na*, *g*) **in** *P-mod-mod*[*THEN sym*, *of J*],
simp add:*polyn-mem*, *assumption*, *simp*, *simp*)

apply (*simp*,
rule ring-tOp-closed,
cut-tac $c = (Suc\ na, g)$ **and** $j = Suc\ na$ **in** *pol-coeff-mem-R*,
assumption, *simp*, *simp*, *rule npClose*, *assumption*+)
done

lemma (**in** *PolynRg*) *P-mod-mult*:⟦*ideal S I*; *ideal S J*; *pol-coeff S* (*n*, *c*);
pol-coeff S (*m*, *d*); *P-mod R S X I* (*polyn-expr R X n* (*n*, *c*));
P-mod R S X J (*polyn-expr R X m* (*m*, *d*))⟧ ⟹
P-mod R S X (*I* $\diamondsuit_{r\,S}$ *J*) ((*polyn-expr R X n* (*n*, *c*)) $\cdot_r$
(*polyn-expr R X m* (*m*, *d*)))

apply (*simp add*:*P-mod-multTr*)
done

lemma (**in** *PolynRg*) *P-mod-mult1*:⟦*ideal S I*; *ideal S J*;
p ∈ *carrier R*; *q* ∈ *carrier R*; *P-mod R S X I p*; *P-mod R S X J q*⟧ ⟹
P-mod R S X (*I* ◇$_{r\,S}$ *J*) (*p* ·$_r$ *q*)
apply (*case-tac p* = **0**$_R$)
apply (*simp add*:*ring-times-0-x*, *simp add*:*P-mod-def*)
apply (*case-tac q* = **0**$_R$)
apply (*simp add*:*ring-times-x-0*, *simp add*:*P-mod-def*)

apply (*frule s-cf-expr*[*of p*], *assumption*+,
frule s-cf-expr[*of q*], *assumption*+, (*erule conjE*)+)
apply (*cut-tac P-mod-mult*[*of I J fst* (*s-cf R S X p*) *snd* (*s-cf R S X p*)
fst (*s-cf R S X q*) *snd* (*s-cf R S X q*)])
apply (*simp add*:*polyn-expr-split*[*THEN sym*], *assumption*+)
apply (*simp add*:*pol-coeff-split*[*THEN sym*])
apply (*simp add*:*polyn-expr-split*[*THEN sym*])+
done

lemma (**in** *PolynRg*) *P-mod-mult2l*:⟦*ideal S I*; *p* ∈ *carrier R*; *q* ∈ *carrier R*;
P-mod R S X I p⟧ ⟹ *P-mod R S X I* (*p* ·$_r$ *q*)
apply (*cut-tac subring*, *frule subring-Ring*[*of S*],
frule Ring.whole-ideal[*of S*])
apply (*frule P-mod-whole*[*of q*])
apply (*frule P-mod-mult1*[*of I carrier S p q*], *assumption*+)
apply (*simp add*:*Ring.idealprod-whole-r*)
done

lemma (**in** *PolynRg*) *P-mod-mult2r*:⟦*ideal S I*; *p* ∈ *carrier R*; *q* ∈ *carrier R*;
P-mod R S X I q⟧ ⟹ *P-mod R S X I* (*p* ·$_r$ *q*)
apply (*cut-tac subring*, *frule subring-Ring*[*of S*],
frule Ring.whole-ideal[*of S*])
apply (*frule P-mod-whole*[*of p*])
apply (*frule P-mod-mult1*[*of carrier S I p q*], *assumption*+)
apply (*simp add*:*Ring.idealprod-whole-l*)
done

lemma (**in** *PolynRg*) *csrp-fn-pol-coeff*:⟦*ideal S P*; *PolynRg R'* (*S* /$_r$ *P*) *Y*;
pol-coeff (*S* /$_r$ *P*) (*n*, *c'*)⟧ ⟹
pol-coeff S (*n*, (*cmp* (*csrp-fn S P*) *c'*))
apply (*cut-tac subring*, *frule subring-Ring*)
apply (*simp add*:*pol-coeff-def*)
apply (*rule allI*, *rule impI*, *simp add*:*cmp-def*)
apply (*rule Ring.csrp-fn-mem*[*of S P*], *assumption*+)
apply *simp*
done

lemma (**in** *PolynRg*) *pj-csrp-mem-coeff*:⟦*ideal S P*; *pol-coeff* (*S* /$_r$ *P*) (*n*, *c'*)⟧

$\Longrightarrow \forall j \leq n.\ (pj\ S\ P)\ ((csrp\text{-}fn\ S\ P)\ (c'\ j)) = c'\ j$
apply (*cut-tac subring*, *frule subring-Ring*)
apply (*rule allI*, *rule impI*, *simp add*:*pol-coeff-def*)
apply (*simp add*:*Ring.csrp-pj*)
done

lemma (**in** *PolynRg*) *pHom-pj-csrp*:⟦*Idomain S*; *ideal S P*;
PolynRg R' (*S* $/_r$ *P*) *Y*; *pol-coeff* (*S* $/_r$ *P*) (*n*, *c'*)⟧ $\Longrightarrow$
erH R S X R' (*S* $/_r$ *P*) *Y* (*pj S P*)
(*polyn-expr R X n* (*n*, (*cmp* (*csrp-fn S P*) *c'*)))
= *polyn-expr R' Y n* (*n*, *c'*)
apply (*cut-tac subring*, *frule subring-Ring*,
frule Ring.qring-ring[*of S P*], *assumption*+)

apply (*subst pHom-mem*[*of R'* (*S* $/_r$ *P*) *Y erH R S X R'* (*S* $/_r$ *P*) *Y* (*pj S P*)
n cmp (*csrp-fn S P*) *c'*], *assumption*+,
rule erH-rHom[*of R' S* $/_r$ *P Y pj S P*],
assumption+,
simp add:*pj-Hom*, *simp add*:*csrp-fn-pol-coeff*)
apply (*rule PolynRg.polyn-exprs-eq*[*of R' S* $/_r$ *P Y*
(*n*, *cmp* (*erH R S X R'* (*S* $/_r$ *P*) *Y* (*pj S P*)) (*cmp* (*csrp-fn S P*) *c'*))
(*n*, *c'*) *n*], *assumption*+)
apply (*frule csrp-fn-pol-coeff*[*of P R' Y n c'*], *assumption*+,
frule erH-rHom [*of R' S* $/_r$ *P Y pj S P*], *assumption*+,
simp add:*pj-Hom*,
rule cmp-pol-coeff-e[*of R' S* $/_r$ *P Y erH R S X R'* (*S* $/_r$ *P*) *Y* (*pj S P*)
n cmp (*csrp-fn S P*) *c'*], *assumption*+, *simp*)
apply (*rule allI*, *rule impI*, *simp add*:*cmp-def*,
frule-tac c = (*n*, *c'*) **and** *j* = *j* **in**
PolynRg.pol-coeff-mem[*of R' S* $/_r$ *P Y*], *assumption*+, *simp*+,
frule-tac x = *c' j* **in** *Ring.csrp-fn-mem*[*of S P*], *assumption*+,
frule-tac s = *csrp-fn S P* (*c' j*) **in**
erH-rHom-cf[*of R' S* $/_r$ *P Y pj S P*], *assumption*+,
simp add:*pj-Hom*, *assumption*+)
apply (*simp add*:*pj-csrp-mem-coeff*)
done

lemma (**in** *PolynRg*) *ext-csrp-fn-nonzero*:⟦*Idomain S*; *ideal S P*;
PolynRg R' (*S* $/_r$ *P*) *Y*; *g'* ∈ *carrier R'*; *g'* ≠ $\mathbf{0}_{R'}$⟧ $\Longrightarrow$
polyn-expr R X (*deg-n R'* (*S* $/_r$ *P*) *Y g'*) ((*deg-n R'* (*S* $/_r$ *P*) *Y g'*),
(*cmp* (*csrp-fn S P*) (*snd* (*s-cf R'* (*S* $/_r$ *P*) *Y g'*)))) ≠ **0**
apply (*cut-tac subring*, *frule subring-Ring*,
frule Ring.qring-ring[*of S P*], *assumption*+,
frule pj-Hom[*of S P*], *assumption*+,
frule PolynRg.s-cf-expr[*of R' S* $/_r$ *P Y g'*], *assumption*+,
(*erule conjE*)+)
apply (*simp add*:*PolynRg.s-cf-deg*[*THEN sym*, *of R' S* $/_r$ *P Y g'*],
frule csrp-fn-pol-coeff[*of P R' Y deg-n R'* (*S* $/_r$ *P*) *Y g'*
snd (*s-cf R'* (*S* $/_r$ *P*) *Y g'*)], *assumption*+,

simp add:*PolynRg.s-cf-deg*[*of* R' S $/_r$ P Y g'])
apply (*subst coeff-nonzero-polyn-nonzero*[*of* (*deg-n* R' (S $/_r$ P) Y g',
cmp (*csrp-fn* S P) (*snd* (*s-cf* R' (S $/_r$ P) Y g')))
deg-n R' (S $/_r$ P) Y g'], *assumption*+, *simp*)
apply (*simp add*:*cmp-def*, *rule contrapos-pp*, *simp*+)
apply (*drule-tac* a = *deg-n* R' (S $/_r$ P) Y g' **in** *forall-spec*, *simp*,
frule pj-csrp-mem-coeff[*of* P *deg-n* R' (S $/_r$ P) Y g'
snd (*s-cf* R' (S $/_r$ P) Y g')],
simp add:*PolynRg.s-cf-deg*[*of* R' S $/_r$ P Y g'])
apply (*drule-tac* a = *deg-n* R' (S $/_r$ P) Y g' **in** *forall-spec*, *simp*,
simp,
frule pj-Hom[*of* S P], *assumption*, *simp add*:*rHom-0-0*)
done

lemma (**in** *PolynRg*) *erH-inv*:⟦*Idomain* S; *ideal* S P; *Ring* R';
PolynRg R' (S $/_r$ P) Y; $g' \in$ *carrier* R'⟧ ⟹
$\exists g \in$*carrier* R. *deg* R S X $g \leq$ (*deg* R' (S $/_r$ P) Y g') ∧
(*erH* R S X R' (S $/_r$ P) Y (*pj* S P)) $g = g'$
apply (*cut-tac subring*, *frule subring-Ring*,
frule Ring.qring-ring[*of* S P], *assumption*+,
frule pj-Hom[*of* S P], *assumption*+)
apply (*frule erH-rHom*[*of* R' S $/_r$ P Y *pj* S P], *assumption*+)
apply (*case-tac* $g' = \mathbf{0}_{R'}$, *simp*,
frule erH-rHom-0[*of* R' S $/_r$ P Y *pj* S P], *assumption*+,
cut-tac ring-zero,
subgoal-tac deg R S X (**0**) ≤ *deg* R' (S $/_r$ P) Y g', *blast*,
simp add:*deg-def*)
apply (*frule PolynRg.s-cf-expr* [*of* R' S $/_r$ P Y g'], *assumption*+,
(*erule conjE*)+)
apply (*frule pHom-pj-csrp*[*of* P R' Y *fst* (*s-cf* R' (S $/_r$ P) Y g')
snd (*s-cf* R' (S $/_r$ P) Y g')], *assumption*+,
simp add:*PolynRg.pol-coeff-split*[*THEN sym*],
drule sym, *simp*)
apply (*subgoal-tac deg* R S X (*polyn-expr* R X (*fst* (*s-cf* R' (S $/_r$ P)
Y g')) (*fst* (*s-cf* R' (S $/_r$ P) Y g'), *cmp* (*csrp-fn* S P)
(*snd* (*s-cf* R' (S $/_r$ P) Y g')))) ≤ *deg* R' (S $/_r$ P) Y g',
subgoal-tac polyn-expr R X (*fst* (*s-cf* R' (S $/_r$ P) Y g'))
(*fst* (*s-cf* R' (S $/_r$ P) Y g'),
cmp (*csrp-fn* S P) (*snd* (*s-cf* R' (S $/_r$ P) Y g'))) ∈ *carrier* R,
blast)
apply(*thin-tac* *deg* R S X
(*polyn-expr* R X (*fst* (*s-cf* R' (S $/_r$ P) Y g'))
(*fst* (*s-cf* R' (S $/_r$ P) Y g'),
cmp (*csrp-fn* S P) (*snd* (*s-cf* R' (S $/_r$ P) Y g'))))
≤ *deg* R' (S $/_r$ P) Y g',
thin-tac polyn-expr R' Y (*fst* (*s-cf* R' (S $/_r$ P) Y g'))
(*s-cf* R' (S $/_r$ P) Y g') = g',
thin-tac erH R S X R' (S $/_r$ P) Y (*pj* S P)
(*polyn-expr* R X (*fst* (*s-cf* R' (S $/_r$ P) Y g'))

```
        (fst (s-cf R' (S /r P) Y g'),
        cmp (csrp-fn S P) (snd (s-cf R' (S /r P) Y g')))) = g',
    thin-tac snd (s-cf R' (S /r P) Y g') (fst (s-cf R' (S /r P) Y g')) ≠
          0S /r P)
 apply (rule-tac c = (fst (s-cf R' (S /r P) Y g'),
        cmp (csrp-fn S P) (snd (s-cf R' (S /r P) Y g'))) and
        k = fst (s-cf R' (S /r P) Y g') in polyn-mem)
 apply (rule csrp-fn-pol-coeff, assumption+,
        simp, simp,
        cut-tac pol-deg-le-n[of polyn-expr R X (fst (s-cf R' (S /r P) Y g'))
         (fst (s-cf R' (S /r P) Y g'),
         cmp (csrp-fn S P) (snd (s-cf R' (S /r P) Y g')))
           (fst (s-cf R' (S /r P) Y g'),
         cmp (csrp-fn S P) (snd (s-cf R' (S /r P) Y g')))])
apply (simp,
       simp add:PolynRg.s-cf-deg[THEN sym, of R' S /r P Y g'],
       frule ext-csrp-fn-nonzero[of P R' Y g'], assumption+,
       simp add:deg-def, simp add:ale-natle,
       rule polyn-mem, simp add:csrp-fn-pol-coeff, simp,
       simp add:csrp-fn-pol-coeff, simp)
done

lemma (in PolynRg) P-mod-0:[[Idomain S; ideal S P; PolynRg R' (S /r P) Y;
      g ∈ carrier R]] ⟹
     (erH R S X R' (S /r P) Y (pj S P) g = 0R') = (P-mod R S X P g)
apply (cut-tac subring, frule subring-Ring,
       frule Ring.qring-ring[of S P], assumption+,
       frule pj-Hom[of S P], assumption+)
apply (case-tac g = 0R,
       cut-tac ring-zero, simp add:P-mod-def,
       rule erH-rHom-0[of R' S /r P Y pj S P], assumption+)
apply (frule s-cf-expr[of g], assumption+, (erule conjE)+,
       cut-tac polyn-expr-split[of fst (s-cf R S X g) s-cf R S X g])
apply (frule erH-map[of R' S /r P Y pj S P fst (s-cf R S X g)
                     snd (s-cf R S X g)], assumption+)
     apply (subst pol-coeff-split[THEN sym], assumption)
     apply (drule sym, simp)
     apply (thin-tac erH R S X R' (S /r P) Y (pj S P) g =
      polyn-expr R' Y (fst (s-cf R S X g))
     (fst (s-cf R S X g), cmp (pj S P) (snd (s-cf R S X g))))
     apply (rotate-tac −1, drule sym)
apply (subst P-mod-mod[THEN sym, of P g s-cf R S X g], assumption+,
       thin-tac g = polyn-expr R X (fst (s-cf R S X g)) (s-cf R S X g),
       frule erH-rHom-coeff[of R' S /r P Y pj S P fst (s-cf R S X g)
       snd (s-cf R S X g)], assumption+, simp)
apply (subst PolynRg.coeff-0-pol-0[THEN sym, of R' S /r P Y
        (fst (s-cf R S X g), cmp (pj S P) (snd (s-cf R S X g)))
        fst (s-cf R S X g)], assumption+, simp,
       thin-tac pol-coeff (S /r P)
```

(*fst* (*s-cf R S X g*), *cmp* (*pj S P*) (*snd* (*s-cf R S X g*))))
apply (*simp add*:*cmp-def*)
apply (*rule iffI*)
apply (*rule allI*, *rule impI*,
drule-tac a = *j* **in** *forall-spec*, *assumption*,
frule-tac j = *j* **in** *pol-coeff-mem*[*of s-cf R S X g*], *assumption*+,
simp add:*pj-zero*[*of S P*])

apply (*rule allI*, *rule impI*,
drule-tac a = *j* **in** *forall-spec*, *assumption*,
frule-tac j = *j* **in** *pol-coeff-mem*[*of s-cf R S X g*], *assumption*+,
simp add:*pj-zero*[*THEN sym*, *of S P*])
done

lemma (**in** *PolynRg*) *P-mod-I-J*:⟦*p* ∈ *carrier R*; *ideal S I*; *ideal S J*;
I ⊆ *J*; *P-mod R S X I p*⟧ ⟹ *P-mod R S X J p*
apply (*case-tac p* = **0**$_R$, *simp*)
apply (*simp add*:*P-mod-def*)

apply (*frule s-cf-expr*[*of p*], *assumption*, (*erule conjE*)+)
apply (*frule P-mod-mod*[*THEN sym*, *of I p s-cf R S X p*], *assumption*+)
apply (*subst P-mod-mod*[*THEN sym*, *of J p s-cf R S X p*], *assumption*+,
thin-tac p = *polyn-expr R X* (*fst* (*s-cf R S X p*)) (*s-cf R S X p*),
simp)
apply (*rule allI*, *rule impI*, *drule-tac a* = *j* **in** *forall-spec*, *assumption*,
simp add:*subsetD*)
done

lemma (**in** *PolynRg*) *P-mod-n-1*:⟦*Idomain S*; *t* ∈ *carrier S*; *g* ∈ *carrier R*;
P-mod R S X (*S* ♢$_p$ (*t*^^{S} (*Suc n*))) *g*⟧ ⟹ *P-mod R S X* (*S* ♢$_p$ *t*) *g*
apply (*cut-tac subring*, *frule subring-Ring*,
frule Ring.npClose[*of S t n*], *assumption*+,
frule Ring.npClose[*of S t Suc n*], *assumption*+,
frule Ring.principal-ideal[*of S t*], *assumption*+,
frule Ring.principal-ideal[*of S t*^^{S} (*Suc n*)], *assumption*+)
apply (*case-tac g* = **0**$_R$, *simp add*:*P-mod-def*)
apply (*frule s-cf-expr*[*of g*], *assumption*,
(*erule conjE*)+,
subst P-mod-mod[*THEN sym*, *of S* ♢$_p$ *t g s-cf R S X g*],
assumption+)
apply (*frule-tac P-mod-mod*[*THEN sym*, *of S* ♢$_p$ (*t*^^{S} (*Suc n*)) *g s-cf R S X g*],
assumption+)
apply (*simp del*:*npow-suc*,
thin-tac g = *polyn-expr R X* (*fst* (*s-cf R S X g*)) (*s-cf R S X g*))
apply (*rule allI*, *rule impI*,
drule-tac a = *j* **in** *forall-spec*, *assumption*+)
apply (*simp add*:*Rxa-def*, *erule bexE*, *simp*,
simp add:*Ring.ring-tOp-assoc*[*THEN sym*, *of S*],
frule-tac x = *r* **and** *y* = *t*^^{S} *n* **in** *Ring.ring-tOp-closed*, *assumption*+,

blast)
done

lemma (**in** *PolynRg*) *P-mod-n-m*:⟦*Idomain S*; *t* ∈ *carrier S*; *g* ∈ *carrier R*;
m ≤ *n*; *P-mod R S X* (*S* ◇$_p$ (*t* ^*S* (*Suc n*))) *g*⟧ ⟹
P-mod R S X (*S* ◇$_p$ (*t* ^*S* (*Suc m*))) *g*
apply (*cut-tac subring*, *frule subring-Ring*)
apply (*rule P-mod-I-J*[*of g S* ◇$_p$ (*t* ^*S* (*Suc n*)) *S* ◇$_p$ (*t* ^*S* (*Suc m*))],
assumption)
apply (*rule Ring.principal-ideal*, *assumption*+,
rule Ring.npClose, *assumption*+)
apply (*rule Ring.principal-ideal*, *assumption*+,
rule Ring.npClose, *assumption*+)
apply (*thin-tac P-mod R S X* (*S* ◇$_p$ (*t* ^*S* (*Suc n*))) *g*)
apply (*rule subsetI*)
apply (*simp del*:*npow-suc add*:*Rxa-def*, *erule bexE*, *simp del*:*npow-suc*)
apply (*frule Ring.npMulDistr*[*THEN sym*, *of S t Suc n* − *Suc m Suc m*],
assumption)
apply (*simp del*:*npow-suc*,
thin-tac t ^*S* (*Suc n*) = *t* ^*S* (*n* − *m*) ·$_{rS}$ *t* ^*S* (*Suc m*),
thin-tac x = *r* ·$_{rS}$ (*t* ^*S* (*n* − *m*) ·$_{rS}$ *t* ^*S* (*Suc m*)))
apply (*subst Ring.ring-tOp-assoc*[*THEN sym*, *of S*], *assumption*+,
(*rule Ring.npClose*, *assumption*+)+)
apply (*frule-tac x* = *r* **and** *y* = *t* ^*S* (*n* − *m*) **in** *Ring.ring-tOp-closed*,
assumption+,
rule Ring.npClose, *assumption*+, *blast*)
apply *assumption*
done

lemma (**in** *PolynRg*) *P-mod-diff*:⟦*Idomain S*; *ideal S P*; *PolynRg R*′ (*S* /$_r$ *P*) *Y*;

g ∈ *carrier R*; *h* ∈ *carrier R*⟧ ⟹
(*erH R S X R*′ (*S* /$_r$ *P*) *Y* (*pj S P*) *g* = (*erH R S X R*′ (*S* /$_r$ *P*) *Y* (*pj S P*)
h))
= (*P-mod R S X P* (*g* ± −$_a$ *h*))
apply (*cut-tac ring-is-ag*,
frule PolynRg.is-Ring[*of R*′],
cut-tac subring,
frule subring-Ring,
frule Ring.qring-ring[*of S P*], *assumption*+,
frule pj-Hom[*of S P*], *assumption*+,
frule erH-rHom[*of R*′ *S* /$_r$ *P Y pj S P*],
assumption+,
frule Ring.ring-is-ag[*of R*′])
apply (*frule erH-mem*[*of R*′ *S* /$_r$ *P Y pj S P g*], *assumption*+,
frule erH-mem[*of R*′ *S* /$_r$ *P Y pj S P h*], *assumption*+)
apply (*rule iffI*)
apply (*frule-tac a* = *erH R S X R*′ (*S* /$_r$ *P*) *Y* (*pj S P*) *g* **and**

```
                b = erH R S X R' (S /r P) Y (pj S P) h in
      aGroup.ag-eq-diffzero[of R'], assumption+, simp,
      simp add:erH-minus[THEN sym, of R' S /r P Y pj S P h],
      drule sym, simp,
      thin-tac erH R S X R' (S /r P) Y (pj S P) h =
                    erH R S X R' (S /r P) Y (pj S P) g,
      frule-tac x = h in aGroup.ag-mOp-closed, assumption+,
      simp add:erH-add[THEN sym, of R' S /r P Y pj S P g −a h])
apply (subst P-mod-0[THEN sym, of P R' Y g ± −a h], assumption+,
       rule aGroup.ag-pOp-closed, assumption+)

apply (frule-tac a = erH R S X R' (S /r P) Y (pj S P) g and
                b = erH R S X R' (S /r P) Y (pj S P) h in
      aGroup.ag-eq-diffzero[of R'], assumption+, simp,
      simp add:erH-minus[THEN sym, of R' S /r P Y pj S P h])
  apply (subst erH-add[THEN sym, of R' S /r P Y pj S P g −a h],
         assumption+,
         rule aGroup.ag-mOp-closed, assumption+)
  apply (subst P-mod-0[of P R' Y g ± −a h], assumption+,
         rule aGroup.ag-pOp-closed, assumption+,
         rule aGroup.ag-mOp-closed, assumption+)
done

lemma (in PolynRg) P-mod-erH:⟦Idomain S; ideal S P; PolynRg R' (S /r P) Y;

       g ∈ carrier R; v ∈ carrier R; t ∈ P ⟧ ⟹
       (erH R S X R' (S /r P) Y (pj S P) g =
              (erH R S X R' (S /r P) Y (pj S P) (g ± (t ·r v))))
apply (cut-tac subring, frule subring-Ring,
      cut-tac ring-is-ag,
      frule Ring.ideal-subset[of S P t], assumption+,
      frule mem-subring-mem-ring[of S t], assumption+,
      frule ring-tOp-closed[of t v], assumption+)
apply (subst P-mod-diff[of P R' Y g g ± (t ·r v)], assumption+,
      rule aGroup.ag-pOp-closed, assumption+)
apply (simp add:aGroup.ag-p-inv,
      frule aGroup.ag-mOp-closed[of R g], assumption+,
      frule aGroup.ag-mOp-closed[of R t ·r v], assumption+,
      subst aGroup.ag-pOp-assoc[THEN sym], assumption+,
      simp add:aGroup.ag-r-inv1, simp add:aGroup.ag-l-zero)
apply (rule P-mod-minus[of P t ·r v], assumption+,
      frule P-mod-mult1[of P carrier S t v],
      simp add:Ring.whole-ideal, assumption+,
      subst P-mod-coeffTr[of P t], assumption+,
      rule P-mod-whole[of v], assumption+,
      simp add:Ring.idealprod-whole-r[of S P])
done

lemma (in PolynRg) coeff-principalTr:⟦t ∈ carrier S⟧ ⟹
```

$\forall f.\ pol\text{-}coeff\ S\ (n, f) \wedge (\forall j \leq n.\ f\, j \in S \diamondsuit_p t) \longrightarrow$
$(\exists f'.\ pol\text{-}coeff\ S\ (n, f') \wedge (\forall j \leq n.\ f\, j = t \cdot_{r S} (f'\, j)))$
apply (*cut-tac subring*, *frule subring-Ring*,
cut-tac ring-is-ag)
apply (*induct-tac n*,
rule allI, *rule impI*, *erule conjE*,
simp add:*Rxa-def*, *erule bexE*,
simp add:*Ring.ring-tOp-commute*[*of S - t*],
subgoal-tac pol-coeff S (*0*, (*λj. r*)),
subgoal-tac t $\cdot_{r S}$ *r* = *t* $\cdot_{r S}$ ((*λj. r*) *0*), *blast*,
simp,
simp add:*pol-coeff-def*)

apply (*rule allI*, *rule impI*, *erule conjE*,
frule-tac n = *n* **and** *f* = *f* **in** *pol-coeff-pre*,
subgoal-tac $\forall j \leq n.\ f\, j \in S \diamondsuit_p t$,
drule-tac a = *f* **in** *forall-spec*, *simp*,
erule exE, *erule conjE*,
frule-tac c = (*Suc n*, *f*) **and** *j* = *Suc n* **in**
pol-coeff-mem, *simp*, *simp*,
drule-tac x = *Suc n* **in** *spec*, *simp*,
simp add:*Rxa-def*,
erule bexE, *simp add*:*Ring.ring-tOp-commute*[*of S - t*])
apply (*subgoal-tac pol-coeff S* ((*Suc n*), (*λj. if j* ≤ *n then* (*f' j*) *else r*))
∧
(∀ *j* ≤ (*Suc n*). *f j* = *t* $\cdot_{r S}$ ((*λj. if j* ≤ *n then* (*f' j*) *else r*) *j*)),
blast)
apply (*rule conjI*, *simp add*:*pol-coeff-def*,
rule allI, *rule impI*,
case-tac j ≤ *n*, *simp*)
apply *simp*
apply (*drule-tac y* = *j* **and** *x* = *n* **in** *not-le-imp-less*,
drule-tac m = *n* **and** *n* = *j* **in** *Suc-leI*)
apply (*frule-tac m* = *j* **and** *n* = *Suc n* **in** *le-antisym*, *assumption*, *simp*,
thin-tac $\forall f.\ pol\text{-}coeff\ S\ (n, f) \wedge (\forall j \leq n.\ f\, j \in S \diamondsuit_p t) \longrightarrow$
$(\exists f'.\ pol\text{-}coeff\ S\ (n, f') \wedge (\forall j \leq n.\ f\, j = t \cdot_{r S} f'\, j)))$
apply (*rule allI*, *rule impI*,
drule-tac a = *j* **in** *forall-spec*, *simp*+)
done

lemma (**in** *PolynRg*) *coeff-principal*:⟦*t* ∈ *carrier S*; *pol-coeff S* (*n*, *f*);
∀ *j* ≤ *n*. *f j* ∈ *S* $\diamondsuit_p$ *t*⟧ ⟹
∃ *f'*. *pol-coeff S* (*n*, *f'*) ∧ (∀ *j* ≤ *n*. *f j* = *t* $\cdot_{r S}$ (*f' j*))
apply (*simp add*:*coeff-principalTr*)
done

lemma (**in** *PolynRg*) *Pmod-0-principal*:⟦*Idomain S*; *t* ∈ *carrier S*; *g* ∈ *carrier R*;
P-mod R S X (*S* $\diamondsuit_p$ *t*) *g*⟧ ⟹ ∃ *h*∈ *carrier R*. *g* = *t* $\cdot_r$ *h*
apply (*cut-tac subring*, *frule subring-Ring*)

apply (*case-tac* $g = \mathbf{0}_R$,
cut-tac ring-zero,
frule mem-subring-mem-ring[*of S t*], *assumption*+,
frule ring-times-x-0[*THEN sym*, *of t*], *blast*)

apply (*frule s-cf-expr*[*of g*], *assumption*+,
(*erule conjE*)+, *frule Ring.principal-ideal*[*of S t*], *assumption*,
simp add:*P-mod-mod*[*THEN sym*, *of S* $\diamondsuit_p$ *t g*],
frule coeff-principal[*of t fst* (*s-cf R S X g*) *snd* (*s-cf R S X g*)],
simp add:*pol-coeff-split*[*THEN sym*], *assumption*+,
erule exE, *erule conjE*)
apply (*frule-tac c* = (*fst* (*s-cf R S X g*), *f* ′) **and** *k* = *fst* (*s-cf R S X g*)
in *polyn-mem*, *simp*,
subgoal-tac g = *t* $\cdot_r$
(*polyn-expr R X* (*fst* (*s-cf R S X g*)) (*fst* (*s-cf R S X g*), *f* ′)),
blast)
apply (*subst scalar-times-pol-expr*[*of t* (*fst* (*s-cf R S X g*), *f* ′)
fst (*s-cf R S X g*)], *assumption*+, *simp*,
drule sym,
subgoal-tac polyn-expr R X (*fst* (*s-cf R S X g*)) (*s-cf R S X g*) =
polyn-expr R X (*fst* (*s-cf R S X g*)) (*sp-cf S t* (*fst* (*s-cf R S X g*), *f* ′)),
simp)
apply (*frule-tac c* = (*fst* (*s-cf R S X g*), *f* ′) **in** *sp-cf-pol-coeff*[*of - t*],
assumption+,
frule-tac d = *sp-cf S t* (*fst* (*s-cf R S X g*), *f* ′) **in**
pol-expr-unique2[*of s-cf R S X g*], *assumption*+,
simp, *simp add*:*sp-cf-len*, *simp add*:*sp-cf-len*,
thin-tac (*g* =
polyn-expr R X (*fst* (*s-cf R S X g*))
(*sp-cf S t* (*fst* (*s-cf R S X g*), *f* ′))) =
(∀ *j*≤*fst* (*s-cf R S X g*).
t $\cdot_{r\,S}$ *f* ′ *j* = *snd* (*sp-cf S t* (*fst* (*s-cf R S X g*), *f* ′)) *j*),
thin-tac polyn-expr R X (*fst* (*s-cf R S X g*)) (*s-cf R S X g*) = *g*,
thin-tac polyn-expr R X (*fst* (*s-cf R S X g*)) (*fst* (*s-cf R S X g*), *f* ′)
∈ *carrier R*)
apply (*rule allI*, *rule impI*,
drule-tac a = *j* **in** *forall-spec*, *assumption*+,
simp add:*sp-cf-def*)
done

lemma (**in** *PolynRg*) *Pmod0-principal-rev*:⟦*Idomain S*; *t* ∈ *carrier S*;
g ∈ *carrier R*; ∃ *h*∈ *carrier R*. *g* = *t* $\cdot_r$ *h*⟧ ⟹
P-mod R S X (*S* $\diamondsuit_p$ *t*) *g*
apply (*cut-tac subring*, *frule subring-Ring*)
apply (*erule bexE*)
apply (*case-tac t* = $\mathbf{0}_S$,
frule Subring-zero-ring-zero, *simp*)
apply (*simp add*:*ring-times-0-x*, *simp add*:*P-mod-def*)

apply (*case-tac* $h = \mathbf{0}_R$, *simp*,
frule mem-subring-mem-ring[*of S t*], *assumption*+,
simp add:*ring-times-x-0*, *simp add*:*P-mod-def*,
cut-tac polyn-ring-integral, *simp*)
apply (*frule-tac p* = *h* **in** *s-cf-expr*, *assumption*+, (*erule conjE*)+,
frule-tac c = *s-cf R S X h* **and** *n* = *fst* (*s-cf R S X h*) **in**
scalar-times-pol-expr[*of t*], *assumption*+, *simp*,
thin-tac g = $t \cdot_r h$,
drule sym, *simp*)
apply (*frule Ring.principal-ideal*[*of S t*], *assumption*+,
frule-tac c1 = *sp-cf S t* (*s-cf R S X h*) **and** *p1* = $t \cdot_r h$ **in**
P-mod-mod[*THEN sym*],
frule-tac x = *t* **in** *mem-subring-mem-ring*, *assumption*,
rule ring-tOp-closed, *assumption*+,
simp add:*sp-cf-pol-coeff*, *simp add*:*sp-cf-len*)
apply (*drule sym*, *simp*,
thin-tac P-mod R S X (*S* $\diamondsuit_p$ *t*) (*t* $\cdot_r$ *h*) =
(∀ *j*≤*fst* (*sp-cf S t* (*s-cf R S X h*)).
snd (*sp-cf S t* (*s-cf R S X h*)) *j* ∈ *S* $\diamondsuit_p$ *t*),
thin-tac polyn-expr R X (*fst* (*s-cf R S X h*))
(*sp-cf S t* (*s-cf R S X h*)) = *t* $\cdot_r$ *h*,
thin-tac polyn-expr R X (*fst* (*s-cf R S X h*)) (*s-cf R S X h*) = *h*)
apply (*rule allI*, *rule impI*, *simp add*:*sp-cf-len*,
subst sp-cf-def, *simp*, *subst Rxa-def*, *simp*,
frule-tac c = *s-cf R S X h* **and** *j* = *j* **in** *pol-coeff-mem*,
assumption)
apply (*simp add*:*Ring.ring-tOp-commute*[*of S t*], *blast*)
done

lemma (**in** *PolynRg*) *Pmod0-principal-rev1*:⟦*Idomain S*; *t* ∈ *carrier S*;
h ∈ *carrier R*⟧ ⟹ *P-mod R S X* (*S* $\diamondsuit_p$ *t*) (*t* $\cdot_r$ *h*)
apply (*rule Pmod0-principal-rev*[*of t t* $\cdot_r$ *h*], *assumption*+)
apply (*cut-tac subring*,
frule mem-subring-mem-ring[*of S t*], *assumption*+,
simp add:*ring-tOp-closed*)
apply *blast*
done

lemma (**in** *PolynRg*) *Pmod0-principal-erH-vanish-t*:⟦*Idomain S*; *ideal S* (*S* $\diamondsuit_p$ *t*);
t ∈ *carrier S*; *t* ≠ $\mathbf{0}_S$; *PolynRg R'* (*S* /$_r$ (*S* $\diamondsuit_p$ *t*)) *Y* ⟧ ⟹
erH R S X R' (*S* /$_r$ (*S* $\diamondsuit_p$ *t*)) *Y* (*pj S* (*S* $\diamondsuit_p$ *t*)) *t* = $\mathbf{0}_{R'}$
apply (*cut-tac subring*, *frule subring-Ring*,
frule mem-subring-mem-ring[*of S t*], *assumption*+)
apply (*subst P-mod-0*[*of S* $\diamondsuit_p$ *t R' Y t*], *assumption*+)
apply (*rule Pmod0-principal-rev*[*of t t*], *assumption*+)
apply (*cut-tac ring-one*,
frule ring-r-one[*THEN sym*, *of t*], *blast*)

done

lemma (**in** *PolynRg*) *P-mod-diffxxx1*:⟦*Idomain S*; $t \in$ *carrier S*; $t \neq \mathbf{0}_S$;
maximal-ideal S (*S* $\diamondsuit_p$ *t*); *PolynRg R'* (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y*;
$f \in$ *carrier R*; $g \in$ *carrier R*; $h \in$ *carrier R*;
$f \neq \mathbf{0}$; $g \neq \mathbf{0}$; $h \neq \mathbf{0}$; $u \in$ *carrier R*; $v \in$ *carrier R*;
erH R S X R' (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y* (*pj S* (*S* $\diamondsuit_p$ *t*)) $g \neq \mathbf{0}_{R'}$;
erH R S X R' (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y* (*pj S* (*S* $\diamondsuit_p$ *t*)) $h \neq \mathbf{0}_{R'}$;
$ra \in$ *carrier R*;
$f \pm_a (g \cdot_r h) = t \hat{}^S m \cdot_r ra$; $0 < m$;
(*erH R S X R'* (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y* (*pj S* (*S* $\diamondsuit_p$ *t*)) *u*)
$\cdot_{r_{R'}}$ *erH R S X R'* (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y* (*pj S* (*S* $\diamondsuit_p$ *t*)) $g \pm_{R'}$
(*erH R S X R'* (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y* (*pj S* (*S* $\diamondsuit_p$ *t*)) *v*)
$\cdot_{r_{R'}}$ *erH R S X R'* (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y* (*pj S* (*S* $\diamondsuit_p$ *t*)) *h* =
erH R S X R' (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y* (*pj S* (*S* $\diamondsuit_p$ *t*)) *ra*⟧
$\Longrightarrow$ *P-mod R S X* (*S* $\diamondsuit_p$ ($t \hat{}^S$ (*Suc m*)))
($f \pm_a ((g \pm t \hat{}^S m \cdot_r v) \cdot_r (h \pm t \hat{}^S m \cdot_r u))$)
apply (*cut-tac is-Ring*,
cut-tac subring, *frule subring-Ring*,
cut-tac ring-is-ag,
frule PolynRg.is-Ring[*of R' S* $/_r$ (*S* $\diamondsuit_p$ *t*) *Y*],
frule Ring.ring-is-ag[*of R'*],
frule Ring.maximal-ideal-ideal[*of S S* $\diamondsuit_p$ *t*], *assumption*+,
frule Ring.qring-ring[*of S S* $\diamondsuit_p$ *t*], *assumption*+,
frule erH-rHom[*of R' S* $/_r$ (*S* $\diamondsuit_p$ *t*) *Y pj S* (*S* $\diamondsuit_p$ *t*)], *assumption*+,
frule mem-subring-mem-ring[*of S t*], *assumption*+)
apply (*rule pj-Hom*[*of S S* $\diamondsuit_p$ *t*], *assumption*+,
frule pHom-rHom[*of R' S* $/_r$ (*S* $\diamondsuit_p$ *t*) *Y*
erH R S X R' (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y* (*pj S* (*S* $\diamondsuit_p$ *t*))], *assumption*+)
apply (*simp del*:*npow-suc add*:*rHom-tOp*[*THEN sym*])
apply (*frule-tac ring-tOp-closed*[*of u g*], *assumption*,
frule-tac ring-tOp-closed[*of v h*], *assumption*)
apply (*simp del*:*npow-suc add*:*rHom-add*[*THEN sym*])
apply (*rotate-tac 17*, *drule sym*)
apply (*frule P-mod-diff*[*of S* $\diamondsuit_p$ *t R' Y ra u* $\cdot_r$ *g* $\pm$ *v* $\cdot_r$ *h*], *assumption*+)
apply (*rule aGroup.ag-pOp-closed*, *assumption*+, *simp del*:*npow-suc*)
apply (*frule Pmod-0-principal*[*of t ra* $\pm_a$ (*u* $\cdot_r$ *g* $\pm$ *v* $\cdot_r$ *h*)], *assumption*+)
apply (*rule aGroup.ag-pOp-closed*, *assumption*+,
rule aGroup.ag-mOp-closed, *assumption*,
rule aGroup.ag-pOp-closed, *assumption*+, *erule bexE*)

apply (*frule Ring.npClose*[*of S t m*], *assumption*+,
frule mem-subring-mem-ring[*of S* $t \hat{}^S m$], *assumption*+,
subst ring-distrib1,
rule aGroup.ag-pOp-closed, *assumption*+,
rule ring-tOp-closed, *simp add*:*mem-subring-mem-ring*,
assumption+)
apply (*rule ring-tOp-closed*, *assumption*+)
apply (*subst ring-distrib2*, *assumption*+,

rule ring-tOp-closed, *assumption+*)
apply (*frule-tac* $x = g$ **and** $y = h$ **in** *ring-tOp-closed*, *assumption+*,
frule-tac $x = t\hat{}^{S\ m}$ **and** $y = v$ **in** *ring-tOp-closed*, *assumption+*,
frule-tac $x = t\hat{}^{S\ m}$ **and** $y = u$ **in** *ring-tOp-closed*, *assumption+*,
frule-tac $x = t\hat{}^{S\ m} \cdot_r v$ **and** $y = h$ **in** *ring-tOp-closed*, *assumption+*)
apply (*subst ring-distrib2*, *assumption+*,
frule-tac $x = t\hat{}^{S\ m} \cdot_r v$ **and** $y = t\hat{}^{S\ m} \cdot_r u$ **in** *ring-tOp-closed*,
assumption+)
apply (*subst aGroup.ag-p-inv*, *assumption+*,
rule aGroup.ag-pOp-closed, *assumption+*,
rule aGroup.ag-pOp-closed, *assumption+*,
rule ring-tOp-closed, *assumption+*)
apply (*subst aGroup.ag-p-inv*, *assumption+*,
frule aGroup.ag-mOp-closed[*of R* $g \cdot_r h$], *assumption+*,
frule aGroup.ag-mOp-closed[*of R* $t\hat{}^{S\ m} \cdot_r v \cdot_r h$], *assumption+*)
apply (*subst aGroup.ag-pOp-assoc*[*of R* $-_a (g \cdot_r h)\ -_a (t\hat{}^{S\ m} \cdot_r v \cdot_r h)$],
assumption+)
apply (*rule aGroup.ag-mOp-closed*, *assumption*,
rule aGroup.ag-pOp-closed, *assumption*,
rule ring-tOp-closed, *assumption+*)
apply (*subst aGroup.ag-pOp-assoc*[*THEN sym*], *assumption+*,
rule aGroup.ag-pOp-closed, *assumption*,
rule aGroup.ag-mOp-closed, *assumption+*,
rule aGroup.ag-mOp-closed, *assumption+*,
rule aGroup.ag-pOp-closed, *assumption+*,
rule ring-tOp-closed, *assumption+*, *simp del*:*npow-suc*)

apply (*subst aGroup.ag-p-inv*, *assumption*,
rule ring-tOp-closed, *assumption+*,
simp del:*npow-suc add*:*ring-tOp-assoc*[*of* $t\hat{}^{S\ m}\ v\ h$],
simp add:*del*:*npow-suc add*:*ring-tOp-commute*[*of* $g\ t\hat{}^{S\ m} \cdot_r u$],
simp del:*npow-suc add*:*ring-tOp-assoc*[*of* $t\hat{}^{S\ m}\ u\ g$],
simp del:*npow-suc add*:*ring-inv1-2*,
subst aGroup.ag-pOp-assoc[*THEN sym*], *assumption+*,
rule ring-tOp-closed, *assumption+*)
apply (*rule aGroup.ag-pOp-closed*, *assumption+*,
(*rule ring-tOp-closed*, *assumption+*)+,
rule aGroup.ag-mOp-closed, *assumption+*,
(*rule ring-tOp-closed*, *assumption+*)+,
rule aGroup.ag-mOp-closed, *assumption+*)
apply (*subst ring-distrib1*[*THEN sym*, *of* $t\hat{}^{S\ m}$ *ra* $v \cdot_r (-_a h)$],
assumption+,
rule ring-tOp-closed, *assumption+*, *rule aGroup.ag-mOp-closed*,
assumption+)
apply (*subst aGroup.ag-pOp-assoc*[*THEN sym*], *assumption+*,
rule ring-tOp-closed, *assumption+*,
rule aGroup.ag-pOp-closed, *assumption+*,
rule ring-tOp-closed, *assumption+*, *rule aGroup.ag-mOp-closed*,

assumption+)
apply ((*rule ring-tOp-closed*, *assumption*+)+,
rule aGroup.ag-mOp-closed, *assumption*+,
(*rule ring-tOp-closed*, *assumption*+)+,
rule aGroup.ag-mOp-closed, *assumption*+)
apply (*subst ring-distrib1*[*THEN sym*, *of* $t\hat{}^S\ m$],
assumption+,
rule aGroup.ag-pOp-closed, *assumption*+,
rule ring-tOp-closed, *assumption*+,
rule aGroup.ag-mOp-closed, *assumption*+,
rule ring-tOp-closed, *assumption*+,
rule aGroup.ag-mOp-closed, *assumption*+)
apply (*subst ring-tOp-assoc*[*of* $t\hat{}^S\ m\ v$], *assumption*+,
rule ring-tOp-closed, *assumption*+,
rule aGroup.ag-mOp-closed, *assumption*+)
apply (*subst ring-distrib1*[*THEN sym*, *of* $t\hat{}^S\ m$],
assumption+,
rule aGroup.ag-pOp-closed, *assumption*+,
rule aGroup.ag-pOp-closed, *assumption*+,
rule ring-tOp-closed, *assumption*,
rule aGroup.ag-mOp-closed, *assumption*+,
rule ring-tOp-closed, *assumption*,
rule aGroup.ag-mOp-closed, *assumption*+,
(*rule ring-tOp-closed*, *assumption*+)+,
rule aGroup.ag-mOp-closed, *assumption*+)
apply (*frule ring-tOp-closed*[*of u g*], *assumption*+,
frule ring-tOp-closed[*of v h*], *assumption*+,
simp del:*npow-suc add*:*aGroup.ag-p-inv*[*of R u* $\cdot_r$ *g v* $\cdot_r$ *h*],
simp del:*npow-suc add*:*add*:*ring-inv1-2*,
frule aGroup.ag-mOp-closed[*of R g*], *assumption*+,
frule aGroup.ag-mOp-closed[*of R h*], *assumption*+,
frule ring-tOp-closed[*of u* $-_a$ *g*], *assumption*+,
frule ring-tOp-closed[*of v* $-_a$ *h*], *assumption*+,
simp del:*npow-suc add*:*aGroup.ag-pOp-commute*[*of R*
u $\cdot_r$ ($-_a$ *g*) *v* $\cdot_r$ ($-_a$ *h*)],
simp del:*npow-suc add*:*aGroup.ag-pOp-assoc*[*THEN sym*,
of R ra v $\cdot_r$ ($-_a$ *h*) *u* $\cdot_r$ ($-_a$ *g*)])
apply (*subst ring-tOp-assoc*[*THEN sym*, *of v* $t\hat{}^S\ m$ $-_a$ *u*], *assumption*+,
rule aGroup.ag-mOp-closed, *assumption*+,
simp only:*ring-tOp-commute*[*of v* $t\hat{}^S\ m$],
subgoal-tac $t\hat{}^S\ m$ $\cdot_r$ *v* = *t* $\cdot_r$ ($t\hat{}^S$ (*m* − *Suc 0*) $\cdot_r$ *v*),
simp del:*npow-suc*)
apply (*subst ring-tOp-assoc*[*of t*],
frule mem-subring-mem-ring[*of S t*], *assumption*+,
rule ring-tOp-closed)
apply (*frule Ring.npClose*[*of S t m* − *Suc 0*], *assumption*+,
simp add:*mem-subring-mem-ring*, *assumption*,
rule aGroup.ag-mOp-closed, *assumption*+,
subst ring-distrib1[*THEN sym*, *of t*],

simp add:*mem-subring-mem-ring*, *assumption*+)
apply ((*rule ring-tOp-closed*)+,
frule Ring.npClose[*of S t m − Suc 0*], *assumption*+,
simp add:*mem-subring-mem-ring*, *assumption*,
rule aGroup.ag-mOp-closed, *assumption*+)
apply (*subst ring-tOp-assoc*[*THEN sym*],
frule Ring.npClose[*of S t m − Suc 0*], *assumption*+,
simp add:*mem-subring-mem-ring*)

apply (*rule aGroup.ag-pOp-closed*, *assumption*+,
rule ring-tOp-closed,
frule Ring.npClose[*of S t m − Suc 0*], *assumption*+,
rule ring-tOp-closed, *simp add*:*mem-subring-mem-ring*,
assumption, *rule aGroup.ag-mOp-closed*, *assumption*+,
simp add:*Subring-tOp-ring-tOp*[*THEN sym*],
simp only:*npow-suc*[*THEN sym*, *of S t m*])
apply (*rule Pmod0-principal-rev1*[*of* $t \hat{}_S$ (*Suc m*)], *assumption*+,
rule Ring.npClose, *assumption*+,
rule aGroup.ag-pOp-closed, *assumption*+,
(*rule ring-tOp-closed*)+,
frule Ring.npClose[*of S t m − Suc 0*], *assumption*+,
simp add:*mem-subring-mem-ring*, *assumption*,
rule aGroup.ag-mOp-closed, *assumption*+)
apply (*frule Ring.npClose*[*of S t m − Suc 0*], *assumption*+,
frule mem-subring-mem-ring[*of S t*], *assumption*,
frule mem-subring-mem-ring[*of S* $t \hat{}_S$ (*m − Suc 0*)], *assumption*,
simp add:*ring-tOp-assoc*[*THEN sym*],
simp add:*ring-tOp-commute*[*of t* $t \hat{}_S$ (*m − Suc 0*)],
subgoal-tac $t \hat{}_S$ *m* = $t \hat{}_S$ (*Suc* (*m − Suc 0*)),
simp del:*Suc-pred add*:*Subring-tOp-ring-tOp*,
simp only:*Suc-pred*)
done

lemma (**in** *PolynRg*) *P-mod-diffxxx2*:⟦*Idomain S*; *t* ∈ *carrier S*; *t* ≠ $\mathbf{0}_S$;
maximal-ideal S (*S* ◇$_p$ *t*); *PolynRg R′* (*S* /$_r$ (*S* ◇$_p$ *t*)) *Y*;
f ∈ *carrier R*; *g* ∈ *carrier R*; *h* ∈ *carrier R*;
deg R S X g ≤ *deg R′* (*S* /$_r$ (*S* ◇$_p$ *t*)) *Y*
(*erH R S X R′* (*S* /$_r$ (*S* ◇$_p$ *t*)) *Y* (*pj S* (*S* ◇$_p$ *t*)) *g*);
deg R S X h +
deg R′ (*S* /$_r$ (*S* ◇$_p$ *t*)) *Y* (*erH R S X R′* (*S* /$_r$ (*S* ◇$_p$ *t*)) *Y* (*pj S* (*S* ◇$_p$ *t*)) *g*)
≤ *deg R S X f*;
0 < *deg R′* (*S* /$_r$ (*S* ◇$_p$ *t*)) *Y*
(*erH R S X R′* (*S* /$_r$ (*S* ◇$_p$ *t*)) *Y* (*pj S* (*S* ◇$_p$ *t*)) *g*);
0 < *deg R′* (*S* /$_r$ (*S* ◇$_p$ *t*)) *Y*
(*erH R S X R′* (*S* /$_r$ (*S* ◇$_p$ *t*)) *Y* (*pj S* (*S* ◇$_p$ *t*)) *h*);
rel-prime-pols R′ (*S* /$_r$ (*S* ◇$_p$ *t*)) *Y*
(*erH R S X R′* (*S* /$_r$ (*S* ◇$_p$ *t*)) *Y* (*pj S* (*S* ◇$_p$ *t*)) *g*)
(*erH R S X R′* (*S* /$_r$ (*S* ◇$_p$ *t*)) *Y* (*pj S* (*S* ◇$_p$ *t*)) *h*);

$P\text{-}mod\ R\ S\ X\ (S \diamondsuit_p (t\hat{}^S\ m))\ (f \pm -_a (g \cdot_r h));\ 0 < m\rrbracket \Longrightarrow$
$\exists\, g1\ h1.\ g1 \in carrier\ R \wedge h1 \in carrier\ R \wedge$
$(deg\ R\ S\ X\ g1 \leq deg\ R'\ (S /_r (S \diamondsuit_p t))\ Y$
$(erH\ R\ S\ X\ R'\ (S /_r (S \diamondsuit_p t))\ Y\ (pj\ S\ (S \diamondsuit_p t))\ g1)) \wedge$
$P\text{-}mod\ R\ S\ X\ (S \diamondsuit_p (t\hat{}^S\ m))\ (g \pm -_a g1) \wedge\ (deg\ R\ S\ X\ h1 +$
$deg\ R'\ (S /_r (S \diamondsuit_p t))\ Y\ (erH\ R\ S\ X\ R'\ (S /_r (S \diamondsuit_p t))\ Y\ (pj\ S\ (S \diamondsuit_p t))\ g1)$
$\leq deg\ R\ S\ X\ f) \wedge$
$P\text{-}mod\ R\ S\ X\ (S \diamondsuit_p (t\hat{}^S\ m))\ (h \pm -_a h1) \wedge$
$P\text{-}mod\ R\ S\ X\ (S \diamondsuit_p (t\hat{}^S\ (Suc\ m)))\ (f \pm (-_a (g1 \cdot_r h1)))$

apply (*cut-tac subring*, *frule subring-Ring*,
cut-tac ring-is-ag,
frule Ring.residue-field-cd[*of S S* $\diamondsuit_p$ *t*], *assumption*+,
frule Ring.maximal-ideal-ideal[*of S S* $\diamondsuit_p$ *t*], *assumption*+,
frule pj-Hom[*of S S* $\diamondsuit_p$ *t*], *assumption*+,
frule mem-subring-mem-ring[*of S t*], *assumption*+,
frule Ring.qring-ring[*of S S* $\diamondsuit_p$ *t*], *assumption*+,
frule PolynRg.pol-nonzero[*of R' S* $/_r$ $(S \diamondsuit_p t)$ *Y*
erH R S X R' $(S /_r (S \diamondsuit_p t))$ *Y* $(pj\ S\ (S \diamondsuit_p t))$ *g*],
rule erH-mem, *assumption*+,
frule erH-rHom-nonzero[*of R' S* $/_r$ $(S \diamondsuit_p t)$ *Y pj S* $(S \diamondsuit_p t)$ *g*],
assumption+, *simp add*:*aless-imp-le*)

apply (*frule PolynRg.pol-nonzero*[*of R' S* $/_r$ $(S \diamondsuit_p t)$ *Y*
erH R S X R' $(S /_r (S \diamondsuit_p t))$ *Y* $(pj\ S\ (S \diamondsuit_p t))$ *h*],
rule erH-mem, *assumption*+,
frule erH-rHom-nonzero[*of R' S* $/_r$ $(S \diamondsuit_p t)$ *Y pj S* $(S \diamondsuit_p t)$ *h*],
assumption+, *simp add*:*aless-imp-le*, *simp del*:*npow-suc add*:*aless-imp-le*)

apply (
frule pol-nonzero[*THEN sym*, *of h*], *simp del*:*npow-suc*,
frule aadd-pos-poss[*of deg R S X h deg R'* $(S /_r (S \diamondsuit_p t))$ *Y*
$(erH\ R\ S\ X\ R'\ (S /_r (S \diamondsuit_p t))\ Y\ (pj\ S\ (S \diamondsuit_p t))\ g)$], *assumption*+,
frule aless-le-trans[*of 0* $(deg\ R\ S\ X\ h)$ +
$(deg\ R'\ (S /_r (S \diamondsuit_p t))\ Y$
$(erH\ R\ S\ X\ R'\ (S /_r (S \diamondsuit_p t))\ Y\ (pj\ S\ (S \diamondsuit_p t))\ g))$
deg R S X f], *assumption*+,
frule pol-nonzero[*of f*], *simp del*:*npow-suc add*:*aless-imp-le*)

apply (*thin-tac* $0 < deg\ R\ S\ X\ f$,
thin-tac $0 < deg\ R\ S\ X\ h$ +
$deg\ R'\ (S /_r (S \diamondsuit_p t))\ Y$
$(erH\ R\ S\ X\ R'\ (S /_r (S \diamondsuit_p t))\ Y\ (pj\ S\ (S \diamondsuit_p t))\ g))$

apply (*frule Pmod-0-principal*[*of* $t\hat{}^S\ m\ f \pm -_a (g \cdot_r h)$],
rule Ring.npClose, *assumption*+)

apply (*rule aGroup.ag-pOp-closed*, *assumption*+,
rule aGroup.ag-mOp-closed, *assumption*+,
rule ring-tOp-closed, *assumption*+,
erule bexE, *rename-tac ra*)

apply (*frule deg-mult-pols1* [*of g h*], *assumption*+,
frule aadd-le-mono[*of deg R S X g deg R'* $(S /_r (S \diamondsuit_p t))$ *Y*

$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ g)$
$deg\ R\ S\ X\ h])$

apply $(simp\ only{:}aadd\text{-}commute[of\ deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ g)$
$deg\ R\ S\ X\ h])$

apply $(frule\ ale\text{-}trans[of\ deg\ R\ S\ X\ g\ +\ deg\ R\ S\ X\ h\ deg\ R\ S\ X\ h\ +$
$deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$(pj\ S\ (S\ \diamondsuit_p\ t))\ g)\ \ deg\ R\ S\ X\ f],\ assumption+)$

apply $(thin\text{-}tac\ deg\ R\ S\ X\ g\ +\ deg\ R\ S\ X\ h$
$\leq deg\ R\ S\ X\ h\ +$
$deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ g))$

apply $(frule\text{-}tac\ ring\text{-}tOp\text{-}closed[of\ g\ h],\ assumption+,$
$frule\ deg\text{-}minus\text{-}eq1[of\ g\ \cdot_r\ h],$
$frule\ polyn\text{-}deg\text{-}add4[of\ f\ -_a\ (g\ \cdot_r\ h)\ deg\text{-}n\ R\ S\ X\ f],$
$rule\ aGroup.ag\text{-}mOp\text{-}closed,\ assumption+)$

apply $(subst\ deg\text{-}an[THEN\ sym],\ assumption+,\ simp\ del{:}npow\text{-}suc)$

apply $(simp\ add{:}deg\text{-}an[THEN\ sym],\ simp\ del{:}npow\text{-}suc\ add{:}deg\text{-}an[THEN\ sym],$
$thin\text{-}tac\ deg\ R\ S\ X\ (g\ \cdot_r\ h) = deg\ R\ S\ X\ g\ +\ deg\ R\ S\ X\ h,$
$thin\text{-}tac\ deg\ R\ S\ X\ g\ +\ deg\ R\ S\ X\ h \leq deg\ R\ S\ X\ f,$
$thin\text{-}tac\ deg\ R\ S\ X\ (-_a\ (g\ \cdot_r\ h)) = deg\ R\ S\ X\ g\ +\ deg\ R\ S\ X\ h)$

apply $(frule\ Ring.npClose[of\ S\ t\ m],\ assumption,$
$frule\ Idomain.idom\text{-}potent\text{-}nonzero[of\ S\ t\ m],\ assumption+,$
$frule\text{-}tac\ p = ra\ \textbf{in}\ const\text{-}times\text{-}polyn1[of\ -\ t\hat{}^S\ m],\ assumption+,$
$simp\ del{:}npow\text{-}suc)$

apply $(frule\text{-}tac\ h =$
$erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ ra\ \textbf{in}$
$PolynRg.rel\text{-}prime\text{-}equation[of\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ g$
$erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ h],$
$assumption+,$
$simp\ del{:}npow\text{-}suc\ add{:}erH\text{-}mem,\ simp\ del{:}npow\text{-}suc\ add{:}erH\text{-}mem,$
$assumption+,$
$simp\ del{:}npow\text{-}suc\ add{:}erH\text{-}mem)$

apply $(erule\ bexE,\ erule\ bexE,\ (erule\ conjE)+,$
$frule\text{-}tac\ erH\text{-}mem[of\ R'\ S\ /_r\ (S\ \diamondsuit_p\ t)\ Y$
$pj\ S\ (S\ \diamondsuit_p\ t)\ g],\ assumption+,$
$frule\text{-}tac\ erH\text{-}mem[of\ R'\ S\ /_r\ (S\ \diamondsuit_p\ t)\ Y$
$pj\ S\ (S\ \diamondsuit_p\ t)\ h],\ assumption+)$

apply $(rename\text{-}tac\ ra\ u'\ v')$

apply $(frule\text{-}tac\ g' = v'\ \textbf{in}\ erH\text{-}inv[of\ S\ \diamondsuit_p\ t\ R'\ Y],\ assumption+,$
$simp\ add{:}PolynRg.is\text{-}Ring[of\ R'],\ assumption+)$

apply $(frule\text{-}tac\ g' = u'\ \textbf{in}\ erH\text{-}inv[of\ S\ \diamondsuit_p\ t\ R'\ Y\ ],\ assumption+,$
$simp\ add{:}PolynRg.is\text{-}Ring[of\ R'],\ assumption+)$

apply ((*erule bexE*)+, *rename-tac ra u′ v′ v u*, (*erule conjE*)+)
apply (
frule-tac p1 = u **in** *erH-mult*[*THEN sym*, *of R′ S* $/_r$ (*S* $\diamondsuit_p$ *t*) *Y*
pj S (*S* $\diamondsuit_p$ *t*) - *g*], *assumption*+,
frule-tac p1 = v **in** *erH-mult*[*THEN sym*, *of R′ S* $/_r$ (*S* $\diamondsuit_p$ *t*) *Y*
pj S (*S* $\diamondsuit_p$ *t*) - *h*], *assumption*+,
thin-tac 0 < *deg R′* (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y*
(*erH R S X R′* (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y* (*pj S* (*S* $\diamondsuit_p$ *t*)) *g*),
thin-tac 0 < *deg R′* (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y*
(*erH R S X R′* (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y* (*pj S* (*S* $\diamondsuit_p$ *t*)) *h*),
thin-tac rel-prime-pols R′ (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y*
(*erH R S X R′* (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y* (*pj S* (*S* $\diamondsuit_p$ *t*)) *g*)
(*erH R S X R′* (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y* (*pj S* (*S* $\diamondsuit_p$ *t*)) *h*))
apply (*subgoal-tac g* $\pm$ (*t* $\hat{}_S$ *m*) $\cdot_r$ *v* $\in$ *carrier R* $\wedge$
h $\pm$ (*t* $\hat{}_S$ *m*) $\cdot_r$ *u* $\in$ *carrier R* $\wedge$
deg R S X (*g* $\pm$ (*t* $\hat{}_S$ *m*) $\cdot_r$ *v*) $\leq$ *deg R′* (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y*
(*erH R S X R′* (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y*
(*pj S* (*S* $\diamondsuit_p$ *t*)) (*g* $\pm$ (*t* $\hat{}_S$ *m*) $\cdot_r$ *v*)) $\wedge$
P-mod R S X (*S* $\diamondsuit_p$ (*t* $\hat{}_S$ *m*)) (*g* $\pm$ $-_a$ (*g* $\pm$ (*t* $\hat{}_S$ *m*) $\cdot_r$ *v*)) $\wedge$
deg R S X (*h* $\pm$ (*t* $\hat{}_S$ *m*) $\cdot_r$ *u*) + *deg R′* (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y*
(*erH R S X R′* (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y* (*pj S* (*S* $\diamondsuit_p$ *t*))
(*g* $\pm$ (*t* $\hat{}_S$ *m*) $\cdot_r$ *v*)) $\leq$ *deg R S X f* $\wedge$
P-mod R S X (*S* $\diamondsuit_p$ (*t* $\hat{}_S$ *m*)) (*h* $\pm$ $-_a$ (*h* $\pm$ (*t* $\hat{}_S$ *m*) $\cdot_r$ *u*)) $\wedge$
P-mod R S X (*S* $\diamondsuit_p$ (*t* $\hat{}_S$ (*Suc m*)))
(*f* $\pm$ $-_a$ ((*g* $\pm$ (*t* $\hat{}_S$ *m*) $\cdot_r$ *v*) $\cdot_r$ (*h* $\pm$ (*t* $\hat{}_S$ *m*) $\cdot_r$ *u*))))
apply (*thin-tac deg R S X h* +
deg R′ (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y*
(*erH R S X R′* (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y* (*pj S* (*S* $\diamondsuit_p$ *t*)) *g*)
$\leq$ *deg R S X f*,
thin-tac deg R′ (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y u′*
$\leq$ *amax*
(*deg R′* (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y*
(*erH R S X R′* (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y* (*pj S* (*S* $\diamondsuit_p$ *t*)) *ra*) $-$
deg R′ (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y*
(*erH R S X R′* (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y* (*pj S* (*S* $\diamondsuit_p$ *t*)) *g*))
(*deg R′* (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y*
(*erH R S X R′* (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y* (*pj S* (*S* $\diamondsuit_p$ *t*)) *h*)),
thin-tac deg R′ (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y v′* $\leq$ *deg R′* (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y*
(*erH R S X R′* (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y* (*pj S* (*S* $\diamondsuit_p$ *t*)) *g*),
thin-tac u′ $\cdot_{r\,R'}$ *erH R S X R′* (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y* (*pj S* (*S* $\diamondsuit_p$ *t*)) *g* $\pm_{R'}$
v′ $\cdot_{r\,R'}$ *erH R S X R′* (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y* (*pj S* (*S* $\diamondsuit_p$ *t*)) *h* =
erH R S X R′ (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y* (*pj S* (*S* $\diamondsuit_p$ *t*)) *ra*,
thin-tac erH R S X R′ (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y* (*pj S* (*S* $\diamondsuit_p$ *t*)) *g*
$\in$ *carrier R′*,
thin-tac erH R S X R′ (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y* (*pj S* (*S* $\diamondsuit_p$ *t*)) *h*
$\in$ *carrier R′*,
thin-tac erH R S X R′ (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y* (*pj S* (*S* $\diamondsuit_p$ *t*)) *v* = *v′*,
thin-tac erH R S X R′ (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y* (*pj S* (*S* $\diamondsuit_p$ *t*)) *u* = *u′*,

thin-tac erH R S X R' (S /$_r$ (S ◇$_p$ t)) Y (pj S (S ◇$_p$ t)) u ·$_r$ R'
erH R S X R' (S /$_r$ (S ◇$_p$ t)) Y (pj S (S ◇$_p$ t)) g =
erH R S X R' (S /$_r$ (S ◇$_p$ t)) Y (pj S (S ◇$_p$ t)) (u ·$_r$ g),
thin-tac erH R S X R' (S /$_r$ (S ◇$_p$ t)) Y (pj S (S ◇$_p$ t)) v ·$_r$ R'
erH R S X R' (S /$_r$ (S ◇$_p$ t)) Y (pj S (S ◇$_p$ t)) h =
erH R S X R' (S /$_r$ (S ◇$_p$ t)) Y (pj S (S ◇$_p$ t)) (v ·$_r$ h))
apply *blast*

apply (*frule mem-subring-mem-ring*[*of S t*^*S m*], *assumption*)
apply (*rule conjI*)
apply (*rule aGroup.ag-pOp-closed*, *assumption*+,
rule ring-tOp-closed, *assumption*+)
apply (*rule conjI*,
rule aGroup.ag-pOp-closed, *assumption*+,
rule ring-tOp-closed, *assumption*+)

apply (*frule Ring.a-in-principal*[*of S t*], *assumption*+,
frule Ring.maximal-ideal-ideal[*of S S* ◇$_p$ *t*], *assumption*+,
frule Ring.ideal-npow-closed[*of S S* ◇$_p$ *t t m*], *assumption*+,
frule PolynRg.is-Ring[*of R' S* /$_r$ (*S* ◇$_p$ *t*) *Y*],
frule Ring.ring-is-ag[*of R'*],
frule erH-rHom[*of R' S* /$_r$ (*S* ◇$_p$ *t*) *Y pj S* (*S* ◇$_p$ *t*)], *assumption*+)

apply (*rule conjI*)
apply (*frule pHom-dec-deg*[*of R' S* /$_r$ (*S* ◇$_p$ *t*) *Y*
erH R S X R' (*S* /$_r$ (*S* ◇$_p$ *t*)) *Y* (*pj S* (*S* ◇$_p$ *t*)) *g*], *assumption*+,
frule-tac i = deg R S X v **and** *j = deg R'* (*S* /$_r$ (*S* ◇$_p$ *t*)) *Y v'* **and**
k = deg R' (*S* /$_r$ (*S* ◇$_p$ *t*)) *Y*
(*erH R S X R'* (*S* /$_r$ (*S* ◇$_p$ *t*)) *Y* (*pj S* (*S* ◇$_p$ *t*)) *g*) **in** *ale-trans*,
assumption,
thin-tac deg R' (*S* /$_r$ (*S* ◇$_p$ *t*)) *Y u'* ≤ *amax* (*deg R'* (*S* /$_r$ (*S* ◇$_p$ *t*)) *Y*
(*erH R S X R'* (*S* /$_r$ (*S* ◇$_p$ *t*)) *Y* (*pj S* (*S* ◇$_p$ *t*)) *ra*) −
deg R' (*S* /$_r$ (*S* ◇$_p$ *t*)) *Y*
(*erH R S X R'* (*S* /$_r$ (*S* ◇$_p$ *t*)) *Y* (*pj S* (*S* ◇$_p$ *t*)) *g*))
(*deg R'* (*S* /$_r$ (*S* ◇$_p$ *t*)) *Y*
(*erH R S X R'* (*S* /$_r$ (*S* ◇$_p$ *t*)) *Y* (*pj S* (*S* ◇$_p$ *t*)) *h*)),
thin-tac u' ·$_r$ *R' erH R S X R'* (*S* /$_r$ (*S* ◇$_p$ *t*)) *Y* (*pj S* (*S* ◇$_p$ *t*)) *g* ±$_{R'}$
v' ·$_r$ *R' erH R S X R'* (*S* /$_r$ (*S* ◇$_p$ *t*)) *Y* (*pj S* (*S* ◇$_p$ *t*)) *h* =
erH R S X R' (*S* /$_r$ (*S* ◇$_p$ *t*)) *Y* (*pj S* (*S* ◇$_p$ *t*)) *ra*,
thin-tac deg R S X h +
deg R' (*S* /$_r$ (*S* ◇$_p$ *t*)) *Y*
(*erH R S X R'* (*S* /$_r$ (*S* ◇$_p$ *t*)) *Y* (*pj S* (*S* ◇$_p$ *t*)) *g*)
≤ *deg R S X f*)
apply (*subst P-mod-erH*[*THEN sym*, *of S* ◇$_p$ *t R' Y g* - *t*^*S m*],
assumption+,
thin-tac deg R' (*S* /$_r$ (*S* ◇$_p$ *t*)) *Y*
(*erH R S X R'* (*S* /$_r$ (*S* ◇$_p$ *t*)) *Y* (*pj S* (*S* ◇$_p$ *t*)) *g*)
≤ *deg R S X g*)
apply (*frule-tac p = v* **in** *const-times-polyn1*[*of* - *t*^*S m*], *assumption*+,

frule-tac $q = (t \hat{}^S m) \cdot_r v$ **in** *polyn-deg-add4*[*of* g -
$\quad$ *deg-n* R' $(S /_r (S \diamondsuit_p t))$ Y
$\quad$ $(erH\ R\ S\ X\ R'\ (S /_r (S \diamondsuit_p t))\ Y\ (pj\ S\ (S \diamondsuit_p t))\ g)$])
apply (*rule ring-tOp-closed*, *assumption*+,
$\quad$ *simp del*:*npow-suc add*:*PolynRg.deg-an*[*THEN sym*],
$\quad$ *simp add*:*PolynRg.deg-an*[*THEN sym*],
$\quad$ *simp add*:*PolynRg.deg-an*[*THEN sym*])

apply (*rule conjI*)
apply (*frule Ring.principal-ideal*[*of* S $t \hat{}^S m$], *assumption*+,
$\quad$ *frule Ring.a-in-principal*[*of* S $t \hat{}^S m$], *assumption*+)
apply (*frule-tac* $y = v$ **in** *ring-tOp-closed*[*of* $t \hat{}^S m$], *assumption*+,
$\quad$ *subst aGroup.ag-p-inv*, *assumption*+,
$\quad$ *frule aGroup.ag-mOp-closed*[*of* R g], *assumption*+,
$\quad$ *frule-tac* $x = (t \hat{}^S m) \cdot_r v$ **in** *aGroup.ag-mOp-closed*[*of* R],
$\quad$ *assumption*+)
apply (*subst aGroup.ag-pOp-assoc*[*THEN sym*], *assumption*+,
$\quad$ *subst aGroup.ag-r-inv1*[*of* R], *assumption*+,
$\quad$ *subst aGroup.ag-l-zero*[*of* R], *assumption*+,
$\quad$ *rule P-mod-minus*, *assumption*+)
apply (*rule-tac* $g = (t \hat{}^S m) \cdot_r v$ **in** *Pmod0-principal-rev*[*of*
$\quad$ $t \hat{}^S m$], *assumption*+)
apply *blast*

apply (*rule conjI*)
apply (*subst P-mod-erH*[*THEN sym*, *of* S $\diamondsuit_p$ t R' Y g - $t \hat{}^S m$], *assumption*+,
$\quad$ *thin-tac P-mod* R S X $(S \diamondsuit_p (t \hat{}^S m))$ $(t \hat{}^S m \cdot_r ra)$,
$\quad$ *thin-tac* $f \pm -_a (g \cdot_r h) = t \hat{}^S m \cdot_r ra$,
$\quad$ *thin-tac*
$\quad$ $u' \cdot_{r R'} erH\ R\ S\ X\ R'\ (S /_r (S \diamondsuit_p t))\ Y\ (pj\ S\ (S \diamondsuit_p t))\ g \pm_{R'}$
$\quad$ $v' \cdot_{r R'} erH\ R\ S\ X\ R'\ (S /_r (S \diamondsuit_p t))\ Y\ (pj\ S\ (S \diamondsuit_p t))\ h =$
$\quad$ $erH\ R\ S\ X\ R'\ (S /_r (S \diamondsuit_p t))\ Y\ (pj\ S\ (S \diamondsuit_p t))\ ra$)
apply (*case-tac*
$\quad$ $(deg\ R'\ (S /_r (S \diamondsuit_p t))\ Y$
$\quad$ $(erH\ R\ S\ X\ R'\ (S /_r (S \diamondsuit_p t))\ Y\ (pj\ S\ (S \diamondsuit_p t))\ ra) -$
$\quad$ $deg\ R'\ (S /_r (S \diamondsuit_p t))\ Y$
$\quad\quad$ $(erH\ R\ S\ X\ R'\ (S /_r (S \diamondsuit_p t))\ Y\ (pj\ S\ (S \diamondsuit_p t))\ g)) \leq$
$\quad$ $(deg\ R'\ (S /_r (S \diamondsuit_p t))\ Y$
$\quad\quad$ $(erH\ R\ S\ X\ R'\ (S /_r (S \diamondsuit_p t))\ Y\ (pj\ S\ (S \diamondsuit_p t))\ h)))$
apply (*simp add*:*amax-def*)
apply (*frule-tac* $i = deg\ R\ S\ X\ u$ **and** $j = deg\ R'\ (S /_r (S \diamondsuit_p t))\ Y\ u'$ **and**
$\quad$ $k = deg\ R'\ (S /_r (S \diamondsuit_p t))\ Y$
$\quad\quad$ $(erH\ R\ S\ X\ R'\ (S /_r (S \diamondsuit_p t))\ Y\ (pj\ S\ (S \diamondsuit_p t))\ h)$ **in** *ale-trans*,
$\quad\quad$ *assumption*+,
$\quad$ *frule pHom-dec-deg*[*of* R' $S /_r (S \diamondsuit_p t)$ Y
$\quad\quad$ $erH\ R\ S\ X\ R'\ (S /_r (S \diamondsuit_p t))\ Y\ (pj\ S\ (S \diamondsuit_p t))\ h$], *assumption*+,
$\quad$ *frule-tac* $i = deg\ R\ S\ X\ u$ **and** $j = deg\ R'\ (S /_r (S \diamondsuit_p t))\ Y$
$\quad$ $(erH\ R\ S\ X\ R'\ (S /_r (S \diamondsuit_p t))\ Y\ (pj\ S\ (S \diamondsuit_p t))\ h)$ **and**

$k = deg\ R\ S\ X\ h$ **in** *ale-trans*, *assumption+*,
frule-tac $p = u$ **and** $c = t\hat{}^{S}\ m$ **in** *const-times-polyn1*,
assumption+)
apply (*frule-tac* $q = (t\hat{}^{S}\ m) \cdot_r u$ **in** *polyn-deg-add4*[*of h - deg-n R S X h*],
rule ring-tOp-closed, *assumption+*,
subst deg-an[*THEN sym*], *assumption+*, *rule ale-refl*,
subst deg-an[*THEN sym*], *assumption+*,
simp, *frule deg-an*[*THEN sym*, *of h*], *assumption+*, *simp*)
apply (*frule-tac* $x = deg\ R\ S\ X\ (\ h \pm (t\hat{}^{S}\ m) \cdot_r u)$ **in** *aadd-le-mono*[*of -*
deg R S X h deg R' $(S\ /_r\ (S \diamondsuit_p t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S \diamondsuit_p t))\ Y\ (pj\ S\ (S \diamondsuit_p t))\ g)$],
rule-tac $i = deg\ R\ S\ X\ (\ h \pm (t\hat{}^{S}\ m) \cdot_r u) + (deg\ R'\ (S\ /_r\ (S \diamondsuit_p t))$
$Y\ (erH\ R\ S\ X\ R'\ (S\ /_r\ (S \diamondsuit_p t))\ Y\ (pj\ S\ (S \diamondsuit_p t))\ g))$ **in**
ale-trans[*of - deg R S X h* + $(deg\ R'\ (S\ /_r\ (S \diamondsuit_p t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S \diamondsuit_p t))\ Y\ (pj\ S\ (S \diamondsuit_p t))\ g))$ *deg R S X f*],
assumption+)
apply (*simp add:amax-def*)
apply (*thin-tac* $\neg$ $deg\ R'\ (S\ /_r\ (S \diamondsuit_p t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S \diamondsuit_p t))\ Y\ (pj\ S\ (S \diamondsuit_p t))\ ra) -$
$deg\ R'\ (S\ /_r\ (S \diamondsuit_p t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S \diamondsuit_p t))\ Y\ (pj\ S\ (S \diamondsuit_p t))\ g)$
$\leq deg\ R'\ (S\ /_r\ (S \diamondsuit_p t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S \diamondsuit_p t))\ Y\ (pj\ S\ (S \diamondsuit_p t))\ h))$

apply (*subst aplus-le-aminus*[*of - deg R'* $(S\ /_r\ (S \diamondsuit_p t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S \diamondsuit_p t))\ Y\ (pj\ S\ (S \diamondsuit_p t))\ g)$ *deg R S X f*])
apply (*rule deg-in-aug-minf*,
rule aGroup.ag-pOp-closed, *assumption+*,
rule ring-tOp-closed, *assumption+*,
rule PolynRg.deg-in-aug-minf, *assumption+*,
rule deg-in-aug-minf, *assumption+*)

apply (*subst PolynRg.deg-an*, *assumption+*, *simp add:minus-an-in-aug-minf*,
frule-tac $y = u$ **in** *ring-tOp-closed*[*of* $t\hat{}^{S}\ m$], *assumption+*,
frule-tac $q = (t\hat{}^{S}\ m) \cdot_r u$ **in** *polyn-deg-add5*[*of h -*
deg R S X f $-$ $deg\ R'\ (S\ /_r\ (S \diamondsuit_p t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S \diamondsuit_p t))\ Y\ (pj\ S\ (S \diamondsuit_p t))\ g)$],
assumption+,
frule deg-in-aug-minf[*of h*],
subst aplus-le-aminus[*THEN sym*, *of deg R S X h*
$deg\ R'\ (S\ /_r\ (S \diamondsuit_p t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S \diamondsuit_p t))\ Y\ (pj\ S\ (S \diamondsuit_p t))\ g)$
deg R S X f], *assumption+*,
rule PolynRg.deg-in-aug-minf, *assumption+*,
rule deg-in-aug-minf, *assumption+*,
subst PolynRg.deg-an, *assumption+*,
simp add:minus-an-in-aug-minf,
assumption)

apply (*subst const-times-polyn1*, *assumption+*,
frule-tac $i = deg\ R\ S\ X\ u$ **and** $j = deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ u'$ **and**
$k = deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ ra) -$
$deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ g)$ **in**
ale-trans, *assumption+*,
frule-tac $p = ra$ **in** *pHom-dec-deg*[*of* $R'\ S\ /_r\ (S\ \diamondsuit_p\ t)\ Y$
$erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))$], *assumption+*,
frule-tac $i = deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ ra)$ **and**
$j = deg\ R\ S\ X\ ra$ **and** $k = deg\ R\ S\ X\ f$ **in** *ale-trans*, *assumption+*,
frule-tac $a = deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ ra)$ **and**
$a' = deg\ R\ S\ X\ f$ **and** $b = deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ g)$ **in**
adiff-le-adiff,
frule-tac $i = deg\ R\ S\ X\ u$ **and** $j = deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ ra) -$
$deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ g)$ **and**
$k = deg\ R\ S\ X\ f - deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ g)$ **in**
ale-trans, *assumption+*)

apply (*rule conjI*)
apply (*frule Ring.principal-ideal*[*of* $S\ t\ \hat{}^S\ m$], *assumption+*,
frule Ring.a-in-principal[*of* $S\ t\ \hat{}^S\ m$], *assumption+*,
frule-tac $y = u$ **in** *ring-tOp-closed*[*of* $t\ \hat{}^S\ m$], *assumption+*,
subst aGroup.ag-p-inv, *assumption+*,
frule aGroup.ag-mOp-closed[*of* $R\ g$], *assumption+*,
frule-tac $x = (t\ \hat{}^S\ m)\ \cdot_r\ u$ **in** *aGroup.ag-mOp-closed*[*of* R],
assumption+,
subst aGroup.ag-pOp-assoc[*THEN sym*], *assumption+*,
rule aGroup.ag-mOp-closed, *assumption+*,
subst aGroup.ag-r-inv1[*of* R], *assumption+*,
subst aGroup.ag-l-zero[*of* R], *assumption+*,
rule P-mod-minus, *assumption+*,
rule-tac $g = (t\ \hat{}^S\ m)\ \cdot_r\ u$ **in** *Pmod0-principal-rev*[*of* $t\ \hat{}^S\ m$],
assumption+,
thin-tac $deg\ R\ S\ X\ g$
$\leq deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ g)$,
thin-tac $deg\ R\ S\ X\ h + deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ g) \leq deg\ R\ S\ X\ f$,
thin-tac $deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ u' \leq amax$
$(deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ ra) -$
$deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$

$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ g))$
$(deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ h)),$
$thin\text{-}tac\ deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ v' \leq deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ g),$
$thin\text{-}tac$
$u' \cdot_r{}_{R'}\ erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ g \pm_{R'}$
$v' \cdot_r{}_{R'}\ erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ h =$
$erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ ra,$
$thin\text{-}tac\ erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ u \cdot_r{}_{R'}$
$erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ g =$
$erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ (u \cdot_r g),$
$thin\text{-}tac\ erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ v \cdot_r{}_{R'}$
$erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ h =$
$erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ (v \cdot_r h))$
apply *blast*
apply (*rule-tac* $u = u$ **and** $v = v$ **and** $ra = ra$ **in** *P-mod-diffxxx1*[*of t R' Y f g h*],
assumption+)
apply (*rotate-tac* -12,
drule sym, *drule sym*, *simp*)
done

definition
Hensel-next :: [('*a*, '*b*) *Ring-scheme*, ('*a*, '*c*) *Ring-scheme*, '*a*, '*a*,
('*a set*, '*m*) *Ring-scheme*, '*a set*,'*a*, *nat*] ⇒ ('*a* × '*a*) ⇒ ('*a* × '*a*)
((*9Hen - - - - - - - -*) [67,67,67,67,67,67,67,68]67) **where**

$Hen_{R\ S\ X\ t\ R'\ Y\ f}\ m\ gh = (SOME\ gh1.$
$gh1 \in carrier\ R \times carrier\ R \wedge$
$(deg\ R\ S\ X\ (fst\ gh1) \leq deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ (fst\ gh1))) \wedge$
$P\text{-}mod\ R\ S\ X\ (S\ \diamondsuit_p\ (t\hat{}^S\ m))\ ((fst\ gh) \pm_R -_a R\ (fst\ gh1)) \wedge$
$(deg\ R\ S\ X\ (snd\ gh1) + deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (erH\ R\ S\ X\ R'$
$(S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ (fst\ gh1)) \leq deg\ R\ S\ X\ f) \wedge$
$P\text{-}mod\ R\ S\ X\ (S\ \diamondsuit_p\ (t\hat{}^S\ m))\ ((snd\ gh) \pm_R -_a R\ (snd\ gh1)) \wedge$
$P\text{-}mod\ R\ S\ X\ (S\ \diamondsuit_p\ (t\hat{}^S\ (Suc\ m)))\ (f \pm_R (-_a R\ ((fst\ gh1) \cdot_r R\ (snd\ gh1)))))$

lemma *cart-prod-fst*:$x \in A \times B \Longrightarrow fst\ x \in A$
by *auto*

lemma *cart-prod-snd*:$x \in A \times B \Longrightarrow snd\ x \in B$
by *auto*

lemma *cart-prod-split*:$((x,y) \in A \times B) = (x \in A \wedge y \in B)$
by *auto*

lemma (**in** *PolynRg*) *P-mod-diffxxx3*:⟦*Idomain S*; $t \in carrier\ S$; $t \neq \mathbf{0}_S$;

maximal-ideal S $(S \diamondsuit_p t)$; *PolynRg* R' $(S /_r (S \diamondsuit_p t))$ Y;
$f \in$ *carrier* R; $gh \in$ *carrier* $R \times$ *carrier* R;
deg R S X (*fst* gh) $\leq$ *deg* R' $(S /_r (S \diamondsuit_p t))$ Y (*erH* R S X R'
$(S /_r (S \diamondsuit_p t))$ Y (*pj* S $(S \diamondsuit_p t)$) (*fst* gh));
deg R S X (*snd* gh) + *deg* R' $(S /_r (S \diamondsuit_p t))$ Y (*erH* R S X R'
$(S /_r (S \diamondsuit_p t))$ Y (*pj* S $(S \diamondsuit_p t)$) (*fst* gh)) $\leq$ *deg* R S X f;
$0 <$ *deg* R' $(S /_r (S \diamondsuit_p t))$ Y (*erH* R S X R' $(S /_r (S \diamondsuit_p t))$ Y
(*pj* S $(S \diamondsuit_p t)$) (*fst* gh));
$0 <$ *deg* R' $(S /_r (S \diamondsuit_p t))$ Y
(*erH* R S X R' $(S /_r (S \diamondsuit_p t))$ Y (*pj* S $(S \diamondsuit_p t)$) (*snd* gh));
rel-prime-pols R' $(S /_r (S \diamondsuit_p t))$ Y
(*erH* R S X R' $(S /_r (S \diamondsuit_p t))$ Y (*pj* S $(S \diamondsuit_p t)$) (*fst* gh))
(*erH* R S X R' $(S /_r (S \diamondsuit_p t))$ Y (*pj* S $(S \diamondsuit_p t)$) (*snd* gh));
P-mod R S X $(S \diamondsuit_p (t \hat{}^S m))$ $(f \pm -_a ((fst\ gh) \cdot_r (snd\ gh)))$; $0 < m$$\rrbracket \Longrightarrow$
$\exists\, gh1.\ gh1 \in$*carrier* $R \times$ *carrier* $R \wedge$
(*deg* R S X (*fst* $gh1$) $\leq$ *deg* R' $(S /_r (S \diamondsuit_p t))$ Y
(*erH* R S X R' $(S /_r (S \diamondsuit_p t))$ Y (*pj* S $(S \diamondsuit_p t)$) (*fst* $gh1$))) $\wedge$
P-mod R S X $(S \diamondsuit_p (t \hat{}^S m))$ $((fst\ gh) \pm -_a (fst\ gh1))$ $\wedge$
(*deg* R S X (*snd* $gh1$) + *deg* R' $(S /_r (S \diamondsuit_p t))$ Y
(*erH* R S X R' $(S /_r (S \diamondsuit_p t))$ Y (*pj* S $(S \diamondsuit_p t)$) (*fst* $gh1$)) $\leq$
deg R S X f) $\wedge$
P-mod R S X $(S \diamondsuit_p (t \hat{}^S m))$ $((snd\ gh) \pm -_a (snd\ gh1))$ $\wedge$
P-mod R S X $(S \diamondsuit_p (t \hat{}^S (Suc\ m)))$ $(f \pm (-_a ((fst\ gh1) \cdot_r (snd\ gh1))))$
apply (*cases* gh)
apply (*simp del*: *npow-suc*)
apply (*rename-tac* g h)
apply (*erule conjE*,
frule-tac $g = g$ **and** $h = h$ **and** $f = f$ **in** *P-mod-diffxxx2*[*of* t R' Y],
assumption+)
apply *blast*
done

lemma (**in** *PolynRg*) *P-mod-diffxxx4*:$\llbracket$*Idomain* S; $t \in$ *carrier* S; $t \neq \mathbf{0}_S$;
maximal-ideal S $(S \diamondsuit_p t)$; *PolynRg* R' $(S /_r (S \diamondsuit_p t))$ Y; $f \in$ *carrier* R;
$gh \in$ *carrier* $R \times$ *carrier* R;
deg R S X (*fst* gh) $\leq$ *deg* R' $(S /_r (S \diamondsuit_p t))$ Y
(*erH* R S X R' $(S /_r (S \diamondsuit_p t))$ Y (*pj* S $(S \diamondsuit_p t)$) (*fst* gh));
deg R S X (*snd* gh) + *deg* R' $(S /_r (S \diamondsuit_p t))$ Y (*erH* R S X R'
$(S /_r (S \diamondsuit_p t))$ Y (*pj* S $(S \diamondsuit_p t)$) (*fst* gh)) $\leq$ *deg* R S X f;
$0 <$ *deg* R' $(S /_r (S \diamondsuit_p t))$ Y
(*erH* R S X R' $(S /_r (S \diamondsuit_p t))$ Y (*pj* S $(S \diamondsuit_p t)$) (*fst* gh));
$0 <$ *deg* R' $(S /_r (S \diamondsuit_p t))$ Y
(*erH* R S X R' $(S /_r (S \diamondsuit_p t))$ Y (*pj* S $(S \diamondsuit_p t)$) (*snd* gh));
rel-prime-pols R' $(S /_r (S \diamondsuit_p t))$ Y
(*erH* R S X R' $(S /_r (S \diamondsuit_p t))$ Y (*pj* S $(S \diamondsuit_p t)$) (*fst* gh))
(*erH* R S X R' $(S /_r (S \diamondsuit_p t))$ Y (*pj* S $(S \diamondsuit_p t)$) (*snd* gh));
P-mod R S X $(S \diamondsuit_p (t \hat{}^S m))$ $(f \pm -_a ((fst\ gh) \cdot_r (snd\ gh)))$; $0 < m$$\rrbracket \Longrightarrow$
$(Hen_{R\ S\ X\ t\ R'\ Y\ f}\ m\ gh) \in$ *carrier* $R \times$ *carrier* $R\ \wedge$ (*deg* R S X

$(fst\ (Hen_{R\ S\ X\ t\ R'\ Y\ f}\ m\ gh)) \leq deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))$
$(fst\ (Hen_{R\ S\ X\ t\ R'\ Y\ f}\ m\ gh)))) \wedge$
$P\text{-}mod\ R\ S\ X\ (S\ \diamondsuit_p\ (t\hat{}^S\ m))\ ((fst\ gh) \pm -_a\ (fst\ (Hen_{R\ S\ X\ t\ R'\ Y\ f}\ m\ gh))) \wedge$

$(deg\ R\ S\ X\ (snd\ (Hen_{R\ S\ X\ t\ R'\ Y\ f}\ m\ gh)) + deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))$
$(fst\ (Hen_{R\ S\ X\ t\ R'\ Y\ f}\ m\ gh))) \leq deg\ R\ S\ X\ f) \wedge$
$P\text{-}mod\ R\ S\ X\ (S\ \diamondsuit_p\ (t\hat{}^S\ m))\ ((snd\ gh) \pm -_a\ (snd\ (Hen_{R\ S\ X\ t\ R'\ Y\ f}\ m\ gh)))$
$\wedge$
$P\text{-}mod\ R\ S\ X\ (S\ \diamondsuit_p\ (t\hat{}^S\ (Suc\ m)))\ (f \pm (-_a\ ((fst\ (Hen_{R\ S\ X\ t\ R'\ Y\ f}\ m\ gh))$
$\cdot_r$
$(snd\ (Hen_{R\ S\ X\ t\ R'\ Y\ f}\ m\ gh)))))$
apply (*unfold Hensel-next-def*)
apply (*rule someI2-ex*)
apply (*rule P-mod-diffxxx3*, *assumption*+)
done

primrec
Hensel-pair :: [('*a*, '*b*) *Ring-scheme*, ('*a*, '*c*) *Ring-scheme*, '*a*, '*a*,
('*a set*, '*m*) *Ring-scheme*, '*a set*, '*a*, '*a*, '*a*, *nat*] ⇒ ('*a* × '*a*)
((*10Hpr* - - - - - - - - - -) [67,67,67,67,67,67,67,67,67,68]67)
where
Hpr-0: $Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ 0 = (g, h)$
| *Hpr-Suc*: $Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ (Suc\ m) =$
$Hen_{R\ S\ X\ t\ R'\ Y\ f}\ (Suc\ m)\ (Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ m)$

lemma (**in** *PolynRg*) *fst-xxx*: ⟦$t \in carrier\ S$; $t \neq \mathbf{0}_S$; *ideal S* $(S\ \diamondsuit_p\ t)$;
$\forall (n::nat).\ (F\ n) \in carrier\ R \times carrier\ R$;
$\forall m.\ P\text{-}mod\ R\ S\ X\ (S\ \diamondsuit_p\ t)\ (fst\ (F\ m) \pm -_a\ (fst\ (F\ (Suc\ m))))$⟧ ⟹
$P\text{-}mod\ R\ S\ X\ (S\ \diamondsuit_p\ t)\ (fst\ (F\ 0) \pm -_a\ (fst\ (F\ n)))$
apply (*cut-tac subring*, *frule subring-Ring*,
cut-tac ring-is-ag)
apply (*induct-tac n*)
apply (*drule-tac x = 0* **in** *spec*)
apply (*frule cart-prod-fst*[*of F 0 carrier R carrier R*])
apply (*simp add*:*aGroup.ag-r-inv1*) **apply** (*simp add*:*P-mod-def*)

apply (*frule-tac x = 0* **in** *spec*,
frule-tac x = n **in** *spec*,
drule-tac x = Suc n **in** *spec*)

apply (*frule-tac x = F 0* **in** *cart-prod-fst*[*of - carrier R carrier R*],
frule-tac x = F n **in** *cart-prod-fst*[*of - carrier R carrier R*],
frule-tac x = F (*Suc n*) **in** *cart-prod-fst*[*of - carrier R*
carrier R])

apply (*drule-tac* $x = n$ **in** *spec*)
apply (*frule-tac* $p = fst\ (F\ 0) \pm -_a\ (fst\ (F\ n))$ **and**
$q = fst\ (F\ n) \pm -_a\ (fst\ (F\ (Suc\ n)))$ **in**
P-mod-add[*of* $S \diamondsuit_p t$])
apply (*rule aGroup.ag-pOp-closed*, *assumption*+, *rule aGroup.ag-mOp-closed*,
assumption+)+
apply (*frule-tac* $x = fst\ (F\ n)$ **in** *aGroup.ag-mOp-closed*, *assumption*+,
frule-tac $x = fst\ (F\ (Suc\ n))$ **in** *aGroup.ag-mOp-closed*, *assumption*+)
apply (*simp add*:*aGroup.pOp-assocTr41*[*of R*, *THEN sym*],
simp add:*aGroup.pOp-assocTr42*[*of R*],
simp add:*aGroup.ag-l-inv1*,
simp add:*aGroup.ag-r-zero*)
done

lemma (**in** *PolynRg*) *snd-xxx*:⟦$t \in carrier\ S$; $t \neq \mathbf{0}_S$;
$ideal\ S\ (S \diamondsuit_p t)$; $\forall (n::nat).\ (F\ n) \in carrier\ R \times carrier\ R$;
$\forall m.\ P\text{-}mod\ R\ S\ X\ (S \diamondsuit_p t)\ (snd\ (F\ m) \pm -_a\ (snd\ (F\ (Suc\ m))))$⟧ ⟹
$P\text{-}mod\ R\ S\ X\ (S \diamondsuit_p t)\ (snd\ (F\ 0) \pm -_a\ (snd\ (F\ n)))$
apply (*cut-tac subring*, *frule subring-Ring*,
cut-tac ring-is-ag)
apply (*induct-tac n*)
apply (*drule-tac* $x = 0$ **in** *spec*)
apply (*frule cart-prod-snd*[*of F 0 carrier R carrier R*])
apply (*simp add*:*aGroup.ag-r-inv1*) **apply** (*simp add*:*P-mod-def*)

apply (*frule-tac* $x = 0$ **in** *spec*,
frule-tac $x = n$ **in** *spec*,
drule-tac $x = Suc\ n$ **in** *spec*)

apply (*frule-tac* $x = F\ 0$ **in** *cart-prod-snd*[*of - carrier R carrier R*],
frule-tac $x = F\ n$ **in** *cart-prod-snd*[*of - carrier R carrier R*],
frule-tac $x = F\ (Suc\ n)$ **in** *cart-prod-snd*[*of - carrier R*
carrier R])
apply (*drule-tac* $x = n$ **in** *spec*)
apply (*frule-tac* $p = snd\ (F\ 0) \pm -_a\ (snd\ (F\ n))$ **and**
$q = snd\ (F\ n) \pm -_a\ (snd\ (F\ (Suc\ n)))$ **in**
P-mod-add[*of* $S \diamondsuit_p t$])
apply (*rule aGroup.ag-pOp-closed*, *assumption*+, *rule aGroup.ag-mOp-closed*,
assumption+)+
apply (*frule-tac* $x = snd\ (F\ n)$ **in** *aGroup.ag-mOp-closed*, *assumption*+,
frule-tac $x = snd\ (F\ (Suc\ n))$ **in** *aGroup.ag-mOp-closed*, *assumption*+)
apply (*simp add*:*aGroup.pOp-assocTr41*[*of R*, *THEN sym*],
simp add:*aGroup.pOp-assocTr42*[*of R*],
simp add:*aGroup.ag-l-inv1*,
simp add:*aGroup.ag-r-zero*)
done

lemma (**in** *PolynRg*) *P-mod-diffxxx5*:⟦*Idomain S*; $t \in carrier\ S$; $t \neq \mathbf{0}_S$;
$maximal\text{-}ideal\ S\ (S \diamondsuit_p t)$; $PolynRg\ R'\ (S /_r (S \diamondsuit_p t))\ Y$;

$f \in carrier\ R$; $(g,\ h) \in carrier\ R \times carrier\ R$;
$deg\ R\ S\ X\ (fst\ (g,\ h)) \leq deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ (fst\ (g,\ h)))$;
$deg\ R\ S\ X\ (snd\ (g,\ h)) + deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ (fst\ (g,\ h))) \leq deg\ R\ S\ X\ f$;
$0 < deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$(pj\ S\ (S\ \diamondsuit_p\ t))\ (fst\ (g,\ h)))$;
$0 < deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$(pj\ S\ (S\ \diamondsuit_p\ t))\ (snd\ (g,\ h)))$;
$rel\text{-}prime\text{-}pols\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ (fst\ (g,\ h)))$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ (snd\ (g,\ h)))$;
$P\text{-}mod\ R\ S\ X\ (S\ \diamondsuit_p\ t)\ (f \pm -_a\ (g \cdot_r h))]\!] \Longrightarrow$
$(Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ (Suc\ m)) \in carrier\ R \times carrier\ R\ \wedge$
$erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))$
$(fst\ (Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ (Suc\ m))) =$
$erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ (fst\ (g,\ h)) \wedge$
$erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))$
$(snd\ (Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ (Suc\ m))) =$
$erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ (snd\ (g,\ h)) \wedge$
$(deg\ R\ S\ X\ (fst\ (Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ (Suc\ m))) \leq deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))$
$(fst\ (Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ (Suc\ m)))) \wedge$
$P\text{-}mod\ R\ S\ X\ (S\ \diamondsuit_p\ (t\hat{}^S\ (Suc\ m)))\ ((fst\ (Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ m)) \pm -_a$
$(fst\ (Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ (Suc\ m)))) \wedge$
$(deg\ R\ S\ X\ (snd\ (Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ (Suc\ m))) + deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ (fst\ (Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}$
$(Suc\ m)))) \leq\ deg\ R\ S\ X\ f) \wedge$
$P\text{-}mod\ R\ S\ X\ (S\ \diamondsuit_p\ (t\hat{}^S\ (Suc\ m)))\ ((snd\ (Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ m)) \pm -_a\ (snd$
$(Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ (Suc\ m)))) \wedge$
$P\text{-}mod\ R\ S\ X\ (S\ \diamondsuit_p\ (t\hat{}^S\ (Suc\ (Suc\ m))))\ (f \pm -_a\ ((fst\ (Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}$
$(Suc\ m))) \cdot_r (snd\ (Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ (Suc\ m)))))$
apply ($cut\text{-}tac\ subring$, $frule\ subring\text{-}Ring$,
$cut\text{-}tac\ ring\text{-}is\text{-}ag$,
$frule\ mem\text{-}subring\text{-}mem\text{-}ring[of\ S\ t]$, $assumption+$,
$frule\ Ring.maximal\text{-}ideal\text{-}ideal[of\ S\ S\ \diamondsuit_p\ t]$, $assumption+$)
apply ($induct\text{-}tac\ m$)
apply ($simp\ del$:$Hpr\text{-}0\ npow\text{-}suc$)
apply ($simp\ only$:$Hpr\text{-}0$)
apply ($frule\ P\text{-}mod\text{-}diffxxx4[of\ t\ R'\ Y\ f\ (g,\ h)\ Suc\ 0]$,
$assumption+$)
apply ($simp\ add$:$cart\text{-}prod\text{-}split$, $simp+$)
apply ($simp\ add$:$Ring.ring\text{-}l\text{-}one$, $simp$)
apply ($simp\ add$:$Ring.ring\text{-}l\text{-}one$, $(erule\ conjE)+$)
apply ($frule\ P\text{-}mod\text{-}diff[THEN\ sym,\ of\ S\ \diamondsuit_p\ t\ R'\ Y\ g$
$fst\ (Hen_{R\ S\ X\ t\ R'\ Y\ f}\ (Suc\ 0)\ (g,\ h))]$, $assumption+$,

simp add:*cart-prod-fst*, *rotate-tac* -1, *drule sym*, *simp*)
apply (*frule P-mod-diff*[*THEN sym*, *of* $S \diamondsuit_p t\ R'\ Y\ h$
$snd\ (Hen_{R\ S\ X\ t\ R'\ Y\ f}\ (Suc\ 0)\ (g, h))$], *assumption*+,
simp add:*cart-prod-snd*, *rotate-tac* -1, *drule sym*, *simp*)

apply ((*erule conjE*)+, *rename-tac m*)
apply (*frule-tac* $m = Suc\ (Suc\ m)$ **and** $gh = Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ (Suc\ m)$ **in**
P-mod-diffxxx4[*of t R' Y f*], *assumption*+)
apply (*simp*, *simp*, *simp*, *simp del*:*npow-suc*, *simp*)
apply (*erule conjE*)+
apply (*simp del*:*npow-suc del*:*Hpr-Suc*
add:*Hpr-Suc*[*THEN sym*, *of R S X t R' Y f - g h*])
apply (*thin-tac deg R S X g*
$\leq deg\ R'\ (S /_r (S \diamondsuit_p t))\ Y$
$(erH\ R\ S\ X\ R'\ (S /_r (S \diamondsuit_p t))\ Y\ (pj\ S\ (S \diamondsuit_p t))\ g)$,
thin-tac deg R S X h +
$deg\ R'\ (S /_r (S \diamondsuit_p t))\ Y$
$(erH\ R\ S\ X\ R'\ (S /_r (S \diamondsuit_p t))\ Y\ (pj\ S\ (S \diamondsuit_p t))\ g)$
$\leq deg\ R\ S\ X\ f$,
thin-tac $0 < deg\ R'\ (S /_r (S \diamondsuit_p t))\ Y$
$(erH\ R\ S\ X\ R'\ (S /_r (S \diamondsuit_p t))\ Y\ (pj\ S\ (S \diamondsuit_p t))\ g)$,
thin-tac $0 < deg\ R'\ (S /_r (S \diamondsuit_p t))\ Y$
$(erH\ R\ S\ X\ R'\ (S /_r (S \diamondsuit_p t))\ Y\ (pj\ S\ (S \diamondsuit_p t))\ h)$,
thin-tac rel-prime-pols $R'\ (S /_r (S \diamondsuit_p t))\ Y$
$(erH\ R\ S\ X\ R'\ (S /_r (S \diamondsuit_p t))\ Y\ (pj\ S\ (S \diamondsuit_p t))\ g)$
$(erH\ R\ S\ X\ R'\ (S /_r (S \diamondsuit_p t))\ Y\ (pj\ S\ (S \diamondsuit_p t))\ h)$,
thin-tac deg R S X $(fst\ (Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ Suc\ m))$
$\leq deg\ R'\ (S /_r (S \diamondsuit_p t))\ Y$
$(erH\ R\ S\ X\ R'\ (S /_r (S \diamondsuit_p t))\ Y\ (pj\ S\ (S \diamondsuit_p t))\ g)$,
thin-tac deg R S X $(snd\ (Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ Suc\ m))$ +
$deg\ R'\ (S /_r (S \diamondsuit_p t))\ Y$
$(erH\ R\ S\ X\ R'\ (S /_r (S \diamondsuit_p t))\ Y\ (pj\ S\ (S \diamondsuit_p t))\ g)$
$\leq deg\ R\ S\ X\ f$,
thin-tac deg R S X $(fst\ (Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ Suc\ (Suc\ m)))$
$\leq deg\ R'\ (S /_r (S \diamondsuit_p t))\ Y$
$(erH\ R\ S\ X\ R'\ (S /_r (S \diamondsuit_p t))\ Y\ (pj\ S\ (S \diamondsuit_p t))$
$(fst\ (Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ Suc\ (Suc\ m))))$,
thin-tac deg R S X $(snd\ (Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ Suc\ (Suc\ m)))$ +
$deg\ R'\ (S /_r (S \diamondsuit_p t))\ Y$
$(erH\ R\ S\ X\ R'\ (S /_r (S \diamondsuit_p t))\ Y\ (pj\ S\ (S \diamondsuit_p t))$
$(fst\ (Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ Suc\ (Suc\ m))))$
$\leq deg\ R\ S\ X\ f$)
apply (*frule-tac* $g = fst\ (Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ (Suc\ m)) \pm -_a\ (fst$
$(Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ (Suc\ (Suc\ m))))$ **and** $n = Suc\ m$ **in** *P-mod-n-1*
[*of t*], *assumption*+,
(*rule aGroup.ag-pOp-closed*, *assumption*+, *simp add*:*cart-prod-fst*,
rule aGroup.ag-mOp-closed, *assumption*+, *simp add*:*cart-prod-fst*,
assumption),

(*frule-tac* x = *Hpr* $_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}$ *Suc m* **in** *cart-prod-fst*[*of* - *carrier R carrier R*],
frule-tac x = *Hpr* $_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}$ (*Suc* (*Suc m*)) **in** *cart-prod-fst*[*of* - *carrier R carrier R*]),
(*frule-tac g1* = *fst* (*Hpr* $_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}$ (*Suc m*)) **and**
h1 = *fst* (*Hpr* $_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}$ (*Suc* (*Suc m*))) **in**
P-mod-diff[*THEN sym*, *of S* $\diamondsuit_p$ *t R' Y*], *assumption*+))
apply (*frule-tac g* = *snd* (*Hpr* $_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}$ (*Suc m*)) $\pm$ $-_a$ (*snd* (*Hpr* $_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}$ (*Suc* (*Suc m*)))) **and** n = *Suc m* **in** *P-mod-n-1*[*of t*], *assumption*+,
(*rule aGroup.ag-pOp-closed*, *assumption*+, *simp add*:*cart-prod-snd*,
rule aGroup.ag-mOp-closed, *assumption*+, *simp add*:*cart-prod-snd*,
assumption),
(*frule-tac* x = *Hpr* $_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}$ *Suc m* **in** *cart-prod-snd*[*of* - *carrier R carrier R*],
frule-tac x = *Hpr* $_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}$ (*Suc* (*Suc m*)) **in** *cart-prod-snd*[*of* - *carrier R carrier R*]),
(*frule-tac g1* = *snd* (*Hpr* $_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}$ (*Suc m*)) **and**
h1 = *snd* (*Hpr* $_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}$ (*Suc* (*Suc m*))) **in**
P-mod-diff[*THEN sym*, *of S* $\diamondsuit_p$ *t R' Y*], *assumption*+))
apply *simp*
done

lemma (**in** *PolynRg*) *P-mod-diffxxx5-1*:⟦*Idomain S*; $t \in$ *carrier S*; $t \neq \mathbf{0}_S$;
maximal-ideal S (*S* $\diamondsuit_p$ *t*); *PolynRg R'* (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y*;
$f \in$ *carrier R*; $g \in$ *carrier R*; $h \in$ *carrier R*;
deg R S X g $\leq$ *deg R'* (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y*
(*erH R S X R'* (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y* (*pj S* (*S* $\diamondsuit_p$ *t*)) *g*);
deg R S X h + *deg R'* (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y* (*erH R S X R'*
(*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y* (*pj S* (*S* $\diamondsuit_p$ *t*)) *g*) $\leq$ *deg R S X f*;
0 $<$ *deg R'* (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y*
(*erH R S X R'* (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y* (*pj S* (*S* $\diamondsuit_p$ *t*)) *g*);
0 $<$ *deg R'* (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y*
(*erH R S X R'* (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y* (*pj S* (*S* $\diamondsuit_p$ *t*)) *h*);
rel-prime-pols R' (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y*
(*erH R S X R'* (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y* (*pj S* (*S* $\diamondsuit_p$ *t*)) *g*)
(*erH R S X R'* (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y* (*pj S* (*S* $\diamondsuit_p$ *t*)) *h*);
P-mod R S X (*S* $\diamondsuit_p$ *t*) (*f* $\pm$ $-_a$ (*g* $\cdot_r$ *h*))⟧ $\Longrightarrow$
(*Hpr* $_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}$ (*Suc m*)) $\in$ *carrier R* $\times$ *carrier R* $\wedge$
erH R S X R' (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y* (*pj S* (*S* $\diamondsuit_p$ *t*))
(*fst* (*Hpr* $_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}$ (*Suc m*))) =
erH R S X R' (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y* (*pj S* (*S* $\diamondsuit_p$ *t*)) (*fst* (*g*, *h*)) $\wedge$
erH R S X R' (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y* (*pj S* (*S* $\diamondsuit_p$ *t*))
(*snd* (*Hpr* $_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}$ (*Suc m*))) =
erH R S X R' (*S* $/_r$ (*S* $\diamondsuit_p$ *t*)) *Y* (*pj S* (*S* $\diamondsuit_p$ *t*)) (*snd* (*g*, *h*)) $\wedge$
(*deg R S X* (*fst* (*Hpr* $_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}$ (*Suc m*))) $\leq$ *deg R'*

$(S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$(pj\ S\ (S\ \diamondsuit_p\ t))\ (fst\ (Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ (Suc\ m)))))\ \wedge$
$P\text{-}mod\ R\ S\ X\ (S\ \diamondsuit_p\ (t\hat{}^S\ (Suc\ m)))\ ((fst\ (Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ m))\ \pm\ -_a$
$(fst\ (Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ (Suc\ m))))\ \wedge$
$(deg\ R\ S\ X\ (snd\ (Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ (Suc\ m)))\ +$
$deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$(pj\ S\ (S\ \diamondsuit_p\ t))\ (fst\ (Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ (Suc\ m))))\ \leq\ deg\ R\ S\ X\ f)\ \wedge$
$P\text{-}mod\ R\ S\ X\ (S\ \diamondsuit_p\ (t\hat{}^S\ (Suc\ m)))\ ((snd\ (Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ m))\ \pm\ -_a$
$(snd\ (Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ (Suc\ m))))\ \wedge$
$P\text{-}mod\ R\ S\ X\ (S\ \diamondsuit_p\ (t\hat{}^S\ (Suc\ (Suc\ m))))\ (f\ \pm\ -_a$
$((fst\ (Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ (Suc\ m)))\ \cdot_r\ (snd\ (Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ (Suc\ m)))))$
apply (*frule P-mod-diffxxx5*[*of t R' Y f g h m*], *assumption*+)
apply (*simp add*:*cart-prod-split*, *simp*, *simp*, *simp*, *simp*, *simp*, *assumption*+)
done

lemma (**in** *PolynRg*) *P-mod-diffxxx5-2*:⟦*Idomain S*; $t \in carrier\ S$; $t \neq \mathbf{0}_S$;
maximal-ideal S $(S\ \diamondsuit_p\ t)$; *PolynRg* $R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$; $f \in carrier\ R$;
$g \in carrier\ R$; $h \in carrier\ R$;
$deg\ R\ S\ X\ g \leq deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ g)$;
$deg\ R\ S\ X\ h + deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (erH\ R\ S\ X\ R'$
$(S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ g) \leq deg\ R\ S\ X\ f$;
$0 < deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ g)$;
$0 < deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ h)$;
rel-prime-pols $R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ g)$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ h)$;
$P\text{-}mod\ R\ S\ X\ (S\ \diamondsuit_p\ t)\ (f\ \pm\ -_a\ (g\ \cdot_r\ h))$⟧ ⟹
$(Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ m) \in carrier\ R \times carrier\ R$
apply (*case-tac m = 0*, *simp*, *simp*)
apply (*frule P-mod-diffxxx5-1*[*of t R' Y f g h*
$m - Suc\ 0$], *assumption*+) **apply** (*erule conjE*)+
apply *simp*
done

lemma (**in** *PolynRg*) *P-mod-diffxxx5-3*:⟦*Idomain S*; $t \in carrier\ S$; $t \neq \mathbf{0}_S$;
maximal-ideal S $(S\ \diamondsuit_p\ t)$; *PolynRg* $R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$; $f \in carrier\ R$;
$g \in carrier\ R$; $h \in carrier\ R$;
$deg\ R\ S\ X\ g \leq deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ g)$;
$deg\ R\ S\ X\ h + deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (erH\ R\ S\ X\ R'$
$(S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ g) \leq deg\ R\ S\ X\ f$;
$0 < deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$

$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ g);$
$0 < deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ h);$
$rel\text{-}prime\text{-}pols\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ g)$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ h);$
$P\text{-}mod\ R\ S\ X\ (S\ \diamondsuit_p\ t)\ (f \pm -_a\ (g \cdot_r h))]\!] \Longrightarrow$
$P\text{-}mod\ R\ S\ X\ (S\ \diamondsuit_p\ (t\hat{}^S\ m))\ ((fst\ (Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ m)) \pm$
$-_a\ (fst\ (Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ (m + n)))) \wedge$
$P\text{-}mod\ R\ S\ X\ (S\ \diamondsuit_p\ (t\hat{}^S\ m))\ ((snd\ (Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ m)) \pm$
$-_a\ (snd\ (Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ (m + n))))$

apply (*cut-tac ring-is-ag*,
cut-tac subring, *frule subring-Ring*)
apply (*induct-tac n*)
apply (*simp del*:*npow-suc Hpr-Suc*)
apply (*frule P-mod-diffxxx5-2*[*of t R' Y f g h m*], *assumption*+)
apply (*frule cart-prod-fst*[*of Hpr* $_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}$ *m carrier R carrier R*],
frule cart-prod-snd[*of Hpr* $_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}$ *m carrier R carrier R*])
apply (*simp add*:*aGroup.ag-r-inv1*, *simp add*:*P-mod-def*)

apply (*frule-tac m = m + n* **in** *P-mod-diffxxx5-1*[*of t R' Y f g h*],
assumption+, (*erule conjE*)+)
apply (*frule-tac m = m* **in** *P-mod-diffxxx5-2*[*of t R' Y f g h*], *assumption*+)
apply (*frule-tac m = m + n* **in** *P-mod-diffxxx5-2*[*of t R' Y f g h*], *assumption*+)
apply (*thin-tac deg R S X g*
$\leq deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ g)$,
thin-tac deg R S X h + $deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ g) \leq deg\ R\ S\ X\ f$,
thin-tac 0 $< deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ g)$,
thin-tac 0 $< deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ h)$,
thin-tac rel-prime-pols $R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ g)$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ h)$,
thin-tac P-mod R S X $(S\ \diamondsuit_p\ t)\ (f \pm -_a\ (g \cdot_r h))$,
thin-tac erH R S X $R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))$
$(fst\ (Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ Suc\ (m + n))) =$
$erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))\ (fst\ (g, h))$,
thin-tac erH R S X $R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))$
$(snd\ (Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ Suc\ (m + n))) =$
$erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))$
$(snd\ (g, h))$,
thin-tac deg R S X $(fst\ (Hpr_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ Suc\ (m + n)))$
$\leq deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y$
$(erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))$

$(fst\ (Hpr_{\ R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ Suc\ (m + n))))$,
thin-tac deg R S X $(snd\ (Hpr_{\ R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ Suc\ (m + n))) +$
$deg\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (erH\ R\ S\ X\ R'\ (S\ /_r\ (S\ \diamondsuit_p\ t))\ Y\ (pj\ S\ (S\ \diamondsuit_p\ t))$
$(fst\ (Hpr_{\ R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ Suc\ (m + n)))) \leq deg\ R\ S\ X\ f$,
thin-tac P-mod R S X $(S\ \diamondsuit_p\ (t\hat{}^S\ (Suc\ (Suc\ (m + n)))))$
$(f\ \pm\ -_a\ (fst\ (Hpr_{\ R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ Suc\ (m + n))\ \cdot_r$
$snd\ (Hpr_{\ R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ Suc\ (m + n)))))$
apply (*simp del*:*npow-suc Hpr-Suc*)
apply (*frule-tac x* $= Hpr_{\ R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ m$ **in**
cart-prod-fst[*of - carrier R carrier R*],
frule-tac x $= Hpr_{\ R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ (m + n)$ **in**
cart-prod-fst[*of - carrier R carrier R*],
frule-tac x $= Hpr_{\ R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ (Suc\ (m + n))$ **in**
cart-prod-fst[*of - carrier R carrier R*],
frule-tac x $= Hpr_{\ R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ m$ **in**
cart-prod-snd[*of - carrier R carrier R*],
frule-tac x $= Hpr_{\ R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ (m + n)$ **in**
cart-prod-snd[*of - carrier R carrier R*],
frule-tac x $= Hpr_{\ R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ (Suc\ (m + n))$ **in**
cart-prod-snd[*of - carrier R carrier R*])
apply (*case-tac m* $= 0$, *simp del*:*npow-suc Hpr-Suc*)
apply (*simp only*:*Ring.Rxa-one*)
apply (*rule conjI*)
apply (*rule-tac p* $= g \pm -_a\ (fst\ (Hpr_{\ R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ (Suc\ n)))$ **in**
P-mod-whole,
rule aGroup.ag-pOp-closed, *assumption+*,
rule aGroup.ag-mOp-closed, *assumption+*)
apply (*rule-tac p* $= h \pm -_a\ (snd\ (Hpr_{\ R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ (Suc\ n)))$ **in**
P-mod-whole,
rule aGroup.ag-pOp-closed, *assumption+*,
rule aGroup.ag-mOp-closed, *assumption+*)
apply (*frule-tac g* $= fst\ (Hpr_{\ R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ (m + n)) \pm$
$-_a\ (fst\ (Hpr_{\ R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ Suc\ (m + n)))$ **and** $n = m + n$ **in**
P-mod-n-m[*of t - m − Suc 0*], *assumption+*)
apply (*rule aGroup.ag-pOp-closed*, *assumption+*, *rule aGroup.ag-mOp-closed*,
assumption+,
arith,
simp del:*npow-suc Hpr-Suc*, *simp del*:*npow-suc Hpr-Suc*)
apply (*frule Ring.npClose*[*of S t m*], *assumption*,
frule Ring.principal-ideal[*of S* $t\hat{}^S\ m$], *assumption*)
apply (*frule-tac p* $= fst\ (Hpr_{\ R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ m) \pm$
$-_a\ (fst\ (Hpr_{\ R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ (m + n)))$ **and**
$q = fst\ (Hpr_{\ R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ (m + n)) \pm$
$-_a\ (fst\ (Hpr_{\ R\ S\ X\ t\ R'\ Y\ f\ g\ h}\ Suc\ (m + n)))$ **in**
P-mod-add[*of S* $\diamondsuit_p\ (t\hat{}^S\ m)$],
(*rule aGroup.ag-pOp-closed*, *assumption+*,

```
        rule aGroup.ag-mOp-closed, assumption+)+,
    simp del:npow-suc Hpr-Suc add:aGroup.pOp-assoc-cancel)
apply (frule-tac g = snd (Hpr R S X t R' Y f g h (m + n)) ±
   −a (snd (Hpr R S X t R' Y f g h Suc (m + n))) and n = m + n in
   P-mod-n-m[of t - m − Suc 0], assumption+)
apply (rule aGroup.ag-pOp-closed, assumption+, rule aGroup.ag-mOp-closed,
     assumption+,
     arith,
     simp del:npow-suc Hpr-Suc, simp del:npow-suc Hpr-Suc)
apply (frule-tac p = snd (Hpr R S X t R' Y f g h m) ±
        −a (snd (Hpr R S X t R' Y f g h (m + n))) and
     q = snd (Hpr R S X t R' Y f g h (m + n)) ±
        −a (snd (Hpr R S X t R' Y f g h Suc (m + n))) in
     P-mod-add[of S ♢p (t ^S m)],
    (rule aGroup.ag-pOp-closed, assumption+,
        rule aGroup.ag-mOp-closed, assumption+)+,
    simp del:npow-suc Hpr-Suc add:aGroup.pOp-assoc-cancel)
done


lemma (in PolynRg) P-mod-diffxxx5-4:⟦Idomain S; t ∈ carrier S; t ≠ 0S;
     maximal-ideal S (S ♢p t); PolynRg R' (S /r (S ♢p t)) Y; f ∈ carrier R;
  g ∈ carrier R; h ∈ carrier R;
  deg R S X g ≤ deg R' (S /r (S ♢p t)) Y
   (erH R S X R' (S /r (S ♢p t)) Y (pj S (S ♢p t)) g);
  deg R S X h + deg R' (S /r (S ♢p t)) Y (erH R S X R'
            (S /r (S ♢p t)) Y (pj S (S ♢p t)) g) ≤ deg R S X f;
    0 < deg R' (S /r (S ♢p t)) Y
      (erH R S X R' (S /r (S ♢p t)) Y (pj S (S ♢p t)) g);
    0 < deg R' (S /r (S ♢p t)) Y
      (erH R S X R' (S /r (S ♢p t)) Y (pj S (S ♢p t)) h);
  rel-prime-pols R' (S /r (S ♢p t)) Y
        (erH R S X R' (S /r (S ♢p t)) Y (pj S (S ♢p t)) g)
        (erH R S X R' (S /r (S ♢p t)) Y (pj S (S ♢p t)) h);
  P-mod R S X (S ♢p t) (f ± −a (g ·r h))⟧ ⟹
     deg R S X (fst (Hpr R S X t R' Y f g h m)) ≤ deg R S X g ∧
     deg R S X (snd (Hpr R S X t R' Y f g h m)) ≤ deg R S X f
apply (cut-tac subring, frule subring-Ring,
     frule Ring.maximal-ideal-ideal[of S S ♢p t], assumption)
apply (case-tac m = 0) apply simp
apply (frule aless-imp-le[of 0 deg R' (S /r (S ♢p t)) Y
       (erH R S X R' (S /r (S ♢p t)) Y (pj S (S ♢p t)) g)])
apply (frule aadd-le-mono[of 0 deg R' (S /r (S ♢p t)) Y
     (erH R S X R' (S /r (S ♢p t)) Y (pj S (S ♢p t)) g) deg R S X h])
 apply (simp add:aadd-0-l, simp add:aadd-commute[of  - deg R S X h])

 apply (frule P-mod-diffxxx5-1[of t R' Y f g h m − Suc 0],
      assumption+, (erule conjE)+)
```

apply (*thin-tac deg R S X g*
$\leq$ *deg R′ (S /r (S ◇p t)) Y*
(*erH R S X R′ (S /r (S ◇p t)) Y (pj S (S ◇p t)) g*))
apply (*thin-tac deg R S X h* + *deg R′ (S /r (S ◇p t)) Y*
(*erH R S X R′ (S /r (S ◇p t)) Y (pj S (S ◇p t)) g*) $\leq$ *deg R S X f*)
apply (*thin-tac rel-prime-pols R′ (S /r (S ◇p t)) Y*
(*erH R S X R′ (S /r (S ◇p t)) Y (pj S (S ◇p t)) g*)
(*erH R S X R′ (S /r (S ◇p t)) Y (pj S (S ◇p t)) h*),
thin-tac P-mod R S X (S ◇p t) (f ± −*a* (*g* ·*r h*)),
thin-tac P-mod R S X (S ◇p (*t*^*S* (*Suc* (*m* − *Suc 0*))))
(*fst* (*Hpr R S X t R′ Y f g h* (*m* − *Suc 0*)) ±
−*a* (*fst* (*Hpr R S X t R′ Y f g h Suc* (*m* − *Suc 0*)))),
thin-tac P-mod R S X (S ◇p (*t*^*S* (*Suc* (*m* − *Suc 0*))))
(*snd* (*Hpr R S X t R′ Y f g h* (*m* − *Suc 0*)) ±
−*a* (*snd* (*Hpr R S X t R′ Y f g h Suc* (*m* − *Suc 0*)))),
thin-tac P-mod R S X (S ◇p (*t*^*S* (*Suc* (*Suc* (*m* − *Suc 0*)))))
(*f* ±
−*a* (*fst* (*Hpr R S X t R′ Y f g h Suc* (*m* − *Suc 0*)) ·*r*
snd (*Hpr R S X t R′ Y f g h Suc* (*m* − *Suc 0*)))))
apply (*simp del*:*npow-suc Hpr-Suc*)
apply (*thin-tac erH R S X R′ (S /r (S ◇p t)) Y (pj S (S ◇p t))*
(*fst* (*Hpr R S X t R′ Y f g h m*)) =
erH R S X R′ (S /r (S ◇p t)) Y (pj S (S ◇p t)) g,
thin-tac erH R S X R′ (S /r (S ◇p t)) Y (pj S (S ◇p t))
(*snd* (*Hpr R S X t R′ Y f g h m*)) =
erH R S X R′ (S /r (S ◇p t)) Y (pj S (S ◇p t)) h)
apply (*frule-tac p* = *g* **in** *pHom-dec-deg*[*of R′ (S /r (S ◇p t)) Y*
erH R S X R′ (S /r (S ◇p t)) Y (pj S (S ◇p t))])
apply (*frule Ring.qring-ring*[*of S S ◇p t*], *assumption*+)
apply (*rule erH-rHom*[*of R′ (S /r (S ◇p t)) Y pj S (S ◇p t)*], *assumption*+,
simp add:*pj-Hom*, *assumption*+)
apply (*frule ale-trans*[*of deg R S X* (*fst* (*Hpr R S X t R′ Y f g h m*))
deg R′ (S /r (S ◇p t)) Y
(*erH R S X R′ (S /r (S ◇p t)) Y (pj S (S ◇p t)) g*) *deg R S X g*],
assumption+)
apply *simp*

apply (*thin-tac deg R S X* (*fst* (*Hpr R S X t R′ Y f g h m*))
$\leq$ *deg R′ (S /r (S ◇p t)) Y*
(*erH R S X R′ (S /r (S ◇p t)) Y (pj S (S ◇p t)) g*),
thin-tac deg R′ (S /r (S ◇p t)) Y
(*erH R S X R′ (S /r (S ◇p t)) Y (pj S (S ◇p t)) g*) $\leq$ *deg R S X g*,
thin-tac deg R S X (*fst* (*Hpr R S X t R′ Y f g h m*)) $\leq$ *deg R S X g*)

apply (*frule aless-imp-le*[*of 0 deg R′ (S /r (S ◇p t)) Y*
(*erH R S X R′ (S /r (S ◇p t)) Y (pj S (S ◇p t)) g*)])

apply (*frule aadd-le-mono*[*of 0 deg* R' (S $/_r$ (S $\diamondsuit_p$ t)) Y
(erH R S X R' (S $/_r$ (S $\diamondsuit_p$ t)) Y (pj S (S $\diamondsuit_p$ t)) g)
deg R S X (*snd* (*Hpr* $_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}$ m))])
apply (*simp add*:*aadd-0-l*, *simp add*:*aadd-commute*[*of* -
deg R S X (*snd* (*Hpr* $_{R\ S\ X\ t\ R'\ Y\ f\ g\ h}$ m))])
done

declare *max.absorb1* [*simp del*] *max.absorb2* [*simp del*]

end

theory *Algebra7* **imports** *Algebra6* **begin**

Chapter 5

Modules

5.1 Basic properties of Modules

record $('a, 'b)$ *Module* $= 'a$ *aGroup* $+$
  sprod $:: 'b \Rightarrow 'a \Rightarrow 'a$ (**infixl** $\cdot_{s1}$ *76*)

locale *Module* $=$ *aGroup* M **for** M (**structure**) $+$
  fixes R (**structure**)
  assumes *sc-Ring*: *Ring* R
  and *sprod-closed* :
    $[\![a \in carrier\ R;\ m \in carrier\ M]\!] \Longrightarrow a \cdot_s m \in carrier\ M$
   and *sprod-l-distr*:
    $[\![a \in carrier\ R;\ b \in carrier\ R;\ m \in carrier\ M]\!] \Longrightarrow$
     $(a \pm_R b) \cdot_s m = a \cdot_s m \pm_M b \cdot_s m$
   and *sprod-r-distr*:
    $[\![a \in carrier\ R;\ m \in carrier\ M;\ n \in carrier\ M\ ]\!] \Longrightarrow$
    $a \cdot_s (m \pm_M n) = a \cdot_s m \pm_M a \cdot_s n$
   and *sprod-assoc*:
    $[\![a \in carrier\ R;\ b \in carrier\ R;\ m \in carrier\ M\ ]\!] \Longrightarrow$
    $(a \cdot_{r R} b) \cdot_s m = a \cdot_s (b \cdot_s m)$
   and *sprod-one*:
    $m \in carrier\ M \Longrightarrow (1_{r R}) \cdot_s m = m$

definition
  submodule $::$ $[('b, 'm)$ *Ring-scheme*, $('a, 'b, 'c)$ *Module-scheme*, $'a$ *set*$] \Rightarrow$
      bool **where**
  submodule $R\ A\ H \longleftrightarrow H \subseteq carrier\ A \wedge A +> H \wedge (\forall a.\ \forall m.$
      $(a \in carrier\ R \wedge m \in H) \longrightarrow (sprod\ A\ a\ m) \in H)$

definition
  mdl $::$ $[('a, 'b, 'm)$ *Module-scheme*, $'a$ *set*$] \Rightarrow ('a, 'b)$ *Module* **where**
  mdl $M\ H = (\!|carrier = H,\ pop = pop\ M,\ mop = mop\ M,\ zero = zero\ M,$
    $sprod = \lambda a.\ \lambda x{\in}H.\ sprod\ M\ a\ x|\!)$

abbreviation

MODULE (**infixl** *module 58*) **where**
R module M == *Module M R*

lemma (**in** *Module*) *module-is-ag*: *aGroup M* **..**

lemma (**in** *Module*) *module-inc-zero*: $\mathbf{0}_M \in$ *carrier M*
apply (*simp add*:*ag-inc-zero*)
done

lemma (**in** *Module*) *submodule-subset*:*submodule R M H* $\Longrightarrow$ *H* $\subseteq$ *carrier M*
apply (*simp add*:*submodule-def*)
done

lemma (**in** *Module*) *submodule-asubg*:*submodule R M H* $\Longrightarrow$ *M* +> *H*
by (*simp add*:*submodule-def*)

lemma (**in** *Module*) *submodule-subset1*:$[\![$*submodule R M H*; *h* $\in$ *H*$]\!] \Longrightarrow$
h $\in$ *carrier M*
apply (*simp add*:*submodule-def*)
apply (*erule conjE*)+
apply (*simp add*:*subsetD*)
done

lemma (**in** *Module*) *submodule-inc-0*:*submodule R M H* $\Longrightarrow$
$\mathbf{0}_M \in H$
apply (*simp add*:*submodule-def*, (*erule conjE*)+)
apply (*rule asubg-inc-zero*, *assumption*+)
done

lemma (**in** *Module*) *sc-un*: *m* $\in$ *carrier M* $\Longrightarrow$ $1_{r\,R} \cdot_s m = m$
apply (*simp add*:*sprod-one*)
done

lemma (**in** *Module*) *sc-mem*:$[\![$*a* $\in$ *carrier R*; *m* $\in$ *carrier M*$]\!] \Longrightarrow$
$a \cdot_s m \in$ *carrier M*
apply (*simp add*:*sprod-closed*)
done

lemma (**in** *Module*) *submodule-sc-closed*:$[\![$*submodule R M H*;
a $\in$ *carrier R*; *h* $\in$ *H*$]\!] \Longrightarrow$ $a \cdot_s h \in H$
apply (*simp add*:*submodule-def*)
done

lemma (**in** *Module*) *sc-assoc*:$[\![$*a* $\in$ *carrier R*; *b* $\in$ *carrier R*;
m $\in$ *carrier M*$]\!] \Longrightarrow (a \cdot_{r\,R} b) \cdot_s m = a \cdot_s (b \cdot_s m)$
apply (*simp add*:*sprod-assoc*)
done

lemma (**in** *Module*) *sc-l-distr*:⟦$a \in$ *carrier* R; $b \in$ *carrier* R;
$m \in$ *carrier* M⟧ $\Longrightarrow$ $(a \pm_R b) \cdot_s m = a \cdot_s m \pm b \cdot_s m$
apply (*simp add*:*sprod-l-distr*)
done

lemma (**in** *Module*) *sc-r-distr*:⟦$a \in$ *carrier* R; $m \in$ *carrier* M; $n \in$ *carrier* M⟧ $\Longrightarrow$
$a \cdot_s (m \pm n) = a \cdot_s m \pm a \cdot_s n$
apply (*simp add*:*sprod-r-distr*)
done

lemma (**in** *Module*) *sc-0-m*:$m \in$ *carrier* $M \Longrightarrow \mathbf{0}_R \cdot_s m = \mathbf{0}_M$
apply (*cut-tac sc-Ring*,
frule Ring.ring-is-ag,
frule aGroup.ag-inc-zero [*of* R],
frule sc-l-distr[*of* $\mathbf{0}_R$ $\mathbf{0}_R$ m], *assumption*+,
frule sc-mem [*of* $\mathbf{0}_R$ m], *assumption*+)
apply (*simp add*:*aGroup.ag-l-zero*, *frule sym*,
thin-tac $\mathbf{0}_R \cdot_s m = \mathbf{0}_R \cdot_s m \pm \mathbf{0}_R \cdot_s m$)
apply (*frule ag-eq-sol1* [*of* $\mathbf{0}_R \cdot_s m$ $\mathbf{0}_R \cdot_s m$ $\mathbf{0}_R \cdot_s m$], *assumption*+,
simp add:*ag-l-inv1*)
done

lemma (**in** *Module*) *sc-a-0*:$a \in$ *carrier* $R \Longrightarrow a \cdot_s \mathbf{0} = \mathbf{0}$
apply (*cut-tac ag-inc-zero*,
frule sc-r-distr[*of* a $\mathbf{0}$ $\mathbf{0}$], *assumption*+,
frule sc-mem [*of* a $\mathbf{0}$], *assumption*+)
apply (*simp add*:*ag-l-zero*, *frule sym*,
thin-tac $a \cdot_s \mathbf{0} = a \cdot_s \mathbf{0} \pm a \cdot_s \mathbf{0}$)
apply (*frule ag-eq-sol1* [*of* $a \cdot_s \mathbf{0}$ $a \cdot_s \mathbf{0}$ $a \cdot_s \mathbf{0}$], *assumption*+,
simp add:*ag-l-inv1*)
done

lemma (**in** *Module*) *sc-minus-am*:⟦$a \in$ *carrier* R; $m \in$ *carrier* M⟧
$\Longrightarrow -_a (a \cdot_s m) = a \cdot_s (-_a m)$
apply (*frule ag-mOp-closed* [*of* m],
frule sc-r-distr[*of* a m $-_a m$], *assumption*+,
simp add:*ag-r-inv1*,
simp add:*sc-a-0*, *frule sym*,
thin-tac $\mathbf{0} = a \cdot_s m \pm a \cdot_s (-_a m)$)
apply (*frule sc-mem* [*of* a m], *assumption*+,
frule sc-mem [*of* a $-_a m$], *assumption*+,
frule ag-eq-sol1 [*of* $a \cdot_s m$ $a \cdot_s (-_a m)$ $\mathbf{0}$], *assumption*+,
simp add:*ag-inc-zero*, *assumption*)
apply (*frule ag-mOp-closed* [*of* $a \cdot_s m$],
simp add:*ag-r-zero*)
done

lemma (**in** *Module*) *sc-minus-am1*:⟦$a \in$ *carrier* R; $m \in$ *carrier* M⟧
$\Longrightarrow -_a (a \cdot_s m) = (-_{a_R} a) \cdot_s m$

apply (*cut-tac sc-Ring*, *frule Ring.ring-is-ag*,
frule aGroup.ag-mOp-closed [*of R a*], *assumption*+,
frule sc-l-distr[*of a* $-_{a\,R}$ *a m*], *assumption*+,
simp add:*aGroup.ag-r-inv1* [*of R*],
simp add:*sc-0-m*, *frule sym*) **apply** (
thin-tac $\mathbf{0} = a \cdot_s m \pm (-_{a\,R}\ a) \cdot_s m$)
apply (*frule sc-mem* [*of a m*], *assumption*+,
frule sc-mem [*of* $-_{a\,R}$ *a m*], *assumption*+)
apply (*frule ag-eq-sol1* [*of a* $\cdot_s$ *m* $(-_{a\,R}\ a) \cdot_s m\ \mathbf{0}$], *assumption*+,
simp add:*ag-inc-zero*, *assumption*)
apply (*frule ag-mOp-closed* [*of a* $\cdot_s$ *m*])
apply (*thin-tac a* $\cdot_s$ *m* $\pm$ $(-_{a\,R}\ a) \cdot_s m = \mathbf{0}$,
simp add:*ag-r-zero*)
done

lemma (**in** *Module*) *submodule-0*:*submodule R M* $\{\mathbf{0}\}$
apply (*simp add*:*submodule-def*)
apply (*simp add*:*ag-inc-zero*)
apply (*simp add*:*asubg-zero*)
apply (*rule allI*, *rule impI*)
apply (*simp add*:*sc-a-0*)
done

lemma (**in** *Module*) *submodule-whole*:*submodule R M* (*carrier M*)
apply (*simp add*:*submodule-def*)
apply (*simp add*:*asubg-whole*)
apply ((*rule allI*)+, *rule impI*, *erule conjE*)
apply (*simp add*:*sc-mem*)
done

lemma (**in** *Module*) *submodule-pOp-closed*:$[\![$*submodule R M H*; *h* $\in$ *H*; *k* $\in$ *H*$]\!]$
$\Longrightarrow$
h $\pm$ *k* $\in$ *H*
apply (*simp add*:*submodule-def*)
apply (*erule conjE*)+
apply (*thin-tac* $\forall$ *a m*. *a* $\in$ *carrier R* $\wedge$ *m* $\in$ *H* $\longrightarrow$ *a* $\cdot_s$ *m* $\in$ *H*)
apply (*simp add*:*asubg-pOp-closed*)
done

lemma (**in** *Module*) *submodule-mOp-closed*:$[\![$*submodule R M H*; *h* $\in$ *H*$]\!]$
$\Longrightarrow$ $-_a$ *h* $\in$ *H*
apply (*simp add*:*submodule-def*,
(*erule conjE*)+,
thin-tac $\forall$ *a m*. *a* $\in$ *carrier R* $\wedge$ *m* $\in$ *H* $\longrightarrow$ *a* $\cdot_s$ *m* $\in$ *H*)
apply (*rule asubg-mOp-closed*, *assumption*+)
done

definition
mHom :: [('*b*, '*m*) *Ring-scheme*, ('*a*, '*b*, '*m1*) *Module-scheme*,

$('c, 'b, 'm2)$ *Module-scheme*] $\Rightarrow$ $('a \Rightarrow 'c)$ *set*
where
mHom R M N = {*f*. *f* ∈ *aHom M N* ∧
(∀ *a*∈*carrier R*. ∀ *m*∈*carrier M*. *f* (*a* ·$_{s M}$ *m*) = *a* ·$_{s N}$ (*f m*))}

definition
mimg :: [$('b, 'm)$ *Ring-scheme*, $('a, 'b, 'm1)$ *Module-scheme*,
$('c, 'b, 'm2)$ *Module-scheme*, $'a \Rightarrow 'c$] $\Rightarrow$ $('c, 'b)$ *Module*
((*4mimg*_ -,-/ -) [*88*,*88*,*88*,*89*]*88*) **where**
mimg$_{R\ M,N}$ *f* = *mdl N* (*f* ‘ (*carrier M*))

definition
mzeromap :: [$('a, 'b, 'm1)$ *Module-scheme*, $('c, 'b, 'm2)$ *Module-scheme*]
$\Rightarrow$ $('a \Rightarrow 'c)$ **where**
mzeromap M N = (λ*x*∈*carrier M*. **0**$_N$)

lemma (**in** *Ring*) *mHom-func*:*f* ∈ *mHom R M N* ⟹ *f* ∈ *carrier M* → *carrier N*
by (*simp add*:*mHom-def aHom-def*)

lemma (**in** *Module*) *mHom-test*:⟦*R module N*; *f* ∈ *carrier M* → *carrier N* ∧
f ∈ *extensional* (*carrier M*) ∧
(∀ *m*∈*carrier M*. ∀ *n*∈*carrier M*. *f* (*m* ±$_M$ *n*) = *f m* ±$_N$ (*f n*)) ∧
(∀ *a*∈*carrier R*. ∀ *m*∈*carrier M*. *f* (*a* ·$_{s M}$ *m*) = *a* ·$_{s N}$ (*f m*))⟧ ⟹
f ∈ *mHom R M N*
apply (*simp add*:*mHom-def*)
apply (*simp add*:*aHom-def*)
done

lemma (**in** *Module*) *mHom-mem*:⟦*R module N*; *f* ∈ *mHom R M N*; *m* ∈ *carrier M*⟧
⟹ *f m* ∈ *carrier N*
apply (*simp add*:*mHom-def aHom-def*) **apply** (*erule conjE*)+
apply (*simp add*:*Pi-def*)
done

lemma (**in** *Module*) *mHom-add*:⟦*R module N*; *f* ∈ *mHom R M N*; *m* ∈ *carrier M*;
n ∈ *carrier M*⟧ ⟹ *f* (*m* ± *n*) = *f m* ±$_N$ (*f n*)
apply (*simp add*:*mHom-def*) **apply** (*erule conjE*)+
apply (*frule Module.module-is-ag* [*of N R*],
cut-tac module-is-ag)
apply (*simp add*:*aHom-add*)
done

lemma (**in** *Module*) *mHom-0*:⟦*R module N*; *f* ∈ *mHom R M N*⟧ ⟹ *f* (**0**) = **0**$_N$
apply (*simp add*:*mHom-def*, (*erule conjE*)+,
frule Module.module-is-ag [*of N*],
cut-tac module-is-ag)
apply (*simp add*:*aHom-0-0*)

done

lemma (**in** *Module*) *mHom-inv*:⟦*R module N*; *m* ∈ *carrier M*; *f* ∈ *mHom R M N*⟧ ⟹
$f\ (-_a\ m) = -_{aN}\ (f\ m)$
apply (*cut-tac module-is-ag*,
frule Module.module-is-ag [*of N*])
apply (*simp add*:*mHom-def*, (*erule conjE*)+)
apply (*rule aHom-inv-inv*, *assumption*+)
done

lemma (**in** *Module*) *mHom-lin*:⟦*R module N*; *m* ∈ *carrier M*; *f* ∈ *mHom R M N*;
$a \in$ *carrier* R⟧ ⟹ $f\ (a \cdot_s\ m) = a \cdot_{sN}\ (f\ m)$
apply (*simp add*:*mHom-def*)
done

lemma (**in** *Module*) *mker-inc-zero*:
⟦*R module N*; *f* ∈ *mHom R M N* ⟧ ⟹ $\mathbf{0} \in (ker_{M,N}\ f)$
apply (*simp add*:*ker-def*)
apply (*simp add*:*module-inc-zero*)
apply (*simp add*:*mHom-0*)
done

lemma (**in** *Module*) *mHom-eq-ker*:⟦*R module N*; *f* ∈ *mHom R M N*; *a* ∈ *carrier M*;
b∈ *carrier M*; $a \pm (-_a\ b) \in ker_{M,N}\ f$⟧ ⟹ $f\ a = f\ b$
apply (*simp add*:*ker-def*, *erule conjE*)
apply (*cut-tac module-is-ag*,
frule aGroup.ag-mOp-closed [*of M b*], *assumption*+,
simp add:*mHom-add*, *simp add*:*mHom-inv*,
thin-tac aGroup M)
apply (*frule mHom-mem* [*of N f a*], *assumption*+,
frule mHom-mem [*of N f b*], *assumption*+,
frule Module.module-is-ag[*of N*])
apply (*subst aGroup.ag-eq-diffzero*[*of N*], *assumption*+)
done

lemma (**in** *Module*) *mHom-ker-eq*:⟦*R module N*; *f* ∈ *mHom R M N*; *a* ∈ *carrier M*;
b∈ *carrier M*; *f a* = *f b*⟧ ⟹ $a \pm (-_a\ b) \in ker_{M,N}\ f$
apply (*simp add*:*ker-def*)
apply (*frule ag-mOp-closed*[*of b*])
apply (*simp add*:*ag-pOp-closed*)
apply (*simp add*:*mHom-add mHom-inv*)
apply (*frule mHom-mem* [*of N f b*], *assumption*+)
apply (*frule-tac* $R = R$ **and** $M = N$ **in** *Module.module-is-ag*,
simp add:*aGroup.ag-r-inv1*)
done

lemma (**in** *Module*) *mker-submodule*:⟦*R module N*; *f* ∈ *mHom R M N*⟧ ⟹
submodule R M (*ker*$_{M,N}$ *f*)
apply (*cut-tac module-is-ag*,
frule Module.module-is-ag [*of N*])
apply (*simp add*:*submodule-def*)
apply (*rule conjI*)
apply (*rule subsetI*, *simp add*:*ker-def*)

apply (*rule conjI*)
apply (*simp add*:*mHom-def*, (*erule conjE*)+, *simp add*:*ker-subg*)

apply ((*rule allI*)+, *rule impI*, *erule conjE*)
apply (*simp add*:*ker-def*, *erule conjE*)
apply (*simp add*:*sc-mem*)
apply (*subst mHom-lin* [*of N - f*], *assumption*+, *simp*)
apply (*simp add*:*Module.sc-a-0*[*of N*])
done

lemma (**in** *Module*) *mker-mzeromap*:*R module N* ⟹
ker$_{M,N}$ (*mzeromap M N*) = *carrier M*
apply (*simp add*:*ker-def mzeromap-def*)
done

lemma (**in** *Module*) *mdl-carrier*:*submodule R M H* ⟹ *carrier* (*mdl M H*) = *H*
apply (*simp add*:*mdl-def*)
done

lemma (**in** *Module*) *mdl-is-ag*:*submodule R M H* ⟹ *aGroup* (*mdl M H*)
apply (*cut-tac module-is-ag*)
apply (*rule aGroup.intro*)
apply (*simp add*:*mdl-def*)
apply (*clarsimp simp*: *submodule-def asubg-pOp-closed*)

apply (*simp add*:*mdl-def*)
apply (*simp add*:*submodule-def*, (*erule conjE*)+,
frule-tac c = a **in** *subsetD*[*of H carrier M*], *assumption*+,
frule-tac c = b **in** *subsetD*[*of H carrier M*], *assumption*+,
frule-tac c = c **in** *subsetD*[*of H carrier M*], *assumption*+,
simp add:*aGroup.ag-pOp-assoc*)

apply (*simp add*:*submodule-def*, (*erule conjE*)+,
simp add:*mdl-def*,
frule-tac c = a **in** *subsetD*[*of H carrier M*], *assumption*+,
frule-tac c = b **in** *subsetD*[*of H carrier M*], *assumption*+,
simp add:*aGroup.ag-pOp-commute*)

apply (*simp add*:*mdl-def*)
apply (*simp add*:*submodule-def aGroup.asubg-mOp-closed*)

apply (*simp add*:*mdl-def*,
simp add:*submodule-def*, (*erule conjE*)+,
frule-tac c = *a* **in** *subsetD*[*of H carrier M*], *assumption*+,
rule aGroup.ag-l-inv1, *assumption*+)

apply (*simp add*:*mdl-def*,
simp add:*submodule-def*, (*erule conjE*)+,
simp add:*asubg-inc-zero*)

apply (*simp add*:*mdl-def*, *simp add*:*submodule-def*, (*erule conjE*)+,
frule-tac c = *a* **in** *subsetD*[*of H carrier M*], *assumption*+)
apply (*simp add*:*ag-l-zero*)
done

lemma (**in** *Module*) *mdl-is-module*:*submodule R M H* ⟹ *R module* (*mdl M H*)
apply (*rule Module.intro*)
apply (*simp add*:*mdl-is-ag*)

apply (*rule Module-axioms.intro*)
apply (*simp add*:*sc-Ring*)

apply (*simp add*:*mdl-def*)
apply (*simp add*:*submodule-def*)

apply (*simp add*:*mdl-def*)
apply (*simp add*:*submodule-def*, (*erule conjE*)+,
frule-tac c = *m* **in** *subsetD*[*of H carrier M*], *assumption*+,
simp add:*sc-l-distr*)

apply (*simp add*:*mdl-def submodule-def*, (*erule conjE*)+,
simp add:*asubg-pOp-closed*,
frule-tac c = *m* **in** *subsetD*[*of H carrier M*], *assumption*+,
frule-tac c = *n* **in** *subsetD*[*of H carrier M*], *assumption*+,
simp add:*sc-r-distr*)
apply (*simp add*:*mdl-def submodule-def*, (*erule conjE*)+,
frule-tac c = *m* **in** *subsetD*[*of H carrier M*], *assumption*+,
simp add:*sc-assoc*)
apply (*simp add*:*mdl-def submodule-def*, (*erule conjE*)+,
frule-tac c = *m* **in** *subsetD*[*of H carrier M*], *assumption*+,
simp add:*sprod-one*)
done

lemma (**in** *Module*) *submodule-of-mdl*:⟦*submodule R M H*; *submodule R M N*; *H* ⊆ *N*⟧
⟹ *submodule R* (*mdl M N*) *H*
apply (*subst submodule-def*)
apply (*rule conjI*, *simp add*:*mdl-def*)
apply (*rule conjI*)
apply (*rule aGroup.asubg-test*[*of mdl M N H*])

apply (*frule mdl-is-module*[*of N*],
simp add:*Module.module-is-ag*, *simp add*:*mdl-def*)
apply (*simp add*:*submodule-def*[*of R M H*], (*erule conjE*)+)
apply (*frule asubg-inc-zero*[*of H*], *simp add*:*nonempty*)

apply ((*rule ballI*)+, *simp add*:*mdl-def*)
apply (*simp add*:*submodule-def*[*of R M H*], (*erule conjE*)+)
apply (*frule-tac x* = *b* **in** *asubg-mOp-closed*[*of H*], *assumption*+)
apply (*rule asubg-pOp-closed*[*of H*], *assumption*+)

apply ((*rule allI*)+, *rule impI*, *erule conjE*)
apply (*simp add*:*mdl-def subsetD*)
apply (*simp add*:*submodule-def*[*of R M H*])
done

lemma (**in** *Module*) *img-set-submodule*:⟦*R module N*; *f* ∈ *mHom R M N*⟧ ⟹
submodule R N (*f* ‘ (*carrier M*))
apply (*simp add*:*submodule-def*)
apply (*rule conjI*)
apply (*rule subsetI*)
apply (*simp add*:*image-def*)
apply (*erule bexE*, *simp*, *thin-tac x* = *f xa*)
apply (*simp add*:*mHom-mem*)
apply (*rule conjI*)
apply (*frule Module.module-is-ag* [*of N*])
apply (*rule aGroup.asubg-test*, *assumption*+)
apply (*rule subsetI*) **apply** (*simp add*:*image-def*)
apply (*erule bexE*) **apply** (*simp add*:*mHom-mem*)
apply (*cut-tac ag-inc-zero*,
simp add:*mHom-mem*, *simp add*:*nonempty*)
apply ((*rule ballI*)+, *simp add*:*image-def*)
apply ((*erule bexE*)+, *simp*)
apply (*simp add*:*mHom-inv*[*THEN sym*],
frule-tac x = *xa* **in** *ag-mOp-closed*,
simp add:*mHom-add*[*THEN sym*, *of N f*],
frule-tac x = *x* **and** *y* = −$_a$ *xa* **in** *ag-pOp-closed*, *assumption*+)
apply *blast*

apply ((*rule allI*)+, *rule impI*, *erule conjE*)
apply (*simp add*:*image-def*, *erule bexE*, *simp*)
apply (*simp add*:*mHom-lin*[*THEN sym*, *of N - f*])
apply (*frule-tac a* = *a* **and** *m* = *x* **in** *sc-mem*, *assumption*)
apply *blast*
done

lemma (**in** *Module*) *mimg-module*:⟦*R module N*; *f* ∈ *mHom R M N*⟧ ⟹
R module (*mimg R M N f*)
apply (*simp add*:*mimg-def*)
apply (*rule Module.mdl-is-module*[*of N R f* ‘ (*carrier M*)], *assumption*)

apply (*simp add*:*img-set-submodule*)
done

lemma (**in** *Module*) *surjec-to-mimg*:⟦*R module N*; *f* ∈ *mHom R M N*⟧ ⟹
surjec$_{M,\ (mimg\ R\ M\ N\ f)}$ *f*
apply (*simp add*:*surjec-def*)
apply (*rule conjI*)
apply (*simp add*:*aHom-def*)
apply (*rule conjI*)
apply (*simp add*:*mimg-def mdl-def*)
apply (*rule conjI*)
apply (*simp add*:*mHom-def aHom-def restrict-def extensional-def*)
apply ((*rule ballI*)+, *simp add*:*mimg-def mdl-def*, *simp add*:*mHom-add*)
apply (*simp add*:*mimg-def mdl-def*)
apply (*simp add*:*surj-to-def image-def*)
done

definition
tOp-mHom :: [('*b*, '*m*) *Ring-scheme*, ('*a*, '*b*, '*m1*) *Module-scheme*,
('*c*, '*b*, '*m2*) *Module-scheme*] ⇒ ('*a* ⇒ '*c*) ⇒ ('*a* ⇒ '*c*) ⇒ ('*a* ⇒ '*c*) **where**
tOp-mHom R M N f g = (λ*x* ∈ *carrier M*. (*f x* ±$_N$ (*g x*)))

definition
iOp-mHom :: [('*b*, '*m*) *Ring-scheme*, ('*a*, '*b*, '*m1*) *Module-scheme*,
('*c*, '*b*, '*m2*) *Module-scheme*] ⇒ ('*a* ⇒ '*c*) ⇒ ('*a* ⇒ '*c*) **where**
iOp-mHom R M N f = (λ*x* ∈ *carrier M*. (−$_{a\,N}$ (*f x*)))

definition
sprod-mHom ::[('*b*, '*m*) *Ring-scheme*, ('*a*, '*b*, '*m1*) *Module-scheme*,
('*c*, '*b*, '*m2*) *Module-scheme*] ⇒ '*b* ⇒ ('*a* ⇒ '*c*) ⇒ ('*a* ⇒ '*c*) **where**
sprod-mHom R M N a f = (λ*x* ∈ *carrier M*. *a* ·$_{s\,N}$ (*f x*))

definition
HOM :: [('*b*, '*more*) *Ring-scheme*, ('*a*, '*b*, '*more1*) *Module-scheme*,
('*c*, '*b*, '*more2*) *Module-scheme*] ⇒ ('*a* ⇒ '*c*, '*b*) *Module*
((*3HOM*_ -/ -) [*90*, *90*, *91*] *90*) **where**
HOM$_R$ *M N* = (|*carrier* = *mHom R M N*, *pop* = *tOp-mHom R M N*,
mop = *iOp-mHom R M N*, *zero* = *mzeromap M N*, *sprod* =*sprod-mHom R M N* |)

lemma (**in** *Module*) *zero-HOM*:*R module N* ⟹
mzeromap M N = **0**$_{HOM_R\ M\ N}$
apply (*simp add*:*HOM-def*)
done

lemma (**in** *Module*) *tOp-mHom-closed*:⟦*R module N*; *f* ∈ *mHom R M N*; *g* ∈ *mHom R M N*⟧
⟹ *tOp-mHom R M N f g* ∈ *mHom R M N*
apply (*rule mHom-test*, *assumption*+)

apply (*rule conjI*)
 apply (*rule Pi-I*)
 apply (*simp add*:*tOp-mHom-def*)
 apply (*frule-tac f* = *f* **and** *m* = *x* **in** *mHom-mem* [*of N*], *assumption*+,
 frule-tac f = *g* **and** *m* = *x* **in** *mHom-mem* [*of N*], *assumption*+,
 frule Module.module-is-ag [*of N*],
 simp add:*aGroup.ag-pOp-closed*[*of N*])
apply (*rule conjI*)
 apply (*simp add*:*tOp-mHom-def restrict-def extensional-def*)
apply (*rule conjI*)
 apply (*rule ballI*)+
 apply (*simp add*:*tOp-mHom-def*)
 apply (*simp add*:*ag-pOp-closed*)

apply (*frule-tac f* = *f* **and** *m* = *m* **in** *mHom-mem* [*of N*], *assumption*+,
 frule-tac f = *f* **and** *m* = *n* **in** *mHom-mem* [*of N*], *assumption*+,
 frule-tac f = *g* **and** *m* = *m* **in** *mHom-mem* [*of N*], *assumption*+,
 frule-tac f = *g* **and** *m* = *n* **in** *mHom-mem* [*of N*], *assumption*+,
 simp add:*mHom-add*,
 frule Module.module-is-ag [*of N*],
 subst aGroup.pOp-assocTr43[*of N*], *assumption*+,
 frule-tac x = *f n* **and** *y* = *g m* **in** *aGroup.ag-pOp-commute* [*of N*],
 assumption+)
apply *simp*
apply (*subst aGroup.pOp-assocTr43*[*of N*], *assumption*+, *simp*)

apply (*rule ballI*)+
apply (*simp add*:*tOp-mHom-def*)
apply (*frule-tac a* = *a* **and** *m* = *m* **in** *sc-mem*, *assumption*, *simp*)
apply (*frule-tac f* = *f* **and** *m* = *m* **in** *mHom-mem* [*of N*], *assumption*+,
 frule-tac f = *g* **and** *m* = *m* **in** *mHom-mem* [*of N*], *assumption*+,
 frule-tac a = *a* **and** *m* = *f m* **and** *n* = *g m* **in**
 Module.sc-r-distr[*of N R*], *assumption*+,
 simp)
apply (*simp add*:*mHom-lin*)
done

lemma (**in** *Module*) *iOp-mHom-closed*:⟦*R module N*; *f* ∈ *mHom R M N*⟧
 ⟹ *iOp-mHom R M N f* ∈ *mHom R M N*
apply (*rule mHom-test*, *assumption*+)
apply (*rule conjI*)
 apply (*rule Pi-I*)
 apply (*simp add*:*iOp-mHom-def*)
 apply (*frule-tac f* = *f* **and** *m* = *x* **in** *mHom-mem* [*of N*], *assumption*+)
 apply (*frule Module.module-is-ag* [*of N*])
 apply (*simp add*:*aGroup.ag-mOp-closed*)
apply (*rule conjI*)
 apply (*simp add*:*iOp-mHom-def restrict-def extensional-def*)
apply (*rule conjI*) **apply** (*rule ballI*)+

apply (*simp add*:*iOp-mHom-def*)
apply (*simp add*:*ag-pOp-closed*)
apply (*simp add*:*mHom-add*)
apply (*frule-tac* $f = f$ **and** $m = m$ **in** *mHom-mem* [*of* N], *assumption*+,
frule-tac $f = f$ **and** $m = n$ **in** *mHom-mem* [*of* N], *assumption*+)
apply (*frule Module.module-is-ag* [*of* N])
apply (*simp add*:*aGroup.ag-p-inv*)

apply (*rule ballI*)+
apply (*simp add*:*iOp-mHom-def*)
apply (*simp add*:*sc-mem*)
apply (*simp add*:*mHom-lin*)
apply (*frule-tac* $f = f$ **and** $m = m$ **in** *mHom-mem* [*of* N], *assumption*+)
apply (*simp add*:*Module.sc-minus-am*[*of* N])
done

lemma (**in** *Module*) *mHom-ex-zero*:R *module* $N \Longrightarrow$ *mzeromap* M $N \in$ *mHom* R M N
apply (*simp add*:*mHom-def*)
apply (*rule conjI*)
apply (*simp add*:*aHom-def*,
rule conjI,
simp add:*mzeromap-def*, *simp add*:*Module.module-inc-zero*)

apply (*simp add*:*mzeromap-def extensional-def*)

apply ((*rule ballI*)+,
simp add:*ag-pOp-closed*,
frule Module.module-is-ag [*of* N],
frule aGroup.ag-inc-zero [*of* N],
simp add:*aGroup.ag-l-zero*)
apply ((*rule ballI*)+,
simp add:*mzeromap-def*,
simp add:*sc-mem*)
apply (*simp add*:*Module.sc-a-0*)
done

lemma (**in** *Module*) *mHom-eq*:⟦R *module* N; $f \in$ *mHom* R M N; $g \in$ *mHom* R M N;
$\forall\, m{\in}$*carrier* M. $f\ m = g\ m$⟧ $\Longrightarrow f = g$
apply (*simp add*:*mHom-def aHom-def*)
apply (*erule conjE*)+
apply (*rule funcset-eq*, *assumption*+)
done

lemma (**in** *Module*) *mHom-l-zero*:⟦R *module* N; $f \in$ *mHom* R M N⟧
$\Longrightarrow$ *tOp-mHom* R M N (*mzeromap* M N) $f = f$
apply (*frule mHom-ex-zero* [*of* N])
apply (*frule tOp-mHom-closed* [*of* N *mzeromap* M N f], *assumption*+)

apply (*rule mHom-eq*, *assumption*+)
apply (*rule ballI*)
apply (*simp add*:*tOp-mHom-def*, *simp add*:*mzeromap-def*)
apply (*frule-tac f* = *f* **and** *m* = *m* **in** *mHom-mem* [*of N*], *assumption*+)
apply (*frule Module.module-is-ag* [*of N*])
apply (*simp add*:*aGroup.ag-l-zero*[*of N*])
done

lemma (**in** *Module*) *mHom-l-inv*:⟦*R module N*; *f* ∈ *mHom R M N*⟧
⟹ *tOp-mHom R M N* (*iOp-mHom R M N f*) *f* = *mzeromap M N*
apply (*frule mHom-ex-zero* [*of N*])
apply (*frule-tac f* = *f* **in** *iOp-mHom-closed* [*of N*], *assumption*,
frule-tac f = *iOp-mHom R M N f* **and** *g* = *f* **in** *tOp-mHom-closed* [*of N*],
assumption+,
frule mHom-ex-zero [*of N*])
apply (*rule mHom-eq*, *assumption*+, *rule ballI*)
apply (*simp add*:*tOp-mHom-def iOp-mHom-def*, *simp add*:*mzeromap-def*)
apply (*frule-tac f* = *f* **and** *m* = *m* **in** *mHom-mem* [*of N*], *assumption*+)
apply (*frule Module.module-is-ag* [*of N*])
apply (*simp add*:*aGroup.ag-l-inv1*)
done

lemma (**in** *Module*) *mHom-tOp-assoc*:⟦*R module N*; *f* ∈ *mHom R M N*; *g* ∈ *mHom R M N*;
h ∈ *mHom R M N*⟧ ⟹ *tOp-mHom R M N* (*tOp-mHom R M N f g*) *h* =
tOp-mHom R M N f (*tOp-mHom R M N g h*)
apply (*frule-tac f* = *f* **and** *g* = *g* **in** *tOp-mHom-closed* [*of N*], *assumption*+,
frule-tac f = *tOp-mHom R M N f g* **and** *g* = *h* **in**
tOp-mHom-closed [*of N*], *assumption*+,
frule-tac f = *g* **and** *g* = *h* **in** *tOp-mHom-closed* [*of N*], *assumption*+,
frule-tac f = *f* **and** *g* = *tOp-mHom R M N g h* **in**
tOp-mHom-closed [*of N*], *assumption*+)
apply (*rule mHom-eq*, *assumption*+, *rule ballI*,
thin-tac tOp-mHom R M N f g ∈ *mHom R M N*,
thin-tac tOp-mHom R M N (*tOp-mHom R M N f g*) *h* ∈ *mHom R M N*,
thin-tac tOp-mHom R M N g h ∈ *mHom R M N*,
thin-tac tOp-mHom R M N f (*tOp-mHom R M N g h*) ∈ *mHom R M N*)
apply (*simp add*:*tOp-mHom-def*)
apply (*frule-tac f* = *f* **and** *m* = *m* **in** *mHom-mem* [*of N*], *assumption*+,
frule-tac f = *g* **and** *m* = *m* **in** *mHom-mem* [*of N*], *assumption*+,
frule-tac f = *h* **and** *m* = *m* **in** *mHom-mem* [*of N*], *assumption*+)
apply (*frule Module.module-is-ag* [*of N*])
apply (*simp add*:*aGroup.ag-pOp-assoc*)
done

lemma (**in** *Module*) *mHom-tOp-commute*:⟦*R module N*; *f* ∈ *mHom R M N*;
g ∈ *mHom R M N*⟧ ⟹ *tOp-mHom R M N f g* = *tOp-mHom R M N g f*
apply (*frule-tac f* = *f* **and** *g* = *g* **in** *tOp-mHom-closed* [*of N*], *assumption*+,
frule-tac f = *g* **and** *g* = *f* **in** *tOp-mHom-closed* [*of N*], *assumption*+)

apply (*rule mHom-eq*, *assumption*+)
 apply (*rule ballI*)
 apply (*thin-tac tOp-mHom R M N f g* ∈ *mHom R M N*,
 thin-tac tOp-mHom R M N g f ∈ *mHom R M N*)
 apply (*simp add*:*tOp-mHom-def*)
 apply (*frule-tac f* = *f* **and** *m* = *m* **in** *mHom-mem* [*of N*], *assumption*+,
 frule-tac f = *g* **and** *m* = *m* **in** *mHom-mem* [*of N*], *assumption*+,
 frule Module.module-is-ag [*of N*])
 apply (*simp add*:*aGroup.ag-pOp-commute*)
done

lemma (**in** *Module*) *HOM-is-ag*:*R module N* ⟹ *aGroup* (*HOM*$_R$ *M N*)
apply (*rule aGroup.intro*)
 apply (*simp add*:*HOM-def*)
 apply (*simp add*:*tOp-mHom-closed*)

apply (*simp add*:*HOM-def*)
 apply (*simp add*:*mHom-tOp-assoc*)

apply (*simp add*:*HOM-def*)
 apply (*simp add*:*mHom-tOp-commute*)

apply (*simp add*:*HOM-def*)
 apply (*simp add*:*iOp-mHom-closed*)

apply (*simp add*:*HOM-def*,
 simp add:*mHom-l-inv*)

apply (*simp add*:*HOM-def*)
 apply (*simp add*:*mHom-ex-zero*)

apply (*simp add*:*HOM-def*,
 simp add:*mHom-l-zero*)
done

lemma (**in** *Module*) *sprod-mHom-closed*:⟦*R module N*; *a* ∈ *carrier R*;
 f ∈ *mHom R M N*⟧ ⟹ *sprod-mHom R M N a f* ∈ *mHom R M N*
apply (*rule mHom-test*, *assumption*+)
apply (*rule conjI*)
 apply (*simp add*:*Pi-def*)
 apply (*rule allI*, *rule impI*, *simp add*:*sprod-mHom-def*,
 frule-tac f = *f* **and** *m* = *x* **in** *mHom-mem* [*of N*], *assumption*+,
 simp add:*Module.sc-mem* [*of N R a*])
apply (*rule conjI*)
 apply (*simp add*:*sprod-mHom-def restrict-def extensional-def*)
apply (*rule conjI*)
 apply (*rule ballI*)+
 apply (*frule-tac x* = *m* **and** *y* = *n* **in** *ag-pOp-closed*, *assumption*+)
 apply (*simp add*:*sprod-mHom-def*)

apply (*subst mHom-add* [*of N f*], *assumption*+)
 apply (*frule-tac f* = *f* **and** *m* = *m* **in** *mHom-mem* [*of N*], *assumption*+,
 frule-tac f = *f* **and** *m* = *n* **in** *mHom-mem* [*of N*], *assumption*+)
 apply (*simp add*:*Module.sc-r-distr*)

apply (*rule ballI*)+
 apply (*simp add*:*sprod-mHom-def*)
 apply (*frule-tac a* = *aa* **and** *m* = *m* **in** *sc-mem*, *assumption*+, *simp*)
 apply (*simp add*:*mHom-lin*)
 apply (*frule-tac f* = *f* **and** *m* = *m* **in** *mHom-mem* [*of N*], *assumption*+)
apply (*simp add*:*Module.sc-assoc*[*THEN sym*, *of N R*])
apply (*cut-tac sc-Ring*, *simp add*:*Ring.ring-tOp-commute*)
done

lemma (**in** *Module*) *HOM-is-module*:*R module N* $\Longrightarrow$ *R module* (*HOM*$_R$ *M N*)
apply (*rule Module.intro*)
apply (*simp add*:*HOM-is-ag*)
apply (*rule Module-axioms.intro*)
 apply (*simp add*:*sc-Ring*)

 apply (*simp add*:*HOM-def*)
 apply (*simp add*:*sprod-mHom-closed*)

 apply (*simp add*:*HOM-def*)
 apply (*cut-tac sc-Ring*,
 frule Ring.ring-is-ag[*of R*],
 frule-tac x = *a* **and** *y* = *b* **in** *aGroup.ag-pOp-closed*[*of R*], *assumption*+,
 frule-tac a = *a* $\pm_R$ *b* **and** *f* = *m* **in** *sprod-mHom-closed*[*of N*],
 assumption+)
 apply(*frule-tac a* = *a* **and** *f* = *m* **in** *sprod-mHom-closed*[*of N*], *assumption*+,
 frule-tac a = *b* **and** *f* = *m* **in** *sprod-mHom-closed*[*of N*], *assumption*+,
 frule-tac f = *sprod-mHom R M N a m* **and** *g* = *sprod-mHom R M N b m* **in**
 tOp-mHom-closed[*of N*], *assumption*+)
 apply (*rule mHom-eq*[*of N*], *assumption*+, *rule ballI*,
 simp add:*sprod-mHom-def tOp-mHom-def*)
 apply (*rename-tac a b f m*)
 apply (*frule-tac f* = *f* **and** *m* = *m* **in** *mHom-mem*[*of N*], *assumption*+)
 apply (*simp add*:*Module.sc-l-distr*[*of N*])

apply (*simp add*:*HOM-def*)
 apply (*rename-tac a f g*,
 frule-tac f = *f* **and** *g* = *g* **in** *tOp-mHom-closed*[*of N*], *assumption*+,
 frule-tac a = *a* **and** *f* = *tOp-mHom R M N f g* **in**
 sprod-mHom-closed[*of N*], *assumption*+,
 frule-tac a = *a* **and** *f* = *f* **in** *sprod-mHom-closed*[*of N*], *assumption*+,
 frule-tac a = *a* **and** *f* = *g* **in** *sprod-mHom-closed*[*of N*], *assumption*+,
 frule-tac f = *sprod-mHom R M N a f* **and** *g* = *sprod-mHom R M N a g*
 in *tOp-mHom-closed*[*of N*], *assumption*+)
 apply (*rule mHom-eq*[*of N*], *assumption*+, *rule ballI*,

simp add:*sprod-mHom-def tOp-mHom-def*,
frule-tac $f = f$ **and** $m = m$ **in** *mHom-mem*[*of N*], *assumption*+,
frule-tac $f = g$ **and** $m = m$ **in** *mHom-mem*[*of N*], *assumption*+)
apply (*simp add*:*Module.sc-r-distr*)

apply (*simp add*:*HOM-def*)
apply (*rename-tac a b f*)
apply (*cut-tac sc-Ring*,
frule-tac $x = a$ **and** $y = b$ **in** *Ring.ring-tOp-closed*, *assumption*+,
frule-tac $a = a \cdot_{r\,R} b$ **and** $f = f$ **in** *sprod-mHom-closed*[*of N*],
assumption+,
frule-tac $a = b$ **and** $f = f$ **in** *sprod-mHom-closed*[*of N*], *assumption*+,
frule-tac $a = a$ **and** $f =$ *sprod-mHom R M N b f* **in**
sprod-mHom-closed[*of N*], *assumption*+)
apply (*rule mHom-eq*[*of N*], *assumption*+, *rule ballI*,
simp add:*sprod-mHom-def*,
frule-tac $f = f$ **and** $m = m$ **in** *mHom-mem*[*of N*], *assumption*+,
simp add:*Module.sc-assoc*)

apply (*simp add*:*HOM-def*)
apply (*cut-tac sc-Ring*,
frule Ring.ring-one,
frule-tac $a = 1_{r\,R}$ **and** $f = m$ **in** *sprod-mHom-closed*[*of N*], *assumption*+)
apply (*rule mHom-eq*, *assumption*+, *rule ballI*, *rename-tac f m*,
simp add:*sprod-mHom-def*,
frule-tac $f = f$ **and** $m = m$ **in** *mHom-mem*[*of N*], *assumption*+,
simp add:*Module.sprod-one*)
done

5.2 Injective hom, surjective hom, bijective hom and inverse hom

definition
invmfun :: [('*b*, '*m*) *Ring-scheme*, ('*a*, '*b*, '*m1*) *Module-scheme*,
('*c*, '*b*, '*m2*) *Module-scheme*, '*a* ⇒ '*c*] ⇒ '*c* ⇒ '*a* **where**
invmfun R M N (*f* :: '*a* ⇒ '*c*) =
(λ*y*∈(*carrier N*). *SOME x*. (*x* ∈ (*carrier M*) ∧ *f x* = *y*))

definition
misomorphic :: [('*b*, '*m*) *Ring-scheme*, ('*a*, '*b*, '*m1*) *Module-scheme*,
('*c*, '*b*, '*m2*) *Module-scheme*] ⇒ *bool* **where**
misomorphic R M N ⟷ (∃*f*. *f* ∈ *mHom R M N* ∧ $bijec_{M,N}\, f$)

definition
mId :: ('*a*, '*b*, '*m1*) *Module-scheme* ⇒ '*a* ⇒ '*a* ((*mId*_/) [*89*]*88*) **where**
mId_M = (λ*m*∈*carrier M*. *m*)

definition

mcompose :: $[('a, 'r, 'm1)$ *Module-scheme*, $'b \Rightarrow 'c, 'a \Rightarrow 'b] \Rightarrow 'a \Rightarrow 'c$ **where**
mcompose M g f = *compose* (*carrier M*) *g f*

abbreviation
MISOM ((*3*- $\cong_-$ -) [*82*,*82*,*83*]*82*) **where**
$M \cong_R N$ == *misomorphic R M N*

lemma (**in** *Module*) *minjec-inj*:$[\![R$ *module* N; $inj_{M,N}\ f]\!] \Longrightarrow$
inj-on f (*carrier M*)
apply (*simp add*:*inj-on-def*, (*rule ballI*)+, *rule impI*)
apply (*simp add*:*injec-def*, *erule conjE*)
apply (*frule Module.module-is-ag*[*of N*])
apply (*cut-tac module-is-ag*)
apply (*frule-tac a* = *x* **in** *aHom-mem*[*of M N f*], *assumption*+,
frule-tac a = *y* **in** *aHom-mem*[*of M N f*], *assumption*+)
apply (*simp add*:*aGroup.ag-eq-diffzero*[*of N*])
apply (*simp add*:*aHom-inv-inv*[*THEN sym*, *of M N f*],
frule-tac x = *y* **in** *aGroup.ag-mOp-closed*, *assumption*+,
simp add:*aHom-add*[*THEN sym*, *of M N f*])
apply (*simp add*:*ker-def*)
apply (*frule-tac x* = *x* **and** *y* = $-_a$ *y* **in** *ag-pOp-closed*, *assumption*+)
apply (*subgoal-tac* ($x \pm_a y$) $\in \{a \in$ *carrier M*. *f a* = $\mathbf{0}_N\}$, *simp*)
apply (*simp add*:*ag-eq-diffzero*)
apply *blast*
done

lemma (**in** *Module*) *invmfun-l-inv*:$[\![R$ *module* N; $bijec_{M,N}\ f$; $m \in$ *carrier* $M]\!] \Longrightarrow$
(*invmfun R M N f*) (*f m*) = *m*
apply (*simp add*:*bijec-def*, *erule conjE*)
apply (*frule minjec-inj* [*of N f*], *assumption*+)
apply (*simp add*:*surjec-def*, *erule conjE*, *simp add*:*aHom-def*)
apply (*frule conjunct1*)
apply (*thin-tac f* $\in$ *carrier M* $\to$ *carrier N* $\wedge$
$f \in$ *extensional* (*carrier M*) $\wedge$
$(\forall\, a \in$ *carrier M*. $\forall\, b \in$ *carrier M*. *f* (*a* $\pm$ *b*) = *f a* $\pm_N$ *f b*))
apply (*frule invfun-l* [*of f carrier M carrier N m*], *assumption*+)
apply (*simp add*:*surj-to-def*)
apply (*simp add*:*invfun-def invmfun-def*)
done

lemma (**in** *Module*) *invmfun-mHom*:$[\![R$ *module* N; $bijec_{M,N}\ f$; $f \in$ *mHom R M N* $]\!] \Longrightarrow$
invmfun R M N f $\in$ *mHom R N M*
apply (*frule minjec-inj* [*of N f*])
apply (*simp add*:*bijec-def*)
apply (*subgoal-tac surjec*$_{M,N}$ *f*) **prefer** *2* **apply** (*simp add*:*bijec-def*)
apply (*rule Module.mHom-test*) **apply** *assumption* **apply** (*rule Module-axioms*)

apply (*rule conjI*)

apply (*simp add*:*surjec-def*, *erule conjE*)
apply (*simp add*:*aHom-def*, *frule conjunct1*)
apply (*thin-tac* $f \in carrier\ M \to carrier\ N \wedge$
$f \in extensional\ (carrier\ M) \wedge$
$(\forall a{\in}carrier\ M.\ \forall b{\in}carrier\ M.\ f\ (a \pm b) = f\ a \pm_N f\ b)$)
apply (*frule inv-func* [*of f carrier M carrier N*], *assumption*+)
apply (*simp add*:*invmfun-def invfun-def*)

apply (*rule conjI*)
apply (*simp add*:*invmfun-def restrict-def extensional-def*)

apply (*rule conjI*)
apply (*rule ballI*)+
apply (*simp add*:*surjec-def*)
apply (*erule conjE*, *simp add*:*surj-to-def*)
apply (*frule sym*, *thin-tac f ‘ carrier M = carrier N*, *simp*,
thin-tac carrier N = f ‘ carrier M)
apply (*simp add*:*image-def*, (*erule bexE*)+, *simp*)
apply (*simp add*:*mHom-add*[*THEN sym*])
apply (*frule-tac x = x* **and** *y = xa* **in** *ag-pOp-closed*, *assumption*+)
apply (*simp add*:*invmfun-l-inv*)

apply (*rule ballI*)+
apply (*simp add*:*surjec-def*, *erule conjE*)
apply (*simp add*:*surj-to-def*, *frule sym*, *thin-tac f ‘ carrier M = carrier N*)
apply (*simp add*:*image-def*, (*erule bexE*)+, *simp*)
apply (*simp add*:*mHom-lin*[*THEN sym*])
apply (*frule-tac a = a* **and** *m = x* **in** *sc-mem*, *assumption*+)
apply (*simp add*:*invmfun-l-inv*)
done

lemma (**in** *Module*) *invmfun-r-inv*:⟦*R module N*; $bijec_{M,N}\ f$; $n \in carrier\ N$⟧ ⟹
$f\ ((invmfun\ R\ M\ N\ f)\ n) = n$
apply (*frule minjec-inj*[*of N f*])
apply (*simp add*:*bijec-def*)
apply (*unfold bijec-def*, *frule conjunct2*, *fold bijec-def*)
apply (*simp add*:*surjec-def*, *erule conjE*, *simp add*:*surj-to-def*)
apply (*frule sym*, *thin-tac f ‘ carrier M = carrier N*, *simp*,
thin-tac carrier N = f ‘ carrier M)
apply (*simp add*:*image-def*, *erule bexE*, *simp*)
apply (*simp add*:*invmfun-l-inv*)
done

lemma (**in** *Module*) *mHom-compos*:⟦*R module L*; *R module N*; $f \in mHom\ R\ L\ M$;
$g \in mHom\ R\ M\ N$ ⟧ ⟹ $compos\ L\ g\ f \in mHom\ R\ L\ N$
apply (*simp add*:*mHom-def* [*of R L N*])
apply (*frule Module.module-is-ag* [*of L*],
frule Module.module-is-ag [*of N*])

apply (*rule conjI*)
 apply (*simp add*:*mHom-def*, (*erule conjE*)+)
  apply (*rule aHom-compos*[*of L M N f*], *assumption*+)
  apply (*cut-tac module-is-ag*, *assumption*+)

apply (*rule ballI*)+
apply (*simp add*:*compos-def compose-def*)
 apply (*simp add*:*Module.sc-mem*)
 apply (*subst Module.mHom-lin*[*of L R M - f*], *assumption*, *rule Module-axioms*, *assumption*+)
 apply (*subst Module.mHom-lin*[*of M R N - g*], *rule Module-axioms*, *assumption*)

 apply (*rule Module.mHom-mem*[*of L R M f*], *assumption*, *rule Module-axioms*, *assumption*+)
 apply *simp*
done

lemma (**in** *Module*) *mcompos-inj-inj*:⟦*R module L*; *R module N*; *f* ∈ *mHom R L M*;
      g ∈ *mHom R M N*; *injec*$_{L,M}$ *f*; *injec*$_{M,N}$ *g* ⟧ ⟹ *injec*$_{L,N}$ (*compos L g f*)
apply (*frule Module.module-is-ag* [*of L*],
      frule Module.module-is-ag [*of N*])
apply (*simp add*:*injec-def* [*of L N*])
apply (*rule conjI*)
 apply (*simp add*:*injec-def*, (*erule conjE*)+,
       rule-tac aHom-compos[*of L M N*], *assumption*+,
       rule module-is-ag)
 apply *assumption*+
 apply (*simp add*:*compos-def compose-def*)
 apply (*rule equalityI*)
 apply (*rule subsetI*, *simp*)
 apply (*simp add*:*injec-def* [*of - - g*], *erule conjE*, *simp add*:*ker-def*)
 apply (*subgoal-tac f x* ∈ {*a*. *a* ∈ *carrier M* ∧ *g a* = **0**$_N$})
 apply *simp*
 apply (*simp add*:*injec-def* [*of - - f*], *erule conjE*)
 apply (*subgoal-tac x* ∈ *ker*$_{L,M}$ *f*, *simp*, *thin-tac ker*$_{L,M}$ *f* = {**0**$_L$})
 apply (*simp add*:*ker-def*)
 apply (*thin-tac* {*a* ∈ *carrier M*. *g a* = **0**$_N$} = {**0**})
 apply (*simp*, *erule conjE*, *simp*)
 apply (*rule Module.mHom-mem*[*of L R M f*], *assumption*, *rule Module-axioms*, *assumption*+)

apply (*rule subsetI*, *simp*)
 apply (*frule Module.module-inc-zero* [*of L R*])
 apply (*frule Module.mHom-0*[*of L R M f*], *rule Module-axioms*, *assumption*+)
 apply (*simp add*:*ker-def*)
 apply (*subst mHom-0*[*of N*], *assumption*+, *simp*)
done

lemma (**in** *Module*) *mcompos-surj-surj*:⟦*R module L*; *R module N*; $surjec_{L,M}\ f$;
$surjec_{M,N}\ g$; *f* ∈ *mHom R L M*; *g* ∈ *mHom R M N* ⟧ ⟹
$surjec_{L,N}$ (*compos L g f*)
apply (*frule Module.module-is-ag* [*of L*],
frule Module.module-is-ag [*of N*],
cut-tac module-is-ag)
apply (*simp add*:*surjec-def* [*of L N*])
apply (*rule conjI*)
apply (*simp add*:*mHom-def*, (*erule conjE*)+)
apply (*rule aHom-compos*[*of L M N f g*], *assumption*+)

apply (*rule surj-to-test*)
apply (*cut-tac Module.mHom-compos* [*of M R L N f g*])
apply (*simp add*:*mHom-def aHom-def*)
apply (*rule Module-axioms*, *assumption*+)

apply (*rule ballI*)
apply (*simp add*: *compos-def compose-def*)
apply (*simp add*:*surjec-def* [*of - - g*])
apply (*erule conjE*) **apply** (*simp add*:*surj-to-def*)
apply (*frule sym*, *thin-tac g ' carrier M = carrier N*, *simp add*:*image-def*,
thin-tac carrier N = {*y*. ∃ *x*∈*carrier M*. *y* = *g x*},
erule bexE, *simp*)
apply (*simp add*:*surjec-def* [*of - - f*], *erule conjE*, *simp add*:*surj-to-def*,
rotate-tac −*1*, *frule sym*, *thin-tac f ' carrier L = carrier M*,
simp add:*image-def*, *erule bexE*, *simp*)
apply *blast*
done

lemma (**in** *Module*) *mId-mHom*:mId_M ∈ *mHom R M M*
apply (*simp add*:*mHom-def*)
apply (*rule conjI*)
apply (*simp add*:*aHom-def*)
apply (*rule conjI*)
apply (*simp add*:*mId-def*)
apply (*simp add*:*mId-def extensional-def*)
apply (*rule ballI*)+
apply (*simp add*:*ag-pOp-closed*)
apply (*rule ballI*)+
apply (*simp add*:*mId-def*)
apply (*simp add*:*sc-mem*)
done

lemma (**in** *Module*) *mHom-mId-bijec*:⟦*R module N*; *f* ∈ *mHom R M N*; *g* ∈ *mHom R N M*;
compose (*carrier M*) *g f* = mId_M; *compose* (*carrier N*) *f g* = mId_N⟧ ⟹
$bijec_{M,N}\ f$
apply (*simp add*:*bijec-def*)

apply (*rule conjI*)
apply (*simp add*:*injec-def*)
apply (*rule conjI*)
apply (*simp add*:*mHom-def*)
apply (*simp add*:*ker-def*)
apply (*rule equalityI*)
apply (*rule subsetI*, *simp*, *erule conjE*)
apply (*frule-tac* $x = f\ x$ **and** $y = \mathbf{0}_N$ **and** $f = g$ **in** *eq-elems-eq-val*)
apply (*frule-tac* $f = compose\ (carrier\ M)\ g\ f$ **and** $g = mId_M$ **and** $x = x$ **in**
eq-fun-eq-val, *thin-tac compose* (*carrier M*) *g f* = mId_M,
simp add:*compose-def*)
apply (*cut-tac Module.mHom-0*[*of N R M g*], *simp add*:*mId-def*, *assumption*,
rule Module-axioms, *assumption*)
apply (*rule subsetI*, *simp*,
simp add:*ag-inc-zero*, *simp add*:*mHom-0*)

apply (*simp add*:*surjec-def*)
apply (*rule conjI*, *simp add*:*mHom-def*)
apply (*rule surj-to-test*)
apply (*simp add*:*mHom-def aHom-def*)
apply (*rule ballI*)
apply (*frule-tac* $f = compose\ (carrier\ N)\ f\ g$ **and** $g = mId_N$ **and** $x = b$ **in**
eq-fun-eq-val, *thin-tac compose* (*carrier M*) *g f* = mId_M,
thin-tac compose (*carrier N*) *f g* = mId_N,
simp add:*compose-def*)
apply (*simp add*:*mId-def*)
apply (*frule-tac* $m = b$ **in** *Module.mHom-mem* [*of N R M g*], *rule Module-axioms*,
assumption+)
apply *blast*
done

definition
sup-sharp :: [('*r*, '*n*) *Ring-scheme*, ('*b*, '*r*, '*m1*) *Module-scheme*,
('*c*, '*r*, '*m2*) *Module-scheme*, ('*a*, '*r*, '*m*) *Module-scheme*, '*b* ⇒ '*c*]
⇒ ('*c* ⇒ '*a*) ⇒ ('*b* ⇒ '*a*) **where**
sup-sharp R M N L u = (λ*f*∈*mHom R N L*. *compos M f u*)

definition
sub-sharp :: [('*r*, '*n*) *Ring-scheme*, ('*a*, '*r*, '*m*) *Module-scheme*,
('*b*, '*r*, '*m1*) *Module-scheme*, ('*c*, '*r*, '*m2*) *Module-scheme*, '*b* ⇒ '*c*]
⇒ ('*a* ⇒ '*b*) ⇒ ('*a* ⇒ '*c*) **where**
sub-sharp R L M N u = (λ*f*∈*mHom R L M*. *compos L u f*)

lemma (**in** *Module*) *sup-sharp-homTr*:⟦*R module N*; *R module L*; *u* ∈ *mHom R M N*;
f ∈ *mHom R N L* ⟧ ⟹ *sup-sharp R M N L u f* ∈ *mHom R M L*
apply (*simp add*:*sup-sharp-def*)

apply (*rule Module.mHom-compos*, *assumption*, *rule Module-axioms*, *assumption*+)

done

lemma (**in** *Module*) *sup-sharp-hom*:⟦*R module N*; *R module L*; *u* ∈ *mHom R M N*⟧ ⟹
sup-sharp R M N L u ∈ *mHom R* (*HOM*$_R$ *N L*) (*HOM*$_R$ *M L*)
apply (*simp add*:*mHom-def* [*of R HOM*$_R$ *N L*])
apply (*rule conjI*)
apply (*simp add*:*aHom-def*)
apply (*rule conjI*)
apply (*simp add*:*HOM-def sup-sharp-homTr*)

apply (*rule conjI*)
apply (*simp add*:*sup-sharp-def extensional-def*,
rule allI, *rule impI*, *simp add*:*HOM-def*)

apply (*rule ballI*)+
apply (*simp add*:*HOM-def*)
apply (*frule-tac f* = *a* **and** *g* = *b* **in** *Module.tOp-mHom-closed*, *assumption*+)
apply (*subgoal-tac R module M*)
apply (*frule-tac f* = *a* **in** *Module.sup-sharp-homTr* [*of M R N L u*], *assumption*+)
apply (*frule-tac f* = *b* **in** *Module.sup-sharp-homTr* [*of M R N L u*], *assumption*+)
apply (*frule-tac f* = *tOp-mHom R N L a b* **in**
Module.sup-sharp-homTr[*of M R N L u*], *assumption*+)
apply (*rule Module.mHom-eq*, *assumption*+)
apply (*rule Module.tOp-mHom-closed*, *assumption*+)

apply (*rule ballI*)
apply (*simp add*:*sup-sharp-def tOp-mHom-def compose-def compos-def*)
apply (*simp add*:*mHom-mem*, *rule Module-axioms*)

apply (*rule ballI*)+
apply (*simp add*:*HOM-def*)
apply (*frule-tac a* = *a* **and** *f* = *m* **in** *Module.sprod-mHom-closed* [*of N R L*],
assumption+)
apply (*subgoal-tac R module M*,
frule-tac f = *sprod-mHom R N L a m* **in**
Module.sup-sharp-homTr [*of M R N L u*], *assumption*+)
apply (*frule-tac f* = *m* **in** *Module.sup-sharp-homTr* [*of M R N L u*], *assumption*+)
apply (*frule-tac a* = *a* **and** *f* = *sup-sharp R M N L u m* **in**
Module.sprod-mHom-closed [*of M R L*], *assumption*+)
apply (*rule mHom-eq*, *assumption*+)
apply (*rule ballI*)
apply (*simp add*:*sprod-mHom-def sup-sharp-def compose-def compos-def*)
apply (*simp add*:*Module.mHom-mem*, *rule Module-axioms*)
done

lemma (**in** *Module*) *sub-sharp-homTr*:⟦*R module N*; *R module L*; *u* ∈ *mHom R M*

N;
$f \in mHom\ R\ L\ M\rrbracket \Longrightarrow sub\text{-}sharp\ R\ L\ M\ N\ u\ f \in mHom\ R\ L\ N$
apply (*simp add*:*sub-sharp-def*)
apply (*simp add*:*mHom-compos*)
done

lemma (**in** *Module*) *sub-sharp-hom*:$\llbracket R\ module\ N;\ R\ module\ L;\ u \in mHom\ R\ M\ N\rrbracket \Longrightarrow$
$sub\text{-}sharp\ R\ L\ M\ N\ u \in mHom\ R\ (HOM_R\ L\ M)\ (HOM_R\ L\ N)$
apply (*simp add*:*mHom-def* [*of* - HOM_R *L M*])
apply (*rule conjI*)
apply (*simp add*:*aHom-def*)
apply (*rule conjI*)
apply (*simp add*:*HOM-def*)
apply (*simp add*:*sub-sharp-homTr*)

apply (*rule conjI*)
apply (*simp add*:*sub-sharp-def extensional-def*)
apply (*simp add*:*HOM-def*)

apply (*rule ballI*)+
apply (*simp add*:*HOM-def*)
apply (*frule-tac f* = *a* **and** *g* = *b* **in** *Module.tOp-mHom-closed* [*of L R M*],
rule Module-axioms, *assumption*+)
apply (*subgoal-tac R module M*)
apply (*frule-tac f* = *tOp-mHom R L M a b* **in** *Module.sub-sharp-homTr*
[*of M R N L u*], *assumption*+)
apply (*frule-tac f* = *b* **in** *Module.sub-sharp-homTr*[*of M R N L u*],
assumption+,
frule-tac f = *a* **in** *Module.sub-sharp-homTr*[*of M R N L u*], *assumption*+)
apply (*frule-tac f* = *sub-sharp R L M N u a* **and**
g = *sub-sharp R L M N u b* **in** *Module.tOp-mHom-closed* [*of L R N*],*assumption*+)
apply (*rule Module.mHom-eq*, *assumption*+)
apply (*rule ballI*)
apply (*simp add*:*tOp-mHom-def sub-sharp-def mcompose-def compose-def*,
simp add:*compos-def compose-def*)
apply (*rule Module.mHom-add* [*of M R*], *assumption*+)
apply (*simp add*:*Module.mHom-mem*, *simp add*:*Module.mHom-mem*)
apply (*rule Module-axioms*)

apply (*rule ballI*)+
apply (*simp add*:*HOM-def*)
apply (*subgoal-tac R module M*)
apply (*frule-tac a* = *a* **and** *f* = *m* **in** *Module.sprod-mHom-closed* [*of L R M*],
assumption+)
apply (*frule-tac f* = *sprod-mHom R L M a m* **in** *Module.sub-sharp-homTr*
[*of M R N L u*], *assumption*+)
apply (*frule-tac f* = *m* **in** *Module.sub-sharp-homTr*
[*of M R N L u*], *assumption*+)

apply (*frule-tac* $a = a$ **and** f = *sub-sharp* R L M N u m **in**
Module.sprod-mHom-closed [*of* L R N], *assumption*+)
apply (*rule Module.mHom-eq*, *assumption*+)
apply (*rule ballI*)
apply (*simp add*:*sprod-mHom-def sub-sharp-def mcompose-def compose-def*)
apply (*frule-tac* $f = m$ **and** $m = ma$ **in** *Module.mHom-mem* [*of* L R M], *assumption*+)
apply (*simp add*:*compos-def compose-def*)
apply (*simp add*:*mHom-lin*)
apply (*rule Module-axioms*)
done

lemma (**in** *Module*) *mId-bijec*:$bijec_{M,M}$ (mId_M)
apply (*simp add*:*bijec-def*)
apply (*cut-tac mId-mHom*)
apply (*rule conjI*)
apply (*simp add*:*injec-def*)
apply (*rule conjI*) **apply** (*simp add*:*mHom-def*)
apply (*simp add*:*ker-def*) **apply** (*simp add*:*mId-def*)
apply (*rule equalityI*) **apply** (*rule subsetI*, *simp*)
apply (*rule subsetI*, *simp*, *simp add*:*ag-inc-zero*)

apply (*simp add*:*surjec-def*)
apply (*rule conjI*, *simp add*:*mHom-def*)
apply (*rule surj-to-test*)
apply (*simp add*:*mHom-def aHom-def*)
apply (*rule ballI*)
apply (*simp add*:*mId-def*)
done

lemma (**in** *Module*) *invmfun-bijec*:⟦R *module* N; $f \in mHom\ R\ M\ N$; $bijec_{M,N}\ f$⟧
⟹
$bijec_{N,M}$ (*invmfun* R M N f)
apply (*frule invmfun-mHom* [*of* N f], *assumption*+)
apply (*simp add*:*bijec-def* [*of* N M])
apply (*rule conjI*)
apply (*simp add*:*injec-def*)
apply (*simp add*:*mHom-def* [*of* R N M]) **apply** (*erule conjE*)+
apply (*thin-tac* ∀ a∈*carrier* R.
∀ m∈*carrier* N. *invmfun* R M N f ($a \cdot_{sN} m$) = $a \cdot_s$ *invmfun* R M N f m)
apply (*rule equalityI*) **apply** (*rule subsetI*) **apply** (*simp add*:*ker-def CollectI*)
apply (*erule conjE*)
apply (*frule-tac* x = *invmfun* R M N f x **and** $y = \mathbf{0}$ **and** $f = f$ **in**
eq-elems-eq-val,
thin-tac invmfun R M N f $x = \mathbf{0}$)
apply (*simp add*:*invmfun-r-inv*)
apply (*simp add*:*mHom-0*)

apply (*rule subsetI*, *simp*)

apply (*simp add*:*ker-def*)
apply (*simp add*:*Module.module-inc-zero*)
apply (*cut-tac ag-inc-zero*,
frule invmfun-l-inv[*of N f* **0**], *assumption*+)
apply (*simp add*:*mHom-0*)

apply (*simp add*:*surjec-def*,
frule invmfun-mHom[*of N f*], *assumption*+)
apply (*rule conjI*, *simp add*:*mHom-def*)
apply (*simp add*:*surj-to-def*)
apply (*rule equalityI*, *rule subsetI*, *simp add*:*image-def*, *erule bexE*,
simp) **thm** *Module.mHom-mem*[*of N R M invmfun R M N f*]
apply (*rule Module.mHom-mem*[*of N R M invmfun R M N f*], *assumption*,
rule Module-axioms, *assumption*+)
apply (*rule subsetI*, *simp add*:*image-def*)
apply (*frule-tac m* = *x* **in** *invmfun-l-inv*[*of N f*], *assumption*+)
apply (*frule-tac m* = *x* **in** *mHom-mem*[*of N f*], *assumption*+)
apply (*frule sym*, *thin-tac invmfun R M N f* (*f x*) = *x*, *blast*)
done

lemma (**in** *Module*) *misom-self*:$M \cong_R M$
apply (*cut-tac mId-bijec*)
apply (*cut-tac mId-mHom*)
apply (*simp add*:*misomorphic-def*)
apply *blast*
done

lemma (**in** *Module*) *misom-sym*:⟦*R module N*; $M \cong_R N$⟧ ⟹ $N \cong_R M$
apply (*simp add*:*misomorphic-def* [*of R M N*])
apply (*erule exE*, *erule conjE*)
apply (*frule-tac f* = *f* **in** *invmfun-mHom* [*of N*], *assumption*+)
apply (*frule-tac f* = *f* **in** *invmfun-bijec* [*of N*], *assumption*+)
apply (*simp add*:*misomorphic-def*)
apply *blast*
done

lemma (**in** *Module*) *misom-trans*:⟦*R module L*; *R module N*; $L \cong_R M$; $M \cong_R N$⟧
⟹
$L \cong_R N$
apply (*simp add*:*misomorphic-def*)
apply ((*erule exE*)+, (*erule conjE*)+)
apply (*subgoal-tac* $bijec_{L,N}$ (*compos L fa f*))
apply (*subgoal-tac* (*compos L fa f*) ∈ *mHom R L N*)
apply *blast*
apply (*rule Module.mHom-compos*[*of M R L N*], *rule Module-axioms*, *assumption*+)

apply (*simp add*:*bijec-def*) **apply** (*erule conjE*)+
apply (*simp add*:*mcompos-inj-inj*)

apply (*simp add*:*mcompos-surj-surj*)
done

definition
mr-coset :: [*'a*, (*'a*, *'b*, *'more*) *Module-scheme*, *'a set*] ⇒ *'a set* **where**
mr-coset a M H = *a* ⊎$_M$ *H*

definition
set-mr-cos :: [(*'a*, *'b*, *'more*) *Module-scheme*, *'a set*] ⇒ *'a set set* **where**
set-mr-cos M H = {*X*. ∃ *a*∈*carrier M*. *X* = *a* ⊎$_M$ *H*}

definition
mr-cos-sprod :: [(*'a*, *'b*, *'more*) *Module-scheme*, *'a set*] ⇒
'b ⇒ *'a set* ⇒ *'a set* **where**
mr-cos-sprod M H a X = {*z*. ∃ *x*∈*X*. ∃ *h*∈*H*. *z* = *h* ±$_M$ (*a* ·$_{sM}$ *x*)}

definition
mr-cospOp :: [(*'a*, *'b*, *'more*) *Module-scheme*, *'a set*] ⇒
'a set ⇒ *'a set* ⇒ *'a set* **where**
mr-cospOp M H = (λ*X*. λ*Y*. *c-top* (*b-ag M*) *H X Y*)

definition
mr-cosmOp :: [(*'a*, *'b*, *'more*) *Module-scheme*, *'a set*] ⇒
'a set ⇒ *'a set* **where**
mr-cosmOp M H = (λ*X*. *c-iop* (*b-ag M*) *H X*)

definition
qmodule :: [(*'a*, *'r*, *'more*) *Module-scheme*, *'a set*] ⇒
(*'a set*, *'r*) *Module* **where**
qmodule M H = (| *carrier* = *set-mr-cos M H*, *pop* = *mr-cospOp M H*,
mop = *mr-cosmOp M H*, *zero* = *H*, *sprod* = *mr-cos-sprod M H* |)

definition
sub-mr-set-cos :: [(*'a*, *'r*, *'more*) *Module-scheme*, *'a set*, *'a set*] ⇒
'a set set **where**
sub-mr-set-cos M H N = {*X*. ∃ *n*∈*N*. *X* = *n* ⊎$_M$ *H*}

abbreviation
QMODULE (**infixl** *'/'*$_m$ *200*) **where**
M /$_m$ *H* == *qmodule M H*

abbreviation
SUBMRSET ((*3-*/ $_s$*'*/*'-*/ -) [*82*,*82*,*83*]*82*) **where**
N $_s$/$_M$ *H* == *sub-mr-set-cos M H N*

lemma (**in** *Module*) *qmodule-carr*:*submodule R M H* ⟹
carrier (*qmodule M H*) = *set-mr-cos M H*
apply (*simp add*:*qmodule-def*)

done

lemma (**in** *Module*) *set-mr-cos-mem*:⟦*submodule R M H*; $m \in$ *carrier M*⟧ ⟹
$m \uplus_M H \in$ *set-mr-cos M H*
apply (*simp add*:*set-mr-cos-def*)
apply *blast*
done

lemma (**in** *Module*) *mem-set-mr-cos*:⟦*submodule R M N*; $x \in$ *set-mr-cos M N*⟧
⟹
$\exists m \in$ *carrier M*. $x = m \uplus_M N$
by (*simp add*:*set-mr-cos-def*)

lemma (**in** *Module*) *m-in-mr-coset*:⟦*submodule R M H*; $m \in$ *carrier M*⟧ ⟹
$m \in m \uplus_M H$
apply (*cut-tac module-is-ag*)
apply (*frule aGroup.b-ag-group*)
apply (*simp add*:*ar-coset-def*)
apply (*simp add*:*aGroup.ag-carrier-carrier* [*THEN sym*])
apply (*simp add*:*submodule-def*) **apply** (*erule conjE*)+
apply (*simp add*:*asubGroup-def*)
apply (*rule Group.a-in-rcs* [*of b-ag M H m*], *assumption*+)
done

lemma (**in** *Module*) *mr-cos-h-stable*:⟦*submodule R M H*; $h \in H$⟧ ⟹
$H = h \uplus_M H$
apply (*cut-tac module-is-ag*)
apply (*frule aGroup.b-ag-group* [*of M*])
apply (*simp add*:*ar-coset-def*)
apply (*rule Group.rcs-Unit2*[*THEN sym*], *assumption*+,
simp add:*submodule-def*, (*erule conjE*)+,
simp add:*asubGroup-def*)
apply *assumption*
done

lemma (**in** *Module*) *mr-cos-h-stable1*:⟦*submodule R M H*; $m \in$ *carrier M*; $h \in H$⟧
⟹ $(m \pm h) \uplus_M H = m \uplus_M H$
apply (*cut-tac module-is-ag*)
apply (*subst aGroup.ag-pOp-commute*, *assumption*+)
apply (*simp add*:*submodule-def*, (*erule conjE*)+, *simp add*:*subsetD*)
apply (*frule aGroup.b-ag-group* [*of M*])
apply (*simp add*:*ar-coset-def*)
apply (*simp add*:*aGroup.agop-gop* [*THEN sym*])
apply (*simp add*:*aGroup.ag-carrier-carrier* [*THEN sym*])
apply (*simp add*:*submodule-def*, (*erule conjE*)+, *simp add*:*asubGroup-def*)
apply (*rule Group.rcs-fixed1* [*THEN sym*, *of b-ag M H m h*], *assumption*+)
done

lemma (**in** *Module*) *x-in-mr-coset*:⟦*submodule R M H*; $m \in$ *carrier M*; $x \in m \uplus_M$

H⟧
⟹ ∃ h∈H. $m \pm h = x$
apply (*cut-tac module-is-ag*)
apply (*frule aGroup.b-ag-group* [*of M*])
apply (*simp add*:*submodule-def*, (*erule conjE*)+,
simp add:*asubGroup-def*)
apply (*simp add*:*aGroup.ag-carrier-carrier* [*THEN sym*])
apply (*simp add*:*aGroup.agop-gop* [*THEN sym*])
apply (*simp add*:*ar-coset-def*)
apply (*frule Group.rcs-tool2*[*of b-ag M H m x*], *assumption*+,
erule bexE)
apply (*frule sym*, *thin-tac h* $\cdot_{b\text{-}ag\ M}$ *m* = *x*, *simp*)
apply (*simp add*:*aGroup.agop-gop*)
apply (*simp add*:*aGroup.ag-carrier-carrier*)
apply (*frule-tac c* = *h* **in** *subsetD*[*of H carrier M*], *assumption*+)
apply (*subst ag-pOp-commute*[*of* - *m*], *assumption*+)
apply *blast*
done

lemma (**in** *Module*) *mr-cos-sprodTr*:⟦*submodule R M H*; *a* ∈ *carrier R*;
m ∈ *carrier M*⟧ ⟹ *mr-cos-sprod M H a* (*m* $\uplus_M$ *H*) = (*a* $\cdot_s$ *m*) $\uplus_M$ *H*
apply (*cut-tac module-is-ag*,
frule aGroup.b-ag-group,
frule sc-mem[*of a m*], *assumption*)
apply (*simp add*:*ar-coset-def*,
simp add:*mr-cos-sprod-def*)
apply (*simp add*:*submodule-def*, (*erule conjE*)+)
apply (*simp add*:*aGroup.ag-carrier-carrier* [*THEN sym*],
simp add:*aGroup.agop-gop* [*THEN sym*])
apply (*simp add*:*asubGroup-def*)
apply (*rule equalityI*)
apply (*rule subsetI*, *simp*)
apply (*erule bexE*)+
apply (*frule-tac x* = *xa* **in** *Group.rcs-tool2*[*of b-ag M H m*], *assumption*+)
apply (*erule bexE*, *rotate-tac* −*1*, *frule sym*, *thin-tac ha* $\cdot_{b\text{-}ag\ M}$ *m* = *xa*,
simp)
apply (*simp add*:*aGroup.agop-gop*, *simp add*:*aGroup.ag-carrier-carrier*)
apply (*frule-tac c* = *ha* **in** *subsetD*[*of H carrier M*], *assumption*+,
simp add:*sc-r-distr*,
drule-tac x = *a* **in** *spec*,
drule-tac a = *ha* **in** *forall-spec*, *simp*,
frule-tac c = *a* $\cdot_s$ *ha* **in** *subsetD*[*of H carrier M*], *assumption*+,
frule-tac c = *h* **in** *subsetD*[*of H carrier M*], *assumption*+,
subst ag-pOp-assoc[*THEN sym*], *assumption*+)
apply (*simp add*:*aGroup.agop-gop*[*THEN sym*],
simp add:*aGroup.ag-carrier-carrier*[*THEN sym*])
apply (*frule-tac x* = *h* **and** *y* = *a* $\cdot_s$ *ha* **in**
Group.sg-mult-closed[*of b-ag M H*], *assumption*+)
apply (*frule-tac a* = *a* $\cdot_s$ *m* **and** *h* = *h* $\cdot_{b\text{-}ag\ M}$ (*a* $\cdot_s$ *ha*) **in**

Group.rcs-fixed1[*of b-ag M H*], *assumption*+)
apply *simp*
apply (*rule Group.a-in-rcs* [*of b-ag M H*], *assumption*+)
apply (*simp add*:*aGroup.agop-gop*, *simp add*:*aGroup.ag-carrier-carrier*)
apply (*rule ag-pOp-closed*, *simp add*:*subsetD*, *assumption*)

apply (*rule subsetI*, *simp*,
frule-tac x = *x* **in** *Group.rcs-tool2*[*of b-ag M H a* $\cdot_s$ *m*], *assumption*+,
erule bexE,
rotate-tac −*1*, *frule sym*, *thin-tac h* $\cdot_{b\text{-}ag\ M}$ (*a* $\cdot_s$ *m*) = *x*,
frule Group.a-in-rcs[*of b-ag M H m*], *assumption*+)
apply *blast*
done

lemma (**in** *Module*) *mr-cos-sprod-mem*:⟦*submodule R M H*; *a* ∈ *carrier R*;
X ∈ *set-mr-cos M H*⟧ ⟹ *mr-cos-sprod M H a X* ∈ *set-mr-cos M H*
apply (*simp add*:*set-mr-cos-def*)
apply (*erule bexE*, *rename-tac m*, *simp*)
apply (*subst mr-cos-sprodTr*, *assumption*+)
apply (*frule-tac m* = *m* **in** *sc-mem* [*of a*], *assumption*)
apply *blast*
done

lemma (**in** *Module*) *mr-cos-sprod-assoc*:⟦*submodule R M H*; *a* ∈ *carrier R*;
b ∈ *carrier R*; *X* ∈ *set-mr-cos M H*⟧ ⟹ *mr-cos-sprod M H* (*a* $\cdot_{r\,R}$ *b*) *X* =
mr-cos-sprod M H a (*mr-cos-sprod M H b X*)
apply (*simp add*:*set-mr-cos-def*, *erule bexE*, *simp*)
apply (*frule-tac m* = *aa* **in** *sc-mem* [*of b*], *assumption*)
apply (*cut-tac sc-Ring*,
frule Ring.ring-tOp-closed [*of R a b*], *assumption*+)
apply (*subst mr-cos-sprodTr*, *assumption*+)+
apply (*simp add*: *sc-assoc*)
done

lemma (**in** *Module*) *mr-cos-sprod-one*:⟦*submodule R M H*; *X* ∈ *set-mr-cos M H*⟧
⟹
mr-cos-sprod M H (1$_{r\,R}$) *X* = *X*
apply (*simp add*:*set-mr-cos-def*, *erule bexE*, *simp*,
thin-tac X = *a* ⊎$_M$ *H*)
apply (*cut-tac sc-Ring*,
frule Ring.ring-one[*of R*])
apply (*subst mr-cos-sprodTr*, *assumption*+)
apply (*simp add*:*sprod-one*)
done

lemma (**in** *Module*) *mr-cospOpTr*:⟦*submodule R M H*; *m* ∈ *carrier M*; *n* ∈ *carrier M*⟧
⟹ *mr-cospOp M H* (*m* ⊎$_M$ *H*) (*n* ⊎$_M$ *H*) = (*m* ± *n*) ⊎$_M$ *H*
apply (*cut-tac module-is-ag*, *frule aGroup.b-ag-group*)

apply (*simp add*:*mr-cospOp-def mr-coset-def agop-gop* [*THEN sym*])
apply (*simp add*:*ag-carrier-carrier* [*THEN sym*])

apply (*simp add*:*submodule-def*, (*erule conjE*)+,
frule aGroup.asubg-nsubg, *assumption*+, *simp add*:*ar-coset-def*)
apply (*simp add*:*Group.c-top-welldef*[*THEN sym*, *of b-ag M H m n*])
done

lemma(**in** *Module*) *mr-cos-sprod-distrib1*:⟦*submodule R M H*; *a* ∈ *carrier R*;
b ∈ *carrier R*; *X* ∈ *set-mr-cos M H*⟧ ⟹
mr-cos-sprod M H (*a* ±$_R$ *b*) *X* =
mr-cospOp M H (*mr-cos-sprod M H a X*) (*mr-cos-sprod M H b X*)
apply (*simp add*:*set-mr-cos-def*, *erule bexE*, *rename-tac m*)
apply *simp*
apply (*cut-tac sc-Ring*,
frule Ring.ring-is-ag[*of R*])
apply (*frule aGroup.ag-pOp-closed* [*of R a b*], *assumption*+)
apply (*subst mr-cos-sprodTr* [*of H*], *assumption*+)+
apply (*subst mr-cospOpTr*, *assumption*+)
apply (*simp add*:*sc-mem*, *simp add*:*sc-mem*)
apply (*simp add*:*sc-l-distr*)
done

lemma (**in** *Module*) *mr-cos-sprod-distrib2*:⟦*submodule R M H*;
a ∈ *carrier R*; *X* ∈ *set-mr-cos M H*; *Y* ∈ *set-mr-cos M H*⟧ ⟹
mr-cos-sprod M H a (*mr-cospOp M H X Y*) =
mr-cospOp M H (*mr-cos-sprod M H a X*) (*mr-cos-sprod M H a Y*)
apply (*simp add*:*set-mr-cos-def*, (*erule bexE*)+, *rename-tac m n*, *simp*,
thin-tac X = *m* ⊎$_M$ *H*, *thin-tac Y* = *n* ⊎$_M$ *H*)
apply (*subst mr-cos-sprodTr* [*of H*], *assumption*+)+
apply (*subst mr-cospOpTr*, *assumption*+)
apply (*subst mr-cospOpTr*, *assumption*+)
apply (*simp add*:*sc-mem*)+
apply (*subst mr-cos-sprodTr* [*of H*], *assumption*+)
apply (*rule ag-pOp-closed*, *assumption*+)
apply (*simp add*:*sc-r-distr*)
done

lemma (**in** *Module*) *mr-cosmOpTr*:⟦*submodule R M H*; *m* ∈ *carrier M*⟧ ⟹
mr-cosmOp M H (*m* ⊎$_M$ *H*) = (−$_a$ *m*) ⊎$_M$ *H*
apply (*simp add*:*ar-coset-def*)
apply (*cut-tac module-is-ag*)
apply (*frule aGroup.b-ag-group*)
apply (*simp add*:*ag-carrier-carrier* [*THEN sym*])
apply (*simp add*:*agiop-giop* [*THEN sym*])
apply (*simp add*:*mr-cosmOp-def*)
apply (*simp add*:*submodule-def*, (*erule conjE*)+,
frule aGroup.asubg-nsubg[*of M H*], *assumption*)
apply (*simp add*:*Group.c-iop-welldef*[*of b-ag M H m*])

done

lemma (**in** *Module*) *mr-cos-oneTr*:*submodule R M H* $\Longrightarrow$ *H* = **0** $\uplus_M$ *H*
apply (*cut-tac module-is-ag*,
  cut-tac ag-inc-zero)
apply (*simp add*:*ar-coset-def*)
apply (*frule aGroup.b-ag-group*)
apply (*simp add*:*ag-carrier-carrier* [*THEN sym*])
apply (*subst aGroup.agunit-gone*[*THEN sym*, *of M*], *assumption*)
apply (*subst Group.rcs-Unit1*, *assumption*)
apply (*simp add*:*submodule-def*, (*erule conjE*)+, *simp add*:*asubGroup-def*)
apply *simp*
done

lemma (**in** *Module*) *mr-cos-oneTr1*:⟦*submodule R M H*; *m* ∈ *carrier M*⟧ $\Longrightarrow$
  mr-cospOp M H H (*m* $\uplus_M$ *H*) = *m* $\uplus_M$ *H*
apply (*subgoal-tac mr-cospOp M H* (**0** $\uplus_M$ *H*) (*m* $\uplus_M$ *H*) = *m* $\uplus_M$ *H*)
apply (*simp add*:*mr-cos-oneTr* [*THEN sym*, *of H*])
apply (*subst mr-cospOpTr*, *assumption*+)
apply (*simp add*:*ag-inc-zero*)
apply *assumption*
apply (*simp add*:*ag-l-zero*)
done

lemma (**in** *Module*) *qmodule-is-ag*:*submodule R M H* $\Longrightarrow$ *aGroup* (*M* /$_m$ *H*)
apply (*cut-tac sc-Ring*)
apply (*rule aGroup.intro*)
apply (*simp add*:*qmodule-def*)
apply (*rule Pi-I*)+
apply (*rename-tac X Y*)
apply (*simp add*:*set-mr-cos-def*, (*erule bexE*)+, *rename-tac n m*, *simp*)
apply (*subst mr-cospOpTr*, *assumption*+,
  frule-tac x = *n* **and** *y* = *m* **in** *ag-pOp-closed*, *assumption*+, *blast*)

apply (*simp add*:*qmodule-def*)
apply (*simp add*:*set-mr-cos-def*, (*erule bexE*)+, *rename-tac a b c m n n'*)
apply (*simp add*:*mr-cospOpTr*,
  frule-tac x = *m* **and** *y* = *n* **in** *ag-pOp-closed*, *assumption*+,
  frule-tac x = *n* **and** *y* = *n'* **in** *ag-pOp-closed*, *assumption*+,
  simp add:*mr-cospOpTr*, *simp add*:*ag-pOp-assoc*)

apply (*simp add*:*qmodule-def*)
apply (*simp add*:*set-mr-cos-def*, (*erule bexE*)+, *rename-tac a b m n*, *simp*)
apply (*simp add*:*mr-cospOpTr*,
  simp add:*ag-pOp-commute*)

apply (*simp add*:*qmodule-def*,
  rule Pi-I,
  simp add:*set-mr-cos-def*, *erule bexE*, *simp*)

apply (*subst mr-cosmOpTr*, *assumption*+,
frule-tac $x = a$ **in** *ag-mOp-closed*, *blast*)

apply (*simp add*:*qmodule-def*,
simp add:*set-mr-cos-def*, *erule bexE*, *simp*,
simp add:*mr-cosmOpTr*,
frule-tac $x = aa$ **in** *ag-mOp-closed*)
apply (*simp add*:*mr-cospOpTr*,
frule-tac $x = -_a\ aa$ **and** $y = aa$ **in** *ag-pOp-closed*, *assumption*+,
simp add:*ag-l-inv1*, *simp add*:*mr-cos-oneTr*[*THEN sym*])
apply (*simp add*:*qmodule-def*,
simp add:*set-mr-cos-def*,
cut-tac mr-cos-oneTr[*of H*],
cut-tac ag-inc-zero, *blast*, *assumption*)

apply (*simp add*:*qmodule-def*)
apply (*simp add*:*set-mr-cos-def*, *erule bexE*, *simp*)
apply (*subgoal-tac mr-cospOp M H* $(\mathbf{0} \uplus_M H)$ $(aa \uplus_M H) = aa \uplus_M H$)
apply (*simp add*:*mr-cos-oneTr*[*THEN sym*, *of H*])
apply (*subst mr-cospOpTr*, *assumption*+,
simp add:*ag-inc-zero*, *assumption*, *simp add*:*ag-l-zero*)
done

lemma (**in** *Module*) *qmodule-module*:*submodule R M H* $\Longrightarrow$ *R module* $(M /_m H)$
apply (*rule Module.intro*)
apply (*simp add*:*qmodule-is-ag*)
apply (*rule Module-axioms.intro*)
apply (*cut-tac sc-Ring*, *simp*)

apply (*simp add*:*qmodule-def*)
apply (*simp add*:*mr-cos-sprod-mem*)

apply (*simp add*:*qmodule-def*)
apply (*simp add*:*mr-cos-sprod-distrib1*[*of H*])

apply (*simp add*:*qmodule-def*)
apply (*simp add*:*mr-cos-sprod-distrib2*[*of H*])

apply (*simp add*:*qmodule-def*)
apply (*simp add*:*mr-cos-sprod-assoc*)

apply (*simp add*:*qmodule-def*)
apply (*simp add*:*mr-cos-sprod-one*)
done

definition
indmhom :: [('*b*, '*m*) *Ring-scheme*, ('*a*, '*b*, '*m1*) *Module-scheme*,
('*c*, '*b*, '*m2*) *Module-scheme*, '*a* $\Rightarrow$ '*c*] $\Rightarrow$ '*a set* $\Rightarrow$ '*c* **where**
indmhom R M N f = $(\lambda X \in (set\text{-}mr\text{-}cos\ M\ (ker_{M,N}\ f)).\ f\ (\ SOME\ x.\ x \in X))$

abbreviation
INDMHOM ((*4*-$^\flat$ - -, -) [*92*,*92*,*92*,*93*]*92*) **where**
$f^\flat{}_{R\ M,N} == indmhom\ R\ M\ N\ f$

lemma (**in** *Module*) *indmhom-someTr*:⟦*R module N*; $f \in mHom\ R\ M\ N$;
$X \in$ *set-mr-cos M* ($ker_{M,N}\ f$)⟧ ⟹ f (*SOME xa*. $xa \in X$) $\in f$ ‘(*carrier M*)
apply (*simp add*:*set-mr-cos-def*)
apply (*erule bexE*, *simp*)
apply (*frule mker-submodule* [*of N f*], *assumption*+)
apply (*simp add*:*submodule-def*) **apply** (*erule conjE*)+
apply (*simp add*:*asubGroup-def*)
apply (*thin-tac* ∀ *a m*. $a \in$ *carrier R* ∧ $m \in ker_{M,N}\ f \longrightarrow a \cdot_s m \in ker_{M,N}\ f$)
apply (*cut-tac module-is-ag*)
apply (*frule aGroup.b-ag-group*)
apply (*rule someI2-ex*)
apply (*simp add*:*ar-coset-def*)
apply (*frule-tac a* = *a* **in** *Group.a-in-rcs*[*of b-ag M* $ker_{M,N}\ f$],
assumption+, *simp add*:*ag-carrier-carrier* [*THEN sym*], *blast*)
apply (*simp add*:*ar-coset-def*)
apply (*frule-tac a* = *a* **and** *x* = *x* **in**
Group.rcs-subset-elem[*of b-ag M* $ker_{M,N}\ f$], *assumption*+)
apply (*simp add*:*ag-carrier-carrier*, *assumption*+)

apply (*simp add*:*image-def*,
simp add:*ag-carrier-carrier*, *blast*)
done

lemma (**in** *Module*) *indmhom-someTr1*:⟦*R module N*; $f \in mHom\ R\ M\ N$; $m \in$ *carrier M*⟧
⟹ f (*SOME xa*. $xa \in$ (*ar-coset m M* ($ker_{M,N}\ f$))) = $f\ m$
apply (*rule someI2-ex*)
apply (*frule mker-submodule*[*of N f*], *assumption*)
apply (*frule-tac m-in-mr-coset*[*of* $ker_{M,N}\ f\ m$], *assumption*+,
blast)

apply (*frule mker-submodule* [*of N f*], *assumption*+)
apply (*frule-tac x* = *x* **in** *x-in-mr-coset* [*of* $ker_{M,N}\ f\ m$],
assumption+, *erule bexE*,
frule sym , *thin-tac m* ± *h* = *x*, *simp*)
apply (*simp add*:*ker-def*, *erule conjE*)
apply (*subst mHom-add*[*of N f*], *assumption*+, *simp*)
apply (*frule Module.module-is-ag* [*of N R*])
apply (*frule mHom-mem* [*of N f m*], *assumption*+)
apply (*simp add*:*aGroup.ag-r-zero*)
done

lemma (**in** *Module*) *indmhom-someTr2*:⟦*R module N*; $f \in mHom\ R\ M\ N$;

$submodule\ R\ M\ H;\ m \in carrier\ M;\ H \subseteq ker_{M,N}\ f\rrbracket \Longrightarrow$
$f\ (SOME\ xa.\ xa \in m \uplus_M H) = f\ m$
apply (*rule someI2-ex*)
apply (*frule-tac m-in-mr-coset*[*of H m*], *assumption*+, *blast*)
apply (*frule-tac x = x* **in** *x-in-mr-coset* [*of H m*],
assumption+, *erule bexE*,
frule sym , *thin-tac m ± h = x*, *simp*)
apply (*frule-tac c = h* **in** *subsetD*[*of H ker*$_{M,N}$ *f*], *assumption*+)
apply (*frule mker-submodule* [*of N f*], *assumption*+,
simp add:*submodule-def*[*of R M ker*$_{M,N}$ *f*], (*erule conjE*)+,
frule-tac c = h **in** *subsetD*[*of ker*$_{M,N}$ *f carrier M*], *assumption*+)
apply (*simp add*:*ker-def mHom-add*,
frule-tac m = m **in** *mHom-mem*[*of N f*], *assumption*+)
apply (*frule Module.module-is-ag*[*of N R*])
apply (*simp add*:*aGroup.ag-r-zero*)
done

lemma (**in** *Module*) *indmhomTr1*:$\llbracket R$ *module* $N;\ f \in mHom\ R\ M\ N;\ m \in carrier\ M\rrbracket \Longrightarrow$
$(f^\flat{}_{R\ M,N})\ (m \uplus_M (ker_{M,N}\ f)) = f\ m$
apply (*simp add*:*indmhom-def*)
apply (*subgoal-tac m* $\uplus_M$ *ker*$_{M,N}$ *f* $\in$ *set-mr-cos M* (*ker*$_{M,N}$ *f*), *simp*)
apply (*rule indmhom-someTr1*, *assumption*+)
apply (*rule set-mr-cos-mem*)
apply (*rule mker-submodule*, *assumption*+)
done

lemma (**in** *Module*) *indmhomTr2*:$\llbracket R$ *module* $N;\ f \in mHom\ R\ M\ N\rrbracket$
$\Longrightarrow (f^\flat{}_{R\ M,N}) \in set\text{-}mr\text{-}cos\ M\ (ker_{M,N}\ f) \to carrier\ N$
apply (*rule Pi-I*)
apply (*simp add*:*set-mr-cos-def*)
apply (*erule bexE*)
apply (*frule-tac m = a* **in** *indmhomTr1* [*of N f*], *assumption*+)
apply (*simp add*:*mHom-mem*)
done

lemma (**in** *Module*) *indmhom*:$\llbracket R$ *module* $N;\ f \in mHom\ R\ M\ N\rrbracket$
$\Longrightarrow (f^\flat{}_{R\ M,N}) \in mHom\ R\ (M\ /_m\ (ker_{M,N}\ f))\ N$
apply (*simp add*:*mHom-def* [*of R M* $/_m$ (*ker*$_{M,N}$ *f*) *N*])
apply (*rule conjI*)
apply (*simp add*:*aHom-def*)
apply (*rule conjI*)
apply (*simp add*:*qmodule-def*)
apply (*simp add*:*indmhomTr2*)

apply (*rule conjI*)
apply (*simp add*:*qmodule-def indmhom-def extensional-def*)

apply (*rule ballI*)+

apply (*simp add*:*qmodule-def*)
apply (*simp add*:*set-mr-cos-def*, (*erule bexE*)+, *simp*, *rename-tac m n*)
apply (*frule mker-submodule* [*of N f*], *assumption*+,
simp add:*mr-cospOpTr*,
frule-tac x = *m* **and** *y* = *n* **in** *ag-pOp-closed*, *assumption*+)
apply (*simp add*:*indmhomTr1*, *simp add*:*mHom-add*)

apply (*rule ballI*)+
apply (*simp add*:*qmodule-def*)
apply (*simp add*:*set-mr-cos-def*, (*erule bexE*)+, *simp*)
apply (*frule mker-submodule* [*of N f*], *assumption*+,
subst mr-cos-sprodTr [*of* $ker_{M,N}\ f$], *assumption*+,
frule-tac a = *a* **and** *m* = *aa* **in** *sc-mem*, *assumption*)
apply (*simp add*:*indmhomTr1*)
apply (*simp add*:*mHom-lin*)
done

lemma (**in** *Module*) *indmhom-injec*:⟦*R module N*; *f* ∈ *mHom R M N*⟧ ⟹
$injec_{(M\ /_m\ (ker_{M,N}\ f)),N}\ (f^\flat\ R\ M,N)$
apply (*simp add*:*injec-def*)
apply (*frule indmhom* [*of N f*], *assumption*+)
apply (*rule conjI*)
apply (*simp add*:*mHom-def*)
apply (*simp add*:*ker-def* [*of* - - $f^\flat R\ M,\ N$])
apply (*simp add*:*qmodule-def*) **apply** (*fold qmodule-def*)
apply (*rule equalityI*)
apply (*rule subsetI*) **apply** (*simp add*:*CollectI*) **apply** (*erule conjE*)
apply (*simp add*:*set-mr-cos-def*, *erule bexE*, *simp*)
apply (*simp add*:*indmhomTr1*)
apply (*frule mker-submodule* [*of N f*], *assumption*+)
apply (*rule-tac h1* = *a* **in** *mr-cos-h-stable* [*THEN sym*, *of* $ker_{M,N}\ f$],
assumption+)
apply (*simp add*:*ker-def*)

apply (*rule subsetI*) **apply** (*simp add*:*CollectI*)
apply (*rule conjI*)
apply (*simp add*:*set-mr-cos-def*)
apply (*frule mker-submodule* [*of N f*], *assumption*+)
apply (*frule mr-cos-oneTr* [*of* $ker_{M,N}\ f$])
apply (*cut-tac ag-inc-zero*)
apply *blast*
apply (*frule mker-submodule* [*of N f*], *assumption*+)
apply (*subst mr-cos-oneTr* [*of* $ker_{M,N}\ f$], *assumption*)
apply (*cut-tac ag-inc-zero*)
apply (*subst indmhomTr1*, *assumption*+)
apply (*simp add*:*mHom-0*)
done

lemma (**in** *Module*) *indmhom-surjec1*:⟦*R module N*; $surjec_{M,N}\ f$;

$f \in mHom\ R\ M\ N\rrbracket \Longrightarrow surjec_{(M\ /_m\ (ker_{M,N}\ f)),N}\ (f^\flat_{R\ M,N})$
apply (*simp add*:*surjec-def*)
apply (*frule indmhom* [*of N f*], *assumption*+)
apply (*rule conjI*)
apply (*simp add*:*mHom-def*)
apply (*rule surj-to-test*)
apply (*simp add*:*mHom-def aHom-def*)
apply (*rule ballI*)
apply (*erule conjE*)
apply (*simp add*:*surj-to-def*, *frule sym* , *thin-tac f ' carrier M = carrier N*,
simp,
thin-tac carrier N = f ' carrier M)
apply (*simp add*:*image-def*, *erule bexE*, *simp*)
apply (*frule-tac m = x* **in** *indmhomTr1* [*of N f*], *assumption*+)
apply (*frule mker-submodule* [*of N f*], *assumption*+)
apply (*simp add*:*qmodule-carr*)
apply (*frule-tac m = x* **in** *set-mr-cos-mem* [*of ker*$_{M,N}$ *f*], *assumption*+)
apply *blast*
done

lemma (**in** *Module*) *module-homTr*:$\llbracket R$ *module* N; $f \in mHom\ R\ M\ N\rrbracket \Longrightarrow$
$f \in mHom\ R\ M\ (mimg_{R\ M,N}\ f)$
apply (*subst mHom-def*, *simp add*:*CollectI*)
apply (*rule conjI*)
apply (*simp add*:*aHom-def*)
apply (*rule conjI*)
apply (*simp add*:*mimg-def mdl-def*)
apply (*rule conjI*)
apply (*simp add*:*mHom-def aHom-def extensional-def*)
apply (*rule ballI*)+
apply (*simp add*:*mimg-def mdl-def*)
apply (*simp add*:*mHom-add*)
apply (*rule ballI*)+
apply (*simp add*:*mimg-def mdl-def*)
apply (*simp add*:*mHom-lin*)
done

lemma (**in** *Module*) *ker-to-mimg*:$\llbracket R$ *module* N; $f \in mHom\ R\ M\ N\rrbracket \Longrightarrow$
$ker_{M,mimg_{R\ M,N}\ f}\ f = ker_{M,N}\ f$
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*simp add*:*ker-def mimg-def mdl-def*)
apply (*rule subsetI*)
apply (*simp add*:*ker-def mimg-def mdl-def*)
done

lemma (**in** *Module*) *module-homTr1*:$\llbracket R$ *module* N; $f \in mHom\ R\ M\ N\rrbracket \Longrightarrow$
$(mimg_{R\ (M\ /_m\ (ker_{M,N}\ f)),N}\ (f^\flat_{R\ M,N})) = mimg_{R\ M,N}\ f$ **apply** (*simp*

add:*mimg-def*)
apply (*subgoal-tac* $f^{\flat}{}_{R\ M,\ N}$ ‘ *carrier* (*M* $/_m$ ($ker_{M,N}$ *f*)) = *f* ‘ *carrier M* ,
simp)
apply (*simp add*:*qmodule-def*)
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*simp add*:*image-def set-mr-cos-def*)
apply (*erule exE*, *erule conjE*, *erule bexE*, *simp*)
apply (*simp add*:*indmhomTr1*, *blast*)
apply (*rule subsetI*,
simp add:*image-def set-mr-cos-def*, *erule bexE*, *simp*)
apply (*frule-tac m1* = *xa* **in** *indmhomTr1* [*THEN sym*, *of N f*],
assumption+)
apply *blast*
done

lemma (**in** *Module*) *module-Homth-1*:⟦*R module N*; *f* ∈ *mHom R M N*⟧ ⟹
M $/_m$ ($ker_{M,N}$ *f*) ≅$_R$ $mimg_{R\ M,N}$ *f*
apply (*frule surjec-to-mimg*[*of N f*], *assumption*,
frule module-homTr[*of N f*], *assumption*,
frule mimg-module[*of N f*], *assumption*,
frule indmhom-surjec1[*of* $mimg_{R\ M,N}$ *f f*], *assumption*+,
frule indmhom-injec[*of* $mimg_{R\ M,N}$ *f f*], *assumption*+,
frule indmhom[*of* $mimg_{R\ M,N}$ *f f*], *assumption*+)
apply (*simp add*:*misomorphic-def*,
simp add:*bijec-def*)
apply (*simp add*:*ker-to-mimg*)
apply *blast*
done

definition
mpj :: [('*a*, '*r*, '*m*) *Module-scheme*, '*a set*] ⇒ ('*a* => '*a set*) **where**
mpj M H = (λ*x*∈*carrier M*. *x* ⊎$_M$ *H*)

lemma (**in** *Module*) *elem-mpj*:⟦*m* ∈ *carrier M*; *submodule R M H*⟧ ⟹
mpj M H m = *m* ⊎$_M$ *H*
by (*simp add*:*mpj-def*)

lemma (**in** *Module*) *mpj-mHom*:*submodule R M H* ⟹ *mpj M H* ∈ *mHom R M*
(*M* $/_m$ *H*)
apply (*simp add*:*mHom-def*)
apply (*rule conjI*)
apply (*simp add*:*aHom-def*)
apply (*rule conjI*)
apply (*simp add*:*mpj-def qmodule-carr set-mr-cos-mem*)
apply (*rule conjI*)
apply (*simp add*:*mpj-def extensional-def*)
apply (*rule ballI*)+
apply (*simp add*:*qmodule-def*)

apply (*simp add*:*mpj-def*, *simp add*:*ag-pOp-closed*)
apply (*simp add*:*mr-cospOpTr*)
apply (*rule ballI*)+
apply (*simp add*:*mpj-def sc-mem*)
apply (*simp add*:*qmodule-def*)
apply (*simp add*:*mr-cos-sprodTr*)
done

lemma (**in** *Module*) *mpj-mem*:⟦*submodule R M H*; *m* ∈ *carrier M*⟧ ⟹
mpj M H m ∈ *carrier* (*M* /$_m$ *H*)
apply (*frule mpj-mHom*[*of H*])
apply (*rule mHom-mem* [*of M* /$_m$ *H mpj M H m*])
apply (*simp add*:*qmodule-module*) **apply** *assumption*+
done

lemma (**in** *Module*) *mpj-surjec*:*submodule R M H* ⟹
surjec$_{M,(M\ /_m\ H)}$ (*mpj M H*)
apply (*simp add*:*surjec-def*)
apply (*frule mpj-mHom* [*of H*])
apply (*rule conjI*, *simp add*:*mHom-def*)
apply (*rule surj-to-test*,
simp add:*mHom-def aHom-def*)
apply (*rule ballI*)
apply (*thin-tac mpj M H* ∈ *mHom R M* (*M* /$_m$ *H*))

apply (*simp add*:*qmodule-def*)
apply (*simp add*:*set-mr-cos-def*, *erule bexE*, *simp*)
apply (*frule-tac m* = *a* **in** *elem-mpj*[*of* - *H*], *assumption*, *blast*)
done

lemma (**in** *Module*) *mpj-0*:⟦*submodule R M H*; *h* ∈ *H*⟧ ⟹
mpj M H h = **0**$_{(M\ /_m\ H)}$
apply (*simp add*:*submodule-def*, (*erule conjE*)+)
apply (*frule-tac c* = *h* **in** *subsetD*[*of H carrier M*], *assumption*+)
apply (*subst elem-mpj*[*of* - *H*], *assumption*+,
simp add:*submodule-def*)
apply (*simp add*:*qmodule-def*)
apply (*rule mr-cos-h-stable*[*THEN sym*],
simp add:*submodule-def*, *assumption*)
done

lemma (**in** *Module*) *mker-of-mpj*:*submodule R M H* ⟹
ker$_{M,(M\ /_m\ H)}$ (*mpj M H*) = *H*
apply (*simp add*:*ker-def*)
apply (*rule equalityI*)
apply (*rule subsetI*, *simp*, *erule conjE*)
apply (*simp add*:*elem-mpj*, *simp add*:*qmodule-def*)
apply (*frule-tac m* = *x* **in** *m-in-mr-coset* [*of H*], *assumption*+)
apply *simp*

apply (*rule subsetI*)
 apply *simp*
 apply (*simp add*:*submodule-def*, (*erule conjE*)+)
 apply (*simp add*:*subsetD*)
 apply (*subst elem-mpj*,
 simp add:*subsetD*, *simp add*:*submodule-def*)
 apply (*simp add*:*qmodule-def*)
 apply (*rule mr-cos-h-stable*[*THEN sym*],
 simp add:*submodule-def*, *assumption*)
done

lemma (**in** *Module*) *indmhom1*:⟦*submodule R M H*; *R module N*; $f \in mHom\ R\ M\ N$; $H \subseteq ker_{M,N}\ f$⟧ ⟹ ∃!*g*. $g \in (mHom\ R\ (M\ /_m\ H)\ N) \wedge (compos\ M\ g\ (mpj\ M\ H)) = f$
apply (*rule ex-ex1I*)
apply (*subgoal-tac* $(\lambda X{\in}set\text{-}mr\text{-}cos\ M\ H.\ f\ (SOME\ x.\ x \in X)) \in mHom\ R\ (M\ /_m\ H)\ N \wedge compos\ M\ (\lambda X{\in}set\text{-}mr\text{-}cos\ M\ H.\ f\ (SOME\ x.\ x \in X))\ (mpj\ M\ H) = f$)
apply *blast*
 apply (*rule conjI*)
 apply (*rule Module.mHom-test*)
 apply (*simp add*:*qmodule-module*, *assumption*+)
 apply (*rule conjI*)
 apply (*rule Pi-I*)
 apply (*simp add*:*qmodule-def*, *simp add*:*set-mr-cos-def*, *erule bexE*, *simp*)
 apply (*simp add*:*indmhom-someTr2*, *simp add*:*mHom-mem*)

 apply (*rule conjI*)
 apply (*simp add*:*qmodule-def*)

 apply (*rule conjI*, (*rule ballI*)+)
 apply (*simp add*:*qmodule-def*, *simp add*:*set-mr-cos-def*, (*erule bexE*)+, *simp*)
 apply (*simp add*:*mr-cospOpTr*,
 frule-tac x = *a* **and** *y* = *aa* **in** *ag-pOp-closed*, *assumption*+)
 apply (*simp add*:*indmhom-someTr2*, *simp add*:*mHom-add*)
 apply (*rule impI*)
 apply (*frule-tac x* = *a* ± *aa* **in** *bspec*, *assumption*+, *simp*)

 apply ((*rule ballI*)+,
 simp add:*qmodule-def*, *simp add*:*set-mr-cos-def*, *erule bexE*, *simp*,
 simp add:*mr-cos-sprodTr*,
 frule-tac a = *a* **and** *m* = *aa* **in** *sc-mem*, *assumption*)
 apply (*simp add*:*indmhom-someTr2*, *simp add*:*mHom-lin*,
 rule impI,
 frule-tac x = *a* ·$_s$ *aa* **in** *bspec*, *assumption*, *simp*)
 apply (*rule mHom-eq*[*of N - f*], *assumption*)
 apply (*rule Module.mHom-compos*[*of M* /$_m$ *H R M N mpj M H*
 $\lambda X{\in}set\text{-}mr\text{-}cos\ M\ H.\ f\ (SOME\ x.\ x \in X)$]) **apply** (
 simp add:*qmodule-module*, *rule Module-axioms*, *assumption*,

simp add:*mpj-mHom*)
apply (*rule Module.mHom-test*,
simp add:*qmodule-module*, *assumption*)
apply (*rule conjI*,
rule Pi-I,
clarsimp simp: *qmodule-def set-mr-cos-def indmhom-someTr2 mHom-mem*)
apply (*rule conjI*,
simp add:*qmodule-def*)
apply (*rule ballI*)+, *simp add*:*qmodule-def*, *simp add*:*set-mr-cos-def*,
(*erule bexE*)+, *simp add*:*mr-cospOpTr*,
frule-tac x = *a* **and** *y* = *aa* **in** *ag-pOp-closed*, *assumption*+,
simp add:*indmhom-someTr2 mHom-add*,
rule impI,
frule-tac x = *a* ± *aa* **in** *bspec*, *assumption*, *simp*)
apply ((*rule ballI*)+, *simp add*:*qmodule-def set-mr-cos-def*, *erule bexE*, *simp*,
simp add:*mr-cos-sprodTr*,
frule-tac a = *a* **and** *m* = *aa* **in** *sc-mem*, *assumption*,
simp add:*indmhom-someTr2 mHom-lin*,
rule impI,
frule-tac x = *a* ·$_s$ *aa* **in** *bspec*, *assumption*, *simp*,
assumption+)
apply (*rule ballI*, *simp add*:*compos-def compose-def elem-mpj*,
simp add:*indmhom-someTr2*,
rule impI, *simp add*:*set-mr-cos-def*,
frule-tac x = *m* **in** *bspec*, *assumption*, *simp*)

apply (*erule conjE*)+
apply (*rule-tac f* = *g* **and** *g* = *y* **in** *Module.mHom-eq*[*of M* /$_m$ *H R N*],
simp add:*qmodule-module*, *assumption*+)
apply (*rule ballI*, *simp add*:*qmodule-def*, *fold qmodule-def*,
simp add:*set-mr-cos-def*, *erule bexE*, *simp*)
apply (*rotate-tac* −*3*, *frule sym*, *thin-tac compos M y* (*mpj M H*) = *f*,
simp)
apply (*frule-tac f* = *compos M g* (*mpj M H*) **and** *g* = *compos M y* (*mpj M H*)
and *x* = *a* **in** *eq-fun-eq-val*,
thin-tac compos M g (*mpj M H*) = *compos M y* (*mpj M H*))
apply (*simp add*:*compos-def compose-def elem-mpj*)
done

definition
mQmp :: [('*a*, '*r*, '*m*) *Module-scheme*, '*a set*, '*a set*] ⇒
('*a set* ⇒ '*a set*) **where**
mQmp M H N = (λ*X*∈ *set-mr-cos M H*. {*z*. ∃ *x* ∈ *X*. ∃ *y* ∈ *N*. (*y* ±$_M$ *x* = *z*)})

abbreviation
MQP ((*3Mp*_ _,_) [*82*,*82*,*83*]*82*) **where**

$Mp_{M\ H,N} == mQmp\ M\ H\ N$

lemma (**in** *Module*) *mQmpTr0*:⟦*submodule R M H*; *submodule R M N*; $H \subseteq N$;
$m \in carrier\ M$⟧ ⟹ $mQmp\ M\ H\ N\ (m \uplus_M H) = m \uplus_M N$
apply (*frule set-mr-cos-mem* [*of H m*], *assumption*+)
apply (*simp add*:*mQmp-def*)
apply (*rule equalityI*)
apply (*rule subsetI*, *simp*, (*erule bexE*)+, *rotate-tac* −*1*, *frule sym*,
thin-tac $y \pm xa = x$, *simp*)
apply (*frule-tac* $x = xa$ **in** *x-in-mr-coset*[*of H m*], *assumption*+, *erule bexE*,
rotate-tac −*1*, *frule sym*, *thin-tac* $m \pm h = xa$, *simp*)
apply (*unfold submodule-def*, *frule conjunct1*, *rotate-tac 1*, *frule conjunct1*,
fold submodule-def,
frule-tac $c = y$ **in** *subsetD*[*of N carrier M*], *assumption*+,
frule-tac $c = h$ **in** *subsetD*[*of H carrier M*], *assumption*+,
simp add:*ag-pOp-assoc*[*THEN sym*],
simp add:*ag-pOp-commute*[*of - m*], *simp add*:*ag-pOp-assoc*,
frule-tac $c = h$ **in** *subsetD*[*of H N*], *assumption*+,
frule-tac $h = y$ **and** $k = h$ **in** *submodule-pOp-closed*[*of N*], *assumption*+,
frule-tac $h1 = y \pm h$ **in** *mr-cos-h-stable1*[*THEN sym*, *of N m*],
assumption+, *simp*)
apply (*rule m-in-mr-coset*, *assumption*+,
rule ag-pOp-closed, *assumption*+, *simp add*:*subsetD*)

apply (*rule subsetI*, *simp*,
frule-tac $x = x$ **in** *x-in-mr-coset*[*of N m*], *assumption*+,
erule bexE, *frule sym*, *thin-tac* $m \pm h = x$, *simp*,
simp add:*submodule-def*[*of R M N*], *frule conjunct1*, *fold submodule-def*,
frule-tac $c = h$ **in** *subsetD*[*of N carrier M*], *assumption*+)
apply (*frule-tac m-in-mr-coset*[*of H m*], *assumption*+,
subst ag-pOp-commute[*of m*], *assumption*+)
apply *blast*
done

lemma (**in** *Module*) *mQmpTr1*:⟦*submodule R M H*; *submodule R M N*; $H \subseteq N$;
$m \in carrier\ M$; $n \in carrier\ M$; $m \uplus_M H = n \uplus_M H$⟧ ⟹ $m \uplus_M N = n \uplus_M N$
apply (*frule-tac m-in-mr-coset* [*of H m*], *assumption*+)
apply *simp*
apply (*frule-tac x-in-mr-coset* [*of H n m*], *assumption*+)
apply (*erule bexE*, *rotate-tac* −*1*, *frule sym*, *thin-tac* $n \pm h = m$, *simp*)
apply (*frule-tac* $c = h$ **in** *subsetD* [*of H N*], *assumption*+)
apply (*rule mr-cos-h-stable1*[*of N n*], *assumption*+)
done

lemma (**in** *Module*) *mQmpTr2*:⟦*submodule R M H*; *submodule R M N*; $H \subseteq N$;
$X \in carrier\ (M\ /_m\ H)$⟧ ⟹ $(mQmp\ M\ H\ N)\ X \in carrier\ (M\ /_m\ N)$

apply (*simp add*:*qmodule-def*)
apply (*simp add*:*set-mr-cos-def*)
apply (*erule bexE*, *simp*)
apply (*frule-tac m* = *a* **in** *mQmpTr0* [*of H N*], *assumption*+)
apply *blast*
done

lemma (**in** *Module*) *mQmpTr2-1*:⟦*submodule R M H*; *submodule R M N*; *H* ⊆ *N* ⟧
⟹ *mQmp M H N* ∈ *carrier* (*M* /$_m$ *H*) → *carrier* (*M* /$_m$ *N*)
by (*simp add*:*mQmpTr2*)

lemma (**in** *Module*) *mQmpTr3*:⟦*submodule R M H*; *submodule R M N*; *H* ⊆ *N* ;
X ∈ *carrier* (*M* /$_m$ *H*); *Y* ∈ *carrier* (*M* /$_m$ *H*)⟧ ⟹ (*mQmp M H N*) (*mr-cospOp M H X Y*) = *mr-cospOp M N* ((*mQmp M H N*) *X*) ((*mQmp M H N*) *Y*)
apply (*simp add*:*qmodule-def*)
apply (*simp add*:*set-mr-cos-def*)
apply ((*erule bexE*)+, *simp*)
apply (*simp add*:*mr-cospOpTr*)
apply (*frule-tac x* = *a* **and** *y* = *aa* **in** *ag-pOp-closed*, *assumption*+)
apply (*subst mQmpTr0*, *assumption*+)+
apply (*subst mr-cospOpTr*, *assumption*+)
apply *simp*
done

lemma (**in** *Module*) *mQmpTr4*:⟦*submodule R M H*; *submodule R M N*; *H* ⊆ *N*;
a ∈ *N*⟧ ⟹ *mr-coset a* (*mdl M N*) *H* = *mr-coset a M H*
apply (*simp add*:*mr-coset-def*)
apply (*unfold submodule-def*[*of R M N*], *frule conjunct1*, *fold submodule-def*,
frule subsetD[*of N carrier M a*], *assumption*+)
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*frule mdl-is-module*[*of N*])
apply (*frule-tac x* = *x* **in** *Module.x-in-mr-coset*[*of mdl M N R H a*])
apply (*simp add*:*submodule-of-mdl*)
apply (*simp add*:*mdl-carrier*)
apply *assumption*+
apply (*erule bexE*)
apply (*unfold submodule-def*[*of R M H*], *frule conjunct1*, *fold submodule-def*)
apply (*frule-tac c* = *h* **in** *subsetD*[*of H carrier M*], *assumption*+)
apply (*thin-tac x* ∈ *a* ⊎$_{mdl\ M\ N}$ *H*, *thin-tac R module mdl M N*,
simp add:*mdl-def*)
apply (*frule sym*, *thin-tac a* ± *h* = *x*, *simp*)
apply (*subst mr-cos-h-stable1*[*THEN sym*, *of H a*], *assumption*+)
apply (*frule-tac x* = *a* **and** *y* = *h* **in** *ag-pOp-closed*, *assumption*+)
apply (*rule m-in-mr-coset*, *assumption*+)

apply (*rule subsetI*)
apply (*frule-tac x* = *x* **in** *x-in-mr-coset*[*of H a*], *assumption*+)

apply (*erule bexE*, *frule sym*, *thin-tac* $a \pm h = x$, *simp*)
apply (*frule mdl-is-module*[*of N*])
apply (*frule submodule-of-mdl*[*of H N*], *assumption*+)
apply (*subst Module.mr-cos-h-stable1*[*THEN sym*, *of mdl M N R H a*],
assumption+, *simp add*:*mdl-carrier*, *simp*)
apply (*subgoal-tac* $a \pm h = a \pm_{mdl\ M\ N} h$, *simp*)
apply (*rule Module.m-in-mr-coset*[*of mdl M N R H*], *assumption*+)
apply (*frule Module.module-is-ag*[*of mdl M N R*])
apply (*rule aGroup.ag-pOp-closed*, *assumption*,
simp add:*mdl-carrier*, *simp add*:*mdl-carrier subsetD*)
apply (*subst mdl-def*, *simp*)
done

lemma (**in** *Module*) *mQmp-mHom*:⟦*submodule R M H*; *submodule R M N*; $H \subseteq N$⟧ ⟹
$(Mp_{M\ H,N}) \in mHom\ R\ (M\ /_m\ H)\ (M\ /_m\ N)$
apply (*simp add*:*mHom-def*)
apply (*rule conjI*)
apply (*simp add*:*aHom-def*)
apply (*simp add*:*mQmpTr2-1*)
apply (*rule conjI*)
apply (*simp add*:*mQmp-def extensional-def qmodule-def*)
apply (*rule ballI*)+
apply (*frule-tac X1* = *a* **and** *Y1* = *b* **in** *mQmpTr3* [*THEN sym*, *of H N*],
assumption+)
apply (*simp add*:*qmodule-def*)

apply (*rule ballI*)+
apply (*simp add*:*qmodule-def*)
apply (*simp add*:*set-mr-cos-def*)
apply (*erule bexE*, *simp*)
apply (*subst mr-cos-sprodTr*, *assumption*+)
apply (*frule-tac a* = *a* **and** *m* = *aa* **in** *sc-mem*, *assumption*)
apply (*simp add*:*mQmpTr0*)
apply (*subst mr-cos-sprodTr*, *assumption*+)
apply *simp*
done

lemma (**in** *Module*) *Mp-surjec*:⟦*submodule R M H*; *submodule R M N*; $H \subseteq N$⟧ ⟹
$surjec_{(M\ /_m\ H),(M\ /_m\ N)}\ (Mp_{M\ H,N})$
apply (*simp add*:*surjec-def*)
apply (*frule mQmp-mHom* [*of H N*], *assumption*+)
apply (*rule conjI*)
apply (*simp add*:*mHom-def*)
apply (*rule surj-to-test*)
apply (*simp add*:*mHom-def aHom-def*)
apply (*rule ballI*)
apply (*thin-tac* $Mp_{M\ \ H,N} \in mHom\ R\ (M\ /_m\ H)\ (M\ /_m\ N)$)

apply (*simp add*:*qmodule-def*)
apply (*simp add*:*set-mr-cos-def*, *erule bexE*, *simp*)
apply (*frule-tac m* = *a* **in** *mQmpTr0* [*of H N*], *assumption*+)
apply *blast*
done

lemma (**in** *Module*) *kerQmp*:⟦*submodule R M H*; *submodule R M N*; *H* ⊆ *N*⟧
⟹ $ker_{(M\ /_m\ H),(M\ /_m\ N)}\ (Mp_{M\ H,N}) = carrier\ ((mdl\ M\ N)\ /_m\ H)$
apply (*simp add*:*ker-def*)
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*simp add*:*CollectI*, *erule conjE*)
apply (*simp add*:*qmodule-def*)
apply (*simp add*:*set-mr-cos-def* [*of mdl M N H*])
apply (*simp add*:*set-mr-cos-def*)
apply (*erule bexE*, *simp*)
apply (*simp add*:*mQmpTr0*)
apply (*frule-tac m* = *a* **in** *m-in-mr-coset*[*of N*], *assumption*+, *simp*)
apply (*frule-tac a* = *a* **in** *mQmpTr4*[*of H N*], *assumption*+,
simp add:*mr-coset-def*,
rotate-tac −1, *frule sym*,*thin-tac* $a \uplus_{mdl\ M\ N} H = a \uplus_M H$,
simp only:*mdl-carrier*, *blast*)

apply (*rule subsetI*)
apply (*simp add*:*qmodule-def*)
apply (*simp add*:*set-mr-cos-def* [*of mdl M N H*])
apply (*erule bexE*, *simp*)
apply (*simp add*:*mdl-carrier*)
apply (*frule-tac a* = *a* **in** *mQmpTr4*[*of H N*], *assumption*+,
simp add:*mr-coset-def*)
apply (*thin-tac* $a \uplus_{mdl\ M\ N} H = a \uplus_M H$)
apply (*unfold submodule-def*[*of R M N*], *frule conjunct1*, *fold submodule-def*,
frule-tac c = *a* **in** *subsetD*[*of N carrier M*], *assumption*+)
apply (*rule conjI*)
apply (*simp add*:*set-mr-cos-def*, *blast*)
apply (*simp add*:*mQmpTr0*)
apply (*simp add*:*mr-cos-h-stable* [*THEN sym*])
done

lemma (**in** *Module*) *misom2Tr*:⟦*submodule R M H*; *submodule R M N*; *H* ⊆ *N*⟧
⟹
$$(M\ /_m\ H)\ /_m\ (carrier\ ((mdl\ M\ N)\ /_m\ H)) \cong_R (M\ /_m\ N)$$
apply (*frule mQmp-mHom* [*of H N*], *assumption*+)
apply (*frule qmodule-module* [*of H*])
apply (*frule qmodule-module* [*of N*]) **thm** *Module.indmhom*
apply (*frule Module.indmhom* [*of M* /*m H R M* /*m N* $Mp_{M\ H,N}$], *assumption*+)
apply (*simp add*:*kerQmp*)
apply (*subgoal-tac bijec*$_{((M\ /_m\ H)\ /_m\ (carrier((mdl\ M\ N)\ /_m\ H))),(M\ /_m\ N)}$ (*indmhom*
R (*M* /*m H*) (*M* /*m N*) (*mQmp M H N*)))

apply (*simp add*:*misomorphic-def*) **apply** *blast*
apply (*simp add*:*bijec-def*)
apply (*rule conjI*)
apply (*simp add*:*kerQmp* [*THEN sym*])
apply (*rule Module.indmhom-injec* [*of M* $/_m$ *H R M* $/_m$ *N Mp*$_{M\ H,N}$], *assumption*+)
apply (*frule Mp-surjec* [*of H N*], *assumption*+)
apply (*simp add*:*kerQmp* [*THEN sym*])
apply (*rule Module.indmhom-surjec1*, *assumption*+)
done

lemma (**in** *Module*) *eq-class-of-Submodule*:⟦*submodule R M H*; *submodule R M N*;

$H \subseteq N$⟧ ⟹ *carrier* ((*mdl M N*) $/_m$ *H*) = *N* $_s/_M$ *H*
apply (*rule equalityI*)
apply (*rule subsetI*) **apply** (*simp add*:*qmodule-def*)
apply (*simp add*:*set-mr-cos-def*) **apply** (*erule bexE*, *simp*)
apply (*frule-tac a* = *a* **in** *mQmpTr4* [*of H N*], *assumption*+)
apply (*simp add*:*mdl-def*) **apply** (*simp add*:*mr-coset-def*)
apply (*simp add*:*sub-mr-set-cos-def*)
apply (*simp add*:*mdl-carrier*, *blast*)

apply (*rule subsetI*)
apply (*simp add*:*qmodule-def*)
apply (*simp add*:*set-mr-cos-def*)
apply (*simp add*:*sub-mr-set-cos-def*)
apply (*erule bexE*, *simp add*:*mdl-carrier*)
apply (*frule-tac a1* = *n* **in** *mQmpTr4*[*THEN sym*, *of H N*], *assumption*+)
apply (*simp add*:*mr-coset-def*)
apply *blast*
done

theorem (**in** *Module*) *misom2*:⟦*submodule R M H*; *submodule R M N*; *H* ⊆ *N*⟧
⟹
(*M* $/_m$ *H*) $/_m$ (*N* $_s/_M$ *H*) ≅$_R$ (*M* $/_m$ *N*)
apply (*frule misom2Tr* [*of H N*], *assumption*+)
apply (*simp add*:*eq-class-of-Submodule*)
done

primrec *natm* :: ('*a*, '*m*) *aGroup-scheme* => *nat* ⇒ '*a* => '*a*
where
natm-0: *natm M 0 x* = **0**$_M$
| *natm-Suc*: *natm M* (*Suc n*) *x* = (*natm M n x*) ±$_M$ *x*

definition
finitesum-base :: [('*a*, '*r*, '*m*) *Module-scheme*, '*b set*, '*b* ⇒ '*a set*]
⇒ '*a set* **where**
finitesum-base M I f = ⋃{*f i* | *i*. *i* ∈ *I*}

definition
finitesum :: [('*a*, '*r*, '*m*) *Module-scheme*, '*b set*, '*b* ⇒ '*a set*]
⇒ '*a set* **where**
finitesum M I f = {*x*. ∃ *n*. ∃ *g*. *g* ∈ {*j*. *j* ≤ (*n*::*nat*)} → *finitesum-base M I f*
∧ *x* = *nsum M g n*}

lemma (**in** *Module*) *finitesumbase-sub-carrier*:*f* ∈ *I* → {*X*. *submodule R M X*}
⟹
finitesum-base M I f ⊆ *carrier M*
apply (*simp add*:*finitesum-base-def*)
apply (*rule subsetI*)
apply (*simp add*:*CollectI*)
apply (*erule exE*, *erule conjE*, *erule exE*, *erule conjE*)
apply (*frule-tac x* = *i* **in** *funcset-mem*[*of f I* {*X*. *submodule R M X*}],
assumption+, *simp*)
apply (*thin-tac f* ∈ *I* → {*X*. *submodule R M X*}, *unfold submodule-def*,
frule conjunct1, *fold submodule-def*, *simp add*:*subsetD*)
done

lemma (**in** *Module*) *finitesum-sub-carrier*:*f* ∈ *I* → {*X*. *submodule R M X*} ⟹
finitesum M I f ⊆ *carrier M*
apply (*rule subsetI*, *simp add*:*finitesum-def*)
apply ((*erule exE*)+, *erule conjE*, *simp*)
apply (*frule finitesumbase-sub-carrier*)
apply (*rule nsum-mem*, *rule allI*, *rule impI*)
apply (*frule-tac x* = *j* **and** *f* = *g* **and** *A* = {*j*. *j* ≤ *n*} **and**
B = *finitesum-base M I f* **in** *funcset-mem*, *simp*)
apply (*simp add*:*subsetD*)
done

lemma (**in** *Module*) *finitesum-inc-zero*:⟦*f* ∈ *I* → {*X*. *submodule R M X*}; *I* ≠ {}⟧
⟹ **0** ∈ *finitesum M I f*
apply (*simp add*:*finitesum-def*)
apply (*frule nonempty-ex*)
apply (*subgoal-tac* ∀ *i*. *i*∈*I* ⟶ (∃ *n g*. *g* ∈ {*j*. *j* ≤ (*n*::*nat*)} →
finitesum-base M I f ∧ **0**$_M$ = Σ$_e$ *M g n*))
apply *blast*
apply (*rule allI*, *rule impI*)
apply (*subgoal-tac* (λ*x*∈{*j*. *j* ≤ (*0*::*nat*)}. **0**) ∈
{*j*. *j* ≤ (*0*::*nat*)} → *finitesum-base M I f* ∧
0$_M$ = Σ$_e$ *M* (λ*x*∈{*j*. *j* ≤ (*0*::*nat*)}. **0**) *0*)
apply *blast*
apply (*rule conjI*)
apply (*rule Pi-I*)
apply (*simp add*:*finitesum-base-def*, *thin-tac* ∃ *x*. *x* ∈ *I*)
apply (*frule-tac x* = *i* **in** *funcset-mem*[*of f I* {*X*. *submodule R M X*}],
assumption+)
apply (*frule-tac x* = *i* **in** *funcset-mem* [*of f I* {*X*. *submodule R M X*}],

assumption+, *simp*)
apply (*frule-tac* $H = f\ i$ **in** *submodule-inc-0*)
apply *blast*

apply *simp*
done

lemma (**in** *Module*) *finitesum-mOp-closed*:
$[\![f \in I \to \{X.\ submodule\ R\ M\ X\};\ I \neq \{\};\ a \in finitesum\ M\ I\ f]\!] \Longrightarrow$
$-_a\ a \in finitesum\ M\ I\ f$
apply (*simp add*:*finitesum-def*)
apply ((*erule exE*)+, *erule conjE*)
apply (*frule finitesumbase-sub-carrier* [*of f I*])
apply (*frule-tac* $f = g$ **and** $A = \{j.\ j \leq n\}$ **and** $B = finitesum\text{-}base\ M\ I\ f$
and *?B1.0* = *carrier M* **in** *extend-fun*, *assumption*+)
apply (*frule sym*, *thin-tac* $a = \Sigma_e\ M\ g\ n$)
apply (*cut-tac* $n = n$ **and** $f = g$ **in** *nsum-minus*,
rule allI, *simp add*:*Pi-def*, *simp*)

apply (*subgoal-tac* $(\lambda x{\in}\{j.\ j \leq n\}.\ -_a\ (g\ x)) \in \{j.\ j \leq n\} \to$
$finitesum\text{-}base\ M\ I\ f$)
apply *blast*
apply (*rule Pi-I*, *simp*)
apply (*frule-tac* $f = g$ **and** $A = \{j.\ j \leq n\}$ **and** $B = finitesum\text{-}base\ M\ I\ f$
and $x = x$ **in** *funcset-mem*, *simp*)
apply (*simp add*:*finitesum-base-def*)
apply (*erule exE*, *erule conjE*, *erule exE*, *erule conjE*)
apply (*frule-tac* $f = f$ **and** $A = I$ **and** $B = \{X.\ submodule\ R\ M\ X\}$ **and**
$x = i$ **in** *funcset-mem*, *assumption*+, *simp add*:*CollectI*)
apply (*thin-tac* $f \in I \to \{X.\ submodule\ R\ M\ X\}$)
apply (*simp add*:*submodule-def*, (*erule conjE*)+,
frule-tac $H = f\ i$ **and** $x = g\ x$ **in** *asubg-mOp-closed*, *assumption*+)
apply *blast*
done

lemma (**in** *Module*) *finitesum-pOp-closed*:
$[\![f \in I \to \{X.\ submodule\ R\ M\ X\};\ a \in finitesum\ M\ I\ f;\ \ b \in finitesum\ M\ I\ f]\!]$
$\Longrightarrow\ \ a \pm b \in finitesum\ M\ I\ f$
apply (*simp add*:*finitesum-def*)
apply ((*erule exE*)+, (*erule conjE*)+)
apply (*frule-tac* $f = g$ **and** $n = n$ **and** $A = finitesum\text{-}base\ M\ I\ f$ **and**
$g = ga$ **and** $m = na$ **and** $B = finitesum\text{-}base\ M\ I\ f$ **in** *jointfun-hom0*,
assumption+, *simp*)
apply (*cut-tac finitesumbase-sub-carrier*[*of f I*],
cut-tac $n1 = n$ **and** $f1 = g$ **and** $m1 = na$ **and** $g1 = ga$ **in**
nsum-add-nm[*THEN sym*], *rule allI*, *rule impI*,
frule-tac $x = j$ **and** $f = g$ **and** $A = \{j.\ j \leq n\}$ **and**
$B = finitesum\text{-}base\ M\ I\ f$ **in** *funcset-mem*, *simp*,
simp add:*subsetD*,

rule allI, *rule impI*,
frule-tac $x = j$ **and** $f = ga$ **and** $A = \{j.\ j \leq na\}$ **and**
$B =$ *finitesum-base* $M\ I\ f$ **in** *funcset-mem*, *simp*,
simp add:*subsetD*)
apply *blast*
apply *assumption*
done

lemma (**in** *Module*) *finitesum-sprodTr*:$[\![f \in I \to \{X.\ submodule\ R\ M\ X\};\ I \neq \{\};$
$r \in carrier\ R]\!] \Longrightarrow g \in \{j.\ j \leq (n::nat)\} \to (finitesum\text{-}base\ M\ I\ f)$
$\longrightarrow r \cdot_s (nsum\ M\ g\ n) = nsum\ M\ (\lambda x.\ r \cdot_s (g\ x))\ n$
apply (*induct-tac n*)
apply (*rule impI*)
apply *simp*
apply (*rule impI*)
apply (*frule func-pre*) **apply** *simp*
apply (*frule finitesumbase-sub-carrier* [*of f I*])
apply (*frule-tac* $f = g$ **and** $A = \{j.\ j \leq Suc\ n\}$ **in** *extend-fun* [*of - - finitesum-base M I f carrier M*], *assumption*+)
apply (*thin-tac* $g \in \{j.\ j \leq Suc\ n\} \to finitesum\text{-}base\ M\ I\ f$,
thin-tac $g \in \{j.\ j \leq n\} \to finitesum\text{-}base\ M\ I\ f$,
frule func-pre)
apply (*cut-tac* $n = n$ **in** *nsum-mem* [*of - g*])
apply (*rule allI*, *simp add*:*Pi-def*)
apply (*frule-tac* $x = Suc\ n$ **in** *funcset-mem* [*of g - carrier M*], *simp*)
apply (*subst sc-r-distr*, *assumption*+)
apply *simp*
done

lemma (**in** *Module*) *finitesum-sprod*:$[\![f \in I \to \{X.\ submodule\ R\ M\ X\};\ I \neq \{\};$
$r \in carrier\ R;\ g \in \{j.\ j \leq (n::nat)\} \to (finitesum\text{-}base\ M\ I\ f)\ ]\!] \Longrightarrow$
$r \cdot_s (nsum\ M\ g\ n) = nsum\ M\ (\lambda x.\ r \cdot_s (g\ x))\ n$
apply (*simp add*:*finitesum-sprodTr*)
done

lemma (**in** *Module*) *finitesum-subModule*:$[\![f \in I \to \{X.\ submodule\ R\ M\ X\};\ I \neq \{\}]\!]$
$\Longrightarrow submodule\ R\ M\ (finitesum\ M\ I\ f)$
apply (*simp add*:*submodule-def* [*of - - (finitesum M I f)*])
apply (*simp add*:*finitesum-sub-carrier*)
apply (*rule conjI*)
apply (*rule asubg-test*)
apply (*simp add*:*finitesum-sub-carrier*)
apply (*frule finitesum-inc-zero*, *assumption*, *blast*)

apply (*rule ballI*)+
apply (*rule finitesum-pOp-closed*, *assumption*+,
rule finitesum-mOp-closed, *assumption*+)

apply ((*rule allI*)+, *rule impI*, *erule conjE*)
apply (*simp add*:*finitesum-def*, (*erule exE*)+, *erule conjE*, *simp*)
apply (*simp add*:*finitesum-sprod*)
apply (*subgoal-tac* $(\lambda x.\ a \cdot_s g\ x) \in \{j.\ j \leq n\} \to$ *finitesum-base M I f*,
blast)
apply (*rule Pi-I*)
apply (*frule-tac* $x = x$ **and** $f = g$ **and** $A = \{j.\ j \leq n\}$ **in**
funcset-mem[*of - - finitesum-base M I f*], *assumption*+,
thin-tac $g \in \{j.\ j \leq n\} \to$ *finitesum-base M I f*,
simp add:*finitesum-base-def*, *erule exE*, *erule conjE*, *erule exE*,
erule conjE, *simp*)
apply (*frule-tac* $x = i$ **and** $f = f$ **and** $A = I$ **in**
funcset-mem[*of - - {X. submodule R M X}*], *assumption*+, *simp*,
frule-tac $H = f\ i$ **and** $a = a$ **and** $h = g\ x$ **in** *submodule-sc-closed*,
assumption+)
apply *blast*
done

lemma (**in** *Module*) *sSum-cont-H*:⟦*submodule R M H*; *submodule R M K*⟧ ⟹
$H \subseteq H \mathbin{+} K$
apply (*rule subsetI*)
apply (*unfold submodule-def*[*of R M H*], *frule conjunct1*, *fold submodule-def*,
unfold submodule-def[*of R M K*], *frule conjunct1*, *fold submodule-def*)
apply (*simp add*:*set-sum*)
apply (*frule submodule-inc-0* [*of K*])
apply (*cut-tac* $t = x$ **in** *ag-r-zero* [*THEN sym*],
rule submodule-subset1, *assumption*+)
apply *blast*
done

lemma (**in** *Module*) *sSum-commute*:⟦*submodule R M H*; *submodule R M K*⟧ ⟹
$H \mathbin{+} K = K \mathbin{+} H$
apply (*unfold submodule-def*[*of R M H*], *frule conjunct1*, *fold submodule-def*,
unfold submodule-def[*of R M K*], *frule conjunct1*, *fold submodule-def*)
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*simp add*:*set-sum*)
apply ((*erule bexE*)+, *simp*)
apply (*frule-tac* $c = h$ **in** *subsetD*[*of H carrier M*], *assumption*+,
frule-tac $c = k$ **in** *subsetD*[*of K carrier M*], *assumption*+)
apply (*subst ag-pOp-commute*, *assumption*+)
apply *blast*

apply (*rule subsetI*)
apply (*simp add*:*set-sum*)
apply ((*erule bexE*)+, *simp*)
apply (*frule-tac* $h = h$ **in** *submodule-subset1*[*of K*], *assumption*+,
frule-tac $h = k$ **in** *submodule-subset1*[*of H*], *assumption*+)
apply (*subst ag-pOp-commute*, *assumption*+)

apply *blast*
done

lemma (**in** *Module*) *Sum-of-SubmodulesTr*:⟦*submodule R M H*; *submodule R M K*⟧ ⟹
$g \in \{j.\ j \leq (n::nat)\} \to H \cup K \longrightarrow \Sigma_e\ M\ g\ n \in H \mp K$
apply (*induct-tac n*)
apply (*rule impI*)
apply *simp*
apply (*frule submodule-subset*[*of H*],
frule submodule-subset[*of K*])
apply (*simp add*:*set-sum*)
apply (*erule disjE*)
apply (*frule-tac c* = *g 0* **in** *subsetD*[*of H carrier M*], *assumption*+,
frule-tac t = *g 0* **in** *ag-r-zero*[*THEN sym*]) **apply** (
frule submodule-inc-0[*of K*], *blast*)
apply (*frule-tac c* = *g 0* **in** *subsetD*[*of K carrier M*], *assumption*+,
frule-tac t = *g 0* **in** *ag-l-zero*[*THEN sym*]) **apply** (
frule submodule-inc-0[*of H*], *blast*)
apply *simp*

apply (*rule impI*, *frule func-pre*, *simp*)
apply (*frule submodule-subset*[*of H*],
frule submodule-subset[*of K*])
apply (*simp add*:*set-sum*[*of H K*], (*erule bexE*)+, *simp*)
apply (*frule-tac x* = *Suc n* **and** *f* = *g* **and** *A* = $\{j.\ j \leq Suc\ n\}$ **and**
B = *H* ∪ *K* **in** *funcset-mem*, *simp*,
thin-tac $g \in \{j.\ j \leq n\} \to H \cup K$,
thin-tac $g \in \{j.\ j \leq Suc\ n\} \to H \cup K$,
thin-tac $\Sigma_e\ M\ g\ n = h \pm k$, *simp*)
apply (*erule disjE*)
apply (*frule-tac h* = *h* **in** *submodule-subset1*[*of H*], *assumption*,
frule-tac h = *g* (*Suc n*) **in** *submodule-subset1*[*of H*], *assumption*,
frule-tac h = *k* **in** *submodule-subset1*[*of K*], *assumption*)
apply (*subst ag-pOp-assoc*, *assumption*+)
apply (*frule-tac x* = *k* **and** *y* = *g* (*Suc n*) **in** *ag-pOp-commute*, *assumption*+,
simp, *subst ag-pOp-assoc*[*THEN sym*], *assumption*+)
apply (*frule-tac h* = *h* **and** *k* = *g* (*Suc n*) **in** *submodule-pOp-closed*[*of H*],
assumption+, *blast*)
apply (*frule-tac h* = *h* **in** *submodule-subset1*[*of H*], *assumption*,
frule-tac h = *g* (*Suc n*) **in** *submodule-subset1*[*of K*], *assumption*,
frule-tac h = *k* **in** *submodule-subset1*[*of K*], *assumption*)
apply (*subst ag-pOp-assoc*, *assumption*+,
frule-tac h = *k* **and** *k* = *g* (*Suc n*) **in** *submodule-pOp-closed*[*of K*],
assumption+, *blast*)
done

lemma (**in** *Module*) *sSum-two-Submodules*:⟦*submodule R M H*; *submodule R M K*⟧ ⟹

submodule R M (H ∔ K)
apply (*subst submodule-def*)
apply (*frule submodule-asubg*[*of H*],
frule submodule-asubg[*of K*])
apply (*frule plus-subgs*[*of H K*], *assumption*, *simp add*:*asubg-subset*)

apply (*rule allI*)+
apply (*rule impI*, *erule conjE*, *frule asubg-subset*[*of H*],
frule asubg-subset[*of K*])
apply (*simp add*:*set-sum*[*of H K*], (*erule bexE*)+, *simp*)
apply (*frule-tac H = H* **and** *a = a* **and** *h = h* **in** *submodule-sc-closed*,
assumption+,
frule-tac H = K **and** *a = a* **and** *h = k* **in** *submodule-sc-closed*,
assumption+)
apply (*frule-tac c = h* **in** *subsetD*[*of H carrier M*], *assumption*+,
frule-tac c = k **in** *subsetD*[*of K carrier M*], *assumption*+,
simp add:*sc-r-distr*)
apply *blast*
done

definition
iotam :: [('*a*, '*r*, '*m*) *Module-scheme*, '*a set*, '*a set*] ⇒ ('*a* ⇒ '*a*)
((*3ιm*_ _,_) [*82*, *82*, *83*]*82*) **where**
$\iota m_{M\ H,K} = (\lambda x{\in}H.\ (x \pm_M \mathbf{0}_M))$

lemma (**in** *Module*) *iotam-mHom*:⟦*submodule R M H*; *submodule R M K*⟧
⟹ $\iota m_{M\ H,K} \in$ *mHom R* (*mdl M H*) (*mdl M* (*H* ∔ *K*))
apply (*simp add*:*mHom-def*)
apply (*rule conjI*)
apply (*simp add*:*aHom-def*)
apply (*simp add*:*mdl-def*)
apply (*rule conjI*)
apply (*rule Pi-I*)
apply (*simp add*:*iotam-def*)
apply (*frule submodule-subset*[*of H*], *frule submodule-subset*[*of K*],
simp add:*set-sum*)
apply (*frule submodule-inc-0* [*of K*])
apply *blast*
apply (*rule conjI*)
apply (*simp add*:*iotam-def extensional-def mdl-def*)
apply (*rule ballI*)+
apply (*simp add*:*mdl-def iotam-def*)
apply (*frule-tac h = a* **and** *k = b* **in** *submodule-pOp-closed* [*of H*],
assumption+, *simp*)
apply (*frule submodule-subset*[*of H*],
frule-tac c = a **in** *subsetD*[*of H carrier M*], *assumption*) **apply** (
simp add:*ag-r-zero*)
apply (*frule-tac c = b* **in** *subsetD*[*of H carrier M*], *assumption*,
subst ag-pOp-assoc, *assumption*+,

simp add:*ag-inc-zero*, *simp*)

apply (*rule ballI*)+
apply (*simp add*:*iotam-def mdl-def*)
apply (*simp add*:*submodule-sc-closed*)
apply (*frule submodule-inc-0*[*of K*])
apply (*frule submodule-asubg*[*of H*], *frule submodule-asubg*[*of K*],
simp add:*mem-sum-subgs*)

apply (*frule-tac a* = *a* **and** *h* = *m* **in** *submodule-sc-closed*, *assumption*+,
frule submodule-subset[*of H*],
frule-tac c = *m* **in** *subsetD*[*of H carrier M*], *assumption*+,
frule-tac c = *a* ·$_s$ *m* **in** *subsetD*[*of H carrier M*], *assumption*+)
apply (*simp add*:*ag-r-zero*)
done

lemma (**in** *Module*) *mhomom3Tr*:⟦*submodule R M H*; *submodule R M K*⟧ ⟹
submodule R (*mdl M* (*H* ∓ *K*)) *K*
apply (*subst submodule-def*)
apply (*rule conjI*)
apply (*simp add*:*mdl-def*)
apply (*subst sSum-commute*, *assumption*+)
apply (*simp add*:*sSum-cont-H*)
apply (*rule conjI*)
apply (*rule aGroup.asubg-test*)
apply (*frule sSum-two-Submodules* [*of H K*], *assumption*+)
apply (*frule mdl-is-module* [*of* (*H* ∓ *K*)])
apply (*rule Module.module-is-ag*, *assumption*+)
apply (*simp add*:*mdl-def*)
apply (*subst sSum-commute*, *assumption*+)
apply (*simp add*:*sSum-cont-H*)
apply (*frule submodule-inc-0* [*of K*])
apply (*simp add*:*nonempty*)
apply (*rule ballI*)+
apply (*simp add*:*mdl-def*)
apply (*rule submodule-pOp-closed*, *assumption*+)
apply (*rule submodule-mOp-closed*, *assumption*+)
apply ((*rule allI*)+, *rule impI*)
apply (*simp add*:*mdl-def*, *erule conjE*)
apply (*frule sSum-cont-H*[*of K H*], *assumption*,
simp add:*sSum-commute*[*of K H*])
apply (*simp add*:*subsetD submodule-sc-closed*)
done

lemma (**in** *Module*) *mhomom3Tr0*:⟦*submodule R M H*; *submodule R M K*⟧
⟹ *compos* (*mdl M H*) (*mpj* (*mdl M* (*H* ∓ *K*)) *K*) (*ιm*$_{M\ H,K}$)
∈ *mHom R* (*mdl M H*) (*mdl M* (*H* ∓ *K*) /$_m$ *K*)
apply (*frule mdl-is-module* [*of H*])
apply (*frule mhomom3Tr*[*of H K*], *assumption*+)

apply (*frule sSum-two-Submodules* [*of H K*], *assumption*+)
apply (*frule mdl-is-module* [*of H* $\mp$ *K*])
apply (*frule iotam-mHom* [*of H K*], *assumption*+) **thm** *Module.mpj-mHom*
apply (*frule Module.mpj-mHom* [*of mdl M* (*H* $\mp$ *K*) *R K*], *assumption*+)
apply (*rule Module.mHom-compos*[*of mdl M* (*H* $\mp$ *K*) *R mdl M H*], *assumption*+)
apply (*simp add*:*Module.qmodule-module*, *assumption*)
apply (*simp add*:*mpj-mHom*)
done

lemma (**in** *Module*) *mhomom3Tr1*:⟦*submodule R M H*; *submodule R M K*⟧ ⟹
$surjec_{(mdl\ M\ H),((mdl\ M\ (H \mp K))/_m\ K)}$
(*compos* (*mdl M H*) (*mpj* (*mdl M* (*H* $\mp$ *K*)) *K*) ($\iota m_{M\ H,K}$))
apply (*simp add*:*surjec-def*)
apply (*frule mhomom3Tr0* [*of H K*], *assumption*+)
apply (*rule conjI*)
apply (*simp add*:*mHom-def*)
apply (*rule surj-to-test*)
apply (*simp add*:*mHom-def aHom-def*)
apply (*rule ballI*)
apply (*simp add*:*compos-def compose-def*)
apply (*thin-tac* (λ*x*∈*carrier* (*mdl M H*). *mpj* (*mdl M* (*H* $\mp$ *K*)) *K* (($\iota m_{M\ H,K}$) *x*))
∈ *mHom R* (*mdl M H*) (*mdl M* (*H* $\mp$ *K*) $/_m$ *K*))
apply (*simp add*:*qmodule-def*)
apply (*simp add*:*set-mr-cos-def*)
apply (*erule bexE*, *simp*)
apply (*simp add*:*mdl-carrier*)
apply (*simp add*:*iotam-def*)
apply (*simp add*:*mpj-def*)
apply (*frule sSum-two-Submodules*[*of H K*], *assumption*+)
apply (*simp add*:*mdl-carrier*)
apply (*subgoal-tac* ∀ *aa*∈*H*. *aa* ± **0** ∈ *H* $\mp$ *K*, *simp*)
apply (*frule submodule-subset*[*of H*], *frule submodule-subset*[*of K*],
thin-tac ∀ *aa*∈*H*. *aa* ± **0** ∈ *H* $\mp$ *K*,
simp add:*set-sum*, (*erule bexE*)+)
apply (*simp add*:*set-sum*[*THEN sym*])
apply (*frule mdl-is-module*[*of H* $\mp$ *K*],
frule mhomom3Tr[*of H K*], *assumption*+)
apply (*frule-tac m* = *h* **and** *h* = *k* **in** *Module.mr-cos-h-stable1*[*of mdl M* (*H* $\mp$ *K*)
R K], *assumption*+)
apply (*simp add*:*mdl-carrier*)
apply (*frule sSum-cont-H*[*of H K*], *assumption*+, *simp add*:*subsetD*, *assumption*)
apply (*simp add*:*mdl-def*, *fold mdl-def*)
apply (*subgoal-tac* ∀ *a*∈*H*. *a* ± **0** = *a*, *simp*, *blast*)
apply (*rule ballI*)
apply (*frule-tac c* = *aa* **in** *subsetD*[*of H carrier M*], *assumption*+,
simp add:*ag-r-zero*)

apply (*rule ballI*)
apply (*frule submodule-inc-0*[*of K*])
apply (*rule mem-sum-subgs*,
simp add:*submodule-def*, *simp add*:*submodule-def*, *assumption*+)
done

lemma (**in** *Module*) *mhomom3Tr2*:⟦*submodule R M H*; *submodule R M K*⟧ ⟹
$ker_{(mdl\ M\ H),((mdl\ M\ (H \mp K))\ /_m\ K)}$
(*compos* (*mdl M H*) (*mpj* (*mdl M* (*H* ∓ *K*)) *K*) ($\iota m_{M\ H,K}$)) = *H* ∩ *K*
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*simp add*:*ker-def*, *erule conjE*)
apply (*simp add*:*qmodule-def*)
apply (*simp add*:*mdl-carrier*)
apply (*simp add*:*compos-def compose-def mdl-def iotam-def*)
apply (*fold mdl-def*)
apply (*simp add*:*iotam-def mpj-def*)
apply (*frule sSum-two-Submodules*[*of H K*], *assumption*+, *simp add*:*mdl-carrier*)
apply (*frule submodule-asubg*[*of H*], *frule submodule-asubg*[*of K*])
apply (*frule-tac h* = *x* **and** *k* = **0** **in** *mem-sum-subgs*[*of H K*], *assumption*+)
apply (*simp add*:*submodule-inc-0*)
apply *simp* **apply** (*frule mhomom3Tr*[*of H K*], *assumption*+)

apply (*frule sSum-two-Submodules*[*of H K*], *assumption*,
frule mdl-is-module [*of* *H* ∓ *K*])
apply (*frule-tac m* = *x* ± **0** **in** *Module.m-in-mr-coset*[*of mdl M* (*H* ∓ *K*) *R K*],
assumption+)
apply (*simp add*:*mdl-carrier*, *simp*)
apply (*frule submodule-subset*[*of H*],
frule-tac c = *x* **in** *subsetD*[*of H carrier M*], *assumption*+)
apply (*simp add*:*ag-r-zero*)

apply (*rule subsetI*)
apply (*simp add*:*ker-def*)
apply (*simp add*:*mdl-carrier*)
apply (*simp add*:*qmodule-def*)
apply (*simp add*:*compos-def compose-def*)
apply (*simp add*:*mdl-carrier*)
apply (*simp add*:*iotam-def mpj-def*)
apply (*frule sSum-two-Submodules*[*of H K*], *assumption*+)
apply (*simp add*:*mdl-carrier*)
apply (*erule conjE*,
frule submodule-inc-0[*of K*],
frule submodule-asubg[*of H*], *frule submodule-asubg*[*of K*],
simp add:*mem-sum-subgs*)
apply (*frule submodule-subset*[*of K*]) **apply** (
frule-tac c = *x* **in** *subsetD*[*of K carrier M*], *assumption*+)
apply (*simp add*:*ag-r-zero*,
frule mdl-is-module [*of* *H* ∓ *K*],

frule mhomom3Tr[*of H K*], *assumption*+)
apply (*frule-tac h1* = *x* **in** *Module.mr-cos-h-stable*[*THEN sym*, *of mdl M* (*H* ∓ *K*)
R K], *assumption*+)
done

lemma (**in** *Module*) *mhomom-3*:⟦*submodule R M H*; *submodule R M K*⟧ ⟹
(*mdl M H*) /$_m$ (*H* ∩ *K*) ≅$_R$ (*mdl M* (*H* ∓ *K*)) /$_m$ *K*
apply (*frule sSum-two-Submodules* [*of H K*], *assumption*+)
apply (*frule mdl-is-module* [*of H*])
apply (*frule mdl-is-module* [*of K*])
apply (*frule mdl-is-module* [*of H* ∓ *K*])
apply (*frule mhomom3Tr* [*of H K*], *assumption*+)
apply (*frule Module.qmodule-module* [*of mdl M* (*H* ∓ *K*) *R K*], *assumption*+)
apply (*simp add*:*misomorphic-def*)
apply (*frule mhomom3Tr0*[*of H K*], *assumption*+)
apply (*frule mhomom3Tr1*[*of H K*], *assumption*+)
apply (*frule Module.indmhom* [*of mdl M H R mdl M* (*H* ∓ *K*) /$_m$ *K compos* (*mdl M H*) (*mpj* (*mdl M* (*H* ∓ *K*)) *K*) (*ι*$m_{M\ H,K}$)], *assumption*+)
apply (*frule Module.indmhom-injec*[*of mdl M H R mdl M* (*H* ∓ *K*) /$_m$ *K*
compos (*mdl M H*) (*mpj* (*mdl M* (*H* ∓ *K*)) *K*) (*ι*$m_{M\ H,K}$)], *assumption*+)
apply (*frule Module.indmhom-surjec1*[*of mdl M H R mdl M* (*H* ∓ *K*) /$_m$ *K compos* (*mdl M H*) (*mpj* (*mdl M* (*H* ∓ *K*)) *K*) (*ι*$m_{M\ H,K}$)], *assumption*+)
apply (*simp add*:*bijec-def*)
apply (*simp add*:*mhomom3Tr2*[*of H K*])
apply *blast*
done

definition
l-comb :: [('*r*, '*m*) *Ring-scheme*, ('*a*, '*r*, '*m1*) *Module-scheme*, *nat*] ⇒
(*nat* ⇒ '*r*) ⇒ (*nat* ⇒ '*a*) ⇒ '*a* **where**
l-comb R M n s m = *nsum M* (λ*j*. (*s j*) ·$_{s M}$ (*m j*)) *n*

definition
linear-span :: [('*r*, '*m*) *Ring-scheme*, ('*a*, '*r*, '*m1*) *Module-scheme*, '*r set*,
'*a set*] ⇒ '*a set* **where**
linear-span R M A H = (*if H* = {} *then* {**0**$_M$} *else*
{*x*. ∃ *n*. ∃ *f* ∈ {*j*. *j* ≤ (*n*::*nat*)} → *H*.
∃ *s*∈{*j*. *j* ≤ (*n*::*nat*)} → *A*. *x* = *l-comb R M n s f*})

definition
coefficient :: [('*r*, '*m*) *Ring-scheme*, ('*a*, '*r*, '*m1*) *Module-scheme*,
nat, *nat* ⇒ '*r*, *nat* ⇒ '*a*] ⇒ *nat* ⇒ '*r* **where**
coefficient R M n s m j = *s j*

definition
body :: [('*r*, '*m*) *Ring-scheme*, ('*a*, '*r*, '*m1*) *Module-scheme*, *nat*, *nat* ⇒ '*r*,
nat ⇒ '*a*] ⇒ *nat* ⇒ '*a* **where**
body R M n s m j = *m j*

lemma (**in** *Module*) *l-comb-mem-linear-span*:⟦*ideal R A*; $H \subseteq$ *carrier M*;
$s \in \{j.\ j \leq (n::nat)\} \to A$; $f \in \{j.\ j \leq n\} \to H$⟧ ⟹
l-comb R M n s f ∈ *linear-span R M A H*
apply (*frule-tac x* = *0* **in** *funcset-mem*[*of f* {*j*. *j* ≤ *n*} *H*], *simp*)
apply (*frule nonempty*[*of f 0 H*])
apply (*simp add*:*linear-span-def*)
apply *blast*
done

lemma (**in** *Module*) *linear-comb-eqTr*:*H* ⊆ *carrier M* ⟹
$s \in \{j.\ j \leq (n::nat)\} \to$ *carrier R* ∧
$f \in \{j.\ j \leq n\} \to H$ ∧
$g \in \{j.\ j \leq n\} \to H$ ∧
$(\forall j \in \{j.\ j \leq n\}.\ f\ j = g\ j) \longrightarrow$
l-comb R M n s f = *l-comb R M n s g*
apply (*induct-tac n*)
apply (*rule impI*) **apply** (*erule conjE*)+ **apply** (*simp add*:*l-comb-def*)

apply (*rule impI*) **apply** (*erule conjE*)+
apply (*frule-tac f* = *s* **in** *func-pre*)
apply (*frule-tac f* = *f* **in** *func-pre*)
apply (*frule-tac f* = *g* **in** *func-pre*)
apply (*cut-tac n* = *n* **in** *Nsetn-sub-mem1*, *simp*)
apply (*thin-tac* $s \in \{j.\ j \leq n\} \to$ *carrier R*,
thin-tac $f \in \{j.\ j \leq n\} \to H$,
thin-tac $g \in \{j.\ j \leq n\} \to H$)
apply (*simp add*:*l-comb-def*)
done

lemma (**in** *Module*) *linear-comb-eq*:⟦*H* ⊆ *carrier M*;
$s \in \{j.\ j \leq (n::nat)\} \to$ *carrier R*; $f \in \{j.\ j \leq n\} \to H$;
$g \in \{j.\ j \leq n\} \to H$; $\forall j \in \{j.\ j \leq n\}.\ f\ j = g\ j$⟧ ⟹
l-comb R M n s f = *l-comb R M n s g*
apply (*simp add*:*linear-comb-eqTr*)
done

lemma (**in** *Module*) *l-comb-Suc*:⟦*H* ⊆ *carrier M*; *ideal R A*;
$s \in \{j.\ j \leq (Suc\ n)\} \to$ *carrier R*; $f \in \{j.\ j \leq (Suc\ n)\} \to H$⟧ ⟹
l-comb R M (*Suc n*) *s f* = *l-comb R M n s f* ± *s* (*Suc n*) ·$_s$ *f* (*Suc n*)
apply (*simp add*:*l-comb-def*)
done

lemma (**in** *Module*) *l-comb-jointfun-jj*:⟦*H* ⊆ *carrier M*; *ideal R A*;
$s \in \{j.\ j \leq (n::nat)\} \to A$; $f \in \{j.\ j \leq (n::nat)\} \to H$;
$t \in \{j.\ j \leq (m::nat)\} \to A$; $g \in \{j.\ j \leq (m::nat)\} \to H$⟧ ⟹
nsum M (λ*j*. (*jointfun n s m t*) *j* ·$_s$ (*jointfun n f m g*) *j*) *n* =
nsum M (λ*j*. *s j* ·$_s$ *f j*) *n*
apply (*cut-tac sc-Ring*)

apply (*rule nsum-eq*)
apply (*rule allI*, *rule impI*, *simp add*:*jointfun-def*,
rule sc-mem,
simp add:*Pi-def Ring.ideal-subset*,
simp add:*Pi-def subsetD*)
apply (*rule allI*, *rule impI*,
rule sc-mem,
simp add:*Pi-def Ring.ideal-subset*,
simp add:*Pi-def subsetD*)
apply (*rule allI*, *simp add*:*jointfun-def*)
done

lemma (**in** *Module*) *l-comb-jointfun-jj1*:⟦*H* ⊆ *carrier M*; *ideal R A*;
s ∈ {*j*. *j* ≤ (*n*::*nat*)} → *A*; *f* ∈ {*j*. *j* ≤ (*n*::*nat*)} → *H*;
t ∈ {*j*. *j* ≤ (*m*::*nat*)} → *A*; *g* ∈ {*j*. *j* ≤ (*m*::*nat*)} → *H*⟧ ⟹
l-comb R M n (*jointfun n s m t*) (*jointfun n f m g*) =
l-comb R M n s f
by (*simp add*:*l-comb-def*, *simp add*:*l-comb-jointfun-jj*)

lemma (**in** *Module*) *l-comb-jointfun-jf*:⟦*H* ⊆ *carrier M*; *ideal R A*;
s ∈ {*j*. *j* ≤ (*n*::*nat*)} → *A*; *f* ∈ {*j*. *j* ≤ *Suc* (*n* + *m*)} → *H*;
t ∈ {*j*. *j* ≤ (*m*::*nat*)} → *A*⟧ ⟹
nsum M (λ*j*. (*jointfun n s m t*) *j* ·$_s$ *f j*) *n* =
nsum M (λ*j*. *s j* ·$_s$ *f j*) *n*
apply (*cut-tac sc-Ring*)
apply (*rule nsum-eq*)
apply (*rule allI*, *rule impI*, *simp add*:*jointfun-def*,
rule sc-mem,
simp add:*Pi-def Ring.ideal-subset*,
simp add:*Pi-def subsetD*)
apply (*rule allI*, *rule impI*,
rule sc-mem,
simp add:*Pi-def Ring.ideal-subset*,
simp add:*Pi-def subsetD*)
apply (*rule allI*, *simp add*:*jointfun-def*)
done

lemma (**in** *Module*) *l-comb-jointfun-jf1*:⟦*H* ⊆ *carrier M*; *ideal R A*;
s ∈ {*j*. *j* ≤ (*n*::*nat*)} → *A*; *f* ∈ {*j*. *j* ≤ *Suc* (*n* + *m*)} → *H*;
t ∈ {*j*. *j* ≤ (*m*::*nat*)} → *A*⟧ ⟹
l-comb R M n (*jointfun n s m t*) *f* = *l-comb R M n s f*
by (*simp add*:*l-comb-def l-comb-jointfun-jf*)

lemma (**in** *Module*) *l-comb-jointfun-fj*:⟦*H* ⊆ *carrier M*; *ideal R A*;
s ∈ {*j*. *j* ≤ *Suc* (*n* + *m*)} → *A*; *f* ∈ {*j*. *j* ≤ (*n*::*nat*)} → *H*;
g ∈ {*j*. *j* ≤ (*m*::*nat*)} → *H*⟧ ⟹
nsum M (λ*j*. *s j* ·$_s$ (*jointfun n f m g*) *j*) *n* =
nsum M (λ*j*. *s j* ·$_s$ *f j*) *n*
apply (*cut-tac sc-Ring*)

apply (*rule nsum-eq*)
apply (*rule allI*, *rule impI*, *simp add*:*jointfun-def*,
rule sc-mem,
simp add:*Pi-def Ring.ideal-subset*,
simp add:*Pi-def subsetD*)
apply (*rule allI*, *rule impI*,
rule sc-mem,
simp add:*Pi-def Ring.ideal-subset*,
simp add:*Pi-def subsetD*)
apply (*rule allI*, *simp add*:*jointfun-def*)
done

lemma (**in** *Module*) *l-comb-jointfun-fj1*:⟦$H \subseteq$ *carrier* M; *ideal* R A;
$s \in \{j.\ j \leq Suc\ (n + m)\} \to A$; $f \in \{j.\ j \leq (n::nat)\} \to H$;
$g \in \{j.\ j \leq (m::nat)\} \to H$⟧ $\Longrightarrow$
l-comb R M n s (*jointfun* n f m g) = *l-comb* R M n s f
by (*simp add*:*l-comb-def l-comb-jointfun-fj*)

lemma (**in** *Module*) *linear-comb0-1Tr*:$H \subseteq$ *carrier* $M \Longrightarrow$
$s \in \{j.\ j \leq (n::nat)\} \to \{\mathbf{0}_R\} \wedge$
$m \in \{j.\ j \leq n\} \to H \longrightarrow$ *l-comb* R M n s $m = \mathbf{0}_M$
apply (*induct-tac* n)
apply (*rule impI*) **apply** (*erule conjE*)
apply (*simp add*:*l-comb-def subsetD sc-0-m*)

apply (*rule impI*) **apply** (*erule conjE*)
apply (*frule func-pre* [*of* - - $\{\mathbf{0}_R\}$])
apply (*frule func-pre* [*of* - - H])
apply *simp*
apply (*thin-tac* $s \in \{j.\ j \leq n\} \to \{\mathbf{0}_R\}$,
thin-tac $m \in \{j.\ j \leq n\} \to H$)
apply (*simp add*:*l-comb-def*)
apply (*frule-tac* $x = Suc\ n$ **and** $f = s$ **and** $A = \{j.\ j \leq Suc\ n\}$ **in**
funcset-mem[*of* - - $\{\mathbf{0}_R\}$], *simp*, *simp*,
frule-tac $x = Suc\ n$ **and** $f = m$ **and** $A = \{j.\ j \leq Suc\ n\}$ **in**
funcset-mem[*of* - - H], *simp*,
frule-tac $c = m$ (*Suc* n) **in** *subsetD*[*of* H *carrier* M], *assumption*+,
simp add:*sc-0-m*)
apply (*cut-tac ag-inc-zero*)
apply (*simp add*:*ag-l-zero*)
done

lemma (**in** *Module*) *linear-comb0-1*:⟦$H \subseteq$ *carrier* M;
$s \in \{j.\ j \leq (n::nat)\} \to \{\mathbf{0}_R\}$; $m \in \{j.\ j \leq n\} \to H$ ⟧ $\Longrightarrow$
l-comb R M n s $m = \mathbf{0}_M$
apply (*simp add*:*linear-comb0-1Tr*)
done

lemma (**in** *Module*) *linear-comb0-2Tr*:*ideal* R $A \Longrightarrow s \in \{j.\ j \leq (n::nat)\} \to A$

$\wedge$ $m \in \{j.\ j \leq n\} \to \{\mathbf{0}_M\} \longrightarrow$ *l-comb R M n s m* $= \mathbf{0}_M$
apply (*induct-tac n*)
apply (*rule impI*) **apply** (*erule conjE*)
apply (*simp add*:*l-comb-def sc-a-0 sc-Ring Ring.ideal-subset*)

apply (*rule impI*)
apply (*erule conjE*)+
apply (*frule func-pre* [*of s*],
frule func-pre [*of m*], *simp*)
apply (*thin-tac* $s \in \{j.\ j \leq n\} \to A$,
thin-tac $m \in \{j.\ j \leq n\} \to \{\mathbf{0}\}$)
apply (*simp add*:*l-comb-def*)
apply (*frule-tac* $A = \{j.\ j \leq Suc\ n\}$ **and** $x = Suc\ n$ **in**
funcset-mem [*of m* - $\{\mathbf{0}\}$], *simp*+,
frule-tac $A = \{j.\ j \leq Suc\ n\}$ **and** $x = Suc\ n$ **in**
funcset-mem[*of s* - *A*], *simp*+,
cut-tac sc-Ring,
frule-tac $h = s\ (Suc\ n)$ **in** *Ring.ideal-subset*[*of R A*], *assumption*+)
apply (*cut-tac ag-inc-zero*, *simp add*:*sc-a-0*)
apply (*simp add*:*ag-l-zero*)
done

lemma (**in** *Module*) *linear-comb0-2*:$[\![$*ideal R A*; $s \in \{j.\ j \leq (n::nat)\} \to A$;
$m \in \{j.\ j \leq n\} \to \{\mathbf{0}_M\}\ ]\!] \Longrightarrow$ *l-comb R M n s m* $= \mathbf{0}_M$
apply (*simp add*:*linear-comb0-2Tr*)
done

lemma (**in** *Module*) *liear-comb-memTr*:$[\![$*ideal R A*; $H \subseteq$ *carrier M*$]\!] \Longrightarrow$
$\forall s.\ \forall m.\ s \in \{j.\ j \leq (n::nat)\} \to A \wedge$
$m \in \{j.\ j \leq n\} \to H \longrightarrow$ *l-comb R M n s m* $\in$ *carrier M*
apply (*induct-tac n*)
apply (*rule allI*)+ **apply** (*rule impI*) **apply** (*erule conjE*)
apply (*simp add*: *l-comb-def sc-mem Ring.ideal-subset*[*of R A*] *subsetD sc-Ring*)

apply (*rule allI*)+ **apply** (*rule impI*) **apply** (*erule conjE*)
apply (*frule func-pre* [*of - - A*],
frule func-pre [*of - - H*],
drule-tac $x = s$ **in** *spec*,
drule-tac $x = m$ **in** *spec*)

apply (*simp add*:*l-comb-def*)
apply (*rule ag-pOp-closed*, *assumption*+)
apply (*rule sc-mem*)
apply (*cut-tac sc-Ring*,
simp add:*Pi-def Ring.ideal-subset subsetD*)
apply (*simp add*:*Pi-def subsetD*)
done

lemma (**in** *Module*) *l-comb-mem*:$[\![$*ideal R A*; $H \subseteq$ *carrier M*;

$s \in \{j.\ j \leq (n::nat)\} \to A;\ m \in \{j.\ j \leq n\} \to H \rrbracket \Longrightarrow$
$l\text{-}comb\ R\ M\ n\ s\ m \in carrier\ M$
apply (*simp add:liear-comb-memTr*)
done

lemma (**in** *Module*) *l-comb-transpos*: $\llbracket ideal\ R\ A;\ H \subseteq carrier\ M;$
$s \in \{l.\ l \leq Suc\ n\} \to A;\ f \in \{l.\ l \leq Suc\ n\} \to H;$
$j < Suc\ n\ \rrbracket \Longrightarrow$
$\Sigma_e\ M\ (cmp\ (\lambda k.\ s\ k \cdot_s f\ k)\ (transpos\ j\ (Suc\ n)))\ (Suc\ n) =$
$\Sigma_e\ M\ (\lambda k.\ (cmp\ s\ (transpos\ j\ (Suc\ n)))\ k \cdot_s$
$(cmp\ f\ (transpos\ j\ (Suc\ n)))\ k)\ (Suc\ n)$
apply (*cut-tac sc-Ring*)
apply (*rule nsum-eq*)
apply (*rule allI*, *rule impI*, *simp add:cmp-def*)
apply (*cut-tac l = ja* **in** *transpos-mem*[*of j Suc n Suc n*],
simp add:less-imp-le, *simp*, *simp*, *assumption*)
apply (*rule sc-mem*,
simp add:Pi-def Ring.ideal-subset,
simp add:Pi-def subsetD)
apply (*rule allI*, *rule impI*, *simp add:cmp-def*)
apply (*frule less-imp-le*[*of j Suc n*],
frule-tac l = ja **in** *transpos-mem*[*of j Suc n Suc n*], *simp*,
simp, *assumption+*)
apply (*rule sc-mem*,
simp add:Pi-def Ring.ideal-subset,
simp add:Pi-def subsetD)
apply (*rule allI*, *rule impI*,
simp add:cmp-def)
done

lemma (**in** *Module*) *l-comb-transpos1*: $\llbracket ideal\ R\ A;\ H \subseteq carrier\ M;$
$s \in \{l.\ l \leq Suc\ n\} \to A;\ f \in \{l.\ l \leq Suc\ n\} \to H;\ j < Suc\ n\ \rrbracket \Longrightarrow$
$l\text{-}comb\ R\ M\ (Suc\ n)\ s\ f =$
$l\text{-}comb\ R\ M\ (Suc\ n)\ (cmp\ s\ (transpos\ j\ (Suc\ n)))\ (cmp\ f\ (transpos\ j\ (Suc\ n)))$
apply (*cut-tac sc-Ring*)
apply (*frule l-comb-transpos*[*THEN sym*, *of A H s n f j*], *assumption+*)
apply (*simp del:nsum-suc add:l-comb-def*,
thin-tac $\Sigma_e\ M\ (\lambda k.\ (cmp\ s\ (transpos\ j\ (Suc\ n)))\ k \cdot_s$
$(cmp\ f\ (transpos\ j\ (Suc\ n)))\ k)\ (Suc\ n) =$
$\Sigma_e\ M\ (cmp\ (\lambda k.\ s\ k \cdot_s f\ k)\ (transpos\ j\ (Suc\ n)))\ (Suc\ n)$)
apply (*cut-tac addition2*[*of* $\lambda j.\ s\ j \cdot_s f\ j\ n\ transpos\ j\ (Suc\ n)$],
simp)
apply (*rule Pi-I*, *rule sc-mem*,
simp add:Pi-def Ring.ideal-subset,
simp add:Pi-def subsetD)
apply (*rule-tac i = j* **and** *n = Suc n* **and** *j = Suc n* **in** *transpos-hom*,
simp add:less-imp-le, *simp*, *simp*)
apply (*rule-tac i = j* **and** *n = Suc n* **and** *j = Suc n* **in** *transpos-inj*,
simp add:less-imp-le, *simp*, *simp*)

done

lemma (**in** *Module*) *sc-linear-span*:⟦*ideal R A*; *H* ⊆ *carrier M*; *a* ∈ *A*;
h ∈ *H*⟧ ⟹ *a* ·$_s$ *h* ∈ *linear-span R M A H*
apply (*simp add*:*linear-span-def*)
apply (*simp add*:*nonempty*)
apply (*simp add*:*l-comb-def*)
apply (*subgoal-tac* (λ*k*∈{*j*. *j* ≤ (*0*::*nat*)}. *a*) ∈{*j*. *j* ≤ *0*} → *A*)
apply (*subgoal-tac* (λ*k*∈{*j*. *j* ≤ *0*}. *h*) ∈ {*j*. *j* ≤ (*0*::*nat*)} → *H*)
apply (*subgoal-tac a* ·$_s$ *h* =
Σ$_e$ *M* (λ*j*. (λ*k*∈{*j*. *j* ≤ (*0*::*nat*)}. *a*) *j* ·$_s$ (λ*k*∈{*j*. *j* ≤ (*0*::*nat*)}. *h*) *j*) *0*)
apply *blast*
apply *simp*+
done

lemma (**in** *Module*) *l-span-cont-H*:*H* ⊆ *carrier M* ⟹
H ⊆ *linear-span R M* (*carrier R*) *H*
apply (*rule subsetI*)
apply (*cut-tac sc-Ring*,
cut-tac Ring.whole-ideal[*of R*])
apply (*frule-tac A* = *carrier R* **and** *H* = *H* **and** *a* = *1*$_r$*R*
and *h* = *x* **in** *sc-linear-span*, *assumption*+)
apply (*simp add*:*Ring.ring-one*, *assumption*+)
apply (*frule-tac c* = *x* **in** *subsetD*[*of H carrier M*], *assumption*+,
simp add:*sprod-one*, *assumption*)
done

lemma (**in** *Module*) *linear-span-inc-0*:⟦*ideal R A*; *H* ⊆ *carrier M*⟧ ⟹
0 ∈ *linear-span R M A H*
apply (*case-tac H* = {})
apply (*simp add*:*linear-span-def*)

apply (*frule nonempty-ex*[*of H*], *erule exE*)
apply (*frule-tac h* = *x* **in** *sc-linear-span*[*of A H* **0**$_R$], *assumption*)
apply (*cut-tac sc-Ring*, *simp add*:*Ring.ideal-zero*, *assumption*)
apply (*frule-tac c* = *x* **in** *subsetD*[*of H carrier M*], *assumption*,
simp add:*sc-0-m*)
done

lemma (**in** *Module*) *linear-span-iOp-closedTr1*:⟦*ideal R A*;
s ∈ {*j*. *j* ≤ (*n*::*nat*)} → *A*⟧ ⟹
(λ*x*∈{*j*. *j* ≤ *n*}. −$_a$*R* (*s x*)) ∈ {*j*. *j* ≤ *n*} → *A*
apply (*rule Pi-I*)
apply *simp*
apply (*cut-tac sc-Ring*,
rule Ring.ideal-inv1-closed, *assumption*+)
apply (*simp add*:*Pi-def*)
done

lemma (**in** *Module*) *l-span-gen-mono*:⟦$K \subseteq H$; $H \subseteq$ *carrier* M; *ideal* R A⟧ ⟹
$\quad$ *linear-span* R M A $K \subseteq$ *linear-span* R M A H
apply (*rule subsetI*)
apply (*case-tac* $K = \{\}$, *simp add*:*linear-span-def*[*of* - - - {}],
$\quad$ *simp add*:*linear-span-inc-0*)
apply (*frule nonempty-ex*[*of* K], *erule exE*,
$\quad$ *frule-tac* $c = xa$ **in** *subsetD*[*of* K H], *assumption*+,
$\quad$ *frule nonempty*[*of* - H])
apply (*simp add*:*linear-span-def*[*of* - - - K],
$\quad$ *erule exE*, (*erule bexE*)+, *simp*,
$\quad$ *frule extend-fun*[*of* - - K H], *assumption*+)
apply (*simp add*: *l-comb-mem-linear-span*)
done

lemma (**in** *Module*) *l-comb-add*:⟦*ideal* R A; $H \subseteq$ *carrier* M;
$\quad$ $s \in \{j.\ j \leq (n::nat)\} \to A$; $f \in \{j.\ j \leq n\} \to H$;
$\quad$ $t \in \{j.\ j \leq (m::nat)\} \to A$; $g \in \{j.\ j \leq m\} \to H$⟧ ⟹
l-comb R M (*Suc* $(n + m)$) (*jointfun* n s m t) (*jointfun* n f m g) =
$\quad$ *l-comb* R M n s f ± *l-comb* R M m t g
apply (*cut-tac sc-Ring*)
apply (*simp del*:*nsum-suc add*:*l-comb-def*)
apply (*subst nsum-split*)
apply (*rule allI*, *rule impI*)
apply (*case-tac* $j \leq n$, *simp add*:*jointfun-def*,
$\quad$ *rule sc-mem*, *simp add*:*Pi-def Ring.ideal-subset*,
$\quad$ *simp add*:*Pi-def subsetD*)
apply (*simp add*:*jointfun-def sliden-def*)
apply (*frule-tac* $m = j$ **and** $n =$ *Suc* $(n + m)$ **and** $l =$ *Suc* n **in** *diff-le-mono*,
$\quad$ *thin-tac* $j \leq$ *Suc* $(n + m)$, *simp*,
$\quad$ *rule sc-mem*, *simp add*:*Pi-def Ring.ideal-subset*,
$\quad$ *simp add*:*Pi-def subsetD*)
apply (*simp add*:*l-comb-jointfun-jj*[*of* H A s n f t m g])
apply (*cut-tac nsum-eq*[*of* m *cmp* ($\lambda j.$ *jointfun* n s m t j $\cdot_s$
$\quad$ *jointfun* n f m g j) (*slide* (*Suc* n)) $\lambda j.$ t j $\cdot_s$ g j], *simp*)
apply (*rule allI*, *rule impI*, *simp add*:*cmp-def*,
$\quad$ *simp add*:*jointfun-def sliden-def slide-def*,
$\quad$ *rule sc-mem*, *simp add*:*Pi-def Ring.ideal-subset*,
$\quad$ *simp add*:*Pi-def subsetD*)
apply (*rule allI*, *rule impI*,
$\quad$ *rule sc-mem*, *simp add*:*Pi-def Ring.ideal-subset*,
$\quad$ *simp add*:*Pi-def subsetD*)
apply (*simp add*:*cmp-def jointfun-def sliden-def slide-def*)
done

lemma (**in** *Module*) *l-comb-add1Tr*:⟦*ideal* R A; $H \subseteq$ *carrier* M⟧ ⟹
$f \in \{j.\ j \leq (n::nat)\} \to H \wedge s \in \{j.\ j \leq n\} \to A \wedge t \in \{j.\ j \leq n\} \to A \longrightarrow$
$\quad$ *l-comb* R M n ($\lambda x \in \{j.\ j \leq n\}.$ $(s\ x) \pm_R (t\ x)$) f =
$\quad$ *l-comb* R M n s f ± *l-comb* R M n t f
apply (*induct-tac* n)

apply (*simp add*:*l-comb-def sc-Ring Ring.ideal-subset subsetD sc-l-distr*)

apply (*rule impI*, (*erule conjE*)+)
apply (*frule func-pre*[*of f*], *frule func-pre*[*of s*], *frule func-pre*[*of t*],
simp)
apply (*simp add*:*l-comb-def*, *cut-tac sc-Ring*)
apply (*cut-tac n* = *n* **and** *f* = $\lambda j.$ (*if j* $\leq$ *n then s j* $\pm_R$ *t j else undefined*) $\cdot_s$ *f j*
and *g* = $\lambda j.$ (*if j* $\leq$ *Suc n then s j* $\pm_R$ *t j else undefined*) $\cdot_s$
f j **in** *nsum-eq*)
apply (*rule allI*, *rule impI*, *simp*,
rule sc-mem, *frule Ring.ring-is-ag*,
rule aGroup.ag-pOp-closed[*of R*], *assumption*,
simp add:*Pi-def Ring.ideal-subset*,
simp add:*Pi-def Ring.ideal-subset*,
simp add:*Pi-def subsetD*)
apply (*rule allI*, *rule impI*, *simp*,
rule sc-mem, *frule Ring.ring-is-ag*,
rule aGroup.ag-pOp-closed[*of R*], *assumption*,
simp add:*Pi-def Ring.ideal-subset*,
simp add:*Pi-def Ring.ideal-subset*,
simp add:*Pi-def subsetD*)
apply (*simp*)
apply *simp*
apply (*thin-tac* Σ_e *M* ($\lambda j.$ (*if j* $\leq$ *n then s j* $\pm_R$ *t j else undefined*) $\cdot_s$ *f j*)
n = Σ_e *M* ($\lambda j.$ *s j* $\cdot_s$ *f j*) *n* $\pm$ Σ_e *M* ($\lambda j.$ *t j* $\cdot_s$ *f j*) *n*,
thin-tac Σ_e *M* ($\lambda j.$ (*if j* $\leq$ *Suc n then s j* $\pm_R$ *t j else undefined*) $\cdot_s$
f j) *n* = Σ_e *M* ($\lambda j.$ *s j* $\cdot_s$ *f j*) *n* $\pm$ Σ_e *M* ($\lambda j.$ *t j* $\cdot_s$ *f j*) *n*)
apply (*frule-tac x* = *Suc n* **and** *A* = {*j*. *j* $\leq$ *Suc n*} **in**
funcset-mem[*of s* - *A*], *simp*,
frule-tac x = *Suc n* **and** *A* = {*j*. *j* $\leq$ *Suc n*} **in**
funcset-mem[*of t* - *A*], *simp*,
frule-tac x = *Suc n* **and** *A* = {*j*. *j* $\leq$ *Suc n*} **in**
funcset-mem[*of f* - *H*], *simp*,
cut-tac sc-Ring,
frule-tac h = *s* (*Suc n*) **in** *Ring.ideal-subset*, *assumption*+,
frule-tac h = *t* (*Suc n*) **in** *Ring.ideal-subset*, *assumption*+,
frule-tac c = *f* (*Suc n*) **in** *subsetD*[*of H carrier M*], *assumption*+)
apply (*simp add*:*sc-l-distr*)
apply (*cut-tac n* = *n* **and** *f* = $\lambda j.$ *s j* $\cdot_s$ *f j* **in** *nsum-mem*,
rule allI, *rule impI*, *rule sc-mem*,
simp add:*Pi-def Ring.ideal-subset*,
simp add:*Pi-def subsetD*)
apply (*cut-tac n* = *n* **and** *f* = $\lambda j.$ *t j* $\cdot_s$ *f j* **in** *nsum-mem*,
rule allI, *rule impI*, *rule sc-mem*,
simp add:*Pi-def Ring.ideal-subset*,
simp add:*Pi-def subsetD*)
apply (*cut-tac a* = *s* (*Suc n*) **and** *m* = *f* (*Suc n*) **in** *sc-mem*,
simp add:*Pi-def Ring.ideal-subset*,
simp add:*Pi-def subsetD*)

apply (*cut-tac* $a = t\ (Suc\ n)$ **and** $m = f\ (Suc\ n)$ **in** *sc-mem*,
simp add:*Pi-def Ring.ideal-subset*,
simp add:*Pi-def subsetD*)
apply (*subst pOp-assocTr41*[*THEN sym*], *assumption*+,
subst pOp-assocTr42, *assumption*+)
apply (*frule-tac* $x = \Sigma_e\ M\ (\lambda j.\ t\ j \cdot_s f\ j)\ n$ **and**
$y = s\ (Suc\ n) \cdot_s f\ (Suc\ n)$ **in** *ag-pOp-commute*, *assumption*+, *simp*)
apply (*subst pOp-assocTr42*[*THEN sym*], *assumption*+,
subst pOp-assocTr41, *assumption*+, *simp*)
done

lemma (**in** *Module*) *l-comb-add1*:⟦*ideal R A*; $H \subseteq$ *carrier M*;
$f \in \{j.\ j \leq (n::nat)\} \to H$; $s \in \{j.\ j \leq n\} \to A$; $t \in \{j.\ j \leq n\} \to A$ ⟧ ⟹
l-comb R M n $(\lambda x{\in}\{j.\ j \leq n\}.\ (s\ x) \pm_R (t\ x))\ f =$
l-comb R M n s f $\pm$ *l-comb R M n t f*
apply (*simp add*:*l-comb-add1Tr*)
done

lemma (**in** *Module*) *linear-span-iOp-closedTr2*:⟦*ideal R A*; $H \subseteq$ *carrier M*;
$f \in \{j.\ j \leq (n::nat)\} \to H$; $s \in \{j.\ j \leq n\} \to A$⟧ ⟹
$-_a$ (*l-comb R M n s f*) =
l-comb R M n $(\lambda x{\in}\{j.\ j \leq n\}.\ -_{a R}\ (s\ x))\ f$
apply (*frule-tac* $f = f$ **and** $A = \{j.\ j \leq n\}$ **and** $B = H$ **and** $x = 0$ **in**
funcset-mem, *simp*)
apply (*frule-tac* $A = A$ **and** $s = s$ **in** *linear-span-iOp-closedTr1*, *assumption*+)
apply (*frule l-comb-add1*[*of A H f n s* $\lambda x{\in}\{j.\ j \leq n\}.\ -_{a R}\ (s\ x)$],
assumption+)
apply (*cut-tac linear-comb0-1*[*of H* $\lambda x{\in}\{j.\ j \leq n\}.\ s\ x \pm_R$
$(\lambda x{\in}\{j.\ j \leq n\}.\ -_{a R}\ (s\ x))\ x\ n\ f$])
apply (*simp*,
thin-tac l-comb R M n
$(\lambda x{\in}\{j.\ j \leq n\}.\ s\ x \pm_R (if\ x \leq n\ then\ -_{a R}\ (s\ x)\ else\ undefined))\ f = \mathbf{0}$)
apply (*frule l-comb-mem*[*of A H s n f*], *assumption*+,
frule l-comb-mem[*of A H* $\lambda x{\in}\{j.\ j \leq n\}.\ -_{a R}\ (s\ x)$ *n f*], *assumption*+)
apply (*frule ag-mOp-closed*[*of l-comb R M n s f*])
apply (*frule ag-pOp-assoc*[*of* $-_a$ (*l-comb R M n s f*) *l-comb R M n s f l-comb R*
M n $(\lambda x{\in}\{j.\ j \leq n\}.\ -_{a R}\ (s\ x))\ f$], *assumption*+)
apply (*simp*, *simp add*:*ag-l-inv1*, *simp add*:*ag-l-zero*, *simp add*:*ag-r-zero*)
apply *assumption*+
apply (*rule Pi-I*, *simp*)
apply (*frule-tac* $x = x$ **in** *funcset-mem*[*of s* $\{j.\ j \leq n\}$ *A*], *simp*,
cut-tac sc-Ring,
frule-tac $h = s\ x$ **in** *Ring.ideal-subset*[*of R A*], *assumption*+)
apply (*frule Ring.ring-is-ag*[*of R*],
simp add:*aGroup.ag-r-inv1*[*of R*])
apply *assumption*
done

lemma (**in** *Module*) *linear-span-iOp-closed*:⟦*ideal R A*; $H \subseteq$ *carrier M*;

$a \in$ *linear-span R M A H*$\rrbracket \Longrightarrow -_a\ a \in$ *linear-span R M A H*
apply (*case-tac H* $= \{\}$)
apply (*simp add*:*linear-span-def*)
apply (*simp add*:*ag-inv-zero*)
apply (*simp add*:*linear-span-def*, *erule exE*, (*erule bexE*)+)
apply *simp*
apply (*frule-tac f* $= f$ **and** $n = n$ **and** $s = s$ **in**
linear-span-iOp-closedTr2[*of A H*], *assumption*+)
apply (*subgoal-tac* $(\lambda x{\in}\{j.\ j \leq n\}.\ -_{aR}\ (s\ x)) \in \{j.\ j \leq n\} \to A$)
apply *blast*
apply (*rule Pi-I*, *simp*)
apply(*cut-tac sc-Ring*,
rule Ring.ideal-inv1-closed, *assumption*+,
simp add:*Pi-def*)
done

lemma (**in** *Module*) *linear-span-pOp-closed*:
$\llbracket$*ideal R A*; $H \subseteq$ *carrier M*; $a \in$ *linear-span R M A H*; $b \in$ *linear-span R M A H*$\rrbracket$
$\Longrightarrow a \pm b \in$ *linear-span R M A H*
apply (*case-tac H* $= \{\}$)
apply (*simp add*:*linear-span-def*)
apply (*cut-tac ag-inc-zero*, *simp add*:*ag-r-zero*)
apply (*simp add*:*linear-span-def*)
apply ((*erule exE*)+, (*erule bexE*)+)
apply (*rename-tac n m f g s t*)
apply (*simp add*:*l-comb-def*)
apply (*cut-tac* $n = n$ **and** $f = \lambda j.\ s\ j \cdot_s f\ j$ **and** $m = m$ **and**
$g = \lambda j.\ t\ j \cdot_s g\ j$ **in** *nsum-add-nm*)
apply (*rule allI*, *rule impI*, *rule sc-mem*,
cut-tac sc-Ring,
simp add:*Pi-def Ring.ideal-subset*,
simp add:*Pi-def subsetD*)
apply (*rule allI*, *rule impI*, *rule sc-mem*,
cut-tac sc-Ring,
simp add:*Pi-def Ring.ideal-subset*,
simp add:*Pi-def subsetD*)
apply (*rotate-tac* -1, *frule sym*,
thin-tac $\Sigma_e\ M\ (jointfun\ n\ (\lambda j.\ s\ j \cdot_s f\ j)\ m\ (\lambda j.\ t\ j \cdot_s g\ j))$
$(Suc\ (n + m)) =$
$\Sigma_e\ M\ (\lambda j.\ s\ j \cdot_s f\ j)\ n \pm \Sigma_e\ M\ (\lambda j.\ t\ j \cdot_s g\ j)\ m$,
simp del:*nsum-suc*)
apply (*cut-tac* $n = Suc\ (n + m)$ **and** $f = jointfun\ n\ (\lambda j.\ s\ j \cdot_s f\ j)\ m$
$(\lambda j.\ t\ j \cdot_s g\ j)$ **and** $g = \lambda j.\ (jointfun\ n\ s\ m\ t)\ j \cdot_s (jointfun\ n\ f\ m\ g)\ j$
in *nsum-eq*)
apply (*rule allI*, *rule impI*)
apply (*simp add*:*jointfun-def*)
apply (*case-tac* $j \leq n$, *simp*)
apply (*rule sc-mem*,

cut-tac sc-Ring,
simp add:*Pi-def Ring.ideal-subset*,
simp add:*Pi-def subsetD*)
apply (*simp*, *rule sc-mem*)
apply (*simp add*:*sliden-def*,
frule-tac $m = j$ **and** $n = Suc\ (n + m)$ **and** $l = Suc\ n$ **in** *diff-le-mono*,
thin-tac $j \leq Suc\ (n + m)$, *simp*,
cut-tac sc-Ring,
simp add:*Pi-def Ring.ideal-subset*)
apply (*simp add*:*sliden-def*,
frule-tac $m = j$ **and** $n = Suc\ (n + m)$ **and** $l = Suc\ n$ **in** *diff-le-mono*,
thin-tac $j \leq Suc\ (n + m)$, *simp*,
cut-tac sc-Ring,
simp add:*Pi-def subsetD*)
apply (*rule allI*, *rule impI*)
apply (*simp add*:*jointfun-def*)
apply (*case-tac* $j \leq n$, *simp*)
apply (*rule sc-mem*,
cut-tac sc-Ring,
simp add:*Pi-def Ring.ideal-subset*,
simp add:*Pi-def subsetD*)
apply (*simp*, *simp add*:*sliden-def*,
rule sc-mem,
frule-tac $m = j$ **and** $n = Suc\ (n + m)$ **and** $l = Suc\ n$ **in** *diff-le-mono*,
thin-tac $j \leq Suc\ (n + m)$, *simp*,
cut-tac sc-Ring,
simp add:*Pi-def Ring.ideal-subset*)
apply (*frule-tac* $m = j$ **and** $n = Suc\ (n + m)$ **and** $l = Suc\ n$ **in** *diff-le-mono*,
thin-tac $j \leq Suc\ (n + m)$, *simp*,
cut-tac sc-Ring,
simp add:*Pi-def subsetD*)
apply (*rule allI*, *rule impI*,
simp add:*jointfun-def*)
apply (*simp del*:*nsum-suc*,
thin-tac $\Sigma_e\ M\ (\lambda j.\ s\ j\ \cdot_s\ f\ j)\ n \pm \Sigma_e\ M\ (\lambda j.\ t\ j\ \cdot_s\ g\ j)\ m =$
$\Sigma_e\ M\ (\lambda j.\ jointfun\ n\ s\ m\ t\ j\ \cdot_s\ jointfun\ n\ f\ m\ g\ j)\ (Suc\ (n + m))$,
thin-tac $\Sigma_e\ M\ (jointfun\ n\ (\lambda j.\ s\ j\ \cdot_s\ f\ j)\ m\ (\lambda j.\ t\ j\ \cdot_s\ g\ j))$
$(Suc\ (n + m)) =$
$\Sigma_e\ M\ (\lambda j.\ jointfun\ n\ s\ m\ t\ j\ \cdot_s\ jointfun\ n\ f\ m\ g\ j)\ (Suc\ (n + m))$)
apply (*frule-tac* $f = s$ **and** $n = n$ **and** $A = A$ **and** $g = t$ **and** $m = m$ **and** $B = A$ **in**
jointfun-hom0, *assumption*+, *simp del*:*nsum-suc*,
frule-tac $f = f$ **and** $n = n$ **and** $A = H$ **and** $g = g$ **and** $m = m$ **and** $B = H$ **in**
jointfun-hom0, *assumption*+, *simp del*:*nsum-suc*)
apply *blast*
done

lemma (**in** *Module*) *l-comb-scTr*:⟦*ideal R A*; $H \subseteq$ *carrier M*;

$r \in carrier\ R;\ H \neq \{\}\rrbracket \Longrightarrow s \in \{j.\ j \leq (n::nat)\} \to A \wedge$
$g \in \{j.\ j \leq n\} \to\ H \longrightarrow r \cdot_s (nsum\ M\ (\lambda k.\ (s\ k) \cdot_s (g\ k))\ \ n) =$
$nsum\ M\ (\lambda k.\ r \cdot_s ((s\ k) \cdot_s (g\ k)))\ n$
apply (*induct-tac n*)
apply (*rule impI*, (*erule conjE*)+, *simp*)

apply (*rule impI*) **apply** (*erule conjE*)
apply (*frule func-pre* [*of* - - *A*]) **apply** (*frule func-pre* [*of* - - *H*])
apply (*simp*)
apply (*cut-tac n = n* **and** $f = \lambda k.\ s\ k \cdot_s g\ k$ **in** *nsum-mem*,
rule allI, *rule impI*,
cut-tac sc-Ring, *rule sc-mem*,
simp add:*Pi-def Ring.ideal-subset*,
simp add:*Pi-def subsetD*)
apply (*cut-tac a = s* (*Suc n*) **and** *m = g* (*Suc n*) **in** *sc-mem*,
cut-tac sc-Ring,
simp add:*Pi-def Ring.ideal-subset*,
simp add:*Pi-def subsetD*)
apply (*simp add*:*sc-r-distr*)
done

lemma (**in** *Module*) *l-comb-sc1Tr*:$\llbracket$*ideal R A*; $H \subseteq$ *carrier M*;
$r \in carrier\ R;\ H \neq \{\}\rrbracket \Longrightarrow s \in \{j.\ j \leq (n::nat)\} \to A \wedge$
$g \in \{j.\ j \leq n\} \to\ H \longrightarrow r \cdot_s (nsum\ M\ (\lambda k.\ (s\ k) \cdot_s (g\ k))\ \ n) =$
$nsum\ M\ (\lambda k.\ (r \cdot_{r\,R} (s\ k)) \cdot_s (g\ k))\ n$
apply (*cut-tac sc-Ring*)
apply (*induct-tac n*)
apply (*rule impI*, (*erule conjE*)+, *simp*)
apply (*subst sc-assoc*, *assumption*+,
simp add:*Pi-def Ring.ideal-subset*,
simp add:*Pi-def subsetD*, *simp*)

apply (*rule impI*) **apply** (*erule conjE*)
apply (*frule func-pre* [*of* - - *A*], *frule func-pre* [*of* - - *H*])
apply *simp*
apply (*cut-tac n = n* **and** $f = \lambda k.\ s\ k \cdot_s g\ k$ **in** *nsum-mem*,
rule allI, *rule impI*,
cut-tac sc-Ring, *rule sc-mem*,
simp add:*Pi-def Ring.ideal-subset*,
simp add:*Pi-def subsetD*)
apply (*cut-tac a = s* (*Suc n*) **and** *m = g* (*Suc n*) **in** *sc-mem*,
cut-tac sc-Ring,
simp add:*Pi-def Ring.ideal-subset*,
simp add:*Pi-def subsetD*)
apply (*simp add*:*sc-r-distr*)
apply (*subst sc-assoc*, *assumption*+,
simp add:*Pi-def Ring.ideal-subset*,
simp add:*Pi-def subsetD*, *simp*)
done

lemma (**in** *Module*) *l-comb-sc*:⟦*ideal R A*; *H* ⊆ *carrier M*; *r* ∈ *carrier R*;
s ∈ {*j*. *j* ≤ (*n*::*nat*)} → *A*; *g* ∈ {*j*. *j* ≤ *n*} → *H*⟧ ⟹
r ·$_s$ (*nsum M* (λ*k*. (*s k*) ·$_s$ (*g k*)) *n*) = *nsum M* (λ*k*. *r* ·$_s$ ((*s k*) ·$_s$ (*g k*))) *n*
apply (*case-tac H* ≠ {})
apply (*simp add*:*l-comb-scTr*)
apply *simp*
apply (*frule-tac x* = *0* **in** *funcset-mem*[*of g* {*j*. *j* ≤ *n*} {}], *simp*)
apply *blast*
done

lemma (**in** *Module*) *l-comb-sc1*:⟦*ideal R A*; *H* ⊆ *carrier M*; *r* ∈ *carrier R*;
s ∈ {*j*. *j* ≤ (*n*::*nat*)} → *A*; *g* ∈ {*j*. *j* ≤ *n*} → *H*⟧ ⟹
r ·$_s$ (*nsum M* (λ*k*. (*s k*) ·$_s$ (*g k*)) *n*) = *nsum M* (λ*k*. (*r* ·$_{r\,R}$ (*s k*)) ·$_s$ (*g k*)) *n*
apply (*case-tac H* ≠ {})
apply (*simp add*:*l-comb-sc1Tr*)
apply *simp*
apply (*frule-tac x* = *0* **in** *funcset-mem*[*of g* {*j*. *j* ≤ *n*} {}], *simp*)
apply *blast*
done

lemma (**in** *Module*) *linear-span-sc-closed*:⟦*ideal R A*; *H* ⊆ *carrier M*;
r ∈ *carrier R*; *x* ∈ *linear-span R M A H*⟧ ⟹ *r* ·$_s$ *x* ∈ *linear-span R M A H*
apply (*case-tac H* = {})
apply (*simp add*:*linear-span-def*)
apply (*simp add*:*sc-a-0*)
apply (*simp add*:*linear-span-def*)
apply (*erule exE*, (*erule bexE*)+)
apply (*simp add*:*l-comb-def*)
apply (*simp add*:*l-comb-sc*)

apply (*cut-tac n* = *n* **and** *f* = λ*j*. *r* ·$_s$ (*s j* ·$_s$ *f j*) **and**
g = λ*j*. (*r* ·$_{r\,R}$ (*s j*)) ·$_s$ *f j* **in** *nsum-eq*)
apply (*rule allI*, *rule impI*,
rule sc-mem, *assumption*, *rule sc-mem*,
cut-tac sc-Ring,
simp add:*Pi-def Ring.ideal-subset*,
simp add:*Pi-def subsetD*)
apply (*rule allI*, *rule impI*,
rule sc-mem,
cut-tac sc-Ring,
rule Ring.ring-tOp-closed, *assumption*+,
simp add:*Pi-def Ring.ideal-subset*,
simp add:*Pi-def subsetD*)
apply (*rule allI*, *rule impI*,
subst sc-assoc, *assumption*,
cut-tac sc-Ring,
simp add:*Pi-def Ring.ideal-subset*,
simp add:*Pi-def subsetD*, *simp*,

```
    simp,
   thin-tac Σe M (λj. r ·s (s j ·s f j)) n =
                          Σe M (λj. (r ·rR s j) ·s f j) n,
   thin-tac x = Σe M (λj. s j ·s f j) n)

 apply (cut-tac n = n and f = λj. (r ·rR s j) ·s f j and
      g = λj. (λx∈{j. j ≤ n}. r ·rR (s x)) j ·s f j in nsum-eq)
  apply (rule allI, rule impI,
       rule sc-mem,
       cut-tac sc-Ring,
       rule Ring.ring-tOp-closed, assumption+,
       simp add:Pi-def Ring.ideal-subset,
       simp add:Pi-def subsetD)
   apply (rule allI, rule impI,
        rule sc-mem, simp) apply (
         cut-tac sc-Ring,
       rule Ring.ring-tOp-closed, assumption+,
       simp add:Pi-def Ring.ideal-subset,
       simp add:Pi-def subsetD)
  apply (rule allI, rule impI)
        apply simp
  apply (subgoal-tac (λx∈{j. j ≤ n}. r ·rR s x) ∈ {j. j ≤ n} → A,
        blast)
  apply (rule Pi-I, simp)
apply (thin-tac Σe M (λj. (r ·rR s j) ·s f j) n =
       Σe M (λj. (if j ≤ n then r ·rR s j else undefined) ·s f j) n,
       cut-tac sc-Ring,
       rule Ring.ideal-ring-multiple, assumption+, simp add:Pi-def,
       assumption)
done

lemma (in Module) mem-single-l-spanTr:⟦ideal R A; h ∈ carrier M⟧ ⟹
      s ∈ {j. j ≤ (n::nat)} → A ∧
      f ∈ {j. j ≤ n} → {h} ∧ l-comb R M n s f ∈ linear-span R M A {h}
      ⟶ (∃ a ∈ A. l-comb R M n s f = a ·s h)
apply (cut-tac sc-Ring)
apply (induct-tac n)
 apply (rule impI, (erule conjE)+)
 apply (simp add:l-comb-def Ring.ideal-subset[of R A] bexI[of - s 0])
apply (rule impI, (erule conjE)+,
       frule func-pre[of - - A], frule func-pre[of - - {h}],
       frule-tac n = n in l-comb-mem-linear-span[of A {h} s - f],
       rule subsetI, simp, assumption+, simp,
       erule bexE)
apply (frule singleton-sub[of h carrier M])
 apply (frule Ring.ideal-subset1[of R A], assumption)
 apply (frule extend-fun[of s - A carrier R], assumption)
 apply (frule-tac n = n in l-comb-Suc[of {h} A s - f], assumption+,
       simp)
```

apply (*frule-tac* $A = \{j.\ j \leq Suc\ n\}$ **and** $x = Suc\ n$ **in**
funcset-mem[*of* f - $\{h\}$], *simp*, *simp*,
frule-tac $A = \{j.\ j \leq Suc\ n\}$ **and** $x = Suc\ n$ **in**
funcset-mem[*of* s - A], *simp*,
frule-tac $h = s\ (Suc\ n)$ **in** *Ring.ideal-subset*[*of* $R\ A$], *assumption*+)
apply (*frule-tac* $h = a$ **in** *Ring.ideal-subset*[*of* $R\ A$], *assumption*+,
frule-tac $h = s\ (Suc\ n)$ **in** *Ring.ideal-subset*[*of* $R\ A$], *assumption*+,
simp add:*sc-l-distr*[*THEN sym*],
frule-tac $x = a$ **and** $y = s\ (Suc\ n)$ **in** *Ring.ideal-pOp-closed*[*of* $R\ A$],
assumption+, *blast*)
done

lemma (**in** *Module*) *mem-single-l-span*:⟦*ideal* $R\ A$; $h \in carrier\ M$;
$s \in \{j.\ j \leq (n::nat)\} \to A$; $f \in \{j.\ j \leq n\} \to \{h\}$;
l-comb $R\ M\ n\ s\ f \in$ *linear-span* $R\ M\ A\ \{h\}$⟧ ⟹
$\exists\, a \in A.$ *l-comb* $R\ M\ n\ s\ f = a \cdot_s h$
apply (*simp add*:*mem-single-l-spanTr*)
done

lemma (**in** *Module*) *mem-single-l-span1*:⟦*ideal* $R\ A$; $h \in carrier\ M$;
$x \in$ *linear-span* $R\ M\ A\ \{h\}$⟧ ⟹ $\exists\, a \in A.\ x = a \cdot_s h$
apply (*simp add*:*linear-span-def*, *erule exE*, (*erule bexE*)+, *simp*)
apply (*frule-tac* $s = s$ **and** $n = n$ **and** $f = f$ **in** *mem-single-l-span*[*of* $A\ h$],
assumption+)
apply (*frule singleton-sub*[*of* h *carrier* M],
rule-tac $s = s$ **and** $f = f$ **in** *l-comb-mem-linear-span*[*of* $A\ \{h\}$],
assumption+)
done

lemma (**in** *Module*) *linear-span-subModule*:⟦*ideal* $R\ A$; $H \subseteq carrier\ M$⟧ ⟹
submodule $R\ M$ (*linear-span* $R\ M\ A\ H$)
apply (*case-tac* $H = \{\}$)
apply (*simp add*:*linear-span-def*)
apply (*simp add*:*submodule-0*)

apply (*simp add*:*submodule-def*)
apply (*rule conjI*)
apply (*simp add*:*linear-span-def*)
apply (*rule subsetI*)
apply (*simp add*:*CollectI*)
apply (*erule exE*, (*erule bexE*)+)
apply *simp*
apply (*simp add*:*l-comb-mem*)
apply (*rule conjI*)
apply (*rule asubg-test*)
apply (*rule subsetI*) **apply** (*simp add*:*linear-span-def*)
apply (*erule exE*, (*erule bexE*)+)
apply (*simp add*:*l-comb-mem*)
apply (*frule linear-span-inc-0*[*of* $A\ H$], *assumption*, *blast*)

apply (*rule ballI*)+
apply (*rule linear-span-pOp-closed*, *assumption*+)
apply (*rule linear-span-iOp-closed*, *assumption*+)
apply (*rule allI*)+
apply (*simp add*:*linear-span-sc-closed*)
done

lemma (**in** *Module*) *l-comb-mem-submoduleTr*:⟦*ideal R A*; *submodule R M N*⟧ ⟹
(*s* ∈ {*j*. *j* ≤ (*n*::*nat*)} → *A* ∧ *f* ∈ {*j*. *j* ≤ *n*} → *carrier M* ∧
(∀ *j* ≤ *n*.(*s j*) ·$_s$ (*f j*) ∈ *N*)) ⟶ *l-comb R M n s f* ∈ *N*
apply (*induct-tac n*)
apply (*simp add*:*l-comb-def*, *rule impI*, (*erule conjE*)+)
apply (*frule func-pre*[*of - - A*], *frule func-pre*[*of - - carrier M*], *simp*)
apply (*simp add*:*l-comb-def*)
apply (*frule-tac a* = *Suc n* **in** *forall-spec*, *simp*)
apply (*rule submodule-pOp-closed*, *assumption*+)
done

lemma (**in** *Module*) *l-span-sub-submodule*:⟦*ideal R A*; *submodule R M N*; *H* ⊆ *N*⟧
⟹
linear-span R M A H ⊆ *N*
apply (*cut-tac sc-Ring*)
apply (*rule subsetI*, *simp add*:*linear-span-def*)
apply (*case-tac H* = {}, *simp*)
apply (*simp add*:*submodule-inc-0*)

apply *simp*
apply (*erule exE*, (*erule bexE*)+)
apply (*cut-tac s* = *s* **and** *A* = *A* **and** *f* = *f* **and** *N* = *N* **and** *n* = *n* **in**
l-comb-mem-submoduleTr, *assumption*+,
frule submodule-subset[*of N*],
frule subset-trans[*of H N carrier M*], *assumption*+,
frule-tac f = *f* **and** *A* = {*j*. *j* ≤ *n*} **and** *B* = *H* **and** *?B1.0* = *carrier M*
in *extend-fun*, *assumption*+)
apply (*subgoal-tac* ∀ *j*≤*n*. *s j* ·$_s$ *f j* ∈ *N*, *simp*)
apply (*rule allI*, *rule impI*)
apply (*rule submodule-sc-closed*[*of N*], *assumption*,
simp add:*Pi-def Ring.ideal-subset*,
simp add:*Pi-def subsetD*)
done

lemma (**in** *Module*) *linear-span-sub*:⟦*ideal R A*; *H* ⊆ *carrier M*⟧ ⟹
(*linear-span R M A H*) ⊆ *carrier M*
apply (*frule linear-span-subModule*[*of A H*], *assumption*+)
apply (*simp add*:*submodule-subset*)
done

definition
smodule-ideal-coeff :: [('*r*, '*m*) *Ring-scheme*, ('*a*, '*r*, '*m1*) *Module-scheme*,

```
    'r set] ⇒ 'a set where
  smodule-ideal-coeff R M A = linear-span R M A (carrier M)

abbreviation
  SMLIDEALCOEFF  ((3-/ ⊙_ -) [64,64,65]64) where
  A ⊙_R M == smodule-ideal-coeff R M A

lemma (in Module) smodule-ideal-coeff-is-Submodule:ideal R A  ⟹
          submodule R M (A ⊙_R M)
apply (simp add:smodule-ideal-coeff-def)
apply (simp add:linear-span-subModule)
done

lemma (in Module) mem-smodule-ideal-coeff:⟦ideal R A; x ∈ A ⊙_R M⟧ ⟹
          ∃ n. ∃ s ∈ {j. j ≤ n} → A. ∃ g ∈ {j. j ≤ n} → carrier M.
           x = l-comb R M n s g
apply (cut-tac ag-inc-zero,
      frule nonempty[of 0 carrier M])
apply (simp add:smodule-ideal-coeff-def linear-span-def,
      erule exE, (erule bexE)+, blast)
done

definition
  quotient-of-submodules :: [('r, 'm) Ring-scheme, ('a, 'r, 'm1) Module-scheme,
          'a set, 'a set] ⇒ 'r set where
  quotient-of-submodules R M N P = {x | x. x∈carrier R ∧
                                (linear-span R M (Rxa R x)  P) ⊆ N}

definition
  Annihilator :: [('r, 'm) Ring-scheme, ('a, 'r, 'm1) Module-scheme]
    ⇒ 'r set ((Ann_ -) [82,83]82) where
  Ann_R M = quotient-of-submodules R M {0_M} (carrier M)

abbreviation
  QOFSUBMDS  ((4- _‡_ -) [82,82,82,83]82) where
  N _R‡M P == quotient-of-submodules R M N P

lemma (in Module) quotient-of-submodules-inc-0:
    ⟦submodule R M P; submodule R M Q⟧ ⟹ 0_R ∈ (P _R‡M Q)
apply (simp add:quotient-of-submodules-def)
apply (cut-tac sc-Ring, simp add:Ring.ring-zero)
apply (simp add:linear-span-def)
 apply (frule submodule-inc-0[of Q], simp add:nonempty)
apply (rule subsetI)
 apply (simp, erule exE, (erule bexE)+)
 apply (simp, thin-tac x = l-comb R M n s f, simp add:l-comb-def)
 apply (cut-tac n = n and f = λj. s j ·_s f j in nsum-zeroA)
 apply (rule allI, rule impI,
      frule-tac x = j and f = s and A = {j. j ≤ n} in
```

funcset-mem[*of* - - R $\diamondsuit_p$ $\mathbf{0}_R$], *simp*)
apply (*simp add*:*Rxa-def*, *erule bexE*, *simp*) **apply** (
simp add:*Ring.ring-times-x-0*,
rule sc-0-m) **apply** (
frule submodule-subset[*of Q*],
simp add:*Pi-def subsetD*)
apply (*simp add*:*submodule-inc-0*)
done

lemma (**in** *Module*) *quotient-of-submodules-is-ideal*:
$[\![$*submodule R M P*; *submodule R M Q*$]\!] \Longrightarrow$ *ideal R* (P $_{R\ddagger M}$ Q)
apply (*frule quotient-of-submodules-inc-0* [*of P Q*], *assumption*+)
apply (*cut-tac sc-Ring*,
rule Ring.ideal-condition[*of R*], *assumption*+)
apply (*simp add*:*quotient-of-submodules-def*)
apply (*rule subsetI*)
apply (*simp add*:*CollectI*)
apply (*simp add*:*nonempty*) **apply** (*thin-tac* $\mathbf{0}_R \in P$ $_{R\ddagger M}$ Q)
apply (*rule ballI*)+
apply (*simp add*:*quotient-of-submodules-def*)
apply (*erule conjE*)+
apply (*rule conjI*)
apply (*frule Ring.ring-is-ag*,
rule aGroup.ag-pOp-closed[*of R*], *assumption*+)
apply (*rule aGroup.ag-mOp-closed*, *assumption*+)
apply (*subst linear-span-def*)
apply (*frule submodule-inc-0* [*of Q*], *simp add*:*nonempty*)
apply (*rule subsetI*, *simp*,
erule exE, (*erule bexE*)+, *simp add*:*l-comb-def*,
thin-tac xa = Σ_e M ($\lambda j.$ s j $\cdot_s$ f j) n)
apply (*cut-tac s* = *s* **and** *n* = *n* **and** *f* = *f* **in**
l-comb-mem-submoduleTr[*of carrier R P*])
apply (*simp add*:*Ring.whole-ideal*, *assumption*+)
apply (*frule Ring.ring-is-ag*[*of R*],
frule-tac x = *y* **in** *aGroup.ag-mOp-closed*[*of R*], *assumption*+,
frule-tac x = *x* **and** *y* = $-_{aR}$ *y* **in** *aGroup.ag-pOp-closed*, *assumption*+,
frule-tac a = *x* $\pm_R$ $-_{aR}$ *y* **in** *Ring.principal-ideal*[*of R*], *assumption*+,
frule-tac I = *R* $\diamondsuit_p$ (*x* $\pm_R$ $-_{aR}$ *y*) **in** *Ring.ideal-subset1*, *assumption*+)
apply (*frule-tac f* = *s* **and** *A* = $\{j.\ j \leq n\}$ **and** *B* = *R* $\diamondsuit_p$ (*x* $\pm_R$ $-_{aR}$ *y*)
and *?B1.0* = *carrier R* **in** *extend-fun*, *assumption*+,
frule-tac submodule-subset[*of Q*],
frule-tac f = *f* **and** *A* = $\{j.\ j \leq n\}$ **and** *B* = *Q*
and *?B1.0* = *carrier M* **in** *extend-fun*, *assumption*+)
apply (*subgoal-tac* $\forall j{\leq}n.$ s j $\cdot_s$ f j $\in$ P, *simp add*:*l-comb-def*,
thin-tac s $\in \{j.\ j \leq n\} \to$ *carrier R* $\wedge$
$f \in \{j.\ j \leq n\} \to$ *carrier M* $\wedge$ ($\forall j{\leq}n.$ s j $\cdot_s$ f j $\in$ P) $\longrightarrow$
l-comb R M n s f $\in$ P,
thin-tac s $\in \{j.\ j \leq n\} \to$ *carrier R*,
thin-tac f $\in \{j.\ j \leq n\} \to$ *carrier M*)

apply (*rule allI*, *rule impI*,
frule-tac $x = j$ **and** $f = s$ **and** $A = \{j.\ j \leq n\}$ **and**
$B = R \diamondsuit_p (x \pm_R -_{aR} y)$ **in** *funcset-mem*, *simp*,
thin-tac $s \in \{j.\ j \leq n\} \to R \diamondsuit_p (x \pm_R -_{aR} y)$,
thin-tac ideal R $(R \diamondsuit_p (x \pm_R -_{aR} y))$)
apply (*simp add*:*Rxa-def*, *fold Rxa-def*, *erule bexE*, *simp*,
thin-tac $s\ j = r \cdot_{rR} (x \pm_R -_{aR} y)$)
apply (*simp add*:*Ring.ring-distrib1*,
frule-tac $x = r$ **and** $y = x$ **in** *Ring.ring-tOp-closed*, *assumption*+,
frule-tac $x = r$ **and** $y = -_{aR}\ y$ **in** *Ring.ring-tOp-closed*, *assumption*+,
frule-tac $x = j$ **and** $A = \{j.\ j \leq n\}$ **and** $B = Q$ **in** *funcset-mem*,
simp,
frule-tac $c = f\ j$ **in** *subsetD*[*of Q carrier M*], *assumption*+,
simp add:*sc-l-distr*)
apply (*subst Ring.ring-inv1-2*[*THEN sym*], *assumption*+,
subst Ring.ring-inv1-1, *assumption*+)
apply (*frule-tac* $a = x$ **in** *Ring.principal-ideal*[*of R*], *assumption*+,
frule-tac $a = x$ **in** *Ring.principal-ideal*[*of R*], *assumption*+,
frule-tac $A = R \diamondsuit_p x$ **and** $H = Q$ **and** $a = r \cdot_{rR} x$ **and** $h = f\ j$ **in**
sc-linear-span, *assumption*+, *simp add*:*Rxa-def*, *blast*,
simp add:*Pi-def*)
apply (*frule-tac* $x = r$ **in** *aGroup.ag-mOp-closed*[*of R*], *assumption*+,
frule-tac $a = y$ **in** *Ring.principal-ideal*[*of R*], *assumption*+,
frule-tac $a = y$ **in** *Ring.principal-ideal*[*of R*], *assumption*+,
frule-tac $A = R \diamondsuit_p y$ **and** $H = Q$ **and** $a = (-_{aR}\ r) \cdot_{rR} y$ **and**
$h = f\ j$ **in** *sc-linear-span*, *assumption*+, *simp add*:*Rxa-def*,
blast,
simp add:*Pi-def*)
apply (*frule-tac* $c = (r \cdot_{rR} x) \cdot_s f\ j$ **and** $A = \mathit{linear\text{-}span}\ R\ M\ (R \diamondsuit_p x)\ Q$
and $B = P$ **in** *subsetD*, *assumption*+) **apply** (
frule-tac $c = ((-_{aR}\ r) \cdot_{rR} y) \cdot_s f\ j$ **and**
$A = \mathit{linear\text{-}span}\ R\ M\ (R \diamondsuit_p y)\ Q$ **and** $B = P$ **in** *subsetD*, *assumption*+)
apply (*rule submodule-pOp-closed*, *assumption*+)

apply ((*rule ballI*)+,
thin-tac $\mathbf{0}_R \in P\ {}_{R\ddagger M}\ Q$,
simp add:*quotient-of-submodules-def*, *erule conjE*)
apply (*simp add*:*Ring.ring-tOp-closed*)
apply (*rule subsetI*)
apply (*frule submodule-inc-0*[*of Q*],
simp add:*linear-span-def nonempty*)
apply (*erule exE*, (*erule bexE*)+)
apply (*rule-tac* $c = xa$ **and** $A = \{xa.\ \exists n.\ \exists f{\in}\{j.\ j \leq n\} \to Q.$
$\exists s{\in}\{j.\ j \leq n\} \to R \diamondsuit_p x.\ xa = \mathit{l\text{-}comb}\ R\ M\ n\ s\ f\}$ **in**
subsetD[*of* - *P*], *assumption*+,
thin-tac $\{xa.\ \exists n.\ \exists f{\in}\{j.\ j \leq n\} \to Q.$
$\exists s{\in}\{j.\ j \leq n\} \to R \diamondsuit_p x.\ xa = \mathit{l\text{-}comb}\ R\ M\ n\ s\ f\} \subseteq P$)
apply *simp*
apply (*frule-tac* $a = r$ **and** $b = x$ **in** *Ring.Rxa-mult-smaller*[*of R*], *assumption*+)

apply (*frule-tac* $f = s$ **and** $A = \{j.\ j \leq n\}$ **and** $B = R \diamondsuit_p (r \cdot_{r R} x)$ **and**
?B1.0 $= R \diamondsuit_p x$ **in** *extend-fun*, *assumption*+)
apply *blast*
done

lemma (**in** *Module*) *Ann-is-ideal*:*ideal* R (Ann_R M)
apply (*simp add*:*Annihilator-def*)
apply (*rule quotient-of-submodules-is-ideal*)
apply (*simp add*:*submodule-0*)
apply (*simp add*:*submodule-whole*)
done

lemma (**in** *Module*) *linmap-im-of-lincombTr*:⟦*ideal* R A; R *module* N;
$f \in mHom\ R\ M\ N$; $H \subseteq carrier\ M$⟧ ⟹
$s \in \{j.\ j \leq (n::nat)\} \to A \wedge g \in \{j.\ j \leq n\} \to H \longrightarrow$
f (*l-comb* R M n s g) = *l-comb* R N n s (*cmp* f g)
apply (*induct-tac* n)
apply (*rule impI*) **apply** (*erule conjE*)
apply (*simp add*:*l-comb-def*)
apply (*cut-tac* $m = g\ 0$ **and** $f = f$ **and** $a = s\ 0$ **in** *mHom-lin* [*of* N],
assumption+,
simp add:*Pi-def subsetD*, *assumption*,
cut-tac sc-Ring,
simp add:*Pi-def Ring.ideal-subset*, *simp add*:*cmp-def*)

apply (*rule impI*, *erule conjE*)
apply (*frule-tac* $f = s$ **in** *func-pre*,
frule-tac $f = g$ **in** *func-pre*, *simp*)
apply (*simp add*:*l-comb-def*)
apply (*subst mHom-add*[*of* N f], *assumption*+)
apply (*rule nsum-mem*,
rule allI, *rule impI*, *rule sc-mem*,
cut-tac sc-Ring,
simp add:*Pi-def Ring.ideal-subset*,
simp add:*Pi-def subsetD*)
apply (*rule sc-mem*,
cut-tac sc-Ring,
simp add:*Pi-def Ring.ideal-subset*,
simp add:*Pi-def subsetD*, *simp*,
frule-tac $x = Suc\ n$ **and** $A = \{j.\ j \leq Suc\ n\}$ **and** $f = s$ **and**
$B = A$ **in** *funcset-mem*, *simp*,
cut-tac sc-Ring,
frule-tac $h = s$ (*Suc* n) **in** *Ring.ideal-subset*[*of* R A], *assumption*+,
frule-tac $x = Suc\ n$ **and** $A = \{j.\ j \leq Suc\ n\}$ **and** $f = g$ **and**
$B = H$ **in** *funcset-mem*, *simp*,
frule-tac $c = g$ (*Suc* n) **in** *subsetD*[*of* H *carrier* M], *assumption*+)
apply (*simp add*:*mHom-lin cmp-def*)
done

lemma (**in** *Module*) *linmap-im-lincomb*:⟦*ideal R A*; *R module N*; *f* ∈ *mHom R M N*;
H ⊆ *carrier M*; *s* ∈ {*j*. *j* ≤ (*n*::*nat*)} → *A*; *g* ∈ {*j*. *j* ≤ *n*} → *H* ⟧ ⟹
f (*l-comb R M n s g*) = *l-comb R N n s* (*cmp f g*)
apply (*simp add*:*linmap-im-of-lincombTr*)
done

lemma (**in** *Module*) *linmap-im-linspan*:⟦*ideal R A*; *R module N*; *f* ∈ *mHom R M N*;
H ⊆ *carrier M*; *s* ∈ {*j*. *j* ≤ (*n*::*nat*)} → *A*; *g* ∈ {*j*. *j* ≤ *n*} → *H* ⟧ ⟹
f (*l-comb R M n s g*) ∈ *linear-span R N A* (*f* ‘ *H*)
apply (*frule l-comb-mem-linear-span*[*of A H s n g*], *assumption*+)
apply (*simp add*:*linmap-im-lincomb*)
apply (*rule Module.l-comb-mem-linear-span*[*of N R A f* ‘ *H s n cmp f g*],
assumption+,
rule subsetI,
simp add:*image-def*, *erule bexE*, *simp*,
frule-tac c = *xa* **in** *subsetD*[*of H carrier M*], *assumption*+,
simp add:*mHom-mem*[*of N f*], *assumption*+)
apply (*rule Pi-I*, *simp add*:*cmp-def*)
apply (*frule-tac f* = *g* **and** *A* = {*j*. *j* ≤ *n*} **and** *B* = *H* **and** *x* = *x* **in**
funcset-mem, *simp*, *simp add*:*image-def*)
apply *blast*
done

lemma (**in** *Module*) *linmap-im-linspan1*:⟦*ideal R A*; *R module N*; *f* ∈ *mHom R M N*;
H ⊆ *carrier M*; *h* ∈ *linear-span R M A H*⟧ ⟹
f h ∈ *linear-span R N A* (*f* ‘ *H*)
apply (*simp add*:*linear-span-def* [*of R M*])
apply (*case-tac H* = {}, *simp add*:*linear-span-def*)
apply (*simp add*:*mHom-0*, *simp*)
apply (*erule exE*, (*erule bexE*)+)
apply (*simp add*:*linmap-im-linspan*)
done

definition
faithful :: [('*r*, '*m*) *Ring-scheme*, ('*a*, '*r*, '*m1*) *Module-scheme*]
⇒ *bool* **where**
faithful R M ⟷ *Ann*$_R$ *M* = {**0**$_R$}

5.3 nsum and Generators

definition
generator :: [('*r*, '*m*) *Ring-scheme*, ('*a*, '*r*, '*m1*) *Module-scheme*,
'*a set*] ⇒ *bool* **where**

generator R M H == H ⊆ carrier M ∧
linear-span R M (carrier R) H = carrier M

definition
finite-generator :: [('r, 'm) Ring-scheme, ('a, 'r, 'm1) Module-scheme,
'a set] ⇒ bool **where**
finite-generator R M H ⟷ finite H ∧ generator R M H

definition
fGOver :: [('a, 'r, 'm1) Module-scheme, ('r, 'm) Ring-scheme] ⇒ bool
where
fGOver M R ⟷ (∃ H. finite-generator R M H)

abbreviation
FGENOVER (**infixl** *fgover 70*) **where**
M fgover R == fGOver M R

lemma (**in** *Module*) *h-in-linear-span*:⟦*H ⊆ carrier M*; *h ∈ H*⟧ ⟹
h ∈ linear-span R M (carrier R) H
apply (*subst sprod-one* [*THEN sym*, *of h*])
apply (*simp add*:*subsetD*)
apply (*cut-tac sc-Ring*)
apply (*frule Ring.ring-one*)
apply (*rule sc-linear-span* [*of carrier R H* $1_{r\,R}$ *h*])
apply (*simp add*:*Ring.whole-ideal*) **apply** *assumption*+
done

lemma (**in** *Module*) *generator-sub-carrier*:*generator R M H* ⟹
H ⊆ carrier M
apply (*simp add*:*generator-def*)
done

lemma (**in** *Module*) *lin-span-sub-carrier*:⟦*ideal R A*;
H ⊆ carrier M⟧ ⟹ *linear-span R M A H ⊆ carrier M*
apply (*cut-tac sc-Ring*)
apply (*rule subsetI*)
apply (*simp add*:*linear-span-def*)
apply (*case-tac H = {}*) **apply** *simp*
apply (*simp add*:*module-inc-zero*)
apply *simp*
apply (*erule exE*, (*erule bexE*)+, *simp*,
thin-tac x = l-comb R M n s f)
apply (*simp add*:*l-comb-def*)
apply (*rule-tac n = n* **in** *nsum-mem*)
apply (*rule allI*, *rule impI*)
apply (*rule sc-mem*)
apply (*simp add*:*Pi-def Ring.ideal-subset*)
apply (*simp add*:*Pi-def subsetD*)
done

lemma (**in** *Module*) *lin-span-coeff-mono*:⟦*ideal R A*; *H* ⊆ *carrier M*⟧⟹
linear-span R M A H ⊆ *linear-span R M* (*carrier R*) *H*
apply (*cut-tac sc-Ring*)
apply (*rule subsetI*)
apply (*simp add*:*linear-span-def*)
apply (*case-tac H* = {}) **apply** *simp* **apply** *simp*
apply (*erule exE*, (*erule bexE*)+)
apply (*frule Ring.ideal-subset1* [*of R A*], *assumption*+)
apply (*frule-tac f* = *s* **in** *extend-fun*, *assumption*+)
apply *blast*
done

lemma (**in** *Module*) *l-span-sum-closedTr*:⟦*ideal R A*; *H* ⊆ *carrier M*⟧⟹
∀ *s*. ∀ *f*. *s*∈{*j*. *j* ≤ (*n*::*nat*)} → *A* ∧
f ∈ {*j*. *j* ≤ *n*} → *linear-span R M A H* ⟶
(*nsum M* (λ*j*. *s j* ·$_s$ (*f j*)) *n* ∈ *linear-span R M A H*)
apply (*cut-tac sc-Ring*)
apply (*induct-tac n*)
apply ((*rule allI*)+, *rule impI*, *simp*)
apply (*erule conjE*)
apply (*rule linear-span-sc-closed*, *assumption*+)
apply (*simp add*:*Pi-def Ring.ideal-subset*,
simp add:*Pi-def*)

apply ((*rule allI*)+, *rule impI*, *erule conjE*)
apply (*frule func-pre* [*of* - - *A*],
frule func-pre [*of* - - *linear-span R M A H*])
apply (*drule-tac x* = *s* **in** *spec*,
drule-tac x = *f* **in** *spec*)

apply *simp*
apply (*rule linear-span-pOp-closed*, *assumption*+)
apply (*rule linear-span-sc-closed*, *assumption*+,
simp add:*Pi-def Ring.ideal-subset*,
simp add:*Pi-def subsetD*)
done

lemma (**in** *Module*) *l-span-closed*:⟦*ideal R A*; *H* ⊆ *carrier M*;
s ∈ {*j*. *j* ≤ (*n*::*nat*)} → *A*; *f* ∈ {*j*. *j* ≤ *n*} → *linear-span R M A H* ⟧ ⟹
l-comb R M n s f ∈ *linear-span R M A H*
apply (*simp add*:*l-comb-def*)
apply (*simp add*: *l-span-sum-closedTr*)
done

lemma (**in** *Module*) *l-span-closed1*:⟦*H* ⊆ *carrier M*;
s ∈ {*j*. *j* ≤ (*n*::*nat*)} → *carrier R*;
f ∈ {*j*. *j* ≤ *n*} → *linear-span R M* (*carrier R*) *H* ⟧ ⟹
Σ$_e$ *M* (λ*j*. *s j* ·$_s$ (*f j*)) *n* ∈ *linear-span R M* (*carrier R*) *H*

apply (*cut-tac sc-Ring*,
frule Ring.whole-ideal [*of R*])
apply (*frule l-span-sum-closedTr*[*of carrier R H n*], *assumption*+)
apply (*drule-tac x* = *s* **in** *spec*,
drule-tac x = *f* **in** *spec*,
simp)
done

lemma (**in** *Module*) *l-span-closed2Tr0*:⟦*ideal R A*; *H* ⊆ *carrier M*; *Ring R*; *s* ∈ *A*;
f ∈ *linear-span R M* (*carrier R*) *H* ⟧ ⟹ *s* ·$_s$ *f* ∈ *linear-span R M A H*
apply (*cut-tac sc-Ring*)
apply (*case-tac H* = {})
apply (*simp add*:*linear-span-def*)
apply (*rule sc-a-0*,
cut-tac sc-Ring,
simp add:*Pi-def Ring.ideal-subset*)

apply (*simp add*:*linear-span-def*)
apply (*erule exE*, (*erule bexE*)+, *simp*,
thin-tac f = *l-comb R M n sa fa*)
apply (*frule Ring.whole-ideal*[*of R*])
apply (*frule-tac h* = *s* **in** *Ring.ideal-subset*[*of R A*], *assumption*+)
apply (*frule-tac s* = *sa* **and** *g* = *f* **in** *l-comb-sc1*[*of carrier R H s*],
assumption+, *simp add*:*l-comb-def*,
thin-tac s ·$_s$ Σ$_e$ *M* (λ*k*. *sa k* ·$_s$ *f k*) *n* =
Σ$_e$ *M* (λ*k*. (*s* ·$_{r\,R}$ *sa k*) ·$_s$ *f k*) *n*)
apply (*cut-tac n* = *n* **and** *f* = λ*j*. (*s* ·$_{r\,R}$ *sa j*) ·$_s$ *f j* **and**
g = λ*j*. ((λ*x*∈{*j*. *j* ≤ *n*}. (*s* ·$_{r\,R}$ *sa x*)) *j*) ·$_s$ *f j* **in** *nsum-eq*)
apply (*rule allI*, *rule impI*, *rule sc-mem*,
rule Ring.ring-tOp-closed, *assumption*+,
simp add:*Pi-def*,
simp add:*Pi-def subsetD*)
apply (*rule allI*, *rule impI*, *simp*,
rule sc-mem,
rule Ring.ring-tOp-closed, *assumption*+,
simp add:*Pi-def*,
simp add:*Pi-def subsetD*)
apply (*rule allI*, *rule impI*, *simp*)
apply (*subgoal-tac* (λ*x*∈{*j*. *j* ≤ *n*}. (*s* ·$_{r\,R}$ *sa x*)) ∈ {*j*. *j* ≤ *n*} → *A*,
blast,
thin-tac Σ$_e$ *M* (λ*j*. (*s* ·$_{r\,R}$ *sa j*) ·$_s$ *f j*) *n* =
Σ$_e$ *M* (λ*j*. (λ*x*∈{*j*. *j* ≤ *n*}. *s* ·$_{r\,R}$ *sa x*) *j* ·$_s$ *f j*) *n*)
apply (*rule Pi-I*, *simp*,
rule-tac x = *s* **and** *r* = *sa x* **in**
Ring.ideal-ring-multiple1[*of R A*], *assumption*+)
apply (*simp add*:*Pi-def*)
done

lemma (**in** *Module*) *l-span-closed2Tr*:⟦*ideal R A*; *H* ⊆ *carrier M*⟧ ⟹

```
      s ∈ {j. j ≤ (n::nat)} → A ∧
      f ∈ {j. j ≤ n} → linear-span R M (carrier R) H ⟶
          l-comb R M n s f ∈ linear-span R M A H
apply (cut-tac sc-Ring)
apply (induct-tac n)
apply (rule impI, (erule conjE)+)
apply (case-tac H = {})
 apply (simp add:linear-span-def)
 apply (simp add:l-comb-def)
 apply (rule sc-a-0,
        cut-tac sc-Ring,
        simp add:Pi-def Ring.ideal-subset)
 apply (simp add:l-comb-def l-span-closed2Tr0)

apply (rule impI, erule conjE,
       frule func-pre[of s], frule func-pre[of f], simp)
 apply (simp add:l-comb-def)
 apply (rule linear-span-pOp-closed, assumption+)
 apply (rule-tac s = s (Suc n) and f = f (Suc n) in
                 l-span-closed2Tr0[of A H], assumption+,
       (simp add:Pi-def)+)
done

lemma (in Module) l-span-closed2:⟦ideal R A; H ⊆ carrier M;
      s ∈ {j. j ≤ (n::nat)} → A ;
      f ∈ {j. j ≤ n} → linear-span R M (carrier R) H⟧ ⟹
      l-comb R M n s f ∈ linear-span R M A H
apply (simp add:l-span-closed2Tr)
done

lemma (in Module) l-span-l-span:H ⊆ carrier M ⟹
      linear-span R M (carrier R) (linear-span R M (carrier R) H) =
                                    linear-span R M (carrier R) H
apply (cut-tac sc-Ring, frule Ring.whole-ideal[of R])
apply (rule equalityI)
 apply (rule subsetI)
 apply (frule linear-span-inc-0[of carrier R H], assumption+,
        frule nonempty[of - linear-span R M (carrier R) H],
        simp add:linear-span-def[of R M carrier R
                          linear-span R M (carrier R) H],
        erule exE, (erule bexE)+, simp)
 apply (frule-tac s = s and n = n and f = f in l-span-closed2[of carrier R],
        assumption+,
        frule lin-span-sub-carrier[of carrier R H], assumption+,
        rule subsetI)
 apply (rule-tac h = x in h-in-linear-span[of linear-span R M (carrier R) H],
        assumption+)
done
```

lemma (**in** *Module*) *l-spanA-l-span*:⟦*ideal R A*; *H* ⊆ *carrier M*⟧ ⟹
linear-span R M A (*linear-span R M* (*carrier R*) *H*) =
linear-span R M A H
apply (*cut-tac sc-Ring*, *frule Ring.whole-ideal*[*of R*])
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*frule linear-span-inc-0*[*of carrier R H*], *assumption*+,
frule nonempty[*of - linear-span R M* (*carrier R*) *H*],
simp add:*linear-span-def*[*of R M A*
linear-span R M (*carrier R*) *H*],
erule exE, (*erule bexE*)+, *simp*)
apply (*frule-tac s* = *s* **and** *n* = *n* **and** *f* = *f* **in** *l-span-closed2*[*of A*],
assumption+)
apply (*frule l-span-cont-H*[*of H*])
apply (*frule l-span-gen-mono*[*of H linear-span R M* (*carrier R*) *H A*],
simp add:*lin-span-sub-carrier*[*of carrier R H*], *assumption*)
apply *assumption*
done

lemma (**in** *Module*) *l-span-zero*:*ideal R A* ⟹ *linear-span R M A* {**0**} = {**0**}
apply (*cut-tac sc-Ring*)
apply (*rule equalityI*)
apply (*rule subsetI*,
frule-tac x = *x* **in** *mem-single-l-span1*[*of A* **0**],
simp add:*ag-inc-zero*, *assumption*,
erule bexE, *frule-tac h* = *a* **in** *Ring.ideal-subset*[*of R A*], *assumption*+,
simp add:*sc-a-0*)
apply (*rule subsetI*, *simp*, *rule linear-span-inc-0*, *assumption*,
rule subsetI, *simp add*:*ag-inc-zero*)
done

lemma (**in** *Module*) *l-span-closed3*:⟦*ideal R A*; *generator R M H*;
A ⊙$_R$ *M* = *carrier M*⟧ ⟹ *linear-span R M A H* = *carrier M*
apply (*cut-tac sc-Ring*)

apply (*rule equalityI*)
apply (*cut-tac linear-span-subModule*[*of A H*],
simp add:*submodule-subset*, *assumption*,
simp add:*generator-def*)

apply (*rule subsetI*)
apply (*simp add*:*generator-def*)
apply (*erule conjE*)
apply (*case-tac H* = {}, *simp*, *simp add*:*linear-span-def*)
apply (*simp add*:*smodule-ideal-coeff-def*)
apply (*rotate-tac* −*2*, *frule sym*,
thin-tac linear-span R M (*carrier R*) *H* = *carrier M*)
apply *simp*
apply (*frule sym*,

thin-tac linear-span R M A (linear-span R M (carrier R) H) =
linear-span R M (carrier R) H)
apply (*frule-tac a = x* **in** *eq-set-inc*[*of - linear-span R M (carrier R) H*
linear-span R M A (linear-span R M (carrier R) H)], *assumption*+,
thin-tac x ∈ linear-span R M (carrier R) H,
thin-tac linear-span R M (carrier R) H =
linear-span R M A (linear-span R M (carrier R) H))
apply (*frule sym*,
thin-tac carrier M = linear-span R M (carrier R) H,
frule subset-trans[*of H linear-span R M (carrier R) H carrier M*],
simp,
thin-tac linear-span R M (carrier R) H = carrier M)
apply (*frule Ring.whole-ideal*,
frule linear-span-inc-0 [*of carrier R H*], *assumption*+,
frule nonempty [*of* **0** *linear-span R M (carrier R) H*])
apply (*simp add:linear-span-def* [*of - - - linear-span R M (carrier R) H*])
apply (*erule exE*, (*erule bexE*)+)
apply (*simp add:l-span-closed2*)
done

lemma (**in** *Module*) *generator-generator*:⟦*generator R M H*; *H1 ⊆ carrier M*;
H ⊆ linear-span R M (carrier R) H1⟧ ⟹ *generator R M H1*
apply (*cut-tac sc-Ring*,
frule Ring.whole-ideal[*of R*],
frule linear-span-subModule[*of carrier R H1*], *assumption*,
frule l-span-sub-submodule[*of carrier R*
linear-span R M (carrier R) H1 H], *assumption*+)
apply (*simp add:generator-def*)
apply (*rule equalityI*,
simp add:submodule-subset, *assumption*)
done

lemma (**in** *Module*) *generator-elimTr*:
f ∈ {j. j ≤ (n::nat)} → carrier M ∧ generator R M (f ‘ {j. j ≤ n}) ∧
(∀ i∈nset (Suc 0) n. f i ∈
linear-span R M (carrier R) (f ‘ {j. j ≤ (i − Suc 0)})) ⟶
linear-span R M (carrier R) {f 0} = carrier M
apply (*induct-tac n*)
apply (*rule impI*, (*erule conjE*)+)
apply (*simp add:nset-def generator-def*)

apply (*rule impI*)
apply (*erule conjE*)+
apply (*frule func-pre* [*of - - carrier M*], *simp*)
apply (*subgoal-tac generator R M (f ‘ {j. j ≤ n})*)
apply (*subgoal-tac ∀ i∈nset (Suc 0) n.*
f i ∈ linear-span R M (carrier R) (f ‘ {j. j ≤ (i − Suc 0)}))
apply *simp*
apply (*thin-tac generator R M (f ‘ {j. j ≤ n}) ∧*

```
    (∀ i∈nset (Suc 0) n. f i ∈ linear-span R M (carrier R)
          (f ‘ {j. j ≤ i − Suc 0})) ⟶
       linear-span R M (carrier R) {f 0} = carrier M)
 apply (rule ballI)
 apply (frule-tac x = i in bspec, simp add:nset-def, assumption)
 apply (thin-tac generator R M (f ‘ {j. j ≤ n}) ∧
       (∀ i∈nset (Suc 0) n.
       f i ∈ linear-span R M (carrier R) (f ‘ {j. j ≤ i − Suc 0})) ⟶
       linear-span R M (carrier R) {f 0} = carrier M)
 apply (frule-tac x = Suc n in bspec, simp add:nset-def,
       thin-tac ∀ i∈nset (Suc 0) (Suc n).
          f i ∈ linear-span R M (carrier R) (f ‘ {j. j ≤ i − Suc 0}),
       simp)
 apply (subgoal-tac f ‘ {j. j ≤ Suc n} ⊆ linear-span R M (carrier R) (f ‘ {j. j ≤
n}))
 apply (frule-tac H = f ‘ {j. j ≤ Suc n} and ?H1.0 = f ‘ {j. j ≤ n}
       in generator-generator,
       rule subsetI, simp add:image-def, erule exE, erule conjE, simp,
       simp add:Pi-def)
 apply assumption+
 apply (rule subsetI, simp add:image-def, erule exE, erule conjE)
 apply (case-tac xa = Suc n, simp)
 apply (frule-tac m = xa and n = Suc n in noteq-le-less, assumption,
       thin-tac xa ≤ Suc n,
       frule-tac x = xa and n = Suc n in less-le-diff,
       thin-tac xa < Suc n, simp)
 apply (rule-tac H = {y. ∃ x≤n. y = f x} and h = f xa in
                     h-in-linear-span,
       rule subsetI, simp add:image-def, erule exE, erule conjE,
       simp add:Pi-def)
 apply (simp, blast)
done

lemma (in Module) generator-generator-elim:
 ⟦f ∈ {j. j ≤ (n::nat)} → carrier M; generator R M (f ‘ {j. j ≤ n});
  (∀ i∈nset (Suc 0) n. f i ∈ linear-span R M (carrier R)
    (f ‘ {j. j ≤ (i − Suc 0)}))⟧ ⟹
   linear-span R M (carrier R) {f 0} = carrier M
apply (simp add:generator-elimTr [of f n])
done

lemma (in Module) surjec-generator:⟦R module N; f ∈ mHom R M N;
 surjec_{M,N} f; generator R M H⟧ ⟹ generator R N (f ‘ H)
apply (cut-tac sc-Ring, frule Ring.whole-ideal)
apply (simp add:generator-def, erule conjE)
 apply (simp add:surjec-def, (erule conjE)+)
 apply (simp add:aHom-def, (erule conjE)+)
 apply (simp add:image-sub [of f carrier M carrier N H])
```

apply (*frule Module.lin-span-sub-carrier*[*of N R carrier R f ‘ H*],
assumption,
simp add:*image-sub* [*of f carrier M carrier N H*])
apply (*rule equalityI*, *assumption*+)
apply (*rule subsetI*)
apply (*simp add*:*surj-to-def*,
thin-tac f ∈ *extensional* (*carrier M*),
thin-tac ∀ *a*∈*carrier M*. ∀ *b*∈*carrier M*. *f* (*a* ± *b*) = *f a* ±$_N$ *f b*)
apply (*frule sym*, *rotate-tac 6*, *frule sym*,
thin-tac f ‘ carrier M = *carrier N*,
frule-tac a = *x* **and** *A* = *carrier N* **and** *B* = *f ‘ carrier M* **in**
eq-set-inc, *assumption*,
thin-tac carrier N = *f ‘ carrier M*,
thin-tac carrier M = *linear-span R M* (*carrier R*) *H*)
apply (*simp add*:*image-def*[*of f carrier M*], *erule bexE*)
apply (*frule sym*, *thin-tac linear-span R M* (*carrier R*) *H* = *carrier M*,
frule-tac a = *xa* **in** *eq-set-inc*[*of* - *carrier M*
linear-span R M (*carrier R*) *H*], *assumption*,
thin-tac carrier M = *linear-span R M* (*carrier R*) *H*,
thin-tac linear-span R N (*carrier R*) (*f ‘ H*) ⊆ *carrier N*)

apply (*simp add*:*linear-span-def*)
apply (*case-tac H* = {}, *simp*)
apply (*simp add*:*mHom-0*, *simp*,
erule exE, (*erule bexE*)+)
apply (*cut-tac sc-Ring*, *frule Ring.whole-ideal*[*of R*],
frule-tac s = *s* **and** *n* = *n* **and** *g* = *fa* **in**
linmap-im-linspan[*of carrier R N f H*], *assumption*+,
rotate-tac −5, *frule sym*,
thin-tac xa = *l-comb R M n s fa*, *simp*,
thin-tac l-comb R M n s fa = *xa*)
apply (*simp add*:*linear-span-def*)
done

lemma (**in** *Module*) *surjec-finitely-gen*:⟦*R module N*; *f* ∈ *mHom R M N*;
surjec$_{M,N}$ *f*; *M fgover R*⟧ ⟹ *N fgover R*
apply (*simp add*:*fGOver-def*)
apply (*erule exE*)
apply (*simp add*:*finite-generator-def* [*of R M*],*erule conjE*)
apply (*frule-tac H* = *H* **in** *surjec-generator*[*of N f*], *assumption*+)
apply (*simp add*:*finite-generator-def* [*of R N*])
apply (*frule-tac F* = *H* **and** *h* = *f* **in** *finite-imageI*)
apply *blast*
done

5.3.1 Sum up coefficients

Symbolic calculation.

lemma (**in** *Module*) *similar-termTr*:⟦*ideal R A*; *a* ∈ *A*⟧ ⟹

$\forall s.\ \forall f.\ s \in \{j.\ j \leq (n::nat)\} \to A \wedge$
$\quad f \in \{j.\ j \leq n\} \to carrier\ M \wedge$
$\quad m \in f\ `\ \{j.\ j \leq n\} \longrightarrow$
$\quad (\exists t{\in}\{j.\ j \leq n\} \to A.\ nsum\ M\ (\lambda j.\ s\ j\ \cdot_s\ (f\ j))\ n \pm a\ \cdot_s\ m =$
$\quad\quad nsum\ M\ (\lambda j.\ t\ j\ \cdot_s\ (f\ j))\ n\ )$
apply (*cut-tac sc-Ring*)
apply (*induct-tac n*)
apply (*rule allI*)+ **apply** (*rule impI*) **apply** (*erule conjE*)+
apply *simp*
apply (*rule-tac x* = $\lambda k{\in}\{0::nat\}.\ (s\ 0\ \pm_R\ a)$ **in** *bexI*)
apply (*simp add*: *Ring.ideal-subset sc-l-distr*)
apply *simp*
apply (*simp add*:*Ring.ideal-pOp-closed*)

apply ((*rule allI*)+, *rule impI*, (*erule conjE*)+)
apply (*simp del*:*nsum-suc add*:*image-def*)
apply (*cut-tac n* = *n* **and** *f* = $\lambda j.\ s\ j\ \cdot_s\ f\ j$ **in** *nsum-mem*,
rule allI, *rule impI*, *rule sc-mem*,
simp add:*funcset-mem Ring.ideal-subset*,
simp add:*funcset-mem*,
frule-tac x = *Suc n* **and** *f* = *s* **and** *A* = $\{j.\ j \leq Suc\ n\}$ **and**
B = *A* **in** *funcset-mem*, *simp*,
frule-tac h = *s* (*Suc n*) **in** *Ring.ideal-subset*, *assumption*+,
frule-tac x = *Suc n* **and** *f* = *f* **and** *A* = $\{j.\ j \leq Suc\ n\}$ **and**
B = *carrier M* **in** *funcset-mem*, *simp*,
frule-tac a = *s* (*Suc n*) **and** *m* = *f* (*Suc n*) **in** *sc-mem*, *assumption*+,
cut-tac a = *a* **and** *m* = *m* **in** *sc-mem*,
simp add:*Ring.ideal-subset*, *erule exE*, *simp add*:*Pi-def*,
erule exE, *erule conjE*)
apply (*case-tac x* = *Suc n*, *simp*)
apply (*subst ag-pOp-assoc*, *assumption*+)
apply (*thin-tac* $\Sigma_e\ M\ (\lambda j.\ s\ j\ \cdot_s\ f\ j)\ n \in carrier\ M$,
thin-tac s (*Suc n*) $\cdot_s$ *f* (*Suc n*) $\in$ *carrier M*,
thin-tac a $\cdot_s$ *f* (*Suc n*) $\in$ *carrier M*,
thin-tac $\forall s\ fa.$
$s \in \{j.\ j \leq n\} \to A \wedge$
$fa \in \{j.\ j \leq n\} \to carrier\ M \wedge (\exists x{\leq}n.\ f\ (Suc\ n) = fa\ x) \longrightarrow$
$(\exists t{\in}\{j.\ j \leq n\} \to A.$
$\Sigma_e\ M\ (\lambda j.\ s\ j\ \cdot_s\ fa\ j)\ n \pm a\ \cdot_s\ f\ (Suc\ n) =$
$\Sigma_e\ M\ (\lambda j.\ t\ j\ \cdot_s\ fa\ j)\ n))$
apply (*subst sc-l-distr*[*THEN sym*], *assumption*+,
simp add:*Ring.ideal-subset*, *assumption*+)
apply (*frule func-pre*[*of* - - *A*],
frule-tac f = *s* **and** *n* = *n* **and** *g* = $\lambda k{\in}\{0::nat\}.\ (s\ (Suc\ n) \pm_R\ a)$ **and**
m = *0* **and** *A* = *A* **and** *B* = *A* **in** *jointfun-hom0*,
simp add: *Ring.ideal-pOp-closed*)
apply (*subgoal-tac* $\Sigma_e\ M\ (\lambda j.\ s\ j\ \cdot_s\ f\ j)\ n \pm (s\ (Suc\ n) \pm_R\ a)\ \cdot_s\ f\ (Suc\ n) =$
$\Sigma_e\ M\ (\lambda j.\ (jointfun\ n\ s\ 0\ (\lambda k{\in}\{0\}.\ s\ (Suc\ n) \pm_R\ a))\ j\ \cdot_s\ f\ j)\ (Suc\ n)$,

simp,
thin-tac $\Sigma_e\ M\ (\lambda j.\ s\ j\ \cdot_s\ f\ j)\ n \pm (s\ (Suc\ n) \pm_R\ a)\ \cdot_s\ f\ (Suc\ n) =$
$\Sigma_e\ M\ (\lambda j.\ jointfun\ n\ s\ 0\ (\lambda k{\in}\{0\}.\ s\ (Suc\ n) \pm_R\ a)\ j\ \cdot_s\ f\ j)\ n \pm$
jointfun n s 0 $(\lambda k{\in}\{0\}.\ s\ (Suc\ n) \pm_R\ a)\ (Suc\ n)\ \cdot_s\ f\ (Suc\ n))$
apply *blast*
apply *simp*
apply (*simp add*:*jointfun-def sliden-def*)
apply (*cut-tac* $n = n$ **and** $f = \lambda j.\ s\ j\ \cdot_s\ f\ j$ **and** $g = \lambda j.$ (*if* $j \leq n$ *then s j*
else $(\lambda k{\in}\{0\}.\ s\ (Suc\ n) \pm_R\ a)$ (*sliden* (*Suc n*) *j*)) $\cdot_s\ f\ j$ **in**
nsum-eq)
apply (*rule allI*, *rule impI*, *rule sc-mem*,
simp add:*Pi-def Ring.ideal-subset*,
simp add:*Pi-def*)
apply (*rule allI*, *rule impI*, *simp*, *rule sc-mem*,
simp add:*Pi-def Ring.ideal-subset*,
simp add:*Pi-def*)
apply (*rule allI*, *rule impI*, *simp*)
apply *simp*

apply (*frule-tac* $m = x$ **and** $n = Suc\ n$ **in** *noteq-le-less*, *assumption*,
thin-tac $x \leq Suc\ n$,
frule-tac $x = x$ **and** $n = Suc\ n$ **in** *less-le-diff*,
thin-tac $x < Suc\ n$, *simp*)
apply (*frule func-pre*[*of - - A*], *frule func-pre*[*of - - carrier M*])
apply (*drule-tac* $x = s$ **in** *spec*,
drule-tac $x = f$ **in** *spec*)
apply (*subgoal-tac* $\exists xa{\leq}n.\ f\ x = f\ xa$, *simp*,
thin-tac $\exists xa{\leq}n.\ f\ x = f\ xa$, *erule bexE*)
apply (*subst ag-pOp-assoc*, *assumption*+,
frule-tac $x = s\ (Suc\ n)\ \cdot_s\ f\ (Suc\ n)$ **and** $y = a\ \cdot_s\ f\ x$ **in**
ag-pOp-commute, *assumption*+, *simp*,
thin-tac $s\ (Suc\ n)\ \cdot_s\ f\ (Suc\ n) \pm a\ \cdot_s\ f\ x =$
$a\ \cdot_s\ f\ x \pm s\ (Suc\ n)\ \cdot_s\ f\ (Suc\ n)$,
subst ag-pOp-assoc[*THEN sym*], *assumption*+, *simp*,
thin-tac $\Sigma_e\ M\ (\lambda j.\ s\ j\ \cdot_s\ f\ j)\ n \pm a\ \cdot_s\ f\ x = \Sigma_e\ M\ (\lambda j.\ t\ j\ \cdot_s\ f\ j)\ n$)
apply (*frule-tac* $f = t$ **and** $n = n$ **and** $g = \lambda k{\in}\{0{::}nat\}.\ s\ (Suc\ n)$ **and**
$m = 0$ **and** $A = A$ **and** $B = A$ **in** *jointfun-hom0*,
simp, *simp*)
apply (*subgoal-tac* $\Sigma_e\ M\ (\lambda j.\ t\ j\ \cdot_s\ f\ j)\ n \pm s\ (Suc\ n)\ \cdot_s\ f\ (Suc\ n) =$
$\Sigma_e\ M\ (\lambda j.$ (*jointfun n t 0* $(\lambda k{\in}\{0\}.\ s\ (Suc\ n)))\ j\ \cdot_s\ f\ j)\ (Suc\ n)$,
simp,
thin-tac $\Sigma_e\ M\ (\lambda j.\ t\ j\ \cdot_s\ f\ j)\ n \pm s\ (Suc\ n)\ \cdot_s\ f\ (Suc\ n) =$
$\Sigma_e\ M\ (\lambda j.\ jointfun\ n\ t\ 0\ (\lambda k{\in}\{0\}.\ s\ (Suc\ n))\ j\ \cdot_s\ f\ j)\ n \pm$
jointfun n t 0 $(\lambda k{\in}\{0\}.\ s\ (Suc\ n))\ (Suc\ n)\ \cdot_s\ f\ (Suc\ n))$
apply *blast*
apply (*simp add*:*jointfun-def sliden-def*)
apply (*cut-tac* $n = n$ **and** $f = \lambda j.\ t\ j\ \cdot_s\ f\ j$ **and**
$g = \lambda j.$ (*if* $j \leq n$ *then t j else* $(\lambda k{\in}\{0\}.\ s\ (Suc\ n))$
(*sliden* (*Suc n*) *j*)) $\cdot_s\ f\ j$ **in** *nsum-eq*)

apply (*rule allI*, *rule impI*, *rule sc-mem*,
simp add:*Pi-def Ring.ideal-subset*,
simp add:*Pi-def*)
apply (*rule allI*, *rule impI*, *simp*, *rule sc-mem*,
simp add:*Pi-def Ring.ideal-subset*,
simp add:*Pi-def*)
apply (*rule allI*, *rule impI*, *simp*, *simp*)
apply *blast*
done

lemma (**in** *Module*) *similar-term1*:⟦*ideal R A*; $a \in A$; $s \in \{j.\ j \leq (n::nat)\} \to A$;
$f \in \{j.\ j \leq n\} \to$ *carrier M*; $m \in f\ `\ \{j.\ j \leq n\}$⟧ ⟹
$\exists t \in \{j.\ j \leq n\} \to A.\ \Sigma_e\ M\ (\lambda j.\ s\ j \cdot_s (f\ j))\ n \pm a \cdot_s m =$
$\Sigma_e\ M\ (\lambda j.\ \ t\ j \cdot_s (f\ j))\ n$
apply (*simp add*:*similar-termTr*)
done

lemma (**in** *Module*) *same-togetherTr*:⟦*ideal R A*; $H \subseteq$ *carrier M* ⟧ ⟹
$\forall s.\ \forall f.\ s \in \{j.\ j \leq (n::nat)\} \to A\ \ \wedge f \in \{j.\ j \leq n\} \to H \longrightarrow$
$(\exists t \in \{j.\ j \leq (card\ (f\ `\ \{j.\ j \leq n\}) - Suc\ 0)\} \to A.$
$\exists g \in \{j.\ j \leq (card\ (f\ `\ \{j.\ j \leq n\}) - Suc\ 0)\} \to f\ `\ \{j.\ j \leq n\}.$
surj-to g $\{j.\ j \leq (card\ (f\ `\ \{j.\ j \leq n\}) - Suc\ 0)\}\ (f\ `\ \{j.\ j \leq n\}) \wedge$
$nsum\ M\ (\lambda j.\ s\ j \cdot_s (f\ j))\ n = nsum\ M\ (\lambda k.\ t\ k \cdot_s (g\ k))$
$(card\ (f\ `\ \{j.\ j \leq n\}) - Suc\ 0))$
apply (*induct-tac n*)
apply ((*rule allI*)+, *rule impI*, *erule conjE*)
apply (*simp del*: *Pi-split-insert-domain*)
apply (*frule-tac f* = *f* **and** *A* = $\{0\}$ **and** *B*= *H* **in** *func-to-img*,
frule-tac f = *f* **and** *A* = $\{0\}$ **and** *B*= *H* **in** *surj-to-image*,
fastforce simp add:*image-def*)

apply ((*rule allI*)+, *rule impI*, *erule conjE*)
apply (*frule func-pre* [*of* - - *A*], *frule func-pre* [*of* - - *H*])
apply (*drule-tac x* = *s* **in** *spec*,
drule-tac x = *f* **in** *spec*,
simp, (*erule bexE*)+ , (*erule conjE*)+, *simp*,
thin-tac $\Sigma_e\ M\ (\lambda j.\ s\ j \cdot_s f\ j)\ n =$
$\Sigma_e\ M\ (\lambda k.\ t\ k \cdot_s g\ k)\ (card\ (f\ `\ \{j.\ j \leq n\}) - Suc\ 0))$

apply (*case-tac f* $(Suc\ n) \in f\ `\ \{j.\ j \leq n\}$)
apply (*frule-tac a* = *s* $(Suc\ n)$ **and** *s* = *t* **and**
n = *card* $(f\ `\ \{j.\ j \leq n\}) - Suc\ 0$ **and** *f* = *g* **and** *m* = *f* $(Suc\ n)$ **in**
similar-term1[*of A*],
simp add:*Pi-def*,
assumption,
frule-tac f = *f* **and** *A* = $\{j.\ j \leq n\}$ **and** *B* = *H* **in** *image-sub0*,
frule-tac A = *f* ‘ $\{j.\ j \leq n\}$ **and** *B* = *H* **and** *C* = *carrier M*
in *subset-trans*, *assumption*,

rule-tac $f = g$ **and** $A = \{j.\ j \leq card\ (f\ `\ \{j.\ j \leq n\}) - Suc\ 0\}$ **and**
$B = f\ `\ \{j.\ j \leq n\}$ **and** *?B1.0* $= carrier\ M$ **in** *extend-fun*,
assumption+)
apply (*simp add*:*surj-to-def*)
apply (*erule bexE*, *simp*,
thin-tac $\Sigma_e\ M\ (\lambda j.\ t\ j \cdot_s g\ j)\ (card\ (f\ `\ \{j.\ j \leq n\}) - Suc\ 0) \pm$
$s\ (Suc\ n) \cdot_s f\ (Suc\ n) =$
$\Sigma_e\ M\ (\lambda j.\ ta\ j \cdot_s g\ j)\ (card\ (f\ `\ \{j.\ j \leq n\}) - Suc\ 0)$)
apply (*simp add*:*Nset-img0*)
apply *blast*

apply (*frule-tac* $f = t$ **and** $n = card\ (f\ `\ \{j.\ j \leq n\}) - Suc\ 0$ **and** $A = A$ **and**
$g = \lambda k{\in}\{0{::}nat\}.\ s\ (Suc\ n)$ **and** $m = 0$ **and** $B = A$ **in** *jointfun-hom0*)
apply (*simp add*:*Pi-def*,
simp)
apply (*frule-tac* $f = g$ **and** $n = card\ (f\ `\ \{j.\ j \leq n\}) - Suc\ 0$ **and**
$A = f\ `\ \{j.\ j \leq n\}$ **and** $g = \lambda k{\in}\{0{::}nat\}.\ f\ (Suc\ n)$ **and** $m = 0$ **and**
$B = \{f\ (Suc\ n)\}$ **in** *jointfun-hom0*)
apply (*simp add*:*Pi-def*,
simp)
apply (*subgoal-tac* $\Sigma_e\ M\ (\lambda k.\ t\ k \cdot_s g\ k)\ (card\ (f\ `\ \{j.\ j \leq n\}) - Suc\ 0) \pm$
$s\ (Suc\ n) \cdot_s f\ (Suc\ n) =$
$\Sigma_e\ M\ (\lambda j.\ (jointfun\ (card\ (f\ `\ \{j.\ j \leq n\}) - Suc\ 0)\ t\ 0\ (\lambda k{\in}\{0\}.$
$s\ (Suc\ n)))\ j \cdot_s (jointfun\ (card\ (f\ `\ \{j.\ j \leq n\}) - Suc\ 0)\ g\ 0$
$(\lambda k{\in}\{0\}.\ f\ (Suc\ n)))\ j)\ (card\ (f\ `\ \{j.\ j \leq (Suc\ n)\}) - Suc\ 0)$, *simp*,
thin-tac $\Sigma_e\ M\ (\lambda k.\ t\ k \cdot_s g\ k)\ (card\ (f\ `\ \{j.\ j \leq n\}) - Suc\ 0) \pm$
$s\ (Suc\ n) \cdot_s f\ (Suc\ n) =$
$\Sigma_e\ M\ (\lambda j.\ jointfun\ (card\ (f\ `\ \{j.\ j \leq n\}) - Suc\ 0)\ t\ 0$
$(\lambda k{\in}\{0\}.\ s\ (Suc\ n))\ j \cdot_s$
$jointfun\ (card\ (f\ `\ \{j.\ j \leq n\}) - Suc\ 0)\ g\ 0$
$(\lambda k{\in}\{0\}.\ f\ (Suc\ n))$
$j)\ (card\ (f\ `\ \{j.\ j \leq Suc\ n\}) - Suc\ 0)$)
apply (*simp del*:*nsum-suc add*:*card-image-Nsetn-Suc*)
apply (*simp del*:*nsum-suc add*:*image-Nset-Suc*[*THEN sym*])
apply (*subgoal-tac surj-to* (*jointfun* (*card* $(f\ `\ \{j.\ j \leq n\}) - Suc\ 0)\ g\ 0$
$(\lambda k{\in}\{0\}.\ f\ (Suc\ n)))\ \{l.\ l \leq Suc\ (card\ (f\ `\ \{j.\ j \leq n\}) - Suc\ 0)\}$
$(f\ `\ \{j.\ j \leq Suc\ n\})$, *blast*)

apply (*simp add*:*surj-to-def*)
apply (*frule-tac* $f = g$ **and** $n = card\ (f\ `\ \{j.\ j \leq n\}) - Suc\ 0$ **and** $A = f\ `\ \{j.$
$j \leq n\}$ **and** $g = \lambda k{\in}\{0\}.\ f\ (Suc\ n)$ **and** $m = 0$ **and** $B = \{f\ (Suc\ n)\}$ **in**
im-jointfun)
apply (*simp add*:*Pi-def*)
apply *simp*
apply (*simp add*:*image-Nset-Suc*[*THEN sym*])
apply (*simp add*:*card-image-Nsetn-Suc*)
apply (*simp add*:*Nset-img*)

apply (*frule-tac* $f = f$ **and** $A = \{j.\ j \leq Suc\ n\}$ **and** $B = H$ **in** *image-sub0*)

apply (*frule-tac* $A = f\ `\ \{j.\ j \leq Suc\ n\}$ **and** $B = H$ **and** $C = carrier\ M$ **in**
subset-trans, *assumption*+)
apply (*cut-tac* $H = H$ **and** $s = t$ **and** $n = card\ (f\ `\ \{j.\ j \leq n\}) - Suc\ 0$
and $f = g$ **and** $t = \lambda k{\in}\{0\}.\ s\ (Suc\ n)$ **and** $m = 0$ **and**
$g = \lambda k{\in}\{0\}.\ f\ (Suc\ n)$ **in**
l-comb-jointfun-jj[*of* - *A*], *assumption*+)
apply (*frule-tac* $f = f$ **and** $A = \{j.\ j \leq n\}$ **and** $B = H$ **in** *image-sub0*)
apply (*rule-tac* $f = g$ **and** $A = \{j.\ j \leq card\ (f\ `\ \{j.\ j \leq n\}) - Suc\ 0\}$ **and**
$B = f\ `\ \{j.\ j \leq n\}$ **in** *extend-fun*[*of* - - - *H*], *assumption*+,
simp add:*Pi-def*, *simp add*:*Pi-def*)
apply *simp*
apply (*simp add*:*jointfun-def sliden-def*)
done

lemma (**in** *Module*) *same-together*:⟦*ideal R A*; $H \subseteq carrier\ M$;
$s \in \{j.\ j \leq (n{::}nat)\} \to A$; $f \in \{j.\ j \leq n\} \to H$⟧ ⟹
$\exists\, t \in \{j.\ j \leq (card\ (f\ `\ \{j.\ j \leq (n{::}nat)\}) - Suc\ 0)\} \to A.$
$\exists\, g \in \{j.\ j \leq (card\ (f\ `\ \{j.\ j \leq n\}) - Suc\ 0)\} \to f\ `\ \{j.\ j \leq n\}.$
surj-to g $\{j.\ j \leq (card\ (f\ `\ \{j.\ j \leq n\}) - Suc\ 0)\}\ (f\ `\ \{j.\ j \leq n\}) \wedge$
$\Sigma_e\ M\ (\lambda j.\ s\ j\ \cdot_s\ (f\ j))\ n =$
$\Sigma_e\ M\ (\lambda k.\ t\ k\ \cdot_s\ (g\ k))\ (card\ (f\ `\ \{j.\ j \leq n\}) - Suc\ 0)$
apply (*simp add*:*same-togetherTr*[*of A H*])
done

lemma (**in** *Module*) *one-last*:⟦*ideal R A*; $H \subseteq carrier\ M$;
$s \in \{j.\ j \leq (Suc\ n)\} \to A$; $f \in \{j.\ j \leq (Suc\ n)\} \to H$;
bij-to f $\{j.\ j \leq (Suc\ n)\}\ H$; $j \leq (Suc\ n)$; $j \neq (Suc\ n)$⟧ ⟹
$\exists\, t \in \{j.\ j \leq (Suc\ n)\} \to A.\ \exists\, g \in \{j.\ j \leq (Suc\ n)\} \to H.$
$\Sigma_e\ M\ (\lambda k.\ s\ k\ \cdot_s\ (f\ k))\ (Suc\ n) = \Sigma_e\ M\ (\lambda k.\ t\ k\ \cdot_s\ (g\ k))\ (Suc\ n) \wedge$
$g\ (Suc\ n) = f\ j \wedge t\ (Suc\ n) = s\ j \wedge$ *bij-to g* $\{j.\ j \leq (Suc\ n)\}\ H$
apply (*cut-tac sc-Ring*)
apply (*subgoal-tac* $(\lambda k.\ s\ k\ \cdot_s\ (f\ k)) \in \{j.\ j \leq Suc\ n\} \to carrier\ M$)
apply (*frule transpos-hom*[*of j Suc n Suc n*], *simp*, *assumption*,
frule transpos-inj[*of j Suc n Suc n*], *simp*, *assumption*,
frule-tac f1 $= \lambda k.\ \ s\ k\ \cdot_s\ (f\ k)$ **and** *n1* $= n$ **and** *h1* $=$
transpos j $(Suc\ n)$ **in** *addition2* [*THEN sym*], *assumption*+,
simp del:*nsum-suc*)
prefer *2*
apply (*rule Pi-I*, *rule sc-mem*,
simp add:*Pi-def Ring.ideal-subset*,
simp add:*Pi-def subsetD*)
apply (*frule cmp-fun*[*of transpos j* $(Suc\ n)$ $\{j.\ j \leq Suc\ n\}$
$\{j.\ j \leq Suc\ n\}$ *s A*], *assumption*+,
frule cmp-fun[*of transpos j* $(Suc\ n)$ $\{j.\ j \leq Suc\ n\}$
$\{j.\ j \leq Suc\ n\}$ *f H*], *assumption*+)
apply (*simp del*:*nsum-suc add*:*l-comb-transpos*[*of A H*])
apply (*subgoal-tac bij-to* (*cmp f* (*transpos j* $(Suc\ n)$)) $\{j.\ j \leq (Suc\ n)\}\ H$)
apply (*subgoal-tac* (*cmp f* (*transpos j* $(Suc\ n)$)) $(Suc\ n) = f\ j$)

apply (*subgoal-tac* (*cmp s* (*transpos j* (*Suc n*))) (*Suc n*) = *s j*)
apply *blast*
apply (*simp add*:*cmp-def*, *simp add*:*transpos-ij-2*,
simp add:*cmp-def*, *simp add*:*transpos-ij-2*)
apply (*simp add*:*bij-to-def*, *rule conjI*,
rule cmp-surj[*of transpos j* (*Suc n*) {*j*. *j* ≤ *Suc n*}
{*j*. *j* ≤ *Suc n*} *f H*], *assumption*+,
simp add:*transpos-surjec*, *assumption*+, *simp*)
apply (*rule cmp-inj*[*of transpos j* (*Suc n*) {*j*. *j* ≤ *Suc n*}
{*j*. *j* ≤ *Suc n*} *f H*], *assumption*+, *simp*)
done

lemma (**in** *Module*) *finite-lin-spanTr1*:⟦*ideal R A*; *z* ∈ *carrier M*⟧ ⟹
h ∈ {*j*. *j* ≤ (*n*::*nat*)} → {*z*} ∧ *t* ∈ {*j*. *j* ≤ *n*} → *A* ⟶
(∃ *s*∈{*0*::*nat*} → *A*. Σ$_e$ *M* (λ*j*. *t j* ·$_s$ (*h j*)) *n* = *s 0* ·$_s$ *z*)
apply (*induct-tac n*)
apply (*rule impI*)
apply ((*erule conjE*)+, *simp*)
apply *blast*

apply (*rule impI*) **apply** (*erule conjE*)+
apply (*frule func-pre* [*of* - - {*z*}], *frule func-pre* [*of* - - *A*])
apply (*simp del*:*nsum-suc*, *erule bexE*, *simp*,
frule-tac f = *h* **and** *A* = {*j*. *j* ≤ *Suc n*} **and** *B* = {*z*} **and** *x* = *Suc n*
in *funcset-mem*, *simp*, *simp*,
frule-tac f = *t* **and** *A* = {*j*. *j* ≤ *Suc n*} **and** *B* = *A* **and** *x* = *Suc n* **in**
funcset-mem, *simp*, *cut-tac sc-Ring*,
frule-tac h = *s 0* **in** *Ring.ideal-subset*[*of R A*], *assumption*+,
frule-tac h = *t* (*Suc n*) **in** *Ring.ideal-subset*[*of R A*], *assumption*+)
apply (*simp add*:*sc-l-distr*[*THEN sym*])
apply (*subgoal-tac* (λ*l*∈{*0*::*nat*}. (*s 0* ±$_R$ (*t* (*Suc n*)))) ∈ {*0*} → *A*)
apply (*subgoal-tac* (*s 0* ±$_R$ *t* (*Suc n*)) ·$_s$ *z* = (λ*l*∈{*0*::*nat*}. (*s 0* ±$_R$ (*t* (*Suc n*))))
0 ·$_s$ *z*) **apply** *blast*
apply *simp*
apply (*rule Pi-I*) **apply** *simp*
apply (*rule Ring.ideal-pOp-closed*, *assumption*+)
done

lemma (**in** *Module*) *single-span*:⟦*ideal R A*; *z* ∈ *carrier M*;
h ∈ {*j*. *j* ≤ (*n*::*nat*)} → {*z*}; *t* ∈ {*j*. *j* ≤ *n*} → *A*⟧ ⟹
∃ *s*∈{*0*::*nat*} → *A*. Σ$_e$ *M* (λ*j*. *t j* ·$_s$ (*h j*)) *n* = *s 0* ·$_s$ *z*
apply (*simp add*:*finite-lin-spanTr1*)
done

definition
coeff-at-k :: [('*r*, '*m*) *Ring-scheme*, '*r*, *nat*] ⇒ (*nat* ⇒ '*r*) **where**
coeff-at-k R a k = (λ*j*. *if j* = *k then a else* (**0**$_R$))

lemma *card-Nset-im*:$f \in \{j.\ j \leq (n::nat)\} \to A \Longrightarrow$
$(Suc\ 0) \leq card\ (f\ `\{j.\ j \leq n\})$
apply (*cut-tac image-Nsetn-card-pos*[*of f n*])
apply (*frule-tac* $m = 0$ **and** $n = card\ (f\ `\ \{i.\ i \leq n\})$ **in** *Suc-leI*,
assumption+)
done

lemma (**in** *Module*) *eSum-changeTr1*:⟦*ideal R A*;
$t \in \{k.\ k \leq (card\ (f\ `\ \{j.\ j \leq (n1::nat)\}) - Suc\ 0)\} \to A$;
$g \in \{k.\ k \leq (card\ (f\ `\ \{j.\ j \leq n1\}) - Suc\ 0)\} \to f\ `\{j.\ j \leq n1\}$;
$Suc\ 0 < card\ (f\ `\{j.\ j \leq n1\})$; $g\ x = h\ (Suc\ n)$; $x = Suc\ n$;
$card\ (f\ `\{j.\ j \leq n1\}) - Suc\ 0 = \ Suc\ (card\ (f\ `\ \{j.\ j \leq n1\}) - Suc\ 0 - Suc\ 0)$⟧
$\Longrightarrow$
$\Sigma_e\ M\ (\lambda k.\ t\ k\ \cdot_s\ (g\ k))\ (card\ (f\ `\ \{j.\ j \leq n1\}) - Suc\ 0) =$
$\Sigma_e\ M\ (\lambda k.\ t\ k\ \cdot_s\ (g\ k))\ (card\ (f\ `\ \{j.\ j \leq n1\}) - Suc\ 0 - Suc\ 0) \pm$
$(t\ (Suc\ (card\ (f\ `\ \{j.\ j \leq n1\}) - Suc\ 0 - Suc\ 0))\ \cdot_s$
$(g\ (\ Suc\ (card\ (f\ `\ \{j.\ j \leq n1\}) - Suc\ 0 - Suc\ 0))))$
apply *simp*
done

definition
zeroi :: [('*r*, '*m*) *Ring-scheme*] ⇒ *nat* ⇒ '*r* **where**
zeroi R = $(\lambda j.\ \mathbf{0}_R)$

lemma *zeroi-func*:⟦*Ring R*; *ideal R A*⟧ $\Longrightarrow$ *zeroi R* $\in \{j.\ j \leq 0\} \to A$
by (*simp add*:*zeroi-def Ring.ideal-zero*)

lemma (**in** *Module*) *prep-arrTr1*:⟦*ideal R A*; $h \in \{j.\ j \leq (Suc\ n)\} \to carrier\ M$;
$f \in \{j.\ j \leq (n1::nat)\} \to h\ `\ \{j.\ j \leq (Suc\ n)\}$; $s \in \{j.\ j \leq n1\} \to A$;
$m = l\text{-}comb\ R\ M\ n1\ s\ f$⟧ $\Longrightarrow$
$\exists\, l \in \{j.\ j \leq (Suc\ n)\}.\ (\exists\, s \in \{j.\ j \leq (l::nat)\} \to A.$
$\exists\, g \in \{j.\ j \leq l\} \to h\ `\{j.\ j \leq (Suc\ n)\}.\ m = l\text{-}comb\ R\ M\ l\ s\ g \wedge$
$bij\text{-}to\ g\ \{j.\ j \leq l\}\ (f\ `\ \{j.\ j \leq n1\}))$
apply (*cut-tac sc-Ring*)
apply (*frule-tac* $s = s$ **and** $n = n1$ **and** $f = f$ **in** *same-together*[*of A*
$h\ `\ \{j.\ j \leq (Suc\ n)\}$])
apply (*simp add*:*image-sub0*, *assumption*+)
apply (*erule bexE*)+
apply (*simp add*:*l-comb-def*, *erule conjE*)
apply (*thin-tac* $\Sigma_e\ M\ (\lambda j.\ s\ j\ \cdot_s\ f\ j)\ n1 =$
$\Sigma_e\ M\ (\lambda k.\ t\ k\ \cdot_s\ g\ k)\ (card\ (f\ `\ \{j.\ j \leq n1\}) - Suc\ 0)$)
apply (*subgoal-tac* $(card\ (f\ `\ \{j.\ j \leq n1\}) - Suc\ 0) \in \{j.\ j \leq Suc\ n\}$)
apply (*subgoal-tac* $g \in \{k.\ k \leq (card\ (f\ `\{j.\ j \leq n1\}) - Suc\ 0)\} \to$
$h\ `\ \{j.\ j \leq Suc\ n\}$)
apply (*subgoal-tac bij-to g* $\{k.\ k \leq (card\ (f\ `\ \{j.\ j \leq n1\}) - Suc\ 0)\}\ (f\ `\ \{j.\ j \leq n1\}))$
apply *blast*
prefer *2*
apply (*frule-tac* $f = f$ **and** $A = \{j.\ j \leq n1\}$ **and** $B = h\ `\ \{j.\ j \leq Suc\ n\}$

```
        in image-sub0, simp)
  apply (rule extend-fun, assumption+)
 apply (simp add:bij-to-def)
apply (rule-tac A = f ‘ {j. j ≤ n1} and n = card (f ‘{j. j ≤ n1}) − Suc 0 and
f = g in Nset2finite-inj)
 apply (rule finite-imageI, simp)
 apply (frule-tac f = f and n = n1 and A = h ‘ {j. j ≤ (Suc n)} in card-Nset-im)
 apply (simp, assumption)
apply (subgoal-tac finite (h ‘ {j. j ≤ (Suc n)}))
apply (frule-tac f = f and A = {j. j ≤ n1} and B = h ‘ {j. j ≤ (Suc n)}
      in image-sub0, simp)
 apply (cut-tac B = h ‘ {j. j ≤ (Suc n)} and A = f ‘ {j. j ≤ n1} in
       card-mono, simp,  assumption+,
       insert finite-Collect-le-nat [of Suc n],
       frule card-image-le [of {j. j ≤ (Suc n)} h],
       frule-tac i = card (f ‘ {j. j ≤ n1}) and
        j = card (h ‘ {j. j ≤ (Suc n)}) and k = card {j. j ≤ (Suc n)} in
       le-trans, assumption+)
 apply simp
 apply (rule finite-imageI, simp)
done

lemma two-func-imageTr:⟦ h ∈ {j. j ≤ Suc n} → B;
  f ∈ {j. j ≤ (m::nat)} → h ‘ {j. j ≤ Suc n};  h (Suc n) ∉ f ‘ {j. j ≤ m}⟧
     ⟹ f ∈ {j. j ≤ m} → h ‘ {j. j ≤ n}
apply (rule Pi-I)
   apply (frule-tac x = x and f = f and A = {j. j ≤ m} and
         B = h ‘ {j. j ≤ Suc n} in funcset-mem, assumption)
  apply (thin-tac h ∈ {j. j ≤ Suc n} → B)
  apply (rule contrapos-pp, simp+)
    apply (simp add:image-def[of h])
    apply (erule exE, erule conjE)
    apply (case-tac xa ≠ Suc n,
          frule-tac m = xa and n = Suc n in noteq-le-less, assumption)
        apply (
          thin-tac xa ≤ Suc n,
          frule-tac x = xa and n = Suc n in less-le-diff,
          thin-tac xa < Suc n, simp) apply blast
    apply simp
    apply (subgoal-tac (f x) ∈ f ‘ {j. j ≤ m}, simp)
     apply (thin-tac h (Suc n) ∉ f ‘ {j. j ≤ m},
               thin-tac ∀ x≤n. h (Suc n) ≠ h x,
               thin-tac f x = h (Suc n),
               thin-tac xa = Suc n)
    apply (simp add:image-def, blast)
done

lemma (in Module) finite-lin-spanTr3-0:⟦bij-to g {j. j ≤ l} (g ‘{j. j ≤ l});
     ideal R A;
```

$\forall\, na.\ \forall\, s{\in}\{j.\ j \le na\} \to A.$
$\quad \forall f{\in}\{j.\ j \le na\} \to h\ `\ \{j.\ j \le n\}.$
$\quad\ \exists\, t{\in}\{j.\ j \le n\} \to A.\ l\text{-}comb\ R\ M\ na\ s\ f = l\text{-}comb\ R\ M\ n\ t\ h;$
$h \in \{j.\ j \le Suc\ n\} \to carrier\ M;\ s \in \{j.\ j \le m\} \to A;$
$f \in \{j.\ j \le m\} \to h\ `\ \{j.\ j \le Suc\ n\};$
$l \le Suc\ n;\ sa \in \{j.\ j \le l\} \to A;\ g \in \{j.\ j \le l\} \to h\ `\ \{j.\ j \le Suc\ n\};$
$0 < l;\ f\ `\ \{j.\ j \le m\} = g\ `\ \{j.\ j \le l\};\ h\ (Suc\ n) = g\ l\rrbracket$
$\Longrightarrow \exists\, t{\in}\{j.\ j \le Suc\ n\} \to A.\ l\text{-}comb\ R\ M\ l\ sa\ g = l\text{-}comb\ R\ M\ (Suc\ n)\ t\ h$
apply (*cut-tac sc-Ring*)
apply (*subgoal-tac l-comb R M l sa g = l-comb R M (Suc (l − Suc 0)) sa g*,
simp del:*Suc-pred*,
thin-tac l-comb R M l sa g = l-comb R M (Suc (l − Suc 0)) sa g,
simp del:*Suc-pred add*:*l-comb-def*)
apply (*drule-tac x = l − Suc 0* **in** *spec*,
drule-tac x = sa **in** *bspec*)

apply (*rule Pi-I*, *simp*)
apply (*rule-tac x = x* **and** *f = sa* **and** *A = {j. j ≤ l}***and** *B = A*
in *funcset-mem*, *assumption*, *simp*)
apply (*drule-tac x = g* **in** *bspec*,
thin-tac f ∈ {j. j ≤ m} → h ` {j. j ≤ Suc n},
thin-tac sa ∈ {j. j ≤ l} → A,
thin-tac f ` {j. j ≤ m} = g ` {j. j ≤ l})
apply (*rule Pi-I*, *simp*)
apply (*frule-tac x = x* **and** *f = g* **and** *A = {j. j ≤ l}* **and**
B = h ` {j. j ≤ Suc n} **in** *funcset-mem*)
apply *simp*
apply (*unfold bij-to-def*, *frule conjunct2*, *fold bij-to-def*,
thin-tac bij-to g {j. j ≤ l} (g ` {j. j ≤ l}),
thin-tac g ∈ {j. j ≤ l} → h ` {j. j ≤ Suc n})
apply (*simp add*:*image-def*, *erule exE*, *erule conjE*)
apply (*case-tac xa = Suc n*, *simp add*:*inj-on-def*,
drule-tac a = x **in** *forall-spec*) **apply** *simp*

apply(*drule-tac a = l* **in** *forall-spec*, *simp*)
apply (*cut-tac n1 = l* **and** *m1 = l − Suc 0* **in** *Suc-le-mono*[*THEN sym*])
apply *simp*
apply (*frule-tac m = xa* **and** *n = Suc n* **in** *noteq-le-less*, *assumption*,
thin-tac xa ≤ Suc n,
frule-tac x = xa **and** *n = Suc n* **in** *less-le-diff*,
thin-tac xa < Suc n, *simp*)
apply *blast*
apply (*erule bexE*, *simp*)
apply (*rotate-tac −4*, *frule sym*, *thin-tac h (Suc n) = g l*, *simp*)
apply (*frule-tac f = t* **and** *n = n* **and** *A = A* **and** *g = λk∈{0::nat}. sa l*
and *m = 0* **and** *B = A* **in** *jointfun-hom0*,
simp add:*Pi-def*, *simp*)
apply (*subgoal-tac* $\Sigma_e\ M\ (\lambda j.\ t\ j \cdot_s h\ j)\ n \pm sa\ l \cdot_s h\ (Suc\ n) =$
$\Sigma_e\ M\ (\lambda j.\ (jointfun\ n\ t\ 0\ (\lambda k{\in}\{0\}.\ sa\ l))\ j \cdot_s h\ j)\ (Suc\ n)$,

```
        simp, blast)
   apply (cut-tac H = carrier M and A = A and s = t and f = h and n = n
and
        m = 0 and t = λk∈{0}. sa l in l-comb-jointfun-jf)
        apply simp+
        apply (simp add:Pi-def)
        apply simp
   apply (simp add:jointfun-def sliden-def, simp)
done

lemma (in Module) finite-lin-spanTr3:ideal R A ⟹
      h ∈ {j. j ≤ (n::nat)} → carrier M ⟶
     (∀ na. ∀ s ∈ {j. j ≤ (na::nat)} → A.
      ∀ f∈ {j. j ≤ na} → (h ‘ {j. j ≤ n}). (∃ t ∈ {j. j ≤ n} → A.
      l-comb R M na s f = l-comb R M n t h))
apply (cut-tac sc-Ring)
apply (induct-tac n)
 apply (rule impI, rule allI, (rule ballI)+)
 apply (insert Nset-nonempty [of 0])
 apply (simp add:l-comb-def)
 apply (frule-tac z = h 0 and h = f and t = s and n = na in
        single-span [of A])
 apply (simp add:Pi-def)
 apply assumption+

apply (rule impI, rule allI, (rule ballI)+)
 apply (frule func-pre, simp)
 apply (case-tac h (Suc n) ∉  f ‘ {j. j ≤ na})
  apply (frule-tac h = h and n = n and B = carrier M and f = f and
        m = na in two-func-imageTr, assumption+)
  apply (drule-tac x = na in spec,
        drule-tac x = s in bspec, assumption,
        drule-tac x = f in bspec, assumption)

  apply (erule bexE, simp )
  apply (thin-tac l-comb R M na s f = l-comb R M n t h)
apply (simp add:l-comb-def)
 apply (subgoal-tac Σe M (λj. t j ·s (h j)) n =
       Σe M (λj. (jointfun n t 0 (zeroi R)) j ·s (h j)) (Suc n), simp,
       thin-tac Σe M (λj. t j ·s h j) n =
       Σe M (λj. jointfun n t 0 (zeroi R) j ·s h j) n ±
       jointfun n t 0 (zeroi R) (Suc n) ·s h (Suc n))
 apply (frule-tac f = t and n = n and g = zeroi R and m = 0 and A = A and

       B = A in jointfun-hom)
       apply (rule zeroi-func, assumption+, simp, blast)
 apply (cut-tac H = carrier M and s = t and n = n and f = h and m = 0
and
       t = zeroi R in l-comb-jointfun-jf[of - A],
```

simp, *assumption*+, *simp*,
rule zeroi-func, *assumption*+)
apply (*simp*,
thin-tac $\Sigma_e\ M\ (\lambda j.\ jointfun\ n\ t\ 0\ (zeroi\ R)\ j\ \cdot_s\ h\ j)\ n =$
$\Sigma_e\ M\ (\lambda j.\ t\ j\ \cdot_s\ h\ j)\ n$,
simp add:*jointfun-def sliden-def zeroi-def*,
subst sc-0-m, *simp add*:*Pi-def*,
subst ag-r-zero,
rule nsum-mem, *rule allI*, *rule impI*, *rule sc-mem*,
simp add:*Pi-def Ring.ideal-subset*,
simp add:*Pi-def*,
simp)

apply *simp*
apply (*frule-tac* $h = h$ **and** $n = n$ **and** $m = l\text{-}comb\ R\ M\ na\ s\ f$ **in**
prep-arrTr1 [*of A*], *assumption*+, *simp*)
apply (*erule bexE*)+
apply (*simp*, (*erule conjE*)+)
apply (*case-tac* $l = 0$, *simp*)
apply (*unfold bij-to-def*, *frule conjunct1*, *frule conjunct2*, *fold bij-to-def*)
apply (*thin-tac l-comb R M na s f* $=$ *l-comb R M 0 sa g*)
apply (*simp add*:*l-comb-def*)
apply (*simp add*:*surj-to-def*, *rotate-tac* -1, *frule sym*,
thin-tac $\{g\ 0\} = f\ `\ \{j.\ j \leq na\}$, *simp*,
rotate-tac -6, *frule sym*, *thin-tac h* $(Suc\ n) = g\ 0$, *simp*)
apply (*cut-tac* $f = zeroi\ R$ **and** $n = n$ **and** $g = \lambda j.\ sa\ 0$ **and** $m = 0$ **and**
$A = A$ **and** $B = A$ **in** *jointfun-hom0*)
apply (*simp add*:*zeroi-def Ring.ideal-zero*)
apply (*simp add*:*Pi-def*)
apply *simp*
apply (*subgoal-tac sa 0* $\cdot_s$ *h* $(Suc\ n) = nsum\ M\ (\lambda j.\ (jointfun\ n\ (zeroi\ R)\ 0$
$(\lambda j.\ sa\ 0)\ j\ \cdot_s\ h\ j))\ (Suc\ n)$, *simp*,
thin-tac sa 0 $\cdot_s$ *h* $(Suc\ n) =$
$\Sigma_e\ M\ (\lambda j.\ jointfun\ n\ (zeroi\ R)\ 0\ (\lambda j.\ sa\ 0)\ j\ \cdot_s\ h\ j)\ n \pm$
jointfun n $(zeroi\ R)\ 0\ (\lambda j.\ sa\ 0)\ (Suc\ n) \cdot_s\ h\ (Suc\ n)$,
blast)
apply *simp*
apply (*cut-tac* $n = n$ **and** $f = \lambda j.\ jointfun\ n\ (zeroi\ R)\ 0\ (\lambda j.\ sa\ 0)\ j\ \cdot_s\ h\ j$
in *nsum-zeroA*)
apply (*rule allI*, *rule impI*,
simp add:*jointfun-def zeroi-def*,
rule sc-0-m, *simp add*:*Pi-def*, *simp*,
thin-tac $\Sigma_e\ M\ (\lambda j.\ jointfun\ n\ (zeroi\ R)\ 0\ (\lambda j.\ sa\ 0)\ j\ \cdot_s\ h\ j)\ n = \mathbf{0}$)
apply (*simp add*:*jointfun-def sliden-def*,
subst ag-l-zero,
rule sc-mem, *simp add*:*Pi-def Ring.ideal-subset*,
simp add:*Pi-def*, *simp*)

apply (*simp*)
 apply (*thin-tac l-comb R M na s f = l-comb R M l sa g*)
 apply (*unfold bij-to-def*, *frule conjunct1*, *frule conjunct2*, *fold bij-to-def*)
 apply (*simp add*:*surj-to-def*, *rotate-tac −2*, *frule sym*,
 thin-tac g ‘ {j. j ≤ l} = f ‘ {j. j ≤ na}, *simp*)
 apply (*subgoal-tac* ∃ *x*∈{*j. j ≤ l*}. *h* (*Suc n*) = *g x*)
 prefer *2* **apply** (*simp add*:*image-def*)
apply (*erule bexE*)
 apply (*case-tac x = l*, *simp*)
apply (*frule-tac g = g* **and** *l = l* **and** *A = A* **and** *h = h* **and** *n = n* **and** *s = s*
and
 m = na **and** *f = f* **and** *l = l* **and** *sa = sa* **in** *finite-lin-spanTr3-0*,
 assumption+)

apply (*subgoal-tac l-comb R M l sa g = l-comb R M* (*Suc* (*l − Suc 0*)) *sa g*)
 prefer *2* **apply** *simp*
 apply (*simp del*:*nsum-suc Suc-pred*,
 thin-tac l-comb R M l sa g = l-comb R M (*Suc* (*l − Suc 0*)) *sa g*,
 simp del:*nsum-suc Suc-pred add*:*l-comb-def*)
 apply (*cut-tac f1 = λj. sa j* ·$_s$ *g j* **and** *n1 = l − Suc 0* **and**
 h1 = transpos x (*Suc* (*l − Suc 0*)) **in** *addition2*[*THEN sym*],
 thin-tac ∀ *na*. ∀ *s*∈{*j. j ≤ na*} → *A*.
 ∀ *f*∈{*j. j ≤ na*} → *h ‘* {*j. j ≤ n*}.
 ∃ *t*∈{*j. j ≤ n*} → *A*.
 Σ$_e$ *M* (*λj. s j* ·$_s$ *f j*) *na* = Σ$_e$ *M* (*λj. t j* ·$_s$ *h j*) *n*,
 rule Pi-I, *simp*)
 apply (*rule sc-mem*,
 simp add:*Pi-def Ring.ideal-subset*,
 frule-tac f = h **and** *A* = {*j. j ≤ Suc n*} **and** *B = carrier M* **in**
 image-sub0,
 frule-tac x = xa **and** *f = g* **and** *A* = {*j. j ≤ l*} **and**
 B = h ‘ {*j. j ≤ Suc n*} **in** *funcset-mem*, *simp*, *simp add*:*subsetD*)
 apply (*simp*,
 rule-tac i = x **and** *n = l* **and** *j = l* **in** *transpos-hom*,
 assumption+, *simp*, *assumption+*)
 apply (*simp*,
 rule-tac i = x **and** *n = l* **and** *j = l* **in** *transpos-inj*,
 assumption+, *simp*, *assumption+*)
 apply (*simp del*:*Suc-pred nsum-suc*)
 apply (*subst l-comb-transpos*[*of A carrier M*], *assumption*, *simp*,
 simp, *simp*,
 rule Pi-I,
 frule-tac f = h **and** *A* = {*j. j ≤ Suc n*} **and** *B = carrier M* **in**
 image-sub0,
 frule-tac x = xa **and** *f = g* **and** *A* = {*j. j ≤ l*} **and**
 B = h ‘ {*j. j ≤ Suc n*} **in** *funcset-mem*, *simp*, *simp add*:*subsetD*,
 simp)
 apply (*simp del*:*Suc-pred*, *simp*,

thin-tac $\Sigma_e\ M\ (\lambda j.\ sa\ j\ \cdot_s\ g\ j)\ (l - Suc\ 0) \pm sa\ l\ \cdot_s\ g\ l =$
$\Sigma_e\ M\ (cmp\ (\lambda j.\ sa\ j\ \cdot_s\ g\ j)\ (transpos\ x\ l))\ (l - Suc\ 0) \pm$
$cmp\ (\lambda j.\ sa\ j\ \cdot_s\ g\ j)\ (transpos\ x\ l)\ l$)

apply (*cut-tac* $g = cmp\ g\ (transpos\ x\ l)$ **and** $l = l$ **and** $A = A$ **and**
$h = h$ **and** $n = n$ **and** $s = s$ **and** $m = na$ **and** $f = f$ **and** $l = l$ **and**
$sa = cmp\ sa\ (transpos\ x\ l)$ **in** *finite-lin-spanTr3-0*)

apply (*frule-tac* $i = x$ **and** $n = l$ **and** $j = l$ **in** *transpos-hom*,
simp, *assumption*)
apply (*cut-tac* $n = l$ **in** *Nat.le-refl*)
apply (*frule-tac* $i = x$ **and** $n = l$ **and** $j = l$ **in** *transpos-surjec*, *assumption*+)
apply (*frule-tac* $f = transpos\ x\ l$ **and** $A = \{j.\ j \leq l\}$ **and**
$B = \{j.\ j \leq l\}$ **and** $g = g$ **and** $C = g\ `\ \{j.\ j \leq l\}$ **in** *cmp-surj*,
assumption+)
apply (*rule-tac* $f = g$ **and** $A = \{j.\ j \leq l\}$ **and** $B = h\ `\ \{j.\ j \leq Suc\ n\}$
in *func-to-img*, *assumption*)
apply (*simp add*:*bij-to-def*)
apply (*subst bij-to-def*, *simp*)
apply (*subgoal-tac cmp g* (*transpos x l*) $`\ \{j.\ j \leq l\} = g\ `\ \{j.\ j \leq l\}$,
simp)
apply (*frule-tac* $f = transpos\ x\ l$ **and** $A = \{j.\ j \leq l\}$ **and**
$B = \{j.\ j \leq l\}$ **and** $g = g$ **and** $C = h\ `\ \{j.\ j \leq Suc\ n\}$ **in** *cmp-inj*,
assumption+)
apply (*rule-tac* $i = x$ **and** $n = l$ **and** $j = l$ **in** *transpos-inj*, *assumption*,
simp, *assumption*, *simp add*:*bij-to-def*,
assumption)
apply (*simp add*:*cmp-fun-image*, *simp add*:*surj-to-def*)

apply *assumption*+
apply (*simp add*:*l-comb-def*)
apply *assumption*+

apply (*rule Pi-I*)
apply (*simp add*:*cmp-def*)
apply (*cut-tac* $n = l$ **in** *Nat.le-refl*,
frule-tac $i = x$ **and** $n = l$ **and** $j = l$ **and** $l = xa$ **in** *transpos-mem*,
assumption+,
simp add:*Pi-def*)
apply (*rule Pi-I*,
simp add:*cmp-def*,
cut-tac $n = l$ **in** *Nat.le-refl*,
frule-tac $i = x$ **and** $n = l$ **and** $j = l$ **and** $l = xa$ **in** *transpos-mem*,
assumption+,
simp add:*Pi-def*)
apply *simp*

apply (*cut-tac* $n = l$ **in** *Nat.le-refl*,
frule-tac $i = x$ **and** $n = l$ **and** $j = l$ **in** *transpos-surjec*, *assumption*+)

apply (*frule-tac* $i = x$ **and** $n = l$ **and** $j = l$ **in** *transpos-hom*,
simp, *assumption*)
apply (*frule-tac* $f = transpos\ x\ l$ **and** $A = \{i.\ i \leq l\}$ **and**
$B = \{i.\ i \leq l\}$ **and** $g = g$ **and** $C = h\ `\ \{j.\ j \leq Suc\ n\}$ **in**
cmp-fun-image, *assumption*+)
apply (*simp add*:*surj-to-def*)
apply (*simp add*:*cmp-def*)
apply (*simp add*:*transpos-ij-2*)
apply (*erule bexE*)
apply (*thin-tac* $\forall na.\ \forall s{\in}\{j.\ j \leq na\} \to A.$
$\forall f{\in}\{j.\ j \leq na\} \to h\ `\ \{j.\ j \leq n\}.$
$\exists t{\in}\{j.\ j \leq n\} \to A.$
$\Sigma_e\ M\ (\lambda j.\ s\ j\ \cdot_s\ f\ j)\ na = \Sigma_e\ M\ (\lambda j.\ t\ j\ \cdot_s\ h\ j)\ n$)
apply (*rename-tac n na s f l sa g x sb*)
apply (*subgoal-tac l-comb R M l* (*cmp sa* (*transpos x l*))
(*cmp g* (*transpos x l*)) = *l-comb R M* (*Suc* (*l − Suc 0*))
(*cmp sa* (*transpos x l*)) (*cmp g* (*transpos x l*)) ,
simp del:*Suc-pred*,
thin-tac l-comb R M l (*cmp sa* (*transpos x l*)) (*cmp g* (*transpos x l*)) =
l-comb R M (*Suc n*) *sb h*)
apply (*simp del*:*Suc-pred add*:*l-comb-def*, *simp*,
thin-tac $\Sigma_e\ M\ (\lambda j.\ cmp\ sa\ (transpos\ x\ l)\ j\ \cdot_s$
cmp g (*transpos x l*) *j*) (*l − Suc 0*) ±
cmp sa (*transpos x l*) *l* $\cdot_s$ *cmp g* (*transpos x l*) *l* =
$\Sigma_e\ M\ (\lambda j.\ sb\ j\ \cdot_s\ h\ j)\ n \pm sb\ (Suc\ n)\ \cdot_s\ g\ x$)
apply (*rotate-tac −3*, *frule sym*, *thin-tac h* (*Suc n*) = *g x*, *simp*)
apply *blast*

apply *simp*
done

lemma (**in** *Module*) *finite-lin-span*:
$[\![ideal\ R\ A;\ \ h \in \{j.\ j \leq (n{::}nat)\} \to carrier\ M;\ s \in \{j.\ j \leq (n1{::}nat)\} \to A;$
$f \in \{j.\ j \leq n1\} \to h\ `\ \{j.\ j \leq n\}]\!] \Longrightarrow \exists t{\in}\{j.\ j \leq n\} \to A.$
l-comb R M n1 s f = *l-comb R M n t h*
apply (*simp add*:*finite-lin-spanTr3*)
done

5.3.2 Free generators

definition
free-generator :: [('*r*, '*m*) *Ring-scheme*, ('*a*, '*r*, '*m1*) *Module-scheme*, '*a set*]
⇒ *bool* **where**
free-generator R M H ⟷ *generator R M H* ∧
$(\forall n.\ (\forall s\ f.\ (s \in \{j.\ j \leq (n{::}nat)\} \to carrier\ R \wedge$
$f \in \{j.\ j \leq n\} \to H \wedge inj\text{-}on\ f\ \{j.\ j \leq n\} \wedge$
l-comb R M n s f = $\mathbf{0}_M$) ⟶ $s \in \{j.\ j \leq n\} \to \{\mathbf{0}_R\}))$

lemma (**in** *Module*) *free-generator-generator*:*free-generator R M H* ⟹

generator R M H
by (*simp add*:*free-generator-def*)

lemma (**in** *Module*) *free-generator-sub*:*free-generator R M H* $\Longrightarrow$
H $\subseteq$ *carrier M*
by (*simp add*:*free-generator-def generator-def*)

lemma (**in** *Module*) *free-generator-nonzero*:$[\![\neg$ (*zeroring R*);
free-generator R M H; *h* $\in$ *H*$]\!] \Longrightarrow h \neq \mathbf{0}$
apply (*cut-tac sc-Ring*)
apply (*rule contrapos-pp*, *simp*+)
apply (*simp add*:*free-generator-def*, (*erule conjE*)+)
apply (*subgoal-tac* $(\lambda t.\ 1_{r\,R}) \in \{j.\ j \leq (0::nat)\} \to$ *carrier R*)
apply (*subgoal-tac* $(\lambda t.\ \mathbf{0}) \in \{j.\ j \leq (0::nat)\} \to H \wedge$
inj-on $(\lambda t.\ \mathbf{0})\ \{j.\ j \leq (0::nat)\} \wedge$
l-comb R M 0 $(\lambda t.\ 1_{r\,R})\ (\lambda t.\ \mathbf{0}) = \mathbf{0}$)
apply (*subgoal-tac* $(\lambda t.\ 1_{r\,R}) \in \{j.\ j \leq (0::nat)\} \to \{\mathbf{0}_R\}$)
prefer *2* **apply** *blast*
apply (*frule-tac f* $= \lambda t.\ 1_{r\,R}$ **and** *A* $= \{j.\ j \leq (0::nat)\}$ **and** *B* $= \{\mathbf{0}_R\}$
and *x* = *0* **in** *funcset-mem*, *simp*, *simp*)
apply (*frule Ring.Zero-ring1* [*of R*], *assumption*+, *simp*)
apply *simp*
apply (*thin-tac* $\forall\, n\ s.\ s \in \{j.\ j \leq n\} \to$ *carrier R* $\wedge$
$(\exists f.\ f \in \{j.\ j \leq n\} \to H \wedge$
inj-on f $\{j.\ j \leq n\} \wedge$ *l-comb R M n s f* $= \mathbf{0}) \longrightarrow$
$s \in \{j.\ j \leq n\} \to \{\mathbf{0}_R\}$)
apply (*simp add*:*l-comb-def*)
apply (*rule sc-a-0*)
apply (*simp add*:*Ring.ring-one*)
apply (*simp add*:*Ring.ring-one*)
done

lemma (**in** *Module*) *has-free-generator-nonzeroring*: $[\![$*free-generator R M H*;
$\exists\, p \in$ *linear-span R M* (*carrier R*) *H*. $p \neq \mathbf{0}\ ]\!] \Longrightarrow \neg$ *zeroring R*
apply (*erule bexE*, *simp add*:*linear-span-def*)
apply (*case-tac H* $= \{\}$, *simp*, *simp*)
apply (*erule exE*, (*erule bexE*)+, *simp*,
thin-tac p $=$ *l-comb R M n s f*)
apply (*rule contrapos-pp*, *simp*+)
apply (*simp add*:*zeroring-def*, *erule conjE*)
apply (*frule Ring.ring-one*[*of R*], *simp*)
apply (*simp add*:*l-comb-def*)
apply (*cut-tac n* $= n$ **and** *f* $= \lambda j.\ s\ j\ \cdot_s\ f\ j$ **in** *nsum-zeroA*)
apply (*rule allI*, *rule impI*)
apply (*simp add*:*free-generator-def generator-def*, *frule conjunct1*,
frule-tac x $= j$ **and** *f* $= f$ **and** *A* $= \{j.\ j \leq n\}$ **and** *B* $= H$ **in**
funcset-mem, *simp*,
frule-tac c $= f\ j$ **in** *subsetD*[*of H carrier M*], *assumption*+,
frule-tac x $= j$ **and** *f* $= s$ **and** *A* $= \{j.\ j \leq n\}$ **and** *B* $= \{\mathbf{0}_R\}$ **in**

funcset-mem, *simp*, *simp add*:*sc-0-m*)
apply *simp*
done

lemma (**in** *Module*) *unique-expression1*:⟦*H* ⊆ *carrier M*; *free-generator R M H*;
s ∈ {*j*. *j* ≤ (*n*::*nat*)} → *carrier R*; *m* ∈ {*j*. *j* ≤ *n*} → *H*;
inj-on m {*j*. *j* ≤ *n*}; *l-comb R M n s m* = **0**⟧ ⟹
∀*j*∈{*j*. *j* ≤ *n*}. *s j* = **0**$_R$
apply (*rule ballI*)
apply (*simp add*:*free-generator-def*, (*erule conjE*)+)
apply (*subgoal-tac s* ∈ {*j*. *j* ≤ *n*} → {**0**$_R$})
apply (*frule-tac f* = *s* **and** *A* = {*j*. *j* ≤ *n*} **and** *B* = {**0**$_R$} **and** *x* = *j* **in**
funcset-mem, *simp*, *simp*)
apply *blast*
done

lemma (**in** *Module*) *free-gen-coeff-zero*:⟦*H* ⊆ *carrier M*; *free-generator R M H*;
h ∈ *H*; *a* ∈ *carrier R*; *a* ·$_s$ *h* = **0**⟧ ⟹ *a* = **0**$_R$
apply (*frule unique-expression1*[*of H* λ*x*∈{*0*::*nat*}. *a 0* λ*x*∈{*0*::*nat*}. *h*],
assumption+,
simp,
simp,
simp add:*inj-on-def*,
simp add:*l-comb-def*,
simp)
done

lemma (**in** *Module*) *unique-expression2*:⟦*H* ⊆ *carrier M*;
f ∈ {*j*. *j* ≤ (*n*::*nat*)} → *H*; *s* ∈ {*j*. *j* ≤ *n*} → *carrier R*⟧ ⟹
∃*m g t*. *g* ∈ ({*j*. *j* ≤ (*m*::*nat*)} → *H*) ∧
bij-to g {*j*. *j* ≤ (*m*::*nat*)} (*f* ‘ {*j*. *j* ≤ *n*}) ∧
t ∈ {*j*. *j* ≤ *m*} → *carrier R* ∧
l-comb R M n s f = *l-comb R M m t g*
apply (*cut-tac sc-Ring*)
apply (*frule Ring.whole-ideal* [*of R*])
apply (*frule-tac A* = *carrier R* **and** *H* = *H* **and** *s* = *s* **and** *f* = *f* **in**
same-together, *assumption*+)
apply ((*erule bexE*)+, *erule conjE*)
apply (*frule-tac f* = *f* **and** *A* = {*j*. *j* ≤ *n*} **in** *image-sub0*,
frule-tac f = *g* **and** *A* = {*j*. *j* ≤ *card* (*f* ‘ {*j*. *j* ≤ *n*}) − *Suc 0*}
and *B* = *f* ‘ {*j*. *j* ≤ *n*} **in** *extend-fun*[*of - - - H*], *assumption*)
apply (*subgoal-tac bij-to g* {*j*. *j* ≤ (*card* (*f* ‘ {*j*. *j* ≤ *n*}) − *Suc 0*)}
(*f* ‘ {*j*. *j* ≤ *n*}))
apply (*simp add*:*l-comb-def*, *blast*)
apply (*simp add*:*bij-to-def*)
apply (*cut-tac finite-Collect-le-nat*[*of n*],
frule finite-imageI[*of* {*j*. *j* ≤ *n*} *f*])
apply (*rule-tac A* = *f* ‘ {*j*. *j* ≤ *n*} **and** *n* = *card* (*f* ‘ {*j*. *j* ≤ *n*}) −
Suc 0 **and** *f* = *g* **in** *Nset2finite-inj*, *assumption*)

using *image-Nsetn-card-pos*[*of f n*] **apply** *simp*
apply *assumption*
done

lemma (**in** *Module*) *unique-expression3-1*:⟦$H \subseteq$ *carrier M*;
$f \in \{l.\ l \leq (Suc\ n)\} \to H$; $s \in \{l.\ l \leq (Suc\ n)\} \to$ *carrier R*;
$(f\ (Suc\ n)) \notin f\ `(\{l.\ l \leq (Suc\ n)\} - \{Suc\ n\})$⟧ $\Longrightarrow$
$\exists g\ m\ t.\ g \in \{l.\ l \leq (m::nat)\} \to H \wedge$
inj-on g $\{l.\ l \leq (m::nat)\} \wedge$
$t \in \{l.\ l \leq (m::nat)\} \to$ *carrier R* $\wedge$
l-comb R M (*Suc n*) *s f* =
l-comb R M m t g $\wedge$ $t\ m = s\ (Suc\ n) \wedge g\ m = f\ (Suc\ n)$
apply (*cut-tac sc-Ring*,
frule Ring.whole-ideal)
apply (*simp add*:*Nset-pre1*)
apply (*subst l-comb-Suc*[*of H carrier R s n f*], *assumption*+)
apply (*frule func-pre*[*of - - H*], *frule func-pre*[*of - - carrier R*])
apply (*frule unique-expression2*[*of H f n s*], *assumption*+)
apply ((*erule exE*)+, (*erule conjE*)+, *simp*,
thin-tac l-comb R M n s f = *l-comb R M m t g*)
apply (*frule-tac f* = *g* **and** *n* = *m* **and** *A* = *H* **and** $g = \lambda k{\in}\{0::nat\}.\ f\ (Suc\ n)$
and *m* = *0* **and** *B* = *H* **in** *jointfun-hom0*,
simp add:*Pi-def*, *simp*)
apply (*frule-tac f* = *t* **and** *n* = *m* **and** *A* = *carrier R* **and**
g = $\lambda k{\in}\{0::nat\}.\ s\ (Suc\ n)$ **and** *m* = *0* **and** *B* = *carrier R* **in**
jointfun-hom0,
simp add:*Pi-def*, *simp*)
apply (*subgoal-tac inj-on* (*jointfun m g 0* $(\lambda k{\in}\{0\}.\ f\ (Suc\ n))$)
$\{l.\ l \leq Suc\ m\}$,
subgoal-tac l-comb R M m t g $\pm$ *s* (*Suc n*) $\cdot_s$ *f* (*Suc n*) =
l-comb R M (*Suc m*) (*jointfun m t 0* $(\lambda k{\in}\{0\}.\ s\ (Suc\ n))$)
(*jointfun m g 0* $(\lambda k{\in}\{0\}.\ f\ (Suc\ n))$),
subgoal-tac (*jointfun m t 0* $(\lambda k{\in}\{0\}.\ s\ (Suc\ n))$) (*Suc m*) = *s* (*Suc n*) $\wedge$
(*jointfun m g 0* $(\lambda k{\in}\{0\}.\ f\ (Suc\ n))$) (*Suc m*) = *f* (*Suc n*),
simp, *blast*)
apply (*simp add*:*jointfun-def sliden-def*)
apply (*frule-tac s* = *t* **and** *n* = *m* **and** *f* = *g* **and** *t* = $\lambda k{\in}\{0\}.\ s\ (Suc\ n)$ **and**
m = *0* **and** *g* = $\lambda k{\in}\{0\}.\ f\ (Suc\ n)$ **in** *l-comb-jointfun-jj*[*of H*
carrier R], *assumption*+,
simp add:*Pi-def*, *simp*,
simp add:*Pi-def*)
apply (*simp add*:*l-comb-def*, *simp add*:*jointfun-def sliden-def*)
apply (*thin-tac jointfun m g 0* $(\lambda k{\in}\{0\}.\ f\ (Suc\ n)) \in \{l.\ l \leq Suc\ m\} \to H$,
thin-tac jointfun m t 0 $(\lambda k{\in}\{0\}.\ s\ (Suc\ n)) \in \{l.\ l \leq Suc\ m\} \to$ *carrier R*,
thin-tac t $\in \{j.\ j \leq m\} \to$ *carrier R*,
thin-tac s $\in \{j.\ j \leq n\} \to$ *carrier R*)
apply (*rule-tac f* = *g* **and** *n* = *m* **and** *b* = *f* (*Suc n*) **and** *B* = *H* **in** *jointfun-inj*,
assumption+)
apply (*simp add*:*bij-to-def*)

apply (*unfold bij-to-def*, *frule conjunct1*, *fold bij-to-def*,
simp add:*surj-to-def*)
done

lemma (**in** *Module*) *unique-expression3-2*:⟦$H \subseteq$ *carrier* M;
$f \in \{k.\ k \leq (Suc\ n)\} \to H$; $s \in \{k.\ k \leq (Suc\ n)\} \to$ *carrier* R;
$l \leq (Suc\ n)$; $(f\ l) \notin f\ `\ (\{k.\ k \leq (Suc\ n)\} - \{l\})$; $l \neq Suc\ n$⟧ $\Longrightarrow$
$\exists g\ m\ t.\ g \in \{l.\ l \leq (m::nat)\} \to H \wedge$ *inj-on* $g\ \{l.\ l \leq (m::nat)\} \wedge$
$t \in \{l.\ l \leq m\} \to$ *carrier* $R \wedge$
l-comb $R\ M\ (Suc\ n)\ s\ f =$ *l-comb* $R\ M\ m\ t\ g \wedge$
$t\ m = s\ l \wedge g\ m = f\ l$
apply (*cut-tac sc-Ring*,
frule Ring.whole-ideal)
apply (*subst l-comb-transpos1*[*of carrier R H s n f l*], *assumption*+,
rule noteq-le-less[*of l Suc n*], *assumption*+)
apply (*cut-tac unique-expression3-1*[*of H cmp f* (*transpos l* (*Suc n*)) *n*
cmp s (*transpos l* (*Suc n*))])
apply ((*erule exE*)+, (*erule conjE*)+, *simp*)
apply (*subgoal-tac* $t\ m = s\ l \wedge g\ m = f\ l$, *blast*)
apply (*thin-tac l-comb R M* (*Suc n*) (*cmp s* (*transpos l* (*Suc n*)))
(*cmp f* (*transpos l* (*Suc n*))) = *l-comb R M m t g*)
apply (*simp add*:*cmp-def*)
apply (*subst transpos-ij-2*[*of l Suc n Suc n*], *simp*+,
subst transpos-ij-2[*of l Suc n Suc n*], *simp*+)
apply (*rule Pi-I*, *simp add*:*cmp-def*,
frule-tac $l = x$ **in** *transpos-mem*[*of l Suc n Suc n*], *simp*,
assumption+, *simp add*:*Pi-def*)
apply (*rule Pi-I*, *simp add*:*cmp-def*,
frule-tac $l = x$ **in** *transpos-mem*[*of l Suc n Suc n*], *simp*,
assumption+, *simp add*:*Pi-def*)
apply (*frule-tac* $i = l$ **and** $n = Suc\ n$ **and** $j = Suc\ n$ **in** *transpos-hom*,
simp, *assumption*)
apply (*frule cmp-fun-sub-image*[*of transpos l* (*Suc n*) $\{i.\ i \leq Suc\ n\}$
$\{i.\ i \leq Suc\ n\}$ *f H* $\{l.\ l \leq Suc\ n\} - \{Suc\ n\}$], *assumption*+)
apply (*rule subsetI*, *simp*)
apply *simp*
apply (*frule-tac* $i = l$ **and** $n = Suc\ n$ **and** $j = Suc\ n$ **in** *transpos-inj*,
simp, *assumption*+)
apply (*subst injfun-elim-image*[*of transpos l* (*Suc n*) $\{i.\ i \leq Suc\ n\}$
$\{i.\ i \leq Suc\ n\}$ *Suc n*], *assumption*+, *simp*)
apply (*thin-tac cmp f* (*transpos l* (*Suc n*)) ‘ ($\{l.\ l \leq Suc\ n\}$ −
$\{Suc\ n\}$) = *f* ‘ *transpos l* (*Suc n*) ‘ ($\{l.\ l \leq Suc\ n\} - \{Suc\ n\}$))
apply (*frule-tac* $i = l$ **and** $n = Suc\ n$ **and** $j = Suc\ n$ **in**
transpos-surjec, *simp*, *assumption*+)
apply (*simp add*:*surj-to-def cmp-def*)
apply (*simp add*:*transpos-ij-2*)
done

lemma (**in** *Module*) *unique-expression3*:
  $[\![H \subseteq carrier\ M;\ f \in \{k.\ k \leq (Suc\ n)\} \to H;$
    $s \in \{k.\ k \leq (Suc\ n)\} \to carrier\ R;\ l \leq (Suc\ n);$
   $(f\ l) \notin f\ `\ (\{k.\ k \leq (Suc\ n)\} - \{l\})]\!] \Longrightarrow$
  $\exists\, g\ m\ t.\ g \in \{k.\ k \leq (m::nat)\} \to H \wedge$
      inj-on $g\ \{k.\ k \leq m\} \wedge$
      $t \in \{k.\ k \leq m\} \to carrier\ R \wedge$
      l-comb $R\ M\ (Suc\ n)\ s\ f = l\text{-}comb\ R\ M\ m\ t\ g \wedge t\ m = s\ l \wedge g\ m = f\ l$
apply (*case-tac* $l = Suc\ n$, *simp*)
 apply (*cut-tac unique-expression3-1*[*of* $H\ f\ n\ s$], *blast*,
      assumption+)
 apply (*rule unique-expression3-2*[*of* $H\ f\ n\ s\ l$], *assumption*+)
done

lemma (**in** *Module*) *unique-expression4*:*free-generator* $R\ M\ H \Longrightarrow$
    $f \in \{k.\ k \leq (n::nat)\} \to H \wedge$ *inj-on* $f\ \{k.\ k \leq n\} \wedge$
    $s \in \{k.\ k \leq n\} \to carrier\ R \wedge$ *l-comb* $R\ M\ n\ s\ f \neq \mathbf{0}\ \longrightarrow$
$(\exists\, m\ g\ t.\ (g \in \{k.\ k \leq m\} \to H) \wedge$ *inj-on* $g\ \{k.\ k \leq m\} \wedge$
      $(g\ `\ \{k.\ k \leq m\} \subseteq f\ `\ \{k.\ k \leq n\}) \wedge (t \in \{k.\ k \leq m\} \to carrier\ R) \wedge$
      $(\forall\, l \in \{k.\ k \leq m\}.\ t\ l \neq \mathbf{0}_R) \wedge l\text{-}comb\ R\ M\ n\ s\ f = l\text{-}comb\ R\ M\ m\ t\ g)$
apply (*cut-tac sc-Ring*)
apply (*frule free-generator-sub*[*of H*])
apply (*induct-tac n*)
 apply (*rule impI*, (*erule conjE*)+)
 apply (*frule has-free-generator-nonzeroring*[*of H*])
  apply (*frule Ring.whole-ideal*,
      frule-tac $s = s$ **and** $n = 0$ **and** $f = f$ **in**
        l-comb-mem-linear-span[*of carrier R H*], *assumption*+)
  apply *blast*
 apply (*simp add*:*l-comb-def*)
 apply (*subgoal-tac* $f \in \{j.\ j \leq (0::nat)\} \to H \wedge$
      inj-on $f\ \{j.\ j \leq 0\} \wedge f\ `\ \{j.\ j \leq 0\} \subseteq f\ `\ \{0\} \wedge$
      $s \in \{j.\ j \leq 0\} \to carrier\ R \wedge (\forall\, l \leq 0.\ s\ l \neq \mathbf{0}_R) \wedge$
      $s\ 0 \cdot_s (f\ 0) = \Sigma_e\ M\ (\lambda j.\ s\ j \cdot_s (f\ j))\ 0$,
      (*erule conjE*)+, *blast*)
 apply *simp*
 apply (*rule contrapos-pp*, *simp*+)
 apply (*cut-tac* $m = f\ 0$ **in** *sc-0-m*,
       simp add:*Pi-def subsetD*, *simp*)

apply (*rule impI*) **apply** (*erule conjE*)+
 apply (*frule func-pre*[*of - - H*],
      frule-tac $f = f$ **and** $A = \{k.\ k \leq Suc\ n\}$ **and** *?A1.0* $= \{k.\ k \leq n\}$ **in**
      restrict-inj, *rule subsetI*, *simp*,
      frule func-pre[*of - - carrier R*], *simp*)
 apply (*frule Ring.whole-ideal*)
 apply (*frule free-generator-sub*[*of H*],

simp add:*l-comb-Suc*[*of H carrier R s - f*])

apply (*case-tac s* (*Suc n*) = $\mathbf{0}_R$, *simp*)
apply (*frule-tac x* = *Suc n* **and** *f* = *f* **and** *A* = {*k*. *k* ≤ *Suc n*} **and**
B = *H* **in** *funcset-mem*, *simp*,
frule-tac c = *f* (*Suc n*) **in** *subsetD*[*of H carrier M*], *simp*)
apply (*frule-tac m* = *f* (*Suc n*) **in** *sc-0-m*, *simp*)
apply (*frule-tac n* = *n* **in** *l-comb-mem*[*of carrier R H s - f*],
assumption+, *simp add*:*ag-r-zero*)
apply ((*erule exE*)+, (*erule conjE*)+)
apply (*frule-tac f* = *f* **and** *A* = {*k*. *k* ≤ *Suc n*} **and** *B* = *H* **and**
?A1.0 = {*k*. *k* ≤ *n*} **and** *?A2.0* = {*k*. *k* ≤ *Suc n*} **in** *im-set-mono*,
rule subsetI, *simp*, *simp*,
frule-tac A = *g* ‘ {*k*. *k* ≤ *m*} **and** *B* = *f* ‘ {*k*. *k* ≤ *n*} **and**
C = *f* ‘ {*k*. *k* ≤ *Suc n*} **in** *subset-trans*, *assumption*+)
apply *blast*

apply (*case-tac l-comb R M n s f* = $\mathbf{0}_M$, *simp*,
frule-tac x = *Suc n* **and** *f* = *s* **and** *A* = {*k*. *k* ≤ *Suc n*} **and**
B = *carrier R* **in** *funcset-mem*, *simp*,
frule-tac x = *Suc n* **and** *f* = *f* **and** *A* = {*k*. *k* ≤ *Suc n*} **and**
B = *H* **in** *funcset-mem*, *simp*,
frule-tac c = *f* (*Suc n*) **in** *subsetD*[*of H carrier M*], *assumption*+,
frule-tac a = *s* (*Suc n*) **and** *m* = *f* (*Suc n*) **in** *sc-mem*, *assumption*+,
simp add:*ag-l-zero*)
apply (*subgoal-tac* (λ*j*∈{*0*::*nat*}. *f* (*Suc n*)) ∈ {*j*. *j* ≤ (*0*::*nat*)} → *H* ∧
inj-on (λ*j*∈{*0*::*nat*}. *f* (*Suc n*)) {*j*. *j* ≤ (*0*::*nat*)} ∧
(λ*j*∈{*0*::*nat*}. *f* (*Suc n*)) ‘ {*j*. *j* ≤ (*0*::*nat*)} ⊆ *f* ‘ {*k*. *k* ≤ (*Suc n*)} ∧
(λ*j*∈{*0*::*nat*}. *s* (*Suc n*))∈ {*k*. *k* ≤ *0*} → *carrier R* ∧
(∀ *l*≤*0*. (λ*j*∈{*0*::*nat*}. *s* (*Suc n*)) *l* ≠ $\mathbf{0}_R$) ∧
s (*Suc n*) ·$_s$ *f* (*Suc n*) =
l-comb R M 0 (λ*j*∈{*0*::*nat*}. *s* (*Suc n*)) (λ*j*∈{*0*::*nat*}. *f* (*Suc n*)))
apply ((*erule conjE*)+, *blast*)
apply *simp*
apply (*simp add*:*l-comb-def*)

apply *simp*
apply ((*erule exE*)+, (*erule conjE*)+, *erule exE*, (*erule conjE*)+, *simp*)
apply (*thin-tac l-comb R M m t g* ≠ $\mathbf{0}$,
thin-tac l-comb R M m t g ± *s* (*Suc n*) ·$_s$ *f* (*Suc n*) ≠ $\mathbf{0}$,
thin-tac l-comb R M n s f = *l-comb R M m t g*)
apply (*frule-tac f* = *g* **and** *n* = *m* **and** *A* = *H* **and** *g* = λ*j*∈{*0*::*nat*}. *f* (*Suc n*)
and *m* = *0* **and** *B* = *H* **in** *jointfun-hom*,
rule Pi-I, *simp add*:*Pi-def*,
frule-tac f = *t* **and** *n* = *m* **and** *A* = *carrier R* **and**
g = λ*j*∈{*0*::*nat*}. *s* (*Suc n*) **and** *m* = *0* **and** *B* = *carrier R* **in**
jointfun-hom, *simp add*:*Pi-def*, *simp*)
apply (*subgoal-tac inj-on* (*jointfun m g 0* (λ*j*∈{*0*}. *f* (*Suc n*)))
{*k*. *k* ≤ *Suc m*} ∧

(*jointfun m g 0* ($\lambda j{\in}\{0\}$. *f* (*Suc n*))) ‘ $\{k.\ k \leq Suc\ m\} \subseteq f$ ‘ $\{k.\ k \leq Suc\ n\} \wedge$
$(\forall\, l \leq (Suc\ m).\ (jointfun\ m\ t\ 0\ (\lambda j{\in}\{0\}.\ s\ (Suc\ n)))\ l \neq \mathbf{0}_R) \wedge$
l-comb R M m t g $\pm$ *s* (*Suc n*) $\cdot_s$ *f* (*Suc n*) =
l-comb R M (*Suc m*) (*jointfun m t 0* ($\lambda j{\in}\{0\}$. *s* (*Suc n*)))
(*jointfun m g 0* ($\lambda j{\in}\{0\}$. *f* (*Suc n*))))
apply (*erule conjE*)+ **apply** *blast*

apply (*rule conjI*)
apply (*rule-tac f* = *g* **and** *n* = *m* **and** *b* = *f* (*Suc n*) **and** *B* = *H* **in**
jointfun-inj, *assumption*+)
apply (*rule contrapos-pp*, *simp*+)
apply (*frule-tac c* = *f* (*Suc n*) **and** *A* = *g* ‘ $\{k.\ k \leq m\}$ **and**
B = *f* ‘ $\{k.\ k \leq n\}$ **in** *subsetD*, *assumption*+)

apply (*thin-tac inj-on f* $\{k.\ k \leq n\}$,
thin-tac g ‘ $\{k.\ k \leq m\} \subseteq f$ ‘ $\{k.\ k \leq n\}$,
thin-tac f (*Suc n*) $\in g$ ‘ $\{j.\ j \leq m\}$, *simp add*:*image-def*,
erule exE, *erule conjE*)
apply (*simp add*:*inj-on-def*,
drule-tac a = *Suc n* **in** *forall-spec*, *simp*,
thin-tac $\forall\, x{\leq}m.\ \forall\, y{\leq}m.\ g\ x = g\ y \longrightarrow x = y$,
thin-tac $\forall\, l{\leq}m.\ t\ l \neq \mathbf{0}_R$,
drule-tac a = *x* **in** *forall-spec*, *simp*, *simp*)

apply (*rule conjI*, *rule subsetI*)
apply (*simp add*:*image-def*, *erule exE*, *erule conjE*)
apply (*case-tac xa* = *Suc m*, *simp add*:*jointfun-def sliden-def*)
apply (*cut-tac n* = *Suc n* **in** *Nat.le-refl*, *blast*)
apply (*frule-tac m* = *xa* **and** *n* = *Suc m* **in** *noteq-le-less*, *assumption*,
thin-tac xa $\leq$ *Suc m*,
frule-tac x = *xa* **and** *n* = *Suc m* **in** *less-le-diff*,
thin-tac xa < *Suc m*, *simp*,
thin-tac jointfun m g 0 ($\lambda j{\in}\{0\}$. *f* (*Suc n*)) $\in \{j.\ j \leq Suc\ m\} \to H$,
thin-tac jointfun m t 0 ($\lambda j{\in}\{0\}$. *s* (*Suc n*)) $\in \{j.\ j \leq Suc\ m\} \to$ *carrier R*,
simp add:*jointfun-def*)
apply (*subgoal-tac g xa* $\in \{y.\ \exists\, x{\leq}n.\ y = f\, x\}$, *simp*, *erule exE*)
apply (*erule conjE*, *frule-tac i* = *xb* **and** *j* = *n* **and** *k* = *Suc n* **in**
le-trans, *simp*, *blast*)
apply (*rule-tac c* = *g xa* **and** *A* = $\{y.\ \exists\, x{\leq}m.\ y = g\ x\}$ **and**
B = $\{y.\ \exists\, x{\leq}n.\ y = f\, x\}$ **in** *subsetD*, *assumption*+,
simp, *blast*)
apply (*rule conjI*, *rule allI*, *rule impI*)
apply (*case-tac l* = *Suc m*, *simp add*:*jointfun-def sliden-def*)
apply (*frule-tac m* = *l* **and** *n* = *Suc m* **in** *noteq-le-less*, *assumption*,
thin-tac l $\leq$ *Suc m*,
frule-tac x = *l* **and** *n* = *Suc m* **in** *less-le-diff*,
thin-tac l < *Suc m*, *simp*,
thin-tac jointfun m g 0 ($\lambda j{\in}\{0\}$. *f* (*Suc n*)) $\in \{j.\ j \leq Suc\ m\} \to H$,
thin-tac jointfun m t 0 ($\lambda j{\in}\{0\}$. *s* (*Suc n*)) $\in \{j.\ j \leq Suc\ m\} \to$ *carrier R*,

simp add:jointfun-def)
apply (*simp add:l-comb-def*,
subst l-comb-jointfun-jj[*of H carrier R*], *assumption*+,
simp add:Pi-def,
simp add:Pi-def)
apply (*simp add:jointfun-def sliden-def*)
done

lemma (**in** *Module*) *unique-prepression5-0*:⟦*free-generator R M H*;
$f \in \{j.\ j \leq n\} \to H$; *inj-on f* $\{j.\ j \leq n\}$;
$s \in \{j.\ j \leq n\} \to$ *carrier R*; $g \in \{j.\ j \leq m\} \to H$;
inj-on g $\{j.\ j \leq m\}$; $t \in \{j.\ j \leq m\} \to$ *carrier R*;
l-comb R M n s f = *l-comb R M m t g*;$\forall j{\leq}n.\ s\ j \neq \mathbf{0}_R$; $\forall k{\leq}m.\ t\ k \neq \mathbf{0}_R$;
$f\ n \notin g\ `\ \{j.\ j \leq m\}$; $0 < n$⟧ ⟹ *False*
apply (*cut-tac sc-Ring*,
frule Ring.ring-is-ag,
frule Ring.whole-ideal,
frule free-generator-sub[*of H*])
apply (*cut-tac l-comb-Suc*[*of H carrier R s n − Suc 0 f*],
simp,
thin-tac l-comb R M n s f = *l-comb R M* $(n - Suc\ 0)\ s\ f \pm s\ n \cdot_s f\ n$)
apply (*frule free-generator-sub*[*of H*],
frule l-comb-mem[*of carrier R H t m g*], *assumption*+,
frule l-comb-mem[*of carrier R H s n − Suc 0 f*], *assumption*+,
rule func-pre, *simp*, *rule func-pre*, *simp*,
cut-tac sc-mem[*of s n f n*])
apply (*frule ag-pOp-closed*[*of l-comb R M* $(n - Suc\ 0)\ s\ f\ s\ n \cdot_s f\ n$],
assumption+,
frule ag-mOp-closed[*of l-comb R M* $(n - Suc\ 0)\ s\ f$])
apply (*frule ag-pOp-add-l*[*of l-comb R M m t g l-comb R M* $(n - Suc\ 0)\ s\ f \pm s\ n \cdot_s f\ n -_a$ (*l-comb R M* $(n - Suc\ 0)\ s\ f$)], *assumption*+,
thin-tac l-comb R M m t g = *l-comb R M* $(n - Suc\ 0)\ s\ f \pm s\ n \cdot_s f\ n$)
apply (*simp add:ag-pOp-assoc*[*THEN sym*, *of* $-_a$ (*l-comb R M* $(n - Suc\ 0)\ s\ f$)
l-comb R M $(n - Suc\ 0)\ s\ f\ s\ n \cdot_s f\ n$],
simp add:ag-l-inv1 ag-l-zero)
apply (*cut-tac func-pre*[*of f n − Suc 0 H*],
cut-tac func-pre[*of s n − Suc 0 carrier R*])
apply (*frule linear-span-iOp-closedTr2*[*of carrier R H f n − Suc 0 s*],
assumption+)
apply (*simp*,
thin-tac $-_a$ (*l-comb R M* $(n - Suc\ 0)\ s\ f$) =
l-comb R M $(n - Suc\ 0)$ $(\lambda x{\in}\{j.\ j \leq n - Suc\ 0\}.\ -_{a\,R}\ (s\ x))\ f$)
apply (*subgoal-tac* $(\lambda x{\in}\{j.\ j \leq n - Suc\ 0\}.\ -_{a\,R}\ (s\ x))$
$\in \{j.\ j \leq n - Suc\ 0\} \to$ *carrier R*)
apply (*simp add:l-comb-add*[*THEN sym*, *of carrier R H*
$\lambda x{\in}\{j.\ j \leq n - Suc\ 0\}.\ -_{a\,R}\ (s\ x)\ n - Suc\ 0\ f\ t\ m\ g$],
thin-tac l-comb R M m t g ∈ *carrier M*,
thin-tac l-comb R M $(n - Suc\ 0)\ s\ f \in$ *carrier M*,

thin-tac l-comb R M $(n - Suc\ 0)$ $s\ f \pm s\ n \cdot_s f\ n \in carrier\ M$,
thin-tac l-comb R M $(n - Suc\ 0)$ $(\lambda x{\in}\{j.\ j \leq n - Suc\ 0\}.\ -_{a\,R}\ (s\ x))\ f$
$\in carrier\ M$)
apply (*frule jointfun-hom*[*of f* $n - Suc\ 0$ *H g m H*], *assumption*+,
frule jointfun-hom[*of* $\lambda x{\in}\{j.\ j \leq n - Suc\ 0\}.\ -_{a\,R}\ (s\ x)\ n - Suc\ 0$
carrier R t m carrier R], *assumption*+, *simp*)

apply (*frule im-jointfun*[*of f* $n - Suc\ 0$ *H g m H*], *assumption*+)
apply (*frule unique-expression3-1*[*of H*
jointfun $(n + m)$ (*jointfun* $(n - Suc\ 0)$ *f m g*) *0* $(\lambda x{\in}\{0{::}nat\}.\ (f\ n))$
$n + m$
jointfun $(n + m)$ (*jointfun* $(n - Suc\ 0)$ $(\lambda x{\in}\{j.\ j \leq n - Suc\ 0\}.\ -_{a\,R}\ (s\ x))$
m t) *0* $(\lambda x{\in}\{0{::}nat\}.\ -_{a\,R}\ (s\ n))$])
apply (*rule Pi-I*,
case-tac $x \leq (n + m)$, *simp*,
simp add:*jointfun-def*[*of* $n{+}m$], *simp add*:*Pi-def*,
simp add:*jointfun-def*[*of* $n{+}m$] *sliden-def*, *simp add*:*Pi-def*)
apply (*rule Pi-I*,
case-tac $x \leq (n + m)$, *simp*,
simp add:*jointfun-def*[*of* $n{+}m$], *simp add*:*Pi-def*)
apply (*simp add*:*jointfun-def*[*of* $n{+}m$] *sliden-def*,
frule Ring.ring-is-ag[*of R*], *rule aGroup.ag-mOp-closed*, *assumption*,
simp add:*Pi-def*)
apply (*thin-tac* $s \in \{j.\ j \leq n\} \to carrier\ R$,
thin-tac $t \in \{j.\ j \leq m\} \to carrier\ R$,
thin-tac $\forall j{\leq}n.\ s\ j \neq \mathbf{0}_R$,
thin-tac $\forall k{\leq}m.\ t\ k \neq \mathbf{0}_R$,
thin-tac l-comb R M $(n + m)$
(*jointfun* $(n - Suc\ 0)$ $(\lambda x{\in}\{j.\ j \leq n - Suc\ 0\}.\ -_{a\,R}\ (s\ x))$ *m t*)
(*jointfun* $(n - Suc\ 0)$ *f m g*) =
$s\ n \cdot_s f\ n$,
thin-tac $s \in \{j.\ j \leq n - Suc\ 0\} \to carrier\ R$)
apply (*thin-tac* $(\lambda x{\in}\{j.\ j \leq n - Suc\ 0\}.\ -_{a\,R}\ (s\ x))$
$\in \{j.\ j \leq n - Suc\ 0\} \to carrier\ R$,
thin-tac jointfun $(n - Suc\ 0)$ $(\lambda x{\in}\{j.\ j \leq n - Suc\ 0\}.\ -_{a\,R}\ (s\ x))\ m\ t$
$\in \{j.\ j \leq n + m\} \to carrier\ R$)
apply (*simp add*:*Nset-pre1*,
simp add:*im-jointfunTr1*[*of* $n + m$ *jointfun* $(n - Suc\ 0)$ *f m g 0*
$\lambda x{\in}\{0\}.\ f\ n$],
thin-tac jointfun $(n - Suc\ 0)$ *f m g* $\in \{j.\ j \leq n + m\} \to H$,
thin-tac jointfun $(n - Suc\ 0)$ *f m g* ‘ $\{j.\ j \leq n + m\} =$
f ‘ $\{j.\ j \leq n - Suc\ 0\} \cup g$ ‘ $\{j.\ j \leq m\}$,
simp add:*jointfun-def*[*of* $n{+}m$] *sliden-def*)
apply (*rule contrapos-pp*, *simp*+, *simp add*:*image-def*, *erule exE*,*erule conjE*,
simp add:*inj-on-def*[*of f*],
drule-tac $a = n$ **in** *forall-spec*, *simp*,
thin-tac $\forall xa{\leq}m.\ f\ x \neq g\ xa$,
drule-tac $a = x$ **in** *forall-spec*,
rule-tac $i = x$ **and** $j = n - Suc\ 0$ **and** $k = n$ **in** *Nat.le-trans*,

assumption+, *subst Suc-le-mono*[*THEN sym*], *simp*,
simp,
cut-tac n1 = *x* **and** *m1* = *x* − *Suc 0* **in**
Suc-le-mono[*THEN sym*], *simp*)

defer
apply (*rule Pi-I*, *simp*,
rule aGroup.ag-mOp-closed, *assumption*,
cut-tac *i* = *x* **and** *j* = *n* − *Suc 0* **and** *k* = *n* **in** *Nat.le-trans*,
assumption, *subst Suc-le-mono*[*THEN sym*], *simp*,
simp add:*Pi-def*, *simp*, *simp*, *simp add*:*Pi-def*,
simp add:*Pi-def*,
simp add:*Pi-def subsetD*, *assumption*+, *simp*, *simp*)
apply ((*erule exE*)+, (*erule conjE*)+, *erule exE*, (*erule conjE*)+)
apply (*cut-tac l-comb-Suc*[*of H carrier R jointfun* (*n* + *m*)
(*jointfun* (*n* − *Suc 0*) (λ*x*∈{*j*. *j* ≤ *n* − *Suc 0*}. −$_{aR}$ (*s x*)) *m t*) *0*
(λ*x*∈{*0*}. −$_{aR}$ (*s n*)) *n* + *m*
jointfun (*n* + *m*) (*jointfun* (*n* − *Suc 0*) *f m g*) *0* (λ*x*∈{*0*}. *f n*)],
simp) **apply** (
thin-tac l-comb R M (*Suc* (*n* + *m*))
(*jointfun* (*n* + *m*)
(*jointfun* (*n* − *Suc 0*) (λ*x*∈{*j*. *j* ≤ *n* − *Suc 0*}. −$_{aR}$ (*s x*)) *m t*) *0*
(λ*x*∈{*0*}. −$_{aR}$ (*s n*)))
(*jointfun* (*n* + *m*) (*jointfun* (*n* − *Suc 0*) *f m g*) *0* (λ*x*∈{*0*}. *f n*)) =
l-comb R M ma ta ga)
apply (*subgoal-tac l-comb R M* (*n* + *m*)
(*jointfun* (*n* + *m*)
(*jointfun* (*n* − *Suc 0*) (λ*x*∈{*j*. *j* ≤ *n* − *Suc 0*}. −$_{aR}$ (*s x*)) *m t*) *0*
(λ*x*∈{*0*}. −$_{aR}$ (*s n*)))
(*jointfun* (*n* + *m*) (*jointfun* (*n* − *Suc 0*) *f m g*) *0* (λ*x*∈{*0*}. *f n*)) ±
jointfun (*n* + *m*)
(*jointfun* (*n* − *Suc 0*) (λ*x*∈{*j*. *j* ≤ *n* − *Suc 0*}. −$_{aR}$ (*s x*)) *m t*) *0*
(λ*x*∈{*0*}. −$_{aR}$ (*s n*)) (*Suc* (*n* + *m*)) ·$_s$
jointfun (*n* + *m*) (*jointfun* (*n* − *Suc 0*) *f m g*) *0* (λ*x*∈{*0*}. *f n*)
(*Suc* (*n* + *m*)) = **0**$_M$, *simp*,
thin-tac l-comb R M (*n* + *m*)
(*jointfun* (*n* + *m*)
(*jointfun* (*n* − *Suc 0*) (λ*x*∈{*j*. *j* ≤ *n* − *Suc 0*}. −$_{aR}$ (*s x*)) *m t*) *0*
(λ*x*∈{*0*}. −$_{aR}$ (*s n*)))
(*jointfun* (*n* + *m*) (*jointfun* (*n* − *Suc 0*) *f m g*) *0* (λ*x*∈{*0*}. *f n*)) ±
jointfun (*n* + *m*)
(*jointfun* (*n* − *Suc 0*) (λ*x*∈{*j*. *j* ≤ *n* − *Suc 0*}. −$_{aR}$ (*s x*)) *m t*) *0*
(λ*x*∈{*0*}. −$_{aR}$ (*s n*)) (*Suc* (*n* + *m*)) ·$_s$
jointfun (*n* + *m*) (*jointfun* (*n* − *Suc 0*) *f m g*) *0* (λ*x*∈{*0*}. *f n*)
(*Suc* (*n* + *m*)) =
l-comb R M ma ta ga,
thin-tac l-comb R M (*n* + *m*)
(*jointfun* (*n* − *Suc 0*) (λ*x*∈{*j*. *j* ≤ *n* − *Suc 0*}. −$_{aR}$ (*s x*)) *m t*)
(*jointfun* (*n* − *Suc 0*) *f m g*) =

```
    s n ·s f n,
    thin-tac jointfun (n − Suc 0) f m g ∈ {j. j ≤ n + m} → H,
    thin-tac jointfun (n − Suc 0) (λx∈{j. j ≤ n − Suc 0}. −aR (s x)) m t
     ∈ {j. j ≤ n + m} → carrier R,
    thin-tac jointfun (n − Suc 0) f m g ‘ {j. j ≤ n + m} =
     f ‘ {j. j ≤ n − Suc 0} ∪ g ‘ {j. j ≤ m})
  apply (simp add:jointfun-def[of n+m] sliden-def)
  apply (rotate-tac −3, frule sym, thin-tac 0 = l-comb R M ma ta ga)
   apply (frule-tac s = ta and n = ma and m = ga in unique-expression1[of
H],
      assumption+)
  apply (rotate-tac −1,
      drule-tac x = ma in bspec, simp)
  apply (frule-tac funcset-mem[of s {j. j ≤ n} carrier R n], simp,
      frule sym, thin-tac ta ma = −aR (s n),
      frule aGroup.ag-inv-inv[of R s n], assumption+, simp,
      thin-tac  −aR (s n) = 0R,
      rotate-tac −1, frule sym, thin-tac  −aR 0R = s n,
      simp add:aGroup.ag-inv-zero[of R])

 apply (thin-tac l-comb R M (n + m)
     (jointfun (n + m)
      (jointfun (n − Suc 0) (λx∈{j. j ≤ n − Suc 0}. −aR (s x)) m t) 0
      (λx∈{0}. −aR (s n)))
     (jointfun (n + m) (jointfun (n − Suc 0) f m g) 0 (λx∈{0}. f n)) ±
     jointfun (n + m)
      (jointfun (n − Suc 0) (λx∈{j. j ≤ n − Suc 0}. −aR (s x)) m t) 0
      (λx∈{0}. −aR (s n)) (Suc (n + m)) ·s
     jointfun (n + m) (jointfun (n − Suc 0) f m g) 0 (λx∈{0}. f n)
      (Suc (n + m)) =
     l-comb R M ma ta ga,
     thin-tac ta ma =
     jointfun (n + m)
      (jointfun (n − Suc 0) (λx∈{j. j ≤ n − Suc 0}. −aR (s x)) m t) 0
      (λx∈{0}. −aR (s n)) (Suc (n + m)))
 apply (subst l-comb-jointfun-jj1[of H carrier R], assumption+,
      rule Pi-I, simp,
      rule aGroup.ag-mOp-closed, assumption, simp add:Pi-def,
      simp add:Pi-def)
 apply (simp,
      thin-tac l-comb R M (n + m) (jointfun (n − Suc 0)
     (λx∈{j. j ≤ n − Suc 0}. −aR (s x)) m t) (jointfun (n − Suc 0) f m g) =
      s n ·s f n,
     thin-tac jointfun (n − Suc 0) f m g ∈ {j. j ≤ n + m} → H,
     thin-tac jointfun (n − Suc 0) (λx∈{j. j ≤ n − Suc 0}. −aR (s x)) m t
      ∈ {j. j ≤ n + m} → carrier R,
     thin-tac jointfun (n − Suc 0) f m g ‘ {j. j ≤ n + m} =
     f ‘ {j. j ≤ n − Suc 0} ∪ g ‘ {j. j ≤ m})
 apply (simp add:jointfun-def[of n+m] sliden-def,
```

subst sc-minus-am1[*THEN sym*],
simp add:*Pi-def*, *simp add*:*Pi-def subsetD*,
simp add:*ag-r-inv1*, *simp add*:*free-generator-sub*)
apply (*assumption*+,
rule Pi-I,
case-tac $x \leq n + m$, *simp add*:*jointfun-def*[*of* $n+m$],
simp add:*Pi-def*,
simp add:*jointfun-def*[*of* $n+m$] *sliden-def*,
rule aGroup.ag-mOp-closed, *assumption*, *simp add*:*Pi-def*,
rule Pi-I, *simp*,
case-tac $x \leq n+m$, *simp add*:*jointfun-def*[*of* $n+m$],
simp add:*Pi-def*,
simp add:*jointfun-def*[*of* $n+m$] *sliden-def*,
simp add:*Pi-def*)
done

lemma (**in** *Module*) *unique-expression5*:⟦*free-generator R M H*;
$f \in \{j.\ j \leq (n::nat)\} \to H$; *inj-on f* $\{j.\ j \leq n\}$;
$s \in \{j.\ j \leq n\} \to$ *carrier R*; $g \in \{j.\ j \leq (m::nat)\} \to H$;
inj-on g $\{j.\ j \leq m\}$; $t \in \{j.\ j \leq m\} \to$ *carrier R*;
l-comb R M n s f = *l-comb R M m t g*;
$\forall j \in \{j.\ j \leq n\}.\ s\ j \neq \mathbf{0}_R$; $\forall k \in \{j.\ j \leq m\}.\ t\ k \neq \mathbf{0}_R$⟧ ⟹
f ‘ $\{j.\ j \leq n\} \subseteq g$ ‘ $\{j.\ j \leq m\}$
apply (*cut-tac sc-Ring*, *frule Ring.ring-is-ag*[*of R*],
frule Ring.whole-ideal,
frule free-generator-sub[*of H*])
apply (*rule contrapos-pp*, *simp*+, *simp add*:*subset-eq*)
apply (*erule exE*, *erule conjE*)
apply (*case-tac* $n = 0$, *simp*)
apply (*frule-tac* $f = t$ **and** $n = m$ **and** $A =$ *carrier R* **and**
$g = \lambda k{\in}\{0::nat\}.\ -_{a\,R}\ (s\ 0)$ **and** $m = 0$ **and** $B =$ *carrier R* **in**
jointfun-hom0,
simp add:*Pi-def*,
rule aGroup.ag-mOp-closed, *assumption*, *simp add*:*Pi-def*,
frule-tac $f = g$ **and** $n = m$ **and** $A = H$ **and**
$g = \lambda k{\in}\{0::nat\}.\ (f\ 0)$ **and** $m = 0$ **and** $B = H$ **in**
jointfun-hom0,
simp add:*Pi-def subsetD*,
simp)
apply (*frule sym*, *thin-tac l-comb R M 0 s f* = *l-comb R M m t g*)
apply (*frule-tac* $n = 0$ **in** *l-comb-mem*[*of carrier R H s - f*],
simp add:*free-generator-sub*, *simp*+,
frule-tac $n = m$ **in** *l-comb-mem*[*of carrier R H t - g*],
simp add:*free-generator-sub*, *assumption*+)
apply (*simp add*:*ag-eq-diffzero*[*of l-comb R M m t g l-comb R M 0 s f*],
simp add:*l-comb-def*[*of R M 0 s f*],
frule free-generator-sub[*of H*],
frule-tac $c = f\ 0$ **in** *subsetD*[*of H carrier M*], *assumption*+,
simp add:*sc-minus-am1*)

apply (*subgoal-tac l-comb R M m t g* $\pm$ $(-_{aR}\ (s\ 0))\ \cdot_s\ f\ 0 =$
l-comb R M (*Suc m*) (*jointfun m t 0* ($\lambda k{\in}\{0\}$. $(-_{aR}\ (s\ 0))$))
(*jointfun m g 0* ($\lambda k{\in}\{0\}$. *f 0*)), *simp*)
apply (*frule-tac f* = *g* **and** *n* = *m* **and** *B* = *H* **and** *b* = *f 0* **in** *jointfun-inj*,
assumption+)
apply (*frule unique-expression1*[*of H jointfun m t 0* ($\lambda k{\in}\{0\}$. $(-_{aR}\ (s\ 0))$)
Suc m jointfun m g 0 ($\lambda k{\in}\{0\}$. *f 0*)], *assumption*+)
apply (*frule-tac x* = *Suc m* **in** *bspec*, *simp*,
thin-tac $\forall j{\in}\{j.\ j \le Suc\ m\}$. *jointfun m t 0* ($\lambda k{\in}\{0\}$. $-_{aR}\ (s\ 0)$) *j*
= $\mathbf{0}_R$)
apply (*simp add*:*jointfun-def sliden-def*)
apply (*frule aGroup.ag-inv-inv*[*THEN sym*, *of R s 0*], *assumption*,
simp add:*aGroup.ag-inv-zero*)

apply (*thin-tac l-comb R M m t g* $\pm$ $(-_{aR}\ (s\ 0))\ \cdot_s\ f\ 0 = \mathbf{0}$,
simp del:*nsum-suc add*:*l-comb-def*)
apply (*cut-tac l-comb-jointfun-jj*[*of H carrier R t m g* $\lambda k{\in}\{0\}$. $-_{aR}\ (s\ 0)$
0 $\lambda k{\in}\{0\}$. *f 0*], *simp*,
thin-tac $\Sigma_e\ M\ (\lambda j.\ jointfun\ m\ t\ 0\ (\lambda k{\in}\{0\}.\ -_{aR}\ (s\ 0))\ j\ \cdot_s$
jointfun m g 0 ($\lambda k{\in}\{0\}$. *f 0*) *j*) *m* =
$\Sigma_e\ M\ (\lambda j.\ t\ j\ \cdot_s\ g\ j)\ m$,
simp add:*jointfun-def sliden-def*, *simp add*:*free-generator-sub*,
assumption+,
rule Pi-I, *simp*,
rule aGroup.ag-mOp-closed, *assumption*+,
simp)
apply (*case-tac x* = *n*, *simp*,
rule unique-prepression5-0[*of H f n s g m t*], *assumption*+)
apply (*frule-tac j* = *x* **in** *l-comb-transpos1*[*of carrier R H s n* $-$ *Suc 0 f*],
rule subsetI, *simp*,
simp+,
rotate-tac -1, *frule sym*,
thin-tac l-comb R M m t g =
l-comb R M n (*cmp s* (*transpos x n*)) (*cmp f* (*transpos x n*)),
frule-tac i = *x* **and** *n* = *n* **and** *j* = *n* **in** *transpos-hom*, *simp*,
assumption,
frule-tac i = *x* **and** *n* = *n* **and** *j* = *n* **in** *transpos-inj*, *simp*,
assumption+,
rule-tac f = *cmp f* (*transpos x n*) **and** *s* = *cmp s* (*transpos x n*) **in**
unique-prepression5-0[*of H - n - g m t*], *assumption*+,
simp add:*cmp-fun*, *simp add*:*cmp-fun*, *simp add*:*cmp-inj*,
simp add:*cmp-fun*, *assumption*+,
rule allI, *rule impI*, *simp add*:*cmp-def*,
frule-tac i = *x* **and** *n* = *n* **and** *j* = *n* **and** *l* = *j* **in** *transpos-mem*,
simp, *assumption*+, *blast*, *assumption*)
apply (*simp add*:*cmp-def transpos-ij-2*)
apply *simp*
done

lemma (**in** *Module*) *unique-expression6*:⟦*free-generator R M H*;
$f \in \{j.\ j \leq (n::nat)\} \to H$; *inj-on f* $\{j.\ j \leq n\}$;
$s \in \{j.\ j \leq n\} \to$ *carrier R*;
$g \in \{j.\ j \leq (m::nat)\} \to H$; *inj-on g* $\{j.\ j \leq m\}$;
$t \in \{j.\ j \leq m\} \to$ *carrier R*;
l-comb R M n s f = *l-comb R M m t g*;
$\forall j{\in}\{j.\ j \leq n\}.\ s\ j \neq \mathbf{0}_R; \forall k{\in}\ \{j.\ j \leq m\}.\ t\ k \neq \mathbf{0}_R\rrbracket \Longrightarrow$
$f\ `\{j.\ j \leq n\} = g\ `\ \{j.\ j \leq m\}$
apply (*rule equalityI*)
apply (*rule-tac* $H = H$ **and** $f = f$ **and** $n = n$ **and** $s = s$ **and** $g = g$ **and** $m = m$ **and**
$t = t$ **in** *unique-expression5*, *assumption*+)
apply (*rule-tac* $H = H$ **and** $f = g$ **and** $n = m$ **and** $s = t$ **and** $g = f$ **and** $m = n$ **and**
$t = s$ **in** *unique-expression5*, *assumption*+)
apply (*rule sym*, *assumption*, *blast*, *blast*)
done

lemma (**in** *Module*) *unique-expression7-1*:⟦*free-generator R M H*;
$f \in \{j.\ j \leq (n::nat)\} \to H$; *inj-on f* $\{j.\ j \leq n\}$;
$s \in \{j.\ j \leq n\} \to$ *carrier R*;
$g \in \{j.\ j \leq (m::nat)\} \to H$; *inj-on g* $\{j.\ j \leq m\}$;
$t \in \{j.\ j \leq m\} \to$ *carrier R*;
l-comb R M n s f = *l-comb R M m t g*;
$\forall j \in \{j.\ j \leq n\}.\ s\ j \neq \mathbf{0}_R; \forall k{\in}\{j.\ j \leq m\}.\ t\ k \neq \mathbf{0}_R\rrbracket \Longrightarrow n = m$
apply (*frule-tac* $A = \{j.\ j \leq n\}$ **and** $f = f$ **in** *card-image*,
frule-tac $A = \{j.\ j \leq m\}$ **and** $f = g$ **in** *card-image*)
apply (*frule-tac* $H = H$ **and** $f = f$ **and** $n = n$ **and** $s = s$ **and** $g = g$ **and** $t = t$ **and**
$m = m$ **in** *unique-expression6*, *assumption*+)
apply (*rotate-tac* -3, *frule sym*,
thin-tac card $(f\ `\ \{j.\ j \leq n\}) =$ *card* $(\{j.\ j \leq n\})$)
apply *simp*
done

lemma (**in** *Module*) *unique-expression7-2*:⟦*free-generator R M H*;
$f \in \{j.\ j \leq (n::nat)\} \to H$; *inj-on f* $\{j.\ j \leq n\}$;
$s \in \{j.\ j \leq n\} \to$ *carrier R*; $t \in \{j.\ j \leq n\} \to$ *carrier R*;
l-comb R M n s f = *l-comb R M n t f*⟧ $\Longrightarrow (\forall l \in \{j.\ j \leq n\}.\ s\ l = t\ l)$
apply (*cut-tac sc-Ring*, *frule Ring.whole-ideal*)
apply (*frule free-generator-sub*[*of H*])
apply (*frule l-comb-mem*[*of carrier R H s n f*], *assumption*+,
frule l-comb-mem[*of carrier R H t n f*], *assumption*+)
apply (*simp add*:*ag-eq-diffzero*[*of l-comb R M n s f l-comb R M n t f*])
apply (*simp add*:*linear-span-iOp-closedTr2*[*of carrier R H f n t*])
apply (*frule l-comb-add1*[*THEN sym*, *of carrier R H f n s* $\lambda j{\in}\{k.\ k \leq n\}.\ -_a R\ (t\ j)$],
assumption+)
apply (*rule Pi-I*)

apply (*simp*, *frule Ring.ring-is-ag*[*of R*],
rule aGroup.ag-mOp-closed[*of R*], *simp add*:*Pi-def*)
apply (*simp add*:*Pi-def*)
apply *simp*
apply (*frule-tac* $s = \lambda x \in \{x.\ x \leq n\}.\ s\ x \pm_R$ (*if* $x \leq n$ *then* $-_{aR}\ (t\ x)$ *else undefined*) **in** *unique-expression1*[*of H - n f*], *assumption*+)
apply (*rule Pi-I*, *simp*)
apply (*frule Ring.ring-is-ag*[*of R*], *rule aGroup.ag-pOp-closed*, *assumption*,
simp add:*Pi-def*,
rule aGroup.ag-mOp-closed, *assumption*,
simp add:*Pi-def*, *assumption*+)
apply (*rule allI*, *rule impI*)
apply (*subst aGroup.ag-eq-diffzero*[*of R*],
simp add:*Ring.ring-is-ag*,
simp add:*Pi-def*, *simp add*:*Pi-def*)
apply (*drule-tac* $x = l$ **in** *bspec*, *simp*)
apply *simp*
done

end

theory *Algebra8* **imports** *Algebra7* **begin**

5.4 nsum and Generators (continued)

lemma (**in** *Module*) *unique-expression-last*: $[\![$*free-generator R M H*;
$f \in \{j.\ j \leq Suc\ n\} \to H$; $s \in \{j.\ j \leq Suc\ n\} \to carrier\ R$;
$g \in \{j.\ j \leq Suc\ n\} \to H$; $t \in \{j.\ j \leq Suc\ n\} \to carrier\ R$;
l-comb R M (*Suc n*) *s f* = *l-comb R M* (*Suc n*) *t g*;
inj-on f $\{j.\ j \leq Suc\ n\}$; *inj-on g* $\{j.\ j \leq Suc\ n\}$;
f (*Suc n*) = *g* (*Suc n*)$]\!] \Longrightarrow s\ (Suc\ n) = t\ (Suc\ n)$
apply (*cut-tac sc-Ring*, *frule Ring.whole-ideal*, *frule free-generator-sub*)
apply (*frule l-comb-mem*[*of carrier R H s Suc n f*], *assumption*+,
frule l-comb-mem[*of carrier R H t Suc n g*], *assumption*+)
apply (*frule ag-r-inv1*[*of l-comb R M* (*Suc n*) *t g*],
simp only:*linear-span-iOp-closedTr2*,
thin-tac l-comb R M (*Suc n*) *t g* $\in$ *carrier M*,
frule sym,
thin-tac l-comb R M (*Suc n*) *s f* = *l-comb R M* (*Suc n*) *t g*,
simp,
thin-tac l-comb R M (*Suc n*) *t g* = *l-comb R M* (*Suc n*) *s f*,
subgoal-tac $(\lambda x \in \{j.\ j \leq Suc\ n\}.\ -_{aR}\ (t\ x)) \in$
$\{j.\ j \leq Suc\ n\} \to carrier\ R$,
simp add:*l-comb-Suc*,
frule funcset-mem[*of s* $\{j.\ j \leq Suc\ n\}$ *carrier R Suc n*],
simp,
frule funcset-mem[*of t* $\{j.\ j \leq Suc\ n\}$ *carrier R Suc n*],
simp,

frule funcset-mem[*of g* {*j*. *j* ≤ *Suc n*} *H Suc n*],
simp,
frule subsetD[*of H carrier M g* (*Suc n*)], *assumption*+,
frule func-pre[*of s*], *frule func-pre*[*of t*], *frule func-pre*[*of f*],
frule func-pre[*of g*],
frule func-pre[*of* λ*x*∈{*j*. *j* ≤ *Suc n*}. −$_{a\,R}$ (*t x*)],
frule l-comb-mem[*of carrier R H s n f*], *assumption*+,
frule l-comb-mem[*of carrier R H* λ*x*∈{*j*. *j* ≤ *Suc n*}. −$_{a\,R}$ (*t x*)
n g], *assumption*+,
frule sc-mem[*of s* (*Suc n*) *g* (*Suc n*)], *assumption*+,
cut-tac sc-mem[*of* −$_{a\,R}$ (*t* (*Suc n*)) *g* (*Suc n*)],
simp add:*pOp-assocTr43*[*of l-comb R M n s f s* (*Suc n*) ·$_s$ *g* (*Suc n*)],
simp add:*ag-pOp-commute*[*of s* (*Suc n*) ·$_s$ *g* (*Suc n*)],
simp add:*pOp-assocTr43*[*THEN sym*, *of l-comb R M n s f*])
apply (*frule Ring.ring-is-ag*[*of R*],
frule aGroup.ag-mOp-closed[*of R t* (*Suc n*)], *assumption*+,
simp add:*sc-l-distr*[*THEN sym*],
simp add:*l-comb-add*[*THEN sym*, *of carrier R H s n f*
λ*x*∈{*j*. *j* ≤ *Suc n*}. −$_{a\,R}$ (*t x*) *n g*])
apply (*frule jointfun-hom*[*of f n H g n H*], *assumption*+,
frule jointfun-hom[*of s n carrier R* λ*x*∈{*j*. *j* ≤ *Suc n*}. −$_{a\,R}$ (*t x*)
n carrier R], *assumption*+, *simp*)
apply (*subgoal-tac* (*s* (*Suc n*) ±$_R$ −$_{a\,R}$ (*t* (*Suc n*))) ·$_s$ *g* (*Suc n*) =
l-comb R M 0 (λ*x*∈{*0*::*nat*}. (*s* (*Suc n*) ±$_R$ −$_{a\,R}$ (*t* (*Suc n*))))
(λ*x*∈{*0*::*nat*}. (*g* (*Suc n*))),
simp,
thin-tac (*s* (*Suc n*) ±$_R$ −$_{a\,R}$ (*t* (*Suc n*))) ·$_s$ *g* (*Suc n*) =
l-comb R M 0 (λ*x*∈{*0*}. *s* (*Suc n*) ±$_R$ −$_{a\,R}$ (*t* (*Suc n*)))
(λ*x*∈{*0*}. *g* (*Suc n*)))
apply (*subgoal-tac* (λ*x*∈{*0*::*nat*}. (*s* (*Suc n*) ±$_R$ −$_{a\,R}$ (*t* (*Suc n*)))) ∈
{*j*. *j* ≤ (*0*::*nat*)} → *carrier R*,
subgoal-tac (λ*x*∈{*0*::*nat*}. *g* (*Suc n*)) ∈ {*j*. *j* ≤ (*0*::*nat*)} → *H*)
apply (*simp add*:*l-comb-add*[*THEN sym*] *del*: *Pi-split-insert-domain*)
apply (*cut-tac unique-expression3-1*[*of H jointfun* (*Suc* (*n* + *n*))
(*jointfun n f n g*) *0* (λ*x*∈{*0*}. *g* (*Suc n*)) *Suc* (*n* + *n*)
jointfun (*Suc* (*n* + *n*))
(*jointfun n s n* (λ*x*∈{*j*. *j* ≤ *Suc n*}. −$_{a\,R}$ (*t x*))) *0*
(λ*x*∈{*0*}. *s* (*Suc n*) ±$_R$ −$_{a\,R}$ (*t* (*Suc n*)))], *simp del*: *Pi-split-insert-domain*,
thin-tac l-comb R M (*Suc* (*Suc* (*n* + *n*)))
(*jointfun* (*Suc* (*n* + *n*)) (*jointfun n s n* (λ*x*∈{*j*. *j* ≤ *Suc n*}. −$_{a\,R}$ (*t x*))) *0*
(λ*x*∈{*0*}. *s* (*Suc n*) ±$_R$ −$_{a\,R}$ (*t* (*Suc n*))))
(*jointfun* (*Suc* (*n* + *n*)) (*jointfun n f n g*) *0* (λ*x*∈{*0*}. *g* (*Suc n*))) =
0,
thin-tac (λ*x*∈{*j*. *j* ≤ *Suc n*}. −$_{a\,R}$ (*t x*)) ∈ {*j*. *j* ≤ *Suc n*} → *carrier R*,
thin-tac l-comb R M n s f ∈ *carrier M*,
thin-tac jointfun n f n g ∈ {*j*. *j* ≤ *Suc* (*n* + *n*)} → *H*,
thin-tac jointfun n s n (λ*x*∈{*j*. *j* ≤ *Suc n*}. −$_{a\,R}$ (*t x*))
∈ {*j*. *j* ≤ *Suc* (*n* + *n*)} → *carrier R*,
thin-tac (λ*x*∈{*0*}. *s* (*Suc n*) ±$_R$ −$_{a\,R}$ (*t* (*Suc n*))) ∈ {*0*} → *carrier R*)

apply (*thin-tac* $(-_{aR}\ (t\ (Suc\ n)))\ \cdot_s\ g\ (Suc\ n) \in carrier\ M$,
thin-tac $-_{aR}\ (t\ (Suc\ n)) \in carrier\ R$,
thin-tac l-comb R M n $(\lambda x{\in}\{j.\ j \leq Suc\ n\}.\ -_{aR}\ (t\ x))\ g \in carrier\ M$)
apply ((*erule exE*)+, (*erule conjE*)+, *erule exE*, (*erule conjE*)+)
apply (*frule-tac* $s = ta$ **and** $n = m$ **and** $m = ga$ **in** *unique-expression1*[*of H*],
assumption+, *simp*)
apply (*drule-tac* $x = m$ **in** *bspec*, *simp*)
apply (*simp add*:*jointfun-def*[*of Suc* $(n + n)$] *sliden-def*,
thin-tac $\mathbf{0} = l\text{-}comb\ R\ M\ m\ ta\ ga$,
thin-tac ta $m = s\ (Suc\ n) \pm_R -_{aR}\ (t\ (Suc\ n))$,
frule Ring.ring-is-ag[*of R*],
rotate-tac -2, *frule sym*,
subst aGroup.ag-eq-diffzero[*of R s* $(Suc\ n)$ *t* $(Suc\ n)$], *assumption*+)
apply (*rule sym*, *assumption*) **apply** *assumption*
apply (*rule Pi-I*,
case-tac $x \leq Suc\ (n + n)$, *simp add*:*jointfun-def del*: *Pi-split-insert-domain*,

simp add:*Pi-def del*: *Pi-split-insert-domain*,
rule impI, *simp add*:*sliden-def del*: *Pi-split-insert-domain*,
simp add:*Pi-def*, *simp add*:*jointfun-def sliden-def del*: *Pi-split-insert-domain*)
apply (*rule Pi-I*,
case-tac $x \leq Suc\ (n + n)$,
simp add:*jointfun-def*[*of Suc* $(n{+}n)$] *del*: *Pi-split-insert-domain*,
simp add:*Pi-def*,
simp add:*jointfun-def*[*of Suc* $(n + n)$] *sliden-def del*: *Pi-split-insert-domain*,
rule aGroup.ag-pOp-closed, *assumption*+,
simp add: *Nset-pre1 del*: *Pi-split-insert-domain*,
simp add:*im-jointfunTr1*[*of Suc* $(n{+}n)$ *jointfun n f n g 0*
$\lambda x{\in}\{0\}.\ g\ (Suc\ n)$] *del*: *Pi-split-insert-domain*,
simp add:*im-jointfun*[*of f n H g n H*] *del*: *Pi-split-insert-domain*,
(*subst jointfun-def*[*of Suc* $(n{+}n)$])+, *simp add*:*sliden-def del*: *Pi-split-insert-domain*)
apply (*rule conjI*, *frule sym*, *thin-tac f* $(Suc\ n) = g\ (Suc\ n)$,
simp add:*inj-on-def*[*of f*],
rule contrapos-pp, *simp*+, *simp add*:*image-def*, *erule exE*,
erule conjE,
drule-tac $a = Suc\ n$ **in** *forall-spec*, *simp*,
drule-tac $a = x$ **in** *forall-spec*, *simp*, *simp*)
apply (*simp add*:*inj-on-def*[*of g*],
rule contrapos-pp, *simp*+, *simp add*:*image-def*, *erule exE*,
erule conjE,
drule-tac $a = Suc\ n$ **in** *forall-spec*, *simp*,
drule-tac $a = x$ **in** *forall-spec*, *simp*, *simp*)

apply (*simp*)
apply (*rule Pi-I*, *simp*,
rule aGroup.ag-pOp-closed, *assumption*+)
apply (*subst l-comb-def*, *simp*,
rule aGroup.ag-mOp-closed, *simp add*:*Ring.ring-is-ag*,
simp add:*Pi-def*, *simp add*:*Pi-def*)

apply (*rule Pi-I*, *simp*,
frule Ring.ring-is-ag[*of R*],
rule aGroup.ag-mOp-closed[*of R*], *assumption*+,
simp add:*Pi-def*)
done

lemma (**in** *Module*) *unique-exprTr7p1*: ⟦*free-generator R M H*;
$\forall f\ s\ g\ t\ m.$
$f \in \{j.\ j \leq n\} \to H \wedge$ *inj-on f* $\{j.\ j \leq n\} \wedge s \in \{j.\ j \leq n\} \to$ *carrier R* $\wedge$
$g \in \{j.\ j \leq m\} \to H \wedge$ *inj-on g* $\{j.\ j \leq m\} \wedge t \in \{j.\ j \leq m\} \to$ *carrier R* $\wedge$
l-comb R M n s f = *l-comb R M m t g* $\wedge$
$(\forall j{\leq}n.\ s\ j \neq \mathbf{0}_R) \wedge (\forall k{\leq}m.\ t\ k \neq \mathbf{0}_R) \longrightarrow$
$n = m \wedge$
$(\exists h.\ h \in \{j.\ j \leq n\} \to \{j.\ j \leq n\} \wedge$
$(\forall l{\leq}n.$ *cmp f h l* = *g l* $\wedge$ *cmp s h l* = *t l*));
$f \in \{j.\ j \leq$ *Suc n*$\} \to H$; $s \in \{j.\ j \leq$ *Suc n*$\} \to$ *carrier R*;
$g \in \{j.\ j \leq$ *Suc n*$\} \to H$; $t \in \{j.\ j \leq$ *Suc n*$\} \to$ *carrier R*;
l-comb R M (*Suc n*) *s f* = *l-comb R M* (*Suc n*) *t g*; $\forall j{\leq}$*Suc n*. *s j* $\neq \mathbf{0}_R$;
$\forall k{\leq}$*Suc n*. *t k* $\neq \mathbf{0}_R$; *inj-on f* $\{j.\ j \leq$ *Suc n*$\}$; *inj-on g* $\{j.\ j \leq$ *Suc n*$\}$;
f (*Suc n*) = *g* (*Suc n*)⟧ $\Longrightarrow \exists h.\ h \in \{j.\ j \leq$ *Suc n*$\} \to \{j.\ j \leq$ *Suc n*$\} \wedge$
$(\forall l{\leq}$*Suc n*. *cmp f h l* = *g l* $\wedge$ *cmp s h l* = *t l*)
apply (*cut-tac sc-Ring*, *frule Ring.whole-ideal*,
frule Ring.ring-is-ag,
frule free-generator-sub)
apply (*frule-tac f* = *f* **and** *n* = *n* **and** *s* = *s* **and** *g* = *g* **and** *t* = *t* **in**
unique-expression-last[*of H*], *assumption*+)
apply (*simp add*:*l-comb-Suc*)
apply (*frule-tac f* = *f* **in** *func-pre*, *frule-tac f* = *s* **in** *func-pre*,
frule-tac f = *g* **in** *func-pre*, *frule-tac f* = *t* **in** *func-pre*,
cut-tac n = *n* **in** *Nsetn-sub*,
frule-tac f = *f* **and** *A* = $\{j.\ j \leq$ *Suc n*$\}$ **and** *?A1.0* = $\{j.\ j \leq n\}$ **in**
restrict-inj, *assumption*+,
frule-tac f = *g* **and** *A* = $\{j.\ j \leq$ *Suc n*$\}$ **and** *?A1.0* = $\{j.\ j \leq n\}$ **in**
restrict-inj, *assumption*+)
apply (*frule-tac s* = *s* **and** *m* = *f* **and** *n* = *n* **in** *l-comb-mem*[*of carrier R H*],
assumption+,
frule-tac s = *t* **and** *m* = *g* **and** *n* = *n* **in** *l-comb-mem*[*of carrier R H*],
assumption+,
frule-tac f = *t* **and** *A* = $\{j.\ j \leq$ *Suc n*$\}$ **and** *B* = *carrier R* **and**
x = *Suc n* **in** *funcset-mem*, *simp*,
frule-tac f = *g* **and** *A* = $\{j.\ j \leq$ *Suc n*$\}$ **and** *B* = *H* **and** *x* = *Suc n*
in *funcset-mem*, *simp*,
frule-tac c = *g* (*Suc n*) **in** *subsetD*[*of H carrier M*], *assumption*+,
frule-tac a = *t* (*Suc n*) **and** *m* = *g* (*Suc n*) **in** *sc-mem*, *assumption*+)
apply (*frule-tac x* = *l-comb R M n s f* **and** *y* = *t* (*Suc n*) $\cdot_s$ *g* (*Suc n*) **and**
z = $-_a$ (*t* (*Suc n*) $\cdot_s$ *g* (*Suc n*)) **in** *ag-pOp-assoc*, *assumption*+,
rule ag-mOp-closed, *assumption*, *simp add*:*ag-r-inv1 ag-r-zero*,
frule-tac x = *t* (*Suc n*) $\cdot_s$ *g* (*Suc n*) **in** *ag-mOp-closed*,
simp add:*ag-pOp-assoc*, *simp add*:*ag-r-inv1 ag-r-zero*,

rotate-tac -2, *frule sym*,
thin-tac l-comb R M n t g $=$ *l-comb R M n s f*)
apply (*drule-tac* $x = f$ **in** *spec*,
rotate-tac -1,
drule-tac $x = s$ **in** *spec*,
rotate-tac -1,
drule-tac $x = g$ **in** *spec*,
rotate-tac -1, *drule-tac* $x = t$ **in** *spec*,
rotate-tac -1, *drule-tac* $x = n$ **in** *spec*)
apply (*cut-tac* $n = n$ **in** *Nsetn-sub-mem1*, *simp*)
apply (*erule exE*, *erule conjE*)
apply (*subgoal-tac* (*jointfun n h 0* $(\lambda j.$ (*Suc n*))) $\in$
$\{j.\ j \leq$ *Suc n*$\} \to \{j.\ j \leq$ *Suc n*$\}$,
subgoal-tac $\forall\, l \leq$*Suc n*. *cmp f* (*jointfun n h 0* $(\lambda j.$ (*Suc n*))) *l* $=$
g l $\wedge$ *cmp s* (*jointfun n h 0* $(\lambda j.$ (*Suc n*))) *l* $=$ *t l*,
thin-tac $\forall\, k \leq$*Suc n*. *t k* $\neq \mathbf{0}_R$, *thin-tac inj-on f* $\{j.\ j \leq$ *Suc n*$\}$,
thin-tac inj-on g $\{j.\ j \leq$ *Suc n*$\}$, *thin-tac f* (*Suc n*) $=$ *g* (*Suc n*),
thin-tac s (*Suc n*) $=$ *t* (*Suc n*),
thin-tac f $\in \{j.\ j \leq n\} \to H$, *thin-tac s* $\in \{j.\ j \leq n\} \to$ *carrier R*,
thin-tac g $\in \{j.\ j \leq n\} \to H$, *thin-tac t* $\in \{j.\ j \leq n\} \to$ *carrier R*,
thin-tac $\{j.\ j \leq n\} \subseteq \{j.\ j \leq$ *Suc n*$\}$,
thin-tac $\forall\, l \leq n$. *cmp f h l* $=$ *g l* $\wedge$ *cmp s h l* $=$ *t l*,
thin-tac l-comb R M n t g $\in$ *carrier M*, *blast*)
apply (*rule allI*, *rule impI*, *case-tac* $l \leq n$,
simp add:*cmp-def jointfun-def*,
simp add:*cmp-def*, *simp add*:*jointfun-def sliden-def*,
frule-tac $y = l$ **and** $x = n$ **in** *not-le-imp-less*, *thin-tac* $\neg\ l \leq n$,
frule-tac $m = n$ **and** $n = l$ **in** *Suc-leI*,
frule-tac $m = l$ **and** $n =$ *Suc n* **in** *Nat.le-antisym*, *assumption*,
simp)
apply (*rule Pi-I*,
case-tac xa $\leq n$, *simp add*:*jointfun-def*, *rule impI*,
frule-tac $x = x$ **and** $f = h$ **and** $A = \{j.\ j \leq n\}$ **and**
$B = \{j.\ j \leq n\}$ **in** *funcset-mem*, *simp*, *simp*,
simp add:*jointfun-def*, *rule impI*,*frule-tac* $x = x$ **and** $f = h$ **and**
$A = \{j.\ j \leq n\}$ **and** $B = \{j.\ j \leq n\}$ **in** *funcset-mem*, *simp*, *simp*)
done

lemma (**in** *Module*) *unique-expression7*:*free-generator R M H* $\Longrightarrow$
$\forall\, f\ s\ g\ t\ m.\ f \in \{j.\ j \leq (n{::}nat)\} \to H \wedge$ *inj-on f* $\{j.\ j \leq n\} \wedge$
$s \in \{j.\ j \leq n\} \to$ *carrier R* $\wedge$
$g \in \{j.\ j \leq (m{::}nat)\} \to H \wedge$ *inj-on g* $\{j.\ j \leq m\} \wedge$
$t \in \{j.\ j \leq m\} \to$ *carrier R* $\wedge$ *l-comb R M n s f* $=$ *l-comb R M m t g* $\wedge$
$(\forall\, j \in \{j.\ j \leq n\}.\ s\ j \neq \mathbf{0}_R) \wedge (\forall\, k \in \{j.\ j \leq m\}.\ t\ k \neq \mathbf{0}_R\ ) \longrightarrow n = m \wedge$
$(\exists\, h.\ h \in \{j.\ j \leq n\} \to \{j.\ j \leq n\} \wedge (\forall\, l \in \{j.\ j \leq n\}.$ (*cmp f h*) $l = g\ l$
$\wedge$ (*cmp s h*) $l = t\ l$))
apply (*cut-tac sc-Ring*,
frule Ring.ring-is-ag[*of R*],
frule free-generator-sub[*of H*])

apply (*induct-tac n*)
apply (*rule allI*)+
apply (*rule impI*, (*erule conjE*)+)
apply (*frule-tac H = H* **and** *f = f* **and** *n = 0* **and** *s = s* **and** *g = g* **and** *m =*
m
and *t = t* **in** *unique-expression7-1*, *assumption*+, *simp del*: *Pi-split-insert-domain*)
apply (*frule-tac H = H* **and** *f = f* **and** *n = 0* **and** *s = s* **and** *g = g* **and** *m =*
m
and *t = t* **in** *unique-expression6*, (*simp del*: *Pi-split-insert-domain*) +)
apply (*simp add*:*l-comb-def del*: *Pi-split-insert-domain*)
apply (*frule-tac f = s* **in** *funcset-mem*[*of - {0} carrier R 0*], *simp del*: *Pi-split-insert-domain*,
frule-tac f = t **in** *funcset-mem*[*of - {0} carrier R 0*], *simp del*: *Pi-split-insert-domain*,
frule-tac f = g **in** *funcset-mem*[*of - {0} H 0*], *simp del*: *Pi-split-insert-domain*,
frule-tac c = g 0 **in** *subsetD*[*of H carrier M*], *assumption*+)
apply (*frule-tac a = s 0* **and** *m = g 0* **in** *sc-mem*, *assumption*+,
frule-tac a = t 0 **and** *m = g 0* **in** *sc-mem*, *assumption*+,
frule-tac a = s 0 $\cdot_s$ *g 0* **and** *b = t 0* $\cdot_s$ *g 0* **in** *ag-eq-diffzero*)
apply *assumption*
apply (*simp only*:*sc-minus-am1*,
thin-tac (*s 0* $\cdot_s$ *g 0 = t 0* $\cdot_s$ *g 0*) = (**0 = 0**))
apply (*frule-tac x = t 0* **in** *aGroup.ag-mOp-closed*[*of R*], *assumption*+)
apply (*simp add*:*sc-l-distr*[*THEN sym*],
frule-tac h = g 0 **and** *a = s 0* $\pm_R$ $-_{a\,R}$ (*t 0*) **in**
free-gen-coeff-zero[*of H*],
assumption+,
rule aGroup.ag-pOp-closed[*of R*], *assumption*+)
apply (*simp add*:*aGroup.ag-eq-diffzero*[*THEN sym*, *of R*],
simp add:*cmp-def*,
cut-tac Pi-I[*of* {*0*::*nat*} *id* %-. {*0*::*nat*}],
subgoal-tac f (*id 0*) = *g 0* $\wedge$ *s* (*id 0*) = *t 0*, *blast*,
simp add:*id-def*, *simp add*:*id-def*)

apply ((*rule allI*)+, *rule impI*)
apply (*erule conjE*)+
apply (*frule-tac H = H* **and** *f = f* **and** *n = Suc n* **and** *s = s* **and** *g = g* **and**
m = m **and** *t = t* **in** *unique-expression6*, *assumption*+)
apply (*cut-tac k = Suc n* **in** *finite-Collect-le-nat*,
frule-tac A = {*j*. *j* $\leq$ *Suc n*} **and** *f = f* **in** *inj-on-iff-eq-card*,
simp)
apply (*cut-tac k = m* **in** *finite-Collect-le-nat*,
frule-tac A = {*j*. *j* $\leq$ *m*} **and** *f = g* **in** *inj-on-iff-eq-card*,
simp,
thin-tac card (*g* ‘ {*j*. *j* $\leq$ *m*}) = *Suc m*)
apply (*frule sym*, *thin-tac Suc n = m*, *simp*)
apply (*cut-tac a = Suc n* **and** *A* = {*j*. *j* $\leq$ *Suc n*} **and** *f = g* **in**
mem-in-image2) **apply** *simp* **apply** (
rotate-tac -6) **apply** (*frule sym*,
thin-tac f ‘ {*j*. *j* $\leq$ *Suc n*} = *g* ‘ {*j*. *j* $\leq$ *Suc n*}, *simp*,
simp add:*image-def*) **apply** (*erule exE*, *erule conjE*)

apply (*case-tac* $x = Suc\ n$, *simp*, *thin-tac* $x = Suc\ n$,
rotate-tac -1, *frule sym*, *thin-tac* $g\ (Suc\ n) = f\ (Suc\ n)$)
apply (*rule-tac* $n = n$ **and** $f = f$ **and** $s = s$ **and** $g = g$ **and** $t = t$ **in**
unique-exprTr7p1[*of H*], *assumption*+)

apply (*rotate-tac* -2, *frule sym*, *thin-tac* $g\ (Suc\ n) = f\ x$)
apply (*frule Ring.whole-ideal*)
apply (*frule-tac* $s = s$ **and** $n = n$ **and** $f = f$ **and** $j = x$ **in**
l-comb-transpos1[*of carrier R H*], *assumption*+,
rule noteq-le-less, *assumption*+, *simp*) **apply** (
thin-tac l-comb R M $(Suc\ n)\ s\ f = l\text{-}comb\ R\ M\ (Suc\ n)\ t\ g$)
apply (*frule-tac* $f = cmp\ f\ (transpos\ x\ (Suc\ n))$ **and** $n = n$ **and**
$s = cmp\ s\ (transpos\ x\ (Suc\ n))$ **and** $g = g$ **and** $t = t$ **in**
unique-exprTr7p1[*of H*], *assumption*+)
apply (*rule Pi-I*,
simp add:*cmp-def*, *rule-tac* $f = f$ **and** $A = \{j.\ j \leq Suc\ n\}$ **and** $B = H$
and $x = transpos\ x\ (Suc\ n)\ xa$ **in** *funcset-mem*, *simp*,
simp add:*transpos-mem*)
apply (*rule Pi-I*,
simp add:*cmp-def*, *rule-tac* $f = s$ **and** $A = \{j.\ j \leq Suc\ n\}$ **and**
$B = carrier\ R$ **and** $x = transpos\ x\ (Suc\ n)\ xa$ **in** *funcset-mem*, *simp*,
simp add:*transpos-mem*)
apply *assumption*+
apply (*rule allI*, *rule impI*, *simp add*:*cmp-def*)
apply (*cut-tac* $i = x$ **and** $n = Suc\ n$ **and** $j = Suc\ n$ **and** $l = j$ **in**
transpos-mem, *simp*+)
apply (*cut-tac* $i = x$ **and** $n = Suc\ n$ **and** $j = Suc\ n$ **in** *transpos-inj*,
simp+)
apply (*cut-tac* $i = x$ **and** $n = Suc\ n$ **and** $j = Suc\ n$ **in**
transpos-hom, *simp*+)
apply (*rule-tac* $f = transpos\ x\ (Suc\ n)$ **and** $A = \{j.\ j \leq Suc\ n\}$ **and**
$B = \{j.\ j \leq Suc\ n\}$ **and** $g = f$ **and** $C = H$ **in** *cmp-inj*,
assumption+)
apply (*simp add*:*cmp-def transpos-ij-2*)
apply (*erule exE*, *erule conjE*)
apply (*cut-tac* $i = x$ **and** $n = Suc\ n$ **and** $j = Suc\ n$ **in**
transpos-hom, *simp*+)
apply (*simp add*:*cmp-assoc*[*THEN sym*])
apply (*frule-tac* $f = h$ **and** $A = \{j.\ j \leq Suc\ n\}$ **and** $B = \{j.\ j \leq Suc\ n\}$
and $g = transpos\ x\ (Suc\ n)$ **and** $C = \{j.\ j \leq Suc\ n\}$ **in**
cmp-fun, *assumption*+)
apply (*thin-tac* $\forall f\ s\ g\ t\ m.\ f \in \{j.\ j \leq n\} \to H \wedge inj\text{-}on\ f\ \{j.\ j \leq n\} \wedge$
$s \in \{j.\ j \leq n\} \to carrier\ R \wedge g \in \{j.\ j \leq m\} \to H \wedge inj\text{-}on\ g\ \{j.\ j \leq m\} \wedge$
$t \in \{j.\ j \leq m\} \to carrier\ R \wedge l\text{-}comb\ R\ M\ n\ s\ f = l\text{-}comb\ R\ M\ m\ t\ g \wedge$
$(\forall j{\leq}n.\ s\ j \neq \mathbf{0}_R) \wedge (\forall k{\leq}m.\ t\ k \neq \mathbf{0}_R) \longrightarrow\ n = m \wedge$
$(\exists h.\ h \in \{j.\ j \leq n\} \to \{j.\ j \leq n\} \wedge$
$(\forall l{\leq}n.\ cmp\ f\ h\ l = g\ l \wedge cmp\ s\ h\ l = t\ l))$,
thin-tac $\forall j{\leq}Suc\ n.\ s\ j \neq \mathbf{0}_R$, *thin-tac* $\forall k{\leq}Suc\ n.\ t\ k \neq \mathbf{0}_R$,
thin-tac $\{y.\ \exists x{\leq}Suc\ n.\ y = g\ x\} = \{y.\ \exists x{\leq}Suc\ n.\ y = f\ x\}$)

apply *blast*
done

lemma (**in** *Module*) *gen-mHom-eq*:⟦*R module N*; *generator R M H*; *f* ∈ *mHom R M N*;
g ∈ *mHom R M N*; ∀ *h*∈*H*. *f h* = *g h* ⟧ ⟹ *f* = *g*
apply (*rule mHom-eq*[*of N*], *assumption*+)
apply (*rule ballI*)
apply (*unfold generator-def*, *frule conjunct2*, *fold generator-def*)
apply (*frule sym*, *thin-tac linear-span R M* (*carrier R*) *H* = *carrier M*,
simp,
thin-tac carrier M = *linear-span R M* (*carrier R*) *H*)
apply (*simp add*:*linear-span-def*)
apply (*case-tac H* = {}, *simp*, *simp add*:*mHom-0*, *simp*)
apply (*erule exE*, (*erule bexE*)+)
apply (*unfold generator-def*, *frule conjunct1*, *fold generator-def*)
apply (*cut-tac sc-Ring*, *frule Ring.whole-ideal*)
apply *simp*
apply (*simp add*:*linmap-im-lincomb*[*of carrier R N*])

apply (*frule-tac s* = *s* **and** *n* = *n* **and** *f* = *cmp f fa* **and** *g* = *cmp g fa* **in**
Module.linear-comb-eq[*of N R f* ‘ *H*],
(*simp add*:*mHom-def aHom-def*, (*erule conjE*)+,
simp add:*image-sub*[*of f carrier M carrier N H*],
assumption),
(*rule Pi-I*, *simp add*:*cmp-def*,
simp add:*image-def*,
frule-tac f = *fa* **and** *A* = {*j*. *j* ≤ *n*} **and** *B* = *H* **and** *x* = *x* **in**
funcset-mem, *simp*, *blast*),
(*rule Pi-I*, *simp add*:*cmp-def*,
simp add:*image-def*,
frule-tac f = *fa* **and** *A* = {*j*. *j* ≤ *n*} **and** *B* = *H* **and** *x* = *x* **in**
funcset-mem, *simp*, *blast*),
(*rule ballI*, *simp add*:*cmp-def*,
frule-tac f = *fa* **and** *A* = {*j*. *j* ≤ *n*} **and** *B* = *H* **and** *x* = *j* **in**
funcset-mem, *simp*))
apply *blast*
apply *assumption*
done

5.5 Existence of homomorphism

definition
fgs :: [('*r*, '*m*) *Ring-scheme*, ('*a*, '*r*, '*m1*) *Module-scheme*, '*a set*] ⇒
'*a set* **where**

fgs R M A = *linear-span R M* (*carrier R*) *A*

definition

fsp :: [('*r*, '*m*) *Ring-scheme*, ('*a*, '*r*, '*m1*) *Module-scheme*,
('*b*, '*r*, '*m2*) *Module-scheme*, '*a* ⇒ '*b*, '*a set*, '*a set*, '*a* ⇒ '*b*] ⇒ *bool* **where**
fsp R M N f H A g ⟷ *g* ∈ *mHom R* (*mdl M* (*fgs R M A*)) *N* ∧ (∀ *z*∈*A*. *f z* = *g z*) ∧ *A* ⊆ *H*

definition
fsps :: [('*r*, '*m*) *Ring-scheme*, ('*a*, '*r*, '*m1*) *Module-scheme*,
('*b*, '*r*, '*m2*) *Module-scheme*, '*a* ⇒ '*b*, '*a set*] ⇒
(('*a set*) ∗ ('*a* ⇒ '*b*)) *set* **where**
fsps R M N f H = {*Z*. ∃ *A g*. *Z* = (*A*, *g*) ∧ *fsp R M N f H A g*}

definition
od-fm-fun :: [('*r*, '*m*) *Ring-scheme*, ('*a*, '*r*, '*m1*) *Module-scheme*,
('*b*, '*r*, '*m2*) *Module-scheme*, '*a* ⇒ '*b*, '*a set*] ⇒
(('*a set*) ∗ ('*a* ⇒ '*b*)) *Order* **where**
od-fm-fun R M N f H = (|*carrier* = *fsps R M N f H*,
rel = {*Y*. *Y* ∈ (*fsps R M N f H*) × (*fsps R M N f H*) ∧
(*fst* (*fst Y*)) ⊆ (*fst* (*snd Y*))} |)

lemma (**in** *Module*) *od-fm-fun-carrier*:*carrier* (*od-fm-fun R M N f H*) =
fsps R M N f H
by (*simp add*:*od-fm-fun-def*)

lemma (**in** *Module*) *fgs-submodule*:*a* ⊆ *carrier M* ⟹
submodule R M (*fgs R M a*)
apply (*cut-tac sc-Ring*, *frule Ring.whole-ideal*)
apply (*subst fgs-def*)
apply (*rule linear-span-subModule*, *assumption*, *assumption*)
done

lemma (**in** *Module*) *fgs-sub-carrier*:*a* ⊆ *carrier M* ⟹ (*fgs R M a*) ⊆ *carrier M*
by (*frule fgs-submodule*[*of a*],
rule submodule-subset, *assumption*+)

lemma (**in** *Module*) *elem-fgs*:⟦*a* ⊆ *carrier M*; *x* ∈ *a*⟧ ⟹ *x* ∈ *fgs R M a*
apply (*simp add*:*fgs-def*)
apply (*frule l-span-cont-H*[*of a*])
apply (*simp add*:*subsetD*)
done

lemma (**in** *Module*) *fst-chain-subset*:⟦*R module N*; *free-generator R M H*;
f ∈ *H* → *carrier N*; *C* ⊆ *fsps R M N f H*; (*a*, *b*) ∈ *C*⟧ ⟹ *a* ⊆ *carrier M*
apply (*subgoal-tac a* ⊆ *H*, *frule free-generator-sub*)
apply (*rule subset-trans*, *assumption*+)
apply (*frule-tac A* = *C* **and** *B* = *fsps R M N f H* **and** *c* = (*a*, *b*) **in** *subsetD*,
assumption+)

apply (*simp add*:*fsps-def fsp-def*)
done

lemma (**in** *Module*) *empty-fsp*:⟦*R module N*; *free-generator R M H*;
f ∈ *H* → *carrier N*⟧ ⟹ ({}, (λ*x*∈{**0**$_M$}. **0**$_N$)) ∈ *fsps R M N f H*
apply (*simp add*:*fsps-def fsp-def*,
simp add:*fgs-def linear-span-def*,
cut-tac submodule-0,
frule mdl-is-module[*of* {**0**}])
apply (*rule Module.mHom-test*, *assumption*+)
apply (*rule conjI*)
apply (*rule Pi-I*, *simp add*:*mdl-carrier*,
rule Module.module-inc-zero[*of N R*], *assumption*+)
apply (*simp add*:*mdl-carrier*)
apply (*simp add*:*mdl-def*, *fold mdl-def*)
apply (*cut-tac ag-inc-zero*, *simp add*:*ag-l-zero*,
frule Module.module-is-ag[*of N R*],
frule aGroup.ag-inc-zero[*of N*], *simp add*:*aGroup.ag-l-zero*[*of N*])
apply (*rule ballI*, *frule-tac a* = *a* **in** *sc-a-0*, *simp*)
apply (*simp add*:*Module.sc-a-0*)
done

lemma (**in** *Module*) *mem-fgs-l-comb*:⟦*K* ≠ {}; *K* ⊆ *carrier M*; *x* ∈ *fgs R M K*⟧
⟹
∃(*n*::*nat*).
∃*f* ∈ {*j*. *j* ≤ (*n*::*nat*)} → *K*. ∃ *s* ∈ {*j*. *j* ≤ *n*} → *carrier R*.
x = *l-comb R M n s f*
apply (*simp add*:*fgs-def linear-span-def*)
done

lemma *PairE-lemma*: *EX x y*. *p* = (*x*, *y*) **by** *auto*

lemma (**in** *Module*) *fsps-chain-boundTr1*:⟦*R module N*; *free-generator R M H*;
f ∈ *H* → *carrier N*; *C* ⊆ *fsps R M N f H*;
∀ *a*∈*C*. ∀ *b*∈*C*. *fst a* ⊆ *fst b* ∨ *fst b* ⊆ *fst a*; ∀ *a b*. (*a*, *b*) ∈ *C* ⟶
(*a*, *b*) ∈ *fsps R M N f H*; ∃ *x*. (∃ *b*. (*x*, *b*) ∈ *C*) ∧ *x* ≠ {}⟧ ⟹
fa ∈ {*j*. *j* ≤ (*n*::*nat*)} → ⋃{*a*. ∃ *b*. (*a*, *b*) ∈ *C*}
⟶ (∃(*c*, *d*) ∈ *C*. *fa* ‘ {*j*. *j* ≤ *n*} ⊆ *c*)
apply (*induct-tac n*)
apply (*rule impI*, *simp del*: *Pi-split-insert-domain*)
apply ((*erule exE*)+, *erule conjE*, *erule exE*)
apply (*frule-tac f* = *fa* **and** *A* = {*0*} **and** *B* = ⋃{*a*. ∃ *b*. (*a*, *b*) ∈ *C*} **and**
?A1.0 = {*0*} **in** *image-sub*, *simp*, *simp*)
apply *blast*

apply (*rule impI*)
apply (*frule func-pre*, *simp*)
apply (*frule-tac f* = *fa* **and** *A* = {*j*. *j* ≤ (*Suc n*)} **and**
B = ⋃{*a*. ∃ *b*. (*a*, *b*) ∈ *C*} **and** *x* = *Suc n* **in** *funcset-mem*, *simp*)

apply *simp*

apply ((*erule exE*)+, *erule conjE*, *erule exE*, *erule conjE*, *erule exE*)
apply (*erule bexE*)
apply (*drule-tac* $x = (xa, ba)$ **in** *bspec*, *assumption*,
drule-tac $x = xb$ **in** *bspec*, *assumption*+)
apply (*erule disjE*, *simp*)
apply (*subgoal-tac* $(\lambda(c, d).\ fa\ `\ \{j.\ j \leq Suc\ n\} \subseteq c)\ xb$)
apply *auto*[*1*]
apply (*subgoal-tac* $(\lambda(c, d).\ fa\ `\ \{j.\ j \leq Suc\ n\} \subseteq c)\ xb$)
apply (*blast*,
cut-tac $p = xb$ **in** *PairE-lemma*,
(*erule exE*)+, *simp*, *rule subsetI*, *simp add*:*image-def*,
erule exE, *erule conjE*,
case-tac $xe = Suc\ n$, *simp*, *simp add*:*subsetD*,
frule-tac $m = xe$ **and** $n = Suc\ n$ **in** *noteq-le-less*, *assumption*,
frule-tac $x = xe$ **and** $n = n$ **in** *Suc-less-le*,
thin-tac $xe \leq Suc\ n$, *thin-tac* $xe < Suc\ n$,
cut-tac $c = xd$ **and** $A = \{y.\ \exists x{\leq}n.\ y = fa\ x\}$ **and** $B = xc$ **in**
subsetD, *simp*, *simp*, *blast*, *assumption*)
apply (*subgoal-tac* $(\lambda(c, d).\ fa\ `\ \{j.\ j \leq Suc\ n\} \subseteq c)\ (xa, ba)$)
apply *auto*[*1*]
apply (*simp*, *rule subsetI*, *simp add*:*image-def*, *erule exE*, *erule conjE*,
case-tac $xd = Suc\ n$, *simp*,
frule-tac $m = xd$ **and** $n = Suc\ n$ **in** *noteq-le-less*, *assumption*,
frule-tac $x = xd$ **and** $n = n$ **in** *Suc-less-le*,
thin-tac $xd \leq Suc\ n$, *thin-tac* $xd < Suc\ n$,
cut-tac $p = xb$ **in** *PairE-lemma*, (*erule exE*)+, *simp*,
cut-tac $c = fa\ xd$ **and** $A = \{y.\ \exists x{\leq}n.\ y = fa\ x\}$ **and** $B = xe$ **in**
subsetD, *assumption*, *simp*, *blast*)
apply (*rule-tac* $c = fa\ xd$ **and** $A = xe$ **and** $B = xa$ **in** *subsetD*, *assumption*+)
done

lemma (**in** *Module*) *fsps-chain-boundTr1-1*:⟦*R module N*; *free-generator R M H*;
$f \in H \to carrier\ N$; $C \subseteq fsps\ R\ M\ N\ f\ H$;
$\forall\, a{\in}C.\ \forall\, b{\in}C.\ fst\ a \subseteq fst\ b \vee fst\ b \subseteq fst\ a$;
$\exists\, x.\ (\exists\, b.\ (x, b) \in C) \wedge x \neq \{\}$;
$fa \in \{j.\ j \leq (n{::}nat)\} \to \bigcup\{a.\ \exists\, b.\ (a, b) \in C\}$⟧ ⟹
$\exists (c, d) \in C.\ fa\ `\ \{j.\ j \leq n\} \subseteq c$
apply (*subgoal-tac* $\forall\, a\ b.\ (a, b) \in C \longrightarrow (a, b) \in fsps\ R\ M\ N\ f\ H$)
apply (*simp add*:*fsps-chain-boundTr1*)
apply ((*rule allI*)+, *rule impI*, *simp add*:*subsetD*)
done

lemma (**in** *Module*) *fsps-chain-boundTr1-2*:⟦*R module N*; *free-generator R M H*;
$f \in H \to carrier\ N$; $C \subseteq fsps\ R\ M\ N\ f\ H$;
$\forall\, a{\in}C.\ \forall\, b{\in}C.\ fst\ a \subseteq fst\ b \vee fst\ b \subseteq fst\ a$;
$\exists\, x.\ (\exists\, b.\ (x, b) \in C) \wedge x \neq \{\}$;
$fa \in \{j.\ j \leq (n{::}nat)\} \to \bigcup\{a.\ \exists\, b.\ (a, b) \in C\}$⟧ ⟹

$\exists P \in C.\ fa\ `\ \{j.\ j \leq n\} \subseteq fst\ P$
apply (*frule-tac* $N = N$ **and** $f = f$ **and** $C = C$ **and** $fa = fa$ **and** $n = n$ **in**
fsps-chain-boundTr1-1, *assumption*+)
apply (*erule bexE*)
apply (*cut-tac* $p = x$ **in** *PairE-lemma*, (*erule exE*)+)
apply (*subgoal-tac* $fa\ `\ \{j.\ j \leq n\} \subseteq fst\ x$)
apply *blast*
apply *simp*
done

lemma (**in** *Module*) *eSum-in-SubmoduleTr*:$[\![H \subseteq carrier\ M;\ K \subseteq H]\!] \Longrightarrow$
$f \in \{j.\ j \leq (n::nat)\} \to K \wedge s \in \{j.\ j \leq n\} \to carrier\ R \longrightarrow$
l-comb R (*mdl M* (*fgs R M K*)) *n s f* = *l-comb R M n s f*
apply (*induct-tac n*)
apply (*rule impI*, (*erule conjE*)+)
apply (*simp add*:*l-comb-def del*: *Pi-split-insert-domain*, *simp add*:*mdl-def del*: *Pi-split-insert-domain*, *rule impI*)
apply (*simp add*:*fgs-def del*: *Pi-split-insert-domain*, *frule subset-trans*[*of K H carrier M*], *assumption*+)
apply (*frule funcset-mem*[*of f* {*0*} *K 0*], *simp del*: *Pi-split-insert-domain*,
frule l-span-cont-H[*of K*], *simp add*:*subsetD*)

apply (*rule impI*, *erule conjE*)
apply (*frule func-pre*[*of* - - *K*], *frule func-pre*[*of* - - *carrier R*],
simp)
apply (*simp add*:*l-comb-def*, *simp add*:*mdl-def*)
apply (*rule impI*, *simp add*:*fgs-def*,*frule subset-trans*[*of K H carrier M*],
assumption+)
apply (*frule-tac* $x = Suc\ n$ **and** $A = \{j.\ j \leq Suc\ n\}$ **in**
funcset-mem[*of f* - *K*], *simp*,
frule l-span-cont-H[*of K*], *simp add*:*subsetD*)
done

lemma (**in** *Module*) *eSum-in-Submodule*:$[\![H \subseteq carrier\ M;\ K \subseteq H;$
$f \in \{j.\ j \leq (n::nat)\} \to K;\ s \in \{j.\ j \leq n\} \to carrier\ R]\!] \Longrightarrow$
l-comb R (*mdl M* (*fgs R M K*)) *n s f* = *l-comb R M n s f*
apply (*simp add*:*eSum-in-SubmoduleTr*)
done

lemma (**in** *Module*) *fgs-generator*:$H \subseteq carrier\ M \Longrightarrow$
generator R (*mdl M* (*fgs R M H*)) *H*
apply (*simp add*:*generator-def*, *simp add*:*mdl-def*, *fold mdl-def*)
apply (*rule conjI*)
apply (*simp add*:*fgs-def*)
apply (*simp add*:*l-span-cont-H*)
apply (*rule equalityI*)
apply (*rule subsetI*, *simp add*:*linear-span-def*)
apply (*case-tac* $H = \{\}$, *simp add*:*mdl-def*)
apply (*simp add*:*fgs-def linear-span-def*)

apply *simp* **apply** (*erule exE*, (*erule bexE*)+)
apply (*cut-tac f = f* **and** *n = n* **and** *s = s* **in** *eSum-in-Submodule*[*of H H*],
assumption+, *simp*, *assumption*+)
apply *simp*
apply (*simp add*:*fgs-def*)
apply (*cut-tac sc-Ring*, *frule Ring.whole-ideal*)
apply (*simp add*:*l-comb-mem-linear-span*)
apply (*rule subsetI*)
apply (*case-tac H = {}*, *simp*)
apply (*simp add*:*fgs-def linear-span-def mdl-def*)
apply (*frule-tac x = x* **in** *mem-fgs-l-comb*[*of H*], *assumption*+)
apply (*erule exE*, *erule bexE*, *erule bexE*)
apply (*simp add*:*eSum-in-Submodule*[*THEN sym*, *of H H*])
apply (*frule fgs-submodule*[*of H*],
frule mdl-is-module[*of fgs R M H*])
apply (*rule Module.l-comb-mem-linear-span*[*of mdl M* (*fgs R M H*) *R carrier R H*], *assumption*+)
apply (*cut-tac sc-Ring*, *rule Ring.whole-ideal*, *assumption*)
apply (*simp add*:*mdl-carrier*, *simp add*:*fgs-def l-span-cont-H*,
assumption+)
done

lemma (**in** *Module*) *fgs-mono*:⟦*free-generator R M H*; *J ⊆ K*; *K ⊆ H*⟧
⟹ *fgs R M J ⊆ fgs R M K*
apply (*cut-tac sc-Ring*)
apply (*case-tac J = {}*)
apply (*simp add*:*fgs-def linear-span-def*)
apply (*case-tac K = {}*) **apply** *simp* **apply** *simp*
apply (*frule nonempty-ex* [*of K*])
apply (*erule exE*)
apply (*subgoal-tac* (*λj. x*) ∈ {*j. j ≤* (*0*::*nat*)} → *K*,
subgoal-tac (*λj.* $\mathbf{0}_R$) ∈ {*j. j ≤* (*0*::*nat*)} → *carrier R*,
subgoal-tac $\mathbf{0}_M$ = *l-comb R M 0* (*λj.* $\mathbf{0}_R$) (*λj. x*))
apply *blast*
apply (*simp add*:*l-comb-def*)
apply (*frule free-generator-sub*[*of H*],
frule subset-trans[*of K H carrier M*], *assumption*+,
frule-tac c = x **in** *subsetD*[*of K carrier M*], *assumption*+,
simp add:*sc-0-m*)
apply (*rule Pi-I*,
frule Ring.ring-is-ag [*of R*],
simp add:*aGroup.ag-inc-zero*[*of R*])
apply (*rule Pi-I*, *assumption*)
apply (*subgoal-tac K ≠ {}*) **prefer** *2* **apply** (*rule contrapos-pp*, *simp*+)
apply (*rule subsetI*)
apply (*simp add*:*fgs-def*, *simp add*:*linear-span-def*)
apply (*erule exE*, (*erule bexE*)+)
apply (*frule-tac f = f* **and** *A = {j. j ≤ n}* **and** *B = J* **and** *?B1.0 = K* **in**
extend-fun, *assumption*+)

apply *blast*
done

lemma (**in** *Module*) *empty-fgs*:*fgs R M* {} = {**0**}
by(*simp add*:*fgs-def linear-span-def*)

lemma (**in** *Module*) *mem-fsps-snd-mHom*:⟦*R module N*; *free-generator R M H*;
f ∈ *H* → *carrier N*; (*a*, *b*) ∈ *fsps R M N f H*⟧ ⟹
b ∈ *mHom R* (*mdl M* (*fgs R M a*)) *N*
by (*simp add*:*fsps-def fsp-def*)

lemma (**in** *Module*) *mem-fsps-fst-sub*:⟦*R module N*; *free-generator R M H*;
f ∈ *H* → *carrier N*; (*a*, *b*) ∈ *fsps R M N f H*⟧ ⟹ *a* ⊆ *H*
by (*simp add*:*fsps-def fsp-def*)

lemma (**in** *Module*) *mem-fsps-fst-sub1*:⟦*R module N*; *free-generator R M H*;
f ∈ *H* → *carrier N*; (*a*, *b*) ∈ *fsps R M N f H*⟧ ⟹ *a* ⊆ *carrier M*
apply (*simp add*:*fsps-def fsp-def*)
apply (*frule free-generator-sub*[*of H*],
rule subset-trans[*of a H carrier M*], *simp*+)
done

lemma (**in** *Module*) *Order-od-fm-fun*:⟦*R module N*; *free-generator R M H*;
f ∈ *H* → *carrier N*⟧ ⟹ *Order* (*od-fm-fun R M N f H*)
apply (*rule Order.intro*)
apply (*simp add*:*od-fm-fun-carrier*)
apply (*rule subsetI*, *simp add*:*od-fm-fun-def*)
apply (*simp add*:*od-fm-fun-def*)
apply (*simp add*:*od-fm-fun-def*)
apply (*cut-tac p* = *a* **in** *PairE-lemma*, (*erule exE*)+,
cut-tac p = *b* **in** *PairE-lemma*, (*erule exE*)+, *simp*)
apply (*frule-tac a* = *x* **and** *b* = *y* **in** *mem-fsps-snd-mHom*[*of N H f*], *assumption*+,
frule-tac a = *x* **and** *b* = *ya* **in** *mem-fsps-snd-mHom*[*of N H f*], *assumption*+)
apply (*frule-tac a* = *x* **and** *b* = *y* **in** *mem-fsps-fst-sub*[*of N H f*], *assumption*+,
frule free-generator-sub,
frule-tac A = *x* **and** *B* = *H* **and** *C* = *carrier M* **in** *subset-trans*,
assumption+)
apply (*frule-tac a* = *x* **in** *fgs-submodule*,
frule-tac H = *fgs R M x* **in** *mdl-is-module*)
apply (*thin-tac a* = (*x*, *y*), *thin-tac b* = (*x*, *ya*))
apply (*simp add*:*mdl-carrier*)
apply (*frule-tac H* = *x* **in** *fgs-generator*)
apply (*frule-tac R* = *R* **and** *M* = *mdl M* (*fgs R M x*) **and** *N* = *N* **and** *H* = *x*
and *f* = *y* **and** *g* = *ya* **in** *Module.gen-mHom-eq*, *assumption*+,
simp add:*fsps-def fsp-def*, *assumption*)
apply (*simp add*:*od-fm-fun-def*)
done

lemma (**in** *Module*) *fsps-chain-boundTr2-1*:⟦*R module N*;
free-generator R M H; *f* ∈ *H* → *carrier N*; *C* ⊆ *fsps R M N f H*;
(*a*, *b*) ∈ *C*; (*aa*, *ba*) ∈ *C*; *x* ∈ *fgs R M a*; *x* ∈ *fgs R M aa*; *a* ⊆ *aa*⟧
⟹ *b x* = *ba x*
apply (*cut-tac sc-Ring*, *frule Ring.whole-ideal*, *frule free-generator-sub*)
apply (*frule mem-fsps-snd-mHom*[*of N H f a b*], *assumption*+, *simp add*:*subsetD*,
frule mem-fsps-snd-mHom[*of N H f aa ba*], *assumption*+, *simp add*:*subsetD*)
apply (*case-tac a* = {}, *simp add*:*empty-fgs*)
apply (*cut-tac submodule-0*, *frule mdl-is-module*[*of* {**0**}],
frule subsetD[*of C fsps R M N f H* (*aa*, *ba*)], *assumption*+)
apply (*cut-tac fgs-submodule*[*of aa*],
frule mdl-is-module[*of fgs R M aa*])
apply (*frule Module.mHom-0*[*of mdl M* {**0**} *R N b*], *assumption*+,
frule Module.mHom-0[*of mdl M* (*fgs R M aa*) *R N ba*], *assumption*+,
simp add:*mdl-def*)
apply (*frule mem-fsps-fst-sub*[*of N H f aa ba*], *assumption*+,
rule subset-trans[*of aa H carrier M*], *assumption*+)
apply (*frule nonempty-ex*[*of a*], *erule exE*,
frule-tac c = *xa* **in** *subsetD*[*of a aa*], *assumption*,
frule-tac x = *xa* **in** *nonempty*[*of* - *aa*],
thin-tac xa ∈ *a*, *thin-tac xa* ∈ *aa*)
apply (*frule mem-fsps-fst-sub*[*of N H f aa ba*], *assumption*+,
simp add:*subsetD*)
apply (*frule mem-fgs-l-comb*[*of a x*],
frule subset-trans[*of a aa H*], *assumption*+,
rule subset-trans[*of a H carrier M*], *assumption*+)
apply (*erule exE*, (*erule bexE*)+)
apply (*frule-tac f* = *fa* **and** *A* = {*j*. *j* ≤ *n*} **and** *B* = *a* **and** *?B1.0* = *aa* **in**
extend-fun, *assumption*+)
apply (*frule subsetD*[*of C fsps R M N f H* (*a*, *b*)], *assumption*+,
frule subsetD[*of C fsps R M N f H* (*aa*, *ba*)], *assumption*+,
thin-tac C ⊆ *fsps R M N f H*)
apply (*cut-tac fgs-submodule*[*of a*],
frule mdl-is-module[*of fgs R M a*],
cut-tac fgs-submodule[*of aa*],
frule mdl-is-module[*of fgs R M aa*])
apply (*frule subset-trans*[*of a aa H*], *assumption*+)
apply (*frule-tac f1* = *fa* **and** *n1* = *n* **and** *s1* = *s* **in** *eSum-in-Submodule*[*THEN sym*,
of H a], *assumption*+,
frule-tac s = *s* **and** *n* = *n* **and** *g* = *fa* **in** *Module.linmap-im-lincomb*[*of*
mdl M (*fgs R M a*) *R carrier R N* - *a*], *assumption*+,
simp add:*mdl-carrier*,
subst fgs-def, *rule-tac l-span-cont-H*,
frule subset-trans[*of a H carrier M*], *assumption*+, *simp*,
thin-tac b (*l-comb R* (*mdl M* (*fgs R M a*)) *n s fa*) =
l-comb R N n s (*cmp b fa*),
rotate-tac −*1*, *frule sym*,

thin-tac l-comb R M n s fa = l-comb R (mdl M (fgs R M a)) n s fa,
simp,
thin-tac l-comb R (mdl M (fgs R M a)) n s fa = l-comb R M n s fa)
apply (*frule-tac f1 = fa* **and** *n1 = n* **and** *s1 = s* **in** *eSum-in-Submodule*[*THEN sym*,
of H aa], *assumption*+,
frule-tac s = s **and** *n = n* **and** *g = fa* **in** *Module.linmap-im-lincomb*[*of mdl M (fgs R M aa) R carrier R N - aa*], *assumption*+,
simp add:*mdl-carrier*,
subst fgs-def, *rule-tac l-span-cont-H*,
frule subset-trans[*of a H carrier M*], *assumption*+, *simp*+,
thin-tac l-comb R M n s fa = l-comb R (mdl M (fgs R M aa)) n s fa,
thin-tac ba (l-comb R (mdl M (fgs R M aa)) n s fa) =
l-comb R N n s (cmp ba fa))
apply (*simp add*:*fsps-def fsp-def*)
apply (*rule-tac s = s* **and** *n = n* **and** *f = cmp b fa* **and** *g = cmp ba fa* **in**
Module.linear-comb-eq[*of N R f ' H*], *assumption*+,
simp add:*image-sub0*, *assumption*)
apply (*rule Pi-I*, *simp add*:*cmp-def*,
frule-tac x = xa **and** *f = fa* **and** *A = {j. j ≤ n}* **and** *B = a* **in**
funcset-mem, *simp*,
frule-tac m = fa xa **in** *Module.mHom-mem*[*of mdl M (fgs R M a) R N b*],
assumption+,
simp add:*mdl-carrier fgs-def*, *rule-tac c = fa xa* **in**
subsetD[*of a linear-span R M (carrier R) a*],
rule l-span-cont-H,
frule subset-trans[*of a aa H*], *assumption*+,
rule subset-trans[*of a H carrier M*], *assumption*+,
drule-tac x = fa xa **in** *bspec*, *assumption*,
frule-tac c = fa xa **in** *subsetD*[*of a H*], *assumption*+,
rotate-tac −2, *frule sym*, *thin-tac f (fa xa) = b (fa xa)*, *simp*)
apply (*rule Pi-I*, *simp add*:*cmp-def*,
frule-tac x = xa **and** *f = fa* **and** *A = {j. j ≤ n}* **and** *B = aa* **in**
funcset-mem, *simp*,
frule-tac c = fa xa **in** *subsetD*[*of aa H*], *assumption*+,
drule-tac x = fa xa **in** *bspec*, *assumption*,
rotate-tac −1, *frule sym*, *thin-tac f (fa xa) = ba (fa xa)*,
simp)
apply (*rule ballI*, *simp add*:*cmp-def*,
frule-tac x = j **and** *f = fa* **and** *A = {j. j ≤ n}* **and** *B = a* **in**
funcset-mem, *simp*)
apply (*frule-tac x = fa j* **in** *bspec*, *assumption*,
thin-tac ∀ z∈a. f z = b z,
frule-tac c = fa j **in** *subsetD*[*of a aa*], *assumption*+,
frule-tac x = fa j **in** *bspec*, *assumption*,
thin-tac ∀ z∈aa. f z = ba z)
apply (*rotate-tac −3*, *frule sym*,
thin-tac f (fa j) = b (fa j), *frule sym*,
thin-tac f (fa j) = ba (fa j), *simp*)

apply (*simp add*:*subset-trans*[*of aa H carrier M*])
apply (*frule subset-trans*[*of a aa H*], *assumption*+,
simp add:*subset-trans*[*of a H carrier M*])
done

lemma (**in** *Module*) *fsps-chain-boundTr2-2*:⟦*R module N*; *free-generator R M H*;
$f \in H \to$ *carrier N*; $C \subseteq$ *fsps R M N f H*;
$\forall\, a{\in}C.\ \forall\, b{\in}C.$ *fst a* $\subseteq$ *fst b* $\vee$ *fst b* $\subseteq$ *fst a*; $C \neq \{\}$; $(a,\ b) \in C$;
$x \in$ *fgs R M a*; $(a1,\ b1) \in C$; $x \in$ *fgs R M a1*⟧ $\Longrightarrow$ $b\ x = b1\ x$
apply (*frule-tac* $x = (a,\ b)$ **in** *bspec*, *assumption*,
thin-tac $\forall\, a{\in}C.\ \forall\, b{\in}C.$ *fst a* $\subseteq$ *fst b* $\vee$ *fst b* $\subseteq$ *fst a*,
frule-tac $x = (a1,\ b1)$ **in** *bspec*, *assumption*,
thin-tac $\forall\, ba{\in}C.$ *fst* $(a,\ b) \subseteq$ *fst ba* $\vee$ *fst ba* $\subseteq$ *fst* $(a,\ b)$, *simp*)
apply (*erule disjE*)
apply (*rule fsps-chain-boundTr2-1*[*of N H f C a b a1 b1*], *assumption*+)
apply (*frule fsps-chain-boundTr2-1*[*of N H f C a1 b1 a b*], *assumption*+)
apply (*rule sym*, *assumption*)
done

lemma (**in** *Module*) *fsps-chain-boundTr2*:$\bigwedge x.$ ⟦*R module N*; *free-generator R M H*;
$f \in H \to$ *carrier N*; $C \subseteq$ *fsps R M N f H*;
$\forall\, a{\in}C.\ \forall\, b{\in}C.$ *fst a* $\subseteq$ *fst b* $\vee$ *fst b* $\subseteq$ *fst a*;
$x \in$ (*fgs R M* $(\bigcup\{a.\ \exists\, b.\ (a,\ b) \in C\})$); $C \neq \{\}$⟧ $\Longrightarrow$
$(THE\ y.\ y \in carrier\ N \wedge (\exists\, a\ b.\ (a,\ b) \in C \wedge x \in fgs\ R\ M\ a \wedge y = b\ x)) \in$
$(carrier\ N) \wedge$
$(\exists\, a1\ b1.\ (a1,\ b1) \in C \wedge x \in fgs\ R\ M\ a1 \wedge$
$(THE\ y.\ y \in carrier\ N \wedge (\exists\, a\ b.\ (a,\ b) \in C \wedge x \in fgs\ R\ M\ a \wedge y = b\ x)) =$
$b1\ x)$
apply (*cut-tac sc-Ring*, *frule Ring.whole-ideal*[*of R*])
apply (*case-tac* $\bigcup\{a.\ \exists\, b.\ (a,\ b) \in C\} = \{\}$, *simp add*: *empty-fgs*,
frule nonempty-ex[*of C*], *erule exE*,
cut-tac $p = xa$ **in** *PairE-lemma*, (*erule exE*)+,
frule-tac $c = xa$ **in** *subsetD*[*of C fsps R M N f H*], *assumption*+, *simp*)
apply (*frule-tac* $a = xb$ **and** $b = y$ **in** *mem-fsps-snd-mHom*[*of N H f*],
assumption+, *simp add*:*subsetD*)
apply (*rename-tac a b a1 b1*)
apply (*subgoal-tac* $\exists!y.\ y \in carrier\ N \wedge (\exists\, a\ b.\ (a,\ b) \in C \wedge$
$\mathbf{0} \in (fgs\ R\ M\ a) \wedge y = b\ \mathbf{0})$,
subgoal-tac $(THE\ y.\ y \in carrier\ N \wedge (\exists\, a\ b.\ (a,\ b) \in C \wedge$
$\mathbf{0} \in fgs\ R\ M\ a \wedge y = b\ \mathbf{0})) \in carrier\ N \wedge (\exists\, a\ b.\ (a,\ b) \in C \wedge$
$\mathbf{0} \in fgs\ R\ M\ a \wedge$
$(THE\ y.\ y \in carrier\ N \wedge (\exists\, a\ b.\ (a,\ b) \in C \wedge \mathbf{0} \in fgs\ R\ M\ a \wedge y = b\ \mathbf{0})) =$
$b\ \mathbf{0})$)
prefer *2*
apply (*rule theI'*, *simp*, *simp*)
apply (*rule ex-ex1I*)
apply (*cut-tac* $a = a1$ **in** *fgs-submodule*)
apply (*frule-tac* $a = a1$ **and** $b = b1$ **in** *mem-fsps-fst-sub*[*of N H f*],

assumption+, *simp add*:*subsetD*, *frule free-generator-sub*[*of H*],
rule subset-trans[*of - H carrier M*], *assumption*+,
frule-tac H = fgs R M a1 **in** *mdl-is-module*,
frule-tac M = mdl M (*fgs R M a1*) **and** *f = b1* **in**
Module.mHom-0[*of - R N*], *assumption*+,
frule-tac H = fgs R M a1 **in** *submodule-inc-0*,
frule Module.module-inc-zero[*of N*],
thin-tac b1 ∈ *mHom R* (*mdl M* (*fgs R M a1*)) *N*,
thin-tac R module mdl M (*fgs R M a1*),
simp add:*mdl-def*)
apply (*rotate-tac* −*3*, *frule sym*, *thin-tac b1* $\mathbf{0} = \mathbf{0}_N$, *blast*)
apply ((*erule conjE*)+, (*erule exE*)+, (*erule conjE*)+)
apply (*cut-tac a = aa* **in** *fgs-submodule*,
frule-tac a = aa **and** *b = ba* **in** *mem-fsps-fst-sub*[*of N H f*],
assumption+, *simp add*:*subsetD*,
frule free-generator-sub, *rule subset-trans*, *assumption*+,
simp add:*subsetD*,
frule-tac H = fgs R M aa **in** *mdl-is-module*,
frule-tac M = mdl M (*fgs R M aa*) **and** *f = ba* **in**
Module.mHom-0[*of - R N*], *assumption*+,
frule-tac H = fgs R M aa **in** *submodule-inc-0*,
frule-tac c = (*aa*, *ba*) **in** *subsetD*[*of C fsps R M N f H*],
assumption+, *simp*,
frule-tac a = aa **and** *b = ba* **in** *mem-fsps-snd-mHom*[*of N H f*],
assumption+, *simp add*:*mdl-def*)
apply (*cut-tac a = ab* **in** *fgs-submodule*,
frule-tac a = ab **and** *b = bb* **in** *mem-fsps-fst-sub*[*of N H f*],
assumption+, *simp add*:*subsetD*,
frule free-generator-sub, *rule subset-trans*[*of - H carrier M*],
assumption+,
frule-tac H = fgs R M ab **in** *mdl-is-module*,
frule-tac M = mdl M (*fgs R M ab*) **and** *f = bb* **in**
Module.mHom-0[*of - R N*], *assumption*+)
apply (*fold mdl-def*,
frule-tac c = (*ab*, *bb*) **in** *subsetD*[*of C fsps R M N f H*],
assumption+,
frule-tac a = ab **and** *b = bb* **in** *mem-fsps-snd-mHom*[*of N H f*],
assumption+, *simp add*:*mdl-def*)

apply (*subgoal-tac* ∃!*y*. *y* ∈ *carrier N* ∧ (∃ *a b*. (*a*, *b*) ∈ *C* ∧
x ∈ (*fgs R M a*) ∧ *y* = *b x*),
subgoal-tac (*THE y*. *y* ∈ *carrier N* ∧ (∃ *a b*. (*a*, *b*) ∈ *C* ∧
x ∈ *fgs R M a* ∧ *y* = *b x*)) ∈ *carrier N* ∧ (∃ *a b*. (*a*, *b*) ∈ *C* ∧
x ∈ *fgs R M a* ∧
(*THE y*. *y* ∈ *carrier N* ∧ (∃ *a b*. (*a*, *b*) ∈ *C* ∧ *x* ∈ *fgs R M a* ∧ *y* = *b x*)) =
b x))
prefer *2*
apply (*rule theI'*, *simp*, *simp*)
apply (*rule ex-ex1I*)

apply (*frule-tac* $K = \bigcup\{a.\ \exists\, b.\ (a,\ b) \in C\}$ **and** $x = x$ **in** *mem-fgs-l-comb*)
apply (*rule subsetI*, *simp*,
thin-tac $\exists\, x.\ (\exists\, b.\ (x,\ b) \in C) \wedge x \neq \{\}$,
erule exE, *erule conjE*, *erule exE*,
frule-tac $a = xb$ **and** $b = b$ **in** *mem-fsps-fst-sub*[*of N H f*],
assumption+, *simp add*:*subsetD*,
frule free-generator-sub[*of H*], *simp add*:*subsetD*, *assumption*)
apply (*erule exE*, (*erule bexE*)+)
apply (*frule-tac fa* = *fa* **and** $n = n$ **in** *fsps-chain-boundTr1-1*[*of N H f C*],
assumption+)
apply (*frule-tac* $A = \bigcup\{a.\ \exists\, b.\ (a,\ b) \in C\}$ **in** *nonempty-ex*,
erule exE, *simp*, *assumption*+, *erule bexE*,
cut-tac $p = xa$ **in** *PairE-lemma*, (*erule exE*)+, *simp*)
apply (*frule-tac* $a = xb$ **and** $b = y$ **in** *mem-fsps-fst-sub*[*of N H f*],
assumption+, *simp add*:*subsetD*,
frule free-generator-sub[*of H*],
frule-tac $A = xb$ **in** *subset-trans*[*of* - *H carrier M*], *assumption*+)
apply (*frule-tac* $H = xb$ **and** $s = s$ **and** $n = n$ **and** $f = fa$ **in**
l-comb-mem-linear-span[*of carrier R*], *assumption*+,
rule Pi-I,
frule-tac $a = xc$ **and** $A = \{j.\ j \leq n\}$ **and** $f = fa$ **in**
mem-in-image2, *simp add*:*image-def*, *erule exE*, *blast*,
fold fgs-def)
apply (*frule-tac* $a = xb$ **and** $b = y$ **in** *mem-fsps-snd-mHom*[*of N H f*],
assumption+, *simp add*:*subsetD*,
frule-tac $a = xb$ **in** *fgs-submodule*,
frule-tac $H = fgs\ R\ M\ xb$ **in** *mdl-is-module*,
frule-tac $R = R$ **and** $M = mdl\ M\ (fgs\ R\ M\ xb)$ **and** $N = N$ **and**
$f = y$ **and** $m = x$ **in** *Module.mHom-mem*, *assumption*+,
simp add:*mdl-carrier*, *simp*, *blast*)
apply ((*erule conjE*)+, (*erule exE*)+, (*erule conjE*)+)
apply (*frule-tac* $x = (a,\ b)$ **in** *bspec*, *assumption*,
rotate-tac -1,
frule-tac $x = (aa,\ ba)$ **in** *bspec*, *assumption*,
thin-tac $\forall\, ba \in C.\ fst\ (a,\ b) \subseteq fst\ ba \vee fst\ ba \subseteq fst\ (a,\ b)$,
simp)
apply (*rule-tac* $a = a$ **and** $b = b$ **and** *?a1.0* = *aa* **and** *?b1.0* = *ba* **and** $x = x$
in
fsps-chain-boundTr2-2[*of N H f C*], *assumption*+)
done

lemma (**in** *Module*) *Un-C-submodule*:$[\![$*R module N*; *free-generator R M H*;
$f \in H \to carrier\ N$; $C \subseteq fsps\ R\ M\ N\ f\ H$; $C \neq \{\}$;
$\forall\, a \in C.\ \forall\, b \in C.\ fst\ a \subseteq fst\ b \vee fst\ b \subseteq fst\ a$$]\!] \Longrightarrow$
submodule R M $(fgs\ R\ M\ (\bigcup\{a.\ \exists\, b.\ (a,\ b) \in C\}))$
apply (*cut-tac sc-Ring*, *frule free-generator-sub*[*of H*],
frule Ring.whole-ideal)
apply (*subst fgs-def*)
apply (*rule linear-span-subModule*, *assumption*+)

apply (*rule subsetI*, *simp*)
apply (*erule exE*, *erule conjE*, *erule exE*,
frule-tac c = (*xa*, *b*) **in** *subsetD*[*of C fsps R M N f H*], *assumption*+,
frule-tac a = *xa* **and** *b* = *b* **in** *mem-fsps-fst-sub*[*of N H f*], *assumption*+,
frule free-generator-sub, *frule-tac A* = *xa* **in**
subset-trans[*of - H carrier M*], *assumption*+,
simp add:*subsetD*)
done

lemma (**in** *Module*) *Un-C-fgs-sub*:⟦*R module N*; *free-generator R M H*;
f ∈ *H* → *carrier N*; *C* ⊆ *fsps R M N f H*; *C* ≠ {};
∀ *a*∈*C*. ∀ *b*∈*C*. *fst a* ⊆ *fst b* ∨ *fst b* ⊆ *fst a*⟧ ⟹
⋃{*a*. ∃ *b*. (*a*, *b*) ∈ *C*} ⊆ *fgs R M* (⋃{*a*. ∃ *b*. (*a*, *b*) ∈ *C*})
apply (*simp add*:*fgs-def*)
apply (*rule l-span-cont-H*)
apply (*frule Un-C-submodule*[*of N H f C*], *assumption*+)
apply (*rule subsetI*, *simp*, *erule exE*, *erule conjE*, *erule exE*)
apply (*frule-tac a* = *xa* **and** *b* = *b* **in** *mem-fsps-fst-sub*[*of N H f*], *assumption*+,
simp add:*subsetD*,
frule free-generator-sub[*of H*],
frule-tac A = *xa* **and** *B* = *H* **and** *C* = *carrier M* **in** *subset-trans*,
assumption+, *simp add*:*subsetD*)
done

lemma (**in** *Module*) *Chain-3-supset*:⟦*R module N*; *free-generator R M H*;
f ∈ *H* → *carrier N*; *C* ⊆ *fsps R M N f H*; *C* ≠ {};
∀ *a*∈*C*. ∀ *b*∈*C*. *fst a* ⊆ *fst b* ∨ *fst b* ⊆ *fst a*; (*a1*, *b1*) ∈ *C*; (*a2*, *b2*) ∈ *C*;
(*a3*, *b3*) ∈ *C*⟧ ⟹ ∃(*g*, *h*)∈*C*. *a1* ⊆ *g* ∧ *a2* ⊆ *g* ∧ *a3* ⊆ *g*
proof −
assume (*a1*, *b1*) ∈ *C*
(*a2*, *b2*) ∈ *C*
(*a3*, *b3*) ∈ *C*
then have *A*: *a1* ∈ *fst* ‘ *C*
a2 ∈ *fst* ‘ *C*
a3 ∈ *fst* ‘ *C*
by (*simp-all add*: *image-iff*) *force*+
assume ∀ *a*∈*C*. ∀ *b*∈*C*. *fst a* ⊆ *fst b* ∨ *fst b* ⊆ *fst a*
with *A* **have** *a1* ⊆ *a2* ∨ *a2* ⊆ *a1*
a1 ⊆ *a3* ∨ *a3* ⊆ *a1*
a2 ⊆ *a3* ∨ *a3* ⊆ *a2*
by *auto*
with *A* **have** ∃ *a*∈*fst* ‘ *C*. *a1* ⊆ *a* ∧ *a2* ⊆ *a* ∧ *a3* ⊆ *a*
by *auto*
then obtain *a* **where** *a* ∈ *fst* ‘ *C* **and** *C*: *a1* ⊆ *a a2* ⊆ *a a3* ⊆ *a* **by** *blast*
then obtain *b* **where** (*a*, *b*) ∈ *C* **by** *auto*
with *C* **show** ∃(*a*, *b*)∈*C*. *a1* ⊆ *a* ∧ *a2* ⊆ *a* ∧ *a3* ⊆ *a*
by *auto*
qed

lemma (**in** *Module*) *fsps-chain-bound1*:⟦*R module N*; *free-generator R M H*;
f ∈ *H* → *carrier N*; *C* ⊆ *fsps R M N f H*;
∀ *a*∈*C*. ∀ *b*∈*C*. *fst a* ⊆ *fst b* ∨ *fst b* ⊆ *fst a*; *C* ≠ {}⟧ ⟹
(λ*x*∈(*fgs R M* (⋃{*a*. ∃ *b*. (*a*, *b*) ∈ *C*})). (*THE y*. *y* ∈ *carrier N* ∧
(∃ *a b*. (*a*, *b*) ∈ *C* ∧ *x* ∈ *fgs R M a* ∧ *y* = *b x*))) ∈
mHom R (*mdl M* (*fgs R M* (⋃{*a*. ∃ *b*. (*a*, *b*) ∈ *C*}))) *N*
apply (*cut-tac sc-Ring*, *frule Ring.whole-ideal*)
apply (*frule Un-C-submodule*[*of N H f C*], *assumption*+)
apply (*frule mdl-is-module*[*of fgs R M* (⋃{*a*. ∃ *b*. (*a*, *b*) ∈ *C*})])
apply (*rule Module.mHom-test*[*of mdl M* (*fgs R M* (⋃{*a*. ∃ *b*. (*a*, *b*) ∈ *C*})) *R N*

λ*x*∈*fgs R M* (⋃{*a*. ∃ *b*. (*a*, *b*) ∈ *C*}). *THE y*. *y* ∈ *carrier N* ∧
(∃ *a b*. (*a*, *b*) ∈ *C* ∧ *x* ∈ *fgs R M a* ∧ *y* = *b x*)], *assumption*+)
apply (*rule conjI*)
apply (*rule Pi-I*,
frule submodule-subset[*of fgs R M* (⋃{*a*. ∃ *b*. (*a*, *b*) ∈ *C*})],
simp add:*mdl-carrier*, *simp add*:*fsps-chain-boundTr2*)
apply (*rule conjI*)
apply (*simp add*:*mdl-carrier*)
apply (*rule conjI*)
apply (*rule ballI*)+
apply (*frule Module.module-is-ag*[*of mdl M* (*fgs R M* (⋃{*a*. ∃ *b*. (*a*, *b*) ∈ *C*}))
R])
apply (*frule-tac x* = *m* **and** *y* = *n* **in** *aGroup.ag-pOp-closed*[*of
mdl M* (*fgs R M* (⋃{*a*. ∃ *b*. (*a*, *b*) ∈ *C*}))], *assumption*+)
apply (*frule-tac x* = *m* **in** *fsps-chain-boundTr2*[*of N H f C*], *assumption*+,
simp add:*mdl-carrier*, *assumption*, *erule conjE*, (*erule exE*)+,
(*erule conjE*)+,
frule-tac x = *n* **in** *fsps-chain-boundTr2*[*of N H f C*], *assumption*+,
simp add:*mdl-carrier*, *assumption*, *erule conjE*, (*erule exE*)+,
(*erule conjE*)+,
frule-tac x = *m* ±$_{mdl\ M\ (fgs\ R\ M\ (\bigcup\{a.\ \exists\, b.\ (a,\ b) \in C\}))}$ *n* **in**
fsps-chain-boundTr2[*of N H f C*], *assumption*+,
simp add:*mdl-carrier*, *assumption*, *erule conjE*, (*erule exE*)+,
(*erule conjE*)+) **apply** (*simp add*:*mdl-carrier*)
apply (*thin-tac* (*THE y*.
y ∈ *carrier N* ∧ (∃ *a b*. (*a*, *b*) ∈ *C* ∧ *m* ∈ *fgs R M a* ∧ *y* = *b m*)) =
b1 m) **apply** (
thin-tac (*THE y*.
y ∈ *carrier N* ∧ (∃ *a b*. (*a*, *b*) ∈ *C* ∧ *n* ∈ *fgs R M a* ∧ *y* = *b n*)) =
b1a n) **apply** (
thin-tac (*THE y*.
y ∈ *carrier N* ∧
(∃ *a b*. (*a*, *b*) ∈ *C* ∧
m ±$_{mdl\ M\ (fgs\ R\ M\ (\bigcup\{a.\ \exists\, b.\ (a,\ b) \in C\}))}$ *n* ∈ *fgs R M a* ∧
y = *b* (*m* ±$_{mdl\ M\ (fgs\ R\ M\ (\bigcup\{a.\ \exists\, b.\ (a,\ b) \in C\}))}$ *n*))) =
b1b (*m* ±$_{mdl\ M\ (fgs\ R\ M\ (\bigcup\{a.\ \exists\, b.\ (a,\ b) \in C\}))}$ *n*))
apply (*thin-tac m* ±$_{mdl\ M\ (fgs\ R\ M\ (\bigcup\{a.\ \exists\, b.\ (a,\ b) \in C\}))}$ *n*

$\in$ *fgs R M* ($\bigcup\{a.\ \exists\, b.\ (a,\ b) \in C\}$),
thin-tac b1b ($m \pm_{mdl\ M\ (fgs\ R\ M\ (\bigcup\{a.\ \exists\, b.\ (a,\ b) \in C\}))}\ n$) $\in$ *carrier N*,
simp add:*mdl-def*, *fold mdl-def*)
apply (*cut-tac ?a1.0 = a1* **and** *?b1.0 = b1* **and** *?a2.0 = a1a* **and** *?b2.0 = b1a* **and**
?a3.0 = a1b **and** *?b3.0 = b1b* **in** *Chain-3-supset*[*of N H f C*], *assumption*+)
apply (*erule bexE*)
apply (*cut-tac p = x* **in** *PairE-lemma*, (*erule exE*)+, *simp*, (*erule conjE*)+)
apply (*frule-tac J = a1b* **and** *K = xa* **in** *fgs-mono*[*of H*], *assumption*+,
rule-tac a = xa **and** *b = y* **in** *mem-fsps-fst-sub*[*of N H f*], *assumption*+,
simp add:*subsetD*,
frule-tac J = a1 **and** *K = xa* **in** *fgs-mono*[*of H*], *assumption*+,
rule-tac a = xa **and** *b = y* **in** *mem-fsps-fst-sub*[*of N H f*], *assumption*+,
simp add:*subsetD*,
frule-tac J = a1a **and** *K = xa* **in** *fgs-mono*[*of H*], *assumption*+,
rule-tac a = xa **and** *b = y* **in** *mem-fsps-fst-sub*[*of N H f*], *assumption*+,
simp add:*subsetD*)
apply (*frule-tac c = m* **and** *A = fgs R M a1* **and** *B = fgs R M xa* **in** *subsetD*,
assumption+,
frule-tac c = n **and** *A = fgs R M a1a* **and** *B = fgs R M xa* **in** *subsetD*,
assumption+,
frule-tac c = m ± n **and** *A = fgs R M a1b* **and** *B = fgs R M xa* **in**
subsetD, *assumption*+)
apply (*frule-tac a = a1* **and** *b = b1* **and** *aa = xa* **and** *ba = y* **in**
fsps-chain-boundTr2-1[*of N H f C*], *assumption*+, *simp*,
frule-tac a = a1a **and** *b = b1a* **and** *aa = xa* **and** *ba = y* **in**
fsps-chain-boundTr2-1[*of N H f C*], *assumption*+, *simp*,
frule-tac a = a1b **and** *b = b1b* **and** *aa = xa* **and** *ba = y* **in**
fsps-chain-boundTr2-1[*of N H f C*], *assumption*+, *simp*)
apply (*thin-tac submodule R M* (*fgs R M* ($\bigcup\{a.\ \exists\, b.\ (a,\ b) \in C\}$)),
thin-tac R module mdl M (*fgs R M* ($\bigcup\{a.\ \exists\, b.\ (a,\ b) \in C\}$)),
thin-tac m $\in$ *fgs R M* ($\bigcup\{a.\ \exists\, b.\ (a,\ b) \in C\}$),
thin-tac n $\in$ *fgs R M* ($\bigcup\{a.\ \exists\, b.\ (a,\ b) \in C\}$),
thin-tac aGroup (*mdl M* (*fgs R M* ($\bigcup\{a.\ \exists\, b.\ (a,\ b) \in C\}$))),
thin-tac y m $\in$ *carrier N*, *thin-tac* (*a1*, *b1*) $\in$ *C*,
thin-tac m $\in$ *fgs R M a1*, *thin-tac y n* $\in$ *carrier N*,
thin-tac (*a1a*, *b1a*) $\in$ *C*, *thin-tac n* $\in$ *fgs R M a1a*,
thin-tac (*a1b*, *b1b*) $\in$ *C*, *thin-tac m ± n* $\in$ *fgs R M a1b*,
thin-tac a1 $\subseteq$ *xa*, *thin-tac a1a* $\subseteq$ *xa*,
thin-tac a1b $\subseteq$ *xa*, *thin-tac fgs R M a1* $\subseteq$ *fgs R M xa*,
thin-tac fgs R M a1a $\subseteq$ *fgs R M xa*,
thin-tac b1b (*m ± n*) = *y* (*m ± n*),
thin-tac b1 m = *y m*, *thin-tac b1a n* = *y n*)
apply (*cut-tac a = xa* **in** *fgs-submodule*,
frule-tac c = (*xa*, *y*) **in** *subsetD*[*of C fsps R M N f H*],
assumption+,
simp add:*mem-fsps-fst-sub1*)
apply (*frule-tac H = fgs R M xa* **in** *mdl-is-module*)
apply (*frule-tac a = xa* **and** *b = y* **in** *mem-fsps-snd-mHom*[*of N H f*],

assumption+, *simp add*:*subsetD*)
apply (*frule-tac* $R = R$ **and** $M = mdl\ M\ (fgs\ R\ M\ xa)$ **and** $N = N$ **and** $f = y$ **and**
$m = m$ **and** $n = n$ **in** *Module.mHom-add*, *assumption*+,
simp add:*mdl-carrier*, *simp add*:*mdl-carrier*, *simp add*:*mdl-def*)

apply (*rule ballI*)+
apply (*frule-tac* $a = a$ **and** $m = m$ **in** *Module.sc-mem*[*of*
$mdl\ M\ (fgs\ R\ M\ (\bigcup\{a.\ \exists\, b.\ (a,\ b) \in C\}))\ R$], *assumption*+)
apply (*frule-tac* $x = m$ **in** *fsps-chain-boundTr2*[*of N H f C*], *assumption*+,
simp add:*mdl-carrier*, *assumption*, *erule conjE*, (*erule exE*)+,
(*erule conjE*)+,
frule-tac $x = a\ \cdot_{s\,mdl\ M\ (fgs\ R\ M\ (\bigcup\{a.\ \exists\, b.\ (a,\ b) \in C\}))}\ m$ **in**
fsps-chain-boundTr2[*of N H f C*], *assumption*+,
simp add:*mdl-carrier*, *assumption*, *erule conjE*, (*erule exE*)+,
(*erule conjE*)+)
apply (*simp add*:*mdl-carrier*)
apply (*thin-tac* (*THE y*.
$y \in carrier\ N \wedge (\exists\, a\ b.\ (a,\ b) \in C \wedge m \in fgs\ R\ M\ a \wedge y = b\ m)) =$
$b1\ m$,
thin-tac $b1a\ (a\ \cdot_{s\,mdl\ M\ (fgs\ R\ M\ (\bigcup\{a.\ \exists\, b.\ (a,\ b) \in C\}))}\ m) \in carrier\ N$,
thin-tac (*THE y*.
$y \in carrier\ N \wedge$
$(\exists\, aa\ b.$
$(aa,\ b) \in C \wedge$
$a\ \cdot_{s\,mdl\ M\ (fgs\ R\ M\ (\bigcup\{a.\ \exists\, b.\ (a,\ b) \in C\}))}\ m \in fgs\ R\ M\ aa \wedge$
$y = b\ (a\ \cdot_{s\,mdl\ M\ (fgs\ R\ M\ (\bigcup\{a.\ \exists\, b.\ (a,\ b) \in C\}))}\ m))) =$
$b1a\ (a\ \cdot_{s\,mdl\ M\ (fgs\ R\ M\ (\bigcup\{a.\ \exists\, b.\ (a,\ b) \in C\}))}\ m)$,
thin-tac $a\ \cdot_{s\,mdl\ M\ (fgs\ R\ M\ (\bigcup\{a.\ \exists\, b.\ (a,\ b) \in C\}))}\ m$
$\in fgs\ R\ M\ (\bigcup\{a.\ \exists\, b.\ (a,\ b) \in C\})$,
thin-tac submodule R M $(fgs\ R\ M\ (\bigcup\{a.\ \exists\, b.\ (a,\ b) \in C\}))$,
thin-tac R module mdl M $(fgs\ R\ M\ (\bigcup\{a.\ \exists\, b.\ (a,\ b) \in C\}))$)
apply (*simp add*:*mdl-def*,
thin-tac $m \in fgs\ R\ M\ (\bigcup\{a.\ \exists\, b.\ (a,\ b) \in C\})$,
thin-tac $b1\ m \in carrier\ N$)
apply (*frule-tac* $x = (a1,\ b1)$ **in** *bspec*, *assumption*,
rotate-tac -1,
drule-tac $x = (a1a,\ b1a)$ **in** *bspec*, *assumption*,
simp)
apply (*erule disjE*,
frule-tac $J = a1$ **and** $K = a1a$ **in** *fgs-mono*[*of H*], *assumption*+,
rule-tac $a = a1a$ **and** $b = b1a$ **in** *mem-fsps-fst-sub*[*of N H f*],
assumption+, *simp add*:*subsetD*,
frule-tac $c = m$ **and** $A = fgs\ R\ M\ a1$ **and** $B = fgs\ R\ M\ a1a$ **in** *subsetD*,
assumption+)
apply (*cut-tac* $a = a1a$ **in** *fgs-submodule*,
frule-tac $c = (a1a,\ b1a)$ **in** *subsetD*[*of C fsps R M N f H*],

assumption+, *simp add*:*mem-fsps-fst-sub1*,
frule-tac H = fgs R M a1a **in** *mdl-is-module*,
frule-tac a = a1a **and** *b = b1a* **in** *mem-fsps-snd-mHom*[*of N H f*],
assumption+, *simp add*:*subsetD*)
apply (*frule-tac R = R* **and** *M = mdl M* (*fgs R M a1a*) **and** *N = N* **and** *m = m*
and *f = b1a* **and** *a = a* **in** *Module.mHom-lin*, *assumption+*,
simp add:*mdl-carrier*, *assumption+*, *simp add*:*mdl-def*,
fold mdl-def)
apply (*frule-tac a = a1* **and** *b = b1* **and** *?a1.0 = a1a* **and** *?b1.0 = b1a* **and** *x = m*
in *fsps-chain-boundTr2-2*[*of N H f C*], *assumption+*, *simp*)

apply (*frule-tac J = a1a* **and** *K = a1* **in** *fgs-mono*[*of H*], *assumption+*,
rule-tac a = a1 **and** *b = b1* **in** *mem-fsps-fst-sub*[*of N H f*],
assumption+, *simp add*:*subsetD*,
frule-tac c = a $\cdot_s$ *m* **and** *A = fgs R M a1a* **and** *B = fgs R M a1* **in**
subsetD, *assumption+*)
apply (*frule-tac a = a1a* **and** *b = b1a* **and** *?a1.0 = a1* **and** *?b1.0 = b1* **and**
x = a $\cdot_s$ *m* **in** *fsps-chain-boundTr2-2*[*of N H f C*], *assumption+*, *simp*)
apply (*cut-tac a = a1* **in** *fgs-submodule*,
frule-tac c = (a1, b1) **in** *subsetD*[*of C fsps R M N f H*],
assumption+, *simp add*:*mem-fsps-fst-sub1*,
frule-tac H = fgs R M a1 **in** *mdl-is-module*,
frule-tac a = a1 **and** *b = b1* **in** *mem-fsps-snd-mHom*[*of N H f*],
assumption+, *simp add*:*subsetD*)
apply (*frule-tac R = R* **and** *M = mdl M* (*fgs R M a1*) **and** *N = N* **and** *m = m*
and *f = b1* **and** *a = a* **in** *Module.mHom-lin*, *assumption+*,
simp add:*mdl-carrier*, *assumption+*, *simp add*:*mdl-def*)
done

lemma (**in** *Module*) *fsps-chain-bound2*:⟦*R module N*; *free-generator R M H*;
f ∈ *H* → *carrier N*; *C* ⊆ *fsps R M N f H*; *C* ≠ {};
∀ *a*∈*C*. ∀ *b*∈*C*. *fst a* ⊆ *fst b* ∨ *fst b* ⊆ *fst a*⟧ ⟹
∀ *y*∈(⋃{*a*. ∃ *b*. (*a*, *b*) ∈ *C*}). (λ*x*∈(*fgs R M* (⋃{*a*. ∃ *b*. (*a*, *b*) ∈ *C*})).
(*THE y*. *y* ∈ *carrier N* ∧ (∃ *a b*. (*a*, *b*) ∈ *C* ∧ *x* ∈ *fgs R M a* ∧ *y* = *b x*))) *y* = *f y*
apply (*rule ballI*)
apply (*subgoal-tac y* ∈ *fgs R M* (⋃{*a*. ∃ *b*. (*a*, *b*) ∈ *C*}))
prefer *2*
apply *simp*
apply (*erule exE*, *erule conjE*, *erule exE*)
apply (*frule Un-C-fgs-sub*[*of N H f C*], *assumption+*)
apply (*rule-tac c = y* **in** *subsetD*[*of* ⋃{*a*. ∃ *b*. (*a*, *b*) ∈ *C*}
fgs R M (⋃{*a*. ∃ *b*. (*a*, *b*) ∈ *C*})], *assumption+*)
apply (*simp*, *blast*)

apply *simp*

apply (*frule-tac* $x = y$ **in** *fsps-chain-boundTr2*[*of N H f C*], *assumption*+)
apply (*erule conjE*, (*erule exE*)+, *erule conjE*, *erule exE*, (*erule conjE*)+)
apply (*simp*,
thin-tac (*THE ya*. $ya \in$ *carrier N* $\wedge$
$(\exists\, a\ b.\ (a,\ b) \in C \wedge y \in$ *fgs R M a* $\wedge\ ya = b\ y)) = b1\ y$)
apply (*frule-tac* $c = (x,\ b)$ **in** *subsetD*[*of C fsps R M N f H*],
assumption+,
simp add:*fsps-def fsp-def*, (*erule conjE*)+,
thin-tac $\forall\, z{\in}x.\ f\ z = b\ z$)
apply (*frule-tac* $x = y$ **and** $a = x$ **and** $b = b$ **and** *?a1.0* = *a1* **and** *?b1.0* = *b1*
in
fsps-chain-boundTr2-2[*of N H f C*], *assumption*+,
simp add:*fsps-def fsp-def*, *assumption*+, *subst fgs-def*,
rule-tac $c = y$ **and** $A = x$ **and** $B =$ *linear-span R M* (*carrier R*) x
in *subsetD*,
rule l-span-cont-H,
frule free-generator-sub[*of H*], *rule subset-trans*[*of - H carrier M*],
assumption+, *simp*)
done

lemma (**in** *Module*) *od-fm-fun-Chain*:⟦*R module N*; *free-generator R M H*;
$f \in H \to$ *carrier N*; *Algebra2.Chain* (*od-fm-fun R M N f H*) *C*; $C \neq \{\}$⟧ $\Longrightarrow$

$\forall\, a{\in}C.\ \forall\, b{\in}C.$ *fst a* $\subseteq$ *fst b* $\vee$ *fst b* $\subseteq$ *fst a*
apply (*frule Order-od-fm-fun*[*of N H f*], *assumption*+)
apply (*rule ballI*)+
apply (*simp add*:*Algebra2.Chain-def*, *erule conjE*, *simp add*:*Torder-def*, *erule conjE*,
simp add:*Torder-axioms-def*)
apply (*simp add*:*Order.Iod-carrier*)
apply (*auto simp add*: *Iod-def od-fm-fun-def ole-def*)
apply *blast*
done

lemma (**in** *Module*) *od-fm-fun-inPr0*:⟦*R module N*; *free-generator R M H*;
$f \in H \to$ *carrier N*; *Algebra2.Chain* (*od-fm-fun R M N f H*) *C*; $C \neq \{\}$;
$\exists\, b.\ (y,\ b) \in C$; $z \in y$⟧ $\Longrightarrow z \in$ *fgs R M* $(\bigcup\{a.\ \exists\, b.\ (a,\ b) \in C\})$
apply (*frule Un-C-fgs-sub*[*of N H f C*], *assumption*+)
apply (*simp add*:*Algebra2.Chain-def od-fm-fun-def*, *assumption*)
apply (*simp add*:*od-fm-fun-Chain*)
apply (*rule-tac subsetD*[*of* $\bigcup\{a.\ \exists\, b.\ (a,\ b) \in C\}$
fgs R M $(\bigcup\{a.\ \exists\, b.\ (a,\ b) \in C\})$ *z*], *assumption*+)
apply (*simp*, *blast*)
done

lemma (**in** *Module*) *od-fm-fun-indPr1*: ⟦*R module N*; *free-generator R M H*;
$f \in H \to$ *carrier N*; *Algebra2.Chain* (*od-fm-fun R M N f H*) *C*; $C \neq \{\}$⟧ $\Longrightarrow$
$(\bigcup\{a.\ \exists\, b.\ (a,\ b) \in C\},$
$\lambda x \in$ *fgs R M* $(\bigcup\{a.\ \exists\, b.\ (a,\ b) \in C\})$. *THE y*. $y \in$ *carrier N* $\wedge$

$(\exists\, a\ b.\ (a,\ b) \in C \wedge x \in \mathit{fgs}\ R\ M\ a \wedge y = b\ x)) \in \mathit{fsps}\ R\ M\ N\ f\ H$
apply (*simp add*:*fsps-def fsp-def*, *rule conjI*)
apply (*rule fsps-chain-bound1*[*of N H f C*], *assumption*+)
apply (*simp add*:*od-fm-fun-def Algebra2.Chain-def*)
apply (*simp add*:*od-fm-fun-Chain*, *assumption*)
apply (*rule conjI*)
apply (*rule allI*, *rule impI*)
apply (*rule ballI*)
apply (*simp add*:*od-fm-fun-inPr0*[*of N H f C*])
apply (*frule fsps-chain-bound2*[*of N H f C*], *assumption*+,
simp add:*Algebra2.Chain-def od-fm-fun-def*, *assumption*)
apply (*simp add*:*od-fm-fun-Chain*)
apply (*drule-tac x* = *z* **in** *bspec*)
apply (*simp*, *blast*)
apply (*simp add*:*od-fm-fun-inPr0*)
apply (*rule subsetI*, *simp*, *erule exE*, *erule conjE*, *erule exE*)
apply (*frule Order-od-fm-fun*[*of N H f*], *assumption*+,
frule Order.Chain-sub[*of od-fm-fun R M N f H C*], *assumption*,
thin-tac Algebra2.Chain (*od-fm-fun R M N f H*) *C*,
thin-tac Order (*od-fm-fun R M N f H*))
apply (*simp add*:*od-fm-fun-def*)
apply (*frule-tac a* = *xa* **and** *b* = *b* **in** *mem-fsps-fst-sub*[*of N H f*], *assumption*+,
simp add:*subsetD*, *simp add*:*subsetD*)
done

lemma (**in** *Module*) *od-fm-fun-indPr2*: $[\![R$ *module* N; *free-generator* $R\ M\ H$;
$f \in H \to$ *carrier* N; *Algebra2.Chain* (*od-fm-fun R M N f H*) C; $C \neq \{\}]\!] \Longrightarrow$
$^{ub}_{\mathit{od\text{-}fm\text{-}fun}\ R\ M\ N\ f\ H}\ C\ (\bigcup\{a.\ \exists\, b.\ (a,\ b) \in C\},$
$\lambda x \in \mathit{fgs}\ R\ M\ (\bigcup\{a.\ \exists\, b.\ (a,\ b) \in C\}).\ \mathit{THE}\ y.\ y \in \mathit{carrier}\ N \wedge$
$(\exists\, a\ b.\ (a,\ b) \in C \wedge x \in \mathit{fgs}\ R\ M\ a \wedge y = b\ x))$
apply (*frule Order-od-fm-fun*[*of N H f*], *assumption*+,
simp add:*upper-bound-def*)

apply (*simp add*:*od-fm-fun-def*, *fold od-fm-fun-def*,
frule od-fm-fun-indPr1[*of N H f C*], *assumption*+, *simp*)

apply (*rule ballI*,
subst od-fm-fun-def, *subst ole-def*, *simp*)
apply (*rule conjI*)
apply (*frule Order.Chain-sub*[*of od-fm-fun R M N f H C*], *assumption*,
simp add:*od-fm-fun-carrier*, *simp add*:*subsetD*)
apply (*unfold split-paired-all*,
rule subsetI, *simp*, *blast*)
done

lemma (**in** *Module*) *od-fm-fun-inductive*:$[\![R$ *module* N; *free-generator* $R\ M\ H$;
$f \in H \to$ *carrier* $N]\!] \Longrightarrow$ *inductive-set* (*od-fm-fun R M N f H*)
apply (*frule Order-od-fm-fun*[*of N H f*], *assumption*+)
apply (*simp add*:*inductive-set-def*)

apply (*rule allI*, *rule impI*)
apply (*case-tac* $C \neq \{\}$)
 apply (*frule-tac* $C = C$ **in** *od-fm-fun-indPr2*[*of N H f*], *assumption*+)
 apply *blast*
apply *simp*
 apply (*simp add*:*upper-bound-def*)
 apply (*subst od-fm-fun-def*, *simp*)
 apply (*frule empty-fsp*[*of N H f*], *assumption*+, *blast*)
done

lemma (**in** *Module*) *sSum-eq*:⟦*R module N*; *free-generator R M H*; $H1 \subseteq H$;
    $h \in H - H1$⟧ ⟹ (*fgs R M H1*) ∓ (*fgs R M* $\{h\}$) = *fgs R M* ($H1 \cup \{h\}$)
apply (*cut-tac sc-Ring*, *frule Ring.whole-ideal*)
apply (*frule free-generator-sub*)
 apply (*frule subset-trans*[*of H1 H carrier M*], *assumption*+,
       frule fgs-sub-carrier[*of H1*])
 apply (*cut-tac subset-trans*[*of* $\{h\}$ $H - H1$ *carrier M*],
       frule fgs-sub-carrier[*of* $\{h\}$])
 apply (*simp add*:*set-sum*[*of fgs R M H1 fgs R M* $\{h\}$])
 apply (*rule equalityI*)
  apply (*rule subsetI*, *simp*, (*erule bexE*)+)
  apply (*case-tac H1* = $\{\}$, *simp add*:*empty-fgs*)
  apply (*frule-tac* $c = k$ **in** *subsetD*[*of fgs R M* $\{h\}$ *carrier M*], *assumption*+)
  apply (*simp add*:*ag-l-zero*)
  apply (*cut-tac fgs-mono*[*of H H1 insert h H1*],
       frule-tac $c = ha$ **in** *subsetD*[*of fgs R M H1 fgs R M* (*insert h H1*)],
        assumption+,
       cut-tac fgs-mono[*of H* $\{h\}$ *insert h H1*],
       frule-tac $c = k$ **in** *subsetD*[*of fgs R M* $\{h\}$ *fgs R M* (*insert h H1*)],
        assumption+)
  apply (*simp add*:*fgs-def*,
       cut-tac sc-Ring, *frule Ring.whole-ideal*,
       rule linear-span-pOp-closed, *assumption*+,
       rule subsetI, *simp*, *erule disjE*, *simp*, *simp add*:*subsetD*, *assumption*+,
       rule subsetI, *simp*, *rule subsetI*, *simp*, *erule disjE*, *simp*,
       simp add:*subsetD*, *assumption*+, *rule subsetI*, *simp*,
       rule subsetI, *simp*, *erule disjE*, *simp*, *simp add*:*subsetD*)
 apply (*rule subsetI*)

 apply (*cut-tac* $x = x$ **in** *mem-fgs-l-comb*[*of insert h H1*],
       simp,
       rule subsetI, *simp*, *erule disjE*, *simp*, *simp add*:*subsetD*, *assumption*+)
 apply (*erule exE*, (*erule bexE*)+)
 apply (*case-tac f* ‘ $\{j.\ j \leq n\} \subseteq H1$)
 apply (*frule-tac* $s = s$ **and** $n = n$ **and** $f = f$ **in**
        l-comb-mem-linear-span[*of carrier R H1*], *assumption*+)
  apply (*rule Pi-I*)
  apply (*frule-tac f* = *f* **and** $A = \{j.\ j \leq n\}$ **and** B = *insert h H1* **and**
       $a = xa$ **in** *mem-in-image*, *assumption*+, *simp add*:*subsetD*)

apply (*fold fgs-def*)
apply (*frule fgs-sub-carrier*[*of H1*])
apply (*frule-tac c = l-comb R M n s f* **in** *subsetD*[*of fgs R M H1*
carrier M], *assumption*+)
apply (*frule-tac t = l-comb R M n s f* **in** *ag-r-zero*[*THEN sym*], *simp*)
apply (*cut-tac fgs-submodule*[*of* {*h*}],
frule submodule-inc-0[*of fgs R M* {*h*}], *blast*,
rule subsetI, *simp*)

apply (*frule-tac f = f* **and** *A* = {*j*. *j* ≤ *n*} **and** *B = insert h H1* **in**
image-sub0)
apply (*frule-tac Y = f* ‘ {*j*. *j* ≤ *n*} **and** *a = h* **and** *X = H1* **in** *sub-inserted1*,
assumption+, *erule conjE*,
thin-tac ¬ *f* ‘ {*j*. *j* ≤ *n*} ⊆ *H1*,
thin-tac f ‘ {*j*. *j* ≤ *n*} ⊆ *insert h H1*, *erule conjE*)
apply (*cut-tac H = insert h H1* **and** *f = f* **and** *n = n* **and** *s = s* **in**
unique-expression2)
apply (*rule subsetI*, *simp*, *erule disjE*, *simp*, *simp add*:*subsetD*)
apply *assumption*+
apply ((*erule exE*)+, (*erule conjE*)+, *simp*)
apply (*simp add*:*bij-to-def*, *erule conjE*,
simp add:*surj-to-def*)
apply (*rotate-tac* −*2*, *frule sym*, *thin-tac g* ‘ {*j*. *j* ≤ *m*} = *f* ‘ {*j*. *j* ≤ *n*},
simp, *thin-tac f* ‘ {*j*. *j* ≤ *n*} = *g* ‘ {*j*. *j* ≤ *m*}, *simp add*:*image-def*,
erule exE,
thin-tac f ∈ {*j*. *j* ≤ *n*} → *insert h H1*,
thin-tac s ∈ {*j*. *j* ≤ *n*} → *carrier R*,
thin-tac l-comb R M n s f = l-comb R M m t g, *erule conjE*)

apply (*case-tac m = 0*, *simp*,
frule-tac a = H1 **in** *fgs-submodule*,
frule submodule-inc-0[*of fgs R M H1*],
cut-tac a = insert (*g 0*) *H1* **in** *fgs-submodule*,
rule subsetI, *simp*, *erule disjE*, *simp*, *simp add*:*subsetD*,
frule-tac H = fgs R M (*insert* (*g 0*) *H1*) **in** *submodule-subset*,
frule-tac c = l-comb R M 0 t g **and** *A = fgs R M* (*insert* (*g 0*) *H1*)
and *B = carrier M* **in** *subsetD*, *assumption*+,
frule-tac t = l-comb R M 0 t g **in** *ag-l-zero*[*THEN sym*],
subgoal-tac l-comb R M 0 t g ∈ *fgs R M* {*g 0*}, *blast*)
apply (*subst fgs-def*,
cut-tac s = t **and** *n = m* **and** *f = g* **in**
l-comb-mem-linear-span[*of carrier R* {*h*}], *assumption*+,
rule subsetI, *simp*, *simp*, *simp*,
simp, *simp*)

apply (*cut-tac f = g* **and** *n = m* − *Suc 0* **and** *s = t* **and** *l = xa* **in**
unique-expression3[*of insert h H1*],
rule subsetI, *simp*, *erule disjE*, *simp*, *simp add*:*subsetD*)
apply *simp*+

apply (*rule contrapos-pp*, *simp*+, *simp add*:*image-def*)
apply (*erule bexE*, *simp add*:*inj-on-def*)
apply (*drule-tac a* = *xa* **in** *forall-spec*, *simp*)
apply (*drule-tac a* = *xb* **in** *forall-spec*,
erule conjE, *assumption*, *simp*)
apply ((*erule exE*)+, (*erule conjE*)+, *simp*)
apply (*thin-tac g* ∈ {*j*. *j* ≤ *m*} → *insert* (*g xa*) *H1*,
thin-tac t ∈ {*j*. *j* ≤ *m*} → *carrier R*,
thin-tac l-comb R M m t g = *l-comb R M ma ta ga*,
thin-tac inj-on g {*j*. *j* ≤ *m*},
thin-tac g xa ∈ *carrier M*,
thin-tac fgs R M {*g xa*} ⊆ *carrier M*)
apply (*rotate-tac 13*, *frule sym*, *thin-tac h* = *g xa*, *simp*)
apply (*thin-tac g xa* = *h*, *thin-tac 0* < *m*, *thin-tac xa* ≤ *m*,
thin-tac ta ma = *t xa*, *simp*)
apply (*rename-tac x g m t*)
apply (*case-tac m* = *0*, *simp*,
frule-tac a = *H1* **in** *fgs-submodule*,
frule submodule-inc-0[*of fgs R M H1*],
cut-tac a = *insert h H1* **in** *fgs-submodule*,
rule subsetI, *simp*, *erule disjE*, *simp*, *simp add*:*subsetD*,
frule subset-trans[*of H1 H carrier M*], *assumption*+,
simp add:*subsetD*,
frule-tac H = *fgs R M* (*insert h H1*) **in** *submodule-subset*,
frule-tac c = *l-comb R M 0 t g* **and** *A* = *fgs R M* (*insert h H1*)
and *B* = *carrier M* **in** *subsetD*, *assumption*+,
frule-tac t = *l-comb R M 0 t g* **in** *ag-l-zero*[*THEN sym*],
subgoal-tac l-comb R M 0 t g ∈ *fgs R M* {*h*}, *blast*)
apply (*subst fgs-def*,
cut-tac s = *t* **and** *n* = *0* **and** *f* = *g* **in**
l-comb-mem-linear-span[*of carrier R* {*h*}],
assumption, *rule subsetI*, *simp*, *simp add*:*subsetD*,
simp, *simp*, *assumption*)
apply (*subgoal-tac l-comb R M m t g* = *l-comb R M* (*Suc* (*m* − *Suc 0*)) *t g*,
simp del:*Suc-pred*,
frule insert-sub[*of H1 H h*], *assumption*+,
frule subset-trans[*of insert h H1 H carrier M*], *assumption*+,
subst l-comb-Suc[*of insert h H1 carrier R*], *assumption*+,
(*subst Suc-pred*, *assumption*,
thin-tac l-comb R M m t g = *l-comb R M* (*Suc* (*m* − *Suc 0*)) *t g*,
simp)+)
apply (*subgoal-tac l-comb R M* (*m* − *Suc 0*) *t g* ∈ *fgs R M H1*,
subgoal-tac t m ·$_s$ *h* ∈ *fgs R M* {*h*}, *blast*,
subst fgs-def,
rule-tac h = *h* **and** *a* = *t m* **in**
sc-linear-span[*of carrier R* {*h*}], *assumption*+,
rule subsetI, *simp*, *simp add*:*Pi-def*, *simp*,
cut-tac a = {*h*} **in** *fgs-submodule*, *rule subsetI*, *simp*,
cut-tac s = *t* **and** *n* = *m* − *Suc 0* **and** *f* = *g* **in**

```
            l-comb-mem-linear-span[of carrier R H1], assumption,
        simp add:subset-trans[of H1 H carrier M],
        rule func-pre, simp,
        rule Pi-I, simp,
        frule-tac f = g and A = {k. k ≤ m} and B = insert h H1
              and x = xa in funcset-mem, simp, simp, simp add:inj-on-def)
  apply (erule disjE)
  apply (drule-tac a = m in forall-spec, simp,
        drule-tac a = xa in forall-spec, simp, simp, simp)
  apply (cut-tac n1 = xa and m1 = xa − Suc 0 in Suc-le-mono[THEN sym],
           simp, simp add:fgs-def, simp)
  apply (rule subsetI, simp)
  apply (rule subsetI, simp add:subsetD)
done

definition
  monofun :: [('r, 'm) Ring-scheme, ('a, 'r, 'm1) Module-scheme,
          ('b, 'r, 'm2) Module-scheme, 'a ⇒ 'b, 'a set, 'a] ⇒ ('a ⇒ 'b) where
  monofun R M N f H h = (λx∈fgs R M {h}.
        (THE y. (∃ r∈carrier R. x = r ·_sM h ∧  y = r ·_sN (f h))))

lemma (in Module) fgs-single-span:⟦h ∈ carrier M; x ∈ (fgs R M {h})⟧ ⟹
           ∃ a∈carrier R. x = a ·_s h
apply (simp add:fgs-def linear-span-def)
apply (erule exE, (erule bexE)+)
 apply (cut-tac sc-Ring, frule Ring.whole-ideal)
 apply (frule-tac A = carrier R and z = h and h = f and n = n and t = s in
       single-span, assumption+)
 apply (erule bexE, simp add:l-comb-def del: Pi-split-insert-domain)
apply (frule-tac f = sa and A = {0} and B = carrier R and
                x = 0 in funcset-mem,  simp, blast)
done

lemma (in Module) monofun-mHomTr:⟦h ∈ H; free-generator R M H;
      a ∈ carrier R; r ∈ carrier R; a ·_s h = r ·_s h⟧ ⟹ a = r
apply (cut-tac sc-Ring, frule free-generator-sub,
       frule subsetD[of H carrier M h], assumption+)
apply (frule-tac a = a and m = h in sc-mem, assumption+,
       frule-tac a = r and m = h in sc-mem, assumption+)
apply (frule ag-eq-diffzero[of a ·_s h r ·_s h], assumption+,
       simp only:sc-minus-am1, thin-tac (a ·_s h = r ·_s h) = (0 = 0))
apply (frule Ring.ring-is-ag[of R],
       frule aGroup.ag-mOp-closed[of R r], assumption+)
       apply (simp add:sc-l-distr[THEN sym])
       apply (frule free-gen-coeff-zero[of H h a ±_R −_aR r], assumption+)
       apply (rule aGroup.ag-pOp-closed, assumption+)
apply (simp add:aGroup.ag-eq-diffzero)
done
```

lemma (**in** *Module*) *monofun-mhomTr1*:⟦*R module N*; *h* ∈ *H*; *free-generator R M H*;
f ∈ *H* → *carrier N*; *a* ∈ *carrier R*⟧ ⟹
monofun R M N f H h (*a* ·$_s$ *h*) = *a* ·$_{sN}$ (*f h*)
apply (*frule free-generator-sub*)
apply (*simp add*:*monofun-def*)
apply (*cut-tac a* = {*h*} **and** *x* = *h* **in** *elem-fgs*, *rule subsetI*,
simp add:*subsetD*, *simp*,
cut-tac fgs-submodule[*of* {*h*}])
apply (*frule submodule-sc-closed*[*of fgs R M* {*h*} *a h*], *assumption*+,
simp)
apply (*subgoal-tac* ∃!*y*. ∃ *r*∈*carrier R*. *a* ·$_s$ *h* = *r* ·$_s$ *h* ∧ *y* = *r* ·$_{sN}$ *f h*,
subgoal-tac ∃ *r*∈*carrier R*. *a* ·$_s$ *h* = *r* ·$_s$ *h* ∧
(*THE y*. ∃ *s*∈*carrier R*. *a* ·$_s$ *h* = *s* ·$_s$ *h* ∧ *y* = *s* ·$_{sN}$ (*f h*)) =
r ·$_{sN}$ (*f h*))
prefer *2* **apply** (*rule theI'*, *simp*)
apply (*thin-tac* ∃!*y*. ∃ *r*∈*carrier R*. *a* ·$_s$ *h* = *r* ·$_s$ *h* ∧ *y* = *r* ·$_{sN}$ *f h*)
apply (*erule bexE*, *erule conjE*, *simp*,
thin-tac (*THE y*. ∃ *s*∈*carrier R*. *r* ·$_s$ *h* = *s* ·$_s$ *h* ∧ *y* = *s* ·$_{sN}$ *f h*) = *r* ·$_{sN}$ *f h*)
apply (*frule-tac r* = *r* **in** *monofun-mHomTr*[*of h H a*], *assumption*+, *simp*)

apply (*rule ex-ex1I*)
apply *blast*
apply ((*erule bexE*)+, (*erule conjE*)+)
apply (*frule-tac r* = *r* **in** *monofun-mHomTr*[*of h H a*], *assumption*+,
frule-tac r = *ra* **in** *monofun-mHomTr*[*of h H a*], *assumption*+)
apply (*thin-tac a* ·$_s$ *h* = *r* ·$_s$ *h*, *thin-tac a* ·$_s$ *h* = *ra* ·$_s$ *h*,
rotate-tac −*2*, *frule sym*, *thin-tac a* = *r*, *frule sym*,
thin-tac a = *ra*, *simp*)

apply (*rule subsetI*, *simp add*:*subsetD*)
done

lemma (**in** *Module*) *monofun-mem*:⟦*R module N*; *h* ∈ *H*; *free-generator R M H*;
x ∈ *fgs R M* {*h*}; *f* ∈ *H* → *carrier N*⟧ ⟹
monofun R M N f H h x ∈ *carrier N*
apply (*frule free-generator-sub*,
frule subsetD[*of H carrier M h*], *assumption*+,
frule fgs-single-span[*of h x*], *assumption*+)
apply (*erule bexE*, *simp add*:*monofun-mhomTr1*,
frule funcset-mem[*of f H carrier N h*], *assumption*)
apply (*simp add*:*Module.sc-mem*[*of N R - f h*])
done

lemma (**in** *Module*) *monofun-eq-f*:⟦*R module N*; *h* ∈ *H*; *free-generator R M H*;
f ∈ *H* → *carrier N*⟧ ⟹ *monofun R M N f H h h* = *f h*
apply (*cut-tac sc-Ring*)
apply (*frule-tac N* = *N* **and** *h* = *h* **and** *f* = *f* **and** *a* = $1_{r\,R}$ **in** *monofun-mhomTr1*,

assumption+)
apply (*simp add*:*Ring.ring-one*)
apply (*frule free-generator-sub*, *frule subsetD*[*of H carrier M h*],
assumption+,
frule funcset-mem[*of f H carrier N h*], *assumption*)
apply (*simp add*:*sprod-one*, *simp add*:*Module.sprod-one*)
done

lemma (**in** *Module*) *sSum-unique*:⟦*R module N*; *free-generator R M H*; *H1* ⊆ *H*;
h ∈ *H* − *H1*; *x1* ∈(*fgs R M H1*); *x2* ∈ (*fgs R M H1*);
y1 ∈ (*fgs R M* {*h*}); *y2* ∈ (*fgs R M* {*h*}); *x1* ± *y1* = *x2* ± *y2*⟧ ⟹
x1 = *x2* ∧ *y1* = *y2*
apply (*cut-tac sc-Ring*, *frule Ring.whole-ideal* [*of R*],
frule free-generator-sub)
apply (*frule-tac subset-trans*[*of H1 H carrier M*], *assumption*+)
apply (*frule subsetD*[*of H carrier M h*], *simp*)
apply (*frule-tac fgs-single-span*[*of h y1*], *assumption*+,
frule-tac fgs-single-span[*of h y2*], *assumption*+)
apply (*erule bexE*, *erule bexE*)
apply (*frule-tac* *a* = *a* **and** *m* = *h* **in** *sc-mem*, *assumption*+,
frule-tac *a* = *aa* **and** *m* = *h* **in** *sc-mem*, *assumption*+)
apply (*frule subset-trans*[*of H1 H carrier M*], *assumption*+)
apply (*frule-tac A* = *carrier R* **and** *H* = *H1* **in** *lin-span-sub-carrier*,
assumption+)
apply (*fold fgs-def*)
apply (*frule subsetD*[*of fgs R M H1 carrier M x1*], *assumption*+,
frule subsetD[*of fgs R M H1 carrier M x2*], *assumption*+)
apply *simp*
apply (*frule ag-mOp-closed*[*of x2*])
apply (*frule-tac z* = *a* ·$_s$ *h* **in** *ag-pOp-assoc*[*of* −$_a$ *x2 x1*], *assumption*+,
simp only:*ag-pOp-assoc*[*THEN sym*], *simp add*:*ag-l-inv1*,
simp add:*ag-l-zero*,
frule ag-pOp-closed[*of* −$_a$ *x2 x1*], *assumption*+,
frule-tac x = *a* ·$_s$ *h* **in** *ag-mOp-closed*)
apply (*frule-tac y* = *a* ·$_s$ *h* **and** *z* = −$_a$ (*a* ·$_s$ *h*) **in**
ag-pOp-assoc[*of* −$_a$ *x2* ± *x1*], *assumption*+,
simp add:*ag-r-inv1 ag-r-zero*, *thin-tac* −$_a$ *x2* ± *x1* ± *a* ·$_s$ *h* = *aa* ·$_s$ *h*)
apply (*frule fgs-submodule*[*of H1*])
apply (*frule submodule-mOp-closed*[*of fgs R M H1 x2*], *assumption*,
frule submodule-pOp-closed[*of fgs R M H1* −$_a$ *x2 x1*], *assumption*+)
apply (*case-tac H1* = {}, *simp add*:*empty-fgs*)
apply (*simp add*:*ag-eq-diffzero*[*THEN sym*])
apply (*frule mem-fgs-l-comb*[*of H1* −$_a$ *x2* ± *x1*], *assumption*+,
erule exE, (*erule bexE*)+)
apply (*case-tac* −$_{aM}$ *x2* ±$_M$ *x1* = **0**$_M$, *simp*, *rotate-tac* −2, *frule sym*,
thin-tac **0** = *l-comb R M n s f*, *simp*,
simp add:*ag-pOp-commute*[*of* −$_a$ *x2 x1*])
apply (*simp add*:*ag-eq-diffzero*[*of x1 x2*])
apply (*simp add*:*ag-eq-diffzero*[*THEN sym*])

apply *simp*
apply (*frule-tac* $f = f$ **and** $n = n$ **and** $s = s$ **in** *unique-expression2*[*of H1*],
assumption+, (*erule exE*)+, (*erule conjE*)+, *simp*,
simp add:*bij-to-def*, *erule conjE*,
thin-tac $f \in \{j.\ j \leq n\} \to H1$,
thin-tac $s \in \{j.\ j \leq n\} \to$ *carrier R*,
thin-tac surj-to g $\{j.\ j \leq m\}$ $(f\ `\ \{j.\ j \leq n\})$,
thin-tac l-comb R M n s f = *l-comb R M m t g*, *simp*)
apply (*frule-tac* $f = g$ **and** $A = \{j.\ j \leq m\}$ **and** $B = H1$ **and** *?B1.0* $= H$ **in**
extend-fun, *assumption*+)
apply (*frule-tac* $f = g$ **and** $n = m$ **and** $s = t$ **in** *unique-expression4*[*of H*],
simp, (*erule exE*)+, (*erule conjE*)+, *erule exE*, (*erule conjE*)+)
apply (*frule-tac* $f = g$ **and** $A = \{j.\ j \leq m\}$ **and** $B = H1$ **in** *image-sub0*,
frule-tac $A = ga\ `\ \{k.\ k \leq ma\}$ **and** $B = g\ `\ \{j.\ j \leq m\}$ **and**
$C = H1$ **in** *subset-trans*, *assumption*+, *simp*,
thin-tac $g \in \{j.\ j \leq m\} \to H1$,
thin-tac $t \in \{j.\ j \leq m\} \to$ *carrier R*,
thin-tac $ga\ `\ \{k.\ k \leq ma\} \subseteq g\ `\ \{k.\ k \leq m\}$,
thin-tac l-comb R M m t g = *l-comb R M ma ta ga*,
thin-tac $g\ `\ \{j.\ j \leq m\} \subseteq H1$,
thin-tac inj-on g $\{j.\ j \leq m\}$,
thin-tac $g \in \{j.\ j \leq m\} \to H$, *simp*)
apply (*rename-tac a aa m g t*)
apply (*simp add*:*sc-minus-am1*,
cut-tac sc-Ring, *frule Ring.ring-is-ag*,
frule-tac $x = a$ **in** *aGroup.ag-mOp-closed*[*of R*], *assumption*)
apply (*simp add*:*sc-l-distr*[*THEN sym*])
apply (*case-tac aa* $\pm_R -_{aR}\ a = \mathbf{0}_R$, *simp add*:*sc-0-m*,
frule-tac $x = aa$ **and** $y = -_{aR}\ a$ **in** *aGroup.ag-pOp-closed*, *assumption*+)
apply (*subgoal-tac* $(\lambda x{\in}\{0{::}nat\}.\ (aa \pm_R -_{aR}\ a)) \in$
$\{j.\ j \leq (0{::}nat)\} \to$ *carrier R*,
subgoal-tac $(\lambda x{\in}\{0{::}nat\}.\ h) \in$
$\{j.\ j \leq (0{::}nat)\} \to H$)
apply (*frule-tac* $f = \lambda x{\in}\{0{::}nat\}.\ h$ **and** $n = 0$ **and**
$s = \lambda x{\in}\{0{::}nat\}.\ (aa \pm_R -_{aR}\ a)$ **and** $g = g$ **and** $m = m$ **and** $t = t$ **in**
unique-expression5[*of H*])
apply (*simp*)
apply (*simp add*:*inj-on-def*)
apply (*simp*, *assumption*+)
apply (*simp add*:*l-comb-def*[*of - - 0*], *rule ballI*, *simp*)
apply *simp*
apply (*frule-tac* $A = (\lambda x{\in}\{0\}.\ h)\ `\ \{j.\ j \leq 0\}$ **and** $B = g\ `\ \{j.\ j \leq m\}$
and $C = H1$ **in** *subset-trans*, *assumption*+,
thin-tac $(\lambda x{\in}\{0\}.\ h) \in \{j.\ j \leq 0\} \to H$,
thin-tac $(\lambda x{\in}\{0\}.\ aa \pm_R -_{aR}\ a) \in \{j.\ j \leq 0\} \to$ *carrier R*,
thin-tac $(\lambda x{\in}\{0\}.\ h)\ `\ \{j.\ j \leq 0\} \subseteq g\ `\ \{j.\ j \leq m\}$,
thin-tac $g\ `\ \{k.\ k \leq m\} \subseteq H1$)
apply (*simp*, *simp*, *simp*)
done

lemma (**in** *Module*) *ex-extensionTr*:⟦*R module N*; *free-generator R M H*;
f ∈ *H* → *carrier N*; *H1* ⊆ *H*; *h* ∈ *H*; *h* ∉ *H1*;
g ∈ *mHom R* (*mdl M* (*fgs R M H1*)) *N*;
x ∈ *fgs R M H1* ∓ (*fgs R M* {*h*})⟧ ⟹
∃ *k1*∈ *fgs R M H1*. ∃ *k2*∈*fgs R M* {*h*}. *x* = *k1* ± *k2* ∧
(*THE y*. ∃ *h1*∈*fgs R M H1*. ∃ *h2*∈*fgs R M* {*h*}. *x* = *h1* ± *h2* ∧ *y* = *g h1* ±$_N$
(*monofun R M N f H h h2*)) = *g k1* ±$_N$ (*monofun R M N f H h k2*)
apply (*frule free-generator-sub*)
apply (*frule sSum-eq*[*THEN sym*, *of N H H1 h*], *assumption*+, *simp*,
simp)
apply (*subgoal-tac* ∃!*y*. ∃ *h1*∈*fgs R M H1*. ∃ *h2*∈*fgs R M* {*h*}. *x* = *h1* ± *h2* ∧
y = *g h1* ±$_N$ (*monofun R M N f H h h2*),
subgoal-tac ∃ *h1*∈*fgs R M H1*. ∃ *h2*∈*fgs R M* {*h*}. *x* = *h1* ± *h2* ∧
(*THE y*. ∃ *h1*∈*fgs R M H1*. ∃ *h2*∈*fgs R M* {*h*}. *x* = *h1* ± *h2* ∧
y = *g h1* ±$_N$ (*monofun R M N f H h h2*)) = *g h1* ±$_N$ (*monofun R M N f H h*
h2))
prefer *2* **apply** (*rule theI'*) **apply** *simp*
apply (*thin-tac* ∃!*y*. ∃ *h1*∈*fgs R M H1*.
∃ *h2*∈*fgs R M* {*h*}. *x* = *h1* ± *h2* ∧ *y* = *g h1* ±$_N$ *monofun R M N f H h h2*)
apply *blast*
apply (*rule ex-ex1I*)
apply (*thin-tac fgs R M* (*insert h H1*) = *fgs R M H1* ∓ (*fgs R M* {*h*}))
apply (*cut-tac fgs-sub-carrier*[*of H1*], *cut-tac fgs-sub-carrier*[*of* {*h*}])
apply (*simp add*:*set-sum*)
apply (*erule bexE*)+
apply (*frule subsetD*[*of H carrier M h*], *assumption*+)
apply (*frule-tac x* = *k* **in** *monofun-mem*[*of N h H - f*], *assumption*+)
apply *blast*
apply (*rule subsetI*, *simp*, *simp add*:*subsetD*,
rule subset-trans[*of H1 H carrier M*], *assumption*+)
apply ((*erule bexE*)+, (*erule conjE*)+, *simp*)
apply (*frule-tac ?x1.0* = *h1a* **and** *?x2.0* = *h1* **and** *?y1.0* = *h2a* **and** *?y2.0* =
h2
in *sSum-unique*[*of N H H1 h*], *assumption*+, *simp*, *assumption*+)
apply ((*erule conjE*)+, *simp*)
done

lemma (**in** *Module*) *monofun-add*:⟦*R module N*; *free-generator R M H*;
f ∈ *H* → *carrier N*; *h* ∈ *H*; *x* ∈ *fgs R M* {*h*}; *y* ∈ *fgs R M* {*h*}⟧ ⟹
monofun R M N f H h (*x* ± *y*) =
monofun R M N f H h x ±$_N$ (*monofun R M N f H h y*)
apply (*frule free-generator-sub*,
cut-tac sc-Ring, *frule Ring.ring-is-ag*,
frule subsetD[*of H carrier M h*], *assumption*+,
frule fgs-single-span[*of h x*], *assumption*+,
frule fgs-single-span[*of h y*], *assumption*+, (*erule bexE*)+, *simp*,
simp add:*sc-l-distr*[*THEN sym*])
apply (*frule-tac x* = *a* **and** *y* = *aa* **in** *aGroup.ag-pOp-closed*[*of R*], *assumption*+)

apply (*simp add*:*monofun-mhomTr1*,
frule funcset-mem[*of f H carrier N h*], *assumption*+)
apply (*simp add*:*Module.sc-l-distr*)
done

lemma (**in** *Module*) *monofun-sprod*:⟦*R module N*; *free-generator R M H*;
$f \in H \to$ *carrier N*; $h \in H$; $x \in$ *fgs R M* $\{h\}$; $a \in$ *carrier R*⟧ $\Longrightarrow$
monofun R M N f H h ($a \cdot_s x$) = $a \cdot_{sN}$ (*monofun R M N f H h x*)
apply (*frule free-generator-sub*,
cut-tac sc-Ring,
frule subsetD[*of H carrier M h*], *assumption*+,
frule fgs-single-span[*of h x*], *assumption*+,
erule bexE, *simp add*:*sc-assoc*[*THEN sym*])
apply (*frule-tac* $x = a$ **and** $y = aa$ **in** *Ring.ring-tOp-closed*, *assumption*+)
apply (*simp add*:*monofun-mhomTr1*,
frule funcset-mem[*of f H carrier N h*], *assumption*+)
apply (*simp add*:*Module.sc-assoc*)
done

lemma (**in** *Module*) *monofun-0*:⟦*R module N*; *free-generator R M H*;
$f \in H \to$ *carrier N*; $h \in H$⟧ $\Longrightarrow$ *monofun R M N f H h* $\mathbf{0} = \mathbf{0}_N$
apply (*cut-tac fgs-submodule*[*of* $\{h\}$],
frule submodule-inc-0[*of fgs R M* $\{h\}$])
apply (*cut-tac sc-Ring*, *frule Ring.ring-zero*,
frule monofun-sprod[*of N H f h* $\mathbf{0}$ $\mathbf{0}_R$], *assumption*+,
cut-tac ag-inc-zero)
apply (*simp add*:*sc-0-m*,
frule monofun-mem[*of N h H* $\mathbf{0}$ *f*], *assumption*+,
simp add:*Module.sc-0-m*)
apply (*frule free-generator-sub*, *rule subsetI*, *simp add*:*subsetD*)
done

lemma (**in** *Module*) *ex-extension*:⟦*R module N*; *free-generator R M H*;
$f \in H \to$ *carrier N*; *H1* $\subseteq H$; $h \in H - H1$; (*H1*, *g*) $\in$ *fsps R M N f H*⟧ $\Longrightarrow$
$\exists k.$ ((*H1* $\cup$ $\{h\}$), *k*) $\in$ *fsps R M N f H*
apply (*frule free-generator-sub*,
cut-tac sc-Ring,
frule subset-trans[*of H1 H carrier M*], *assumption*+,
frule subsetD[*of H carrier M h*], *simp*+,
frule singleton-sub[*of h carrier M*])
apply (*frule fgs-submodule*[*of* $\{h\}$],
frule fgs-submodule[*of H1*],
frule submodule-subset[*of fgs R M* $\{h\}$],
frule submodule-subset[*of fgs R M H1*],
frule insert-sub[*of H1 carrier M h*], *assumption*,
frule fgs-submodule[*of insert h H1*])
apply (*simp add*:*fsps-def fsp-def*)
apply (*erule conjE*)+
apply (*subgoal-tac* ($\lambda x \in$ (*fgs R M* (*H1* $\cup$ $\{h\}$))).

(*THE y.* (∃ *h1*∈(*fgs R M H1*). ∃ *h2*∈(*fgs R M* {*h*}). *x* = *h1* ± *h2* ∧
y = *g h1* ±$_N$ (*monofun R M N f H h h2*)))) ∈
mHom R (*mdl M* (*fgs R M* (*insert h H1*))) *N* ∧
f h = (λ*x* ∈ (*fgs R M* (*H1* ∪ {*h*})).
(*THE y.* (∃ *h1*∈(*fgs R M H1*). ∃ *h2*∈(*fgs R M* {*h*}). *x* = *h1* ± *h2* ∧
y = *g h1* ±$_N$ (*monofun R M N f H h h2*)))) *h* ∧
(∀ *z*∈*H1*. *g z* = (λ*x* ∈ (*fgs R M* (*H1* ∪ {*h*})).
(*THE y.* (∃ *h1*∈(*fgs R M H1*). ∃ *h2*∈(*fgs R M* {*h*}). *x* = *h1* ± *h2* ∧
y = *g h1* ±$_N$ (*monofun R M N f H h h2*)))) *z*))
apply *blast*
apply (*rule conjI*)
apply (*rule Module.mHom-test*)
apply (*rule mdl-is-module*, *assumption*+)
apply (*rule conjI*,
rule Pi-I, *simp add*:*mdl-carrier*)
apply (*frule sSum-eq*[*THEN sym*, *of N H H1 h*], *assumption*+, *simp*, *simp*)
apply (*frule-tac h* = *h* **and** *H* = *H* **and** *?H1.0* = *H1* **and** *g* = *g* **and** *x* = *x* **and** *N* = *N*
in *ex-extensionTr*, *assumption*+,
(*erule bexE*)+, *erule conjE*, *rotate-tac* −*1*,
frule sym,
thin-tac (*THE y.* ∃ *h1*∈*fgs R M H1*. ∃ *h2*∈*fgs R M* {*h*}.
x = *h1* ± *h2* ∧ *y* = *g h1* ±$_N$ *monofun R M N f H h h2*) =
g k1 ±$_N$ *monofun R M N f H h k2*)
apply (*frule mdl-is-module*[*of fgs R M H1*],
frule-tac f = *g* **and** *m* = *k1* **in** *Module.mHom-mem*[*of mdl M* (*fgs R M H1*)
R N], *assumption*+, *simp add*:*mdl-carrier*)
apply (*frule-tac x* = *k2* **in** *monofun-mem*[*of N h H - f*], *assumption*+)
apply (*frule Module.module-is-ag*[*of N*],
frule-tac x = *g k1* **and** *y* = *monofun R M N f H h k2* **in**
aGroup.ag-pOp-closed[*of N*], *assumption*+, *simp*)
apply (*simp add*:*mdl-carrier*)
apply (*rule conjI*, (*rule ballI*)+)
apply (*frule mdl-is-module*[*of fgs R M* (*insert h H1*)],
frule Module.module-is-ag[*of mdl M* (*fgs R M* (*insert h H1*))],
frule-tac x = *m* **and** *y* = *n* **in** *aGroup.ag-pOp-closed*[*of mdl M* (*fgs R M* (*insert h H1*))],
(*simp add*:*mdl-carrier*)+)
apply (*frule sSum-eq*[*THEN sym*, *of N H H1 h*], *assumption*+, *simp*, *simp*)
apply (*frule-tac h* = *h* **and** *H* = *H* **and** *?H1.0* = *H1* **and** *g* = *g* **and** *N* = *N* **and**
x = *m* ±$_{mdl\ M\ (fgs\ R\ M\ H1\ \mp\ fgs\ R\ M\ \{h\})}$ *n* **in** *ex-extensionTr*, *assumption*+,
(*erule bexE*)+, (*erule conjE*)+, *simp*,
thin-tac (*THE y.* ∃ *h1*∈*fgs R M H1*. ∃ *h2*∈*fgs R M* {*h*}.
k1 ± *k2* = *h1* ± *h2* ∧ *y* = *g h1* ±$_N$ *monofun R M N f H h h2*) =
g k1 ±$_N$ *monofun R M N f H h k2*)
apply (*frule-tac h* = *h* **and** *H* = *H* **and** *?H1.0* = *H1* **and** *g* = *g* **and** *N* = *N*

and
$x = m$ **in** *ex-extensionTr*, *assumption*+, (*erule bexE*)+, (*erule conjE*)+,
simp,
thin-tac (*THE y*. ∃*h1*∈*fgs R M H1*. ∃*h2*∈*fgs R M* {*h*}.
k1a ± *k2a* = *h1* ± *h2* ∧ *y* = *g h1* ±$_N$ *monofun R M N f H h h2*) =
g k1a ±$_N$ *monofun R M N f H h k2a*)
apply (*frule-tac h* = *h* **and** *H* = *H* **and** *?H1.0* = *H1* **and** *g* = *g* **and** *N* = *N*
and
$x = n$ **in** *ex-extensionTr*, *assumption*+, (*erule bexE*)+, (*erule conjE*)+,
simp,
thin-tac (*THE y*. ∃*h1*∈*fgs R M H1*. ∃*h2*∈*fgs R M* {*h*}.
k1b ± *k2b* = *h1* ± *h2* ∧ *y* = *g h1* ±$_N$ *monofun R M N f H h h2*) =
g k1b ±$_N$ *monofun R M N f H h k2b*)
apply (*simp add*:*mdl-def*[*of M fgs R M H1* ∓ *fgs R M* {*h*}],
fold mdl-def)
apply (*frule-tac c* = *k1a* **in** *subsetD*[*of fgs R M H1 carrier M*],
assumption+,
frule-tac c = *k1b* **in** *subsetD*[*of fgs R M H1 carrier M*],
assumption+) **apply** (
frule-tac c = *k2a* **in** *subsetD*[*of fgs R M* {*h*} *carrier M*],
assumption+,
frule-tac c = *k2b* **in** *subsetD*[*of fgs R M* {*h*} *carrier M*],
assumption+)
apply (*simp add*:*pOp-assocTr41*[*THEN sym*], *simp add*:*pOp-assocTr42*,
frule-tac x = *k2a* **and** *y* = *k1b* **in** *ag-pOp-commute*, *assumption*+, *simp*,
thin-tac k2a ± *k1b* = *k1b* ± *k2a*, *simp add*:*pOp-assocTr42*[*THEN sym*],
simp add:*pOp-assocTr41*)
apply(*frule-tac h* = *k1a* **and** *k* = *k1b* **in** *submodule-pOp-closed*[*of fgs R M H1*],
assumption+)
apply(*frule-tac h* = *k2a* **and** *k* = *k2b* **in**
submodule-pOp-closed[*of fgs R M* {*h*}], *assumption*+)
apply (*frule-tac ?x1.0* = *k1a* ± *k1b* **and** *?x2.0* = *k1* **and** *?y1.0* = *k2a* ± *k2b*
and *?y2.0* = *k2* **in** *sSum-unique*[*of N H H1 h*], *assumption*+, *simp*,
assumption+, *erule conjE*, *rotate-tac* −*2*, *frule sym*,
thin-tac k1a ± *k1b* = *k1*, *frule sym*, *thin-tac k2a* ± *k2b* = *k2*,
thin-tac k1a ± *k1b* ± (*k2a* ± *k2b*) = *k1* ± *k2*, *simp*,
frule mdl-is-module[*of fgs R M H1*])
apply (*frule-tac m* = *k1a* **and** *n* = *k1b* **in** *Module.mHom-add*[*of*
mdl M (*fgs R M H1*) *R N g*], *assumption*+,
simp add:*mdl-carrier*, *simp add*:*mdl-carrier*,
simp add:*mdl-def*, *fold mdl-def*)
apply (*simp add*: *monofun-add*)
apply (*frule-tac m* = *k1a* **in** *Module.mHom-mem*[*of mdl M* (*fgs R M H1*) *R N*
g],
assumption+, *simp add*:*mdl-carrier*,
frule-tac m = *k1b* **in** *Module.mHom-mem*[*of mdl M* (*fgs R M H1*) *R N g*],
assumption+, *simp add*:*mdl-carrier*,
frule-tac x = *k2a* **in** *monofun-mem*[*of N h H - f*], *assumption*+,
frule-tac x = *k2b* **in** *monofun-mem*[*of N h H - f*], *assumption*+)

apply (*frule Module.module-is-ag*[*of N*],
simp add:*aGroup.pOp-assocTr41*[*of N*, *THEN sym*],
simp add:*aGroup.pOp-assocTr42*[*of N*],
simp add:*aGroup.ag-pOp-commute*[*of N*])
apply (*rule ballI*)+
apply (*frule mdl-is-module*[*of fgs R M* (*insert h H1*)])
apply (*frule-tac* $a = a$ **and** $m = m$ **in** *Module.sc-mem*[*of*
mdl M (*fgs R M* (*insert h H1*)) *R*], *assumption*+,
simp add:*mdl-carrier*, *simp add*:*mdl-carrier*)
apply (*frule sSum-eq*[*THEN sym*, *of N H H1 h*], *assumption*+, *simp*)
apply (*frule-tac* $h = h$ **and** $H = H$ **and** *?H1.0* = *H1* **and** $g = g$ **and** $N = N$
and
$x = a \cdot_{s mdl\ M\ (fgs\ R\ M\ (insert\ h\ H1))}\ m$ **in** *ex-extensionTr*, *assumption*+,
simp)
apply ((*erule bexE*)+, *erule conjE*, *simp*,
thin-tac (*THE y*. ∃ *h1*∈*fgs R M H1*. ∃ *h2*∈*fgs R M* {*h*}.
k1 ± *k2* = *h1* ± *h2* ∧ *y* = *g h1* $\pm_N$ *monofun R M N f H h h2*) =
g k1 $\pm_N$ *monofun R M N f H h k2*)
apply (*simp add*:*mdl-def*, *fold mdl-def*)
apply (*frule-tac* $h = h$ **and** $H = H$ **and** *?H1.0* = *H1* **and** $g = g$ **and** $N = N$
and
$x = m$ **in** *ex-extensionTr*, *assumption*+,
(*erule bexE*)+, *erule conjE*, *simp*,
thin-tac (*THE y*. ∃ *h1*∈*fgs R M H1*. ∃ *h2*∈*fgs R M* {*h*}.
k1a ± *k2a* = *h1* ± *h2* ∧ *y* = *g h1* $\pm_N$ *monofun R M N f H h h2*) =
g k1a $\pm_N$ *monofun R M N f H h k2a*,
frule-tac $c = k1a$ **in** *subsetD*[*of fgs R M H1 carrier M*], *assumption*+,
frule-tac $c = k1$ **in** *subsetD*[*of fgs R M H1 carrier M*], *assumption*+,
frule-tac $c = k2a$ **in** *subsetD*[*of fgs R M* {*h*} *carrier M*], *assumption*+,
frule-tac $c = k2$ **in** *subsetD*[*of fgs R M* {*h*} *carrier M*], *assumption*+)
apply (*simp add*:*sc-r-distr*,
frule-tac $a = a$ **and** $h = k1a$ **in**
submodule-sc-closed[*of fgs R M H1*], *assumption*+,
frule-tac $a = a$ **and** $h = k2a$ **in**
submodule-sc-closed[*of fgs R M* {*h*}], *assumption*+)
apply (*frule-tac* *?x1.0* = *k1* **and** *?x2.0* = $a \cdot_s$ *k1a* **and** *?y1.0* = *k2* **and**
?y2.0 = $a \cdot_s$ *k2a* **in** *sSum-unique*[*of N H H1 h*], *assumption*+,
simp, *assumption*+, *rule sym*, *assumption*,
thin-tac $a \cdot_s$ *k1a* ± $a \cdot_s$ *k2a* = *k1* ± *k2*, *simp*)
apply (*frule subset-trans*[*of H1 H carrier M*], *assumption*+,
frule mdl-is-module[*of fgs R M H1*])
apply (*frule-tac* $a = a$ **and** $m = k1a$ **in** *Module.mHom-lin*[*of mdl M* (*fgs R M H1*)
R N - *g*], *assumption*+, *simp add*:*mdl-carrier*, *assumption*+)
apply (*simp add*:*mdl-def*, *fold mdl-def*)
apply (*simp add*:*monofun-sprod*)
apply (*frule-tac* $m = k1a$ **in** *Module.mHom-mem*[*of mdl M* (*fgs R M H1*) *R N*
g],
assumption+, *simp add*:*mdl-carrier*,

frule-tac $x = k2a$ **in** *monofun-mem*[*of* N h H - f], *assumption*+,
simp add:*Module.sc-r-distr*[*of* N])
apply (*rule conjI*) **apply** *simp*
apply (*cut-tac insert-sub*[*of H1 carrier M h*],
frule elem-fgs[*of insert h H1 h*], *simp*, *simp*)
apply (*frule-tac* $h = h$ **and** $H = H$ **and** *?H1.0* $= H1$ **and** $g = g$ **and** $N = N$ **and**
$x = h$ **in** *ex-extensionTr*, *assumption*+)
apply (*frule sSum-eq*[*THEN sym*, *of N H H1 h*], *assumption*+, *simp*, *simp*)
apply ((*erule bexE*)+, (*erule conjE*), *simp*,
thin-tac (*THE y*.
$\exists\, h1 \in fgs\ R\ M\ H1$.
$\exists\, h2 \in fgs\ R\ M\ \{k1 \pm k2\}$.
$k1 \pm k2 = h1 \pm h2 \wedge$
$y = g\ h1 \pm_N monofun\ R\ M\ N\ f\ H\ (k1 \pm k2)\ h2) =$
$g\ k1 \pm_N monofun\ R\ M\ N\ f\ H\ (k1 \pm k2)\ k2$)
apply (*rotate-tac* -1, *frule sym*, *thin-tac* $h = k1 \pm k2$, *simp*)
apply (*cut-tac elem-fgs*[*of* $\{h\}$ h],
frule ag-l-zero[*THEN sym*, *of h*])
apply (*frule submodule-inc-0*[*of fgs R M H1*])
apply (*frule-tac ?x1.0* $= k1$ **and** *?x2.0* $= \mathbf{0}$ **and** *?y1.0* $= k2$ **and** *?y2.0* $= h$ **in**
sSum-unique[*of N H H1 h*], *assumption*+, *simp*, *assumption*+,
simp, *simp*, *simp add*: *monofun-eq-f*)
apply (*cut-tac Module.mHom-0*[*of mdl M* (*fgs R M H1*) *R N g*],
simp add:*mdl-def*, *fold mdl-def*)
apply (*frule funcset-mem*[*of f H carrier N h*], *assumption*+,
frule Module.module-is-ag[*of N*], *simp add*:*aGroup.l-zero*,
rule mdl-is-module, *assumption*+,
rule subsetI, *simp*, *simp*, *assumption*+)
apply (*rule ballI*, *simp*,
cut-tac $x = z$ **in** *elem-fgs*[*of insert h H1*],
rule subsetI, *simp*, *erule disjE*, *simp*, *simp add*:*subsetD*,
rule-tac $c = z$ **in** *subsetD*[*of H1 insert h H1*],
rule subsetI, *simp*, *assumption*, *simp*)

apply (*frule-tac* $h = h$ **and** $H = H$ **and** *?H1.0* $= H1$ **and** $g = g$ **and** $N = N$ **and**
$x = z$ **in** *ex-extensionTr*, *assumption*+)
apply (*frule sSum-eq*[*THEN sym*, *of N H H1 h*], *assumption*+, *simp*, *simp*)
apply ((*erule bexE*)+, (*erule conjE*), *simp*)
apply (*thin-tac* (*THE y*. $\exists\, h1 \in fgs\ R\ M\ H1$. $\exists\, h2 \in fgs\ R\ M\ \{h\}$.
$k1 \pm k2 = h1 \pm h2 \wedge y = g\ h1 \pm_N monofun\ R\ M\ N\ f\ H\ h\ h2) =$
$g\ k1 \pm_N monofun\ R\ M\ N\ f\ H\ h\ k2$)
apply (*rotate-tac* -1, *frule sym*, *thin-tac* $z = k1 \pm k2$, *simp*)
apply (*frule-tac* $x = z$ **in** *elem-fgs*[*of H1*], *assumption*,
frule-tac $c = z$ **in** *subsetD*[*of H1 carrier M*], *assumption*+,
frule-tac $t = z$ **in** *ag-r-zero*[*THEN sym*])
apply (*frule-tac ?x1.0* $= k1$ **and** *?x2.0* $= z$ **and** *?y1.0* $= k2$ **and** *?y2.0* $= \mathbf{0}$ **in**

sSum-unique[*of N H H1 h*], *assumption*+, *simp*, *assumption*+,
simp add:*submodule-inc-0*, *simp*, *simp*)
apply (*simp add*: *monofun-0*)
apply (*frule mdl-is-module*[*of fgs R M H1*],
frule-tac m = *z* **in** *Module.mHom-mem*[*of mdl M* (*fgs R M H1*) *R N g*],
assumption+, *simp add*:*mdl-carrier*)
apply (*frule Module.module-is-ag*[*of N*],
simp add:*aGroup.ag-r-zero*)
done

lemma (**in** *Module*) *mHom-mHom*:⟦*R module N*; *g* ∈ *mHom R* (*mdl M* (*carrier M*)) *N*⟧
⟹ *g* ∈ *mHom R M N*
apply (*rule mHom-test*, *assumption*)
apply (*rule conjI*)
apply (*simp add*:*mHom-def aHom-def mdl-def*)
apply (*simp add*:*mHom-def aHom-def mdl-def*)
done

lemma (**in** *Module*) *exist-extension-mhom*:⟦*R module N*; *free-generator R M H*;
f ∈ *H* → *carrier N*⟧ ⟹ ∃ *g*∈*mHom R M N*. ∀ *x*∈*H*. *g x* = *f x*
apply (*frule Order-od-fm-fun*[*of N H f*], *assumption*+)
apply (*frule-tac N* = *N* **and** *H* = *H* **and** *f* = *f* **in** *od-fm-fun-inductive*,
assumption+)
apply (*frule-tac D* = *od-fm-fun R M N f H* **in** *Order.g-Zorn-lemma3*,
assumption+)
apply (*erule bexE*)
apply (*rule contrapos-pp*, *simp*+)
apply (*simp add*:*maximal-element-def*)
apply (*simp add*:*od-fm-fun-carrier*)
apply (*cut-tac p* = *m* **in** *PairE-lemma*, (*erule exE*)+, *simp*)
apply (*frule-tac a* = *x* **and** *b* = *y* **in** *mem-fsps-fst-sub*[*of N H f*], *assumption*+)
apply (*case-tac x* ≠ *H*, *frule not-sym*)
apply (*frule-tac A* = *H* **and** *B* = *x* **in** *diff-nonempty*, *assumption*,
frule-tac A = *H* − *x* **in** *nonempty-ex*, *erule exE*,
rename-tac m H1 g h)
apply (*frule-tac ?H1.0* = *H1* **and** *h* = *h* **and** *g* = *g* **in** *ex-extension*[*of N H f*],
assumption+, *erule exE*)
apply (*drule-tac x* = (*H1* ∪ {*h*}, *k*) **in** *bspec*, *assumption*)
apply (*simp add*:*od-fm-fun-def ole-def*, *fold od-fm-fun-def*,
simp add:*insert-inc1*, (*erule conjE*)+,
cut-tac a = *h* **and** *A* = *H1* **in** *insert-inc2*,
rotate-tac −*3*, *frule sym*, *thin-tac H1* = *insert h H1*, *simp*)
apply *simp*
apply (*frule-tac a* = *H* **and** *b* = *y* **in** *mem-fsps-snd-mHom*[*of N H f*], *assumption*+)
apply (*simp add*:*fgs-def free-generator-def generator-def*)

apply (*frule-tac* $g = y$ **in** *mHom-mHom*[*of* N], *assumption*+,
(*erule conjE*)+, *thin-tac* $\forall n\ s.\ s \in \{j.\ j \leq n\} \to carrier\ R \wedge$
$(\exists f.\ f \in \{j.\ j \leq n\} \to H \wedge$
$inj\text{-}on\ f\ \{j.\ j \leq n\} \wedge l\text{-}comb\ R\ M\ n\ s\ f = \mathbf{0}) \longrightarrow$
$s \in \{j.\ j \leq n\} \to \{\mathbf{0}_R\}$)
apply (*drule-tac* $x = y$ **in** *bspec*, *assumption*+,
simp add:*fsps-def fsp-def*)
done

5.6 Nakayama lemma

definition
$Lcg :: [('r, 'm)\ Ring\text{-}scheme, ('a, 'r, 'm1)\ Module\text{-}scheme, nat] \Rightarrow bool$ **where**
$Lcg\ R\ M\ j \longleftrightarrow (\exists H.\ finite\text{-}generator\ R\ M\ H \wedge j = card\ H)$

lemma (**in** *Module*) *NAKTr1*:$M\ fgover\ R \Longrightarrow$
$\exists H.\ finite\text{-}generator\ R\ M\ H \wedge (LEAST\ j.$
$\exists L.\ finite\text{-}generator\ R\ M\ L \wedge j = card\ L) = card\ H$
apply (*simp add*:*fGOver-def*)
apply (*erule exE*)
apply (*rule LeastI*)
apply (*rule-tac* $x = H$ **in** *exI*) **apply** *simp*
done

lemma (**in** *Module*) *NAKTr2*:$[\![Lcg\ R\ M\ j;\ k < (LEAST\ j.\ Lcg\ R\ M\ j)]\!] \Longrightarrow$
$\neg\ Lcg\ R\ M\ k$
apply (*rule not-less-Least*, *assumption*+)
done

lemma (**in** *Module*) *NAKTr3*:$[\![M\ fgover\ R;\ H \subseteq carrier\ M;\ finite\ H;$
$card\ H < (LEAST\ j.\ \exists L.\ finite\text{-}generator\ R\ M\ L \wedge j = card\ L)]\!] \Longrightarrow$
$\neg\ finite\text{-}generator\ R\ M\ H$
apply (*frule NAKTr1*, *erule exE*, *erule conjE*)
apply (*cut-tac* $j = card\ Ha$ **and** $k = card\ H$ **in** *NAKTr2*)
apply (*simp add*:*Lcg-def*, *blast*)
apply (*simp add*:*Lcg-def*)
apply (*simp add*:*Lcg-def*)
apply (*rule contrapos-pp*, *simp*+)
apply (*drule-tac* $a = H$ **in** *forall-spec*, *assumption*)
apply *simp*
done

lemma (**in** *Module*) *finite-gen-over-ideal*:$[\![ideal\ R\ A;\ h \in \{j.\ j \leq (n::nat)\} \to$
$carrier\ M;\ generator\ R\ M\ (h\ `\ \{j.\ j \leq n\});\ A \odot_R M = carrier\ M;$
$m \in carrier\ M\ ]\!] \Longrightarrow\ \exists s{\in}\{j.\ j \leq n\} \to A.\ m = l\text{-}comb\ R\ M\ n\ s\ h$
apply (*frule l-span-closed3*[*of A h* ‘ $\{j.\ j \leq n\}$], *assumption*+)
apply (*rotate-tac* -1, *frule sym*,
thin-tac linear-span R M A $(h\ `\ \{j.\ j \leq n\}) = carrier\ M$)

apply (*frule eq-set-inc*[*of m carrier M*
linear-span *R M A* (*h* ‘ {*j*. *j* ≤ *n*})], *assumption*+)
apply (*thin-tac carrier M* = *linear-span R M A* (*h* ‘ {*j*. *j* ≤ *n*}))
apply (*simp add*:*linear-span-def*)
apply (*cut-tac Nset-nonempty*[*of n*], *simp*)
apply (*case-tac* ∀ *x*. ¬ *x* ≤ *n*, *simp*, *simp*)
apply (*erule exE*, (*erule bexE*)+)
apply (*simp add*:*finite-lin-span*)
done

lemma (**in** *Module*) *NAKTr4*:⟦*ideal R A*; *h* ∈ {*j*. *j* ≤ (*k*::*nat*)} → *carrier M*;
0 < *k*; *h* ‘ {*j*. *j* ≤ *k*} ⊆ *carrier M*; *s* ∈ {*j*. *j* ≤ *k*} → *A*;
h k = Σ*e M* (λ*j*. *s j* ·*s* (*h j*)) (*k* − *Suc 0*) ± (*s k* ·*s* (*h k*))⟧ ⟹
(*1r R* ±*R* (−*aR* (*s k*))) ·*s* (*h k*) = Σ*e M* (λ*j*. *s j* ·*s* (*h j*)) (*k* − *Suc 0*)
apply (*cut-tac sc-Ring*)
apply (*frule funcset-mem*[*of h* {*j*. *j* ≤ *k*} *carrier M k*], *simp*)
apply (*frule-tac funcset-mem*[*of s* {*j*. *j* ≤ *k*} *A k*], *simp*,
frule-tac h = *s k* **in** *Ring.ideal-subset*[*of R A*], *assumption*+)
apply (*cut-tac n* = *k* − *Suc 0* **and** *f* = λ*j*. *s j* ·*s h j* **in** *nsum-mem*)
apply (*rule allI*, *rule impI*, *rule sc-mem*,
frule-tac x = *j* **in** *funcset-mem*[*of s* {*j*. *j* ≤ *k*} *A*],
simp)
apply (*simp add*:*Ring.ideal-subset*)
apply (*cut-tac i* = *j* **and** *j* = *k* − *Suc 0* **and** *k* = *k* **in**
le-trans, *assumption*+, *simp*,
rule-tac x = *j* **in** *funcset-mem*[*of h* {*j*. *j* ≤ *k*} *carrier M*],
assumption, *simp*)

apply (*frule sc-mem*[*of s k h k*], *assumption*+,
frule ag-mOp-closed[*of s k* ·*s h k*])
apply (*frule-tac ag-pOp-add-r*[*of h k*
Σ*e M* (λ*j*. *s j* ·*s h j*) (*k* − *Suc 0*) ± *s k* ·*s h k* −*a* (*s k* ·*s h k*)],
rule ag-pOp-closed, *assumption*+,
thin-tac h k = Σ*e M* (λ*j*. *s j* ·*s h j*) (*k* − *Suc 0*) ± *s k* ·*s h k*,
simp add:*ag-pOp-assoc ag-r-inv1 ag-r-zero*,
simp add:*sc-minus-am1*)
apply (*frule Ring.ring-is-ag*,
frule aGroup.ag-mOp-closed[*of R s k*], *assumption*+)

apply (*subgoal-tac* (*1r R* ±*R* (−*aR* (*s k*))) ·*s h k* = *h k* ± (−*aR* (*s k*)) ·*s h k*,
simp,
thin-tac h k ± (−*aR* (*s k*)) ·*s h k* = Σ*e M* (λ*j*. *s j* ·*s h j*) (*k* − *Suc 0*))
apply (*cut-tac Ring.ring-one*[*of R*],
simp add:*sc-l-distr sprod-one*, *assumption*)
done

lemma (**in** *Module*) *NAKTr5*:⟦¬ *zeroring R*; *ideal R A*; *A* ⊆ *J-rad R*;
A ⊙*R M* = *carrier M*; *finite-generator R M H*; *card H* = *Suc k*; *0* < *k*⟧ ⟹
∃ *h* ∈ {*j*. *j* ≤ *k*} → *carrier M*. *H* = *h* ‘ {*j*. *j* ≤ *k*} ∧

$h\ k \in$ *linear-span R M A* $(h$ ‘ $\{j.\ j \leq (k - Suc\ 0)\})$
apply (*cut-tac sc-Ring*)
apply (*insert Nset2-finiteTr* [*of k*])
apply (*subgoal-tac* $\exists h.\ h \in \{j.\ j \leq k\} \to H \wedge$ *surj-to h* $\{j.\ j \leq k\}$ *H*)
prefer *2*
apply (*subgoal-tac finite H*)
apply *blast* **apply** (*simp add*:*finite-generator-def*)
apply (*thin-tac* $\forall A.$ *finite A* $\wedge$ *card A* $=$ *Suc k* $\longrightarrow$
$(\exists f.\ f \in \{j.\ j \leq k\} \to A \wedge$ *surj-to f* $\{j.\ j \leq k\}$ *A*))
apply (*erule exE*)
apply (*simp add*:*surj-to-def*, *erule conjE*) **apply** (*rotate-tac* -1)
apply (*drule sym*)
apply (*frule-tac f* $=$ *h* **and** *A* $= \{j.\ j \leq k\}$ **and** *B* $=$ *H* **and** *x* $=$ *k* **in**
funcset-mem)
apply *simp*
apply (*simp add*:*finite-generator-def*,
simp add:*generator-def*, *erule conjE*)
apply (*cut-tac A* $=$ *h* ‘ $\{j.\ j \leq k\}$ **and** *B* $=$ *carrier M* **and** *c* $=$ *h k* **in**
subsetD, *assumption*, *simp*)
apply (*frule-tac h* $=$ *h* **and** *n* $=$ *k* **and** *m* $=$ *h k* **in** *finite-gen-over-ideal* [*of*
A])
apply (*fastforce simp add*:*Pi-def image-def*)
apply (*simp add*:*generator-def*)
apply *assumption*+
apply (*erule bexE*)
apply (*frule-tac f* $=$ *s* **and** *A* $= \{j.\ j \leq k\}$ **and** *B* $=$ *A* **and** *x* $=$ *k* **in**
funcset-mem, *simp*, *simp add*:*subsetD*)
apply (*subgoal-tac l-comb R M k s h* $=$ *l-comb R M* $(Suc\ (k - Suc\ 0))\ s\ h$,
simp del:*Suc-pred*,
thin-tac l-comb R M k s h $=$ *l-comb R M* $(Suc\ (k - Suc\ 0))\ s\ h$)
apply (*cut-tac s* $=$ *s* **and** *n* $=$ *k* $-$ *Suc 0* **and** *f* $=$ *h* **and** *H* $=$ *h* ‘ $\{j.\ j \leq k\}$
in *l-comb-Suc*[*of* - *A*], *assumption*+,
frule Ring.ideal-subset1[*of R A*], *assumption*,
simp add:*extend-fun*, *simp*)
apply (*rotate-tac* -3, *frule sym*,
thin-tac h k $=$ *l-comb R M* $(Suc\ (k - Suc\ 0))\ s\ h$, *simp*)
apply (*frule-tac h* $=$ *h* **and** *k* $=$ *k* **and** *s* $=$ *s* **in** *NAKTr4*[*of A*])
apply (*fastforce simp add*:*Pi-def image-def*)
apply(*assumption*+, *rule sym*, *simp add*:*l-comb-def*)
apply (*cut-tac H* $=$ *h* ‘ $\{j.\ j \leq (k - Suc\ 0)\}$ **and** *s* $=$ *s* **and** *n* $=$ *k* $-$ *Suc 0*
and *f* $=$ *h* **in** *l-comb-mem-linear-span*[*of A*], *assumption*+)
apply (*rule-tac A* $=$ *h* ‘ $\{j.\ j \leq k - Suc\ 0\}$ **and**
B $=$ *h* ‘ $\{j.\ j \leq k\}$ **and** *C* $=$ *carrier M* **in** *subset-trans*)
apply (*rule subsetI*, *simp add*:*image-def*, *erule exE*, *erule conjE*,
cut-tac i $=$ *xa* **and** *j* $=$ *k* $-$ *Suc 0* **and** *k* $=$ *k* **in** *le-trans*,
assumption, *subst Suc-le-mono*[*THEN sym*], *simp*, *blast*, *assumption*)
apply (*rule Pi-I*, *simp*)
apply (*frule-tac x* $=$ *x* **in** *le-pre-le*[*of* - *k*], *simp add*:*Pi-def*)
apply (*rule Pi-I*, *simp*)

apply (*unfold l-comb-def*)
apply (*rotate-tac −2*, *frule sym*,
thin-tac ($1_{r\,R} \pm_R -_{a\,R} (s\ k)) \cdot_s h\ k = \Sigma_e\ M\ (\lambda j.\ s\ j \cdot_s h\ j)\ (k - Suc\ 0)$,
thin-tac $\Sigma_e\ M\ (\lambda j.\ s\ j \cdot_s h\ j)\ (k - Suc\ 0) \pm s\ k \cdot_s h\ k = h\ k$,
simp,
thin-tac $\Sigma_e\ M\ (\lambda j.\ s\ j \cdot_s h\ j)\ (k - Suc\ 0) = (1_{r\,R} \pm_R -_{a\,R} (s\ k)) \cdot_s h\ k$)
apply (*frule-tac* $x = s\ k$ **in** *Ring.J-rad-unit* [*of R*], *assumption*,
frule-tac $f = s$ **and** $A = \{j.\ j \leq k\}$ **and** $B = A$ **and** $x = k$ **in**
funcset-mem, *simp*, *simp add*:*subsetD*)
apply (*frule Ring.ring-one*[*of R*],
frule-tac $a = 1_{r\,R}$ **in** *forall-spec*, *assumption*)
apply (*frule-tac* $h = s\ k$ **in** *Ring.ideal-subset*[*of R A*], *assumption+*,
frule Ring.ring-is-ag[*of R*],
frule-tac $x = s\ k$ **in** *aGroup.ag-mOp-closed*[*of R*], *assumption*,
thin-tac $\forall y.\ y \in carrier\ R \longrightarrow Unit\ R\ (1_{r\,R} \pm_R (-_{a\,R} (s\ k)) \cdot_{r\,R} y)$,
simp add:*Ring.ring-r-one*)
apply (*simp add*:*Unit-def*, *erule conjE*, *erule bexE*)
apply (*simp add*:*Ring.ring-tOp-commute*)
apply (*frule-tac* $H = h$ ‘ $\{j.\ j \leq k - Suc\ 0\}$ **and** $r = b$ **and**
$x = (1_{r\,R} \pm_R (-_{a\,R} (s\ k))) \cdot_s h\ k$ **in** *linear-span-sc-closed*[*of A*])
apply (*rule-tac* $A = h$ ‘ $\{j.\ j \leq k - Suc\ 0\}$ **and** $B = h$ ‘ $\{j.\ j \leq k\}$
and $C = carrier\ M$ **in** *subset-trans*,
rule subsetI, *simp add*:*image-def*, *erule exE*, *erule conjE*)
apply (*frule-tac* $x = xa$ **and** $n = k$ **in** *le-pre-le*, *blast*, *assumption+*)
apply (*simp add*:*sc-assoc*[*THEN sym*] *sprod-one*)
apply(*simp add*:*Pi-def*)
apply *blast*
apply *simp*
done

lemma (**in** *Module*) *NAK*:⟦¬ *zeroring R*; *M fgover R*; *ideal R A*; $A \subseteq$ *J-rad R*;
$A \odot_R M = carrier\ M$ ⟧ ⟹ $carrier\ M = \{\mathbf{0}\}$
apply (*cut-tac sc-Ring*)
apply (*frule NAKTr1*, *erule exE*, *erule conjE*)
apply (*simp add*:*finite-generator-def*, *frule conjunct1*, *frule conjunct2*,
fold finite-generator-def)
apply (*case-tac* $H = \{\}$, *simp add*:*generator-def linear-span-def*,
frule-tac $A = H$ **in** *nonempty-card-pos1*, *assumption+*)
apply (*case-tac card* $H = Suc\ 0$, *simp*,
frule-tac $A = H$ **in** *nonempty-ex*, *erule exE*, *rename-tac H h*,
frule-tac $t = H$ **and** $a1 = h$ **in** *card1-tr0*[*THEN sym*], *assumption+*, *simp*)
apply (*frule-tac* $H = \{h\}$ **in** *generator-sub-carrier*,
frule sym, *thin-tac* $A \odot_R M = carrier\ M$, *simp*,
thin-tac carrier $M = A \odot_R M$, *simp add*:*smodule-ideal-coeff-def*,
simp add:*generator-def*)
apply (*erule conjE*, *rotate-tac −1*, *frule sym*,
thin-tac linear-span R M (*carrier R*) $\{h\} = carrier\ M$, *simp*)
apply (*rotate-tac −1*, *frule sym*,
frule-tac $a = h$ **and** $A = linear\text{-}span\ R\ M\ (carrier\ R)\ \{h\}$ **and**

B = *carrier* M **in** *eq-set-inc*, *assumption*+,
frule-tac $a = h$ **and** A = *carrier* M **in** *singleton-sub*)
apply (*frule-tac* $H = \{h\}$ **in** *l-spanA-l-span*[*of* A], *assumption*+,
thin-tac carrier M = *linear-span* R M (*carrier* R) $\{h\}$,
thin-tac linear-span R M (*carrier* R) $\{h\}$ = *carrier* M, *simp*,
thin-tac $h \in$ *linear-span* R M (*carrier* R) $\{h\}$,
thin-tac linear-span R M A (*linear-span* R M (*carrier* R) $\{h\}$) =
linear-span R M A $\{h\}$,
thin-tac (*LEAST* j. $\exists L$. *finite* $L \wedge$
$L \subseteq$ *linear-span* R M (*carrier* R) $\{h\} \wedge$
linear-span R M (*carrier* R) L =
linear-span R M (*carrier* R) $\{h\} \wedge$
j = *card* L) = *Suc 0*)
apply (*frule-tac* $h = h$ **and** $x = h$ **in** *mem-single-l-span1*[*of* A],
assumption+, *erule bexE*,
frule-tac $h = a$ **in** *Ring.ideal-subset*[*of* R A], *assumption*+,
frule-tac $a = a$ **and** $m = h$ **in** *sc-mem*, *assumption*+,
frule-tac $x = a \cdot_s h$ **in** *ag-mOp-closed*)
apply (*frule-tac* $a = h$ **and** $b = a \cdot_s h$ **and** $c = -_a (a \cdot_s h)$ **in**
ag-pOp-add-r, *assumption*+,
thin-tac $h = a \cdot_s h$, *simp add*:*ag-r-inv1*,
simp add:*sc-minus-am1*,
frule Ring.ring-is-ag[*of* R],
frule-tac $x = a$ **in** *aGroup.ag-mOp-closed*, *assumption*)
apply (*subgoal-tac* $h \pm (-_{a\,R}\ a) \cdot_s h = 1_{r\,R} \cdot_s h \pm\ (-_{a\,R}\ a) \cdot_s h$, *simp*)
apply (*frule Ring.ring-one*[*of* R],
simp add:*sc-l-distr*[*THEN sym*],
frule-tac $x = a$ **in** *Ring.J-rad-unit* [*of* R], *assumption*,
simp add:*subsetD*,
drule-tac $a = 1_{r\,R}$ **in** *forall-spec*, *assumption*,
simp add:*Ring.ring-r-one*,
simp add:*Unit-def*, *erule conjE*, *erule bexE*,
simp add:*Ring.ring-tOp-commute*,
frule-tac $a = b$ **and** $b = 1_{r\,R} \pm_R -_{a\,R}\ a$ **and** $m = h$ **in** *sc-assoc*,
rule aGroup.ag-pOp-closed, *assumption*+, *simp add*:*sprod-one sc-a-0*,
simp add:*l-span-zero*, *simp add*:*sprod-one*)

apply (*rotate-tac* -1, *frule not-sym*,
frule-tac m = *Suc 0* **and** n = *card* H **in** *noteq-le-less*, *assumption*,
thin-tac Suc 0 $\leq$ *card* H, *thin-tac card* $H \neq$ *Suc 0*)
apply (*frule-tac* a = *Suc 0* **and** b = *card* H **and** c = *Suc 0* **in**
diff-less-mono, *simp*,
thin-tac Suc 0 $<$ *card* H)
apply (*frule-tac* $H = H$ **and** k = *card* H $-$ *Suc 0* **in** *NAKTr5*[*of* A],
assumption+, *simp*, *simp*)
apply (*erule bexE*)
apply (*frule-tac* $H = H$ **and** *?H1.0* $= h$ ‘ $\{j.\ j \leq$ *card* H $-$ *Suc 0* $-$ *Suc 0*$\}$ **in**
generator-generator,
frule-tac $f = h$ **and** $A = \{j.\ j \leq$ *card* H $-$ *Suc 0*$\}$ **and** B = *carrier* M

and *?A1.0* = {*j*. *j* ≤ *card H* − *Suc 0* − *Suc 0*} **in** *image-sub*,
rule subsetI, *simp*+)
apply (*erule conjE*)
apply (*cut-tac f* = *h* **and** *n* = *card H* − *Suc* (*Suc 0*) **in** *image-Nset-Suc*)
apply (*frule-tac m* = *Suc 0* **and** *n* = *card H* **in** *Suc-leI*)
apply (*simp add*:*Suc-Suc-Tr*)
apply (*frule-tac f* = *h* **and** *A* = {*j*. *j* ≤ *card H* − *Suc 0*} **and**
B = *carrier M* **and** *?A1.0* = {*j*. *j* ≤ *card H* − *Suc 0* − *Suc 0*} **in**
image-sub,
rule subsetI, *simp*)
apply (*frule-tac H* = *h* ‘ {*j*. *j* ≤ *card H* − *Suc 0* − *Suc 0*} **in**
lin-span-coeff-mono[*of A*], *simp*, *simp*)
apply (*frule-tac c* = *h* (*card H* − *Suc 0*) **and** *A* =
linear-span R M A (*h* ‘ {*j*. *j* ≤ *card H* − *Suc* (*Suc 0*)}) **and**
B = *linear-span R M* (*carrier R*)
(*h* ‘ {*j*. *j* ≤ *card H* − *Suc* (*Suc 0*)}) **in** *subsetD*, *assumption*)
apply (*rule-tac A* = *H* **and** *B* = *insert* (*h* (*card H* − *Suc 0*))
(*h* ‘ {*j*. *j* ≤ *card H* − *Suc* (*Suc 0*)}) **and**
C = *linear-span R M* (*carrier R*)
(*h* ‘ {*j*. *j* ≤ *card H* − *Suc* (*Suc 0*)}) **in** *subset-trans*)
apply *simp*
apply (*rule-tac A* = *h* ‘ {*j*. *j* ≤ *card H* − *Suc* (*Suc 0*)} **and**
B = *linear-span R M* (*carrier R*)
(*h* ‘ {*j*. *j* ≤ *card H* − *Suc* (*Suc 0*)}) **and** *a* = *h* (*card H* − *Suc 0*)
in *insert-sub*)
apply (*rule subsetI*)
apply (*rule-tac H* = *h* ‘ {*j*. *j* ≤ *card H* − *Suc* (*Suc 0*)} **and** *h* = *x* **in**
h-in-linear-span, *assumption*+)
apply (*frule-tac H* = *h* ‘ {*j*. *j* ≤ *card H* − *Suc 0* − *Suc 0*} **in** *NAKTr3*)
apply (*simp add*:*generator-sub-carrier*)
apply (*rule finite-imageI*, *simp*)
apply (*thin-tac H* = *h* ‘ {*j*. *j* ≤ *card H* − *Suc 0*} ∧
h (*card H* − *Suc 0*)
∈ *linear-span R M A* (*h* ‘ {*j*. *j* ≤ *card H* − *Suc 0* − *Suc 0*}))
apply (*cut-tac k* = *card H* − *Suc 0* − *Suc 0* **in** *finite-Collect-le-nat*,
frule-tac A = {*j*. *j* ≤ *card H* − *Suc 0* − *Suc 0*} **and** *f* = *h* **in**
card-image-le,
simp)
apply (*frule-tac m* = *Suc 0* **and** *n* = *card H* **in** *Suc-leI*,
simp add:*Suc-Suc-Tr*, *simp add*:*finite-generator-def*)
apply (*cut-tac k* = *card H* − *Suc 0* − *Suc 0* **in** *finite-Collect-le-nat*,
frule-tac F = {*j*. *j* ≤ *card H* − *Suc 0* − *Suc 0*} **and** *h* = *h* **in**
finite-imageI, *simp add*:*finite-generator-def*)
done

lemma (**in** *Module*) *fg-qmodule*:⟦ *M fgover R*; *submodule R M N*⟧ ⟹
(*M* /$_m$ *N*) *fgover R*
apply (*frule mpj-mHom*[*of N*])
apply (*frule mpj-surjec* [*of N*])

apply (*rule surjec-finitely-gen* [*of* M $/_m$ N *mpj* M N])
apply (*simp add*:*qmodule-module*)
apply *assumption*+
done

lemma (**in** *Module*) *NAK1*:⟦¬ *zeroring* R; M *fgover* R; *submodule* R M N;
ideal R A; $A \subseteq$ *J-rad* R; *carrier* $M = A \odot_R M \mp N$ ⟧ ⟹ *carrier* $M = N$
apply (*subgoal-tac* $A \odot_R (M /_m N) =$ *carrier* $(M /_m N)$)
apply (*frule fg-qmodule* [*of* N], *assumption*+)
apply (*frule qmodule-module* [*of* N])
apply (*frule Module.NAK* [*of* M $/_m$ N R A], *assumption*+,
thin-tac R *module* M $/_m$ N, *thin-tac* M $/_m$ N *fgover* R,
thin-tac $A \odot_R M /_m N =$ *carrier* $(M /_m N)$,
thin-tac *carrier* $M = A \odot_R M \mp N$)
apply (*simp add*:*qmodule-def*)
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*frule-tac* $m = x$ **in** *set-mr-cos-mem*[*of* N], *assumption*)
apply *simp*
apply (*frule-tac* $m = x$ **in** *m-in-mr-coset*[*of* N], *assumption*, *simp*)
apply (*simp add*:*submodule-subset*)
apply (*subst smodule-ideal-coeff-def*)
apply (*frule qmodule-module* [*of* N])
apply (*frule Module.lin-span-sub-carrier*[*of* M $/_m$ N R A *carrier* $(M /_m N)$],
assumption, *simp*)
apply (*rule equalityI*, *assumption*+)
apply (*thin-tac linear-span* R $(M /_m N)$ A (*carrier* $(M /_m N)$) ⊆
carrier $(M /_m N)$)
apply (*rule subsetI*)
apply (*simp add*:*qmodule-carr*)
apply (*frule-tac* $x = x$ **in** *mem-set-mr-cos*[*of* N], *assumption*, *erule bexE*)
apply (*frule smodule-ideal-coeff-is-Submodule*[*of* A],
frule submodule-subset,
frule submodule-subset[*of* $A \odot_R M$])
apply (*frule-tac* $x = m$ **in** *mem-set-sum*[*of* $A \odot_R M$ N], *assumption*+,
simp, (*erule bexE*)+)
apply *simp*
apply (*frule submodule-asubg*[*of* N])
apply (*frule sym*, *thin-tac* *carrier* $M = A \odot_R M \mp N$, *simp*)
apply (*frule-tac* $c = h$ **in** *subsetD*[*of* $A \odot_R M$ *carrier* M], *assumption*+,
frule-tac $c = k$ **in** *subsetD*[*of* N *carrier* M], *assumption*+,
frule-tac $x = h$ **and** $y = k$ **in** *ag-pOp-commute*, *assumption*+, *simp*)
apply (*simp add*:*arcos-fixed*[*THEN sym*, *of* N], *thin-tac* $h \pm k = k \pm h$,
thin-tac $m = k \pm h$,
thin-tac *submodule* R M $(A \odot_R M)$,
thin-tac $A \odot_R M \subseteq$ *carrier* M,
thin-tac $A \odot_R M \mp N =$ *carrier* M)
apply (*simp add*:*smodule-ideal-coeff-def*)
apply (*frule-tac* $h = h$ **in** *linmap-im-linspan1* [*of* A M $/_m$ N *mpj* M N

carrier M], *assumption*+,
rule mpj-mHom[*of N*], *assumption*, *simp*, *assumption*)
apply (*frule-tac m = h* **in** *elem-mpj*[*of* - *N*], *assumption*,
frule-tac mpj-surjec[*of N*],
simp add:*surjec-def surj-to-def qmodule-carr*)
done

5.7 Direct sum and direct products of modules

definition
prodM-sprod :: [('*r*, '*m*) *Ring-scheme*, '*i set*,
'*i* ⇒ ('*a*, '*r*, '*m1*) *Module-scheme*] ⇒ '*r* ⇒ ('*i* ⇒ '*a*) ⇒ ('*i* ⇒ '*a*) **where**
prodM-sprod R I A = (λ*a*∈*carrier R*. λ*g*∈*carr-prodag I A*.
(λ*j*∈*I*. *a* ·$_{s(A\ j)}$ (*g j*)))

definition
prodM :: [('*r*, '*m*) *Ring-scheme*, '*i set*, '*i* ⇒ ('*a*, '*r*, '*m1*) *Module-scheme*] ⇒
(| *carrier*:: ('*i* ⇒ '*a*) *set*,
pop::['*i* ⇒ '*a*, '*i* ⇒ '*a*] ⇒ ('*i* ⇒ '*a*),
mop:: ('*i* ⇒ '*a*) ⇒ ('*i* ⇒ '*a*), *zero*::('*i* ⇒ '*a*),
sprod :: ['*r*, '*i* ⇒ '*a*] ⇒ ('*i* ⇒ '*a*) |) **where**
prodM R I A = (|*carrier* = *carr-prodag I A*,
pop = *prod-pOp I A*, *mop* = *prod-mOp I A*,
zero = *prod-zero I A*, *sprod* = *prodM-sprod R I A* |)

definition
mProject :: [('*r*, '*m*) *Ring-scheme*, '*i set*,
'*i* ⇒ ('*a*, '*r*, '*more*) *Module-scheme*, '*i*] ⇒ ('*i* ⇒ '*a*) ⇒ '*a* **where**
mProject R I A j = (λ*f*∈*carr-prodag I A*. *f j*)

abbreviation
PRODMODULES ((*3*mΠ_ _ -) [*72*,*72*,*73*]*72*) **where**
mΠ$_{R\ I}$ *A* == *prodM R I A*

lemma (**in** *Ring*) *prodM-carr*:⟦∀ *i*∈*I*. (*R module* (*M i*))⟧ ⟹
carrier (*prodM R I M*) = *carr-prodag I M*
apply (*simp add*:*prodM-def*)
done

lemma (**in** *Ring*) *prodM-mem-eq*:⟦∀ *i*∈*I*. (*R module* (*M i*));
m1 ∈ *carrier* (*prodM R I M*); *m2* ∈ *carrier* (*prodM R I M*);
∀ *i*∈*I*. *m1 i* = *m2 i* ⟧ ⟹ *m1* = *m2*
apply (*simp add*:*prodM-carr* [*of I M*])
apply (*rule carr-prodag-mem-eq* [*of I M*])
apply (*rule ballI*, *drule-tac x* = *k* **in** *bspec*, *assumption*,
simp add:*Module.module-is-ag*)
apply *assumption*+
done

lemma (**in** *Ring*) *prodM-sprod-mem*:⟦∀ *i*∈*I*. (*R module* (*M i*)); *a* ∈ *carrier R*;
m ∈ *carr-prodag I M*⟧ ⟹ *prodM-sprod R I M a m* ∈ *carr-prodag I M*
apply (*simp add*:*prodM-sprod-def carr-prodag-def*)
apply (*erule conjE*)+
apply (*rule conjI*)
apply (*rule Pi-I*)
apply (*simp add*:*Un-carrier-def*)
apply (*drule-tac x* = *x* **in** *bspec*, *assumption*,
drule-tac x = *x* **in** *bspec*, *assumption*+)

apply (*frule-tac R* = *R* **and** *M* = *M x* **and** *a* = *a* **and** *m* = *m x* **in**
Module.sc-mem, *assumption*+, *blast*)
apply (*rule ballI*,
drule-tac x = *i* **in** *bspec*, *assumption*,
drule-tac x = *i* **in** *bspec*, *assumption*+)

apply (*frule-tac M* = *M i* **and** *a* = *a* **and** *m* = *m i* **in**
Module.sc-mem [*of* - *R*], *assumption*+)
done

lemma (**in** *Ring*) *prodM-sprod-val*:⟦∀ *i*∈*I*. (*R module* (*M i*)); *a* ∈ *carrier R*;
m ∈ *carr-prodag I M*; *j* ∈ *I*⟧ ⟹ (*prodM-sprod R I M a m*) *j* = *a* ·$_{s(M\ j)}$ (*m j*)
apply (*simp add*:*prodM-sprod-def*)
done

lemma (**in** *Ring*) *prodM-Module*:∀ *i*∈*I*. (*R module* (*M i*)) ⟹
R module (*prodM R I M*)
apply (*rule Module.intro*)
apply (*cut-tac prodag-aGroup*[*of I M*])
apply (*simp add*:*prodM-def prodag-def aGroup-def*)
apply (*erule conjE*)+
apply (*rule allI*, *rule impI*)
apply (*rotate-tac* −*2*,
frule-tac a = *a* **in** *forall-spec*, *assumption*)
apply (*thin-tac* ∀ *a*. *a* ∈ *carr-prodag I M* ⟶
prod-pOp I M (*prod-zero I M*) *a* = *a*,
thin-tac ∀ *i*∈*I*. *R module M i*,
thin-tac ∀ *a*. *a* ∈ *carr-prodag I M* ⟶ (∀ *b*. *b* ∈ *carr-prodag I M* ⟶
(∀ *c*. *c* ∈ *carr-prodag I M* ⟶ *prod-pOp I M* (*prod-pOp I M a b*) *c* =
prod-pOp I M a (*prod-pOp I M b c*))),
thin-tac ∀ *a*. *a* ∈ *carr-prodag I M* ⟶
prod-pOp I M (*prod-mOp I M a*) *a* = *prod-zero I M*,
frule-tac a = *prod-zero I M* **in** *forall-spec*, *assumption*,
thin-tac ∀ *a*. *a* ∈ *carr-prodag I M* ⟶ (∀ *b*. *b* ∈ *carr-prodag I M* ⟶
prod-pOp I M a b = *prod-pOp I M b a*),
frule-tac a = *a* **in** *forall-spec*, *assumption*,
thin-tac ∀ *b*. *b* ∈ *carr-prodag I M* ⟶ *prod-pOp I M* (*prod-zero I M*) *b* =
prod-pOp I M b (*prod-zero I M*), *simp*)
apply (*rule ballI*, *drule-tac x* = *k* **in** *bspec*, *assumption*,

simp add:*Module.module-is-ag*)
apply (*rule Module-axioms.intro*)
apply (*cut-tac Ring*, *assumption*)
apply (*simp add*:*prodM-carr*, *simp add*:*prodM-def prodM-sprod-mem*)
apply (*simp add*:*prodM-carr*, *simp add*:*prodM-def*)
apply (*frule-tac* $a = a$ **and** $m = m$ **in** *prodM-sprod-mem*[*of I M*], *assumption*+,
frule-tac $a = b$ **and** $m = m$ **in** *prodM-sprod-mem*[*of I M*], *assumption*+,
(*cut-tac ring-is-ag*,
frule-tac $x = a$ **and** $y = b$ **in** *aGroup.ag-pOp-closed*[*of R*], *assumption*+),
frule-tac $a = a \pm b$ **and** $m = m$ **in** *prodM-sprod-mem*[*of I M*],
assumption+)
apply (*rule prodM-mem-eq*[*of I M*], *assumption*, *simp add*:*prodM-carr*)
apply (*simp add*:*prodM-carr*,
rule-tac $X = $ *prodM-sprod R I M a m* **and** $Y = $ *prodM-sprod R I M b m*
in *prod-pOp-mem*[*of I M*],
(*rule ballI*, *drule-tac* $x = k$ **in** *bspec*, *assumption*,
simp add:*Module.module-is-ag*),
assumption+)
apply (*rule ballI*,
simp add:*prodM-sprod-def prod-pOp-def*,
frule-tac $x = i$ **in** *bspec*, *assumption*,
thin-tac $\forall i \in I.$ *R module M i*,
thin-tac $(\lambda j \in I.\ a \cdot_{s\,M\,j} m\ j) \in$ *carr-prodag I M*,
thin-tac $(\lambda j \in I.\ b \cdot_{s\,M\,j} m\ j) \in$ *carr-prodag I M*,
thin-tac $(\lambda j \in I.\ (a \pm b) \cdot_{s\,M\,j} m\ j) \in$ *carr-prodag I M*,
simp add:*carr-prodag-def*, (*erule conjE*)+,
drule-tac $x = i$ **in** *bspec*, *assumption*+)

apply (*cut-tac Ring*, *simp add*:*Module.sc-l-distr*)
apply (*simp add*:*prodM-carr*)
apply (*simp add*:*prodM-def*)
apply (*cut-tac* $X = m$ **and** $Y = n$ **in** *prod-pOp-mem*[*of I M*],
(*rule ballI*, *frule-tac* $x = k$ **in** *bspec*, *assumption*,
thin-tac $\forall i \in I.$ *R module M i*, *simp add*:*Module.module-is-ag*),
assumption+)
apply (*frule-tac* $a = a$ **and** $m =$ *prod-pOp I M m n* **in** *prodM-sprod-mem*[*of I M*],
assumption+)

apply (*rule prodM-mem-eq*[*of I M*], *assumption*, *simp add*:*prodM-carr*)
apply (*simp add*:*prodM-carr*,
frule-tac $a = a$ **and** $m = m$ **in** *prodM-sprod-mem*[*of I M*], *assumption*+,
frule-tac $a = a$ **and** $m = n$ **in** *prodM-sprod-mem*[*of I M*], *assumption*+,
rule-tac $X = $ *prodM-sprod R I M a m* **and** $Y = $ *prodM-sprod R I M a n*
in *prod-pOp-mem*[*of I M*],
(*rule ballI*, *drule-tac* $x = k$ **in** *bspec*, *assumption*,
simp add:*Module.module-is-ag*),
assumption+)
apply (*rule ballI*)

apply (*frule-tac* $a = a$ **and** $m = m$ **in** *prodM-sprod-mem*[*of* I M], *assumption*+,
frule-tac $a = a$ **and** $m = n$ **in** *prodM-sprod-mem*[*of* I M], *assumption*+,
simp add:*prod-pOp-def prodM-sprod-def*,
drule-tac $x = i$ **in** *bspec*, *assumption*,
thin-tac $(\lambda x{\in}I.\ m\ x \pm_{M\ x} n\ x) \in$ *carr-prodag* I M,
thin-tac $(\lambda j{\in}I.\ a \cdot_{s M\ j}$ (*if* $j \in I$ *then* $m\ j \pm_{M\ j} n\ j$ *else undefined*))
$\in$ *carr-prodag* I M,
thin-tac $(\lambda j{\in}I.\ a \cdot_{s M\ j} m\ j) \in$ *carr-prodag* I M,
thin-tac $(\lambda j{\in}I.\ a \cdot_{s M\ j} n\ j) \in$ *carr-prodag* I M)
apply (*simp add*:*carr-prodag-def*, (*erule conjE*)+,
drule-tac $x = i$ **in** *bspec*, *assumption*,
drule-tac $x = i$ **in** *bspec*, *assumption*)

apply (*simp add*:*Module.sc-r-distr*)

apply (*simp add*:*prodM-carr*,
frule-tac $a = b$ **and** $m = m$ **in** *prodM-sprod-mem*[*of* I M], *assumption*+,
frule-tac $a = a$ **and** $m =$ *prodM-sprod* R I M b m **in** *prodM-sprod-mem*[*of* I M], *assumption*+)
apply (*frule-tac* $x = a$ **and** $y = b$ **in** *ring-tOp-closed*, *assumption*+,
frule-tac $a = a \cdot_r b$ **and** $m = m$ **in** *prodM-sprod-mem*[*of* I M],
assumption+)
apply (*simp add*:*prodM-def*,
rule prodM-mem-eq, *assumption*+, *simp add*:*prodM-carr*,
simp add:*prodM-carr*)
apply (*rule ballI*, *simp add*:*prodM-sprod-def*,
drule-tac $x = i$ **in** *bspec*, *assumption*,
thin-tac $(\lambda j{\in}I.\ b \cdot_{s M\ j} m\ j) \in$ *carr-prodag* I M,
thin-tac $(\lambda j{\in}I.\ a \cdot_{s M\ j}$ (*if* $j \in I$ *then* $b \cdot_{s M\ j} m\ j$ *else undefined*))
$\in$ *carr-prodag* I M,
thin-tac $(\lambda j{\in}I.\ (a \cdot_r b) \cdot_{s M\ j} m\ j) \in$ *carr-prodag* I M,
simp add:*carr-prodag-def*, (*erule conjE*)+,
drule-tac $x = i$ **in** *bspec*, *assumption*)

apply (*simp add*:*Module.sc-assoc*)

apply (*simp add*:*prodM-carr*, *cut-tac ring-one*,
frule-tac $a = 1_r$ **and** $m = m$ **in** *prodM-sprod-mem*[*of* I M], *assumption*+,
rule prodM-mem-eq, *assumption*+, *simp add*:*prodM-carr*,
simp add:*prodM-def*, *simp add*:*prodM-def*)
apply (*rule ballI*, *simp add*:*prodM-def*,
thin-tac prodM-sprod R I M 1_r $m \in$ *carr-prodag* I M,
simp add:*prodM-sprod-def*,
drule-tac $x = i$ **in** *bspec*, *assumption*,
simp add:*carr-prodag-def*, (*erule conjE*)+,
drule-tac $x = i$ **in** *bspec*, *assumption*,
simp add:*Module.sprod-one*)
done

definition
dsumM :: [('*r*, '*m*) *Ring-scheme*, '*i set*, '*i* ⇒ ('*a*, '*r*, '*more*) *Module-scheme*]
⇒ (| *carrier*:: ('*i* ⇒ '*a*) *set*,
pop::['*i* ⇒ '*a*, '*i* ⇒ '*a*] ⇒ ('*i* ⇒ '*a*),
mop:: ('*i* ⇒ '*a*) ⇒ ('*i* ⇒ '*a*),
zero::('*i* ⇒ '*a*),
sprod :: ['*r*, '*i* ⇒ '*a*] ⇒ ('*i* ⇒ '*a*) |) **where**

dsumM R I A = (| *carrier* = *carr-dsumag I A*,
pop = *prod-pOp I A*, *mop* = *prod-mOp I A*,
zero = *prod-zero I A*, *sprod* = *prodM-sprod R I A*|)

abbreviation
DSUMMOD ((*3*-Σ$_{d}$- -) [*72*,*72*,*73*]*72*) **where**
$_{R}\Sigma_{d I}$ *A* == *dsumM R I A*

lemma (**in** *Ring*) *dsumM-carr*:*carrier* (*dsumM R I M*) = *carr-dsumag I M*
by (*simp add*:*dsumM-def*)

lemma (**in** *Ring*) *dsum-sprod-mem*:⟦∀ *i*∈*I*. *R module M i*; *a* ∈ *carrier R*;
b ∈ *carr-dsumag I M*⟧ ⟹ *prodM-sprod R I M a b* ∈ *carr-dsumag I M*
apply (*simp add*:*carr-dsumag-def*)
apply (*simp add*:*finiteHom-def*, *erule conjE*, *erule exE*, (*erule conjE*)+,
frule-tac a = *a* **and** *m* = *b* **in** *prodM-sprod-mem*[*of I M*], *assumption*+)
apply (*subgoal-tac* ∀ *j*∈*I* − *H*. *prodM-sprod R I M a b j* = $\mathbf{0}_{M\,j}$, *blast*)
apply (*rule ballI*,
drule-tac x = *j* **in** *bspec*, *assumption*,
drule-tac x = *j* **in** *bspec*, *simp*)

apply (*subst prodM-sprod-def*, *simp*)
apply (*simp add*:*Module.sc-a-0*)
done

lemma (**in** *Ring*) *carr-dsum-prod*:*carr-dsumag I M* ⊆ *carr-prodag I M*
apply (*rule subsetI*) **apply** (*simp add*:*carr-dsumag-def finiteHom-def*)
done

lemma (**in** *Ring*) *carr-dsum-prod1*:
∀ *x*. *x* ∈ *carr-dsumag I M* ⟶ *x* ∈ *carr-prodag I M*
apply (*rule allI*) **apply** (*rule impI*)
apply (*cut-tac carr-dsum-prod* [*of I M*])
apply (*simp add*:*subsetD*)
done

lemma (**in** *Ring*) *carr-dsumM-mem-eq*:⟦∀ *i*∈*I*. *R module M i*; *x* ∈ *carr-dsumag I M*;
y ∈ *carr-dsumag I M*; ∀ *j*∈*I*. *x j* = *y j*⟧ ⟹ *x* = *y*
apply (*cut-tac carr-dsum-prod* [*of I M*])
apply (*frule subsetD* [*of carr-dsumag I M carr-prodag I M x*], *assumption*+,

frule subsetD [*of carr-dsumag I M carr-prodag I M y*], *assumption*+)
apply (*subgoal-tac* $\forall\, i{\in}I.$ *aGroup* (*M i*))
apply (*rule carr-prodag-mem-eq* [*of I M*], *assumption*+)
apply (*rule ballI*)
apply (*drule-tac x* = *i* **in** *bspec*, *assumption*,
simp add:*Module.module-is-ag*)
done

lemma (**in** *Ring*) *dsumM-Module*:$\forall\, i{\in}I.$ *R module* (*M i*) $\Longrightarrow$ *R module* ($_R\Sigma_{d\,I}$ *M*)
apply (*rule Module.intro*)
apply (*cut-tac prodag-aGroup*[*of I M*])
apply (*cut-tac carr-dsum-prod1*[*of I M*])
apply (*simp add*:*dsumM-def prodag-def aGroup-def*,
cut-tac dsum-pOp-func[*of I M*], *simp*,
cut-tac dsum-iOp-func[*of I M*], *simp*,
cut-tac dsum-zero-func[*of I M*], *simp*)
apply (*erule conjE*)+
apply (*rule allI*, *rule impI*,
thin-tac $\forall\, a.\ a \in$ *carr-prodag I M* $\longrightarrow$
($\forall\, b.\ b \in$ *carr-prodag I M* $\longrightarrow$
($\forall\, c.\ c \in$ *carr-prodag I M* $\longrightarrow$
prod-pOp I M (*prod-pOp I M a b*) *c* =
prod-pOp I M a (*prod-pOp I M b c*))),
thin-tac $\forall\, i{\in}I.$ *R module M i*)
apply (*frule-tac a* = *a* **in** *forall-spec*, *assumption*,
drule-tac a = *prod-zero I M* **in** *forall-spec*, *assumption*)
apply (*drule-tac a* = *a* **in** *forall-spec*, *assumption*,
rotate-tac −*1*,
drule-tac a = *prod-zero I M* **in** *forall-spec*, *assumption*, *simp*)

apply (*rule ballI*, *drule-tac x* = *k* **in** *bspec*, *assumption*,
simp add:*Module.module-is-ag*)
apply (*rule ballI*, *drule-tac x* = *k* **in** *bspec*, *assumption*,
(*simp add*:*Module.module-is-ag*)+)
apply (*rule ballI*, *drule-tac x* = *k* **in** *bspec*, *assumption*+,
simp add:*Module.module-is-ag*)+

apply (*rule Module-axioms.intro*)
apply (*cut-tac Ring*, *simp del*: *Pi-split-insert-domain*,
simp add:*dsumM-carr dsumM-def del*: *Pi-split-insert-domain*,
simp add:*dsum-sprod-mem del*: *Pi-split-insert-domain*)

apply (*cut-tac ring-is-ag*,
frule-tac x = *a* **and** *y* = *b* **in** *aGroup.ag-pOp-closed*, *assumption*+,
simp add:*dsumM-carr del*: *Pi-split-insert-domain*,
frule-tac a = *a* **and** *b* = *m* **in** *dsum-sprod-mem*[*of I M*], *assumption*+,
frule-tac a = *b* **and** *b* = *m* **in** *dsum-sprod-mem*[*of I M*], *assumption*+,
frule-tac a = *a* ± *b* **and** *b* = *m* **in** *dsum-sprod-mem*[*of I M*], *assumption*+,

simp add:*dsumM-def del*: *Pi-split-insert-domain*,
cut-tac X = *prodM-sprod* R I M a m **and** Y = *prodM-sprod* R I M b m **in**
prod-pOp-mem[*of* I M],
(*rule ballI*, *frule-tac* x = k **in** *bspec*, *assumption*,
thin-tac $\forall i{\in}I.$ R *module* M i, *simp add*:*Module.module-is-ag del*: *Pi-split-insert-domain*),
(*cut-tac carr-dsum-prod*[*of* I M], *simp add*:*subsetD del*: *Pi-split-insert-domain*)+)
apply(*cut-tac carr-dsum-prod1*[*of* I M])
apply(*rule prodM-mem-eq*)
apply(*assumption*+)
apply((*simp add*:*prodM-carr del*: *Pi-split-insert-domain*)+)
apply(*rule ballI*)
apply(*simp add*:*prodM-sprod-def prod-pOp-def del*: *Pi-split-insert-domain PiE-restrict*)
apply(*thin-tac* $(\lambda j{\in}I.\ a \cdot_{s\,M\,j} m\ j) \in$ *carr-dsumag* I M,
thin-tac $(\lambda j{\in}I.\ b \cdot_{s\,M\,j} m\ j) \in$ *carr-dsumag* I M,
thin-tac $(\lambda j{\in}I.\ (a \pm b) \cdot_{s\,M\,j} m\ j) \in$ *carr-dsumag* I M,
thin-tac $(\lambda x{\in}I.$ (*if* $x \in I$ *then* $a \cdot_{s\,M\,x} m\ x$ *else undefined*) $\pm_{M\,x}$
(*if* $x \in I$ *then* $b \cdot_{s\,M\,x} m\ x$ *else undefined*))
$\in$ *carr-prodag* I M)
apply (*frule-tac* a = m **in** *forall-spec*, *assumption*,
thin-tac $\forall x.\ x \in$ *carr-dsumag* I $M \longrightarrow x \in$ *carr-prodag* I M,
frule-tac x = i **in** *bspec*, *assumption*,
thin-tac $\forall i{\in}I.$ R *module* M i,
thin-tac $m \in$ *carr-dsumag* I M,
simp add:*carr-prodag-def*, (*erule conjE*)+,
frule-tac x = i **in** *bspec*, *assumption*,
thin-tac $\forall i{\in}I.$ $m\ i \in$ *carrier* $(M\ i)$)
apply (*simp add*:*Module.sc-l-distr*)

apply (*simp add*:*dsumM-carr*,
frule-tac a = a **and** b = m **in** *dsum-sprod-mem*[*of* I M], *assumption*+,
frule-tac a = a **and** b = n **in** *dsum-sprod-mem*[*of* I M], *assumption*+)
apply (*simp add*:*dsumM-def*, *cut-tac carr-dsum-prod1*[*of* I M],
cut-tac X = m **and** Y = n **in** *prod-pOp-mem*[*of* I M],
(*rule ballI*, *frule-tac* x = k **in** *bspec*, *assumption*,
thin-tac $\forall i{\in}I.$ R *module* M i, *simp add*:*Module.module-is-ag*), *simp*+,
frule-tac a = a **and** m = *prod-pOp* I M m n **in** *prodM-sprod-mem*[*of* I M],
assumption, *simp*)
apply (*cut-tac* X = *prodM-sprod* R I M a m **and** Y = *prodM-sprod* R I M a n
in
prod-pOp-mem[*of* I M],
(*rule ballI*, *frule-tac* x = k **in** *bspec*, *assumption*,
thin-tac $\forall i{\in}I.$ R *module* M i, *simp add*:*Module.module-is-ag*), *simp*+,
rule-tac *?m1.0* = *prodM-sprod* R I M a (*prod-pOp* I M m n) **and**
?m2.0 = *prod-pOp* I M (*prodM-sprod* R I M a m) (*prodM-sprod* R I M a n)
in *prodM-mem-eq*[*of* I M], *assumption*, (*simp add*:*prodM-carr*)+)
apply (*rule ballI*,
frule-tac x = i **in** *bspec*, *assumption*,
thin-tac prodM-sprod R I M a $m \in$ *carr-dsumag* I M,

thin-tac prodM-sprod R I M a n ∈ carr-dsumag I M)
apply (*rotate-tac 1*,
frule-tac a = m **in** *forall-spec*, *assumption*,
frule-tac a = n **in** *forall-spec*, *assumption*,
thin-tac ∀ *x*. *x ∈ carr-dsumag I M* ⟶ *x ∈ carr-prodag I M*,
thin-tac m ∈ carr-dsumag I M,
thin-tac n ∈ carr-dsumag I M)
apply (*frule-tac a = a* **and** *m = m* **in** *prodM-sprod-mem*[*of I M*], *assumption*+,
frule-tac a = a **and** *m = n* **in** *prodM-sprod-mem*[*of I M*], *assumption*+)
apply (*simp add*:*prodM-sprod-def prod-pOp-def del*: *PiE-restrict*,
thin-tac (λ*j*∈*I*. *a* $\cdot_{s\,M\,j}$ (*if j ∈ I then m j* $\pm_{M\,j}$ *n j else undefined*))
∈ *carr-prodag I M*,
thin-tac (λ*x*∈*I*. (*if x ∈ I then a* $\cdot_{s\,M\,x}$ *m x else undefined*) $\pm_{M\,x}$
(*if x ∈ I then a* $\cdot_{s\,M\,x}$ *n x else undefined*))
∈ *carr-prodag I M*,
thin-tac (λ*j*∈*I*. *a* $\cdot_{s\,M\,j}$ *m j*) ∈ *carr-prodag I M*,
thin-tac (λ*j*∈*I*. *a* $\cdot_{s\,M\,j}$ *n j*) ∈ *carr-prodag I M*,
thin-tac (λ*x*∈*I*. *m x* $\pm_{M\,x}$ *n x*) ∈ *carr-prodag I M*)
apply (*simp add*:*carr-prodag-def*, (*erule conjE*)+,
frule-tac x = i **in** *bspec*, *assumption*,
thin-tac ∀ *i*∈*I*. *R module M i*,
frule-tac x = i **in** *bspec*, *assumption*,
thin-tac ∀ *i*∈*I*. *m i ∈ carrier* (*M i*),
frule-tac x = i **in** *bspec*, *assumption*,
simp add:*Module.sc-r-distr*)

apply (*frule-tac x = a* **and** *y = b* **in** *ring-tOp-closed*, *assumption*+,
simp add:*dsumM-carr*,
frule-tac a = b **and** *b = m* **in** *dsum-sprod-mem*[*of I M*], *assumption*+,
frule-tac a = a **and** *b = prodM-sprod R I M b m* **in** *dsum-sprod-mem*[*of I M*],
assumption+,
frule-tac a = a $\cdot_r$ *b* **and** *b = m* **in** *dsum-sprod-mem*[*of I M*], *assumption*+)
apply (*cut-tac carr-dsum-prod1*[*of I M*])
apply (*simp add*:*dsumM-def*,
rule-tac ?m1.0 = prodM-sprod R I M (*a* $\cdot_r$ *b*) *m* **and**
?m2.0 = prodM-sprod R I M a (*prodM-sprod R I M b m*)
in *prodM-mem-eq*[*of I M*], *assumption*, (*simp add*:*prodM-carr*)+)
apply (*rule ballI*, *simp add*:*prodM-sprod-def*,
frule-tac x = i **in** *bspec*, *assumption*,
thin-tac ∀ *i*∈*I*. *R module M i*,
thin-tac (λ*j*∈*I*. *b* $\cdot_{s\,M\,j}$ *m j*) ∈ *carr-dsumag I M*,
thin-tac (λ*j*∈*I*. *a* $\cdot_{s\,M\,j}$ (*if j ∈ I then b* $\cdot_{s\,M\,j}$ *m j else undefined*))
∈ *carr-dsumag I M*,
thin-tac (λ*j*∈*I*. (*a* $\cdot_r$ *b*) $\cdot_{s\,M\,j}$ *m j*) ∈ *carr-dsumag I M*,
frule-tac a = m **in** *forall-spec*, *assumption*,
thin-tac ∀ *x*. *x ∈ carr-dsumag I M* ⟶ *x ∈ carr-prodag I M*,
thin-tac m ∈ carr-dsumag I M,
simp add:*carr-prodag-def*, (*erule conjE*)+,
frule-tac x = i **in** *bspec*, *assumption*,

thin-tac $\forall i{\in}I.\ m\ i \in carrier\ (M\ i)$,
simp add:*Module.sc-assoc*)

apply (*simp add*:*dsumM-carr*)
apply (*simp add*:*dsumM-def*)
apply (*cut-tac ring-one*,
cut-tac carr-dsum-prod1[*of I M*],
frule-tac $a = 1_r$ **and** $b = m$ **in** *dsum-sprod-mem*[*of I M*], *assumption*+,
rule-tac $?m1.0 = prodM\text{-}sprod\ R\ I\ M\ 1_r\ m$ **and**
$?m2.0 = m$ **in** *prodM-mem-eq*[*of I M*], *assumption*, (*simp add*:*prodM-carr*)+,
frule-tac $a = m$ **in** *forall-spec*, *assumption*,
frule-tac $a = prodM\text{-}sprod\ R\ I\ M\ 1_r\ m$ **in** *forall-spec*, *assumption*,
thin-tac $\forall x.\ x \in carr\text{-}dsumag\ I\ M \longrightarrow x \in carr\text{-}prodag\ I\ M$)
apply (*rule ballI*,
frule-tac $x = i$ **in** *bspec*, *assumption*,
thin-tac $\forall i{\in}I.\ R\ module\ M\ i$)
apply (*simp add*:*prodM-sprod-def*)
apply (*thin-tac* $(\lambda j{\in}I.\ 1_r \cdot_{s M\ j} m\ j) \in carr\text{-}dsumag\ I\ M$,
thin-tac $(\lambda j{\in}I.\ 1_r \cdot_{s M\ j} m\ j) \in carr\text{-}prodag\ I\ M$)
apply (*simp add*:*carr-prodag-def*, (*erule conjE*)+,
frule-tac $x = i$ **in** *bspec*, *assumption*,
simp add:*Module.sprod-one*)
done

definition
$ringModule :: ('r, 'b)\ Ring\text{-}scheme \Rightarrow ('r, 'r)\ Module$
$((M_)\ [998]999)$ **where**
$M_R = (\!|carrier = carrier\ R, pop = pop\ R, mop = mop\ R,$
zero $= zero\ R$, *sprod* $= tp\ R|\!)$

lemma (**in** *Ring*) *ringModule-Module*:*R module* M_R
apply (*rule Module.intro*)
apply (*simp add*:*aGroup-def*, *simp add*:*ringModule-def*,
simp add:*pop-closed*, *simp add*:*pop-aassoc*,
simp add:*pop-commute*, *simp add*:*mop-closed*,
simp add:*l-m*, *simp add*:*ex-zero*,
simp add:*l-zero*)

apply (*rule Module-axioms.intro*)
apply (*cut-tac Ring*, *assumption*,
simp add:*ringModule-def*, *simp add*:*ring-tOp-closed*,
simp add:*ringModule-def*, *simp add*:*ring-distrib2*,
simp add:*ringModule-def*, *simp add*:*ring-distrib1*,
simp add:*ringModule-def*, *simp add*:*ring-tOp-assoc*,
simp add:*ringModule-def*, *simp add*:*rg-l-unit*)
done

definition
$dsumMhom :: ['i\ set, 'i \Rightarrow ('a, 'r, 'm)\ Module\text{-}scheme,$

$'i \Rightarrow ('b, 'r, 'm1)$ *Module-scheme*, $'i \Rightarrow ('a \Rightarrow 'b)] \Rightarrow ('i \Rightarrow 'a) \Rightarrow$
$('i \Rightarrow 'b)$ **where**

dsumMhom I A B S = $(\lambda f{\in}$*carr-dsumag I A*. $(\lambda k{\in}I.\ (S\ k)\ (f\ k)))$

lemma (**in** *Ring*) *dsumMhom-mem*:$[\![\forall\, i{\in}I.\ R$ *module* $M\ i; \forall\, i{\in}I.\ R$ *module* $N\ i;$
$\forall\, i{\in}I.\ S\ i \in$ *mHom* $R\ (M\ i)\ (N\ i); x \in$ *carr-dsumag I M*$]\!]$
$\Longrightarrow$ *dsumMhom I M N S x* $\in$ *carr-dsumag I N*
apply (*subst carr-dsumag-def*, *simp*, *simp add*:*finiteHom-def*)
apply (*rule conjI*)
apply (*simp add*:*dsumMhom-def*)
apply (*cut-tac carr-dsum-prod1*[*of I M*],
drule-tac a = *x* **in** *forall-spec*, *assumption*) **apply** (
subst carr-prodag-def, *simp*)
apply (*rule conjI*,
rule Pi-I,
thin-tac x $\in$ *carr-dsumag I M*, *simp*,
simp add:*Un-carrier-def*,
drule-tac x = *xa* **in** *bspec*, *assumption*,
drule-tac x = *xa* **in** *bspec*, *assumption*,
drule-tac x = *xa* **in** *bspec*, *assumption*,
simp add:*carr-prodag-def*, (*erule conjE*)+,
drule-tac x = *xa* **in** *bspec*, *assumption*)

apply (*frule-tac R* = *R* **and** *M* = *M xa* **and** *N* = *N xa* **and** *f* = *S xa* **and**
m = *x xa* **in** *Module.mHom-mem*, *assumption*+, *blast*)

apply (*rule ballI*,
drule-tac x = *i* **in** *bspec*, *assumption*,
drule-tac x = *i* **in** *bspec*, *assumption*,
drule-tac x = *i* **in** *bspec*, *assumption*,
simp add:*carr-prodag-def*, (*erule conjE*)+,
drule-tac x = *i* **in** *bspec*, *assumption*)
apply (*simp add*:*Module.mHom-mem*)
apply (*cut-tac carr-dsum-prod1*[*of I M*],
drule-tac a = *x* **in** *forall-spec*, *assumption*)

apply (*simp add*:*dsumMhom-def*)
apply (*simp add*:*carr-prodag-def*, (*erule conjE*)+)
apply (*simp add*:*carr-dsumag-def*, *simp add*:*finiteHom-def*)
apply (*erule conjE*, *erule exE*)
apply (*subgoal-tac* $\forall\, j{\in}I - H.\ S\ j\ (x\ j) = \mathbf{0}_{N\ j}$, *blast*)
apply (*rule ballI*,
drule-tac x = *j* **in** *bspec*, *simp*,
drule-tac x = *j* **in** *bspec*, *simp*,
drule-tac x = *j* **in** *bspec*, *simp*,

```
        simp add:carr-prodag-def, (erule conjE)+,
        drule-tac x = j in bspec, simp)
  apply (simp add:Module.mHom-0)
done

lemma (in Ring) dsumMhom-mHom:⟦∀ i∈I. (R module (M i));
     ∀ i∈I. (R module (N i)); ∀ i∈I. ((S i) ∈ mHom R (M i) (N i))⟧ ⟹
            dsumMhom I M N S ∈ mHom R (dsumM R I M) (dsumM R I N)
apply (subst mHom-def, simp)
apply (rule conjI)
 apply (simp add:aHom-def,
        rule conjI,
        simp add:dsumM-def dsumMhom-mem)
apply (rule conjI,
        simp add:dsumMhom-def extensional-def dsumM-def)
apply (rule ballI)+
 apply (simp add:dsumM-def)
 apply (subgoal-tac ∀ i∈I. aGroup (M i),
        frule-tac X = a and Y = b in dsum-pOp-mem [of I M], assumption+,
        frule-tac x = prod-pOp I M a b in dsumMhom-mem [of I M N S],
        assumption+,
        frule-tac x = a in dsumMhom-mem [of I M N S], assumption+,
        frule-tac x = b in dsumMhom-mem [of I M N S], assumption+)
 apply (subgoal-tac ∀ i∈I. aGroup (N i))
 apply (frule-tac X = dsumMhom I M N S a and Y = dsumMhom I M N S b in

                                    dsum-pOp-mem [of I N], assumption+)
 apply (rule carr-dsumM-mem-eq [of I N], assumption+, rule ballI)
 apply (cut-tac carr-dsum-prod1 [of  I M], cut-tac carr-dsum-prod1 [of I N])
 apply (simp add:prod-pOp-def)
 apply (simp add:dsumMhom-def)
 apply (rule-tac R = R and M = M j and N = N j and m = a j and n = b j
       in  Module.mHom-add)
  apply simp+
  apply (simp add:carr-dsumag-def carr-prodag-def finiteHom-def,
         simp add:carr-dsumag-def carr-prodag-def finiteHom-def,
         rule ballI, rotate-tac 1,
         drule-tac x = i in bspec, assumption,
         simp add:Module.module-is-ag,
         rule ballI, drule-tac x = i in bspec, assumption,
         simp add:Module.module-is-ag)
apply (rule ballI)+
 apply (simp add:dsumM-def)
 apply (frule-tac a = a and b = m in dsum-sprod-mem [of I M], assumption+)
 apply (frule-tac x = prodM-sprod R I M a m in
                dsumMhom-mem [of I M N S], assumption+)
 apply (frule-tac x = m in dsumMhom-mem [of I M N S], assumption+)
 apply (frule-tac a = a and b = dsumMhom I M N S m in
                  dsum-sprod-mem [of I N], assumption+)
```

apply (*rule carr-dsumM-mem-eq* [*of I N*], *assumption*+)
apply (*rule ballI*)
apply (*cut-tac carr-dsum-prod1* [*of I*])
apply (*drule-tac a = m* **in** *forall-spec*, *assumption*)

apply (*cut-tac carr-dsum-prod1* [*of I N*],
frule-tac a = dsumMhom I M N S m **in** *forall-spec*,
assumption)
apply (*simp add*:*dsumMhom-def prodM-sprod-def*)
apply (*thin-tac* ($\lambda j{\in}I.\ a \cdot_{s\,M\ j} m\ j) \in$ *carr-dsumag I M*,
thin-tac ($\lambda k{\in}I.\ S\ k$ (*if* $k \in I$ *then* $a \cdot_{s\,M\ k} m\ k$ *else undefined*))
$\in$ *carr-dsumag I N*,
thin-tac ($\lambda k{\in}I.\ S\ k\ (m\ k)$) $\in$ *carr-dsumag I N*,
thin-tac ($\lambda j{\in}I.\ a \cdot_{s\,N\ j}$ (*if* $j \in I$ *then* $S\ j\ (m\ j)$ *else undefined*))
$\in$ *carr-dsumag I N*,
thin-tac $\forall x.\ x \in$ *carr-dsumag I N* $\longrightarrow x \in$ *carr-prodag I N*)
apply (*drule-tac x = j* **in** *bspec*, *assumption*,
drule-tac x = j **in** *bspec*, *assumption*,
drule-tac x = j **in** *bspec*, *assumption*,
simp add:*carr-prodag-def*, (*erule conjE*)+,
drule-tac x = j **in** *bspec*, *assumption*)
apply (*simp add*:*Module.mHom-lin*)
done

end

theory *Algebra9* **imports** *Algebra8* **begin**

5.8 Exact sequence

definition
Zm :: [('*r*, '*m*) *Ring-scheme*, '*a*] ⇒ ('*a*, '*r*) *Module* **where**
Zm R e = (| *carrier* = {*e*}, *pop* = $\lambda x{\in}\{e\}.\ \lambda y{\in}\{e\}.\ e$, *mop* =
$\lambda x{\in}\{e\}.\ e$, *zero* = *e*, *sprod* = $\lambda r{\in}$*carrier R*. $\lambda x{\in}\{e\}.\ e$ |)

lemma (**in** *Ring*) *Zm-Module*:*R module* (*Zm R e*)
apply (*simp add*:*Module-def aGroup-def Zm-def Module-axioms-def*)
apply (*simp add*:*ring-one*, *cut-tac Ring*, *simp*)
apply (*rule conjI*)
apply (*rule allI*, *rule impI*, *rule allI*, *rule impI*, *rule impI*)
apply (*cut-tac ring-is-ag*,
frule-tac x = a **and** *y = b* **in** *aGroup.ag-pOp-closed*, *assumption*+, *simp*)
apply (*rule allI*, *rule impI*, *rule allI*, *rule impI*, *rule impI*,
frule-tac x = a **and** *y = b* **in** *ring-tOp-closed*, *assumption*+, *simp*)
done

lemma (**in** *Ring*) *Zm-carrier*:*carrier* (*Zm R e*) = {*e*}
apply (*simp add*:*Zm-def*)

done

lemma (**in** *Ring*) *Zm-to-M-0*:⟦*R module M*; *f* ∈ *mHom R* (*Zm R e*) *M*⟧ ⟹
f e = $\mathbf{0}_M$
apply (*cut-tac Zm-Module* [*of e*])
apply (*frule Module.mHom-add* [*of Zm R e R M f e e*], *assumption*+,
(*simp add*:*Zm-carrier*)+,
frule-tac R = *R* **and** *M* = *Zm R e* **in** *Module.module-is-ag*,
frule-tac x = *e* **and** *y* = *e* **in** *aGroup.ag-pOp-closed*[*of Zm R e*],
(*simp add*:*Zm-carrier*)+)
apply (*frule-tac R* = *R* **and** *M* = *Zm R e* **and** *N* = *M* **and** *f* = *f* **and** *m* = *e*
in
Module.mHom-mem, *assumption*+, *simp add*:*Zm-carrier*,
frule sym, *thin-tac f e* = *f e* $\pm_M$ (*f e*),
frule-tac R = *R* **and** *M* = *M* **in** *Module.module-is-ag*,
frule aGroup.ag-eq-sol2 [*of M f e f e f e*], *assumption*+)
apply (*simp add*:*aGroup.ag-r-inv1*)
done

lemma (**in** *Ring*) *Z-to-M*:⟦*R module M*; *f* ∈ *mHom R* (*Zm R e*) *M*;
g ∈ *mHom R* (*Zm R e*) *M* ⟧ ⟹ *f* = *g*
apply (*rule-tac R* = *R* **and** *M* = *Zm R e* **and** *N* = *M* **in** *Module.mHom-eq*)
apply (*simp add*:*Zm-Module*)
apply *assumption*+
apply (*rule ballI*)
apply (*simp add*:*Zm-carrier*)
apply (*simp add*:*Zm-to-M-0* [*of - - e*])
done

lemma (**in** *Ring*) *mzeromap-mHom*:⟦*R module M*; *R module N*⟧ ⟹
mzeromap M N ∈ *mHom R M N*
apply (*simp add*:*mHom-def aHom-def*)
apply (*rule conjI*)
apply (*simp add*:*mzeromap-def*, *simp add*:*Module-def aGroup-def*)
apply (*rule conjI*)
apply (*simp add*:*mzeromap-def extensional-def*)
apply (*rule conjI*)
apply ((*rule ballI*)+,
frule-tac R = *R* **and** *M* = *M* **in** *Module.module-is-ag*,
frule-tac x = *a* **and** *y* = *b* **in** *aGroup.ag-pOp-closed* [*of M*],
assumption+,
simp add:*mzeromap-def*,
frule-tac R = *R* **and** *M* = *N* **in** *Module.module-is-ag*,
rule aGroup.ag-l-zero[*THEN sym*, *of N*], *assumption*+,
simp add:*aGroup.ag-inc-zero*)
apply (*rule ballI*)+
apply (*frule-tac a* = *a* **and** *m* = *m* **in** *Module.sc-mem* [*of M R*], *assumption*+,
simp add:*mzeromap-def*,
rule Module.sc-a-0 [*THEN sym*], *assumption*+)

done

lemma (**in** *Ring*) *HOM-carrier*:*carrier* (*HOM*$_R$ *M N*) = *mHom R M N*
apply (*simp add*:*HOM-def*)
done

lemma (**in** *Ring*) *mHom-Z-M*:*R module M* ⟹
  mHom R (*Zm R e*) *M* = {*mzeromap* (*Zm R e*) *M*}
apply (*rule equalityI*)
 apply (*rule subsetI*)
 apply *simp*
 apply (*cut-tac Zm-Module* [*of e*])
 apply (*frule mzeromap-mHom* [*of Zm R e M*], *assumption*+)
 apply (*simp add*:*Z-to-M*)
apply (*rule subsetI*) **apply** *simp*
 apply (*cut-tac Zm-Module*[*of e*],
  simp add:*mzeromap-mHom*)
done

lemma (**in** *Module*) *Modules-single-carrier-isom*:⟦*R module N*; *carrier M* = {**0**};
  carrier N = {**0**$_N$}⟧ ⟹ *M* ≅$_R$ *N*
apply (*subgoal-tac bijec*$_{M,\ N}$ (λ*x*∈{**0**}. **0**$_N$) ∧
  (λ*x*∈{**0**}. **0**$_N$) ∈ *mHom R M N*)
apply (*simp add*:*misomorphic-def*, *blast*,
  subgoal-tac (λ*x*∈{**0**}. **0**$_N$) ∈ *mHom R M N*, *simp*)
apply (*simp add*:*bijec-def injec-def surjec-def mHom-def*,
  simp add:*ker-def surj-to-def*)

apply (*simp add*:*mHom-def aHom-def*)
 apply (*cut-tac ag-inc-zero*, *simp add*:*ag-l-zero*)
 apply (*frule-tac R* = *R* **and** *M* = *N* **in** *Module.module-is-ag*,
  frule aGroup.ag-inc-zero[*of N*],
  simp add:*aGroup.ag-l-zero*[*of N*])
 apply (*simp add*:*sc-a-0 Module.sc-a-0*)
done

lemma (**in** *Ring*) *Zm-isom*:(*Zm R* (*e*::'*a*)) ≅$_R$ (*Zm R* (*u*::'*b*))
apply (*cut-tac Zm-Module*[*of e*], *cut-tac Zm-Module*[*of u*])
apply (*rule-tac R* = *R* **and** *M* = *Zm R e* **and** *N* = *Zm R u* **in**
  Module.Modules-single-carrier-isom, *assumption*+)
apply (*simp add*:*Zm-def*)+
done

lemma (**in** *Ring*) *HOM-Z-M-0*:*R module M* ⟹ *HOM*$_R$ (*Zm R e*) *M* ≅$_R$ (*Zm R e*)
 apply (*cut-tac Zm-Module*[*of e*],
  frule-tac M = *Zm R e* **and** *N* = *M* **in** *Module.HOM-is-module*,
  assumption+)
 apply (*cut-tac M* = *Zm R e* **and** *N* = *M* **in** *HOM-carrier*)

apply (*simp add*: *mHom-Z-M*)
apply (*simp add*:*Module.zero-HOM*)
apply (*rule-tac* $R = R$ **and** $M = HOM_R\ (Zm\ R\ e)\ M$ **and** $N = Zm\ R\ e$ **in**
Module.Modules-single-carrier-isom, *assumption*+)
apply (*simp add*:*Zm-def*)
done

lemma (**in** *Ring*) *M-to-Z*:⟦*R module M*; $f \in mHom\ R\ M\ (Zm\ R\ e)$;
$g \in mHom\ R\ M\ (Zm\ R\ e)$⟧ $\Longrightarrow f = g$
apply (*rule Module.mHom-eq* [*of M - Zm R e*], *assumption*+)
apply (*simp add*:*Zm-Module*, *assumption*+)
apply (*rule ballI*)
apply (*frule-tac* $m = m$ **in** *Module.mHom-mem* [*of M - Zm R e f*],
simp add:*Zm-Module*, *assumption*+)
apply (*frule-tac* $m = m$ **in** *Module.mHom-mem* [*of M - Zm R e g*],
simp add:*Zm-Module*, *assumption*+)
apply (*simp add*:*Zm-carrier*)
done

lemma (**in** *Ring*) *mHom-to-zero*:*R module M* $\Longrightarrow$ $mHom\ R\ M\ (Zm\ R\ e) =$
$\{mzeromap\ M\ (Zm\ R\ e)\}$
apply (*frule mzeromap-mHom* [*of M Zm R e*])
apply (*simp add*:*Zm-Module*)
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*frule-tac* $f = mzeromap\ M\ (Zm\ R\ e)$ **and** $g = x$ **in** *M-to-Z* [*of M*],
assumption+)
apply *simp*
apply (*rule subsetI*)
apply *simp*
done

lemma (**in** *Ring*) *carrier-HOM-M-Z*:*R module M* $\Longrightarrow$
$carrier\ (HOM_R\ M\ (Zm\ R\ e)) = \{mzeromap\ M\ (Zm\ R\ e)\}$
apply (*subst HOM-carrier*)
apply (*simp add*:*mHom-to-zero*)
done

lemma (**in** *Ring*) *HOM-M-Z-0*:*R module M* $\Longrightarrow HOM_R\ M\ (Zm\ R\ e) \cong_R (Zm\ R\ e)$
apply (*cut-tac Zm-Module*[*of e*],
frule-tac $M = M$ **and** $N = Zm\ R\ e$ **in** *Module.HOM-is-module*,
assumption+)
apply (*frule-tac* $M = M$ **and** $e = e$ **in** *carrier-HOM-M-Z*)
apply (*simp add*:*Module.zero-HOM*)
apply (*rule Module.Modules-single-carrier-isom*, *assumption*+)
apply (*simp add*:*Zm-def*)
done

lemma (**in** *Ring*) *M-to-Z-0*:$[\![R$ *module* $M; f \in$ *mHom* $R\ M\ (Zm\ R\ e)]\!] \Longrightarrow$
$ker_{M,(Zm\ R\ e)}\ f = carrier\ M$
apply (*simp add*:*ker-def*)
apply (*simp add*:*Zm-def*) **apply** (*fold Zm-def*)
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*simp add*:*CollectI*)
apply (*rule subsetI*, *simp*)
apply (*cut-tac Zm-Module*[*of e*])
apply (*frule-tac* $R = R$ **and** $M = M$ **and** $N = Zm\ R\ e$ **and** $f = f$ **and** $m = x$
in
Module.mHom-mem, *assumption*+)
apply (*simp add*:*Zm-carrier*)
done

definition
exact3 :: [('*r*, '*m*) *Ring-scheme*, ('*a*, '*r*, '*m1*) *Module-scheme*, '*a* ⇒ '*b*,
('*b*, '*r*, '*m1*) *Module-scheme*, '*b* ⇒ '*c*, ('*c*, '*r*, '*m1*) *Module-scheme*] ⇒ *bool*
where
exact3 R L0 h0 L1 h1 L2 == *h0* ‘ (*carrier L0*) = $ker_{(L1),(L2)}\ h1$

definition
exact4 :: [('*r*, '*m*) *Ring-scheme*, ('*a0*, '*r*, '*m1*) *Module-scheme*, '*a0* ⇒ '*a1*,
('*a1*, '*r*, '*m1*) *Module-scheme*, '*a1* ⇒ '*a2*, ('*a2*, '*r*, '*m1*) *Module-scheme*,
'*a2* ⇒ '*a3*, ('*a3*, '*r*, '*m1*) *Module-scheme*] ⇒ *bool* **where**
exact4 R L0 h0 L1 h1 L2 h2 L3 ⟷ *h0* ‘ (*carrier L0*) = $ker_{(L1),(L2)}\ h1$ ∧
h1 ‘ (*carrier L1*) = $ker_{(L2),(L3)}\ h2$

definition
exact5 :: [('*r*, '*m*) *Ring-scheme*, ('*a0*, '*r*, '*m1*) *Module-scheme*, '*a0* ⇒ '*a1*,
('*a1*, '*r*, '*m1*) *Module-scheme*, '*a1* ⇒ '*a2*, ('*a2*, '*r*, '*m1*) *Module-scheme*,
'*a2* ⇒ '*a3*, ('*a3*, '*r*, '*m1*) *Module-scheme*, '*a3* ⇒ '*a4*,
('*a4*, '*r*, '*m1*) *Module-scheme*] ⇒ *bool* **where**
exact5 R L0 h0 L1 h1 L2 h2 L3 h3 L4 == *h0* ‘ (*carrier L0*) = $ker_{(L1),(L2)}\ h1$
∧
h1 ‘ (*carrier L1*) = $ker_{(L2),(L3)}\ h2$ ∧ *h2* ‘(*carrier L2*) = $ker_{(L3),(L4)}\ h3$

definition
exact8 :: [('*r*, '*m*) *Ring-scheme*, ('*a0*, '*r*, '*m1*) *Module-scheme*, '*a0* ⇒ '*a1*,
('*a1*, '*r*, '*m1*) *Module-scheme*, '*a1* ⇒ '*a2*, ('*a2*, '*r*, '*m1*) *Module-scheme*,
'*a2* ⇒ '*a3*, ('*a3*, '*r*, '*m1*) *Module-scheme*, '*a3* ⇒ '*a4*,
('*a4*, '*r*, '*m1*) *Module-scheme*, '*a4* ⇒ '*a5*, ('*a5*, '*r*, '*m1*) *Module-scheme*,
'*a5* ⇒ '*a6*, ('*a6*, '*r*, '*m1*) *Module-scheme*, '*a6* ⇒ '*a7*,
('*a7*, '*r*, '*m1*) *Module-scheme*] ⇒ *bool* **where**
exact8 R L0 h0 L1 h1 L2 h2 L3 h3 L4 h4 L5 h5 L6 h6 L7 ⟷
h0 ‘ (*carrier L0*) = $ker_{(L1),(L2)}\ h1$ ∧ *h1* ‘ (*carrier L1*) = $ker_{(L2),(L3)}\ h2$ ∧
h2 ‘ (*carrier L2*) = $ker_{(L3),(L4)}\ h3$ ∧ *h3* ‘ (*carrier L3*) = $ker_{(L4),(L5)}\ h4$ ∧
h4 ‘ (*carrier L4*) = $ker_{(L5),(L6)}\ h5$ ∧ *h5* ‘ (*carrier L5*) = $ker_{(L6),(L7)}\ h6$

lemma (**in** *Ring*) *exact3-comp-0*:⟦*R module L*; *R module M*; *R module N*;
f ∈ *mHom R L M*; *g* ∈ *mHom R M N*; *exact3 R L f M g N*⟧ ⟹
compos L g f = *mzeromap L N*
apply (*frule Module.mHom-compos* [*of M R L N f g*], *assumption*+,
frule mzeromap-mHom [*of L N*], *assumption*,
rule Module.mHom-eq [*of L R N*], *assumption*+)
apply (*rule ballI*)
apply (*subst compos-def*)+
apply (*simp add*:*exact3-def*)
apply (*cut-tac mHom-func*[*of f L M*])
apply (*frule-tac a* = *m* **in** *mem-in-image* [*of f carrier L carrier M*],
assumption+)
apply *simp*
apply (*simp add*:*ker-def mzeromap-def compose-def*, *assumption*)
done

lemma (**in** *Ring*) *exact-im-sub-kern*:⟦*R module L*; *R module M*; *R module N*;
f ∈ *mHom R L M*; *g* ∈ *mHom R M N*; *exact3 R L f M g N*⟧ ⟹
f ‘ (*carrier L*) ⊆ $ker_{M,N}$ *g*
apply (*simp add*:*exact3-def*)
done

lemma (**in** *Ring*) *mzero-im-sub-ker*:⟦*R module L*; *R module M*; *R module N*;
f ∈ *mHom R L M*; *g* ∈ *mHom R M N*; *compos L g f* = *mzeromap L N*⟧ ⟹

f ‘ (*carrier L*) ⊆ $ker_{M,N}$ *g*
apply (*rule subsetI*)
apply (*simp add*:*image-def*)
apply *auto*
apply (*simp add*:*ker-def*)
apply (*simp add*:*Module.mHom-mem*)
apply (*simp add*:*compos-def compose-def*)
apply (*subgoal-tac* (λ*x*∈*carrier L*. *g* (*f x*)) *xa* = *mzeromap L N xa*)
prefer *2* **apply** *simp*
apply (*thin-tac* (λ*x*∈*carrier L*. *g* (*f x*)) = *mzeromap L N*)
apply *simp*
apply (*simp add*:*mzeromap-def*)
done

lemma (**in** *Ring*) *left-exact-injec*:⟦*R module M*; *R module N*;
z ∈ *mHom R* (*Zm R e*) *M*; *f* ∈ *mHom R M N*; *exact3 R* (*Zm R e*) *z M f N*⟧
⟹
$injec_{M,N}$ *f*
apply (*simp add*:*injec-def*)
apply (*rule conjI*)
apply (*simp add*:*mHom-def*)
apply (*simp add*:*exact3-def*)
apply (*simp add*:*Zm-def*, *fold Zm-def*)
apply (*simp add*: *Zm-to-M-0* [*of M z e*])

done

lemma (**in** *Ring*) *injec-left-exact*:⟦*R module M*; *R module N*;
z ∈ *mHom R* (*Zm R e*) *M*; *f* ∈ *mHom R M N*; $injec_{M,N}\ f$⟧ ⟹
exact3 R (*Zm R e*) *z M f N*
apply (*simp add*:*exact3-def*)
apply (*simp add*:*Zm-def*, *fold Zm-def*)
apply (*simp add*:*Zm-to-M-0* [*of M z e*])
apply (*simp add*:*injec-def*)
done

lemma (**in** *Ring*) *injec-mHom-image*:⟦*R module N*; *R module M1*; *R module M2*;

x ∈ *mHom R N M2*; *f* ∈ *mHom R M1 M2*; *x* ‘ (*carrier N*) ⊆ *f* ‘ (*carrier M1*);
$injec_{M1,M2}\ f$⟧⟹
(λ*n* ∈(*carrier N*). (*SOME m*. (*m* ∈ *carrier M1* ∧ *x n* = *f m*))) ∈ *mHom R N M1* ∧
compos N f (λ*n* ∈ (*carrier N*). (*SOME m*. *m* ∈ *carrier M1* ∧ *x n* = *f m*)) = *x*
apply (*subgoal-tac* (λ*n*∈*carrier N*. *SOME m*. *m* ∈ *carrier M1* ∧ *x n* = *f m*) ∈
mHom R N M1, *simp*)
apply (*rule Module.mHom-eq*, *assumption*+)
apply (*simp add*:*Module.mHom-compos*, *assumption*)
apply (*rule ballI*)
apply (*simp add*:*compos-def compose-def*)
apply (*thin-tac* (λ*n*∈*carrier N*. *SOME m*. *m* ∈ *carrier M1* ∧ *x n* = *f m*) ∈
mHom R N M1)
apply (*frule-tac m* = *m* **in** *Module.mHom-mem* [*of N R M2 x*], *assumption*+)
apply (*cut-tac mHom-func*[*of x N M2*],
frule-tac a = *m* **in** *mem-in-image*[*of x carrier N carrier M2*],
assumption+,
frule-tac c = *x m* **in** *subsetD* [*of x* ‘ *carrier N f* ‘ *carrier M1*],
assumption+)
apply (*simp add*:*image-def*)
apply (*rule someI2-ex*, *blast*)
apply (*thin-tac* ∃ *xa*∈*carrier M1*. *x m* = *f xa*, *erule conjE*)
apply (*rotate-tac* −*1*, *rule sym*, *assumption*+)

apply (*simp add*:*mHom-def*[*of R N M1*] *aHom-def*)
apply (*rule conjI*)
apply (*rule Pi-I*)
apply (*cut-tac mHom-func*[*of x N M2*],
frule-tac a = *xa* **in** *mem-in-image*[*of x carrier N carrier M2*],
assumption+,
frule-tac c = *x xa* **in** *subsetD* [*of x* ‘ *carrier N f* ‘ *carrier M1*],
assumption+)
apply (*simp add*:*image-def*)
apply (*rule someI2-ex*, *blast*, *simp*, *assumption*)

apply (*frule-tac* $R = R$ **and** $M = N$ **in** *Module.module-is-ag*,
simp add:*aGroup.ag-pOp-closed*,
simp add:*Module.sc-mem*)
apply (*rule conjI*)
apply (*rule ballI*)+
apply (*frule-tac* $x = a$ **and** $y = b$ **in** *aGroup.ag-pOp-closed* [*of N*],
assumption+)
apply (*cut-tac mHom-func*[*of x N M2*],
frule-tac $a = a \pm_N b$ **in** *mem-in-image*[*of x carrier N carrier M2*],
assumption+,
frule-tac $c = x$ $(a \pm_N b)$ **in** *subsetD*[*of x ‘ carrier N
f ‘ carrier M1*], *assumption*+)
apply (*frule-tac* $a = a$ **in** *mem-in-image*[*of x carrier N carrier M2*],
assumption+,
frule-tac $c = x$ a **in** *subsetD*[*of x ‘ carrier N
f ‘ carrier M1*], *assumption*+,
frule-tac $a = b$ **in** *mem-in-image*[*of x carrier N carrier M2*],
assumption+,
frule-tac $c = x$ b **in** *subsetD*[*of x ‘ carrier N
f ‘ carrier M1*], *assumption*+)
apply (*simp add*:*image-def*)
apply (*rule someI2-ex*, *blast*)
apply (*rule someI2-ex*, *blast*)
apply (*rule someI2-ex*, *blast*)
apply (*thin-tac* $\exists xa \in carrier\ N.\ x\ (\ a \pm_N b) = x\ xa$,
thin-tac $\exists xa \in carrier\ M1.\ x\ (\ a \pm_N b) = f\ xa$,
thin-tac $\exists xa \in carrier\ N.\ x\ a = x\ xa$,
thin-tac $\exists xa \in carrier\ M1.\ x\ a = f\ xa$,
thin-tac $\exists xa \in carrier\ N.\ x\ b = x\ xa$,
thin-tac $\exists xa \in carrier\ M1.\ x\ b = f\ xa$)
apply ((*erule conjE*)+, *fold image-def*)
apply (*frule-tac* $R = R$ **and** $M = N$ **and** $N = M2$ **and** $f = x$ **and** $m = a$ **and** $n = b$ **in**
Module.mHom-add, *assumption*+, *simp*)
apply (*simp add*:*Module.mHom-add*[*THEN sym*, *of - - - f*])
apply (*frule-tac* $R = R$ **and** $M = M1$ **and** $N = M2$ **and** $f = f$ **in** *Module.minjec-inj*,
assumption+)
apply (*frule-tac* $R = R$ **and** $M = M1$ **in** *Module.module-is-ag*,
frule-tac $x = xaa$ **and** $y = xa$ **in** *aGroup.ag-pOp-closed*[*of M1*],
assumption+)
apply (*simp add*:*inj-on-def*, *assumption*)

apply (*rule ballI*)+
apply (*frule-tac* $a = a$ **and** $m = m$ **in** *Module.sc-mem* [*of N R*], *assumption*+)
apply (*cut-tac mHom-func*[*of x N M2*],
frule-tac $a = a \cdot_{s N} m$ **in** *mem-in-image*[*of x carrier N carrier M2*],
assumption+,

frule-tac $c = x\ (a \cdot_{s N} m)$ **in** *subsetD*[*of* x ‘ *carrier* N
f ‘ *carrier M1*], *assumption*+)
apply (*frule-tac* $a = m$ **in** *mem-in-image*[*of* x *carrier* N *carrier M2*],
assumption+,
frule-tac $c = x\ m$ **in** *subsetD*[*of* x ‘ *carrier* N
f ‘ *carrier M1*], *assumption*+)
apply (*thin-tac* $x\ (a \cdot_{s N} m) \in x$ ‘ *carrier* N,
thin-tac $x\ m \in x$ ‘ *carrier* N)
apply (*simp add*:*image-def*)
apply (*rule someI2-ex*, *blast*)
apply (*rule someI2-ex*, *blast*)
apply (*thin-tac* $\exists\, xa{\in}$*carrier M1*. $x\ m = f\ xa$,
thin-tac $\exists\, xa{\in}$*carrier M1*. $x\ (a \cdot_{s N} m) = f\ xa$)
apply (*erule conjE*)+
apply (*simp add*:*Module.mHom-lin*)
apply (*simp add*:*Module.mHom-lin*[*THEN sym*])
apply (*rule sym*)
apply (*frule-tac* $R = R$ **and** $M = M1$ **and** $N = M2$ **and** $f = f$ **in** *Module.minjec-inj*,
assumption+,
frule-tac $R = R$ **and** $M = M1$ **and** $a = a$ **and** $m = xa$ **in** *Module.sc-mem*,
assumption+)
apply (*simp add*:*inj-on-def*, *assumption*)
done

lemma (**in** *Ring*) *right-exact-surjec*:⟦R *module* M; R *module* N; $f \in$ *mHom* $R\ M\ N$;
$p \in$ *mHom* $R\ N\ (Zm\ R\ e)$; *exact3* $R\ M\ f\ N\ p\ (Zm\ R\ e)$⟧ $\Longrightarrow$ $surjec_{M,N}\ f$
apply (*simp add*:*surjec-def*)
apply (*rule conjI*)
apply (*simp add*:*mHom-def*)
apply (*simp add*:*surj-to-def*)
apply (*simp add*:*exact3-def*)
apply (*simp add*:*M-to-Z-0*)
done

lemma (**in** *Ring*) *surjec-right-exact*:⟦R *module* M; R *module* N; $f \in$ *mHom* $R\ M\ N$;
$p \in$ *mHom* $R\ N\ (Zm\ R\ e)$; $surjec_{M,N}\ f$⟧ $\Longrightarrow$ *exact3* $R\ M\ f\ N\ p\ (Zm\ R\ e)$
apply (*simp add*:*exact3-def*)
apply (*simp add*:*ker-def*)
apply (*frule-tac* $f = p$ **and** $M = N$ **and** $N = Zm\ R\ e$ **in** *mHom-func*,
simp add:*Zm-carrier*)
apply (*simp add*:*surjec-def surj-to-def*,
thin-tac $f \in aHom\ M\ N \wedge f$ ‘ *carrier* $M = $ *carrier* N)
apply (*subst Zm-def*, *simp*)
apply (*rule equalityI*, *rule subsetI*, *simp*,
frule funcset-mem[*of* p *carrier* N $\{e\}$], *assumption*, *simp*)
apply (*rule subsetI*, *simp*)

done

lemma (**in** *Ring*) *exact4-exact3*:⟦*R module M*; *R module N*; *z* ∈ *mHom R* (*Zm R e*) *M*;
f ∈ *mHom R M N*; *z1* ∈ *mHom R N* (*Zm R e*);
exact4 R (*Zm R e*) *z M f N z1* (*Zm R e*) ⟧ ⟹
exact3 R (*Zm R e*) *z M f N* ∧ *exact3 R M f N z1* (*Zm R e*)
apply (*simp add*:*exact4-def exact3-def*)
done

lemma (**in** *Ring*) *exact4-bijec*:⟦*R module M*; *R module N*; *z* ∈ *mHom R* (*Zm R e*) *M*;
f ∈ *mHom R M N*; *z1* ∈ *mHom R N* (*Zm R e*);
exact4 R (*Zm R e*) *z M f N z1* (*Zm R e*)⟧ ⟹ $bijec_{M,N}\ f$
apply (*frule exact4-exact3* [*of M N z e f z1*], *assumption*+)
apply (*erule conjE*)
apply (*simp add*:*bijec-def*)
apply (*simp add*:*left-exact-injec*)
apply (*simp add*:*right-exact-surjec*)
done

lemma (**in** *Ring*) *exact-im-sub-ker*:⟦*R module L*; *R module M*; *R module N*;
f ∈ *mHom R L M*; *g* ∈ *mHom R M N*; *z1* ∈ *mHom R N* (*Zm R e*); *R module Z*;
exact4 R L f M g N z1 (*Zm R e*); *x* ∈ *mHom R M Z*; *compos L x f* = *mzeromap L Z*⟧
⟹ (λ*z*∈(*carrier N*). *x* (*SOME y*. *y* ∈ *carrier M* ∧ *g y* = *z*)) ∈ *mHom R N Z*
apply (*subst mHom-def*, *simp*)
apply (*rule conjI*)
apply (*simp add*:*aHom-def*)
apply (*rule conjI*)
apply (*rule Pi-I*)
apply *simp*
apply (*subgoal-tac exact3 R M g N z1* (*Zm R e*))
prefer *2* **apply** (*simp add*:*exact4-def exact3-def*)
apply (*frule right-exact-surjec* [*of M N g z1*], *assumption*+)
apply (*simp add*:*surjec-def*, *frule conjunct2*)
apply (*simp add*:*surj-to-def*, *erule conjE*)
apply (*rotate-tac* −*1*, *frule sym*, *thin-tac g* ‘ *carrier M* = *carrier N*,
simp, *thin-tac carrier N* = *g* ‘ *carrier M*)
apply (*simp add*:*image-def*)
apply (*rule someI2-ex*, *blast*)
apply (*erule conjE*, *simp add*:*Module.mHom-mem*)

apply (*frule-tac R* = *R* **and** *M* = *N* **in** *Module.module-is-ag*,
simp add:*aGroup.ag-pOp-closed*)
apply (*rule ballI*)+
apply (*frule-tac x* = *a* **and** *y* = *b* **in** *aGroup.ag-pOp-closed* [*of N*],
assumption+)

apply (*subgoal-tac exact3 R M g N z1* (*Zm R e*))
prefer *2* **apply** (*simp add*:*exact4-def exact3-def*,
frule right-exact-surjec[*of M N g z1*], *assumption*+)
apply (*simp add*:*surjec-def surj-to-def*, *erule conjE*)
apply (*rotate-tac* −*1*, *frule sym*, *thin-tac g* ‘ *carrier M* = *carrier N*,
simp)
apply (*simp add*:*image-def*)
apply (*rule someI2-ex*, *blast*)
apply (*rule someI2-ex*, *blast*)
apply (*rule someI2-ex*)
apply (*thin-tac* ∃ *x*∈*carrier M*. *a* = *g x*, *thin-tac* ∃ *x*∈*carrier M*. *b* = *g x*,
thin-tac xa ∈ *carrier M* ∧ *g xa* = *b*,
thin-tac xaa ∈ *carrier M* ∧ *g xaa* = *a*)
apply (*erule bexE*, *rotate-tac* −*1*, *frule sym*, *thin-tac a* ±$_N$ *b* = *g xb*)
apply *blast*
apply (*thin-tac* ∃ *x*∈*carrier M*. *a* = *g x*, *thin-tac* ∃ *x*∈*carrier M*. *b* = *g x*,
thin-tac ∃ *x*∈*carrier M*. *a* ±$_N$ *b* = *g x*)
apply (*erule conjE*)+
apply (*rotate-tac* −*5*)
apply (*frule sym*, *thin-tac g xa* = *b*, *frule sym*, *thin-tac g xaa* = *a*,
frule sym, *thin-tac g xb* = *a* ±$_N$ *b*, *simp*)
apply (*simp add*:*Module.mHom-add*[*THEN sym*])
apply (*frule mzero-im-sub-ker* [*of L M Z f x*], *assumption*+)
apply (*simp add*:*exact4-def*, *fold image-def*,
thin-tac f ‘ *carrier L* = *ker*$_{M,N}$ *g* ∧ *g* ‘ *carrier M* = *ker*$_{N,Zm\ R\ e}$ *z1*)
apply (*frule-tac R* = *R* **and** *M* = *M* **in** *Module.module-is-ag*,
frule-tac x = *xaa* **and** *y* = *xa* **in** *aGroup.ag-pOp-closed*[*of M*], *assumption*+,
frule-tac R = *R* **and** *M* = *M* **and** *N* = *N* **and** *f* = *g* **and** *a* = *xaa* ±$_M$ *xa*
and *b* = *xb* **in** *Module.mHom-ker-eq*, *assumption*+)
apply (*frule-tac c* = *xaa* ±$_M$ *xa* ±$_M$ −$_{aM}$ *xb* **in** *subsetD*[*of ker*$_{M,N}$ *g*
ker$_{M,Z}$ *x*], *assumption*+)
apply (*frule-tac a* = *xaa* ±$_M$ *xa* **and** *b* = *xb* **in**
Module.mHom-eq-ker[*of M R Z x*], *assumption*+)
apply (*rule sym*, *assumption*)

apply (*simp add*:*Module.sc-mem*)
apply (*rule ballI*)+
apply (*frule right-exact-surjec*[*of M N g z1*], *assumption*+,
simp add:*exact4-def exact3-def*)
apply (*frule-tac a* = *a* **and** *m* = *m* **in** *Module.sc-mem*[*of N*], *assumption*+)
apply (*simp add*:*surjec-def surj-to-def*, *erule conjE*,
rotate-tac −*1*, *frule sym*, *thin-tac g* ‘ *carrier M* = *carrier N*,
simp, *thin-tac carrier N* = *g* ‘ *carrier M*)
apply (*simp add*:*image-def*)
apply (*rule someI2-ex*, *blast*,
thin-tac ∃ *x*∈*carrier M*. *m* = *g x*)
apply (*rule someI2-ex*)
apply (*erule bexE*, *rotate-tac* −*1*, *frule sym*, *thin-tac a* ·$_{sN}$ *m* = *g xaa*)
apply *blast*

apply (*thin-tac* $\exists x{\in}carrier\ M.\ a \cdot_{s\,N} m = g\ x$)
apply (*erule conjE*)+
apply (*rotate-tac* -3, *frule sym*, *thin-tac* $g\ xa = m$, *simp*)
apply (*simp add*:*Module.mHom-lin*[*THEN sym*])
apply (*frule-tac* $a = a$ **and** $m = xa$ **in** *Module.sc-mem*, *assumption*+)
apply (*frule-tac* $R = R$ **and** $M = M$ **and** $N = N$ **and** $f = g$ **and** $a = xaa$ **and**
$b = a \cdot_{s\,M} xa$ **in** *Module.mHom-ker-eq*, *assumption*+)

apply (*frule mzero-im-sub-ker* [*of L M Z f x*], *assumption*+,
simp add:*exact4-def*,
thin-tac f ' carrier L $= ker_{M,N}\ g \wedge g$ *' carrier M* $= ker_{N,Zm\ R\ e}$ *z1*)
apply (*frule-tac* $c = xaa \pm_M -_{a\,M} (a \cdot_{s\,M} xa)$ **in**
subsetD[*of* $ker_{M,N}\ g\ \ ker_{M,Z}\ x$], *assumption*+)
apply (*rule-tac* $a = xaa$ **and** $b = a \cdot_{s\,M} xa$ **in**
Module.mHom-eq-ker[*of M R Z x*], *assumption*+)
done

lemma (**in** *Ring*) *exact-im-sub-ker1*:⟦*R module L*; *R module M*; *R module N*;
$f \in mHom\ R\ L\ M$; $g \in mHom\ R\ M\ N$; *z1* $\in mHom\ R\ N\ (Zm\ R\ e)$; *R module Z*;
exact4 R L f M g N z1 (*Zm R e*); $x \in mHom\ R\ M\ Z$;
compos L x f $= mzeromap\ L\ Z$ ⟧ $\Longrightarrow$
compos M $(\lambda z{\in}(carrier\ N).\ x\ (SOME\ y.\ y \in carrier\ M \wedge g\ y = z))\ g = x$
apply (*frule exact-im-sub-ker* [*of L M N f g z1 e Z x*], *assumption*+)
apply (*frule-tac* $g = (\lambda z{\in}carrier\ N.\ x\ (SOME\ y.\ y \in carrier\ M \wedge g\ y = z))$ **in**
Module.mHom-compos [*of N R M Z g*], *assumption*+)
apply (*rule Module.mHom-eq* [*of M R Z - x*], *assumption*+)
apply (*rule ballI*)
apply (*subst compos-def*, *subst compose-def*, *simp*)
apply (*simp add*:*Module.mHom-mem*)
apply (*thin-tac compos M* $(\lambda z{\in}carrier\ N.\ x$
$(SOME\ y.\ y \in carrier\ M \wedge g\ y = z))\ g \in mHom\ R\ M\ Z$)
apply (*thin-tac* $(\lambda z{\in}carrier\ N.\ x\ (SOME\ y.\ y \in carrier\ M \wedge g\ y = z)) \in$
$mHom\ R\ N\ Z$)
apply (*frule right-exact-surjec* [*of M N g z1 e*], *assumption*+)
apply (*simp add*:*exact4-def exact3-def*)
apply (*frule mHom-func*[*of g M N*],
frule-tac $a = m$ **in** *mem-in-image*[*of g carrier M carrier N*],
assumption+)
apply (*simp add*:*image-def*)
apply (*rule someI2-ex*) **apply** *blast*
apply (*thin-tac* $\exists x{\in}carrier\ M.\ g\ m = g\ x$) **apply** (*erule conjE*)
apply (*frule mzero-im-sub-ker* [*of L M Z f x*], *assumption*+)
apply (*simp add*:*surjec-def surj-to-def exact4-def*)
apply (*frule-tac* $a = xa$ **and** $b = m$ **in** *Module.mHom-ker-eq*[*of M R N g*],
assumption+)
apply (*frule-tac* $c = xa \pm_M -_{a\,M} m$ **in** *subsetD*[*of* $ker_{M,N}\ g\ ker_{M,Z}\ x$],

assumption+)
apply (*rule-tac a = xa* **and** *b = m* **in** *Module.mHom-eq-ker*[*of M R Z x*],
assumption+)
done

definition
module-iota :: [('*r*, '*m*) *Ring-scheme*, ('*a*, '*r*, '*m1*) *Module-scheme*] ⇒
'*a* ⇒ '*a* ((*mι*_ -) [*92*, *93*]*92*) **where**
mι$_R$ *M* = (λ*x*∈*carrier M*. *x*)

lemma (**in** *Ring*) *short-exact-sequence*:⟦*R module M*; *submodule R M N*;
z ∈ *mHom R* (*Zm R e*) (*mdl M N*); *z1* ∈ *mHom R* (*M* /$_m$ *N*) (*Zm R e*)⟧ ⟹
exact5 R (*Zm R e*) *z* (*mdl M N*)(*mι*$_R$ (*mdl M N*)) *M* (*mpj M N*) (*M* /$_m$ *N*) *z1*
(*Zm R e*)
apply (*simp add*:*exact5-def*)
apply (*rule conjI*)
apply (*simp add*:*Zm-def*, *fold Zm-def*)
apply (*frule Module.mdl-is-module* [*of M R N*], *assumption+*)
apply (*simp add*:*ker-def*, *simp add*:*module-iota-def*)
apply (*simp add*:*Zm-to-M-0* [*of mdl M N z*])
apply (*simp add*:*mdl-def*, *fold mdl-def*)
apply (*rule equalityI*)
apply *simp*
apply (*simp add*:*Module.submodule-inc-0*)
apply (*rule subsetI*)
apply (*simp add*:*CollectI*)
apply (*rule conjI*)
apply (*simp add*:*module-iota-def*)
apply (*simp add*:*mdl-def*, *fold mdl-def*)
apply (*simp add*:*Module.mker-of-mpj*[*THEN sym*])
apply (*frule Module.qmodule-module* [*of M R N*], *assumption*)
apply (*subst M-to-Z-0* [*of M* /$_m$ *N z1 e*], *assumption+*)
apply (*frule Module.mpj-surjec* [*of M R N*], *assumption+*)
apply (*simp add*:*surjec-def surj-to-def*)
done

lemma (**in** *Ring*) *rexact4-lexact4-HOM*:⟦*R module M1*; *R module M2*; *R module M3*;
f ∈ *mHom R M1 M2*; *g* ∈ *mHom R M2 M3*; *z1* ∈ *mHom R M3* (*Zm R e*);
exact4 R M1 f M2 g M3 z1 (*Zm R e*)⟧ ⟹
∀ *N*. *R module N* ⟶
exact4 R (*HOM*$_R$ (*Zm R e*) *N*) (*sup-sharp R M3* (*Zm R e*) *N z1*) (*HOM*$_R$ *M3*
N)
(*sup-sharp R M2 M3 N g*) (*HOM*$_R$ *M2 N*) (*sup-sharp R M1 M2 N f*) (*HOM*$_R$
M1 N)

apply (*rule allI*) **apply** (*rule impI*)
apply (*subst exact4-def*)

apply (*rule conjI*)
apply (*cut-tac Zm-Module* [*of e*])
apply (*subst HOM-carrier* [*of Zm R e*])
apply (*simp add*:*mHom-Z-M*)
apply (*simp add*:*sup-sharp-def*)
apply (*simp add*:*mzeromap-mHom*)
apply (*simp add*:*ker-def*)
apply (*simp add*:*HOM-def*)
apply (*rule equalityI*)
apply (*rule subsetI*)
apply *simp*
apply (*frule-tac N = N* **in** *mzeromap-mHom* [*of Zm R e*], *assumption*+)
thm *Module.mHom-compos*[*of Zm R e R M3 - z1*]
apply (*frule-tac N = N* **and** *g = mzeromap* (*Zm R e*) *N* **in**
Module.mHom-compos[*of Zm R e R M3 - z1*], *assumption*+,
simp)
apply (*frule-tac N = N* **in** *mzeromap-mHom* [*of M2*], *assumption*+)
apply (*rule-tac N = N* **and** *f = compos M2* (*compos M3* (*mzeromap* (*Zm R e*) *N*) *z1*) *g* **and** *g = mzeromap M2 N* **in** *Module.mHom-eq* [*of M2 -*], *assumption*+)
apply (*rule Module.mHom-compos*, *assumption*+)
apply (*rule ballI*)
apply (*simp add*:*compos-def mzeromap-def compose-def*)
apply (*simp add*:*Module.mHom-mem*)+
apply (*rule subsetI*, *simp*, *erule conjE*, *simp*)
apply (*frule-tac N = N* **in** *mzeromap-mHom* [*of Zm R e*], *assumption*+)
apply (*frule-tac N = N* **and** *g = mzeromap* (*Zm R e*) *N* **in**
Module.mHom-compos [*of Zm R e R M3 - z1*], *assumption*+)
apply (*rule-tac N = N* **and** *f = x* **and**
g = compos M3 (*mzeromap* (*Zm R e*) *N*) *z1* **in** *Module.mHom-eq*[*of M3 -*],
assumption+)
apply (*rule ballI*)
apply (*subst compos-def*, *subst compose-def*)
apply (*subst mzeromap-def*) **apply** (*simp add*:*Module.mHom-mem*)
apply (*simp add*:*exact4-def*)
apply (*subgoal-tac exact3 R M2 g M3 z1* (*Zm R e*))
prefer *2* **apply** (*simp add*:*exact3-def*)
apply (*frule right-exact-surjec* [*of M2 M3 g z1*], *assumption*+)
apply (*simp add*:*surjec-def surj-to-def*, *erule conjE*)
apply (*thin-tac f* ‘ *carrier M1* = $ker_{M2,M3}$ *g*,
thin-tac g ∈ *aHom M2 M3* ∧ $ker_{M3,Zm\ R\ e}$ *z1* = *carrier M3*,
rotate-tac −*1*, *frule sym*, *thin-tac g* ‘ *carrier M2* = *carrier M3*,
simp add:*image-def*, *fold image-def*)
apply (*erule bexE*, *simp*) **apply** (*simp add*:*compos-def compose-def mzeromap-def*)
apply (*frule-tac f* = λ*xa*∈*carrier M2*. *x* (*g xa*) **and** *g* = λ*x*∈*carrier M2*. $\mathbf{0}_N$
and *x = xa* **in** *eq-fun-eq-val*,
thin-tac (λ*xa*∈*carrier M2*. *x* (*g xa*)) = (λ*x*∈*carrier M2*. $\mathbf{0}_N$),
simp)

apply (*rule equalityI*)

apply (*rule subsetI*)
apply (*simp add*:*image-def*)
apply (*simp add*:*ker-def*)
apply (*simp add*:*HOM-carrier*)
apply (*erule bexE*)
apply (*frule-tac L = N* **and** *f = xa* **in** *Module.sup-sharp-homTr*[*of M2 R M3 - g*],
assumption+, *simp*)
thm *Module.sup-sharp-homTr*[*of M1 R M2 - f*]
apply (*frule-tac L = N* **and** *f = sup-sharp R M2 M3 N g xa* **in**
Module.sup-sharp-homTr[*of M1 R M2 - f*], *assumption*+)
apply (*simp add*:*HOM-def*)
apply (*frule-tac N = N* **in** *mzeromap-mHom*[*of M1*], *assumption*)
apply (*rule Module.mHom-eq*, *assumption*+)
apply (*rule ballI*)
apply (*subst sup-sharp-def*)
apply *simp*
apply (*subst compos-def*, *subst compose-def*)
apply (*subst sup-sharp-def*, *simp*)
apply (*subst compos-def*, *subst compose-def*, *simp*)
apply (*simp add*:*Module.mHom-mem*)
apply (*subgoal-tac exact3 R M1 f M2 g M3*)
prefer *2* **apply** (*simp add*:*exact4-def exact3-def*)
apply (*frule-tac exact3-comp-0* [*of M1 M2 M3 f g*], *assumption*+)
apply (*frule-tac f = compos M1 g f* **and** *g = mzeromap M1 M3*
and *x = m* **in** *eq-fun-eq-val*,
thin-tac compos M1 g f = mzeromap M1 M3,
simp add:*compos-def compose-def mzeromap-def*)
apply (*simp add*:*Module.mHom-0*)
apply (*rule subsetI*)
apply (*simp add*:*ker-def*)
apply (*erule conjE*)
apply (*simp add*:*HOM-carrier*)
apply (*simp add*:*HOM-def*)
apply (*simp add*:*sup-sharp-def*)
apply (*simp add*:*image-def*)
apply (*frule-tac Z = N* **and** *x = x* **in** *exact-im-sub-ker1* [*of M1 M2 M3 f g z1 e*],
assumption+)
apply (*rotate-tac −1*) **apply** (*frule sym*)
apply (*thin-tac compos M2* (λ*z*∈*carrier M3*. *x*
(*SOME y*. *y* ∈ *carrier M2* ∧ *g y = z*)) *g = x*)
apply (*frule-tac Z = N* **and** *x = x* **in** *exact-im-sub-ker* [*of M1 M2 M3 f g z1 e*],
assumption+)
apply *blast*
done

lemma *exact-HOM-exactTr*:⟦*Ring* (*R*::('*r*, '*m1*) *Ring-scheme*); *f* ∈ *mHom R M1 M2*;

$g \in mHom\ R\ M2\ M3$; $z1 \in mHom\ R\ M3\ (Zm\ R\ e)$; $R\ module\ NV$;
$\forall (N::('a, 'r, 'm)\ Module\text{-}scheme).\ R\ module\ N \longrightarrow$
$exact4\ R\ (HOM_R\ (Zm\ R\ e)\ N)(sup\text{-}sharp\ R\ M3\ (Zm\ R\ e)\ N\ z1)$
$(HOM_R\ M3\ N)\ (sup\text{-}sharp\ R\ M2\ M3\ N\ g)\ (HOM_R\ M2\ N)\ (sup\text{-}sharp\ R\ M1\ M2\ N\ f)$
$(HOM_R\ M1\ N)$; $R\ module\ (L::('a, 'r, 'm)\ Module\text{-}scheme)]\!] \Longrightarrow$
$exact4\ R\ (HOM_R\ (Zm\ R\ e)\ L)\ (sup\text{-}sharp\ R\ M3\ (Zm\ R\ e)\ L\ z1)$
$(HOM_R\ M3\ L)\ (sup\text{-}sharp\ R\ M2\ M3\ L\ g)\ (HOM_R\ M2\ L)\ (sup\text{-}sharp\ R\ M1\ M2\ L\ f)$
$(HOM_R\ M1\ L)$
apply *simp*
done

lemma *lexact4-rexact4-HOM*:$[\![Ring\ R$; $R\ module\ M1$; $R\ module\ M2$; $R\ module\ M3$;
$f \in mHom\ R\ M1\ M2$; $g \in mHom\ R\ M2\ M3$; $z \in mHom\ R\ (Zm\ R\ e)\ M1$;
$exact4\ R\ (Zm\ R\ e)\ z\ M1\ f\ M2\ g\ M3\ ]\!] \Longrightarrow$
$\forall N.\ R\ module\ N \longrightarrow exact4\ R\ (HOM_R\ N\ (Zm\ R\ e))\ (sub\text{-}sharp\ R\ N\ (Zm\ R\ e)\ M1\ z)$
$(HOM_R\ N\ M1)\ (sub\text{-}sharp\ R\ N\ M1\ M2\ f)\ (HOM_R\ N\ M2)\ (sub\text{-}sharp\ R\ N\ M2\ M3\ g)$
$(HOM_R\ N\ M3)$

apply (*rule allI*) **apply** (*rule impI*)
apply (*subst exact4-def*)
apply (*rule conjI*)
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*simp add*:*image-def*)
apply (*simp add*:*HOM-def*) **apply** (*fold HOM-def*)
apply (*erule bexE*)
apply (*simp add*:*ker-def*) **apply** (*simp add*:*HOM-def*)
apply (*simp add*:*sub-sharp-def*)
apply (*cut-tac Ring.Zm-Module* [*of R e*])
apply (*simp add*:*Module.mHom-compos*)
apply (*frule-tac L = N* **and** *f = xa* **in**
Module.mHom-compos[*of Zm R e R - M1 - z*], *assumption*+)
apply (*frule-tac L = N* **and** *f = compos N z xa* **in**
Module.mHom-compos[*of M1 R - M2 - f*], *assumption*+)
apply (*frule-tac M = N* **in** *Ring.mzeromap-mHom* [*of R - M2*], *assumption*+)
apply (*rule Module.mHom-eq*, *assumption*+)
apply (*rule ballI*)
apply (*simp add*:*mzeromap-def*) **apply** (*simp add*:*compos-def compose-def*)
apply (*frule-tac M = N* **and** *f = xa* **and** *m = m* **in** *Module.mHom-mem* [*of - R*
Zm R e], *assumption*+)

apply (*simp add*:*exact4-def*)
apply (*frule conjunct1*)
apply (*thin-tac z ‘ carrier* (*Zm R e*) = $ker_{M1,M2}$ *f*)
apply (*erule conjE*, *thin-tac f ‘ carrier M1* = $ker_{M2,M3}$ *g*)
apply (*frule Ring.mHom-func*[*of R z* (*Zm R e*) *M1*], *assumption*)
apply (*frule-tac a* = *xa m* **in** *mem-in-image*
[*of z carrier* (*Zm R e*) *carrier M1*], *assumption*+, *simp*)
apply (*simp add*:*ker-def*, *assumption*)

apply (*rule subsetI*)
apply (*simp add*:*ker-def*) **apply** (*simp add*:*HOM-def*)
apply (*erule conjE*)
apply (*frule-tac M* = *N* **in** *Ring.mHom-to-zero* [*of R - e*], *assumption*+)
apply *simp* **apply** (*simp add*:*sub-sharp-def*)
apply (*simp add*:*exact4-def*) **apply** (*frule conjunct1*)
apply (*thin-tac z ‘ carrier* (*Zm R e*) = $ker_{M1,M2}$ *f* ∧
f ‘ carrier M1 = $ker_{M2,M3}$ *g*)
apply (*simp add*:*Zm-def*, *fold Zm-def*)
apply (*frule-tac L* = *N* **and** *f* = *x* **in** *Ring.mzero-im-sub-ker* [*of R - M1 M2 - f*], *assumption*+) **apply** (*rotate-tac* −2) **apply** (*frule sym*)
apply (*thin-tac* {*z e*} = $ker_{M1,M2}$ *f*, *simp*)
apply (*subgoal-tac mzeromap N* (*Zm R e*) ∈ *mHom R N* (*Zm R e*))
prefer *2* **apply** *simp*
apply (*frule Ring.Zm-Module*[*of R e*])
apply (*frule-tac L* = *N* **and** *f* = *mzeromap N* (*Zm R e*) **in**
Module.mHom-compos [*of Zm R e R - M1 - z*], *assumption*+)
apply (*rule Module.mHom-eq*, *assumption*+) **apply** (*rule ballI*)
apply (*simp add*:*compos-def compose-def mzeromap-def*)
apply (*simp add*:*Zm-def*, *fold Zm-def*)
apply (*frule-tac M* = *N* **and** *f* = *x* **in** *Ring.mHom-func*[*of R - - M1*], *assumption*+)
apply (*frule-tac f* = *x* **and** *A* = *carrier N* **and** *B* = *carrier M1* **and** *a* = *m* **in**
mem-in-image, *assumption*+)
apply (*frule-tac c* = *x m* **and** *A* = *x ‘ carrier N* **and** *B* = {*z e*} **in**
subsetD, *assumption*+) **apply** *simp*

apply (*simp add*:*image-def ker-def HOM-def*)
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*simp*, *erule bexE*)
apply (*simp add*:*sub-sharp-def*)
apply (*frule-tac L* = *N* **and** *f* = *xa* **in** *Module.mHom-compos* [*of M1 R - M2 - f*],
assumption+) **apply** *simp*
apply (*frule-tac L* = *N* **and** *f* = *compos N f xa* **in** *Module.mHom-compos* [*of M2 R*
- M3 - g], *assumption*+)
apply (*rule Module.mHom-eq*, *assumption*+)
apply (*simp add*:*Ring.mzeromap-mHom*)

apply (*rule ballI*)
apply (*simp add*:*compos-def compose-def mzeromap-def*)
apply (*thin-tac* $x = (\lambda x{\in}carrier\ N.\ f\ (xa\ x))$)
apply (*thin-tac* $(\lambda x{\in}carrier\ N.\ f\ (xa\ x)) \in mHom\ R\ N\ M2$)
apply (*thin-tac* $(\lambda x{\in}carrier\ N.\ g\ (if\ x \in carrier\ N\ then\ f\ (xa\ x)\ else\ undefined)) \in mHom\ R\ N\ M3$)
apply (*frule-tac* $M = N$ **and** $f = xa$ **and** $m = m$ **in** *Module.mHom-mem* [*of - R M1*],
assumption+)
apply (*simp add*:*exact4-def*) **apply** (*frule conjunct2*)
apply (*thin-tac* z ‘ *carrier* $(Zm\ R\ e) = ker_{M1,M2}\ f \wedge$
f ‘ *carrier M1* $= ker_{M2,M3}\ g$)
apply (*frule Ring.mHom-func*[*of R f M1 M2*], *assumption*+,
frule-tac $f = f$ **and** $A =$ *carrier M1* **and** $B =$ *carrier M2* **and**
$a = xa\ m$ **in** *mem-in-image*, *assumption*+, *simp*)
apply (*simp add*:*ker-def*)

apply (*rule subsetI*)
apply *simp* **apply** (*erule conjE*)
apply (*simp add*:*sub-sharp-def*)
apply (*frule-tac* $L = N$ **and** $f = x$ **in** *Ring.mzero-im-sub-ker* [*of R - M2 M3 - g*],
assumption+)
apply (*simp add*:*exact4-def*)
apply (*frule conjunct2*) **apply** (*rotate-tac* -1) **apply** (*frule sym*)
apply (*thin-tac* f ‘ *carrier M1* $= ker_{M2,M3}\ g$)
apply *simp* **apply** (*thin-tac* $ker_{M2,M3}\ g = f$ ‘ *carrier M1*)
apply (*frule Ring.left-exact-injec*[*of R M1 M2 z e f*], *assumption*+)
apply (*simp add*:*exact3-def exact4-def*)
apply (*frule-tac* $N = N$ **and** $x = x$ **in** *Ring.injec-mHom-image*[*of R - M1 M2 - f*],
assumption+)
apply (*erule conjE*) **apply** (*rotate-tac* -1) **apply** (*frule sym*)
apply (*thin-tac compos N f* $(\lambda n{\in}carrier\ N.\ SOME\ m.\ m \in carrier\ M1 \wedge$
$x\ n = f\ m) = x$)
apply *blast*
done

5.9 Tensor product

definition
prod-carr :: [(*'a*, *'r*, *'m*) *Module-scheme*, (*'b*, *'r*, *'m*) *Module-scheme*]
⇒ (*'a* ∗ *'b*) *set* (**infixl** $\times_c$ *100*) **where**
$M \times_c N = carrier\ M \times carrier\ N$

definition
bilinear-map :: [*'a* ∗ *'b* ⇒ *'c*, (*'r*, *'m*) *Ring-scheme*,
(*'a*, *'r*, *'m1*) *Module-scheme*, (*'b*, *'r*, *'m1*) *Module-scheme*,
(*'c*, *'r*, *'m1*) *Module-scheme*] ⇒ *bool* **where**

$$\begin{aligned}
&\mathit{bilinear\text{-}map}\ f\ R\ M1\ M2\ N \longleftrightarrow f \in M1 \times_c M2 \to \mathit{carrier}\ N \wedge \\
&\qquad f \in \mathit{extensional}\ (M1 \times_c M2) \wedge \\
&(\forall\, x1 \in \mathit{carrier}\ M1.\ \forall\, x2 \in \mathit{carrier}\ M1. \\
&\qquad \forall\, y{\in}\mathit{carrier}\ M2.(f\ (x1 \pm_{M1} x2,\ y) = f\ (x1,\ y) \pm_N (f\ (x2,\ y)))) \wedge \\
&(\forall\, x{\in}\mathit{carrier}\ M1.\ \forall\, y1{\in}\mathit{carrier}\ M2. \\
&\qquad \forall\, y2{\in}\mathit{carrier}\ M2.\ f\ (x,\ y1 \pm_{M2} y2) = f\ (x,\ y1) \pm_N (f\ (x,\ y2))) \wedge \\
&(\forall\, x{\in}\mathit{carrier}\ M1.\ \forall\, y{\in}\mathit{carrier}\ M2. \\
&\qquad \forall\, r{\in}\mathit{carrier}\ R.\ f\ (r \cdot_{sM1} x,\ y) = r \cdot_{sN} (f\ (x,\ y)) \wedge \\
&\qquad\qquad f\ (x,\ r \cdot_{sM2} y) = r \cdot_{sN} (f\ (x,\ y)))
\end{aligned}$$

lemma (**in** *Ring*) *prod-carr-mem*:⟦*R module M*; *R module N*; $m \in \mathit{carrier}\ M$; $n \in \mathit{carrier}\ N$⟧ ⟹ $(m,\ n) \in M \times_c N$
by (*simp add*:*prod-carr-def*)

lemma (**in** *Ring*) *bilinear-func*:*bilinear-map f R M N Z* ⟹ $f \in M \times_c N \to \mathit{carrier}\ Z$
by (*simp add*:*bilinear-map-def*)

lemma (**in** *Ring*) *bilinear-mem*:⟦*R module M1*; *R module M2*; *R module N*; $m1 \in \mathit{carrier}\ M1$; $m2 \in \mathit{carrier}\ M2$; *bilinear-map f R M1 M2 N*⟧ ⟹ $f\ (m1,\ m2) \in \mathit{carrier}\ N$
apply (*simp add*:*bilinear-map-def*) **apply** (*erule conjE*)+
apply (*rule funcset-mem* [*of f M1* $\times_c$ *M2 carrier N*], *assumption*+)
apply (*simp add*:*prod-carr-def*)
done

lemma (**in** *Ring*) *bilinear-l-add*:⟦*R module M1*; *R module M2*; *R module N*; $m11 \in \mathit{carrier}\ M1$; $m12 \in \mathit{carrier}\ M1$; $m2 \in \mathit{carrier}\ M2$; *bilinear-map f R M1 M2 N*⟧ ⟹ $f\ (m11 \pm_{M1} m12,\ m2) = f\ (m11,\ m2) \pm_N (f\ (m12,\ m2))$
apply (*simp add*:*bilinear-map-def*)
done

lemma (**in** *Ring*) *bilinear-l-add1*:⟦*R module M1*; *R module M2*; *R module N*; $m11 \in \mathit{carrier}\ M1$; $m12 \in \mathit{carrier}\ M1$; $m2 \in \mathit{carrier}\ M2$; *bilinear-map f R M1 M2 N*⟧ ⟹ $f\ (m11 \pm_{M1} m12,\ m2) \pm_N -_{aN} (f\ (m11,\ m2) \pm_N (f\ (m12,\ m2))) = \mathbf{0}_N$
apply (*frule Module.module-is-ag*[*of N*],
frule Module.module-is-ag[*of M1*],
subst aGroup.ag-eq-diffzero[*of N*, *THEN sym*], *assumption*+,
frule-tac x = *m11* **and** *y* = *m12* **in** *aGroup.ag-pOp-closed*, *assumption*+)
apply (*simp add*:*bilinear-mem*,
rule aGroup.ag-pOp-closed, *assumption*+)
apply ((*simp add*:*bilinear-mem*)+, *simp add*:*bilinear-l-add*)
done

lemma (**in** *Ring*) *bilinear-r-add*:⟦*R module M1*; *R module M2*; *R module N*; $m \in \mathit{carrier}\ M1$; $m21 \in \mathit{carrier}\ M2$; $m22 \in \mathit{carrier}\ M2$; *bilinear-map f R M1 M2 N*⟧ ⟹

$f\ (m,\ m21\ \pm_{M2}\ m22) = f\ (m,\ m21)\ \pm_N\ (f\ (m,\ m22))$
apply (*simp add*:*bilinear-map-def*)
done

lemma (**in** *Ring*) *bilinear-r-add1*:⟦*R module M1*; *R module M2*; *R module N*;
m ∈ *carrier M1*; *m21* ∈ *carrier M2*; *m22* ∈ *carrier M2*;
bilinear-map f R M1 M2 N⟧ ⟹
$f\ (m,\ m21\ \pm_{M2}\ m22)\ \pm_N\ -_{a\,N}\ (f\ (m,\ m21)\ \pm_N\ (f\ (m,\ m22))) = \mathbf{0}_N$
apply (*frule Module.module-is-ag*[*of N*],
frule Module.module-is-ag[*of M2*],
subst aGroup.ag-eq-diffzero[*of N*, *THEN sym*], *assumption*+,
frule-tac x = *m21* **and** *y* = *m22* **in** *aGroup.ag-pOp-closed*, *assumption*+)
apply (*simp add*:*bilinear-mem*,
rule aGroup.ag-pOp-closed, *assumption*+)
apply ((*simp add*:*bilinear-mem*)+, *simp add*:*bilinear-r-add*)
done

lemma (**in** *Ring*) *bilinear-l-lin*:⟦*R module M1*; *R module M2*; *R module N*;
m1 ∈ *carrier M1*; *m2* ∈ *carrier M2*; *r* ∈ *carrier R*;
bilinear-map f R M1 M2 N⟧ ⟹ $f\ (r\ \cdot_{s\,M1}\ m1,\ m2) = r\ \cdot_{s\,N}\ (f\ (m1,\ m2))$
by (*simp add*:*bilinear-map-def*)

lemma (**in** *Ring*) *bilinear-l-lin1*:⟦*R module M1*; *R module M2*; *R module N*;
m1 ∈ *carrier M1*; *m2* ∈ *carrier M2*; *r* ∈ *carrier R*;
bilinear-map f R M1 M2 N⟧ ⟹
$f\ (r\ \cdot_{s\,M1}\ m1,\ m2)\ \pm_N\ -_{a\,N}\ (r\ \cdot_{s\,N}\ (f\ (m1,\ m2))) = \mathbf{0}_N$
apply (*frule Module.module-is-ag*[*of N*],
subst aGroup.ag-eq-diffzero[*of N*, *THEN sym*], *assumption*+,
frule-tac a = *r* **and** *m* = *m1* **in** *Module.sc-mem*[*of M1 R*], *assumption*+,
simp add:*bilinear-mem*)
apply (*rule Module.sc-mem*, *assumption*+, *simp add*:*bilinear-mem*,
simp add:*bilinear-l-lin*)
done

lemma (**in** *Ring*) *bilinear-r-lin*:⟦*R module M1*; *R module M2*; *R module N*;
m1 ∈ *carrier M1*; *m2* ∈ *carrier M2*; *r* ∈ *carrier R*;
bilinear-map f R M1 M2 N⟧ ⟹ $f\ (m1,\ r\ \cdot_{s\,M2}\ m2) = r\ \cdot_{s\,N}\ (f\ (m1,\ m2))$
apply (*simp add*:*bilinear-map-def*)
done

lemma (**in** *Ring*) *bilinear-r-lin1*:⟦*R module M1*; *R module M2*; *R module N*;
m1 ∈ *carrier M1*; *m2* ∈ *carrier M2*; *r* ∈ *carrier R*;
bilinear-map f R M1 M2 N⟧ ⟹
$f\ (m1,\ r\ \cdot_{s\,M2}\ m2)\ \pm_N\ -_{a\,N}\ (r\ \cdot_{s\,N}\ (f\ (m1,\ m2))) = \mathbf{0}_N$
apply (*frule Module.module-is-ag*[*of N*],
subst aGroup.ag-eq-diffzero[*of N*, *THEN sym*], *assumption*+,
frule-tac a = *r* **and** *m* = *m2* **in** *Module.sc-mem*[*of M2 R*], *assumption*+,
simp add:*bilinear-mem*)
apply (*rule Module.sc-mem*, *assumption*+, *simp add*:*bilinear-mem*,

simp add:*bilinear-r-lin*)
done

lemma (**in** *Ring*) *bilinear-l-0*:⟦*R module M1*; *R module M2*; *R module N*;
m2 ∈ *carrier M2*; *bilinear-map f R M1 M2 N*⟧ ⟹ *f* (**0**$_{M1}$, *m2*) = **0**$_N$
apply (*frule Module.module-inc-zero* [*of M1 R*])
apply (*frule bilinear-l-add* [*of M1 M2 N* **0**$_{M1}$ **0**$_{M1}$ *m2 f*], *assumption*+)
apply (*frule Module.module-is-ag* [*of M1 R*], *simp add*:*aGroup.ag-l-zero*)
apply (*frule bilinear-mem* [*of M1 M2 N* **0**$_{M1}$ *m2 f*], *assumption*+)
apply (*frule Module.module-is-ag* [*of N R*])
apply (*frule aGroup.ag-eq-sol1* [*of N f* (**0**$_{M1}$, *m2*) *f* (**0**$_{M1}$, *m2*)
f (**0**$_{M1}$, *m2*)], *assumption*+)
apply (*rule sym*, *assumption*+)
apply (*simp add*:*aGroup.ag-l-inv1*)
done

lemma (**in** *Ring*) *bilinear-r-0*:⟦*R module M1*; *R module M2*; *R module N*;
m1 ∈ *carrier M1*; *bilinear-map f R M1 M2 N*⟧ ⟹ *f* (*m1*, **0**$_{M2}$) = **0**$_N$
apply (*frule Module.module-inc-zero* [*of M2 R*])
apply (*frule bilinear-r-add* [*of M1 M2 N m1* **0**$_{M2}$ **0**$_{M2}$ *f*], *assumption*+)
apply (*frule Module.module-is-ag* [*of M2 R*])
apply (*simp add*:*aGroup.ag-l-zero*)
apply (*frule bilinear-mem* [*of M1 M2 N m1* **0**$_{M2}$ *f*], *assumption*+)
apply (*frule Module.module-is-ag* [*of N R*])
apply (*frule aGroup.ag-eq-sol1* [*of N f* (*m1*, **0**$_{M2}$) *f* (*m1*, **0**$_{M2}$)
f (*m1*, **0**$_{M2}$)], *assumption*+)
apply (*rule sym*, *assumption*+)
apply (*simp add*:*aGroup.ag-l-inv1*)
done

definition
universal-property :: [('*r*, '*m*) *Ring-scheme*, ('*a*, '*r*, '*m1*) *Module-scheme*,
('*b*, '*r*, '*m1*) *Module-scheme*, ('*c*, '*r*, '*m1*) *Module-scheme*,
'*a* ∗ '*b* ⇒'*c*] ⇒ *bool* **where**
universal-property (*R*::('*r*, '*m*) *Ring-scheme*) (*M*::('*a*, '*r*, '*m1*) *Module-scheme*)
(*N*:: ('*b*, '*r*, '*m1*) *Module-scheme*) (*MN*::('*c*, '*r*, '*m1*) *Module-scheme*)
(*f*:: '*a* ∗ '*b* ⇒ '*c*) ⟷ (*bilinear-map f R M N MN*) ∧
(∀(*Z* :: ('*c*, '*r*, '*m1*) *Module-scheme*). ∀(*g* :: '*a* ∗ '*b* ⇒ '*c*). (*R module Z*) ∧
(*bilinear-map g R M N Z*) ⟶ ((∃!*h*. (*h* ∈ *mHom R MN Z*) ∧
(*compose* (*M* ×$_c$ *N*) *h f* = *g*))))

lemma *tensor-prod-uniqueTr*:⟦*Ring R*; *R module* (*M*::('*a*, '*r*, '*m1*) *Module-scheme*);

R module (*N*:: ('*b*, '*r*, '*m1*) *Module-scheme*);
R module (*MN*:: ('*c*, '*r*, '*m1*) *Module-scheme*);
R module (*MN1*::('*c*, '*r*, '*m1*) *Module-scheme*);
universal-property R M N MN f; *universal-property R M N MN1 g*⟧ ⟹

$\exists!k.\ k \in mHom\ R\ MN1\ MN \wedge compose\ (M \times_c N)\ k\ g = f$
apply (*simp add*: *universal-property-def* [*of* - - - - *f*])
apply (*frule conjunct1*) **apply** (*fold universal-property-def*)
apply (*simp add*:*universal-property-def* [*of* - - - - *g*])
done

lemma *tensor-prod-unique*:⟦*Ring* (*R*:: ('*r*, '*m*) *Ring-scheme*);
R module (*M* :: ('*a*, '*r*, '*m1*) *Module-scheme*);
R module (*N*:: ('*b*, '*r*, '*m1*) *Module-scheme*);
R module (*MN*:: ('*c*, '*r*, '*m1*) *Module-scheme*);
R module (*MN1*::('*c*, '*r*, '*m1*) *Module-scheme*);
universal-property R M N MN f; *universal-property R M N MN1 g*⟧ ⟹
$MN \cong_R MN1$
apply (*frule tensor-prod-uniqueTr*[*of R M N MN MN1 f g*], *assumption*+,
erule ex1E,
thin-tac $\forall y.\ y \in mHom\ R\ MN1\ MN \wedge compose\ (M \times_c N)\ y\ g = f \longrightarrow y = k$,
frule tensor-prod-uniqueTr [*of R M N MN1 MN g f*], *assumption*+)
apply (*erule ex1E*,
thin-tac $\forall y.\ y \in mHom\ R\ MN\ MN1 \wedge compose\ (M \times_c N)\ y\ f = g \longrightarrow y = ka$,
(*erule conjE*)+,
rename-tac k h,
frule-tac f = *k* **in** *Ring.mHom-func*[*of R - MN1 MN*], *assumption*)
apply (*subgoal-tac f* ∈ (*M* ×$_c$ *N*) → (*carrier MN*))
prefer *2* **apply** (*simp add*:*universal-property-def bilinear-map-def*)
apply (*frule-tac f* = *h* **in** *Ring.mHom-func*[*of R - MN MN1*], *assumption*,
frule-tac g = *h* **and** *h* = *k* **in** *compose-assoc* [*of f M* ×$_c$ *N*
carrier MN - carrier MN1 - carrier MN], *assumption*+,
simp)
apply (*subgoal-tac g* ∈ (*M* ×$_c$ *N*) → (*carrier MN1*))
prefer *2* **apply** (*simp add*:*universal-property-def bilinear-map-def*)
apply (*frule-tac g* = *k* **and** *h* = *h* **in** *compose-assoc* [*of g M* ×$_c$ *N*
carrier MN1 - carrier MN - carrier MN1], *assumption*+,
simp)
apply (*subgoal-tac compose* (*M* ×$_c$ *N*) (mId_{MN}) *f* = *f*)
prefer *2*
apply (*frule Module.mId-mHom* [*of MN R*],
frule-tac f = mId_{MN} **in** *Ring.mHom-func*[*of R - MN MN*], *assumption*,
frule composition [*of f M* ×$_c$ *N carrier MN* mId_{MN} *carrier MN*],
assumption+,
rule funcset-eq [*of - M* ×$_c$ *N*])
apply (*simp add*:*compose-def restrict-def extensional-def*,
simp add:*universal-property-def bilinear-map-def*)
apply (*simp add*:*compose-def mId-def funcset-mem del*:*Pi-I'*)
apply (*rotate-tac* −*4*)
apply (*frule sym*,
thin-tac f = *compose* (*M* ×$_c$ *N*) (*compose* (*carrier MN*) *k h*) *f*)
apply (*subgoal-tac* (*compose* (*carrier MN*) *k h*) = mId_{MN})

apply (*subgoal-tac* (*compose* (*carrier MN1*) *h k*) = (mId_{MN1}))
apply (*simp add*:*misomorphic-def*)
apply (*frule-tac f* = *h* **and** *g* = *k* **in** *Module.mHom-mId-bijec* [*of MN R MN1*],
assumption+)
apply *blast*
apply (*subgoal-tac compose* (*M* $\times_c$ *N*) (mId_{MN1}) *g* = *g*)
prefer *2*
apply (*frule Module.mId-mHom* [*of MN1 R*])
apply (*subgoal-tac* mId_{MN1} $\in$ *carrier MN1* $\rightarrow$ *carrier MN1*)
prefer *2* **apply** (*simp add*:*mHom-def aHom-def*)
apply (*frule composition* [*of g M* $\times_c$ *N carrier MN1* mId_{MN1}
carrier MN1], *assumption*+,
rule funcset-eq [*of* - *M* $\times_c$ *N*],
simp add:*compose-def restrict-def extensional-def*,
simp add:*universal-property-def bilinear-map-def*)
apply (*simp add*:*compose-def mId-def*,
simp add:*funcset-mem del*:*Pi-I'*,
frule sym,
thin-tac g = *compose* (*M* $\times_c$ *N*) (*compose* (*carrier MN1*) *h k*) *g*,
frule tensor-prod-uniqueTr [*of R M N MN1 MN1 g g*], *assumption*+)
apply (*erule ex1E*,
frule Module.mId-mHom [*of MN1 R*])
apply (*subgoal-tac* mId_{MN1} = *ka*) **prefer** *2*
apply (*thin-tac compose* (*M* $\times_c$ *N*) *k g* = *f*,
thin-tac compose (*M* $\times_c$ *N*) *h f* = *g*,
thin-tac compose (*M* $\times_c$ *N*) (*compose* (*carrier MN*) *k h*) *f* = *f*,
thin-tac compose (*carrier MN*) *k h* = mId_{MN},
thin-tac ka $\in$ *mHom R MN1 MN1* $\wedge$ *compose* (*M* $\times_c$ *N*) *ka g* = *g*,
thin-tac compose (*M* $\times_c$ *N*) (*compose* (*carrier MN1*) *h k*) *g* = *g*,
blast)
apply (*subgoal-tac compose* (*carrier MN1*) *h k* = *ka*,
thin-tac k $\in$ *mHom R MN1 MN*,
thin-tac compose (*M* $\times_c$ *N*) *k g* = *f*,
thin-tac compose (*M* $\times_c$ *N*) *h f* = *g*,
thin-tac k $\in$ *carrier MN1* $\rightarrow$ *carrier MN*,
thin-tac f $\in$ *M* $\times_c$ *N* $\rightarrow$ *carrier MN*,
thin-tac h $\in$ *carrier MN* $\rightarrow$ *carrier MN1*,
thin-tac compose (*M* $\times_c$ *N*) (*compose* (*carrier MN*) *k h*) *f* = *f*,
thin-tac compose (*carrier MN*) *k h* = mId_{MN},
thin-tac compose (*M* $\times_c$ *N*) (mId_{MN1}) *g* = *g*,
thin-tac compose (*M* $\times_c$ *N*) (*compose* (*carrier MN1*) *h k*) *g* = *g*,
thin-tac ka $\in$ *mHom R MN1 MN1* $\wedge$ *compose* (*M* $\times_c$ *N*) *ka g* = *g*,
thin-tac $\forall$ *y*. *y* $\in$ *mHom R MN1 MN1* $\wedge$
compose (*M* $\times_c$ *N*) *y g* = *g* $\longrightarrow$ *y* = *ka*)
apply *simp*
apply (*thin-tac* mId_{MN1} $\in$ *mHom R MN1 MN1*,
thin-tac compose (*M* $\times_c$ *N*) (mId_{MN1}) *g* = *g*,
thin-tac ka $\in$ *mHom R MN1 MN1* $\wedge$ *compose* (*M* $\times_c$ *N*) *ka g* = *g*,
thin-tac mId_{MN1} = *ka*)

apply (*subgoal-tac* (*compose* (*carrier MN1*) *h k*) ∈ *mHom R MN1 MN1*)
apply *simp*
apply (*thin-tac* ∀ *y*. *y* ∈ *mHom R MN1 MN1* ∧
compose (*M* ×$_c$ *N*) *y g* = *g* ⟶ *y* = *ka*)

apply (*frule-tac f* = *k* **and** *g* = *h* **in** *Module.mHom-compos*[*of MN R MN1 MN1*],
assumption+)
apply (*simp add*:*compos-def*)

apply (*frule Module.mId-mHom* [*of MN R*])
apply (*subgoal-tac compose* (*M* ×$_c$ *N*) (*mId*$_{MN}$) *f* = *f*)
prefer *2*
apply (*frule-tac f* = *mId*$_{MN}$ **in** *Ring.mHom-func*[*of R - MN MN*], *assumption*)
apply (*frule composition* [*of f M* ×$_c$ *N carrier MN mId*$_{MN}$ *carrier MN*],
assumption+)
apply (*frule tensor-prod-uniqueTr* [*of R M N MN MN f f*],
assumption+)
apply (*erule ex1E*)
apply (*subgoal-tac mId*$_{MN}$ = *ka*) **prefer** *2*
apply (*thin-tac compose* (*M* ×$_c$ *N*) *k g* = *f*,
thin-tac compose (*M* ×$_c$ *N*) *h f* = *g*,
thin-tac ka ∈ *mHom R MN MN* ∧ *compose* (*M* ×$_c$ *N*) *ka f* = *f*)
apply *blast*
apply (*rotate-tac* −*1*) **apply** (*frule sym*, *thin-tac mId*$_{MN}$ = *ka*)
apply (*thin-tac compose* (*M* ×$_c$ *N*) *k g* = *f*,
thin-tac compose (*M* ×$_c$ *N*) *h f* = *g*,
thin-tac k ∈ *carrier MN1* → *carrier MN*,
thin-tac f ∈ *M* ×$_c$ *N* → *carrier MN*,
thin-tac h ∈ *carrier MN* → *carrier MN1*,
thin-tac compose (*M* ×$_c$ *N*) (*mId*$_{MN}$) *f* = *f*)
apply (*subgoal-tac* (*compose* (*carrier MN*) *k h*) ∈ *mHom R MN MN*)
apply *simp*
apply (*frule-tac f* = *h* **and** *g* = *k* **in** *Module.mHom-compos*[*of MN1 R MN MN*],

assumption+)
apply (*metis compos-def*)
done

Chapter 6

Construction of an abelian group

6.1 Free generated abelian group I, direct sum and direct product 2

definition
bpp :: [$'a \Rightarrow 'a \Rightarrow 'a$, $'a$, $'a$] $\Rightarrow$ $'a$ **where**
bpp f a b = *f a b*

definition
ipp :: [$'a \Rightarrow 'a$, $'a$] $\Rightarrow$ $'a$ ((-$-_i$/ -) [64,65]64) **where**
$-_i$ *a* == *i a*

definition
sop :: [$'r \Rightarrow 'a \Rightarrow 'a$, $'r$, $'a$] $\Rightarrow$ $'a$ **where**
sop s r a = *s r a*

abbreviation
BOP :: [$'a$, $'a \Rightarrow 'a \Rightarrow 'a$, $'a$] $\Rightarrow$ $'a$
((3-/ -$+_f$/ -) [62,62,63]62) **where**
a $+_f$ *b* == *bpp f a b*

abbreviation
SOP :: [$'r$, $'r \Rightarrow 'a \Rightarrow 'a$, $'a$] $\Rightarrow$ $'a$
((3-/ -$\cdot_s$ -) [68,68,69]68) **where**
r $\cdot_s$ *a* == *sop s r a*

definition
minus-set :: [$'a \Rightarrow 'a$, $'a$ *set*] $\Rightarrow$ $'a$ *set* **where**
minus-set i A = $\{x.\ \exists y{\in}A.\ x = -_i\ y\}$

definition
pm-set :: [$'a \Rightarrow 'a$, $'a$ *set*] $\Rightarrow$ $'a$ *set* **where**

pm-set i A = A ∪ (minus-set i A)

definition
s-set :: [('r, 'm) *Ring-scheme*, 'r ⇒ 'a ⇒ 'a, 'a *set*] ⇒ 'a *set* **where**
s-set R s A = {x. ∃ r∈*carrier R*. ∃ a∈A. x = r $_s$· a} ∪ A

primrec *add-set* :: ['a ⇒ 'a ⇒ 'a, 'a *set*] ⇒ *nat* ⇒ 'a *set*
where
add-set-0 : *add-set f A 0* = A
| *add-set-Suc*: *add-set f A* (*Suc n*) =
{x. ∃ s∈ (*add-set f A n*). ∃ t∈ A. x = s $_f$+ t}

definition
aug-pm-set :: ['a, 'a ⇒ 'a, 'a *set*] ⇒ 'a *set* **where**
aug-pm-set z i A = {z} ∪ A ∪ (*minus-set i A*)

definition
addition-set :: ['a ⇒ 'a ⇒ 'a, 'a *set*] ⇒ 'a *set* **where**
addition-set f A = ⋃{*add-set f A n* | n. (*0*::*nat*)≤ n}

definition
assoc-bpp :: ['a *set*, 'a ⇒ 'a ⇒ 'a] ⇒ *bool* **where**
assoc-bpp A f ⟷
(∀ a∈(*addition-set f A*). ∀ b∈(*addition-set f A*). ∀ c∈(*addition-set f A*). (a $_f$+ b) $_f$+ c = a $_f$+ (b $_f$+ c))

definition
commute-bpp :: ['a ⇒ 'a ⇒ 'a, 'a *set*] ⇒ *bool* **where**
commute-bpp f A ⟷ (∀ x∈*addition-set f A*. ∀ y∈*addition-set f A*. x $_f$+ y = y $_f$+ x)

definition
zeroA :: ['a, 'a ⇒ 'a, 'a ⇒ 'a ⇒ 'a, 'a *set*] ⇒ 'a ⇒ *bool* **where**
zeroA z i f A z1 ⟷ (∀ x ∈ *addition-set f* (*aug-pm-set z i A*). *z1* $_f$+ x = x)

definition
inv-ipp :: ['a, 'a ⇒ 'a, 'a ⇒ 'a ⇒ 'a, 'a *set*] ⇒ *bool* **where**
inv-ipp z i f A ⟷ (∀ a∈*addition-set f* (*aug-pm-set z i A*). *zeroA z i f A* (($_i$− a) $_f$+ a))

definition
ipp-cond1 :: ['a *set*, 'a ⇒ 'a] ⇒ *bool* **where**
ipp-cond1 A i ⟷ (∀ x∈A. $_i$− ($_i$− x) = x)

definition
ipp-cond2 :: ['a, 'a *set*, 'a ⇒ 'a, 'a ⇒ 'a ⇒ 'a] ⇒ *bool* **where**
ipp-cond2 z A i f == ∀ x∈(*addition-set f* (*aug-pm-set z i A*)).
∀ y∈ (*addition-set f* (*aug-pm-set z i A*)). $_i$−(x $_f$+ y) = $_i$− y $_f$+ ($_i$− x)

definition
ipp-cond3 :: [*'a*, *'a* ⇒ *'a*] ⇒ *bool* **where**
ipp-cond3 z i ⟷ $_i$− *z* = *z*

lemma *add-set-mono*:*A* ⊆ *B* ⟹ *add-set f A n* ⊆ *add-set f B n*
apply (*induct-tac n*)
apply *simp*
apply (*rule subsetI*, *simp*)
apply (*erule bexE*)+
apply (*frule-tac A* = *add-set f A n* **and** *B* = *add-set f B n* **and** *c* = *s* **in**
subsetD, *assumption*+)
apply (*frule-tac A* = *A* **and** *B* = *B* **and** *c* = *t* **in** *subsetD*, *assumption*+)
apply *blast*
done

lemma *addition-inc-add*:*add-set f A n* ⊆ *addition-set f A*
apply (*rule subsetI*)
apply (*simp add*:*addition-set-def*)
apply *blast*
done

lemma *addition-inc-add0*: *A* ⊆ *addition-set f A*
apply (*rule subsetI*)
apply (*insert addition-inc-add* [*of f A 0*])
apply *simp*
apply (*simp add*:*subsetD*)
done

lemma *addition-set-mono*:*A* ⊆ *B* ⟹ *addition-set f A* ⊆ *addition-set f B*
apply (*rule subsetI*)
apply (*simp add*:*addition-set-def* [*of f A*])
apply (*erule exE*, *erule conjE*, *erule exE*, *simp*)
apply (*frule-tac n* = *n* **in** *add-set-mono* [*of A B f*],
frule-tac A = *add-set f A n* **and** *B* = *add-set f B n* **and** *c* = *x* **in**
subsetD, *assumption*+)
apply (*cut-tac n* = *n* **in** *addition-inc-add*[*of f B*])
apply (*simp add*:*subsetD*)
done

lemma *a-in-aug-pm-set*:*a* ∈ *A* ⟹ *a* ∈ *aug-pm-set z i A*
apply (*simp add*:*aug-pm-set-def*)
done

lemma *A-sub-aug-pm-set*:*A* ⊆ *aug-pm-set z i A*
by (*rule subsetI*, *simp add*:*aug-pm-set-def*)

lemma *addition-sub-aug-pm-addition*:
addition-set f A ⊆ *addition-set f* (*aug-pm-set z i A*)
apply (*cut-tac A-sub-aug-pm-set*[*of A z i*])

apply (*simp add*:*addition-set-mono*)
done

lemma *assoc-bpp-restrict*:⟦ $A \subseteq B$; *assoc-bpp* B f⟧ ⟹ *assoc-bpp* A f
apply (*simp add*:*assoc-bpp-def*)
apply (*rule ballI*)+
apply (*frule addition-set-mono*[*of* A B f])
apply *blast*
done

lemma *addition-assoc*:⟦*assoc-bpp* A f; $x \in$ *addition-set* f A;
$y \in$ *addition-set* f A; $z \in$ *addition-set* f A⟧ ⟹
$(x\ {}_f{+}\ y)\ {}_f{+}\ z = x\ {}_f{+}\ (y\ {}_f{+}\ z)$
apply (*simp add*:*assoc-bpp-def*)
done

lemma *bpp-closedTr*:*assoc-bpp* A f ⟹
$\forall x\ y.\ x \in$ *add-set* f A $n \wedge y \in$ *add-set* f A $m \longrightarrow$
$x\ {}_f{+}\ y \in$ *add-set* f A $(n + m + Suc\ 0)$
apply (*induct-tac* m, *simp*, *blast*)
apply ((*rule allI*)+, *rule impI*, *erule conjE*)
apply (*simp*, (*erule bexE*)+)
apply (*cut-tac addition-inc-add*[*of* f A n],
cut-tac $n = na$ **in** *addition-inc-add*[*of* f A],
cut-tac addition-inc-add0[*of* A f])
apply (*drule-tac* $x = x$ **in** *spec*,
drule-tac $a = s$ **in** *forall-spec*, *simp*)

apply ((*erule bexE*)+, *simp*)
apply (*cut-tac* $n = n + na$ **in** *addition-inc-add*[*of* f A],
frule-tac $c = sa$ **and** $A =$ *add-set* f A $(n + na)$ **in** *subsetD*[*of* -
addition-set f A], *assumption*+,
frule-tac $c = x$ **in** *subsetD*[*of add-set* f A n *addition-set* f A],
assumption+,
frule-tac $c = s$ **and** $A =$ *add-set* f A na **in**
subsetD[*of* - *addition-set* f A], *assumption*+,
frule-tac $c = t$ **in** *subsetD*[*of* A *addition-set* f A], *assumption*+,
frule-tac $c = ta$ **in** *subsetD*[*of* A *addition-set* f A], *assumption*+)
apply (*frule-tac* $x1 = x$ **and** $y1 = s$ **and** $z1 = t$ **in** *addition-assoc*[*THEN sym*,
of A f], *assumption*+) **apply** *simp* **apply** *blast*
done

lemma *bpp-closed1*:⟦*assoc-bpp* A f; $x \in$ *add-set* f A n; $y \in$ *add-set* f A m⟧ ⟹
$x\ {}_f{+}\ y \in$ *add-set* f A $(n + m + Suc\ 0)$
apply (*insert bpp-closedTr*[*of* A f])
apply *blast*
done
lemma *bpp-closed*:⟦*assoc-bpp* A f; $x \in$ *addition-set* f A; $y \in$ *addition-set* f A⟧
⟹ $x\ {}_f{+}\ y \in$ *addition-set* f A

apply (*simp add*:*addition-set-def*)
apply ((*erule exE*)+, (*erule conjE*)+, (*erule exE*)+, *simp*)
apply (*frule-tac* $x = x$ **and** $n = n$ **and** $y = y$ **and** $m = na$ **in** *bpp-closed1*,
assumption+)
apply *blast*
done

lemma *aug-addition-inc-z*: $z \in$ *addition-set* f (*aug-pm-set* z i A)
apply (*subgoal-tac* $z \in$ *aug-pm-set* z i A)
apply (*subgoal-tac aug-pm-set* z i $A \subseteq$ *addition-set* f (*aug-pm-set* z i A))
apply (*simp add*:*subsetD*)
apply (*simp add*:*addition-inc-add0*)
apply (*simp add*:*aug-pm-set-def*)
done

lemma *aug-bpp-closed*:⟦*assoc-bpp* (*aug-pm-set* z i A) f;
$x \in$ *addition-set* f (*aug-pm-set* z i A);
$y \in$ *addition-set* f (*aug-pm-set* z i A)⟧ ⟹
x $_f$+ $y \in$ *addition-set* f (*aug-pm-set* z i A)
apply (*simp add*:*bpp-closed* [*of aug-pm-set* z i A f])
done

lemma *aug-commute*:⟦*commute-bpp* f (*aug-pm-set* z i A);
$x \in$ *addition-set* f (*aug-pm-set* z i A);
$y \in$ *addition-set* f (*aug-pm-set* z i A)⟧ ⟹ x $_f$+ $y = y$ $_f$+ x
apply (*simp add*: *commute-bpp-def*)
done

lemma *addition-set-inc-z*:$z \in$ *addition-set* f (*aug-pm-set* z i A)
apply (*simp add*:*addition-set-def*)
apply (*subgoal-tac* $z \in$ *add-set* f (*aug-pm-set* z i A) *0*)
apply *blast*
apply (*simp add*:*aug-pm-set-def*)
done

lemma *aug-ipp-closed0*:⟦*commute-bpp* f (*aug-pm-set* z i A);
assoc-bpp (*aug-pm-set* z i A) f; *ipp-cond1* A i; *ipp-cond2* z A i f;
ipp-cond3 z i; $x \in$ *add-set* f (*aug-pm-set* z i A) *0*⟧ ⟹
$_i$− $x \in$ *add-set* f (*aug-pm-set* z i A) *0*
apply (*simp add*:*aug-pm-set-def*)
apply (*case-tac* $x \in A$, *simp add*:*minus-set-def*, *blast*)
apply *simp*
apply (*simp add*:*minus-set-def*)
apply (*case-tac* $x = z$, *simp*, *simp add*:*ipp-cond3-def*)
apply *simp*
apply (*erule bexE*)
apply (*simp add*:*ipp-cond1-def*)
done

lemma *aug-ipp-closedTr*:⟦*commute-bpp f* (*aug-pm-set z i A*);
assoc-bpp (*aug-pm-set z i A*) *f*; *ipp-cond1 A i*; *ipp-cond2 z A i f*;
ipp-cond3 z i⟧ ⟹
∀ *x*. *x* ∈ *add-set f* (*aug-pm-set z i A*) *n* ⟶
$_i-$ *x* ∈ *add-set f* (*aug-pm-set z i A*) *n*
apply (*induct-tac n*, *rule allI*, *rule impI*)
apply (*simp add*:*aug-pm-set-def*)
apply (*case-tac x* = *z*, *simp add*:*ipp-cond3-def*)
apply *simp*
apply (*case-tac x* ∈ *A*, *simp add*:*minus-set-def*, *blast*)
apply *simp*
apply (*simp add*:*minus-set-def*)
apply (*erule bexE*, *simp add*:*ipp-cond1-def*)
apply (*rule allI*, *rule impI*, *simp*)
apply (*erule bexE*)+
apply (*drule-tac a* = *s* **in** *forall-spec*, *assumption*)
apply (*cut-tac ipp-cond2-def*[*of z A i f*], *simp*)
apply (*cut-tac n* = *n* **in** *addition-inc-add*[*of f aug-pm-set z i A*],
frule-tac c = *s* **and** *A* = *add-set f* (*aug-pm-set z i A*) *n* **in**
subsetD[*of* - *addition-set f* (*aug-pm-set z i A*)], *assumption*+,
cut-tac addition-inc-add0[*of aug-pm-set z i A f*],
frule-tac c = *t* **in** *subsetD*[*of aug-pm-set z i A*
addition-set f (*aug-pm-set z i A*)], *assumption*+)
apply (*drule-tac x* = *s* **in** *bspec*, *assumption*,
drule-tac x = *t* **in** *bspec*, *assumption*)

apply (*frule-tac x* = *t* **in** *aug-ipp-closed0*[*of f z i A*], *assumption*+,
simp, *assumption*+, *simp*, *simp*,
frule-tac c = $_i-$ *t* **in** *subsetD*[*of aug-pm-set z i A*
addition-set f (*aug-pm-set z i A*)], *assumption*+)
apply (*frule-tac c* = $_i-$ *s* **and** *A* = *add-set f* (*aug-pm-set z i A*) *n* **in**
subsetD[*of* - *addition-set f* (*aug-pm-set z i A*)], *assumption*+,
frule-tac x = $_i-$ *t* **and** *y* = $_i-$ *s* **in** *aug-commute*[*of f z i A*],
assumption+, *simp*)
apply *blast*
done

lemma *aug-ipp-closedTr2*:⟦*commute-bpp f* (*aug-pm-set z i A*);
assoc-bpp (*aug-pm-set z i A*) *f*; *ipp-cond1 A i*; *ipp-cond2 z A i f*;
ipp-cond3 z i; *x* ∈ *add-set f* (*aug-pm-set z i A*) *n*⟧ ⟹
$_i-$ *x* ∈ *add-set f* (*aug-pm-set z i A*) *n*
apply (*simp add*:*aug-ipp-closedTr*)
done

lemma *aug-ipp-closed*:⟦*commute-bpp f* (*aug-pm-set z i A*);
assoc-bpp (*aug-pm-set z i A*) *f*; *ipp-cond1 A i*; *ipp-cond2 z A i f*;
ipp-cond3 z i; *x* ∈ *addition-set f* (*aug-pm-set z i A*)⟧ ⟹
$_i-$ *x* ∈ *addition-set f* (*aug-pm-set z i A*)
apply (*simp add*:*addition-set-def*, *erule exE*, *erule conjE*, *erule exE*, *simp*)

apply (*frule-tac n = n* **in** *aug-ipp-closedTr2*[*of f z i A x*], *assumption*+)
apply *blast*
done

lemma *aug-zero-unique*:⟦*commute-bpp f* (*aug-pm-set z i A*);
z1 ∈ *addition-set f* (*aug-pm-set z i A*); *zeroA z i f A z*;
zeroA z i f A z1⟧ ⟹ *z = z1*
apply (*simp add*:*zeroA-def*[*of z - - - z*])
apply (*drule-tac x = z1* **in** *bspec*, *assumption*)

apply (*cut-tac addition-set-inc-z* [*of z f i A*])
apply (*frule aug-commute* [*of f z i A z z1*], *assumption*+)
apply *simp*
apply (*simp add*:*zeroA-def*[*of - - - - z1*])
done

lemma *inv-aug-addition*:⟦*commute-bpp f* (*aug-pm-set z i A*);
assoc-bpp (*aug-pm-set z i A*) *f*; *ipp-cond1 A i*; *ipp-cond2 z A i f*;
ipp-cond3 z i; *inv-ipp z i f A*; *commute-bpp f* (*aug-pm-set z i A*);
zeroA z i f A z⟧ ⟹
∀ *a*∈*addition-set f* (*aug-pm-set z i A*). (*i*−*a*) *f*+ *a = z*
apply (*simp add*:*inv-ipp-def*)
apply (*rule ballI*)
apply (*drule-tac x = a* **in** *bspec*, *assumption*)
apply (*frule-tac ?z1.0 = (i− a f+ a)* **in** *aug-zero-unique* [*of f z i A*])
apply (*frule-tac x = a* **in** *aug-ipp-closed* [*of f z i A*], *assumption*+)
apply (*rule-tac x = i− a* **and** *y = a* **in** *aug-bpp-closed* [*of z i A f*],
assumption+)
apply (*simp add*:*zeroA-def*)
done

definition
fag-gen-by :: ['*a set*, '*a* ⇒ '*a* ⇒ '*a*, '*a* ⇒ '*a*, '*a*] ⇒ '*a aGroup* **where**
fag-gen-by A f i z = (|*carrier* = *addition-set f* (*aug-pm-set z i A*),
pop = λ*x*∈(*addition-set f* (*aug-pm-set z i A*)).
λ*y*∈(*addition-set f* (*aug-pm-set z i A*)). *x f*+ *y*,
mop = λ*x*∈(*addition-set f* (*aug-pm-set z i A*)). *i*− *x*, *zero* = *z*|)

lemma *fag-gen-carrier*:⟦*commute-bpp f* (*aug-pm-set z i A*);
assoc-bpp (*aug-pm-set z i A*) *f*; *ipp-cond1 A i*; *ipp-cond2 z A i f*;
ipp-cond3 z i; *inv-ipp z i f A*; *commute-bpp f* (*aug-pm-set z i A*);
zeroA z i f A z⟧ ⟹
carrier (*fag-gen-by A f i z*) = *addition-set f* (*aug-pm-set z i A*)
by (*simp add*:*fag-gen-by-def*)

lemma *addition-set-sub-fag-gen-carrier*:⟦*commute-bpp f* (*aug-pm-set z i A*);
assoc-bpp (*aug-pm-set z i A*) *f*; *ipp-cond1 A i*; *ipp-cond2 z A i f*;
ipp-cond3 z i; *inv-ipp z i f A*; *commute-bpp f* (*aug-pm-set z i A*);

zeroA z i f A z⟧ ⟹ *addition-set f A* ⊆ *carrier* (*fag-gen-by A f i z*)
apply (*simp add*:*fag-gen-carrier*)
apply (*simp add*:*addition-sub-aug-pm-addition*)
done

lemma *fag-aGroup*:⟦*commute-bpp f* (*aug-pm-set z i A*);
assoc-bpp (*aug-pm-set z i A*) *f*; *ipp-cond1 A i*; *ipp-cond2 z A i f*;
ipp-cond3 z i; *inv-ipp z i f A*; *commute-bpp f* (*aug-pm-set z i A*);
zeroA z i f A z⟧ ⟹ *aGroup* (*fag-gen-by A f i z*)
apply (*rule aGroup.intro*)
apply (*simp add*:*fag-gen-by-def aug-bpp-closed*)

apply (*simp add*:*fag-gen-by-def*)
apply (*simp add*:*aug-bpp-closed*)
apply (*simp add*:*assoc-bpp-def*)

apply (*simp add*:*fag-gen-by-def*)
apply (*simp add*:*aug-commute*)

apply (*simp add*:*fag-gen-by-def aug-ipp-closed*)

apply (*simp add*:*fag-gen-by-def inv-aug-addition aug-ipp-closed*)

apply (*simp add*:*fag-gen-by-def addition-set-inc-z*)

apply (*simp add*:*fag-gen-by-def addition-set-inc-z zeroA-def*)
done

6.2 Abelian group generated by a singleton (constructive)

definition
fag-single :: [*'a*, *'a* ⇒ *'a* ⇒ *'a*, *'a* ⇒ *'a*, *'a*] ⇒ *'a aGroup* **where**
fag-single a f i z = *fag-gen-by* {*a*} *f i z*

lemma *aug-pm-aug-pm-minus*:*ipp-cond1* {*a*} *i* ⟹
aug-pm-set z i {*a*} = *aug-pm-set z i* {*i*− *a*}
apply (*simp add*:*aug-pm-set-def minus-set-def*)
apply (*simp add*:*ipp-cond1-def*)
apply (*rule equalityI*, *rule subsetI*, *simp*, *blast*)
apply (*rule subsetI*, *simp*, *blast*)
done

lemma *ipp-cond1-minus*:*ipp-cond1* {*a*} *i* ⟹ *ipp-cond1* {*i*− *a*} *i*
by (*simp add*:*ipp-cond1-def*)

lemma *ipp-cond2-minus*:⟦*ipp-cond1* {*a*} *i*; *ipp-cond2 z* {*a*} *i f*⟧ ⟹
ipp-cond2 z {*i*− *a*} *i f*

by (*simp add*:*ipp-cond2-def*, *simp add*:*aug-pm-aug-pm-minus*)

lemma *zeroA-minus*:⟦*ipp-cond1* {*a*} *i*; *zeroA z i f* {*a*} *z1*⟧ ⟹
zeroA z i f {$_i-$ *a*} *z1*
apply (*simp add*:*zeroA-def*)
apply (*simp add*:*aug-pm-aug-pm-minus*)
done

lemma *inv-ipp-minus*:⟦*ipp-cond1* {*a*} *i*; *inv-ipp z i f* {*a*}⟧ ⟹
inv-ipp z i f {$_i-$ *a*}
apply (*simp add*:*inv-ipp-def* [*of* - - - {*a*}])
apply (*simp add*:*aug-pm-aug-pm-minus*)
apply (*simp add*:*inv-ipp-def*)
apply (*simp add*:*zeroA-minus*)
done

lemma *fag-single-additionTr1*:⟦*commute-bpp f* (*aug-pm-set z i* {*a*});
assoc-bpp (*aug-pm-set z i* {*a*}) *f*; *ipp-cond1* {*a*} *i*; *ipp-cond2 z* {*a*} *i f*;
ipp-cond3 z i; *inv-ipp z i f* {*a*}; *commute-bpp f* (*aug-pm-set z i* {*a*});
zeroA z i f {*a*} *z*⟧ ⟹
∀ *s*. *s*∈ *add-set f* {*a*} (*Suc n*) ⟶ *s* $_f+$ $_i-$ *a* ∈ *add-set f* {*a*} *n*
apply (*cut-tac addition-inc-add0*[*of aug-pm-set z i* {*a*} *f*])
apply (*cut-tac a-in-aug-pm-set*[*of a* {*a*} *z i*], *simp*)
apply (*frule subsetD*[*of aug-pm-set z i* {*a*}
addition-set f (*aug-pm-set z i* {*a*}) *a*], *assumption*+)
apply (*induct-tac n*)
apply (*rule allI*, *rule impI*, *simp*)
apply (*frule aug-ipp-closed* [*of f z i* {*a*} *a*], *assumption*+)
apply (*simp add*:*addition-assoc*)
apply (*frule aug-commute* [*of f z i* {*a*} *a* $_i-$ *a*], *assumption*+)
apply *simp* **apply** (*thin-tac a* $_f+$ $_i-$ *a* = $_i-$ *a* $_f+$ *a*)
apply (*simp add*:*inv-aug-addition*)
apply (*cut-tac addition-set-inc-z*[*of z f i* {*a*}])
apply (*frule aug-commute* [*of f z i* {*a*} *a z*], *assumption*+)
apply *simp* **apply** (*thin-tac a* $_f+$ *z* = *z* $_f+$ *a*)
apply (*simp add*:*zeroA-def*)

apply (*rule allI*) **apply** (*rule impI*)
apply (*erule bexE*)
apply (*thin-tac* ∀ *s*. (∃ *sa*∈*add-set f* {*a*} *n*. *s* = *sa* $_f+$ *a*) ⟶
s $_f+$ $_i-$ *a* ∈ *add-set f* {*a*} *n*)
apply (*simp del*:*add-set-Suc*)
apply (*frule fag-aGroup*[*of f z i* {*a*}], *assumption*+)
apply (*cut-tac n* = *Suc n* **in** *addition-inc-add*[*of f* {*a*}],
cut-tac addition-sub-aug-pm-addition[*of f* {*a*} *z i*],
frule-tac c = *sa* **and** *A* = *add-set f* {*a*} (*Suc n*) **in** *subsetD*[*of* -
addition-set f {*a*}], *assumption*+,
frule-tac c = *sa* **in** *subsetD*[*of addition-set f* {*a*}
addition-set f (*aug-pm-set z i* {*a*})], *assumption*+)

```
 apply (cut-tac x = sa and y = a and z =  _i− a in
            aGroup.ag-pOp-assoc[of fag-gen-by {a} f i z], assumption,
       simp del:add-set-Suc add:fag-gen-carrier)
 apply (simp add:fag-gen-carrier,
       cut-tac addition-inc-add0[of aug-pm-set z i {a}],
       simp add:subsetD)
 apply (subst fag-gen-carrier, assumption+)
       apply (rule aug-ipp-closed[of f z i {a} a], assumption+)
       apply (simp add:subsetD)
apply (simp del:add-set-Suc add:fag-gen-by-def, fold fag-gen-by-def,
      frule aug-ipp-closed[of f z i {a} a], assumption+,
      simp del:add-set-Suc add:aug-bpp-closed,
      simp del:add-set-Suc add:aug-commute[of f z i {a} a _i− a],
      thin-tac sa _f+ a _f+ _i− a = sa _f+ (_i− a _f+ a),
      simp del:add-set-Suc add:inv-aug-addition,
      cut-tac addition-set-inc-z[of z f i {a}])
 apply (subst aug-commute[of f z i {a} - z], assumption+)
 apply (simp del:add-set-Suc add:zeroA-def[of z i f {a} z])

 apply simp
done

lemma fag-single-additionTr2:⟦commute-bpp f (aug-pm-set z i {a});
     assoc-bpp (aug-pm-set z i {a}) f; ipp-cond1 {a} i; ipp-cond2 z {a} i f;
     ipp-cond3 z i; inv-ipp z i f {a}; commute-bpp f (aug-pm-set z i {a});
     zeroA z i f {a} z; s ∈ add-set f {a} 0⟧ ⟹ s _f+ _i− a = z
 apply simp
 apply (cut-tac a-in-aug-pm-set[of a {a} z i],
       cut-tac addition-inc-add0[of aug-pm-set z i {a} f],
       frule subsetD[of aug-pm-set z i {a}
                   addition-set f (aug-pm-set z i {a}) a], assumption+)
 apply (frule aug-ipp-closed [of f z i {a} a], assumption+)
  apply (frule aug-commute [of f z i {a} a _i− a], assumption+)
  apply simp apply (thin-tac a _f+ _i− a = _i− a _f+ a)
  apply (simp add:inv-aug-addition)
 apply simp
done

lemma ipp-conditions:⟦assoc-bpp (aug-pm-set z i {a}) f; ipp-cond1 {a} i;
       ipp-cond2 z {a} i f; ipp-cond3 z i; inv-ipp z i f {a};
       commute-bpp f (aug-pm-set z i {a}); zeroA z i f {a} z⟧ ⟹
       assoc-bpp (aug-pm-set z i { _i− a}) f ∧ ipp-cond1 { _i− a} i ∧
       ipp-cond2 z { _i− a} i f ∧ inv-ipp z i f { _i− a} ∧
       commute-bpp f (aug-pm-set z i { _i− a}) ∧ zeroA z i f { _i− a} z
apply (simp add:aug-pm-aug-pm-minus[THEN sym])
apply (rule conjI)
 apply (subst ipp-cond1-def, rule ballI, simp, simp add:ipp-cond1-def)

apply (rule conjI)
```

apply (*subst ipp-cond2-def*,
simp add:*aug-pm-aug-pm-minus*[*THEN sym*] *ipp-cond2-def*)

apply (*simp add*:*zeroA-def inv-ipp-def*,
simp add:*aug-pm-aug-pm-minus*)
done

lemma *fag-single-additionTr3*:⟦*commute-bpp f* (*aug-pm-set z i* {*a*});
assoc-bpp (*aug-pm-set z i* {*a*}) *f*; *ipp-cond1* {*a*} *i*; *ipp-cond2 z* {*a*} *i f*;
ipp-cond3 z i; *inv-ipp z i f* {*a*}; *commute-bpp f* (*aug-pm-set z i* {*a*});
zeroA z i f {*a*} *z*; *s*∈ *add-set f* {$_i$− *a*} *n*⟧ ⟹
s $_f$+ $_i$− *a* ∈ *add-set f* {$_i$− *a*} (*Suc n*)
apply *simp* **apply** *blast*
done

lemma *fag-single-elemTr*:⟦*commute-bpp f* (*aug-pm-set z i* {*a*});
assoc-bpp (*aug-pm-set z i* {*a*}) *f*; *ipp-cond1* {*a*} *i*; *ipp-cond2 z* {*a*} *i f*;
ipp-cond3 z i; *inv-ipp z i f* {*a*}; *commute-bpp f* (*aug-pm-set z i* {*a*});
zeroA z i f {*a*} *z*⟧ ⟹
∀ *x*. *x* ∈ *add-set f* (*aug-pm-set z i* {*a*}) *n* ⟶
(∃ *n1*. *x* ∈ *add-set f* {*a*} *n1*) ∨ (∃ *m1*. *x* ∈ *add-set f* {$_i$− *a*} *m1*) ∨ *x* = *z*
apply (*cut-tac a-in-aug-pm-set*[*of a* {*a*} *z i*],
cut-tac addition-inc-add0[*of aug-pm-set z i* {*a*} *f*],
frule subsetD[*of aug-pm-set z i* {*a*}
addition-set f (*aug-pm-set z i* {*a*}) *a*], *assumption*+)
apply (*cut-tac addition-set-inc-z*[*of z f i* {*a*}])
prefer *2* **apply** *simp*
apply (*cut-tac assoc-bpp-restrict*[*of* {*a*} *aug-pm-set z i* {*a*} *f*])
apply (*induct-tac n*)
apply (*rule allI*, *rule impI*, *simp add*:*aug-pm-set-def*)
apply (*erule disjE*, *simp*)
apply (*subgoal-tac a* ∈ *add-set f* {*a*} *0*, *blast*)
apply *simp*
apply (*erule disjE*, *simp*)
apply (*simp add*:*minus-set-def*)
apply (*subgoal-tac* $_i$− *a* ∈ *add-set f* {$_i$− *a*} *0*, *blast*, *simp*)
apply (*rule allI*, *rule impI*)
apply (*simp*, (*erule bexE*)+)
apply (*drule-tac a* = *s* **in** *forall-spec*, *assumption*)
apply (*subgoal-tac t* = *a* ∨ *t* = *z* ∨ *t* = $_i$− *a*)
prefer *2* **apply** (*simp add*:*aug-pm-set-def minus-set-def*, *blast*)
apply (*erule disjE*, *erule exE*)
apply (*case-tac n1* = *0*, *simp*)
apply (*case-tac t* = *a*, *simp*)
apply (*cut-tac A-sub-aug-pm-set*[*of* {*a*} *z i*])
apply (*cut-tac addition-inc-add0*[*of* {*a*}*f*])
apply (*frule bpp-closed1*[*of* {*a*} *f a 0 a 0*],
simp, *simp*, *blast*)

apply *simp*
apply (*case-tac* $t = z$, *simp*)
apply (*simp add*:*aug-commute*[*of f z i* {*a*} *a z*])
apply (*simp add*:*zeroA-def*,
subgoal-tac $a \in$ *add-set f* {*a*} *0*, *blast*, *simp*)

apply *simp*
apply (*simp add*:*fag-single-additionTr2*[*of f z i a a*])

apply *simp*
apply (*case-tac* $t = a$, *simp*,
frule-tac $x = s$ **and** $n = n1$ **and** $y = a$ **in** *bpp-closed1*[*of* {*a*} *f*
- - - 0], *assumption*, *simp*, *blast*)

apply (*case-tac* $t = z$, *simp*)
apply (*cut-tac* $n = n$ **in** *addition-inc-add*[*of f aug-pm-set z i* {*a*}],
frule-tac $c = s$ **and** $A =$ *add-set f* (*aug-pm-set z i* {*a*}) *n* **in**
subsetD[*of - addition-set f* (*aug-pm-set z i* {*a*})], *assumption*+)
apply (*simp add*:*aug-commute*[*of f z i* {*a*} *- z*])
apply (*simp add*:*zeroA-def*, *blast*)

apply *simp*
apply (*frule-tac* $n = n1 - Suc\ 0$ **in** *fag-single-additionTr1*[*of f z i a*],
assumption+, *simp*)
apply (*drule-tac* $a = s$ **in** *forall-spec*, *assumption*, *blast*)

apply (*rotate-tac* -1, *erule disjE*)
apply (*erule exE*)

apply (*frule ipp-conditions*[*of z i a f*], *assumption*+, (*erule conjE*)+)

apply (*case-tac* $m1 = 0$, *simp*)
apply (*case-tac* $t = a$, *simp*)
apply (*cut-tac aug-pm-aug-pm-minus*[*of a i z*])
apply (*cut-tac addition-inc-add0*[*of* {$_i-$ *a*} *f*])
apply (*cut-tac* $A =$ {$_i-$ *a*} **and** $B =$ *aug-pm-set z i* {*a*} **in**
addition-set-mono[*of - - f*], *simp add*:*aug-pm-set-def*)
apply (*frule subsetD*[*of* {$_i-$ *a*} *addition-set f* {$_i-$ *a*} $_i-$ *a*], *simp*,
frule subsetD[*of addition-set f* {$_i-$ *a*}
addition-set f (*aug-pm-set z i* {*a*})], *assumption*+)
apply (*frule inv-aug-addition*[*of f z i* {*a*}], *assumption*+)
apply (*drule-tac* $x = a$ **in** *bspec*, *assumption*,
simp, *assumption*+)

apply (*case-tac* $t = z$, *simp*,
cut-tac $n = n$ **in** *addition-inc-add*[*of f aug-pm-set z i* {*a*}],
frule-tac $c =$ $_i-$ *a* **and** $A =$ *add-set f* (*aug-pm-set z i* {*a*}) *n* **in**
subsetD[*of - addition-set f* (*aug-pm-set z i* {*a*})], *assumption*+)
apply (*simp add*:*aug-commute*)

apply (*simp add*:*zeroA-def*)
apply (*subgoal-tac* $_i-$ *a* ∈ *add-set f* {$_i-$ *a*} *0*, *blast*, *simp*)

apply *simp*
apply (*cut-tac assoc-bpp-restrict*[*of* {$_i-$ *a*} *aug-pm-set z i* {*a*}],
frule bpp-closed1[*of* {$_i-$ *a*} *f* $_i-$ *a 0* $_i-$ *a 0*], *simp*, *simp*, *blast*)
apply (*simp add*:*aug-pm-set-def*, *assumption*+)

apply *simp*
apply (*case-tac t* = *a*, *simp*)
apply (*frule-tac n* = *m1* − *Suc 0* **in** *fag-single-additionTr1*[*of f z i* $_i-$ *a*],
assumption+, *simp*)
apply (*thin-tac s* ∈ *add-set f* (*aug-pm-set z i* {*a*}) *n*,
drule-tac a = *s* **in** *forall-spec*, *assumption*,
simp add:*ipp-cond1-def*, *blast*)

apply (*case-tac t* = *z*, *simp*,
cut-tac n = *n* **in** *addition-inc-add*[*of f aug-pm-set z i* {*a*}],
frule-tac c = *s* **and** *A* = *add-set f* (*aug-pm-set z i* {*a*}) *n* **in**
subsetD[*of* - *addition-set f* (*aug-pm-set z i* {*a*})], *assumption*+)
apply (*frule-tac x* = *s* **and** *y* = *z* **in** *aug-commute*[*of f z i* {*a*}], *assumption*+,
simp)
apply (*simp add*:*zeroA-def*, *blast*)

apply *simp*
apply (*cut-tac assoc-bpp-restrict*[*of* {$_i-$ *a*} *aug-pm-set z i* {$_i-$ *a*} *f*],
frule-tac x = *s* **and** *n* = *m1* **and** *y* = $_i-$ *a* **in**
bpp-closed1[*of* {$_i-$ *a*} *f* - - - *0*], *assumption*, *simp*, *blast*,
simp add:*aug-pm-set-def*, *assumption*)

apply *simp*
apply (*case-tac t* = *a*, *simp*)
apply (*simp add*:*zeroA-def*)
apply (*subgoal-tac a* ∈ *add-set f* {*a*} *0*, *blast*, *simp*)

apply (*case-tac t* = *z*, *simp*, *simp add*:*zeroA-def*)

apply *simp*
apply (*frule subsetD*[*of aug-pm-set z i* {*a*}
addition-set f (*aug-pm-set z i* {*a*}) $_i-$ *a*], *assumption*+,
simp add:*zeroA-def*,
subgoal-tac $_i-$ *a* ∈ *add-set f* {$_i-$ *a*} *0*, *blast*, *simp*)

apply (*simp add*:*aug-pm-set-def*)
apply *assumption*
done

lemma *fag-single-elem*:⟦*commute-bpp f* (*aug-pm-set z i* {*a*});
assoc-bpp (*aug-pm-set z i* {*a*}) *f*; *ipp-cond1* {*a*} *i*; *ipp-cond2 z* {*a*} *i f*;

ipp-cond3 z i; *inv-ipp z i f* {*a*}; *commute-bpp f* (*aug-pm-set z i* {*a*});
zeroA z i f {*a*} *z*; *x* ∈ *addition-set f* (*aug-pm-set z i* {*a*})⟧ ⟹
(∃ *n1*. *x* ∈ *add-set f* {*a*} *n1*) ∨ (∃ *m1*. *x* ∈ *add-set f* {*a*} −*i* *m1*) ∨ *x* = *z*
apply (*simp add*:*addition-set-def*)
apply (*erule exE*, *erule conjE*, *erule exE*, *simp*)
apply (*simp add*:*fag-single-elemTr*)
done

lemma *add-set-single1Tr*:⟦*commute-bpp f* (*aug-pm-set z i* {*a*});
assoc-bpp (*aug-pm-set z i* {*a*}) *f*; *ipp-cond1* {*a*} *i*; *ipp-cond2 z* {*a*} *i f*;
ipp-cond3 z i; *inv-ipp z i f* {*a*}; *commute-bpp f* (*aug-pm-set z i* {*a*});
zeroA z i f {*a*} *z*⟧ ⟹
∀ *x y*. *x* ∈ *add-set f* {*a*} *n* ∧ *y* ∈ *add-set f* {*a*} *n* ⟶ *x* = *y*
apply (*induct-tac n*)
apply ((*rule allI*)+, *rule impI*, *erule conjE*)
apply *simp*
apply ((*rule allI*)+, *rule impI*, *erule conjE*, *simp*, (*erule bexE*)+)
apply (*drule-tac x* = *s* **in** *spec*,
drule-tac a = *sa* **in** *forall-spec*, *simp*)

apply *simp*
done

lemma *add-set-single-nonempty1*:⟦*commute-bpp f* (*aug-pm-set z i* {*a*});
assoc-bpp (*aug-pm-set z i* {*a*}) *f*; *ipp-cond1* {*a*} *i*; *ipp-cond2 z* {*a*} *i f*;
ipp-cond3 z i; *inv-ipp z i f* {*a*}; *commute-bpp f* (*aug-pm-set z i* {*a*});
zeroA z i f {*a*} *z*⟧ ⟹ ∃ *x*. *x*∈*add-set f* {*a*} *n*
apply (*induct-tac n*)
apply *simp*
apply (*erule exE*)
apply *simp* **apply** *blast*
done

lemma *add-set-single-nonempty2*:⟦*commute-bpp f* (*aug-pm-set z i* {*a*});
assoc-bpp (*aug-pm-set z i* {*a*}) *f*; *ipp-cond1* {*a*} *i*; *ipp-cond2 z* {*a*} *i f*;
ipp-cond3 z i; *inv-ipp z i f* {*a*}; *commute-bpp f* (*aug-pm-set z i* {*a*});
zeroA z i f {*a*} *z*⟧ ⟹ ∃ *x*. *x*∈*add-set f* {*a*} −*i* *a* *n*
apply (*simp add*:*aug-pm-aug-pm-minus*,
frule ipp-cond1-minus[*of a i*],
frule ipp-cond2-minus[*of a i z f*], *assumption*+,
frule inv-ipp-minus[*of a i z f*], *assumption*+,
frule zeroA-minus[*of a i z f z*], *assumption*+)
apply (*simp add*:*add-set-single-nonempty1* [*of f z i* −*i* *a n*])
done

lemma *add-set-single1*:⟦*commute-bpp f* (*aug-pm-set z i* {*a*});
assoc-bpp (*aug-pm-set z i* {*a*}) *f*; *ipp-cond1* {*a*} *i*; *ipp-cond2 z* {*a*} *i f*;
ipp-cond3 z i; *inv-ipp z i f* {*a*}; *commute-bpp f* (*aug-pm-set z i* {*a*});
zeroA z i f {*a*} *z*; *x* ∈ *add-set f* {*a*} *n*; *y* ∈ *add-set f* {*a*} *n*⟧ ⟹ *x* = *y*

apply (*frule add-set-single1Tr* [*of f z i a n*], *assumption*+)
apply *blast*
done

lemma *add-set-single2*:⟦*commute-bpp f* (*aug-pm-set z i* {*a*});
assoc-bpp (*aug-pm-set z i* {*a*}) *f*; *ipp-cond1* {*a*} *i*; *ipp-cond2 z* {*a*} *i f*;
ipp-cond3 z i; *inv-ipp z i f* {*a*}; *commute-bpp f* (*aug-pm-set z i* {*a*});
zeroA z i f {*a*} *z*; *x* ∈ *add-set f* {$_i$− *a*} *n*; *y* ∈ *add-set f* {$_i$− *a*} *n*⟧ ⟹
x = *y*
apply (*simp add*:*aug-pm-aug-pm-minus*)
apply (*frule ipp-cond1-minus*[*of a i*])
apply (*frule ipp-cond2-minus*[*of a i z f*], *assumption*+)
apply (*frule inv-ipp-minus*[*of a i z f*], *assumption*+)
apply (*frule zeroA-minus*[*of a i z f z*], *assumption*+)
apply (*rule add-set-single1* [*of f z i* $_i$− *a - n -*], *assumption*+)
done

lemma *fag-single-additionTr4*:⟦*commute-bpp f* (*aug-pm-set z i* {*a*});
assoc-bpp (*aug-pm-set z i* {*a*}) *f*; *ipp-cond1* {*a*} *i*; *ipp-cond2 z* {*a*} *i f*;
ipp-cond3 z i; *inv-ipp z i f* {*a*}; *commute-bpp f* (*aug-pm-set z i* {*a*});
zeroA z i f {*a*} *z* ⟧ ⟹
∀ *s t*. *s* ∈ *add-set f* {*a*} *n* ∧ *t* ∈ *add-set f* {$_i$− *a*} *n*⟶ *s* $_f$+ *t* = *z*
apply (*cut-tac a-in-aug-pm-set*[*of a* {*a*} *z i*], *simp*,
cut-tac addition-inc-add0[*of aug-pm-set z i* {*a*} *f*],
frule subsetD[*of aug-pm-set z i* {*a*}
addition-set f (*aug-pm-set z i* {*a*}) *a*], *assumption*+)

apply (*induct-tac n*)
apply *simp*
apply (*frule inv-aug-addition* [*of f z i* {*a*}], *assumption*+)
apply (*frule aug-ipp-closed* [*of f z i* {*a*} *a*], *assumption*+)
apply (*frule fag-single-additionTr2* [*of f z i a a*], *assumption*+)
apply (*simp*, *assumption*)
apply ((*rule allI*)+, *rule impI*, *erule conjE*)
apply *simp*
apply (*erule bexE*)+
apply (*frule aug-ipp-closed* [*of f z i* {*a*} *a*], *assumption*+)
apply *simp*
apply (*drule-tac x* = *sa* **in** *spec*,
drule-tac a = *sb* **in** *forall-spec*, *simp*)
apply (*cut-tac n* = *n* **in** *addition-inc-add*[*of f* {*a*}],
cut-tac addition-set-mono[*of* {*a*} *aug-pm-set z i* {*a*} *f*],
frule-tac c = *sa* **and** *A* = *add-set f* {*a*} *n* **in** *subsetD*[*of -
addition-set f* {*a*}], *assumption*+,
frule-tac c = *sa* **in** *subsetD*[*of addition-set f* {*a*}
addition-set f (*aug-pm-set z i* {*a*})], *assumption*+)
apply (*cut-tac n* = *n* **in** *addition-inc-add*[*of f* {$_i$− *a*}],
cut-tac addition-set-mono[*of* {$_i$− *a*} *aug-pm-set z i* {$_i$− *a*} *f*],
frule-tac c = *sb* **and** *A* = *add-set f* {$_i$− *a*} *n* **in** *subsetD*[*of -*

addition-set f $\{_i-\ a\}$], *assumption*+,
frule-tac c = sb **in** *subsetD*[*of addition-set f* $\{_i-\ a\}$
addition-set f (*aug-pm-set z i* $\{_i-\ a\}$)], *assumption*+)
apply (*simp add*:*aug-pm-aug-pm-minus*[*THEN sym*])
apply (*frule-tac x = sb* **in** *aug-bpp-closed* [*of z i* {*a*} *f* - $_i-$ *a*],
assumption+)
apply (*frule-tac x = sa* **and** *y = a* **and** *z = sb* $_f+$ $_i-$ *a* **in**
addition-assoc [*of aug-pm-set z i* {*a*} *f*], *assumption*+)
apply *simp* **apply** (*thin-tac sa* $_f+$ *a* $_f+$ (*sb* $_f+$ $_i-$ *a*) = *sa* $_f+$ (*a* $_f+$ (*sb* $_f+$ $_i-$ *a*)))
apply (*frule-tac x1 = a* **and** *y1 = sb* **and** *z1* = $_i-$ *a* **in**
addition-assoc [*THEN sym*, *of aug-pm-set z i* {*a*} *f*], *assumption*+)
apply *simp*
apply (*frule-tac y = sb* **in** *aug-commute* [*of f z i* {*a*} *a*],
assumption+, *simp*)
apply (*frule-tac x = sb* **and** *y = a* **and** *z* = $_i-$ *a* **in**
addition-assoc [*of aug-pm-set z i* {*a*} *f*], *assumption*+)
apply *simp* **apply** (*thin-tac a* $_f+$ (*sb* $_f+$ $_i-$ *a*) = *sb* $_f+$ (*a* $_f+$ $_i-$ *a*))
apply (*thin-tac a* $_f+$ *sb* = *sb* $_f+$ *a*,
thin-tac sb $_f+$ *a* $_f+$ $_i-$ *a* = *sb* $_f+$ (*a* $_f+$ $_i-$ *a*),
thin-tac sb $_f+$ $_i-$ *a* ∈ *addition-set f* (*aug-pm-set z i* {*a*}))
apply (*frule-tac y* = $_i-$ *a* **in** *aug-commute* [*of f z i* {*a*} *a*],
assumption+, *simp*)
apply (*frule-tac x = sa* **and** *y = a* **and** *z = sb* $_f+$ $_i-$ *a* **in**
addition-assoc [*of aug-pm-set z i* {*a*} *f*], *assumption*+)
apply (*frule-tac x = sb* **in** *aug-bpp-closed* [*of z i* {*a*} *f* - $_i-$ *a*],
assumption+)
apply (*thin-tac sa* $_f+$ *a* $_f+$ (*sb* $_f+$ $_i-$ *a*) = *sa* $_f+$ (*a* $_f+$ (*sb* $_f+$ $_i-$ *a*)))
apply (*thin-tac a* $_f+$ $_i-$ *a* = $_i-$ *a* $_f+$ *a*)
apply (*frule inv-aug-addition* [*of f z i* {*a*}], *assumption*+)
apply (*drule-tac x = a* **in** *bspec*, *assumption*, *simp*)

apply (*frule-tac x = sb* **and** *y = z* **in** *aug-commute* [*of f z i* {*a*}],
assumption+)
apply (*simp add*:*addition-set-inc-z*)
apply (*frule-tac x = sb* **and** *y = z* **in** *aug-commute* [*of f z i* {*a*}],
assumption+, *simp add*:*addition-set-inc-z*, *simp*)
apply (*simp add*:*zeroA-def*)
apply (*rule subsetI*, *simp add*:*aug-pm-set-def minus-set-def*)
apply (*simp add*:*aug-pm-set-def*)
apply *simp*
done

lemma *fag-single-additionTr4-1*:⟦*commute-bpp f* (*aug-pm-set z i* {*a*});
assoc-bpp (*aug-pm-set z i* {*a*}) *f*; *ipp-cond1* {*a*} *i*; *ipp-cond2 z* {*a*} *i f*;
ipp-cond3 z i; *inv-ipp z i f* {*a*}; *commute-bpp f* (*aug-pm-set z i* {*a*});
zeroA z i f {*a*} *z*;*s* ∈ *add-set f* {*a*} *n*; *t* ∈ *add-set f* $\{_i-\ a\}$ *n* ⟧ ⟹
s $_f+$ *t = z*
apply (*frule fag-single-additionTr4*[*of f z i a n*], *assumption*+)

apply *blast*
done

lemma *fag-single-additionTr5*:⟦*assoc-bpp* (*aug-pm-set z i* {*a*}) *f*;
ipp-cond1 {*a*} *i*; *ipp-cond2 z* {*a*} *i f*; *ipp-cond3 z i*; *inv-ipp z i f* {*a*};
commute-bpp f (*aug-pm-set z i* {*a*}); *zeroA z i f* {*a*} *z*⟧ ⟹
∀ *m*. *m* < *Suc n* ⟶ (*THE x*. *x* ∈ *add-set f* {*a*} (*Suc n*)) $_f$+
(*THE x*. *x* ∈ *add-set f* {$_i$− *a*} *m*) = (*THE x*. *x* ∈ *add-set f* {*a*} (*n* − *m*))
apply (*cut-tac a-in-aug-pm-set*[*of a* {*a*} *z i*],
cut-tac addition-inc-add0[*of aug-pm-set z i* {*a*} *f*],
frule subsetD[*of aug-pm-set z i* {*a*}
addition-set f (*aug-pm-set z i* {*a*}) *a*], *assumption*+)
prefer *2* **apply** *simp*
apply (*cut-tac aug-addition-inc-z*[*of z f i* {*a*}])
apply (*frule aug-ipp-closed* [*of f z i* {*a*} *a*], *assumption*+)

apply (*induct-tac n*)
apply (*rule allI*, *rule impI*, *simp*)
apply (*simp add*: *addition-assoc* [*of aug-pm-set z i* {*a*} *f a a* $_i$− *a*])
apply (*frule fag-single-additionTr2* [*of f z i a a*], *assumption*+)
apply *simp* **apply** *simp*
apply (*thin-tac a* $_f$+ $_i$− *a* = *z*)
apply (*simp add*:*aug-commute* [*of f z i* {*a*} *a z*])
apply (*simp add*:*zeroA-def*)

apply (*rule allI*, *rule impI*)
apply (*subgoal-tac* (*THE x*. *x* ∈ *add-set f* {*a*} (*Suc* (*Suc n*))) =
(*THE x*. *x* ∈ *add-set f* {*a*} (*Suc n*)) $_f$+ *a*)
apply (*simp del*:*add-set-Suc*)
apply (*frule-tac m* = *m* **and** *n* = *Suc* (*Suc n*) **in** *Suc-leI*,
thin-tac Suc m ≤ *Suc* (*Suc n*))
apply (*case-tac Suc m* = *Suc* (*Suc n*))
apply (*frule-tac x* = *m* **and** *y* = *Suc n* **in** *Suc-inject*,
thin-tac Suc m = *Suc* (*Suc n*))
apply (*rotate-tac* −*1*, *frule sym*, *thin-tac m* = *Suc n*,
thin-tac ∀ *m*. *m* < *Suc n* ⟶ (*THE x*. *x* ∈ *add-set f* {*a*} (*Suc n*)) $_f$+
(*THE x*. *x* ∈ *add-set f* {$_i$− *a*} *m*) = (*THE x*. *x* ∈ *add-set f* {*a*} (*n* − *m*)),
thin-tac (*THE x*. *x* ∈ *add-set f* {*a*} (*Suc* (*Suc n*))) =
(*THE x*. *x* ∈ *add-set f* {*a*} (*Suc n*)) $_f$+ *a*)
apply *simp*
apply (*subgoal-tac* (*THE x*. *x* ∈ *add-set f* {*a*} *m*) ∈
addition-set f (*aug-pm-set z i* {*a*}))
apply (*subgoal-tac* (*THE x*. *x* ∈ *add-set f* {$_i$− *a*} *m*) ∈
addition-set f (*aug-pm-set z i* {*a*}))
apply (*frule-tac x* = *THE x*. *x* ∈ *add-set f* {*a*} *m* **and** *y* = *a* **and**
z = *THE x*. *x* ∈ *add-set f* {$_i$− *a*} *m* **in**
addition-assoc [*of aug-pm-set z i* {*a*} *f*], *assumption*+)
apply *simp*
apply (*thin-tac* (*THE x*. *x* ∈ *add-set f* {*a*} *m*) $_f$+ *a* $_f$+

```
      (THE x. x ∈ add-set f {i− a} m) =
     (THE x. x ∈ add-set f {a} m) f+ (a f+ (THE x. x ∈ add-set f {i− a} m)))
apply (frule-tac x = a and y = THE x. x ∈ add-set f {i− a} m in
       aug-commute [of f z i {a}], assumption+) apply simp
apply (thin-tac a f+ (THE x. x ∈ add-set f {i− a} m) =
       (THE x. x ∈ add-set f {i− a} m) f+ a)
apply (frule-tac x1 = THE x. x ∈ add-set f {a} m and
        y1 = THE x. x ∈ add-set f {i− a} m and z1 = a in
       addition-assoc[THEN sym, of aug-pm-set z i {a} f], assumption+)
apply simp
apply (frule-tac n = m in fag-single-additionTr4 [of f z i a],
       assumption+)
apply (subgoal-tac (THE x. x ∈ add-set f {a} m) f+
       (THE x. x ∈ add-set f {i− a} m) = z)
prefer 2
apply (thin-tac (THE x. x ∈ add-set f {a} m)
           ∈ addition-set f (aug-pm-set z i {a}),
       thin-tac (THE x. x ∈ add-set f {i− a} m)
           ∈ addition-set f (aug-pm-set z i {a}),
       thin-tac (THE x. x ∈ add-set f {a} m) f+
       ((THE x. x ∈ add-set f {i− a} m) f+ a) =
      (THE x. x ∈ add-set f {a} m) f+ (THE x. x ∈ add-set f {i− a} m) f+ a)
apply (subgoal-tac (THE x. x ∈ add-set f {a} m) ∈ add-set f {a} m)
apply (subgoal-tac (THE x. x ∈ add-set f {i− a} m) ∈ add-set f {i− a} m)
 apply simp
 apply (thin-tac ∀ s t. s ∈ add-set f {a} m ∧ t ∈ add-set f {i− a} m ⟶
       s f+ t = z,
        thin-tac (THE x. x ∈ add-set f {a} m) ∈ add-set f {a} m)
apply (rule theI′) apply (rule ex-ex1I)
 apply (simp add:add-set-single-nonempty2)
 apply (simp add:add-set-single2)
apply (rule theI′) apply (rule ex-ex1I)
 apply (simp add:add-set-single-nonempty1)
 apply (simp add:add-set-single1)
 apply (thin-tac ∀ s t. s ∈ add-set f {a} m ∧ t ∈ add-set f {i− a} m ⟶
             s f+ t = z)
 apply simp apply (simp add:zeroA-def)
apply (subgoal-tac (THE x. x ∈ add-set f {i− a} m) ∈ add-set f {i− a} m)
apply (subgoal-tac {i− a} ⊆ (aug-pm-set z i {a}))
apply (frule-tac n = m in add-set-mono[of {i− a} aug-pm-set z i {a} f])
apply (frule-tac A = add-set f {i− a} m and
       B = add-set f (aug-pm-set z i {a}) m and
       c = THE x. x ∈ add-set f {i− a} m in subsetD, assumption+)
apply (subgoal-tac add-set f (aug-pm-set z i {a}) m ⊆
                          addition-set f (aug-pm-set z i {a}))
apply (simp add:subsetD) apply (simp add:addition-inc-add)
apply (rule subsetI) apply (simp add:aug-pm-set-def minus-set-def)
apply (thin-tac (THE x. x ∈ add-set f {a} m)
           ∈ addition-set f (aug-pm-set z i {a}))
```

apply (*rule theI′*) **apply** (*rule ex-ex1I*)
apply (*simp add*:*add-set-single-nonempty2*)
apply (*simp add*:*add-set-single2*)
apply (*subgoal-tac* (*THE x*. *x* ∈ *add-set f* {*a*} *m*) ∈ *add-set f* {*a*} *m*)
apply (*subgoal-tac* {*a*} ⊆ (*aug-pm-set z i* {*a*}))
apply (*frule-tac n* = *m* **in** *add-set-mono*[*of* {*a*} *aug-pm-set z i* {*a*} *f*])
apply (*frule-tac A* = *add-set f* {*a*} *m* **and**
B = *add-set f* (*aug-pm-set z i* {*a*}) *m* **and**
c = *THE x*. *x* ∈ *add-set f* {*a*} *m* **in** *subsetD*, *assumption*+)
apply (*subgoal-tac add-set f* (*aug-pm-set z i* {*a*}) *m* ⊆
addition-set f (*aug-pm-set z i* {*a*}))
apply (*simp add*:*subsetD*) **apply** (*simp add*:*addition-inc-add*)
apply (*rule subsetI*) **apply** (*simp add*:*aug-pm-set-def minus-set-def*)
apply (*rule theI′*) **apply** (*rule ex-ex1I*)
apply (*simp add*:*add-set-single-nonempty1*)
apply (*simp add*:*add-set-single1*) **apply** (*simp del*:*add-set-Suc*)
apply (*subgoal-tac* (*THE x*. *x* ∈ *add-set f* {*a*} (*Suc n*)) $_f$+
(*THE x*. *x* ∈ *add-set f* {$_i$− *a*} *m*) = (*THE x*. *x* ∈ *add-set f* {*a*} (*n* − *m*)))
prefer *2* **apply** *simp*
apply (*thin-tac* ∀ *m*. *m* < *Suc n* ⟶ (*THE x*. *x* ∈ *add-set f* {*a*} (*Suc n*)) $_f$+
(*THE x*. *x* ∈ *add-set f* {$_i$− *a*} *m*) = (*THE x*. *x* ∈ *add-set f* {*a*} (*n* − *m*)))
apply (*thin-tac* (*THE x*. *x* ∈ *add-set f* {*a*} (*Suc* (*Suc n*))) =
(*THE x*. *x* ∈ *add-set f* {*a*} (*Suc n*)) $_f$+ *a*)
apply (*subgoal-tac* (*THE x*. *x* ∈ *add-set f* {*a*} (*Suc n*)) ∈ *add-set f* {*a*} (*Suc n*))
prefer *2*
apply (*rule theI′*) **apply** (*rule ex-ex1I*)
apply (*simp del*:*add-set-Suc add*:*add-set-single-nonempty1*)
apply (*simp del*:*add-set-Suc add*:*add-set-single1*)
apply (*subgoal-tac* (*THE x*. *x* ∈ *add-set f* {$_i$− *a*} *m*) ∈ *add-set f* {$_i$− *a*} *m*)
apply (*subgoal-tac* (*THE x*. *x* ∈ *add-set f* {*a*} (*Suc n*)) ∈ *addition-set f* (*aug-pm-set z i* {*a*}))
apply (*subgoal-tac* (*THE x*. *x* ∈ *add-set f* {$_i$− *a*} *m*) ∈ *addition-set f* (*aug-pm-set z i* {*a*}))
apply (*frule-tac x* = *THE x*. *x* ∈ *add-set f* {*a*} (*Suc n*) **and** *y* = *a* **and** *z* = *THE x*. *x* ∈ *add-set f* {$_i$− *a*} *m* **in** *addition-assoc* [*of aug-pm-set z i* {*a*} *f*], *assumption*+) **apply** (*simp del*:*add-set-Suc*)
apply (*frule-tac x* = *a* **and** *y* = *THE x*. *x* ∈ *add-set f* {$_i$− *a*} *m* **in** *aug-commute* [*of f z i* {*a*}], *assumption*+) **apply** (*simp del*:*add-set-Suc*)
apply (*frule-tac x1* = *THE x*. *x* ∈ *add-set f* {*a*} (*Suc n*) **and** *y1* = *THE x*. *x* ∈ *add-set f* {$_i$− *a*} *m* **and** *z1* = *a* **in** *addition-assoc*[*THEN sym*, *of aug-pm-set z i* {*a*} *f*], *assumption*+) **apply** (*simp del*:*add-set-Suc*)
apply (*thin-tac* (*THE x*. *x* ∈ *add-set f* {*a*} (*Suc n*)) $_f$+
(*THE x*. *x* ∈ *add-set f* {$_i$− *a*} *m*) = (*THE x*. *x* ∈ *add-set f* {*a*} (*n* − *m*)),
thin-tac (*THE x*. *x* ∈ *add-set f* {*a*} (*Suc n*)) $_f$+ *a* $_f$+
(*THE x*. *x* ∈ *add-set f* {$_i$− *a*} *m*) = (*THE x*. *x* ∈ *add-set f* {*a*} (*n* − *m*)) $_f$+ *a*,
thin-tac (*THE x*. *x* ∈ *add-set f* {*a*} (*Suc n*)) $_f$+
((*THE x*. *x* ∈ *add-set f* {$_i$− *a*} *m*) $_f$+ *a*) = (*THE x*. *x* ∈
add-set f {*a*} (*n* − *m*)) $_f$+ *a*,
thin-tac a $_f$+ (*THE x*. *x* ∈ *add-set f* {$_i$− *a*} *m*) =

$(THE\ x.\ x \in add\text{-}set\ f\ \{_i-\ a\}\ m)\ _f+\ a)$
apply (*subgoal-tac Suc n − m = Suc* (*n − m*), *simp del*:*add-set-Suc*)
apply (*thin-tac Suc n − m = Suc* (*n − m*))
apply (*subgoal-tac* (*THE x. x ∈ add-set f* {*a*} (*Suc* (*n − m*))) ∈ *add-set f* {*a*} (*Suc* (*n − m*)))
apply (*subgoal-tac* ∃ *s*∈*add-set f* {*a*} (*n − m*). (*THE x. x* ∈*add-set f* {*a*} (*Suc* (*n − m*))) = *s* $_f$+ *a*) **prefer** *2* **apply** *simp*
apply (*subgoal-tac* ∀ *s*∈*add-set f* {*a*} (*n − m*). (*THE x. x*∈*add-set f* {*a*} (*Suc* (*n − m*))) = *s* $_f$+ *a* ⟶ (*THE x. x ∈ add-set f* {*a*} (*n − m*)) $_f$+ *a* =
(*THE x. x ∈ add-set f* {*a*} (*Suc* (*n − m*))))
apply *blast* **apply** (*thin-tac* ∃ *s*∈*add-set f* {*a*} (*n − m*).
(*THE x. x ∈ add-set f* {*a*} (*Suc* (*n − m*))) = *s* $_f$+ *a*)
apply (*rule ballI*) **apply** (*rule impI*) **apply** (*simp del*:*add-set-Suc*)
apply (*thin-tac s* $_f$+ *a* ∈ *add-set f* {*a*} (*Suc* (*n − m*)))
apply (*thin-tac* (*THE x. x ∈ add-set f* {*a*} (*Suc* (*n − m*))) = *s* $_f$+ *a*)
apply (*subgoal-tac* (*THE x. x ∈ add-set f* {*a*} (*n − m*)) = *s*)
apply *simp*
apply (*subgoal-tac* (*THE x. x ∈ add-set f* {*a*} (*n − m*)) ∈
add-set f {*a*} (*n − m*))
apply (*simp add*:*add-set-single1*)
apply (*rule theI′*) **apply** (*rule ex-ex1I*)
apply (*simp add*:*add-set-single-nonempty1*)
apply (*simp add*:*add-set-single1*)
apply (*thin-tac* (*THE x. x ∈ add-set f* {$_i-$ *a*} *m*) ∈
addition-set f (*aug-pm-set z i* {*a*}),
thin-tac (*THE x. x ∈ add-set f* {*a*} (*Suc n*)) ∈ *add-set f* {*a*} (*Suc n*),
thin-tac (*THE x. x ∈ add-set f* {$_i-$ *a*} *m*) ∈ *add-set f* {$_i-$ *a*} *m*,
thin-tac (*THE x. x ∈ add-set f* {*a*} (*Suc n*))
∈ *addition-set f* (*aug-pm-set z i* {*a*}))
apply (*rule theI′*) **apply** (*rule ex-ex1I*)
apply (*simp del*:*add-set-Suc add*:*add-set-single-nonempty1*)
apply (*simp del*:*add-set-Suc add*:*add-set-single1*)
apply (*simp add*:*Suc-diff-le*)
apply(*thin-tac* (*THE x. x ∈ add-set f* {*a*} (*Suc n*)) $_f$+
(*THE x. x ∈ add-set f* {$_i-$ *a*} *m*) = (*THE x. x ∈ add-set f* {*a*} (*n − m*)),
thin-tac (*THE x. x ∈ add-set f* {*a*} (*Suc n*)) ∈ *add-set f* {*a*} (*Suc n*),
thin-tac (*THE x. x ∈ add-set f* {$_i-$ *a*} *m*) ∈ *add-set f* {$_i-$ *a*} *m*,
thin-tac (*THE x. x ∈ add-set f* {*a*} (*Suc n*))
∈ *addition-set f* (*aug-pm-set z i* {*a*}))
apply (*subgoal-tac* {$_i-$ *a*} ⊆ (*aug-pm-set z i* {*a*}))
apply (*frule-tac A* = {$_i-$ *a*} **and** *B* = (*aug-pm-set z i* {*a*}) **and**
n = *m* **and** *f* = *f* **in** *add-set-mono*)
apply (*subgoal-tac* (*THE x. x ∈ add-set f* {$_i-$ *a*} *m*) ∈ *add-set f* {$_i-$ *a*} *m*)
apply (*cut-tac n* = *m* **in** *addition-inc-add*[*of f aug-pm-set z i* {*a*}])
apply (*simp add*:*subsetD*)+
apply (*thin-tac add-set f* {$_i-$ *a*} *m* ⊆ *add-set f* (*aug-pm-set z i* {*a*}) *m*)
apply (*rule theI′*, *rule ex-ex1I*)
apply (*simp add*:*add-set-single-nonempty2*)
apply (*simp add*:*add-set-single2*)

apply (*rule subsetI*, *simp add*:*aug-pm-set-def minus-set-def*)
apply (*thin-tac* (*THE x*. *x* ∈ *add-set f* {*a*} (*Suc n*)) $_f$+
(*THE x*. *x* ∈ *add-set f* {$_i$− *a*} *m*) = (*THE x*. *x* ∈ *add-set f* {*a*} (*n* − *m*)),
thin-tac (*THE x*. *x* ∈ *add-set f* {$_i$− *a*} *m*) ∈ *add-set f* {$_i$− *a*} *m*)
apply (*subgoal-tac* {*a*} ⊆ (*aug-pm-set z i* {*a*}))
apply (*frule-tac A* = {*a*} **and** *B* = *aug-pm-set z i* {*a*} **and**
n = *Suc n* **and** *f* = *f* **in** *add-set-mono*)
apply (*cut-tac n* = *Suc n* **in** *addition-inc-add*[*of f aug-pm-set z i* {*a*}])
apply (*simp del*:*add-set-Suc add*:*subsetD*)+
apply (*thin-tac* (*THE x*. *x* ∈ *add-set f* {*a*} (*Suc n*)) $_f$+
(*THE x*. *x* ∈ *add-set f* {$_i$− *a*} *m*) = (*THE x*. *x* ∈ *add-set f* {*a*} (*n* − *m*)),
thin-tac (*THE x*. *x* ∈ *add-set f* {*a*} (*Suc n*)) ∈ *add-set f* {*a*} (*Suc n*))
apply (*rule theI'*) **apply** (*rule ex-ex1I*)
apply (*simp add*:*add-set-single-nonempty2*)
apply (*simp add*:*add-set-single2*)
apply (*thin-tac* ∀ *m*. *m* < *Suc n* ⟶ (*THE x*. *x* ∈ *add-set f* {*a*} (*Suc n*)) $_f$+
(*THE x*. *x* ∈ *add-set f* {$_i$− *a*} *m*) = (*THE x*. *x* ∈ *add-set f* {*a*} (*n* − *m*)))
apply (*subgoal-tac* (*THE x*. *x* ∈ *add-set f* {*a*} (*Suc* (*Suc n*))) ∈
add-set f {*a*} (*Suc* (*Suc n*)))
apply (*subgoal-tac* ∃ *s*∈*add-set f* {*a*} (*Suc n*).
(*THE x*. *x* ∈ *add-set f* {*a*} (*Suc* (*Suc n*))) = *s* $_f$+ *a*)
prefer *2* **apply** *simp*
apply (*erule bexE*)
apply (*simp del*:*add-set-Suc*)
apply (*thin-tac* (*THE x*. *x* ∈ *add-set f* {*a*} (*Suc* (*Suc n*))) = *s* $_f$+ *a*)
apply (*subgoal-tac s* = (*THE x*. *x* ∈ *add-set f* {*a*} (*Suc n*)))
apply (*simp del*:*add-set-Suc*)
apply (*subgoal-tac* (*THE x*. *x* ∈ *add-set f* {*a*} (*Suc n*)) ∈
add-set f {*a*} (*Suc n*))
apply (*simp del*:*add-set-Suc add*:*add-set-single1*)
apply (*rule theI'*) **apply** (*rule ex-ex1I*)
apply (*simp del*:*add-set-Suc add*:*add-set-single-nonempty1*)
apply (*simp del*:*add-set-Suc add*:*add-set-single1*)
apply (*rule theI'*) **apply** (*rule ex-ex1I*)
apply (*simp del*:*add-set-Suc add*:*add-set-single-nonempty1*)
apply (*simp del*:*add-set-Suc add*:*add-set-single1*)
done

lemma *fag-single-additionTr5-1*:⟦*assoc-bpp* (*aug-pm-set z i* {*a*}) *f*;
ipp-cond1 {*a*} *i*; *ipp-cond2 z* {*a*} *i f*; *ipp-cond3 z i*; *inv-ipp z i f* {*a*};
commute-bpp f (*aug-pm-set z i* {*a*}); *zeroA z i f* {*a*} *z*; *m* < *Suc n*⟧ ⟹
(*THE x*. *x* ∈ *add-set f* {*a*} (*Suc n*)) $_f$+ (*THE x*. *x* ∈ *add-set f* {$_i$− *a*} *m*) =
(*THE x*. *x* ∈ *add-set f* {*a*} (*n* − *m*))
apply (*frule-tac n* = *n* **in** *fag-single-additionTr5* [*of z i a f*],
assumption+) **apply** *simp*
done

lemma *fag-single-additionTr5-2*:⟦*assoc-bpp* (*aug-pm-set z i* {*a*}) *f*;
ipp-cond1 {*a*} *i*; *ipp-cond2 z* {*a*} *i f*; *ipp-cond3 z i*; *inv-ipp z i f* {*a*};

commute-bpp f (aug-pm-set z i {a}); zeroA z i f {a} z; n < Suc m⟧ ⟹
(THE x. x ∈ add-set f {$_i$− a} (Suc m)) $_f$+ (THE x. x ∈ add-set f {a} n) =
(THE x. x ∈ add-set f {$_i$− a} (m − n))
apply (*simp del*:*add-set-Suc add*:*aug-pm-aug-pm-minus*)
apply (*frule ipp-cond1-minus*[*of a i*])
apply (*frule ipp-cond2-minus*[*of a i z f*], *assumption*+)
apply (*frule inv-ipp-minus*[*of a i z f*], *assumption*+)
apply (*frule zeroA-minus*[*of a i z f z*], *assumption*+)
apply (*subgoal-tac* $_i$− ($_i$− a) = a)
apply (*frule fag-single-additionTr5-1* [*of z i* $_i$− *a f n m*], *assumption*+)
apply (*simp del*:*add-set-Suc*)
apply (*simp add*:*ipp-cond1-def*)
done

definition
free-gen-condition :: ['*a* ⇒ '*a* ⇒ '*a*, '*a* ⇒ '*a*, '*a*, '*a*] ⇒ *bool* **where**
free-gen-condition f i a z ⟷ (∀ *n*. *z* ∉ *add-set f* {*a*} *n*)

definition
fg-elem-single :: ['*a* ⇒ '*a* ⇒ '*a*, '*a* ⇒ '*a*, '*a*, '*a*] ⇒ *int* ⇒ '*a* **where**
fg-elem-single f i a z n = (*if 0* = *n then z else*
(*if 0* < *n then* (*THE x*. *x* ∈ (*add-set f* {*a*} (*nat* (*n* − *1*))))
else (*THE x*. *x* ∈ (*add-set f* {$_i$− *a*} (*nat* (− *n* − *1*))))))

abbreviation
FGELEMSNGLE ((*5-*⊙ -,-,-) [*99*,*98*,*98*,*98*,*98*]*99*) **where**
n⊙ $a_{f,i,z}$ == *fg-elem-single f i a z n*

lemma *single-addition-pm-mem*:⟦*assoc-bpp* (*aug-pm-set z i* {*a*}) *f*;
ipp-cond1 {*a*} *i*; *ipp-cond2 z* {*a*} *i f*; *ipp-cond3 z i*; *inv-ipp z i f* {*a*};
commute-bpp f (*aug-pm-set z i* {*a*}); *zeroA z i f* {*a*} *z*⟧ ⟹
(*n*⊙ $a_{f,i,z}$) ∈ *addition-set f* (*aug-pm-set z i* {*a*})
apply (*case-tac n* = *0*)
apply (*simp add*:*fg-elem-single-def*) **apply** (*simp add*:*aug-addition-inc-z*)
apply (*frule-tac non-zero-int* [*of n*])
apply (*case-tac 0* < *n*)
apply (*simp add*:*fg-elem-single-def*)
apply (*subgoal-tac* (*THE x*. *x* ∈ *add-set f* {*a*} (*nat* (*n* − *1*))) ∈
add-set f {*a*} (*nat* (*n* − *1*)))
apply (*subgoal-tac add-set f* {*a*} (*nat* (*n* − *1*)) ⊆ *addition-set f* {*a*})
apply (*subgoal-tac addition-set f* {*a*}
⊆ *addition-set f* (*aug-pm-set z i* {*a*}))
apply (*simp add*:*subsetD*)+
apply (*rule addition-set-mono*)
apply (*rule subsetI*) **apply** (*simp add*:*aug-pm-set-def minus-set-def*)
apply (*simp add*:*addition-inc-add*)
apply (*rule theI'*)
apply (*rule ex-ex1I*)

apply (*simp add*:*add-set-single-nonempty1*)
apply (*simp add*:*add-set-single1*) **apply** (*thin-tac* $n \neq 0$)
apply *simp*
apply (*simp add*:*fg-elem-single-def*)
apply (*subgoal-tac* (*THE x*. $x \in$ *add-set f* $\{_i-$ *a*$\}$ (*nat* (− *n* − *1*))) ∈
add-set f $\{_i-$ *a*$\}$ (*nat* (− *n* − *1*)))
apply (*subgoal-tac add-set f* $\{_i-$ *a*$\}$ (*nat* (− *n* − *1*)) ⊆
addition-set f $\{_i-$ *a*$\}$)
apply (*subgoal-tac addition-set f* $\{_i-$ *a*$\}$ ⊆
addition-set f (*aug-pm-set z i* {*a*}))
apply (*simp add*:*subsetD*)+
apply (*rule addition-set-mono*)
apply (*rule subsetI*) **apply** (*simp add*:*aug-pm-set-def minus-set-def*)
apply (*simp add*:*addition-inc-add*)
apply (*rule theI'*)
apply (*rule ex-ex1I*)
apply (*simp add*:*add-set-single-nonempty2*)
apply (*simp add*:*add-set-single2*)
done

lemma *assoc-aug-assoc*:*assoc-bpp* (*aug-pm-set z i* {*a*}) *f* ⟹ *assoc-bpp* {*a*} *f*
apply (*simp add*:*assoc-bpp-def*)
apply (*rule ballI*)+
apply (*subgoal-tac* {*a*} ⊆ *aug-pm-set z i* {*a*})
apply (*frule addition-set-mono*[*of* {*a*} *aug-pm-set z i* {*a*} *f*])
apply (*frule-tac c* = *aa* **in** *subsetD*[*of addition-set f* {*a*}
addition-set f (*aug-pm-set z i* {*a*})], *assumption*+)
apply (*frule-tac c* = *b* **in** *subsetD*[*of addition-set f* {*a*}
addition-set f (*aug-pm-set z i* {*a*})], *assumption*+)
apply (*frule-tac c* = *c* **in** *subsetD*[*of addition-set f* {*a*}
addition-set f (*aug-pm-set z i* {*a*})], *assumption*+)
apply *simp*
apply (*rule subsetI*)
apply (*simp add*:*aug-pm-set-def minus-set-def*)
done

lemma *single-addition-posTr*:⟦*commute-bpp f* (*aug-pm-set z i* {(*a*::′*a*)});
assoc-bpp (*aug-pm-set z i* {*a*}) *f*; *ipp-cond1* {*a*} *i*; *ipp-cond2 z* {*a*} *i f*;
ipp-cond3 z i; *inv-ipp z i f* {*a*}; *commute-bpp f* (*aug-pm-set z i* {*a*});
zeroA z i f {*a*} *z*; *0* < (*n*::*int*); *0* < (*m*::*int*)⟧ ⟹
(*THE x*. $x \in$ *add-set f* {*a*} (*nat* (*n* − *1*))) $_f+$
(*THE x*. $x \in$ *add-set f* {*a*} (*nat* (*m* − *1*))) =
(*THE x*. $x \in$ *add-set f* {*a*} (*nat* (*n* + *m* − *1*)))
apply (*subgoal-tac* (*THE x*. $x \in$ *add-set f* {*a*} (*nat* (*n* − *1*))) ∈
add-set f {*a*} (*nat* (*n* − *1*)))
apply (*subgoal-tac* (*THE x*. $x \in$ *add-set f* {*a*} (*nat* (*m* − *1*)))
∈
add-set f {*a*} (*nat* (*m* − *1*)))
apply (*subgoal-tac* (*THE x*. $x \in$ *add-set f* {*a*} (*nat* (*n* + *m* − *1*)))

$\in$ *add-set f* {*a*} (*nat* (*n* + *m* − *1*)))
apply (*frule assoc-aug-assoc* [*of z i a f*])
apply (*frule-tac x* = *THE x*. *x* ∈ *add-set f* {*a*} (*nat* (*n* − *1*)) **and**
y = *THE x*. *x* ∈ *add-set f* {*a*} (*nat* (*m* − *1*)) **in**
bpp-closed1 [*of* {*a*} *f* - *nat* (*n* − *1*) - *nat* (*m* − *1*)], *assumption*+)
apply (*subgoal-tac nat* (*n* − *1*) + *nat* (*m* − *1*) + *Suc 0* = *nat* (*n* + *m* − *1*))
apply (*simp del*:*add-set-Suc*)
apply (*simp add*:*add-set-single1*)
prefer *2*
apply (*thin-tac* (*THE x*. *x* ∈ *add-set f* {*a*} (*nat* (*n* − *1*)))
∈ *add-set f* {*a*} (*nat* (*n* − *1*)))
apply (*thin-tac* (*THE x*. *x* ∈ *add-set f* {*a*} (*nat* (*m* − *1*)))
∈ *add-set f* {*a*} (*nat* (*m* − *1*)))
apply (*rule theI'*)
apply (*rule ex-ex1I*)
apply (*simp add*:*add-set-single-nonempty1*)
apply (*simp add*:*add-set-single1*)
prefer *2*
apply (*rule theI'*)
apply (*rule ex-ex1I*)
apply (*simp add*:*add-set-single-nonempty1*)
apply (*simp add*:*add-set-single1*)
prefer *2*
apply (*rule theI'*)
apply (*rule ex-ex1I*)
apply (*simp add*:*add-set-single-nonempty1*)
apply (*simp add*:*add-set-single1*)
apply (*rule int-nat-add*, *assumption*+)
done

lemma *single-addition-pos*:⟦*commute-bpp f* (*aug-pm-set z i* {(*a*::'*a*)});
assoc-bpp (*aug-pm-set z i* {*a*}) *f*; *ipp-cond1* {*a*} *i*; *ipp-cond2 z* {*a*} *i f*;
ipp-cond3 z i; *inv-ipp z i f* {*a*}; *commute-bpp f* (*aug-pm-set z i* {*a*});
zeroA z i f {*a*} *z*; *0* < (*n*::*int*); *0* < (*m*::*int*)⟧ ⟹
(*n*⊙ $a_{f,i,z}$) $_f$+ (*m*⊙ $a_{f,i,z}$) = (*n* + *m*)⊙ $a_{f,i,z}$
apply (*frule-tac single-addition-posTr* [*of f z i a n m*], *assumption*+)
apply (*simp add*:*fg-elem-single-def*)
done

lemma *single-addition-neg*:⟦*commute-bpp f* (*aug-pm-set z i* {(*a*::'*a*)});
assoc-bpp (*aug-pm-set z i* {*a*}) *f*; *ipp-cond1* {*a*} *i*; *ipp-cond2 z* {*a*} *i f*;
ipp-cond3 z i; *inv-ipp z i f* {*a*}; *commute-bpp f* (*aug-pm-set z i* {*a*});
zeroA z i f {*a*} *z*; (*n*::*int*) < *0*; (*m*::*int*) < *0* ⟧ ⟹
(*n*⊙ $a_{f,i,z}$) $_f$+ (*m*⊙ $a_{f,i,z}$) = (*n* + *m*)⊙ $a_{f,i,z}$
apply (*simp add*:*fg-elem-single-def*)
apply (*simp del*:*add-set-Suc add*:*aug-pm-aug-pm-minus*)
apply (*frule ipp-cond1-minus*[*of a i*])
apply (*frule ipp-cond2-minus*[*of a i z f*], *assumption*+)
apply (*frule inv-ipp-minus*[*of a i z f*], *assumption*+)

apply (*frule zeroA-minus*[*of a i z f z*], *assumption*+)
apply (*frule single-addition-posTr* [*of f z i* $_i$− *a* − *n* − *m*], *assumption*+)
apply *simp*+
done

lemma *single-addition-zero*:⟦*commute-bpp f* (*aug-pm-set z i* {(*a*::′*a*)});
assoc-bpp (*aug-pm-set z i* {*a*}) *f*; *ipp-cond1* {*a*} *i*; *ipp-cond2 z* {*a*} *i f*;
ipp-cond3 z i; *inv-ipp z i f* {*a*}; *commute-bpp f* (*aug-pm-set z i* {*a*});
zeroA z i f {*a*} *z*⟧ ⟹ *0*⊙ $a_{f,i,z}$ = *z*
apply (*simp add*:*fg-elem-single-def*)
done

lemma *s-a-p-1*:⟦*assoc-bpp* (*aug-pm-set z i* {*a*}) *f*; *ipp-cond1* {*a*} *i*;
ipp-cond2 z {*a*} *i f*; *ipp-cond3 z i*; *inv-ipp z i f* {*a*};
commute-bpp f (*aug-pm-set z i* {*a*}); *zeroA z i f* {*a*} *z*;
m < *0*; *0* < *n*⟧ ⟹ (*n*⊙ $a_{f,i,z}$) $_f$+ (*m*⊙ $a_{f,i,z}$) = (*n* + *m*)⊙ $a_{f,i,z}$

apply (*case-tac* − *m* < *n*)
apply (*subst zminus-zadd-cancel* [*THEN sym*, *of n m*])
apply (*subgoal-tac 0* < −*m*) **apply** (*subgoal-tac 0* < *m* + *n*)
apply (*subst single-addition-pos*[*THEN sym*, *of f z i a* −*m m* + *n*],
assumption+)
apply (*simp add*:*zminus-zadd-cancel* [*of m m* + *n*])
apply (*frule single-addition-pm-mem*[*of z i a f* −*m*], *assumption*+)
apply (*frule single-addition-pm-mem*[*of z i a f m* + *n*], *assumption*+)
apply (*frule single-addition-pm-mem*[*of z i a f m*], *assumption*+)
apply (*subst addition-assoc*, *assumption*+)
apply (*simp add*:*aug-commute*[*of f z i* {*a*} (*m* + *n*)⊙ $a_{f,i,z}$ *m*⊙ $a_{f,i,z}$])
apply (*subst addition-assoc*[*THEN sym*], *assumption*+)
apply (*subgoal-tac* ((− *m*)⊙ $a_{f,i,z}$) $_f$+ (*m*⊙ $a_{f,i,z}$) = *z*) **apply** *simp*
apply (*simp add*:*zeroA-def*) **apply** (*simp add*:*add.commute*)
apply (*simp add*:*fg-elem-single-def*)
apply (*rule fag-single-additionTr4-1*[*of f z i a- nat* (− *m* − *1*)], *assumption*+)
apply (*rule theI* ′) **apply** (*rule ex-ex1I*)
apply (*simp add*:*add-set-single-nonempty1*)
apply (*simp add*:*add-set-single1*)
apply (*rule theI* ′) **apply** (*rule ex-ex1I*)
apply (*simp add*:*add-set-single-nonempty2*)
apply (*simp add*:*add-set-single2*)
apply *simp*+ **apply** (*subgoal-tac n* ≤ − *m*) **prefer** *2* **apply** *simp*
apply (*thin-tac* ¬ − *m* < *n*)
apply (*frule zle-imp-zless-or-eq*) **apply** (*thin-tac n* ≤ − *m*)
apply (*case-tac n* = −*m*)
apply (*thin-tac n* < − *m* ∨ *n* = − *m*) **apply** *simp*
apply (*subgoal-tac 0* < −*m*) **apply** (*thin-tac n* = − *m*)
apply (*simp add*:*fg-elem-single-def*)
apply (*rule fag-single-additionTr4-1* [*of f z i a - nat* (−*m* − *1*)], *assumption*+)
apply (*rule theI* ′) **apply** (*rule ex-ex1I*)
apply (*simp add*:*add-set-single-nonempty1*)

apply (*simp add*:*add-set-single1*)
apply (*rule theI'*) **apply** (*rule ex-ex1I*)
apply (*simp add*:*add-set-single-nonempty2*)
apply (*simp add*:*add-set-single2*)
apply *simp*
apply *simp*
apply (*subst zminus-zadd-cancel* [*THEN sym*, *of m n*])
apply (*subst single-addition-neg*[*THEN sym*, *of f z i a* $-n$ $n + m$], *assumption*+)
apply *simp* **apply** *simp*
apply (*frule single-addition-pm-mem*[*of z i a f* $-n$], *assumption*+)
apply (*frule single-addition-pm-mem*[*of z i a f n* + *m*], *assumption*+) **apply** (*frule single-addition-pm-mem*[*of z i a f n*], *assumption*+)
apply (*simp add*:*addition-assoc*[*THEN sym*])
apply (*subgoal-tac* $(n\bigodot a_{f,i,z})\ {}_f{+}\ ((-n)\bigodot a_{f,i,z}) = z$) **apply** (*simp add*:*zeroA-def*)
apply (*thin-tac* $(-n)\bigodot a_{f,i,z} \in$ *addition-set f* (*aug-pm-set z i* {*a*}))
apply (*thin-tac* $(n + m)\bigodot a_{f,i,z} \in$ *addition-set f* (*aug-pm-set z i* {*a*}))
apply (*thin-tac* $n\bigodot a_{f,i,z} \in$ *addition-set f* (*aug-pm-set z i* {*a*}))
apply (*simp add*:*fg-elem-single-def*)
apply (*subgoal-tac* (*THE x*. $x \in$ *add-set f* {*a*} (*nat* $(n - 1)$)) $\in$
add-set f {*a*} (*nat* $(n - 1)$))
apply (*subgoal-tac* (*THE x*. $x \in$ *add-set f* $\{_i{-}\ a\}$ (*nat* $(n - 1)$)) $\in$
add-set f $\{_i{-}\ a\}$ (*nat* $(n - 1)$))
apply (*simp add*:*fag-single-additionTr4-1*)
apply (*rule theI'*) **apply** (*rule ex-ex1I*)
apply (*simp add*:*add-set-single-nonempty2*)
apply (*simp add*:*add-set-single2*)
apply (*rule theI'*) **apply** (*rule ex-ex1I*)
apply (*simp add*:*add-set-single-nonempty1*)
apply (*simp add*:*add-set-single1*)
done

lemma *single-addition-pm*:⟦*commute-bpp f* (*aug-pm-set z i* {(*a*::'*a*)});
assoc-bpp (*aug-pm-set z i* {*a*}) *f*; *ipp-cond1* {*a*} *i*; *ipp-cond2 z* {*a*} *i f*;
ipp-cond3 z i; *inv-ipp z i f* {*a*}; *commute-bpp f* (*aug-pm-set z i* {*a*});
zeroA z i f {*a*} *z*⟧ ⟹ $(n\bigodot a_{f,i,z})\ {}_f{+}\ (m\bigodot a_{f,i,z}) = (n + m)\bigodot a_{f,i,z}$
apply (*frule single-addition-pm-mem*[*of z i a f n*], *assumption*+)
apply (*frule single-addition-pm-mem*[*of z i a f m*], *assumption*+)
apply (*case-tac* $n = 0$)
apply (*subgoal-tac* $(n\bigodot a_{f,i,z}) = z$, *simp*)
apply (*simp add*:*zeroA-def*, *subst fg-elem-single-def*, *simp*)
apply (*case-tac* $m = 0$)
apply (*simp add*:*aug-commute*)
apply (*subgoal-tac* $(0\bigodot a_{f,i,z}) = z$, *simp*)
apply (*simp add*:*zeroA-def*, *subst fg-elem-single-def*, *simp*)
apply (*frule-tac non-zero-int* [*of n*], *thin-tac* $n \neq 0$)
apply (*frule-tac non-zero-int* [*of m*], *thin-tac* $m \neq 0$)
apply (*case-tac* $0 < n$, *thin-tac* $0 < n \vee n < 0$)
apply (*case-tac* $0 < m$, *thin-tac* $0 < m \vee m < 0$)
apply (*simp add*:*single-addition-pos*)

apply (*simp*, *thin-tac* $m\bigodot a_{f,i,z} \in$ *addition-set f* (*aug-pm-set z i* {*a*}))
apply (*simp add*:*s-a-p-1*)
apply *simp*
apply (*subst aug-commute*, *assumption*+)
apply (*case-tac* $0 < m$, *thin-tac* $0 < m \vee m < 0$)
apply (*simp add*:*s-a-p-1*, *simp add*:*add.commute*)
apply *simp*
apply (*simp add*:*single-addition-neg*, *simp add*:*add.commute*)
done

lemma *single-inv*:⟦*commute-bpp f* (*aug-pm-set z i* {(*a*::'*a*)});
assoc-bpp (*aug-pm-set z i* {*a*}) *f*; *ipp-cond1* {*a*} *i*; *ipp-cond2 z* {*a*} *i f*;
ipp-cond3 z i; *inv-ipp z i f* {*a*}; *commute-bpp f* (*aug-pm-set z i* {*a*});
zeroA z i f {*a*} *z*⟧ ⟹ $_i-\ (m\bigodot a_{f,i,z}) = (-m)\bigodot a_{f,i,z}$
apply (*frule single-addition-pm-mem*[*of z i a f m*], *assumption*+,
frule single-addition-pm-mem[*of z i a f* $-$ *m*], *assumption*+,
frule single-addition-pm[*THEN sym*, *of f z i a* $-m$ *m*],
assumption+, *simp*)
apply (*simp add*:*single-addition-zero*)
apply (*subgoal-tac z* $_f+\ (_i-\ (m\bigodot a_{f,i,z}) =$
$((-\ m)\bigodot a_{f,i,z})\ _f+\ (m\bigodot a_{f,i,z})\ _f+\ (_i-\ (m\bigodot a_{f,i,z}))$) **prefer** *2* **apply** *simp*
apply (*frule aug-ipp-closed* [*of f z i* {*a*} $m\bigodot a_{f,i,z}$], *assumption*+)
apply (*thin-tac z* $= ((-\ m)\bigodot a_{f,i,z})\ _f+\ (m\bigodot a_{f,i,z})$)
apply (*simp add*:*addition-assoc*)
apply (*simp add*:*aug-commute* [*of f z i* {*a*} $(m\bigodot a_{f,i,z})\ _i-\ (m\bigodot a_{f,i,z})$])
apply (*subgoal-tac* $_i-\ (m\bigodot a_{f,i,z})\ _f+\ (m\bigodot a_{f,i,z}) = z$) **apply** *simp*
apply (*subgoal-tac z* ∈ *addition-set f* (*aug-pm-set z i* {*a*}))
prefer *2* **apply** (*simp add*:*addition-set-inc-z*)
apply (*simp add*:*aug-commute*[*of f z i* {*a*} $(-\ m)\bigodot a_{f,i,z}$ *z*])
apply (*simp add*:*zeroA-def*)
apply (*simp add*:*inv-ipp-def*)
apply (*subgoal-tac zeroA z i f* {*a*} $(_i-\ (m\bigodot a_{f,i,z})\ _f+\ (m\bigodot a_{f,i,z}))$)
prefer *2* **apply** *simp*
apply (*frule aug-zero-unique*[*of f z i* {*a*} $_i-\ (m\bigodot a_{f,i,z})\ _f+\ (m\bigodot a_{f,i,z})$])
apply (*rule aug-bpp-closed*, *assumption*+)
apply (*rule sym*) **apply** *assumption*
done

lemma *free-ag-single*:⟦*commute-bpp f* (*aug-pm-set z i* {*a*});
assoc-bpp (*aug-pm-set z i* {*a*}) *f*; *ipp-cond1* {*a*} *i*; *ipp-cond2 z* {*a*} *i f*;
ipp-cond3 z i; *inv-ipp z i f* {*a*}; *commute-bpp f* (*aug-pm-set z i* {*a*});
zeroA z i f {*a*} *z*; *free-gen-condition f i a z*; $n \neq m$⟧ ⟹
$(n\bigodot a_{f,i,z}) \neq (m\bigodot a_{f,i,z})$
apply (*rule contrapos-pp*, *simp*+)
apply (*frule single-addition-pm*[*THEN sym*, *of f z i a n* $-m$], *assumption*+)
apply *simp*
apply (*frule single-addition-pm-mem*[*of z i a f m*], *assumption*+)

apply (*frule single-addition-pm-mem*[*of z i a f − m*], *assumption*+)
apply (*simp add*:*aug-commute* [*of f z i* {*a*} ($m\bigodot a_{f,i,z}$) (− *m*)$\bigodot a_{f,i,z}$])
apply (*simp add*:*single-inv*[*THEN sym*, *of f z i a m*])
apply (*simp add*:*inv-aug-addition*)
apply (*thin-tac* $n\bigodot a_{f,i,z} = m\bigodot a_{f,i,z}$)
apply (*thin-tac* $_i-$ ($m\bigodot a_{f,i,z}$) ∈ *addition-set f* (*aug-pm-set z i* {*a*}))
apply (*thin-tac* $m\bigodot a_{f,i,z}$ ∈ *addition-set f* (*aug-pm-set z i* {*a*}))
apply (*insert int-neq-iff*[*of n m*]) **apply** *simp*
apply (*case-tac n* < *m*) **apply** (*thin-tac n* < *m* ∨ *m* < *n*)
apply (*frule single-inv* [*THEN sym*, *of f z i a n − m*], *assumption*+)
apply *simp* **apply** (*subgoal-tac* $_i-$ *z* = *z*) **apply** *simp*
apply (*thin-tac* $_i-$ *z* = *z*) **prefer** *2* **apply** (*simp add*:*ipp-cond3-def*)
apply (*subgoal-tac 0* < *m* − *n*) **prefer** *2* **apply** *simp*
apply (*thin-tac* (*n* − *m*)$\bigodot a_{f,i,z} = z$)
apply (*simp add*:*fg-elem-single-def*)
apply (*subgoal-tac* (*THE x*. *x* ∈ *add-set f* {*a*} (*nat* (*m* − *n* − *1*))) ∈ *add-set f* {*a*} (*nat* (*m* − *n* − *1*))) **apply** *simp*
apply (*thin-tac* (*THE x*. *x* ∈ *add-set f* {*a*} (*nat* (*m* − *n* − *1*))) = *z*)
apply (*simp add*:*free-gen-condition-def*)
apply (*rule theI'*) **apply** (*rule ex-ex1I*)
apply (*simp add*:*add-set-single-nonempty1*)
apply (*simp add*:*add-set-single1*) **apply** *simp*
apply (*subgoal-tac 0* < *n* − *m*) **prefer** *2* **apply** *simp*
apply (*simp add*:*fg-elem-single-def*)
apply (*subgoal-tac* (*THE x*. *x* ∈ *add-set f* {*a*} (*nat* (*n* − *m* − *1*))) ∈ *add-set f* {*a*} (*nat* (*n* − *m* − *1*))) **apply** *simp*
apply (*thin-tac* (*THE x*. *x* ∈ *add-set f* {*a*} (*nat* (*n* − *m* − *1*))) = *z*)
apply (*simp add*:*free-gen-condition-def*)
apply (*rule theI'*) **apply** (*rule ex-ex1I*)
apply (*simp add*:*add-set-single-nonempty1*)
apply (*simp add*:*add-set-single1*)
done

definition
fags-cond :: [*'a* ⇒ *'a* ⇒ *'a*, *'a*, *'a* ⇒ *'a*, *'a*] ⇒ *bool* **where**
fags-cond f z i a ⟷ *commute-bpp f* (*aug-pm-set z i* {*a*}) ∧
assoc-bpp (*aug-pm-set z i* {*a*}) *f* ∧ *ipp-cond1* {*a*} *i* ∧
ipp-cond2 z {*a*} *i f* ∧ *ipp-cond3 z i* ∧ *inv-ipp z i f* {*a*} ∧
commute-bpp f (*aug-pm-set z i* {*a*}) ∧ *zeroA z i f* {*a*} *z* ∧
free-gen-condition f i a z

lemma *fag-single-free*:⟦*fags-cond f z i a*; *n* ≠ *m*⟧ ⟹ ($n\bigodot a_{f,i,z}$) ≠ ($m\bigodot a_{f,i,z}$)
apply (*simp add*:*fags-cond-def*) **apply** (*erule conjE*)+
apply (*simp add*:*free-ag-single*)
done

lemma *fag-single-free1*:⟦*fags-cond f z i a*;($n\bigodot a_{f,i,z}$) = ($m\bigodot a_{f,i,z}$)⟧ ⟹ *n* = *m*
apply (*rule contrapos-pp*, *simp*+)
apply (*frule fag-single-free* [*of f z i a n m*], *assumption*+)

apply *simp*
done

definition
fags-carr :: [$'a \Rightarrow 'a \Rightarrow 'a$, $'a$, $'a \Rightarrow 'a$, $'a$] $\Rightarrow$ $'a$ *set* **where**
fags-carr f z i a = $\{x.\ \exists n.\ x = n \odot a_{f,i,z}\}$

definition
fags-bpp :: [$'a \Rightarrow 'a \Rightarrow 'a$, $'a$, $'a \Rightarrow 'a$, $'a$] $\Rightarrow$ $'a \Rightarrow 'a \Rightarrow 'a$ **where**
fags-bpp f z i a = ($\lambda x \in$(*fags-carr f z i a*). $\lambda y \in$(*fags-carr f z i a*).
((*THE n. x* = $n \odot a_{f,i,z}$) + (*THE m. y* = $m \odot a_{f,i,z}$)) $\odot a_{f,i,z}$)

definition
fags-ipp :: [$'a \Rightarrow 'a \Rightarrow 'a$, $'a$, $'a \Rightarrow 'a$, $'a$] $\Rightarrow$ $'a \Rightarrow 'a$ **where**
fags-ipp f z i a = ($\lambda x \in$(*fags-carr f z i a*).
(− (*THE n. x* = $n \odot a_{f,i,z}$)) $\odot a_{f,i,z}$)

lemma *fags-mem*:*fags-cond f z i a* $\Longrightarrow$ ($n \odot a_{f,i,z}$) $\in$ *fags-carr f z i a*
apply (*simp add*:*fags-carr-def*)
apply *blast*
done

lemma *fags-ippTr*:*fags-cond f z i a* $\Longrightarrow$
fags-ipp f z i a ($n \odot a_{f,i,z}$) = (− *n*) $\odot a_{f,i,z}$
apply (*subgoal-tac* ($n \odot a_{f,i,z}$) $\in$ *fags-carr f z i a*)
apply (*simp add*:*fags-ipp-def*)
apply (*subgoal-tac n* $\odot a_{f,i,z}$ = (*THE na. n* $\odot a_{f,i,z}$ = *na* $\odot a_{f,i,z}$) $\odot a_{f,i,z}$)
apply (*frule-tac m* = *THE na. n* $\odot a_{f,i,z}$ = *na* $\odot a_{f,i,z}$ **in**
fag-single-free1[*of f z i a n*], *assumption*+, *simp*)
apply (*rule theI'*, *rule ex-ex1I*, *blast*)
apply (*simp add*:*fag-single-free1* [*of f z i a*])
apply (*simp add*:*fags-mem*)
done

lemma *fags-bppTr*:*fags-cond f z i a* $\Longrightarrow$
fags-bpp f z i a ($n \odot a_{f,i,z}$) ($m \odot a_{f,i,z}$) = (*n* + *m*) $\odot a_{f,i,z}$
apply (*subgoal-tac* ($n \odot a_{f,i,z}$) $\in$ *fags-carr f z i a*)
apply (*subgoal-tac* ($m \odot a_{f,i,z}$) $\in$ *fags-carr f z i a*)
apply (*simp add*:*fags-bpp-def*)
apply (*subgoal-tac n* $\odot a_{f,i,z}$ = (*THE na. n* $\odot a_{f,i,z}$ = *na* $\odot a_{f,i,z}$) $\odot a_{f,i,z}$)
apply (*frule-tac m* = *THE na. n* $\odot a_{f,i,z}$ = *na* $\odot a_{f,i,z}$ **in**
fag-single-free1 [*of f z i a n*], *assumption*+)
apply (*subgoal-tac m* $\odot a_{f,i,z}$ = (*THE ma. m* $\odot a_{f,i,z}$ = *ma* $\odot a_{f,i,z}$) $\odot a_{f,i,z}$)
apply (*frule-tac m* = *THE ma. m* $\odot a_{f,i,z}$ = *ma* $\odot a_{f,i,z}$ **in**
fag-single-free1 [*of f z i a m*], *assumption*+)
apply *simp*
apply (*rule theI'*, *rule ex-ex1I*, *blast*)
apply (*simp add*:*fag-single-free1* [*of f z i a*])

apply (*rule theI'*, *rule ex-ex1I*, *blast*)
apply (*simp add*:*fag-single-free1* [*of f z i a*])
apply (*simp add*:*fags-carr-def*, *blast*)
apply (*simp add*:*fags-carr-def*, *blast*)
done

definition
fags :: ['*a* ⇒ '*a* ⇒ '*a*, '*a*, '*a* ⇒ '*a*, '*a*] ⇒ '*a aGroup* **where**
fags f z i a = (|*carrier* = *fags-carr f z i a*,
pop = *fags-bpp f z i a*,
mop = *fags-ipp f z i a*, *zero* = *z*|)

lemma *fags-ag*:*fags-cond f z i a* ⟹ *aGroup* (*fags f z i a*)
apply (*rule aGroup.intro*)
apply (*rule Pi-I*)+
apply (*simp add*:*fags-def fags-carr-def*)
apply ((*erule exE*)+, *simp*)
apply (*simp add*:*fags-bppTr*)
apply *blast*

apply (*simp add*:*fags-def*, *simp add*:*fags-carr-def*,
(*erule exE*)+, *simp*)
apply (*simp add*:*fags-bppTr*, *simp add*:*add.assoc*)

apply (*simp add*:*fags-def*, *simp add*:*fags-carr-def*,
(*erule exE*)+, *simp*)
apply (*simp add*:*fags-bppTr*, *simp add*:*add.commute*)

apply (*rule Pi-I*)
apply (*simp add*:*fags-def*, *simp add*:*fags-carr-def*, *erule exE*, *simp*)
apply (*simp add*:*fags-ippTr*, *blast*)

apply (*simp add*:*fags-def*, *simp add*:*fags-carr-def*, *erule exE*, *simp*)
apply (*simp add*:*fags-ippTr fags-bppTr*,
simp add:*fags-cond-def single-addition-zero*[*of f z i a*])

apply (*simp add*:*fags-def fags-carr-def*,
simp add:*fags-cond-def*, (*erule conjE*)+,
frule single-addition-zero, *assumption*+, *rotate-tac* −*1*, *frule sym*,
blast)

apply (*simp add*:*fags-def fags-carr-def*, *erule exE*)
apply (*subgoal-tac fags-bpp f z i a z aa* = *fags-bpp f z i a* (*0* ⨀ $a_{f,i,z}$) *aa*,
simp add:*fags-bppTr*,
thin-tac aa = *n* ⨀ $a_{f,i,z}$)
apply (*cut-tac single-addition-zero*[*of f z i a*], *simp*)
apply (*simp add*:*fags-cond-def*)+
done

6.3 Abelian Group generated by one element II (nonconstructive)

definition
ag-single-gen :: $[('a, 'm)\ aGroup\text{-}scheme, 'a] \Rightarrow bool$ **where**
ag-single-gen $A\ a \longleftrightarrow aGroup\ A \wedge carrier\ A = \bigcap \{H.\ asubGroup\ A\ H \wedge a \in H\}$

primrec *aSum* :: $[('a, 'm)\ aGroup\text{-}scheme, nat, 'a] \Rightarrow 'a$ **where**
aSum-0: $aSum\ A\ 0\ a = \mathbf{0}_A$
| *aSum-Suc*: $aSum\ A\ (Suc\ n)\ a = aSum\ A\ n\ a \pm_A a$

definition
sprod-n-a ::$[('a, 'm)\ aGroup\text{-}scheme, int, 'a] \Rightarrow 'a$ **where**
sprod-n-a $A\ n\ x = (if\ 0 \leq n\ then\ (aSum\ A\ (nat\ n)\ x)$
$else\ (aSum\ A\ (nat\ (-\ n))\ (-_{a\,A}\ x)))$

abbreviation
SPRODNA $((3\text{-}\triangleright\text{-}_)\ [95,95,96]95)$ **where**
$n \triangleright a_A == sprod\text{-}n\text{-}a\ A\ n\ a$

lemma (**in** *aGroup*) *asum-mem*:$a \in carrier\ A \Longrightarrow aSum\ A\ n\ a \in carrier\ A$
apply (*induct-tac n*)
apply *simp* **apply** (*simp add*:*ag-inc-zero*)
apply *simp*
apply (*rule ag-pOp-closed*, *assumption*+)
done

lemma (**in** *aGroup*) *nt-mem0*:$a \in carrier\ A \Longrightarrow n \triangleright a_A \in carrier\ A$
apply (*case-tac* $n = 0$, *simp add*:*sprod-n-a-def*)
apply (*simp add*:*ag-inc-zero*)
apply (*frule non-zero-int*[*of n*])
apply (*case-tac* $0 < n$)
apply (*simp add*:*sprod-n-a-def*)
apply (*simp add*:*asum-mem*, *thin-tac* $n \neq 0$, *simp*)
apply (*simp add*:*sprod-n-a-def*)
apply (*rule asum-mem*)
apply (*simp add*:*ag-mOp-closed*)
done

lemma (**in** *aGroup*) *nt-zero0*:$a \in carrier\ A \Longrightarrow 0 \triangleright a_A = \mathbf{0}$
apply (*simp add*:*sprod-n-a-def*)
done

lemma (**in** *aGroup*) *nt-1*:$a \in carrier\ A \Longrightarrow 1 \triangleright a_A = a$
apply (*simp add*:*sprod-n-a-def*)
apply (*simp add*:*ag-l-zero*)
done

lemma (**in** *aGroup*) *asumTr*:$a \in$ *carrier* $A \Longrightarrow$
$aSum\ A\ (n + m)\ a = aSum\ A\ n\ a \pm (aSum\ A\ m\ a)$
apply (*induct-tac* m)
apply *simp*
apply (*frule asum-mem*[*of a n*])
apply (*rule ag-r-zero* [*THEN sym*, *of aSum A n a*], *assumption*+)
apply (*subgoal-tac* $n + Suc\ na = Suc\ (n + na)$, *simp*)
apply (*rule ag-pOp-assoc*)
apply (*simp add*:*asum-mem*)+
done

lemma (**in** *aGroup*) *aSum-zero*:$a \in$ *carrier* $A \Longrightarrow aSum\ A\ n\ \mathbf{0} = \mathbf{0}$
apply (*induct-tac* n)
apply (*simp*, *simp*, *rule ag-r-zero*)
apply (*simp add*:*ag-inc-zero*)
done

lemma (**in** *aGroup*) *agsum-add1p*:$[\![\ a \in$ *carrier* $A;\ 0 \leq n;\ 0 \leq m\ ]\!] \Longrightarrow$
$(n + m) \triangleright a_A = n \triangleright a_A \pm (m \triangleright a_A)$
apply (*simp add*:*sprod-n-a-def*)
apply (*subst nat-add-distrib*[*of n m*], *assumption*+)
apply (*simp add*:*asumTr*)
done

lemma (**in** *aGroup*) *agsum-add1m*:$[\![\ a \in$ *carrier* $A;\ n < 0;\ m < 0\ ]\!] \Longrightarrow$
$(n + m) \triangleright a_A = n \triangleright a_A \pm (m \triangleright a_A)$
apply (*simp add*:*sprod-n-a-def*)
apply (*subst zdiff*)
apply (*subgoal-tac* $0 \leq -n$) **apply** (*thin-tac* $n < 0$)
apply (*subgoal-tac* $0 \leq -m$) **apply** (*thin-tac* $m < 0$)
apply (*subst nat-add-distrib* [*of* $-n\ -m$], *assumption*+)
apply (*rule asumTr*)
apply (*simp add*:*ag-mOp-closed*) **apply** *simp*+
done

lemma (**in** *aGroup*) *agsum-add2Tr*:$a \in$ *carrier* $A \Longrightarrow$
$\mathbf{0}\ = aSum\ A\ n\ a \pm (aSum\ A\ n\ (-_a\ a))$
apply (*induct-tac* n)
apply *simp*
apply (*cut-tac ag-inc-zero*)
apply (*simp add*:*ag-r-zero*[*THEN sym*])
apply *simp*
apply (*frule ag-mOp-closed*[*of a*])
apply (*frule-tac* $n = n$ **in** *asum-mem* [*of* $-_a\ a$])
apply (*frule-tac* $x = aSum\ A\ n\ (-_a\ a)$ **in** *ag-pOp-closed* [*of* $-\ -_a\ a$],
assumption)
apply (*frule-tac* $n = n$ **in** *asum-mem* [*of a*])
apply (*subst ag-pOp-assoc*[*of* - *a*], *assumption*+)

apply (*subst ag-pOp-assoc*[*THEN sym*, *of a* - $-_a$ *a*], *assumption*+)
apply (*subst ag-pOp-commute*[*of a*], *assumption*+)
apply (*subst ag-pOp-assoc*[*of* - *a* $-_a$ *a*], *assumption*+)
apply (*frule sym*) **apply** (*thin-tac* $\mathbf{0} =$ *aSum A n a* $\pm$ (*aSum A n* ($-_a$ *a*)))
apply (*simp add*:*ag-r-inv1*[*of a*])
apply (*subst ag-r-zero*, *assumption*+) **apply** *simp*
done

lemma (**in** *aGroup*) *agsum-add2p*:⟦$a \in$ *carrier A*; $0 \leq n$⟧ $\Longrightarrow$
$\mathbf{0} = n{\triangleright}a_A \pm ((-n){\triangleright}a_A)$
apply (*case-tac* $n = 0$)
apply (*simp add*:*sprod-n-a-def*)
apply (*cut-tac ag-inc-zero*)
apply (*simp add*:*ag-r-zero*[*THEN sym*])
apply (*frule non-zero-int*[*of n*], *thin-tac* $n \neq 0$, *simp*)
apply (*subgoal-tac* $-n < 0$) **prefer** *2* **apply** *simp*
apply (*simp add*:*sprod-n-a-def*)
apply (*simp add*:*agsum-add2Tr*)
done

lemma (**in** *aGroup*) *agsum-add2m*:⟦$a \in$ *carrier A*; $n < 0$⟧ $\Longrightarrow$
$\mathbf{0} = n{\triangleright}a_A \pm ((-n){\triangleright}a_A)$
apply (*simp add*:*sprod-n-a-def*)
apply (*subst ag-pOp-commute*)
apply (*rule asum-mem*)
apply (*simp add*:*ag-mOp-closed*)
apply (*rule asum-mem*, *assumption*+)
apply (*simp add*:*agsum-add2Tr*)
done

lemma (**in** *aGroup*) *agsum-add3pm*:⟦$a \in$ *carrier A*; $0 < n$; $m < 0$⟧ $\Longrightarrow$
$(n + m){\triangleright}a_A = n{\triangleright}a_A \pm (m{\triangleright}a_A)$
apply (*cut-tac less-linear*[*of* $n - m$])
apply (*case-tac* $n = -m$)
apply *simp*
apply (*subst ag-pOp-commute*)
apply (*simp add*:*nt-mem0*)+
apply (*subgoal-tac* $0{\triangleright}a_A = \mathbf{0}$) **apply** (*simp add*:*agsum-add2m*)
apply (*simp add*:*sprod-n-a-def*) **apply** *simp*
apply (*case-tac* $n < -m$)
apply (*subst zminus-zadd-cancel*[*THEN sym*, *of m n*])
apply (*subgoal-tac* $-n < 0$) **prefer** *2* **apply** *simp*
apply (*subgoal-tac* $n + m < 0$) **prefer** *2* **apply** *simp*
apply *simp*
apply (*cut-tac agsum-add1m* [*of a* $-n$ $n + m$])
apply *simp*
apply (*subst ag-pOp-assoc* [*THEN sym*])
apply (*simp add*:*nt-mem0*)+
apply (*subst agsum-add2p* [*THEN sym*, *of a n*], *assumption*+)

apply *simp*
apply (*rule ag-l-zero*[*THEN sym*, *of* $(n + m)\triangleright a_A$])
apply (*simp add*:*nt-mem0*, *assumption*) **apply** *arith* **apply** *assumption*
apply (*thin-tac* $n \neq -m$) **apply** *simp*
apply (*subst zminus-zadd-cancel*[*THEN sym*, *of n m*])
apply (*frule zminus-minus-pos*[*of m*], *frule zless-imp-zle*[*of* $0 - m$])
apply (*subgoal-tac* $0 \leq n + m$) **prefer** *2* **apply** *simp*
apply (*frule agsum-add1p* [*of a* $-m\ n + m$], *assumption*+)
apply *simp* **apply** (*thin-tac* $n\triangleright a_A = (-m)\triangleright a_A \pm (n + m)\triangleright a_A$)
apply (*subst ag-pOp-commute*[*of* $(-m)\triangleright a_A\ (n + m)\triangleright a_A$])
apply (*simp add*:*nt-mem0*)+
apply (*subst ag-pOp-assoc*)
apply (*simp add*:*nt-mem0*)+
apply (*subst ag-pOp-commute* [*of* $(-m)\triangleright a_A\ m\triangleright a_A$])
apply (*simp add*:*nt-mem0*)+
apply (*simp add*:*agsum-add2m*[*THEN sym*])
apply (*rule ag-r-zero*[*THEN sym*])
apply (*simp add*:*nt-mem0*)
done

lemma (**in** *aGroup*) *agsum-add3mp*:$[\![\ a \in carrier\ A;\ n < 0;\ 0 < m\ ]\!] \Longrightarrow$
$(n + m)\triangleright a_A = n\triangleright a_A \pm (m\triangleright a_A)$
apply (*simp add*:*add.commute*)
apply (*subst ag-pOp-commute*, (*simp add*:*nt-mem0*)+)
apply (*simp add*:*agsum-add3pm*)
done

lemma (**in** *aGroup*) *nt-sum0*:$[\![\ a \in carrier\ A\ ]\!] \Longrightarrow (n + m)\triangleright a_A = n\triangleright a_A \pm (m\triangleright a_A)$
apply (*cut-tac less-linear*[*of n 0*])
apply (*case-tac* $n < 0$)
apply (*cut-tac less-linear*[*of m 0*])
apply (*case-tac* $m < 0$, *simp add*:*agsum-add1m*, *simp*)
apply (*thin-tac* $\neg\ m < 0$)
apply (*case-tac* $m = 0$, *simp*)
apply (*simp add*:*nt-zero0*)
apply (*rule ag-r-zero*[*THEN sym*], *simp add*:*nt-mem0*)
apply (*simp*, *simp add*:*agsum-add3mp*, *simp*)
apply (*thin-tac* $\neg\ n < 0$)
apply (*case-tac* $n = 0$, *simp*, *simp add*:*nt-zero0*)
apply (*rule ag-l-zero*[*THEN sym*], *simp add*:*nt-mem0*)
apply *simp*
apply (*cut-tac less-linear*[*of m 0*])
apply (*case-tac* $m < 0$)
apply (*simp add*:*agsum-add3pm*, *simp*, *thin-tac* $\neg\ m < 0$)
apply (*case-tac* $m = 0$, *simp*, *simp add*:*nt-zero0*)
apply (*rule ag-r-zero*[*THEN sym*], *simp add*:*nt-mem0*)
apply *simp*
apply (*simp add*:*agsum-add1p*)

done

lemma (**in** *aGroup*) *nt-inv0*:$a \in carrier\ A \Longrightarrow -_a\ (n\triangleright a_A) = (-\ n)\triangleright a_A$
apply (*subgoal-tac* $(n + -n)\triangleright a_A = n\triangleright a_A \pm ((-n)\triangleright a_A)$)
prefer *2* **apply** (*rule nt-sum0*, *assumption*+) **apply** (*simp add*:*nt-zero0*)
apply (*subgoal-tac* $-_a\ (n\triangleright a_A) \pm \mathbf{0} = -_a\ (n\triangleright a_A) \pm (n\triangleright a_A \pm (-\ n)\triangleright a_A)$)
apply (*subgoal-tac* $n\triangleright a_A \in carrier\ A$)
apply (*frule ag-mOp-closed* [*of* $n\triangleright a_A$])
apply (*thin-tac* $\mathbf{0} = \ n\triangleright a_A \pm (-\ n)\triangleright a_A$)
apply (*simp add*:*ag-r-zero*)
apply (*subgoal-tac* $(-\ n)\triangleright a_A \in carrier\ A$)
apply (*simp add*:*ag-pOp-assoc*[*THEN sym*])
apply (*simp add*:*ag-l-inv1*) **apply** (*simp add*:*ag-l-zero*)
apply (*simp add*:*nt-mem0*)+
done

lemma (**in** *aGroup*) *m-x-asum*:$[\![\ a \in carrier\ A;\ b \in carrier\ A]\!]$
$\Longrightarrow aSum\ A\ m\ (a \pm b) = (aSum\ A\ m\ a) \pm (aSum\ A\ m\ b)$
apply (*induct-tac m*) **apply** *simp*
apply (*rule ag-r-zero*[*THEN sym*])
apply (*simp add*:*ag-inc-zero*)
apply *simp*
apply (*frule-tac* $n = n$ **in** *asum-mem*[*of a*])
apply (*frule-tac* $n = n$ **in** *asum-mem*[*of b*])
apply (*frule-tac* $a = aSum\ A\ n\ a$ **and** $c = aSum\ A\ n\ b$ **in**
pOp-assocTr43 [*of* - *a* - *b*], *assumption*+) **apply** *simp*
apply (*frule-tac* $x = a$ **and** $y = aSum\ A\ n\ b$ **in** *ag-pOp-commute*, *assumption*+)

apply *simp*
apply (*simp add*:*pOp-assocTr43*[*THEN sym*])
done

lemma (**in** *aGroup*) *asum-multTr-pp*:$a \in carrier\ A \Longrightarrow$
$aSum\ A\ m\ (aSum\ A\ n\ a) = aSum\ A\ (m * n)\ a$
apply (*induct-tac n*)
apply *simp*
apply (*induct-tac m*, *simp*)
apply (*simp*, *rule ag-r-zero*, *simp add*:*ag-inc-zero*)
apply *simp*
apply (*frule-tac* $n = n$ **in** *asum-mem*[*of a*])
apply (*frule-tac* $a = aSum\ A\ n\ a$ **and** $b = a$ **and** $m = m$ **in** *m-x-asum*,
assumption+, *simp*)
apply (*frule-tac* $n = m$ **and** $m = m * n$ **in** *asumTr* [*of a*])
apply *simp*
apply (*frule-tac* $n = m * n$ **in** *asum-mem*[*of a*])
apply (*frule-tac* $n = m$ **in** *asum-mem*[*of a*])
apply (*simp add*:*ag-pOp-commute*)
done

lemma (**in** *aGroup*) *nt-mult-pp*:⟦ $a \in carrier\ A$; $0 \leq m$; $0 \leq n$⟧
$\Longrightarrow m{\triangleright}(n{\triangleright}a_A)_A = (m * n){\triangleright}a_A$
apply (*simp add*:*sprod-n-a-def*)
apply (*subgoal-tac* $0 \leq m * n$, *simp*)
apply (*simp add*:*asum-multTr-pp*)
apply (*simp add*:*nat-mult-distrib*)
apply (*frule int-mult-le* [*of* 0 *m n*], *assumption*+)
apply (*simp add*:*mult.commute*)
done

lemma (**in** *aGroup*) *asum-multTr-pm*:⟦$a \in carrier\ A$; $0 \leq m$; $n < 0$⟧ $\Longrightarrow$
aSum A (*nat m*) (*aSum A* (*nat* (− *n*)) ($-_a$ *a*)) =
aSum A (*nat* (*m* ∗ (− *n*))) ($-_a$ *a*)
apply (*frule ag-mOp-closed* [*of a*])
apply (*simp add*:*asum-multTr-pp*)
apply (*subgoal-tac nat m* ∗ *nat* (− *n*) = *nat* (− (*m* ∗ *n*)), *simp*)
apply (*subgoal-tac* (*nat m*) ∗ (*nat* (− *n*)) = *nat* (*m* ∗ (− *n*)), *simp*)
apply *simp*
apply (*subst zmult-zminus-right*[*THEN sym*, *of m n*])
apply (*rule nat-mult-distrib* [*THEN sym*, *of m*], *assumption*+)
done

lemma (**in** *aGroup*) *nt-mult-pm*:⟦$a \in carrier\ A$; $0 \leq m$; $n < 0$⟧ $\Longrightarrow$
$m{\triangleright}(n{\triangleright}a_A)_A = (m * n){\triangleright}a_A$
apply (*frule zmult-zle-mono1-neg* [*of* 0 *m n*])
apply (*simp add*:*zless-imp-zle*, *simp*)
apply (*simp add*:*sprod-n-a-def*)
apply (*rule impI*)
apply (*simp add*: *asum-multTr-pm*)
done

lemma (**in** *aGroup*) *asum-multTr-mp*:⟦$a \in carrier\ A$; $m < 0$; $0 \leq n$⟧ $\Longrightarrow$
aSum A (*nat* (−*m*))($-_a$ (*aSum A* (*nat n*) *a*)) = *aSum A* (*nat* ((− *m*) ∗ *n*)) ($-_a$ *a*)
apply (*frule asum-mem* [*of a nat n*])
apply (*frule ag-mOp-closed* [*of aSum A* (*nat n*) *a*])
apply (*simp add*:*sprod-n-a-def*)
apply (*subgoal-tac* $-_a$ (*aSum A* (*nat n*) *a*) = *aSum A* (*nat n*) ($-_a$ *a*))
apply *simp*
apply (*subst asum-multTr-pp*)
apply (*simp add*:*ag-mOp-closed*)
apply (*subgoal-tac* (*nat* (− *m*)) ∗ (*nat n*) = *nat* ((− *m*) ∗ *n*), *simp*)
apply (*subst nat-mult-distrib*, *simp*, *simp*)
apply (*frule nt-inv0* [*of a n*])
apply (*simp add*:*sprod-n-a-def*)
done

lemma (**in** *aGroup*) *nt-mult-mp*:⟦$a \in carrier\ A$; $m < 0$; $0 \leq n$⟧ $\Longrightarrow$
$m{\triangleright}(n{\triangleright}a_A)_A = (m * n){\triangleright}a_A$

apply (*simp add*:*sprod-n-a-def*)
apply (*cut-tac zless-imp-zle*[*of m 0*])
apply (*frule int-mult-le* [*of m 0 n*], *assumption*, *simp*)
apply (*case-tac* $0 \leq m * n$, *simp*)
apply (*frule zle-imp-zless-or-eq* [*of 0* $m * n$])
apply (*thin-tac* $0 \leq m * n$, *simp add*:*zle mult.commute*)
apply (*simp add*:*ag-inv-zero*, *simp add*:*aSum-zero*)
apply *simp*
apply (*simp add*:*asum-multTr-mp*)
apply (*simp add*:*zle-imp-zless-or-eq*)
done

lemma (**in** *aGroup*) *asum-multTr-mm*:$[\![a \in carrier\ A;\ m < 0;\ n < 0]\!] \Longrightarrow$
$aSum\ A\ (nat\ (-m))(-_a\ (aSum\ A\ (nat\ (-\ n))\ (-_a\ a))) =$
$aSum\ A\ (nat\ ((-\ m) * (-\ n)))\ a$
apply (*simp add*:*sprod-n-a-def*)
apply (*subgoal-tac* $-_a\ (aSum\ A\ (nat\ (-\ n))\ (-_a\ a)) = aSum\ A\ (nat\ (-\ n))\ a$)
apply *simp*
apply (*simp add*:*asum-multTr-pp*)
apply (*subst nat-mult-distrib*[*THEN sym*]) **apply** *simp*
apply *simp*
apply (*frule ag-mOp-closed* [*of* $-_a\ a$])
apply (*frule nt-inv0* [*of* $-_a\ a - n$])
apply (*simp add*:*sprod-n-a-def*)
apply (*simp add*:*ag-inv-inv*)
done

lemma (**in** *aGroup*) *nt-mult-mm*:$[\![\ a \in carrier\ A;\ m < 0;\ n < 0]\!] \Longrightarrow$
$m \triangleright (n \triangleright a_A)_A = (m * n) \triangleright a_A$
apply (*simp add*:*sprod-n-a-def*)
apply (*subgoal-tac* $0 \leq m * n$) **apply** *simp*
apply (*simp add*:*asum-multTr-mm*)
apply (*frule zmult-neg-neg*[*of m n*], *assumption*+)
apply (*simp add*:*zle-imp-zless-or-eq*)
done

lemma (**in** *aGroup*) *nt-mult-assoc0*:$a \in carrier\ A \Longrightarrow m \triangleright n \triangleright a_{A\,A} = (m * n) \triangleright a_A$
apply (*case-tac* $0 \leq n$)
apply (*case-tac* $0 \leq m$)
apply (*simp add*:*nt-mult-pp*, *simp add*:*zle*)
apply (*frule nt-mult-mp*[*of a m n*], *assumption*, *simp*, *simp*)
apply (*cut-tac less-linear*[*of 0 m*])
apply (*case-tac* $0 \leq m$)
apply (*simp add*:*nt-mult-pm*)
apply (*simp add*:*zle*)
apply (*simp add*:*nt-mult-mm*)
done

lemma (**in** *aGroup*) *single-gen-carrTr*:$a \in carrier\ A \Longrightarrow$

asubGroup A $\{x.\ \exists\, n.\ x = (n \triangleright a_A)\}$
apply (*rule asubg-test*)
apply (*rule subsetI*, *simp*)
apply (*erule exE*, *simp add*:*nt-mem0*)
apply (*simp*, *blast*)
apply ((*rule ballI*)+, *simp*)
apply (*erule exE*)+
apply (*simp add*:*nt-inv0*)
apply (*subst nt-sum0*[*THEN sym*], *assumption*+)
apply *blast*
done

lemma (**in** *aGroup*) *ag-single-inc-a*:*ag-single-gen A a* $\Longrightarrow$ *a* $\in$ *carrier A*
apply (*simp add*:*ag-single-gen-def*)
done

lemma (**in** *aGroup*) *single-gen*:*ag-single-gen A a* $\Longrightarrow$
carrier A $= \{g.\ \exists\, n.\ g = (n \triangleright a_A)\}$
apply (*rule equalityI*)
apply (*frule ag-single-inc-a* [*of a*])
apply (*rule subsetI*, *simp*)
apply (*unfold ag-single-gen-def*, *erule conjE*)
apply (*frule single-gen-carrTr* [*of a*])
apply (*subgoal-tac a* $\in \{x.\ \exists\, n.\ x = n \triangleright a_A\}$)
apply (*subgoal-tac* $\bigcap \{H.\ A +> H \wedge a \in H\} \subseteq \{x.\ \exists\, n.\ x = n \triangleright a_A\}$)
apply (*frule-tac A* $= \bigcap \{H.\ A +> H \wedge a \in H\}$ **and** *B* $= \{x.\ \exists\, n.\ x = n \triangleright a_A\}$
and
c $= x$ **in** *subsetD*)
apply (*frule sym*, *thin-tac carrier A* $= \bigcap \{H.\ A +> H \wedge a \in H\}$)
apply (*simp*, *simp*)
apply (*thin-tac carrier A* $= \bigcap \{H.\ A +> H \wedge a \in H\}$)
apply (*rule subsetI*, *blast*)
apply (*thin-tac carrier A* $= \bigcap \{H.\ A +> H \wedge a \in H\}$,
thin-tac A $+> \{x.\ \exists\, n.\ x = n \triangleright a_A\}$)
apply (*subgoal-tac a* $= 1 \triangleright a_A$, *blast*)
apply (*simp add*:*sprod-n-a-def*, *simp add*:*ag-l-zero*[*THEN sym*])
apply (*fold ag-single-gen-def*)
apply (*frule ag-single-inc-a* [*of a*])
apply (*unfold ag-single-gen-def*)
apply (*erule conjE*)
apply (*thin-tac carrier A* $= \bigcap \{H.\ A +> H \wedge a \in H\}$)
apply (*rule subsetI*, *simp*)
apply (*erule exE*)
apply (*simp*, *simp add*:*nt-mem0*)
done

definition
single-gen-free :: [('*a*, '*m*) *aGroup-scheme*, '*a*] $\Rightarrow$ *bool* **where**
single-gen-free A a $==$ $\forall\, n.\ n \neq 0 \longrightarrow \mathbf{0}_A \neq n \triangleright a_A$

definition
sfg :: [('*a*, '*m*) *aGroup-scheme*, '*a*] ⇒ *bool* **where**
sfg A a ⟷ *ag-single-gen A a* ∧ *single-gen-free A a*

lemma (**in** *aGroup*) *single-gen-free-neg*:⟦*sfg A a*; *n*▷a_A = **0**⟧ ⟹ *n* = *0*
apply (*simp add*:*sfg-def*, *erule conjE*)
apply (*rule contrapos-pp*, *simp*+)
apply (*simp add*:*single-gen-free-def*)
apply (*drule-tac a* = *n* **in** *forall-spec*, *simp*)
apply *simp*
done

lemma (**in** *aGroup*) *sfg-G-inc-a*:*sfg A a* ⟹ *a* ∈ *carrier A*
apply (*simp add*:*sfg-def ag-single-inc-a*)
done

lemma *sfg-agroup*:*sfg A a* ⟹ *aGroup A*
apply (*simp add*:*sfg-def ag-single-gen-def*)
done

lemma (**in** *aGroup*) *mem-G-nt*:⟦*sfg A a*; *x* ∈ *carrier A*⟧ ⟹ ∃ *n*. *x* = *n*▷a_A
apply (*simp add*:*sfg-def*) **apply** (*erule conjE*)
apply (*frule single-gen* [*of a*]) **apply** *simp*
done

lemma (**in** *aGroup*) *nt-mem*:*sfg A a* ⟹ *n*▷a_A ∈ *carrier A*
apply (*frule sfg-G-inc-a*)
apply (*frule sfg-agroup*)
apply (*simp add*:*nt-mem0*)
done

lemma (**in** *aGroup*) *nt-zero*:*sfg A a* ⟹ *0*▷a_A = **0**
apply (*frule sfg-G-inc-a*)
apply (*frule sfg-agroup*)
apply (*simp add*:*nt-zero0*)
done

lemma (**in** *aGroup*) *nt-sum*:*sfg A a* ⟹ (*n* + *m*)▷a_A = *n*▷a_A ± (*m*▷a_A)
apply (*frule sfg-G-inc-a*)
apply (*frule sfg-agroup*)
apply (*simp add*:*nt-sum0*)
done

lemma (**in** *aGroup*) *nt-inv*:*sfg A a* ⟹ $-_a$(*n*▷a_A) = (− *n*)▷a_A
apply (*frule sfg-G-inc-a*)
apply (*frule sfg-agroup*)
apply (*simp add*:*nt-inv0*)

done

lemma (**in** *aGroup*) *nt-mult-assoc*:*sfg A a* $\Longrightarrow m \triangleright n \triangleright a_{A\,A} = (m * n) \triangleright a_A$
apply (*frule sfg-G-inc-a*)
apply (*frule sfg-agroup*)
apply (*simp add*:*nt-mult-assoc0*)
done

lemma (**in** *aGroup*) *sfg-free*:$[\![$*sfg A a*; $n \neq m\ ]\!] \Longrightarrow n \triangleright a_A \neq (m \triangleright a_A)$
apply (*rule contrapos-pp*, *simp*+)
apply (*frule sfg-G-inc-a*)
apply (*frule sfg-agroup*)
apply (*frule nt-mem* [*of a m*])
apply (*frule nt-mem* [*of a n*])
apply (*subgoal-tac* $n \triangleright a_A \pm (-_a\ (m \triangleright a_A)) = \mathbf{0}$)
apply (*thin-tac* $n \triangleright a_A = m \triangleright a_A$)
apply (*simp add*:*nt-inv*)
apply (*simp add*:*nt-sum*[*THEN sym*])
apply (*frule single-gen-free-neg*[*of a n* − *m*], *assumption*+)
apply *simp*
apply (*simp add*:*ag-r-inv1*)
done

lemma (**in** *aGroup*) *sfg-free-inj*:$[\![$*sfg A a*; $n \triangleright a_A = (m \triangleright a_A)\ ]\!] \Longrightarrow n = m$
apply (*rule contrapos-pp*, *simp*+)
apply (*simp add*:*sfg-free*)
done

6.4 Free Generated Modules (constructive)

definition
sop-one::[('*r*, '*m*) *Ring-scheme*, '*r* ⇒ '*a* ⇒ '*a*, '*a set*] ⇒ *bool* **where**
sop-one R s A ⟷ (∀ *x*∈*A*. $(1_{r\,R})\ _s\cdot\ x = x$)

definition
sop-assoc :: [('*r*, '*m*) *Ring-scheme*, '*r* ⇒ '*a* ⇒ '*a*, '*a set*] ⇒ *bool* **where**
sop-assoc R s A ⟷ (∀ *a*∈*carrier R*. ∀ *b*∈*carrier R*. ∀ *x*∈*A*.
$(a\ \cdot_{r\,R}\ b)\ _s\cdot\ x = a\ _s\cdot\ (b\ _s\cdot\ x)$)

definition
sop-inv :: [('*r*, '*m*) *Ring-scheme*, '*r* ⇒ '*a* ⇒ '*a*, '*a* ⇒ '*a*, '*a set*]
⇒ *bool* **where**
sop-inv R s i A ⟷ (∀ *r*∈*carrier R*. ∀ *x*∈*A*. $r\ _s\cdot\ (_i- x) = (-_{a\,R}\ r)\ _s\cdot\ x$)

definition
sop-distr1 :: [('*r*, '*m*) *Ring-scheme*, '*r* ⇒ '*a* ⇒ '*a*, '*a* ⇒ '*a* ⇒ '*a*,
'*a* ⇒ '*a*, '*a set*, '*a*] ⇒ *bool* **where**
sop-distr1 R s f i A z ⟷ (∀ *a*∈*carrier R*. ∀ *b*∈*carrier R*.
∀ *x*∈(*aug-pm-set z i A*). $(a \pm_R b)\ _s\cdot\ x = (a\ _s\cdot\ x)\ _f+\ (b\ _s\cdot\ x)$)

definition
sop-distr2 :: [('r, 'm) *Ring-scheme*, 'r ⇒ 'a ⇒ 'a, 'a ⇒ 'a ⇒ 'a,
'a ⇒ 'a, 'a *set*, 'a] ⇒ *bool* **where**
sop-distr2 R s f i A z ⟷ (∀ a∈*carrier R*.
∀ x∈*addition-set f* (*aug-pm-set z i A*).
∀ y∈*addition-set f* (*aug-pm-set z i A*).
a $_s$· (x $_f$+ y) = (a $_s$· x) $_f$+ (a $_s$· y))

definition
sop-z :: [('r, 'm) *Ring-scheme*, 'r ⇒ 'a ⇒ 'a, 'a] ⇒ *bool* **where**
sop-z R s z ⟷ (∀ r∈*carrier R*. r $_s$· z = z)

definition
fgmodule :: [('r, 'm) *Ring-scheme*, 'a *set*, 'a, 'a ⇒ 'a, 'a ⇒ 'a ⇒ 'a,
'r ⇒ 'a ⇒ 'a] ⇒ ('a, 'r) *Module* **where**
fgmodule R A z i f s =
(|*carrier* = *addition-set f* (*aug-pm-set z i* (*s-set R s A*)),
pop = λx∈*addition-set f* (*aug-pm-set z i* (*s-set R s A*)).
λy∈*addition-set f* (*aug-pm-set z i* (*s-set R s A*)). x $_f$+ y,
mop = λx∈*addition-set f* (*aug-pm-set z i* (*s-set R s A*)). $_i$− x,
zero = z,
sprod = λr∈*carrier R*.
λx∈*addition-set f* (*aug-pm-set z i* (*s-set R s A*)). r $_s$· x |)

lemma *fgmodule-carr*:*carrier* (*fgmodule R A z i f s*) =
addition-set f (*aug-pm-set z i* (*s-set R s A*))
by (*simp add*:*fgmodule-def*)

lemma *a-in-s-set*:a ∈ A ⟹ a ∈ *s-set R s A*
by (*simp add*:*s-set-def*)

lemma (**in** *Ring*) *ra-in-s-set*:⟦r ∈ *carrier R*; a ∈ A⟧ ⟹ r $_s$· a ∈ *s-set R s A*
by (*simp add*:*s-set-def*, *blast*)

lemma *in-aug-pm-set*:
x ∈ *aug-pm-set z i A* = (x = z ∨ x ∈ A ∨ x ∈ *minus-set i A*)
by (*simp add*:*aug-pm-set-def*)

lemma (**in** *Ring*) *in-s-set*:x ∈ *s-set R s A* ⟹ (∃ r ∈ *carrier R*. ∃ a ∈ A.
x = r $_s$· a) ∨ x ∈ A
by (*simp add*:*s-set-def*)

lemma (**in** *Ring*) *sop-closedTr0*:⟦*ipp-cond1* (*s-set R s A*) *i*;
ipp-cond2 z (*s-set R s A*) *i f*; *ipp-cond3 z i*;
inv-ipp z i f (*s-set R s A*); *zeroA z i f* (*s-set R s A*) *z*;
sop-distr2 R s f i (*s-set R s A*) *z*;
sop-assoc R s (*aug-pm-set z i* (*s-set R s A*));
sop-inv R s i (*s-set R s A*);

sop-one R s (aug-pm-set z i (s-set R s A)); *sop-z R s z*;
r ∈ carrier R; *x ∈ aug-pm-set z i (s-set R s A)*⟧ ⟹
r $_s\cdot$ *x ∈ aug-pm-set z i (s-set R s A)*
apply (*simp add*:*in-aug-pm-set*)
apply (*case-tac x = z*, *simp*, *simp add*:*sop-z-def*)

apply (*case-tac x ∈ s-set R s A*, *simp add*:*s-set-def*, *fold s-set-def*)
apply (*case-tac x ∈ A*, *simp*, *blast*)

apply *simp*
apply ((*erule bexE*)+, *simp*)
apply (*simp add*:*sop-assoc-def*)
apply (*drule-tac x = r* **in** *bspec*, *assumption*,
drule-tac x = ra **in** *bspec*, *assumption*,
frule-tac x = a **in** *bspec*,
subst in-aug-pm-set, *simp add*:*a-in-s-set*)
apply (*rotate-tac −1*, *frule sym*, *thin-tac r* $\cdot_r$ *ra* $_s\cdot$ *a = r* $_s\cdot$ (*ra* $_s\cdot$ *a*),
simp)
apply (*frule-tac x = r* **and** *y = ra* **in** *ring-tOp-closed*, *assumption*+, *blast*)

apply *simp* **apply** (*thin-tac x ≠ z*, *thin-tac x ∉ s-set R s A*)
apply (*simp add*:*minus-set-def*, *erule bexE*, *simp*)
apply (*simp add*:*sop-inv-def*[*of R s i s-set R s A*])
apply (*cut-tac ring-is-ag*,
frule-tac x = r **in** *aGroup.ag-mOp-closed*[*of R*], *assumption*)
apply (*frule-tac a = y* **in** *ra-in-s-set*[*of* $-_a$ *r - s-set R s A s*],
assumption+)
apply (*frule-tac x = y* **in** *in-s-set*[*of - s A*])
apply (*case-tac y ∈ A*, *simp*, *simp add*:*ra-in-s-set*)

apply *simp* **apply** ((*erule bexE*)+, *simp*)
apply (*simp add*:*sop-assoc-def*)
apply (*drule-tac x* $= -_a$ *r* **in** *bspec*, *assumption*,
thin-tac ∀ *r*∈*carrier R*. ∀ *x*∈*s-set R s A*. *r* $_s\cdot$ ($_i-$ *x*) = ($-_a$ *r*) $_s\cdot$ *x*,
drule-tac x = ra **in** *bspec*, *assumption*,
drule-tac x = a **in** *bspec*,
simp add:*aug-pm-set-def*, *simp add*:*a-in-s-set*)
apply (*rotate-tac −1*, *frule sym*,
thin-tac ($-_a$ *r*) $\cdot_r$ *ra* $_s\cdot$ *a* = ($-_a$ *r*) $_s\cdot$ (*ra* $_s\cdot$ *a*), *simp*,
frule-tac x $= -_a$ *r* **and** *y = ra* **in** *ring-tOp-closed*, *assumption*,
simp add:*ra-in-s-set*)
done

lemma (**in** *Ring*) *sop-closedTr*:⟦*ipp-cond1* (*s-set R s A*) *i*;
ipp-cond2 z (*s-set R s A*) *i f*; *ipp-cond3 z i*;
inv-ipp z i f (*s-set R s A*); *zeroA z i f* (*s-set R s A*) *z*;
sop-distr2 R s f i (*s-set R s A*) *z*;
sop-assoc R s (*aug-pm-set z i* (*s-set R s A*));
sop-inv R s i (*s-set R s A*);

```
        sop-one R s (aug-pm-set z i (s-set R s A)); sop-z R s z⟧ ⟹
    ∀ r∈carrier R. ∀ x∈add-set f (aug-pm-set z i (s-set R s A)) n.
                r ₛ· x ∈ add-set f (aug-pm-set z i (s-set R s A)) n
apply (induct-tac n)
 apply (simp, (rule ballI)+, simp add:sop-closedTr0)

apply (rule ballI)+ apply simp
 apply (erule bexE)+
 apply (frule-tac r = r and x = t in sop-closedTr0 [of s A i z f],
        assumption+)

 apply (subgoal-tac sa ∈ addition-set f (aug-pm-set z i (s-set R s A)))
 apply (subgoal-tac t ∈ addition-set f (aug-pm-set z i (s-set R s A)))
 apply (simp add:sop-distr2-def) apply blast
 apply (cut-tac addition-inc-add0[of aug-pm-set z i (s-set R s A) f])
 apply (simp add:subsetD)
 apply (cut-tac n = n in addition-inc-add[of f
                                          aug-pm-set z i (s-set R s A)])
 apply (simp add:subsetD)
done

lemma (in Ring) sop-closed:⟦ipp-cond1 (s-set R s A) i;
     ipp-cond2 z (s-set R s A) i f; ipp-cond3 z i;
      inv-ipp z i f (s-set R s A); zeroA z i f (s-set R s A) z;
       sop-distr2 R s f i (s-set R s A) z;
        sop-assoc R s (aug-pm-set z i (s-set R s A));
         sop-inv R s i (s-set R s A);
          sop-one R s (aug-pm-set z i (s-set R s A)); sop-z R s z⟧ ⟹
    ∀ r∈carrier R. ∀ x∈addition-set f (aug-pm-set z i (s-set R s A)).
         r ₛ· x ∈ addition-set f (aug-pm-set z i (s-set R s A))
apply (subst addition-set-def)
 apply simp
 apply (rule ballI) apply (rule allI) apply (rule impI)
 apply (erule exE)
 apply (rule ballI) apply simp
 apply (thin-tac y = add-set f (aug-pm-set z i (s-set R s A)) n)
 apply (frule-tac n = n in sop-closedTr[of s A i z f], assumption+)
 apply (drule-tac x = r in bspec, assumption,
        drule-tac x = x in bspec, assumption)

 apply (cut-tac n = n in addition-inc-add [of f
                                  aug-pm-set z i (s-set R s A)],
        simp add:subsetD)
done

lemma (in Ring) sop-oneTr:⟦commute-bpp f (aug-pm-set z i (s-set R s A));
  assoc-bpp (aug-pm-set z i (s-set R s A)) f;
   ipp-cond1 (s-set R s A) i; ipp-cond2 z (s-set R s A) i f;
    ipp-cond3 z i; inv-ipp z i f (s-set R s A); zeroA z i f (s-set R s A) z;
```

```
    sop-distr2 R s f i (s-set R s A) z;
     sop-assoc R s (aug-pm-set z i (s-set R s A));
      sop-one R s (aug-pm-set z i (s-set R s A))⟧ ⟹
  ∀x∈add-set f (aug-pm-set z i (s-set R s A)) n. (1_r) _s· x = x
apply (induct-tac n)
 apply (rule ballI, simp, simp add:sop-one-def)

apply (rule ballI)
 apply (simp, (erule bexE)+, simp)
 apply (subgoal-tac t ∈ addition-set f (aug-pm-set z i (s-set R s A)))
 apply (subgoal-tac sa ∈ addition-set f (aug-pm-set z i (s-set R s A)))
 apply (cut-tac ring-one)
 apply (simp add:sop-distr2-def)
 apply (thin-tac ∀x∈add-set f (aug-pm-set z i (s-set R s A)) n. (1_r) _s· x
                 = x) apply (simp add:sop-one-def)
 apply (cut-tac n = n in addition-inc-add[of f
                aug-pm-set z i (s-set R s A)], simp add:subsetD,
       cut-tac addition-inc-add0[of aug-pm-set z i (s-set R s A) f],
                    simp add:subsetD)
done

lemma (in Ring) sop-one:⟦commute-bpp f (aug-pm-set z i (s-set R s A));
   assoc-bpp (aug-pm-set z i (s-set R s A)) f; ipp-cond1 (s-set R s A) i;
    ipp-cond2 z (s-set R s A) i f; ipp-cond3 z i;
     inv-ipp z i f (s-set R s A); zeroA z i f (s-set R s A) z;
      sop-distr2 R s f i (s-set R s A) z;
       sop-assoc R s (aug-pm-set z i (s-set R s A));
        sop-one R s (aug-pm-set z i (s-set R s A))⟧ ⟹
  ∀x∈addition-set f (aug-pm-set z i (s-set R s A)). (1_r) _s· x = x
apply (rule ballI) apply (simp add:addition-set-def)
apply (erule exE, erule conjE, erule exE, simp,
      thin-tac xa = add-set f (aug-pm-set z i (s-set R s A)) n)
apply (simp add:sop-oneTr)
done

lemma (in Ring) sop-assocTr:⟦ipp-cond1 (s-set R s A) i;
      ipp-cond2 z (s-set R s A) i f; ipp-cond3 z i;
      inv-ipp z i f (s-set R s A); zeroA z i f (s-set R s A) z;
      sop-distr2 R s f i (s-set R s A) z;
      sop-assoc R s (aug-pm-set z i (s-set R s A));
      sop-inv R s i (s-set R s A);
      sop-one R s (aug-pm-set z i (s-set R s A)); sop-z R s z⟧ ⟹
     ∀a∈carrier R. ∀b∈carrier R.
      ∀x∈add-set f (aug-pm-set z i (s-set R s A)) n.
                    a _s· ( b _s· x) = (a ·_r b) _s· x
apply (induct-tac n)
apply (rule ballI)+
apply (simp add:sop-assoc-def)
apply (rule ballI)+ apply simp
```

apply (*erule bexE*)+
 apply (*drule-tac* $x = a$ **in** *bspec*, *assumption*,
 drule-tac $x = b$ **in** *bspec*, *assumption*,
 drule-tac $x = sa$ **in** *bspec*, *assumption*)

 apply *simp*
 apply (*cut-tac* $n = n$ **in** *addition-inc-add*[*of f*
 aug-pm-set z i (*s-set R s A*)],
 cut-tac addition-inc-add0[*of aug-pm-set z i* (*s-set R s A*) *f*])
 apply (*frule-tac* $c = sa$ **and** $A = add\text{-}set\ f\ (aug\text{-}pm\text{-}set\ z\ i\ (s\text{-}set\ R\ s\ A))\ n$
 in *subsetD*[*of - addition-set f* (*aug-pm-set z i* (*s-set R s A*))],
 assumption+,
 frule-tac $c = t$ **in** *subsetD*[*of aug-pm-set z i* (*s-set R s A*)
 addition-set f (*aug-pm-set z i* (*s-set R s A*))], *assumption*+,
 frule-tac $x = a$ **and** $y = b$ **in** *ring-tOp-closed*, *assumption*+)
 apply (*simp add*:*sop-distr2-def*[*of R s f i s-set R s A z*])
 apply (*frule sop-closed*[*of s A i z f*], *assumption*+, *simp add*:*sop-distr2-def*,
 assumption+, *rotate-tac* -1)
 apply (*drule-tac* $x = b$ **in** *bspec*, *assumption*,
 rotate-tac -1,
 frule-tac $x = sa$ **in** *bspec*, *assumption*,
 drule-tac $x = t$ **in** *bspec*, *assumption*)
 apply (*simp*,
 thin-tac $\forall a{\in}carrier\ R.$
 $\forall x{\in}addition\text{-}set\ f\ (aug\text{-}pm\text{-}set\ z\ i\ (s\text{-}set\ R\ s\ A)).$
 $\forall y{\in}addition\text{-}set\ f\ (aug\text{-}pm\text{-}set\ z\ i\ (s\text{-}set\ R\ s\ A)).$
 $a\ {}_s{\cdot}\ (x\ {}_f{+}\ y) = a\ {}_s{\cdot}\ x\ {}_f{+}\ a\ {}_s{\cdot}\ y$)
 apply (*simp add*:*sop-assoc-def*)
done

lemma (**in** *Ring*) *sop-assoc*:⟦*ipp-cond1* (*s-set R s A*) *i*;
 ipp-cond2 z (*s-set R s A*) *i f*; *ipp-cond3 z i*;
 inv-ipp z i f (*s-set R s A*); *zeroA z i f* (*s-set R s A*) *z*;
 sop-distr2 R s f i (*s-set R s A*) *z*;
 sop-assoc R s (*aug-pm-set z i* (*s-set R s A*));
 sop-inv R s i (*s-set R s A*); *sop-z R s z*;
 sop-one R s (*aug-pm-set z i* (*s-set R s A*))⟧ ⟹
 $\forall a{\in}carrier\ R.\ \forall b{\in}carrier\ R.$
 $\forall x{\in}addition\text{-}set\ f\ (aug\text{-}pm\text{-}set\ z\ i\ (s\text{-}set\ R\ s\ A)).$
 $a\ {}_s{\cdot}\ (b\ {}_s{\cdot}\ x) = (\ a\ {\cdot}_r\ b)\ {}_s{\cdot}\ x$
apply (*rule ballI*)+ **apply** (*simp add*:*addition-set-def*)
 apply (*erule exE*, *erule conjE*, *erule exE*, *simp*)
 apply (*simp add*:*sop-assocTr*)
done

lemma (**in** *Ring*) *s-set-commute*:⟦*commute-bpp f* (*aug-pm-set z i* (*s-set R s A*));
 $x \in addition\text{-}set\ f\ (aug\text{-}pm\text{-}set\ z\ i\ (s\text{-}set\ R\ s\ A))$;
 $y \in addition\text{-}set\ f\ (aug\text{-}pm\text{-}set\ z\ i\ (s\text{-}set\ R\ s\ A))$⟧ ⟹
 $x\ {}_f{+}\ y = y\ {}_f{+}\ x$

```
apply (simp add:commute-bpp-def)
done

lemma (in Ring) add-s-set-inc-add-set:
      add-set f (aug-pm-set z i A) n ⊆
            add-set f (aug-pm-set z i (s-set R s A)) n
apply (rule add-set-mono[of aug-pm-set z i A
                           aug-pm-set z i (s-set R s A) f n])
apply (rule subsetI, simp add:aug-pm-set-def s-set-def)
apply (case-tac x = z, simp)
 apply (case-tac x ∈ A, simp)

 apply simp
 apply (simp add:minus-set-def, erule bexE)
 apply blast
done

lemma (in Ring) sop-distr1Tr:⟦commute-bpp f (aug-pm-set z i (s-set R s A));
    assoc-bpp (aug-pm-set z i (s-set R s A)) f; ipp-cond1 (s-set R s A) i;
     ipp-cond2 z (s-set R s A) i f; ipp-cond3 z i;
      inv-ipp z i f (s-set R s A); zeroA z i f (s-set R s A) z;
       sop-distr1 R s f i (s-set R s A) z;
        sop-distr2 R s f i (s-set R s A) z;
         sop-assoc R s (aug-pm-set z i (s-set R s A));
          sop-inv R s i (s-set R s A);
           sop-one R s (aug-pm-set z i (s-set R s A)); sop-z R s z⟧ ⟹
∀ a∈carrier R. ∀ b∈carrier R. ∀ x∈ add-set f (aug-pm-set z i (s-set R s A)) n.
        (a ± b) s· x = a s· x f+ (b s· x)
apply (induct-tac n)
 apply ((rule ballI)+, simp add:sop-distr1-def)

apply (rule ballI)+ apply simp
 apply (erule bexE)+
 apply (cut-tac ring-is-ag,
       frule-tac x = a and y = b in aGroup.ag-pOp-closed [of R],
        assumption+)
  apply (cut-tac n = n in addition-inc-add[of f
                   aug-pm-set z i (s-set R s A)],
       cut-tac addition-inc-add0[of aug-pm-set z i (s-set R s A) f])
 apply (frule-tac c = sa and A = add-set f (aug-pm-set z i (s-set R s A)) n
        in subsetD[of - addition-set f (aug-pm-set z i (s-set R s A))],
        assumption+,
       frule-tac c = t in subsetD[of aug-pm-set z i (s-set R s A)
        addition-set f (aug-pm-set z i (s-set R s A))], assumption+,
       frule-tac x = a and y = b in  ring-tOp-closed, assumption+)
 apply simp
 apply (simp add:sop-distr2-def sop-distr1-def)
 apply (frule sop-closed[of s A i z f], assumption+,
       simp add:sop-distr2-def, simp add:sop-assoc-def, assumption+)
```

apply (*rotate-tac −1*,
frule-tac x = a **in** *bspec*, *assumption*,
rotate-tac −1,
frule-tac x = sa **in** *bspec*, *assumption*,
drule-tac x = t **in** *bspec*, *assumption*)
apply (*frule-tac x = b* **in** *bspec*, *assumption*,
rotate-tac −1,
frule-tac x = sa **in** *bspec*, *assumption*,
drule-tac x = t **in** *bspec*, *assumption*,
thin-tac ∀ *r*∈*carrier R*.
∀ *x*∈*addition-set f* (*aug-pm-set z i* (*s-set R s A*)).
r $_s\cdot$ *x* ∈ *addition-set f* (*aug-pm-set z i* (*s-set R s A*)))
apply (*frule-tac x* = *a* $_s\cdot$ *t* **and** *y* = *b* $_s\cdot$ *t* **in**
bpp-closed [*of* (*aug-pm-set z i* (*s-set R s A*)) *f*], *assumption*+)
apply (*subst addition-assoc*, *assumption*+)
apply (*frule-tac ?x1* = *b* $_s\cdot$ *sa* **and** *?y1* = *a* $_s\cdot$ *t* **and** *?z1* = *b* $_s\cdot$ *t* **in**
addition-assoc[*THEN sym*, *of aug-pm-set z i* (*s-set R s A*) *f*],
assumption+, *simp*,
thin-tac b $_s\cdot$ *sa* $_f$+ (*a* $_s\cdot$ *t* $_f$+ *b* $_s\cdot$ *t*) = *b* $_s\cdot$ *sa* $_f$+ *a* $_s\cdot$ *t* $_f$+ *b* $_s\cdot$ *t*)
apply (*frule-tac x* = *b* $_s\cdot$ *sa* **and** *y* = *a* $_s\cdot$ *t* **in**
s-set-commute[*of f z i s A*], *assumption*+, *simp*)
apply (*frule-tac x* = *a* $_s\cdot$ *t* **and** *y* = *b* $_s\cdot$ *sa* **and** *z* = *b* $_s\cdot$ *t* **in**
addition-assoc[*of aug-pm-set z i* (*s-set R s A*) *f*],
assumption+, *simp*,
thin-tac a $_s\cdot$ *t* $_f$+ *b* $_s\cdot$ *sa* $_f$+ *b* $_s\cdot$ *t* = *a* $_s\cdot$ *t* $_f$+ (*b* $_s\cdot$ *sa* $_f$+ *b* $_s\cdot$ *t*),
frule-tac x = *b* $_s\cdot$ *sa* **and** *y* = *b* $_s\cdot$ *t* **in**
bpp-closed [*of* (*aug-pm-set z i* (*s-set R s A*)) *f*], *assumption*+,
subst addition-assoc, *assumption*+, *simp*)
done

lemma (**in** *Ring*) *sop-distr1*:⟦*commute-bpp f* (*aug-pm-set z i* (*s-set R s A*));
assoc-bpp (*aug-pm-set z i* (*s-set R s A*)) *f*; *ipp-cond1* (*s-set R s A*) *i*;
ipp-cond2 z (*s-set R s A*) *i f*; *ipp-cond3 z i*;
inv-ipp z i f (*s-set R s A*); *zeroA z i f* (*s-set R s A*) *z*;
sop-distr1 R s f i (*s-set R s A*) *z*;
sop-distr2 R s f i (*s-set R s A*) *z*;
sop-assoc R s (*aug-pm-set z i* (*s-set R s A*));
sop-inv R s i (*s-set R s A*);
sop-one R s (*aug-pm-set z i* (*s-set R s A*)); *sop-z R s z*⟧ ⟹
∀ *a*∈*carrier R*. ∀ *b*∈*carrier R*.
∀ *x*∈ *addition-set f* (*aug-pm-set z i* (*s-set R s A*)).
(*a* ± *b*) $_s\cdot$ *x* = *a* $_s\cdot$ *x* $_f$+ (*b* $_s\cdot$ *x*)
apply (*rule ballI*)+
apply (*simp add*:*addition-set-def*)
apply (*erule exE*, *erule conjE*, *erule exE*, *simp*,
thin-tac xa = *add-set f* (*aug-pm-set z i* (*s-set R s A*)) *n*)
apply (*simp add*:*sop-distr1Tr*)
done

definition
fgmodule-condition ::[('*r*, '*m*) *Ring-scheme*, '*a* ⇒ '*a* ⇒ '*a*, '*a* ⇒ '*a*,
'*r* ⇒ '*a* ⇒ '*a*, '*a set*, '*a*] ⇒ *bool* **where**
fgmodule-condition R f i s A z ⟷
commute-bpp f (*aug-pm-set z i* (*s-set R s A*)) ∧
assoc-bpp (*aug-pm-set z i* (*s-set R s A*)) *f* ∧
ipp-cond1 (*s-set R s A*) *i* ∧ *ipp-cond2 z* (*s-set R s A*) *i f* ∧
ipp-cond3 z i ∧ *inv-ipp z i f* (*s-set R s A*) ∧
zeroA z i f (*s-set R s A*) *z* ∧ *sop-distr1 R s f i* (*s-set R s A*) *z* ∧
sop-distr2 R s f i (*s-set R s A*) *z* ∧
sop-assoc R s (*aug-pm-set z i* (*s-set R s A*)) ∧
sop-inv R s i (*s-set R s A*) ∧
sop-one R s (*aug-pm-set z i* (*s-set R s A*)) ∧ *sop-z R s z*

lemma (**in** *Ring*) *sop-closed1*:⟦*fgmodule-condition R f i s A z*; *r* ∈ *carrier R*;
x ∈ *addition-set f* (*aug-pm-set z i* (*s-set R s A*))⟧ ⟹
r ₛ· *x* ∈ *addition-set f* (*aug-pm-set z i* (*s-set R s A*))
apply(*simp add*:*fgmodule-condition-def*, (*erule conjE*)+)
apply (*simp add*:*sop-closed*)
done

lemma (**in** *Ring*) *fgmodule-is-module*:*fgmodule-condition R f i s A z*
⟹ *R module* (*fgmodule R A z i f s*)
apply (*simp add*:*fgmodule-condition-def*, (*erule conjE*)+)
apply (*rule Module.intro*)
apply (*frule fag-aGroup* [*of f z i s-set R s A*], *assumption*+)
apply (*simp add*:*fag-gen-by-def fgmodule-def*, *simp add*:*aGroup-def*)
apply (*rule allI*, *rule impI*)
apply (*simp add*:*zeroA-def*)

apply (*rule Module-axioms.intro*)
apply (*rule Ring-axioms*)

apply (*simp add*:*fgmodule-carr*, *subst fgmodule-def*, *simp*,
simp add:*sop-closed*)

apply (*simp add*:*fgmodule-carr*, (*subst fgmodule-def*)+, *simp*,
simp add:*sop-closed*,
cut-tac ring-is-ag,
frule-tac x = *a* **and** *y* = *b* **in** *aGroup.ag-pOp-closed*, *assumption*+, *simp*)
apply (*simp add*:*sop-distr1*)

apply (*simp add*:*fgmodule-carr*, (*subst fgmodule-def*)+, *simp*,
simp add:*sop-closed bpp-closed*)
apply (*simp add*:*sop-distr2-def*)

apply (*frule-tac x* = *a* **and** *y* = *b* **in** *ring-tOp-closed*, *assumption*+,
simp add:*fgmodule-carr*, (*subst fgmodule-def*)+, *simp*,
simp add:*sop-closed bpp-closed*)

apply (*simp add*:*sop-assoc*)

apply (*cut-tac ring-one*)
apply (*simp add*:*fgmodule-carr*, (*subst fgmodule-def*)+, *simp*)
apply (*simp add*:*sop-one*)
done

lemma (**in** *Ring*) *a-in-carr-fgmodule*:$a \in A$
$\Longrightarrow$ $a \in$ *carrier* (*fgmodule R A z i f s*)
apply (*simp add*:*fgmodule-carr*)
apply (*cut-tac addition-inc-add0*[*of aug-pm-set z i* (*s-set R s A*)],
rule subsetD[*of aug-pm-set z i* (*s-set R s A*)
addition-set f (*aug-pm-set z i* (*s-set R s A*)) *a*], *assumption*)
apply (*simp add*:*aug-pm-set-def s-set-def*)
done

6.5 A fgmodule and a free module

lemma (**in** *Ring*) *fg-zeroTr*:⟦*fgmodule-condition R f i s A z*; $a \in A$⟧ $\Longrightarrow$
$\mathbf{0}_{\ s} \cdot a = z$
apply (*frule fgmodule-is-module* [*of f i s A z*])
apply (*frule a-in-carr-fgmodule*[*of a A z i f s*])
apply (*frule Module.sc-0-m* [*of fgmodule R A z i f s R a*],
simp add:*a-in-carr-fgmodule*)
apply (*cut-tac ring-zero*,
simp add:*fgmodule-def*)
done

lemma (**in** *Ring*) *fg-genTr0*:⟦*fgmodule-condition R f i s A z*;
$x \in$ *aug-pm-set z i* (*s-set R s A*)⟧ $\Longrightarrow$
$x \in$ *linear-span R* (*fgmodule R A z i f s*) (*carrier R*) *A*
apply (*simp add*:*aug-pm-set-def s-set-def*)
apply (*case-tac* $x = z$)
apply *simp*
apply (*simp add*:*linear-span-def*)
apply (*case-tac* $A = \{\}$, *simp*, *simp add*:*fgmodule-def*)
apply *simp*
apply (*frule nonempty-ex*[*of A*], *thin-tac* $A \neq \{\}$)
apply (*erule exE*)
apply (*subgoal-tac* $(\lambda j \in \{j.\ j \leq (0::nat)\}.\ xa) \in \{j.\ j \leq (0::nat)\} \to A$)
apply (*subgoal-tac* $(\lambda k \in \{j.\ j \leq (0::nat)\}.\ \mathbf{0}) \in \{j.\ j \leq (0::nat)\} \to$
carrier R)
apply (*subgoal-tac z = l-comb R* (*fgmodule R A z i f s*) *0*
$(\lambda k \in \{j.\ j \leq (0::nat)\}.\ \mathbf{0})$ $(\lambda j \in \{j.\ j \leq (0::nat)\}.\ xa)$) **apply** *blast*
apply (*simp add*:*l-comb-def*)
apply (*frule-tac a1 = xa* **in** *fg-zeroTr* [*THEN sym*, *of f i s A z*],
assumption+)
apply (*frule-tac a = xa* **in** *a-in-carr-fgmodule* [*of - A z i f s*])
apply (*frule fgmodule-is-module* [*of f i s A z*])

apply (*cut-tac ring-zero*, *simp add*:*fgmodule-def*)
apply (*simp add*:*ring-zero*)
apply (*simp*)
apply (*case-tac* $x \in A$,
frule-tac $x = x$ **and** $A = A$ **in** *nonempty*,
simp add:*linear-span-def*)
apply (*subgoal-tac* $(\lambda j{\in}\{j.\ j \leq (0::nat)\}.\ x) \in \{j.\ j \leq (0::nat)\} \to A$)
apply (*subgoal-tac* $(\lambda k{\in}\{j.\ j \leq (0::nat)\}.\ 1_r) \in \{j.\ j \leq (0::nat)\} \to$
carrier R)
apply (*subgoal-tac* x = *l-comb R* (*fgmodule R A z i f s*) *0*
$(\lambda k{\in}\{j.\ j \leq (0::nat)\}.\ 1_r)$ $(\lambda j{\in}\{j.\ j \leq (0::nat)\}.\ x)$)
apply *blast*
apply (*simp add*:*l-comb-def*)
apply (*frule fgmodule-is-module* [*of f i s A z*])
apply (*frule-tac* $a = x$ **in** *a-in-carr-fgmodule*[*of - A z i f s*])
apply (*simp add*:*fgmodule-def*, *fold fgmodule-def*,
simp add:*ring-one*, *simp add*:*fgmodule-condition-def*,
(*erule conjE*)+, *simp add*:*sop-one*)

apply (*simp add*:*ring-one*)
apply (*simp*)

apply (*case-tac* $\exists\, r{\in}$*carrier R*. $\exists\, a{\in}A.\ x = r\ {}_s{\cdot}\ a$)
apply ((*erule bexE*)+, *simp*)
apply (*frule-tac* $x = a$ **and** $A = A$ **in** *nonempty*, *simp add*:*linear-span-def*)
apply (*subgoal-tac* $(\lambda j{\in}\{j.\ j \leq (0::nat)\}.\ a) \in \{j.\ j \leq (0::nat)\} \to A$)
apply (*subgoal-tac* $(\lambda k{\in}\{j.\ j \leq (0::nat)\}.\ r) \in \{j.\ j \leq (0::nat)\} \to$
carrier R)
apply (*subgoal-tac* $r\ {}_s{\cdot}\ a$ = *l-comb R* (*fgmodule R A z i f s*) *0*
$(\lambda k{\in}\{j.\ j \leq (0::nat)\}.\ r)$ $(\lambda j{\in}\{j.\ j \leq (0::nat)\}.\ a)$)
apply *blast*
apply (*frule-tac* $a = a$ **in** *a-in-carr-fgmodule*[*of - A z i f s*])
apply (*simp add*:*l-comb-def fgmodule-def*)
apply (*simp*)
apply (*simp*)

apply (*simp add*:*minus-set-def*,
thin-tac $x \neq z$, *thin-tac* $x \notin A$,
thin-tac $\forall\, r{\in}$*carrier R*. $\forall\, a{\in}A.\ x \neq r\ {}_s{\cdot}\ a$)
apply (*erule bexE*)
apply *simp*
apply (*erule disjE*)
apply ((*erule bexE*)+, *simp*)
apply (*frule fgmodule-is-module* [*of f i s A z*])
apply (*frule-tac* $a = r$ **and** $m = a$ **in**
Module.sc-minus-am1[*of fgmodule R A z i f s R*], *assumption*+)
apply (*frule-tac* $a = a$ **in** *a-in-carr-fgmodule*[*of - A z i f s*], *assumption*)
apply (*cut-tac ring-is-ag*,
frule-tac $x = r$ **in** *aGroup.ag-mOp-closed*, *assumption*+)

apply (*frule-tac a = a* **in** *a-in-carr-fgmodule*[*of - A z i f s*])
apply (*simp add*:*fgmodule-def*, *fold fgmodule-def*,
simp add:*sop-closed1*)
apply (*frule-tac x = a* **and** *A = A* **in** *nonempty*, *simp add*:*linear-span-def*)
apply (*subgoal-tac* ($\lambda j \in \{j.\ j \leq (0::nat)\}.\ a$) $\in \{j.\ j \leq (0::nat)\} \to A$)
apply (*subgoal-tac* ($\lambda k \in \{j.\ j \leq (0::nat)\}.\ (-_a\ r)$) $\in \{j.\ j \leq (0::nat)\} \to$
carrier R)
apply (*subgoal-tac* $(-_a\ r)\ {}_s\cdot\ a$ = *l-comb R* (*fgmodule R A z i f s*) *0*
($\lambda k \in \{j.\ j \leq (0::nat)\}.\ (-_a\ r)$) ($\lambda j \in \{j.\ j \leq (0::nat)\}.\ a$))
apply *blast*
apply (*simp add*:*l-comb-def fgmodule-def*)
apply (*simp*)
apply (*simp*)

apply (*frule-tac x = y* **and** *A = A* **in** *nonempty*, *simp add*:*linear-span-def*)
apply (*subgoal-tac* ($\lambda j \in \{j.\ j \leq (0::nat)\}.\ y$) $\in \{j.\ j \leq (0::nat)\} \to A$)
apply (*subgoal-tac* ($\lambda k \in \{j.\ j \leq (0::nat)\}.\ (-_a\ 1_r)$) $\in \{j.\ j \leq (0::nat)\} \to$
carrier R)
apply (*subgoal-tac* $_i-\ y$ = *l-comb R* (*fgmodule R A z i f s*) *0*
($\lambda k \in \{j.\ j \leq (0::nat)\}.\ (-_a\ 1_r)$) ($\lambda j \in \{j.\ j \leq (0::nat)\}.\ y$))
apply *blast*
apply (*simp add*:*l-comb-def*)
apply (*frule fgmodule-is-module* [*of f i s A z*])
apply (*frule-tac a = y* **in** *a-in-carr-fgmodule* [*of - A z i f s*])
apply (*cut-tac ring-one* , *cut-tac ring-is-ag*,
frule aGroup.ag-mOp-closed [*of R* 1_r], *assumption*)
apply (*simp add*:*Module.sc-minus-am1*[*THEN sym*, *of fgmodule R A z i f s R*])
apply (*simp add*:*Module.sprod-one*)
apply (*simp add*:*fgmodule-def*)
apply (*rule Pi-I*, *simp*,
cut-tac ring-one, *cut-tac ring-is-ag*,
simp add:*aGroup.ag-mOp-closed*[*of R*])
apply (*simp*)
done

lemma (**in** *Ring*) *fg-genTr*:*fgmodule-condition R f i s A z* $\Longrightarrow$
$\forall x.\ x \in$ (*add-set f* (*aug-pm-set z i* (*s-set R s A*)) *n*) $\longrightarrow$
$x \in$ *linear-span R* (*fgmodule R A z i f s*) (*carrier R*) *A*
apply (*induct-tac n*)
apply (*rule allI*, *rule impI*, *simp*)
apply (*simp add*:*fg-genTr0*)
apply (*rule allI*, *rule impI*, *simp*)
apply (*erule bexE*)+
apply (*drule-tac x = sa* **in** *spec*)
apply *simp*
apply (*frule-tac x = t* **in** *fg-genTr0*[*of f i s A z*], *assumption*+)
apply (*cut-tac whole-ideal*)
apply (*frule fgmodule-is-module*[*of f i s A z*])

apply (*frule-tac* $a = sa$ **and** $b = t$ **in** *Module.linear-span-pOp-closed*[*of*
fgmodule R A z i f s R carrier R A], *assumption*+,
rule subsetI, *simp add*:*a-in-carr-fgmodule*, *assumption*+)
apply (*cut-tac* $n = n$ **in** *addition-inc-add*[*of f aug-pm-set z i* (*s-set R s A*)],
cut-tac addition-inc-add0[*of aug-pm-set z i* (*s-set R s A*) *f*],
frule-tac $c = sa$ **and** $A = add\text{-}set\ f\ (aug\text{-}pm\text{-}set\ z\ i\ (s\text{-}set\ R\ s\ A))\ n$
in *subsetD*[*of* - *addition-set f* (*aug-pm-set z i* (*s-set R s A*))],
assumption+,
frule-tac $c = t$ **and** $A = aug\text{-}pm\text{-}set\ z\ i\ (s\text{-}set\ R\ s\ A)$
in *subsetD*[*of* - *addition-set f* (*aug-pm-set z i* (*s-set R s A*))],
assumption+)
apply (*simp add*:*fgmodule-def*)
done

lemma (**in** *Ring*) *generator-of-fgm*:*fgmodule-condition R f i s A z* $\Longrightarrow$
generator R (*fgmodule R A z i f s*) *A*
apply (*cut-tac whole-ideal*)
apply (*simp add*:*generator-def*)
apply (*frule fgmodule-is-module* [*of f i s A z*])
apply (*rule conjI*)
apply (*rule subsetI*, *simp add*:*a-in-carr-fgmodule*)
apply (*frule Module.linear-span-sub*[*of fgmodule R A z i f s R*
carrier R A], *assumption*+)
apply (*rule subsetI*,
simp add:*a-in-carr-fgmodule*)
apply (*rule equalityI*, *assumption*,
thin-tac linear-span R (*fgmodule R A z i f s*) (*carrier R*) *A*
$\subseteq$ *carrier* (*fgmodule R A z i f s*))
apply (*rule subsetI*)
apply (*simp add*:*fgmodule-carr*, *simp add*:*addition-set-def*)
apply (*erule exE*, *erule conjE*, *erule exE*, *simp*)
apply (*simp add*: *fg-genTr*)
done

lemma (**in** *Ring*) *fg-freeTr1*:$[\![$*R module M*; *free-generator R M A*;
R module fgmodule R A z i f s; *free-generator R* (*fgmodule R A z i f s*) *A*;
$g \in mHom\ R\ M\ (fgmodule\ R\ A\ z\ i\ f\ s)$; $\forall x{\in}A.\ g\ x = x$$]\!] \Longrightarrow$
$\forall fa\ sa.\ fa \in \{j.\ j \leq (n{::}nat)\} \to A \wedge sa \in \{j.\ j \leq n\} \to carrier\ R \longrightarrow$
l-comb R (*fgmodule R A z i f s*) *n sa* (*cmp g fa*) =
l-comb R (*fgmodule R A z i f s*) *n sa fa*
apply (*induct-tac n*, (*rule allI*)+, *rule impI*, *erule conjE*)
apply (*simp add*:*l-comb-def*)
apply (*simp add*:*cmp-def*)
apply ((*rule allI*)+, *rule impI*, *erule conjE*)
apply (*frule-tac* $f = fa$ **and** $n = n$ **and** $A = A$ **in** *func-pre*,
frule-tac $f = sa$ **and** $n = n$ **and** $A = carrier\ R$ **in** *func-pre*)
apply (*drule-tac* $x = fa$ **in** *spec*,
drule-tac $x = sa$ **in** *spec*, *simp*)

apply (*frule-tac* $s = sa$ **and** $f = fa$ **and** $n = n$ **in** *Module.l-comb-Suc*[*of*
fgmodule R A z i f s R A carrier R],
rule subsetI,*simp add*:*a-in-carr-fgmodule*,
cut-tac whole-ideal, *simp*, *assumption*+, *simp*)
apply (*frule-tac* $s = sa$ **and** $f = cmp\ g\ fa$ **and** $n = n$ **in** *Module.l-comb-Suc*[*of*
fgmodule R A z i f s R A carrier R],
rule subsetI,*simp add*:*a-in-carr-fgmodule*,
cut-tac whole-ideal, *simp*, *simp*,
rule Pi-I, *simp add*:*cmp-def*,
frule-tac $x = x$ **and** $f = fa$ **and** $A = \{j.\ j \leq Suc\ n\}$ **and** $B = A$
in *funcset-mem*, *simp*+)
apply (*frule-tac* $x = Suc\ n$ **and** $f = fa$ **and** $A = \{j.\ j \leq Suc\ n\}$ **and** $B = A$
in *funcset-mem*, *simp*+, *simp add*:*cmp-def*)
done

lemma (**in** *Ring*) *fg-freeTr*:⟦*R module M*; *free-generator R M A*;
R module fgmodule R A z i f s;
free-generator R (*fgmodule R A z i f s*) *A*;
$g \in mHom\ R\ M$ (*fgmodule R A z i f s*); $\forall x{\in}A.\ g\ x = x$;
$fa \in \{j.\ j \leq (n::nat)\} \to A$; $sa \in \{j.\ j \leq n\} \to carrier\ R$⟧ ⟹
l-comb R (*fgmodule R A z i f s*) *n sa* (*cmp g fa*) =
l-comb R (*fgmodule R A z i f s*) *n sa fa*
apply (*simp add*:*fg-freeTr1*)
done

lemma (**in** *Ring*) *fg-free1*:⟦ $A \neq \{\}$; *fgmodule-condition R f i s A z*;
free-generator R (*fgmodule R A z i f s*) *A*; *R module M*;
free-generator R M A⟧ ⟹ $M \cong_R$ (*fgmodule R A z i f s*)
apply (*subgoal-tac* $(\lambda x{\in}A.\ x) \in A \to carrier$ (*fgmodule R A z i f s*))
prefer *2*
apply (*rule Pi-I*, *simp*, *simp add*:*a-in-carr-fgmodule*)
apply (*frule fgmodule-is-module* [*of f i s A z*])
apply (*frule Module.exist-extension-mhom*[*of M R fgmodule R A z i f s A*
$\lambda x{\in}A.\ x$], *assumption*+)
apply (*erule bexE*)
apply (*thin-tac* $(\lambda x{\in}A.\ x) \in A \to carrier$ (*fgmodule R A z i f s*))
apply (*simp add*:*misomorphic-def*)
apply (*subgoal-tac bijec*$_{M,(fgmodule\ R\ A\ z\ i\ f\ s)}$ *g*, *blast*)
apply (*simp add*:*bijec-def*)
apply (*rule conjI*)
apply (*simp add*:*injec-def*)
apply (*rule conjI*, *simp add*:*mHom-def*)
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*simp add*:*ker-def*, *erule conjE*)
apply (*frule Module.free-generator-generator*[*of M R A*], *assumption*)
apply (*simp add*:*generator-def*, *erule conjE*)
apply (*rotate-tac* −*1*, *frule sym*,
thin-tac linear-span R M (*carrier R*) *A* = *carrier M*,

frule-tac $a = x$ **and** $A =$ *carrier* M **and**
$B =$ *linear-span* R M (*carrier* R) A **in** *eq-set-inc*, *assumption*+,
thin-tac carrier $M =$ *linear-span* R M (*carrier* R) A,
simp add:*linear-span-def*)
apply (*erule exE*, (*erule bexE*)+)
apply (*cut-tac whole-ideal*)
apply (*frule-tac* $s = sa$ **and** $n = n$ **and** $f = fa$ **and** $H = A$ **in**
Module.same-together[*of M R carrier R*], *assumption*+)
apply ((*erule bexE*)+, *erule conjE*)
apply (*fold l-comb-def*[*of R M*], *simp*)
apply (*frule-tac* $f = g$ **and** $s = t$ **and** $n =$ *card* (fa ‘ $\{j.\ j \leq n\}$) $-$ *Suc 0*
and $g = ga$ **in** *Module.linmap-im-lincomb*[*of M R carrier R*
fgmodule R A z i f s - *A*], *assumption*+)
apply (*frule-tac* $f = fa$ **and** $A = \{j.\ j \leq n\}$ **in** *img-subset*[*of* - - *A*],
frule-tac $f = ga$ **and** $A = \{j.\ j \leq$ *card* (fa ‘ $\{j.\ j \leq n\}$) $-$ *Suc 0*$\}$
and $B = fa$ ‘ $\{j.\ j \leq n\}$ **and** *?B1.0* $= A$ **in** *extend-fun*, *assumption*+)
apply (*rotate-tac* -1, *frule sym*,
thin-tac g (*l-comb* R M (*card* (fa ‘ $\{j.\ j \leq n\}$) $-$ *Suc 0*) t ga) =
l-comb R (*fgmodule R A z i f s*) (*card* (fa ‘ $\{j.\ j \leq n\}$) $-$ *Suc 0*) t
(*cmp g ga*), *simp*)

apply (*frule-tac* $g = g$ **and** $fa = ga$ **and** $sa = t$ **and**
$n =$ *card* (fa ‘ $\{j.\ j \leq n\}$) $-$ *Suc 0* **in** *fg-freeTr*[*of M A z i f s*],
assumption+)
apply (*frule-tac* $f = fa$ **and** $A = \{j.\ j \leq n\}$ **in** *img-subset*[*of* - - *A*],
frule-tac $f = ga$ **and** $A = \{j.\ j \leq$ *card* (fa ‘ $\{j.\ j \leq n\}$) $-$ *Suc 0*$\}$
and $B = fa$ ‘ $\{j.\ j \leq n\}$ **and** *?B1.0* $= A$ **in** *extend-fun*, *assumption*+)
apply *simp*
apply (*cut-tac* $k = n$ **in** *finite-Collect-le-nat*,
cut-tac $k =$ *card* (fa ‘ $\{j.\ j \leq n\}$) $-$ *Suc 0* **in** *finite-Collect-le-nat*,
cut-tac $F = \{j.\ j \leq n\}$ **and** $h = fa$ **in** *finite-imageI*,
assumption)
apply (*frule-tac* $f = fa$ **and** $A = \{j.\ j \leq n\}$ **and** $B = A$ **and** $a = 0$ **in**
mem-in-image, *simp*,
frule-tac $x = fa\ 0$ **and** $A = fa$ ‘ $\{j.\ j \leq n\}$ **in** *nonempty*,
frule-tac $A = fa$ ‘ $\{j.\ j \leq n\}$ **in** *nonempty-card-pos*, *assumption*)
apply (*frule-tac* $A = fa$ ‘ $\{j.\ j \leq n\}$ **and**
$n =$ *card* (fa ‘ $\{j.\ j \leq n\}$) $-$ *Suc 0* **and** $f = ga$ **in** *Nset2finite-inj*,
simp, *assumption*)
apply (*frule Module.free-generator-sub*[*of fgmodule R A z i f s R A*],
assumption)
apply (*frule-tac* $s = t$ **and** $m = ga$ **and** $n =$ *card* (fa ‘ $\{j.\ j \leq n\}$) $-$ *Suc 0*
in *Module.unique-expression1*[*of fgmodule R A z i f s R A*],
assumption+)
apply (*frule-tac* $f = fa$ **and** $A = \{j.\ j \leq n\}$ **in** *img-subset*[*of* - - *A*],
rule-tac $f = ga$ **and** $A = \{j.\ j \leq$ *card* (fa ‘ $\{j.\ j \leq n\}$) $-$ *Suc 0*$\}$
and $B = fa$ ‘ $\{j.\ j \leq n\}$ **and** *?B1.0* $= A$ **in** *extend-fun*, *assumption*+)
apply (*rule-tac* $s = t$ **and** $n =$ *card* (fa ‘ $\{j.\ j \leq n\}$) $-$ *Suc 0* **and**
$m = ga$ **in** *Module.linear-comb0-1*[*of M R A*], *assumption*+,

```
      simp)
   apply (frule-tac f = fa and A = {j. j ≤ n} in img-subset[of - - A],
      rule-tac f = ga and A = {j. j ≤ card (fa ‘ {j. j ≤ n}) − Suc 0}
      and B = fa ‘ {j. j ≤ n} and ?B1.0 = A in extend-fun, assumption+)

apply (simp add:ker-def,
     frule Module.module-is-ag[of M],
     simp add: aGroup.ag-inc-zero[of M],
     simp add:Module.mHom-0)

apply (simp add:surjec-def,
      rule conjI, simp add:mHom-def)
 apply (rule surj-to-test, simp add:mHom-def aHom-def)
 apply (rule ballI)
 apply (frule Module.free-generator-generator[of fgmodule R A z i f s R A],
      assumption+)
 apply (cut-tac generator-def[of R fgmodule R A z i f s A], simp,
      erule conjE, rotate-tac −1, frule sym,
      thin-tac linear-span R (fgmodule R A z i f s) (carrier R) A =
        carrier (fgmodule R A z i f s), simp,
      thin-tac carrier (fgmodule R A z i f s) =
        linear-span R (fgmodule R A z i f s) (carrier R) A,
      thin-tac A ⊆ linear-span R (fgmodule R A z i f s) (carrier R) A,
      thin-tac generator R (fgmodule R A z i f s) A ≡ True)
 apply (simp add:linear-span-def, erule exE, (erule bexE)+)
 apply (frule-tac g1 = g and fa1 = fa and n1 = n and sa1 = sa in
      fg-freeTr[THEN sym, of M A z i f s], assumption+, simp,
      thin-tac l-comb R (fgmodule R A z i f s) n sa fa =
      l-comb R (fgmodule R A z i f s) n sa (cmp g fa))
 apply (frule-tac f1 = g and s1 = sa and g1 = fa and n1 = n in
      Module.linmap-im-lincomb[THEN sym, of M R carrier R
           fgmodule R A z i f s -  A],
      simp add:whole-ideal, assumption+,
      simp add:Module.free-generator-sub, assumption+, simp)
 apply (cut-tac whole-ideal,
      frule-tac s = sa and n = n and m = fa in
      Module.l-comb-mem[of M R carrier R A], assumption+,
      simp add:Module.free-generator-sub, assumption+)
 apply blast
done

lemma (in Ring) fg-free:⟦fgmodule-condition R f i s A z;
     free-generator R (fgmodule R A z i f s) A; R module M;
     free-generator R M A⟧ ⟹ M ≅_R (fgmodule R A z i f s)
apply (case-tac A = {})
apply (simp add:free-generator-def generator-def linear-span-def)
 apply (erule conjE)+
apply (cut-tac fgmodule-is-module[of f i s A z])
apply (rule Module.Modules-single-carrier-isom[of M R fgmodule R {} z i f s],
```

assumption+, *simp*, *rule sym*, *assumption*, *rule sym*, *assumption*, *simp*)
apply (*simp add*: *fg-free1*)
done

6.6 Direct sum, again

definition
miota :: [('*r*, '*m*) *Ring-scheme*, ('*a*, '*r*, '*m1*) *Module-scheme*,
('*a*, '*r*, '*m1*) *Module-scheme*] ⇒ '*a* ⇒ '*a* **where**
miota R M1 M = (λ*x*∈*carrier M1*. *x*)

definition
msubmodule ::[('*r*, '*m*) *Ring-scheme*, ('*a*, '*r*, '*m1*) *Module-scheme*,
('*a*, '*r*, '*m1*) *Module-scheme*] ⇒ *bool* **where**
msubmodule R M M1 ⟷ *miota R M1 M* ∈ *mHom R M1 M* ∧
(*carrier M1*) ⊆ (*carrier M*)

definition
ds2 :: [('*r*, '*m*) *Ring-scheme*, ('*a*, '*r*, '*m1*) *Module-scheme*,
('*a*, '*r*, '*m1*) *Module-scheme*, ('*a*, '*r*, '*m1*) *Module-scheme*] ⇒ *bool* **where**

ds2 R M M1 M2 ⟷ *R module M* ∧ *msubmodule R M M1* ∧ *msubmodule R M M2* ∧
(∀ *x*∈*carrier M*. ∃ *m1*∈*carrier M1*. ∃ *m2*∈*carrier M2*. *x* = *m1* $\pm_M$ *m2*) ∧

(*carrier M1*) ∩ (*carrier M2*) = {$\mathbf{0}_M$}

abbreviation
DS2 ((4-/ $\bigoplus$ -,- -) [92,93,92,92]92) **where**
M1 $\bigoplus_{R,M}$ *M2* == *ds2 R M M1 M2*

lemma (**in** *Ring*) *ds2-commute*:⟦*R module M1*; *R module M2*; *R module M*;
M1 $\bigoplus_{R,M}$ *M2*⟧ ⟹ *M2* $\bigoplus_{R,M}$ *M1*
apply (*simp add*:*ds2-def*)
apply (*erule conjE*)+
apply (*subst Int-commute*, *simp*)
apply (*rule ballI*,
drule-tac x = *x* **in** *bspec*, *assumption*,
(*erule bexE*)+)
apply (*simp add*:*msubmodule-def*, (*erule conjE*)+,
frule-tac c = *m1* **in** *subsetD*[*of carrier M1 carrier M*], *assumption*+,
frule-tac c = *m2* **in** *subsetD*[*of carrier M2 carrier M*], *assumption*+)
apply (*frule Module.module-is-ag*[*of M R*],
frule-tac x = *m1* **and** *y* = *m2* **in** *aGroup.ag-pOp-commute*, *assumption*+,
simp)
apply *blast*
done

lemma (**in** *Ring*) *msub-addition*:⟦*R module M*; *R module M1*; *msubmodule R M M1*;
x ∈ *carrier M1*; *y* ∈ *carrier M1*⟧ ⟹ $x \pm_{M1} y = x \pm_M y$
apply (*simp add*:*msubmodule-def*, (*erule conjE*)+)
apply (*frule Module.mHom-add*[*of M1 R M miota R M1 M x y*], *assumption*+)
apply (*frule Module.module-is-ag*[*of M1*],
frule aGroup.ag-pOp-closed[*of M1 x y*], *assumption*+,
simp add:*miota-def*)
done

lemma (**in** *Ring*) *msub-mOp*:⟦*R module M*; *R module M1*; *msubmodule R M M1*;
x ∈ *carrier M1*⟧ ⟹ $-_{aM1}\ x\ = -_{aM}\ x$
apply (*simp add*:*msubmodule-def*, (*erule conjE*)+)
apply (*frule Module.module-is-ag*[*of M1*],
frule-tac x = *x* **in** *aGroup.ag-mOp-closed*[*of M1*], *assumption*+)
apply (*frule Module.mHom-inv*[*of M1 R M x miota R M1 M*], *assumption*+,
simp add:*miota-def*)
done

lemma (**in** *Ring*) *msub-sprod*:⟦*R module M*; *R module M1*; *msubmodule R M M1*;
a ∈ *carrier R*; *x* ∈ *carrier M1*⟧ ⟹ $a \cdot_{sM1} x = a \cdot_{sM} x$
apply (*simp add*:*msubmodule-def*, (*erule conjE*)+)
apply (*frule Module.mHom-lin*[*of M1 R M x miota R M1 M a*], *assumption*+)
apply (*frule Module.sc-mem*[*of M1 R a x*], *assumption*+)
apply (*simp add*:*miota-def*)
done

lemma (**in** *Ring*) *msub-submodule*:⟦*R module M*; *R module M1*; *msubmodule R M M1*⟧
⟹ *submodule R M* (*carrier M1*)
apply (*simp add*:*submodule-def msubmodule-def*, *erule conjE*)
apply (*rule conjI*)
apply (*frule Module.module-is-ag*[*of M R*],
frule Module.module-is-ag[*of M1 R*])
apply (*rule aGroup.asubg-test*, *assumption*+,
frule Module.module-inc-zero[*of M1 R*], *blast*)
apply ((*rule ballI*)+,
frule-tac x = *b* **in** *aGroup.ag-mOp-closed*[*of M1*], *assumption*+)
apply (*frule-tac m* = *a* **and** *n* = $-_{aM1}\ b$ **in** *Module.mHom-add*[*of M1 R M miota R M1 M*], *assumption*+,
frule-tac x = *a* **and** *y* = $-_{aM1}\ b$ **in** *aGroup.ag-pOp-closed*, *assumption*+, *simp add*:*miota-def*)
apply (*frule-tac m* = *b* **in** *Module.mHom-inv*[*of M1 R M - miota R M1 M*], *assumption*+, *simp add*:*miota-def*, *simp add*:*miota-def*)
apply ((*rule allI*)+, *rule impI*, *erule conjE*)
apply (*frule-tac m* = *m* **and** *a* = *a* **in** *Module.mHom-lin* [*of M1 R M - miota R M1 M -*], *assumption*+)
apply (*frule-tac a* = *a* **and** *m* = *m* **in** *Module.sc-mem*[*of M1 R*], *assumption*+)
apply (*simp add*:*miota-def*)

done

lemma (**in** *Ring*) *ds2-unique*:⟦*R module M*; *R module M1*; *R module M2*;
ds2 R M M1 M2; *m1* ∈ *carrier M1*; *m1'* ∈ *carrier M1*;
m2 ∈ *carrier M2*; *m2'* ∈ *carrier M2*;
m1 ±$_M$ *m2* = *m1'* ±$_M$ *m2*'⟧ ⟹ *m1* = *m1'* ∧ *m2* = *m2'*
apply (*frule msub-submodule* [*of M M1*], *assumption*+,
simp add:*ds2-def*)
apply (*frule msub-submodule* [*of M M2*], *assumption*+,
simp add:*ds2-def*)
apply (*frule Module.submodule-subset*[*of M R carrier M1*], *assumption*,
frule Module.submodule-subset[*of M R carrier M2*], *assumption*)
apply (*frule subsetD* [*of carrier M1 carrier M m1*], *assumption*+,
frule subsetD [*of carrier M1 carrier M m1'*], *assumption*+,
frule subsetD [*of carrier M2 carrier M m2*], *assumption*+,
frule subsetD [*of carrier M2 carrier M m2'*], *assumption*+)
apply (*frule-tac Module.module-is-ag*[*of M R*],
frule-tac x = *m1* **and** *y* = *m2* **in** *aGroup.ag-pOp-closed*, *assumption*+,
frule-tac x = *m1'* **and** *y* = *m2'* **in** *aGroup.ag-pOp-closed*, *assumption*+,
frule-tac x = *m2'* **in** *aGroup.ag-mOp-closed*, *assumption*+,
frule-tac x = *m2* **in** *aGroup.ag-mOp-closed*, *assumption*+)
apply (*frule-tac a* = *m1* ±$_M$ *m2* **and** *b* = *m1'* ±$_M$ *m2'* **and** *c* = −$_{aM}$ *m2* **in**
aGroup.ag-pOp-add-r, *assumption*+,
thin-tac m1 ±$_M$ *m2* = *m1'* ±$_M$ *m2'*,
simp add:*aGroup.ag-pOp-assoc*[*of M m1 m2* −$_{aM}$ *m2*]
aGroup.ag-r-inv1 aGroup.ag-r-zero,
simp add:*aGroup.ag-pOp-assoc*[*of M m1' m2'* −$_{aM}$ *m2*],
frule-tac x = *m2'* **and** *y* = −$_{aM}$ *m2* **in** *aGroup.ag-pOp-closed*,
assumption+,
frule-tac x = *m1'* **in** *aGroup.ag-mOp-closed*, *assumption*+,
frule-tac x = *m1'* **and** *y* = *m2'* ±$_M$ −$_{aM}$ *m2* **in**
aGroup.ag-pOp-closed, *assumption*+,
frule-tac a = *m1* **and** *b* = *m1'* ±$_M$ (*m2'* ±$_M$ −$_{aM}$ *m2*) **and**
c = −$_{aM}$ *m1'* **in** *aGroup.ag-pOp-add-l*[*of M*], *simp*, *assumption*+)
apply (*frule aGroup.ag-pOp-assoc*[*THEN sym*, *of M* −$_{aM}$ *m1' m1'*
(*m2'* ±$_M$ −$_{aM}$ *m2*)],
assumption+)
apply (*rotate-tac* −*6*, *frule sym*, *thin-tac m1* = *m1'* ±$_M$ (*m2'* ±$_M$ −$_{aM}$ *m2*),
thin-tac −$_{aM}$ *m1'* ±$_M$ *m1* = −$_{aM}$ *m1'* ±$_M$ (*m1'* ±$_M$ (*m2'* ±$_M$ −$_{aM}$ *m2*)),
simp, *simp add*:*aGroup.ag-l-inv1 aGroup.ag-l-zero*)
apply (*frule Module.submodule-mOp-closed* [*of M R carrier M1 m1'*],
assumption+,
frule Module.submodule-pOp-closed [*of M R carrier M1* −$_{aM}$ *m1'*
m1], *assumption*+,
frule Module.submodule-pOp-closed [*of M R carrier M2 m2'* −$_{aM}$ *m2*],
assumption+,
rule Module.submodule-mOp-closed [*of M R carrier M2 m2*], *assumption*+)
apply (*subgoal-tac* (−$_{aM}$ *m1'* ±$_M$ *m1*) ∈ *carrier M1* ∩ *carrier M2*)
prefer *2* **apply** *simp*

apply (*subgoal-tac carrier M1* $\cap$ *carrier M2* = $\{\mathbf{0}_M\}$)
apply (*frule sym*, *thin-tac* $-_{aM}$ *m1'* $\pm_M$ *m1* = *m2'* $\pm_M$ $-_{aM}$ *m2*)
apply *simp*
apply (*simp add*:*aGroup.ag-eq-diffzero*[*THEN sym*, *of M m2' m2*] *aGroup.ag-r-zero*)

apply (*simp add*:*ds2-def*)
done

lemma (**in** *Ring*) *miota-injec*:⟦*R module M*; *R module M1*; *R module M2*;
ds2 R M M1 M2; *msubmodule R M M1*⟧ ⟹
miota R M1 M ∈ *mHom R M1 M* ∧ *injec*$_{M1,M}$ (*miota R M1 M*)
apply (*rule conjI*)
apply (*simp add*:*msubmodule-def*)
apply (*simp add*:*injec-def*)
apply (*rule conjI*)
apply (*simp add*:*msubmodule-def mHom-def*)
apply (*rule equalityI*)
prefer *2*
apply (*rule subsetI*, *simp*, *simp add*:*ker-def*)
apply (*simp add*:*Module.module-inc-zero*, *simp add*:*msubmodule-def*)
apply (*erule conjE*)
apply (*simp add*:*Module.mHom-0*)
apply (*rule subsetI*)
apply (*simp add*:*ker-def*, *erule conjE*)
apply (*simp add*:*miota-def*, *simp add*:*msubmodule-def*)
apply (*erule conjE*)
apply (*frule Module.mHom-0* [*of M1 R M miota R M1 M*], *assumption*+)
apply (*frule Module.module-inc-zero* [*of M1 R*])
apply (*simp add*:*miota-def*)
done

definition
mproj1 :: [('*r*, '*m*) *Ring-scheme*, ('*a*, '*r*, '*m1*) *Module-scheme*,
('*a*, '*r*, '*m1*) *Module-scheme*, ('*a*, '*r*, '*m1*) *Module-scheme*] ⇒ '*a* ⇒ '*a* **where**
mproj1 R M1 M2 M = (λ*x*∈*carrier M*. *THE x1*. *x1* ∈ *carrier M1* ∧
(*x* $\pm_M$ ($-_{aM}$ *x1*)) ∈ *carrier M2*)

definition
mproj2 :: [('*r*, '*m*) *Ring-scheme*, ('*a*, '*r*, '*m1*) *Module-scheme*,
('*a*, '*r*, '*m1*) *Module-scheme*, ('*a*, '*r*, '*m1*) *Module-scheme*] ⇒ '*a* ⇒ '*a* **where**
mproj2 R M1 M2 M = *mproj1 R M2 M1 M*

lemma (**in** *Ring*) *ds2-components*:⟦*R module M1*; *R module M2*; *R module M*;
M1 $\bigoplus_{R,M}$ *M2*; *a* ∈ *carrier M*⟧ ⟹
∃ *a1*∈*carrier M1*. ∃ *a2*∈*carrier M2*. *a* = *a1* $\pm_M$ *a2*
by (*simp add*:*ds2-def*)

lemma (**in** *Ring*) *ds2-components1*:⟦*R module M1*; *R module M2*; *R module M*;
M1 $\bigoplus_{R,M}$ *M2*; *a* ∈ *carrier M*⟧ ⟹
∃ *a1*∈*carrier M1*. *a* $\pm_M$ $-_{aM}$ *a1* ∈ *carrier M2*
apply (*frule ds2-components*[*of M1 M2 M a*], *assumption*+, (*erule bexE*)+,
frule Module.module-is-ag[*of M R*],
unfold ds2-def, *frule conjunct2*[*THEN conjunct1*],
frule conjunct2[*THEN conjunct2*[*THEN conjunct1*]], *fold ds2-def*)
apply (*simp add*:*msubmodule-def*, (*erule conjE*)+)
apply (*frule-tac c* = *a1* **in** *subsetD*[*of carrier M1 carrier M*], *assumption*+,
frule-tac c = *a2* **in** *subsetD*[*of carrier M2 carrier M*], *assumption*+)

apply (*frule Module.module-is-ag*[*of M*],
frule-tac x = *a1* **and** *y* = *a2* **in** *aGroup.ag-pOp-commute*, *assumption*+,
simp)
apply (*frule-tac x* = *a1* **and** *y* = *a2* **in** *aGroup.ag-pOp-closed*[*of M*], *assumption*+,
frule-tac x = *a1* **in** *aGroup.ag-mOp-closed*[*of M*], *assumption*+,
frule-tac a = *a1* $\pm_M$ *a2* **and** *b* = *a2* $\pm_M$ *a1* **and** *c* = $-_{aM}$ *a1* **in**
aGroup.ag-pOp-add-r[*of M*], *assumption*+,
frule-tac x = *a2* **and** *y* = *a1* **and** *z* = $-_{aM}$ *a1* **in**
aGroup.ag-pOp-assoc[*of M*], *assumption*+,
simp add:*aGroup.ag-r-inv1 aGroup.ag-r-zero*)
apply (*thin-tac a2* $\pm_M$ *a1* ∈ *carrier M*,
thin-tac a = *a2* $\pm_M$ *a1*,
thin-tac a2 ∈ *carrier M*,
thin-tac a1 $\pm_M$ *a2* = *a2* $\pm_M$ *a1*,
frule sym, *thin-tac a2* $\pm_M$ *a1* $\pm_M$ $-_{aM}$ *a1* = *a2*)
apply (*frule-tac a* = *a2* **and** *b* = *a2* $\pm_M$ *a1* $\pm_M$ $-_{aM}$ *a1* **in**
eq-elem-in[*of - carrier M2*], *assumption*, *blast*)
done

lemma (**in** *Ring*) *mprojTr1*:⟦*R module M1*; *R module M2*; *R module M*; *ds2 R M M1 M2*;
x ∈ *carrier M* ⟧ ⟹ ∃!*x1*. *x1* ∈ *carrier M1* ∧ (*x* $\pm_M$ $-_{aM}$ *x1*) ∈ *carrier M2*
apply (*frule Module.module-is-ag*[*of M R*])
apply (*rule ex-ex1I*)
apply (*frule ds2-components1* [*of M1 M2 M x*], *assumption*+, *erule bexE*, *blast*)

apply (*simp add*:*ds2-def*, (*erule conjE*)+, *simp add*:*msubmodule-def*,
(*erule conjE*)+,
frule-tac c = *x1* **in** *subsetD*[*of carrier M1 carrier M*], *assumption*+,
frule-tac x = *x1* **in** *aGroup.ag-mOp-closed*, *assumption*+,
frule-tac c = *y* **in** *subsetD*[*of carrier M1 carrier M*], *assumption*+,
frule-tac x = *y* **in** *aGroup.ag-mOp-closed*, *assumption*+,
frule-tac ?m1.0 = *x1* **and** *?m1'* = *y* **and** *?m2.0* = *x* $\pm_M$ $-_{aM}$ *x1* **and**
?m2' = *x* $\pm_M$ $-_{aM}$ *y* **in** *ds2-unique*[*of M M1 M2*], *assumption*+)
apply (*simp add*:*ds2-def msubmodule-def*, *assumption*+)
apply (*simp add*:*aGroup.ag-pOp-commute*[*of M x*])
apply (*simp add*:*aGroup.ag-pOp-assoc*[*THEN sym*, *of M - - x*])

apply ((*subst aGroup.ag-r-inv1*, *assumption*+)+, *simp*, *simp*)
done

lemma (**in** *Ring*) *mprojTr2*:⟦*R module M1*; *R module M2*; *R module M*; *ds2 R M M1 M2*;
x ∈ *carrier M*; *x1* ∈ *carrier M1*; (*x* ±$_M$ (−$_{aM}$ *x1*)) ∈ *carrier M2*;
y1 ∈ *carrier M1*;(*x* ±$_M$ (−$_{aM}$ *y1*)) ∈ *carrier M2* ⟧ ⟹ *x1* = *y1*
apply (*frule-tac x* = *x* **in** *mprojTr1*[*of M1 M2 M*], *assumption*+)
apply *blast*
done

lemma (**in** *Ring*) *mprojTr3*:⟦*R module M1*; *R module M2*; *R module M*; *ds2 R M M1 M2*;
a ∈ *carrier M*; *a1* ∈ *carrier M1*; (*a* ±$_M$ (−$_{aM}$ *a1*)) ∈ *carrier M2*⟧ ⟹
(*THE x1*. *x1* ∈ *carrier M1* ∧ *a* ±$_M$ −$_{aM}$ *x1* ∈ *carrier M2*) = *a1*
apply (*subgoal-tac* (*THE x1*. *x1* ∈ *carrier M1* ∧ *a* ±$_M$ −$_{aM}$ *x1* ∈ *carrier M2*) ∈
carrier M1 ∧ *a* ±$_M$ −$_{aM}$ (*THE x1*. *x1* ∈ *carrier M1* ∧
a ±$_M$ −$_{aM}$ *x1* ∈ *carrier M2*) ∈ *carrier M2*,
rule mprojTr2[*of M1 M2 M a*
THE x1. *x1* ∈ *carrier M1* ∧ *a* ±$_M$ −$_{aM}$ *x1* ∈ *carrier M2 a1*], *assumption*+,
simp, *simp*, *assumption*+)
apply (*rule theI'*)
apply (*simp add*:*mprojTr1*)
done

lemma (**in** *Ring*) *mproj*:⟦*R module M1*; *R module M2*; *R module M*; *ds2 R M M1 M2*⟧
⟹ *mproj1 R M1 M2 M* ∈ *mHom R M M1*
apply (*simp add*:*mHom-def*)
apply (*rule conjI*)
apply (*simp add*:*aHom-def*)
apply (*rule conjI*)
apply (*rule Pi-I*)
apply (*simp add*:*mproj1-def*)
apply (*frule-tac a* = *x* **in** *ds2-components1*[*of M1 M2 M*], *assumption*+,
erule bexE,
frule-tac a = *x* **and** *?a1.0* = *a1* **in** *mprojTr3*[*of M1 M2 M*], *assumption*+,
simp)

apply (*simp add*:*restrict-def mproj1-def extensional-def*)

apply ((*rule ballI*)+,
frule Module.module-is-ag[*of M R*], *simp add*:*aGroup.ag-pOp-closed*,
frule-tac x = *a* **and** *y* = *b* **in** *aGroup.ag-pOp-closed*, *assumption*+,
frule-tac a = *a* **in** *ds2-components1*[*of M1 M2 M*], *assumption*+,
frule-tac a = *b* **in** *ds2-components1*[*of M1 M2 M*], *assumption*+,
frule-tac a = *a* ±$_M$ *b* **in** *ds2-components1*[*of M1 M2 M*], *assumption*+,
(*erule bexE*)+, *rename-tac b1 ab*,
frule-tac a = *a* ±$_M$ *b* **and** *?a1.0* = *ab* **in** *mprojTr3*[*of M1 M2 M*],

assumption+,
frule-tac $a = a$ **and** *?a1.0* $= a1$ **in** *mprojTr3*[*of M1 M2 M*], *assumption*+,
frule-tac $a = b$ **and** *?a1.0* $= b1$ **in** *mprojTr3*[*of M1 M2 M*], *assumption*+,
simp,
thin-tac (*THE x1*. $x1 \in$ *carrier M1* $\wedge$
$a \pm_M b \pm_M -_{aM} x1 \in$ *carrier M2*) $= ab$,
thin-tac (*THE x1*. $x1 \in$ *carrier M1* $\wedge$ $a \pm_M -_{aM} x1 \in$ *carrier M2*) $= a1$,
thin-tac (*THE x1*. $x1 \in$ *carrier M1* $\wedge$ $b \pm_M -_{aM} x1 \in$ *carrier M2*) $= b1$)
apply (*frule Module.module-is-ag*[*of M2*],
frule-tac $x = a \pm_M -_{aM} a1$ **and** $y = b \pm_M -_{aM} b1$ **in**
aGroup.ag-pOp-closed[*of M2*], *assumption*+,
frule-tac $x = a \pm_M -_{aM} a1$ **and** $y = b \pm_M -_{aM} b1$ **in**
msub-addition[*of M M2*], *assumption*+,
simp add:*ds2-def*, *assumption*+, *simp*,
thin-tac $a \pm_M -_{aM} a1 \pm_{M2} (b \pm_M -_{aM} b1) =$
$a \pm_M -_{aM} a1 \pm_M (b \pm_M -_{aM} b1)$,
unfold ds2-def, *frule conjunct2*[*THEN conjunct1*], *fold ds2-def*,
simp add:*msubmodule-def*, (*erule conjE*)+,
frule-tac $c = a1$ **in** *subsetD*[*of carrier M1 carrier M*], *assumption*+,
frule-tac $c = b1$ **in** *subsetD*[*of carrier M1 carrier M*], *assumption*+,
frule-tac $x = a1$ **in** *aGroup.ag-mOp-closed*, *assumption*+,
frule-tac $x = b1$ **in** *aGroup.ag-mOp-closed*, *assumption*+,
frule-tac $a = a$ **and** $b = -_{aM} a1$ **and** $c = b$ **and** $d = -_{aM} b1$ **in**
aGroup.pOp-assocTr43[*of M*], *assumption*+, *simp*,
thin-tac $a \pm_M -_{aM} a1 \pm_M (b \pm_M -_{aM} b1) =$
$a \pm_M (-_{aM} a1 \pm_M b) \pm_M -_{aM} b1$,
frule-tac $x = -_{aM} a1$ **and** $y = b$ **in** *aGroup.ag-pOp-commute*,
assumption+, *simp*,
frule-tac $a1 = a$ **and** $b1 = b$ **and** $c1 = -_{aM} a1$ **and** $d1 = -_{aM} b1$ **in**
aGroup.pOp-assocTr43[*THEN sym*], *assumption*+, *simp*,
thin-tac $a \pm_M (b \pm_M -_{aM} a1) \pm_M -_{aM} b1 =$
$a \pm_M b \pm_M (-_{aM} a1 \pm_M -_{aM} b1)$,
frule-tac $x1 = a1$ **and** $y1 = b1$ **in** *aGroup.ag-p-inv*[*THEN sym*, *of M*],
assumption+, *simp*,
thin-tac $-_{aM} a1 \pm_M -_{aM} b1 = -_{aM} (a1 \pm_M b1)$)
apply (*rule-tac* $x = a \pm_M b$ **and** *?x1.0* $= ab$ **and** *?y1.0* $= a1 \pm_{M1} b1$ **in**
mprojTr2[*of M1 M2 M*], *assumption*+,
frule Module.module-is-ag[*of M1*],
simp add:*aGroup.ag-pOp-closed*[*of M1*], *unfold ds2-def*,
frule conjunct2[*THEN conjunct1*], *fold ds2-def*,
subst msub-addition[*of M M1*], *assumption*+)

apply (*rule ballI*)+
apply (*simp add*:*mproj1-def*, *simp add*:*Module.sc-mem*)
apply (*frule-tac* $a = m$ **in** *ds2-components1*[*of M1 M2 M*], *assumption*+,
erule bexE,
unfold ds2-def, *frule conjunct2*[*THEN conjunct1*], *fold ds2-def*,
simp add:*msubmodule-def*, (*erule conjE*)+,
frule-tac $c = a1$ **in** *subsetD*[*of carrier M1 carrier M*], *assumption*+,

frule-tac $a = a$ **and** $m = m \pm_M -_{a\,M}\ a1$ **in** *Module.sc-mem*[*of M2 R*],
assumption+,
unfold ds2-def, *frule conjunct2*[*THEN conjunct1*],
frule conjunct2[*THEN conjunct2*[*THEN conjunct1*]], *fold ds2-def*,
simp add:*msub-sprod*[*of M M2*],
frule Module.module-is-ag[*of M*],
frule-tac $x = a1$ **in** *aGroup.ag-mOp-closed*[*of M*], *assumption*+,
simp add:*Module.sc-r-distr*,
frule Module.module-is-ag[*of M1*],
frule-tac $x = a1$ **in** *aGroup.ag-mOp-closed*[*of M1*], *assumption*+,
frule-tac $a1 = a$ **and** $x1 = -_{a\,M}\ a1$ **in** *msub-sprod*[*THEN sym*, *of M M1*],
assumption+)
apply (*simp add*:*msub-mOp*,
frule-tac $x1 = a1$ **in** *msub-mOp*[*THEN sym*, *of M M1*], *assumption*+, *simp*)
apply (
thin-tac $a \cdot_{s\,M} (-_{a\,M1}\ a1) = a \cdot_{s\,M1} (-_{a\,M1}\ a1)$,
thin-tac $-_{a\,M}\ a1 = -_{a\,M1}\ a1$,
thin-tac $-_{a\,M1}\ a1 \in carrier\ M$,
frule-tac $a = a$ **and** $m = a1$ **in** *Module.sc-mem*[*of M1 R*],
assumption+,
frule-tac $a = m$ **and** *?a1.0* $= a1$ **in** *mprojTr3*[*of M1 M2 M*],
assumption+, *simp add*:*msub-mOp*, *simp*,
thin-tac (*THE x1*. $x1 \in carrier\ M1 \wedge m \pm_M -_{a\,M}\ x1 \in carrier\ M2$) $= a1$,
frule-tac $a = a \cdot_{s\,M}\ m$ **and** *?a1.0* $= a \cdot_{s\,M1}\ a1$ **in**
mprojTr3[*of M1 M2 M*], *assumption*+,
simp add:*Module.sc-mem*, *assumption*)
apply (*simp add*:*msub-mOp msub-sprod*,
simp add:*Module.sc-minus-am*[*of M R*])
apply *simp*
done

lemma (**in** *Ring*) *mproj2*:⟦*R module M1*; *R module M2*; *R module M*; *M1* $\bigoplus_{R,M}$ *M2*⟧
$\Longrightarrow$ *mproj2 R M1 M2 M* $\in$ *mHom R M M2*
apply (*frule ds2-commute*[*of M1 M2 M*], *assumption*+)
apply (*simp add*:*mproj2-def*)
apply (*simp add*:*mproj*)
done

6.6.1 Existence of the tensor product

definition
fm-gen-by-prod :: [('*r*, '*m*) *Ring-scheme*, (('*a* $*$ '*b*), '*r*, '*m1*) *Module-scheme*,
('*a*, '*r*, '*m1*) *Module-scheme*, ('*b*, '*r*, '*m1*) *Module-scheme*] $\Rightarrow$ *bool*
((*4FM*_/ - - -) [*100*,*100*,*101*]*100*) **where**
$FM_R\ P\ M\ N \longleftrightarrow R$ *module* $P \wedge$ *free-generator R P* ($M \times_c N$)

lemma (**in** *Ring*) *free-gen-gen*:$FM_R\ P\ M\ N \Longrightarrow$ *generator R P* ($M \times_c N$)
apply (*simp add*:*fm-gen-by-prod-def*)

apply (*erule conjE*)
apply (*simp add*:*free-generator-def*)
done

lemma (**in** *Ring*) *free-gen-mem*:⟦*FM*$_R$ *P M N*; *a* ∈ (*M* ×$_c$ *N*)⟧ ⟹ *a* ∈ *carrier P*
apply (*simp add*:*fm-gen-by-prod-def*)
apply (*erule conjE*)
apply (*simp add*:*free-generator-def*, (*erule conjE*)+)
apply (*simp add*:*generator-def*)
apply (*erule conjE*)+
apply (*simp add*:*subsetD*)
done

lemma (**in** *Ring*) *mHom-lin-nsumTr*:⟦*R module M*; *R module N*; *t* ∈ *mHom R M N*⟧ ⟹
f ∈ {*j*. *j* ≤ (*n*::*nat*)} → *carrier M* ⟶ *t* (*nsum M f n*) = *nsum N* (*cmp t f*) *n*
apply (*induct-tac n*)
apply (*rule impI*, *simp add*:*cmp-def*)
apply (*rule impI*)
apply *simp*
apply (*frule-tac func-pre*, *simp*)
apply (*frule Module.module-is-ag* [*of M R*])
apply (*frule-tac n* = *n* **in** *aGroup.nsum-mem* [*of M - f*],
rule allI, *simp add*:*Pi-def*,
frule-tac x = *Suc n* **and** *A* = {*j*. *j* ≤ *Suc n*} **and** *f* = *f* **and**
B = *carrier M* **in** *funcset-mem*, *simp*)
apply (*simp add*:*Module.mHom-add cmp-def*)
done

lemma (**in** *Ring*) *mHom-lin-nsum*:⟦*R module M*; *R module N*; *t* ∈ *mHom R M N*;
f ∈ {*j*. *j* ≤ (*n*::*nat*)} → *carrier M*⟧ ⟹
t (*nsum M f n*) = *nsum N* (*cmp t f*) *n*
apply (*simp add*:*mHom-lin-nsumTr*)
done

lemma (**in** *Ring*) *module-over-zeroring*:⟦*zeroring R*; *R module M*⟧ ⟹
carrier M = {**0**$_M$}
apply (*simp add*:*zeroring-def*, *erule conjE*)
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*frule-tac m* = *x* **in** *Module.sprod-one* [*of M R*], *assumption*)
apply (*cut-tac ring-one*, *simp*)
apply (*simp add*:*Module.sc-0-m*[*of M R*])
apply (*frule Module.module-is-ag* [*of M R*])
apply (*simp add*:*aGroup.ag-inc-zero*)
done

lemma (**in** *Ring*) *submodule-over-zeroring*:⟦*zeroring R*; *R module M*;
submodule R M N⟧ ⟹ $N = \{\mathbf{0}_M\}$
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*simp add*:*submodule-def*, (*erule conjE*)+)
apply (*thin-tac* ∀ *a m*. *a* ∈ *carrier R* ∧ *m* ∈ *N* ⟶ *a* ·$_{s M}$ *m* ∈ *N*)
apply (*cut-tac module-over-zeroring* [*of M*])
apply *simp*
apply (*frule-tac A* = *N* **and** *B* = $\{\mathbf{0}_M\}$ **and** *c* = *x* **in** *subsetD*, *assumption*+)
apply (*simp*, *assumption*+)
apply (*frule Module.submodule-inc-0* [*of M R N*], *assumption*+)
apply *simp*
done

definition
Least-submodule :: [('*r*, '*m*) *Ring-scheme*, ('*a*, '*r*, '*m1*) *Module-scheme*,
'*a set*] ⇒ '*a set*
((*3LSM*_/ -/ -) [*100*,*100*,*101*]*100*) **where**
$LSM_R\ M\ T = \bigcap\{N.\ submodule\ R\ M\ N \wedge T \subseteq N\}$

lemma (**in** *Ring*) *LSM-mem*:⟦*R module M*; *T* ⊆ *carrier M*; *t* ∈ *T*⟧ ⟹
t ∈ (LSM_R *M T*)
apply (*simp add*:*Least-submodule-def*, *rule allI*, *rule impI*)
apply (*erule conjE*)
apply (*simp add*:*subsetD*)
done

lemma (**in** *Ring*) *LSM-sub-M*:⟦*R module M*; *T* ⊆ *carrier M*⟧ ⟹
(LSM_R *M T*) ⊆ *carrier M*
apply (*rule subsetI*, *simp add*:*Least-submodule-def*)
apply (*frule Module.submodule-whole*[*of M R*])
apply (*drule-tac x* = *carrier M* **in** *spec*,
simp)
done

lemma (**in** *Ring*) *LSM-sub-submodule*:⟦*R module M*; *T* ⊆ *carrier M*;
submodule R M N; *T* ⊆ *N* ⟧ ⟹ (LSM_R *M T*) ⊆ *N*
by (*rule subsetI*, *simp add*:*Least-submodule-def*)

lemma (**in** *Ring*) *LSM-inc-T*:⟦*R module M*; *T* ⊆ *carrier M*⟧ ⟹ *T* ⊆ (LSM_R *M T*)
apply (*rule subsetI*)
apply (*simp add*:*LSM-mem*)
done

lemma (**in** *Ring*) *LSM-submodule*:⟦*R module M*; *T* ⊆ *carrier M*⟧ ⟹
submodule R M (LSM_R *M T*)
apply (*frule LSM-sub-M*[*of M T*], *assumption*+)

apply (*subst Least-submodule-def*)
apply (*simp add*:*submodule-def*)
apply (*rule conjI*)
apply (*rule subsetI*, *simp*,
drule-tac a = *carrier M* **in** *forall-spec*)
apply *simp*
apply (*frule Module.submodule-whole*[*of M R*])
apply (*simp add*:*submodule-def*[*of R M carrier M*], *assumption*)

apply (*frule Module.module-is-ag* [*of M R*])
apply (*rule aGroup.asubg-test*, *assumption*+)
apply (*rule subsetI*, *simp*,
drule-tac a = *carrier M* **in** *forall-spec*, *simp*,
frule Module.submodule-whole[*of M R*],
simp add:*submodule-def*[*of R M carrier M*], *assumption*)
apply (*cut-tac x* = $\mathbf{0}_M$ **and** $A = \bigcap \{N.\ N \subseteq carrier\ M \wedge$
$M +> N \wedge (\forall a\ m.\ a \in carrier\ R \wedge m \in N \longrightarrow a \cdot_{s M} m \in N) \wedge T \subseteq N\}$
in *nonempty*)
apply *simp*
apply (*rule allI*, *rule impI*, (*erule conjE*)+,
frule-tac H = *x* **in** *aGroup.asubg-inc-zero*[*of M*], *assumption*+)

apply ((*rule ballI*)+, *simp*, *rule allI*, *rule impI*)
apply (*drule-tac x* = *x* **in** *spec*,
drule-tac x = *x* **in** *spec*)
apply *simp*
apply (*frule-tac H* = *x* **and** *x* = *b* **in** *aGroup.asubg-mOp-closed*[*of M*], *simp*+)
apply (*rule-tac H* = *x* **and** *x* = *a* **and** *y* = $-_{a M}$ *b* **in**
aGroup.asubg-pOp-closed[*of M*], *assumption*+, *simp*+)
done

lemma (**in** *Ring*) *linear-comb-memTr*:⟦*R module M*; *submodule R M N*; *T* ⊆ *N*⟧
⟹
$\forall f\ s.\ f \in \{j.\ j \leq (n::nat)\} \to T \wedge s \in \{j.\ j \leq n\} \to carrier\ R \longrightarrow$
l-comb R M n s f ∈ *N*
apply (*induct-tac n*)
apply ((*rule allI*)+, *rule impI*, (*erule conjE*)+)
apply (*simp add*:*l-comb-def del*: *Pi-split-insert-domain*)
apply (*rule Module.submodule-sc-closed*, *assumption*+)
apply (*simp add*:*Pi-def*, *simp add*:*Pi-def subsetD*)

apply ((*rule allI*)+, *rule impI*, *erule conjE*)
apply (*frule-tac f* = *f* **and** *n* = *n* **and** *A* = *T* **in** *func-pre*)
apply (*frule-tac f* = *s* **and** *n* = *n* **and** *A* = *carrier R* **in** *func-pre*)
apply (*drule-tac x* = *f* **in** *spec*,
drule-tac x = *s* **in** *spec*, *simp*)

apply (*cut-tac whole-ideal*,

frule-tac $s = s$ **and** $n = n$ **and** $f = f$ **in**
Module.l-comb-Suc[*of M R T carrier R*],
frule Module.submodule-subset[*of M R N*], *assumption*+,
rule subset-trans[*of T N carrier M*], *assumption*+, *simp*,
thin-tac l-comb R M (*Suc n*) *s f* =
l-comb R M n s f $\pm_M$ *s* (*Suc n*) $\cdot_{sM}$ *f* (*Suc n*))
apply (*rule Module.submodule-pOp-closed*[*of M R N*], *assumption*+,
rule-tac a = *s* (*Suc n*) **and** *h* = *f* (*Suc n*) **in**
Module.submodule-sc-closed[*of M R N*], *assumption*+,
simp add:*Pi-def*, *simp add*:*Pi-def subsetD*)
done

lemma (**in** *Ring*) *linear-comb-mem*:⟦*R module M*; *submodule R M N*; $T \subseteq N$;
$f \in \{j.\ j \leq (n::nat)\} \to T$; $s \in \{j.\ j \leq n\} \to$ *carrier R*⟧ ⟹
l-comb R M n s f $\in N$
apply (*simp add*:*linear-comb-memTr*)
done

lemma (**in** *Ring*) *LSM-eq-linear-span*:⟦*R module M*; $T \subseteq$ *carrier M*⟧ ⟹
(LSM_R *M T*) = *linear-span R M* (*carrier R*) *T*
apply (*cut-tac whole-ideal*)
apply (*rule equalityI*)
apply (*frule Module.linear-span-subModule*[*of M R carrier R T*], *assumption*+)
apply (*rule LSM-sub-submodule*[*of M T linear-span R M* (*carrier R*) *T*],
assumption+, *simp add*:*Module.l-span-cont-H*[*of M R T*])
apply (*frule LSM-submodule*[*of M T*], *assumption*)
apply (*frule LSM-inc-T*[*of M T*], *assumption*)
apply (*rule Module.l-span-sub-submodule*[*of M R carrier R* LSM_R *M T T*],
assumption+)
done

lemma (**in** *Ring*) *LSM-sub-ker*:⟦*R module M*; *R module N*; $T \subseteq$ *carrier M*;
$f \in$ *mHom R M N*; $T \subseteq ker_{M,N}\, f$⟧ ⟹ LSM_R *M T* $\subseteq ker_{M,N}\, f$
apply (*frule Module.mker-submodule*[*of M R N f*], *assumption*+)
apply (*rule LSM-sub-submodule*[*of M T* $ker_{M,N}\, f$], *assumption*+)
done

definition
tensor-relations1 :: [('*r*, '*m*) *Ring-scheme*, ('*a*, '*r*, '*m1*) *Module-scheme*,
('*b*, '*r*, '*m1*) *Module-scheme*, (('*a* ∗ '*b*), '*r*, '*m1*) *Module-scheme*] ⇒
('*a* ∗ '*b*) *set*
((*4TR1*/ -/ -/ -/ -) [*100*,*100*,*100*,*101*]*100*) **where**
TR1 R M N MN = {*x*. ∃ *m1*∈*carrier M*. ∃ *m2*∈*carrier M*. ∃ *n*∈*carrier N*.
x = (*m1* $\pm_M$ *m2*, *n*) $\pm_{MN}$ ($-_{aMN}$ ((*m1*, *n*) $\pm_{MN}$ (*m2*, *n*)))}

definition
tensor-relations2 :: [('*r*, '*m*) *Ring-scheme*, ('*a*, '*r*, '*m1*) *Module-scheme*,
('*b*, '*r*, '*m1*) *Module-scheme*, (('*a* ∗ '*b*), '*r*, '*m1*) *Module-scheme*] ⇒

$('a * 'b)$ *set*
$((4TR2/\ -/\ -/\ -/\ -)\ [100,100,100,\ 101]100)$ **where**
TR2 R M N MN = $\{x.\ \exists\, m{\in}carrier\ M.\ \exists\, n1{\in}carrier\ N.\ \exists\, n2{\in}carrier\ N.$
$x = (m,\ n1\ \pm_N\ n2)\ \pm_{MN}\ (-_{a}MN\ ((m,\ n1)\ \pm_{MN}\ (m,\ n2)))\}$

definition
tensor-relations3 :: $[('r,\ 'm)$ *Ring-scheme*, $('a,\ 'r,\ 'm1)$ *Module-scheme*,
$('b,\ 'r,\ 'm1)$ *Module-scheme*, $(('a * 'b),\ 'r,\ 'm1)$ *Module-scheme*$]$ $\Rightarrow$
$('a * 'b\ )$ *set*
$((4TR3/\ -/\ -/\ -/\ -)\ [100,100,100,101]100)$ **where**
TR3 R M N P = $\{x.\ \exists\, m{\in}carrier\ M.\ \exists\, n{\in}carrier\ N.\ \exists\ \ a{\in}carrier\ R.$
$x = (a\ \cdot_{s}M\ m,\ n)\ \pm_P\ (-_{a}P\ (a\ \cdot_{s}P\ (m,\ n)))\}$

definition
tensor-relations4 :: $[('r,\ 'm)$ *Ring-scheme*, $('a,\ 'r,\ 'm1)$ *Module-scheme*,
$('b,\ 'r,\ 'm1)$ *Module-scheme*, $(('a * 'b),\ 'r,\ 'm1)$ *Module-scheme*$]$ $\Rightarrow$
$('a * 'b)$ *set*
$((4TR4/\ -/\ -/\ -/\ -)\ [100,100,100,101]100)$ **where**
TR4 R M N MN = $\{x.\ \exists\, m{\in}carrier\ M.\ \exists\, n{\in}carrier\ N.\ \exists\ \ a{\in}carrier\ R.$
$x = (m,\ a\ \cdot_{s}N\ n)\ \pm_{MN}\ (-_{a}MN\ (a\ \cdot_{s}\ MN\ (m,\ n)))\}$

definition
tensor-relations :: $[('r,\ 'm)$ *Ring-scheme*, $('a,\ 'r,\ 'm1)$ *Module-scheme*,
$('b,\ 'r,\ 'm1)$ *Module-scheme*, $(('a * 'b),\ 'r,\ 'm1)$ *Module-scheme*$]$ $\Rightarrow$
$('a * 'b)$ *set*
$((4TR_/\ -/\ -/\ -)\ [100,100,101]100)$ **where**
TR_R *M N MN* = LSM_R *MN* $((TR1\ R\ M\ N\ MN) \cup (TR2\ R\ M\ N\ MN) \cup$
$(TR3\ R\ M\ N\ MN) \cup (TR4\ R\ M\ N\ MN))$

definition
tensor-product :: $[('r,\ 'm)$ *Ring-scheme*, $('a,\ 'r,\ 'm1)$ *Module-scheme*,
$('b,\ 'r,\ 'm1)$ *Module-scheme*, $(('a * 'b),\ 'r,\ 'm1)$ *Module-scheme*$]$ $\Rightarrow$
$(('a * 'b)$ *set*, $'r)$ *Module* **where**
tensor-product R M N MN = $MN\ /_m\ (TR_R\ M\ N\ MN)$

abbreviation
TENSORPROD $((4-/\ _\bigotimes_/\ -)\ [92,92,92,93]92)$ **where**
$M\ {}_P\bigotimes_R\ N$ == *tensor-product R M N P*

lemma (**in** *Ring*) *mem-cartesian*:$[\![R$ *module* M; R *module* N; $m \in$ *carrier* M;
$n \in$ *carrier* $N\ ]\!] \Longrightarrow (m,\ n) \in M \times_c N$
by (*simp add*:*prod-carr-def*)

lemma (**in** *Ring*) *cartesianTr*:$[\![R$ *module* M; R *module* N; $x \in M \times_c N]\!] \Longrightarrow$
$\exists\, m\ n.\ m{\in}carrier\ M \wedge n \in carrier\ N \wedge x = (m,\ n)$
by (*cases x*) (*simp add*: *prod-carr-def*)

lemma (**in** *Ring*) *free-module-mem*:$[\![R$ *module* M; R *module* N; $m \in$ *carrier* M;
$n \in$ *carrier* N; $FM_R\ P\ M\ N]\!] \Longrightarrow\ (m,\ n) \in$ *carrier* P

apply (*rule free-gen-mem* [*of P M N*], *assumption*+)
apply (*simp add*:*prod-carr-def*)
done

lemma (**in** *Ring*) *FM-P-module*:⟦*R module M*; *R module N*; FM_R *P M N*⟧
⟹ *R module P*
by (*simp add*:*fm-gen-by-prod-def*)

lemma (**in** *Ring*) *TR1-sub-carr*:⟦*R module M*; *R module N*; FM_R *P M N*⟧ ⟹
(*TR1 R M N P*) ⊆ *carrier P*
apply (*simp add*:*fm-gen-by-prod-def*)
apply (*erule conjE*)
apply (*rule subsetI*, *simp add*:*tensor-relations1-def*)
apply ((*erule bexE*)+,
frule Module.module-is-ag [*of P*], *simp*)
apply (*rule aGroup.ag-pOp-closed*, *assumption*,
rule free-module-mem, *assumption*+)
apply (*frule Module.module-is-ag*[*of M R*],
rule aGroup.ag-pOp-closed [*of M*], *assumption*+,
simp add:*fm-gen-by-prod-def*)
apply (*rule aGroup.ag-mOp-closed*[*of P*], *assumption*,
rule aGroup.ag-pOp-closed [*of P*], *assumption*+,
rule free-module-mem, *assumption*+,
simp add:*fm-gen-by-prod-def*)
apply (*rule free-module-mem*, *assumption*+, *simp add*:*fm-gen-by-prod-def*)
done

lemma (**in** *Ring*) *TR2-sub-carr*:⟦*R module M*; *R module N*; FM_R *P M N*⟧ ⟹
(*TR2 R M N P*) ⊆ *carrier P*
apply (*simp add*:*fm-gen-by-prod-def*, *erule conjE*)
apply (*rule subsetI*)
apply (*simp add*:*tensor-relations2-def*, (*erule bexE*)+, *simp*)
apply (*frule Module.module-is-ag*[*of P R*])
apply (*rule aGroup.ag-pOp-closed*, *assumption*)
apply (*rule free-module-mem*, *assumption*+)
apply (*frule Module.module-is-ag* [*of N R*])
apply (*rule aGroup.ag-pOp-closed* [*of N*], *assumption*+,
simp add:*fm-gen-by-prod-def*)
apply (*rule aGroup.ag-mOp-closed* [*of P*], *assumption*+,
rule aGroup.ag-pOp-closed [*of P*], *assumption*+,
rule free-module-mem, *assumption*+, *simp add*:*fm-gen-by-prod-def*)
apply (*rule free-module-mem*, *assumption*+,
simp add:*fm-gen-by-prod-def*)
done

lemma (**in** *Ring*) *TR3-sub-carr*:⟦*R module M*; *R module N*; FM_R *P M N*⟧ ⟹
(*TR3 R M N P*) ⊆ *carrier P*
apply (*simp add*:*fm-gen-by-prod-def*)
apply (*erule conjE*)

apply (*rule subsetI*)
apply (*simp add*:*tensor-relations3-def*)
apply ((*erule bexE*)+, *simp*,
thin-tac $x = (a \cdot_{s\,M} m,\ n) \pm_P -_{a\,P} (a \cdot_{s\,P} (m,\ n))$)

apply (*frule Module.module-is-ag* [*of P R*])
apply (*rule aGroup.ag-pOp-closed*, *assumption*)
apply (*rule free-module-mem*, *assumption*+)
apply (*simp add*:*Module.sc-mem*, *assumption*,
simp add:*fm-gen-by-prod-def*)
apply (*rule aGroup.ag-mOp-closed*[*of P*], *assumption*+)
apply (*rule Module.sc-mem*, *assumption*+)
apply (*rule free-module-mem*, *assumption*+,
simp add:*fm-gen-by-prod-def*)
done

lemma (**in** *Ring*) *TR4-sub-carr*:$[\![R$ *module* M; R *module* N; $FM_R\ P\ M\ N]\!] \Longrightarrow$
$(TR4\ R\ M\ N\ P) \subseteq$ *carrier* P
apply (*simp add*:*fm-gen-by-prod-def*)
apply (*erule conjE*)
apply (*rule subsetI*)
apply (*simp add*:*tensor-relations4-def*)
apply ((*erule bexE*)+, *simp*)

apply (*frule Module.module-is-ag* [*of P R*])
apply (*rule aGroup.ag-pOp-closed*, *assumption*)
apply (*rule free-module-mem*, *assumption*+,
simp add:*Module.sc-mem*, *simp add*:*fm-gen-by-prod-def*)
apply (*rule aGroup.ag-mOp-closed*[*of P*], *assumption*+)
apply (*rule Module.sc-mem*, *assumption*+)
apply (*rule free-module-mem*, *assumption*+,
simp add:*fm-gen-by-prod-def*)
done

lemma (**in** *Ring*) *TR-sub-carr*:$[\![R$ *module* M; R *module* N; $FM_R\ P\ M\ N]\!] \Longrightarrow$
$(TR1\ R\ M\ N\ P) \cup (TR2\ R\ M\ N\ P) \cup (TR3\ R\ M\ N\ P) \cup (TR4\ R\ M\ N\ P) \subseteq$
carrier P
apply (*rule subsetI*)
apply (*case-tac* $x \in TR1\ R\ M\ N\ P$, *simp*)
apply (*frule TR1-sub-carr* [*of M N P*], *assumption*+, *simp add*:*subsetD*)

apply (*case-tac* $x \in TR2\ R\ M\ N\ P$, *simp*,
frule TR2-sub-carr [*of M N P*], *assumption*+, *simp add*:*subsetD*)

apply (*case-tac* $x \in TR3\ R\ M\ N\ P$, *simp*,
frule TR3-sub-carr [*of M N P*], *assumption*+, *simp add*:*subsetD*)

apply *simp*
apply (*frule TR4-sub-carr* [*of M N P*], *assumption*+, *simp add*:*subsetD*)

done

lemma (**in** *Ring*) *TR-submodule*:⟦*R module M*; *R module N*; FM_R *P M N*⟧ ⟹
submodule R P (TR_R *M N P*)
apply (*simp add*:*tensor-relations-def*)
apply (*rule LSM-submodule*[*of P* -])
apply (*simp add*:*fm-gen-by-prod-def*)
apply (*rule TR-sub-carr*, *assumption*+)
done

lemma (**in** *Ring*) *TR-cont-TR1234*:⟦*R module M*; *R module N*; FM_R *P M N*⟧
⟹
TR1 R M N P ∪ *TR2 R M N P* ∪ *TR3 R M N P* ∪ *TR4 R M N P* ⊆ TR_R *M N P*
apply (*simp add*:*tensor-relations-def*)
apply (*cut-tac LSM-inc-T* [*of P TR1 R M N P* ∪ *TR2 R M N P* ∪ *TR3 R M N P* ∪
TR4 R M N P], *simp*)
apply (*simp add*:*fm-gen-by-prod-def*)
apply (*rule TR-sub-carr*, *assumption*+)
done

lemma (**in** *Ring*) *TR1-mem*:⟦*R module M*; *R module N*; FM_R *P M N*; *m1* ∈ *carrier M*;
m2 ∈ *carrier M*; *n* ∈ *carrier N*⟧ ⟹ (*m1* $\pm_M$ *m2*, *n*) $\pm_P$ $-_{aP}$ ((*m1*, *n*) $\pm_P$ (*m2*, *n*))
∈ TR_R *M N P*
apply (*rule subsetD*[*of TR1 R M N P* TR_R *M N P*
(*m1* $\pm_M$ *m2*, *n*) $\pm_P$ $-_{aP}$ ((*m1*, *n*) $\pm_P$ (*m2*, *n*))])
apply (*rule subset-trans*[*of TR1 R M N P TR1 R M N P* ∪ *TR2 R M N P* ∪
TR3 R M N P ∪ *TR4 R M N P* TR_R *M N P*],
rule subsetI, *simp*,
cut-tac TR-cont-TR1234[*of M N P*], *assumption*+)
apply (*simp add*:*tensor-relations1-def*)
apply *blast*
done

lemma (**in** *Ring*) *TR2-mem*:⟦*R module M*; *R module N*; FM_R *P M N*; *m* ∈ *carrier M*;
n1 ∈ *carrier N*; *n2* ∈ *carrier N* ⟧ ⟹
(*m*, *n1* $\pm_N$ *n2*) $\pm_P$ $-_{aP}$ ((*m*, *n1*) $\pm_P$ (*m*, *n2*)) ∈ TR_R *M N P*
apply (*rule subsetD*[*of TR2 R M N P* TR_R *M N P*
(*m*, *n1* $\pm_N$ *n2*) $\pm_P$ $-_{aP}$ ((*m*, *n1*) $\pm_P$ (*m*, *n2*))])
apply (*rule subset-trans*[*of TR2 R M N P TR1 R M N P* ∪ *TR2 R M N P* ∪
TR3 R M N P ∪ *TR4 R M N P* TR_R *M N P*],
rule subsetI, *simp*,
cut-tac TR-cont-TR1234[*of M N P*], *assumption*+)
apply (*simp add*:*tensor-relations2-def*)
apply *blast*

done

lemma (**in** *Ring*) *TR3-mem*:⟦*R module M*; *R module N*; FM_R *P M N*; *m* ∈ *carrier M*;
n ∈ *carrier N*; *a* ∈ *carrier R*⟧ ⟹
$(a \cdot_{sM} m,\ n) \pm_P -_{aP} (a \cdot_{sP} (m,\ n)) \in TR_R\ M\ N\ P$
apply (*rule subsetD*[*of TR3 R M N P* TR_R *M N P*
$(a \cdot_{sM} m,\ n) \pm_P -_{aP} (a \cdot_{sP} (m,\ n))$])
apply (*rule subset-trans*[*of TR3 R M N P TR1 R M N P* ∪ *TR2 R M N P* ∪
TR3 R M N P ∪ *TR4 R M N P* TR_R *M N P*],
rule subsetI, *simp*,
cut-tac TR-cont-TR1234[*of M N P*], *assumption*+)
apply (*simp add*:*tensor-relations3-def*)
apply *blast*
done

lemma (**in** *Ring*) *TR4-mem*:⟦*R module M*; *R module N*; FM_R *P M N*; *m* ∈ *carrier M*;
n ∈ *carrier N*; *a* ∈ *carrier R*⟧ ⟹
$(m,\ a \cdot_{sN} n) \pm_P -_{aP} (a \cdot_{sP} (m,\ n)) \in TR_R\ M\ N\ P$
apply (*rule subsetD*[*of TR4 R M N P* TR_R *M N P*
$(m,\ a \cdot_{sN} n) \pm_P -_{aP} (a \cdot_{sP} (m,\ n))$])
apply (*rule subset-trans*[*of TR4 R M N P TR1 R M N P* ∪ *TR2 R M N P* ∪
TR3 R M N P ∪ *TR4 R M N P* TR_R *M N P*],
rule subsetI, *simp*,
cut-tac TR-cont-TR1234[*of M N P*], *assumption*+)
apply (*simp add*:*tensor-relations4-def*)
apply *blast*
done

lemma (**in** *Ring*) *tensor-product-module*:⟦*R module M*; *R module N*; FM_R *P M N*⟧ ⟹
R module (*tensor-product R M N P*)
apply (*simp add*:*fm-gen-by-prod-def*, *erule conjE*)
apply (*frule TR-submodule* [*of M N P*], *assumption*+)
apply (*simp add*:*fm-gen-by-prod-def*)
apply (*simp add*:*tensor-product-def*)
apply (*simp add*:*Module.qmodule-module* [*of P R* TR_R *M N P*])
done

lemma (**in** *Ring*) *tau-mpj-bilin1*:⟦ *R module M*; *R module N*; FM_R *P M N*;
x1 ∈ *carrier M*; *x2* ∈ *carrier M*; *y* ∈ *carrier N*⟧ ⟹
(*mpj P* (TR_R *M N P*)) (*x1* $\pm_M$ *x2*, *y*) =
(*mpj P* (TR_R *M N P*)) (*x1*, *y*) $\pm_{(M\ _P\otimes_R\ N)}$ (*mpj P* (TR_R *M N P*) (*x2*, *y*))
apply (*frule FM-P-module*[*of M N P*], *assumption*+)
apply (*subgoal-tac* (*x1* $\pm_M$ *x2*, *y*) $\pm_P$ ($-_{aP}$ ((*x1*, *y*) $\pm_P$ (*x2*, *y*))) ∈
$ker_{P,(P\ /_m\ (TR_R\ M\ N\ P))}$ (*mpj P* (TR_R *M N P*)))
apply (*frule TR-submodule* [*of M N P*], *assumption*+,

frule Module.qmodule-module [*of P R* TR_R *M N P*], *assumption*+,
frule Module.mpj-mHom [*of P R* TR_R *M N P*], *assumption*+,
frule Module.mHom-eq-ker[*of P R P* $/_m$ (TR_R *M N P*) *mpj P* (TR_R *M N P*)
(*x1* $\pm_M$ *x2*, *y*) (*x1*, *y*) $\pm_P$ (*x2*, *y*)], *assumption*+,
rule free-module-mem, *assumption*+)
apply (*frule Module.module-is-ag* [*of M R*],
simp add:*aGroup.ag-pOp-closed*, *assumption*+,
frule Module.module-is-ag [*of P R*],
rule aGroup.ag-pOp-closed, *assumption*+,
rule free-module-mem, *assumption*+,
rule free-module-mem, *assumption*+)
apply (*frule-tac m* = *x1* **and** *n* = *y* **in** *free-module-mem*[*of M N - - P*],
assumption+,
frule-tac m = *x2* **and** *n* = *y* **in** *free-module-mem*[*of M N - - P*],
assumption+,
simp add:*Module.mHom-add*[*of P R P* $/_m$ (TR_R *M N P*) *mpj P* (TR_R *M N P*)
(*x1*, *y*) (*x2*, *y*)],
simp add:*tensor-product-def*)
apply (*frule TR-submodule* [*of M N P*], *assumption*+,
simp add:*Module.mker-of-mpj*, *thin-tac submodule R P* (TR_R *M N P*),
simp add:*TR1-mem*)
done

lemma (**in** *Ring*) *tau-mpj-bilin2*:⟦*R module M*; *R module N*; FM_R *P M N*;
m ∈ *carrier M*; *n1* ∈ *carrier N*; *n2* ∈ *carrier N*⟧ ⟹
(*mpj P* (TR_R *M N P*)) (*m*, *n1* $\pm_N$ *n2*) =
(*mpj P* (TR_R *M N P*)) (*m*, *n1*) $\pm_{(M\ P\otimes_R\ N)}$ (*mpj P* (TR_R *M N P*) (*m*, *n2*))
apply (*frule FM-P-module*[*of M N P*], *assumption*+)
apply (*subgoal-tac* (*m*, *n1* $\pm_N$ *n2*) $\pm_P$ ($-_a P$ ((*m*, *n1*) $\pm_P$ (*m*, *n2*))) ∈
$ker_{P,(P\ /_m\ (TR_R\ M\ N\ P))}$ (*mpj P* (TR_R *M N P*)))
apply (*frule TR-submodule* [*of M N P*], *assumption*+,
frule Module.qmodule-module [*of P R* TR_R *M N P*], *assumption*+,
frule Module.mpj-mHom [*of P R* TR_R *M N P*], *assumption*+,
frule Module.mHom-eq-ker[*of P R P* $/_m$ (TR_R *M N P*) *mpj P* (TR_R *M N P*)
(*m*, *n1* $\pm_N$ *n2*) (*m*, *n1*) $\pm_P$ (*m*, *n2*)], *assumption*+,
rule free-module-mem, *assumption*+)
apply (*frule Module.module-is-ag* [*of N R*],
simp add:*aGroup.ag-pOp-closed*, *assumption*+,
frule Module.module-is-ag [*of P R*],
rule aGroup.ag-pOp-closed, *assumption*+,
rule free-module-mem, *assumption*+,
rule free-module-mem, *assumption*+)
apply (*frule-tac m* = *m* **and** *n* = *n1* **in** *free-module-mem*[*of M N - - P*],
assumption+,
frule-tac m = *m* **and** *n* = *n2* **in** *free-module-mem*[*of M N - - P*],
assumption+, *simp add*:*tensor-product-def*)

apply (*rule Module.mHom-add*[*of P R P* /$_m$ (*TR*$_R$ *M N P*) *mpj P* (*TR*$_R$ *M N P*)
(*m*, *n1*) (*m*, *n2*)], *assumption*+)
apply (*frule TR-submodule* [*of M N P*], *assumption*+,
simp add:*Module.mker-of-mpj*, *thin-tac submodule R P* (*TR*$_R$ *M N P*),
simp add:*TR2-mem*)
done

lemma (**in** *Ring*) *tau-mpj-bilin3*:⟦*R module M*; *R module N*; *FM*$_R$ *P M N*;
m ∈ *carrier M*; *n* ∈ *carrier N*; *a* ∈ *carrier R*⟧ ⟹
(*mpj P* (*TR*$_R$ *M N P*)) (*a* ·$_{sM}$ *m*, *n*) = *a* ·$_{s(M\ P\bigotimes R\ N)}$
(*mpj P* (*TR*$_R$ *M N P*) (*m*, *n*))
apply (*frule FM-P-module*[*of M N P*], *assumption*+)
apply (*subgoal-tac* (*a* ·$_{sM}$ *m*, *n*) ±$_P$ −$_{aP}$ (*a* ·$_{sP}$ (*m*, *n*)) ∈
ker$_{P,(P\ /_m\ (TR_R\ M\ N\ P))}$ (*mpj P* (*TR*$_R$ *M N P*)))
apply (*frule TR-submodule* [*of M N P*], *assumption*+,
frule Module.qmodule-module [*of P R TR*$_R$ *M N P*], *assumption*+,
frule Module.mpj-mHom [*of P R TR*$_R$ *M N P*], *assumption*+,
frule Module.mHom-eq-ker[*of P R P* /$_m$ (*TR*$_R$ *M N P*) *mpj P* (*TR*$_R$ *M N P*)
(*a* ·$_{sM}$ *m*, *n*) *a* ·$_{sP}$ (*m*, *n*)], *assumption*+,
rule free-module-mem, *assumption*+)
apply (*simp add*:*Module.sc-mem*, *assumption*+)
apply (*frule-tac m* = *m* **and** *n* = *n* **in** *free-module-mem*[*of M N - - P*],
assumption+,
simp add:*Module.sc-mem*,
assumption+, *simp add*:*tensor-product-def*)
apply (*rule Module.mHom-lin*[*of P R P* /$_m$ (*TR*$_R$ *M N P*) (*m*, *n*)
mpj P (*TR*$_R$ *M N P*) *a*], *assumption*+, *simp add*:*free-module-mem*,
assumption+)
apply (*frule TR-submodule* [*of M N P*], *assumption*+,
simp add:*Module.mker-of-mpj*, *thin-tac submodule R P* (*TR*$_R$ *M N P*),
simp add:*TR3-mem*)
done

lemma (**in** *Ring*) *tau-mpj-bilin4*:⟦*R module M*; *R module N*; *FM*$_R$ *P M N*;
m ∈ *carrier M*; *n* ∈ *carrier N*; *a* ∈ *carrier R*⟧ ⟹
(*mpj P* (*TR*$_R$ *M N P*)) (*m*, *a* ·$_{sN}$ *n*) = *a* ·$_{s(M\ P\bigotimes R\ N)}$
(*mpj P* (*TR*$_R$ *M N P*) (*m*, *n*))
apply (*frule FM-P-module*[*of M N P*], *assumption*+)
apply (*subgoal-tac* (*m*, *a* ·$_{sN}$ *n*) ±$_P$ −$_{aP}$ (*a* ·$_{sP}$ (*m*, *n*)) ∈
ker$_{P,(P\ /_m\ (TR_R\ M\ N\ P))}$ (*mpj P* (*TR*$_R$ *M N P*)))
apply (*frule TR-submodule* [*of M N P*], *assumption*+,
frule Module.qmodule-module [*of P R TR*$_R$ *M N P*], *assumption*+,
frule Module.mpj-mHom [*of P R TR*$_R$ *M N P*], *assumption*+,
frule Module.mHom-eq-ker[*of P R P* /$_m$ (*TR*$_R$ *M N P*) *mpj P* (*TR*$_R$ *M N P*)
(*m*, *a* ·$_{sN}$ *n*) *a* ·$_{sP}$ (*m*, *n*)], *assumption*+,

rule free-module-mem, *assumption*+)
apply (*simp add*:*Module.sc-mem*, *assumption*+)
apply (*frule-tac* $m = m$ **and** $n = n$ **in** *free-module-mem*[*of M N - - P*],
assumption+,
simp add:*Module.sc-mem*,
assumption+, *simp add*:*tensor-product-def*)
apply (*rule Module.mHom-lin*[*of P R P* $/_m$ (TR_R *M N P*) (*m*, *n*)
mpj P (TR_R *M N P*) *a*], *assumption*+, *simp add*:*free-module-mem*,
assumption+)
apply (*frule TR-submodule* [*of M N P*], *assumption*+,
simp add:*Module.mker-of-mpj*, *thin-tac submodule R P* (TR_R *M N P*),
simp add:*TR4-mem*)
done

definition
tau :: [('*r*, '*m*) *Ring-scheme*, ('*a*, '*r*, '*m1*) *Module-scheme*,
('*b*, '*r*, '*m1*) *Module-scheme*, (('*a* ∗ '*b*), '*r*, '*m1*) *Module-scheme*] ⇒
('*a* ∗ '*b*) ⇒ ('*a* ∗ '*b*) **where**
tau R M N P = (λ*x*∈(*M* ×$_c$ *N*). *x*)

lemma (**in** *Ring*) *tau-func*:⟦*R module M*; *R module N*; FM_R *P M N*⟧ ⟹
tau R M N P ∈ *M* ×$_c$ *N* → *carrier P*
apply (*rule Pi-I*)
apply (*frule-tac* $x = x$ **in** *cartesianTr* [*of M N*], *assumption*+)
apply ((*erule exE*)+, (*erule conjE*)+, *simp add*:*tau-def*)
apply (*simp add*:*free-module-mem*)
done

lemma (**in** *Ring*) *tau-mem*:⟦*R module M*; *R module N*; *m* ∈ *carrier M*;
n ∈ *carrier N*; FM_R *P M N*⟧ ⟹ *tau R M N P* (*m*, *n*) ∈ *carrier P*
apply (*frule tau-func* [*of M N P*], *assumption*+)
apply (*rule funcset-mem*, *assumption*+)
apply (*simp add*:*prod-carr-mem*)
done

lemma (**in** *Ring*) *tau-inj0*:⟦¬ *zeroring R*; *R module M*; *R module N*; FM_R *P M N*⟧
⟹ *inj-on* (*tau R M N P*) (*M* ×$_c$ *N*)
apply (*simp add*:*inj-on-def*, (*rule ballI*)+)
apply (*rule impI*)
apply (*simp add*:*tau-def*)
done

lemma (**in** *Ring*) *tau-inj1*:⟦*zeroring R*; *R module M*; *R module N*; FM_R *P M N*⟧
⟹
inj-on (*tau R M N P*) (*M* ×$_c$ *N*)
apply (*simp add*:*inj-on-def*)
apply ((*rule ballI*)+, *rule impI*)
apply (*frule module-over-zeroring*[*of M*], *assumption*+)

apply (*frule module-over-zeroring*[*of N*], *assumption*+)
apply (*simp add*:*zeroring-def*, *erule conjE*)
apply (*frule-tac x* = *x* **in** *cartesianTr*[*of M N*], *assumption*+)
apply ((*erule exE*)+, (*erule conjE*)+, *simp add*:*tau-def*)
done

lemma (**in** *Ring*) *tau-inj*:⟦*R module M*; *R module N*; FM_R *P M N*⟧ ⟹
inj-on (*tau R M N P*) (*M* $\times_c$ *N*)
apply (*case-tac zeroring R*)
apply (*simp add*:*tau-inj1*)
apply (*simp add*:*tau-inj0*)
done

lemma (**in** *Ring*) *tau-mpj-bilinear*:⟦*R module M*; *R module N*; FM_R *P M N*⟧ ⟹

bilinear-map (*compose* (*M* $\times_c$ *N*) (*mpj P* (TR_R *M N P*)) (*tau R M N P*))
R M N (*M* $_P\bigotimes_R$ *N*)
apply (*simp add*:*bilinear-map-def*)
apply (*rule conjI*)
apply (*rule Pi-I*)
apply (*simp add*:*compose-def tau-def tensor-product-def*)
apply (*frule TR-submodule* [*of M N P*], *assumption*+)
apply (*frule FM-P-module*[*of M N P*], *assumption*+,
frule Module.mpj-mHom[*of P R* TR_R *M N P*], *assumption*+)
apply (*frule Module.qmodule-module* [*of P R* TR_R *M N P*], *assumption*+,
rule Module.mHom-mem [*of P R P* $/_m$ (TR_R *M N P*) (*mpj P* (TR_R *M N P*))],
assumption+, *simp add*:*free-gen-mem*)
apply (*rule conjI*)
apply (*rule ballI*)+
apply (*simp add*:*compose-def tau-def mem-cartesian*)
apply (*frule Module.module-is-ag*[*of M R*],
frule-tac x = *x1* **and** *y* = *x2* **in** *aGroup.ag-pOp-closed*, *assumption*+,
simp add:*mem-cartesian*, *simp add*:*tau-mpj-bilin1*)
apply (*rule conjI*)
apply (*rule ballI*)+
apply (*simp add*:*compose-def tau-def mem-cartesian*)
apply (*frule Module.module-is-ag*[*of N R*],
frule-tac x = *y1* **and** *y* = *y2* **in** *aGroup.ag-pOp-closed*, *assumption*+,
simp add:*mem-cartesian*, *simp add*:*tau-mpj-bilin2*)
apply (*rule ballI*)+
apply (*rule conjI*)
apply (*simp add*:*compose-def tau-def mem-cartesian*)
apply (*frule-tac a* = *r* **and** *m* = *x* **in** *Module.sc-mem*[*of M R*], *assumption*+,
simp add:*mem-cartesian*, *simp add*:*tau-mpj-bilin3*)
apply (*simp add*:*compose-def tau-def mem-cartesian*)
apply (*frule-tac a* = *r* **and** *m* = *y* **in** *Module.sc-mem*[*of N R*], *assumption*+,
simp add:*mem-cartesian*, *simp add*:*tau-mpj-bilin4*)
done

definition
tnm :: [('*r*, '*m*) *Ring-scheme*, (('*a* ∗ '*b*), '*r*, '*m1*) *Module-scheme*,
('*a*, '*r*, '*m1*) *Module-scheme*, ('*b*, '*r*, '*m1*) *Module-scheme*] ⇒
('*a* ∗ '*b*) ⇒ ('*a* ∗ '*b*) *set* **where**
tnm R P M N = *compose* (*M* ×$_c$ *N*) (*mpj P* (*TR*$_R$ *M N P*)) (*tau R M N P*)

lemma (**in** *Ring*) *tnm-bilinear*:⟦*R module M*; *R module N*; *FM*$_R$ *P M N*⟧ ⟹
bilinear-map (*tnm R P M N*) *R M N* (*M* $_P\bigotimes_R$ *N*)
apply (*simp add*:*tnm-def*)
apply (*simp add*:*tau-mpj-bilinear*)
done

lemma (**in** *Ring*) *tnm-mem*:⟦ *R module M*; *R module N*; *FM*$_R$ *P M N*; *m* ∈ *carrier M*;
n ∈ *carrier N*⟧ ⟹ *tnm R P M N* (*m*, *n*) ∈ *carrier* (*M* $_P\bigotimes_R$ *N*)
apply (*simp add*:*tnm-def*)
apply (*frule tau-mem* [*of M N m n*], *assumption*+)
apply (*simp add*:*compose-def*, *simp add*:*prod-carr-def*)
apply (*simp add*:*tensor-product-def*)
apply (*frule TR-submodule* [*of M N P*], *assumption*+)
apply (*rule Module.mpj-mem*[*of P R TR*$_R$ *M N P tau R M N P* (*m*, *n*)],
rule FM-P-module[*of M N P*], *assumption*+)
done

definition
tensor-elem :: [('*r*, '*m*) *Ring-scheme*, (('*a* ∗ '*b*), '*r*, '*m1*) *Module-scheme*,
('*a*, '*r*, '*m1*) *Module-scheme*, ('*b*, '*r*, '*m1*) *Module-scheme*] ⇒ '*a* ⇒ '*b*
⇒ ('*a* ∗ '*b*) *set* **where**
tensor-elem R P M N m n = *tnm R P M N* (*m*, *n*)

abbreviation
TNSELEM ((*6*- -,-⊗-,-/ -) [*100*,*100*,*100*,*100*,*100*,*101*]*101*) **where**
m $_{R,P}\otimes_{M,N}$ *n* == *tensor-elem R P M N m n*

lemma (**in** *Ring*) *tensor-univ-propTr*:⟦*R module M*; *R module N*; *FM*$_R$ *P M N*;
R module Z; *bilinear-map f R M N Z*⟧ ⟹
∃ *g*. *g* ∈ *mHom R P Z* ∧ (*compose* (*M* ×$_c$ *N*) *g* (*tau R M N P*)) = *f*
apply (*unfold fm-gen-by-prod-def*)
apply (*frule conjunct1*, *frule conjunct2*)
apply (*fold fm-gen-by-prod-def*)
apply (*frule bilinear-func*[*of f M N Z*])
apply (*frule Module.exist-extension-mhom* [*of P R Z M* ×$_c$ *N f*], *assumption*+)
apply (*erule bexE*, *rename-tac h*)
apply (*subgoal-tac compose* (*M* ×$_c$ *N*) *h* (*tau R M N P*) = *f*)
apply *blast*
apply (*rule funcset-eq* [*of* - *M* ×$_c$ *N*])
apply (*simp add*:*compose-def*, *simp add*:*bilinear-map-def*)

apply (*rule ballI*)
apply (*simp add*:*compose-def tau-def*)
done

lemma (**in** *Ring*) *tensor-univ-propTr1*:⟦*R module M*; *R module N*; FM_R *P M N*;

R module Z; *bilinear-map f R M N Z*⟧ ⟹
$\exists!g.\ g \in (mHom\ R\ (M\ {}_P\bigotimes_R\ N)\ Z) \wedge (compose\ (M \times_c N)\ g\ (tnm\ R\ P\ M\ N)) = f$
apply (*frule Module.module-is-ag*[*of M*],
frule Module.module-is-ag[*of N*],
frule FM-P-module[*of M N P*], *assumption*+,
frule Module.module-is-ag[*of P*])
apply (*simp add*:*tnm-def*)

apply (*frule tensor-univ-propTr*[*of M N P Z f*], *assumption*+)
apply (*erule exE*, *erule conjE*,
frule TR-submodule[*of M N P*], *assumption*+,
frule-tac Module.indmhom1[*of P R* TR_R *M N P Z*], *assumption*+,
frule Module.submodule-subset[*of P R* TR_R *M N P*], *assumption*+,
simp add:*tensor-relations-def*)
apply (*rule-tac f* = *g* **in** *LSM-sub-ker*[*of P Z* (*TR1 R M N P* ∪ *TR2 R M N P* ∪ *TR3 R M N P* ∪ *TR4 R M N P*)], *assumption*+,
rule TR-sub-carr, *assumption*+,
thin-tac submodule R P
(*LSM*$_R$ *P* (*TR1 R M N P* ∪ *TR2 R M N P* ∪ *TR3 R M N P* ∪ *TR4 R M N P*)),
thin-tac LSM$_R$ *P* (*TR1 R M N P* ∪ *TR2 R M N P* ∪ *TR3 R M N P* ∪ *TR4 R M N P*) ⊆ *carrier P*)

apply (*rule subsetI*, *simp*, *erule disjE*,
frule TR1-sub-carr[*of M N P*], *assumption*+,
frule-tac c = *x* **in** *subsetD*[*of TR1 R M N P carrier P*], *assumption*+,
thin-tac TR1 R M N P ⊆ *carrier P*,
simp add:*tensor-relations1-def*, (*erule bexE*)+, *simp add*:*ker-def*)
apply (*frule-tac x* = *m1* **and** *y* = *m2* **in** *aGroup.ag-pOp-closed*, *assumption*+,
frule-tac m = *m1* ±$_M$ *m2* **and** *n* = *n* **in** *free-module-mem*[*of M N*],
assumption+,
frule-tac m = *m1* **and** *n* = *n* **in** *free-module-mem*[*of M N*], *assumption*+,
frule-tac m = *m2* **and** *n* = *n* **in** *free-module-mem*[*of M N*], *assumption*+)
apply (*subst Module.mHom-add*[*of P R Z*], *assumption*+,
rule aGroup.ag-mOp-closed, *assumption*+,
rule aGroup.ag-pOp-closed, *assumption*+,
frule-tac x = (*m1*, *n*) **and** *y* = (*m2*, *n*) **in** *aGroup.ag-pOp-closed*,
assumption+,
simp add:*Module.mHom-inv*[*of P R Z*] *Module.mHom-add*)
apply (*frule-tac f* = *compose* (*M* ×$_c$ *N*) *g* (*tau R M N P*) **and** *g* = *f* **and**
x = (*m1* ±$_M$ *m2*, *n*) **in** *eq-fun-eq-val*,

frule-tac f = compose (M ×c N) g (tau R M N P) **and** *g = f* **and**
x = (m1, n) **in** *eq-fun-eq-val*,
frule-tac f = compose (M ×c N) g (tau R M N P) **and** *g = f* **and**
x = (m2, n) **in** *eq-fun-eq-val*,
thin-tac compose (M ×c N) g (tau R M N P) = f,
simp add:*compose-def cmp-def*, *simp add*:*mem-cartesian tau-def*,
simp add:*bilinear-l-add1*)

apply (*erule disjE*,
frule TR2-sub-carr[*of M N P*], *assumption*+,
frule-tac c = x **in** *subsetD*[*of TR2 R M N P carrier P*], *assumption*+,
thin-tac TR2 R M N P ⊆ carrier P,
simp add:*tensor-relations2-def*, (*erule bexE*)+, *simp add*:*ker-def*)
apply (*frule-tac x = n1* **and** *y = n2* **in** *aGroup.ag-pOp-closed*, *assumption*+,
frule-tac m = m **and** *n = n1 ±N n2* **in** *free-module-mem*[*of M N*],
assumption+,
frule-tac m = m **and** *n = n1* **in** *free-module-mem*[*of M N*], *assumption*+,
frule-tac m = m **and** *n = n2* **in** *free-module-mem*[*of M N*], *assumption*+)
apply (*subst Module.mHom-add*[*of P R Z*], *assumption*+,
rule aGroup.ag-mOp-closed, *assumption*+,
rule aGroup.ag-pOp-closed, *assumption*+,
frule-tac x = (m, n1) **and** *y = (m, n2)* **in** *aGroup.ag-pOp-closed*,
assumption+,
simp add:*Module.mHom-inv*[*of P R Z*] *Module.mHom-add*)
apply (*frule-tac f = compose (M ×c N) g (tau R M N P)* **and** *g = f* **and**
x = (m, n1 ±N n2) **in** *eq-fun-eq-val*,
frule-tac f = compose (M ×c N) g (tau R M N P) **and** *g = f* **and**
x = (m, n1) **in** *eq-fun-eq-val*,
frule-tac f = compose (M ×c N) g (tau R M N P) **and** *g = f* **and**
x = (m, n2) **in** *eq-fun-eq-val*,
thin-tac compose (M ×c N) g (tau R M N P) = f,
simp add:*compose-def cmp-def*, *simp add*:*mem-cartesian tau-def*,
simp add:*bilinear-r-add1*)

apply (*erule disjE*,
frule TR3-sub-carr[*of M N P*], *assumption*+,
frule-tac c = x **in** *subsetD*[*of TR3 R M N P carrier P*], *assumption*+,
thin-tac TR3 R M N P ⊆ carrier P,
simp add:*tensor-relations3-def*, (*erule bexE*)+, *simp add*:*ker-def*)
apply (*frule-tac a = a* **and** *m = m* **in** *Module.sc-mem*, *assumption*+,
frule-tac m = a ·sM m **and** *n = n* **in** *free-module-mem*[*of M N*],
assumption+,
frule-tac m = m **and** *n = n* **in** *free-module-mem*[*of M N*], *assumption*+,
frule-tac a = a **and** *m = (m, n)* **in** *Module.sc-mem*, *assumption*+,
frule-tac x = (a ·sP (m, n)) **in** *aGroup.ag-mOp-closed*, *assumption*+)
apply (*subst Module.mHom-add*[*of P R Z*], *assumption*+,
simp add:*Module.mHom-inv*[*of P R Z*],

simp add:*Module.mHom-lin*)
apply (*frule-tac f* = *compose* (*M* $\times_c$ *N*) *g* (*tau R M N P*) **and** *g* = *f* **and**
x = (*a* $\cdot_{s M}$ *m*, *n*) **in** *eq-fun-eq-val*,
frule-tac f = *compose* (*M* $\times_c$ *N*) *g* (*tau R M N P*) **and** *g* = *f* **and**
x = (*m*, *n*) **in** *eq-fun-eq-val*)
apply (*thin-tac compose* (*M* $\times_c$ *N*) *g* (*tau R M N P*) = *f*,
simp add:*compose-def cmp-def*, *simp add*:*mem-cartesian tau-def*,
simp add:*bilinear-l-lin1*)

apply (*frule TR4-sub-carr*[*of M N P*], *assumption*+,
frule-tac c = *x* **in** *subsetD*[*of TR4 R M N P carrier P*], *assumption*+,
thin-tac TR4 R M N P ⊆ *carrier P*,
simp add:*tensor-relations4-def*, (*erule bexE*)+, *simp add*:*ker-def*)
apply (*frule-tac a* = *a* **and** *m* = *n* **in** *Module.sc-mem*, *assumption*+,
frule-tac m = *m* **and** *n* = *a* $\cdot_{s N}$ *n* **in** *free-module-mem*[*of M N*],
assumption+,
frule-tac m = *m* **and** *n* = *n* **in** *free-module-mem*[*of M N*], *assumption*+,
frule-tac a = *a* **and** *m* = (*m*, *n*) **in** *Module.sc-mem*, *assumption*+,
frule-tac x = (*a* $\cdot_{s P}$ (*m*, *n*)) **in** *aGroup.ag-mOp-closed*, *assumption*+)
apply (*subst Module.mHom-add*[*of P R Z*], *assumption*+,
simp add:*Module.mHom-inv*[*of P R Z*],
simp add:*Module.mHom-lin*)
apply (*frule-tac f* = *compose* (*M* $\times_c$ *N*) *g* (*tau R M N P*) **and** *g* = *f* **and**
x = (*m*, *a* $\cdot_{s N}$ *n*) **in** *eq-fun-eq-val*,
frule-tac f = *compose* (*M* $\times_c$ *N*) *g* (*tau R M N P*) **and** *g* = *f* **and**
x = (*m*, *n*) **in** *eq-fun-eq-val*)
apply (*thin-tac compose* (*M* $\times_c$ *N*) *g* (*tau R M N P*) = *f*,
simp add:*compose-def cmp-def*, *simp add*:*mem-cartesian tau-def*)
apply (*simp add*:*bilinear-r-lin1*)
apply (*frule Module.qmodule-module* [*of P R* TR_R *M N P*], *assumption*+)
apply (*rule ex-ex1I*)
apply (*erule ex1E*, *erule conjE*,
thin-tac ∀ *y*. *y* ∈ *mHom R* (*P* $/_m$ (TR_R *M N P*)) *Z* ∧
compos P y (*mpj P* (TR_R *M N P*)) = *g* ⟶ *y* = *ga*)
apply (*simp add*:*tensor-product-def*)
apply (*cut-tac h* = *ga* **in** *compose-assoc*[*of tau R M N P M* $\times_c$ *N*
carrier P mpj P (TR_R *M N P*) *carrier* (*P* $/_m$ (TR_R *M N P*)) -
carrier Z], *simp add*:*tau-func*,
simp add:*Module.mpj-mem*,
rule Pi-I,
rule-tac f = *ga* **and** *m* = *x* **in**
Module.mHom-mem[*of P* $/_m$ (TR_R *M N P*) *R Z*], *assumption*+,
simp add:*compos-def*, *blast*)

apply (*simp add*:*tensor-product-def*,
thin-tac ∃!*ga*. *ga* ∈ *mHom R* (*P* $/_m$ (TR_R *M N P*)) *Z* ∧
compos P ga (*mpj P* (TR_R *M N P*)) = *g*)
apply (*frule free-gen-gen*[*of P M N*],

frule Module.surjec-generator[*of P R P* $/_m$ (TR_R *M N P*)
mpj P (TR_R *M N P*) *M* $\times_c$ *N*],
simp add:*Module.qmodule-module*,
simp add:*Module.mpj-mHom*,
simp add:*Module.mpj-surjec*, *assumption*)
apply (*rule-tac f* = *ga* **and** *g* = *y* **in** *Module.gen-mHom-eq*[*of P* $/_m$ (TR_R *M N P*)
R Z mpj P (TR_R *M N P*) ‘ *M* $\times_c$ *N*], *assumption*+, *simp*, *simp*)
apply (*erule conjE*)+
apply (*rule ballI*,
thin-tac generator R (*P* $/_m$ (TR_R *M N P*)) (*mpj P* (TR_R *M N P*) ‘ *M* $\times_c$ *N*),
simp add:*image-def*, *erule bexE*)
apply (*frule-tac f* = *compose* (*M* $\times_c$ *N*) *ga*
(*compose* (*M* $\times_c$ *N*) (*mpj P* (TR_R *M N P*)) (*tau R M N P*)) **and** *g* = *f* **and**
x = *x* **in** *eq-fun-eq-val*,
thin-tac compose (*M* $\times_c$ *N*) *ga* (*compose* (*M* $\times_c$ *N*) (*mpj P* (TR_R *M N P*))
(*tau R M N P*)) = *f*,
frule-tac f = *compose* (*M* $\times_c$ *N*) *y*
(*compose* (*M* $\times_c$ *N*) (*mpj P* (TR_R *M N P*)) (*tau R M N P*)) **and** *g* = *f* **and**
x = *x* **in** *eq-fun-eq-val*,
thin-tac compose (*M* $\times_c$ *N*) *y* (*compose* (*M* $\times_c$ *N*) (*mpj P* (TR_R *M N P*))
(*tau R M N P*)) = *f*)
apply (*simp add*:*compose-def tau-def*)
done

lemma (**in** *Ring*) *tensor-universal-property*:⟦*R module M*; *R module N*; FM_R *P M N* ⟧
⟹ *universal-property R M N* (*M* $_P\bigotimes_R$ *N*) (*tnm R P M N*)
apply (*simp add*:*universal-property-def*)
apply (*frule tau-mpj-bilinear* [*of M N*], *assumption*+)
apply (*rule conjI*, *simp add*:*tnm-def*)
apply ((*rule allI*)+, *rule impI*, *erule conjE*)
apply (*rule-tac Z* = *Z* **and** *f* = *g* **in** *tensor-univ-propTr1* [*of M N*],
assumption+)
done

end