Representations of Finite Groups

Jeremy Sylvestre
University of Alberta, Augustana Campus
jeremy.sylvestre@ualberta.ca

February 23, 2021

Abstract

We provide a formal framework for the theory of representations of finite groups, as modules over the group ring. Along the way, we develop the general theory of groups (relying on the *group_add* class for the basics), modules, and vector spaces, to the extent required for theory of group representations. We then provide formal proofs of several important introductory theorems in the subject, including Maschke's theorem, Schur's lemma, and Frobenius reciprocity. We also prove that every irreducible representation is isomorphic to a submodule of the group ring, leading to the fact that for a finite group there are only finitely many isomorphism classes of irreducible representations. In all of this, no restriction is made on the characteristic of the ring or field of scalars until the definition of a group representation, and then the only restriction made is that the characteristic must not divide the order of the group.

Contents

Note: A number of the proofs in this theory were modelled on or inspired by proofs in the books listed in the bibliography.

theory *Rep-Fin-Groups*

imports
HOL−Library.Function-Algebras
HOL−Library.Set-Algebras
HOL−Computational-Algebra.Polynomial

begin

1 Preliminaries

In this section, we establish some basic facts about logic, sets, and functions that are not available in the HOL library. As well, we develop some theory for almost-everywhere-zero functions in preparation of the definition of the group ring.

1.1 Logic

lemma *conjcases* [*case-names BothTrue OneTrue OtherTrue BothFalse*] :
  assumes *BothTrue*: $P \wedge Q \Longrightarrow R$
  and *OneTrue*: $P \wedge \neg Q \Longrightarrow R$
  and *OtherTrue*: $\neg P \wedge Q \Longrightarrow R$
  and *BothFalse*: $\neg P \wedge \neg Q \Longrightarrow R$
  shows R
  ⟨*proof*⟩

1.2 Sets

lemma *empty-set-diff-single* : $A - \{x\} = \{\} \Longrightarrow A = \{\} \vee A = \{x\}$
  ⟨*proof*⟩

lemma *seteqI* : $(\bigwedge a.\ a \in A \Longrightarrow a \in B) \Longrightarrow (\bigwedge b.\ b \in B \Longrightarrow b \in A) \Longrightarrow A = B$
  ⟨*proof*⟩

lemma *prod-ballI* : $(\bigwedge a\ b.\ (a,b) \in AxB \Longrightarrow P\ a\ b) \Longrightarrow \forall (a,b){\in}AxB.\ P\ a\ b$
  ⟨*proof*⟩

lemma *good-card-imp-finite* : $\mathit{of\text{-}nat}\ (\mathit{card}\ A) \neq (0{::}'a{::}\mathit{semiring\text{-}1}) \Longrightarrow \mathit{finite}\ A$
  ⟨*proof*⟩

1.3 Lists

1.3.1 *zip*

lemma *zip-truncate-left* : $\mathit{zip}\ xs\ ys = \mathit{zip}\ (\mathit{take}\ (\mathit{length}\ ys)\ xs)\ ys$
  ⟨*proof*⟩

lemma *zip-truncate-right* : $\mathit{zip}\ xs\ ys = \mathit{zip}\ xs\ (\mathit{take}\ (\mathit{length}\ xs)\ ys)$
  ⟨*proof*⟩

Lemmas *zip-append1* and *zip-append2* in theory *HOL.List* have unnecessary *take* (*length* -) in them. Here are replacements.

lemma *zip-append-left* :
  $\mathit{zip}\ (xs@ys)\ zs = \mathit{zip}\ xs\ zs\ @\ \mathit{zip}\ ys\ (\mathit{drop}\ (\mathit{length}\ xs)\ zs)$
  ⟨*proof*⟩

lemma *zip-append-right* :
  $\mathit{zip}\ xs\ (ys@zs) = \mathit{zip}\ xs\ ys\ @\ \mathit{zip}\ (\mathit{drop}\ (\mathit{length}\ ys)\ xs)\ zs$

$\langle proof \rangle$

lemma *length-concat-map-split-zip* :
$length\ [f\ x\ y.\ (x,y) \leftarrow zip\ xs\ ys] = min\ (length\ xs)\ (length\ ys)$
$\langle proof \rangle$

lemma *concat-map-split-eq-map-split-zip* :
$[f\ x\ y.\ (x,y) \leftarrow zip\ xs\ ys] = map\ (case\text{-}prod\ f)\ (zip\ xs\ ys)$
$\langle proof \rangle$

lemma *set-zip-map2* :
$(a,z) \in set\ (zip\ xs\ (map\ f\ ys)) \Longrightarrow \exists\, b.\ (a,b) \in set\ (zip\ xs\ ys) \wedge z = f\ b$
$\langle proof \rangle$

1.3.2 *concat*

lemma *concat-eq* :
$list\text{-}all2\ (\lambda xs\ ys.\ length\ xs = length\ ys)\ xss\ yss \Longrightarrow concat\ xss = concat\ yss$
$\Longrightarrow xss = yss$
$\langle proof \rangle$

lemma *match-concat* :
fixes $bss :: 'b\ list\ list$
defines $eq\text{-}len$: $eq\text{-}len \equiv \lambda xs\ ys.\ length\ xs = length\ ys$
shows $\forall\, as{::}'a\ list.\ length\ as = length\ (concat\ bss)$
$\longrightarrow (\exists\, css{::}'a\ list\ list.\ as = concat\ css \wedge list\text{-}all2\ eq\text{-}len\ css\ bss)$
$\langle proof \rangle$

1.3.3 *strip-while*

lemma *strip-while-0-nnil* :
$as \neq [] \Longrightarrow set\ as \neq 0 \Longrightarrow strip\text{-}while\ ((=)\ 0)\ as \neq []$
$\langle proof \rangle$

1.3.4 *sum-list*

lemma *const-sum-list* :
$\forall\, x \in set\ xs.\ f\ x = a \Longrightarrow sum\text{-}list\ (map\ f\ xs) = a * (length\ xs)$
$\langle proof \rangle$

lemma *sum-list-prod-cong* :
$\forall\, (x,y) \in set\ xys.\ f\ x\ y = g\ x\ y$
$\Longrightarrow (\sum (x,y) \leftarrow xys.\ f\ x\ y) = (\sum (x,y) \leftarrow xys.\ g\ x\ y)$
$\langle proof \rangle$

lemma *sum-list-prod-map2* :
$(\sum (a,y) \leftarrow zip\ as\ (map\ f\ bs).\ g\ a\ y) = (\sum (a,b) \leftarrow zip\ as\ bs.\ g\ a\ (f\ b))$
$\langle proof \rangle$

lemma *sum-list-fun-apply* : $(\sum x \leftarrow xs.\ f\ x)\ y = (\sum x \leftarrow xs.\ f\ x\ y)$

$\langle proof \rangle$

lemma *sum-list-prod-fun-apply* : $(\sum (x,y){\leftarrow}xys.\ f\ x\ y)\ z = (\sum (x,y){\leftarrow}xys.\ f\ x\ y\ z)$
$\langle proof \rangle$

lemma (**in** *comm-monoid-add*) *sum-list-plus* :
$length\ xs = length\ ys$
$\Longrightarrow sum\text{-}list\ xs + sum\text{-}list\ ys = sum\text{-}list\ [a{+}b.\ (a,b){\leftarrow}zip\ xs\ ys]$
$\langle proof \rangle$

lemma *sum-list-const-mult-prod* :
fixes $f :: {'a} \Rightarrow {'b} \Rightarrow {'r}::semiring\text{-}0$
shows $r * (\sum (x,y){\leftarrow}xys.\ f\ x\ y) = (\sum (x,y){\leftarrow}xys.\ r * (f\ x\ y))$
$\langle proof \rangle$

lemma *sum-list-mult-const-prod* :
fixes $f :: {'a} \Rightarrow {'b} \Rightarrow {'r}::semiring\text{-}0$
shows $(\sum (x,y){\leftarrow}xys.\ f\ x\ y) * r = (\sum (x,y){\leftarrow}xys.\ (f\ x\ y) * r)$
$\langle proof \rangle$

lemma *sum-list-update* :
fixes $xs :: {'a}::ab\text{-}group\text{-}add\ list$
shows $n < length\ xs \Longrightarrow sum\text{-}list\ (xs[n := y]) = sum\text{-}list\ xs - xs!n + y$
$\langle proof \rangle$

lemma *sum-list-replicate0* : $sum\text{-}list\ (replicate\ n\ 0) = 0$
$\langle proof \rangle$

1.3.5 *listset*

lemma *listset-ConsI* : $x \in X \Longrightarrow xs \in listset\ Xs \Longrightarrow x\#xs \in listset\ (X\#Xs)$
$\langle proof \rangle$

lemma *listset-ConsD* : $x\#xs \in listset\ (A\ \#\ As) \Longrightarrow x \in A \wedge xs \in listset\ As$
$\langle proof \rangle$

lemma *listset-Cons-conv* :
$xs \in listset\ (A\ \#\ As) \Longrightarrow (\exists\, y\ ys.\ y \in A \wedge ys \in listset\ As \wedge xs = y\#ys)$
$\langle proof \rangle$

lemma *listset-length* : $xs \in listset\ Xs \Longrightarrow length\ xs = length\ Xs$
$\langle proof \rangle$

lemma *set-sum-list-element* :
$x \in (\sum A{\leftarrow}As.\ A) \Longrightarrow \exists\, as \in listset\ As.\ x = (\sum a{\leftarrow}as.\ a)$
$\langle proof \rangle$

lemma *set-sum-list-element-Cons* :
assumes $x \in (\sum X{\leftarrow}(A\#As).\ X)$

shows $\exists\, a\ as.\ a{\in}A \wedge as \in listset\ As \wedge x = a + (\sum b{\leftarrow}as.\ b)$
⟨*proof*⟩

lemma *sum-list-listset* : $as \in listset\ As \Longrightarrow sum\text{-}list\ as \in (\sum A{\leftarrow}As.\ A)$
⟨*proof*⟩

lemma *listsetI-nth* :
$length\ xs = length\ Xs \Longrightarrow \forall\, n{<}length\ xs.\ xs!n \in Xs!n \Longrightarrow xs \in listset\ Xs$
⟨*proof*⟩

lemma *listsetD-nth* : $xs \in listset\ Xs \Longrightarrow \forall\, n{<}length\ xs.\ xs!n \in Xs!n$
⟨*proof*⟩

lemma *set-listset-el-subset* :
$xs \in listset\ Xs \Longrightarrow \forall\, X{\in}set\ Xs.\ X \subseteq A \Longrightarrow set\ xs \subseteq A$
⟨*proof*⟩

1.4 Functions

1.4.1 Miscellaneous facts

lemma *sum-fun-apply* : $finite\ A \Longrightarrow (\sum a{\in}A.\ f\ a)\ x = (\sum a{\in}A.\ f\ a\ x)$
⟨*proof*⟩

lemma *sum-single-nonzero* :
$finite\ A \Longrightarrow (\forall\, x{\in}A.\ \forall\, y{\in}A.\ f\ x\ y = (if\ y = x\ then\ g\ x\ else\ 0))$
$\Longrightarrow (\forall\, x{\in}A.\ sum\ (f\ x)\ A = g\ x)$
⟨*proof*⟩

lemma *distrib-comp-sum-right* : $(T + T') \circ S = (T \circ S) + (T' \circ S)$
⟨*proof*⟩

1.4.2 Support of a function

definition *supp* :: $('a \Rightarrow 'b{::}zero) \Rightarrow 'a\ set$ **where** $supp\ f = \{x.\ f\ x \neq 0\}$

lemma *suppI*: $f\ x \neq 0 \Longrightarrow x \in supp\ f$
⟨*proof*⟩

lemma *suppI-contra*: $x \notin supp\ f \Longrightarrow f\ x = 0$
⟨*proof*⟩

lemma *suppD*: $x \in supp\ f \Longrightarrow f\ x \neq 0$
⟨*proof*⟩

lemma *suppD-contra*: $f\ x = 0 \Longrightarrow x \notin supp\ f$
⟨*proof*⟩

lemma *zerofun-imp-empty-supp* : $supp\ 0 = \{\}$
⟨*proof*⟩

lemma *supp-zerofun-subset-any* : *supp 0* $\subseteq$ *A*
⟨*proof*⟩

lemma *supp-sum-subset-union-supp* :
fixes *f g* :: *'a* $\Rightarrow$ *'b*::*monoid-add*
shows *supp* (*f* + *g*) $\subseteq$ *supp f* $\cup$ *supp g*
⟨*proof*⟩

lemma *supp-neg-eq-supp* :
fixes *f* :: *'a* $\Rightarrow$ *'b*::*group-add*
shows *supp* (− *f*) = *supp f*
⟨*proof*⟩

lemma *supp-diff-subset-union-supp* :
fixes *f g* :: *'a* $\Rightarrow$ *'b*::*group-add*
shows *supp* (*f* − *g*) $\subseteq$ *supp f* $\cup$ *supp g*
⟨*proof*⟩

abbreviation *restrict0* :: (*'a*$\Rightarrow$*'b*::*zero*) $\Rightarrow$ *'a set* $\Rightarrow$ (*'a*$\Rightarrow$*'b*) (**infix** ↓ *70*)
where *restrict0 f A* $\equiv$ (λ*a*. *if a* $\in$ *A then f a else 0*)

lemma *supp-restrict0* : *supp* (*f*↓*A*) $\subseteq$ *A*
⟨*proof*⟩

lemma *bij-betw-restrict0* : *bij-betw f A B* $\Longrightarrow$ *bij-betw* (*f* ↓ *A*) *A B*
⟨*proof*⟩

1.4.3 Convolution

definition *convolution* ::
(*'a*::*group-add* $\Rightarrow$ *'b*::{*comm-monoid-add*,*times*}) $\Rightarrow$ (*'a*$\Rightarrow$*'b*) $\Rightarrow$ (*'a*$\Rightarrow$*'b*)
where *convolution f g*
= ($\lambda x. \sum y | x - y \in supp\ f \wedge y \in supp\ g.\ (f\ (x - y)) * g\ y$)
— More often than not, this definition will be used in the case that *'b* is of class *mult-zero*, in which case the conditions $x - y \in supp\ f$ and $y \in supp\ g$ are obviously mathematically unnecessary. However, they also serve to ensure that the sum is taken over a finite set in the case that at least one of *f* and *g* is almost everywhere zero.

lemma *convolution-zero* :
fixes *f g* :: *'a*::*group-add* $\Rightarrow$ *'b*::{*comm-monoid-add*,*mult-zero*}
shows *f* = *0* $\vee$ *g* = *0* $\Longrightarrow$ *convolution f g* = *0*
⟨*proof*⟩

lemma *convolution-symm* :
fixes *f g* :: *'a*::*group-add* $\Rightarrow$ *'b*::{*comm-monoid-add*,*times*}
shows *convolution f g*
= ($\lambda x. \sum y | y \in supp\ f \wedge -y + x \in supp\ g.\ (f\ y) * g\ (-y + x)$)

⟨*proof*⟩

lemma *supp-convolution-subset-sum-supp* :
fixes $f\ g :: 'a::group\text{-}add \Rightarrow 'b::\{comm\text{-}monoid\text{-}add,times\}$
shows $supp\ (convolution\ f\ g) \subseteq supp\ f + supp\ g$
⟨*proof*⟩

1.5 Almost-everywhere-zero functions

1.5.1 Definition and basic properties

definition $aezfun\text{-}set = \{f::'a \Rightarrow 'b::zero.\ finite\ (supp\ f)\}$

lemma *aezfun-setD*: $f \in aezfun\text{-}set \Longrightarrow finite\ (supp\ f)$
⟨*proof*⟩

lemma *aezfun-setI*: $finite\ (supp\ f) \Longrightarrow f \in aezfun\text{-}set$
⟨*proof*⟩

lemma *zerofun-is-aezfun* : $0 \in aezfun\text{-}set$
⟨*proof*⟩

lemma *sum-of-aezfun-is-aezfun* :
fixes $f\ g :: 'a \Rightarrow 'b::monoid\text{-}add$
shows $f \in aezfun\text{-}set \Longrightarrow g \in aezfun\text{-}set \Longrightarrow f + g \in aezfun\text{-}set$
⟨*proof*⟩

lemma *neg-of-aezfun-is-aezfun* :
fixes $f :: 'a \Rightarrow 'b::group\text{-}add$
shows $f \in aezfun\text{-}set \Longrightarrow -\ f \in aezfun\text{-}set$
⟨*proof*⟩

lemma *diff-of-aezfun-is-aezfun* :
fixes $f\ g :: 'a \Rightarrow 'b::group\text{-}add$
shows $f \in aezfun\text{-}set \Longrightarrow g \in aezfun\text{-}set \Longrightarrow f - g \in aezfun\text{-}set$
⟨*proof*⟩

lemma *restrict-and-extend0-aezfun-is-aezfun* :
assumes $f \in aezfun\text{-}set$
shows $f{\downarrow}A \in aezfun\text{-}set$
⟨*proof*⟩

1.5.2 Delta (impulse) functions

The notation is set up in the order output-input so that later when these are used to define the group ring RG, it will be in order ring-element-group-element.

definition $deltafun :: 'b::zero \Rightarrow 'a \Rightarrow ('a \Rightarrow 'b)$ (**infix** δ *70*)
where $b\ \delta\ a = (\lambda x.\ if\ x = a\ then\ b\ else\ 0)$

lemma *deltafun-apply-eq* : $(b\ \delta\ a)\ a = b$
⟨*proof*⟩

lemma *deltafun-apply-neq* : $x \neq a \Longrightarrow (b\ \delta\ a)\ x = 0$
⟨*proof*⟩

lemma *deltafun0* : $0\ \delta\ a = 0$
⟨*proof*⟩

lemma *deltafun-plus* :
fixes $b\ c$ :: $'b$::*monoid-add*
shows $(b+c)\ \delta\ a = (b\ \delta\ a) + (c\ \delta\ a)$
⟨*proof*⟩

lemma *supp-delta0fun* : *supp* $(0\ \delta\ a) = \{\}$
⟨*proof*⟩

lemma *supp-deltafun* : $b \neq 0 \Longrightarrow$ *supp* $(b\ \delta\ a) = \{a\}$
⟨*proof*⟩

lemma *deltafun-is-aezfun* : $b\ \delta\ a \in$ *aezfun-set*
⟨*proof*⟩

lemma *aezfun-common-supp-spanning-set′* :
finite $A \Longrightarrow \exists$ *as. distinct as* $\wedge$ *set as* $= A$
$\wedge$ ($\forall f$::$'a \Rightarrow 'b$::*semiring-1. supp* $f \subseteq A$
$\longrightarrow$ ($\exists$ *bs. length bs* $=$ *length as* $\wedge f = (\sum (b,a) \leftarrow$ *zip bs as.* $b\ \delta\ a))$)
⟨*proof*⟩

1.5.3 Convolution of almost-everywhere-zero functions

lemma *convolution-eq-sum-over-supp-right* :
fixes $g\ f$ :: $'a$::*group-add* $\Rightarrow$ $'b$::{*comm-monoid-add*,*mult-zero*}
assumes $g \in$ *aezfun-set*
shows *convolution* $f\ g = (\lambda x.\ \sum y \in$ *supp* $g.\ (f\ (x - y)) * g\ y$)
⟨*proof*⟩

lemma *convolution-symm-eq-sum-over-supp-left* :
fixes $f\ g$ :: $'a$::*group-add* $\Rightarrow$ $'b$::{*comm-monoid-add*,*mult-zero*}
assumes $f \in$ *aezfun-set*
shows *convolution* $f\ g = (\lambda x.\ \sum y \in$ *supp* $f.\ (f\ y) * g\ (-y + x))$
⟨*proof*⟩

lemma *convolution-delta-left* :
fixes b :: $'b$::{*comm-monoid-add*,*mult-zero*}
and a :: $'a$::*group-add*
and f :: $'a \Rightarrow 'b$
shows *convolution* $(b\ \delta\ a)\ f = (\lambda x.\ b * f\ (-a + x))$

⟨*proof*⟩

lemma *convolution-delta-right* :
fixes b :: $'b$::{*comm-monoid-add*,*mult-zero*}
and f :: $'a$::*group-add* ⇒ $'b$ **and** a::$'a$
shows *convolution* f $(b\ \delta\ a) = (\lambda x.\ f\ (x - a) * b)$
⟨*proof*⟩

lemma *convolution-delta-delta* :
fixes $b1\ b2$ :: $'b$::{*comm-monoid-add*,*mult-zero*}
and $a1\ a2$ :: $'a$::*group-add*
shows *convolution* $(b1\ \delta\ a1)\ (b2\ \delta\ a2) = (b1 * b2)\ \delta\ (a1 + a2)$
⟨*proof*⟩

lemma *convolution-of-aezfun-is-aezfun* :
fixes $f\ g$ :: $'a$::*group-add* ⇒ $'b$::{*comm-monoid-add*,*times*}
shows $f \in$ *aezfun-set* $\Longrightarrow g \in$ *aezfun-set* $\Longrightarrow$ *convolution* $f\ g \in$ *aezfun-set*
⟨*proof*⟩

lemma *convolution-assoc* :
fixes $f\ h\ g$ :: $'a$::*group-add* ⇒ $'b$::*semiring-0*
assumes *f-aez*: $f \in$ *aezfun-set* **and** *h-aez*: $h \in$ *aezfun-set*
shows *convolution* (*convolution* $f\ g$) h = *convolution* f (*convolution* $g\ h$)
⟨*proof*⟩

lemma *convolution-distrib-left* :
fixes $g\ h\ f$ :: $'a$::*group-add* ⇒ $'b$::*semiring-0*
assumes $g \in$ *aezfun-set* $h \in$ *aezfun-set*
shows *convolution* f $(g + h)$ = *convolution* $f\ g$ + *convolution* $f\ h$
⟨*proof*⟩

lemma *convolution-distrib-right* :
fixes $f\ g\ h$ :: $'a$::*group-add* ⇒ $'b$::*semiring-0*
assumes $f \in$ *aezfun-set* $g \in$ *aezfun-set*
shows *convolution* $(f + g)$ h = *convolution* $f\ h$ + *convolution* $g\ h$
⟨*proof*⟩

1.5.4 Type definition, instantiations, and instances

typedef (**overloaded**) ($'a$::*zero*,$'b$) *aezfun* = *aezfun-set* :: ($'b$⇒$'a$) *set*
morphisms *aezfun Abs-aezfun*
⟨*proof*⟩

setup-lifting *type-definition-aezfun*

lemma *aezfun-finite-supp* : *finite* (*supp* (*aezfun* a))
⟨*proof*⟩

lemma *aezfun-transfer* : *aezfun* a = *aezfun* $b \Longrightarrow a = b$ ⟨*proof*⟩

instantiation *aezfun* :: (*zero*, *type*) *zero*
begin
  lift-definition *zero-aezfun* :: ($'a$,$'b$) *aezfun* **is** $0::'b\Rightarrow'a$
    ⟨*proof*⟩
  instance ⟨*proof*⟩
end

lemma *zero-aezfun-transfer* : *Abs-aezfun* $((0::'b::zero)\ \delta\ (0::'a::zero)) = 0$
⟨*proof*⟩

lemma *zero-aezfun-apply* [*simp*]: *aezfun* $0\ x = 0$
  ⟨*proof*⟩

instantiation *aezfun* :: (*monoid-add*, *type*) *plus*
begin
  lift-definition *plus-aezfun* :: ($'a$, $'b$) *aezfun* ⇒ ($'a$, $'b$) *aezfun* ⇒ ($'a$, $'b$) *aezfun*
    is $\lambda f\ g.\ f + g$
    ⟨*proof*⟩
  instance ⟨*proof*⟩
end

lemma *plus-aezfun-apply* [*simp*]: *aezfun* $(a+b)\ x =$ *aezfun* $a\ x +$ *aezfun* $b\ x$
  ⟨*proof*⟩

instance *aezfun* :: (*monoid-add*, *type*) *semigroup-add*
⟨*proof*⟩

instance *aezfun* :: (*monoid-add*, *type*) *monoid-add*
⟨*proof*⟩

lemma *sum-list-aezfun-apply* [*simp*] :
  aezfun (*sum-list as*) $x = (\sum a\leftarrow as.\ aezfun\ a\ x)$
  ⟨*proof*⟩

lemma *sum-list-map-aezfun-apply* [*simp*] :
  aezfun $(\sum a\leftarrow as.\ f\ a)\ x = (\sum a\leftarrow as.\ aezfun\ (f\ a)\ x)$
  ⟨*proof*⟩

lemma *sum-list-map-aezfun* [*simp*] :
  aezfun $(\sum a\leftarrow as.\ f\ a) = (\sum a\leftarrow as.\ aezfun\ (f\ a))$
  ⟨*proof*⟩

lemma *sum-list-prod-map-aezfun-apply* :
  aezfun $(\sum (x,y)\leftarrow xys.\ f\ x\ y)\ a = (\sum (x,y)\leftarrow xys.\ aezfun\ (f\ x\ y)\ a)$
  ⟨*proof*⟩

lemma *sum-list-prod-map-aezfun* :
  aezfun $(\sum (x,y)\leftarrow xys.\ f\ x\ y) = (\sum (x,y)\leftarrow xys.\ aezfun\ (f\ x\ y))$

⟨*proof*⟩

instance *aezfun* :: (*comm-monoid-add*, *type*) *comm-monoid-add*
⟨*proof*⟩

lemma *sum-aezfun-apply* [*simp*] :
finite $A \Longrightarrow$ *aezfun* $(\sum A)$ $x = (\sum a{\in}A.$ *aezfun* a $x)$
⟨*proof*⟩

instantiation *aezfun* :: (*group-add*, *type*) *minus*
begin
lift-definition *minus-aezfun* :: ($'a$, $'b$) *aezfun* ⇒ ($'a$, $'b$) *aezfun* ⇒ ($'a$, $'b$) *aezfun*
is $\lambda f\ g.\ f - g$
⟨*proof*⟩
instance ⟨*proof*⟩
end

lemma *minus-aezfun-apply* [*simp*]: *aezfun* $(a-b)$ $x =$ *aezfun* a $x -$ *aezfun* b x
⟨*proof*⟩

instantiation *aezfun* :: (*group-add*, *type*) *uminus*
begin
lift-definition *uminus-aezfun* :: ($'a$, $'b$) *aezfun* ⇒ ($'a$, $'b$) *aezfun* **is** $\lambda f.\ - f$
⟨*proof*⟩
instance ⟨*proof*⟩
end

lemma *uminus-aezfun-apply* [*simp*]: *aezfun* $(-a)$ $x = -$ *aezfun* a x
⟨*proof*⟩

lemma *aezfun-left-minus* [*simp*] :
fixes a :: ($'a$::*group-add*, $'b$) *aezfun*
shows $- a + a = 0$
⟨*proof*⟩

lemma *aezfun-diff-minus* [*simp*] :
fixes a b :: ($'a$::*group-add*, $'b$) *aezfun*
shows $a - b = a + - b$
⟨*proof*⟩

instance *aezfun* :: (*group-add*, *type*) *group-add*
⟨*proof*⟩

instance *aezfun* :: (*ab-group-add*, *type*) *ab-group-add*
⟨*proof*⟩

instantiation *aezfun* :: ({*one*,*zero*}, *zero*) *one*
begin
lift-definition *one-aezfun* :: ($'a$,$'b$) *aezfun* **is** $1\ \delta\ 0$

⟨proof⟩
instance ⟨proof⟩
end

lemma *one-aezfun-transfer* : *Abs-aezfun* $(1\ \delta\ 0) = 1$
⟨proof⟩

lemma *one-aezfun-apply* [*simp*]: *aezfun* $1\ x = (1\ \delta\ 0)\ x$
⟨proof⟩

lemma *one-aezfun-apply-eq* [*simp*]: *aezfun* $1\ 0 = 1$
⟨proof⟩

lemma *one-aezfun-apply-neq* [*simp*]: $x \neq 0 \Longrightarrow$ *aezfun* $1\ x = 0$
⟨proof⟩

instance *aezfun* :: (*zero-neq-one*, *zero*) *zero-neq-one*
⟨proof⟩

instantiation *aezfun* :: ({*comm-monoid-add*,*times*}, *group-add*) *times*
begin
lift-definition *times-aezfun* :: ('a, 'b) *aezfun* ⇒ ('a, 'b) *aezfun* ⇒ ('a, 'b) *aezfun*
is λ *f g. convolution f g*
⟨proof⟩
instance ⟨proof⟩
end

lemma *convolution-transfer* :
assumes $f \in$ *aezfun-set* $g \in$ *aezfun-set*
shows *Abs-aezfun* (*convolution f g*) = *Abs-aezfun f* ∗ *Abs-aezfun g*
⟨proof⟩

instance *aezfun* :: ({*comm-monoid-add*,*mult-zero*}, *group-add*) *mult-zero*
⟨proof⟩

instance *aezfun* :: (*semiring-0*, *group-add*) *semiring-0*
⟨proof⟩

instance *aezfun* :: (*ring*, *group-add*) *ring* ⟨proof⟩

instance *aezfun* :: ({*semiring-0*,*monoid-mult*,*zero-neq-one*}, *group-add*) *monoid-mult*
⟨proof⟩

instance *aezfun* :: (*ring-1*, *group-add*) *ring-1* ⟨proof⟩

1.5.5 Transfer facts

abbreviation *aezdeltafun* :: 'b::*zero* ⇒ 'a ⇒ ('b,'a) *aezfun* (**infix** $\delta\delta$ *70*)
where $b\ \delta\delta\ a \equiv$ *Abs-aezfun* $(b\ \delta\ a)$

lemma *aezdeltafun* : *aezfun* $(b\ \delta\delta\ a) = b\ \delta\ a$
⟨*proof*⟩

lemma *aezdeltafun-plus* : $(b{+}c)\ \delta\delta\ a = (b\ \delta\delta\ a) + (c\ \delta\delta\ a)$
⟨*proof*⟩

lemma *times-aezdeltafun-aezdeltafun* :
fixes *b1 b2* :: *'b*::{*comm-monoid-add*,*mult-zero*}
shows $(b1\ \delta\delta\ a1) * (b2\ \delta\delta\ a2) = (b1 * b2)\ \delta\delta\ (a1 + a2)$
⟨*proof*⟩

lemma *aezfun-restrict-and-extend0* : (*aezfun* x)$\downarrow A \in$ *aezfun-set*
⟨*proof*⟩

lemma *aezdeltafun-decomp* :
fixes b :: *'b*::*semiring-1*
shows $b\ \delta\delta\ a = (b\ \delta\delta\ 0) * (1\ \delta\delta\ a)$
⟨*proof*⟩

lemma *aezdeltafun-decomp'* :
fixes b :: *'b*::*semiring-1*
shows $b\ \delta\delta\ a = (1\ \delta\delta\ a) * (b\ \delta\delta\ 0)$
⟨*proof*⟩

lemma *supp-aezfun1* :
supp (*aezfun* (*1* :: (*'a*::*zero-neq-one*,*'b*::*zero*) *aezfun*)) = *0*
⟨*proof*⟩

lemma *supp-aezfun-diff* :
supp (*aezfun* $(x - y)$) $\subseteq$ *supp* (*aezfun* x) $\cup$ *supp* (*aezfun* y)
⟨*proof*⟩

lemma *supp-aezfun-times* :
supp (*aezfun* $(x * y)$) $\subseteq$ *supp* (*aezfun* x) + *supp* (*aezfun* y)
⟨*proof*⟩

1.5.6 Almost-everywhere-zero functions with constrained support

The name of the next definition anticipates *aezfun-common-supp-spanning-set* below.

definition *aezfun-setspan* :: *'a set* $\Rightarrow$ (*'b*::*zero*,*'a*) *aezfun set*
where *aezfun-setspan* $A = \{x.$ *supp* (*aezfun* x) $\subseteq A\}$

lemma *aezfun-setspanD* : $x \in$ *aezfun-setspan* $A \Longrightarrow$ *supp* (*aezfun* x) $\subseteq A$
⟨*proof*⟩

lemma *aezfun-setspanI* : *supp* (*aezfun* x) $\subseteq A \Longrightarrow x \in$ *aezfun-setspan* A
⟨*proof*⟩

lemma *aezfun-common-supp-spanning-set* :
  assumes *finite A*
  shows $\exists\, as.\ distinct\ as \wedge set\ as = A \wedge ($
    $\forall\, x{::}('b{::}semiring\text{-}1,'a)\ aezfun \in aezfun\text{-}setspan\ A.$
      $\exists\, bs.\ length\ bs = length\ as \wedge x = (\sum (b,a){\leftarrow}zip\ bs\ as.\ b\ \delta\delta\ a)$
  $)$
⟨*proof*⟩

lemma *aezfun-common-supp-spanning-set-decomp* :
  fixes $G :: 'g{::}group\text{-}add\ set$
  assumes *finite G*
  shows $\exists\, gs.\ distinct\ gs \wedge set\ gs = G \wedge ($
    $\forall\, x{::}('r{::}semiring\text{-}1,'g)\ aezfun \in aezfun\text{-}setspan\ G.$
      $\exists\, rs.\ length\ rs = length\ gs$
        $\wedge\ x = (\sum (r,g){\leftarrow}zip\ rs\ gs.\ (r\ \delta\delta\ 0) * (1\ \delta\delta\ g))$
  $)$
⟨*proof*⟩

lemma *aezfun-decomp-aezdeltafun* :
  fixes $c :: ('r{::}semiring\text{-}1,'a)\ aezfun$
  shows $\exists\, ras.\ set\ (map\ snd\ ras) = supp\ (aezfun\ c) \wedge c = (\sum (r,a){\leftarrow}ras.\ r\ \delta\delta\ a)$
⟨*proof*⟩

lemma *aezfun-setspan-el-decomp-aezdeltafun* :
  fixes $c :: ('r{::}semiring\text{-}1,'a)\ aezfun$
  shows $c \in aezfun\text{-}setspan\ A$
    $\Longrightarrow \exists\, ras.\ set\ (map\ snd\ ras) \subseteq A \wedge c = (\sum (r,a){\leftarrow}ras.\ r\ \delta\delta\ a)$
  ⟨*proof*⟩

lemma *aezdelta0fun-commutes'* :
  fixes $b1\ b2 :: 'b{::}comm\text{-}semiring\text{-}1$
  shows $b1\ \delta\delta\ a * (b2\ \delta\delta\ 0) = b2\ \delta\delta\ 0 * (b1\ \delta\delta\ a)$
  ⟨*proof*⟩

lemma *aezdelta0fun-commutes* :
  fixes $b :: 'b{::}comm\text{-}semiring\text{-}1$
  shows $c * (b\ \delta\delta\ 0) = b\ \delta\delta\ 0 * c$
⟨*proof*⟩

The following definition constrains the support of arbitrary almost-everywhere-zero functions, as a sort of projection onto a *aezfun-setspan*.

definition $aezfun\text{-}setspan\text{-}proj :: 'a\ set \Rightarrow ('b{::}zero,'a)\ aezfun \Rightarrow ('b{::}zero,'a)\ aezfun$
  where $aezfun\text{-}setspan\text{-}proj\ A\ x \equiv Abs\text{-}aezfun\ ((aezfun\ x){\downarrow}A)$

lemma *aezfun-setspan-projD1* :
  $a \in A \Longrightarrow aezfun\ (aezfun\text{-}setspan\text{-}proj\ A\ x)\ a = aezfun\ x\ a$
  ⟨*proof*⟩

lemma *aezfun-setspan-projD2* :
$a \notin A \Longrightarrow aezfun\ (aezfun\text{-}setspan\text{-}proj\ A\ x)\ a = 0$
⟨*proof*⟩

lemma *aezfun-setspan-proj-in-setspan* :
$aezfun\text{-}setspan\text{-}proj\ A\ x \in aezfun\text{-}setspan\ A$
⟨*proof*⟩

lemma *aezfun-setspan-proj-zero* : $aezfun\text{-}setspan\text{-}proj\ A\ 0 = 0$
⟨*proof*⟩

lemma *aezfun-setspan-proj-aezdeltafun* :
$aezfun\text{-}setspan\text{-}proj\ A\ (b\ \delta\delta\ a) = (if\ a \in A\ then\ b\ \delta\delta\ a\ else\ 0)$
⟨*proof*⟩

lemma *aezfun-setspan-proj-add* :
$aezfun\text{-}setspan\text{-}proj\ A\ (x+y)$
$= aezfun\text{-}setspan\text{-}proj\ A\ x + aezfun\text{-}setspan\text{-}proj\ A\ y$
⟨*proof*⟩

lemma *aezfun-setspan-proj-sum-list* :
$aezfun\text{-}setspan\text{-}proj\ A\ (\sum x \leftarrow xs.\ f\ x) = (\sum x \leftarrow xs.\ aezfun\text{-}setspan\text{-}proj\ A\ (f\ x))$
⟨*proof*⟩

lemma *aezfun-setspan-proj-sum-list-prod* :
$aezfun\text{-}setspan\text{-}proj\ A\ (\sum (x,y) \leftarrow xys.\ f\ x\ y)$
$= (\sum (x,y) \leftarrow xys.\ aezfun\text{-}setspan\text{-}proj\ A\ (f\ x\ y))$
⟨*proof*⟩

1.6 Polynomials

lemma *nonzero-coeffs-nonzero-poly* : $as \neq [] \Longrightarrow set\ as \neq 0 \Longrightarrow Poly\ as \neq 0$
⟨*proof*⟩

lemma *const-poly-nonzero-coeff* :
assumes $degree\ p = 0\ p \neq 0$
shows $coeff\ p\ 0 \neq 0$
⟨*proof*⟩

lemma *pCons-induct2* [*case-names 00 lpCons rpCons pCons2*]:
assumes *00*: $P\ 0\ 0$
and *lpCons*: $\bigwedge a\ p.\ a \neq 0 \lor p \neq 0 \Longrightarrow P\ (pCons\ a\ p)\ 0$
and *rpCons*: $\bigwedge b\ q.\ b \neq 0 \lor q \neq 0 \Longrightarrow P\ 0\ (pCons\ b\ q)$
and *pCons2*: $\bigwedge a\ p\ b\ q.\ a \neq 0 \lor p \neq 0 \Longrightarrow b \neq 0 \lor q \neq 0 \Longrightarrow P\ p\ q$
$\Longrightarrow P\ (pCons\ a\ p)\ (pCons\ b\ q)$
shows $P\ p\ q$
⟨*proof*⟩

1.7 Algebra of sets

1.7.1 General facts

lemma *zeroset-eqI*: $0 \in A \Longrightarrow (\bigwedge a.\ a \in A \Longrightarrow a = 0) \Longrightarrow A = 0$
⟨*proof*⟩

lemma *sum-list-sets-single* : $(\sum X{\leftarrow}[A].\ X) = A$
⟨*proof*⟩

lemma *sum-list-sets-double* : $(\sum X{\leftarrow}[A,B].\ X) = A + B$
⟨*proof*⟩

1.7.2 Additive independence of sets

primrec *add-independentS* :: *'a::monoid-add set list* ⇒ *bool*
where *add-independentS* [] = *True*
| *add-independentS* (*A#As*) = (*add-independentS As*
∧ $(\forall x{\in}(\sum B{\leftarrow}As.\ B).\ \forall a{\in}A.\ a + x = 0 \longrightarrow a = 0)$)

lemma *add-independentS-doubleI*:
assumes $\bigwedge b\ a.\ b{\in}B \Longrightarrow a{\in}A \Longrightarrow a + b = 0 \Longrightarrow a = 0$
shows *add-independentS* [*A*,*B*]
⟨*proof*⟩

lemma *add-independentS-doubleD*:
assumes *add-independentS* [*A*,*B*]
shows $\bigwedge b\ a.\ b{\in}B \Longrightarrow a{\in}A \Longrightarrow a + b = 0 \Longrightarrow a = 0$
⟨*proof*⟩

lemma *add-independentS-double-iff* :
add-independentS [*A*,*B*] = $(\forall b{\in}B.\ \forall a{\in}A.\ a + b = 0 \longrightarrow a = 0)$
⟨*proof*⟩

lemma *add-independentS-Cons-conv-sum-right* :
add-independentS (*A#As*)
= (*add-independentS* [$A,\sum B{\leftarrow}As.\ B$] ∧ *add-independentS As*)
⟨*proof*⟩

lemma *add-independentS-double-sum-conv-append* :
⟦ $\forall X{\in}set\ As.\ 0 \in X$; *add-independentS As*; *add-independentS Bs*;
add-independentS [$\sum X{\leftarrow}As.\ X,\ \sum X{\leftarrow}Bs.\ X$] ⟧
⟹ *add-independentS* (*As@Bs*)
⟨*proof*⟩

lemma *add-independentS-ConsI* :
assumes *add-independentS As*
$\bigwedge x\ a.$ ⟦ $x{\in}(\sum X{\leftarrow}As.\ X)$; $a \in A$; $a{+}x = 0$ ⟧ $\Longrightarrow a = 0$
shows *add-independentS* (*A#As*)
⟨*proof*⟩

lemma *add-independentS-append-reduce-right* :
  $add\text{-}independentS\ (As@Bs) \Longrightarrow add\text{-}independentS\ Bs$
  ⟨*proof*⟩

lemma *add-independentS-append-reduce-left* :
  $add\text{-}independentS\ (As@Bs) \Longrightarrow 0 \in (\sum X{\leftarrow}Bs.\ X) \Longrightarrow add\text{-}independentS\ As$
⟨*proof*⟩

lemma *add-independentS-append-conv-double-sum* :
  $add\text{-}independentS\ (As@Bs) \Longrightarrow add\text{-}independentS\ [\sum X{\leftarrow}As.\ X,\ \sum X{\leftarrow}Bs.\ X]$
⟨*proof*⟩

1.7.3 Inner direct sums

definition *inner-dirsum* :: $'a{::}monoid\text{-}add\ set\ list \Rightarrow\ 'a\ set$
  where $inner\text{-}dirsum\ As = (if\ add\text{-}independentS\ As\ then\ \sum A{\leftarrow}As.\ A\ else\ 0)$

Some syntactic sugar for *inner-dirsum*, borrowed from theory *HOL.List*.

syntax
  -inner-dirsum :: *pttrn => 'a list => 'b => 'b*
  $((3\bigoplus\text{-}{\leftarrow}\text{-}.\ \text{-})\ [0,\ 51,\ 10]\ 10)$
translations — Beware of argument permutation!
  $\bigoplus M{\leftarrow}Ms.\ b == CONST\ inner\text{-}dirsum\ (CONST\ map\ (\%M.\ b)\ Ms)$

abbreviation *inner-dirsum-double* ::
  $'a{::}monoid\text{-}add\ set \Rightarrow\ 'a\ set \Rightarrow\ 'a\ set$ (**infixr** $\oplus$ *70*)
  where $inner\text{-}dirsum\text{-}double\ A\ B \equiv inner\text{-}dirsum\ [A,B]$

lemma *inner-dirsumI* :
  $M = (\sum N{\leftarrow}Ns.\ N) \Longrightarrow add\text{-}independentS\ Ns \Longrightarrow M = (\bigoplus N{\leftarrow}Ns.\ N)$
  ⟨*proof*⟩

lemma *inner-dirsum-doubleI* :
  $M = A + B \Longrightarrow add\text{-}independentS\ [A,B] \Longrightarrow M = A \oplus B$
  ⟨*proof*⟩

lemma *inner-dirsumD* :
  $add\text{-}independentS\ Ms \Longrightarrow (\bigoplus M{\leftarrow}Ms.\ M) = (\sum M{\leftarrow}Ms.\ M)$
  ⟨*proof*⟩

lemma *inner-dirsumD2* : $\neg\ add\text{-}independentS\ Ms \Longrightarrow (\bigoplus M{\leftarrow}Ms.\ M) = 0$
  ⟨*proof*⟩

lemma *inner-dirsum-Nil* : $(\bigoplus A{\leftarrow}[].\ A) = 0$
  ⟨*proof*⟩

lemma *inner-dirsum-singleD* : $(\bigoplus N{\leftarrow}[M].\ N) = M$
  ⟨*proof*⟩

lemma *inner-dirsum-doubleD* : *add-independentS* $[M,N] \Longrightarrow M \oplus N = M + N$
⟨*proof*⟩

lemma *inner-dirsum-Cons* :
add-independentS $(A \# As) \Longrightarrow (\bigoplus X{\leftarrow}(A\#As).\ X) = A \oplus (\bigoplus X{\leftarrow}As.\ X)$
⟨*proof*⟩

lemma *inner-dirsum-append* :
add-independentS $(As@Bs) \Longrightarrow 0 \in (\sum X{\leftarrow}Bs.\ X)$
$\Longrightarrow (\bigoplus X{\leftarrow}(As@Bs).\ X) = (\bigoplus X{\leftarrow}As.\ X) \oplus (\bigoplus X{\leftarrow}Bs.\ X)$
⟨*proof*⟩

lemma *inner-dirsum-double-left0*: $0 \oplus A = A$
⟨*proof*⟩

lemma *add-independentS-Cons-conv-dirsum-right* :
add-independentS $(A\#As) = ($*add-independentS* $[A,\bigoplus B{\leftarrow}As.\ B]$
$\wedge$ *add-independentS* $As)$
⟨*proof*⟩

lemma *sum-list-listset-dirsum* :
add-independentS $As \Longrightarrow as \in$ *listset* $As \Longrightarrow$ *sum-list* $as \in (\bigoplus A{\leftarrow}As.\ A)$
⟨*proof*⟩

2 Groups

2.1 Locales and basic facts

2.1.1 Locale *Group* and finite variant *FinGroup*

Define a *Group* to be a closed subset of *UNIV* for the *group-add* class.

locale *Group* =
fixes G :: *'g::group-add set*
assumes *nonempty* : $G \neq \{\}$
and *diff-closed*: $\bigwedge g\ h.\ g \in G \Longrightarrow h \in G \Longrightarrow g - h \in G$

lemma *trivial-Group* : *Group* 0
⟨*proof*⟩

locale *FinGroup* = *Group G*
for G :: *'g::group-add set*
\+ **assumes** *finite*: *finite G*

lemma (**in** *FinGroup*) *Group* : *Group G* ⟨*proof*⟩

lemma (**in** *Group*) *FinGroupI* : *finite G* $\Longrightarrow$ *FinGroup G* ⟨*proof*⟩

context *Group*

begin

abbreviation *Subgroup* ::
$'g$ *set* $\Rightarrow$ *bool* **where** *Subgroup* $H \equiv$ *Group* $H \wedge H \subseteq G$

lemma *SubgroupD1* : *Subgroup* $H \Longrightarrow$ *Group* H ⟨*proof*⟩

lemma *zero-closed* : $0 \in G$
⟨*proof*⟩

lemma *obtain-nonzero*: **assumes** $G \neq 0$ **obtains** g **where** $g \in G$ **and** $g \neq 0$
⟨*proof*⟩

lemma *zeroS-closed* : $0 \subseteq G$
⟨*proof*⟩

lemma *neg-closed* : $g \in G \Longrightarrow -g \in G$
⟨*proof*⟩

lemma *add-closed* : $g \in G \Longrightarrow h \in G \Longrightarrow g + h \in G$
⟨*proof*⟩

lemma *neg-add-closed* : $g \in G \Longrightarrow h \in G \Longrightarrow -g + h \in G$
⟨*proof*⟩

lemma *sum-list-closed* : *set* (*map* f *as*) $\subseteq G \Longrightarrow (\sum a \leftarrow as.\ f\ a) \in G$
⟨*proof*⟩

lemma *sum-list-closed-prod* :
set (*map* (*case-prod* f) *xys*) $\subseteq G \Longrightarrow (\sum (x,y) \leftarrow xys.\ f\ x\ y) \in G$
⟨*proof*⟩

lemma *set-plus-closed* : $A \subseteq G \Longrightarrow B \subseteq G \Longrightarrow A + B \subseteq G$
⟨*proof*⟩

lemma *zip-add-closed* :
set as $\subseteq G \Longrightarrow$ *set bs* $\subseteq G \Longrightarrow$ *set* $[a + b.\ (a,b) \leftarrow zip\ as\ bs] \subseteq G$
⟨*proof*⟩

lemma *list-diff-closed* :
set gs $\subseteq G \Longrightarrow$ *set hs* $\subseteq G \Longrightarrow$ *set* $[x-y.\ (x,y) \leftarrow zip\ gs\ hs] \subseteq G$
⟨*proof*⟩

lemma *add-closed-converse-right* : $g+x \in G \Longrightarrow g \in G \Longrightarrow x \in G$
⟨*proof*⟩

lemma *add-closed-inverse-right* : $x \notin G \Longrightarrow g \in G \Longrightarrow g+x \notin G$
⟨*proof*⟩

lemma *add-closed-converse-left* : $g{+}x \in G \Longrightarrow x \in G \Longrightarrow g \in G$
⟨*proof*⟩

lemma *add-closed-inverse-left* : $g \notin G \Longrightarrow x \in G \Longrightarrow g{+}x \notin G$
⟨*proof*⟩

lemma *right-translate-bij* :
assumes $g \in G$
shows *bij-betw* $(\lambda x.\ x + g)\ G\ G$
⟨*proof*⟩

lemma *right-translate-sum* : $g \in G \Longrightarrow (\sum h{\in}G.\ f\ h) = (\sum h{\in}G.\ f\ (h + g))$
⟨*proof*⟩

end

2.1.2 Abelian variant locale *AbGroup*

locale *AbGroup* = *Group G*
for G :: $'g$::*ab-group-add set*
begin

lemmas *nonempty* = *nonempty*
lemmas *zero-closed* = *zero-closed*
lemmas *diff-closed* = *diff-closed*
lemmas *add-closed* = *add-closed*
lemmas *neg-closed* = *neg-closed*

lemma *sum-closed* : *finite* $A \Longrightarrow f$ ‘ $A \subseteq G \Longrightarrow (\sum a{\in}A.\ f\ a) \in G$
⟨*proof*⟩

lemma *subset-plus-right* : $A \subseteq G + A$
⟨*proof*⟩

lemma *subset-plus-left* : $A \subseteq A + G$
⟨*proof*⟩

end

2.2 Right cosets

context *Group*
begin

definition *rcoset-rel* :: $'g$ *set* $\Rightarrow$ $('g{\times}'g)$ *set*
where *rcoset-rel* $H \equiv \{(g,g').\ g \in G \wedge g' \in G \wedge g - g' \in H\}$

lemma (**in** *Group*) *rcosets* :
assumes *subgrp*: *Subgroup H* **and** *g*: $g \in G$
shows (*rcoset-rel H*)‘‘$\{g\} = H + \{g\}$

$\langle proof \rangle$

lemma *rcoset-equiv* :
assumes *Subgroup H*
shows *equiv G (rcoset-rel H)*
$\langle proof \rangle$

lemma *rcoset0* : *Subgroup H* $\Longrightarrow$ *(rcoset-rel H)*‘‘$\{0\}$ = *H*
$\langle proof \rangle$

definition *is-rcoset-replist* :: $'g$ *set* $\Rightarrow$ $'g$ *list* $\Rightarrow$ *bool*
where *is-rcoset-replist H gs*
$\equiv$ *set gs* $\subseteq$ *G* $\wedge$ *distinct (map* $(\lambda g.$ *(rcoset-rel H)*‘‘$\{g\})$ *gs)*
$\wedge$ *G* = $(\bigcup g \in$ *set gs. (rcoset-rel H)*‘‘$\{g\})$

lemma *is-rcoset-replistD-set* : *is-rcoset-replist H gs* $\Longrightarrow$ *set gs* $\subseteq$ *G*
$\langle proof \rangle$

lemma *is-rcoset-replistD-distinct* :
is-rcoset-replist H gs $\Longrightarrow$ *distinct (map* $(\lambda g.$ *(rcoset-rel H)*‘‘$\{g\})$ *gs)*
$\langle proof \rangle$

lemma *is-rcoset-replistD-cosets* :
is-rcoset-replist H gs $\Longrightarrow$ *G* = $(\bigcup g \in$ *set gs. (rcoset-rel H)*‘‘$\{g\})$
$\langle proof \rangle$

lemma *group-eq-subgrp-rcoset-un* :
Subgroup H $\Longrightarrow$ *is-rcoset-replist H gs* $\Longrightarrow$ *G* = $(\bigcup g \in$ *set gs. H* + $\{g\})$
$\langle proof \rangle$

lemma *is-rcoset-replist-imp-nrelated-nth* :
assumes *Subgroup H is-rcoset-replist H gs*
shows $\bigwedge i\ j.\ i <$ *length gs* $\Longrightarrow j <$ *length gs* $\Longrightarrow i \neq j \Longrightarrow gs!i - gs!j \notin H$
$\langle proof \rangle$

lemma *is-rcoset-replist-Cons* :
is-rcoset-replist H (g#gs) $\longleftrightarrow$
$g \in G \wedge$ *set gs* $\subseteq G$
$\wedge$ *(rcoset-rel H)*‘‘$\{g\} \notin$ *set (map* $(\lambda x.$ *(rcoset-rel H)*‘‘$\{x\})$ *gs)*
$\wedge$ *distinct (map* $(\lambda x.$ *(rcoset-rel H)*‘‘$\{x\})$ *gs)*
$\wedge$ *G* = *(rcoset-rel H)*‘‘$\{g\} \cup (\bigcup x \in$ *set gs. (rcoset-rel H)*‘‘$\{x\})$
$\langle proof \rangle$

lemma *rcoset-replist-Hrep* :
assumes *Subgroup H is-rcoset-replist H gs*
shows $\exists g \in$ *set gs.* $g \in H$
$\langle proof \rangle$

lemma *rcoset-replist-reorder* :

$is\text{-}rcoset\text{-}replist\ H\ (gs\ @\ g\ \#\ gs') \Longrightarrow is\text{-}rcoset\text{-}replist\ H\ (g\ \#\ gs\ @\ gs')$
⟨*proof*⟩

lemma *rcoset-replist-replacehd* :
assumes $Subgroup\ H\ g' \in (rcoset\text{-}rel\ H)``\{g\}\ is\text{-}rcoset\text{-}replist\ H\ (g\ \#\ gs)$
shows $is\text{-}rcoset\text{-}replist\ H\ (g'\ \#\ gs)$
⟨*proof*⟩

end

lemma (**in** *FinGroup*) *ex-rcoset-replist* :
assumes *Subgroup H*
shows $\exists\, gs.\ is\text{-}rcoset\text{-}replist\ H\ gs$
⟨*proof*⟩

lemma (**in** *FinGroup*) *ex-rcoset-replist-hd0* :
assumes *Subgroup H*
shows $\exists\, gs.\ is\text{-}rcoset\text{-}replist\ H\ (0\#gs)$
⟨*proof*⟩

2.3 Group homomorphisms

2.3.1 Preliminaries

definition $ker :: ('a \Rightarrow 'b::zero) \Rightarrow 'a\ set$
where $ker\ f = \{a.\ f\ a = 0\}$

lemma *kerI* : $f\ a = 0 \Longrightarrow a \in ker\ f$
⟨*proof*⟩

lemma *kerD* : $a \in ker\ f \Longrightarrow f\ a = 0$
⟨*proof*⟩

lemma *ker-im-iff* : $(A \neq \{\} \wedge A \subseteq ker\ f) = (f\ `\ A = 0)$
⟨*proof*⟩

2.3.2 Locales

The *supp* condition is not strictly necessary, but helps with equality and uniqueness arguments.

locale *GroupHom* = *Group G*
for $G :: 'g::group\text{-}add\ set$
+ **fixes** $T :: 'g \Rightarrow 'h::group\text{-}add$
assumes *hom* : $\bigwedge g\ g'.\ g \in G \Longrightarrow g' \in G \Longrightarrow T\ (g + g') = T\ g + T\ g'$
and *supp*: $supp\ T \subseteq G$

abbreviation (**in** *GroupHom*) $Ker \equiv ker\ T \cap G$
abbreviation (**in** *GroupHom*) $ImG \equiv T\ `\ G$

locale *GroupEnd* = *GroupHom G T*
for G :: $'g$::*group-add set*
and T :: $'g \Rightarrow 'g$
+ **assumes** *endomorph*: $ImG \subseteq G$

locale *GroupIso* = *GroupHom G T*
for G :: $'g$::*group-add set*
and T :: $'g \Rightarrow 'h$::*group-add*
+ **fixes** H :: $'h$ *set*
assumes *bijective*: *bij-betw T G H*

2.3.3 Basic facts

lemma (**in** *Group*) *trivial-GroupHom* : *GroupHom G* $(0::('g \Rightarrow 'h::group\text{-}add))$
⟨*proof*⟩

lemma (**in** *Group*) *GroupHom-idhom* : *GroupHom G* $(id{\downarrow}G)$
⟨*proof*⟩

context *GroupHom*
begin

lemma *im-zero* : $T\ 0 = 0$
⟨*proof*⟩

lemma *zero-in-Ker* : $0 \in Ker$
⟨*proof*⟩

lemma *comp-zero* : $T \circ 0 = 0$
⟨*proof*⟩

lemma *im-neg* : $T\ (-\ g) = -\ T\ g$
⟨*proof*⟩

lemma *im-diff* : $g \in G \Longrightarrow g' \in G \Longrightarrow T\ (g - g') = T\ g - T\ g'$
⟨*proof*⟩

lemma *eq-im-imp-diff-in-Ker* : $[\![\ g \in G;\ h \in G;\ T\ g = T\ h\ ]\!] \Longrightarrow g - h \in Ker$
⟨*proof*⟩

lemma *im-sum-list-prod* :
$set\ (map\ (case\text{-}prod\ f)\ xys) \subseteq G$
$\Longrightarrow T\ (\sum (x,y){\leftarrow}xys.\ f\ x\ y) = (\sum (x,y){\leftarrow}xys.\ T\ (f\ x\ y))$
⟨*proof*⟩

lemma *distrib-comp-sum-left* :
$range\ S \subseteq G \Longrightarrow range\ S' \subseteq G \Longrightarrow T \circ (S + S') = (T \circ S) + (T \circ S')$
⟨*proof*⟩

lemma *Ker-Im-iff* : $(Ker = G) = (ImG = 0)$
⟨*proof*⟩

lemma *Ker0-imp-inj-on* :
assumes $Ker \subseteq 0$
shows *inj-on T G*
⟨*proof*⟩

lemma *inj-on-imp-Ker0* :
assumes *inj-on T G*
shows $Ker = 0$
⟨*proof*⟩

lemma *nonzero-Ker-el-imp-n-inj* :
$g \in G \Longrightarrow g \neq 0 \Longrightarrow T\ g = 0 \Longrightarrow \neg\ inj\text{-}on\ T\ G$
⟨*proof*⟩

lemma *Group-Ker* : *Group Ker*
⟨*proof*⟩

lemma *Group-Im* : *Group ImG*
⟨*proof*⟩

lemma *GroupHom-restrict0-subgroup* :
assumes *Subgroup H*
shows $GroupHom\ H\ (T \downarrow H)$
⟨*proof*⟩

lemma *im-subgroup* :
assumes *Subgroup H*
shows *Group.Subgroup ImG (T ‘ H)*
⟨*proof*⟩

lemma *GroupHom-composite-left* :
assumes $ImG \subseteq H$ *GroupHom H S*
shows $GroupHom\ G\ (S \circ T)$
⟨*proof*⟩

lemma *idhom-left* : $T\ ‘\ G \subseteq H \Longrightarrow (id{\downarrow}H) \circ T = T$
⟨*proof*⟩

end

2.3.4 Basic facts about endomorphisms

context *GroupEnd*
begin

lemmas *hom = hom*

lemma *range* : *range* $T \subseteq G$
⟨*proof*⟩

lemma *proj-decomp* :
  assumes $\bigwedge g.\ g \in G \Longrightarrow T\ (T\ g) = T\ g$
  shows $G = Ker \oplus ImG$
⟨*proof*⟩

end

2.3.5 Basic facts about isomorphisms

context *GroupIso*
begin

abbreviation $invT \equiv (the\text{-}inv\text{-}into\ G\ T) \downarrow H$

lemma *ImG* : $ImG = H$ ⟨*proof*⟩

lemma *GroupH* : *Group H* ⟨*proof*⟩

lemma *invT-onto* : $invT$ ‘ $H = G$
  ⟨*proof*⟩

lemma *inj-on-invT* : *inj-on invT H*
  ⟨*proof*⟩

lemma *bijective-invT* : *bij-betw invT H G*
  ⟨*proof*⟩

lemma *invT-into* : $h \in H \Longrightarrow invT\ h \in G$
  ⟨*proof*⟩

lemma *T-invT* : $h \in H \Longrightarrow T\ (invT\ h) = h$
  ⟨*proof*⟩

lemma *invT-eq*: $g \in G \Longrightarrow T\ g = h \Longrightarrow invT\ h = g$
  ⟨*proof*⟩

lemma *inv* : *GroupIso H invT G*
⟨*proof*⟩

end

2.3.6 Hom-sets

definition *GroupHomSet* :: $'g::group\text{-}add\ set \Rightarrow\ 'h::group\text{-}add\ set \Rightarrow ('g \Rightarrow\ 'h)\ set$
  where $GroupHomSet\ G\ H \equiv \{T.\ GroupHom\ G\ T\} \cap \{T.\ T$ ‘ $G \subseteq H\}$

lemma *GroupHomSetI* :
$GroupHom\ G\ T \Longrightarrow T\ `\ G \subseteq H \Longrightarrow T \in GroupHomSet\ G\ H$
⟨*proof*⟩

lemma *GroupHomSetD-GroupHom* :
$T \in GroupHomSet\ G\ H \Longrightarrow GroupHom\ G\ T$
⟨*proof*⟩

lemma *GroupHomSetD-Im* : $T \in GroupHomSet\ G\ H \Longrightarrow T\ `\ G \subseteq H$
⟨*proof*⟩

lemma (**in** *Group*) *Group-GroupHomSet* :
fixes $H :: 'h{::}ab\text{-}group\text{-}add\ set$
assumes $AbGroup\ H$
shows $Group\ (GroupHomSet\ G\ H)$
⟨*proof*⟩

2.4 Facts about collections of groups

lemma *listset-Group-plus-closed* :
$[\![\ \forall\ G{\in}set\ Gs.\ Group\ G;\ as \in listset\ Gs;\ bs \in listset\ Gs]\!]$
$\Longrightarrow [a{+}b.\ (a,b){\leftarrow}zip\ as\ bs] \in listset\ Gs$
⟨*proof*⟩

lemma *AbGroup-set-plus* :
assumes $AbGroup\ H\ AbGroup\ G$
shows $AbGroup\ (H + G)$
⟨*proof*⟩

lemma *AbGroup-sum-list* :
$(\forall\ G{\in}set\ Gs.\ AbGroup\ G) \Longrightarrow AbGroup\ (\sum G{\leftarrow}Gs.\ G)$
⟨*proof*⟩

lemma *AbGroup-subset-sum-list* :
$\forall\ G \in set\ Gs.\ AbGroup\ G \Longrightarrow H \in set\ Gs \Longrightarrow H \subseteq (\sum G{\leftarrow}Gs.\ G)$
⟨*proof*⟩

lemma *independent-AbGroups-pairwise-int0* :
$[\![\ \forall\ G{\in}set\ Gs.\ AbGroup\ G;\ add\text{-}independentS\ Gs;\ G \in set\ Gs;\ G' \in set\ Gs;$
$G \neq G'\ ]\!] \Longrightarrow G \cap G' = 0$
⟨*proof*⟩

lemma *independent-AbGroups-pairwise-int0-double* :
assumes $AbGroup\ G\ AbGroup\ G'\ add\text{-}independentS\ [G,G']$
shows $G \cap G' = 0$
⟨*proof*⟩

2.5 Inner direct sums of Abelian groups

2.5.1 General facts

lemma *AbGroup-inner-dirsum* :
$\forall G{\in}set\ Gs.\ AbGroup\ G \Longrightarrow AbGroup\ (\bigoplus G{\leftarrow}Gs.\ G)$
$\langle proof \rangle$

lemma *inner-dirsum-double-leftfull-imp-right0*:
assumes $Group\ A\ B \neq \{\}\ A = A \oplus B$
shows $B = 0$
$\langle proof \rangle$

lemma *AbGroup-subset-inner-dirsum* :
$[\![\ \forall G \in set\ Gs.\ AbGroup\ G;\ add\text{-}independentS\ Gs;\ H \in set\ Gs\ ]\!]$
$\Longrightarrow H \subseteq (\bigoplus G{\leftarrow}Gs.\ G)$
$\langle proof \rangle$

lemma *AbGroup-nth-subset-inner-dirsum* :
$[\![\ \forall G \in set\ Gs.\ AbGroup\ G;\ add\text{-}independentS\ Gs;\ n < length\ Gs\ ]\!]$
$\Longrightarrow Gs!n \subseteq (\bigoplus G{\leftarrow}Gs.\ G)$
$\langle proof \rangle$

lemma *AbGroup-inner-dirsum-el-decomp-ex1-double* :
assumes $AbGroup\ G\ AbGroup\ H\ add\text{-}independentS\ [G,H]\ x \in G \oplus H$
shows $\exists!gh.\ fst\ gh \in G \wedge snd\ gh \in H \wedge x = fst\ gh + snd\ gh$
$\langle proof \rangle$

lemma *AbGroup-inner-dirsum-el-decomp-ex1* :
$[\![\ \forall G \in set\ Gs.\ AbGroup\ G;\ add\text{-}independentS\ Gs\ ]\!]$
$\Longrightarrow \forall x \in (\bigoplus G{\leftarrow}Gs.\ G).\ \exists!gs{\in}listset\ Gs.\ x = sum\text{-}list\ gs$
$\langle proof \rangle$

lemma *AbGroup-inner-dirsum-pairwise-int0* :
$[\![\ \forall G \in set\ Gs.\ AbGroup\ G;\ add\text{-}independentS\ Gs;\ G \in set\ Gs;\ G' \in set\ Gs;$
$G \neq G'\ ]\!] \Longrightarrow G \cap G' = 0$
$\langle proof \rangle$

lemma *AbGroup-inner-dirsum-pairwise-int0-double* :
assumes $AbGroup\ G\ AbGroup\ G'\ add\text{-}independentS\ [G,G']$
shows $G \cap G' = 0$
$\langle proof \rangle$

2.5.2 Element decomposition and projection

definition *inner-dirsum-el-decomp* ::
$'g{::}ab\text{-}group\text{-}add\ set\ list \Rightarrow ('g \Rightarrow 'g\ list)\ (\bigoplus\text{-}{\leftarrow})$
where $\bigoplus Gs{\leftarrow} = (\lambda x.\ if\ x \in (\bigoplus G{\leftarrow}Gs.\ G)$
$then\ THE\ gs.\ gs \in listset\ Gs \wedge x = sum\text{-}list\ gs\ else\ [])$

abbreviation *inner-dirsum-el-decomp-double* ::
$'g$::*ab-group-add set* $\Rightarrow$ $'g$ *set* $\Rightarrow$ ($'g$ $\Rightarrow$ $'g$ *list*) (-$\oplus$-$\leftarrow$) **where** $G \oplus H \leftarrow \equiv \bigoplus [G,H] \leftarrow$

abbreviation *inner-dirsum-el-decomp-nth* ::
$'g$::*ab-group-add set list* $\Rightarrow$ *nat* $\Rightarrow$ ($'g \Rightarrow 'g$) ($\bigoplus$-$\downarrow$-)
where $\bigoplus Gs{\downarrow}n \equiv restrict0\ (\lambda x.\ (\bigoplus Gs{\leftarrow}x)!n)\ (\bigoplus G{\leftarrow}Gs.\ G)$

lemma *AbGroup-inner-dirsum-el-decompI* :
$[\![\ \forall G \in set\ Gs.\ AbGroup\ G;\ add\text{-}independentS\ Gs;\ x \in (\bigoplus G{\leftarrow}Gs.\ G)\ ]\!]$
$\Longrightarrow (\bigoplus Gs{\leftarrow}x) \in listset\ Gs \wedge x = sum\text{-}list\ (\bigoplus Gs{\leftarrow}x)$
⟨*proof*⟩

lemma (**in** *AbGroup*) *abSubgroup-inner-dirsum-el-decomp-set* :
$[\![\ \forall H \in set\ Hs.\ Subgroup\ H;\ add\text{-}independentS\ Hs;\ x \in (\bigoplus H{\leftarrow}Hs.\ H)\ ]\!]$
$\Longrightarrow set\ (\bigoplus Hs{\leftarrow}x) \subseteq G$
⟨*proof*⟩

lemma *AbGroup-inner-dirsum-el-decomp-eq* :
$[\![\ \forall G \in set\ Gs.\ AbGroup\ G;\ add\text{-}independentS\ Gs;\ x \in (\bigoplus G{\leftarrow}Gs.\ G);$
$gs \in listset\ Gs;\ x = sum\text{-}list\ gs\ ]\!] \Longrightarrow (\bigoplus Gs{\leftarrow}x) = gs$
⟨*proof*⟩

lemma *AbGroup-inner-dirsum-el-decomp-plus* :
assumes $\forall G \in set\ Gs.\ AbGroup\ G\ add\text{-}independentS\ Gs\ x \in (\bigoplus G{\leftarrow}Gs.\ G)$
$y \in (\bigoplus G{\leftarrow}Gs.\ G)$
shows $(\bigoplus Gs{\leftarrow}(x+y)) = [a+b.\ (a,b){\leftarrow}zip\ (\bigoplus Gs{\leftarrow}x)\ (\bigoplus Gs{\leftarrow}y)]$
⟨*proof*⟩

lemma *AbGroup-length-inner-dirsum-el-decomp* :
$[\![\ \forall G \in set\ Gs.\ AbGroup\ G;\ add\text{-}independentS\ Gs;\ x \in (\bigoplus G{\leftarrow}Gs.\ G)\ ]\!]$
$\Longrightarrow length\ (\bigoplus Gs{\leftarrow}x) = length\ Gs$
⟨*proof*⟩

lemma *AbGroup-inner-dirsum-el-decomp-in-nth* :
assumes $\forall G \in set\ Gs.\ AbGroup\ G\ add\text{-}independentS\ Gs\ n < length\ Gs$
$x \in Gs!n$
shows $(\bigoplus Gs{\leftarrow}x) = (replicate\ (length\ Gs)\ 0)[n := x]$
⟨*proof*⟩

lemma *AbGroup-inner-dirsum-el-decomp-nth-in-nth* :
$[\![\ \forall G \in set\ Gs.\ AbGroup\ G;\ add\text{-}independentS\ Gs;\ k < length\ Gs;$
$n < length\ Gs;\ x \in Gs!n\ ]\!] \Longrightarrow (\bigoplus Gs{\downarrow}k)\ x = (if\ k = n\ then\ x\ else\ 0)$
⟨*proof*⟩

lemma *AbGroup-inner-dirsum-el-decomp-nth-id-on-nth* :
$[\![\ \forall G \in set\ Gs.\ AbGroup\ G;\ add\text{-}independentS\ Gs;\ n < length\ Gs;\ x \in Gs!n\ ]\!]$
$\Longrightarrow (\bigoplus Gs{\downarrow}n)\ x = x$
⟨*proof*⟩

lemma *AbGroup-inner-dirsum-el-decomp-nth-onto-nth* :
assumes $\forall G \in set\ Gs.\ AbGroup\ G\ add\text{-}independentS\ Gs\ n < length\ Gs$
shows $(\bigoplus Gs\downarrow n)$ ‘ $(\bigoplus G\leftarrow Gs.\ G) = Gs!n$
⟨*proof*⟩

lemma *AbGroup-inner-dirsum-subset-proj-eq-0* :
assumes $Gs \neq []\ \forall G \in set\ Gs.\ AbGroup\ G\ add\text{-}independentS\ Gs$
$X \subseteq (\bigoplus G\leftarrow Gs.\ G)\ \forall i < length\ Gs.\ (\bigoplus Gs\downarrow i)$ ‘ $X = 0$
shows $X = 0$
⟨*proof*⟩

lemma *GroupEnd-inner-dirsum-el-decomp-nth* :
assumes $\forall G \in set\ Gs.\ AbGroup\ G\ add\text{-}independentS\ Gs\ n < length\ Gs$
shows $GroupEnd\ (\bigoplus G\leftarrow Gs.\ G)\ (\bigoplus Gs\downarrow n)$
⟨*proof*⟩

2.6 Rings

2.6.1 Preliminaries

lemma (**in** *ring-1*) *map-times-neg1-eq-map-uminus* : $[(-1)*r.\ r\leftarrow rs] = [-r.\ r\leftarrow rs]$
⟨*proof*⟩

2.6.2 Locale and basic facts

Define a *Ring1* to be a multiplicatively closed additive subgroup of *UNIV* for the *ring-1* class.

locale $Ring1 = Group\ R$
for $R :: 'r::ring\text{-}1\ set$
\+ **assumes** *one-closed* : $1 \in R$
and *mult-closed*: $\bigwedge r\ s.\ r \in R \Longrightarrow s \in R \Longrightarrow r * s \in R$
begin

lemma *AbGroup* : *AbGroup R*
⟨*proof*⟩

lemmas *zero-closed* = *zero-closed*
lemmas *add-closed* = *add-closed*
lemmas *neg-closed* = *neg-closed*
lemmas *diff-closed* = *diff-closed*
lemmas *zip-add-closed* = *zip-add-closed*
lemmas *sum-closed* = *AbGroup.sum-closed*[*OF AbGroup*]
lemmas *sum-list-closed* = *sum-list-closed*
lemmas *sum-list-closed-prod* = *sum-list-closed-prod*
lemmas *list-diff-closed* = *list-diff-closed*

abbreviation *Subring1* :: $'r\ set \Rightarrow bool$ **where** $Subring1\ S \equiv Ring1\ S \wedge S \subseteq R$

lemma *Subring1D1* : *Subring1 S* $\Longrightarrow$ *Ring1 S* ⟨*proof*⟩

end

lemma (**in** *ring-1*) *full-Ring1* : *Ring1 UNIV*
⟨*proof*⟩

2.7 The group ring

2.7.1 Definition and basic facts

Realize the group ring as the set of almost-every-zero functions from group to ring. One can recover the usual notion of group ring element by considering such a function to send group elements to their coefficients. Here the codomain of such functions is not restricted to some *Ring1* subset since we will not be interested in having the ability to change the ring of scalars for a group ring.

context *Group*
begin

abbreviation *group-ring* :: (*'a*::*zero*, *'g*) *aezfun set*
where *group-ring* $\equiv$ *aezfun-setspan G*

lemmas *group-ringD* = *aezfun-setspan-def*[*of G*]

lemma *RG-one-closed* : (*1*::(*'r*::*zero-neq-one*,*'g*) *aezfun*) $\in$ *group-ring*
⟨*proof*⟩

lemma *RG-zero-closed* : (*0*::(*'r*::*zero*,*'g*) *aezfun*) $\in$ *group-ring*
⟨*proof*⟩

lemma *RG-n0* : *group-ring* $\neq$ (*0*::(*'r*::*zero-neq-one*, *'g*) *aezfun set*)
⟨*proof*⟩

lemma *RG-mult-closed* :
defines *RG*: *RG* $\equiv$ *group-ring* :: (*'r*::*ring-1*, *'g*) *aezfun set*
shows $x \in RG \Longrightarrow y \in RG \Longrightarrow x * y \in RG$
⟨*proof*⟩

lemma *Ring1-RG* :
defines *RG*: *RG* $\equiv$ *group-ring* :: (*'r*::*ring-1*, *'g*) *aezfun set*
shows *Ring1 RG*
⟨*proof*⟩

lemma *RG-aezdeltafun-closed* :
defines *RG*: *RG* $\equiv$ *group-ring* :: (*'r*::*ring-1*, *'g*) *aezfun set*
assumes $g \in G$
shows $r\ \delta\delta\ g \in RG$

⟨*proof*⟩

lemma *RG-aezdelta0fun-closed* : $(r::'r::ring\text{-}1)\ \delta\delta\ 0 \in group\text{-}ring$
⟨*proof*⟩

lemma *RG-sum-list-rddg-closed* :
defines *RG*: $RG \equiv group\text{-}ring :: ('r::ring\text{-}1,\ 'g)\ aezfun\ set$
assumes $set\ (map\ snd\ rgs) \subseteq G$
shows $(\sum (r,g)\leftarrow rgs.\ r\ \delta\delta\ g) \in RG$
⟨*proof*⟩

lemmas *RG-el-decomp-aezdeltafun* = *aezfun-setspan-el-decomp-aezdeltafun*[*of - G*]

lemma *Subgroup-imp-Subring* :
fixes $H :: 'g\ set$
and $FH :: ('r::ring\text{-}1,'g)\ aezfun\ set$
and $FG :: ('r,'g)\ aezfun\ set$
defines $FH \equiv Group.group\text{-}ring\ H$
and $FG \equiv group\text{-}ring$
shows $Subgroup\ H \Longrightarrow Ring1.Subring1\ FG\ FH$
⟨*proof*⟩

end

lemma (**in** *FinGroup*) *group-ring-spanning-set* :
$\exists gs.\ distinct\ gs \wedge set\ gs = G$
$\wedge (\forall f\in (group\text{-}ring :: ('b::semiring\text{-}1,\ 'g)\ aezfun\ set).\ \exists bs.$
$length\ bs = length\ gs \wedge f = (\sum (b,g)\leftarrow zip\ bs\ gs.\ (b\ \delta\delta\ 0) * (1\ \delta\delta\ g)))$
⟨*proof*⟩

2.7.2 Projecting almost-everywhere-zero functions onto a group ring

context *Group*
begin

abbreviation $RG\text{-}proj \equiv aezfun\text{-}setspan\text{-}proj\ G$

lemmas *RG-proj-in-RG* = *aezfun-setspan-proj-in-setspan*
lemmas *RG-proj-sum-list-prod* = *aezfun-setspan-proj-sum-list-prod*[*of G*]

lemma *RG-proj-mult-leftdelta'* :
fixes $r\ s :: 'r::\{comm\text{-}monoid\text{-}add,mult\text{-}zero\}$
shows $g \in G \Longrightarrow RG\text{-}proj\ (r\ \delta\delta\ g * (s\ \delta\delta\ g')) = r\ \delta\delta\ g * RG\text{-}proj\ (s\ \delta\delta\ g')$
⟨*proof*⟩

lemma *RG-proj-mult-leftdelta* :
fixes $r :: 'r::semiring\text{-}1$
assumes $g \in G$

shows $RG\text{-}proj\ ((r\ \delta\delta\ g) * x) = r\ \delta\delta\ g * RG\text{-}proj\ x$
⟨*proof*⟩

lemma *RG-proj-mult-rightdelta′* :
fixes $r\ s :: {'}r{::}\{comm\text{-}monoid\text{-}add, mult\text{-}zero\}$
assumes $g' \in G$
shows $RG\text{-}proj\ (r\ \delta\delta\ g * (s\ \delta\delta\ g')) = RG\text{-}proj\ (r\ \delta\delta\ g) * (s\ \delta\delta\ g')$
⟨*proof*⟩

lemma *RG-proj-mult-rightdelta* :
fixes $r :: {'}r{::}semiring\text{-}1$
assumes $g \in G$
shows $RG\text{-}proj\ (x * (r\ \delta\delta\ g)) = (RG\text{-}proj\ x) * (r\ \delta\delta\ g)$
⟨*proof*⟩

lemma *RG-proj-mult-right* :
$x \in (group\text{-}ring :: ({'}r{::}ring\text{-}1,\ {'}g)\ aezfun\ set)$
$\Longrightarrow RG\text{-}proj\ (y * x) = RG\text{-}proj\ y * x$
⟨*proof*⟩

end

3 Modules

3.1 Locales and basic facts

3.1.1 Locales

locale *scalar-mult* =
fixes $smult :: {'}r{::}ring\text{-}1 \Rightarrow {'}m{::}ab\text{-}group\text{-}add \Rightarrow {'}m$ (**infixr** · *70*)

locale *R-scalar-mult* = *scalar-mult smult* + *Ring1 R*
for $R :: {'}r{::}ring\text{-}1\ set$
and $smult :: {'}r \Rightarrow {'}m{::}ab\text{-}group\text{-}add \Rightarrow {'}m$ (**infixr** · *70*)

lemma (**in** *scalar-mult*) *R-scalar-mult* : *R-scalar-mult UNIV*
⟨*proof*⟩

lemma (**in** *R-scalar-mult*) *Ring1* : *Ring1 R* ⟨*proof*⟩

locale *RModule* = *R-scalars?*: *R-scalar-mult R smult* + *VecGroup?*: *Group M*
for $R :: {'}r{::}ring\text{-}1\ set$
and $smult :: {'}r \Rightarrow {'}m{::}ab\text{-}group\text{-}add \Rightarrow {'}m$ (**infixr** · *70*)
and $M :: {'}m\ set$
\+ **assumes** *smult-closed* : $[\![\ r \in R;\ m \in M\ ]\!] \Longrightarrow r \cdot m \in M$
and *smult-distrib-left* [*simp*] : $[\![\ r \in R;\ m \in M;\ n \in M\ ]\!]$
$\Longrightarrow r \cdot (m + n) = r \cdot m + r \cdot n$
and *smult-distrib-right* [*simp*] : $[\![\ r \in R;\ s \in R;\ m \in M\ ]\!]$
$\Longrightarrow (r + s) \cdot m = r \cdot m + s \cdot m$

and *smult-assoc* [*simp*] : $\llbracket\ r \in R;\ s \in R;\ m \in M\ \rrbracket$
$\Longrightarrow r \cdot s \cdot m = (r * s) \cdot m$
and *one-smult* [*simp*] : $m \in M \Longrightarrow 1 \cdot m = m$

lemmas *RModuleI* = *RModule.intro*[*OF R-scalar-mult.intro*]

locale *Module* = *RModule UNIV smult M*
for *smult* :: *'r::ring-1* ⇒ *'m::ab-group-add* ⇒ *'m* (**infixr** · *70*)
and *M* :: *'m set*

lemmas *ModuleI* = *RModuleI*[*of UNIV*, *OF full-Ring1*, *THEN Module.intro*]

3.1.2 Basic facts

lemma *trivial-RModule* :
fixes *smult* :: *'r::ring-1* ⇒ *'m::ab-group-add* ⇒ *'m* (**infixr** · *70*)
assumes *Ring1 R* ∀ *r*∈*R*. *smult r* (*0::'m::ab-group-add*) = *0*
shows *RModule R smult* (*0::'m set*)
⟨*proof*⟩

context *RModule*
begin

abbreviation *RSubmodule* :: *'m set* ⇒ *bool*
where *RSubmodule N* ≡ *RModule R smult N* ∧ *N* ⊆ *M*

lemma *Group* : *Group M*
⟨*proof*⟩

lemma *Subgroup-RSubmodule* : *RSubmodule N* ⟹ *Subgroup N*
⟨*proof*⟩

lemma *AbGroup* : *AbGroup M*
⟨*proof*⟩

lemmas *zero-closed* = *zero-closed*
lemmas *diff-closed* = *diff-closed*
lemmas *set-plus-closed* = *set-plus-closed*
lemmas *sum-closed* = *AbGroup.sum-closed*[*OF AbGroup*]

lemma *map-smult-closed* :
$r \in R \Longrightarrow$ *set ms* ⊆ *M* ⟹ *set* (*map* ((·) *r*) *ms*) ⊆ *M*
⟨*proof*⟩

lemma *zero-smult* : $m \in M \Longrightarrow 0 \cdot m = 0$
⟨*proof*⟩

lemma *smult-zero* : $r \in R \Longrightarrow r \cdot 0 = 0$
⟨*proof*⟩

lemma *neg-smult* : $r \in R \Longrightarrow m \in M \Longrightarrow (-r) \cdot m = - (r \cdot m)$
⟨*proof*⟩

lemma *neg-eq-neg1-smult* : $m \in M \Longrightarrow (-1) \cdot m = - m$
⟨*proof*⟩

lemma *smult-neg* : $r \in R \Longrightarrow m \in M \Longrightarrow r \cdot (- m) = - (r \cdot m)$
⟨*proof*⟩

lemma *smult-distrib-left-diff* :
$[\![r \in R; m \in M; n \in M]\!] \Longrightarrow r \cdot (m - n) = r \cdot m - r \cdot n$
⟨*proof*⟩

lemma *smult-distrib-right-diff* :
$[\![r \in R; s \in R; m \in M]\!] \Longrightarrow (r - s) \cdot m = r \cdot m - s \cdot m$
⟨*proof*⟩

lemma *smult-sum-distrib* :
assumes $r \in R$
shows $\mathit{finite}\ A \Longrightarrow f\ `\ A \subseteq M \Longrightarrow r \cdot (\sum a{\in}A.\ f\ a) = (\sum a{\in}A.\ r \cdot f\ a)$
⟨*proof*⟩

lemma *sum-smult-distrib* :
assumes $m \in M$
shows $\mathit{finite}\ A \Longrightarrow f\ `\ A \subseteq R \Longrightarrow (\sum a{\in}A.\ f\ a) \cdot m = (\sum a{\in}A.\ (f\ a) \cdot m)$
⟨*proof*⟩

lemma *smult-sum-list-distrib* :
$r \in R \Longrightarrow \mathit{set}\ \mathit{ms} \subseteq M \Longrightarrow r \cdot (\mathit{sum\text{-}list}\ \mathit{ms}) = (\sum m{\leftarrow}\mathit{ms}.\ r \cdot m)$
⟨*proof*⟩

lemma *sum-list-prod-map-smult-distrib* :
$m \in M \Longrightarrow \mathit{set}\ (\mathit{map}\ (\mathit{case\text{-}prod}\ f)\ \mathit{xys}) \subseteq R$
$\Longrightarrow (\sum (x,y){\leftarrow}\mathit{xys}.\ f\ x\ y) \cdot m = (\sum (x,y){\leftarrow}\mathit{xys}.\ f\ x\ y \cdot m)$
⟨*proof*⟩

lemma *RSubmoduleI* :
assumes $\mathit{Subgroup}\ N\ \bigwedge r\ n.\ r \in R \Longrightarrow n \in N \Longrightarrow r \cdot n \in N$
shows *RSubmodule N*
⟨*proof*⟩

end

lemma (**in** *R-scalar-mult*) *listset-RModule-Rsmult-closed* :
$[\![\forall M{\in}\mathit{set}\ \mathit{Ms}.\ \mathit{RModule}\ R\ \mathit{smult}\ M; r \in R; \mathit{ms} \in \mathit{listset}\ \mathit{Ms}]\!]$
$\Longrightarrow [r \cdot m.\ m{\leftarrow}\mathit{ms}] \in \mathit{listset}\ \mathit{Ms}$
⟨*proof*⟩

context *Module*
begin

abbreviation *Submodule* :: *'m set* $\Rightarrow$ *bool*
  where *Submodule* $\equiv$ *RModule.RSubmodule UNIV smult M*

lemmas *AbGroup* = *AbGroup*
lemmas *SubmoduleI* = *RSubmoduleI*

end

3.1.3 Module and submodule instances

lemma (**in** *R-scalar-mult*) *trivial-RModule* :
  $(\bigwedge r.\ r \in R \Longrightarrow r \cdot 0 = 0) \Longrightarrow$ *RModule R smult 0*
  ⟨*proof*⟩

context *RModule*
begin

lemma *trivial-RSubmodule* : *RSubmodule 0*
  ⟨*proof*⟩

lemma *RSubmodule-set-plus* :
  assumes *RSubmodule L RSubmodule N*
  shows *RSubmodule* $(L + N)$
⟨*proof*⟩

lemma *RSubmodule-sum-list* :
  $(\forall N \in set\ Ns.\ RSubmodule\ N) \Longrightarrow RSubmodule\ (\sum N \leftarrow Ns.\ N)$
  ⟨*proof*⟩

lemma *RSubmodule-inner-dirsum* :
  assumes $(\forall N \in set\ Ns.\ RSubmodule\ N)$
  shows $RSubmodule\ (\bigoplus N \leftarrow Ns.\ N)$
⟨*proof*⟩

lemma *RModule-inner-dirsum* :
  $(\forall N \in set\ Ns.\ RSubmodule\ N) \Longrightarrow RModule\ R\ smult\ (\bigoplus N \leftarrow Ns.\ N)$
  ⟨*proof*⟩

lemma *SModule-restrict-scalars* :
  assumes *Subring1 S*
  shows *RModule S smult M*
⟨*proof*⟩

end

3.2 Linear algebra in modules

3.2.1 Linear combinations: *lincomb*

context *scalar-mult*
begin

definition *lincomb* :: *′r list* ⇒ *′m list* ⇒ *′m* (**infix** ·· *70*)
where $rs \cdot\cdot\ ms = (\sum (r,m) \leftarrow zip\ rs\ ms.\ r \cdot m)$

Note: *zip* will truncate if lengths of coefficient and vector lists differ.

lemma *lincomb-Nil* : $rs = [] \vee ms = [] \Longrightarrow rs \cdot\cdot\ ms = 0$
⟨*proof*⟩

lemma *lincomb-singles* : $[a] \cdot\cdot\ [m] = a \cdot m$
⟨*proof*⟩

lemma *lincomb-Cons* : $(r\ \#\ rs) \cdot\cdot\ (m\ \#\ ms) = r \cdot m + rs \cdot\cdot\ ms$
⟨*proof*⟩

lemma *lincomb-append* :
$length\ rs = length\ ms \Longrightarrow (rs@ss) \cdot\cdot\ (ms@ns) = rs \cdot\cdot\ ms + ss \cdot\cdot\ ns$
⟨*proof*⟩

lemma *lincomb-append-left* :
$(rs\ @\ ss) \cdot\cdot\ ms = rs \cdot\cdot\ ms + ss \cdot\cdot\ drop\ (length\ rs)\ ms$
⟨*proof*⟩

lemma *lincomb-append-right* :
$rs \cdot\cdot\ (ms@ns) = rs \cdot\cdot\ ms + (drop\ (length\ ms)\ rs) \cdot\cdot\ ns$
⟨*proof*⟩

lemma *lincomb-conv-take-right* : $rs \cdot\cdot\ ms = rs \cdot\cdot\ take\ (length\ rs)\ ms$
⟨*proof*⟩

end

context *RModule*
begin

lemmas *lincomb-Nil* = *lincomb-Nil*
lemmas *lincomb-Cons* = *lincomb-Cons*

lemma *lincomb-closed* : $set\ rs \subseteq R \Longrightarrow set\ ms \subseteq M \Longrightarrow rs \cdot\cdot\ ms \in M$
⟨*proof*⟩

lemma *smult-lincomb* :
$[\![\ set\ rs \subseteq R;\ s \in R;\ set\ ms \subseteq M\ ]\!] \Longrightarrow s \cdot (rs \cdot\cdot\ ms) = [s*r.\ r \leftarrow rs] \cdot\cdot\ ms$
⟨*proof*⟩

lemma *neg-lincomb* :
$set\ rs \subseteq R \Longrightarrow set\ ms \subseteq M \Longrightarrow -\ (rs \cdot\cdot\ ms) = [-r.\ r \leftarrow rs] \cdot\cdot\ ms$
$\langle proof \rangle$

lemma *lincomb-sum-left* :
$[\![\ set\ rs \subseteq R;\ set\ ss \subseteq R;\ set\ ms \subseteq M;\ length\ rs \leq length\ ss\ ]\!]$
$\Longrightarrow [r + s.\ (r,s) \leftarrow zip\ rs\ ss] \cdot\cdot\ ms = rs \cdot\cdot\ ms + (take\ (length\ rs)\ ss) \cdot\cdot\ ms$
$\langle proof \rangle$

lemma *lincomb-sum* :
assumes $set\ rs \subseteq R\ set\ ss \subseteq R\ set\ ms \subseteq M\ length\ rs \leq length\ ss$
shows $rs \cdot\cdot\ ms + ss \cdot\cdot\ ms$
$= ([a + b.\ (a,b) \leftarrow zip\ rs\ ss]\ @\ (drop\ (length\ rs)\ ss)) \cdot\cdot\ ms$
$\langle proof \rangle$

lemma *lincomb-diff-left* :
$[\![\ set\ rs \subseteq R;\ set\ ss \subseteq R;\ set\ ms \subseteq M;\ length\ rs \leq length\ ss\ ]\!]$
$\Longrightarrow [r - s.\ (r,s) \leftarrow zip\ rs\ ss] \cdot\cdot\ ms = rs \cdot\cdot\ ms - (take\ (length\ rs)\ ss) \cdot\cdot\ ms$
$\langle proof \rangle$

lemma *lincomb-replicate-left* :
$r \in R \Longrightarrow set\ ms \subseteq M \Longrightarrow (replicate\ k\ r) \cdot\cdot\ ms = r \cdot (\ \sum m \leftarrow (take\ k\ ms).\ m\ )$
$\langle proof \rangle$

lemma *lincomb-replicate0-left* : $set\ ms \subseteq M \Longrightarrow (replicate\ k\ 0) \cdot\cdot\ ms = 0$
$\langle proof \rangle$

lemma *lincomb-0coeffs* : $set\ ms \subseteq M \Longrightarrow \forall s \in set\ rs.\ s = 0 \Longrightarrow rs \cdot\cdot\ ms = 0$
$\langle proof \rangle$

lemma *delta-scalars-lincomb-eq-nth* :
$set\ ms \subseteq M \Longrightarrow n < length\ ms$
$\Longrightarrow ((replicate\ (length\ ms)\ 0)[n := 1]) \cdot\cdot\ ms = ms!n$
$\langle proof \rangle$

lemma *lincomb-obtain-same-length-Rcoeffs* :
$set\ rs \subseteq R \Longrightarrow set\ ms \subseteq M$
$\Longrightarrow \exists ss.\ set\ ss \subseteq R \wedge length\ ss = length\ ms$
$\wedge\ take\ (length\ rs)\ ss = take\ (length\ ms)\ rs \wedge rs \cdot\cdot\ ms = ss \cdot\cdot\ ms$
$\langle proof \rangle$

lemma *lincomb-concat* :
$list\text{-}all2\ (\lambda rs\ ms.\ length\ rs = length\ ms)\ rss\ mss$
$\Longrightarrow (concat\ rss) \cdot\cdot\ (concat\ mss) = (\sum (rs,ms) \leftarrow zip\ rss\ mss.\ rs \cdot\cdot\ ms)$
$\langle proof \rangle$

lemma *lincomb-snoc0* : $set\ ms \subseteq M \Longrightarrow (as@[0]) \cdot\cdot\ ms = as \cdot\cdot\ ms$
$\langle proof \rangle$

lemma *lincomb-strip-while-0coeffs* :
assumes *set ms* $\subseteq$ *M*
shows (*strip-while* ((=) *0*) *as*) $\cdot\cdot$ *ms* = *as* $\cdot\cdot$ *ms*
⟨*proof*⟩

end

lemmas (**in** *Module*) *lincomb-obtain-same-length-coeffs* = *lincomb-obtain-same-length-Rcoeffs*
lemmas (**in** *Module*) *lincomb-concat* = *lincomb-concat*

3.2.2 Spanning: *RSpan* and *Span*

context *R-scalar-mult*
begin

primrec *RSpan* :: *'m list* ⇒ *'m set*
where *RSpan* [] = *0*
| *RSpan* (*m*#*ms*) = { *r* · *m* | *r*. *r* ∈ *R* } + *RSpan ms*

lemma *RSpan-single* : *RSpan* [*m*] = { *r* · *m* | *r*. *r* ∈ *R* }
⟨*proof*⟩

lemma *RSpan-Cons* : *RSpan* (*m*#*ms*) = *RSpan* [*m*] + *RSpan ms*
⟨*proof*⟩

lemma *in-RSpan-obtain-same-length-coeffs* :
n ∈ *RSpan ms* ⟹ ∃ *rs*. *set rs* ⊆ *R* ∧ *length rs* = *length ms* ∧ *n* = *rs* $\cdot\cdot$ *ms*
⟨*proof*⟩

lemma *in-RSpan-Cons-obtain-same-length-coeffs* :
n ∈ *RSpan* (*m*#*ms*) ⟹ ∃ *r rs*. *set* (*r*#*rs*) ⊆ *R* ∧ *length rs* = *length ms*
∧ *n* = *r* · *m* + *rs* $\cdot\cdot$ *ms*
⟨*proof*⟩

lemma *RSpanD-lincomb* :
RSpan ms = { *rs* $\cdot\cdot$ *ms* | *rs*. *set rs* ⊆ *R* ∧ *length rs* = *length ms*}
⟨*proof*⟩

lemma *RSpan-append* : *RSpan* (*ms* @ *ns*) = *RSpan ms* + *RSpan ns*
⟨*proof*⟩

end

context *scalar-mult*
begin

abbreviation *Span* ≡ *R-scalar-mult.RSpan UNIV smult*

lemmas *Span-append* = *R-scalar-mult.RSpan-append*[*OF R-scalar-mult*, *of smult*]

lemmas *SpanD-lincomb*
 $= R\text{-}scalar\text{-}mult.RSpanD\text{-}lincomb$ [*OF R-scalar-mult, of smult*]

lemmas *in-Span-obtain-same-length-coeffs*
 $= R\text{-}scalar\text{-}mult.in\text{-}RSpan\text{-}obtain\text{-}same\text{-}length\text{-}coeffs$[
 OF R-scalar-mult, of - smult
 ]

end

context *RModule*
begin

lemma *RSpan-contains-spanset-single* : $m \in M \Longrightarrow m \in RSpan\ [m]$
 ⟨*proof*⟩

lemma *RSpan-single-nonzero* : $m \in M \Longrightarrow m \neq 0 \Longrightarrow RSpan\ [m] \neq 0$
 ⟨*proof*⟩

lemma *Group-RSpan-single* :
 assumes $m \in M$
 shows $Group\ (RSpan\ [m])$
⟨*proof*⟩

lemma *Group-RSpan* : $set\ ms \subseteq M \Longrightarrow Group\ (RSpan\ ms)$
⟨*proof*⟩

lemma *RSpanD-lincomb-arb-len-coeffs* :
 $set\ ms \subseteq M \Longrightarrow RSpan\ ms = \{\ rs \cdot\cdot\ ms \mid rs.\ set\ rs \subseteq R\ \}$
⟨*proof*⟩

lemma *RSpanI-lincomb-arb-len-coeffs* :
 $set\ rs \subseteq R \Longrightarrow set\ ms \subseteq M \Longrightarrow rs \cdot\cdot\ ms \in RSpan\ ms$
 ⟨*proof*⟩

lemma *RSpan-contains-RSpans-Cons-left* :
 $set\ ms \subseteq M \Longrightarrow RSpan\ [m] \subseteq RSpan\ (m\#ms)$
 ⟨*proof*⟩

lemma *RSpan-contains-RSpans-Cons-right* :
 $m \in M \Longrightarrow RSpan\ ms \subseteq RSpan\ (m\#ms)$
 ⟨*proof*⟩

lemma *RSpan-contains-RSpans-append-left* :
 $set\ ns \subseteq M \Longrightarrow RSpan\ ms \subseteq RSpan\ (ms@ns)$
 ⟨*proof*⟩

lemma *RSpan-contains-spanset* : $set\ ms \subseteq M \Longrightarrow set\ ms \subseteq RSpan\ ms$
⟨*proof*⟩

lemma *RSpan-contains-spanset-append-left* :
  $set\ ms \subseteq M \Longrightarrow set\ ns \subseteq M \Longrightarrow set\ ms \subseteq RSpan\ (ms@ns)$
  ⟨*proof*⟩

lemma *RSpan-contains-spanset-append-right* :
  $set\ ms \subseteq M \Longrightarrow set\ ns \subseteq M \Longrightarrow set\ ns \subseteq RSpan\ (ms@ns)$
  ⟨*proof*⟩

lemma *RSpan-zero-closed* : $set\ ms \subseteq M \Longrightarrow 0 \in RSpan\ ms$
  ⟨*proof*⟩

lemma *RSpan-single-closed* : $m \in M \Longrightarrow RSpan\ [m] \subseteq M$
  ⟨*proof*⟩

lemma *RSpan-closed* : $set\ ms \subseteq M \Longrightarrow RSpan\ ms \subseteq M$
⟨*proof*⟩

lemma *RSpan-smult-closed* :
  assumes $r \in R$ $set\ ms \subseteq M$ $n \in RSpan\ ms$
  shows $r \cdot n \in RSpan\ ms$
⟨*proof*⟩

lemma *RSpan-add-closed* :
  assumes $set\ ms \subseteq M$ $n \in RSpan\ ms$ $n' \in RSpan\ ms$
  shows $n + n' \in RSpan\ ms$
⟨*proof*⟩

lemma *RSpan-lincomb-closed* :
  $[\![\ set\ rs \subseteq R;\ set\ ms \subseteq M;\ set\ ns \subseteq RSpan\ ms\ ]\!] \Longrightarrow rs \cdot\cdot\ ns \in RSpan\ ms$
  ⟨*proof*⟩

lemma *RSpanI* : $set\ ms \subseteq M \Longrightarrow M \subseteq RSpan\ ms \Longrightarrow M = RSpan\ ms$
  ⟨*proof*⟩

lemma *RSpan-contains-RSpan-take* :
  $set\ ms \subseteq M \Longrightarrow RSpan\ (take\ k\ ms) \subseteq RSpan\ ms$
  ⟨*proof*⟩

lemma *RSubmodule-RSpan-single* :
  assumes $m \in M$
  shows $RSubmodule\ (RSpan\ [m])$
⟨*proof*⟩

lemma *RSubmodule-RSpan* : $set\ ms \subseteq M \Longrightarrow RSubmodule\ (RSpan\ ms)$
⟨*proof*⟩

lemma *RSpan-RSpan-closed* :
  $set\ ms \subseteq M \Longrightarrow set\ ns \subseteq RSpan\ ms \Longrightarrow RSpan\ ns \subseteq RSpan\ ms$

⟨*proof*⟩

lemma *spanset-reduce-Cons* :
  $set\ ms \subseteq M \Longrightarrow m \in RSpan\ ms \Longrightarrow RSpan\ (m\#ms) = RSpan\ ms$
  ⟨*proof*⟩

lemma *RSpan-replace-hd* :
  assumes $n \in M$ $set\ ms \subseteq M$ $m \in RSpan\ (n\ \#\ ms)$
  shows $RSpan\ (m\ \#\ ms) \subseteq RSpan\ (n\ \#\ ms)$
⟨*proof*⟩

end

lemmas (**in** *scalar-mult*)
  Span-Cons = *R-scalar-mult.RSpan-Cons*[*OF R-scalar-mult*, *of smult*]

context *Module*
begin

lemmas *SpanD-lincomb-arb-len-coeffs* = *RSpanD-lincomb-arb-len-coeffs*
lemmas *SpanI* = *RSpanI*
lemmas *SpanI-lincomb-arb-len-coeffs* = *RSpanI-lincomb-arb-len-coeffs*
lemmas *Span-contains-Spans-Cons-right* = *RSpan-contains-RSpans-Cons-right*
lemmas *Span-contains-spanset* = *RSpan-contains-spanset*
lemmas *Span-contains-spanset-append-left* = *RSpan-contains-spanset-append-left*
lemmas *Span-contains-spanset-append-right* = *RSpan-contains-spanset-append-right*
lemmas *Span-closed* = *RSpan-closed*
lemmas *Span-smult-closed* = *RSpan-smult-closed*
lemmas *Span-contains-Span-take* = *RSpan-contains-RSpan-take*
lemmas *Span-replace-hd* = *RSpan-replace-hd*
lemmas *Submodule-Span* = *RSubmodule-RSpan*

end

3.2.3 Finitely generated modules

context *R-scalar-mult*
begin

abbreviation *R-fingen M* $\equiv (\exists\, ms.\ set\ ms \subseteq M \wedge RSpan\ ms = M)$

Similar to definition of *card* for finite sets, we default *dim* to 0 if no finite spanning set exists. Note that $RSpan\ [] = 0$ implies that $dim\text{-}R\ \{0{::}'b\} = (0{::}'a)$.

definition *dim-R* :: $'m\ set \Rightarrow nat$
  where $dim\text{-}R\ M = (if\ R\text{-}fingen\ M\ then\ ($
      $LEAST\ n.\ \exists\, ms.\ length\ ms = n \wedge set\ ms \subseteq M \wedge RSpan\ ms = M$
    $)\ else\ 0)$

lemma *dim-R-nonzero* :
assumes *dim-R M* $>$ *0*
shows $M \neq 0$
⟨*proof*⟩

end

hide-const *real-vector.dim*
hide-const (**open**) *Real-Vector-Spaces.dim*

abbreviation (**in** *scalar-mult*) *fingen* $\equiv$ *R-scalar-mult.R-fingen UNIV smult*
abbreviation (**in** *scalar-mult*) *dim* $\equiv$ *R-scalar-mult.dim-R UNIV smult*

lemmas (**in** *Module*) *dim-nonzero* = *dim-R-nonzero*

3.2.4 R-linear independence

context *R-scalar-mult*
begin

primrec *R-lin-independent* :: *'m list* $\Rightarrow$ *bool* **where**
R-lin-independent-Nil: *R-lin-independent* [] = *True* |
R-lin-independent-Cons:
R-lin-independent (*m#ms*) = (*R-lin-independent ms*
$\wedge$ ($\forall$ *r rs*. (*set* (*r#rs*) $\subseteq$ *R* $\wedge$ (*r#rs*) ·· (*m#ms*) = *0*) $\longrightarrow$ *r* = *0*))

lemma *R-lin-independent-ConsI* :
assumes *R-lin-independent ms*
$\bigwedge$*r rs*. *set* (*r#rs*) $\subseteq$ *R* $\Longrightarrow$ (*r#rs*) ·· (*m#ms*) = *0* $\Longrightarrow$ *r* = *0*
shows *R-lin-independent* (*m#ms*)
⟨*proof*⟩

lemma *R-lin-independent-ConsD1* :
R-lin-independent (*m#ms*) $\Longrightarrow$ *R-lin-independent ms*
⟨*proof*⟩

lemma *R-lin-independent-ConsD2* :
⟦ *R-lin-independent* (*m#ms*); *set* (*r#rs*) $\subseteq$ *R*; (*r#rs*) ·· (*m#ms*) = *0* ⟧
$\Longrightarrow$ *r* = *0*
⟨*proof*⟩

end

context *RModule*
begin

lemma *R-lin-independent-imp-same-scalars* :
⟦ *length rs* = *length ss*; *length rs* $\leq$ *length ms*; *set rs* $\subseteq$ *R*; *set ss* $\subseteq$ *R*;

$set\ ms \subseteq M;\ R\text{-}lin\text{-}independent\ ms;\ rs \cdot\cdot\ ms = ss \cdot\cdot\ ms\ ]\!] \Longrightarrow rs = ss$
⟨proof⟩

lemma *R-lin-independent-obtain-unique-scalars* :
$[\![\ set\ ms \subseteq M;\ R\text{-}lin\text{-}independent\ ms;\ n \in RSpan\ ms\ ]\!]$
$\Longrightarrow (\exists!\ rs.\ set\ rs \subseteq R \wedge length\ rs = length\ ms \wedge n = rs \cdot\cdot\ ms)$
⟨proof⟩

lemma *R-lin-independentI-all-scalars* :
$set\ ms \subseteq M \Longrightarrow$
$(\forall rs.\ set\ rs \subseteq R \wedge length\ rs = length\ ms \wedge rs \cdot\cdot\ ms = 0 \longrightarrow set\ rs \subseteq 0)$
$\Longrightarrow R\text{-}lin\text{-}independent\ ms$
⟨proof⟩

lemma *R-lin-independentI-concat-all-scalars* :
defines $eq\text{-}len$: $eq\text{-}len \equiv \lambda xs\ ys.\ length\ xs = length\ ys$
assumes $set\ (concat\ mss) \subseteq M$
$\bigwedge rss.\ set\ (concat\ rss) \subseteq R \Longrightarrow list\text{-}all2\ eq\text{-}len\ rss\ mss$
$\Longrightarrow (concat\ rss) \cdot\cdot\ (concat\ mss) = 0 \Longrightarrow (\forall rs \in set\ rss.\ set\ rs \subseteq 0)$
shows $R\text{-}lin\text{-}independent\ (concat\ mss)$
⟨proof⟩

lemma *R-lin-independentD-all-scalars* :
$[\![\ set\ rs \subseteq R;\ set\ ms \subseteq M;\ length\ rs \le length\ ms;\ R\text{-}lin\text{-}independent\ ms;$
$rs \cdot\cdot\ ms = 0\ ]\!] \Longrightarrow set\ rs \subseteq 0$
⟨proof⟩

lemma *R-lin-independentD-all-scalars-nth* :
assumes $set\ rs \subseteq R\ set\ ms \subseteq M\ R\text{-}lin\text{-}independent\ ms\ rs \cdot\cdot\ ms = 0$
$k < min\ (length\ rs)\ (length\ ms)$
shows $rs!k = 0$
⟨proof⟩

lemma *R-lin-dependent-dependence-relation* :
$set\ ms \subseteq M \Longrightarrow \neg\ R\text{-}lin\text{-}independent\ ms$
$\Longrightarrow \exists rs.\ set\ rs \subseteq R \wedge set\ rs \neq 0 \wedge length\ rs = length\ ms \wedge rs \cdot\cdot\ ms = 0$
⟨proof⟩

lemma *R-lin-independent-imp-distinct* :
$set\ ms \subseteq M \Longrightarrow R\text{-}lin\text{-}independent\ ms \Longrightarrow distinct\ ms$
⟨proof⟩

lemma *R-lin-independent-imp-independent-take* :
$set\ ms \subseteq M \Longrightarrow R\text{-}lin\text{-}independent\ ms \Longrightarrow R\text{-}lin\text{-}independent\ (take\ n\ ms)$
⟨proof⟩

lemma *R-lin-independent-Cons-imp-independent-RSpans* :
assumes $m \in M\ R\text{-}lin\text{-}independent\ (m\#ms)$
shows $add\text{-}independentS\ [RSpan\ [m],\ RSpan\ ms]$

⟨*proof*⟩

lemma *hd0-imp-R-lin-dependent* : ¬ *R-lin-independent* (*0#ms*)
⟨*proof*⟩

lemma *R-lin-independent-imp-hd-n0* : *R-lin-independent* (*m#ms*) $\Longrightarrow$ *m* ≠ *0*
⟨*proof*⟩

lemma *R-lin-independent-imp-hd-independent-from-RSpan* :
assumes *m* ∈ *M* *set ms* ⊆ *M* *R-lin-independent* (*m#ms*)
shows *m* ∉ *RSpan ms*
⟨*proof*⟩

lemma *R-lin-independent-reduce* :
assumes *n* ∈ *M*
shows *set ms* ⊆ *M* $\Longrightarrow$ *R-lin-independent* (*ms* @ *n* # *ns*)
$\Longrightarrow$ *R-lin-independent* (*ms* @ *ns*)
⟨*proof*⟩

lemma *R-lin-independent-vs-lincomb0* :
assumes *set* (*ms@n#ns*) ⊆ *M* *R-lin-independent* (*ms* @ *n* # *ns*)
set (*rs@s#ss*) ⊆ *R* *length rs* = *length ms*
(*rs@s#ss*) ·· (*ms@n#ns*) = *0*
shows *s* = *0*
⟨*proof*⟩

lemma *R-lin-independent-append-imp-independent-RSpans* :
set ms ⊆ *M* $\Longrightarrow$ *R-lin-independent* (*ms@ns*)
$\Longrightarrow$ *add-independentS* [*RSpan ms*, *RSpan ns*]
⟨*proof*⟩

end

3.2.5 Linear independence over *UNIV*

context *scalar-mult*
begin

abbreviation *lin-independent ms*
≡ *R-scalar-mult.R-lin-independent UNIV smult ms*

lemmas *lin-independent-ConsI*
= *R-scalar-mult.R-lin-independent-ConsI* [*OF R-scalar-mult*, *of smult*]
lemmas *lin-independent-ConsD1*
= *R-scalar-mult.R-lin-independent-ConsD1*[*OF R-scalar-mult*, *of smult*]

end

context *Module*

begin

lemmas *lin-independent-imp-independent-take* = *R-lin-independent-imp-independent-take*
lemmas *lin-independent-reduce* = *R-lin-independent-reduce*
lemmas *lin-independent-vs-lincomb0* = *R-lin-independent-vs-lincomb0*
lemmas *lin-dependent-dependence-relation* = *R-lin-dependent-dependence-relation*
lemmas *lin-independent-imp-distinct* = *R-lin-independent-imp-distinct*

lemmas *lin-independent-imp-hd-independent-from-Span*
= *R-lin-independent-imp-hd-independent-from-RSpan*
lemmas *lin-independent-append-imp-independent-Spans*
= *R-lin-independent-append-imp-independent-RSpans*

end

3.2.6 Rank

context *R-scalar-mult*
begin

definition *R-finrank* :: $'m$ *set* $\Rightarrow$ *bool*
where *R-finrank M* = ($\exists\, n.\ \forall\, ms.$ *set ms* $\subseteq$ *M*
$\wedge$ *R-lin-independent ms* $\longrightarrow$ *length ms* $\leq$ *n*)

lemma *R-finrankI* :
($\bigwedge ms.$ *set ms* $\subseteq$ *M* $\Longrightarrow$ *R-lin-independent ms* $\Longrightarrow$ *length ms* $\leq$ *n*)
$\Longrightarrow$ *R-finrank M*
⟨*proof*⟩

lemma *R-finrankD* :
R-finrank M $\Longrightarrow \exists\, n.\ \forall\, ms.$ *set ms* $\subseteq$ *M* $\wedge$ *R-lin-independent ms*
$\longrightarrow$ *length ms* $\leq$ *n*
⟨*proof*⟩

lemma *submodule-R-finrank* : *R-finrank M* $\Longrightarrow$ *N* $\subseteq$ *M* $\Longrightarrow$ *R-finrank N*
⟨*proof*⟩

end

context *scalar-mult*
begin

abbreviation *finrank* :: $'m$ *set* $\Rightarrow$ *bool*
where *finrank* $\equiv$ *R-scalar-mult.R-finrank UNIV smult*

lemmas *finrankI* = *R-scalar-mult.R-finrankI*[*OF R-scalar-mult*, *of - smult*]
lemmas *finrankD* = *R-scalar-mult.R-finrankD*[*OF R-scalar-mult*, *of smult*]
lemmas *submodule-finrank*
= *R-scalar-mult.submodule-R-finrank* [*OF R-scalar-mult*, *of smult*]

end

3.3 Module homomorphisms

3.3.1 Locales

locale *RModuleHom* = *Domain?*: *RModule R smult M*
+ *Codomain?*: *scalar-mult smult'*
+ *GroupHom?*: *GroupHom M T*
  for *R* :: *'r::ring-1 set*
  and *smult* :: $'r \Rightarrow 'm$::*ab-group-add* $\Rightarrow 'm$ (**infixr** $\cdot$ *70*)
  and *M* :: *'m set*
  and *smult'* :: $'r \Rightarrow 'n$::*ab-group-add* $\Rightarrow 'n$ (**infixr** $\star$ *70*)
  and *T* :: $'m \Rightarrow 'n$
+ **assumes** *R-map*: $\bigwedge r\ m.\ r \in R \Longrightarrow m \in M \Longrightarrow T\ (r \cdot m) = r \star T\ m$

abbreviation (**in** *RModuleHom*) *lincomb'* :: $'r$ *list* $\Rightarrow 'n$ *list* $\Rightarrow 'n$ (**infix** $\cdot\star$ *70*)
  where *lincomb'* $\equiv$ *Codomain.lincomb*

lemma (**in** *RModule*) *RModuleHomI* :
  assumes *GroupHom M T*
    $\bigwedge r\ m.\ r \in R \Longrightarrow m \in M \Longrightarrow T\ (r \cdot m) = smult'\ r\ (T\ m)$
  shows *RModuleHom R smult M smult' T*
  ⟨*proof*⟩

locale *RModuleEnd* = *RModuleHom R smult M smult T*
  for *R* :: *'r::ring-1 set*
  and *smult* :: $'r \Rightarrow 'm$::*ab-group-add* $\Rightarrow 'm$ (**infixr** $\cdot$ *70*)
  and *M* :: *'m set*
  and *T* :: $'m \Rightarrow 'm$
+ **assumes** *endomorph*: $ImG \subseteq M$

locale *ModuleHom* = *RModuleHom UNIV smult M smult' T*
  for *smult* :: $'r$::*ring-1* $\Rightarrow 'm$::*ab-group-add* $\Rightarrow 'm$ (**infixr** $\cdot$ *70*)
  and *M* :: *'m set*
  and *smult'* :: $'r \Rightarrow 'n$::*ab-group-add* $\Rightarrow 'n$ (**infixr** $\star$ *70*)
  and *T* :: $'m \Rightarrow 'n$

lemmas (**in** *ModuleHom*) *hom* = *hom*

lemmas (**in** *Module*) *ModuleHomI* = *RModuleHomI*[*THEN ModuleHom.intro*]

locale *ModuleEnd* = *ModuleHom smult M smult T*
  for *smult* :: $'r$::*ring-1* $\Rightarrow 'm$::*ab-group-add* $\Rightarrow 'm$ (**infixr** $\cdot$ *70*)
  and *M* :: *'m set* **and** *T* :: $'m \Rightarrow 'm$
+ **assumes** *endomorph*: $ImG \subseteq M$

locale *RModuleIso* = *RModuleHom R smult M smult' T*
  for *R* :: *'r::ring-1 set*

and *smult* :: $'r \Rightarrow 'm$::*ab-group-add* $\Rightarrow 'm$ (**infixr** $\cdot$ *70*)
and *M* :: $'m$ *set*
and *smult'* :: $'r \Rightarrow 'n$::*ab-group-add* $\Rightarrow 'n$ (**infixr** $\star$ *70*)
and *T* :: $'m \Rightarrow 'n$
\+ **fixes** *N* :: $'n$ *set*
assumes *bijective*: *bij-betw T M N*

lemma (**in** *RModule*) *RModuleIsoI* :
assumes *GroupIso M T N*
$\bigwedge r\ m.\ r \in R \Longrightarrow m \in M \Longrightarrow T\ (r \cdot m) = smult'\ r\ (T\ m)$
shows *RModuleIso R smult M smult' T N*
⟨*proof*⟩

3.3.2 Basic facts

lemma (**in** *RModule*) *trivial-RModuleHom* :
$\forall r{\in}R.\ smult'\ r\ 0 = 0 \Longrightarrow$ *RModuleHom R smult M smult' 0*
⟨*proof*⟩

lemma (**in** *RModule*) *RModHom-idhom* : *RModuleHom R smult M smult* (*id*↓*M*)
⟨*proof*⟩

context *RModuleHom*
begin

lemmas *additive* = *hom*
lemmas *supp* = *supp*
lemmas *im-zero* = *im-zero*
lemmas *im-diff* = *im-diff*
lemmas *Ker-Im-iff* = *Ker-Im-iff*
lemmas *Ker0-imp-inj-on* = *Ker0-imp-inj-on*

lemma *GroupHom* : *GroupHom M T* ⟨*proof*⟩

lemma *codomain-smult-zero* : $r \in R \Longrightarrow r \star 0 = 0$
⟨*proof*⟩

lemma *RSubmodule-Ker* : *Domain.RSubmodule Ker*
⟨*proof*⟩

lemma *RModule-Im* : *RModule R smult' ImG*
⟨*proof*⟩

lemma *im-submodule* :
assumes *RSubmodule N*
shows *RModule.RSubmodule R smult' ImG* (*T* ‘ *N*)
⟨*proof*⟩

lemma *RModHom-composite-left* :

assumes T ‘ $M \subseteq N$ *RModuleHom R smult′ N smult″ S*
shows *RModuleHom R smult M smult″* $(S \circ T)$
⟨*proof*⟩

lemma *RModuleHom-restrict0-submodule* :
assumes *RSubmodule N*
shows *RModuleHom R smult N smult′* $(T \downarrow N)$
⟨*proof*⟩

lemma *distrib-lincomb* :
set rs $\subseteq R \Longrightarrow$ *set ms* $\subseteq M \Longrightarrow T$ (*rs* ·· *ms*) = *rs* ·⋆ *map T ms*
⟨*proof*⟩

lemma *same-image-on-RSpanset-imp-same-hom* :
assumes *RModuleHom R smult M smult′ S set ms* $\subseteq M$
M = *Domain.R-scalars.RSpan ms* $\forall m \in$ *set ms*. $S\ m = T\ m$
shows $S = T$
⟨*proof*⟩

end

lemma *RSubmodule-eigenspace* :
fixes *smult* :: ′*r*::*ring-1* ⇒ ′*m*::*ab-group-add* ⇒ ′*m* (**infixr** · *70*)
assumes *RModHom*: *RModuleHom R smult M smult T*
and *r*: $r \in R$ $\bigwedge s\ m.\ s \in R \Longrightarrow m \in M \Longrightarrow s \cdot r \cdot m = r \cdot s \cdot m$
defines *E*: $E \equiv \{m \in M.\ T\ m = r \cdot m\}$
shows *RModule.RSubmodule R smult M E*
⟨*proof*⟩

3.3.3 Basic facts about endomorphisms

lemma (**in** *RModule*) *Rmap-endomorph-is-RModuleEnd* :
assumes *grpend*: *GroupEnd M T*
and *Rmap* : $\bigwedge r\ m.\ r \in R \Longrightarrow m \in M \Longrightarrow T\ (r \cdot m) = r \cdot (T\ m)$
shows *RModuleEnd R smult M T*
⟨*proof*⟩

lemma (**in** *RModuleEnd*) *GroupEnd* : *GroupEnd M T*
⟨*proof*⟩

lemmas (**in** *RModuleEnd*) *proj-decomp* = *GroupEnd.proj-decomp*[*OF GroupEnd*]

lemma (**in** *ModuleEnd*) *RModuleEnd* : *RModuleEnd UNIV smult M T*
⟨*proof*⟩

lemmas (**in** *ModuleEnd*) *GroupEnd* = *RModuleEnd.GroupEnd*[*OF RModuleEnd*]

lemma *RModuleEnd-over-UNIV-is-ModuleEnd* :
RModuleEnd UNIV rsmult M T ⟹ *ModuleEnd rsmult M T*

⟨*proof*⟩

3.3.4 Basic facts about isomorphisms

context *RModuleIso*
begin

abbreviation $invT \equiv (the\text{-}inv\text{-}into\ M\ T) \downarrow N$

lemma *GroupIso* : *GroupIso M T N*
⟨*proof*⟩

lemmas *ImG* = *GroupIso.ImG* [*OF GroupIso*]
lemmas *GroupHom-inv* = *GroupIso.inv* [*OF GroupIso*]
lemmas *invT-into* = *GroupIso.invT-into* [*OF GroupIso*]
lemmas *T-invT* = *GroupIso.T-invT* [*OF GroupIso*]
lemmas *invT-eq* = *GroupIso.invT-eq* [*OF GroupIso*]

lemma *RModuleN* : *RModule R smult′ N* ⟨*proof*⟩

lemma *inv* : *RModuleIso R smult′ N smult invT M*
⟨*proof*⟩

end

3.4 Inner direct sums of RModules

lemma (**in** *RModule*) *RModule-inner-dirsum-el-decomp-Rsmult* :
assumes $\forall N{\in}set\ Ns.\ RSubmodule\ N\ add\text{-}independentS\ Ns\ r \in R$
$x \in (\bigoplus N{\leftarrow}Ns.\ N)$
shows $(\bigoplus Ns{\leftarrow}(r \cdot x)) = [r \cdot m.\ m{\leftarrow}(\bigoplus Ns{\leftarrow}x)]$
⟨*proof*⟩

lemma (**in** *RModule*) *RModuleEnd-inner-dirsum-el-decomp-nth* :
assumes $\forall N \in set\ Ns.\ RSubmodule\ N\ add\text{-}independentS\ Ns\ n < length\ Ns$
shows $RModuleEnd\ R\ smult\ (\bigoplus N{\leftarrow}Ns.\ N)\ (\bigoplus Ns{\downarrow}n)$
⟨*proof*⟩

4 Vector Spaces

4.1 Locales and basic facts

Here we don't care about being able to switch scalars.

locale *fscalar-mult* = *scalar-mult smult*
for *smult* :: ′*f*::*field* ⇒ ′*v*::*ab-group-add* ⇒ ′*v* (**infixr** · *70*)

abbreviation (**in** *fscalar-mult*) *findim* ≡ *fingen*

locale *VectorSpace* = *Module smult V*

for *smult* :: *'f*::*field* ⇒ *'v*::*ab-group-add* ⇒ *'v* (**infixr** · *70*)
and *V* :: *'v set*

lemmas *VectorSpaceI* = *ModuleI*[*THEN VectorSpace.intro*]

sublocale *VectorSpace* < *fscalar-mult* ⟨*proof*⟩

locale *FinDimVectorSpace* = *VectorSpace*
\+ **assumes** *findim*: *findim V*

lemma (**in** *VectorSpace*) *FinDimVectorSpaceI* :
findim V ⟹ *FinDimVectorSpace* (·) *V*
⟨*proof*⟩

context *VectorSpace*
begin

abbreviation *Subspace* :: *'v set* ⇒ *bool* **where** *Subspace* ≡ *Submodule*

lemma *SubspaceD1* : *Subspace U* ⟹ *VectorSpace smult U*
⟨*proof*⟩

lemmas *AbGroup* = *AbGroup*
lemmas *add-closed* = *add-closed*
lemmas *smult-closed* = *smult-closed*
lemmas *one-smult* = *one-smult*
lemmas *smult-assoc* = *smult-assoc*
lemmas *smult-distrib-left* = *smult-distrib-left*
lemmas *smult-distrib-right* = *smult-distrib-right*
lemmas *zero-closed* = *zero-closed*
lemmas *zero-smult* = *zero-smult*
lemmas *smult-zero* = *smult-zero*
lemmas *smult-lincomb* = *smult-lincomb*
lemmas *smult-distrib-left-diff* = *smult-distrib-left-diff*
lemmas *smult-sum-distrib* = *smult-sum-distrib*
lemmas *sum-smult-distrib* = *sum-smult-distrib*
lemmas *lincomb-sum* = *lincomb-sum*
lemmas *lincomb-closed* = *lincomb-closed*
lemmas *lincomb-concat* = *lincomb-concat*
lemmas *lincomb-replicate0-left* = *lincomb-replicate0-left*
lemmas *delta-scalars-lincomb-eq-nth* = *delta-scalars-lincomb-eq-nth*
lemmas *SpanI* = *SpanI*
lemmas *Span-closed* = *Span-closed*
lemmas *SpanD-lincomb-arb-len-coeffs* = *SpanD-lincomb-arb-len-coeffs*
lemmas *SpanI-lincomb-arb-len-coeffs* = *SpanI-lincomb-arb-len-coeffs*
lemmas *in-Span-obtain-same-length-coeffs* = *in-Span-obtain-same-length-coeffs*
lemmas *SubspaceI* = *SubmoduleI*
lemmas *subspace-finrank* = *submodule-finrank*

lemma *cancel-scalar*: $[\![\ a \neq 0;\ u \in V;\ v \in V;\ a \cdot u = a \cdot v\ ]\!] \Longrightarrow u = v$
⟨*proof*⟩

end

4.2 Linear algebra in vector spaces

4.2.1 Linear independence and spanning

context *VectorSpace*
begin

lemmas *Subspace-Span* = *Submodule-Span*
lemmas *lin-independent-Nil* = *R-lin-independent-Nil*
lemmas *lin-independentI-concat-all-scalars* = *R-lin-independentI-concat-all-scalars*
lemmas *lin-independentD-all-scalars* = *R-lin-independentD-all-scalars*
lemmas *lin-independent-obtain-unique-scalars* = *R-lin-independent-obtain-unique-scalars*

lemma *lincomb-Cons-0-imp-in-Span* :
$[\![\ v \in V;\ set\ vs \subseteq V;\ a \neq 0;\ (a\#as) \cdot\cdot\ (v\#vs) = 0\ ]\!] \Longrightarrow v \in Span\ vs$
⟨*proof*⟩

lemma *lin-independent-Cons-conditions* :
$[\![\ v \in V;\ set\ vs \subseteq V;\ v \notin Span\ vs;\ lin\text{-}independent\ vs\ ]\!]$
$\Longrightarrow lin\text{-}independent\ (v\#vs)$
⟨*proof*⟩

lemma *coeff-n0-imp-in-Span-others* :
assumes $v \in V$ $set\ us \subseteq V$ $set\ vs \subseteq V$ $b \neq 0$ $length\ as = length\ us$
$w = (as\ @\ b\ \#\ bs) \cdot\cdot\ (us\ @\ v\ \#\ vs)$
shows $v \in Span\ (w\ \#\ us\ @\ vs)$
⟨*proof*⟩

lemma *lin-independent-replace1-by-lincomb* :
assumes $set\ us \subseteq V$ $v \in V$ $set\ vs \subseteq V$ $lin\text{-}independent\ (us\ @\ v\ \#\ vs)$
$length\ as = length\ us$ $b \neq 0$
shows $lin\text{-}independent\ (\ ((as\ @\ b\ \#\ bs) \cdot\cdot\ (us\ @\ v\ \#\ vs))\ \#\ us\ @\ vs\ )$
⟨*proof*⟩

lemma *build-lin-independent-seq* :
assumes *us-V*: $set\ us \subseteq V$
and *indep-us*: $lin\text{-}independent\ us$
shows $\exists\,ws.\ set\ ws \subseteq V \wedge lin\text{-}independent\ (ws\ @\ us) \wedge (Span\ (ws\ @\ us) = V$
$\vee\ length\ ws = n)$
⟨*proof*⟩

end

4.2.2 Basis for a vector space: *basis-for*

abbreviation (**in** *fscalar-mult*) *basis-for* :: $'v\ set \Rightarrow 'v\ list \Rightarrow bool$
where $basis\text{-}for\ V\ vs \equiv (set\ vs \subseteq V \wedge V = Span\ vs \wedge lin\text{-}independent\ vs)$

context *VectorSpace*
begin

lemma *spanset-contains-basis* :
$set\ vs \subseteq V \Longrightarrow \exists us.\ set\ us \subseteq set\ vs \wedge basis\text{-}for\ (Span\ vs)\ us$
⟨*proof*⟩

lemma *basis-for-Span-ex* : $set\ vs \subseteq V \Longrightarrow \exists us.\ basis\text{-}for\ (Span\ vs)\ us$
⟨*proof*⟩

lemma *replace-basis-one-step* :
assumes *closed*: $set\ vs \subseteq V$ $set\ us \subseteq V$ **and** *indep*: $lin\text{-}independent\ (us@vs)$
and *new-w*: $w \in Span\ (us@vs) - Span\ us$
shows $\exists xs\ y\ ys.\ vs = xs\ @\ y\ \#\ ys$
$\wedge\ basis\text{-}for\ (Span\ (us@vs))\ (w\ \#\ us\ @\ xs\ @\ ys)$
⟨*proof*⟩

lemma *replace-basis* :
assumes *closed*: $set\ vs \subseteq V$ **and** *indep-vs*: $lin\text{-}independent\ vs$
shows $[\![\ length\ us \le length\ vs;\ set\ us \subseteq Span\ vs;\ lin\text{-}independent\ us\ ]\!]$
$\Longrightarrow \exists pvs.\ length\ pvs = length\ vs \wedge set\ pvs = set\ vs$
$\wedge\ basis\text{-}for\ (Span\ vs)\ (take\ (length\ vs)\ (us\ @\ pvs))$
⟨*proof*⟩

lemma *replace-basis-completely* :
$[\![\ set\ vs \subseteq V;\ lin\text{-}independent\ vs;\ length\ us = length\ vs;$
$set\ us \subseteq Span\ vs;\ lin\text{-}independent\ us\ ]\!] \Longrightarrow basis\text{-}for\ (Span\ vs)\ us$
⟨*proof*⟩

lemma *basis-for-obtain-unique-scalars* :
$basis\text{-}for\ V\ vs \Longrightarrow v \in V \Longrightarrow \exists!\ as.\ length\ as = length\ vs \wedge v = as \cdot\cdot\ vs$
⟨*proof*⟩

lemma *add-unique-scalars* :
assumes *vs*: $basis\text{-}for\ V\ vs$ **and** *v*: $v \in V$ **and** *v'*: $v' \in V$
defines *as*: $as \equiv (THE\ ds.\ length\ ds = length\ vs \wedge v = ds \cdot\cdot\ vs)$
and *bs*: $bs \equiv (THE\ ds.\ length\ ds = length\ vs \wedge v' = ds \cdot\cdot\ vs)$
and *cs*: $cs \equiv (THE\ ds.\ length\ ds = length\ vs \wedge v{+}v' = ds \cdot\cdot\ vs)$
shows $cs = [a{+}b.\ (a,b) \leftarrow zip\ as\ bs]$
⟨*proof*⟩

lemma *smult-unique-scalars* :
fixes $a::'f$
assumes *vs*: $basis\text{-}for\ V\ vs$ **and** *v*: $v \in V$
defines *as*: $as \equiv (THE\ cs.\ length\ cs = length\ vs \wedge v = cs \cdot\cdot\ vs)$

and bs: $bs \equiv (THE\ cs.\ length\ cs = length\ vs \wedge a \cdot v = cs \cdot\cdot\ vs)$
shows $bs = map\ ((*)\ a)\ as$
⟨*proof*⟩

lemma *max-lin-independent-set-in-Span* :
assumes $set\ vs \subseteq V\ set\ us \subseteq Span\ vs\ lin\text{-}independent\ us$
shows $length\ us \leq length\ vs$
⟨*proof*⟩

lemma *finrank-Span* : $set\ vs \subseteq V \Longrightarrow finrank\ (Span\ vs)$
⟨*proof*⟩

end

4.3 Finite dimensional spaces

context *VectorSpace*
begin

lemma *dim-eq-size-basis* : $basis\text{-}for\ V\ vs \Longrightarrow length\ vs = dim\ V$
⟨*proof*⟩

lemma *finrank-imp-findim* :
assumes *finrank V*
shows *findim V*
⟨*proof*⟩

lemma *subspace-Span-is-findim* :
$[\![\ set\ vs \subseteq V;\ Subspace\ W;\ W \subseteq Span\ vs\ ]\!] \Longrightarrow findim\ W$
⟨*proof*⟩

end

context *FinDimVectorSpace*
begin

lemma *Subspace-is-findim* : $Subspace\ U \Longrightarrow findim\ U$
⟨*proof*⟩

lemma *basis-ex* : $\exists\, vs.\ basis\text{-}for\ V\ vs$
⟨*proof*⟩

lemma *lin-independent-length-le-dim* :
$set\ us \subseteq V \Longrightarrow lin\text{-}independent\ us \Longrightarrow length\ us \leq dim\ V$
⟨*proof*⟩

lemma *too-long-lin-dependent* :
$set\ us \subseteq V \Longrightarrow length\ us > dim\ V \Longrightarrow \neg\ lin\text{-}independent\ us$
⟨*proof*⟩

lemma *extend-lin-independent-to-basis* :
assumes *set us* $\subseteq$ *V lin-independent us*
shows $\exists$ *vs. basis-for V* (*vs* @ *us*)
⟨*proof*⟩

lemma *extend-Subspace-basis* :
$U \subseteq V \Longrightarrow$ *basis-for U us* $\Longrightarrow \exists$ *vs. basis-for V* (*vs*@*us*)
⟨*proof*⟩

lemma *Subspace-dim-le* :
assumes *Subspace U*
shows *dim U* $\leq$ *dim V*
⟨*proof*⟩

lemma *Subspace-eqdim-imp-equal* :
assumes *Subspace U dim U* = *dim V*
shows $U = V$
⟨*proof*⟩

lemma *Subspace-dim-lt* : *Subspace U* $\Longrightarrow U \neq V \Longrightarrow$ *dim U* $<$ *dim V*
⟨*proof*⟩

lemma *semisimple* :
assumes *Subspace U*
shows $\exists$ *W*. *Subspace W* $\wedge$ ($V = W \oplus U$)
⟨*proof*⟩

end

4.4 Vector space homomorphisms

4.4.1 Locales

locale *VectorSpaceHom* = *ModuleHom smult V smult′ T*
for *smult* :: ′*f*::*field* ⇒ ′*v*::*ab-group-add* ⇒ ′*v* (**infixr** · *70*)
and *V* :: ′*v set*
and *smult′* :: ′*f* ⇒ ′*w*::*ab-group-add* ⇒ ′*w* (**infixr** ⋆ *70*)
and *T* :: ′*v* ⇒ ′*w*

sublocale *VectorSpaceHom* < *VectorSpace* ⟨*proof*⟩

lemmas (**in** *VectorSpace*)
VectorSpaceHomI = *ModuleHomI*[*THEN VectorSpaceHom.intro*]

lemma (**in** *VectorSpace*) *VectorSpaceHomI-fromaxioms* :
assumes $\bigwedge g\ g'.\ g \in V \Longrightarrow g' \in V \Longrightarrow T\ (g + g') = T\ g + T\ g'$
supp T $\subseteq V$
$\bigwedge r\ m.\ r \in UNIV \Longrightarrow m \in V \Longrightarrow T\ (r \cdot m) = smult'\ r\ (T\ m)$
shows *VectorSpaceHom smult V smult′ T*

$\langle proof \rangle$

locale *VectorSpaceEnd* = *VectorSpaceHom smult V smult T*
  for *smult* :: $'f$::*field* ⇒ $'v$::*ab-group-add* ⇒ $'v$ (**infixr** · *70*)
  and *V* :: $'v$ *set*
  and *T* :: $'v$ ⇒ $'v$
+ **assumes** *endomorph*: *ImG* ⊆ *V*

abbreviation (**in** *VectorSpace*) *VEnd* ≡ *VectorSpaceEnd smult V*

lemma *VectorSpaceEndI* :
  fixes *smult* :: $'f$::*field* ⇒ $'v$::*ab-group-add* ⇒ $'v$
  assumes *VectorSpaceHom smult V smult T T ‘ V* ⊆ *V*
  shows *VectorSpaceEnd smult V T*
  $\langle proof \rangle$

lemma (**in** *VectorSpaceEnd*) *VectorSpaceHom*: *VectorSpaceHom smult V smult T*
  $\langle proof \rangle$

lemma (**in** *VectorSpaceEnd*) *ModuleEnd* : *ModuleEnd smult V T*
  $\langle proof \rangle$

locale *VectorSpaceIso* = *VectorSpaceHom smult V smult′ T*
  for *smult* :: $'f$::*field* ⇒ $'v$::*ab-group-add* ⇒ $'v$ (**infixr** · *70*)
  and *V* :: $'v$ *set*
  and *smult′* :: $'f$ ⇒ $'w$::*ab-group-add* ⇒ $'w$ (**infixr** ⋆ *70*)
  and *T* :: $'v$ ⇒ $'w$
+ **fixes** *W* :: $'w$ *set*
  assumes *bijective*: *bij-betw T V W*

abbreviation (**in** *VectorSpace*) *isomorphic* ::
  ($'f$ ⇒ $'w$::*ab-group-add* ⇒ $'w$) ⇒ $'w$ *set* ⇒ *bool*
  where *isomorphic smult′ W* ≡ (∃ *T*. *VectorSpaceIso smult V smult′ T W*)

4.4.2 Basic facts

lemma (**in** *VectorSpace*) *trivial-VectorSpaceHom* :
  ($\bigwedge$*a*. *smult′ a 0* = *0*) ⟹ *VectorSpaceHom smult V smult′ 0*
  $\langle proof \rangle$

lemma (**in** *VectorSpace*) *VectorSpaceHom-idhom* :
  VectorSpaceHom smult V smult (*id*↓*V*)
  $\langle proof \rangle$

context *VectorSpaceHom*
begin

lemmas *hom* = *hom*
lemmas *supp* = *supp*

lemmas *f-map* = *R-map*
lemmas *im-zero* = *im-zero*
lemmas *im-sum-list-prod* = *im-sum-list-prod*
lemmas *additive* = *additive*
lemmas *GroupHom* = *GroupHom*
lemmas *distrib-lincomb* = *distrib-lincomb*

lemmas *same-image-on-spanset-imp-same-hom*
= *same-image-on-RSpanset-imp-same-hom*[
OF ModuleHom.axioms(*1*), *OF VectorSpaceHom.axioms*(*1*)
]

lemma *VectorSpace-Im* : *VectorSpace smult′ ImG*
⟨*proof*⟩

lemma *VectorSpaceHom-scalar-mul* :
VectorSpaceHom smult V smult′ ($\lambda v.\ a \star T\ v$)
⟨*proof*⟩

lemma *VectorSpaceHom-composite-left* :
assumes *ImG* ⊆ *W VectorSpaceHom smult′ W smult″ S*
shows *VectorSpaceHom smult V smult″* ($S \circ T$)
⟨*proof*⟩

lemma *findim-domain-findim-image* :
assumes *findim V*
shows *fscalar-mult.findim smult′ ImG*
⟨*proof*⟩

end

lemma (**in** *VectorSpace*) *basis-im-defines-hom* :
fixes *smult′* :: *′f* ⇒ *′w*::*ab-group-add* ⇒ *′w* (**infixr** ⋆ *70*)
and *lincomb′* :: *′f list* ⇒ *′w list* ⇒ *′w* (**infixr** ·⋆ *70*)
defines *lincomb′* : *lincomb′* ≡ *scalar-mult.lincomb smult′*
assumes *VSpW* : *VectorSpace smult′ W*
and *basisV* : *basis-for V vs*
and *basisV-im* : *set ws* ⊆ *W length ws* = *length vs*
shows ∃! *T*. *VectorSpaceHom smult V smult′ T* ∧ *map T vs* = *ws*
⟨*proof*⟩

4.4.3 Hom-sets

definition *VectorSpaceHomSet* ::
(*′f*::*field* ⇒ *′v*::*ab-group-add* ⇒ *′v*) ⇒ *′v set* ⇒ (*′f* ⇒ *′w*::*ab-group-add* ⇒ *′w*)
⇒ *′w set* ⇒ (*′v* ⇒ *′w*) *set*
where *VectorSpaceHomSet fsmult V fsmult′ W*
≡ {*T*. *VectorSpaceHom fsmult V fsmult′ T*} ∩ {*T*. *T* ‘ *V* ⊆ *W*}

abbreviation (**in** *VectorSpace*) *VectorSpaceEndSet* $\equiv$ $\{S.\ VEnd\ S\}$

lemma *VectorSpaceHomSetI* :
  $VectorSpaceHom\ fsmult\ V\ fsmult'\ T \Longrightarrow T\ `\ V \subseteq W$
    $\Longrightarrow T \in VectorSpaceHomSet\ fsmult\ V\ fsmult'\ W$
  ⟨*proof*⟩

lemma *VectorSpaceHomSetD-VectorSpaceHom* :
  $T \in VectorSpaceHomSet\ fsmult\ V\ fsmult'\ N$
    $\Longrightarrow VectorSpaceHom\ fsmult\ V\ fsmult'\ T$
  ⟨*proof*⟩

lemma *VectorSpaceHomSetD-Im* :
  $T \in VectorSpaceHomSet\ fsmult\ V\ fsmult'\ W \Longrightarrow T\ `\ V \subseteq W$
  ⟨*proof*⟩

context *VectorSpace*
begin

lemma *VectorSpaceHomSet-is-fmaps-in-GroupHomSet* :
  fixes $smult' :: 'f \Rightarrow 'w::ab\text{-}group\text{-}add \Rightarrow 'w$ (**infixr** $\star$ *70*)
  shows $VectorSpaceHomSet\ smult\ V\ smult'\ W$
      $= (GroupHomSet\ V\ W) \cap \{T.\ \forall\, a.\ \forall\, v{\in}V.\ T\ (a \cdot v) = a \star (T\ v)\}$
⟨*proof*⟩

lemma *Group-VectorSpaceHomSet* :
  fixes $smult' :: 'f \Rightarrow 'w::ab\text{-}group\text{-}add \Rightarrow 'w$ (**infixr** $\star$ *70*)
  assumes *VectorSpace smult′ W*
  shows *Group* (*VectorSpaceHomSet smult V smult′ W*)
⟨*proof*⟩

lemma *VectorSpace-VectorSpaceHomSet* :
  fixes $smult' :: 'f \Rightarrow 'w::ab\text{-}group\text{-}add \Rightarrow 'w$ (**infixr** $\star$ *70*)
  and $hom\text{-}smult :: 'f \Rightarrow ('v \Rightarrow 'w) \Rightarrow ('v \Rightarrow 'w)$ (**infixr** $\star\cdot$ *70*)
  defines $hom\text{-}smult$: $hom\text{-}smult \equiv \lambda a\ T\ v.\ a \star T\ v$
  assumes *VSpW*: *VectorSpace smult′ W*
  shows *VectorSpace hom-smult* (*VectorSpaceHomSet smult V smult′ W*)
⟨*proof*⟩

end

4.4.4 Basic facts about endomorphisms

lemma *ModuleEnd-over-field-is-VectorSpaceEnd* :
  fixes $smult :: 'f::field \Rightarrow 'v::ab\text{-}group\text{-}add \Rightarrow 'v$
  assumes *ModuleEnd smult V T*
  shows *VectorSpaceEnd smult V T*
⟨*proof*⟩

context *VectorSpace*
begin

lemmas *VectorSpaceEnd-inner-dirsum-el-decomp-nth* =
  RModuleEnd-inner-dirsum-el-decomp-nth[
    THEN RModuleEnd-over-UNIV-is-ModuleEnd,
    THEN ModuleEnd-over-field-is-VectorSpaceEnd
  ]

abbreviation *end-smult* :: $'f \Rightarrow ('v \Rightarrow 'v) \Rightarrow ('v \Rightarrow 'v)$ (**infixr** ·· *70*)
  where $a \cdot\cdot\ T \equiv (\lambda v.\ a \cdot T\ v)$

abbreviation *end-lincomb*
  :: $'f\ list \Rightarrow (('v \Rightarrow 'v)\ list) \Rightarrow ('v \Rightarrow 'v)$ (**infixr** ··· *70*)
  where *end-lincomb* ≡ *scalar-mult.lincomb end-smult*

lemma *end-smult0*: $a \cdot\cdot\ 0 = 0$
  ⟨*proof*⟩

lemma *end-0smult*: *range* $T \subseteq V \Longrightarrow 0 \cdot\cdot\ T = 0$
  ⟨*proof*⟩

lemma *end-smult-distrib-left* :
  assumes *range* $S \subseteq V$ *range* $T \subseteq V$
  shows $a \cdot\cdot\ (S + T) = a \cdot\cdot\ S + a \cdot\cdot\ T$
⟨*proof*⟩

lemma *end-smult-distrib-right* :
  assumes *range* $T \subseteq V$
  shows $(a+b) \cdot\cdot\ T = a \cdot\cdot\ T + b \cdot\cdot\ T$
⟨*proof*⟩

lemma *end-smult-assoc* :
  assumes *range* $T \subseteq V$
  shows $a \cdot\cdot\ b \cdot\cdot\ T = (a * b) \cdot\cdot\ T$
⟨*proof*⟩

lemma *end-smult-comp-comm-left* : $(a \cdot\cdot\ T) \circ S = a \cdot\cdot\ (T \circ S)$
  ⟨*proof*⟩

lemma *end-idhom* : *VEnd* (*id*↓*V*)
  ⟨*proof*⟩

lemma *VectorSpaceEndSet-is-VectorSpaceHomSet* :
  VectorSpaceHomSet smult V smult V = {*T*. *VEnd T*}
⟨*proof*⟩

lemma *VectorSpace-VectorSpaceEndSet* : *VectorSpace end-smult VectorSpaceEnd-Set*

⟨*proof*⟩

end

context *VectorSpaceEnd*
begin

lemmas *f-map* = *R-map*
lemmas *supp* = *supp*
lemmas *GroupEnd* = *ModuleEnd.GroupEnd*[*OF ModuleEnd*]
lemmas *idhom-left* = *idhom-left*
lemmas *range* = *GroupEnd.range*[*OF GroupEnd*]
lemmas *Ker0-imp-inj-on* = *Ker0-imp-inj-on*
lemmas *inj-on-imp-Ker0* = *inj-on-imp-Ker0*
lemmas *nonzero-Ker-el-imp-n-inj* = *nonzero-Ker-el-imp-n-inj*
lemmas *VectorSpaceHom-composite-left*
= *VectorSpaceHom-composite-left*[*OF endomorph*]

lemma *in-VEndSet* : $T \in VectorSpaceEndSet$
⟨*proof*⟩

lemma *end-smult-comp-comm-right* :
$range\ S \subseteq V \Longrightarrow T \circ (a \cdot\cdot S) = a \cdot\cdot (T \circ S)$
⟨*proof*⟩

lemma *VEnd-end-smult-VEnd* : $VEnd\ (a \cdot\cdot T)$
⟨*proof*⟩

lemma *VEnd-composite-left* :
assumes *VEnd S*
shows $VEnd\ (S \circ T)$
⟨*proof*⟩

lemma *VEnd-composite-right* : $VEnd\ S \Longrightarrow VEnd\ (T \circ S)$
⟨*proof*⟩

end

lemma (**in** *VectorSpace*) *inj-comp-end* :
assumes *VEnd S inj-on S V VEnd T inj-on T V*
shows $inj\text{-}on\ (S \circ T)\ V$
⟨*proof*⟩

lemma (**in** *VectorSpace*) *n-inj-comp-end* :
$[\![\ VEnd\ S;\ VEnd\ T;\ \neg\ inj\text{-}on\ (S \circ T)\ V\ ]\!] \Longrightarrow \neg\ inj\text{-}on\ S\ V \lor \neg\ inj\text{-}on\ T\ V$
⟨*proof*⟩

4.4.5 Polynomials of endomorphisms

context *VectorSpaceEnd*
begin

primrec *endpow* :: *nat* ⇒ ('*v*⇒'*v*)
 where *endpow0*: *endpow 0* = *id*↓*V*
 | *endpowSuc*: *endpow* (*Suc n*) = *T* ∘ (*endpow n*)

definition *polymap* :: '*f poly* ⇒ ('*v*⇒'*v*)
 where *polymap p* ≡ (*coeffs p*) ··· (*map endpow* [*0..<Suc* (*degree p*)])

lemma *VEnd-endpow* : *VEnd* (*endpow n*)
⟨*proof*⟩

lemma *endpow-list-apply-closed* :
 v ∈ *V* ⟹ *set* (*map* (λ*S*. *S v*) (*map endpow* [*0..<k*])) ⊆ *V*
 ⟨*proof*⟩

lemma *map-endpow-Suc* :
 map endpow [*0..<Suc n*] = (*id*↓*V*) # *map* ((∘) *T*) (*map endpow* [*0..<n*])
⟨*proof*⟩

lemma *T-endpow-list-apply-commute* :
 map T (*map* (λ*S*. *S v*) (*map endpow* [*0..<n*]))
  = *map* (λ*S*. *S v*) (*map* ((∘) *T*) (*map endpow* [*0..<n*]))
 ⟨*proof*⟩

lemma *polymap0* : *polymap 0* = *0*
 ⟨*proof*⟩

lemma *VEnd-polymap* : *VEnd* (*polymap p*)
⟨*proof*⟩

lemma *polymap-pCons* : *polymap* (*pCons a p*) = *a* ·· (*id*↓*V*) + (*T* ∘ (*polymap p*))
⟨*proof*⟩

lemma *polymap-plus* : *polymap* (*p* + *q*) = *polymap p* + *polymap q*
⟨*proof*⟩

lemma *polymap-polysmult* : *polymap* (*Polynomial.smult a p*) = *a* ·· *polymap p*
⟨*proof*⟩

lemma *polymap-times* : *polymap* (*p* ∗ *q*) = (*polymap p*) ∘ (*polymap q*)
⟨*proof*⟩

lemma *polymap-apply* :
 assumes *v* ∈ *V*
 shows *polymap p v* = (*coeffs p*)
  ·· (*map* (λ*S*. *S v*) (*map endpow* [*0..<Suc* (*degree p*)]))

$\langle proof \rangle$

lemma *polymap-apply-linear* : $v \in V \Longrightarrow polymap$ $[:-c,\ 1:]$ $v = T\ v - c \cdot v$
$\langle proof \rangle$

lemma *polymap-const-inj* :
assumes $degree\ p = 0\ p \neq 0$
shows $inj\text{-}on\ (polymap\ p)\ V$
$\langle proof \rangle$

lemma *n-inj-polymap-times* :
$\neg\ inj\text{-}on\ (polymap\ (p * q))\ V$
$\Longrightarrow \neg\ inj\text{-}on\ (polymap\ p)\ V \vee \neg\ inj\text{-}on\ (polymap\ q)\ V$
$\langle proof \rangle$

In the following lemma, $[:-\ c,\ 1::'a:]$ is the linear polynomial $x - c$.

lemma *n-inj-polymap-findlinear* :
assumes *alg-closed*: $\bigwedge p::'f\ poly.\ degree\ p > 0 \Longrightarrow \exists\, c.\ poly\ p\ c = 0$
shows $p \neq 0 \Longrightarrow \neg\ inj\text{-}on\ (polymap\ p)\ V$
$\Longrightarrow \exists\, c.\ \neg\ inj\text{-}on\ (polymap\ [:-c,\ 1:])\ V$
$\langle proof \rangle$

end

4.4.6 Existence of eigenvectors of endomorphisms of finite-dimensional vector spaces

lemma (**in** *FinDimVectorSpace*) *endomorph-has-eigenvector* :
assumes *alg-closed*: $\bigwedge p::'a\ poly.\ degree\ p > 0 \Longrightarrow \exists\, c.\ poly\ p\ c = 0$
and *dim* : $dim\ V > 0$
and *endo* : $VectorSpaceEnd\ smult\ V\ T$
shows $\exists\, c\ u.\ u \in V \wedge u \neq 0 \wedge T\ u = c \cdot u$
$\langle proof \rangle$

5 Modules Over a Group Ring

5.1 Almost-everywhere-zero functions as scalars

locale *aezfun-scalar-mult* = *scalar-mult smult*
for *smult* ::
$('r::ring\text{-}1,\ 'g::group\text{-}add)\ aezfun \Rightarrow 'v::ab\text{-}group\text{-}add \Rightarrow 'v$ (**infixr** $\cdot$ *70*)
begin

definition *fsmult* :: $'r \Rightarrow 'v \Rightarrow 'v$ (**infixr** $\sharp\cdot$ *70*) **where** $a\ \sharp\cdot\ v \equiv (a\ \delta\delta\ 0) \cdot v$
abbreviation *flincomb* :: $'r\ list \Rightarrow 'v\ list \Rightarrow 'v$ (**infixr** $\cdot\sharp\cdot$ *70*)
where $as\ \cdot\sharp\cdot\ vs \equiv scalar\text{-}mult.lincomb\ fsmult\ as\ vs$
abbreviation *f-lin-independent* :: $'v\ list \Rightarrow bool$
where $f\text{-}lin\text{-}independent \equiv scalar\text{-}mult.lin\text{-}independent\ fsmult$
abbreviation *fSpan* :: $'v\ list \Rightarrow 'v\ set$ **where** $fSpan \equiv scalar\text{-}mult.Span\ fsmult$

definition *Gmult* :: $'g \Rightarrow 'v \Rightarrow 'v$ (**infixr** $*\cdot$ *70*) **where** $g *\cdot v \equiv (1\ \delta\delta\ g) \cdot v$

lemmas *R-scalar-mult* = *R-scalar-mult*

lemma *fsmultD* : $a \sharp\cdot v = (a\ \delta\delta\ 0) \cdot v$
⟨*proof*⟩

lemma *GmultD* : $g *\cdot v = (1\ \delta\delta\ g) \cdot v$
⟨*proof*⟩

definition *negGorbit-list* :: $'g\ list \Rightarrow ('a \Rightarrow 'v) \Rightarrow 'a\ list \Rightarrow 'v\ list\ list$
where *negGorbit-list gs T as* $\equiv$ *map* ($\lambda g.$ *map* (*Gmult* $(-g) \circ T$) *as*) *gs*

lemma *negGorbit-Cons* :
negGorbit-list (*g#gs*) *T as*
= (*map* (*Gmult* $(-g) \circ T$) *as*) # *negGorbit-list gs T as*
⟨*proof*⟩

lemma *length-negGorbit-list* : *length* (*negGorbit-list gs T as*) = *length gs*
⟨*proof*⟩

lemma *length-negGorbit-list-sublist* :
fs $\in$ *set* (*negGorbit-list gs T as*) $\Longrightarrow$ *length fs* = *length as*
⟨*proof*⟩

lemma *length-concat-negGorbit-list* :
length (*concat* (*negGorbit-list gs T as*)) = (*length gs*) $*$ (*length as*)
⟨*proof*⟩

lemma *negGorbit-list-nth* :
$\bigwedge i.\ i <$ *length gs* $\Longrightarrow$ (*negGorbit-list gs T as*)!*i* = *map* (*Gmult* $(-gs!i) \circ T$) *as*
⟨*proof*⟩

end

5.2 Locale and basic facts

locale *FGModule* = *ActingGroup?*: *Group G*
+ *FGMod?*: *RModule ActingGroup.group-ring smult V*
for *G* :: $'g$::*group-add set*
and *smult* :: ($'f$::*field*, $'g$) *aezfun* $\Rightarrow$ $'v$::*ab-group-add* $\Rightarrow$ $'v$ (**infixr** $\cdot$ *70*)
and *V* :: $'v$ *set*

sublocale *FGModule* < *aezfun-scalar-mult* ⟨*proof*⟩

lemma (**in** *Group*) *trivial-FGModule* :
fixes *smult* :: ($'f$::*field*, $'g$) *aezfun* $\Rightarrow$ $'v$::*ab-group-add* $\Rightarrow$ $'v$
assumes *smult-zero*: $\forall\, a \in$ *group-ring*. *smult a* (*0*::$'v$) = *0*
shows *FGModule G smult* (*0*::$'v$ *set*)

$\langle proof \rangle$

context *FGModule*
begin

abbreviation $FG :: ('f,'g)\ aezfun\ set$ **where** $FG \equiv ActingGroup.group\text{-}ring$
abbreviation $FGSubmodule \equiv RSubmodule$
abbreviation $FG\text{-}proj \equiv ActingGroup.RG\text{-}proj$

lemma *GroupG*: *Group G* $\langle proof \rangle$

lemmas *zero-closed* = *zero-closed*
lemmas *neg-closed* = *neg-closed*
lemmas *diff-closed* = *diff-closed*
lemmas *zero-smult* = *zero-smult*
lemmas *smult-zero* = *smult-zero*
lemmas *AbGroup* = *AbGroup*
lemmas *sum-closed* = *AbGroup.sum-closed*[*OF AbGroup*]
lemmas *FGSubmoduleI* = *RSubmoduleI*
lemmas *FG-proj-mult-leftdelta* = *ActingGroup.RG-proj-mult-leftdelta*
lemmas *FG-proj-mult-right* = *ActingGroup.RG-proj-mult-right*
lemmas *FG-el-decomp* = *ActingGroup.RG-el-decomp-aezdeltafun*

lemma *FG-n0*: $FG \neq 0$ $\langle proof \rangle$

lemma *FG-proj-in-FG* : $FG\text{-}proj\ x \in FG$
$\langle proof \rangle$

lemma *FG-fddg-closed* : $g \in G \Longrightarrow a\ \delta\delta\ g \in FG$
$\langle proof \rangle$

lemma *FG-fdd0-closed* : $a\ \delta\delta\ 0 \in FG$
$\langle proof \rangle$

lemma *Gmult-closed* : $g \in G \Longrightarrow v \in V \Longrightarrow g *\cdot v \in V$
$\langle proof \rangle$

lemma *map-Gmult-closed* :
$g \in G \Longrightarrow set\ vs \subseteq V \Longrightarrow set\ (map\ ((*\cdot)\ g)\ vs) \subseteq V$
$\langle proof \rangle$

lemma *Gmult0* :
assumes $v \in V$
shows $0 *\cdot v = v$
$\langle proof \rangle$

lemma *Gmult-assoc* :
assumes $g \in G\ h \in G\ v \in V$
shows $g *\cdot h *\cdot v = (g + h) *\cdot v$

$\langle proof \rangle$

lemma *Gmult-distrib-left* :
$[\![\ g \in G;\ v \in V;\ v' \in V\ ]\!] \Longrightarrow g *\cdot (v + v') = g *\cdot v + g *\cdot v'$
$\langle proof \rangle$

lemma *neg-Gmult* : $g \in G \Longrightarrow v \in V \Longrightarrow g *\cdot (- v) = - (g *\cdot v)$
$\langle proof \rangle$

lemma *Gmult-neg-left* : $g \in G \Longrightarrow v \in V \Longrightarrow (- g) *\cdot g *\cdot v = v$
$\langle proof \rangle$

lemma *fddg-smult-decomp* : $g \in G \Longrightarrow v \in V \Longrightarrow (f\ \delta\delta\ g) \cdot v = f \sharp\cdot g *\cdot v$
$\langle proof \rangle$

lemma *sum-list-aezdeltafun-smult-distrib* :
assumes $v \in V$ $set\ (map\ snd\ fgs) \subseteq G$
shows $(\sum (f,g){\leftarrow}fgs.\ f\ \delta\delta\ g) \cdot v = (\sum (f,g){\leftarrow}fgs.\ f \sharp\cdot g *\cdot v)$
$\langle proof \rangle$

abbreviation *GSubspace* $\equiv$ *RSubmodule*
abbreviation *GSpan* $\equiv$ *RSpan*
abbreviation *G-fingen* $\equiv$ *R-fingen*

lemma *GSubspaceI* : $FGModule\ G\ smult\ U \Longrightarrow U \subseteq V \Longrightarrow GSubspace\ U$
$\langle proof \rangle$

lemma *GSubspace-is-FGModule* :
assumes *GSubspace U*
shows *FGModule G smult U*
$\langle proof \rangle$

lemma *restriction-to-subgroup-is-module* :
fixes $H :: 'g\ set$
assumes *subgrp*: *Group.Subgroup G H*
shows *FGModule H smult V*
$\langle proof \rangle$

lemma *negGorbit-list-V* :
assumes $set\ gs \subseteq G$ $T\ `\ (set\ as) \subseteq V$
shows $set\ (concat\ (negGorbit\text{-}list\ gs\ T\ as)) \subseteq V$
$\langle proof \rangle$

lemma *negGorbit-list-Cons0* :
$T\ `\ (set\ as) \subseteq V$
$\Longrightarrow negGorbit\text{-}list\ (0\#gs)\ T\ as = (map\ T\ as)\ \#\ (negGorbit\text{-}list\ gs\ T\ as)$
$\langle proof \rangle$

end

5.3 Modules over a group ring as a vector spaces

context *FGModule*
begin

lemma *fVectorSpace* : *VectorSpace fsmult V*
⟨*proof*⟩

abbreviation *fSubspace* ≡ *VectorSpace.Subspace fsmult V*
abbreviation *fbasis-for* ≡ *fscalar-mult.basis-for fsmult*
abbreviation *fdim* ≡ *scalar-mult.dim fsmult V*

lemma *VectorSpace-fSubspace* : *fSubspace W* $\Longrightarrow$ *VectorSpace fsmult W*
⟨*proof*⟩

lemma *fsmult-closed* : $v \in V \Longrightarrow a \sharp\cdot v \in V$
⟨*proof*⟩

lemmas *one-fsmult* [*simp*] = *VectorSpace.one-smult* [*OF fVectorSpace*]
lemmas *fsmult-assoc* [*simp*] = *VectorSpace.smult-assoc* [*OF fVectorSpace*]
lemmas *fsmult-zero* [*simp*] = *VectorSpace.smult-zero* [*OF fVectorSpace*]
lemmas *fsmult-distrib-left* [*simp*] = *VectorSpace.smult-distrib-left*
[*OF fVectorSpace*]
lemmas *flincomb-closed* = *VectorSpace.lincomb-closed* [*OF fVectorSpace*]
lemmas *fsmult-sum-distrib* = *VectorSpace.smult-sum-distrib* [*OF fVectorSpace*]
lemmas *sum-fsmult-distrib* = *VectorSpace.sum-smult-distrib* [*OF fVectorSpace*]
lemmas *flincomb-concat* = *VectorSpace.lincomb-concat* [*OF fVectorSpace*]
lemmas *fSpan-closed* = *VectorSpace.Span-closed* [*OF fVectorSpace*]
lemmas *flin-independentD-all-scalars*
= *VectorSpace.lin-independentD-all-scalars*[*OF fVectorSpace*]
lemmas *in-fSpan-obtain-same-length-coeffs*
= *VectorSpace.in-Span-obtain-same-length-coeffs* [*OF fVectorSpace*]

lemma *fsmult-smult-comm* : $r \in FG \Longrightarrow v \in V \Longrightarrow a \sharp\cdot r \cdot v = r \cdot a \sharp\cdot v$
⟨*proof*⟩

lemma *fsmult-Gmult-comm* : $g \in G \Longrightarrow v \in V \Longrightarrow a \sharp\cdot g *\cdot v = g *\cdot a \sharp\cdot v$
⟨*proof*⟩

lemma *Gmult-flincomb-comm* :
assumes $g \in G$ *set vs* $\subseteq V$
shows $g *\cdot as \cdot\sharp\cdot vs = as \cdot\sharp\cdot (map\ (Gmult\ g)\ vs)$
⟨*proof*⟩

lemma *GSubspace-is-Subspace* :
GSubspace U $\Longrightarrow$ *VectorSpace.Subspace fsmult V U*
⟨*proof*⟩

end

5.4 Homomorphisms of modules over a group ring

5.4.1 Locales

locale *FGModuleHom* = *ActingGroup?*: *Group G*
+ *RModHom?*: *RModuleHom ActingGroup.group-ring smult V smult' T*
  for *G* :: *'g::group-add set*
  and *smult* :: (*'f::field*, *'g*) *aezfun* ⇒ *'v::ab-group-add* ⇒ *'v* (**infixr** · *70*)
  and *V* :: *'v set*
  and *smult'* :: (*'f*, *'g*) *aezfun* ⇒ *'w::ab-group-add* ⇒ *'w* (**infixr** ⋆ *70*)
  and *T* :: *'v* ⇒ *'w*

sublocale *FGModuleHom* < *FGModule* ⟨*proof*⟩

lemma (**in** *FGModule*) *FGModuleHomI-fromaxioms* :
  assumes $\bigwedge v\ v'.\ v \in V \Longrightarrow v' \in V \Longrightarrow T\ (v + v') = T\ v + T\ v'$
    $supp\ T \subseteq V$ $\bigwedge r\ m.\ r \in FG \Longrightarrow m \in V \Longrightarrow T\ (r \cdot m) = smult'\ r\ (T\ m)$
  shows *FGModuleHom G smult V smult' T*
  ⟨*proof*⟩

locale *FGModuleEnd* = *FGModuleHom G smult V smult T*
  for *G* :: *'g::group-add set*
  and *FG* :: (*'f::field*,*'g*) *aezfun set*
  and *smult* :: (*'f*, *'g*) *aezfun* ⇒ *'v::ab-group-add* ⇒ *'v* (**infixr** · *70*)
  and *V* :: *'v set*
  and *T* :: *'v* ⇒ *'v*
+ **assumes** *endomorph*: *ImG* ⊆ *V*

locale *FGModuleIso* = *FGModuleHom G smult V smult' T*
  for *G* :: *'g::group-add set*
  and *smult* :: (*'f::field*, *'g*) *aezfun* ⇒ *'v::ab-group-add* ⇒ *'v* (**infixr** · *70*)
  and *V* :: *'v set*
  and *smult'* :: (*'f*, *'g*) *aezfun* ⇒ *'w::ab-group-add* ⇒ *'w* (**infixr** ⋆ *70*)
  and *T* :: *'v* ⇒ *'w*
+ **fixes** *W* :: *'w set*
  assumes *bijective*: *bij-betw T V W*

abbreviation (**in** *FGModule*) *isomorphic* ::
  ((*'f*,*'g*) *aezfun* ⇒ *'w::ab-group-add* ⇒ *'w*) ⇒ *'w set* ⇒ *bool*
  where *isomorphic smult' W* ≡ (∃ *T*. *FGModuleIso G smult V smult' T W*)

5.4.2 Basic facts

context *FGModule*
begin

lemma *trivial-FGModuleHom* :
  assumes $\bigwedge r.\ r \in FG \Longrightarrow smult'\ r\ 0 = 0$
  shows *FGModuleHom G smult V smult' 0*
⟨*proof*⟩

lemma *FGModHom-idhom* : *FGModuleHom G smult V smult* $(id\downarrow V)$
⟨*proof*⟩

lemma *VecHom-GMap-is-FGModuleHom* :
fixes *smult′* :: $('f, 'g)$ *aezfun* ⇒ *′w::ab-group-add* ⇒ *′w* (**infixr** ⋆ *70*)
and *fsmult′* :: *′f* ⇒ *′w* ⇒ *′w* (**infixr** ♯⋆ *70*)
and *Gmult′* :: *′g* ⇒ *′w* ⇒ *′w* (**infixr** ∗⋆ *70*)
defines *fsmult′*: *fsmult′* ≡ *aezfun-scalar-mult.fsmult smult′*
and *Gmult′* : *Gmult′* ≡ *aezfun-scalar-mult.Gmult smult′*
assumes *hom* : *VectorSpaceHom fsmult V fsmult′ T*
and *Im-W* : *FGModule G smult′ W T ‘ V* ⊆ *W*
and *G-map* : $\bigwedge g\ v.\ g \in G \Longrightarrow v \in V \Longrightarrow T\ (g *\cdot v) = g ** (T\ v)$
shows *FGModuleHom G smult V smult′ T*
⟨*proof*⟩

lemma *VecHom-GMap-on-fbasis-is-FGModuleHom* :
fixes *smult′* :: $('f, 'g)$ *aezfun* ⇒ *′w::ab-group-add* ⇒ *′w* (**infixr** ⋆ *70*)
and *fsmult′* :: *′f* ⇒ *′w* ⇒ *′w* (**infixr** ♯⋆ *70*)
and *Gmult′* :: *′g* ⇒ *′w* ⇒ *′w* (**infixr** ∗⋆ *70*)
and *flincomb′* :: *′f list* ⇒ *′w list* ⇒ *′w* (**infixr** ·♯⋆ *70*)
defines *fsmult′* : *fsmult′* ≡ *aezfun-scalar-mult.fsmult smult′*
and *Gmult′* : *Gmult′* ≡ *aezfun-scalar-mult.Gmult smult′*
and *flincomb′* : *flincomb′* ≡ *aezfun-scalar-mult.flincomb smult′*
assumes *fbasis* : *fbasis-for V vs*
and *hom* : *VectorSpaceHom fsmult V fsmult′ T*
and *Im-W* : *FGModule G smult′ W T ‘ V* ⊆ *W*
and *G-map* : $\bigwedge g\ v.\ g \in G \Longrightarrow v \in set\ vs \Longrightarrow T\ (g *\cdot v) = g ** (T\ v)$
shows *FGModuleHom G smult V smult′ T*
⟨*proof*⟩

end

context *FGModuleHom*
begin

abbreviation *fsmult′* :: *′f* ⇒ *′w* ⇒ *′w* (**infixr** ♯⋆ *70*)
where *fsmult′* ≡ *aezfun-scalar-mult.fsmult smult′*
abbreviation *Gmult′* :: *′g* ⇒ *′w* ⇒ *′w* (**infixr** ∗⋆ *70*)
where *Gmult′* ≡ *aezfun-scalar-mult.Gmult smult′*

lemmas *supp* = *supp*
lemmas *additive* = *additive*
lemmas *FG-map* = *R-map*
lemmas *FG-fdd0-closed* = *FG-fdd0-closed*
lemmas *fsmult-smult-domain-comm* = *fsmult-smult-comm*
lemmas *GSubspace-Ker* = *RSubmodule-Ker*
lemmas *Ker-Im-iff* = *Ker-Im-iff*
lemmas *Ker0-imp-inj-on* = *Ker0-imp-inj-on*

lemmas *eq-im-imp-diff-in-Ker* = *eq-im-imp-diff-in-Ker*
lemmas *im-submodule* = *im-submodule*
lemmas *fsmultD'* = *aezfun-scalar-mult.fsmultD*[*of smult'*]
lemmas *GmultD'* = *aezfun-scalar-mult.GmultD*[*of smult'*]

lemma *f-map* : $v \in V \Longrightarrow T\ (a\ \sharp\cdot\ v) = a\ \sharp\star\ T\ v$
⟨*proof*⟩

lemma *G-map* : $g \in G \Longrightarrow v \in V \Longrightarrow T\ (g\ *\cdot\ v) = g\ **\ T\ v$
⟨*proof*⟩

lemma *VectorSpaceHom* : *VectorSpaceHom fsmult V fsmult' T*
⟨*proof*⟩

lemmas *distrib-flincomb* = *VectorSpaceHom.distrib-lincomb*[*OF VectorSpaceHom*]

lemma *FGModule-Im* : *FGModule G smult' ImG*
⟨*proof*⟩

lemma *FGModHom-composite-left* :
assumes *FGModuleHom G smult' W smult'' S T* ‘ $V \subseteq W$
shows *FGModuleHom G smult V smult''* $(S \circ T)$
⟨*proof*⟩

lemma *restriction-to-subgroup-is-hom* :
fixes *H* :: *'g set*
assumes *subgrp*: *Group.Subgroup G H*
shows *FGModuleHom H smult V smult' T*
⟨*proof*⟩

lemma *FGModuleHom-restrict0-GSubspace* :
assumes *GSubspace U*
shows *FGModuleHom G smult U smult'* $(T \downarrow U)$
⟨*proof*⟩

lemma *FGModuleHom-fscalar-mul* :
FGModuleHom G smult V smult' $(\lambda v.\ a\ \sharp\star\ T\ v)$
⟨*proof*⟩

end

lemma *GSubspace-eigenspace* :
fixes *e* :: *'f::field*
and *E* :: *'v::ab-group-add set*
and *smult* :: (*'f::field*, *'g::group-add*) *aezfun* ⇒ *'v* ⇒ *'v* (**infixr** · *70*)
assumes *FGModHom*: *FGModuleHom G smult V smult T*
defines *E* : $E \equiv \{v \in V.\ T\ v = $ *aezfun-scalar-mult.fsmult smult e v*$\}$
shows *FGModule.GSubspace G smult V E*
⟨*proof*⟩

5.4.3 Basic facts about endomorphisms

lemma *RModuleEnd-over-group-ring-is-FGModuleEnd* :
 fixes G :: $'g$::*group-add set*
 and *smult* :: ($'f$::*field*, $'g$) *aezfun* $\Rightarrow$ $'v$::*ab-group-add* $\Rightarrow$ $'v$
 assumes G : *Group G* **and** *endo*: *RModuleEnd* (*Group.group-ring G*) *smult V T*
 shows *FGModuleEnd G smult V T*
⟨*proof*⟩

lemma (**in** *FGModule*) *VecEnd-GMap-is-FGModuleEnd* :
 assumes *endo* : *VectorSpaceEnd fsmult V T*
 and *G-map*: $\bigwedge g\ v.\ g \in G \Longrightarrow v \in V \Longrightarrow T\ (g *\cdot v) = g *\cdot (T\ v)$
 shows *FGModuleEnd G smult V T*
⟨*proof*⟩

lemma (**in** *FGModule*) *GEnd-inner-dirsum-el-decomp-nth* :
 $[\![\ \forall\ U{\in}set\ Us.\ GSubspace\ U;\ add\text{-}independentS\ Us;\ n < length\ Us\ ]\!]$
 $\Longrightarrow FGModuleEnd\ G\ smult\ (\bigoplus U{\leftarrow}Us.\ U)\ (\bigoplus Us{\downarrow}n)$
 ⟨*proof*⟩

context *FGModuleEnd*
begin

lemma *RModuleEnd* : *RModuleEnd ActingGroup.group-ring smult V T*
 ⟨*proof*⟩

lemma *VectorSpaceEnd* : *VectorSpaceEnd fsmult V T*
 ⟨*proof*⟩

lemmas *proj-decomp* = *RModuleEnd.proj-decomp*[*OF RModuleEnd*]
lemmas *GSubspace-Ker* = *GSubspace-Ker*
lemmas *FGModuleHom-restrict0-GSubspace* = *FGModuleHom-restrict0-GSubspace*

end

5.4.4 Basic facts about isomorphisms

context *FGModuleIso*
begin

lemmas *VectorSpaceHom* = *VectorSpaceHom*

abbreviation *invT* ≡ (*the-inv-into V T*) ↓ *W*

lemma *RModuleIso* : *RModuleIso FG smult V smult′ T W*
⟨*proof*⟩

lemmas *ImG* = *RModuleIso.ImG*[*OF RModuleIso*]

lemma *FGModuleIso-restrict0-GSubspace* :

assumes *GSubspace U*
shows $FGModuleIso\ G\ smult\ U\ smult'\ (T \downarrow U)\ (T\ `\ U)$
⟨*proof*⟩

lemma *inv* : $FGModuleIso\ G\ smult'\ W\ smult\ invT\ V$
⟨*proof*⟩

lemma *FGModIso-composite-left* :
assumes $FGModuleIso\ G\ smult'\ W\ smult''\ S\ X$
shows $FGModuleIso\ G\ smult\ V\ smult''\ (S \circ T)\ X$
⟨*proof*⟩

lemma *isomorphic-sym* : $FGModule.isomorphic\ G\ smult'\ W\ smult\ V$
⟨*proof*⟩

lemma *isomorphic-trans* :
$FGModule.isomorphic\ G\ smult'\ W\ smult''\ X$
$\Longrightarrow FGModule.isomorphic\ G\ smult\ V\ smult''\ X$
⟨*proof*⟩

lemma *isomorphic-to-zero-left* : $V = 0 \Longrightarrow W = 0$
⟨*proof*⟩

lemma *isomorphic-to-zero-right* : $W = 0 \Longrightarrow V = 0$
⟨*proof*⟩

lemma *isomorphic-to-irr-right'* :
assumes $\bigwedge U.\ FGModule.GSubspace\ G\ smult'\ W\ U \Longrightarrow U = 0 \lor U = W$
shows $\bigwedge U.\ GSubspace\ U \Longrightarrow U = 0 \lor U = V$
⟨*proof*⟩

end

context *FGModule*
begin

lemma *isomorphic-sym* :
$isomorphic\ smult'\ W \Longrightarrow FGModule.isomorphic\ G\ smult'\ W\ smult\ V$
⟨*proof*⟩

lemma *isomorphic-trans* :
$isomorphic\ smult'\ W \Longrightarrow FGModule.isomorphic\ G\ smult'\ W\ smult''\ X$
$\Longrightarrow isomorphic\ smult''\ X$
⟨*proof*⟩

lemma *isomorphic-to-zero-left* : $V = 0 \Longrightarrow isomorphic\ smult'\ W \Longrightarrow W = 0$
⟨*proof*⟩

lemma *isomorphic-to-zero-right* : $isomorphic\ smult'\ 0 \Longrightarrow V = 0$

⟨*proof*⟩

lemma *FGModIso-idhom* : *FGModuleIso G smult V smult* $(id{\downarrow}V)$ *V*
⟨*proof*⟩

lemma *isomorphic-refl* : *isomorphic smult V* ⟨*proof*⟩

end

5.4.5 Hom-sets

definition *FGModuleHomSet* ::
$'g$::*group-add set* ⇒ (($'f$::*field*,$'g$) *aezfun* ⇒ $'v$::*ab-group-add* ⇒ $'v$) ⇒ $'v$ *set*
⇒ (($'f$,$'g$) *aezfun* ⇒ $'w$::*ab-group-add* ⇒ $'w$) ⇒ $'w$ *set*
⇒ ($'v$ ⇒ $'w$) *set*
where *FGModuleHomSet G fgsmult V fgsmult′ W*
≡ {*T*. *FGModuleHom G fgsmult V fgsmult′ T*} ∩ {*T*. *T* ‘ *V* ⊆ *W*}

lemma *FGModuleHomSetI* :
FGModuleHom G fgsmult V fgsmult′ T ⟹ *T* ‘ *V* ⊆ *W*
⟹ *T* ∈ *FGModuleHomSet G fgsmult V fgsmult′ W*
⟨*proof*⟩

lemma *FGModuleHomSetD-FGModuleHom* :
T ∈ *FGModuleHomSet G fgsmult V fgsmult′ W*
⟹ *FGModuleHom G fgsmult V fgsmult′ T*
⟨*proof*⟩

lemma *FGModuleHomSetD-Im* :
T ∈ *FGModuleHomSet G fgsmult V fgsmult′ W* ⟹ *T* ‘ *V* ⊆ *W*
⟨*proof*⟩

context *FGModule*
begin

lemma *FGModuleHomSet-is-Gmaps-in-VectorSpaceHomSet* :
fixes *smult′* :: ($'f$, $'g$) *aezfun* ⇒ $'w$::*ab-group-add* ⇒ $'w$ (**infixr** ⋆ *70*)
and *fsmult′* :: $'f$ ⇒ $'w$ ⇒ $'w$ (**infixr** ♯⋆ *70*)
and *Gmult′* :: $'g$ ⇒ $'w$ ⇒ $'w$ (**infixr** ∗⋆ *70*)
defines *fsmult′* : *fsmult′* ≡ *aezfun-scalar-mult.fsmult smult′*
and *Gmult′* : *Gmult′* ≡ *aezfun-scalar-mult.Gmult smult′*
assumes *FGModW* : *FGModule G smult′ W*
shows *FGModuleHomSet G smult V smult′ W*
= (*VectorSpaceHomSet fsmult V fsmult′ W*)
∩ {*T*. ∀ *g*∈*G*. ∀ *v*∈*V*. *T* (*g* ∗· *v*) = *g* ∗⋆ (*T v*)}
⟨*proof*⟩

lemma *Group-FGModuleHomSet* :
fixes *smult′* :: ($'f$, $'g$) *aezfun* ⇒ $'w$::*ab-group-add* ⇒ $'w$ (**infixr** ⋆ *70*)

and $fsmult'$ :: $'f \Rightarrow 'w \Rightarrow 'w$ (**infixr** $\sharp\star$ *70*)
and $Gmult'$ :: $'g \Rightarrow 'w \Rightarrow 'w$ (**infixr** $**\star$ *70*)
defines $fsmult'$: $fsmult' \equiv$ *aezfun-scalar-mult.fsmult smult'*
and $Gmult'$: $Gmult' \equiv$ *aezfun-scalar-mult.Gmult smult'*
assumes *FGModW* : *FGModule G smult' W*
shows *Group* (*FGModuleHomSet G smult V smult' W*)
⟨*proof*⟩

lemma *Subspace-FGModuleHomSet* :
fixes $smult'$ :: $('f, 'g)$ *aezfun* $\Rightarrow$ $'w$::*ab-group-add* $\Rightarrow 'w$ (**infixr** $\star$ *70*)
and $fsmult'$ :: $'f \Rightarrow 'w \Rightarrow 'w$ (**infixr** $\sharp\star$ *70*)
and $Gmult'$ :: $'g \Rightarrow 'w \Rightarrow 'w$ (**infixr** $**\star$ *70*)
and *hom-fsmult* :: $'f \Rightarrow ('v \Rightarrow 'w) \Rightarrow ('v \Rightarrow 'w)$ (**infixr** $\sharp\star\cdot$ *70*)
defines $fsmult'$: $fsmult' \equiv$ *aezfun-scalar-mult.fsmult smult'*
and $Gmult'$: $Gmult' \equiv$ *aezfun-scalar-mult.Gmult smult'*
defines *hom-fsmult* : *hom-fsmult* $\equiv \lambda a\ T\ v.\ a \sharp\star T\ v$
assumes *FGModW* : *FGModule G smult' W*
shows *VectorSpace.Subspace hom-fsmult*
(*VectorSpaceHomSet fsmult V fsmult' W*)
(*FGModuleHomSet G smult V smult' W*)
⟨*proof*⟩

lemma *VectorSpace-FGModuleHomSet* :
fixes $smult'$ :: $('f, 'g)$ *aezfun* $\Rightarrow$ $'w$::*ab-group-add* $\Rightarrow 'w$ (**infixr** $\star$ *70*)
and $fsmult'$ :: $'f \Rightarrow 'w \Rightarrow 'w$ (**infixr** $\sharp\star$ *70*)
and *hom-fsmult* :: $'f \Rightarrow ('v \Rightarrow 'w) \Rightarrow ('v \Rightarrow 'w)$ (**infixr** $\sharp\star\cdot$ *70*)
defines $fsmult' \equiv$ *aezfun-scalar-mult.fsmult smult'*
defines *hom-fsmult* $\equiv \lambda a\ T\ v.\ a \sharp\star T\ v$
assumes *FGModule G smult' W*
shows *VectorSpace hom-fsmult* (*FGModuleHomSet G smult V smult' W*)
⟨*proof*⟩

end

5.5 Induced modules

5.5.1 Additive function spaces

definition *addfunset* ::
$'a$::*monoid-add set* $\Rightarrow$ $'m$::*monoid-add set* $\Rightarrow ('a \Rightarrow 'm)$ *set*
where *addfunset A M* $\equiv \{f.\ supp\ f \subseteq A \wedge range\ f \subseteq M$
$\wedge\ (\forall x\in A.\ \forall y\in A.\ f\ (x+y) = f\ x + f\ y)\ \}$

lemma *addfunsetI* :
$[\![\ supp\ f \subseteq A;\ range\ f \subseteq M;\ \forall x\in A.\ \forall y\in A.\ f\ (x+y) = f\ x + f\ y\ ]\!]$
$\Longrightarrow f \in addfunset\ A\ M$
⟨*proof*⟩

lemma *addfunsetD-supp* : $f \in addfunset\ A\ M \Longrightarrow supp\ \ f \subseteq A$
⟨*proof*⟩

lemma *addfunsetD-range* : $f \in$ *addfunset* $A\ M \Longrightarrow$ *range* $f \subseteq M$
⟨*proof*⟩

lemma *addfunsetD-range'* : $f \in$ *addfunset* $A\ M \Longrightarrow f\,x \in M$
⟨*proof*⟩

lemma *addfunsetD-add* :
$[\![\ f \in$ *addfunset* $A\ M;\ x{\in}A;\ y{\in}A\ ]\!] \Longrightarrow f\ (x{+}y) = f\,x + f\,y$
⟨*proof*⟩

lemma *addfunset0* : *addfunset* A (0::'m::*monoid-add set*) = 0
⟨*proof*⟩

lemma *Group-addfunset* :
fixes M::'g::*ab-group-add set*
assumes *Group M*
shows *Group* (*addfunset R M*)
⟨*proof*⟩

5.5.2 Spaces of functions which transform under scalar multiplication by almost-everywhere-zero functions

context *aezfun-scalar-mult*
begin

definition *smultfunset* :: 'g *set* ⇒ ('r,'g) *aezfun set* ⇒ (('r,'g) *aezfun* ⇒ 'v) *set*
where *smultfunset G FH* ≡ $\{f.\ (\forall\, a{::}'r.\ \forall\, g{\in}G.\ \forall\, x{\in}FH.$
$f\ (\ a\ \delta\delta\ g * x\ ) = (a\ \delta\delta\ g) \cdot (f\,x))\}$

lemma *smultfunsetD* :
$[\![f \in$ *smultfunset* $G\ FH;\ g{\in}G;\ x{\in}FH\ ]\!] \Longrightarrow f\ (\ a\ \delta\delta\ g * x\ ) = (a\ \delta\delta\ g) \cdot (f\,x)$
⟨*proof*⟩

lemma *smultfunsetI* :
$\forall\, a{::}'r.\ \forall\, g{\in}G.\ \forall\, x{\in}FH.\ f\ (\ a\ \delta\delta\ g * x\ ) = (a\ \delta\delta\ g) \cdot (f\,x)$
$\Longrightarrow f \in$ *smultfunset* $G\ FH$
⟨*proof*⟩

end

5.5.3 General induced spaces of functions on a group ring

context *aezfun-scalar-mult*
begin

definition *indspace* ::
'g *set* ⇒ ('r,'g) *aezfun set* ⇒ 'v *set* ⇒ (('r,'g) *aezfun* ⇒ 'v) *set*
where *indspace G FH V* = *addfunset FH V* ∩ *smultfunset G FH*

lemma *indspaceD* :
$f \in indspace\ G\ FH\ V \Longrightarrow f \in addfunset\ FH\ V \cap smultfunset\ G\ FH$
⟨*proof*⟩

lemma *indspaceD-supp* : $f \in indspace\ G\ FH\ V \Longrightarrow supp\ \ f \subseteq FH$
⟨*proof*⟩

lemma *indspaceD-supp′* : $f \in indspace\ G\ FH\ V \Longrightarrow x \notin FH \Longrightarrow f\, x = 0$
⟨*proof*⟩

lemma *indspaceD-range* : $f \in indspace\ G\ FH\ V \Longrightarrow range\ f \subseteq V$
⟨*proof*⟩

lemma *indspaceD-range′*: $f \in indspace\ G\ FH\ V \Longrightarrow f\, x \in V$
⟨*proof*⟩

lemma *indspaceD-add* :
$[\![\ f \in indspace\ G\ FH\ V;\ x{\in}FH;\ y{\in}FH\ ]\!] \Longrightarrow f\ (x{+}y) = f\, x + f\, y$
⟨*proof*⟩

lemma *indspaceD-transform* :
$[\![f \in indspace\ G\ FH\ V;\ g{\in}G;\ x{\in}FH\ ]\!] \Longrightarrow f\ (\ a\ \delta\delta\ g * x\ ) = (a\ \delta\delta\ g) \cdot (f\, x)$
⟨*proof*⟩

lemma *indspaceI* :
$f \in addfunset\ FH\ V \Longrightarrow f \in smultfunset\ G\ FH \Longrightarrow f \in indspace\ G\ FH\ V$
⟨*proof*⟩

lemma *indspaceI′* :
$[\![\ supp\ f \subseteq FH;\ range\ f \subseteq V;\ \forall\, x{\in}FH.\ \forall\, y{\in}FH.\ f\ (x{+}y) = f\, x + f\, y;$
$\forall\, a{::}'r.\ \forall\, g{\in}G.\ \forall\, x{\in}FH.\ f\ (\ a\ \delta\delta\ g * x\ ) = (a\ \delta\delta\ g) \cdot (f\, x)\ ]\!]$
$\Longrightarrow f \in indspace\ G\ FH\ V$
⟨*proof*⟩

lemma *mono-indspace* : $mono\ (indspace\ G\ FH)$
⟨*proof*⟩

end

context *FGModule*
begin

lemma *zero-transforms* : $0 \in smultfunset\ G\ FH$
⟨*proof*⟩

lemma *indspace0* : $indspace\ G\ FH\ 0 = 0$
⟨*proof*⟩

lemma *Group-indspace* :

assumes *Ring1 FH*
shows *Group* (*indspace G FH V*)
⟨*proof*⟩

end

5.5.4 The right regular action

context *Ring1*
begin

definition *rightreg-scalar-mult* ::
$'r::ring\text{-}1 \Rightarrow ('r \Rightarrow 'm::ab\text{-}group\text{-}add) \Rightarrow ('r \Rightarrow 'm)$ (**infixr** ¤ *70*)
where $rightreg\text{-}scalar\text{-}mult\ r\ f = (\lambda x.\ if\ x \in R\ then\ f\ (x*r)\ else\ 0)$

lemma *rightreg-scalar-multD1* : $x \in R \Longrightarrow (r ¤ f)\ x = f\ (x*r)$
⟨*proof*⟩

lemma *rightreg-scalar-multD2* : $x \notin R \Longrightarrow (r ¤ f)\ x = 0$
⟨*proof*⟩

lemma *rrsmult-supp* : $supp\ (r ¤ f) \subseteq R$
⟨*proof*⟩

lemma *rrsmult-range* : $range\ (r ¤ f) \subseteq \{0\} \cup range\ f$
⟨*proof*⟩

lemma *rrsmult-distrib-left* : $r ¤ (f + g) = r ¤ f + r ¤ g$
⟨*proof*⟩

lemma *rrsmult-distrib-right* :
assumes $\bigwedge x\ y.\ x \in R \Longrightarrow y \in R \Longrightarrow f\ (x+y) = f\ x + f\ y\ r \in R\ s \in R$
shows $(r + s) ¤ f = r ¤ f + s ¤ f$
⟨*proof*⟩

lemma *RModule-addfunset* :
fixes $M::'g::ab\text{-}group\text{-}add\ set$
assumes *Group M*
shows *RModule R rightreg-scalar-mult* (*addfunset R M*)
⟨*proof*⟩

end

5.5.5 Locale and basic facts

In the following locale, G is a subgroup of H, V is a module over the group ring for G, and the induced space *indV* will be shown to be a module over the group ring for H under the right regular scalar multiplication *rrsmult*.

locale *InducedFHModule* = *Supgroup?*: *Group H*

\+ *BaseFGMod?* : *FGModule G smult V*
\+ *induced-smult?*: *aezfun-scalar-mult rrsmult*
for *H* :: *'g::group-add set*
and *G* :: *'g set*
and *FG* :: *('f::field, 'g) aezfun set*
and *smult* :: *('f, 'g) aezfun ⇒ 'v::ab-group-add ⇒ 'v* (**infixl** · *70*)
and *V* :: *'v set*
and *rrsmult* :: *('f,'g) aezfun ⇒ (('f,'g) aezfun ⇒ 'v) ⇒ (('f,'g) aezfun ⇒ 'v)* (**infixl** ¤ *70*)
\+ **fixes** *FH* :: *('f, 'g) aezfun set*
and *indV* :: *(('f, 'g) aezfun ⇒ 'v) set*
defines *FH* : *FH ≡ Supgroup.group-ring*
and *indV* : *indV ≡ BaseFGMod.indspace G FH V*
assumes *rrsmult* : *rrsmult = Ring1.rightreg-scalar-mult FH*
and *Subgroup*: *Supgroup.Subgroup G*
begin

abbreviation *indfsmult* ::
'f ⇒ (('f, 'g) aezfun ⇒ 'v) ⇒ (('f, 'g) aezfun ⇒ 'v) (**infixl** ¤¤ *70*)
where *indfsmult ≡ induced-smult.fsmult*
abbreviation *indflincomb* ::
'f list ⇒ (('f, 'g) aezfun ⇒ 'v) list ⇒ (('f, 'g) aezfun ⇒ 'v) (**infixl** •¤¤ *70*)
where *indflincomb ≡ induced-smult.flincomb*
abbreviation *Hmult* ::
'g ⇒ (('f, 'g) aezfun ⇒ 'v) ⇒ (('f, 'g) aezfun ⇒ 'v) (**infixl** ∗¤ *70*)
where *Hmult ≡ induced-smult.Gmult*

lemma *Ring1-FH* : *Ring1 FH* ⟨*proof*⟩

lemma *FG-subring-FH* : *Ring1.Subring1 FH BaseFGMod.FG*
⟨*proof*⟩

lemma *rrsmultD1* : $x \in FH \Longrightarrow (r ¤ f)\ x = f\ (x*r)$
⟨*proof*⟩

lemma *rrsmultD2* : $x \notin FH \Longrightarrow (r ¤ f)\ x = 0$
⟨*proof*⟩

lemma *rrsmult-supp* : $supp\ (r ¤ f) \subseteq FH$
⟨*proof*⟩

lemma *rrsmult-range* : $range\ (r ¤ f) \subseteq \{0\} \cup range\ f$
⟨*proof*⟩

lemma *FHModule-addfunset* : *FGModule H rrsmult (addfunset FH V)*
⟨*proof*⟩

lemma *FHSubmodule-indspace* :
FGModule.FGSubmodule H rrsmult (addfunset FH V) indV

$\langle proof \rangle$

lemma *FHModule-indspace* : *FGModule H rrsmult indV*
$\langle proof \rangle$

lemmas *fVectorSpace-indspace* = *FGModule.fVectorSpace*[*OF FHModule-indspace*]
lemmas *restriction-is-FGModule*
= *FGModule.restriction-to-subgroup-is-module*[*OF FHModule-indspace*]

definition *induced-vector* :: $'v \Rightarrow (('f, 'g)\ aezfun \Rightarrow 'v)$
where *induced-vector* $v \equiv$ (*if* $v \in V$
then (λy. *if* $y \in FH$ *then* (*FG-proj* y) $\cdot$ v *else* 0) *else* 0)

lemma *induced-vector-apply1* :
$v \in V \Longrightarrow x \in FH \Longrightarrow$ *induced-vector* $v\ x$ = (*FG-proj* x) $\cdot$ v
$\langle proof \rangle$

lemma *induced-vector-apply2* : $v \in V \Longrightarrow x \notin FH \Longrightarrow$ *induced-vector* $v\ x = 0$
$\langle proof \rangle$

lemma *induced-vector-indV* :
assumes v: $v \in V$
shows *induced-vector* $v \in indV$
$\langle proof \rangle$

lemma *induced-vector-additive* :
$v \in V \Longrightarrow v' \in V$
$\Longrightarrow$ *induced-vector* $(v+v')$ = *induced-vector* v + *induced-vector* v'
$\langle proof \rangle$

lemma *hom-induced-vector* : *FGModuleHom G smult V rrsmult induced-vector*
$\langle proof \rangle$

lemma *indspace-sum-list-fddh*:
$[\![$ *fhs* $\neq$ []; *set* (*map snd fhs*) $\subseteq H$; $f \in indV$ $]\!]$
$\Longrightarrow f\ (\sum (a,h) \leftarrow fhs.\ a\ \delta\delta\ h) = (\sum (a,h) \leftarrow fhs.\ f\ (a\ \delta\delta\ h))$
$\langle proof \rangle$

lemma *induced-fsmult-conv-fsmult-1ddh* :
$f \in indV \Longrightarrow h \in H \Longrightarrow (r\ ¤¤\ f)\ (1\ \delta\delta\ h) = r\ \sharp\cdot\ (f\ (1\ \delta\delta\ h))$
$\langle proof \rangle$

lemma *indspace-el-eq-on-1ddh-imp-eq-on-rddh* :
assumes $HmodG \subseteq H$ $H = (\bigcup h \in HmodG.\ G + \{h\})$ $f \in indV$ $f' \in indV$
$\forall h \in HmodG.\ f\ (1\ \delta\delta\ h) = f'\ (1\ \delta\delta\ h)$ $h \in H$
shows $f\ (r\ \delta\delta\ h) = f'\ (r\ \delta\delta\ h)$
$\langle proof \rangle$

lemma *indspace-el-eq* :

assumes $HmodG \subseteq H$ $H = (\bigcup h{\in}HmodG.\ G + \{h\})$ $f \in indV$ $f' \in indV$
$\forall h{\in}HmodG.\ f\ (1\ \delta\delta\ h) = f'\ (1\ \delta\delta\ h)$
shows $f = f'$
⟨*proof*⟩

lemma *indspace-el-eq'* :
assumes $set\ hs \subseteq H$ $H = (\bigcup h{\in}set\ hs.\ G + \{h\})$ $f \in indV$ $f' \in indV$
$\forall i{<}length\ hs.\ f\ (1\ \delta\delta\ (hs!i)) = f'\ (1\ \delta\delta\ (hs!i))$
shows $f = f'$
⟨*proof*⟩

end

6 Representations of Finite Groups

6.1 Locale and basic facts

Define a group representation to be a module over the group ring that is finite-dimensional as a vector space. The only restriction on the characteristic of the field is that it does not divide the order of the group. Also, we don't explicitly assume that the group is finite; instead, the *good-char* assumption implies that the cardinality of G is not zero, which implies G is finite. (See lemma *good-card-imp-finite*.)

locale *FinGroupRepresentation* = *FGModule G smult V*
for G :: *'g::group-add set*
and *smult* :: *('f::field, 'g) aezfun* ⇒ *'v::ab-group-add* ⇒ *'v* (**infixl** · *70*)
and V :: *'v set*
\+
assumes *good-char*: *of-nat* (*card G*) ≠ (*0::'f*)
and *findim* : *fscalar-mult.findim fsmult V*

lemma (**in** *Group*) *trivial-FinGroupRep* :
fixes *smult* :: *('f::field, 'g) aezfun* ⇒ *'v::ab-group-add* ⇒ *'v*
assumes *good-char* : *of-nat* (*card G*) ≠ (*0::'f*)
and *smult-zero* : ∀ *a*∈*group-ring*. *smult a* (*0::'v*) = *0*
shows *FinGroupRepresentation G smult* (*0::'v set*)
⟨*proof*⟩

context *FinGroupRepresentation*
begin

abbreviation *ordG* :: *'f* **where** *ordG* ≡ *of-nat* (*card G*)
abbreviation *GRepHom* ≡ *FGModuleHom G smult V*
abbreviation *GRepIso* ≡ *FGModuleIso G smult V*
abbreviation *GRepEnd* ≡ *FGModuleEnd G smult V*

lemmas *zero-closed* = *zero-closed*
lemmas *Group* = *Group*

lemmas *GSubmodule-GSpan-single* = *RSubmodule-RSpan-single*
lemmas *GSpan-single-nonzero* = *RSpan-single-nonzero*

lemma *finiteG*: *finite G*
⟨*proof*⟩

lemma *FinDimVectorSpace*: *FinDimVectorSpace fsmult V*
⟨*proof*⟩

lemma *GSubspace-is-FinGroupRep* :
assumes *GSubspace U*
shows *FinGroupRepresentation G smult U*
⟨*proof*⟩

lemma *isomorphic-imp-GRep* :
assumes *isomorphic smult′ W*
shows *FinGroupRepresentation G smult′ W*
⟨*proof*⟩

end

6.2 Irreducible representations

locale *IrrFinGroupRepresentation* = *FinGroupRepresentation*
+ **assumes** *irr*: *GSubspace U* ⟹ *U* = *0* ∨ *U* = *V*
begin

lemmas *AbGroup* = *AbGroup*

lemma *zero-isomorphic-to-FG-zero* :
assumes *V* = *0*
shows *isomorphic* (∗) (*0*::(′*b*,′*a*) *aezfun set*)
⟨*proof*⟩

lemma *eq-GSpan-single* : *v* ∈ *V* ⟹ *v* ≠ *0* ⟹ *V* = *GSpan* [*v*]
⟨*proof*⟩

end

lemma (**in** *Group*) *trivial-IrrFinGroupRepI* :
fixes *smult* :: (′*f*::*field*, ′*g*) *aezfun* ⇒ ′*v*::*ab-group-add* ⇒ ′*v*
assumes *of-nat* (*card G*) ≠ (*0*::′*f*)
and ∀ *a*∈*group-ring*. *smult a* (*0*::′*v*) = *0*
shows *IrrFinGroupRepresentation G smult* (*0*::′*v set*)
⟨*proof*⟩

lemma (**in** *Group*) *trivial-IrrFinGroupRepresentation-in-FG* :
of-nat (*card G*) ≠ (*0*::′*f*::*field*)
⟹ *IrrFinGroupRepresentation G* (∗) (*0*::(′*f*,′*g*) *aezfun set*)

⟨*proof*⟩

context *FinGroupRepresentation*
begin

lemma *IrrFinGroupRep-trivialGSubspace* :
IrrFinGroupRepresentation G smult $(0::'v\ set)$
⟨*proof*⟩

lemma *IrrI* :
assumes $\bigwedge U.$ *FGModule.GSubspace G smult V U* $\Longrightarrow U = 0 \vee U = V$
shows *IrrFinGroupRepresentation G smult V*
⟨*proof*⟩

lemma *notIrr* :
$\neg$ *IrrFinGroupRepresentation G smult V*
$\Longrightarrow \exists\, U.$ *GSubspace* $U \wedge U \neq 0 \wedge U \neq V$
⟨*proof*⟩

end

6.3 Maschke's theorem

6.3.1 Averaged projection onto a G-subspace

context *FinGroupRepresentation*
begin

lemma *avg-proj-eq-id-on-right* :
assumes *VectorSpace fsmult W add-independentS* $[W,V]$ $v \in V$
defines $P : P \equiv (\bigoplus[W,V]\downarrow 1)$
defines CP: $CP \equiv (\lambda g.\ Gmult\ (-\ g) \circ P \circ Gmult\ g)$
defines $T : T \equiv fsmult\ (1/ordG) \circ (\sum g{\in}G.\ CP\ g)$
shows $T\ v = v$
⟨*proof*⟩

lemma *avg-proj-onto-right* :
assumes *VectorSpace fsmult W GSubspace U add-independentS* $[W,U]$
$V = W \oplus U$
defines $P : P \equiv (\bigoplus[W,U]\downarrow 1)$
defines CP: $CP \equiv (\lambda g.\ Gmult\ (-\ g) \circ P \circ Gmult\ g)$
defines $T : T \equiv fsmult\ (1/ordG) \circ (\sum g{\in}G.\ CP\ g)$
shows $T\ `\ V = U$
⟨*proof*⟩

lemma *FGModuleEnd-avg-proj-right* :
assumes *fSubspace W GSubspace U add-independentS* $[W,U]$ $V = W \oplus U$
defines $P : P \equiv (\bigoplus[W,U]\downarrow 1)$
defines CP: $CP \equiv (\lambda g.\ Gmult\ (-\ g) \circ P \circ Gmult\ g)$
defines $T : T \equiv (fsmult\ (1/ordG) \circ (\sum g{\in}G.\ CP\ g))\downarrow V$

shows *FGModuleEnd G smult V T*
⟨*proof*⟩

lemma *avg-proj-is-proj-right* :
assumes *VectorSpace fsmult W GSubspace U add-independentS* $[W,U]$
$V = W \oplus U$ $v \in V$
defines $P : P \equiv (\bigoplus [W,U]\downarrow 1)$
defines CP: $CP \equiv (\lambda g.\ Gmult\ (-\ g) \circ P \circ Gmult\ g)$
defines $T : T \equiv fsmult\ (1/ordG) \circ (\sum g{\in}G.\ CP\ g)$
shows $T\ (T\ v) = T\ v$
⟨*proof*⟩

end

6.3.2 The theorem

context *FinGroupRepresentation*
begin

theorem *Maschke* :
assumes *GSubspace U*
shows $\exists\, W.\ GSubspace\ W \wedge V = W \oplus U$
⟨*proof*⟩

corollary *Maschke-proper* :
assumes *GSubspace U* $U \neq 0$ $U \neq V$
shows $\exists\, W.\ GSubspace\ W \wedge W \neq 0 \wedge W \neq V \wedge V = W \oplus U$
⟨*proof*⟩

end

6.3.3 Consequence: complete reducibility

lemma (**in** *FinGroupRepresentation*) *notIrr-decompose* :
$\neg$ *IrrFinGroupRepresentation G smult V*
$\Longrightarrow \exists\, U\ W.\ GSubspace\ U \wedge U \neq 0 \wedge U \neq V \wedge GSubspace\ W \wedge W \neq 0$
$\wedge\ W \neq V \wedge V = U \oplus W$
⟨*proof*⟩

In the following decomposition lemma, we do not need to explicitly include the condition that all *U* in *set Us* are subsets of *V*. (See lemma *AbGroup-subset-inner-dirsum.*)

lemma *FinGroupRepresentation-reducible′* :
fixes *n::nat*
shows $\bigwedge V.$ *FinGroupRepresentation G fgsmult V*
$\wedge\ n = FGModule.fdim\ fgsmult\ V$
$\Longrightarrow (\exists\, Us.\ Ball\ (set\ Us)\ (IrrFinGroupRepresentation\ G\ fgsmult)$
$\wedge\ (0 \notin set\ Us) \wedge V = (\bigoplus U{\leftarrow}Us.\ U)\ )$
⟨*proof*⟩

theorem (**in** *FinGroupRepresentation*) *reducible* :
$\exists$ *Us*. ($\forall$ *U*$\in$*set Us*. *IrrFinGroupRepresentation G smult U*) $\wedge$ (*0* $\notin$ *set Us*)
$\wedge$ *V* = ($\bigoplus$ *U*$\leftarrow$*Us*. *U*)
⟨*proof*⟩

6.3.4 Consequence: decomposition relative to a homomomorphism

lemma (**in** *FinGroupRepresentation*) *GRepHom-decomp* :
fixes *T* :: $'v \Rightarrow 'w$::*ab-group-add*
defines *KerT* : *KerT* $\equiv$ (*ker T* $\cap$ *V*)
assumes *hom* : *GRepHom smult′ T* **and** *nonzero*: *V* $\neq$ *0*
shows $\exists$ *U*. *GSubspace U* $\wedge$ *V* = *U* $\oplus$ *KerT*
$\wedge$ *FGModule.isomorphic G smult U smult′* (*T* ‘ *V*)
⟨*proof*⟩

6.4 Schur's lemma

lemma (**in** *IrrFinGroupRepresentation*) *Schur-Ker* :
GRepHom smult′ T $\Longrightarrow$ *T* ‘ *V* $\neq$ *0* $\Longrightarrow$ *inj-on T V*
⟨*proof*⟩

lemma (**in** *FinGroupRepresentation*) *Schur-Im* :
assumes *IrrFinGroupRepresentation G smult′ W GRepHom smult′ T*
T ‘ *V* $\subseteq$ *W*
T ‘ *V* $\neq$ *0*
shows *T* ‘ *V* = *W*
⟨*proof*⟩

theorem (**in** *IrrFinGroupRepresentation*) *Schur1* :
assumes *IrrFinGroupRepresentation G smult′ W*
GRepHom smult′ T T ‘ *V* $\subseteq$ *W T* ‘ *V* $\neq$ *0*
shows *GRepIso smult′ T W*
⟨*proof*⟩

theorem (**in** *IrrFinGroupRepresentation*) *Schur2* :
assumes *GRep* : *GRepEnd T*
and *fdim* : *fdim* > *0*
and *alg-closed*: $\bigwedge$*p*::′*b poly*. *degree p* > *0* $\Longrightarrow$ $\exists$ *c*. *poly p c* = *0*
shows $\exists$ *c*. $\forall$ *v* $\in$ *V*. *T v* = *c* ♯· *v*
⟨*proof*⟩

6.5 The group ring as a representation space

6.5.1 The group ring is a representation space

lemma (**in** *Group*) *FGModule-FG* :
defines *FG*: *FG* $\equiv$ *group-ring* :: (′*f*::*field*, ′*g*) *aezfun set*
shows *FGModule G* ($*$) *FG*
⟨*proof*⟩

theorem (**in** *Group*) *FinGroupRepresentation-FG* :
defines *FG*: $FG \equiv$ *group-ring* :: ($'f$::*field*, $'g$) *aezfun set*
assumes *good-char*: *of-nat* (*card G*) $\neq$ (0::$'f$)
shows *FinGroupRepresentation G* ($*$) *FG*
⟨*proof*⟩

lemma (**in** *FinGroupRepresentation*) *FinGroupRepresentation-FG* :
FinGroupRepresentation G ($*$) *FG*
⟨*proof*⟩

lemma (**in** *Group*) *FG-reducible* :
assumes *of-nat* (*card G*) $\neq$ (0::$'f$::*field*)
shows $\exists$ *Us*::($'f$,$'g$) *aezfun set list*.
($\forall$ $U \in$ *set Us*. *IrrFinGroupRepresentation G* ($*$) *U*) $\wedge$ $0 \notin$ *set Us*
$\wedge$ *group-ring* = ($\bigoplus$ $U \leftarrow Us$. *U*)
⟨*proof*⟩

6.5.2 Irreducible representations are constituents of the group ring

lemma (**in** *FGModuleIso*) *isomorphic-to-irr-right* :
assumes *IrrFinGroupRepresentation G smult′ W*
shows *IrrFinGroupRepresentation G smult V*
⟨*proof*⟩

lemma (**in** *FinGroupRepresentation*) *IrrGSubspace-iso-constituent* :
assumes *nonzero* : $V \neq 0$
and *subsp* : $W \subseteq V$ $W \neq 0$ *IrrFinGroupRepresentation G smult W*
and *V-decomp*: $\forall$ $U \in$ *set Us*. *IrrFinGroupRepresentation G smult U*
$0 \notin$ *set Us* V = ($\bigoplus$ $U \leftarrow Us$. *U*)
shows $\exists$ $U \in$ *set Us*. *FGModule.isomorphic G smult W smult U*
⟨*proof*⟩

theorem (**in** *IrrFinGroupRepresentation*) *iso-FG-constituent* :
assumes *nonzero* : $V \neq 0$
and *FG-decomp*: $\forall$ $U \in$ *set Us*. *IrrFinGroupRepresentation G* ($*$) *U*
$0 \notin$ *set Us* FG = ($\bigoplus$ $U \leftarrow Us$. *U*)
shows $\exists$ $U \in$ *set Us*. *isomorphic* ($*$) *U*
⟨*proof*⟩

6.6 Isomorphism classes of irreducible representations

We have already demonstrated that the relation *FGModule.isomorphic* is reflexive (lemma *FGModule.isomorphic-refl*), symmetric (lemma *FGModule.isomorphic-sym*), and transitive (lemma *FGModule.isomorphic-trans*). In this section, we provide a finite set of representatives for the resulting isomorphism classes of irreducible representations.

context *Group*
begin

primrec *remisodups* :: ('*f*::*field*,'*g*) *aezfun set list* ⇒ ('*f*,'*g*) *aezfun set list* **where**
  remisodups [] = []
| *remisodups* (*U* # *Us*) = (*if*
  (∃ *W*∈*set Us*. *FGModule.isomorphic G* (∗) *U* (∗) *W*)
  then remisodups Us else U # *remisodups Us*)

lemma *set-remisodups* : *set* (*remisodups Us*) ⊆ *set Us*
  ⟨*proof*⟩

lemma *isodistinct-remisodups* :
  ⟦ ∀ *U*∈*set Us*. *FGModule G* (∗) *U*; *V* ∈ *set* (*remisodups Us*);
  W ∈ *set* (*remisodups Us*); *V* ≠ *W* ⟧
  ⟹ ¬ (*FGModule.isomorphic G* (∗) *V* (∗) *W*)
⟨*proof*⟩

definition *FG-constituents* ≡ *SOME Us*.
  (∀ *U*∈*set Us*. *IrrFinGroupRepresentation G* (∗) *U*)
  ∧ *0* ∉ *set Us* ∧ *group-ring* = (⨁ *U*←*Us*. *U*)

lemma *FG-constituents-irr* :
  of-nat (*card G*) ≠ (*0*::'*f*::*field*)
  ⟹ ∀ *U*∈*set* (*FG-constituents*::('*f*,'*g*) *aezfun set list*).
  IrrFinGroupRepresentation G (∗) *U*
  ⟨*proof*⟩

lemma *FG-consitutents-n0*:
  of-nat (*card G*) ≠ (*0*::'*f*::*field*)
  ⟹ *0* ∉ *set* (*FG-constituents*::('*f*,'*g*) *aezfun set list*)
  ⟨*proof*⟩

lemma *FG-constituents-constituents* :
  of-nat (*card G*) ≠ (*0*::'*f*::*field*)
  ⟹ (*group-ring*::('*f*,'*g*) *aezfun set*) = (⨁ *U*←*FG-constituents*. *U*)
  ⟨*proof*⟩

definition *GIrrRep-repset* ≡ *0* ∪ *set* (*remisodups FG-constituents*)

lemma *finite-GIrrRep-repset* : *finite GIrrRep-repset*
  ⟨*proof*⟩

lemma *all-irr-GIrrRep-repset* :
  assumes *of-nat* (*card G*) ≠ (*0*::'*f*::*field*)
  shows ∀ *U*∈(*GIrrRep-repset*::('*f*,'*g*) *aezfun set set*).
  IrrFinGroupRepresentation G (∗) *U*
⟨*proof*⟩

lemma *isodistinct-GIrrRep-repset* :
  defines *GIRRS* ≡ *GIrrRep-repset* :: ('*f*::*field*,'*g*) *aezfun set set*
  assumes *of-nat* (*card G*) ≠ (*0*::'*f*) *V* ∈ *GIRRS W* ∈ *GIRRS V* ≠ *W*
  shows ¬ (*FGModule.isomorphic G* (∗) *V* (∗) *W*)
⟨*proof*⟩

end

lemma (**in** *FGModule*) *iso-in-list-imp-iso-in-remisodups* :
  ∃ *U*∈*set Us*. *isomorphic* (∗) *U*
    ⟹ ∃ *U*∈*set* (*ActingGroup.remisodups Us*). *isomorphic* (∗) *U*
⟨*proof*⟩

lemma (**in** *IrrFinGroupRepresentation*) *iso-to-GIrrRep-rep* :
  ∃ *U*∈*ActingGroup.GIrrRep-repset*. *isomorphic* (∗) *U*
  ⟨*proof*⟩

theorem (**in** *Group*) *iso-class-reps* :
  defines *GIRRS* ≡ *GIrrRep-repset* :: ('*f*::*field*,'*g*) *aezfun set set*
  assumes *of-nat* (*card G*) ≠ (*0*::'*f*)
  shows *finite GIRRS*
    ∀ *U*∈*GIRRS*. *IrrFinGroupRepresentation G* (∗) *U*
    ⋀*U W*. ⟦ *U* ∈ *GIRRS*; *W* ∈ *GIRRS*; *U* ≠ *W* ⟧
      ⟹ ¬ (*FGModule.isomorphic G* (∗) *U* (∗) *W*)
    ⋀*fgsmult V*. *IrrFinGroupRepresentation G fgsmult V*
      ⟹ ∃ *U*∈*GIRRS*. *FGModule.isomorphic G fgsmult V* (∗) *U*
  ⟨*proof*⟩

6.7 Induced representations

6.7.1 Locale and basic facts

locale *InducedFinGroupRepresentation* = *Supgroup?*: *Group H*
+ *BaseRep?*: *FinGroupRepresentation G smult V*
+ *induced-smult?*: *aezfun-scalar-mult rrsmult*
  for *H* :: '*g*::*group-add set*
  and *G* :: '*g set*
  and *smult* :: ('*f*::*field*, '*g*) *aezfun* ⇒ '*v*::*ab-group-add* ⇒ '*v* (**infixl** · *70*)
  and *V* :: '*v set*
  and *rrsmult* :: ('*f*,'*g*) *aezfun* ⇒ (('*f*,'*g*) *aezfun* ⇒'*v*)
    ⇒ (('*f*,'*g*) *aezfun* ⇒'*v*) (**infixl** ¤ *70*)
+ **fixes** *FH* :: ('*f*, '*g*) *aezfun set*
  and *indV* :: (('*f*, '*g*) *aezfun* ⇒ '*v*) *set*
  defines *FH* : *FH* ≡ *Supgroup.group-ring*
  and *indV* : *indV* ≡ *BaseRep.indspace G FH V*
  assumes *rrsmult* : *rrsmult* = *Ring1.rightreg-scalar-mult FH*
  and *good-ordSupgrp*: *of-nat* (*card H*) ≠ (*0*::'*f*)
  and *Subgroup* : *Supgroup.Subgroup G*

sublocale *InducedFinGroupRepresentation* < *InducedFHModule*

⟨*proof*⟩

context *InducedFinGroupRepresentation*
begin

abbreviation *ordH* :: *'f* **where** *ordH* ≡ *of-nat* (*card H*)
abbreviation *is-Vfbasis* ≡ *fbasis-for V*
abbreviation *GRepHomSet* ≡ *FGModuleHomSet G smult V*
abbreviation *HRepHom* ≡ *FGModuleHom H rrsmult indV*
abbreviation *HRepHomSet* ≡ *FGModuleHomSet H rrsmult indV*

lemma *finiteSupgroup*: *finite H*
⟨*proof*⟩

lemma *FinGroupSupgroup* : *FinGroup H*
⟨*proof*⟩

lemmas *fVectorSpace* = *fVectorSpace*
lemmas *FinDimVectorSpace* = *FinDimVectorSpace*
lemmas *ex-rcoset-replist-hd0*
= *FinGroup.ex-rcoset-replist-hd0*[*OF FinGroupSupgroup Subgroup*]

end

6.7.2 A basis for the induced space

context *InducedFinGroupRepresentation*
begin

abbreviation *negHorbit-list* ≡ *induced-smult.negGorbit-list*

lemmas *ex-rcoset-replist*
= *FinGroup.ex-rcoset-replist*[*OF FinGroupSupgroup Subgroup*]
lemmas *length-negHorbit-list* = *induced-smult.length-negGorbit-list*
lemmas *length-negHorbit-list-sublist* = *induced-smult.length-negGorbit-list-sublist*
lemmas *negHorbit-list-indV* = *FGModule.negGorbit-list-V*[*OF FHModule-indspace*]

lemma *flincomb-Horbit-induced-veclist-reduce* :
fixes *vs* :: *'v list*
and *hs* :: *'g list*
defines *hfvss* : *hfvss* ≡ *negHorbit-list hs induced-vector vs*
assumes *vs* : *set vs* ⊆ *V* **and** *i*: *i* < *length hs*
and *scalars* : *list-all2* (λ*rs ms*. *length rs* = *length ms*) *css hfvss*
and *rcoset-reps* : *Supgroup.is-rcoset-replist G hs*
shows ((*concat css*) •¤¤ (*concat hfvss*)) (*1* δδ (*hs*!*i*)) = (*css*!*i*) •♯· *vs*
⟨*proof*⟩

lemma *indspace-fspanning-set* :

fixes *vs* :: *'v list*
and *hs* :: *'g list*
defines *hfvss* : *hfvss* ≡ *negHorbit-list hs induced-vector vs*
assumes *base-spset* : *set vs* ⊆ *V V* = *BaseRep.fSpan vs*
and *rcoset-reps* : *Supgroup.is-rcoset-replist G hs*
shows *indV* = *induced-smult.fSpan* (*concat hfvss*)
⟨*proof*⟩

lemma *indspace-basis* :
fixes *vs* :: *'v list*
and *hs* :: *'g list*
defines *hfvss* : *hfvss* ≡ *negHorbit-list hs induced-vector vs*
assumes *base-spset* : *BaseRep.fbasis-for V vs*
and *rcoset-reps* : *Supgroup.is-rcoset-replist G hs*
shows *fscalar-mult.basis-for induced-smult.fsmult indV* (*concat hfvss*)
⟨*proof*⟩

lemma *induced-vector-decomp* :
fixes *vs* :: *'v list*
and *hs* :: *'g list*
and *cs* :: *'f list*
defines *hfvss* : *hfvss* ≡ *negHorbit-list* (*0#hs*) *induced-vector vs*
and *extrazeros* : *extrazeros* ≡ *replicate* ((*length hs*)∗(*length vs*)) *0*
assumes *base-spset* : *BaseRep.fbasis-for V vs*
and *rcoset-reps* : *Supgroup.is-rcoset-replist G* (*0#hs*)
and *cs* : *length cs* = *length vs*
and *v* : *v* = *cs* ·♯· *vs*
shows *induced-vector v* = (*cs@extrazeros*) ·¤¤ (*concat hfvss*)
⟨*proof*⟩

end

6.7.3 The induced space is a representation space

context *InducedFinGroupRepresentation*
begin

lemma *indspace-findim* :
fscalar-mult.findim induced-smult.fsmult indV
⟨*proof*⟩

theorem *FinGroupRepresentation-indspace* :
FinGroupRepresentation H rrsmult indV
⟨*proof*⟩

end

6.8 Frobenius reciprocity

6.8.1 Locale and basic facts

There are a number of defined objects and lemmas concerning those objects leading up to the theorem of Frobenius reciprocity, so we create a locale to contain it all.

locale *FrobeniusReciprocity*
= *GRep?*: *InducedFinGroupRepresentation H G smult V rrsmult*
+ *HRep?*: *FinGroupRepresentation H smult′ W*
  for *H* :: *′g::group-add set*
  and *G* :: *′g set*
  and *smult* :: *(′f::field, ′g) aezfun ⇒ ′v::ab-group-add ⇒ ′v* (**infixl** · *70*)
  and *V* :: *′v set*
  and *rrsmult* :: *(′f,′g) aezfun ⇒ ((′f,′g) aezfun ⇒ ′v)*
    ⇒ ((′f,′g) aezfun ⇒ ′v) (**infixl** ¤ *70*)
  and *smult′* :: *(′f, ′g) aezfun ⇒ ′w::ab-group-add ⇒ ′w* (**infixr** ⋆ *70*)
  and *W* :: *′w set*
begin

abbreviation *fsmult′* :: *′f ⇒ ′w ⇒ ′w* (**infixr** ♯⋆ *70*)
  where *fsmult′ ≡ HRep.fsmult*
abbreviation *flincomb′* :: *′f list ⇒ ′w list ⇒ ′w* (**infixr** ·♯⋆ *70*)
  where *flincomb′ ≡ HRep.flincomb*
abbreviation *Hmult′* :: *′g ⇒ ′w ⇒ ′w* (**infixr** ∗⋆ *70*)
  where *Hmult′ ≡ HRep.Gmult*

definition *Tsmult1* ::
  ′f ⇒ (((′f, ′g) aezfun ⇒ ′v)⇒′w) ⇒ (((′f, ′g) aezfun ⇒ ′v)⇒′w) (**infixr** ⋆¤ *70*)
  where *Tsmult1 ≡ λa T. λf. a ♯⋆ (T f)*

definition *Tsmult2* :: *′f ⇒ (′v⇒′w) ⇒ (′v⇒′w)* (**infixr** ⋆· *70*)
  where *Tsmult2 ≡ λa T. λv. a ♯⋆ (T v)*

lemma *FHModuleW* : *FGModule H (⋆) W* ⟨*proof*⟩

lemma *FGModuleW*: *FGModule G smult′ W*
⟨*proof*⟩

In order to build an inverse for the required isomorphism of Hom-sets, we will need a basis for the induced H-space.

definition *Vfbasis* :: *′v list* **where** *Vfbasis ≡ (SOME vs. is-Vfbasis vs)*

lemma *Vfbasis* : *is-Vfbasis Vfbasis*
  ⟨*proof*⟩

lemma *Vfbasis-V* : *set Vfbasis ⊆ V*
  ⟨*proof*⟩

definition *nonzero-H-rcoset-reps* :: $'g$ *list*
where *nonzero-H-rcoset-reps* ≡ (*SOME hs. Group.is-rcoset-replist H G* (*0#hs*))

definition *H-rcoset-reps* :: $'g$ *list* **where** *H-rcoset-reps* ≡ *0* # *nonzero-H-rcoset-reps*

lemma *H-rcoset-reps* : *Group.is-rcoset-replist H G H-rcoset-reps*
⟨*proof*⟩

lemma *H-rcoset-reps-H* : *set H-rcoset-reps* ⊆ *H*
⟨*proof*⟩

lemma *nonzero-H-rcoset-reps-H* : *set nonzero-H-rcoset-reps* ⊆ *H*
⟨*proof*⟩

abbreviation *negHorbit-homVfbasis* :: ($'v$ ⇒ $'w$) ⇒ $'w$ *list list*
where *negHorbit-homVfbasis T* ≡ *HRep.negGorbit-list H-rcoset-reps T Vfbasis*

lemma *negHorbit-Hom-indVfbasis-W* :
T ‘ *V* ⊆ *W* ⟹ *set* (*concat* (*negHorbit-homVfbasis T*)) ⊆ *W*
⟨*proof*⟩

lemma *negHorbit-HomSet-indVfbasis-W* :
T ∈ *GRepHomSet smult'* *W* ⟹ *set* (*concat* (*negHorbit-homVfbasis T*)) ⊆ *W*
⟨*proof*⟩

definition *indVfbasis* :: (($'f$, $'g$) *aezfun* ⇒ $'v$) *list list*
where *indVfbasis* ≡ *GRep.negHorbit-list H-rcoset-reps induced-vector Vfbasis*

lemma *indVfbasis* :
fscalar-mult.basis-for induced-smult.fsmult indV (*concat indVfbasis*)
⟨*proof*⟩

lemma *indVfbasis-indV* : *hfvs* ∈ *set indVfbasis* ⟹ *set hfvs* ⊆ *indV*
⟨*proof*⟩

end

6.8.2 The required isomorphism of Hom-sets

context *FrobeniusReciprocity*
begin

The following function will demonstrate the required isomorphism of Hom-sets (as vector spaces).

definition φ :: ((($'f$, $'g$) *aezfun* ⇒ $'v$) ⇒ $'w$) ⇒ ($'v$ ⇒ $'w$)
where φ ≡ *restrict0* (λ*T*. *T* ∘ *GRep.induced-vector*) (*HRepHomSet smult'* *W*)

lemma φ*-im* : φ ‘ *HRepHomSet* (⋆) *W* ⊆ *GRepHomSet* (⋆) *W*
⟨*proof*⟩

end

6.8.3 The inverse map of Hom-sets

In this section we build an inverse for the required isomorphism, φ.

context *FrobeniusReciprocity*
begin

definition ψ*-condition* :: $('v \Rightarrow 'w) \Rightarrow ((('f, 'g)\ aezfun \Rightarrow 'v) \Rightarrow 'w) \Rightarrow bool$
 where ψ*-condition T S*
  $\equiv$ *VectorSpaceHom induced-smult.fsmult indV fsmult′ S*
  $\wedge$ *map (map S) indVfbasis = negHorbit-homVfbasis T*

lemma *inverse-im-exists′* :
 assumes $T \in$ *GRepHomSet* $(\star)$ *W*
 shows $\exists!$ *S. VectorSpaceHom induced-smult.fsmult indV fsmult′ S*
  $\wedge$ *map S (concat indVfbasis) = concat (negHorbit-homVfbasis T)*
⟨*proof*⟩

lemma *inverse-im-exists* :
 assumes $T \in$ *GRepHomSet* $(\star)$ *W*
 shows $\exists!$ *S.* ψ*-condition T S*
⟨*proof*⟩

definition ψ :: $('v \Rightarrow 'w) \Rightarrow ((('f, 'g)\ aezfun \Rightarrow 'v) \Rightarrow 'w)$
 where $\psi \equiv$ *restrict0* (λT. *THE S.* ψ*-condition T S*) (*GRepHomSet* $(\star)$ *W*)

lemma ψD : $T \in$ *GRepHomSet* $(\star)$ $W \Longrightarrow \psi$*-condition T* $(\psi\ T)$
 ⟨*proof*⟩

lemma ψD*-VectorSpaceHom* :
 $T \in$ *GRepHomSet* $(\star)$ *W*
  $\Longrightarrow$ *VectorSpaceHom induced-smult.fsmult indV fsmult′* $(\psi\ T)$
 ⟨*proof*⟩

lemma ψD*-im* :
 $T \in$ *GRepHomSet* $(\star)$ $W \Longrightarrow$ *map (map* $(\psi\ T)$*) indVfbasis*
  = *aezfun-scalar-mult.negGorbit-list* $(\star)$ *H-rcoset-reps T Vfbasis*
 ⟨*proof*⟩

lemma ψD*-im-single* :
 assumes $T \in$ *GRepHomSet* $(\star)$ *W h* $\in$ *set H-rcoset-reps v* $\in$ *set Vfbasis*
 shows $\psi\ T\ ((-\ h) *¤ (induced\text{-}vector\ v)) = (-h) *\star (T\ v)$
⟨*proof*⟩

lemma ψT*-W* :
 assumes $T \in$ *GRepHomSet* $(\star)$ *W*
 shows $\psi\ T$ ‘ *indV* $\subseteq W$

$\langle proof \rangle$

lemma *ψT-Hmap-on-indVfbasis* :
assumes $T \in GRepHomSet\ (\star)\ W$
shows $\bigwedge x\, f.\ x \in H \Longrightarrow f \in set\ (concat\ indVfbasis)$
$\Longrightarrow \psi\ T\ (x * ¤\ f) = x ** (\psi\ T\ f)$
$\langle proof \rangle$

lemma *ψT-hom* :
assumes $T \in GRepHomSet\ (\star)\ W$
shows $HRepHom\ (\star)\ (\psi\ T)$
$\langle proof \rangle$

lemma *ψ-im* : ψ ‘ $GRepHomSet\ (\star)\ W \subseteq HRepHomSet\ (\star)\ W$
$\langle proof \rangle$

end

6.8.4 Demonstration of bijectivity

Now we demonstrate that φ is bijective via the inverse ψ.

context *FrobeniusReciprocity*
begin

lemma *φψ* :
assumes $T \in GRepHomSet\ smult'\ W$
shows $(\varphi \circ \psi)\ T = T$
$\langle proof \rangle$

lemma *φ-inverse-im* : φ ‘ $HRepHomSet\ (\star)\ W \supseteq GRepHomSet\ (\star)\ W$
$\langle proof \rangle$

lemma *bij-φ* : $bij\text{-}betw\ \varphi\ (HRepHomSet\ (\star)\ W)\ (GRepHomSet\ (\star)\ W)$
$\langle proof \rangle$

end

6.8.5 The theorem

Finally we demonstrate that φ is an isomorphism of vector spaces between the two hom-sets, leading to Frobenius reciprocity.

context *FrobeniusReciprocity*
begin

lemma *VectorSpaceIso-φ* :
$VectorSpaceIso\ Tsmult1\ (HRepHomSet\ (\star)\ W)\ Tsmult2\ \varphi$
$(GRepHomSet\ (\star)\ W)$
$\langle proof \rangle$

theorem *FrobeniusReciprocity* :
VectorSpace.isomorphic Tsmult1 (*HRepHomSet smult′ W*) *Tsmult2*
(*GRepHomSet smult′ W*)
⟨*proof*⟩

end

end

7 Bibliography

[1] S. Axler. *Linear Algebra Done Right*. Undergraduate Texts in Mathematics. Springer, New York, third edition, 2015.

[2] D. S. Dummit and R. M. Foote. *Abstract Algebra*. John Wiley & Sons, New York, second edition, 1999.

[3] G. James and M. Liebeck. *Representations and Characters of Groups*. Cambridge Mathematical Textbooks. Cambridge University Press, Cambridge, UK, 1993.