p-adic Fields

Aaron Crighton

March 17, 2025

Contents

Abstract

We formalize the fields $\mathbb{Q}_p$ of p-adic numbers within the framework of the HOL-Algebra library. The p-adic field is defined simply as the fraction field of the ring of p-adic integers. The valuation, and basic topological properties of $\mathbb{Q}_p$ are developed, including deducing Hensel's Lemma for $\mathbb{Q}_p$ from the same theorem for $\mathbb{Z}_p$. The theory of semialgebraic subsets of $\mathbb{Q}_p^n$ and semialgebraic functions is also developed, as outlined in [1]. In order to formulate these results, general theory about multivariable polynomial rings and cartesian powers of a ring must also be developed. This work is done with a view to formalizing the proof in [1] of Macintyre's quantifier elimination theorem for semialgebraic subsets of $\mathbb{Q}_p^n$.

theory *Fraction-Field*
imports *HOL−Algebra.UnivPoly*
Localization-Ring.Localization
HOL−Algebra.Subrings
Padic-Ints.Supplementary-Ring-Facts
begin

1 The Field of Fractions of a Ring

This theory defines the fraction field of a domain and establishes its basic properties. The fraction field is defined in the standard way as the localization of a domain at its nonzero elements. This is done by importing the AFP session `Localization_Ring`. Choice functions for numerator and denominators of fractions are introduced, and the inclusion of a domain into its ring of fractions is defined.

1.1 The Monoid of Nonzero Elements in a Domain

locale *domain-frac* = *domain*

lemma *zero-not-in-nonzero*: $\mathbf{0}_R \notin$ *nonzero R*
unfolding *nonzero-def* **by** *blast*

lemma(**in** *domain*) *nonzero-is-submonoid*: *submonoid R* (*nonzero R*)
proof
show *nonzero R* $\subseteq$ *carrier R*
using *nonzero-def* **by** *fastforce*
show $\bigwedge x\ y.\ x \in$ *nonzero R* $\Longrightarrow y \in$ *nonzero R* $\Longrightarrow x \otimes y \in$ *nonzero R*
by (*metis* (*mono-tags*, *lifting*) *local.integral m-closed mem-Collect-eq nonzero-def*)
show $\mathbf{1} \in$ *nonzero R*
by (*simp add*: *nonzero-def*)
qed

lemma(**in** *domain*) *nonzero-closed*:
assumes $a \in$ *nonzero R*
shows $a \in$ *carrier R*
using *assms*
by (*simp add*: *nonzero-def*)

lemma(**in** *domain*) *nonzero-mult-closed*:
assumes $a \in$ *nonzero R*
assumes $b \in$ *nonzero R*
shows $a \otimes b \in$ *carrier R*
using *assms*
by (*simp add*: *nonzero-def*)

lemma(**in** *domain*) *nonzero-one-closed*:
$\mathbf{1} \in$ *nonzero R*
by (*simp add*: *nonzero-def*)

lemma(**in** *domain*) *nonzero-memI*:
assumes $a \in$ *carrier R*
assumes $a \neq \mathbf{0}$
shows $a \in$ *nonzero R*
using *assms* **by**(*simp add*: *nonzero-def*)

lemma(**in** *domain*) *nonzero-memE*:
assumes $a \in$ *nonzero* R
shows $a \in$ *carrier* R $a \neq \mathbf{0}$
using *assms* **by**(*auto simp*: *nonzero-def*)

lemma(**in** *domain*) *not-nonzero-memE*:
assumes $a \notin$ *nonzero* R
assumes $a \in$ *carrier* R
shows $a = \mathbf{0}$
using *assms*
by (*simp add*: *nonzero-def*)

lemma(**in** *domain*) *not-nonzero-memI*:
assumes $a = \mathbf{0}$
shows $a \notin$ *nonzero* R
using *assms nonzero-memE*(*2*) **by** *auto*

lemma(**in** *domain*) *one-nonzero*:
$\mathbf{1} \in$ *nonzero* R
by (*simp add*: *nonzero-one-closed*)

lemma(**in** *domain-frac*) *eq-obj-rng-of-frac-nonzero*:
eq-obj-rng-of-frac R (*nonzero* R)
using *nonzero-is-submonoid*
by (*simp add*: *eq-obj-rng-of-frac.intro*
is-cring local.ring-axioms mult-submonoid-of-crng-def mult-submonoid-of-rng-def)

1.2 Numerator and Denominator Choice Functions

definition(**in** *eq-obj-rng-of-frac*) *denom* **where**
denom a = (*if* ($a = \mathbf{0}_{\textit{rec-rng-of-frac}}$) *then* $\mathbf{1}$ *else* (*snd* (*SOME* x. $x \in a$)))

The choice function for numerators must be compatible with denom:

definition(**in** *eq-obj-rng-of-frac*) *numer* **where**
numer a = (*if* ($a = \mathbf{0}_{\textit{rec-rng-of-frac}}$) *then* $\mathbf{0}$ *else* (*fst* (*SOME* x. $x \in a \wedge$ (*snd* x = *denom* a))))

Basic properties of numer and denom:

lemma(**in** *eq-obj-rng-of-frac*) *numer-denom-facts0*:
assumes *domain* R
assumes $\mathbf{0} \notin S$
assumes $a \in$ *carrier rec-rng-of-frac*
assumes $a \neq \mathbf{0}_{\textit{rec-rng-of-frac}}$
shows $a = ((\textit{numer}\ a) \mid_{\textit{rel}} (\textit{denom}\ a))$
(*numer* a) $\in$ *carrier* R
(*denom* a) $\in S$
numer $a = \mathbf{0} \Longrightarrow a = \mathbf{0}_{\textit{rec-rng-of-frac}}$

$a \otimes_{rec\text{-}rng\text{-}of\text{-}frac} (rng\text{-}to\text{-}rng\text{-}of\text{-}frac(denom\ a)) = rng\text{-}to\text{-}rng\text{-}of\text{-}frac\ (numer\ a)$

$(rng\text{-}to\text{-}rng\text{-}of\text{-}frac(denom\ a)) \otimes_{rec\text{-}rng\text{-}of\text{-}frac} a = rng\text{-}to\text{-}rng\text{-}of\text{-}frac\ (numer\ a)$

proof −
have *0*: $carrier\ rel \neq \{\}$
by (*metis* (*no-types*, *lifting*) *SigmaI empty-iff one-closed partial-object.select-convs*(*1*) *rel-def zero-closed*)
have *1*: $(numer\ a,\ denom\ a) \in a$
proof −
have $\exists\ r\ s.\ (r \in carrier\ R \wedge s \in S \wedge (a = (r\ |_{rel}\ s)))$
using *rel-def assms*(*3*) *assms*(*1*) *set-eq-class-of-rng-of-frac-def rec-rng-of-frac-def*
by (*smt mem-Collect-eq mem-Sigma-iff partial-object.select-convs*(*1*))
then obtain *r s* **where** $r \in carrier\ R \wedge s \in S \wedge (a = (r\ |_{rel}\ s))$
by *blast*
hence $a = class\text{-}of_{rel}\ (r,\ s)$
by (*simp add*: *class-of-to-rel*)
hence $(r,s) \in a$ **using** *eq-class-of-def rel-def*
using ‹$r \in carrier\ R \wedge s \in S \wedge a = (r\ |_{rel}\ s)$› *equiv-obj-rng-of-frac equivalence.refl* **by** *fastforce*
hence $(\exists\ x.\ x \in a)$
by *blast*
hence $(SOME\ x.\ x \in a) \in a$
by (*meson some-eq-ex*)
hence $(\exists\ x.\ x \in a \wedge (snd\ x = denom\ a))$
using *denom-def assms* **by** *metis*
hence $\exists x.\ x \in a \wedge (snd\ x = denom\ a) \wedge (fst\ x = numer\ a)$
using *numer-def assms* **by** (*metis* (*mono-tags*, *lifting*) *exE-some*)
thus *?thesis*
by *simp*
qed
have $a \in \{r\ |_{rel}\ s\ |r\ s.\ (r,\ s) \in carrier\ rel\}$
using *assms*(*3*) *rec-rng-of-frac-def set-eq-class-of-rng-of-frac-def* **by** *auto*
hence $\exists\ x\ y.\ a = (x\ |_{rel}\ y) \wedge (x,y) \in carrier\ rel$
using *rec-rng-of-frac-def eq-class-of-rng-of-frac-def set-eq-class-of-rng-of-frac-def*
by *blast*
then obtain *x y* **where** $a = (x\ |_{rel}\ y)$ **and** *P0*:$(x,y) \in carrier\ rel$
by *blast*
hence *P1*: $(numer\ a,\ denom\ a) \in (x\ |_{rel}\ y)$
using *1* **by** *blast*
thus *2*:$a = ((numer\ a)\ |_{rel}\ (denom\ a))$
proof −
have *P2*:$(numer\ a,\ denom\ a) \in carrier\ rel \wedge (x,\ y).=_{rel} (numer\ a,\ denom\ a)$
using *eq-class-of-rng-of-frac-def P1* **by** *fastforce*
hence $(x,\ y).=_{rel} (numer\ a,\ denom\ a)$
by *blast*
hence $(numer\ a,\ denom\ a).=_{rel}(x,\ y)$
using *equiv-obj-rng-of-frac* **by** (*simp add*: *equivalence.sym P0 P2*)

thus *?thesis*
by (*metis P0 P2* ‹$a = (x \mid_{rel} y)$› *class-of-to-rel elem-eq-class equiv-obj-rng-of-frac*)

qed
have *3*:(*numer a*, *denom a*) ∈ *carrier rel*
using *P1* **by** (*simp add*: *eq-class-of-rng-of-frac-def*)
thus *4*: *numer a* ∈ *carrier R*
by (*simp add*: *rel-def*)
show *5*: *denom a* ∈ *S*
using *3 rel-def* **by** *auto*
show *numer a* = **0** ⟹ *a* = **0**$_{rec\text{-}rng\text{-}of\text{-}frac}$
proof−
assume *numer a* = **0**
hence *a* = (**0** $\mid_{rel}$ (*denom a*))
using *2* **by** *auto*
thus *?thesis*
using *5 class-of-zero-rng-of-frac* **by** *auto*
qed
show *a* $\otimes_{rec\text{-}rng\text{-}of\text{-}frac}$ *rng-to-rng-of-frac* (*denom a*) = *rng-to-rng-of-frac* (*numer a*)
proof−
have *S*: (*denom a*, **1**) ∈*carrier rel*
using *5 rel-def subset* **by** *auto*
hence *a* $\otimes_{rec\text{-}rng\text{-}of\text{-}frac}$ *rng-to-rng-of-frac* (*denom a*) = (((*numer a*) ⊗ (*denom a*)) $\mid_{rel}$ ((*denom a*) ⊗ **1**))
using *2 3 mult-rng-of-frac-fundamental-lemma rng-to-rng-of-frac-def rec-monoid-rng-of-frac-def rec-rng-of-frac-def* **by** *fastforce*
hence *a* $\otimes_{rec\text{-}rng\text{-}of\text{-}frac}$ *rng-to-rng-of-frac* (*denom a*) = (((*denom a*)⊗ (*numer a*)) $\mid_{rel}$ ((*denom a*) ⊗ **1**))
using *4 5 m-comm subset* **by** *auto*
hence *a* $\otimes_{rec\text{-}rng\text{-}of\text{-}frac}$ *rng-to-rng-of-frac* (*denom a*) = ((*denom a*) $\mid_{rel}$ (*denom a*)) $\otimes_{rec\text{-}rng\text{-}of\text{-}frac}$((*numer a*) $\mid_{rel}$ **1**)
using *mult-rng-of-frac-fundamental-lemma 4 5 S rec-monoid-rng-of-frac-def rec-rng-of-frac-def rel-def* **by** *auto*
thus *?thesis*
using *4 5* ‹*a* $\otimes_{rec\text{-}rng\text{-}of\text{-}frac}$ *rng-to-rng-of-frac* (*denom a*) = (*denom a* ⊗ *numer a* $\mid_{rel}$ *denom a* ⊗ **1**)› *rel-def rng-to-rng-of-frac-def simp-in-frac* **by** *auto*
qed
thus (*rng-to-rng-of-frac*(*denom a*)) $\otimes_{rec\text{-}rng\text{-}of\text{-}frac}$ *a* = *rng-to-rng-of-frac* (*numer a*)
by (*smt 5 assms*(*3*) *cring.cring-simprules*(*14*) *crng-rng-of-frac ring-hom-closed rng-to-rng-of-frac-is-ring-hom subset subsetD*)
qed

lemma(**in** *eq-obj-rng-of-frac*) *frac-zero*:
assumes *domain R*
assumes **0** ∉ *S*
assumes *a* ∈ *carrier R*

assumes $b \in S$
assumes $(a \mid_{rel} b) = \mathbf{0}_{rec\text{-}rng\text{-}of\text{-}frac}$
shows $a = \mathbf{0}$
proof−
have *0*: $(a \mid_{rel} b) = (\mathbf{0} \mid_{rel} \mathbf{1})$
by (*simp add*: *assms*(*5*) *class-of-zero-rng-of-frac*)
have *1*: $(a,b) \in$ *carrier rel*
by (*simp add*: *assms*(*3*) *assms*(*4*) *rel-def*)
have *2*: $(\mathbf{0}, \mathbf{1}) \in$ *carrier rel*
by (*simp add*: *rel-def*)
have *3*: $(b, \mathbf{1}) \in$ *carrier rel*
using *assms*(*4*) *rel-def subset* **by** *auto*
have $(a,b) .=_{rel} (\mathbf{0}, \mathbf{1})$
by (*metis* (*no-types*, *lifting*) *0 1 2 eq-class-to-rel partial-object.select-convs*(*1*) *rel-def*)
have $(a \mid_{rel} b) \otimes_{rec\text{-}rng\text{-}of\text{-}frac} (b \mid_{rel} \mathbf{1}) = (\mathbf{0} \mid_{rel} b)$
by (*metis* (*no-types*, *opaque-lifting*) *assms*(*4*) *assms*(*5*) *basic-trans-rules*(*31*) *class-of-zero-rng-of-frac cring.axioms*(*1*) *crng-rng-of-frac ring.ring-simprules*(*24*) *ring-hom-closed rng-to-rng-of-frac-def rng-to-rng-of-frac-is-ring-hom subset*)
hence *4*: $((a \otimes b) \mid_{rel} b) = (\mathbf{0} \mid_{rel} b)$
using *1 3 assms*(*4*) *mult-rng-of-frac-fundamental-lemma rec-monoid-rng-of-frac-def rec-rng-of-frac-def subset* **by** *auto*
have *S*: $((a \otimes b) , b) \in$ *carrier rel*
using *assms*(*3*) *assms*(*4*) *rel-def subset* **by** *auto*
have *T*: $(\mathbf{0}, b) \in$*carrier rel*
by (*simp add*: *assms*(*4*) *rel-def*)
hence $((a \otimes b) , b).=_{rel} (\mathbf{0} , b)$
using *4 S eq-class-to-rel rel-def* **by** *auto*
hence *eq rel* $((a \otimes b) , b)$ $(\mathbf{0} , b)$
by *blast*
hence $\exists t{\in}S.\ t \otimes (b \otimes (a \otimes b) \ominus b \otimes \mathbf{0}) = \mathbf{0}$
using *rel-def* **by** *auto*
then obtain t **where** $t \in S$ **and** $t \otimes (b \otimes (a \otimes b) \ominus b \otimes \mathbf{0}) = \mathbf{0}$
by *blast*
have $t \neq \mathbf{0}$
using ‹$t \in S$› *assms*(*2*) **by** *blast*
hence $(b \otimes (a \otimes b) \ominus b \otimes \mathbf{0}) = \mathbf{0}$
by (*meson* ‹$t \in S$› ‹$t \otimes (b \otimes (a \otimes b) \ominus b \otimes \mathbf{0}) = \mathbf{0}$› *assms*(*1*) *assms*(*3*) *assms*(*4*) *domain.integral-iff minus-closed semiring-simprules*(*3*) *set-mp subset zero-closed*)
hence $b \otimes (a \otimes b) = \mathbf{0}$
using *3 S rel-def abelian-group.minus-to-eq is-abelian-group* **by** *fastforce*
thus $a = \mathbf{0}$
by (*metis* (*no-types*, *lifting*) *assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *domain.integral-iff semiring-simprules*(*3*) *subset subsetCE*)
qed

When S does not contain 0, and R is a domain, the localization is a domain.

lemma(**in** *eq-obj-rng-of-frac*) *rec-rng-of-frac-is-domain*:
assumes *domain R*
assumes $\mathbf{0} \notin S$
shows *domain rec-rng-of-frac*
proof(*rule domainI*)
show *cring rec-rng-of-frac*
by (*simp add*: *crng-rng-of-frac*)
show $\mathbf{1}_{rec\text{-}rng\text{-}of\text{-}frac} \neq \mathbf{0}_{rec\text{-}rng\text{-}of\text{-}frac}$
proof−
have $\mathbf{1}_R \neq \mathbf{0}_R$
by (*simp add*: *assms domain.one-not-zero*)
hence *0*: $\mathbf{1}_R \notin$ (*a-kernel R rec-rng-of-frac rng-to-rng-of-frac*)
using *assms*(*1*) *rng-to-rng-of-frac-without-zero-div-is-inj*
by (*simp add*: *assms*(*2*) *domain-axioms-def domain-def*)
hence *rng-to-rng-of-frac* $\mathbf{1} \neq \mathbf{0}_{rec\text{-}rng\text{-}of\text{-}frac}$
by (*simp add*: *a-kernel-def'*)
thus *?thesis*
using *ring-hom-one rng-to-rng-of-frac-is-ring-hom* **by** *blast*
qed
show $\bigwedge a\ b.\ a \otimes_{rec\text{-}rng\text{-}of\text{-}frac} b = \mathbf{0}_{rec\text{-}rng\text{-}of\text{-}frac} \Longrightarrow$
$a \in$ *carrier rec-rng-of-frac* $\Longrightarrow$
$b \in$ *carrier rec-rng-of-frac* $\Longrightarrow$
$a = \mathbf{0}_{rec\text{-}rng\text{-}of\text{-}frac} \vee b = \mathbf{0}_{rec\text{-}rng\text{-}of\text{-}frac}$
proof−
fix *a b*
assume *A1*: $a \otimes_{rec\text{-}rng\text{-}of\text{-}frac} b = \mathbf{0}_{rec\text{-}rng\text{-}of\text{-}frac}$
assume *A2*: $a \in$ *carrier rec-rng-of-frac*
assume *A3*: $b \in$ *carrier rec-rng-of-frac*
show $a = \mathbf{0}_{rec\text{-}rng\text{-}of\text{-}frac} \vee b = \mathbf{0}_{rec\text{-}rng\text{-}of\text{-}frac}$
proof(*rule disjCI*)
assume $b \neq \mathbf{0}_{rec\text{-}rng\text{-}of\text{-}frac}$
have $\neg\ a \neq \mathbf{0}_{rec\text{-}rng\text{-}of\text{-}frac}$
proof
assume $a \neq \mathbf{0}_{rec\text{-}rng\text{-}of\text{-}frac}$
have *B1*: *numer a* $\neq \mathbf{0}$
using *A2* ‹$a \neq \mathbf{0}_{rec\text{-}rng\text{-}of\text{-}frac}$› *assms*(*1*) *assms*(*2*) *numer-denom-facts0*(*4*)
by *blast*
have *B2*: *numer b* $\neq \mathbf{0}$
using *A3* ‹$b \neq \mathbf{0}_{rec\text{-}rng\text{-}of\text{-}frac}$› *assms*(*1*) *assms*(*2*) *numer-denom-facts0*(*4*)
by *blast*
have *B3*: *denom a* $\neq \mathbf{0}$
using *A2* ‹$a \neq \mathbf{0}_{rec\text{-}rng\text{-}of\text{-}frac}$› *assms*(*1*) *assms*(*2*)
eq-obj-rng-of-frac.numer-denom-facts0(*1*) *eq-obj-rng-of-frac-axioms*
using *numer-denom-facts0*(*3*) **by** *force*
have *B4*: *denom b* $\neq \mathbf{0}$
using *A3* ‹$b \neq \mathbf{0}_{rec\text{-}rng\text{-}of\text{-}frac}$› *assms*(*1*) *assms*(*2*)
eq-obj-rng-of-frac-axioms **by** (*metis* (*no-types*, *lifting*) *numer-denom-facts0*(*3*))

have $a \otimes_{rec\text{-}rng\text{-}of\text{-}frac} b = (((numer\ a) \otimes (numer\ b)) \mid_{rel} ((denom\ a) \otimes (denom\ b)))$

proof −

have *S0*: $a = (numer\ a \mid_{rel} denom\ a)$

by (*simp add*: *A2* ‹$a \neq \mathbf{0}_{rec\text{-}rng\text{-}of\text{-}frac}$› *assms*(*1*) *assms*(*2*) *numer-denom-facts0*(*1*))

have *S1*: $b = (numer\ b \mid_{rel} denom\ b)$

by (*simp add*: *A3* ‹$b \neq \mathbf{0}_{rec\text{-}rng\text{-}of\text{-}frac}$› *assms*(*1*) *assms*(*2*) *numer-denom-facts0*(*1*))

have *S2*: $(numer\ a, denom\ a) \in carrier\ rel$

using *A2* ‹$a \neq \mathbf{0}_{rec\text{-}rng\text{-}of\text{-}frac}$› *assms*(*1*) *assms*(*2*) *eq-obj-rng-of-frac.numer-denom-facts0*(*2*) *eq-obj-rng-of-frac.numer-denom-facts0*(*3*) *eq-obj-rng-of-frac-axioms rel-def* **by** *fastforce*

have *S3*: $(numer\ b, denom\ b) \in carrier\ rel$

using *A3* ‹$b \neq \mathbf{0}_{rec\text{-}rng\text{-}of\text{-}frac}$› *assms*(*1*) *assms*(*2*) *eq-obj-rng-of-frac.numer-denom-facts0*(*2*) *eq-obj-rng-of-frac-axioms numer-denom-facts0*(*3*) *rel-def* **by** *auto*

show *?thesis* **using** *S0 S1 S2 S3 mult-rng-of-frac-fundamental-lemma*

using *rec-monoid-rng-of-frac-def rec-rng-of-frac-def* **by** *force*

qed

hence $(((numer\ a) \otimes (numer\ b)) \mid_{rel} ((denom\ a) \otimes (denom\ b))) = \mathbf{0}_{rec\text{-}rng\text{-}of\text{-}frac}$

using *A1* **by** *blast*

hence $(numer\ a) \otimes (numer\ b) = \mathbf{0}$

by (*meson A2 A3* ‹$a \neq \mathbf{0}_{rec\text{-}rng\text{-}of\text{-}frac}$› ‹$b \neq \mathbf{0}_{rec\text{-}rng\text{-}of\text{-}frac}$› *assms*(*1*) *assms*(*2*) *eq-obj-rng-of-frac.numer-denom-facts0*(*2*) *eq-obj-rng-of-frac.numer-denom-facts0*(*3*) *eq-obj-rng-of-frac-axioms frac-zero m-closed semiring-simprules*(*3*))

thus *False*

by (*meson A2 A3 B1 B2* ‹$a \neq \mathbf{0}_{rec\text{-}rng\text{-}of\text{-}frac}$› ‹$b \neq \mathbf{0}_{rec\text{-}rng\text{-}of\text{-}frac}$› *assms*(*1*) *assms*(*2*) *domain.integral-iff eq-obj-rng-of-frac.numer-denom-facts0*(*2*) *eq-obj-rng-of-frac-axioms*)

qed

thus $a = \mathbf{0}_{rec\text{-}rng\text{-}of\text{-}frac}$

by *auto*

qed

qed

qed

lemma(**in** *eq-obj-rng-of-frac*) *numer-denom-facts*:

assumes *domain R*

assumes $\mathbf{0} \notin S$

assumes $a \in carrier\ rec\text{-}rng\text{-}of\text{-}frac$

shows $a = (numer\ a \mid_{rel} denom\ a)$

$(numer\ a) \in carrier\ R$

$(denom\ a) \in S$

$a \neq \mathbf{0}_{rec\text{-}rng\text{-}of\text{-}frac} \Longrightarrow (numer\ a) \neq \mathbf{0}$

$a \otimes_{rec\text{-}rng\text{-}of\text{-}frac} (rng\text{-}to\text{-}rng\text{-}of\text{-}frac(denom\ a)) = rng\text{-}to\text{-}rng\text{-}of\text{-}frac\ (numer\ a)$
$(rng\text{-}to\text{-}rng\text{-}of\text{-}frac(denom\ a)) \otimes_{rec\text{-}rng\text{-}of\text{-}frac} a = rng\text{-}to\text{-}rng\text{-}of\text{-}frac\ (numer\ a)$
using *assms(1) assms(2) assms(3) class-of-zero-rng-of-frac denom-def numer-def numer-denom-facts0(1)* **apply** *force*
using *assms(1) assms(2) assms(3) numer-def numer-denom-facts0(2)* **apply** *force*
using *assms(1) assms(2) assms(3) denom-def numer-denom-facts0(3)* **apply** *force*
using *assms(1) assms(2) assms(3) numer-denom-facts0(4)* **apply** *blast*
apply (*metis* (*no-types*, *lifting*) *assms(1) assms(2) assms(3) class-of-zero-rng-of-frac denom-def monoid.r-one monoid.simps(2) numer-def numer-denom-facts0(5) one-closed rec-rng-of-frac-def ringE(2) rng-rng-of-frac rng-to-rng-of-frac-def*)
by (*metis* (*no-types*, *lifting*) *assms(1) assms(2) assms(3) class-of-zero-rng-of-frac cring.cring-simprules(12) crng-rng-of-frac denom-def monoid.simps(2) numer-def numer-denom-facts0(6) one-closed rec-rng-of-frac-def rng-to-rng-of-frac-def*)

lemma(**in** *eq-obj-rng-of-frac*) *numer-denom-closed*:
assumes *domain R*
assumes $\mathbf{0} \notin S$
assumes $a \in carrier\ rec\text{-}rng\text{-}of\text{-}frac$
shows $(numer\ a,\ denom\ a) \in carrier\ rel$
by (*simp add*: *assms(1) assms(2) assms(3) numer-denom-facts(2) numer-denom-facts(3) rel-def*)

Fraction function which suppresses the "rel" argument.

definition(**in** *eq-obj-rng-of-frac*) *fraction* **where**
$fraction\ x\ y = (x\ |_{rel}\ y)$

lemma(**in** *eq-obj-rng-of-frac*) *a-is-fraction*:
assumes *domain R*
assumes $\mathbf{0} \notin S$
assumes $a \in carrier\ rec\text{-}rng\text{-}of\text{-}frac$
shows $a = fraction\ (numer\ a)\ (denom\ a)$
by (*simp add*: *assms(1) assms(2) assms(3) fraction-def numer-denom-facts(1)*)

lemma(**in** *eq-obj-rng-of-frac*) *add-fraction*:
assumes *domain R*
assumes $\mathbf{0} \notin S$
assumes $a \in carrier\ R$
assumes $b \in S$
assumes $c \in carrier\ R$
assumes $d \in S$

shows (*fraction a b*) $\oplus_{\textit{rec-rng-of-frac}}$ (*fraction c d*) = (*fraction* ((*a* $\otimes$ *d*) $\oplus$ (*b* $\otimes$ *c*)) (*b* $\otimes$ *d*))
proof−
have *0*:(*a*,*b*) ∈ *carrier rel*
by (*simp add*: *assms*(*3*) *assms*(*4*) *rel-def*)
have *1*:(*c*,*d*) ∈ *carrier rel*
by (*simp add*: *assms*(*5*) *assms*(*6*) *rel-def*)
show *?thesis*
using *0 1 add-rng-of-frac-fundamental-lemma assms numer-denom-facts fraction-def*
domain-def m-comm subset
by *auto*
qed

lemma(**in** *eq-obj-rng-of-frac*) *mult-fraction*:
assumes *domain R*
assumes **0** ∉ *S*
assumes *a* ∈ *carrier R*
assumes *b* ∈ *S*
assumes *c* ∈ *carrier R*
assumes *d* ∈ *S*
shows (*fraction a b*) $\otimes_{\textit{rec-rng-of-frac}}$ (*fraction c d*) = (*fraction* (*a* $\otimes$ *c*) (*b* $\otimes$ *d*))
proof−
have *0*:(*a*,*b*) ∈ *carrier rel*
by (*simp add*: *assms*(*3*) *assms*(*4*) *rel-def*)
have *1*:(*c*,*d*) ∈ *carrier rel*
by (*simp add*: *assms*(*5*) *assms*(*6*) *rel-def*)
show *?thesis* **using** *0 1 mult-rng-of-frac-fundamental-lemma*
by (*simp add*: *fraction-def rec-monoid-rng-of-frac-def rec-rng-of-frac-def*)
qed

lemma(**in** *eq-obj-rng-of-frac*) *fraction-zero*:
$\mathbf{0}_{\textit{rec-rng-of-frac}}$ = *fraction* **0** **1**
by (*simp add*: *class-of-zero-rng-of-frac fraction-def*)

lemma(**in** *eq-obj-rng-of-frac*) *fraction-zero′*:
assumes *a* ∈ *S*
assumes **0** ∉ *S*
shows $\mathbf{0}_{\textit{rec-rng-of-frac}}$ = *fraction* **0** *a*
by (*simp add*: *assms*(*1*) *class-of-zero-rng-of-frac fraction-def*)

lemma(**in** *eq-obj-rng-of-frac*) *fraction-one*:
$\mathbf{1}_{\textit{rec-rng-of-frac}}$ = *fraction* **1** **1**
by (*simp add*: *fraction-def rec-rng-of-frac-def*)

lemma(**in** *eq-obj-rng-of-frac*) *fraction-one′*:
assumes *domain R*
assumes **0** ∉ *S*
assumes *a* ∈ *S*

shows *fraction a a* = $\mathbf{1}_{rec\text{-}rng\text{-}of\text{-}frac}$
using *assms fraction-def fraction-one one-closed simp-in-frac subset*
by (*smt mem-Sigma-iff partial-object.select-convs*(*1*) *r-one rel-def subsetD*)

lemma(**in** *eq-obj-rng-of-frac*) *fraction-closed*:
assumes *domain R*
assumes **0** $\notin$ *S*
assumes *a* $\in$ *carrier R*
assumes *b* $\in$ *S*
shows *fraction a b* $\in$ *carrier rec-rng-of-frac*
proof−
have (*a*,*b*) $\in$ *carrier rel*
by (*simp add*: *assms*(*3*) *assms*(*4*) *rel-def*)
hence (*a* $|_{rel}$ *b*) $\in$ *set-class-of* $_{rel}$
using *set-eq-class-of-rng-of-frac-def*
by *blast*
thus *?thesis* **using** *fraction-def*
by (*simp add*: *rec-rng-of-frac-def*)
qed

1.3 Defining the Field of Fractions

definition *Frac* **where**
Frac R = *eq-obj-rng-of-frac.rec-rng-of-frac R* (*nonzero R*)

lemma(**in** *domain-frac*) *fraction-field-is-domain*:
domain (*Frac R*)
using *domain-axioms eq-obj-rng-of-frac.rec-rng-of-frac-is-domain*
eq-obj-rng-of-frac-nonzero Frac-def
by (*metis nonzero-memE*(*2*))

1.3.1 Numerator and Denominator Choice Functions for `domain_frac`

definition *numerator* **where**
numerator R = *eq-obj-rng-of-frac.numer R* (*nonzero R*)

abbreviation(**in** *domain-frac*)(*input*) *numer* **where**
numer ≡ *numerator R*

definition *denominator* **where**
denominator R = *eq-obj-rng-of-frac.denom R* (*nonzero R*)

abbreviation(**in** *domain-frac*)(*input*) *denom* **where**
denom ≡ *denominator R*

definition *fraction* **where**
fraction R = *eq-obj-rng-of-frac.fraction R* (*nonzero R*)

abbreviation(**in** *domain-frac*)(*input*) *frac* **where**
frac ≡ *fraction R*

1.3.2 The inclusion of R into its fraction field

definition *Frac-inc* **where**
Frac-inc R = *eq-obj-rng-of-frac.rng-to-rng-of-frac R* (*nonzero R*)

abbreviation(**in** *domain-frac*)(*input*) *inc* (‹ι›) **where**
inc ≡ *Frac-inc R*

lemma(**in** *domain-frac*) *inc-equation*:
assumes *a* ∈ *carrier R*
shows ι *a* = *frac a* **1**
unfolding *Frac-inc-def fraction-def*
using *assms*
by (*simp add*: *eq-obj-rng-of-frac.fraction-def eq-obj-rng-of-frac.rng-to-rng-of-frac-def eq-obj-rng-of-frac-nonzero*)

lemma(**in** *domain-frac*) *inc-is-hom*:
inc ∈ *ring-hom R* (*Frac R*)
by (*simp add*: *eq-obj-rng-of-frac.rng-to-rng-of-frac-is-ring-hom Frac-def eq-obj-rng-of-frac-nonzero Frac-inc-def*)

lemma(**in** *domain-frac*) *inc-is-hom1*:
ring-hom-ring R (*Frac R*) *inc*
apply(*rule ring-hom-ringI2*)
using *cring-def domain.axioms*(*1*) *fraction-field-is-domain*
by(*auto simp*:*inc-is-hom local.ring-axioms*)

Inclusion map is injective:

lemma(**in** *domain-frac*) *inc-inj0*:
a-kernel R (*Frac R*) *inc* = {**0**}
proof−
have *0*: **0** ∉ *nonzero R*
by (*simp add*: *nonzero-def*)
have *1*: *eq-obj-rng-of-frac R* (*nonzero R*)
by (*simp add*: *eq-obj-rng-of-frac-nonzero*)
have *2*: ∀ *a*∈ *carrier R*. ∀ *b*∈*carrier R*. *a* ⊗ *b* = **0** ⟶ *a* = **0** ∨ *b* = **0**
using *local.integral* **by** *blast*
show *?thesis* **using** *0 1 2*
by (*simp add*: *eq-obj-rng-of-frac.rng-to-rng-of-frac-without-zero-div-is-inj Frac-def Frac-inc-def*)
qed

lemma(**in** *domain-frac*) *inc-inj1*:
assumes *a* ∈ *carrier R*
assumes *inc a* = **0**$_{Frac\ R}$
shows *a* = **0**
proof−
have *a* ∈ *a-kernel R* (*Frac R*) *inc* **using** *a-kernel-def′ assms*(*2*)
by (*simp add*: *a-kernel-def′ assms*(*1*))
thus *?thesis*

using *inc-inj0* **by** *blast*
qed

lemma(**in** *domain-frac*) *inc-inj2*:
assumes $a \in$ *carrier* R
assumes $b \in$ *carrier* R
assumes *inc* $a =$ *inc* b
shows $a = b$
proof−
have *inc* $(a \ominus b) =$ (*inc* a) $\oplus_{Frac\ R}$ (*inc* ($\ominus$ b))
using *inc-is-hom* **by** (*simp add*: *ring-hom-add assms*(*1*) *assms*(*2*) *minus-eq*)
hence *inc* $(a \ominus b) = \mathbf{0}_{Frac\ R}$ **using** *assms inc-is-hom*
by (*smt Frac-def add.inv-closed eq-obj-rng-of-frac.rng-rng-of-frac*
eq-obj-rng-of-frac-nonzero local.ring-axioms r-neg ring-hom-add ring-hom-zero)

thus *?thesis*
by (*meson abelian-group.minus-to-eq assms*(*1*) *assms*(*2*) *domain-frac.inc-inj1*
domain-frac-axioms is-abelian-group minus-closed)
qed

Image of inclusion map is a subring:

lemma(**in** *domain-frac*) *inc-im-is-subring*:
subring (ι ‘ (*carrier* R)) (*Frac* R)
using *carrier-is-subring inc-is-hom1 ring-hom-ring.img-is-subring* **by** *blast*

1.3.3 Basic Properties of numer, denom, and frac

lemma(**in** *domain-frac*) *numer-denom-facts*:
assumes $a \in$ *carrier* (*Frac* R)
shows $a =$ *frac* (*numer* a) (*denom* a)
(*numer* a) $\in$ *carrier* R
(*denom* a) $\in$ *nonzero* R
$a \neq \mathbf{0}_{Frac\ R} \Longrightarrow$ *numer* $a \neq \mathbf{0}$
$a \otimes_{Frac\ R}$ (*inc* (*denom* a)) = *inc* (*numer* a)
unfolding *fraction-def numerator-def denominator-def Frac-inc-def*
apply (*metis Frac-def assms domain-axioms eq-obj-rng-of-frac.a-is-fraction eq-obj-rng-of-frac-nonzero*
not-nonzero-memI)
apply (*metis Frac-def assms domain-axioms eq-obj-rng-of-frac.numer-denom-facts*(*2*)
eq-obj-rng-of-frac-nonzero nonzero-memE(*2*))
apply (*metis Frac-def assms domain-axioms eq-obj-rng-of-frac.numer-denom-facts*(*3*)
eq-obj-rng-of-frac-nonzero not-nonzero-memI)
apply (*metis Frac-def assms domain-axioms eq-obj-rng-of-frac.numer-denom-facts0*(*4*)
eq-obj-rng-of-frac-nonzero not-nonzero-memI)
by (*metis Frac-def assms domain-axioms eq-obj-rng-of-frac.numer-denom-facts*(*5*)
eq-obj-rng-of-frac-nonzero nonzero-memE(*2*))

lemma(**in** *domain-frac*) *frac-add*:
assumes $a \in$ *carrier* R
assumes $b \in$ *nonzero* R

assumes $c \in carrier\ R$
assumes $d \in nonzero\ R$
shows $(frac\ a\ b) \oplus_{Frac\ R} (frac\ c\ d) = (frac\ ((a \otimes d) \oplus (b \otimes c))\ (b \otimes d))$
using *eq-obj-rng-of-frac.add-fraction*[*of R nonzero R a b c d*]
eq-obj-rng-of-frac-nonzero assms zero-not-in-nonzero[*of R*]
by (*simp add*: *Frac-def domain-axioms fraction-def*)

lemma(**in** *domain-frac*) *frac-mult*:
assumes $a \in carrier\ R$
assumes $b \in nonzero\ R$
assumes $c \in carrier\ R$
assumes $d \in nonzero\ R$
shows $(frac\ a\ b) \otimes_{Frac\ R} (frac\ c\ d) = (frac\ (a \otimes c)\ (b \otimes d))$
by (*simp add*: *Frac-def assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *domain-axioms*
eq-obj-rng-of-frac.mult-fraction eq-obj-rng-of-frac-nonzero fraction-def not-nonzero-memI)

lemma(**in** *domain-frac*) *frac-one*:
assumes $a \in nonzero\ R$
shows $frac\ a\ a = \mathbf{1}_{Frac\ R}$
by (*metis Frac-def assms domain-axioms eq-obj-rng-of-frac.fraction-one' eq-obj-rng-of-frac-nonzero fraction-def nonzero-memE*(*2*))

lemma(**in** *domain-frac*) *frac-closed*:
assumes $a \in carrier\ R$
assumes $b \in nonzero\ R$
shows $frac\ a\ b \in carrier\ (Frac\ R)$
by (*metis Frac-def assms*(*1*) *assms*(*2*) *domain-axioms eq-obj-rng-of-frac.fraction-closed eq-obj-rng-of-frac-nonzero fraction-def nonzero-memE*(*2*))

lemma(**in** *domain-frac*) *nonzero-fraction*:
assumes $a \in nonzero\ R$
assumes $b \in nonzero\ R$
shows $frac\ a\ b \neq \mathbf{0}_{Frac\ R}$
proof
assume $frac\ a\ b = \mathbf{0}_{Frac\ R}$
hence $(frac\ a\ b) \otimes_{Frac\ R} (frac\ b\ a) = \mathbf{0}_{Frac\ R} \otimes_{Frac\ R} (frac\ b\ a)$
by *simp*
hence $(frac\ a\ b) \otimes_{Frac\ R} (frac\ b\ a) = \mathbf{0}_{Frac\ R}$
by (*metis Localization.submonoid.subset assms*(*1*) *assms*(*2*) *cring.cring-simprules*(*26*)
domain.axioms(*1*) *frac-closed fraction-field-is-domain nonzero-is-submonoid subsetCE*)
hence $frac\ (a \otimes b)\ \ (b \otimes a)\ = \mathbf{0}_{Frac\ R}$
by (*metis* (*no-types*, *lifting*) *Localization.submonoid.subset*
assms(*1*) *assms*(*2*) *frac-mult nonzero-is-submonoid subsetCE*)
hence $frac\ (a \otimes b)\ \ (a \otimes b)\ = \mathbf{0}_{Frac\ R}$
by (*metis* (*no-types*, *lifting*) *Localization.submonoid.subset assms*(*1*) *assms*(*2*)
m-comm nonzero-is-submonoid subsetCE)
hence $\mathbf{1}_{Frac\ R} = \mathbf{0}_{Frac\ R}$

```
  using Localization.submonoid.m-closed assms(1) assms(2) frac-one nonzero-is-submonoid
by force
  thus False
    using domain.one-not-zero fraction-field-is-domain by blast
qed

lemma(in comm-monoid) UnitsI:
  assumes a ∈ carrier G
  assumes b ∈ carrier G
  assumes a ⊗ b = 1
  shows a ∈ Units G b ∈ Units G
  unfolding Units-def using comm-monoid-axioms-def assms m-comm[of a b]
  by auto

lemma(in comm-monoid) is-invI:
  assumes a ∈ carrier G
  assumes b ∈ carrier G
  assumes a ⊗ b = 1
  shows inv_G b = a inv_G a = b
  using assms inv-char m-comm
  by auto

lemma(in ring) ring-in-Units-imp-not-zero:
  assumes 1 ≠ 0
  assumes a ∈ Units R
  shows a ≠ 0
  using assms monoid.Units-l-cancel
  by (metis l-null  monoid-axioms one-closed zero-closed)

lemma(in domain-frac) in-Units-imp-not-zero:
  assumes a ∈ Units R
  shows a ≠ 0
  using assms ring-in-Units-imp-not-zero domain-axioms
  by simp

lemma(in domain-frac) units-of-fraction-field0:
  assumes a ∈ carrier (Frac R)
  assumes a ≠ 0_{Frac R}
  shows inv_{Frac R} a = frac (denom a) (numer a)
        a⊗_{Frac R} (inv_{Frac R} a)  = 1_{Frac R}
        (inv_{Frac R} a)⊗_{Frac R}a   = 1_{Frac R}
proof−
  have 0: a ⊗_{Frac R} (frac (denom a) (numer a)) =
    frac ((numer a) ⊗ (denom a)) ((numer a) ⊗ (denom a))
  proof −
    have denom a ∈ nonzero R
      by (simp add: assms(1) numer-denom-facts(3))
    hence frac (numer a) (denom a) ⊗_{Frac R} frac (denom a) (numer a)
      = frac (numer a ⊗ denom a) (denom a ⊗ numer a)
```

by (*simp add*: *assms*(*1*) *assms*(*2*) *domain-frac.numer-denom-facts*(*2*) *domain-frac-axioms frac-mult nonzero-closed nonzero-memI numer-denom-facts*(*4*))

thus *?thesis*
using *assms*(*1*) *numer-denom-facts*(*5*) *domain-frac.numer-denom-facts*(*2*) *domain-axioms m-comm nonzero-closed numer-denom-facts*(*1*)
by (*simp add*: *domain-frac.numer-denom-facts*(*2*) ‹*denominator R a* ∈ *nonzero R*› *domain-frac-axioms*)
qed
have *1*:((*numer a*) ⊗ (*denom a*)) ∈ *nonzero R*
by (*metis assms*(*1*) *assms*(*2*) *domain-frac.numer-denom-facts*(*2*) *domain-frac-axioms local.integral m-closed nonzero-closed nonzero-memI numer-denom-facts*(*3*) *numer-denom-facts*(*4*))
have *2*: *a* ⊗$_{Frac\ R}$ (*frac* (*denom a*) (*numer a*)) = **1**$_{Frac\ R}$
using *0 1 frac-one* **by** *blast*
have *3*: (*frac* (*denom a*) (*numer a*)) ∈ *carrier* (*Frac R*)
by (*simp add*: *assms*(*1*) *assms*(*2*) *frac-closed nonzero-closed nonzero-memI numer-denom-facts*(*2*) *numer-denom-facts*(*3*) *numer-denom-facts*(*4*))
hence *4*: (*frac* (*denom a*) (*numer a*)) ∈ *carrier* (*Frac R*) ∧
a ⊗$_{Frac\ R}$ (*frac* (*denom a*) (*numer a*)) = **1**$_{Frac\ R}$ ∧
(*frac* (*denom a*) (*numer a*)) ⊗$_{Frac\ R}$ *a* = **1**$_{Frac\ R}$
by (*simp add*: *2 assms*(*1*) *cring.cring-simprules*(*14*) *domain.axioms*(*1*) *fraction-field-is-domain*)
thus *5*: *inv*$_{Frac\ R}$ *a* = *frac* (*denom a*) (*numer a*)
using *m-inv-def 2 assms*(*1*) *comm-monoid.comm-inv-char cring-def domain-def fraction-field-is-domain* **by** *fastforce*
thus *6*: *a*⊗$_{Frac\ R}$ (*inv*$_{Frac\ R}$ *a*) = **1**$_{Frac\ R}$
by (*simp add*: *2*)
thus (*inv*$_{Frac\ R}$ *a*)⊗$_{Frac\ R}$*a* = **1**$_{Frac\ R}$
using *assms*
by (*simp add*: *4 5*)
qed

lemma(**in** *domain-frac*) *units-of-fraction-field*:
Units (*Frac R*) = *carrier* (*Frac R*) − {**0**$_{Frac\ R}$}
proof
show *Units* (*Frac R*) ⊆ *carrier* (*Frac R*) − {**0**$_{Frac\ R}$}
proof fix *x* **assume** *A*: *x* ∈ *Units* (*Frac R*)
have *x* ∈ *carrier* (*Frac R*)
using *Units-def* ‹*x* ∈ *Units* (*Frac R*)› **by** *force*
hence *x* ≠ **0**$_{Frac\ R}$
using *fraction-field-is-domain*
by (*simp add*: *A domain-frac.in-Units-imp-not-zero domain-frac.intro*)
thus *x* ∈ *carrier* (*Frac R*) − {**0**$_{Frac\ R}$}
by (*simp add*: ‹*x* ∈ *carrier* (*Frac R*)›)
qed
show *carrier* (*Frac R*) − {**0**$_{Frac\ R}$} ⊆ *Units* (*Frac R*)
proof fix *x* **assume** *A*: *x* ∈ *carrier* (*Frac R*) − {**0**$_{Frac\ R}$}

show $x \in$ *Units* (*Frac R*)
using *comm-monoid.UnitsI*(*1*)[*of Frac R x* $inv_{Frac\ R}$ *x*]
by (*metis A Diff-iff cring.axioms*(*2*) *domain.axioms*(*1*) *domain-frac.numer-denom-facts*(*2*)
domain-frac.numer-denom-facts(*3*) *domain-frac.units-of-fraction-field0*(*1*)
domain-frac.units-of-fraction-field0(*2*) *domain-frac-axioms frac-closed*
fraction-field-is-domain insert-iff nonzero-closed nonzero-memI numer-denom-facts(*4*))

qed
qed

1.4 The Fraction Field as a Field

lemma(**in** *domain-frac*) *fraction-field-is-field*:
field (*Frac R*)
proof(*rule Ring.field.intro*)
show *domain* (*Frac R*) **by**(*auto simp*: *fraction-field-is-domain*)
show *field-axioms* (*Frac R*)
proof
show *Units* (*Frac R*) = *carrier* (*Frac R*) − {$\mathbf{0}_{Frac\ R}$}
using *units-of-fraction-field* **by** *blast*
qed
qed

lemma(**in** *domain-frac*) *frac-inv*:
assumes *a* ∈ *nonzero R*
assumes *b* ∈ *nonzero R*
shows $inv_{Frac\ R}$ (*frac a b*) = (*frac b a*)
using *cring-def*[*of Frac R*] *domain-def*[*of Frac R*] *fraction-field-is-domain*
frac-closed[*of a b*] *frac-closed*[*of b a*] *nonzero-closed*[*of a*]
nonzero-closed[*of b*] *assms comm-monoid.is-invI*(*2*)[*of Frac R frac a b frac b a*]
by (*metis frac-mult frac-one integral-iff m-comm nonzero-memE*(*2*) *nonzero-memI nonzero-mult-closed*)

lemma(**in** *domain-frac*) *frac-inv-id*:
assumes *a* ∈ *nonzero R*
assumes *b* ∈ *nonzero R*
assumes *c* ∈ *nonzero R*
assumes *d* ∈ *nonzero R*
assumes *frac a b* = *frac c d*
shows *frac b a* = *frac d c*
using *frac-inv assms* **by** *metis*

lemma(**in** *domain-frac*) *frac-uminus*:
assumes *a* ∈ *carrier R*
assumes *b* ∈ *nonzero R*
shows $\ominus_{Frac\ R}$ (*frac a b*) = *frac* (⊖ *a*) *b*
proof−
have *frac* (⊖ *a*) *b* $\oplus_{Frac\ R}$ (*frac a b*) = *frac* (((⊖ *a*)⊗*b*) ⊕ (*a* ⊗*b*)) (*b*⊗*b*)

using *frac-add* **by** (*smt Localization.submonoid.subset add.inv-closed assms*(1) *assms*(2) *m-comm nonzero-is-submonoid subsetCE*)
hence *frac* (⊖ *a*) *b* ⊕$_{Frac\ R}$ (*frac a b*) = *frac* (*b* ⊗((⊖ *a*) ⊕ *a*)) (*b*⊗*b*)
by (*metis* (*no-types*, *lifting*) *add.inv-closed assms*(1) *assms*(2) *local.ring-axioms m-comm mem-Collect-eq nonzero-def ringE*(4))
hence *frac* (⊖ *a*) *b* ⊕$_{Frac\ R}$ (*frac a b*) = (*frac* **0** (*b* ⊗*b*))
using *Localization.submonoid.subset assms*(1) *assms*(2) *l-neg nonzero-is-submonoid*
by *fastforce*
hence *frac* (⊖ *a*) *b* ⊕$_{Frac\ R}$ (*frac a b*) = (*frac* **0** **1**) ⊗$_{Frac\ R}$ (*frac* **0** (*b* ⊗*b*))
using *frac-mult* **by** (*smt Localization.submonoid.m-closed Localization.submonoid.one-closed Localization.submonoid.subset assms*(2) *l-one nonzero-is-submonoid r-null subsetCE zero-closed*)
hence *frac* (⊖ *a*) *b* ⊕$_{Frac\ R}$ (*frac a b*) = **0**$_{Frac\ R}$ ⊗$_{Frac\ R}$ (*frac* **0** (*b* ⊗*b*))
using *Frac-def eq-obj-rng-of-frac.fraction-zero*′ *eq-obj-rng-of-frac-nonzero*
by (*simp add*: *Frac-def eq-obj-rng-of-frac.fraction-zero fraction-def*)
hence *frac* (⊖ *a*) *b* ⊕$_{Frac\ R}$ (*frac a b*) = **0**$_{Frac\ R}$
using *fraction-field-is-field*
by (*simp add*: *Localization.submonoid.m-closed assms*(2) *cring.cring-simprules*(26) *domain.axioms*(1) *frac-closed fraction-field-is-domain nonzero-is-submonoid*)
thus *0*: ⊖$_{Frac\ R}$ (*frac a b*) = *frac* (⊖ *a*) *b*
by (*metis add.inv-closed assms*(1) *assms*(2) *cring.sum-zero-eq-neg domain.axioms*(1) *frac-closed fraction-field-is-domain*)
qed

lemma(**in** *domain-frac*) *frac-eqI*:
assumes *a* ∈ *carrier R*
assumes *b* ∈ *nonzero R*
assumes *c* ∈ *carrier R*
assumes *d* ∈ *nonzero R*
assumes *a* ⊗ *d* ⊖ *b* ⊗ *c* = **0**
shows *frac a b* = *frac c d*
proof−
have *frac a b* ⊖$_{Frac\ R}$ *frac c d* = *frac* (*a* ⊗ *d* ⊖ *b* ⊗ *c*) (*b*⊗*d*)
by (*simp add*: *a-minus-def assms*(1) *assms*(2) *assms*(3) *assms*(4) *frac-uminus local.frac-add nonzero-closed r-minus*)
hence *frac a b* ⊖$_{Frac\ R}$ *frac c d* = **0**$_{Frac\ R}$
by (*metis Frac-def assms*(2) *assms*(4) *assms*(5) *eq-obj-rng-of-frac.fraction-zero*′ *eq-obj-rng-of-frac-nonzero fraction-def local.integral nonzero-memE*(1) *nonzero-memE*(2) *nonzero-memI nonzero-mult-closed*)
thus *?thesis*
by (*meson assms*(1) *assms*(2) *assms*(3) *assms*(4) *field.is-ring frac-closed fraction-field-is-field ring.r-right-minus-eq*)
qed

lemma(**in** *domain-frac*) *frac-eqI*′:
assumes *a* ∈ *carrier R*
assumes *b* ∈ *nonzero R*

assumes $c \in$ *carrier* R
assumes $d \in$ *nonzero* R
assumes $a \otimes d = b \otimes c$
shows *frac* $a\ b =$ *frac* $c\ d$
using *assms frac-eqI*[*of* $a\ b\ c\ d$]
by (*simp add*: *nonzero-closed*)

lemma(**in** *domain-frac*) *frac-cancel-l*:
assumes $a \in$*nonzero* R
assumes $b \in$*nonzero* R
assumes $c \in$*carrier* R
shows *frac* $(a \otimes c)\ (a \otimes b) =$ *frac* $c\ b$
proof−
have *0*: *frac* $(a \otimes c)\ (a \otimes b) =$(*frac* $b\ b) \otimes_{Frac\ R}$ (*frac* $c\ b$)
by (*metis* (*no-types*, *lifting*) *assms*(*1*) *assms*(*2*) *assms*(*3*)
frac-mult frac-one mem-Collect-eq nonzero-def)
have *1*: *frac* $b\ b = \mathbf{1}_{Frac\ R}$
by (*simp add*: *assms*(*2*) *frac-one*)
show *?thesis* **using** *0 1*
using *Frac-def assms*(*2*) *assms*(*3*) *eq-obj-rng-of-frac.rng-rng-of-frac*
eq-obj-rng-of-frac-nonzero frac-closed ring.ring-simprules(*12*)
by (*simp add*: *Frac-def eq-obj-rng-of-frac.rng-rng-of-frac ring.ring-simprules*(*12*))
qed

lemma(**in** *domain-frac*) *frac-cancel-r*:
assumes $a \in$*nonzero* R
assumes $b \in$*nonzero* R
assumes $c \in$*carrier* R
shows *frac* $(c \otimes a)\ (b \otimes a) =$ *frac* $c\ b$
by (*metis* (*no-types*, *lifting*) *Localization.submonoid.subset assms*(*1*)
assms(*2*) *assms*(*3*) *frac-cancel-l m-comm nonzero-is-submonoid subsetCE*)

lemma(**in** *domain-frac*) *frac-cancel-lr*:
assumes $a \in$*nonzero* R
assumes $b \in$*nonzero* R
assumes $c \in$*carrier* R
shows *frac* $(a \otimes c)\ (b \otimes a) =$ *frac* $c\ b$
by (*metis* (*no-types*, *lifting*) *Localization.submonoid.subset assms*(*1*)
assms(*2*) *assms*(*3*) *frac-cancel-l m-comm nonzero-is-submonoid subsetCE*)

lemma(**in** *domain-frac*) *frac-cancel-rl*:
assumes $a \in$*nonzero* R
assumes $b \in$*nonzero* R
assumes $c \in$*carrier* R
shows *frac* $(c \otimes a)\ (a \otimes b) =$ *frac* $c\ b$
by (*metis* (*no-types*, *lifting*) *Localization.submonoid.subset assms*(*1*)
assms(*2*) *assms*(*3*) *frac-cancel-l m-comm nonzero-is-submonoid subsetCE*)

lemma(**in** *domain-frac*) *i-mult*:

assumes $a \in carrier\ R$
assumes $c \in carrier\ R$
assumes $d \in nonzero\ R$
shows $(\iota\ a) \otimes_{Frac\ R} (frac\ c\ d) = frac\ (a \otimes c)\ d$
proof−
have $(\iota\ a) \otimes_{Frac\ R} (frac\ c\ d) = (frac\ a\ \mathbf{1}) \otimes_{Frac\ R} (frac\ c\ d)$
by (*simp add*: *assms*(*1*) *inc-equation*)
thus *?thesis*
by (*metis* (*mono-tags*, *lifting*) *assms*(*1*) *assms*(*2*) *assms*(*3*) *cring-simprules*(*12*)
cring-simprules(*6*) *frac-mult local.one-not-zero mem-Collect-eq nonzero-def*)
qed

lemma(**in** *domain-frac*) *frac-eqE*:
assumes $a \in carrier\ R$
assumes $b \in nonzero\ R$
assumes $c \in carrier\ R$
assumes $d \in nonzero\ R$
assumes $frac\ a\ b = frac\ c\ d$
shows $a \otimes d = b \otimes c$
proof−
have $(\iota\ b) \otimes_{Frac\ R} (frac\ a\ b) = (\iota\ b) \otimes_{Frac\ R} (frac\ c\ d)$
by (*simp add*: *assms*(*5*))
hence $(frac\ (a \otimes b)\ \ b) = (frac\ (c \otimes b)\ d)$
using *i-mult*
by (*metis* (*no-types*, *lifting*) *Localization.submonoid.subset assms*(*1*)
assms(*2*) *assms*(*3*) *assms*(*4*) *m-comm nonzero-is-submonoid subsetCE*)
hence $(frac\ a\ \ \mathbf{1}) = (frac\ (c \otimes b)\ d)$
by (*smt assms*(*1*) *assms*(*2*) *frac-cancel-r l-one mem-Collect-eq nonzero-def*
one-closed zero-not-one)
hence $(\iota\ d) \otimes_{Frac\ R} (frac\ a\ \ \mathbf{1}) = (\iota\ d) \otimes_{Frac\ R} (frac\ (c \otimes b)\ d)$
by *auto*
hence $(frac\ (a \otimes d)\ \mathbf{1}) = (frac\ ((c \otimes b) \otimes d)\ d)$
using *i-mult*
by (*smt Localization.submonoid.m-closed Localization.submonoid.subset assms*(*1*)
assms(*2*) *assms*(*3*)
assms(*4*) *cring-simprules*(*27*) *cring-simprules*(*6*) *local.one-not-zero m-comm*
mem-Collect-eq nonzero-def nonzero-is-submonoid)
hence $(frac\ (a \otimes d)\ \mathbf{1}) = (frac\ (c \otimes b)\ \mathbf{1})$
by (*smt Localization.submonoid.subset assms*(*2*) *assms*(*3*) *assms*(*4*) *cring-simprules*(*5*)
cring-simprules(*6*) *frac-one i-mult inc-equation inc-is-hom nonzero-is-submonoid*
r-one ring-hom-mult ring-hom-one subsetCE)
thus *?thesis* **using** *assms*
unfolding *fraction-def*
by (*simp add*: *fraction-def inc-equation inc-inj2 m-comm nonzero-closed*)
qed

lemma(**in** *domain-frac*) *frac-add-common-denom*:
assumes $a \in carrier\ R$

assumes $b \in carrier\ R$
assumes $c \in nonzero\ R$
shows $(frac\ a\ c) \oplus_{Frac\ R} (frac\ b\ c) = frac\ (a \oplus b)\ c$
proof−
have $(frac\ a\ c) \oplus_{Frac\ R} (frac\ b\ c) = frac\ ((a \otimes c) \oplus (b \otimes c))\ (c \otimes c)$
using *assms*(*1*) *assms*(*2*) *assms*(*3*) *domain-frac.frac-add domain-axioms frac-eqE local.frac-add*
by *auto*
hence $(frac\ a\ c) \oplus_{Frac\ R} (frac\ b\ c) = frac\ ((a \oplus b) \otimes c)\ (c \otimes c)$
by (*metis Localization.submonoid.subset assms*(*1*) *assms*(*2*) *assms*(*3*) *l-distr nonzero-is-submonoid subsetCE*)
thus *?thesis*
by (*simp add*: *assms*(*1*) *assms*(*2*) *assms*(*3*) *frac-cancel-r*)
qed

lemma(**in** *domain-frac*) *common-denominator*:
assumes $x \in carrier\ (Frac\ R)$
assumes $y \in carrier\ (Frac\ R)$
obtains *a b c* **where**
$a \in carrier\ R$
$b \in carrier\ R$
$c \in nonzero\ R$
$x = frac\ a\ c$
$y = frac\ b\ c$
proof−
obtain *x1 x2* **where** *X1*: $x1 \in carrier\ R$ **and** *X2*: $x2 \in nonzero\ R$ **and** *Fx*: $x = frac\ x1\ x2$
by (*meson assms*(*1*) *numer-denom-facts*(*1*) *numer-denom-facts*(*2*) *numer-denom-facts*(*3*))

obtain *y1 y2* **where** *Y1*: $y1 \in carrier\ R$ **and** *Y2*: $y2 \in nonzero\ R$ **and** *Fy*: $y = frac\ y1\ y2$
by (*meson assms*(*2*) *numer-denom-facts*(*1*) *numer-denom-facts*(*2*) *numer-denom-facts*(*3*))

let $?a = x1 \otimes y2$
let $?b = y1 \otimes x2$
let $?c = x2 \otimes y2$
have *0*: $?a \in carrier\ R$
using *X1 Y2* **by** (*simp add*: *nonzero-def*)
have *1*: $?b \in carrier\ R$ **using** *Y1 X2*
by (*simp add*: *nonzero-def*)
have *2*: $?c \in nonzero\ R$ **using** *X2 Y2*
by (*simp add*: *Localization.submonoid.m-closed nonzero-is-submonoid*)
have *3*: $x = frac\ ?a\ ?c$
using *Fx X1 X2 Y2 frac-cancel-r* **by** *auto*
have *4*: $y = frac\ ?b\ ?c$
using *Fy X2 Y1 Y2 frac-cancel-rl* **by** *auto*
thus *?thesis* **using** *0 1 2 3 4*
using *that* **by** *blast*
qed

sublocale *domain-frac* < *F*: *field Frac R*
by (*simp add*: *fraction-field-is-field*)

Inclusions of natural numbers into `(Frac R)`:

lemma(**in** *domain-frac*) *nat-0*:
$[(0::nat)]\cdot\mathbf{1} = \mathbf{0}$
by *simp*

lemma(**in** *domain-frac*) *nat-suc*:
$[Suc\ n]\cdot\mathbf{1} = \mathbf{1} \oplus [n]\cdot\mathbf{1}$
using *add.nat-pow-Suc2* **by** *auto*

lemma(**in** *domain-frac*) *nat-inc-rep*:
fixes *n*::*nat*
shows $[n]\cdot_{Frac\ R}\ \mathbf{1}_{Frac\ R} = frac\ ([n]\cdot\mathbf{1})\ \mathbf{1}$
proof(*induction n*)
case *0*
have $[(0::nat)]\ \cdot_{Frac\ R}\ \mathbf{1}_{Frac\ R} = \mathbf{0}_{Frac\ R}$
using *fraction-field-is-domain*
by (*simp add*: *domain-frac.intro domain-frac.nat-0*)
thus *?case*
by (*simp add*: *Frac-def eq-obj-rng-of-frac.fraction-zero eq-obj-rng-of-frac-nonzero fraction-def*)
next
case (*Suc n*)
assume *A*: $[n]\ \cdot_{Frac\ R}\ \mathbf{1}_{Frac\ R} = frac\ ([n]\cdot\mathbf{1})\ \mathbf{1}$
have $[Suc\ n]\ \cdot_{Frac\ R}\ \mathbf{1}_{Frac\ R} = \mathbf{1}_{Frac\ R} \oplus_{Frac\ R} [n]\ \cdot_{Frac\ R}\ \mathbf{1}_{Frac\ R}$
using *F.add.nat-pow-Suc2* **by** *auto*
hence $[Suc\ n]\ \cdot_{Frac\ R}\ \mathbf{1}_{Frac\ R} = (frac\ \mathbf{1}\ \mathbf{1}) \oplus_{Frac\ R} (frac\ ([n]\cdot\mathbf{1})\ \mathbf{1})$
by (*simp add*: *Suc.IH frac-one nonzero-def*)
hence $[Suc\ n]\ \cdot_{Frac\ R}\ \mathbf{1}_{Frac\ R} = (frac\ (\mathbf{1}\oplus[n]\cdot\mathbf{1})\ \mathbf{1})$
by (*simp add*: *frac-add-common-denom nonzero-def*)
thus *?case*
using *nat-suc* **by** *auto*
qed

lemma(**in** *domain-frac*) *pow-0*:
assumes $a \in nonzero\ R$
shows $a[\hat{}](0::nat) = \mathbf{1}$
by *simp*

lemma(**in** *domain-frac*) *pow-suc*:
assumes $a \in carrier\ R$
shows $a[\hat{}](Suc\ n) = a \otimes (a[\hat{}]n)$
using *assms nat-pow-Suc2* **by** *auto*

lemma(**in** *domain-frac*) *nat-inc-add*:
$[((n::nat) + (m::nat))]\cdot\mathbf{1} = [n]\cdot\mathbf{1} \oplus [m]\cdot\mathbf{1}$

using *domain-def add-pow-def*
by (*simp add*: *add.nat-pow-mult*)

lemma(**in** *domain-frac*) *int-inc-add*:
$[((n::int) + (m::int))]\cdot\mathbf{1} = [n]\cdot\mathbf{1} \oplus [m]\cdot\mathbf{1}$
using *domain-def add-pow-def*
by (*simp add*: *add.int-pow-mult*)

lemma(**in** *domain-frac*) *nat-inc-mult*:
$[((n::nat) * (m::nat))]\cdot\mathbf{1} = [n]\cdot\mathbf{1} \otimes [m]\cdot\mathbf{1}$
using *domain-def add-pow-def*
by (*simp add*: *Groups.mult-ac*(*2*) *add.nat-pow-pow add-pow-ldistr*)

lemma(**in** *domain-frac*) *int-inc-mult*:
$[((n::int) * (m::int))]\cdot\mathbf{1} = [n]\cdot\mathbf{1} \otimes [m]\cdot\mathbf{1}$
using *domain-def add-pow-def*
by (*simp add*: *Groups.mult-ac*(*2*) *add.int-pow-pow add-pow-ldistr-int*)

1.5 Facts About Ring Units

lemma(**in** *ring*) *Units-nonzero*:
assumes $u \in Units\ R$
assumes $\mathbf{1}_R \neq \mathbf{0}_R$
shows $u \in nonzero\ R$
proof−
have $u \in carrier\ R$
using *Units-closed assms* **by** *auto*
have $u \neq \mathbf{0}$
using *Units-r-inv-ex assms*(*1*) *assms*(*2*)
by *force*
thus *?thesis*
by (*simp add*: ‹$u \in carrier\ R$› *nonzero-def*)
qed

lemma(**in** *ring*) *Units-inverse*:
assumes $u \in Units\ R$
shows $inv\ u \in Units\ R$
by (*simp add*: *assms*)

lemma(**in** *cring*) *invI*:
assumes $x \in carrier\ R$
assumes $y \in carrier\ R$
assumes $x \otimes_R y = \mathbf{1}_R$
shows $y = inv_R\ x$
$x = inv_R\ y$
using *assms*(*1*) *assms*(*2*) *assms*(*3*) *is-invI*
by *auto*

lemma(**in** *cring*) *inv-cancelR*:

assumes $x \in Units\ R$
assumes $y \in carrier\ R$
assumes $z \in carrier\ R$
assumes $y = x \otimes_R z$
shows $inv_R\ x \otimes_R y = z$
$y \otimes_R (inv_R\ x) = z$
apply (*metis Units-closed assms*(*1*) *assms*(*3*) *assms*(*4*) *cring.cring-simprules*(*12*)

is-cring m-assoc monoid.Units-inv-closed monoid.Units-l-inv monoid-axioms)
by (*metis Units-closed assms*(*1*) *assms*(*3*) *assms*(*4*) *m-assoc m-comm monoid.Units-inv-closed*

monoid.Units-r-inv monoid.r-one monoid-axioms)

lemma(**in** *cring*) *inv-cancelL*:
assumes $x \in Units\ R$
assumes $y \in carrier\ R$
assumes $z \in carrier\ R$
assumes $y = z \otimes_R x$
shows $inv_R\ x \otimes_R y = z$
$y \otimes_R (inv_R\ x) = z$
apply (*simp add*: *Units-closed assms*(*1*) *assms*(*3*) *assms*(*4*) *m-lcomm*)
by (*simp add*: *Units-closed assms*(*1*) *assms*(*3*) *assms*(*4*) *m-assoc*)

lemma(**in** *domain-frac*) *nat-pow-nonzero*:
assumes $x \in nonzero\ R$
shows $x[\widehat{}](n::nat) \in nonzero\ R$
unfolding *nonzero-def*
apply(*induction n*)
using *assms integral-iff nonzero-closed zero-not-in-nonzero* **by** *auto*

lemma(**in** *monoid*) *Units-int-pow-closed*:
assumes $x \in Units\ G$
shows $x[\widehat{}](n::int) \in Units\ G$
by (*metis Units-pow-closed assms int-pow-def2 monoid.Units-inv-Units monoid-axioms*)

1.6 Facts About Fraction Field Units

lemma(**in** *domain-frac*) *frac-nonzero-Units*:
$nonzero\ (Frac\ R) = Units\ (Frac\ R)$
unfolding *nonzero-def* **using** *F.field-Units*
by *blast*

lemma(**in** *domain-frac*) *frac-nonzero-inv-Unit*:
assumes $b \in nonzero\ (Frac\ R)$
shows $inv_{Frac\ R}\ b \in Units\ (Frac\ R)$
using *assms frac-nonzero-Units*
by *simp*

lemma(**in** *domain-frac*) *frac-nonzero-inv-closed*:

assumes $b \in nonzero\ (Frac\ R)$
shows $inv_{Frac\ R}\ b \in carrier\ (Frac\ R)$
using *frac-nonzero-inv-Unit*
by (*simp add*: *Units-def assms*)

lemma(**in** *domain-frac*) *frac-nonzero-inv*:
assumes $b \in nonzero\ (Frac\ R)$
shows $b \otimes_{Frac\ R} inv_{Frac\ R}\ b = \mathbf{1}_{Frac\ R}$
$inv_{Frac\ R}\ b \otimes_{Frac\ R} b = \mathbf{1}_{Frac\ R}$
using *frac-nonzero-Units assms* **by** *auto*

lemma(**in** *domain-frac*) *fract-cancel-right*[*simp*]:
assumes $a \in carrier\ (Frac\ R)$
assumes $b \in nonzero\ (Frac\ R)$
shows $b \otimes_{Frac\ R} (a \otimes_{Frac\ R} inv_{Frac\ R}\ b) = a$
by (*metis F.Units-inv-inv F.inv-cancelL*(*1*) *F.m-closed assms*(*1*) *assms*(*2*) *frac-nonzero-Units frac-nonzero-inv-Unit frac-nonzero-inv-closed*)

lemma(**in** *domain-frac*) *fract-cancel-left*[*simp*]:
assumes $a \in carrier\ (Frac\ R)$
assumes $b \in nonzero\ (Frac\ R)$
shows $(a \otimes_{Frac\ R} inv_{Frac\ R}\ b) \otimes_{Frac\ R} b = a$
by (*metis Diff-iff assms*(*1*) *assms*(*2*) *cring.cring-simprules*(*14*) *cring.cring-simprules*(*5*)

domain.axioms(*1*) *frac-nonzero-Units frac-nonzero-inv-closed fract-cancel-right*

fraction-field-is-domain units-of-fraction-field)

lemma(**in** *domain-frac*) *fract-mult-inv*:
assumes $b \in nonzero\ (Frac\ R)$
assumes $d \in nonzero\ (Frac\ R)$
shows $(inv_{Frac\ R}\ b) \otimes_{Frac\ R} (inv_{Frac\ R}\ d) = (inv_{Frac\ R}\ (b \otimes_{Frac\ R} d))$
by (*metis F.Units-inv-closed F.Units-m-closed F.inv-cancelR*(*2*) *F.nonzero-closed assms*(*1*) *assms*(*2*) *frac-nonzero-Units*)

lemma(**in** *domain-frac*) *fract-mult*:
assumes $a \in carrier\ (Frac\ R)$
assumes $b \in nonzero\ (Frac\ R)$
assumes $c \in carrier\ (Frac\ R)$
assumes $d \in nonzero\ (Frac\ R)$
shows $(a \otimes_{Frac\ R} inv_{Frac\ R}\ b) \otimes_{Frac\ R} (c \otimes_{Frac\ R} inv_{Frac\ R}\ d) = ((a \otimes_{Frac\ R} c) \otimes_{Frac\ R} inv_{Frac\ R}\ (b \otimes_{Frac\ R} d))$
using *F.m-assoc F.m-lcomm assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *frac-nonzero-Units fract-mult-inv* **by** *auto*

lemma(**in** *domain-frac*) *Frac-nat-pow-nonzero*:
assumes $x \in nonzero\ (Frac\ R)$
shows $x[\hat{}]_{Frac\ R}(n::nat) \in nonzero\ (Frac\ R)$
by (*simp add*: *assms domain-frac.intro domain-frac.nat-pow-nonzero fraction-field-is-domain*)

lemma(**in** *domain-frac*) *Frac-nonzero-nat-pow*:
assumes $x \in$ *carrier* (*Frac R*)
assumes $n > 0$
assumes $x[\widehat{}]_{Frac\ R}(n::nat) \in$ *nonzero* (*Frac R*)
shows $x \in$ *nonzero* (*Frac R*)
proof(*rule ccontr*)
assume $x \notin$ *nonzero* (*Frac R*)
hence *0*: $x = \mathbf{0}_{Frac\ R}$
by (*simp add*: *assms*(*1*) *nonzero-def*)
have $x[\widehat{}]_{Frac\ R}(n::nat) = \mathbf{0}_{Frac\ R}$
proof−
obtain k **where** $n = Suc\ k$
using *assms*(*2*) *lessE* **by** *blast*
hence *00*: $x[\widehat{}]_{Frac\ R}(n::nat) = x[\widehat{}]_{Frac\ R} k \otimes_{Frac\ R} x$
by *simp*
have $x[\widehat{}]_{Frac\ R} k \in$ *carrier* (*Frac R*)
using *assms monoid.nat-pow-closed*[*of Frac R x k*]
by (*meson field.is-ring fraction-field-is-field ring-def*)
thus *?thesis* **using** *0 assms*
using *00 cring.cring-simprules*(*27*) *domain.axioms*(*1*) *fraction-field-is-domain*
by *fastforce*
qed
thus *False*
using *0* ‹$x \notin$ *nonzero* (*Frac R*)› *assms*(*3*) **by** *auto*
qed

lemma(**in** *domain-frac*) *Frac-int-pow-nonzero*:
assumes $x \in$ *nonzero* (*Frac R*)
shows $x[\widehat{}]_{Frac\ R}(n::int) \in$ *nonzero* (*Frac R*)
using *assms field.is-ring frac-nonzero-Units fraction-field-is-field monoid.Units-pow-closed*[*of Frac R x*]
by (*simp add*: *field.is-ring monoid.Units-int-pow-closed ring.is-monoid*)

lemma(**in** *domain-frac*) *Frac-nonzero-int-pow*:
assumes $x \in$ *carrier* (*Frac R*)
assumes $n > 0$
assumes $x[\widehat{}]_{Frac\ R}(n::int) \in$ *nonzero* (*Frac R*)
shows $x \in$ *nonzero* (*Frac R*)
by (*metis* (*mono-tags*, *opaque-lifting*) *Frac-nonzero-nat-pow assms int-pow-int pos-int-cases*)

lemma(**in** *domain-frac*) *numer-denom-frac*[*simp*]:
assumes $a \in$ *nonzero R*
assumes $b \in$ *nonzero R*
shows *frac* (*numer* (*frac a b*)) (*denom* (*frac a b*)) = *frac a b*
using *assms*(*1*) *assms*(*2*) *domain-frac.numer-denom-facts*(*1*) *domain-axioms frac-closed nonzero-closed numer-denom-facts*(*1*) **by** *auto*

lemma(**in** *domain-frac*) *numer-denom-swap*:
assumes $a \in$ *nonzero* R
assumes $b \in$ *nonzero* R
shows $a \otimes$ (*denom* (*frac* a b)) = $b \otimes$ (*numer* (*frac* a b))
using *numer-denom-frac*[*of* a b] *assms*
by (*simp add*: *frac-closed frac-eqE nonzero-closed numer-denom-facts*(*2*) *numer-denom-facts*(*3*))

lemma(**in** *domain-frac*) *numer-nonzero*:
assumes $a \in$ *nonzero* (*Frac* R)
shows *numer* $a \in$ *nonzero* R
using *assms* *numer-denom-facts*(*4*)[*of* a] *zero-not-in-nonzero*[*of* R]
by (*simp add*: *frac-nonzero-Units nonzero-memI numer-denom-facts*(*2*) *units-of-fraction-field*)

lemma(**in** *domain-frac*) *fraction-zero*[*simp*]:
assumes $b \in$ *nonzero* R
shows *frac* $\mathbf{0}$ $b = \mathbf{0}_{Frac\ R}$
by (*metis Frac-def assms eq-obj-rng-of-frac.fraction-zero' eq-obj-rng-of-frac-nonzero fraction-def nonzero-memE*(*2*))

end
theory *Cring-Multivariable-Poly*
imports *HOL−Algebra.Indexed-Polynomials Padic-Ints.Cring-Poly*
begin

2 Multivariable Polynomials Over a Commutative Ring

This theory extends the content of `HOL-Algebra.Indexed_Polynomials`, mainly in the context of a commutative base ring. In particular, the ring of polynomials over an arbitrary variable set is explicitly witnessed as a ring itself, which is commutative if the base is commutative, and a domain if the base ring is a domain. A universal property for polynomial evaluation is proved, which allows us to embed polynomial rings in a ring of functions over the base ring, as well as construe multivariable polynomials as univariate polynomials over a small base polynomial ring.

type-synonym *'a monomial* = *'a multiset*
type-synonym (*'b*,*'a*) *mvar-poly* = *'a multiset* ⇒ *'b*
type-synonym (*'a*,*'b*) *ring-hom* = *'a* ⇒ *'b*
type-synonym *'a u-poly* = *nat* ⇒ *'a*

2.1 Lemmas about multisets

Since multisets function as monomials in this formalization, we'll need some simple lemmas about multisets in order to define degree functions and certain lemmas about factorizations. Those are provided in this section.

lemma *count-size*:
assumes *size* $m \leq K$
shows *count* m $i \leq K$
using *assms*
by (*metis count-le-replicate-mset-subset-eq dual-order.trans order-refl size-mset-mono size-replicate-mset*)

The following defines the set of monomials with nonzero coefficients for a given polynomial:

definition *monomials-of* :: ('*a*,'*c*) *ring-scheme* ⇒ ('*a*, '*b*) *mvar-poly* ⇒ ('*b monomial*) *set* **where**
monomials-of R $P = \{m.\ P\ m \neq \mathbf{0}_R\}$

context *ring*
begin

lemma *monomials-of-index-free*:
assumes *index-free* P i
assumes $m \in$ *monomials-of* R P
shows *count* m $i = 0$
using *assms*
unfolding *monomials-of-def index-free-def*
by (*meson count-inI mem-Collect-eq*)

lemma *index-freeI*:
assumes $\bigwedge m.\ m \in$ *monomials-of* R $P \Longrightarrow$ *count* m $i = 0$
shows *index-free* P i
unfolding *index-free-def*
proof *safe*
fix m
assume $i \in\# m$
then have *count* m $i \neq 0$
by *simp*
then have $m \notin$ *monomials-of* R P
using *assms*
by *meson*
then show P $m = \mathbf{0}$
unfolding *monomials-of-def*
by *blast*
qed

`monomials_of` R is subadditive

lemma *monomials-of-add*:
monomials-of R $(P \bigoplus Q) \subseteq$ (*monomials-of* R P) ∪ (*monomials-of* R Q)
proof
fix x
assume $x \in$ *monomials-of* R $(P \bigoplus Q)$
then have $P\ x \oplus Q\ x \neq \mathbf{0}$
by (*simp add*: *indexed-padd-def monomials-of-def*)

```
  then have P x ≠0 ∨ Q x ≠ 0
    by auto
  then show x ∈ (monomials-of R P) ∪ (monomials-of R Q)
    by (simp add: monomials-of-def)
qed

lemma monomials-of-add-finite:
  assumes finite (monomials-of R P)
  assumes finite (monomials-of R Q)
  shows finite (monomials-of R (P ⊕ Q))
  by (meson assms(1) assms(2) finite-Un finite-subset monomials-of-add)

lemma monomials-ofE:
  assumes m ∈ monomials-of R p
  shows p m ≠ 0
  using assms
  unfolding monomials-of-def
  by simp

lemma complement-of-monomials-of:
  assumes m ∉ monomials-of R P
  shows P m = 0
  using assms
  unfolding monomials-of-def
  by blast
```

Multiplication by an indexed variable corresponds to adding that index to each monomial:

```
lemma monomials-of-p-mult:
monomials-of R (P ⨂ i) = {m. ∃ n ∈ (monomials-of R P). m = add-mset i n}
proof
  show monomials-of R (P ⨂ i) ⊆ {m. ∃n∈monomials-of R P. m = add-mset i
n}
    proof
      fix m
      assume A: m ∈ monomials-of R (P ⨂ i)
      show m ∈ {m. ∃ n∈monomials-of R P. m = add-mset i n}
      proof−
        have (P ⨂ i) m ≠ 0
          by (simp add: A monomials-ofE)
        then have P (m − {# i #}) ≠0
          unfolding indexed-pmult-def
          by presburger
        then have 0: (m − {# i #}) ∈ monomials-of R P
          by (meson complement-of-monomials-of)
        have 1:  m = add-mset i  (m − {# i #})
         by (metis ‹(P ⨂ i) m ≠ 0› add-mset-remove-trivial-If indexed-pmult-def)
        then show ?thesis using 0 1
          by blast
```

qed
qed
show $\{m.\ \exists n{\in}$*monomials-of R P*$.\ m =$ *add-mset i n*$\} \subseteq$ *monomials-of R* $(P \bigotimes i)$
unfolding *monomials-of-def indexed-pmult-def*
by *auto*
qed

lemma *monomials-of-p-mult'*:
monomials-of R $(p \bigotimes i)$ = *add-mset i* ‘ (*monomials-of R p*)
using *monomials-of-p-mult*
by (*simp add*: *monomials-of-p-mult image-def*)

lemma *monomials-of-p-mult-finite*:
assumes *finite* (*monomials-of R P*)
shows *finite* (*monomials-of R* $(P \bigotimes i)$)
using *assms monomials-of-p-mult'*[*of P i*]
by *simp*

Monomials of a constant either consist of the empty multiset, or nothing:

lemma *monomials-of-const*:
(*monomials-of R* (*indexed-const k*)) = (*if* $(k = \mathbf{0})$ *then* {} *else* {{#}})
unfolding *monomials-of-def indexed-const-def*
by *simp*

lemma *monomials-of-const-finite*:
finite (*monomials-of R* (*indexed-const k*))
by (*simp add*: *monomials-of-const*)

A polynomial always has finitely many monomials:

lemma *monomials-finite*:
assumes $P \in$ *indexed-pset K I*
shows *finite* (*monomials-of R P*)
using *assms*
apply(*induction P*)
using *monomials-of-const-finite* **apply** *blast*
using *monomials-of-add-finite* **apply** *blast*
by (*simp add*: *monomials-of-p-mult-finite*)
end

2.2 Turning monomials into polynomials

Constructor for turning a monomial into a polynomial

definition *mset-to-IP* :: (*'a*, *'b*) *ring-scheme* ⇒ *'c monomial* ⇒ (*'a*,*'c*) *mvar-poly*
where
mset-to-IP R m n= (*if* $(n = m)$ *then* $\mathbf{1}_R$ *else* $\mathbf{0}_R$)

definition *var-to-IP* :: (*'a*, *'b*) *ring-scheme* ⇒ *'c* ⇒ (*'a*,*'c*) *mvar-poly* (‹*pvar*›)
where

var-to-IP R i = mset-to-IP R {#i#}

context *ring*
begin

lemma *mset-to-IP-simp*[*simp*]:
mset-to-IP R m m = **1**
by (*simp add*: *mset-to-IP-def*)

lemma *mset-to-IP-simp'*[*simp*]:
assumes *n* ≠*m*
shows *mset-to-IP R m n* = **0**
by (*simp add*: *assms mset-to-IP-def*)

lemma(**in** *cring*) *monomials-of-mset-to-IP*:
assumes 1 ≠**0**
shows *monomials-of R* (*mset-to-IP R m*) = {*m*}
unfolding *monomials-of-def mset-to-IP-def*
proof
show {*ma*. (*if ma* = *m then* **1** *else* **0**) ≠ **0**} ⊆ {*m*}
by *auto*
show {*m*} ⊆ {*ma*. (*if ma* = *m then* **1** *else* **0**) ≠ **0**}
using *assms* **by** *auto*
qed

end

The set of monomials of a fixed P which include a given variable:

definition *monomials-with* :: ('*a*, '*b*) *ring-scheme* ⇒ '*c* ⇒ ('*a*,'*c*) *mvar-poly* ⇒ ('*c monomial*) *set* **where**
monomials-with R i P = {*m*. *m* ∈ *monomials-of R P* ∧ *i* ∈# *m*}

context *ring*
begin

lemma *monomials-withE*:
assumes *m* ∈ *monomials-with R i P*
shows *i* ∈# *m*
m ∈ *monomials-of R P*
using *assms* **unfolding** *monomials-with-def*
apply *blast*
using *assms* **unfolding** *monomials-with-def*
by *blast*

lemma *monomials-withI*:
assumes *i* ∈# *m*
assumes *m* ∈ *monomials-of R P*
shows *m* ∈ *monomials-with R i P*
using *assms*

unfolding *monomials-with-def*
by *blast*

end

Restricting a polynomial to be zero outside of a given monomial set:

definition *restrict-poly-to-monom-set* ::
(*'a*, *'b*) *ring-scheme* ⇒ (*'a*,*'c*) *mvar-poly* ⇒ (*'c monomial*) *set* ⇒(*'a*,*'c*) *mvar-poly*
where
restrict-poly-to-monom-set R P m-set m = (*if m* ∈ *m-set then P m else* $\mathbf{0}_R$)

context *ring*
begin

lemma *restrict-poly-to-monom-set-coeff*:
assumes *carrier-coeff P*
shows *carrier-coeff* (*restrict-poly-to-monom-set R P Ms*)
by (*metis assms carrier-coeff-def restrict-poly-to-monom-set-def zero-closed*)

lemma *restrict-poly-to-monom-set-coeff'*:
assumes *P* ∈ *indexed-pset K I*
assumes *I* ≠ {}
assumes ⋀*m*. *P m* ∈ *S*
assumes **0** ∈ *S*
shows ⋀*m*. (*restrict-poly-to-monom-set R P m-set m*) ∈ *S*
using *assms*
unfolding *restrict-poly-to-monom-set-def*
by *simp*

lemma *restrict-poly-to-monom-set-monoms*:
assumes *P* ∈ *indexed-pset I K*
assumes *m-set* ⊆ *monomials-of R P*
shows *monomials-of R* (*restrict-poly-to-monom-set R P m-set*) = *m-set*
using *assms*
unfolding *monomials-of-def restrict-poly-to-monom-set-def*
by (*simp add*: *subset-iff*)
end

2.3 Degree Functions

2.3.1 Total Degree Function

lemma *multi-set-size-count*:
fixes *m* :: *'c monomial*
shows *size m* ≥ *count m i*
by (*metis count-le-replicate-mset-subset-eq order-refl size-mset-mono size-replicate-mset*)

Total degree function

definition *total-degree* :: (*'a*, *'b*) *ring-scheme* ⇒ (*'a*, *'c*) *mvar-poly* ⇒ *nat* **where**

total-degree R P = (*if* (*monomials-of R P* = {}) *then 0 else Max* (*size* ‘ (*monomials-of R P*)))

context *ring*
begin

lemma *total-degree-ineq*:
assumes $m \in$ *monomials-of R P*
assumes *finite* (*monomials-of R P*)
shows *total-degree R P* $\geq$ *size m*
unfolding *total-degree-def* **using** *assms*
by *force*

lemma *total-degree-in-monomials-of*:
assumes *monomials-of R P* $\neq$ {}
assumes *finite* (*monomials-of R P*)
obtains *m* **where** $m \in$ *monomials-of R P* $\wedge$ *size m* = *total-degree R P*
using *assms Max-in*[*of* (*size* ‘ *monomials-of R P*)] **unfolding** *total-degree-def*
by (*metis* (*mono-tags*, *lifting*) *empty-is-image finite-imageI image-iff*)

lemma *total-degreeI*:
assumes *finite* (*monomials-of R P*)
assumes $\exists$ *m*. $m \in$ *monomials-of R P* $\wedge$ *size m* = *n*
assumes $\bigwedge$*m*. $m \in$ *monomials-of R P* $\Longrightarrow$ *size m* $\leq$ *n*
shows *n* = *total-degree R P*
proof(*cases monomials-of R P* = {})
case *True*
then show *?thesis*
using *assms* **by** *blast*
next
case *False*
obtain *m* **where** *m-def*: $m \in$ *monomials-of R P* $\wedge$ *size m* = *n*
using *assms* **by** *blast*
have *0*: *n* $\in$ (*size* ‘ (*monomials-of R P*))
using *m-def*
by *blast*
have $\bigwedge$*k*. *k* $\in$ (*size* ‘ (*monomials-of R P*)) $\Longrightarrow$ *k* $\leq$ *n*
using *assms*
by *blast*
then have *1*: $\bigwedge$*m*. $m \in$ (*monomials-of R P*) $\Longrightarrow$ *size m*$\leq$ *n*
by *blast*
obtain *m′* **where** *m′-def*: *m′* $\in$ *monomials-of R P* $\wedge$ *size m′* = *total-degree R P*
using *assms total-degree-in-monomials-of*
by *blast*
then have *2*: *size m′* $\leq$ *n*
using *1*
by *blast*
have *3*: *n* $\leq$*size m′*
using *assms m′-def total-degree-ineq* **by** *auto*

show *?thesis* **using** *2 3*
using *dual-order.antisym m'-def* **by** *blast*
qed
end

2.3.2 Degree in One Variable

definition *degree-in-var* ::
(*'a*, *'b*) *ring-scheme* ⇒ (*'a*, *'c*) *mvar-poly* ⇒ *'c* ⇒ *nat* **where**
degree-in-var R P i = (*if* (*monomials-of R P* = {}) *then 0 else Max* ((λ*m*. *count m i*) ‘ (*monomials-of R P*)))

context *ring*
begin

lemma *degree-in-var-ineq*:
assumes *m* ∈ *monomials-of R P*
assumes *finite* (*monomials-of R P*)
shows *degree-in-var R P i* ≥ *count m i*
unfolding *degree-in-var-def* **using** *assms*
by *force*

lemma *degree-in-var-in-monomials-of*:
assumes *monomials-of R P* ≠ {}
assumes *finite* (*monomials-of R P*)
obtains *m* **where** *m* ∈ *monomials-of R P* ∧ *count m i* = *degree-in-var R P i*
using *assms Max-in*[*of* ((λ*m*. *count m i*) ‘ (*monomials-of R P*))] **unfolding** *degree-in-var-def*
by (*metis* (*no-types*, *lifting*) *empty-is-image finite-imageI image-iff*)

lemma *degree-in-varI*:
assumes *finite* (*monomials-of R P*)
assumes ∃ *m*. *m* ∈ *monomials-of R P* ∧ *count m i* = *n*
assumes ⋀*c*. *c* ∈ *monomials-of R P* ⟹ *count c i* ≤ *n*
shows *n* = *degree-in-var R P i*
proof−
obtain *l* **where** *l-def*: *l* ∈ *monomials-of R P* ∧ *count l i* = *degree-in-var R P i*
by (*metis assms*(*1*) *assms*(*2*) *degree-in-var-in-monomials-of equals0D*)
have *degree-in-var R P i* ≤ *n*
using *assms*(*3*) *l-def*
by *force*
then show *?thesis*
using *assms*(*1*) *assms*(*2*) *dual-order.antisym degree-in-var-ineq*
by *fastforce*
qed

Total degree bounds degree in a single variable:

lemma *total-degree-degree-in-var*:
assumes *finite* (*monomials-of R P*)

shows *total-degree R P* ≥ *degree-in-var R P i*
proof(*cases* (*monomials-of R P*) = {})
case *True*
then show *?thesis*
unfolding *total-degree-def degree-in-var-def*
by *simp*
next
case *False*
then obtain *m1* **where** *m1-def*: *m1* ∈ *monomials-of R P* ∧ *count m1 i* = *degree-in-var R P i*
by (*meson assms degree-in-var-in-monomials-of*)
have *size m1* ≥*count m1 i*
by (*simp add*: *multi-set-size-count*)
then show *?thesis*
by (*simp add*: *assms local.ring-axioms m1-def order.trans ring.total-degree-ineq*)
qed
end

The set of monomials of maximal degree in variable i:

definition *max-degree-monoms-in-var* ::
('*a*, '*b*) *ring-scheme* ⇒ ('*a*, '*c*) *mvar-poly* ⇒ '*c* ⇒ ('*c monomial*) *set* **where**
max-degree-monoms-in-var R P i = {*m*. *m* ∈ *monomials-of R P* ∧ *count m i* = *degree-in-var R P i*}

context *ring*
begin

lemma *max-degree-monoms-in-var-memI*:
assumes *m* ∈ *monomials-of R P*
assumes *count m i* = *degree-in-var R P i*
shows *m* ∈ *max-degree-monoms-in-var R P i*
using *assms* **unfolding** *max-degree-monoms-in-var-def*
by *blast*

lemma *max-degree-monoms-in-var-memE*:
assumes *m* ∈ *max-degree-monoms-in-var R P i*
shows *m* ∈ *monomials-of R P*
count m i = *degree-in-var R P i*
using *assms* **unfolding** *max-degree-monoms-in-var-def*
apply *blast*
using *assms* **unfolding** *max-degree-monoms-in-var-def*
by *blast*
end

The set of monomials of P of fixed degree in variable i:

definition *fixed-degree-in-var* ::
('*a*, '*b*) *ring-scheme* ⇒ ('*a*, '*c*) *mvar-poly* ⇒ '*c* ⇒ *nat* ⇒ ('*c monomial*) *set* **where**
fixed-degree-in-var R P i n = {*m*. *m* ∈ *monomials-of R P* ∧ *count m i* = *n*}

context *ring*
begin

lemma *fixed-degree-in-var-subset*:
fixed-degree-in-var R P i n ⊆ *monomials-of R P*
unfolding *fixed-degree-in-var-def*
by *blast*

lemma *fixed-degree-in-var-max-degree-monoms-in-var*:
max-degree-monoms-in-var R P i = *fixed-degree-in-var R P i* (*degree-in-var R P i*)
unfolding *max-degree-monoms-in-var-def fixed-degree-in-var-def*
by *auto*

lemma *fixed-degree-in-varI*:
assumes *m* ∈ *monomials-of R P*
assumes *count m i* = *n*
shows *m* ∈ *fixed-degree-in-var R P i n*
unfolding *fixed-degree-in-var-def*
using *assms*
by *blast*

lemma *fixed-degree-in-varE*:
assumes *m* ∈ *fixed-degree-in-var R P i n*
shows *m* ∈ *monomials-of R P*
count m i = *n*
apply (*meson assms fixed-degree-in-var-subset in-mono*)
using *assms fixed-degree-in-var-def* **by** *force*

definition *restrict-to-var-deg* ::
('*a*,'*c*) *mvar-poly* ⇒ '*c* ⇒ *nat* ⇒ '*c monomial* ⇒ '*a* **where**
restrict-to-var-deg P i n = *restrict-poly-to-monom-set R P* (*fixed-degree-in-var R P i n*)

lemma *restrict-to-var-deg-var-deg*:
assumes *finite* (*monomials-of R P*)
assumes *Q* = *restrict-to-var-deg P i n*
assumes *monomials-of R Q* ≠ {}
shows *n* = *degree-in-var R Q i*
apply(*rule degree-in-varI*)
apply (*metis assms*(*1*) *assms*(*2*) *fixed-degree-in-varE*(*1*) *monomials-ofE restrict-poly-to-monom-set-def*
restrict-to-var-deg-def rev-finite-subset subsetI)
apply (*metis* (*full-types*) *assms*(*2*) *assms*(*3*) *equals0I fixed-degree-in-varE*(*2*)
monomials-ofE restrict-poly-to-monom-set-def restrict-to-var-deg-def)
by (*metis assms*(*2*) *eq-iff fixed-degree-in-varE*(*2*) *monomials-ofE restrict-poly-to-monom-set-def restrict-to-var-deg-def*)

lemma *index-free-degree-in-var*[*simp*]:
assumes *index-free P i*

shows *degree-in-var R P i = 0*
proof(*cases monomials-of R P = {}*)
case *True*
then show *?thesis*
using *assms*
unfolding *degree-in-var-def*
by *simp*
next
case *False*
then have *0*: *degree-in-var R P i = Max ((λm. count m i) ‘ (monomials-of R P))*
unfolding *degree-in-var-def*
by *simp*
have((*λm. count m i*) ‘ (*monomials-of R P*)) = {*0*}
proof
show (*λm. count m i*) ‘ *monomials-of R P* ⊆ {*0*}
using *False assms monomials-of-index-free*[*of P i*]
by *auto*
show {*0*} ⊆ (*λm. count m i*) ‘ *monomials-of R P*
using *False* ‹(*λm. count m i*) ‘ *monomials-of R P* ⊆ {*0*}›
by *auto*
qed
then show *?thesis* **using** *0*
by *simp*
qed

lemma *degree-in-var-index-free*:
assumes *degree-in-var R P i = 0*
assumes *finite* (*monomials-of R P*)
shows *index-free P i*
apply(*rule index-freeI*)
by (*metis assms*(*1*) *assms*(*2*) *degree-in-var-ineq le-zero-eq*)

end

2.4 Constructing the Multiplication Operation on the Ring of Indexed Polynomials

2.4.1 The Set of Factors of a Fixed Monomial

The following function maps a monomial to the set of monomials which divide it:

definition *mset-factors* :: *'c monomial* ⇒ (*'c monomial*) *set* **where**
mset-factors m = {*n*. *n* ⊆# *m*}

context *ring*
begin

lemma *monom-divides-factors*:

$n \in (mset\text{-}factors\ m) \longleftrightarrow n \subseteq\# m$
unfolding *mset-factors-def* **by** *auto*

lemma *mset-factors-mono*:
assumes $n \subseteq\# m$
shows *mset-factors* $n \subseteq$ *mset-factors* m
unfolding *mset-factors-def*
by (*simp add*: *Collect-mono-iff assms subset-mset.order-trans*)

lemma *mset-factors-size-bound*:
assumes $n \in$ *mset-factors* m
shows *size* $n \leq$ *size* m
using *assms*
unfolding *mset-factors-def*
by (*simp add*: *size-mset-mono*)

lemma *sets-to-inds-finite*:
assumes *finite* I
shows *finite* $S \Longrightarrow$ *finite* $(Pi_E\ S\ (\lambda\text{-}.\ I))$
using *assms*
by (*simp add*: *finite-PiE*)

end

2.4.2 Finiteness of the Factor Set of a Monomial

This section shows that any monomial m has only finitely many factors. This is done by mapping the set of factors injectively into a finite extensional function set. Explicitly, a monomial is just mapped to its count function, restricted to the set of indices where the count is nonzero.

definition *mset-factors-to-fun* ::
$('a,'b)$ *ring-scheme* $\Rightarrow$ $'c$ *monomial* $\Rightarrow$ $'c$ *monomial* $\Rightarrow$ $('c \Rightarrow nat)$ **where**
mset-factors-to-fun R m n = (*if* $(n \in$ *mset-factors* $m)$ *then*
(*restrict* (*count n*) (*set-mset m*)) *else undefined*)
context *ring*
begin

lemma *mset-factors-to-fun-prop*:
assumes *size* $m = n$
shows *mset-factors-to-fun R m* $\in$ (*mset-factors m*) $\rightarrow$ (Pi_E (*set-mset m*) (λ-. $\{0..\ n\}$))
proof
fix x
assume A: $x \in$ (*mset-factors* m)
show *mset-factors-to-fun R m x* $\in$ (*set-mset m*) $\rightarrow_E \{0..n\}$
proof
show $\bigwedge xa.\ xa \in\# m \Longrightarrow$ *mset-factors-to-fun R m x xa* $\in \{0..n\}$
proof−

```
      fix y
      assume Ay: y ∈# m
      show P0: mset-factors-to-fun R m x y ∈ {0..n}
      proof−
        have mset-factors-to-fun R m x = restrict (count x) (set-mset m)
          using A  unfolding mset-factors-to-fun-def
          by simp
        then have mset-factors-to-fun R m x y  = count x y
          using Ay
          by simp
        then show ?thesis
         using A INTEG.R.mset-factors-size-bound assms count-size by fastforce
      qed
    qed
   fix z
   assume Ay: z ∉#m
   show mset-factors-to-fun R m x z = undefined
    using A Ay unfolding mset-factors-to-fun-def
    by simp
  qed
qed

lemma mset-factors-to-fun-inj:
  shows inj-on (mset-factors-to-fun R m) (mset-factors m)
proof
  fix x y
  assume A: x ∈ mset-factors m y ∈ mset-factors m
  show mset-factors-to-fun R m x = mset-factors-to-fun R m y ⟹ x = y
  proof−
    assume A0:  mset-factors-to-fun R m x = mset-factors-to-fun R m y
    show  x = y
    proof−
      have ⋀i. count x i = count y i
      proof− fix i
        show count x i = count y i
        proof(cases i ∈# m)
          case True
          then show ?thesis using A0 A unfolding mset-factors-to-fun-def
            by (metis restrict-def)
        next
          case False
          then show ?thesis
            using A0 A unfolding mset-factors-to-fun-def
            by (metis monom-divides-factors count-inI mset-subset-eqD)
        qed
      qed
      then show ?thesis
        using multiset-eqI by blast
    qed
```

qed
qed

lemma *finite-target*:
finite (*Pi*$_E$ (*set-mset m*) (λ-. {*0*..(*n*::*nat*)}))
proof−
have *0*: *finite* (*set-mset m*)
by *simp*
have *1*: *finite* ({*0*..*n*})
by *simp*
then show *?thesis* **using** *0*
by (*simp add*: *finite-PiE*)
qed

A multiset has only finitely many factors:

lemma *mset-factors-finite*[*simp*]:
finite (*mset-factors m*)
proof−
have *0*: *inj-on* (*mset-factors-to-fun R m*) (*mset-factors m*)
by (*simp add*: *mset-factors-to-fun-inj*)
have *1*: (*mset-factors-to-fun R m*) ∈ (*mset-factors m*) → (*Pi*$_E$ (*set-mset m*) (λ-. {*0* .. (*size m*)}))
by (*metis mset-factors-to-fun-prop*)
have *2*: *finite* (*Pi*$_E$ (*set-mset m*) (λ-. {*0* .. (*size m*)}))
by (*simp add*: *finite-target*)
have *3*: ((*mset-factors-to-fun R m*) ‘ (*mset-factors m*)) ⊆ (*Pi*$_E$ (*set-mset m*) (λ-. {*0* .. (*size m*)}))
using *1 2*
by *blast*
then have *finite* ((*mset-factors-to-fun R m*) ‘ (*mset-factors m*))
using *2 finite-subset* **by** *auto*
then show *?thesis* **using** *0 1 2*
finite-imageD[*of mset-factors-to-fun R m mset-factors m*]
by *blast*
qed

end

2.4.3 Definition of Indexed Polynomial Multiplication.

context *ring*
begin

Monomial division:

lemma *monom-divide*:
assumes *n* ∈ *mset-factors m*
shows (*THE k*. *n* + *k* = *m*) = *m* − *n*
apply(*rule the-equality*)
using *assms* **unfolding** *mset-factors-def*

apply *simp*
using *assms* **unfolding** *mset-factors-def*
by *auto*

A monomial is a factor of itself:

lemma *m-factor*[*simp*]:
$m \in$ *mset-factors m*
using *local.ring-axioms ring.monom-divides-factors* **by** *blast*

end

The definition of polynomial multiplication:

definition *P-ring-mult* :: ('*a*, '*b*) *ring-scheme* ⇒ ('*a*,'*c*) *mvar-poly* ⇒ ('*a*,'*c*) *mvar-poly* ⇒ '*c monomial* ⇒ '*a*
where
P-ring-mult R P Q m = (*finsum R* (λ*x*. (*P x*) $\otimes_R$ (*Q* (*m* − *x*))) (*mset-factors m*))

abbreviation(**in** *ring*) *P-ring-mult-in-ring* (**infixl** ‹$\bigotimes_p$› *65*)**where**
P-ring-mult-in-ring ≡ *P-ring-mult R*

2.4.4 Distributivity Laws for Polynomial Multiplication

context *ring*
begin

lemma *P-ring-rdistr*:
assumes *carrier-coeff a*
assumes *carrier-coeff b*
assumes *carrier-coeff c*
shows $a \bigotimes_p (b \bigoplus c) = (a \bigotimes_p b) \bigoplus (a \bigotimes_p c)$
proof
fix *m*
show $(a \bigotimes_p (b \bigoplus c))\ m = (a \bigotimes_p b \bigoplus (a \bigotimes_p c))\ m$
proof−
have *RHS*: $(a \bigotimes_p b \bigoplus (a \bigotimes_p c))\ m =$
$(\bigoplus x \in$ *mset-factors m*. *a x* ⊗ *b* (*m* − *x*)) ⊕ ($\bigoplus x \in$ *mset-factors m*. *a x* ⊗ *c* (*m* − *x*))
unfolding *indexed-padd-def P-ring-mult-def* **by** *auto*
have *LHS*: $(a \bigotimes_p (b \bigoplus c))\ m =$
$(\bigoplus x \in$ *mset-factors m*. *a x* ⊗ *b* (*m* − *x*) ⊕ *a x* ⊗ *c* (*m* − *x*))
unfolding *indexed-padd-def P-ring-mult-def*
by (*meson assms*(*1*) *assms*(*2*) *assms*(*3*) *local.ring-axioms r-distr ring.carrier-coeff-def*)
have *RHS'*: $(a \bigotimes_p b \bigoplus (a \bigotimes_p c))\ m =$
$(\bigoplus x \in$ *mset-factors m*. *a x* ⊗ *b* (*m* − *x*) ⊕ *a x* ⊗ *c* (*m* − *x*))
proof−
have *0*: (λ*x*. *a x* ⊗ *b* (*m* − *x*)) ∈ *mset-factors m* → *carrier R*
proof
fix *x* **assume** *x* ∈ *mset-factors m*
then show *a x* ⊗ *b* (*m* − *x*) ∈ *carrier R*

```
      using assms carrier-coeffE
      by blast
    qed
    have 1: (λx. a x ⊗ c (m − x)) ∈ mset-factors m → carrier R
    proof
      fix x assume x ∈ mset-factors m
      then show a x ⊗ c (m − x) ∈ carrier R
        using assms carrier-coeffE
        by blast
    qed
    then show ?thesis
      using 0 1 RHS assms finsum-addf[of λx. a x ⊗ b (m − x) mset-factors m
                                        λx. a x ⊗ c (m − x)]
      by metis
  qed
  show ?thesis using LHS RHS' by auto
 qed
qed

lemma P-ring-ldistr:
 assumes carrier-coeff a
 assumes carrier-coeff b
 assumes carrier-coeff c
 shows  (b ⊕ c) ⊗p a = (b ⊗p a)⊕ (c ⊗p a)
proof
 fix m
 show ((b ⊕ c) ⊗p a) m = ((b ⊗p a)⊕ (c ⊗p a)) m
 proof−
  have RHS: ((b ⊗p a)⊕ (c ⊗p a)) m =
            (⊕x∈mset-factors m. b x ⊗ a (m − x)) ⊕ (⊕x∈mset-factors m. c
x ⊗ a (m − x))
    unfolding indexed-padd-def P-ring-mult-def by auto
  have LHS: ((b ⊕ c) ⊗p a) m=
            ( ⊕x∈mset-factors m. b x ⊗ a (m − x) ⊕ c x ⊗ a (m − x))
    unfolding indexed-padd-def P-ring-mult-def
  by (meson assms(1) assms(2) assms(3) l-distr local.ring-axioms ring.carrier-coeff-def)
  have RHS': ((b ⊗p a)⊕ (c ⊗p a)) m =
            (⊕x∈mset-factors m. b x ⊗ a (m − x) ⊕ c x ⊗ a (m − x))
  proof−
    have 0: (λx. b x ⊗ a (m − x)) ∈ mset-factors m → carrier R
    proof
      fix x assume x ∈ mset-factors m
      then show b x ⊗ a (m − x) ∈ carrier R
        using assms carrier-coeffE
        by blast
    qed
    have 1: (λx. c x ⊗ a (m − x)) ∈ mset-factors m → carrier R
    proof
      fix x assume x ∈ mset-factors m
```

then show $c\ x \otimes a\ (m - x) \in$ *carrier* R
using *assms carrier-coeffE*
by *blast*
qed
then show *?thesis*
using *0 1 RHS assms finsum-addf*[*of* $\lambda x.\ b\ x \otimes a\ (m - x)$ *mset-factors* m $\lambda x.\ c\ x \otimes a\ (m - x)$]
by *metis*
qed
show *?thesis* **using** *LHS RHS'* **by** *auto*
qed
qed
end

2.4.5 Multiplication Commutes with `indexed_pmult`

context *ring*
begin

This lemma shows how we can write the factors of a monomial m times a variable i in terms of the factors of m:

lemma *mset-factors-add-mset*:
mset-factors (*add-mset i m*) = *mset-factors m* ∪ *add-mset i* ‘ (*mset-factors m*)
proof
show *mset-factors* (*add-mset i m*) ⊆ *mset-factors m* ∪ *add-mset i* ‘ *mset-factors m*
proof fix *x*
assume *A*: *x* ∈ *mset-factors* (*add-mset i m*)
show *x* ∈ *mset-factors m* ∪ *add-mset i* ‘ *mset-factors m*
proof(*cases i* ∈# *x*)
case *True*
then have *x* − {#*i*#} ⊆# *m*
using *A INTEG.R.monom-divides-factors subset-eq-diff-conv* **by** *force*
then show *?thesis*
by (*metis INTEG.R.monom-divides-factors True UnI2 add-mset-remove-trivial image-iff multi-member-split*)
next
case *False*
have *0*: *x* ⊆# *add-mset i m*
using *A INTEG.R.monom-divides-factors* **by** *blast*
hence *x* ⊆# *m*
using *mset-subset-eqI*[*of x m*] *0 False*
by (*metis count-add-mset count-greater-zero-iff count-inI less-Suc-eq-le subseteq-mset-def union-single-eq-member*)
thus *?thesis*
using *INTEG.R.monom-divides-factors* **by** *blast*
qed
qed
show *mset-factors m* ∪ *add-mset i* ‘ *mset-factors m* ⊆ *mset-factors* (*add-mset i*

m)
proof fix *x* **assume** *A*: *x* ∈ *mset-factors m* ∪ *add-mset i* ‘ *mset-factors m*
have *x* ⊆# *add-mset i m*
proof(*cases x* ∈ *mset-factors m*)
case *True*
then have *x* ⊆# *m*
by (*simp add*: *INTEG.R.monom-divides-factors*)
hence (⋀*a*. *count x a* ≤ *count* (*add-mset i m*) *a*)
using *mset-subset-eq-count*[*of x m*] *count-add-mset*[*of i m*]
nat-le-linear not-less-eq-eq **by** *fastforce*
thus *?thesis*
using *mset-subset-eqI* **by** *blast*
next
case *False*
then obtain *n* **where** *n-def*: *n* ∈ *mset-factors m* ∧ *x* = *add-mset i n*
using *A* **by** *blast*
then have *x* ⊆# *add-mset i m*
by (*simp add*: *INTEG.R.monom-divides-factors*)
then show *?thesis*
by *simp*
qed
thus *x* ∈ *mset-factors* (*add-mset i m*)
by (*simp add*: *INTEG.R.monom-divides-factors*)
qed
qed

end

2.4.6 Associativity of Polynomial Multiplication.

context *ring*
begin

lemma *finsum-eq*:
assumes *f* ∈ *S* → *carrier R*
assumes *g* ∈ *S* → *carrier R*
assumes (λ *x* ∈ *S*. *f x*) = (λ *x* ∈ *S*. *g x*)
shows *finsum R f S* = *finsum R g S*
by (*metis assms*(*1*) *assms*(*3*) *finsum-cong*′ *restrict-apply*′)

lemma *finsum-eq-induct*:
assumes *f* ∈ *S* → *carrier R*
assumes *g* ∈ *T* → *carrier R*
assumes *finite S*
assumes *finite T*
assumes *bij-betw h S T*
assumes ⋀*s*. *s* ∈ *S* ⟹ *f s* = *g* (*h s*)
shows *finite U* ⟹ *U* ⊆ *S* ⟹ *finsum R f U* = *finsum R g* (*h* ‘ *U*)
apply(*induct rule*: *finite-induct*)

```
  apply (simp; fail)
proof−
  fix x
  fix F :: 'c set
  assume F-fin: finite F
  assume x-not-in: x ∉ F
  assume IH: (F ⊆ S ⟹ finsum R f F = finsum R g (h ‘ F))
  show insert x F ⊆ S ⟹ finsum R f (insert x F) = finsum R g (h ‘ insert x F)
  proof−
    assume A: insert x F ⊆ S
    then have F-sub: F ⊆S
      by simp
    have x-in: x ∈ S
      using A by blast
    have fx-in: f x ∈  carrier R
      using x-in assms(1)
      by auto
    have ghx-fx-eq:
      f x = g (h x)
      using x-in assms
      by blast
    have I: finsum R f F = finsum R g (h ‘ F)
      using F-sub IH by blast
    have f-fact: f ∈ F → carrier R
      using assms(1) F-sub
      by blast
    have finsum R f (insert x F) = (f x) ⊗add-monoid R (finsum R f F)
      using F-fin  x-not-in comm-monoid.finprod-insert[of (add-monoid R) F x f ]
      unfolding finsum-def
      using f-fact fx-in local.add.comm-monoid-axioms
      by auto
    have finsum R g (h ‘ insert x F) = finsum R g (insert (h x) (h ‘ F))
      by simp
    then have finsum R g (h ‘ insert x F) = (g (h x)) ⊗add-monoid R finsum R g
(h ‘ F)
    proof−
      have 0: finite (h ‘ F)
        by (simp add: F-fin)
      have 1: h x ∉ h ‘ F
      proof−
        have 10: bij-betw h F (h‘ F)
          using assms(5) F-sub bij-betw-subset
          by blast
        show ?thesis
        proof
          assume h x ∈ h ‘ F
          then  obtain s where s-def: s ∈ F ∧ h s = h x
            using 10
            by auto
```

```
        have s ∈ S
          using F-sub s-def by blast
        then have s = x
          using x-in assms(5)
            s-def
          unfolding bij-betw-def inj-on-def
          by blast
        then show False using x-not-in
          using s-def by blast
      qed
    qed
    have 2: g ∈ h ‘ F → carrier (add-monoid R)
    proof
      fix y
      assume Ay: y ∈ h ‘ F
      then obtain s where s-def: y = h s ∧ s ∈ F
        by blast
      then have s-S: s ∈ S
        using F-sub by blast
      have h ‘ F ⊆ T
        using F-sub assms(5)
        unfolding bij-betw-def
        by blast
      then have g y ∈ carrier R
        using Ay assms(2)
        by blast
      then show g y ∈ carrier (add-monoid R)
        by simp
    qed
    have g (h x) ∈ carrier (add-monoid R)
      using fx-in ghx-fx-eq
      by auto
    then show ?thesis
     using 0 1 2 comm-monoid.finprod-insert[of (add-monoid R) (h ‘ F)  h x g ]
      unfolding finsum-def
      by (simp add: local.add.comm-monoid-axioms)
  qed
  then have finsum R g (h ‘ insert x F) = f x  ⊗add-monoid R (finsum R f F)
    using I ghx-fx-eq by auto
  then show ?thesis
    by (simp add: ‹finsum R f (insert x F) = f x ⊗add-monoid R finsum R f F›)
 qed
qed

lemma finsum-bij-eq:
 assumes f ∈ S → carrier R
 assumes g ∈ T → carrier R
 assumes finite S
 assumes bij-betw h S T
```

assumes $\bigwedge s.\ s \in S \Longrightarrow f\ s = g\ (h\ s)$
shows *finsum* $R\ f\ S =$ *finsum* $R\ g\ T$
proof−
have *0*: *finite* T
using *assms bij-betw-finite*
by *blast*
have *1*: $(\bigwedge s.\ s \in S \Longrightarrow f\ s = g\ (h\ s))$
using *assms*
by *blast*
have $(\bigwedge s.\ s \in S \Longrightarrow f\ s = g\ (h\ s)) \Longrightarrow$ *finite* $S \Longrightarrow S \subseteq S \Longrightarrow$ *finsum* $R\ f\ S =$ *finsum* $R\ g\ (h\ `\ S)$
using *0 assms finsum-eq-induct*[*of f S g T h S*]
by *blast*
then have *finsum* $R\ f\ S =$ *finsum* $R\ g\ (h\ `\ S)$
using *assms*(*5*) *assms*(*3*)
by *blast*
then show *?thesis*
using *assms*(*5*) *assms*(*4*) *bij-betw-imp-surj-on*
by *blast*
qed

lemma *monom-comp*:
assumes $x \subseteq\# m$
assumes $y \subseteq\# m - x$
shows $x \subseteq\# m - y$
using *assms*
by (*metis add-diff-cancel-left' subset-eq-diff-conv*
subset-mset.le-diff-conv2 subset-mset.order-refl subset-mset.order-trans)

lemma *monom-comp'*:
assumes $x \subseteq\# m$
assumes $y = m - x$
shows $x = m - y$
using *assms*
by (*metis add-diff-cancel-right' subset-mset.add-diff-inverse*)

This lemma turns iterated sums into sums over a product set. The first lemma is only a technical phrasing of `double_finsum'` to facilitate an inductive proof, and likely can and should be simplified.

lemma *double-finsum-induct*:
assumes *finite* S
assumes $\bigwedge s.\ s \in S \Longrightarrow$ *finite* $(F\ s)$
assumes $P = (\lambda S.\ \{(s,\ t).\ s \in S \wedge\ \ t \in (F\ s)\})$
assumes $\bigwedge s\ y.\ s \in S \Longrightarrow y \in (F\ s) \Longrightarrow g\ s\ y \in$ *carrier* R
shows *finite* $T \Longrightarrow T \subseteq S \Longrightarrow$ *finsum* $R\ (\lambda s.$ *finsum* $R\ (g\ s)\ (F\ s))\ T =$
finsum $R\ (\lambda\ c.\ g\ (fst\ c)\ (snd\ c))\ (P\ \ T)$
apply(*induct rule*: *finite-induct*)
apply (*simp add*: *assms*(*3*); *fail*)
proof−

fix x
fix $T :: 'c\ set$
assume AT: $finite\ T$
assume $x\text{-}notin$: $x \notin T$
assume IH: $(T \subseteq S \Longrightarrow (\bigoplus s{\in}T.\ finsum\ R\ (g\ s)\ (F\ s)) = (\bigoplus c{\in}P\ T.\ g\ (fst\ c)\ (snd\ c)))$
assume A: $insert\ x\ T \subseteq S$
show $(\bigoplus s{\in}insert\ x\ T.\ finsum\ R\ (g\ s)\ (F\ s)) = (\bigoplus c{\in}P\ (insert\ x\ T).\ g\ (fst\ c)\ (snd\ c))$
proof−
have 0: $(\lambda s.\ finsum\ R\ (g\ s)\ (F\ s)) \in T \to carrier\ R$
proof
fix v
assume $A0$: $v \in T$
then have $A1$: $v \in S$
using A **by** $blast$
then show $finsum\ R\ (g\ v)\ (F\ v) \in carrier\ R$
proof−
have 00: $finite\ (F\ v)$
using $assms\ A$
using ‹$v \in T$› **by** $blast$
have $g\ v \in F\ v \to carrier\ R$
proof
fix l **assume** $l \in F\ v$
then show $g\ v\ l \in carrier\ R$
using $A1\ assms(4)[of\ v\ l]$
by $blast$
qed
then show $?thesis$
using $finsum\text{-}closed$ **by** $blast$
qed
qed
have 1: $finsum\ R\ (g\ x)\ (F\ x) \in carrier\ R$
proof−
have $x \in S$
using A **by** $blast$
then show $?thesis$ **using** $assms(4)\ finsum\text{-}closed[of\ (g\ x)\ F\ x]$
by $blast$
qed
have 2: $(\bigoplus s{\in}insert\ x\ T.\ finsum\ R\ (g\ s)\ (F\ s)) = finsum\ R\ (g\ x)\ (F\ x) \oplus (\bigoplus s{\in}\ T.\ finsum\ R\ (g\ s)\ (F\ s))$
using $0\ 1\ finsum\text{-}insert[of\ T\ x\ (\lambda s.\ finsum\ R\ (g\ s)\ (F\ s))]\ AT\ x\text{-}notin$
by $blast$
have 3: $P\ (insert\ x\ T) = P\ \{x\} \cup P\ T$
proof
show $P\ (insert\ x\ T) \subseteq P\ \{x\} \cup P\ T$
proof
fix y
assume $y \in P\ (insert\ x\ T)$

```
    then show y ∈ P {x} ∪ P T
      using assms
      by blast
  qed
  show P {x} ∪ P T ⊆ P (insert x T)
  proof
    fix y
    assume Ay: y ∈ P {x} ∪ P T
    show y  ∈ P (insert x T)
    proof(cases y ∈ P {x})
      case True
      then have y ∈ ({(s, t). s ∈ {x} ∧ t ∈ F s})
        using assms(3) by auto
      then have fst y = x ∧ snd y ∈ F x
        using Product-Type.Collect-case-prodD by auto
      then show ?thesis using assms(3)
        using fst-def by auto
    next
      case False
      then have y ∈ P T
        using Ay
        by blast
      then have y ∈ ({(s, t). s ∈ T ∧ t ∈ F s})
        using assms(3) by blast
      then obtain s t where st-def: y = (s, t) ∧s ∈ T ∧ t ∈ F s
        by blast
      then have y = (s, t) ∧s ∈ (insert x T) ∧ t ∈ F s
        by blast
      then show ?thesis using assms(3)
        by simp
    qed
  qed
qed
have 4: P {x} ∩ P T = {}
proof
  show P {x} ∩ P T ⊆ {}
  proof
    fix y
    assume B: y ∈ P {x} ∩ P T
    then have fst y = x
    proof−
      have y ∈ {(s, t). s ∈ {x} ∧ t ∈ F s}
        using assms(3) B by auto
      then obtain s t where y = (s, t) ∧ s ∈ {x} ∧ t ∈ F s
        by blast
      then show ?thesis
        by auto
    qed
    have fst y ∈ T
```

```
    proof−
      have y ∈ {(s, t). s ∈ T ∧ t ∈ F s}
        using assms(3) B by auto
      then obtain s t where y = (s, t) ∧ s ∈ T ∧ t ∈ F s
        by blast
      then show ?thesis
        by simp
    qed
    then have False
      using x-notin
      by (simp add: ‹fst y = x›)
    then show y ∈ {}
      by simp
  qed
  show {} ⊆ P {x} ∩ P T
    by simp
qed
have 5: (λc. g (fst c) (snd c)) ∈ P {x} → carrier R
proof
  fix y
  assume X0: y ∈ P {x}
  obtain s t where st-def: y = (s, t) ∧ (s, t) ∈ P {x}
    by (metis X0 old.prod.exhaust)
  then have st: s = x ∧ t ∈ F x
    using assms(3) by blast
  then have g (fst y) (snd y) = g x t ∧t ∈ F x
    by (simp add: st-def)
  then show g (fst y) (snd y) ∈ carrier R
    using assms(4)[of x t] A
    by simp
qed
have 6: (λc. g (fst c) (snd c)) ∈ P T → carrier R
proof
  fix y
  assume X0: y ∈ P T
  obtain s t where st-def: y = (s, t) ∧ (s, t) ∈ P T
    by (metis X0 old.prod.exhaust)
  then have st: s ∈ T ∧ t ∈ F s
    using assms(3) by blast
  then have g (fst y) (snd y) = g s t ∧t ∈ F s
    by (simp add: st-def)
  then show g (fst y) (snd y) ∈ carrier R
    using assms(4)[of s t] A st
    by auto
qed
have 07: ⋀x. x ∈ S ⟹ finite (P {x})
proof−
  fix x
  assume x ∈ S
```

have *bij-betw snd* $(P\ \{x\})\ (F\ x)$
unfolding *bij-betw-def*
proof
show *inj-on snd* $(P\ \{x\})$
proof
fix $a\ b$
assume *Aa*: $a \in P\ \{x\}$
assume *Ab*: $b \in P\ \{x\}$
have *0*: $fst\ a = x$
proof−
have $a \in \{(s,\ t).\ s \in \{x\} \wedge t \in F\ s\}$
using *Aa assms*(*3*)
by *blast*
then obtain $s\ t$ **where** *st-def*: $a = (s,\ t) \wedge s \in \{x\} \wedge t \in F\ s$
by *blast*
then show *?thesis*
by *auto*
qed
have *1*: $fst\ b = x$
proof−
have $b \in \{(s,\ t).\ s \in \{x\} \wedge t \in F\ s\}$
using *Ab assms*(*3*)
by *blast*
then obtain $s\ t$ **where** *st-def*: $b = (s,\ t) \wedge s \in \{x\} \wedge t \in F\ s$
by *blast*
then show *?thesis*
by *auto*
qed
show $snd\ a = snd\ b \Longrightarrow a = b$
using *0 1*
by (*simp add*: *prod-eqI*)
qed
show $snd\ `\ P\ \{x\} = F\ x$
proof
show $snd\ `\ P\ \{x\} \subseteq F\ x$
proof
fix y
assume *0*: $y \in snd\ `\ P\ \{x\}$
then obtain q **where** *q-def*: $q \in P\ \{x\} \wedge y = snd\ q$
by *blast*
then obtain $s\ t$ **where** *st*: $q = (s,\ t) \wedge s \in \{x\} \wedge t \in F\ s$
using *assms*(*3*) **by** *blast*
then have *1*: $s = x$
by *blast*
have *2*: $snd\ q = t$
by (*simp add*: *st*)
then show $y \in F\ x$
using *q-def st* **by** *blast*
qed

```
        show F x ⊆ snd ‘ P {x}
        proof
          fix y
          assume y ∈ F x
          then have C: (x, y) ∈ P {x}
            using assms(3)
            by simp
          then have y = snd (x, y)
            by auto
          then show y ∈ snd ‘ P {x}
            using C
            by blast
        qed
      qed
      then show finite (P {x})
        using assms(2)[of x] bij-betw-finite
        ‹x ∈ S›
        by blast
    qed
    have 7: finite (P {x})
      using 07 A
      by blast
    have 8: finite (P T)
    proof−
      have ⋀D. finite D ⟹ D ⊆ S ⟹ finite (P D)
      proof−
        fix D
        show finite D ⟹ D ⊆ S  ⟹ finite (P D)
          apply(induct rule: finite-induct)
        proof−
          have P {} = {} using assms(3)
            by blast
          then show finite (P {})
            by auto
          show ⋀x F. finite F ⟹ x ∉ F ⟹ (F ⊆ S ⟹ finite (P F)) ⟹ insert
x F ⊆ S ⟹ finite (P (insert x F))
          proof−
            fix x
            fix Q :: 'c set
            assume fin: finite Q
            assume notin: x ∉ Q
            assume fin-pf: (Q ⊆ S ⟹ finite (P Q))
            assume I: insert x Q ⊆ S
            show finite (P (insert x Q))
            proof−
              have x-in: x ∈ S
                using I by blast
              have 0: (P (insert x Q)) ⊆ (P Q) ∪ P {x}
```

```
          proof
            fix y
            assume y ∈ P (insert x Q)
            obtain s t where st: y = (s, t) ∧ s ∈ (insert x Q) ∧ t ∈ F s
              using assms(3) x-in ‹y ∈ P (insert x Q)›
              by blast
            show y ∈ (P Q) ∪ P {x}
            proof(cases s ∈ Q)
              case True
              then have y ∈ P Q
                using st assms(3)
                by simp
              then show ?thesis using st assms(3)
                by blast
            next
              case False
              then have s = x
                using st by blast
              then have  s ∈{x} ∧ t ∈ F s
                using st by blast
              then have y ∈ P {x}
                using st assms(3)
                by auto
              then show ?thesis by auto
            qed
          qed
          have 1: finite  (P Q)
            using I fin-pf by blast
          have finite (P {x})
            using 07 x-in by blast
          then show ?thesis using 0 1
            using finite-subset by auto
        qed
      qed
    qed
   qed
   then show ?thesis
     using A AT by blast
  qed
  have 9: (⨁ q∈P (insert x T). g (fst q) (snd q)) =
      (⨁ q∈P T. g (fst q) (snd q)) ⊕ (⨁ q∈P {x}. g (fst q) (snd q))
    using 8 7 6 5 4 3 finsum-Un-disjoint[of P {x} P T λq. g (fst q) (snd q)]
   by (simp add: 7 ‹[[finite (P {x}); finite (P T); P {x} ∩ P T = {}; (λp. g (fst
p) (snd p))
        ∈ P {x} → carrier R; (λp. g (fst p) (snd p)) ∈ P T → carrier R]] ⟹
        (⨁ p∈P {x} ∪ P T. g (fst p) (snd p)) = (⨁ p∈P {x}. g (fst p) (snd p))
⊕
         (⨁ p∈P T. g (fst p) (snd p))› add.m-comm)
  have 10: (⨁ p∈P (insert x T). g (fst p) (snd p))
```

```
    = (⨁ s∈T. finsum R (g s) (F s))⊕ (⨁ p∈P {x}. g (fst p) (snd p))
  using IH 9 A
  by auto
have 11: (⨁ p∈P {x}. g (fst p) (snd p)) = finsum R (g x) (F x)
proof−
  obtain h :: ('c × 'd) ⇒ 'd where h-def: h = snd
    by simp
  have 110: bij-betw h (P {x}) (F x)
    unfolding bij-betw-def
  proof
    show inj-on h (P {x})
      unfolding inj-on-def
    proof
      fix q
      assume Ap: q ∈  P {x}
      show  ∀ y∈P {x}. h q = h y ⟶ q = y
      proof
        fix y
        assume q-def: y ∈ P {x}
        then have C0: fst y = x
          using assms(3)
          by (simp add: case-prod-beta)
        have C1: fst q = x
          using assms(3) Ap
          by (simp add: case-prod-beta)
         show  h q = h y ⟶ q = y
          using C0 C1 h-def
          by (simp add: prod-eq-iff)
      qed
    qed
    show h ‘ P {x} = F x
    proof
      show h ‘ P {x} ⊆ F x
      proof
        fix y
        assume y ∈ h ‘ P {x}
        then obtain d where d-def: y = h d ∧ d ∈ P {x}
          by blast
        then obtain s t where st-def: d = (s, t)
          by (meson surj-pair)
        then have s = x ∧ t ∈ F x
          using assms(3) d-def
          by blast
        then  show y ∈ F x
          using assms(3)
          by (simp add: d-def h-def st-def)
      qed
      show F x ⊆ h ‘ P {x}
      proof
```

```
        fix y
        assume E0:  y ∈ F x
        have E1: y = h (x , y)
          by (simp add: h-def)
        have (x, y) ∈ P {x}
          using E0 assms(3) by blast
        then show y ∈ h ‘ P {x}
          using E1 assms(3) by blast
      qed
    qed
  qed
  have 111: g x ∈ F x → carrier R
  proof
    fix y
    assume y ∈ F x
    then show g x y ∈ carrier R
      using assms A
      by blast
  qed
  have 112: (⋀s. s ∈ P {x} ⟹ g (fst s) (snd s) = g x (h s))
  proof−
    fix s
    assume s ∈ P{x}
    then have s ∈ {(s, t). s = x ∧ t ∈ F s}
      using assms(3) by blast
    then obtain t where s = (x, t) ∧ t ∈ F x
      using assms(3)
      by blast
    then show g (fst s) (snd s) = g x (h s)
      by (simp add: h-def)
  qed



  show ?thesis using 5 7  110 111 112 finsum-bij-eq[of λp. g (fst p) (snd p) P
{x} g x F x h ]
    by auto
 qed
 have 12: (⨁ p∈P (insert x T). g (fst p) (snd p))
    =  (⨁ s∈T. finsum R (g s) (F s))⊕ finsum R (g x) (F x)
  using 10 11 by auto
 then show ?thesis using 2
  by (simp add: 2 0 1 add.m-comm)
qed
qed

lemma double-finsum:
  assumes finite S
  assumes ⋀s. s ∈ S ⟹ finite (F s)
```

assumes $P = \{(s,\ t).\ s \in S \wedge\ \ t \in (F\ s)\}$
assumes $\bigwedge s\ y.\ s \in S \Longrightarrow y \in (F\ s) \Longrightarrow g\ s\ y \in \mathit{carrier}\ R$
shows *finsum* R $(\lambda s.\ \mathit{finsum}\ R\ (g\ s)\ (F\ s))\ S =$
finsum R $(\lambda\ p.\ g\ (\mathit{fst}\ p)\ (\mathit{snd}\ p))\ P$
proof−
obtain P' **where** P'-*def*: $P' =\ \ (\lambda S.\ \{(s,\ t).\ s \in S \wedge\ \ t \in (F\ s)\})$
by *simp*
have *finsum* R $(\lambda s.\ \mathit{finsum}\ R\ (g\ s)\ (F\ s))\ S =$
finsum R $(\lambda\ p.\ g\ (\mathit{fst}\ p)\ (\mathit{snd}\ p))\ (P'\ S)$
using *double-finsum-induct*[*of* S F P' g S]
assms P'-*def*
by *blast*
then show *?thesis*
using P'-*def assms*
by *blast*
qed

end

The product index set for the double sums in the coefficients of the $((a \otimes_p b) \otimes_p c)$. It is the set of pairs (x, y) of monomials, such that x is a factor of monomial m, and y is a factor of monomial x.

definition *right-cuts* :: *'c monomial* ⇒ (*'c monomial* × *'c monomial*) *set* **where**
right-cuts $m = \{(x,\ y).\ x \subseteq\# m \wedge y \subseteq\# x\}$

context *ring*
begin

lemma *right-cuts-alt-def*:
right-cuts $m = \{(x,\ y).\ x \in \mathit{mset\text{-}factors}\ m \wedge y \in \mathit{mset\text{-}factors}\ x\}$
unfolding *mset-factors-def right-cuts-def*
by *simp*

lemma *right-cuts-finite*:
finite (*right-cuts* m)
proof−
have *finite* (*mset-factors* m × *mset-factors* m)
using *mset-factors-finite*
by *blast*
have *right-cuts* $m \subseteq$ (*mset-factors* m × *mset-factors* m)
proof
fix p
assume p-*def*: $p \in$ *right-cuts* m
obtain $x\ y$ **where** *xy*: $p = (x\ ,\ y) \wedge x \subseteq\# m \wedge y \subseteq\# x$
using p-*def* **unfolding** *right-cuts-def*
by *blast*
then have $x \in \mathit{mset\text{-}factors}\ m \wedge y \in \mathit{mset\text{-}factors}\ m$
using *monom-divides-factors*
by *auto*

then show $p \in (\mathit{mset\text{-}factors}\ m \times \mathit{mset\text{-}factors}\ m)$
by (*simp add*: *xy*)
qed
then show *?thesis*
using ‹*finite* (*mset-factors* $m \times$ *mset-factors* m)› *finite-subset*
by *blast*
qed

lemma *assoc-aux0*:
assumes *carrier-coeff a*
assumes *carrier-coeff b*
assumes *carrier-coeff c*
assumes $g = (\lambda x\ y.\ \ a\ x \otimes (b\ y \otimes c\ (m - x - y)))$
shows $\bigwedge s\ y.\ s \in \mathit{mset\text{-}factors}\ m \Longrightarrow y \in \mathit{mset\text{-}factors}\ (m - x) \Longrightarrow g\ s\ y \in \mathit{carrier}\ R$
using *assms carrier-coeffE* **by** *blast*

lemma *assoc-aux1*:
assumes *carrier-coeff a*
assumes *carrier-coeff b*
assumes *carrier-coeff c*
assumes $g = (\lambda x\ y.\ \ (a\ y\ \ \otimes b\ (x - y)) \otimes c\ (m - x))$
shows $\bigwedge s\ y.\ s \in \mathit{mset\text{-}factors}\ m \Longrightarrow y \in \mathit{mset\text{-}factors}\ x \Longrightarrow g\ s\ y \in \mathit{carrier}\ R$
using *assms carrier-coeffE* **by** *blast*
end

The product index set for the double sums in the coefficients of the $(a \otimes_p (b \otimes_p c))$. It is the set of pairs (x, y) such that x is a factor of m and y is a factor of m/x.

definition *left-cuts* :: *'c monomial* ⇒ (*'c monomial* × *'c monomial*) *set* **where**
left-cuts $m = \{(x,\ y).\ x \subseteq\# m \wedge y \subseteq\# (m - x)\}$

context *ring*
begin

lemma *left-cuts-alt-def*:
left-cuts $m = \{(x,\ y).\ x \in \mathit{mset\text{-}factors}\ m \wedge y \in \mathit{mset\text{-}factors}\ (m - x)\}$
unfolding *mset-factors-def left-cuts-def*
by *simp*

This lemma witnesses the bijection between left and right cuts for the proof of associativity:

lemma *left-right-cuts-bij*:
bij-betw $(\lambda\ (x,y).\ (x + y,\ x))$ (*left-cuts* m) (*right-cuts* m)
unfolding *bij-betw-def right-cuts-def left-cuts-def*
proof
show *inj-on* $(\lambda(x,\ y).\ (x + y,\ x))\ \{(x,\ y).\ x \subseteq\# m \wedge y \subseteq\# m - x\}$
unfolding *inj-on-def*
by *auto*

show $(\lambda(x,\ y).\ (x + y,\ x))$ ‘ $\{(x,\ y).\ x \subseteq\# m \wedge y \subseteq\# m - x\} = \{(x,\ y).\ x \subseteq\# m \wedge y \subseteq\# x\}$
proof
show $(\lambda(x,\ y).\ (x + y,\ x))$ ‘ $\{(x,\ y).\ x \subseteq\# m \wedge y \subseteq\# m - x\} \subseteq \{(x,\ y).\ x \subseteq\# m \wedge y \subseteq\# x\}$
proof
fix p
assume $p \in (\lambda(x,\ y).\ (x + y,\ x))$ ‘ $\{(x,\ y).\ x \subseteq\# m \wedge y \subseteq\# m - x\}$
then obtain $a\ b$ **where** *ab*: $a \subseteq\# m \wedge b \subseteq\# m - a \wedge p = (\lambda(x,\ y).\ (x + y,\ x))\ (a,\ b)$
by *blast*
then have $p = (a + b,\ a)$
by *simp*
then show $p \in \{(x,\ y).\ x \subseteq\# m \wedge y \subseteq\# x\}$
using *ab*
by (*metis* (*no-types*, *lifting*) *case-prodI mem-Collect-eq mset-subset-eq-add-left subset-mset.le-diff-conv2 union-commute*)
qed
show $\{(x,\ y).\ x \subseteq\# m \wedge y \subseteq\# x\} \subseteq (\lambda(x,\ y).\ (x + y,\ x))$ ‘ $\{(x,\ y).\ x \subseteq\# m \wedge y \subseteq\# m - x\}$
proof
fix p
assume *p-def*: $p \in \{(x,\ y).\ x \subseteq\# m \wedge y \subseteq\# x\}$
then obtain $a\ b$ **where** *ab*: $p = (a,\ b) \wedge a \subseteq\# m \wedge b \subseteq\# a$
by *blast*
then have $p = (\lambda(x,\ y).\ (x + y,\ x))\ (b,\ a - b)$
using *ab*
by *simp*
then show $p \in (\lambda(x,\ y).\ (x + y,\ x))$ ‘ $\{(x,\ y).\ x \subseteq\# m \wedge y \subseteq\# m - x\}$
by (*metis* (*mono-tags*, *lifting*) *ab case-prodI image-eqI mem-Collect-eq subset-mset.diff-add subset-mset.dual-order.trans subset-mset.le-diff-conv2*)
qed
qed
qed

lemma *left-cuts-sum*:
assumes *carrier-coeff a*
assumes *carrier-coeff b*
assumes *carrier-coeff c*
shows $(a \bigotimes_p (b \bigotimes_p c))\ m = (\bigoplus p \in \textit{left-cuts}\ m.\ a\ (\textit{fst}\ p) \otimes (b\ (\textit{snd}\ p) \otimes c\ (m - (\textit{fst}\ p) - (\textit{snd}\ p))))$
proof−
have *U*: $(a \bigotimes_p (b \bigotimes_p c))\ m = (\bigoplus x \in \textit{mset-factors}\ m.\ (\bigoplus xa \in \textit{mset-factors}\ (m - x).\ a\ x \otimes (b\ xa \otimes c\ (m - x - xa))))$
unfolding *P-ring-mult-def*
proof−
obtain f **where** *f-def*:
$f = (\lambda x\ .\ a\ x \otimes (\bigoplus xa \in \textit{mset-factors}\ (m - x).\ b\ xa \otimes c\ (m - x - xa)))$
by *simp*

obtain *g* **where** *g-def*:
g = (λ*x* . ⨁ *xa*∈*mset-factors* (*m* − *x*). *a x* ⊗ (*b xa* ⊗ *c* (*m* − *x* − *xa*)))
by *simp*
have *0*: *restrict f* (*mset-factors m*) = *restrict g* (*mset-factors m*)
proof
fix *x*
show *restrict f* (*mset-factors m*) *x* = *restrict g* (*mset-factors m*) *x*
proof(*cases x* ∈ *mset-factors m*)
case *True*
have *T0*: *restrict f* (*mset-factors m*) *x* = *a x* ⊗ (⨁ *xa*∈*mset-factors* (*m* − *x*). *b xa* ⊗ *c* (*m* − *x* − *xa*))
using *f-def True* **by** *auto*
have *T1*: *restrict g* (*mset-factors m*) *x* = (⨁ *xa*∈*mset-factors* (*m* − *x*). *a x* ⊗ (*b xa* ⊗ *c* (*m* − *x* − *xa*)))
using *True g-def* **by** *auto*
show *restrict f* (*mset-factors m*) *x* = *restrict g* (*mset-factors m*) *x*
using *assms finsum-rdistr*[*of mset-factors* (*m* − *x*) *a x* λ *xa*. *b xa* ⊗ *c* (*m* − *x* − *xa*)]
by (*metis* (*mono-tags*, *lifting*) *mset-factors-finite Pi-I T0 T1 carrier-coeffE m-closed*)
next
case *False*
then have *restrict f* (*mset-factors m*) *x* = *undefined* **using** *f-def*
by (*simp add*: *restrict-def*)
have *restrict g* (*mset-factors m*) *x* = *undefined* **using** *g-def False*
using *restrict-def* **by** *auto*
then show *?thesis* **using** *f-def g-def*
using ‹*restrict f* (*mset-factors m*) *x* = *undefined*›
by *auto*

qed
qed
have *1*: *f* ∈ *mset-factors m* → *carrier R*
using *f-def assms*
by (*simp add*: *carrier-coeffE*)
have *2*: *g* ∈ *mset-factors m* → *carrier R*
using *g-def assms*
by (*simp add*: *carrier-coeffE*)
show (⨁ *x*∈*mset-factors m*. *a x* ⊗ (⨁ *xa*∈*mset-factors* (*m* − *x*). *b xa* ⊗ *c* (*m* − *x* − *xa*))) =
(⨁ *x*∈*mset-factors m*. ⨁ *xa*∈*mset-factors* (*m* − *x*). *a x* ⊗ (*b xa* ⊗ *c* (*m* − *x* − *xa*)))
using *f-def g-def finsum-eq*[*of f*
mset-factors m g] *0 1 2*
by *blast*
qed
have *0*: (⋀*s*. *s* ∈ *mset-factors m* ⟹ *finite* (*mset-factors* (*m* − *s*)))
by *simp*
have *1*: *finite* (*mset-factors m*)

by *simp*
have *2*: $(\bigwedge s\ y.\ s \in \textit{mset-factors}\ m \Longrightarrow y \in \textit{mset-factors}\ (m - s) \Longrightarrow a\ s \otimes (b\ y \otimes c\ (m - s - y)) \in \textit{carrier}\ R)$
using *assms assoc-aux0*[*of a b c*]
by *blast*
have $(\bigoplus x{\in}\textit{mset-factors}\ m.\ (\bigoplus xa{\in}\textit{mset-factors}\ (m - x).\ a\ x \otimes (b\ xa \otimes c\ (m - x - xa)))) =$
$(\bigoplus p \in \textit{left-cuts}\ m.\ a\ (\textit{fst}\ p) \otimes (b\ (\textit{snd}\ p) \otimes c\ (m - (\textit{fst}\ p) - (\textit{snd}\ p))))$
using *assms left-cuts-alt-def*[*of m*] *0 1 2*
double-finsum[*of mset-factors m* $\lambda x.$ *mset-factors* $(m - x)$ *left-cuts m* $(\lambda x\ y.\ a\ x \otimes (b\ y \otimes c\ (m - x - y)))$]
by *blast*
then show *?thesis* **using** *U*
by *auto*
qed

lemma *right-cuts-sum*:
assumes *carrier-coeff a*
assumes *carrier-coeff b*
assumes *carrier-coeff c*
shows $(a \bigotimes_p b \bigotimes_p c)\ m = (\bigoplus p \in \textit{right-cuts}\ m.\ a\ (\textit{snd}\ \ p) \otimes (b\ ((\textit{fst}\ p) - (\textit{snd}\ p)) \otimes c\ (m - (\textit{fst}\ p))))$
proof−
have *0*: *finite* (*mset-factors m*)
by *simp*
have *1*: $(\bigwedge s.\ s \in \textit{mset-factors}\ m \Longrightarrow \textit{finite}\ (\textit{mset-factors}\ s))$
by *auto*
have *2*: $\textit{right-cuts}\ m = \{(s,\ t).\ s \in \textit{mset-factors}\ m \wedge t \in \textit{mset-factors}\ s\}$
unfolding *right-cuts-def*
by (*simp add*: *monom-divides-factors*)
have *3*: $(\bigwedge s\ y.\ s \in \textit{mset-factors}\ m \Longrightarrow y \in \textit{mset-factors}\ s \Longrightarrow a\ y \otimes b\ (s - y) \otimes c\ (m - s) \in \textit{carrier}\ R)$
using *assoc-aux1 assms*(*1*) *assms*(*2*) *assms*(*3*)
by *blast*
have *4*: $(\bigoplus s{\in}\textit{mset-factors}\ m.\ (\bigoplus y{\in}\textit{mset-factors}\ s.\ a\ y \otimes b\ (s - y) \otimes c\ (m - s))) =$
$(\bigoplus p{\in}\textit{right-cuts}\ m.\ a\ (\textit{snd}\ p) \otimes b\ (\textit{fst}\ p - \textit{snd}\ p) \otimes c\ (m - \textit{fst}\ p))$
using *0 1 2 3*
double-finsum[*of mset-factors m - right-cuts m*
$(\lambda x\ y.\ \ (a\ y\ \ \otimes b\ (x - y)) \otimes c\ (m - x))$]
by *auto*
have *5*: $(\bigoplus x{\in}\textit{mset-factors}\ m.\ (\bigoplus xa{\in}\textit{mset-factors}\ x.\ a\ xa \otimes b\ (x - xa)) \otimes c\ (m - x)) =$
$(\bigoplus x{\in}\textit{mset-factors}\ m.\ \bigoplus xa{\in}\textit{mset-factors}\ x.\ a\ xa \otimes b\ (x - xa) \otimes c\ (m - x))$
proof−
obtain *f* **where** *f-def*: $f = (\ \lambda x.\ (\bigoplus xa{\in}\textit{mset-factors}\ x.\ a\ xa \otimes b\ (x - xa)) \otimes c\ (m - x))$
by *simp*

obtain *g* **where** *g-def*: $g = (\lambda x.\ \bigoplus xa{\in}mset\text{-}factors\ x.\ a\ xa \otimes b\ (x - xa) \otimes c\ (m - x))$
by *simp*
have *50*: $\bigwedge s.\ s \in (mset\text{-}factors\ m) \Longrightarrow f\ s = g\ s$
proof−
fix *x*
assume *As*: $x \in mset\text{-}factors\ m$
show $f\ x = g\ x$
proof−
have *f-eq*: $f\ x = (\bigoplus xa{\in}mset\text{-}factors\ x.\ a\ xa \otimes b\ (x - xa)) \otimes c\ (m - x)$
using *f-def*
by *auto*
have *g-eq*: $g\ x = (\bigoplus xa{\in}mset\text{-}factors\ x.\ a\ xa \otimes b\ (x - xa) \otimes c\ (m - x))$
using *g-def*
by *auto*
have *f-eq'*: $f\ x = (\bigoplus xa{\in}mset\text{-}factors\ x.\ (a\ xa \otimes b\ (x - xa)) \otimes c\ (m - x))$
using *f-eq finsum-ldistr*[*of mset-factors x c* $(m - x)$ $\lambda xa.\ (a\ xa \otimes b\ (x - xa))$] *assms*
by (*simp add*: ‹⟦*finite* (*mset-factors x*); $c\ (m - x) \in carrier\ R$; $(\lambda xa.\ a\ xa \otimes b\ (x - xa)) \in mset\text{-}factors\ x \to carrier\ R$⟧ $\Longrightarrow (\bigoplus i{\in}mset\text{-}factors\ x.\ a\ i \otimes b\ (x - i)) \otimes c\ (m - x) = (\bigoplus i{\in}mset\text{-}factors\ x.\ a\ i \otimes b\ (x - i) \otimes c\ (m - x))$› *Pi-I carrier-coeffE*)
then show $f\ x = g\ x$
by (*simp add*: *g-eq*)
qed
qed
have *51*: $f \in mset\text{-}factors\ m \to carrier\ R$
proof
fix *x*
assume $x \in mset\text{-}factors\ m$
have $(\bigoplus xa{\in}mset\text{-}factors\ x.\ a\ xa \otimes b\ (x - xa)) \otimes c\ (m - x) \in carrier\ R$
using *assms carrier-coeffE finsum-closed*[*of* $\lambda xa.\ a\ xa \otimes b\ (x - xa)$ *mset-factors x*]
by *blast*
then show $f\ x \in carrier\ R$ **using** *assms f-def* **by** *auto*
qed
have *52*: $g \in mset\text{-}factors\ m \to carrier\ R$
proof
fix *x*
assume $x \in mset\text{-}factors\ m$
show $g\ x \in carrier\ R$
using *assms finsum-closed*[*of* $\lambda xa.\ a\ xa \otimes b\ (x - xa) \otimes c\ (m - x)$ *mset-factors x*] *g-def*
by (*metis* (*no-types*, *lifting*) *50 51 Pi-iff* ‹$x \in mset\text{-}factors\ m$›)
qed

show *?thesis*
using *50 51 52 finsum-eq*[*of f* (*mset-factors m*) *g*]
by (*metis* (*mono-tags*, *lifting*) *f-def finsum-cong'' g-def*)

qed
then have *6*: $(\bigoplus x{\in}$*mset-factors* $m.\ (\bigoplus xa{\in}$*mset-factors* $x.\ a\ xa \otimes b\ (x - xa)) \otimes c\ (m - x)) =$
$(\bigoplus p{\in}$*right-cuts* $m.\ a\ (snd\ p) \otimes b\ (fst\ p - snd\ p) \otimes c\ (m - fst\ p))$
by (*simp add*: *4*)
have *7*: $(\bigoplus p{\in}$*right-cuts* $m.\ a\ (snd\ p) \otimes b\ (fst\ p - snd\ p) \otimes c\ (m - fst\ p))$
$= (\bigoplus p{\in}$*right-cuts* $m.\ a\ (snd\ p) \otimes (b\ (fst\ p - snd\ p) \otimes c\ (m - fst\ p)))$
using *assms*
by (*meson local.ring-axioms m-assoc ring.carrier-coeff-def*)
then show *?thesis*
using *assms 5 6*
unfolding *P-ring-mult-def*
by *simp*
qed

The Associativity of Polynomial Multiplication:

lemma *P-ring-mult-assoc*:
assumes *carrier-coeff a*
assumes *carrier-coeff b*
assumes *carrier-coeff c*
shows $a \bigotimes_p (b \bigotimes_p c) = (a \bigotimes_p b) \bigotimes_p c$
proof
fix *m*
show $(a \bigotimes_p (b \bigotimes_p c))\ m = (a \bigotimes_p b \bigotimes_p c)\ m$
proof−
obtain *f* **where** *f-def*: $f = (\lambda p.\ \ a\ (snd\ \ p) \otimes (b\ ((fst\ p) -(snd\ p)) \otimes c\ (m - (fst\ p))))$
by *simp*
obtain *g* **where** *g-def*: $g = (\lambda p.\ a\ (fst\ p) \otimes (b\ (snd\ p) \otimes c\ (m - (fst\ p) - (snd\ p))))$
by *simp*
have *f-dom*: $f \in$ *right-cuts* $m \to$ *carrier* R
using *assms f-def* **unfolding** *right-cuts-def*
by (*simp add*: *carrier-coeffE*)
have *g-dom*: $g \in$ *left-cuts* $m \to$ *carrier* R
using *assms g-def* **unfolding** *left-cuts-def*
by (*simp add*: *carrier-coeffE*)
have *0*: *finite* (*right-cuts m*)
by (*simp add*: *right-cuts-finite*)
have *1*: *bij-betw* $(\lambda\ (x,y).\ (x + y,\ x))$ (*left-cuts m*) (*right-cuts m*)
by (*simp add*: *left-right-cuts-bij*)
have *2*: $(\bigwedge s.\ s \in$ *left-cuts* $m \Longrightarrow g\ s = f\ (case\ s\ of\ (x,\ y) \Rightarrow (x + y,\ x)))$
proof−
fix *s*
assume $s \in$ *left-cuts m*
then obtain *x y* **where** *xy*: $s = (x,\ y) \wedge x \subseteq\# m \wedge y \subseteq\# m - x$
using *left-cuts-def*
by *blast*
then have *g-eq*: $g\ s = a\ x \otimes (b\ y \otimes c\ (m - x - y))$

using *g-def fst-conv*
by *auto*
have *f-eq*: $f\ (case\ s\ of\ (x,\ y) \Rightarrow (x + y,\ x)) = f\ (x + y,\ x)$
by (*simp add*: *xy*)
then have *f-eq'*: $f\ (case\ s\ of\ (x,\ y) \Rightarrow (x + y,\ x)) = a\ x \otimes (b\ ((x + y) - x) \otimes c\ (m - (x + y)))$
using *f-def*
by *simp*
then show $g\ s = f\ (case\ s\ of\ (x,\ y) \Rightarrow (x + y,\ x))$
by (*simp add*: *g-eq*)
qed
have *3*: *finsum R g* (*left-cuts m*) = *finsum R f* (*right-cuts m*)
using *0 1 2 finsum-bij-eq*[*of g left-cuts m f right-cuts m* λ (*x*,*y*). $(x + y,\ x)$]
using *bij-betw-finite f-dom g-dom* **by** *blast*
then show *?thesis* **using** *assms right-cuts-sum left-cuts-sum*
by (*metis* (*mono-tags*, *lifting*) *f-def f-dom finsum-cong' g-def g-dom*)
qed
qed

end

2.4.7 Commutativity of Polynomial Multiplication

context *ring*
begin

lemma *mset-factors-bij*:
bij-betw ($\lambda x.\ m - x$) (*mset-factors m*) (*mset-factors m*)
apply(*rule bij-betwI'*)
apply (*metis monom-comp' monom-divides-factors*)
apply (*simp add*: *monom-divides-factors*)
by (*meson monom-comp' monom-divides-factors diff-subset-eq-self*)

lemma(**in** *cring*) *P-ring-mult-comm*:
assumes *carrier-coeff a*
assumes *carrier-coeff b*
shows $a \bigotimes_p b = b \bigotimes_p a$
proof
fix *m*
show $(a \bigotimes_p b)\ m = (b \bigotimes_p a)\ m$
unfolding *P-ring-mult-def*
apply (*rule finsum-bij-eq*[*of* $\lambda\ x.\ a\ x \otimes b\ (m - x)$ *mset-factors m* $\lambda x.\ b\ x \otimes a\ (m - x)$ *mset-factors m* $\lambda x.\ m - x$])
proof
show $\bigwedge x.\ x \in$ *mset-factors m* $\Longrightarrow a\ x \otimes b\ (m - x) \in$ *carrier R*
proof−
fix *x*
assume $x \in$ *mset-factors m*
show $a\ x \otimes b\ (m - x) \in$ *carrier R*

```
      using assms carrier-coeffE
      by blast
  qed
  show (λx. b x ⊗ a (m − x)) ∈ mset-factors m → carrier R
  proof
    fix x
    assume x ∈ mset-factors m
    show b x ⊗ a (m − x) ∈ carrier R
      using assms  carrier-coeffE
      by blast
  qed
  show finite (mset-factors m)
    by simp
  show bij-betw ((−) m) (mset-factors m) (mset-factors m)
    by (simp add: mset-factors-bij)
  show ⋀s. s ∈ mset-factors m ⟹ a s ⊗ b (m − s) = b (m − s) ⊗ a (m − (m
− s))
  proof−
    fix s
    assume s ∈ mset-factors m
    then show a s ⊗ b (m − s) = b (m − s) ⊗ a (m − (m − s))
      using assms carrier-coeffE
    by (metis local.ring-axioms m-comm ring.monom-comp' ring.monom-divides-factors)
  qed
 qed
qed
```

2.4.8 Closure properties for multiplication

Building monomials from polynomials:

```
lemma indexed-const-P-mult-eq:
 assumes a ∈ carrier R
 assumes b ∈ carrier R
 shows (indexed-const a) ⨂p (indexed-const b) = indexed-const (a ⊗ b)
proof−
 have 0: monomials-of R (indexed-const a) = (if (a = 0) then {} else {{#}})
   unfolding monomials-of-def indexed-const-def
   by auto
 have 1: monomials-of R (indexed-const b) = (if (b = 0) then {} else {{#}})
   unfolding monomials-of-def indexed-const-def
   by auto
 show ?thesis
   unfolding P-ring-mult-def
 proof
   fix m:: 'c monomial
   show (⨁x∈mset-factors m. indexed-const a x ⊗ indexed-const b (m − x)) =
indexed-const (a ⊗ b) m
   proof(cases m = {#})
     case True
```

then have *T0*: *mset-factors m* = {{#}}
unfolding *mset-factors-def*
by *simp*
have (⨁ *x*∈*mset-factors m*. *indexed-const a x* ⊗ *indexed-const b* (*m* − *x*)) =
indexed-const a {#} ⊗ *indexed-const b* ({#} − {#})
proof−
have *0*: (λ*x*. *indexed-const a x* ⊗ *indexed-const b* (*m* − *x*)) ∈ {} → *carrier R*
by *blast*
have *1*: *indexed-const a* {#} ⊗ *indexed-const b* (*m* − {#}) ∈ *carrier R*
by (*simp add*: *assms*(*1*) *assms*(*2*) *indexed-const-def*)
have *2*: (⨁ *x*∈{{#}}. *indexed-const a x* ⊗ *indexed-const b* (*m* − *x*)) =
indexed-const a {#} ⊗ *indexed-const b* (*m* − {#}) ⊕ (⨁ *x*∈{}. *indexed-const a x* ⊗ *indexed-const b* (*m* − *x*))
using *True T0 0 1 finsum-insert*[*of* {} {#} λ*x*. *indexed-const a x* ⊗ *indexed-const b* (*m* − *x*)]
by (*simp add*: *indexed-const-def*)
then show *?thesis*
using *True T0 0 1 finsum-insert*[*of* {} {#} λ*x*. *indexed-const a x* ⊗ *indexed-const b* (*m* − *x*)]
finsum-empty[*of* λ*x*. *indexed-const a x* ⊗ *indexed-const b* (*m* − *x*)]
by (*simp add*: ‹⟦*finite* {}; {#} ∉ {}; (λ*x*. *indexed-const a x* ⊗ *indexed-const b* (*m* − *x*)) ∈ {} → *carrier R*; *indexed-const a* {#} ⊗ *indexed-const b* (*m* − {#}) ∈ *carrier R*⟧ ⟹ (⨁ *x*∈{{#}}. *indexed-const a x* ⊗ *indexed-const b* (*m* − *x*)) = *indexed-const a* {#} ⊗ *indexed-const b* (*m* − {#}) ⊕ (⨁ *x*∈{}. *indexed-const a x* ⊗ *indexed-const b* (*m* − *x*))› *indexed-const-def*)
qed
then show *?thesis*
by (*simp add*: *True indexed-const-def*)
next
case *False*
then show *?thesis* **using** *0 1*
by (*simp add*: *add.finprod-one-eqI assms*(*1*) *assms*(*2*) *indexed-const-def*)
qed
qed
qed

lemma *indexed-const-P-multE*:
assumes *P* ∈ *indexed-pset I* (*carrier R*)
assumes *c* ∈ *carrier R*
shows (*P* ⨂$_p$ (*indexed-const c*)) *m* = (*P m*) ⊗ *c*
unfolding *P-ring-mult-def*
proof−
have *3*: *m* ∉ *mset-factors m* − {*m*}
by *simp*
have *4*: *finite* ((*mset-factors m*) − {*m*})
by *simp*
have *5*: *P m* ⊗ *indexed-const c* (*m* − *m*) ∈ *carrier R*

by (*metis assms*(*1*) *assms*(*2*) *cancel-comm-monoid-add-class.diff-cancel carrier-coeffE indexed-const-def indexed-pset-in-carrier local.ring-axioms ring.ring-simprules*(*5*) *subsetI*)
have *0*: $(\lambda x.\ P\ x \otimes \textit{indexed-const}\ c\ (m - x)) \in \textit{mset-factors}\ m - \{m\} \to \textit{carrier}\ R$
proof
fix *x*
assume $x \in \textit{mset-factors}\ m - \{m\}$
then show $P\ x \otimes \textit{indexed-const}\ c\ (m - x) \in \textit{carrier}\ R$
using *assms*
by (*meson carrier-coeffE indexed-const-in-carrier indexed-pset-in-carrier m-closed subsetI*)
qed
have *1*: $P\ m \otimes \textit{indexed-const}\ c\ (\ m - m) = \ (P\ m \otimes c)$
by (*simp add*: *indexed-const-def*)
have *2*: $\bigwedge x.\ x \in \textit{mset-factors}\ m \Longrightarrow\ P\ x \otimes \textit{indexed-const}\ c\ (m - x)\ = (\textit{if}\ x = m\ \textit{then}\ (P\ m \otimes c)\ \textit{else}\ \mathbf{0})$
proof−
fix *x*
assume $x \in \textit{mset-factors}\ m$
then have $\textit{indexed-const}\ c\ (m - x) = (\textit{if}\ x = m\ \textit{then}\ c\ \textit{else}\ \mathbf{0})$
unfolding *indexed-const-def*
by (*metis cancel-comm-monoid-add-class.diff-cancel diff-zero monom-comp' monom-divides-factors*)
then show $P\ x \otimes \textit{indexed-const}\ c\ (m - x)\ = (\textit{if}\ x = m\ \textit{then}\ (P\ m \otimes c)\ \textit{else}\ \mathbf{0})$
by (*metis assms*(*1*) *carrier-coeffE indexed-pset-in-carrier local.semiring-axioms semiring.r-null set-eq-subset*)
qed
then have $(\bigoplus x{\in}(\textit{mset-factors}\ m) - \{m\}.\ P\ x \otimes \textit{indexed-const}\ c\ (m - x)) = \mathbf{0}$
using *assms*
by (*metis* (*no-types*, *lifting*) *DiffD1 DiffD2 add.finprod-one-eqI singletonI*)
then show $(\bigoplus x{\in}(\textit{mset-factors}\ m).\ P\ x \otimes \textit{indexed-const}\ c\ (m - x)) = P\ m \otimes c$
using *assms 0 1 3 4 5 finsum-insert*[*of* $(\textit{mset-factors}\ m) - \{m\}$ $m\ \lambda x.\ P\ x \otimes \textit{indexed-const}\ c\ (m - x)$]
by (*metis* (*no-types*, *lifting*) *m-factor add.l-cancel-one insert-Diff-single insert-absorb zero-closed*)
qed

lemma *indexed-const-var-mult*:
assumes $P \in \textit{indexed-pset}\ I\ (\textit{carrier}\ R)$
assumes $c \in \textit{carrier}\ R$
assumes $i \in I$
shows $P \bigotimes\ i \bigotimes_p \textit{indexed-const}\ c = (P \bigotimes_p (\textit{indexed-const}\ c)) \bigotimes\ i$
proof
fix *m*
show $(P \bigotimes\ i \bigotimes_p \textit{indexed-const}\ c)\ m = (P \bigotimes_p \textit{indexed-const}\ c \bigotimes\ i)\ m$
proof(*cases* $i \in\# m$)
case *True*

then have *T0*: $(P \bigotimes\ i \bigotimes_p \textit{indexed-const}\ c)\ m\ = (P \bigotimes\ i)\ m \otimes c$
using *assms indexed-const-P-multE*[*of* $P \bigotimes\ i\ I\ c\ m$]
by (*simp add*: *indexed-pset.indexed-pmult*)
then show *?thesis* **using** *assms indexed-const-P-multE*[*of P I c m*]
unfolding *indexed-pmult-def*
using *True indexed-const-P-multE* **by** *fastforce*
next
case *False*
then have *T0*: $(P \bigotimes\ i \bigotimes_p \textit{indexed-const}\ c)\ m\ = (P \bigotimes\ i)\ m \otimes c$
using *assms indexed-const-P-multE*[*of* $P \bigotimes\ i\ I\ c\ m$]
by (*simp add*: *indexed-pset.indexed-pmult*)
then show *?thesis* **using** *assms indexed-const-P-multE*[*of P I c m*]
unfolding *indexed-pmult-def*
using *False* **by** *auto*
qed
qed

lemma *indexed-const-P-mult-closed*:
assumes $a \in \textit{indexed-pset}\ I\ (\textit{carrier}\ R)$
assumes $c \in \textit{carrier}\ R$
shows $a \bigotimes_p (\textit{indexed-const}\ c) \in \textit{indexed-pset}\ I\ (\textit{carrier}\ R)$
apply(*rule indexed-pset.induct*[*of a I* (*carrier R*)])
apply (*simp add*: *assms*(*1*); *fail*)
proof−
show $\bigwedge k.\ (k \in \textit{carrier}\ R) \Longrightarrow ((\textit{indexed-const}\ k) \bigotimes_p (\textit{indexed-const}\ c)) \in ((\textit{carrier}\ R)\ [\mathcal{X}_I])$
using *assms*
by (*metis indexed-const-P-mult-eq indexed-pset.simps m-closed*)
show $\bigwedge P\ Q.\ P \in ((\textit{carrier}\ R)\ [\mathcal{X}_I]) \Longrightarrow P \bigotimes_p \textit{indexed-const}\ c \in ((\textit{carrier}\ R)\ [\mathcal{X}_I]) \Longrightarrow Q \in ((\textit{carrier}\ R)\ [\mathcal{X}_I]) \Longrightarrow Q \bigotimes_p \textit{indexed-const}\ c \in ((\textit{carrier}\ R)\ [\mathcal{X}_I]) \Longrightarrow P \bigoplus\ Q \bigotimes_p \textit{indexed-const}\ c \in ((\textit{carrier}\ R)\ [\mathcal{X}_I])$
using *P-ring-ldistr*
by (*metis assms*(*2*) *carrier-coeff-def indexed-const-in-carrier indexed-pset.indexed-padd indexed-pset-in-carrier subsetI*)
show $\bigwedge P\ i.\ P \in ((\textit{carrier}\ R)\ [\mathcal{X}_I]) \Longrightarrow P \bigotimes_p \textit{indexed-const}\ c \in ((\textit{carrier}\ R)\ [\mathcal{X}_I]) \Longrightarrow i \in I \Longrightarrow P \bigotimes\ i \bigotimes_p \textit{indexed-const}\ c \in ((\textit{carrier}\ R)\ [\mathcal{X}_I])$
proof−
fix *P i*
assume *A0*: $P \in ((\textit{carrier}\ R)\ [\mathcal{X}_I])$
assume *A1*: $(P \bigotimes_p \textit{indexed-const}\ c) \in ((\textit{carrier}\ R)\ [\mathcal{X}_I])$
assume *A2*: $i \in I$
show $P \bigotimes\ i \bigotimes_p \textit{indexed-const}\ c \in (\textit{carrier}\ R\ [\mathcal{X}_I])$
using *assms A0 A1 A2 indexed-const-var-mult local.ring-axioms ring.indexed-pset.simps*
by (*metis* (*no-types*, *opaque-lifting*))
qed
qed

lemma *monom-add-mset*:

mset-to-IP R (add-mset i m) = mset-to-IP R m $\bigotimes$ *i*
unfolding *indexed-pmult-def*
by (*metis* (*no-types*, *opaque-lifting*) *add-mset-diff-bothsides diff-empty mset-to-IP-def multi-member-split union-single-eq-member*)

Monomials are closed under multiplication:

lemma *monom-mult*:
mset-to-IP R m $\bigotimes_p$ *mset-to-IP R n = mset-to-IP R (m + n)*
proof−
have (*λma.* $\bigoplus$ *x∈mset-factors ma.* (*if x = m then* **1** *else* **0**) ⊗ (*if ma − x = n then* **1** *else* **0**))
= (*λma.* (*if ma = n + m then* **1** *else* **0**))
proof
fix *ma*
show ($\bigoplus$ *x∈mset-factors ma.* (*if x = m then* **1** *else* **0**) ⊗ (*if ma − x = n then* **1** *else* **0**)) = (*if ma = n + m then* **1** *else* **0**)

proof−
have *0*: $\bigwedge$*x. x ∈ mset-factors ma ⟹*
(*λ x.* (*if x = m then* **1** *else* **0**) ⊗ (*if ma − x = n then* **1** *else* **0**)) *x* =
(*if* (*x = m*) *then* (*if ma − x = n then* **1** *else* **0**) *else* **0**)
by *simp*
then have *1*: ($\bigoplus$ *x∈*((*mset-factors ma*) − {*m*}). (*if x = m then* **1** *else* **0**) ⊗ (*if ma − x = n then* **1** *else* **0**)) =
0
by (*metis* (*no-types*, *lifting*) *add.finprod-one-eqI mem-Collect-eq set-diff-eq singletonI*)
have *2*: *finite* (*mset-factors ma* − {*m*})
by *simp*
have *3*: *m* ∉ *mset-factors ma* − {*m*}
by *simp*
have *4*: (*λx.* (*if x = m then* **1** *else* **0**) ⊗ (*if ma − x = n then* **1** *else* **0**)) ∈ *mset-factors ma* − {*m*} → *carrier R*
proof
fix *x*
assume *x* ∈ *mset-factors ma* − {*m*}
show (*if x = m then* **1** *else* **0**) ⊗ (*if ma − x = n then* **1** *else* **0**) ∈ *carrier R*
by *auto*
qed
show *?thesis*
proof(*cases m* ∈ (*mset-factors ma*))
case *True*
have *T0*: ($\bigoplus$ *x∈insert m* (*mset-factors ma* − {*m*}). (*if x = m then* **1** *else* **0**) ⊗ (*if ma − x = n then* **1** *else* **0**)) =
(*if m = m then* **1** *else* **0**) ⊗ (*if ma − m = n then* **1** *else* **0**) ⊕
($\bigoplus$ *x∈mset-factors ma* − {*m*}. (*if x = m then* **1** *else* **0**) ⊗ (*if ma − x = n then* **1** *else* **0**))
using *True 1 2 3 4 finsum-insert*[*of* ((*mset-factors ma*) − {*m*}) *m*
λx. (*if x = m then* **1** *else* **0**) ⊗ (*if ma − x = n then* **1**

else **0**)]
using *m-closed one-closed zero-closed* **by** *presburger*
have *T1*: (*mset-factors ma*) = *insert m* (*mset-factors ma* − {*m*})
using *True*
by *blast*
then have (⨁ *x*∈(*mset-factors ma*). (*if x* = *m then* **1** *else* **0**) ⊗ (*if ma* − *x* = *n then* **1** *else* **0**))
= (*if m* = *m then* **1** *else* **0**) ⊗ (*if ma* − *m* = *n then* **1** *else* **0**)
using *T0 T1 1 add.l-cancel-one insert-Diff-single insert-absorb2 l-one mk-disjoint-insert one-closed zero-closed*
by *presburger*
then have (⨁ *x*∈(*mset-factors ma*). (*if x* = *m then* **1** *else* **0**) ⊗ (*if ma* − *x* = *n then* **1** *else* **0**))
= (*if ma* − *m* = *n then* **1** *else* **0**)
by *simp*
then show *?thesis*
by (*metis* (*no-types*, *lifting*) *monom-divides-factors True add.commute add-diff-cancel-right′ subset-mset.add-diff-inverse*)
next
case *False*
then show *?thesis*
by (*metis* (*no-types*, *lifting*) *0 monom-divides-factors add.finprod-one-eqI mset-subset-eq-add-right*)
qed
qed
qed
then show *?thesis*
unfolding *mset-to-IP-def P-ring-mult-def*
by (*simp add*: *union-commute*)
qed

lemma *poly-index-mult*:
assumes *a* ∈ *indexed-pset I* (*carrier R*)
assumes *i* ∈ *I*
shows *a* ⨂ *i* = *a* ⨂$_p$ *mset-to-IP R* {#*i*#}
proof
fix *m*
show (*a* ⨂ *i*) *m* = (*a* ⨂$_p$ *mset-to-IP R* {#*i*#}) *m*
proof(*cases i* ∈# *m*)
case *True*
have *T0*: (*a* ⨂ *i*) *m* = *a* (*m* − {#*i*#})
by (*simp add*: *True indexed-pmult-def*)
have *T1*: (*a* ⨂$_p$ *mset-to-IP R* {#*i*#}) *m* = *a* (*m* − {#*i*#})
proof−
have *T10*: (*a* ⨂$_p$ *mset-to-IP R* {#*i*#}) *m*
= (⨁ *x*∈*mset-factors m*. *a x* ⊗ *mset-to-IP R* {#*i*#} (*m* − *x*))
unfolding *P-ring-mult-def*
by *auto*
have *T11*: ⋀*x*. *x* ∈ *mset-factors m* ⟹

mset-to-IP R {#i#} (m − x) = (if x = (m − {#i#}) then **1** *else* **0**)
unfolding *mset-to-IP-def*
by (*metis Multiset.diff-cancel Multiset.diff-right-commute True diff-single-eq-union diff-single-trivial diff-zero monom-comp′ monom-divides-factors multi-drop-mem-not-eq*)
have *T12*: $\bigwedge x.\ x \in$ *mset-factors m* $\Longrightarrow$ *a x ⊗ mset-to-IP R {#i#} (m − x) = (if x = (m − {#i#}) then a (m − {#i#}) else* **0**)
proof−
fix *x*
assume *x ∈ mset-factors m*
then show *a x ⊗ mset-to-IP R {#i#} (m − x) = (if x = m − {#i#} then a (m − {#i#}) else* **0**)
apply(*cases x = m − {#i#}*)
apply (*metis T0 T11 ‹x ∈ mset-factors m› assms(1) carrier-coeffE empty-subsetI ideal-is-subalgebra indexed-pset-in-carrier oneideal r-one subalgebra-in-carrier*)
by (*metis T11 ‹x ∈ mset-factors m› assms(1) carrier-coeffE carrier-is-subalgebra empty-subsetI indexed-pset-in-carrier r-null subalgebra-in-carrier*)
qed
have *T13*: $(\bigoplus x \in$ *(mset-factors m) − {m − {#i#}}. a x ⊗ mset-to-IP R {#i#} (m − x)) =* **0**
using *T12*
by (*metis* (*no-types*, *lifting*) *DiffE add.finprod-one-eqI singletonI*)
have *T14*: *finite* (*mset-factors m − {m − {#i#}}*)
using *mset-factors-finite* **by** *blast*
have *T15*: *m − {#i#} ∉ mset-factors m − {m − {#i#}}*
by *simp*
have *T16*: $(\lambda x.$ *a x ⊗ mset-to-IP R {#i#} (m − x)) ∈ mset-factors m − {m − {#i#}} → carrier R*
proof
fix *x*
assume *x ∈ mset-factors m − {m − {#i#}}*
then show *a x ⊗ mset-to-IP R {#i#} (m − x) ∈ carrier R*
using *assms T12* **by** *auto*
qed
have *m − (m − {#i#}) = {#i#}*
by (*metis monom-comp′ True single-subset-iff*)
then have *T17*: *a (m − {#i#}) ⊗ mset-to-IP R {#i#} (m − (m − {#i#})) ∈ carrier R*
unfolding *mset-to-IP-def* **apply** *auto* **using** *assms*
by (*meson carrier-coeffE carrier-is-subalgebra empty-subsetI indexed-pset-in-carrier m-closed one-closed subalgebra-in-carrier*)
have *T18*:$(\bigoplus x \in$*insert (m − {#i#}) (mset-factors m − {m − {#i#}}). a x ⊗ mset-to-IP R {#i#} (m − x)) =*
a (m − {#i#}) ⊗ mset-to-IP R {#i#} (m − (m − {#i#})) ⊕ ($\bigoplus x \in$*mset-factors m − {m − {#i#}}. a x ⊗ mset-to-IP R {#i#} (m − x))*
using *T12 T13 T14 T15 T16 T17* **unfolding** *P-ring-mult-def*

using *finsum-insert*[*of mset-factors* $m - \{m - \{\#i\#\}\}$ $m - \{\#i\#\}$
$\lambda x.\ a\ x \otimes$ *mset-to-IP* R $\{\#i\#\}$ $(m - x)$]
by *blast*
have *T19*: $a\ (m - \{\#i\#\}) \otimes$ *mset-to-IP* R $\{\#i\#\}$ $(m - (m - \{\#i\#\})) = a\ (m - \{\#i\#\})$
proof−
have $(m - (m - \{\#i\#\})) = \{\#i\#\}$
using *True*
by (*metis monom-comp*′ *single-subset-iff*)
then have *mset-to-IP* R $\{\#i\#\}$ $(m - (m - \{\#i\#\})) = \mathbf{1}$
by (*metis mset-to-IP-def*)
have $a\ (m - \{\#i\#\}) \in$ *carrier* R
using *assms*
by (*meson carrier-coeffE carrier-is-subalgebra exp-base-closed indexed-pset-in-carrier one-closed subalgebra-in-carrier*)
then show *?thesis* **using** *assms*
using ‹*mset-to-IP* R $\{\#i\#\}$ $(m - (m - \{\#i\#\})) = \mathbf{1}$› **by** *auto*
qed
have *T20*: $(\bigoplus x \in$ *insert* $(m - \{\#i\#\})$ $((mset\text{-}factors\ m) - \{m - \{\#i\#\}\}).$ $a\ x \otimes$ *mset-to-IP* R $\{\#i\#\}$ $(m - x)) =$
$a\ (m - \{\#i\#\}) \oplus (\bigoplus x \in mset\text{-}factors\ m - \{m - \{\#i\#\}\}.\ a\ x \otimes$ *mset-to-IP* R $\{\#i\#\}$ $(m - x))$
using *T18 T19* **by** *auto*
have *T21*: *insert* $(m - \{\#i\#\})$ $((mset\text{-}factors\ m) - \{m - \{\#i\#\}\}) =$ *mset-factors* m
proof−
have $(m - \{\#i\#\}) \in mset\text{-}factors\ m$
by (*simp add*: *monom-divides-factors*)
then show *?thesis*
by *blast*
qed
then show *?thesis* **using** *T13 T20 True* **unfolding** *P-ring-mult-def*
using *T17 T19* **by** *auto*
qed
then show *?thesis*
by (*simp add*: *T0*)
next
case *False*
have *F0*: $(a \bigotimes i)\ m = \mathbf{0}$
by (*simp add*: *False indexed-pmult-def*)
have *F1*: $(a \bigotimes_p$ *mset-to-IP* R $\{\#i\#\})$ $m = \mathbf{0}$
unfolding *P-ring-mult-def*
proof−
have $\bigwedge x.\ x \in mset\text{-}factors\ m \Longrightarrow\ a\ x \otimes$ *mset-to-IP* R $\{\#i\#\}$ $(m - x) = \mathbf{0}$
proof−
fix x
assume *A*: $x \in mset\text{-}factors\ m$
have *B*: $m - x \neq \{\#i\#\}$ **using** *False A*
by (*metis diff-subset-eq-self single-subset-iff*)

```
      then show a x ⊗ mset-to-IP R {#i#} (m − x) = 0
        using assms False unfolding mset-to-IP-def
        using  carrier-coeffE indexed-pset-in-carrier by fastforce
    qed
    then show (⨁x∈mset-factors m. a x ⊗ mset-to-IP R {#i#} (m − x)) = 0
      by (simp add: add.finprod-one-eqI)
  qed
  then show ?thesis
    by (simp add: F0)
 qed
qed

lemma mset-to-IP-mult-closed:
 assumes a ∈ indexed-pset I (carrier R)
 shows set-mset m ⊆ I ⟹ a ⨂p mset-to-IP R m ∈ indexed-pset I (carrier R)
proof(induction m)
 case empty
 then have mset-to-IP R {#} = indexed-const 1
  unfolding mset-to-IP-def indexed-const-def by auto
 then show ?case
  by (simp add: ‹mset-to-IP R {#} = indexed-const 1› assms indexed-const-P-mult-closed)
next
 case (add x m)
 fix x
 fix m :: 'c monomial
 assume A: set-mset (add-mset x m) ⊆ I
 assume B: set-mset m ⊆ I ⟹ a ⨂p mset-to-IP R m ∈ (carrier R [𝒳I])
 show  a ⨂p mset-to-IP R (add-mset x m) ∈ (carrier R [𝒳I])
 proof−
  have 0: x ∈ I
   using A by auto
  have 1: set-mset m ⊆ I
   using A by auto
  have 2:  a ⨂p mset-to-IP R m ∈ (carrier R [𝒳I])
   using 1 B by blast
  have 3:  a ⨂p mset-to-IP R (add-mset x m) = (a ⨂p mset-to-IP R m) ⨂ x
  proof−
   have 30:  a ⨂p mset-to-IP R (add-mset x m) = a ⨂p (mset-to-IP R m ⨂
x)
    using monom-add-mset[of x m]
    by auto
   have 31: (a ⨂p mset-to-IP R m) ⨂ x = a ⨂p (mset-to-IP R m ⨂ x)
   proof
    fix y
   show (a ⨂p mset-to-IP R m ⨂ x) y = (a ⨂p (mset-to-IP R m ⨂ x)) y
    proof−
     have 310: (a ⨂p mset-to-IP R m ⨂ x) y = (a ⨂p mset-to-IP R m ⨂p
mset-to-IP R {#x#}) y
       using poly-index-mult 0 2
```

```
        by fastforce
      have 311: (mset-to-IP R m ⨂ x) = mset-to-IP R m ⨂_p mset-to-IP R
{#x#}
        using 0 2
        by (metis add-mset-add-single monom-add-mset monom-mult)
      have 312: carrier-coeff a
        using assms indexed-pset-in-carrier by blast
      have 313: carrier-coeff (mset-to-IP R m)
        by (simp add: carrier-coeff-def mset-to-IP-def)
      have 314: carrier-coeff (mset-to-IP R {#x#})
        by (metis carrier-coeff-def mset-to-IP-def one-closed zero-closed)
      show ?thesis
        using 310 311 312 313 314
            P-ring-mult-assoc[of a mset-to-IP R m mset-to-IP R {#x#}]
        by simp
    qed
   qed
   then show ?thesis
    by (simp add: monom-add-mset)
  qed
  then show ?thesis
   by (simp add: 0 1 B indexed-pset.indexed-pmult)
 qed
qed
```

A predicate which identifies when the variables used in a given polynomial P are only drawn from a fixed variable set I.

```
abbreviation monoms-in where
monoms-in I P ≡ (∀ m ∈ monomials-of R P. set-mset m ⊆ I)

lemma monoms-of-const:
monomials-of R (indexed-const k) = (if k = 0 then {} else {{#}})
 unfolding indexed-const-def monomials-of-def
 by auto

lemma const-monoms-in:
monoms-in I (indexed-const k)
 unfolding indexed-const-def monomials-of-def
 using count-empty count-eq-zero-iff monomials-ofE subsetI
 by simp

lemma mset-to-IP-indices:
 shows P ∈ indexed-pset I (carrier R) ⟹ monoms-in I P
 apply(erule indexed-pset.induct[of])
  apply (simp add: const-monoms-in; fail)
proof−
 show ⋀P Q. P ∈ (carrier R [𝒳_I]) ⟹
        ∀ m∈monomials-of R P. set-mset m ⊆ I ⟹
         Q ∈ (carrier R [𝒳_I]) ⟹ ∀ m∈monomials-of R Q. set-mset m ⊆ I ⟹
```

$\forall\, m{\in}$*monomials-of* R $(P \bigoplus Q)$. *set-mset* $m \subseteq I$
proof
fix P Q
fix m
assume *A*: $P \in$ (*carrier* R $[\mathcal{X}_I]$) $\forall\, m{\in}$*monomials-of* R P. *set-mset* $m \subseteq I$
$Q \in$ (*carrier* R $[\mathcal{X}_I]$) $\forall\, m{\in}$*monomials-of* R Q. *set-mset* $m \subseteq I$
$m \in$ *monomials-of* R $(P \bigoplus Q)$
show *set-mset* $m \subseteq I$
proof −
have $m \in$ *monomials-of* R $P \vee m \in$ *monomials-of* R Q
using A **using** *monomials-of-add*[*of* P Q]
by *blast*
then show *?thesis* **using** A
by *blast*
qed
qed
show $\bigwedge P$ *i*. $P \in$ (*carrier* R $[\mathcal{X}_I]$) $\Longrightarrow \forall\, m{\in}$*monomials-of* R P. *set-mset* $m \subseteq I$
$\Longrightarrow i \in I \Longrightarrow \forall\, m{\in}$*monomials-of* R $(P \bigotimes i)$. *set-mset* $m \subseteq I$
proof
fix P
fix i
fix m
assume *A*: $P \in$ (*carrier* R $[\mathcal{X}_I]$) $\forall\, m{\in}$*monomials-of* R P. *set-mset* $m \subseteq I$ $i \in I$ $m \in$ *monomials-of* R $(P \bigotimes i)$
obtain n **where** $n \in$ *monomials-of* R $P \wedge m =$ *add-mset* i n
using A
by (*metis image-iff monomials-of-p-mult'*)
then show *set-mset* $m \subseteq I$
using A
by *auto*
qed
qed

lemma *mset-to-IP-indices'*:
assumes $P \in$ *indexed-pset* I (*carrier* R)
assumes $m \in$ *monomials-of* R P
shows *set-mset* $m \subseteq I$
using *assms*(*1*) *assms*(*2*) *mset-to-IP-indices* **by** *blast*

lemma *one-mset-to-IP*:
mset-to-IP R $\{\#\}$ = *indexed-const* **1**
unfolding *mset-to-IP-def indexed-const-def*
by *blast*

lemma *mset-to-IP-closed*:
shows *set-mset* $m \subseteq I \Longrightarrow$*mset-to-IP* R $m \in$ *indexed-pset* I (*carrier* R)
apply(*induction* m)
apply (*simp add*: *indexed-pset.indexed-const one-mset-to-IP*)
by (*metis* (*no-types*, *lifting*) *add-mset-commute indexed-pset.simps*

monom-add-mset mset-add subset-iff union-single-eq-member)

lemma *mset-to-IP-closed′*:
assumes $P \in$ *indexed-pset I* (*carrier R*)
assumes $m \in$ *monomials-of R P*
shows *mset-to-IP R m* $\in$ *indexed-pset I* (*carrier R*)
by (*meson assms*(*1*) *assms*(*2*) *mset-to-IP-closed mset-to-IP-indices′*)

This lemma expresses closure under multiplcation for indexed polynomials.

lemma *P-ring-mult-closed*:
assumes *carrier-coeff P*
assumes *carrier-coeff Q*
shows *carrier-coeff* (*P-ring-mult R P Q*)
unfolding *carrier-coeff-def*
proof
fix m
have $(\lambda x.\ P\ x \otimes Q\ (m - x)) \in$ *mset-factors m* $\to$ *carrier R*
proof
fix x
assume $x \in$ *mset-factors m*
then show $P\ x \otimes Q\ (m - x) \in$ *carrier R*
using *assms carrier-coeffE*
by *blast*
qed
then show $(P \bigotimes_p Q)\ m \in$ *carrier R*
using *assms finsum-closed*[*of* $(\lambda x.\ (P\ x) \otimes (Q\ (m - x)))$ *mset-factors m*]
unfolding *carrier-coeff-def P-ring-mult-def*
by *blast*
qed

2.5 Multivariable Polynomial Induction

lemma *mpoly-induct*:
assumes $\bigwedge Q.\ Q \in$ *indexed-pset I* (*carrier R*) $\wedge$ *card* (*monomials-of R Q*) $= 0 \Longrightarrow P\ Q$
assumes $\bigwedge n.\ (\bigwedge Q.\ Q \in$ *indexed-pset I* (*carrier R*) $\wedge$ *card* (*monomials-of R Q*) $\leq n \Longrightarrow P\ Q) \Longrightarrow (\bigwedge Q.\ Q \in$ *indexed-pset I* (*carrier R*) $\wedge$ *card* (*monomials-of R Q*) $\leq$ (*Suc n*) $\Longrightarrow P\ Q)$
assumes $Q \in$ *indexed-pset I* (*carrier R*)
shows $P\ Q$
proof−
have $\bigwedge m.\ \bigwedge Q.\ Q \in$ *indexed-pset I* (*carrier R*) $\wedge$ *card*(*monomials-of R Q*) $\leq m \Longrightarrow P\ Q$
proof−
fix m
show $\bigwedge Q.\ Q \in$ *indexed-pset I* (*carrier R*) $\wedge$ *card*(*monomials-of R Q*) $\leq m \Longrightarrow P\ Q$
apply(*induction m*)

```
    using assms(1)
     apply blast
    using assms
    by blast
  qed
  then show ?thesis
    using assms(3) by blast
qed

lemma monomials-of-card-zero:
  assumes Q ∈ indexed-pset I (carrier R) ∧ card (monomials-of R Q) = 0
  shows Q = indexed-const 0
proof−
  have 0: carrier-coeff  Q
    using assms  indexed-pset-in-carrier
    by blast
  have ⋀m. Q m = 0
    unfolding monomials-of-def
    using assms monomials-finite
    by (metis card-0-eq complement-of-monomials-of empty-iff)
  then show  ?thesis
  using 0 assms unfolding indexed-const-def
  by auto
qed
```

Polynomial induction on the number of monomials with nonzero coefficient:

```
lemma mpoly-induct′:
  assumes P (indexed-const 0)
  assumes ⋀n. (⋀Q. Q ∈ indexed-pset I (carrier R) ∧ card (monomials-of R Q)
≤  n ⟹ P Q)
            ⟹ (⋀Q. Q ∈ indexed-pset I (carrier R) ∧ card (monomials-of R Q)
= (Suc n) ⟹ P Q)
  assumes Q ∈ indexed-pset I (carrier R)
  shows P Q
  apply(rule mpoly-induct)
  using assms(1) monomials-of-card-zero apply blast
proof−
  show ⋀n Q. (⋀Q. Q ∈ (carrier R [𝒳_I]) ∧ card (monomials-of R Q) ≤ n ⟹ P
Q) ⟹ Q ∈ (carrier R [𝒳_I]) ∧ card (monomials-of R Q) ≤ Suc n ⟹ P Q
  proof−
    fix n
    fix Q
    assume A0: (⋀Q. Q ∈ (carrier R [𝒳_I]) ∧ card (monomials-of R Q) ≤ n ⟹
P Q)
    assume A1: Q ∈ (carrier R [𝒳_I]) ∧ card (monomials-of R Q) ≤ Suc n
    show P Q
      apply(cases  card (monomials-of R Q) = Suc n)
      using assms A0 A1
       apply blast
```

```
    using assms A0 A1
    using le-SucE by blast
  qed
  show Q ∈ (carrier R [𝒳_I])
    using assms by auto
qed

lemma monomial-poly-split:
  assumes P ∈ indexed-pset I (carrier R)
  assumes m ∈ monomials-of R P
  shows (restrict-poly-to-monom-set R P ((monomials-of R P) − {m})) ⊕ ((mset-to-IP
R m) ⊗_p (indexed-const (P m))) = P
proof fix x
  show (restrict-poly-to-monom-set R P (monomials-of R P − {m}) ⊕ (mset-to-IP
R m ⊗_p indexed-const (P m))) x = P x
  proof(cases x = m)
    case True
    have T0: (restrict-poly-to-monom-set R P (monomials-of R P − {m})) x = 0
      unfolding restrict-poly-to-monom-set-def
      by (simp add: True)
    have T1: (mset-to-IP R m ⊗_p indexed-const (P m)) x = P x
      using assms True indexed-const-P-multE[of mset-to-IP R m I P m m]
            mset-to-IP-simp
      by (metis Idl-subset-ideal' carrier-coeffE genideal-one indexed-pset-in-carrier
          l-one mset-to-IP-closed' one-closed)
    then show ?thesis
      using T0 True
      unfolding indexed-padd-def
      using assms(1) carrier-coeffE indexed-pset-in-carrier l-zero
      by fastforce
  next
    case False
    then have F0: (restrict-poly-to-monom-set R P (monomials-of R P − {m}))
x = P x
    proof(cases x ∈ monomials-of R P)
      case True
      then show ?thesis
        by (simp add: False restrict-poly-to-monom-set-def)
    next
      case False
      then show ?thesis
        by (simp add: complement-of-monomials-of restrict-poly-to-monom-set-def)
    qed
    have F1: mset-to-IP R m ∈ (carrier R [𝒳_I])
      using assms(1) assms(2) mset-to-IP-closed' by blast
    have F2: P m ∈ carrier R
      using assms(1) carrier-coeffE indexed-pset-in-carrier by blast
    have F3: (mset-to-IP R m ⊗_p indexed-const (P m)) x = 0
      using False F1 F2 assms indexed-const-P-multE[of mset-to-IP R m I P m x]
```

```
    by (simp add: F1 ‹m ∈ monomials-of R P›)
  then show ?thesis using F0 unfolding indexed-padd-def
    using assms(1) carrier-coeffE indexed-pset-in-carrier
    by fastforce
 qed
qed

lemma restrict-not-in-monoms:
 assumes a ∉ monomials-of R P
 shows restrict-poly-to-monom-set R P A = restrict-poly-to-monom-set R P (insert
a A)
proof
 fix x
  show  restrict-poly-to-monom-set R P A x = restrict-poly-to-monom-set R P
(insert a A) x
   unfolding restrict-poly-to-monom-set-def using assms unfolding monomi-
als-of-def
 by simp
qed

lemma restriction-closed':
 assumes P ∈ indexed-pset I (carrier R)
 assumes finite ms
 shows (restrict-poly-to-monom-set R P ms) ∈ indexed-pset I (carrier R)
 apply(rule finite.induct[of ms])
 apply (simp add: assms(2); fail)
proof−
 show restrict-poly-to-monom-set R P {} ∈ (carrier R [𝒳_I])
 proof−
  have restrict-poly-to-monom-set R P {} = indexed-const 0
   unfolding restrict-poly-to-monom-set-def indexed-const-def by auto
  then show ?thesis
   by (simp add: indexed-pset.indexed-const)
 qed
 show ⋀A a. finite A ⟹ restrict-poly-to-monom-set R P A ∈ (carrier R [𝒳_I])
⟹ restrict-poly-to-monom-set R P (insert a A) ∈ (carrier R [𝒳_I])
 proof−
  fix A
  fix a
  assume A: restrict-poly-to-monom-set R P A ∈ (carrier R [𝒳_I])
  show restrict-poly-to-monom-set R P (insert a A) ∈ (carrier R [𝒳_I])
  proof(cases a ∈ monomials-of R P)
   case True
   then have T0: mset-to-IP R a ∈ (carrier R [𝒳_I])
    using assms(1) mset-to-IP-closed' by blast
   then have T1: (mset-to-IP R a ⨂_p indexed-const (P a)) ∈  (carrier R [𝒳_I])
    by (meson assms(1) carrier-coeffE indexed-pset-in-carrier
   local.ring-axioms ring.indexed-const-P-mult-closed subset-refl)
```

show *?thesis*
proof(*cases a ∈ A*)
case *True*
then show *?thesis*
by (*simp add*: *A insert-absorb*)
next
case *False*
have *restrict-poly-to-monom-set R P* (*insert a A*) = *restrict-poly-to-monom-set R P A* ⊕ (*mset-to-IP R a* ⊗$_p$ *indexed-const* (*P a*))
proof
fix *x*
show *restrict-poly-to-monom-set R P* (*insert a A*) *x* = (*restrict-poly-to-monom-set R P A* ⊕ (*mset-to-IP R a* ⊗$_p$ *indexed-const* (*P a*))) *x*
proof(*cases x = a*)
case *True*
then have *FT0*: (*if x ∈ insert a A then P x else* **0**) = *P x*
by *simp*
have *FT1*: (*if x ∈ A then P x else* **0**) = **0**
by (*simp add*: *False True*)
have *FT2*: *P a ∈ carrier R*
using *assms*(*1*) *carrier-coeffE indexed-pset-in-carrier* **by** *blast*
have *FT3*: (*mset-to-IP R a* ⊗$_p$ *indexed-const* (*P a*)) *x* = *P x*
using *T0 FT2 True indexed-const-P-multE*[*of mset-to-IP R a I P a x*] *mset-to-IP-simp*[*of a*]
by *simp*
then show *?thesis*
using *False*
unfolding *restrict-poly-to-monom-set-def indexed-padd-def*
using *FT0 FT1 FT3 assms*
by (*simp add*: *FT2 True*)
next
case *F*: *False*
then have *FT0*: (*if x ∈ insert a A then P x else* **0**) = (*if x ∈ A then P x else* **0**)
by *simp*
have *FT2*: *P a ∈ carrier R*
using *assms*(*1*) *carrier-coeffE indexed-pset-in-carrier* **by** *blast*
have *FT3*: (*mset-to-IP R a* ⊗$_p$ *indexed-const* (*P a*)) *x* = **0**
using *T0 FT2 True indexed-const-P-multE*[*of mset-to-IP R a I P a x*] *mset-to-IP-simp*[*of a*]
by (*simp add*: *F mset-to-IP-def*)
show *?thesis*
unfolding *restrict-poly-to-monom-set-def indexed-padd-def*
proof−

show (*if x ∈ insert a A then P x else* **0**) = (*if x ∈ A then P x else* **0**) ⊕ (*mset-to-IP R a* ⊗$_p$ *indexed-const* (*P a*)) *x*
apply(*cases x ∈ A*)
using *FT0 FT2 FT3 F False assms carrier-coeffE indexed-pset-in-carrier*

local.ring-axioms
apply *fastforce*
using *FT0 FT2 FT3 F False assms*
by *simp*
qed
qed
qed
then show *?thesis*
using *A T1 indexed-pset.simps* **by** *blast*
qed
next
case *False*
then show *?thesis*
using *A restrict-not-in-monoms* **by** *fastforce*
qed
qed
qed

lemma *restriction-restrict*:
restrict-poly-to-monom-set R P ms = restrict-poly-to-monom-set R P (ms ∩ monomials-of R P)
proof
fix *x*
show *restrict-poly-to-monom-set R P ms x = restrict-poly-to-monom-set R P (ms ∩ monomials-of R P) x*
unfolding *restrict-poly-to-monom-set-def monomials-of-def*
by *simp*
qed

lemma *restriction-closed*:
assumes *P ∈ indexed-pset I (carrier R)*
assumes *Q = restrict-poly-to-monom-set R P ms*
shows *Q ∈ indexed-pset I (carrier R)*
proof−
have *Q = restrict-poly-to-monom-set R P (ms ∩ monomials-of R P)*
using *assms restriction-restrict*
by *blast*
then show *?thesis*
using *assms restriction-closed'*[*of P I (ms ∩ monomials-of R P)*]
using *monomials-finite*
by *blast*
qed

lemma *monomial-split-card*:
assumes *P ∈ indexed-pset I (carrier R)*
assumes *m ∈ monomials-of R P*
shows *card (monomials-of R (restrict-poly-to-monom-set R P ((monomials-of R P) − {m})))=*
card (monomials-of R P) −1

proof−
have *0*: (*monomials-of R* (*restrict-poly-to-monom-set R P* ((*monomials-of R P*) − {*m*}))) =
(*monomials-of R P*) − {*m*}
using *assms*(*1*)
by (*meson Diff-subset restrict-poly-to-monom-set-monoms*)
then have *1*: *card* (*monomials-of R* (*restrict-poly-to-monom-set R P* ((*monomials-of R P*) − {*m*}))) =
card ((*monomials-of R P*) − {*m*})
by *auto*
have *card* ((*monomials-of R P*) − {*m*}) = *card* (*monomials-of R P*) − *1*
using *assms*(*2*)
using *assms*(*1*) *monomials-finite* **by** *fastforce*
then show *?thesis*
by (*simp add*: *1*)
qed

lemma *P-ring-mult-closed′*:
assumes *a* ∈ *indexed-pset I* (*carrier R*)
assumes *b* ∈ *indexed-pset I* (*carrier R*)
shows *a* ⨂$_p$ *b* ∈ *indexed-pset I* (*carrier R*)
apply(*rule mpoly-induct′*[*of* λ*b*. *a* ⨂$_p$ *b* ∈ *indexed-pset I* (*carrier R*) *I b*])
using *assms*(*1*) *indexed-const-P-mult-closed* **apply** *blast*
proof−
show *b* ∈ (*carrier R* [$\mathcal{X}_I$])
using *assms* **by** *auto*
show ⋀*n Q*. (⋀*Q*. *Q* ∈ (*carrier R* [$\mathcal{X}_I$]) ∧ *card* (*monomials-of R Q*) ≤ *n* ⟹ *a* ⨂$_p$ *Q* ∈ (*carrier R* [$\mathcal{X}_I$])) ⟹
Q ∈ (*carrier R* [$\mathcal{X}_I$]) ∧ *card* (*monomials-of R Q*) = *Suc n* ⟹ *a* ⨂$_p$ *Q* ∈ (*carrier R* [$\mathcal{X}_I$])
proof−
fix *n*
fix *Q*
assume *A0*: (⋀*Q*. *Q* ∈ (*carrier R* [$\mathcal{X}_I$]) ∧ *card* (*monomials-of R Q*) ≤ *n* ⟹ *a* ⨂$_p$ *Q* ∈ (*carrier R* [$\mathcal{X}_I$]))
assume *A1*: *Q* ∈ (*carrier R* [$\mathcal{X}_I$]) ∧ *card* (*monomials-of R Q*) = *Suc n*
obtain *m* **where** *m-def*: *m* ∈ *monomials-of R Q*
using *A1*
by *fastforce*
obtain *P* **where** *P-def*: *P* = (*restrict-poly-to-monom-set R Q* ((*monomials-of R Q*) − {*m*}))
by *simp*
have *P* ∈ (*carrier R* [$\mathcal{X}_I$])
using *P-def A1 restriction-closed*
by *blast*
have *0*: *a* ⨂$_p$ *P* ∈ (*carrier R* [$\mathcal{X}_I$])
using *A0 P-def*
by (*metis A1 One-nat-def* ‹*P* ∈ (*carrier R* [$\mathcal{X}_I$])› *diff-Suc-Suc m-def minus-nat.diff-0 monomial-split-card nat-le-linear*)

have *1*: $a \bigotimes_p (\textit{mset-to-IP } R\ m) \in (\textit{carrier } R\ [\mathcal{X}_I])$
using *assms mset-to-IP-mult-closed*[*of a I m*] *A1 m-def mset-to-IP-indices*
by *blast*
have *2*: $a \bigotimes_p (\textit{mset-to-IP } R\ m \bigotimes_p \textit{indexed-const } (Q\ m)) \in (\textit{carrier } R\ [\mathcal{X}_I])$
using *P-ring-mult-assoc*[*of a mset-to-IP R m indexed-const* (*Q m*)]
1
by (*metis A1 assms*(*1*) *carrier-coeffE indexed-pset.indexed-const indexed-pset-in-carrier local.ring-axioms m-def mset-to-IP-closed′ ring.indexed-const-P-mult-closed set-eq-subset*)
have *3*: $(P \bigoplus (\textit{mset-to-IP } R\ m \bigotimes_p \textit{indexed-const } (Q\ m))) = Q$
using *P-def monomial-poly-split*[*of Q I m*]
using *A1 m-def* **by** *blast*
have *4*: $(a \bigotimes_p P) \bigoplus (a \bigotimes_p (\textit{mset-to-IP } R\ m \bigotimes_p \textit{indexed-const } (Q\ m))) = a \bigotimes_p Q$
using *assms 2 3 P-ring-rdistr*[*of a P* (*mset-to-IP R m* $\bigotimes_p$ *indexed-const* (*Q m*))]
by (*metis A1 Idl-subset-ideal′ P-ring-mult-closed* ‹$P \in (\textit{carrier } R\ [\mathcal{X}_I])$› *carrier-coeffE indexed-pset.indexed-const indexed-pset-in-carrier m-def mset-to-IP-closed′ onepideal principalideal.generate*)
then show $a \bigotimes_p Q \in (\textit{carrier } R\ [\mathcal{X}_I])$
by (*metis 0 2 indexed-pset.indexed-padd*)
qed
qed

end

2.6 Subtraction of Polynomials

definition *P-ring-uminus* :: ('*a*,'*b*) *ring-scheme* ⇒ ('*a*,'*c*) *mvar-poly* ⇒ ('*a*,'*c*) *mvar-poly* **where**
P-ring-uminus R P = $(\lambda m.\ \ominus_R (P\ m))$

context *ring*

begin

lemma *P-ring-uminus-eq*:
assumes $a \in \textit{indexed-pset } I\ (\textit{carrier } R)$
shows $\textit{P-ring-uminus } R\ a = a \bigotimes_p (\textit{indexed-const } (\ominus \mathbf{1}))$
proof
fix *x*
have $(a \bigotimes_p \textit{indexed-const } (\ominus \mathbf{1}))\ x = a\ x \otimes \ominus \mathbf{1}$
using *indexed-const-P-multE*[*of a I* $\ominus \mathbf{1}$ *x*] *assms*
by *blast*
then show $\textit{P-ring-uminus } R\ a\ x = (a \bigotimes_p \textit{indexed-const } (\ominus \mathbf{1}))\ x$
unfolding *P-ring-uminus-def*
using *assms*
by (*metis Idl-subset-ideal′ carrier-coeffE indexed-pset-in-carrier*

one-closed onepideal principalideal.generate r-minus r-one)
qed

lemma *P-ring-uminus-closed*:
assumes $a \in$ *indexed-pset I* (*carrier R*)
shows *P-ring-uminus R a* $\in$ *indexed-pset I* (*carrier R*)
using *assms P-ring-uminus-eq*
by (*metis add.l-inv-ex indexed-const-P-mult-closed minus-equality one-closed*)

lemma *P-ring-uminus-add*:
assumes $a \in$ *indexed-pset I* (*carrier R*)
shows *P-ring-uminus R a* $\bigoplus$ a = *indexed-const* **0**
proof
fix x
show (*P-ring-uminus R a* $\bigoplus$ a) x = *indexed-const* **0** x
using *assms*
unfolding *P-ring-uminus-def indexed-const-def indexed-padd-def*
by (*meson carrier-coeffE indexed-pset-in-carrier*
local.ring-axioms ring.ring-simprules(*9*) *set-eq-subset*)
qed

multiplication by 1

lemma *one-mult-left*:
assumes $a \in$ *indexed-pset I* (*carrier R*)
shows (*indexed-const* **1**) $\bigotimes_p$ a = a
proof
fix m
show (*indexed-const* **1** $\bigotimes_p$ a) m = a m
unfolding *indexed-const-def P-ring-mult-def*
proof−
have *0*: ($\bigoplus$ x∈((*mset-factors m*) − {{#}}). (*if* x = {#} *then* **1** *else* **0**) ⊗ a (m − x)) = **0**
proof−
have $\bigwedge x.$ x∈((*mset-factors m*) − {{#}}) ⟹ (*if* x = {#} *then* **1** *else* **0**) ⊗ a (m − x) = **0** ⊗ a (m − x)
by *simp*
then have $\bigwedge x.$ x∈((*mset-factors m*) − {{#}}) ⟹ (*if* x = {#} *then* **1** *else* **0**) ⊗ a (m − x) = **0**
using *assms*
by (*metis carrier-coeffE indexed-pset-in-carrier local.ring-axioms ring.ring-simprules*(*24*) *set-eq-subset*)
then show *?thesis*
by (*meson add.finprod-one-eqI*)
qed
have *1*: (λx. (*if* x = {#} *then* **1** *else* **0**) ⊗ a (m − x)) {#} = a m
by (*metis* (*full-types*) *assms carrier-coeffE diff-zero*
indexed-pset-in-carrier local.ring-axioms ring.ring-simprules(*12*) *set-eq-subset*)
have *2*: ($\bigoplus$ x∈*insert* {#} (*mset-factors m* − {{#}}). (*if* x = {#} *then* **1** *else* **0**) ⊗ a (m − x)) =

```
          (λx. (if x = {#} then 1 else 0) ⊗ a (m − x)) {#} ⊕
          (⨁ x∈((mset-factors m) − {{#}}). (if x = {#} then 1 else 0) ⊗ a (m
− x))
    proof−
      have 20:finite (mset-factors m − {{#}})
        by simp
      have 21: {#} ∉ mset-factors m − {{#}}
        by blast
      have 22: (λx. (if x = {#} then 1 else 0) ⊗ a (m − x)) ∈ mset-factors m −
{{#}} → carrier R
      proof
        fix x
        assume A: x ∈ mset-factors m − {{#}}
        show (if x = {#} then 1 else 0) ⊗ a (m − x) ∈ carrier R
          apply(cases x = {#})
          using A
          apply blast
              using assms carrier-coeffE indexed-pset-in-carrier  local.ring-axioms
              one-closed ring.ring-simprules(5) set-eq-subset zero-closed
              by (metis (mono-tags, opaque-lifting))
      qed
      have 23: (if {#} = {#} then 1 else 0) ⊗ a (m − {#}) ∈ carrier R
        by (metis 1 assms carrier-coeffE indexed-pset-in-carrier set-eq-subset)
      show ?thesis
        using 20 21 22 23 finsum-insert[of  ((mset-factors m) − {{#}}) {#}
            (λx. (if x = {#} then 1 else 0) ⊗ a (m − x))]
        by blast
    qed
    have 3: insert {#} (mset-factors m − {{#}}) = mset-factors m
    proof−
      have {#} ∈ mset-factors m
        using monom-divides-factors subset-mset.zero-le
        by blast
      then show ?thesis
        by blast
    qed
    show (⨁ x∈mset-factors m. (if x = {#} then 1 else 0) ⊗ a (m − x)) = a m
      using 0 1 2 3 assms
      by (metis (no-types, lifting) Idl-subset-ideal' carrier-coeffE
          genideal-one indexed-pset-in-carrier one-closed r-zero)
  qed
qed

end
```

2.7 The Carrier of the Ring of Indexed Polynomials

```
abbreviation(input) Pring-set where
Pring-set R I ≡ ring.indexed-pset R I (carrier R)
```

context *ring*

begin

lemma *Pring-set-zero*:
assumes $f \in$ *Pring-set R I*
assumes $\neg$ *set-mset m* $\subseteq$ *I*
shows *f m* $= \mathbf{0}_R$
proof−
have $\neg$ *set-mset m* $\subseteq$ *I* $\Longrightarrow$ *f m* $= \mathbf{0}_R$
apply(*induction m*)
apply *simp*
by (*meson assms complement-of-monomials-of mset-to-IP-indices'*)
then show *?thesis*
using *assms*(*2*) **by** *blast*
qed

lemma(**in** *ring*) *Pring-cfs-closed*:
assumes $P \in$ *Pring-set R I*
shows *P m* $\in$ *carrier R*
using *assms carrier-coeffE indexed-pset-in-carrier*
by *blast*

lemma *indexed-pset-mono-aux*:
assumes $P \in$ *indexed-pset I S*
shows $S \subseteq T \Longrightarrow P \in$ *indexed-pset I T*
using *assms*
apply(*induction P*)
using *indexed-pset.indexed-const* **apply** *blast*
using *indexed-pset.indexed-padd* **apply** *blast*
by (*simp add*: *indexed-pset.indexed-pmult*)

lemma *indexed-pset-mono*:
assumes $S \subseteq T$
shows *indexed-pset I S* $\subseteq$ *indexed-pset I T*
using *assms indexed-pset-mono-aux*
by *blast*

end

2.8 Scalar Multiplication

definition *poly-scalar-mult* :: ('*a*, '*b*) *ring-scheme* $\Rightarrow$ '*a* $\Rightarrow$ ('*a*,'*c*) *mvar-poly* $\Rightarrow$ ('*a*,'*c*) *mvar-poly* **where**
poly-scalar-mult R c P = (λ *m*. *c* $\otimes_R$ *P m*)

lemma(**in** *cring*) *poly-scalar-mult-eq*:
assumes *c* $\in$ *carrier R*

shows $P \in$ *Pring-set R* (*I* :: *'c set*) $\Longrightarrow$ *poly-scalar-mult R c P* = *indexed-const c* $\bigotimes_p$ *P*
proof(*erule indexed-pset.induct*)
show $\bigwedge k.\ k \in$ *carrier R* $\Longrightarrow$ *poly-scalar-mult R c* (*indexed-const k*) = *indexed-const c* $\bigotimes_p$ *indexed-const k*
proof−
fix *k*
assume *A0*: *k* ∈ *carrier R*
show *poly-scalar-mult R c* (*indexed-const k*) = (*indexed-const c* $\bigotimes_p$ *indexed-const k*)
unfolding *poly-scalar-mult-def*
proof
show $\bigwedge m.$ *c* ⊗ *indexed-const k m* = (*indexed-const c* $\bigotimes_p$ *indexed-const k*) *m*
using *indexed-const-P-mult-eq*
by (*metis A0 assms indexed-const-P-mult-eq indexed-const-def local.semiring-axioms semiring.r-null*)
qed
qed
show $\bigwedge P\ Q.\ P \in$ *Pring-set R I* $\Longrightarrow$
poly-scalar-mult R c P = *indexed-const c* $\bigotimes_p$ *P* $\Longrightarrow$
Q ∈ *Pring-set R I* $\Longrightarrow$ *poly-scalar-mult R c Q* = *indexed-const c* $\bigotimes_p$ *Q* $\Longrightarrow$ *poly-scalar-mult R c* (*P* $\bigoplus$ *Q*) = *indexed-const c* $\bigotimes_p$ (*P* $\bigoplus$ *Q*)
proof
fix *P Q* :: *'c monomial* ⇒ *'a*
fix *x* :: *'c monomial*
assume *A0*: *P* ∈ *Pring-set R I*
assume *A1*: *poly-scalar-mult R c P* = *indexed-const c* $\bigotimes_p$ *P*
assume *A2*: *Q* ∈ *Pring-set R I*
assume *A3*: *poly-scalar-mult R c Q* = *indexed-const c* $\bigotimes_p$ *Q*
show *poly-scalar-mult R c* (*P* $\bigoplus$ *Q*) *x* = (*indexed-const c* $\bigotimes_p$ (*P* $\bigoplus$ *Q*)) *x*
proof−
have *P0*: *poly-scalar-mult R c* (*P* $\bigoplus$ *Q*) = (*poly-scalar-mult R c P*) $\bigoplus$ (*poly-scalar-mult R c Q*)
unfolding *poly-scalar-mult-def*
proof
fix *m*
show *c* ⊗ (*P* $\bigoplus$ *Q*) *m* = ((λ*m*. *c* ⊗ *P m*) $\bigoplus$ (λ*m*. *c* ⊗ *Q m*)) *m*
proof−
have *LHS*: *c* ⊗ (*P* $\bigoplus$ *Q*) *m* = *c* ⊗ (*P m* ⊕ *Q m*)
by (*simp add*: *indexed-padd-def*)
then have *LHS1*: *c* ⊗ (*P* $\bigoplus$ *Q*) *m* = (*c* ⊗ *P m*) ⊕ (*c* ⊗*Q m*)
proof−
have *0*: *carrier-coeff P*
using *A0 indexed-pset-in-carrier*
by *blast*
have *1*: *P m* ∈ *carrier R*
using *0 carrier-coeffE* **by** *blast*
have *2*: *carrier-coeff Q*
using *A2 indexed-pset-in-carrier*

```
        by blast
      have 3: Q m ∈ carrier R using 2
        using carrier-coeff-def
        by blast
      show ?thesis using 1 3 assms
        by (simp add: LHS r-distr)
    qed
    then show ?thesis
      by (simp add: indexed-padd-def)
  qed
 qed
  have P1: poly-scalar-mult R c (P ⊕ Q) = (indexed-const c ⊗_p P) ⊕
(indexed-const c ⊗_p Q)
    using P0 A1 A3
    by simp
  have P2: indexed-const c ∈ Pring-set R I
    using assms indexed-pset.indexed-const by blast
  show ?thesis
    using P2 A0 A2 P1
    by (metis P-ring-rdistr indexed-pset-in-carrier set-eq-subset)
 qed
qed
show ⋀P i. P ∈ Pring-set R I ⟹
          poly-scalar-mult R c P = indexed-const c ⊗_p P ⟹ i ∈ I ⟹
poly-scalar-mult R c (P ⊗ i) = indexed-const c ⊗_p (P ⊗ i)
proof−
 fix P
 fix i
 assume A0: P ∈ Pring-set R I
 assume A1: poly-scalar-mult R c P = indexed-const c ⊗_p P
 assume A2: i ∈ I
 show poly-scalar-mult R c (P ⊗ i) = indexed-const c ⊗_p (P ⊗ i)
 proof
  fix x
  show poly-scalar-mult R c (P ⊗ i) x = (indexed-const c ⊗_p (P ⊗ i)) x
  proof−
   have RHS: (indexed-const c ⊗_p (P ⊗ i)) = (indexed-const c ⊗_p P) ⊗ i
   proof−
    have B0: P ⊗ i = P ⊗_p (mset-to-IP R {#i#})
     by (meson A0 A2 poly-index-mult)
    have B1: (indexed-const c ⊗_p P) ⊗ i = (indexed-const c ⊗_p P) ⊗_p
(mset-to-IP R {#i#})
     by (meson A0 A2 P-ring-mult-closed' assms indexed-pset.simps poly-index-mult)
     have B2: (indexed-const c ⊗_p (P ⊗ i)) = indexed-const c ⊗_p P ⊗_p
(mset-to-IP R {#i#})
    by (metis A0 A2 B1 assms cring.P-ring-mult-comm indexed-const-var-mult
      indexed-pmult-in-carrier indexed-pset.indexed-const indexed-pset-in-carrier
        is-cring set-eq-subset)
    show ?thesis using A0 A1 A2 B2 B1 assms
```

```
      by (simp add: A0)
    then have RHS': (indexed-const c ⊗_p (P ⊗ i)) = (poly-scalar-mult R c
P) ⊗ i
      using A0 A1 A2 assms
      by simp
    qed
    show ?thesis apply(cases i ∈# x)
      unfolding poly-scalar-mult-def
       apply (metis A1 RHS poly-scalar-mult-def indexed-pmult-def )
      by (metis RHS assms indexed-pmult-def r-null)
  qed
 qed
qed

lemma(in cring) poly-scalar-mult-const:
 assumes c ∈ carrier R
 assumes k ∈ carrier R
 shows poly-scalar-mult R k (indexed-const c) = indexed-const (k ⊗ c)
 using assms poly-scalar-mult-eq
 by (simp add: poly-scalar-mult-eq indexed-const-P-mult-eq indexed-pset.indexed-const)

lemma(in cring) poly-scalar-mult-closed:
 assumes c ∈ carrier R
 assumes P ∈ Pring-set R I
 shows poly-scalar-mult R c P ∈ Pring-set R I
 using assms poly-scalar-mult-eq
 by (metis P-ring-mult-closed' indexed-pset.indexed-const)

lemma(in cring) poly-scalar-mult-zero:
 assumes P ∈ Pring-set R I
 shows poly-scalar-mult R 0 P = indexed-const 0
proof
 fix x
 show poly-scalar-mult R 0 P x = indexed-const 0 x
   unfolding poly-scalar-mult-def
   using assms
   by (metis Pring-cfs-closed indexed-zero-def l-null)
qed

lemma(in cring) poly-scalar-mult-one:
 assumes P ∈ Pring-set R I
 shows poly-scalar-mult R 1 P = P
proof
 fix x show poly-scalar-mult R 1 P x = P x
   using assms
   by (metis one-closed one-mult-left poly-scalar-mult-eq)
qed
```

lemma(**in** *cring*) *times-poly-scalar-mult*:
assumes $P \in$ *Pring-set R I*
assumes $Q \in$ *Pring-set R I*
assumes $k \in$ *carrier R*
shows $P \bigotimes_p$ (*poly-scalar-mult R k Q*) = *poly-scalar-mult R k* ($P \bigotimes_p Q$)
proof−
have $P \bigotimes_p$ (*poly-scalar-mult R k Q*) = $P \bigotimes_p$ (*indexed-const k*) $\bigotimes_p Q$
by (*metis assms*(*1*) *assms*(*2*) *assms*(*3*) *indexed-pset-mono-aux*
local.ring-axioms poly-scalar-mult-eq ring.P-ring-mult-assoc ring.indexed-pset.intros(*1*)
ring.indexed-pset-in-carrier subset-refl)
then have $P \bigotimes_p$ (*poly-scalar-mult R k Q*) = (*indexed-const k*) $\bigotimes_p P \bigotimes_p Q$
by (*metis P-ring-mult-comm assms*(*1*) *assms*(*3*) *local.ring-axioms ring.indexed-pset.indexed-const ring.indexed-pset-in-carrier subset-refl*)
then show *?thesis*
by (*metis* (*no-types*, *opaque-lifting*) *P-ring-mult-closed*′ *Pring-cfs-closed assms*(*1*) *assms*(*2*) *assms*(*3*)
carrier-coeff-def indexed-pset.indexed-const local.ring-axioms poly-scalar-mult-eq ring.P-ring-mult-assoc)
qed

lemma(**in** *cring*) *poly-scalar-mult-times*:
assumes $P \in$ *Pring-set R I*
assumes $Q \in$ *Pring-set R I*
assumes $k \in$ *carrier R*
shows *poly-scalar-mult R k* ($Q \bigotimes_p P$) = (*poly-scalar-mult R k Q*) $\bigotimes_p P$
using *assms times-poly-scalar-mult*
by (*metis* (*no-types*, *opaque-lifting*) *P-ring-mult-comm cring.P-ring-mult-comm*
cring.poly-scalar-mult-closed is-cring local.ring-axioms ring.indexed-pset-in-carrier subset-refl)

2.9 Defining the Ring of Indexed Polynomials

definition *Pring* :: ('*b*, '*e*) *ring-scheme* ⇒ '*a set* ⇒ ('*b*, ('*b*,'*a*) *mvar-poly*) *module*
where

Pring R I ≡ (| *carrier* = *Pring-set R I*,
Group.monoid.mult = *P-ring-mult R* ,
one = *ring.indexed-const R* $\mathbf{1}_R$,
zero = *ring.indexed-const R* $\mathbf{0}_R$,
add = *ring.indexed-padd R*,
smult = *poly-scalar-mult R*|)

context *ring*

begin

lemma *Pring-car*:
carrier (*Pring R I*) = *Pring-set R I*
unfolding *Pring-def*

by *auto*

Definitions of the operations and constants:

lemma *Pring-mult*:
$a \otimes_{Pring\ R\ I} b = a \bigotimes_p b$
unfolding *Pring-def*
by *simp*

lemma *Pring-add*:
$a \oplus_{Pring\ R\ I} b = a \bigoplus b$
unfolding *Pring-def*
by *simp*

lemma *Pring-zero*:
$\mathbf{0}_{Pring\ R\ I} = indexed\text{-}const\ \mathbf{0}$
unfolding *Pring-def* **by** *simp*

lemma *Pring-one*:
$\mathbf{1}_{Pring\ R\ I} = indexed\text{-}const\ \mathbf{1}$
unfolding *Pring-def* **by** *simp*

lemma *Pring-smult*:
$(\odot_{Pring\ R\ I}) = (poly\text{-}scalar\text{-}mult\ R)$
unfolding *Pring-def* **by** *simp*

lemma *Pring-carrier-coeff*:
assumes $a \in carrier\ (Pring\ R\ I)$
shows *carrier-coeff a*
using *assms indexed-pset-in-carrier*[*of* (*carrier R*) *a I*] *Pring-car*
by *blast*

lemma *Pring-carrier-coeff'*[*simp*]:
assumes $a \in carrier\ (Pring\ R\ I)$
shows $a\ m \in carrier\ R$
using *Pring-car*[*of I*] *assms carrier-coeffE indexed-pset-in-carrier*
by *blast*

lemma *Pring-add-closed*:
assumes $a \in carrier\ (Pring\ R\ I)$
assumes $b \in carrier\ (Pring\ R\ I)$
shows $a \oplus_{Pring\ R\ I} b \in carrier\ (Pring\ R\ I)$
using *assms Pring-def*[*of R I*]
by (*simp add*: *Pring-def*[*of R I*] *indexed-pset.indexed-padd*)

lemma *Pring-mult-closed*:
assumes $a \in carrier\ (Pring\ R\ I)$
assumes $b \in carrier\ (Pring\ R\ I)$
shows $a \otimes_{Pring\ R\ I} b \in carrier\ (Pring\ R\ I)$
using *assms P-ring-mult-closed'*[*of a I b*] *Pring-car*[*of I*] *Pring-mult*[*of I a b*]

by (*simp add*: ‹$a \otimes_{Pring\ R\ I} b = a \bigotimes_p b$› ‹*carrier* (*Pring R I*) = *Pring-set R I*›)

lemma *Pring-one-closed*:
$\mathbf{1}_{Pring\ R\ I} \in carrier\ (Pring\ R\ I)$
proof−
have *indexed-const* $\mathbf{1} \in carrier\ (Pring\ R\ I)$
using *Pring-car indexed-pset.simps* **by** *blast*
then show *?thesis*
unfolding *Pring-def* **by** *auto*
qed

lemma *Pring-zero-closed*:
$\mathbf{0}_{Pring\ R\ I} \in carrier\ (Pring\ R\ I)$
proof−
have *indexed-const* $\mathbf{0} \in carrier\ (Pring\ R\ I)$
using *Pring-car indexed-pset.simps* **by** *blast*
then show *?thesis*
unfolding *Pring-def* **by** *auto*
qed

lemma *Pring-var-closed*:
assumes $i \in I$
shows *var-to-IP R i* $\in$ *carrier* (*Pring R I*)
unfolding *var-to-IP-def*
by (*metis Pring-car Pring-one-closed assms indexed-pset.indexed-pmult local.ring-axioms monom-add-mset one-mset-to-IP ring.Pring-one*)

Properties of addition:

lemma *Pring-add-assoc*:
assumes $a \in carrier\ (Pring\ R\ I)$
assumes $b \in carrier\ (Pring\ R\ I)$
assumes $c \in carrier\ (Pring\ R\ I)$
shows $a \oplus_{Pring\ R\ I} (b \oplus_{Pring\ R\ I} c) = (a \oplus_{Pring\ R\ I} b) \oplus_{Pring\ R\ I} c$
proof−
have $a \bigoplus (b \bigoplus c) = (a \bigoplus b) \bigoplus c$
proof
fix *x*
have *carrier-coeff a carrier-coeff b carrier-coeff c*
using *assms Pring-car*[*of I*] *Pring-carrier-coeff* **apply** *blast*
using *assms Pring-car*[*of I*] *Pring-carrier-coeff* **apply** *blast*
using *assms Pring-car*[*of I*] *Pring-carrier-coeff* **by** *blast*
then have *a x* $\in$ *carrier R b x* $\in$ *carrier R c x* $\in$ *carrier R*
using *carrier-coeffE* **apply** *blast*
using ‹*carrier-coeff b*› *carrier-coeffE* **apply** *blast*
using ‹*carrier-coeff c*› *carrier-coeffE* **by** *blast*
show $(a \bigoplus (b \bigoplus c))\ x = (a \bigoplus b \bigoplus c)\ x$
unfolding *indexed-padd-def*
using *assms*

by (*simp add*: ‹*a x* ∈ *carrier R*› ‹*b x* ∈ *carrier R*› ‹*c x* ∈ *carrier R*› *add.m-assoc*)
qed
then show *?thesis*
using *assms*
by (*simp add*: *Pring-add*)
qed

lemma *Pring-add-comm*:
assumes *a* ∈ *carrier* (*Pring R I*)
assumes *b* ∈ *carrier* (*Pring R I*)
shows *a* $\oplus_{Pring\ R\ I}$ *b* = *b* $\oplus_{Pring\ R\ I}$ *a*
proof−
have *a* ⨁ *b* = *b* ⨁ *a*
proof fix *x*
show (*a* ⨁ *b*) *x* = (*b* ⨁ *a*) *x*
using *assms*
by (*metis abelian-monoid.a-comm abelian-monoid-axioms*
indexed-padd-def local.ring-axioms ring.Pring-carrier-coeff')
qed
then show *?thesis*
by (*simp add*: *Pring-add*)
qed

lemma *Pring-add-zero*:
assumes *a* ∈ *carrier* (*Pring R I*)
shows *a* $\oplus_{Pring\ R\ I}$ $\mathbf{0}_{Pring\ R\ I}$ = *a*
$\mathbf{0}_{Pring\ R\ I}$ $\oplus_{Pring\ R\ I}$ *a* = *a*
using *assms Pring-zero Pring-add*
apply (*metis Pring-carrier-coeff indexed-padd-zero*(*1*))
using *assms Pring-zero Pring-add*
by (*metis Pring-carrier-coeff indexed-padd-zero*(*2*))

Properties of multiplication

lemma *Pring-mult-assoc*:
assumes *a* ∈ *carrier* (*Pring R I*)
assumes *b* ∈ *carrier* (*Pring R I*)
assumes *c* ∈ *carrier* (*Pring R I*)
shows *a* $\otimes_{Pring\ R\ I}$ (*b* $\otimes_{Pring\ R\ I}$ *c*) = (*a* $\otimes_{Pring\ R\ I}$ *b*) $\otimes_{Pring\ R\ I}$ *c*
using *assms P-ring-mult-assoc*[*of a b c*]
by (*metis Pring-carrier-coeff Pring-mult*)

lemma *Pring-mult-comm*:
assumes *cring R*
assumes *a* ∈ *carrier* (*Pring R I*)
assumes *b* ∈ *carrier* (*Pring R I*)
shows *a* $\otimes_{Pring\ R\ I}$ *b* = *b* $\otimes_{Pring\ R\ I}$ *a*
using *assms Pring-carrier-coeff*[*of a I*] *Pring-carrier-coeff*[*of b I*]
Pring-mult[*of I b a*] *Pring-mult*[*of I a b*] *cring.P-ring-mult-comm*[*of R a b*]
by *metis*

lemma *Pring-mult-one*:
assumes $a \in carrier\ (Pring\ R\ I)$
shows $a \otimes_{Pring\ R\ I} \mathbf{1}_{Pring\ R\ I} = a$
proof
fix x
show $(a \otimes_{Pring\ R\ I} \mathbf{1}_{Pring\ R\ I})\ x = a\ x$
proof−
have *0*: $(a \otimes_{Pring\ R\ I} \mathbf{1}_{Pring\ R\ I})\ x = (a \bigotimes_p indexed\text{-}const\ \mathbf{1})\ x$
using *assms Pring-mult*[*of I a* $\mathbf{1}_{Pring\ R\ I}$] *Pring-one*[*of I*]
by *metis*
have *1*: $\mathbf{1} \in carrier\ R$
by *simp*
have *2*: $(a \otimes_{Pring\ R\ I} \mathbf{1}_{Pring\ R\ I})\ x = a\ x \otimes \mathbf{1}$
using *0 1 indexed-const-P-multE*[*of a I* $\mathbf{1}$ *x*]
assms Pring-car[*of I*]
by *metis*
then show *?thesis*
using *assms* **by** *auto*
qed
qed

lemma *Pring-mult-one′*:
assumes $a \in carrier\ (Pring\ R\ I)$
shows $\mathbf{1}_{Pring\ R\ I} \otimes_{Pring\ R\ I} a = a$
using *one-mult-left*[*of a I*]
assms Pring-one Pring-mult
by (*simp add*: *Pring-mult Pring-one Pring-car*)

Distributive laws

lemma *Pring-mult-rdistr*:
assumes $a \in carrier\ (Pring\ R\ I)$
assumes $b \in carrier\ (Pring\ R\ I)$
assumes $c \in carrier\ (Pring\ R\ I)$
shows $a \otimes_{Pring\ R\ I} (b \oplus_{Pring\ R\ I} c) = (a \otimes_{Pring\ R\ I} b) \oplus_{Pring\ R\ I} (a \otimes_{Pring\ R\ I} c)$
proof−
have $a \bigotimes_p (b \bigoplus c) = a \bigotimes_p b \bigoplus (a \bigotimes_p c)$
using *P-ring-rdistr*[*of a b c*]
assms Pring-carrier-coeff
by *metis*
then have $a \bigotimes_p (b \oplus_{Pring\ R\ I} c) = a \bigotimes_p b \oplus_{Pring\ R\ I} (a \bigotimes_p c)$
using *Pring-add*[*of I b c*] *Pring-add*[*of I a* $\bigotimes_p$ *b a* $\bigotimes_p$ *c*]
by *auto*
then have $a \otimes_{Pring\ R\ I} (b \oplus_{Pring\ R\ I} c) = (a \bigotimes_p b) \oplus_{Pring\ R\ I} (a \bigotimes_p c)$
using *Pring-mult*[*of I a* $(b \oplus_{Pring\ R\ I} c)$]
by *auto*
then have $a \otimes_{Pring\ R\ I} (b \oplus_{Pring\ R\ I} c) = (a \otimes_{Pring\ R\ I} b) \oplus_{Pring\ R\ I} (a \bigotimes_p c)$

using *Pring-mult*[*of I a b*] **by** *metis*
then show *?thesis*
using *Pring-mult*[*of I a c*] **by** *metis*
qed

lemma *Pring-mult-ldistr*:
assumes $a \in carrier\ (Pring\ R\ I)$
assumes $b \in carrier\ (Pring\ R\ I)$
assumes $c \in carrier\ (Pring\ R\ I)$
shows $(b \oplus_{Pring\ R\ I} c) \otimes_{Pring\ R\ I} a = (b \otimes_{Pring\ R\ I} a) \oplus_{Pring\ R\ I} (c \otimes_{Pring\ R\ I} a)$
proof−
have $(b \bigoplus c) \bigotimes_p a = b \bigotimes_p a \bigoplus (c \bigotimes_p a)$
using *P-ring-ldistr*[*of a b c*]
assms Pring-carrier-coeff
by *metis*
then have $(b \oplus_{Pring\ R\ I} c) \bigotimes_p a = b \bigotimes_p a \oplus_{Pring\ R\ I} (c \bigotimes_p a)$
using *Pring-add*[*of I b c*] *Pring-add*[*of I b* $\bigotimes_p$ *a c* $\bigotimes_p$ *a*]
by *auto*
then have $(b \oplus_{Pring\ R\ I} c) \otimes_{Pring\ R\ I} a = (b \bigotimes_p a) \oplus_{Pring\ R\ I} (c \bigotimes_p a)$
using *Pring-mult*[*of I* $(b \oplus_{Pring\ R\ I} c)$ *a*]
by *auto*
then have $(b \oplus_{Pring\ R\ I} c) \otimes_{Pring\ R\ I} a = (b \otimes_{Pring\ R\ I} a) \oplus_{Pring\ R\ I} (c \bigotimes_p a)$
using *Pring-mult*[*of I b a*] **by** *metis*
then show *?thesis*
using *Pring-mult*[*of I c a*] **by** *metis*
qed

Properties of subtraction:

lemma *Pring-uminus*:
assumes $a \in carrier\ (Pring\ R\ I)$
shows *P-ring-uminus R a* $\in carrier\ (Pring\ R\ I)$
using *P-ring-uminus-closed*[*of a I*] *Pring-car*[*of I*] *assms*
by *metis*

lemma *Pring-subtract*:
assumes $a \in carrier\ (Pring\ R\ I)$
shows *P-ring-uminus R a* $\oplus_{Pring\ R\ I}$ *a* $= \mathbf{0}_{Pring\ R\ I}$
a $\oplus_{Pring\ R\ I}$ *P-ring-uminus R a* $= \mathbf{0}_{Pring\ R\ I}$
using *assms Pring-add*[*of I P-ring-uminus R a a*] *Pring-zero*[*of I*]
apply (*simp add*: *Pring-car local.ring-axioms ring.P-ring-uminus-add*)
using *assms Pring-add*[*of I P-ring-uminus R a a*] *Pring-zero*[*of I*]
by (*metis P-ring-uminus-add P-ring-uminus-closed Pring-add-comm Pring-car*)

Pring R I is a ring

lemma *Pring-is-abelian-group*:
shows *abelian-group* (*Pring R I*)
apply(*rule abelian-groupI*)

```
  apply (simp add: Pring-add-closed)
   apply (simp add: local.ring-axioms ring.Pring-zero-closed)
    apply (simp add: Pring-add-assoc)
     apply (simp add: Pring-add-comm)
      apply (simp add: local.ring-axioms ring.Pring-add-zero(2))
  using Pring-subtract(1) Pring-uminus
  by blast

lemma Pring-is-monoid:
Group.monoid (Pring R I)
  apply(rule monoidI)
  using Pring-mult-closed apply blast
   apply (simp add: Pring-one-closed)
    apply (metis Pring-mult-assoc)
     using Pring-mult-one'
     apply blast
  using Pring-mult-one by blast

lemma Pring-is-ring:
  shows ring (Pring R I)
  apply(rule ringI)
  apply (simp add: Pring-is-abelian-group)
   apply (simp add: Pring-is-monoid)
    apply (simp add: Pring-mult-ldistr)
  by (simp add: Pring-mult-rdistr)

lemma Pring-is-cring:
  assumes cring R
  shows cring (Pring R I)
  apply(rule cringI)
  apply (simp add: Pring-is-abelian-group)
   apply (simp add: Pring-is-monoid assms local.ring-axioms
         monoid.monoid-comm-monoidI ring.Pring-mult-comm)
    by (simp add: Pring-mult-ldistr)

lemma Pring-a-inv:
  assumes P ∈ carrier (Pring R I)
  shows ⊖Pring R I P = P-ring-uminus R P
proof−
  have 0: P-ring-uminus R P ∈ carrier (Pring R I)
   using Pring-uminus assms
   by blast
  have 1: P-ring-uminus R P ⊕Pring R I P = 0Pring R I
   using Pring-subtract(1) assms
   by blast
  show ?thesis using 0 1 assms Pring-is-ring
  by (simp add: Pring-car Pring-is-abelian-group abelian-group.minus-equality)
qed
```

end

2.10 Defining the R-Algebra of Indexed Polynomials

context *cring*
begin

lemma *Pring-smult-cfs*:
assumes $a \in carrier\ R$
assumes $P \in carrier\ (Pring\ R\ I)$
shows $(a \odot_{Pring\ R\ I} P)\ m = a \otimes (P\ m)$
using *assms Pring-smult*
by (*simp add*: *Pring-smult poly-scalar-mult-def*)

lemma *Pring-smult-closed*:
$\bigwedge a\ x.$ [| $a \in carrier\ R$; $x \in carrier\ (Pring\ R\ I)$ |] ==> $a \odot_{(Pring\ R\ I)} x \in carrier\ (Pring\ R\ I)$
by (*simp add*: *Pring-car Pring-smult poly-scalar-mult-closed*)

lemma *Pring-smult-l-distr*:
!!$a\ b\ x.$ [| $a \in carrier\ R$; $b \in carrier\ R$; $x \in carrier\ (Pring\ R\ I)$ |] ==>
$(a \oplus b) \odot_{(Pring\ R\ I)} x = (a \odot_{(Pring\ R\ I)} x) \oplus_{(Pring\ R\ I)} (b \odot_{(Pring\ R\ I)} x)$
proof− **fix** $a\ b\ x$ **assume** A: $a \in carrier\ R\ b \in carrier\ R\ x \in carrier\ (Pring\ R\ I)$
show $(a \oplus b) \odot_{Pring\ R\ I} x = a \odot_{Pring\ R\ I} x \oplus_{Pring\ R\ I} b \odot_{Pring\ R\ I} x$
proof fix m
have $(a \oplus b) \otimes x\ m = (a \otimes (x\ m)) \oplus (b \otimes (x\ m))$
by (*meson A(1) A(2) A(3) Pring-carrier-coeff' local.semiring-axioms semiring.l-distr*)
thus $((a \oplus b) \odot_{Pring\ R\ I} x)\ m = (a \odot_{Pring\ R\ I} x \oplus_{Pring\ R\ I} b \odot_{Pring\ R\ I} x)\ m$
using *Pring-smult-cfs*[*of* $a \oplus b\ x\ I\ m$]
Pring-smult-cfs[*of* $a\ x\ I\ m$]
Pring-smult-cfs[*of* $b\ x\ I\ m$]
Pring-smult-closed[*of* $a\ x\ I$]
Pring-smult-closed[*of* $b\ x\ I$]
Pring-add A
by (*simp add*: ‹$(a \oplus b) \otimes x\ m = a \otimes x\ m \oplus b \otimes x\ m$› *Pring-def indexed-padd-def poly-scalar-mult-def*)
qed
qed

lemma *Pring-smult-r-distr*:
!!$a\ x\ y.$ [| $a \in carrier\ R$; $x \in carrier\ (Pring\ R\ I)$; $y \in carrier\ (Pring\ R\ I)$ |] ==>
$a \odot_{(Pring\ R\ I)} (x \oplus_{(Pring\ R\ I)} y) = (a \odot_{(Pring\ R\ I)} x) \oplus_{(Pring\ R\ I)} (a \odot_{(Pring\ R\ I)} y)$
proof fix $a\ x\ y\ m$ **assume** A: $a \in carrier\ R\ x \in carrier\ (Pring\ R\ I)\ y \in carrier\ (Pring\ R\ I)$

show $(a \odot_{Pring\ R\ I} (x \oplus_{Pring\ R\ I} y))\ m =$
$(a \odot_{Pring\ R\ I} x \oplus_{Pring\ R\ I} a \odot_{Pring\ R\ I} y)\ m$
using *Pring-smult-cfs*[*of a x* $\oplus_{Pring\ R\ I}$ *y I m*]
Pring-smult-cfs[*of a x I m*]
Pring-smult-cfs[*of a y I m*]
Pring-smult-closed[*of a x I*]
Pring-smult-closed[*of a y I*]
Pring-add A
by (*metis* (*no-types*, *lifting*) *Pring-add-closed Pring-carrier-coeff' indexed-padd-def l-distr m-comm*)
qed

lemma *Pring-smult-assoc1*:
!!*a b x*. [| *a* ∈ *carrier R*; *b* ∈ *carrier R*; *x* ∈ *carrier* (*Pring R I*) |] ==>
$(a \otimes b) \odot_{Pring\ R\ I} x = a \odot_{Pring\ R\ I} (b \odot_{Pring\ R\ I} x)$
proof fix *a b x m* **assume** *A*: *a* ∈ *carrier R b* ∈ *carrier R x* ∈ *carrier* (*Pring R I*)
show $(a \otimes b \odot_{Pring\ R\ I} x)\ m = (a \odot_{Pring\ R\ I} (b \odot_{Pring\ R\ I} x))\ m$
using *Pring-smult-cfs*[*of a* ⊗ *b x I m*]
Pring-smult-cfs[*of a b* $\odot_{Pring\ R\ I}$ *x I m*]
Pring-smult-cfs[*of b x I m*]
Pring-smult-closed[*of a b* $\odot_{Pring\ R\ I}$ *x I*]
Pring-smult-closed[*of b x I*]
A(*1*) *A*(*2*) *A*(*3*) *m-assoc m-closed* **by** *auto*
qed

lemma *Pring-smult-one*:
!!*x*. *x* ∈ *carrier* (*Pring R I*) ==> $(one\ R) \odot_{Pring\ R\ I} x = x$
by (*simp add*: *Pring-car Pring-smult poly-scalar-mult-one*)

lemma *Pring-smult-assoc2*:
!!*a x y*. [| *a* ∈ *carrier R*; *x* ∈ *carrier* (*Pring R I*); *y* ∈ *carrier* (*Pring R I*) |]
==>
$(a \odot_{Pring\ R\ I} x) \otimes_{Pring\ R\ I} y = a \odot_{Pring\ R\ I} (x \otimes_{Pring\ R\ I} y)$
by (*simp add*: *Pring-def poly-scalar-mult-times*)

lemma *Pring-algebra*:
algebra R (*Pring R I*)
apply(*rule algebraI*)
apply (*simp add*: *is-cring*)
apply (*simp add*: *Pring-is-cring is-cring*)
apply (*simp add*: *Pring-smult-closed*)
apply (*simp add*: *Pring-smult-l-distr*)
apply (*simp add*: *Pring-smult-r-distr*)
apply (*simp add*: *Pring-smult-assoc1*)
apply (*simp add*: *Pring-smult-one*)
by (*simp add*: *Pring-smult-assoc2*)

end

2.11 Evaluation of Polynomials and Subring Structure

In this section the aim is to define the evaluation of a polynomial over its base ring. We define both total evaluation of a polynomial at all variables, and partial evaluation at only a subset of variables. The basic input for evaluation is a variable assignment function mapping variables to ring elements. Once we can evaluate a polynomial P in variables I over a ring R at an assignment $f : I \to R$, this can be generalized to evaluation of P in some other ring S, given a variable assignment $f : I \to S$ and a ring homomorphism $\phi : R \to S$. We chose to define this by simply applying ϕ to the coefficients of P, and then using the first evaluation function over S. This could also have been done the other way around: define general polynomial evaluation over any ring, given a ring hom ϕ, and then defining evaluation over the base ring R as the specialization of this function to the case there $\phi = id_R$.

definition *remove-monom* ::
(*'a*,*'b*) *ring-scheme* ⇒ *'c monomial* ⇒ (*'a*, *'c*) *mvar-poly* ⇒ (*'a*, *'c*) *mvar-poly*
where
remove-monom R m P = *ring.indexed-padd R P* (*poly-scalar-mult R* (⊖$_R$ *P m*) (*mset-to-IP R m*))

context *cring*
begin

lemma *remove-monom-alt-def*:
assumes *P* ∈ *Pring-set R I*
shows *remove-monom R m P n* = (*if n* = *m then* **0** *else P n*)
unfolding *remove-monom-def*
apply(*cases n* = *m*)
using *assms*
apply (*metis Pring-cfs-closed cring.cring-simprules*(*3*) *poly-scalar-mult-def indexed-padd-def is-cring mset-to-IP-simp r-neg r-one*)
using *assms*
by (*metis Pring-cfs-closed add.l-cancel-one cring.cring-simprules*(*3*) *poly-scalar-mult-def indexed-padd-def is-cring mset-to-IP-simp' r-null zero-closed*)

lemma *remove-monom-zero*:
assumes *m* ∉ *monomials-of R P*
assumes *P* ∈ *Pring-set R I*
shows *remove-monom R m P* = *P*
proof
fix *x*
show *remove-monom R m P x* = *P x*

apply(*cases* $x \in$ *monomials-of* R P)
using *assms* **unfolding** *monomials-of-def remove-monom-def*
apply (*metis cring.remove-monom-alt-def is-cring remove-monom-def*)
using *assms* **unfolding** *monomials-of-def remove-monom-def*
by (*metis assms*(*1*) *cring.remove-monom-alt-def is-cring local.ring-axioms remove-monom-def ring.complement-of-monomials-of*)
qed

lemma *remove-monom-closed*:
assumes $P \in$ *Pring-set* R I
shows *remove-monom* R m $P \in$ *Pring-set* R I

apply(*cases* $m \in$ *monomials-of* R P)
using *assms poly-scalar-mult-closed*[*of* ($\ominus$ P m) (*mset-to-IP* R m) I] *mset-to-IP-closed*[*of* m I]
unfolding *remove-monom-def*
apply (*meson Pring-cfs-closed add.inv-closed indexed-pset.indexed-padd mset-to-IP-closed'*)
by (*metis assms remove-monom-def remove-monom-zero*)

lemma *remove-monom-monomials*:
assumes $P \in$ *Pring-set* R I
shows *monomials-of* R (*remove-monom* R m P) = *monomials-of* R $P - \{m\}$
proof
show *monomials-of* R (*remove-monom* R m P) $\subseteq$ *monomials-of* R $P - \{m\}$
using *assms remove-monom-alt-def*[*of* P I m]
unfolding *monomials-of-def*
by *auto*
show *monomials-of* R $P - \{m\} \subseteq$ *monomials-of* R (*remove-monom* R m P)
using *assms remove-monom-alt-def*[*of* P I m]
unfolding *monomials-of-def*
by *auto*
qed

The additive decomposition of a polynomial by a monomial

lemma *remove-monom-eq*:
assumes $P \in$ *Pring-set* R I
shows P = (*remove-monom* R a P) $\bigoplus$ *poly-scalar-mult* R (P a) (*mset-to-IP* R a)
unfolding *remove-monom-def poly-scalar-mult-def*
proof
fix x
show $P\ x$ = ($P \bigoplus$ ($\lambda m.$ $\ominus$ P $a \otimes$ *mset-to-IP* R a m) $\bigoplus$ ($\lambda m.$ P $a \otimes$ *mset-to-IP* R a m)) x
apply(*cases* $x = a$)
apply (*metis Pring-cfs-closed assms l-minus l-zero local.ring-axioms mset-to-IP-simp one-closed r-neg r-one ring.indexed-padd-def*)
proof−
assume A: $x \neq a$
show $P\ x$ = ($P \bigoplus$ ($\lambda m.$ $\ominus$ P $a \otimes$ *mset-to-IP* R a m) $\bigoplus$ ($\lambda m.$ P $a \otimes$ *mset-to-IP*

R a m)) *x*
using *assms A*
unfolding *mset-to-IP-def indexed-padd-def*
using *Pring-cfs-closed* **by** *fastforce*
qed
qed

lemma *remove-monom-restrict-poly-to-monom-set*:
assumes $P \in$ *Pring-set R I*
assumes *monomials-of R P* = *insert a M*
assumes $a \notin M$
shows (*remove-monom R a P*) = *restrict-poly-to-monom-set R P M*
proof
fix *m*
show *remove-monom R a P m*= *restrict-poly-to-monom-set R P M m*
apply(*cases m* = *a*)
using *assms* **apply** (*metis remove-monom-alt-def restrict-poly-to-monom-set-def*)
using *assms*
by (*metis complement-of-monomials-of insert-iff remove-monom-alt-def restrict-poly-to-monom-set-def*)
qed

end

2.11.1 Nesting of Polynomial Rings According to Nesting of Index Sets

lemma(**in** *ring*) *Pring-carrier-subset*:
assumes $J \subseteq I$
shows (*Pring-set R J*) $\subseteq$ (*Pring-set R I*)
proof
fix *P*
show $P \in$ *Pring-set R J* $\Longrightarrow P \in$ *Pring-set R I*
apply(*erule indexed-pset.induct*)
using *indexed-pset.indexed-const* **apply** *blast*
using *indexed-pset.indexed-padd* **apply** *blast*
by (*meson assms indexed-pset.indexed-pmult subsetD*)
qed

lemma(**in** *cring*) *Pring-set-restrict-induct*:
shows *finite S* $\Longrightarrow \forall P.$ *monomials-of R P* = *S* $\wedge$ $P \in$ *Pring-set R I* $\wedge$ ($\forall$ *m* $\in$ *monomials-of R P. set-mset m* $\subseteq$ *J*) $\longrightarrow$ $P \in$ *Pring-set R J*
proof(*erule finite.induct*)
show $\forall P.$ *monomials-of R P* = $\{\}$ $\wedge$ $P \in$ *Pring-set R I* $\wedge$ ($\forall$ *m*$\in$*monomials-of R P. set-mset m* $\subseteq$ *J*) $\longrightarrow$ $P \in$ *Pring-set R J*
proof
fix *P*
show *monomials-of R P* = $\{\}$ $\wedge$ $P \in$ *Pring-set R I* $\wedge$ ($\forall$ *m*$\in$*monomials-of R P. set-mset m* $\subseteq$ *J*) $\longrightarrow$ $P \in$ *Pring-set R J*

proof
assume *A0*: *monomials-of R P* = {} ∧ *P* ∈ *Pring-set R I* ∧ (∀ *m*∈*monomials-of R P*. *set-mset m* ⊆ *J*)
show *P* ∈ *Pring-set R J*
by (*metis A0 card-eq-0-iff indexed-pset.indexed-const monomials-of-card-zero zero-closed*)
qed
qed
show ⋀*A a*. *finite A* ⟹ ∀ *P*. *monomials-of R P* = *A* ∧ *P* ∈ *Pring-set R I* ∧ (∀ *m*∈*monomials-of R P*. *set-mset m* ⊆ *J*) ⟶ *P* ∈ *Pring-set R J* ⟹ ∀ *P*. *monomials-of R P* = *insert a A* ∧ *P* ∈ *Pring-set R I* ∧ (∀ *m*∈*monomials-of R P*. *set-mset m* ⊆ *J*) ⟶ *P* ∈ *Pring-set R J*
proof
fix *A* :: ('*c monomial*) *set* **fix** *a* **fix** *P*
assume *A1*: *finite A*
assume *A2*: ∀ *P*. *monomials-of R P* = *A* ∧ *P* ∈ *Pring-set R I* ∧ (∀ *m*∈*monomials-of R P*. *set-mset m* ⊆ *J*) ⟶ *P* ∈ *Pring-set R J*
show *monomials-of R P* = *insert a A* ∧ *P* ∈ *Pring-set R I* ∧ (∀ *m*∈*monomials-of R P*. *set-mset m* ⊆ *J*) ⟶ *P* ∈ *Pring-set R J*
proof
assume *A3*: *monomials-of R P* = *insert a A* ∧ *P* ∈ *Pring-set R I* ∧ (∀ *m*∈*monomials-of R P*. *set-mset m* ⊆ *J*)
show *P* ∈ *Pring-set R J*
apply(*cases a* ∈ *A*)
apply (*metis A2 A3 insert-absorb*)
proof−
assume *N*: *a* ∉ *A*
obtain *Q* **where** *Q-def*: *Q* = *remove-monom R a P*
by *simp*
have *Q0*: *monomials-of R Q* = *A*
proof−
have *0*: *monomials-of R P* = *insert a A*
by (*simp add*: *A3*)
have *1*: *P* ∈ *Pring-set R I*
using *A3* **by** *blast*
have *2*: *monomials-of R* (*remove-monom R a P*) = *monomials-of R P* − {*a*}
using *A3 remove-monom-monomials* **by** *blast*
then show *?thesis*
using *Q-def 0*
by (*simp add*: *N*)
qed
have *Q1*: *Q* ∈ *Pring-set R I*
using *A3 Q-def remove-monom-closed* **by** *blast*
have *Q2*: (∀ *m*∈*monomials-of R Q*. *set-mset m* ⊆ *J*)
using *Q0 A3*
by *blast*
have *Q* ∈ *Pring-set R J*

```
        using A2 Q0 Q1 Q2 by blast
      then show P ∈ Pring-set R J
      proof−
        have P = Q ⊕ poly-scalar-mult R (P a) (mset-to-IP R a)
          using Q-def remove-monom-eq
          using A3 by blast
        then show ?thesis
      by (metis A3 Pring-cfs-closed ‹Q ∈ Pring-set R J› indexed-pset.indexed-padd
insertCI mset-to-IP-closed poly-scalar-mult-closed)
      qed
    qed
  qed
qed

lemma(in cring) Pring-set-restrict:
  assumes P ∈ Pring-set R I
  assumes (⋀m. m ∈ monomials-of R P ⟹ set-mset m ⊆ J)
  shows P ∈ Pring-set R J
  using assms Pring-set-restrict-induct[of monomials-of R P]
  by (metis monomials-finite)

lemma(in ring) Pring-mult-eq:
  fixes I:: 'c set
  fixes J:: 'c set
  shows (⊗Pring R I) = (⊗Pring R J)
  by (simp add: Pring-def)

lemma(in ring) Pring-add-eq:
  fixes I:: 'c set
  fixes J:: 'c set
  shows (⊕Pring R I) = (⊕Pring R J)
  using Pring-def
  by (simp add: Pring-def)

lemma(in ring) Pring-one-eq:
  fixes I:: 'c set
  fixes J:: 'c set
  shows (1Pring R I) = (1Pring R J)
  using Pring-def
  by (simp add: Pring-def)

lemma(in ring) Pring-zero-eq:
  fixes I:: 'c set
  fixes J:: 'c set
  shows (0Pring R I) = (0Pring R J)
  using Pring-def
  by (simp add: Pring-def)
```

lemma(**in** *ring*) *index-subset-Pring-subring*:
assumes $J \subseteq I$
shows *subring* (*carrier* (*Pring R J*)) (*Pring R I*)
apply(*rule ring.subringI*)
apply (*simp add*: *Pring-is-ring*; *fail*)
using *assms*
apply (*simp add*: *Pring-car Pring-carrier-subset*; *fail*)
using *Pring-def*
apply (*simp add*: *Pring-def indexed-pset.indexed-const*; *fail*)
proof−
show $\bigwedge h.\ h \in carrier\ (Pring\ R\ J) \Longrightarrow \ominus_{Pring\ R\ I}\ h \in carrier\ (Pring\ R\ J)$
proof−
fix *h*
assume *A*: $h \in carrier\ (Pring\ R\ J)$
then have *A0*: $\ominus_{Pring\ R\ J}\ h = P\text{-}ring\text{-}uminus\ R\ h$
using *Pring-a-inv*[*of h J*]
by *auto*
have *A1*: $\ominus_{Pring\ R\ I}\ h = P\text{-}ring\text{-}uminus\ R\ h$
using *assms A Pring-carrier-subset*[*of J I*] *Pring-a-inv*[*of h I*] *Pring-car*
by *blast*
show $\ominus_{Pring\ R\ I}\ h \in carrier\ (Pring\ R\ J)$
using *A0 A1 A*
by (*simp add*: *Pring-uminus*)
qed
show $\bigwedge h1\ h2.\ h1 \in carrier\ (Pring\ R\ J) \Longrightarrow h2 \in carrier\ (Pring\ R\ J) \Longrightarrow h1 \otimes_{Pring\ R\ I}\ h2 \in carrier\ (Pring\ R\ J)$
using *assms Pring-mult-eq*
by (*metis Pring-mult-closed*)
show $\bigwedge h1\ h2.\ h1 \in carrier\ (Pring\ R\ J) \Longrightarrow h2 \in carrier\ (Pring\ R\ J) \Longrightarrow h1 \oplus_{Pring\ R\ I}\ h2 \in carrier\ (Pring\ R\ J)$
using *assms Pring-add-eq*
by (*metis Pring-add-closed*)
qed

2.11.2 Inclusion Maps

definition *Pring-inc* :: ('*a*, '*c*) *mvar-poly* ⇒ ('*a*, '*c*) *mvar-poly* **where**
Pring-inc ≡ (λ*P*. *P*)

lemma(**in** *ring*) *Princ-inc-is-ring-hom*:
assumes $J \subseteq I$
shows *ring-hom-ring* (*Pring R J*) (*Pring R I*) *Pring-inc*
unfolding *Pring-inc-def*
apply(*rule ring-hom-ringI*)
apply (*simp add*: *Pring-is-ring*)
using *Pring-is-ring* **apply** *blast*
using *index-subset-Pring-subring*[*of I J*] *assms index-subset-Pring-subring subringE*(*1*)
apply *blast*

using *Pring-mult-eq*[*of I J*]
apply (*simp add*: *Pring-mult*)
using *Pring-add-eq*[*of I J*]
apply (*simp add*: *Pring-add*)
using *Pring-one-eq*
by (*simp add*: *Pring-one-eq*)

2.11.3 Restricting a Multiset to a Subset of Variables

definition *restrict-to-indices* :: *'c monomial* ⇒ *'c set* ⇒ *'c monomial* **where**
restrict-to-indices m S = *filter-mset* (λ*i*. *i* ∈ *S*) *m*

lemma *restrict-to-indicesE*:
assumes *i* ∈# *restrict-to-indices m S*
shows *i* ∈ *S*
using *assms*
unfolding *restrict-to-indices-def*
by *simp*

lemma *restrict-to-indicesI*[*simp*]:
assumes *i* ∈# *m*
assumes *i* ∈ *S*
shows *i* ∈# *restrict-to-indices m S*
using *assms*
unfolding *restrict-to-indices-def*
by *simp*

lemma *restrict-to-indices-not-in*[*simp*]:
assumes *i* ∈# *m*
assumes *i* ∉ *S*
shows *i* ∉# *restrict-to-indices m S*
using *assms*
unfolding *restrict-to-indices-def*
by (*meson count-eq-zero-iff count-filter-mset*)

lemma *restrict-to-indices-submultiset*[*simp*]:
restrict-to-indices m S ⊆# *m*
unfolding *restrict-to-indices-def*
using *multiset-filter-subset*
by *blast*

lemma *restrict-to-indices-add-element*:
restrict-to-indices (*add-mset x m*) *S* = (*if x* ∈ *S then* (*add-mset x* (*restrict-to-indices m S*))
else (*restrict-to-indices m S*))
unfolding *restrict-to-indices-def*
by (*metis filter-mset-add-mset*)

lemma *restrict-to-indices-count*[*simp*]:

count (*restrict-to-indices m S*) *i* = (*if* (*i* ∈ *S*) *then* (*count m i*) *else 0*)
unfolding *restrict-to-indices-def*
by (*metis count-filter-mset*)

lemma *restrict-to-indices-subset*:
restrict-to-indices m S = *restrict-to-indices m* (*set-mset m* ∩ *S*)
proof(*induction m*)
case *empty*
then show *?case* **unfolding** *restrict-to-indices-def*
by (*metis filter-empty-mset*)
next
case (*add x m*)
assume *IH*: *restrict-to-indices m S* = *restrict-to-indices m* (*set-mset m* ∩ *S*)
show *restrict-to-indices* (*add-mset x m*) *S* = *restrict-to-indices* (*add-mset x m*)
(*set-mset* (*add-mset x m*) ∩ *S*)
proof−
have ⋀*i*. *count* (*restrict-to-indices* (*add-mset x m*) *S*) *i* =
count (*restrict-to-indices* (*add-mset x m*) (*set-mset* (*add-mset x m*) ∩
S)) *i*
proof−
fix *i*
show *count* (*restrict-to-indices* (*add-mset x m*) *S*) *i* = *count* (*restrict-to-indices*
(*add-mset x m*) (*set-mset* (*add-mset x m*) ∩ *S*)) *i*
apply(*cases i* ∈ *S*)
using *restrict-to-indices-count*
apply (*metis IntI count-inI*)
by (*metis restrict-to-indices-count Int-iff*)
qed
then show *?thesis*
using *multiset-eq-iff* **by** *blast*
qed
qed

`Restrict_to_indices` only depends on the intersection of the index set with the set of indices in m:

lemma *restrict-to-indices-subset′*:
assumes (*set-mset m*) ∩ *S* = (*set-mset m*) ∩ *S′*
shows *restrict-to-indices m S* = *restrict-to-indices m S′*
by (*metis restrict-to-indices-subset assms*)

lemma *mset-add-plus*:
assumes *m* = *n* + *k*
shows *add-mset x m* = (*add-mset x n*) + *k*
using *assms*
by *simp*

Restricting to S and the complement of S partitions m:

lemma *restrict-to-indices-decomp*:
m = (*restrict-to-indices m S*) + (*restrict-to-indices m* ((*set-mset m*) − *S*))

apply(*induction m*)
apply (*metis add.right-neutral empty-Diff restrict-to-indices-submultiset set-mset-empty subset-mset.le-zero-eq*)
proof−
fix *x*
fix *m*
assume *A*: *m* = *restrict-to-indices m S* + *restrict-to-indices m* (*set-mset m* − *S*)
show *add-mset x m* = *restrict-to-indices* (*add-mset x m*) *S* + *restrict-to-indices* (*add-mset x m*) (*set-mset* (*add-mset x m*) − *S*)
proof(*cases x* ∈ *S*)
case *True*
then have *T0*: *restrict-to-indices* (*add-mset x m*) *S* = (*add-mset x* (*restrict-to-indices m S*))
by (*simp add*: *True restrict-to-indices-add-element*)
have *T1*: *restrict-to-indices* (*add-mset x m*) (*set-mset* (*add-mset x m*) − *S*) = *restrict-to-indices m* (*set-mset* (*add-mset x m*) − *S*)
using *True* **by** (*metis DiffD2 restrict-to-indices-add-element*)
have *T2*: *restrict-to-indices* (*add-mset x m*) *S* + *restrict-to-indices* (*add-mset x m*) (*set-mset* (*add-mset x m*) − *S*)
= (*add-mset x* (*restrict-to-indices m S*)) + *restrict-to-indices m* (*set-mset* (*add-mset x m*) − *S*)
using *T0 T1*
by *presburger*
have *T3*: *add-mset x m* = *add-mset x* (*restrict-to-indices m S*) + *restrict-to-indices m* (*set-mset m* − *S*)
using *T2 A* **using** *mset-add-plus*[*of m restrict-to-indices m S restrict-to-indices m* (*set-mset m* − *S*) *x*]
by *blast*
have *T4*: *set-mset m* ∩ (*set-mset* (*add-mset x m*) − *S*) = *set-mset m* ∩ (*set-mset m* − *S*)
proof
show *set-mset m* ∩ (*set-mset* (*add-mset x m*) − *S*) ⊆ *set-mset m* ∩ (*set-mset m* − *S*)
by *blast*
show *set-mset m* ∩ (*set-mset m* − *S*) ⊆ *set-mset m* ∩ (*set-mset* (*add-mset x m*) − *S*)
by (*simp add*: *True*)
qed
have *T5*: *restrict-to-indices m* (*set-mset* (*add-mset x m*) − *S*) = *restrict-to-indices m* (*set-mset m* − *S*)
using *T4 restrict-to-indices-subset'*[*of m* (*set-mset* (*add-mset x m*) − *S*) (*set-mset m* − *S*)]
by *blast*
then show *?thesis* **using** *T4*
by (*metis T0 T1 T3*)
next
case *False*
then have *F0*: *restrict-to-indices* (*add-mset x m*) *S* = (*restrict-to-indices m S*)

by (*simp add*: *False restrict-to-indices-add-element*)
have *F1*: *restrict-to-indices* (*add-mset x m*) (*set-mset* (*add-mset x m*) − *S*) =
(*add-mset x* (*restrict-to-indices m* (*set-mset* (*add-mset x m*) − *S*)))
using *False*
by (*meson DiffI restrict-to-indices-add-element union-single-eq-member*)

have *F2*: *restrict-to-indices* (*add-mset x m*) *S* + *restrict-to-indices* (*add-mset x m*) (*set-mset* (*add-mset x m*) − *S*)
= (*restrict-to-indices m S*) + (*add-mset x* (*restrict-to-indices m* (*set-mset* (*add-mset x m*) − *S*)))
using *F0 F1*
by *presburger*
have *F3*: *add-mset x m* = (*restrict-to-indices m S*) + (*add-mset x* (*restrict-to-indices m* (*set-mset m* − *S*)))
using *F2 A mset-add-plus*[*of m restrict-to-indices m* (*set-mset m* − *S*) *restrict-to-indices m S x*]
by (*metis union-mset-add-mset-right*)
have *F4*: *add-mset x m* = *restrict-to-indices* (*add-mset x m*) *S*+ (*add-mset x* (*restrict-to-indices m* (*set-mset m* − *S*)))
using *F0 F3* **by** *auto*
have *F5*: *add-mset x* (*restrict-to-indices m* (*set-mset m* − *S*)) = *restrict-to-indices* (*add-mset x m*) (*set-mset* (*add-mset x m*) − *S*)
proof(*cases x* ∈ *set-mset m*)
case *True*
then show *?thesis*
by (*metis F1 add-mset-remove-trivial more-than-one-mset-mset-diff*)
next
case *F*: *False*
have *set-mset m* ∩ (*set-mset* (*add-mset x m*) − *S*) = *set-mset m* ∩(*set-mset m* − *S*)
proof
show *set-mset m* ∩ (*set-mset* (*add-mset x m*) − *S*) ⊆ *set-mset m* ∩ (*set-mset m* − *S*)
using *F False*
by *blast*
show *set-mset m* ∩ (*set-mset m* − *S*) ⊆ *set-mset m* ∩ (*set-mset* (*add-mset x m*) − *S*)
using *F False*
by (*metis Diff-mono mset-add-plus Int-mono add-cancel-right-left*
set-eq-subset subsetI subset-iff union-commute union-iff)
qed
then show *?thesis*
by (*metis F1 restrict-to-indices-subset*′)
qed
then show *?thesis*
using *False F4*
by *presburger*
qed
qed

definition *remove-indices* :: *'c monomial* ⇒ *'c set* ⇒ *'c monomial* **where**
remove-indices m S = (*restrict-to-indices m* (*set-mset m* − *S*))

lemma *remove-indices-decomp*:
m = (*restrict-to-indices m S*) + (*remove-indices m S*)
unfolding *remove-indices-def*
using *restrict-to-indices-decomp*
by *blast*

lemma *remove-indices-indices*[*simp*]:
assumes *set-mset m* ⊆ *I*
shows *set-mset* (*remove-indices m S*) ⊆ *I* − *S*
unfolding *remove-indices-def* **using** *assms*
by (*meson Diff-iff restrict-to-indicesE subsetD subsetI*)

2.11.4 Total evaluation of a monomial

We define total evaluation of a monomial first, and then define the partial evaluation of a monomial in terms of this.

abbreviation(*input*) *closed-fun* **where**
closed-fun R g ≡ *g* ∈ *UNIV* → *carrier R*

definition *monom-eval* :: ('*a*, '*b*) *ring-scheme* ⇒ *'c monomial* ⇒ ('*c* ⇒ '*a*) ⇒ *'a* **where**
monom-eval R (*m*:: *'c monomial*) *g* = *fold-mset* (λ *x* . λ *y*. *if y* ∈ *carrier R then* (*g x*) ⊗$_R$ *y else* **0**$_R$) **1**$_R$ *m*

context *cring*
begin

lemma *closed-fun-simp*:
assumes *closed-fun R g*
shows *g n* ∈ *carrier R*
using *assms*
by *blast*

lemma *closed-funI*:
assumes ⋀*x*. *g x* ∈ *carrier R*
shows *closed-fun R g*
by (*meson Pi-I assms*)

The following are necessary technical lemmas to prove properties of about folds over multisets:

lemma *monom-eval-comp-fun*:
fixes *g*:: *'c* ⇒ *'a*
assumes *closed-fun R g*
shows *comp-fun-commute* (λ *x* . λ*y*. *if y* ∈ *carrier R then* (*g x*) ⊗ *y else* **0**)

unfolding *comp-fun-commute-def*
proof−
have $\bigwedge x\ y.\ (\lambda ya.\ \textit{if } ya \in \textit{carrier } R \textit{ then } g\ y \otimes ya \textit{ else } \mathbf{0}) \circ (\lambda y.\ \textit{if } y \in \textit{carrier } R \textit{ then } g\ x \otimes y \textit{ else } \mathbf{0}) =$
$(\lambda y.\ \textit{if } y \in \textit{carrier } R \textit{ then } g\ x \otimes y \textit{ else } \mathbf{0}) \circ (\lambda ya.\ \textit{if } ya \in \textit{carrier } R \textit{ then } g\ y \otimes ya \textit{ else } \mathbf{0})$
proof
fix *x y a*
show $((\lambda ya.\ \textit{if } ya \in \textit{carrier } R \textit{ then } g\ y \otimes ya \textit{ else } \mathbf{0}) \circ (\lambda y.\ \textit{if } y \in \textit{carrier } R \textit{ then } g\ x \otimes y \textit{ else } \mathbf{0}))\ a =$
$((\lambda y.\ \textit{if } y \in \textit{carrier } R \textit{ then } g\ x \otimes y \textit{ else } \mathbf{0}) \circ (\lambda ya.\ \textit{if } ya \in \textit{carrier } R \textit{ then } g\ y \otimes ya \textit{ else } \mathbf{0}))\ a$
proof(*cases* $a \in \textit{carrier } R$)
case *True*
then show *?thesis*
proof−
have *LHS*: $((\lambda ya.\ \textit{if } ya \in \textit{carrier } R \textit{ then } g\ y \otimes ya \textit{ else } \mathbf{0}) \circ (\lambda y.\ \textit{if } y \in \textit{carrier } R \textit{ then } g\ x \otimes y \textit{ else } \mathbf{0}))\ a =$
$((\lambda ya.\ \textit{if } ya \in \textit{carrier } R \textit{ then } g\ y \otimes ya \textit{ else } \mathbf{0})\ \ (g\ x \otimes a))$
using *True assms*(*1*) *m-closed m-lcomm*
unfolding *o-def*
by *presburger*

then have *LHS'*: $((\lambda ya.\ \textit{if } ya \in \textit{carrier } R \textit{ then } g\ y \otimes ya \textit{ else } \mathbf{0}) \circ (\lambda y.\ \textit{if } y \in \textit{carrier } R \textit{ then } g\ x \otimes y \textit{ else } \mathbf{0}))\ a = \ \ g\ y \otimes (g\ x \otimes a)$
using *True assms m-closed*
by (*meson PiE UNIV-I*)
then show *?thesis*
unfolding *o-def*
using *True assms m-closed m-lcomm*
by (*smt PiE UNIV-I*)
qed
next
case *False*
then show *?thesis*
unfolding *o-def*
using *assms r-null closed-fun-simp*
by *smt*
qed
qed
then show $\forall\, y\ x.\ (\lambda ya.\ \textit{if } ya \in \textit{carrier } R \textit{ then } g\ y \otimes ya \textit{ else } \mathbf{0}) \circ (\lambda y.\ \textit{if } y \in \textit{carrier } R \textit{ then } g\ x \otimes y \textit{ else } \mathbf{0}) =$
$(\lambda y.\ \textit{if } y \in \textit{carrier } R \textit{ then } g\ x \otimes y \textit{ else } \mathbf{0}) \circ (\lambda ya.\ \textit{if } ya \in \textit{carrier } R \textit{ then } g\ y \otimes ya \textit{ else } \mathbf{0})$
by *blast*
qed

lemma *monom-eval-car*:
assumes *closed-fun R g*

shows *monom-eval R* (*m*:: *'c monomial*) *g* ∈ *carrier R*
proof(*induction m*)
case *empty*
then show *?case*
unfolding *monom-eval-def*
by (*metis fold-mset-empty one-closed*)
next
case (*add x m*)
fix *x m*
assume *A*: *monom-eval R m g* ∈ *carrier R*
obtain *f* **where** *f-def*: *f* = (λ *x* . λ*y*. *if y* ∈ *carrier R then* (*g x*) ⊗ *y else* **0**)
by *blast*
have *0*: *comp-fun-commute f*
using *assms monom-eval-comp-fun*[*of g*] *f-def*
by *blast*
have *1*: ⋀*m*. *monom-eval R m g* = *fold-mset f* **1** *m*
using *f-def monom-eval-def*
by *blast*
have *2*: *monom-eval R* (*add-mset x m*) *g* = *fold-mset f* **1** (*add-mset x m*)
using *1* **by** *blast*
have *3*: *g x* ∈ *carrier R*
using *assms* **by** *blast*
then show *monom-eval R* (*add-mset x m*) *g* ∈ *carrier R*
using *assms 0 1 2 3*
by (*metis A comp-fun-commute.fold-mset-add-mset f-def m-closed*)
qed

Formula for recursive (total) evaluation of a monomial:

lemma *monom-eval-add*:
assumes *closed-fun R g*
shows *monom-eval R* (*add-mset x M*) *g* = (*g x*) ⊗ (*monom-eval R M g*)
proof−
obtain *f* **where** *f-def*: *f* = (λ *x* . λ*y*. *if y* ∈ *carrier R then* (*g x*) ⊗ *y else* **0**)
by *blast*
have *0*: *comp-fun-commute f*
using *assms monom-eval-comp-fun f-def*
by *blast*
have *1*: ⋀*m*. *monom-eval R m g* = *fold-mset f* **1** *m*
using *f-def monom-eval-def*
by *blast*
have *2*: *monom-eval R* (*add-mset x M*) *g* = *fold-mset f* **1** (*add-mset x M*)
using *1* **by** *blast*
have *3*: *g x* ∈ *carrier R*
using *assms* **by** *blast*
have *4*: (*g x*) ⊗ (*monom-eval R M g*) = *f x* (*monom-eval R M g*)
using *f-def 3*
by (*meson assms monom-eval-car*)
then show *?thesis*
by (*metis 0 1 comp-fun-commute.fold-mset-add-mset*)

qed

end

This function maps a polynomial P to the set of monomials in P which, after evaluating all variables in the set S to values in the ring R, reduce to the monomial n.

definition *monomials-reducing-to* ::
('*a*,'*b*) *ring-scheme* ⇒ '*c monomial* ⇒ ('*a*,'*c*) *mvar-poly* ⇒ '*c set* ⇒ ('*c monomial*) *set* **where**
monomials-reducing-to R n P S = {*m* ∈ *monomials-of R P*. *remove-indices m S* = *n*}

lemma *monomials-reducing-to-subset*[*simp*]:
monomials-reducing-to R n P s ⊆ *monomials-of R P*
unfolding *monomials-reducing-to-def*
by *blast*

context *cring*
begin

lemma *monomials-reducing-to-finite*:
assumes *P* ∈ *Pring-set R I*
shows *finite* (*monomials-reducing-to R n P s*)
by (*meson assms monomials-finite monomials-reducing-to-subset rev-finite-subset*)

lemma *monomials-reducing-to-disjoint*:
assumes *n1* ≠ *n2*
shows *monomials-reducing-to R n1 P S* ∩ *monomials-reducing-to R n2 P S* = {}
unfolding *monomials-reducing-to-def*
using *assms*
by *blast*

lemma *monomials-reducing-to-submset*:
assumes *n* ⊂# *m*
shows *n* ∉ *monomials-reducing-to R m P S*
proof(*rule ccontr*)
assume *C*: ¬ *n* ∉ *monomials-reducing-to R m P S*
then have *n* ∈ *monomials-reducing-to R m P S*
by *blast*
then have *remove-indices n S* = *m*
unfolding *monomials-reducing-to-def*
by *blast*
then show *False*
by (*metis* (*full-types*) *remove-indices-def restrict-to-indices-submultiset assms subset-mset.less-asym' subset-mset.less-irrefl subset-mset-def*)
qed

end

2.11.5 Partial Evaluation of a Polynomial

This function takes as input a set S of variables, an evaluation function g, and a polynomial to evaluate P. The output is a polynomial which is the result of evaluating the variables from the set S which occur in P, according to the evaluation function g.

definition *poly-eval* ::
  ('*a*,'*b*) *ring-scheme* ⇒ '*c set* ⇒ ('*c* ⇒ '*a*) ⇒ ('*a*, '*c*) *mvar-poly* ⇒ ('*a*, '*c*) *mvar-poly***where**
poly-eval R S g P m = (*finsum R* (λ*n*. *monom-eval R* (*restrict-to-indices n S*) *g* ⊗$_R$ (*P n*)) (*monomials-reducing-to R m P S*))

context *cring*
begin

lemma *finsum-singleton*:
  assumes $S = \{s\}$
  assumes *f s* ∈ *carrier R*
  shows *finsum R f S* = *f s*
proof−
  have *finsum R f S* = *finsum R f* (*insert s* {})
    using *assms*(*1*)
    by *blast*
  then show *?thesis* **using** *finsum-insert*[*of* {} *s f*] *assms*
    by (*metis Pi-I empty-iff finite.emptyI finsum-empty r-zero*)
qed

lemma *poly-eval-constant*:
  assumes *k* ∈ *carrier R*
  shows *poly-eval R S g* (*indexed-const k*) = (*indexed-const k*)
proof
  have *S*: *monomials-of R* (*indexed-const k*) ⊆ {{#}}
    unfolding *indexed-const-def monomials-of-def*
    by (*metis* (*mono-tags*, *lifting*) *mem-Collect-eq singletonI subset-iff*)
  fix *x*
  show *poly-eval R S g* (*indexed-const k*) *x* = *indexed-const k x*
  proof(*cases x* = {#})
    case *True*
    have (*monomials-reducing-to R x* (*indexed-const k*) *S*) ⊆ {{#}}
      using *S monomials-reducing-to-subset*
      by *blast*
    then show *?thesis*
    proof(*cases k* = **0**)
      case *True*
      then have (*monomials-reducing-to R x* (*indexed-const k*) *S*) = {}
        by (*metis S* ‹*monomials-reducing-to R x* (*indexed-const k*) *S* ⊆ {{#}}›

monomials-reducing-to-subset monoms-of-const subset-antisym subset-singletonD)
then show *?thesis*
unfolding *poly-eval-def*
by (*metis True finsum-empty indexed-const-def*)
next
case *False*
then have *monomials-of R* (*indexed-const k*) = {{#}}
by (*meson monoms-of-const*)
have *remove-indices* {#} *S* = {#}
using *remove-indices-decomp* **by** *blast*
then have {#} ∈ *monomials-reducing-to R x* (*indexed-const k*) *S*
using *True False* **unfolding** *monomials-reducing-to-def*
‹*monomials-of R* (*indexed-const k*) = {{#}}›
by *blast*
then have *0*: *monomials-reducing-to R x* (*indexed-const k*) *S* = {{#}}
using ‹*monomials-reducing-to R x* (*indexed-const k*) *S* ⊆ {{#}}›
by *blast*
have *1*: *restrict-to-indices* {#} *S* = {#}
using *restrict-to-indices-submultiset remove-indices-decomp* **by** *blast*
have *2*: *monom-eval R* (*restrict-to-indices* {#} *S*) *g* = **1**
unfolding *monom-eval-def*
using *1*
by (*metis fold-mset-empty*)
have *3*: *poly-eval R S g* (*indexed-const k*) *x* =
(⨁ *n*∈{{#}}. *monom-eval R* (*restrict-to-indices n S*) *g* ⊗ *indexed-const k n*)
unfolding *poly-eval-def*
using *0*
by *presburger*
have *4*: (⨁ *n*∈{{#}}. *monom-eval R* (*restrict-to-indices n S*) *g* ⊗ *indexed-const k n*) = *monom-eval R* (*restrict-to-indices* {#} *S*) *g* ⊗ *indexed-const k* {#}
using *finsum-singleton*[*of* {{#}} {#} λ*n*. *monom-eval R* (*restrict-to-indices n S*) *g* ⊗ *indexed-const k n*]
by (*metis 2 assms indexed-const-def l-one*)
then show *?thesis* **unfolding** *poly-eval-def*
using *0 1 2*
by (*metis True assms indexed-const-def l-one*)
qed
next
case *False*
then have *F0*: (*indexed-const k*) *x* = **0**
by (*meson indexed-const-def*)
have (*monomials-reducing-to R x* (*indexed-const k*) *S*) = {}
unfolding *monomials-reducing-to-def*
proof(*rule ccontr*)
assume *A*: {*m* ∈ *monomials-of R* (*indexed-const k*). *remove-indices m S* = *x*} ≠ {}

```
      then obtain m where m-def: m ∈ monomials-of R (indexed-const k) ∧
remove-indices m S = x
       by blast
       then show False using A F0
        by (metis False S empty-eq-union empty-iff
            remove-indices-decomp singletonD subset-singletonD)
    qed
    then show ?thesis
     unfolding poly-eval-def
     by (metis False finsum-empty indexed-const-def)
  qed
qed

lemma finsum-partition:
  assumes finite S
  assumes f ∈ S → carrier R
  assumes T ⊆ S
  shows finsum  R f S  = finsum R f T  ⊕ finsum  R f (S − T)
proof−
  have ⋀U. finite U ⟹ U ⊆ S ⟶ finsum  R f S  = finsum R f U  ⊕ finsum  R
f (S − U)
  proof−
    fix U
    show finite U ⟹ U ⊆ S ⟶ finsum R f S = finsum R f U ⊕ finsum R f (S
− U)
    apply(erule finite.induct)
      apply (metis Diff-empty assms(2) finsum-closed finsum-empty l-zero)
    proof
     fix A :: 'c set fix a
     assume A0: finite A
     show A ⊆ S ⟶ finsum R f S = finsum R f A ⊕ finsum R f (S − A) ⟹
insert a A ⊆ S ⟹ finsum R f S = finsum R f (insert a A) ⊕ finsum R f (S −
insert a A)

     proof−
       assume A1: A ⊆ S ⟶ finsum R f S = finsum R f A ⊕ finsum R f (S −
A)
       assume A2: insert a A ⊆ S
       show finsum R f S = finsum R f (insert a A) ⊕ finsum R f (S − insert a
A)
         apply(cases a ∈ A)
          apply (metis A1 A2 insert-absorb)
       proof−
         assume A3: a ∉ A
         have A4: f a ∈ carrier R
           by (metis A2 Pi-iff assms(2) insert-subset)
         have A5:  finsum R f (insert a A) = f a ⊕ finsum R f A

           using A0 A1 A2 finsum-insert[of A a f] assms  A3
```

by *blast*
have *A6*: $a \in S$
using *A2* **by** *blast*
have *A7*: *finsum R f S* $\in$ *carrier R*
using *assms*(*2*) *finsum-closed* **by** *blast*
have *A8*: *finsum R f* (*S* − *A*)$\in$ *carrier R*
using *Diff-subset*[*of S A*] *Pi-iff assms*(*2*) *finsum-closed*[*of f*] *subsetD*[*of - S*]
by (*meson Pi-anti-mono in-mono*)
have *A9*: *finsum R f A* $\in$ *carrier R*
by (*meson A2 Pi-anti-mono assms*(*2*) *finsum-closed insert-subset subsetD*)
have *A10*: *finsum R f A* = *finsum R f S* $\ominus$ *finsum R f* (*S* − *A*)
using *A7 A8 A9*
by (*metis A1 A2 add.inv-solve-right*′ *insert-subset minus-eq*)
have *A11*: *finsum R f* (*insert a A*) = *f a* $\oplus$ (*finsum R f S* $\ominus$ *finsum R f* (*S* − *A*))
using *A5 A6 A1 A2 assms A10*
by *presburger*
then have *A12*: *finsum R f* (*insert a A*) =*finsum R f S* $\oplus$(*f a* $\ominus$ *finsum R f* (*S* − *A*))
using *A4 A7 A8 add.inv-closed add.m-lcomm minus-eq* **by** *presburger*
have *A13*: *finsum R f* (*insert a A*) $\in$ *carrier R*
using *A4 A5 A9 add.m-closed*
by *presburger*
have *A14*: *finsum R f* (*S* − *insert a A*) $\in$ *carrier R*
by (*meson Diff-subset Pi-anti-mono assms*(*2*) *finsum-closed in-mono*)
have *A15*: *finsum R f S* = *finsum R f* (*insert a A*) $\ominus$ (*f a* $\ominus$ *finsum R f* (*S* − *A*))
by (*metis A12 A13 A4 A7 A8 add.inv-solve-right*′ *minus-closed minus-eq*)
have *A16*: *finsum R f S* = *finsum R f* (*insert a A*) $\oplus$ *finsum R f* (*S* − *A*) $\ominus$ *f a*
using *A1 A2 A4 A5 A8 A9 add.inv-closed add.m-assoc add.m-comm insert-subset minus-closed minus-eq r-neg1*
unfolding *a-minus-def*
by (*metis add.m-closed*)
have *A16*: *finsum R f S* = *finsum R f* (*insert a A*) $\oplus$ (*finsum R f* (*S* − *A*) $\ominus$ *f a*)
unfolding *a-minus-def*
using *A13 A16 A4 A8 add.inv-closed add.m-assoc minus-eq* **by** *presburger*
have *A17*: (*finsum R f* (*S* − *A*) $\ominus$ *f a*) = *finsum R f* (*S* − *insert a A*)
proof−
have *A170*: (*S* − *A*) = *insert a* (*S* − *insert a A*)
using *A3 A6* **by** *blast*
have *A171*: *a* $\notin$ *S* − *insert a A*
by *blast*
then have *finsum R f* (*S* − *A*) =(*f a*) $\oplus$ *finsum R f* (*S* − *insert a A*)
using *A170 finsum-insert*[*of* (*S* − *insert a A*) *a f*]
by (*metis A4 Diff-subset Pi-anti-mono assms*(*1*) *assms*(*2*)
rev-finite-subset subsetD)

```
          then show ?thesis
            by (metis A1 A13 A14 A16 A2 A4 A5 A8 A9 add.l-cancel
                add.m-assoc add.m-comm insert-subset minus-closed)
        qed
          then show finsum R f S = finsum R f (insert a A) ⊕ finsum R f (S −
insert a A)
           using A16
           by presburger
       qed
     qed
   qed
  qed
  then show ?thesis
    by (meson assms(1) assms(3) rev-finite-subset)
qed

lemma finsum-eq-parition:
  assumes finite S
  assumes f ∈ S → carrier R
  assumes T ⊆ S
  assumes ⋀x. x ∈ S − T ⟹ f x = 0
  shows finsum  R f S  = finsum R f T
  using assms
  by (metis add.finprod-mono-neutral-cong-right)

lemma poly-eval-scalar-mult:
  assumes k ∈ carrier R
  assumes closed-fun R g
  assumes P ∈ Pring-set R I
  shows poly-eval R  S g (poly-scalar-mult R k P)=
        (poly-scalar-mult R k (poly-eval R S g P))
proof
  fix m
 show poly-eval R S g (poly-scalar-mult R k P) m = poly-scalar-mult R k (poly-eval
R S g  P) m
    unfolding poly-eval-def poly-scalar-mult-def
  proof−
   have 0: (⨁ n∈monomials-reducing-to R m P S. monom-eval R (restrict-to-indices
n S) g ⊗ (k ⊗ P n)) =
             (⨁ n∈monomials-reducing-to R m (λm. k ⊗ P m) S. monom-eval R
(restrict-to-indices n S) g ⊗ (k ⊗ P n))
    proof−
    have 00: monomials-reducing-to R m (λm. k ⊗ P m) S ⊆ monomials-reducing-to
R m P S
      proof
        fix x show  x ∈ monomials-reducing-to R m (λm. k ⊗ P m) S ⟹ x ∈
monomials-reducing-to R m P S
       unfolding monomials-reducing-to-def
      using assms  assms(1) monomials-ofE complement-of-monomials-of  r-null[of
```

k]
by (*metis* (*no-types*, *lifting*) *mem-Collect-eq*)
have *01*: (λ*n*. *monom-eval R* (*restrict-to-indices n S*) *g* ⊗ (*k* ⊗ *P n*)) ∈ *monomials-reducing-to R m P S* → *carrier R*
by (*smt Pi-I Pring-cfs-closed assms*(*1*) *assms*(*2*) *assms*(*3*) *closed-fun-simp m-closed monom-eval-car*)
have *02*: *finite* (*monomials-reducing-to R m P S*)
using *assms*(*3*) *monomials-reducing-to-finite*
by *blast*
have *03*: (⋀*x*. *x* ∈ *monomials-reducing-to R m P S* − *monomials-reducing-to R m* (λ*m*. *k* ⊗ *P m*) *S* ⟹
monom-eval R (*restrict-to-indices x S*) *g* ⊗ (*k* ⊗ *P x*) = **0**)
proof−
fix *x*
assume *A*: *x* ∈ *monomials-reducing-to R m P S* − *monomials-reducing-to R m* (λ*m*. *k* ⊗ *P m*) *S*
have *x* ∉ *monomials-of R* ((λ*m*. *k* ⊗ *P m*))
proof
assume *x* ∈ *monomials-of R* (λ*m*. *k* ⊗ *P m*)
then have *x* ∈ *monomials-reducing-to R m* (λ*m*. *k* ⊗ *P m*) *S*
using *A*
unfolding *monomials-reducing-to-def*
by *blast*
then show *False*
using *A* **by** *blast*
qed
then show *monom-eval R* (*restrict-to-indices x S*) *g* ⊗ (*k* ⊗ *P x*) = **0**
by (*metis assms*(*2*) *complement-of-monomials-of monom-eval-car r-null*)
qed
qed
have *01*: (⋀*x*. *x* ∈ *monomials-reducing-to R m P S* − *monomials-reducing-to R m* (λ*m*. *k* ⊗ *P m*) *S* ⟹ *monom-eval R* (*restrict-to-indices x S*) *g* ⊗ (*k* ⊗ *P x*) = **0**)
proof− **fix** *x* **assume** *A*: *x* ∈ *monomials-reducing-to R m P S* − *monomials-reducing-to R m* (λ*m*. *k* ⊗ *P m*) *S*
hence *x* ∉ *monomials-reducing-to R m* (λ*m*. *k* ⊗ *P m*) *S*
by *blast*
hence (*k* ⊗ *P x*) = **0** **unfolding** *monomials-reducing-to-def*
by (*metis* (*no-types*, *lifting*) *A DiffD1 complement-of-monomials-of mem-Collect-eq monomials-reducing-to-def*)
thus *monom-eval R* (*restrict-to-indices x S*) *g* ⊗ (*k* ⊗ *P x*) = **0**
using *monom-eval-car*[*of g*] *assms*(*2*) *r-null* **by** *presburger*
qed
have *02*: *finite* (*monomials-reducing-to R m P S*)
using *assms*(*3*) *monomials-reducing-to-finite* **by** *blast*
have *04*: (λ*n*. *monom-eval R* (*restrict-to-indices n S*) *g* ⊗ (*k* ⊗ *P n*)) ∈ *monomials-reducing-to R m P S* → *carrier R*
by (*smt Pi-I assms*(*1*) *assms*(*2*) *assms*(*3*) *closed-fun-simp cring.axioms*(*1*) *is-cring m-closed monom-eval-car ring.Pring-cfs-closed*)

show *?thesis*
using *00 01 02 04*
finsum-eq-parition[*of monomials-reducing-to R m P S*
(λn. *monom-eval R* (*restrict-to-indices n S*) *g* $\otimes$ (*k* $\otimes$ *P n*))
monomials-reducing-to R m (λm. *k* $\otimes$ *P m*) *S*]
by *blast*
qed
then have *1*: ($\bigoplus$ *n*$\in$*monomials-reducing-to R m* (λm. *k* $\otimes$ *P m*) *S*. *monom-eval R* (*restrict-to-indices n S*) *g* $\otimes$ (*k* $\otimes$ *P n*))
= ($\bigoplus$ *n*$\in$*monomials-reducing-to R m P S*. *k* $\otimes$ (*monom-eval R* (*restrict-to-indices n S*) *g* $\otimes$ (*P n*)))
proof −
have $\bigwedge n$. *monom-eval R* (*restrict-to-indices n S*) *g* $\otimes$ (*k* $\otimes$ *P n*) = *k* $\otimes$ (*monom-eval R* (*restrict-to-indices n S*) *g* $\otimes$ (*P n*))
by (*metis Pring-cfs-closed assms*(*1*) *assms*(*2*) *assms*(*3*) *m-lcomm monom-eval-car*)
then show *?thesis*
using *0*
by *presburger*
qed
show ($\bigoplus$ *n*$\in$*monomials-reducing-to R m* (λm. *k* $\otimes$ *P m*) *S*. *monom-eval R* (*restrict-to-indices n S*) *g* $\otimes$ (*k* $\otimes$ *P n*))
= *k* $\otimes$($\bigoplus$ *n*$\in$*monomials-reducing-to R m P S*. *monom-eval R* (*restrict-to-indices n S*) *g* $\otimes$ (*P n*))
proof −
have (λn. *monom-eval R* (*restrict-to-indices n S*) *g* $\otimes$ (*P n*)) $\in$ (*monomials-reducing-to R m P S*) $\to$ *carrier R*
by (*meson Pi-I Pring-cfs-closed assms*(*2*) *assms*(*3*) *m-closed monom-eval-car*)
then have *k* $\otimes$ ($\bigoplus$ *i*$\in$*monomials-reducing-to R m P S*. *monom-eval R* (*restrict-to-indices i S*) *g* $\otimes$ *P i*) =
($\bigoplus$ *i*$\in$*monomials-reducing-to R m P S*. *k* $\otimes$ (*monom-eval R* (*restrict-to-indices i S*) *g* $\otimes$ *P i*))
using *finsum-rdistr*[*of monomials-reducing-to R m P S*
k
(λn. *monom-eval R* (*restrict-to-indices n S*) *g* $\otimes$ (*P n*))]
assms monomials-reducing-to-finite **by** *blast*
then show *?thesis*
using *1*
by *presburger*
qed
qed
qed

lemma *poly-eval-monomial*:
assumes *closed-fun R g*
assumes 1 $\neq$**0**
shows *poly-eval R S g* (*mset-to-IP R m*)
= *poly-scalar-mult R* (*monom-eval R* (*restrict-to-indices m S*) *g*)
(*mset-to-IP R* (*remove-indices m S*))

proof
have *0*: *monomials-of R* (*mset-to-IP R m*) = {*m*}
using *assms monomials-of-mset-to-IP*
by *blast*

fix *x*
show *poly-eval R S g* (*mset-to-IP R m*) *x* =
poly-scalar-mult R (*monom-eval R* (*restrict-to-indices m S*) *g*)
(*mset-to-IP R* (*remove-indices m S*)) *x*
proof(*cases x* = (*remove-indices m S*))
case *True*
have *monomials-reducing-to R x* (*mset-to-IP R m*) *S* = {*m*}
unfolding *monomials-reducing-to-def*
using *True 0*
by *auto*
then have *poly-eval R S g* (*mset-to-IP R m*) *x* = *monom-eval R* (*restrict-to-indices m S*) *g* ⊗ (*mset-to-IP R m m*)
unfolding *poly-eval-def*
by (*metis* (*mono-tags*, *lifting*) *assms*(*1*) *finsum-singleton monom-eval-car mset-to-IP-simp r-one*)
then show *?thesis*
by (*metis True mset-to-IP-simp poly-scalar-mult-def*)
next
case *False*
then have *monomials-reducing-to R x* (*mset-to-IP R m*) *S* = {}
unfolding *monomials-reducing-to-def*
using *0*
by *auto*
then have *poly-eval R S g* (*mset-to-IP R m*) *x* = **0**
unfolding *poly-eval-def*
by (*metis finsum-empty*)
then show *?thesis*
using *False*
by (*metis assms*(*1*) *monom-eval-car mset-to-IP-simp′ poly-scalar-mult-def r-null*)
qed
qed

lemma(**in** *cring*) *poly-eval-monomial-closed*:
assumes *closed-fun R g*
assumes **1** ≠**0**
assumes *set-mset m* ⊆ *I*
shows *poly-eval R S g* (*mset-to-IP R m*) ∈ *Pring-set R* (*I* − *S*)
proof−
have (*mset-to-IP R* (*remove-indices m S*)) ∈ *Pring-set R* (*I* − *S*)
using *assms mset-to-IP-closed*[*of* (*remove-indices m S*) *I* − *S*]
by (*metis remove-indices-indices*)
then show *?thesis*

using *assms poly-eval-monomial*[*of g S m*]
poly-scalar-mult-closed[*of* (*monom-eval R* (*restrict-to-indices m S*) *g*)
(*mset-to-IP R* (*remove-indices m S*))]
by (*metis monom-eval-car*)
qed

lemma *poly-scalar-mult-iter*:
assumes 1 $\neq$**0**
assumes $P \in$ *Pring-set R I*
assumes $k \in$ *carrier R*
assumes $n \in$ *carrier R*
shows *poly-scalar-mult R k* (*poly-scalar-mult R n P*) = *poly-scalar-mult R* ($k \otimes n$) *P*
using *assms*
unfolding *poly-scalar-mult-def*
by (*metis Pring-cfs-closed m-assoc*)

lemma *poly-scalar-mult-comm*:
assumes 1 $\neq$**0**
assumes $P \in$ *Pring-set R I*
assumes $a \in$ *carrier R*
assumes $b \in$ *carrier R*
shows *poly-scalar-mult R a* (*poly-scalar-mult R b P*) = *poly-scalar-mult R b* (*poly-scalar-mult R a P*)
using *assms poly-scalar-mult-iter m-comm*[*of a b*]
by *metis*

lemma *poly-eval-monomial-term*:
assumes *closed-fun R g*
assumes 1 $\neq$**0**
assumes *set-mset m* $\subseteq$ *I*
assumes $k \in$ *carrier R*
shows *poly-eval R S g* (*poly-scalar-mult R k* (*mset-to-IP R m*)) = *poly-scalar-mult R* ($k \otimes$(*monom-eval R* (*restrict-to-indices m S*) *g*))
(*mset-to-IP R* (*remove-indices m S*))
proof−
have *0*: *poly-eval R S g* (*poly-scalar-mult R k* (*mset-to-IP R m*)) =
poly-scalar-mult R k (*poly-eval R S g* (*mset-to-IP R m*))
using *assms poly-eval-scalar-mult*[*of k g mset-to-IP R m I S*] *mset-to-IP-closed*
by *blast*
have *1*: *poly-eval R S g* (*poly-scalar-mult R k* (*mset-to-IP R m*)) =
poly-scalar-mult R k (*poly-scalar-mult R* (*monom-eval R* (*restrict-to-indices m S*) *g*)
(*mset-to-IP R* (*remove-indices m S*)))
using *0 assms*
by (*metis poly-eval-monomial*)
have *2*: *mset-to-IP R* (*remove-indices m S*) $\in$ *Pring-set R I*
using *assms mset-to-IP-closed*
by (*metis Diff-subset remove-indices-def restrict-to-indicesE subset-iff*)

have *3*: *monom-eval R* (*restrict-to-indices m S*) *g* ∈ *carrier R*
using *assms monom-eval-car* **by** *blast*
show *?thesis*
using *1 2 3 assms poly-scalar-mult-iter*[*of mset-to-IP R* (*remove-indices m S*) *I k* (*monom-eval R* (*restrict-to-indices m S*) *g*)]
by *presburger*
qed

lemma *poly-eval-monomial-term-closed*:
assumes *closed-fun R g*
assumes **1** ≠**0**
assumes *set-mset m* ⊆ *I*
assumes *k* ∈ *carrier R*
shows *poly-eval R S g* (*poly-scalar-mult R k* (*mset-to-IP R m*)) ∈ *Pring-set R* (*I* − *S*)
proof−
have (*mset-to-IP R* (*remove-indices m S*)) ∈ *Pring-set R* (*I* − *S*)
using *assms*
by (*meson remove-indices-indices mset-to-IP-closed*)
then show *?thesis*
using *assms poly-eval-monomial-term remove-indices-indices*[*of m I S*]
by (*metis cring.cring-simprules*(*5*) *cring.monom-eval-car is-cring poly-scalar-mult-closed*)
qed

lemma *finsum-split*:
assumes *finite S*
assumes *f* ∈ *S* → *carrier R*
assumes *g* ∈ *S* → *carrier R*
assumes *k* ∈ *carrier R*
assumes *c* ∈ *S*
assumes ⋀*s*. *s* ∈ *S* ∧ *s* ≠ *c* ⟹ *f s* = *g s*
assumes *g c* = *f c* ⊕ *k*
shows *finsum R g S* = *k* ⊕ *finsum R f S*
proof−
have *0*: *finsum R f S* = *f c* ⊕ *finsum R f* (*S* − {*c*})
proof−
have *f* ∈ *S* − {*c*} → *carrier R*
by (*metis Pi-split-insert-domain assms*(*2*) *assms*(*5*) *insert-Diff*)
then show *?thesis*
using *assms finsum-insert*[*of S* − {*c*} *c f*]
by (*metis DiffD2 Pi-iff finite-Diff insert-Diff singletonI*)
qed
have *1*: *finsum R g S* = *g c* ⊕ *finsum R g* (*S* − {*c*})
proof−
have *g* ∈ *S* − {*c*} → *carrier R*
by (*metis Pi-split-insert-domain assms*(*3*) *assms*(*5*) *insert-Diff*)
then show *?thesis*
using *assms finsum-insert*[*of S* − {*c*} *c g*]
by (*metis DiffD2 Pi-iff finite-Diff insert-Diff singletonI*)

```
  qed
  have finsum R f (S − {c}) = finsum R g (S − {c})
    using assms Diff-iff Pi-split-insert-domain finsum-cong'[of S− {c} S − {c} g
f]
      insert-Diff singletonI
    by blast
  then have finsum R g S = f c ⊕ k ⊕ finsum R g (S − {c})
    using assms ‹finsum R g S = g c ⊕ finsum R g (S − {c})›
    by presburger
  then have 1: finsum R g S = f c ⊕ k ⊕ finsum R f (S − {c})
    using ‹finsum R f (S − {c}) = finsum R g (S − {c})› by presburger
  have finsum R g S = k ⊕ ( f c ⊕ finsum R f (S − {c}))
  proof−
    have f c ∈ carrier R
      by (metis PiE assms(2) assms(5))
    have finsum R f (S − {c}) ∈ carrier R
     by (metis Pi-split-insert-domain assms(2) assms(5) finsum-closed insert-Diff)
    then show ?thesis using assms(4) 1
      using ‹f c ∈ carrier R› add.m-assoc add.m-lcomm
      by presburger
  qed
  then show ?thesis
    using 0
    by presburger
qed

lemma poly-monom-induction:
assumes P (indexed-const 0)
assumes ⋀m k. set-mset m ⊆ I ∧ k ∈ carrier R ⟹ P (poly-scalar-mult R k
(mset-to-IP R m))
assumes ⋀Q m k. Q ∈ Pring-set R I ∧ (P Q) ∧ set-mset m ⊆ I ∧ k ∈ carrier
R⟹ P (Q ⨁ (poly-scalar-mult R k (mset-to-IP R m)))
shows ⋀Q. Q ∈ Pring-set R I ⟹ P Q
proof−
  have 0: ⋀Ms. finite Ms ⟹ (∀ Q ∈ Pring-set R I. monomials-of R Q = Ms
⟶ P Q)
  proof−
    fix Ms
   show finite Ms ⟹ (∀ Q ∈ Pring-set R I. monomials-of R Q = Ms ⟶ P Q)
    proof(erule finite.induct)
      show ∀ Q∈Pring-set R I. monomials-of R Q = {} ⟶ P Q
      proof
        fix Q
        assume Q ∈ Pring-set R I
        show monomials-of R Q = {} ⟶ P Q
          using assms
              by (metis ‹Q ∈ Pring-set R I› card-0-eq monomials-finite monomi-
als-of-card-zero)
      qed
```

show $\bigwedge A\ a.\ \mathit{finite}\ A \Longrightarrow \forall\, Q{\in}\mathit{Pring\text{-}set}\ R\ I.\ \mathit{monomials\text{-}of}\ R\ Q = A \longrightarrow P\ Q \Longrightarrow$
$\forall\, Q{\in}\mathit{Pring\text{-}set}\ R\ I.\ \mathit{monomials\text{-}of}\ R\ Q = \mathit{insert}\ a\ A \longrightarrow P\ Q$
proof
fix $A :: 'c\ \mathit{monomial\ set}$ **fix** a **fix** Q
assume *A0*: *finite A*
assume *A1*: $\forall\, Q{\in}\mathit{Pring\text{-}set}\ R\ I.\ \mathit{monomials\text{-}of}\ R\ Q = A \longrightarrow P\ Q$
assume *A2*: $Q \in \mathit{Pring\text{-}set}\ R\ I$
show $\mathit{monomials\text{-}of}\ R\ Q = \mathit{insert}\ a\ A \longrightarrow P\ Q$
proof
assume *A3*: *monomials-of R Q = insert a A*
show *P Q*
apply(*cases a ∈ A*)
apply (*metis A1 A2 A3 insert-absorb*)
proof−
assume *A4*: $a \notin A$
show *P Q*
proof−
have *A5*: *set-mset a ⊆ I*
by (*metis A2 A3 insert-iff mset-to-IP-indices*)
have *A6*: *set-mset a ⊆ I ∧ Q a ∈ carrier R*
using *A2 A5 Pring-cfs-closed* **by** *blast*
obtain *Q′* **where** *Q′-def*: $Q' = \mathit{remove\text{-}monom}\ R\ a\ Q$
by *simp*
then have $Q = Q' \bigoplus \mathit{poly\text{-}scalar\text{-}mult}\ R\ (Q\ a)\ (\mathit{mset\text{-}to\text{-}IP}\ R\ a)$
using *A2 cring.remove-monom-eq is-cring* **by** *blast*
then show *?thesis* **using** *A6 assms*
by (*metis A1 A2 A3 A4 Diff-empty Diff-insert0 Q′-def insert-Diff1 remove-monom-closed remove-monom-monomials singletonI*)
qed
qed
qed
qed
qed
show $\bigwedge Q.\ Q \in \mathit{Pring\text{-}set}\ R\ I \Longrightarrow P\ Q$
proof−
fix *Q*
assume $Q \in \mathit{Pring\text{-}set}\ R\ I$
show *P Q*
using *0*[*of monomials-of R Q*] ‹$Q \in \mathit{Pring\text{-}set}\ R\ I$› *monomials-finite*
by *blast*
qed
qed

lemma *Pring-car-induct*:
assumes $q \in \mathit{carrier}\ (\mathit{Pring}\ R\ I)$
assumes $P\ \mathbf{0}_{\mathit{Pring}\ R\ I}$
assumes $\bigwedge m\ k.\ \mathit{set\text{-}mset}\ m \subseteq I \wedge k \in \mathit{carrier}\ R \Longrightarrow P\ (k \odot_{\mathit{Pring}\ R\ I} (\mathit{mset\text{-}to\text{-}IP}$

R m))
assumes $\bigwedge$*Q m k. Q* $\in$ *carrier* (*Pring R I*) $\wedge$ (*P Q*) $\wedge$ *set-mset m* $\subseteq$ *I* $\wedge$ *k* $\in$ *carrier R*$\Longrightarrow$
P (*Q* $\bigoplus$ (*k* $\odot_{Pring\ R\ I}$ (*mset-to-IP R m*)))
shows *P q*
using *poly-monom-induction*[*of P I q*] *assms Pring-smult*[*of I*] *Pring-car*[*of I*] *Pring-zero*
by *metis*

lemma *poly-monom-induction2*:
assumes *P* (*indexed-const* **0**)
assumes $\bigwedge$*m k. set-mset m* $\subseteq$ *I* $\wedge$ *k* $\in$ *carrier R* $\Longrightarrow$ *P* (*poly-scalar-mult R k* (*mset-to-IP R m*))
assumes $\bigwedge$*Q m k. Q* $\in$ *Pring-set R I* $\wedge$ (*P Q*) $\wedge$ *set-mset m* $\subseteq$ *I* $\wedge$ *k* $\in$ *carrier R* $\Longrightarrow$ *P* (*Q* $\bigoplus$ (*poly-scalar-mult R k* (*mset-to-IP R m*)))
assumes *Q* $\in$ *Pring-set R I*
shows *P Q*
using *assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *poly-monom-induction* **by** *blast*

lemma *poly-monom-induction3*:
assumes *Q* $\in$ *Pring-set R I*
assumes *P* (*indexed-const* **0**)
assumes $\bigwedge$*m k. set-mset m* $\subseteq$ *I* $\wedge$ *k* $\in$ *carrier R* $\Longrightarrow$ *P* (*poly-scalar-mult R k* (*mset-to-IP R m*))
assumes $\bigwedge$*p q. p* $\in$ *Pring-set R I* $\Longrightarrow$ (*P p*) $\Longrightarrow$ *q* $\in$ *Pring-set R I* $\Longrightarrow$ (*P q*) $\Longrightarrow$ *P* (*p* $\bigoplus$ *q*)
shows *P Q*
apply(*rule poly-monom-induction2*[*of - I*])
using *assms*(*2*) **apply** *blast*
using *assms*(*3*) **apply** *blast*
apply (*meson assms*(*3*) *assms*(*4*) *mset-to-IP-closed poly-scalar-mult-closed*)
using *assms*(*1*) **by** *blast*

lemma *Pring-car-induct′*:
assumes *Q* $\in$ *carrier* (*Pring R I*)
assumes *P* $\mathbf{0}_{Pring\ R\ I}$
assumes $\bigwedge$*m k. set-mset m* $\subseteq$ *I* $\wedge$ *k* $\in$ *carrier R* $\Longrightarrow$ *P* (*k* $\odot_{Pring\ R\ I}$ *mset-to-IP R m*)
assumes $\bigwedge$*p q. p* $\in$ *carrier* (*Pring R I*) $\Longrightarrow$ (*P p*) $\Longrightarrow$ *q* $\in$ *carrier* (*Pring R I*) $\Longrightarrow$ (*P q*) $\Longrightarrow$ *P* (*p* $\oplus_{Pring\ R\ I}$ *q*)
shows *P Q*
using *poly-monom-induction3*[*of Q I P*] *assms Pring-smult Pring-add Pring-zero Pring-car*
by *metis*

lemma *poly-eval-mono*:
assumes *P* $\in$ *Pring-set R I*
assumes *closed-fun R g*
assumes *finite F*

assumes *monomials-reducing-to R m P S* $\subseteq$ *F*
assumes $\bigwedge n.\ n \in F \Longrightarrow$ *remove-indices n S* = *m*
shows *poly-eval R S g P m* = ($\bigoplus n \in F.$ *monom-eval R* (*restrict-to-indices n S*) *g* $\otimes$ *P n*)
proof−
have *0*: ($\bigoplus n \in F.$ *monom-eval R* (*restrict-to-indices n S*) *g* $\otimes$ *P n*)=
($\bigoplus n \in F -$ (*monomials-reducing-to R m P S*). *monom-eval R* (*restrict-to-indices n S*) *g* $\otimes$ *P n*) $\oplus$ *poly-eval R S g P m*
proof−
have *00*: ($\lambda n.$ *monom-eval R* (*restrict-to-indices n S*) *g* $\otimes$ *P n*) $\in$ *F* $\rightarrow$ *carrier R*
by (*meson Pi-I Pring-cfs-closed assms*(*1*) *assms*(*2*) *m-closed monom-eval-car*)
have *01*: *monomials-reducing-to R m P S* $\subseteq$ *F*
by (*simp add*: *assms*(*4*))
have *02*: ($\bigoplus n \in F.$ *monom-eval R* (*restrict-to-indices n S*) *g* $\otimes$ *P n*) =
($\bigoplus n \in$ *monomials-reducing-to R m P S*. *monom-eval R* (*restrict-to-indices n S*) *g* $\otimes$ *P n*) $\oplus$
($\bigoplus n \in F -$ *monomials-reducing-to R m P S*. *monom-eval R* (*restrict-to-indices n S*) *g* $\otimes$ *P n*)
using *00 01 assms*(*3*) *finsum-partition* **by** *blast*
have *03*: ($\bigoplus n \in F -$ *monomials-reducing-to R m P S*. *monom-eval R* (*restrict-to-indices n S*) *g* $\otimes$ *P n*)$\in$ *carrier R*
by (*metis* (*mono-tags*, *lifting*) *00 DiffD1 PiE Pi-I finsum-closed*)
have ($\bigoplus n \in$ *monomials-reducing-to R m P S*. *monom-eval R* (*restrict-to-indices n S*) *g* $\otimes$ *P n*) $\in$ *carrier R*
proof −
have ($\lambda m.$ *monom-eval R* (*restrict-to-indices m S*) *g* $\otimes$ *P m*) $\in$ *monomials-reducing-to R m P S* $\rightarrow$ *carrier R*
using *00 01* **by** *blast*
then show *?thesis*
using *finsum-closed* **by** *blast*
qed
then show *?thesis*
unfolding *poly-eval-def*
using *00 01 02 03 add.m-comm*
by *presburger*
qed
have ($\bigoplus n \in F -$ (*monomials-reducing-to R m P S*). *monom-eval R* (*restrict-to-indices n S*) *g* $\otimes$ *P n*) = **0**
proof−
have $\bigwedge n.\ n \in F -$ (*monomials-reducing-to R m P S*) $\Longrightarrow$ *monom-eval R* (*restrict-to-indices n S*) *g* $\otimes$ *P n* = **0**
proof−
fix *n*
assume *A*: *n* $\in$ *F* − (*monomials-reducing-to R m P S*)
have *n* $\notin$ *monomials-of R P*
proof
assume *n* $\in$ *monomials-of R P*
then have *n* $\in$ (*monomials-reducing-to R m P S*)

unfolding *monomials-reducing-to-def*
using *assms*(*5*) *A*
by *blast*
then show *False* **using** *A* **by** *blast*
qed
then show *monom-eval R* (*restrict-to-indices n S*) *g* ⊗ *P n* = **0**
by (*metis assms*(*2*) *monom-eval-car complement-of-monomials-of r-null*)
qed
then show *?thesis*
by (*meson add.finprod-one-eqI*)
qed
then have *1*: (⨁ *n*∈*F*. *monom-eval R* (*restrict-to-indices n S*) *g* ⊗ *P n*) =
0 ⊕ *poly-eval R S g P m*
using *0 Pi-I Pring-cfs-closed add.r-cancel-one assms*(*1*) *assms*(*2*)
by *presburger*
have (λ*n*. *monom-eval R* (*restrict-to-indices n S*) *g* ⊗ *P n*) ∈ *monomials-reducing-to R m P S* → *carrier R*
by (*meson Pi-I Pring-cfs-closed assms*(*1*) *assms*(*2*) *m-closed monom-eval-car*)
hence *poly-eval R S g P m* ∈ *carrier R*
using *assms*
unfolding *poly-eval-def*
using *finsum-closed*[*of* λ *n*. *monom-eval R* (*restrict-to-indices n S*) *g* ⊗ *P n*
monomials-reducing-to R m P S]
by (*meson Pi-I Pring-cfs-closed m-closed monom-eval-car*)
then have (⨁ *n*∈*F*. *monom-eval R* (*restrict-to-indices n S*) *g* ⊗ *P n*) = *poly-eval R S g P m*
using *1 add.r-cancel-one zero-closed*
by *presburger*
then show *?thesis*
by *presburger*
qed

lemma *finsum-group*:
assumes ⋀*n*. *f n* ∈ *carrier R*
assumes ⋀*n*. *g n* ∈ *carrier R*
shows *finite S* ⟹ *finsum R f S* ⊕ *finsum R g S* = *finsum R* (λ*n*. *f n* ⊕ *g n*) *S*
apply(*erule finite.induct*)
apply (*metis finsum-empty r-zero zero-closed*)
proof−
fix *A* :: *'c set*
fix *a*
assume *A0*: *finite A*
assume *A1*: *finsum R f A* ⊕ *finsum R g A* = (⨁ *n*∈*A*. *f n* ⊕ *g n*)
show *finsum R f* (*insert a A*) ⊕ *finsum R g* (*insert a A*) = (⨁ *n*∈*insert a A*. *f n* ⊕ *g n*)
proof(*cases a* ∈ *A*)
case *True*
then show *?thesis*
by (*metis A1 insert-absorb*)

```
  next
    case False
    have LHS: finsum R f (insert a A) ⊕ finsum R g (insert a A) =
                (f a ⊕ finsum R f A) ⊕ (g a ⊕ finsum R g A)
      using assms  finsum-insert[of A a f] finsum-insert[of A a g]
      by (metis A0 False Pi-I)
    have F0: (λn. f n ⊕ g n) ∈ A → carrier R
      using assms
      by blast
    have F1: (f a ⊕ g a) ∈ carrier  R
      using assms
      by blast
    have RHS:  (⨁ n∈insert a A. f n ⊕ g n) = (f a ⊕ g a) ⊕ (⨁ n∈A. f n ⊕ g n)
      using F0 F1 assms finsum-insert[of A a  (λn. f n ⊕ g n)] False A0
      by blast
   have F2:  f a ⊕ finsum R f A ⊕ (g a ⊕ finsum R g A) = (f a ⊕ g a) ⊕ (finsum
R f A ⊕  finsum R g A)
    proof−
      have F20: f a ∈ carrier R
        using assms(1) by blast
      have F21: g a ∈ carrier R
        using assms(2) by blast
      have F22: finsum R f A ∈ carrier R
        by (metis Pi-iff assms(1) finsum-closed)
      have F23: finsum R g A ∈ carrier R
        by (metis Pi-I assms(2) finsum-closed)
      show ?thesis using F21 F20 F22 F23
        using add.m-assoc add.m-closed add.m-lcomm
        by presburger
    qed
    show ?thesis
      using RHS LHS assms A1 F2
      by presburger
  qed
qed

lemma poly-eval-add:
  assumes P ∈ Pring-set R I
  assumes Q ∈ Pring-set R I
  assumes closed-fun R g
  shows poly-eval R S g (P ⨁  Q)   = poly-eval R S g P  ⨁  poly-eval R S g Q
proof
  fix m
  show poly-eval R S g (P ⨁  Q)  m = (poly-eval R S g P ⨁  poly-eval R S g Q)
m
  proof−
     obtain F where F-def: F = monomials-reducing-to R m (P ⨁  Q) S ∪
monomials-reducing-to R m P S ∪
                          monomials-reducing-to R m Q S
```

by *simp*
have *0*: *finite F*
proof −
have *00*: *finite* (*monomials-reducing-to R m* (*P* $\bigoplus$ *Q*) *S*)
using *assms*
by (*meson finite-subset monomials-finite monomials-of-add-finite monomials-reducing-to-subset*)
have *01*: *finite* (*monomials-reducing-to R m P S*)
using *assms*(*1*) *monomials-reducing-to-finite* **by** *blast*
have *02*: *finite* (*monomials-reducing-to R m Q S*)
using *assms*(*2*) *monomials-reducing-to-finite* **by** *blast*
show *?thesis*
using *F-def 00 01 02*
by *blast*
qed
have *1*: $\bigwedge n.\ n \in F \Longrightarrow$ *remove-indices n S* = *m*
proof −
fix *n*
assume *A*: *n* ∈ *F*
show *remove-indices n S* = *m*
using *F-def*
unfolding *monomials-reducing-to-def*
using *A*
by *blast*
qed
have *2*: *poly-eval R S g* (*P* $\bigoplus$ *Q*) *m* = ($\bigoplus$ *n*∈ *F*. *monom-eval R* (*restrict-to-indices n S*) *g* ⊗ (*P* $\bigoplus$ *Q*) *n*)
using *assms 0 1 poly-eval-mono*[*of P* $\bigoplus$ *Q I g F m*] *F-def indexed-pset.indexed-padd*
by *blast*
have *3*: *poly-eval R S g P m* = ($\bigoplus$ *n*∈ *F*. *monom-eval R* (*restrict-to-indices n S*) *g* ⊗ *P n*)
using *assms 0 1 poly-eval-mono*[*of P I g F m*] *F-def indexed-pset.indexed-padd*
by *blast*
have *4*: *poly-eval R S g Q m* = ($\bigoplus$ *n*∈ *F*. *monom-eval R* (*restrict-to-indices n S*) *g* ⊗ *Q n*)
using *assms 0 1 poly-eval-mono*[*of Q I g F m*] *F-def indexed-pset.indexed-padd*
by *blast*
have *5*: *poly-eval R S g P m* ⊕ *poly-eval R S g Q m* = ($\bigoplus$ *n*∈ *F*. *monom-eval R* (*restrict-to-indices n S*) *g* ⊗ *P n*
⊕ *monom-eval R* (*restrict-to-indices n S*) *g* ⊗ *Q n*)
proof −
have *50*: (λ*n*. *monom-eval R* (*restrict-to-indices n S*) *g* ⊗ *P n*) ∈ *F* → *carrier R*
by (*meson Pi-I Pring-cfs-closed assms*(*1*) *assms*(*3*) *m-closed monom-eval-car*)
have *51*: (λ*n*. *monom-eval R* (*restrict-to-indices n S*) *g* ⊗ *Q n*) ∈ *F* → *carrier R*
by (*meson Pi-I Pring-cfs-closed assms*(*2*) *assms*(*3*) *m-closed monom-eval-car*)
then show *?thesis*

using *0 2 3 50 finsum-group*[*of* (λn. *monom-eval R* (*restrict-to-indices n S*) $g \otimes P\ n$)
(λn. *monom-eval R* (*restrict-to-indices n S*) $g \otimes Q\ n$) *F*]
by (*metis* (*mono-tags*, *lifting*) *4 Pring-cfs-closed assms*(*1*) *assms*(*2*) *assms*(*3*) *m-closed monom-eval-car*)
qed
have *6*: *poly-eval R S g P m* $\oplus$ *poly-eval R S g Q m*
$= (\bigoplus n \in F.$ *monom-eval R* (*restrict-to-indices n S*) $g \otimes (P\ n \oplus Q\ n))$
proof −
have *0* : (λn. *monom-eval R* (*restrict-to-indices n S*) $g \otimes P\ n \oplus$ *monom-eval R* (*restrict-to-indices n S*) $g \otimes Q\ n) \in F \to$ *carrier R*
apply(*rule Pi-I*)
by (*meson Pring-cfs-closed add.m-closed assms*(*1*) *assms*(*2*) *assms*(*3*) *m-closed monom-eval-car*)
have *1*: (λn. *monom-eval R* (*restrict-to-indices n S*) $g \otimes (P\ n \oplus Q\ n)) \in F \to$ *carrier R*
apply(*rule Pi-I*)
by (*meson Pring-cfs-closed add.m-closed assms*(*1*) *assms*(*2*) *assms*(*3*) *m-closed monom-eval-car*)
have $\bigwedge n.\ n \in F \Longrightarrow$ *monom-eval R* (*restrict-to-indices n S*) $g \otimes (P\ n \oplus Q\ n) =$ *monom-eval R* (*restrict-to-indices n S*) $g \otimes P\ n$
$\oplus$ *monom-eval R* (*restrict-to-indices n S*) $g \otimes Q\ n$
using *assms Pring-cfs-closed cring.monom-eval-car is-cring r-distr*
by *metis*
then have ($\lambda x \in F.$ *monom-eval R* (*restrict-to-indices x S*) $g \otimes P\ x \oplus$ *monom-eval R* (*restrict-to-indices x S*) $g \otimes Q\ x$)
$= (\lambda x \in F.$ *monom-eval R* (*restrict-to-indices x S*) $g \otimes (P\ x \oplus Q\ x))$
by (*metis* (*no-types*, *lifting*) *restrict-ext*)
then show *?thesis*
using *5 finsum-eq*[*of* (λn. *monom-eval R* (*restrict-to-indices n S*) $g \otimes P\ n \oplus$ *monom-eval R* (*restrict-to-indices n S*) $g \otimes Q\ n$)
F (λn. *monom-eval R* (*restrict-to-indices n S*) $g \otimes (P\ n \oplus Q\ n))$] *0 1*
by *presburger*
qed
have *7*: *monomials-reducing-to R m* $(P \bigoplus Q)\ S \subseteq F$
using *F-def*
by *blast*
have *8*: *poly-eval R S g* $(P \bigoplus Q)\ \ m = (\bigoplus n \in F.$ *monom-eval R* (*restrict-to-indices n S*) $g \otimes (P \bigoplus Q)\ n)$
using *7 0 1 2*
by *blast*
obtain *f* **where** *f-def*: $f = (\lambda n.$ *monom-eval R* (*restrict-to-indices n S*) $g \otimes (P \bigoplus Q)\ n)$
by *blast*
obtain *h* **where** *h-def*: $h = (\lambda n.$ *monom-eval R* (*restrict-to-indices n S*) $g \otimes (P\ n \oplus Q\ n))$

by *blast*
have *9*: $f \in F \to carrier\ R$
using *f-def*
by (*metis* (*mono-tags*, *lifting*) *Pi-I Pring-cfs-closed add.m-closed assms(1) assms(2) assms(3) indexed-padd-def m-closed monom-eval-car*)
have *10*: $h \in F \to carrier\ R$
using *h-def*
by (*metis* (*mono-tags*, *lifting*) *9 Pi-cong f-def indexed-padd-def*)
have *11*: *restrict f F = restrict h F*
using *f-def h-def*
by (*metis indexed-padd-def*)
have *finsum R f F = finsum R h F*
using *9 10 11 finsum-eq*[*of f F h*]
by *blast*
then show *?thesis*
using *f-def h-def*
by (*metis* (*no-types*, *lifting*) *6 8 indexed-padd-def*)
qed
qed

lemma *poly-eval-Pring-add*:
assumes $P \in carrier\ (Pring\ R\ I)$
assumes $Q \in carrier\ (Pring\ R\ I)$
assumes *closed-fun R g*
shows *poly-eval R S g* $(P \oplus_{Pring\ R\ I} Q)$ $=$ *poly-eval R S g P* $\oplus_{Pring\ R\ I}$ *poly-eval R S g Q*
using *assms poly-eval-add*[*of P I Q g S*]
by (*metis Pring-add Pring-car*)

Closure of partial evaluation maps:

lemma(**in** *cring*) *poly-eval-closed*:
assumes *closed-fun R g*
assumes *P* ∈ *Pring-set R I*
shows *poly-eval R S g P* ∈ *Pring-set R* (*I* − *S*)
proof−
obtain *Pr* **where** *Pr-def*[*simp*]: *Pr* = (λ*Q*. *poly-eval R S g Q* ∈ *Pring-set R* (*I* − *S*))
by *blast*
have *Pr P*
apply(*rule poly-monom-induction2*[*of* - *I* -])
apply (*metis Pr-def indexed-pset.indexed-const poly-eval-constant zero-closed*)
apply (*metis Pr-def assms*(*1*) *indexed-pset.indexed-const mset-to-IP-closed poly-eval-constant poly-eval-monomial-term-closed poly-scalar-mult-zero r-null r-one*)
proof−
show *P* ∈ *Pring-set R I* **using** *assms* **by** *blast*
show ⋀*Q m k*. *Q* ∈ *Pring-set R I* ∧ *Pr Q* ∧ *set-mset m* ⊆ *I* ∧ *k* ∈ *carrier R* ⟹ *Pr* (*Q* ⨁ *poly-scalar-mult R k* (*mset-to-IP R m*))
proof−

fix Q
fix m
fix k
assume A: $Q \in$ *Pring-set* $R\ I \wedge$ *Pr* $Q \wedge$ *set-mset* $m \subseteq I \wedge k \in$ *carrier* R
then have *0*: *poly-eval* $R\ S\ g\ Q \in$ *Pring-set* $R\ (I - S)$
using *Pr-def* **by** *blast*
have *1*: *poly-scalar-mult* $R\ k$ (*mset-to-IP* $R\ m) \in$ *Pring-set* $R\ I$
using *A mset-to-IP-closed poly-scalar-mult-closed*
by *blast*
have *poly-eval* $R\ \ S\ g$ $(Q \bigoplus$ *poly-scalar-mult* $R\ k$ (*mset-to-IP* $R\ m)) =$
poly-eval $R\ S\ g\ Q \bigoplus$ *poly-eval* $R\ S\ g$ (*poly-scalar-mult* $R\ k$ (*mset-to-IP* $R\ m$))

using *assms poly-eval-add*[*of* $Q\ I$ (*poly-scalar-mult* $R\ k$ (*mset-to-IP* $R\ m$)) *g*] *1 A*
by *blast*
then show *Pr* $(Q \bigoplus$ *poly-scalar-mult* $R\ k$ (*mset-to-IP* $R\ m$))
using *Pr-def*
by (*metis 1 A assms*(*1*) *indexed-pset.indexed-padd poly-eval-monomial-term-closed poly-scalar-mult-one poly-scalar-mult-zero*)
qed
qed
then show *?thesis*
using *Pr-def*
by *blast*
qed

lemma *poly-scalar-mult-indexed-pmult*:
assumes $P \in$ *Pring-set* $R\ I$
assumes $k \in$ *carrier* R
shows *poly-scalar-mult* $R\ k$ $(P \bigotimes i) =$ (*poly-scalar-mult* $R\ k\ P$) $\bigotimes i$
proof−
have *0*: *mset-to-IP* R $\{\#i\#\} \in$ *Pring-set* R $(I \cup \{i\})$
by (*metis Un-upper2 mset-to-IP-closed set-mset-add-mset-insert set-mset-empty*)
have *1*: $P \in$ *Pring-set* R $(I \cup \{i\})$
using *Pring-carrier-subset Un-upper1 assms*(*1*) **by** *blast*
show *?thesis*
using *0 1 poly-scalar-mult-times*[*of mset-to-IP* R $\{\#i\#\}$ $I \cup \{i\}$ $P\ \ k$]
poly-index-mult[*of* $P\ I \cup \{i\}\ i$] *assms*
by (*metis Un-upper2 insert-subset poly-index-mult poly-scalar-mult-closed*)
qed

lemma *remove-indices-add-mset*:
assumes $i \notin S$
shows *remove-indices* (*add-mset* $i\ m$) $S =$ *add-mset* i (*remove-indices* $m\ S$)
apply(*induction* m)
apply (*smt assms empty-eq-union remove-indices-decomp restrict-to-indicesE single-is-union union-single-eq-member*)
by (*metis assms multi-union-self-other-eq remove-indices-decomp restrict-to-indices-add-element union-mset-add-mset-right*)

lemma *poly-eval-monom-insert*:
assumes *closed-fun R g*
assumes **1** $\neq$**0**
assumes $i \in S$
shows *poly-eval R S g* (*mset-to-IP R* (*add-mset i m*))
= *poly-scalar-mult R* (*g i*)
(*poly-eval R S g* (*mset-to-IP R m*))
proof−
have *0*: *poly-eval R S g* (*mset-to-IP R* (*add-mset i m*)) =
poly-scalar-mult R (*monom-eval R* (*restrict-to-indices* (*add-mset i m*) *S*) *g*)
(*mset-to-IP R* (*remove-indices* (*add-mset i m*) *S*))
using *assms*(*1*) *assms*(*2*) *poly-eval-monomial* **by** *blast*
have *1*: (*mset-to-IP R* (*remove-indices* (*add-mset i m*) *S*)) =
(*mset-to-IP R* (*remove-indices m S*))
using *assms*
by (*metis* (*full-types*) *Diff-iff insert-Diff1 remove-indices-def restrict-to-indices-add-element set-mset-add-mset-insert*)
have *2*: (*monom-eval R* (*restrict-to-indices* (*add-mset i m*) *S*) *g*) =
(*g i*) $\otimes$ ((*monom-eval R* (*restrict-to-indices m S*) *g*))
using *assms*
by (*metis monom-eval-add restrict-to-indices-add-element*)
have *3*: *poly-eval R S g* (*mset-to-IP R* (*add-mset i m*)) =
poly-scalar-mult R ((*g i*) $\otimes$ ((*monom-eval R* (*restrict-to-indices m S*) *g*)))
(*mset-to-IP R* (*remove-indices m S*))
using *0 1 2 assms*
by *presburger*
hence *poly-eval R S g* (*mset-to-IP R* (*add-mset i m*)) =
poly-scalar-mult R (*g i*)
(*poly-scalar-mult R* ((*monom-eval R* (*restrict-to-indices m S*) *g*))
(*mset-to-IP R* (*remove-indices m S*)))
using *assms poly-scalar-mult-iter*[*of* (*mset-to-IP R* (*remove-indices m S*))
UNIV g i monom-eval R (*restrict-to-indices m S*) *g*]
mset-to-IP-closed[*of remove-indices m S UNIV*]
by (*metis PiE UNIV-I monom-eval-car subsetI*)
thus *?thesis* **using** *assms*
by (*metis poly-eval-monomial*)
qed

lemma *poly-eval-monom-insert′*:
assumes *closed-fun R g*
assumes **1** $\neq$**0**
assumes $i \notin S$
shows *poly-eval R S g* (*mset-to-IP R* (*add-mset i m*))
= (*poly-eval R S g* (*mset-to-IP R m*)) $\bigotimes$ *i*
proof−

have *0*: *poly-eval R S g* (*mset-to-IP R* (*add-mset i m*)) =
poly-scalar-mult R (*monom-eval R* (*restrict-to-indices* (*add-mset i m*) *S*) *g*)
(*mset-to-IP R* (*remove-indices* (*add-mset i m*) *S*))
using *assms*(*1*) *assms*(*2*) *poly-eval-monomial* **by** *blast*
hence *poly-eval R S g* (*mset-to-IP R* (*add-mset i m*)) =
poly-scalar-mult R (*monom-eval R* (*restrict-to-indices m S*) *g*)
(*mset-to-IP R* (*remove-indices* (*add-mset i m*) *S*))
by (*metis assms*(*3*) *restrict-to-indices-add-element*)
hence *poly-eval R S g* (*mset-to-IP R* (*add-mset i m*)) =
poly-scalar-mult R (*monom-eval R* (*restrict-to-indices m S*) *g*)
(*mset-to-IP R* (*add-mset i* (*remove-indices m S*)))
by (*metis assms*(*3*) *remove-indices-add-mset*)
hence *poly-eval R S g* (*mset-to-IP R* (*add-mset i m*)) =
poly-scalar-mult R (*monom-eval R* (*restrict-to-indices m S*) *g*)
(*mset-to-IP R* (*remove-indices m S*) ⨂ *i*)
by (*metis monom-add-mset*)
hence *poly-eval R S g* (*mset-to-IP R* (*add-mset i m*)) =
(*poly-scalar-mult R* (*monom-eval R* (*restrict-to-indices m S*) *g*)
(*mset-to-IP R* (*remove-indices m S*))) ⨂ *i*
by (*metis assms*(*1*) *cring.monom-eval-car is-cring local.ring-axioms*
poly-scalar-mult-indexed-pmult ring.mset-to-IP-closed set-eq-subset)
thus *?thesis*
by (*metis assms*(*1*) *assms*(*2*) *poly-eval-monomial*)
qed

lemma *poly-eval-indexed-pmult-monomial*:
assumes *closed-fun R g*
assumes $k \in$ *carrier R*
assumes $i \in S$
assumes $\mathbf{1} \neq \mathbf{0}$
shows *poly-eval R S g* (*poly-scalar-mult R k* (*mset-to-IP R m*) ⨂ *i*) =
poly-scalar-mult R (*g i*) (*poly-eval R S g* (*poly-scalar-mult R k* (*mset-to-IP R m*)))
proof−
have *0*: *poly-scalar-mult R k* (*mset-to-IP R m*) ⨂ *i* =
poly-scalar-mult R k (*mset-to-IP R* (*add-mset i m*))
using *monom-add-mset*[*of i m*] *poly-scalar-mult-indexed-pmult*
by (*metis* (*no-types*, *opaque-lifting*) *assms*(*2*) *local.ring-axioms ring.mset-to-IP-closed subsetD subset-refl*)
hence *1*: *poly-eval R S g* (*poly-scalar-mult R k* (*mset-to-IP R m*) ⨂ *i*) =
poly-scalar-mult R k (*poly-eval R S g* (*mset-to-IP R* (*add-mset i m*)))
by (*metis assms*(*1*) *assms*(*2*) *cring.poly-eval-scalar-mult is-cring local.ring-axioms ring.mset-to-IP-closed subsetI*)
have *2*: *poly-scalar-mult R* (*g i*) (*poly-eval R S g* (*poly-scalar-mult R k* (*mset-to-IP R m*)))
= *poly-scalar-mult R* (*g i*)
(*poly-scalar-mult R k* (*poly-eval R S g* (*mset-to-IP R m*)))
by (*smt assms*(*1*) *assms*(*2*) *local.ring-axioms poly-eval-scalar-mult ring.mset-to-IP-closed*

subsetD subset-refl)
hence *3*: *poly-scalar-mult R* (*g i*) (*poly-eval R S g* (*poly-scalar-mult R k* (*mset-to-IP R m*)))
= *poly-scalar-mult R k*
(*poly-scalar-mult R* (*g i*) (*poly-eval R S g* (*mset-to-IP R m*)))
using *assms poly-scalar-mult-comm*[*of* (*poly-eval R S g* (*mset-to-IP R m*)) *UNIV*]
poly-eval-closed[*of g* (*mset-to-IP R m*) *UNIV*]
by (*metis UNIV-I closed-fun-simp cring.poly-scalar-mult-comm is-cring local.ring-axioms ring.mset-to-IP-closed subsetI*)
have *4*: (*poly-eval R S g* (*mset-to-IP R* (*add-mset i m*))) = (*poly-scalar-mult R* (*g i*) (*poly-eval R S g* (*mset-to-IP R m*)))
using *assms poly-eval-monom-insert*[*of g i S m*]
by *blast*
thus *?thesis*
using *1 3 poly-scalar-mult-comm assms*
by *presburger*
qed

lemma *poly-eval-indexed-pmult-monomial'*:
assumes *closed-fun R g*
assumes $k \in$ *carrier R*
assumes $i \notin S$
assumes $\mathbf{1} \neq \mathbf{0}$
shows *poly-eval R S g* (*poly-scalar-mult R k* (*mset-to-IP R m*) $\bigotimes$ *i*) =
(*poly-eval R S g* (*poly-scalar-mult R k* (*mset-to-IP R m*))) $\bigotimes$ *i*
proof−
have *poly-eval R S g* (*poly-scalar-mult R k* (*mset-to-IP R m*) $\bigotimes$ *i*) =
poly-scalar-mult R k (*poly-eval R S g* ((*mset-to-IP R m*) $\bigotimes$ *i*))
using *poly-eval-scalar-mult*[*of k g*]
by (*metis UNIV-I assms*(*1*) *assms*(*2*) *local.ring-axioms poly-scalar-mult-indexed-pmult ring.monom-add-mset ring.mset-to-IP-closed subsetI*)
hence *poly-eval R S g* (*poly-scalar-mult R k* (*mset-to-IP R m*) $\bigotimes$ *i*) =
poly-scalar-mult R k (*poly-eval R S g* (*mset-to-IP R* (*add-mset i m*)))
by (*simp add*: *monom-add-mset*)
hence *poly-eval R S g* (*poly-scalar-mult R k* (*mset-to-IP R m*) $\bigotimes$ *i*) =
(*poly-scalar-mult R k* (*poly-eval R S g* (*mset-to-IP R m*))) $\bigotimes$ *i*
using *poly-eval-monom-insert'*[*of g i S m*]
by (*smt assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *poly-eval-monomial-closed poly-scalar-mult-indexed-pmult subsetD subset-refl*)
thus *?thesis* **using** *assms poly-eval-scalar-mult*[*of k g - UNIV S*]
by (*metis UNIV-I mset-to-IP-closed subsetI*)
qed

lemma *indexed-pmult-add*:
assumes $p \in$ *Pring-set R I*
assumes $q \in$ *Pring-set R I*
shows $p \bigoplus q \bigotimes i = (p \bigotimes i) \bigoplus (q \bigotimes i)$
using *assms poly-index-mult*[*of - I* $\cup \{i\}$]

by (*smt Pring-carrier-subset Set.basic-monos*(*1*) *Un-upper1 Un-upper2 cring.axioms*(*1*) *insert-subset is-cring mk-disjoint-insert mset-to-IP-closed ring.P-ring-ldistr ring.indexed-pset.intros*(*2*) *ring.indexed-pset-in-carrier set-mset-add-mset-insert set-mset-empty*)

lemma *poly-eval-indexed-pmult*:
assumes $P \in$ *Pring-set R I*
assumes *closed-fun R g*
shows *poly-eval R S g* ($P \bigotimes i$) = (*if* $i \in S$ *then poly-scalar-mult R* (*g i*) (*poly-eval R S g P*) *else* (*poly-eval R S g P*)$\bigotimes i$)
proof(*cases* $i \in S$)
case *True*
have *poly-eval R S g* ($P \bigotimes i$) = *poly-scalar-mult R* (*g i*) (*poly-eval R S g P*)
apply(*rule poly-monom-induction3*[*of P I*])
using *assms* **apply** *blast*
apply (*metis PiE UNIV-I assms*(*2*) *indexed-pmult-zero poly-eval-constant poly-scalar-mult-const r-null zero-closed*)
apply (*meson True assms*(*2*) *poly-eval-indexed-pmult-monomial*)
apply (*smt PiE True UNIV-I assms*(*2*) *genideal-one genideal-zero indexed-pmult-zero mset-to-IP-closed poly-eval-constant poly-eval-indexed-pmult-monomial poly-scalar-mult-one poly-scalar-mult-zero singletonD*)
proof− **fix** *p q* **assume** *A*:
$p \in$ *Pring-set R I*
poly-eval R S g ($p \bigotimes i$) = *poly-scalar-mult R* (*g i*) (*poly-eval R S g p*)
$q \in$ *Pring-set R I*
poly-eval R S g ($q \bigotimes i$) = *poly-scalar-mult R* (*g i*) (*poly-eval R S g q*)
have *poly-eval R S g* ($p \bigoplus q \bigotimes i$) = *poly-eval R S g* ($p \bigotimes i$) $\bigoplus$ *poly-eval R S g* ($q \bigotimes i$)
using *assms poly-eval-add*[*of* $p \bigotimes i$ $I \cup \{i\}$ $q \bigotimes i$ *g S*] *indexed-pmult-add*[*of p I q i*]
by (*smt A*(*1*) *A*(*3*) *Pring-carrier-subset Un-insert-right Un-upper1 indexed-pset.indexed-pmult insert-iff insert-subset mk-disjoint-insert*)
hence *poly-eval R S g* ($p \bigoplus q \bigotimes i$) = *poly-scalar-mult R* (*g i*) (*poly-eval R S g p*) $\bigoplus$ *poly-scalar-mult R* (*g i*) (*poly-eval R S g q*)
using *A*
by *presburger*
hence *poly-eval R S g* ($p \bigoplus q \bigotimes i$) = *poly-scalar-mult R* (*g i*) ((*poly-eval R S g p*) $\bigoplus$ (*poly-eval R S g q*))
using *Pring-smult poly-eval-closed*[*of g*] *A Pring-add Pring-car* ‹*poly-eval R S g* ($p \bigoplus q \bigotimes i$) = *poly-eval R S g* ($p \bigotimes i$) $\bigoplus$ *poly-eval R S g* ($q \bigotimes i$)› *assms*(*1*) *assms*(*2*) *closed-fun-simp Pring-smult-r-distr*[*of g i poly-eval R S g p - poly-eval R S g q*] *Pring-add Pring-car*
by *metis*
thus *poly-eval R S g* ($p \bigoplus q \bigotimes i$) = *poly-scalar-mult R* (*g i*) (*poly-eval R S g* ($p \bigoplus q$))
by (*metis A*(*1*) *A*(*3*) *assms*(*2*) *poly-eval-add*)
qed

then show *?thesis*
using *True* **by** *presburger*
next
case *False*
have *poly-eval R S g (P ⊗ i) = (poly-eval R S g P) ⊗ i*
apply(*rule poly-monom-induction3*[*of P I*])
apply (*simp add*: *assms*(*1*))
apply (*metis indexed-pmult-zero poly-eval-constant zero-closed*)
apply (*metis False assms*(*2*) *indexed-pmult-zero inv-unique l-null mset-to-IP-closed one-closed one-mset-to-IP poly-eval-constant poly-eval-indexed-pmult-monomial′ poly-scalar-mult-zero zero-closed*)
proof−**fix** *p q* **assume** *A*:
p ∈ Pring-set R I
poly-eval R S g (p ⊗ i) = poly-eval R S g p ⊗ i
q ∈ Pring-set R I
poly-eval R S g (q ⊗ i) = poly-eval R S g q ⊗ i
have *poly-eval R S g (p ⊕ q ⊗ i) = poly-eval R S g (p ⊗ i) ⊕ poly-eval R S g (q ⊗ i)*
using *assms poly-eval-add*[*of p ⊗ i I ∪ {i} q ⊗ i g S*] *indexed-pmult-add*[*of p I q i*]
by (*smt A*(*1*) *A*(*3*) *Pring-carrier-subset Un-insert-right Un-upper1 indexed-pset.indexed-pmult insert-iff insert-subset mk-disjoint-insert*)
thus *poly-eval R S g (p ⊕ q ⊗ i) = poly-eval R S g (p ⊕ q) ⊗ i*
by (*metis A*(*1*) *A*(*2*) *A*(*3*) *A*(*4*) *assms*(*2*) *indexed-pmult-add poly-eval-add poly-eval-closed*)
qed
then show *?thesis*
by (*simp add*: *False*)
qed

lemma *poly-eval-index*:
assumes 1 ≠0
assumes *closed-fun R g*
shows *poly-eval R S g* (*mset-to-IP R* {#*i*#})= (*if i ∈ S then* (*indexed-const* (*g i*)) *else mset-to-IP R* {#*i*#})
proof−
have *0*: *poly-eval R S g* (*mset-to-IP R* {#*i*#})= *poly-scalar-mult R* (*monom-eval R* (*restrict-to-indices* {#*i*#} *S*) *g*)
(*mset-to-IP R* (*remove-indices* {#*i*#} *S*))
using *poly-eval-monomial*[*of g S* {#*i*#}] *assms*(*1*) *assms*(*2*) **by** *blast*
show *?thesis* **proof**(*cases i ∈ S*)
case *True*
then have *T0*: (*restrict-to-indices* {#*i*#} *S*) = {#*i*#}
by (*metis restrict-to-indices-add-element restrict-to-indices-submultiset add-mset-subseteq-single-iff*)
then have *T1*: (*monom-eval R* (*restrict-to-indices* {#*i*#} *S*) *g*) = (*monom-eval R* {#*i*#} *g*)
by *presburger*
then have (*monom-eval R* (*restrict-to-indices* {#*i*#} *S*) *g*) = *g i* ⊗ *monom-eval R* {#} *g*

```
    using assms monom-eval-add[of g i {#}]
    by presburger
  then have T2: (monom-eval R (restrict-to-indices {#i#} S) g)  = g i
    unfolding monom-eval-def
    using T0
    by (metis PiE UNIV-I assms(2) fold-mset-empty r-one)
  have T3: (remove-indices {#i#} S) = {#}
    by (metis Diff-iff remove-indices-indices restrict-to-indicesE
        T0 multiset-cases subset-iff union-single-eq-member)
  then have T4:  (mset-to-IP R (remove-indices {#i#} S)) = indexed-const 1
    by (metis one-mset-to-IP)
   have T5: poly-eval R S g (mset-to-IP R {#i#})= poly-scalar-mult R (g i)
(indexed-const 1)
    using 0 T2 T4
    by presburger
  then show ?thesis using True  poly-scalar-mult-const
    by (metis T2 assms(2) monom-eval-car one-closed r-one)
 next
  case False
  have F0: (restrict-to-indices {#i#} S) = {#}
    using False restrict-to-indices-def
    by (metis restrict-to-indices-add-element filter-empty-mset)
  have F1: (monom-eval R (restrict-to-indices {#i#} S) g) = 1
    using F0
    unfolding monom-eval-def
    by (metis fold-mset-empty)
  have F2: (remove-indices {#i#} S) = {#i#}
    using False
    by (metis Diff-iff remove-indices-def restrict-to-indices-add-element
        restrict-to-indices-def filter-empty-mset set-mset-single singletonI)
  have F3: (mset-to-IP R (remove-indices {#i#} S)) = mset-to-IP R {#i#}
    by (simp add: F2)
   have F4:  poly-eval R S g (mset-to-IP R {#i#})= poly-scalar-mult R 1
(mset-to-IP R {#i#})
    using 0 F1 F3
    by presburger
  show ?thesis using False
    by (metis F4 mset-to-IP-closed poly-scalar-mult-one subset-iff)
 qed
qed

lemma poly-eval-indexed-pmult':
 assumes P ∈ Pring-set R I
 assumes closed-fun R g
 assumes i ∈ I
 shows poly-eval R  S g (P ⨂p  (mset-to-IP R {#i#}))  =  poly-eval R S g P
⨂p  poly-eval R S g (mset-to-IP R {#i#})
proof(cases i ∈ S)
 case True
```

have $(P \bigotimes_p mset\text{-}to\text{-}IP\ R\ \{\#i\#\}) = (P \bigotimes i)$
using *assms poly-index-mult*
by *metis*
then have *0*: $poly\text{-}eval\ R\ S\ g\ (P \bigotimes_p mset\text{-}to\text{-}IP\ R\ \{\#i\#\}) = poly\text{-}scalar\text{-}mult\ R\ (g\ i)\ (poly\text{-}eval\ R\ S\ g\ P)$
using *True assms poly-eval-indexed-pmult*[*of P I g S i*]
by *presburger*
have *1*: $poly\text{-}eval\ R\ S\ g\ (mset\text{-}to\text{-}IP\ R\ \{\#i\#\}) = indexed\text{-}const\ (g\ i)$
using *assms True*
by (*smt PiE UNIV-I genideal-one genideal-zero indexed-pmult-zero monom-add-mset one-mset-to-IP poly-eval-constant poly-eval-index singletonD*)
then have $poly\text{-}eval\ R\ S\ g\ P \bigotimes_p poly\text{-}eval\ R\ S\ g\ (mset\text{-}to\text{-}IP\ R\ \{\#i\#\}) = poly\text{-}eval\ R\ S\ g\ P \bigotimes_p indexed\text{-}const\ (g\ i)$
by *presburger*
then have $poly\text{-}eval\ R\ S\ g\ P \bigotimes_p poly\text{-}eval\ R\ S\ g\ (mset\text{-}to\text{-}IP\ R\ \{\#i\#\}) = indexed\text{-}const\ (g\ i) \bigotimes_p poly\text{-}eval\ R\ S\ g\ P$
using *assms P-ring-mult-comm*[*of indexed-const* (*g i*) *poly-eval R S g P*]
unfolding *carrier-coeff-def*
by (*metis 1 Pring-cfs-closed cring.closed-fun-simp indexed-pset.indexed-const is-cring poly-eval-closed*)
then have $poly\text{-}eval\ R\ S\ g\ P \bigotimes_p poly\text{-}eval\ R\ S\ g\ (mset\text{-}to\text{-}IP\ R\ \{\#i\#\}) = poly\text{-}scalar\text{-}mult\ R\ (g\ i)\ (poly\text{-}eval\ R\ S\ g\ P)$
using *assms*
by (*metis 0 1* ‹$P \bigotimes_p mset\text{-}to\text{-}IP\ R\ \{\#i\#\} = P \bigotimes i$› *cring.closed-fun-simp is-cring poly-eval-closed poly-scalar-mult-eq*)
then show *?thesis* **using** *0*
by *presburger*
next
case *False*
then have *0*: $poly\text{-}eval\ R\ S\ g\ (P \bigotimes_p (mset\text{-}to\text{-}IP\ R\ \{\#i\#\})) = (poly\text{-}eval\ R\ S\ g\ P) \bigotimes i$
using *assms*
by (*metis poly-eval-indexed-pmult poly-index-mult*)
have *1*: $poly\text{-}eval\ R\ S\ g\ (mset\text{-}to\text{-}IP\ R\ \{\#i\#\}) = mset\text{-}to\text{-}IP\ R\ \{\#i\#\}$
using *False*
by (*metis assms*(*2*) *indexed-pmult-zero monom-add-mset one-closed one-mset-to-IP poly-eval-constant poly-eval-index*)
then show *?thesis*
using *0 False assms poly-eval-index*[*of g*]
by (*metis UNIV-I cring.Pring-set-restrict is-cring local.ring-axioms poly-eval-closed ring.poly-index-mult subsetI*)
qed

lemma *poly-eval-monom-mult*:
assumes $P \in Pring\text{-}set\ R\ I$
assumes *closed-fun R g*
shows $poly\text{-}eval\ R\ S\ g\ (P \bigotimes_p (mset\text{-}to\text{-}IP\ R\ m)) = poly\text{-}eval\ R\ S\ g\ P \bigotimes_p poly\text{-}eval\ R\ S\ g\ (mset\text{-}to\text{-}IP\ R\ m)$
proof(*induct m*)

case *empty*
have *0*: *mset-to-IP R* {#} = *indexed-const* **1**
using *one-mset-to-IP* **by** *blast*
then have *1*: (*P* ⨂$_p$ *mset-to-IP R* {#}) = *P*
using *assms*
by (*metis P-ring-mult-comm Pring-cfs-closed carrier-coeff-def mset-to-IP-simp mset-to-IP-simp′ one-closed one-mult-left zero-closed*)
have *2*: *poly-eval R S g* (*mset-to-IP R* {#}) = *indexed-const* **1**
by (*metis 0 one-closed poly-eval-constant*)
show *?case* **using** *0 1 2*
by (*metis assms(1) assms(2) cring.P-ring-mult-comm indexed-pset.indexed-const is-cring local.ring-axioms one-closed one-mult-left poly-eval-closed ring.indexed-pset-in-carrier set-eq-subset*)
next
case (*add x m*)
fix *x*
fix *m*
assume *A*: *poly-eval R S g* (*P* ⨂$_p$ *mset-to-IP R m*) = *poly-eval R S g P* ⨂$_p$ *poly-eval R S g* (*mset-to-IP R m*)
show *poly-eval R S g* (*P* ⨂$_p$ *mset-to-IP R* (*add-mset x m*))= *poly-eval R S g P* ⨂$_p$ *poly-eval R S g* (*mset-to-IP R* (*add-mset x m*))
proof−
obtain *J* **where** *J-def*: *J* = *I* ∪ *set-mset m* ∪{*x*}
by *blast*
have *I0*: *P* ∈ *Pring-set R J*
using *J-def assms*
by (*meson Pring-carrier-subset Un-upper1 subsetD*)
have *I1*: *set-mset m* ⊆ *J*
using *J-def* **by** *blast*
have *I2*: *x* ∈ *J*
using *J-def* **by** *blast*
have *mset-to-IP R* (*add-mset x m*)= (*mset-to-IP R m*) ⨂ *x*
by (*simp add*: *monom-add-mset*)
then have (*P* ⨂$_p$ *mset-to-IP R* (*add-mset x m*)) = *P* ⨂$_p$((*mset-to-IP R m*) ⨂ *x*)
by *simp*
then have (*P* ⨂$_p$ *mset-to-IP R* (*add-mset x m*)) = *P* ⨂$_p$((*mset-to-IP R m*) ⨂$_p$ (*mset-to-IP R* {#*x*#}))
by (*metis add-mset-add-single monom-mult*)
then have *I3*: (*P* ⨂$_p$ *mset-to-IP R* (*add-mset x m*)) = (*P* ⨂$_p$(*mset-to-IP R m*)) ⨂$_p$ (*mset-to-IP R* {#*x*#})
by (*metis I0 P-ring-mult-assoc indexed-pset-in-carrier mset-to-IP-closed set-eq-subset*)
have *poly-eval R S g* (*P* ⨂$_p$ *mset-to-IP R m* ⨂$_p$ *mset-to-IP R* {#*x*#}) = *poly-eval R S g* (*P* ⨂$_p$ *mset-to-IP R m*) ⨂$_p$ *poly-eval R S g* (*mset-to-IP R* {#*x*#})
using *poly-eval-indexed-pmult′*[*of* (*P* ⨂$_p$(*mset-to-IP R m*)) *J g x*] *I0 I1 I2 assms(2) assms(1) mset-to-IP-mult-closed*
by *blast*
then have *poly-eval R S g* (*P* ⨂$_p$ *mset-to-IP R* (*add-mset x m*)) =

poly-eval R S g $(P \bigotimes_p$ *(mset-to-IP R m)) $\bigotimes_p$ *poly-eval R S g (mset-to-IP R {#x#})*
using *I3*
by *simp*
then have *poly-eval R S g (P* $\bigotimes_p$ *mset-to-IP R (add-mset x m)) = poly-eval R S g P* $\bigotimes_p$ *poly-eval R S g (mset-to-IP R m)* $\bigotimes_p$ *poly-eval R S g (mset-to-IP R {#x#})*
by (*simp add*: *A*)
then have *poly-eval R S g (P* $\bigotimes_p$ *mset-to-IP R (add-mset x m)) = poly-eval R S g P* $\bigotimes_p$ *(poly-eval R S g (mset-to-IP R m)* $\bigotimes_p$ *poly-eval R S g (mset-to-IP R {#x#}))*
using *P-ring-mult-assoc*[*of poly-eval R S g P poly-eval R S g (mset-to-IP R m) poly-eval R S g (mset-to-IP R {#x#})*]
by (*metis assms*(*1*) *assms*(*2*) *indexed-pset-in-carrier mset-to-IP-closed poly-eval-closed set-eq-subset*)
then have *poly-eval R S g (P* $\bigotimes_p$ *mset-to-IP R (add-mset x m)) = poly-eval R S g P* $\bigotimes_p$ *(poly-eval R S g ((mset-to-IP R m)* $\bigotimes_p$ *(mset-to-IP R {#x#})))*
by (*metis I1 I2 assms*(*2*) *mset-to-IP-closed poly-eval-indexed-pmult*′)
then show *?thesis*
by (*metis add-mset-add-single monom-mult*)
qed
qed

abbreviation *mon-term* (‹*Mt*›) **where**
Mt k m ≡ *poly-scalar-mult R k (mset-to-IP R m)*

lemma *poly-eval-monom-term-mult*:
assumes *P* ∈ *Pring-set R I*
assumes *closed-fun R g*
assumes *k* ∈ *carrier R*
shows *poly-eval R S g (P* $\bigotimes_p$ *(Mt k m)) = poly-eval R S g P* $\bigotimes_p$ *poly-eval R S g (Mt k m)*
proof−
obtain *J* **where** *J-def*: *J* = *I* ∪ (*set-mset m*)
by *blast*
have *J0*: *P* ∈ *Pring-set R J*
using *J-def Pring-carrier-subset Un-upper1 assms*(*1*) **by** *blast*
have *J1*: *mset-to-IP R m* ∈ *Pring-set R J*
by (*metis J-def Un-upper2 mset-to-IP-closed*)
have *0*: *(P* $\bigotimes_p$ *(Mt k m)) = poly-scalar-mult R k (P* $\bigotimes_p$ *mset-to-IP R m)*
using *times-poly-scalar-mult*[*of P J mset-to-IP R m k*] *J0 J1 assms*(*3*)
by *blast*
have *1*: *poly-eval R S g (P* $\bigotimes_p$ *(Mt k m)) = poly-scalar-mult R k (poly-eval R S g (P* $\bigotimes_p$ *mset-to-IP R m))*
by (*metis 0 J0 J1 P-ring-mult-closed*′ *assms*(*2*) *assms*(*3*) *cring.poly-eval-scalar-mult is-cring*)
have *2*: *poly-eval R S g (P* $\bigotimes_p$ *(Mt k m)) = poly-scalar-mult R k ((poly-eval R S g P)* $\bigotimes_p$ *(poly-eval R S g (mset-to-IP R m)))*

by (*metis 1 assms*(*1*) *assms*(*2*) *poly-eval-monom-mult*)
have *3*: *poly-eval R S g* (*P* ⨂$_p$ (*Mt k m*)) = (*poly-eval R S g P*) ⨂$_p$ *poly-scalar-mult R k* (*poly-eval R S g* (*mset-to-IP R m*))
by (*metis 0 2 J0 J1 assms*(*2*) *assms*(*3*) *poly-eval-closed times-poly-scalar-mult*)
then show *?thesis*
by (*metis J1 assms*(*3*) *assms*(*2*) *poly-eval-scalar-mult*)
qed

lemma *poly-eval-mult*:
assumes *P* ∈ *Pring-set R I*
assumes *Q* ∈ *Pring-set R I*
assumes *closed-fun R g*
shows *poly-eval R S g* (*P* ⨂$_p$ *Q*) = *poly-eval R S g P* ⨂$_p$ *poly-eval R S g Q*
proof−
obtain *Pr* **where** *Pr-def*: *Pr* = (λ*Q*. *poly-eval R S g* (*P* ⨂$_p$ *Q*) = *poly-eval R S g P* ⨂$_p$ *poly-eval R S g Q*)
by *blast*
have *Pr Q*
proof(*rule poly-monom-induction2*[*of - I*])
show *Q* ∈ *Pring-set R I*
by (*simp add*: *assms*(*2*))
show *Pr* (*indexed-const* **0**)
proof−
have *0*: (*P* ⨂$_p$ *indexed-const* **0**) = *indexed-const* **0**
by (*metis assms*(*1*) *cring.P-ring-mult-comm indexed-pset.indexed-const indexed-pset-in-carrier is-cring poly-scalar-mult-eq poly-scalar-mult-zero set-eq-subset zero-closed*)
have *1*: *poly-eval R S g* (*indexed-const* **0**) = *indexed-const* **0**
using *poly-eval-constant* **by** *blast*
have *2*: *poly-eval R S g P* ⨂$_p$ *poly-eval R S g* (*indexed-const* **0**) = *indexed-const* **0**
using *1*
by (*metis 0 assms*(*1*) *assms*(*3*) *local.ring-axioms one-closed poly-eval-monom-term-mult poly-scalar-mult-const r-one ring.one-mset-to-IP zero-closed*)
have *3*: *poly-eval R S g* (*P* ⨂$_p$ *indexed-const* **0**) = *indexed-const* **0**
using *0 1* **by** *presburger*
show *?thesis*
using *2 3 Pr-def*
by *presburger*
qed
show ⋀*m k*. *set-mset m* ⊆ *I* ∧ *k* ∈ *carrier R* ⟹ *Pr* (*poly-scalar-mult R k* (*mset-to-IP R m*))
using *Pr-def assms*(*1*) *assms*(*3*) *poly-eval-monom-term-mult* **by** *blast*
show ⋀*Q m k*. *Q* ∈ *Pring-set R I* ∧ *Pr Q* ∧ *set-mset m* ⊆ *I* ∧ *k* ∈ *carrier R* ⟹ *Pr* (*Q* ⨁ *Mt k m*)
proof−
fix *Q*
fix *m*
fix *k*

assume *A*: $Q \in$ *Pring-set R I* $\wedge$ *Pr Q* $\wedge$ *set-mset m* $\subseteq I \wedge k \in$ *carrier R*
show *Pr* $(Q \bigoplus Mt\ k\ m)$
proof−
have *0*: *poly-eval R S g* $(P \bigotimes_p Q)$ = *poly-eval R S g P* $\bigotimes_p$ *poly-eval R S g Q*
using *A Pr-def* **by** *blast*
have *1*: *poly-eval R S g* $(P \bigotimes_p (Mt\ k\ m))$ = *poly-eval R S g P* $\bigotimes_p$ *poly-eval R S g* $(Mt\ k\ m)$
using *A assms*(*1*) *assms*(*3*) *poly-eval-monom-term-mult*
by *blast*
have *2*: $P \bigotimes_p (Q \bigoplus Mt\ k\ m) = (P \bigotimes_p Q) \bigoplus (P \bigotimes_p (Mt\ k\ m))$
by (*meson A assms*(*1*) *local.ring-axioms mset-to-IP-closed poly-scalar-mult-closed ring.P-ring-rdistr ring.indexed-pset-in-carrier set-eq-subset*)
have *3*: *poly-eval R S g* $(P \bigotimes_p (Q \bigoplus Mt\ k\ m))$ = *poly-eval R S g* $(P \bigotimes_p Q) \bigoplus$ *poly-eval R S g* $(P \bigotimes_p (Mt\ k\ m))$
by (*metis 2 A P-ring-mult-closed*′ *assms*(*1*) *assms*(*3*) *mset-to-IP-closed poly-eval-add poly-scalar-mult-closed*)
have *4*: *poly-eval R S g* $(P \bigotimes_p (Q \bigoplus Mt\ k\ m)) =$ (*poly-eval R S g P* $\bigotimes_p$ *poly-eval R S g Q*) $\bigoplus$ (*poly-eval R S g P* $\bigotimes_p$ *poly-eval R S g* $(Mt\ k\ m)$)
by (*simp add*: *0 1 3*)
have *5*: *poly-eval R S g P* $\bigotimes_p$ (*poly-eval R S g Q* $\bigoplus$ *poly-eval R S g* $(Mt\ k\ m)$) = *poly-eval R S g P* $\bigotimes_p$ *poly-eval R S g Q* $\bigoplus$ (*poly-eval R S g P* $\bigotimes_p$ *poly-eval R S g* $(Mt\ k\ m)$)
using *ring.P-ring-rdistr*[*of R* (*poly-eval R S g P*) (*poly-eval R S g Q*) (*poly-eval R S g* $(Mt\ k\ m)$)]
by (*meson A assms*(*1*) *assms*(*3*) *indexed-pset-in-carrier local.ring-axioms poly-eval-closed poly-scalar-mult-closed ring.mset-to-IP-closed subset-refl*)
have *6*: *poly-eval R S g* $(P \bigotimes_p (Q \bigoplus Mt\ k\ m)) =$ (*poly-eval R S g P*) $\bigotimes_p$ ((*poly-eval R S g Q*) $\bigoplus$ (*poly-eval R S g* $(Mt\ k\ m)$))
using *4 5*
by *simp*
have *7*: *poly-eval R S g* $(P \bigotimes_p (Q \bigoplus Mt\ k\ m)) =$ (*poly-eval R S g P*) $\bigotimes_p$ (*poly-eval R S g* $(Q \bigoplus (Mt\ k\ m))$)
using *6 poly-eval-add*[*of* (*poly-eval R S g Q*) *I* $(Mt\ k\ m)$ *g*]
by (*metis A assms*(*3*) *mset-to-IP-closed poly-eval-add poly-scalar-mult-closed*)
show *?thesis* **using** *7*
using *Pr-def* **by** *blast*
qed
qed
qed
then show *?thesis*
using *Pr-def* **by** *blast*
qed

lemma *poly-eval-Pring-mult*:
assumes *P* $\in$ *Pring-set R I*
assumes *Q* $\in$ *Pring-set R I*

assumes *closed-fun R g*
shows *poly-eval R S g* ($P \otimes_{Pring\ R\ I} Q$) $=$ *poly-eval R S g P* $\otimes_{Pring\ R\ I}$ *poly-eval R S g Q*
by (*metis Pring-mult assms*(*1*) *assms*(*2*) *assms*(*3*) *poly-eval-mult*)

lemma *poly-eval-smult*:
assumes $P \in$ *Pring-set R I*
assumes $a \in$ *carrier R*
assumes *closed-fun R g*
shows *poly-eval R S g* ($a \odot_{Pring\ R\ I} P$) $=a \odot_{Pring\ R\ I}$ *poly-eval R S g P*
using *poly-eval-scalar-mult assms*
by (*metis Pring-smult*)

2.11.6 Partial Evaluation is a Homomorphism

lemma *poly-eval-ring-hom*:
assumes $I \subseteq J$
assumes *closed-fun R g*
assumes $J - S \subseteq I$
shows *ring-hom-ring* (*Pring R J*) (*Pring R I*) (*poly-eval R S g*)
apply(*rule ring-hom-ringI*)
apply (*simp add*: *Pring-is-ring*)
apply (*simp add*: *Pring-is-ring*)
apply (*metis* (*full-types*) *Pring-car Pring-carrier-subset assms*(*2*) *assms*(*3*) *poly-eval-closed subset-iff*)
apply (*metis Pring-car assms*(*2*) *local.ring-axioms poly-eval-mult ring.Pring-mult*)
apply (*metis Pring-add Pring-car assms*(*2*) *poly-eval-add*)
by (*metis Pring-one one-closed poly-eval-constant*)

`poly_eval` R at the zero function is an inverse to the inclusion of polynomial rings

lemma *poly-eval-zero-function*:
assumes $g = (\lambda n.\ \mathbf{0})$
assumes $J - S = I$
shows $P \in$ *Pring-set R I* $\Longrightarrow$ *poly-eval R S g P* $= P$
apply(*erule indexed-pset.induct*)
using *poly-eval-constant* **apply** *blast*
using *assms poly-eval-add*[*of - I - g S*] *zero-closed*
apply (*metis Pi-I*)
using *Diff-iff assms*(*1*) *assms*(*2*) *Pi-I*[*of UNIV g*]
poly-eval-indexed-pmult set-eq-subset subsetD zero-closed
by *smt*

lemma *poly-eval-eval-function-eq*:
assumes *closed-fun R g*
assumes *closed-fun R g′*
assumes *restrict g S* $=$ *restrict g′ S*
assumes $P \in$ *Pring-set R I*
shows *poly-eval R S g P* $=$ *poly-eval R S g′ P*

apply(*rule indexed-pset.induct*[*of P I carrier R*])
apply (*simp add*: *assms*(*4*))
apply (*metis poly-eval-constant*)
apply (*metis assms*(*1*) *assms*(*2*) *poly-eval-add*)
proof− **fix** *P i* **assume** *A*: *P* ∈ *Pring-set R I poly-eval R S g P* = *poly-eval R S g*′ *P i* ∈ *I*
show *poly-eval R S g* (*P* ⨂ *i*) = *poly-eval R S g*′ (*P* ⨂ *i*)
proof(*cases i* ∈ *S*)
case *True*
then have *g i* = *g*′ *i*
using *assms*
unfolding *restrict-def*
by *meson*
then show *?thesis*
using *assms A poly-eval-indexed-pmult*[*of P I g S i*] *poly-eval-indexed-pmult*[*of P I g*′ *S i*]
by *presburger*
next
case *False*
then show *?thesis*
using *assms A poly-eval-indexed-pmult*[*of P I g S i*] *poly-eval-indexed-pmult*[*of P I g*′ *S i*]
by *presburger*
qed
qed

lemma *poly-eval-eval-set-eq*:
assumes *closed-fun R g*
assumes *S* ∩ *I* = *S*′ ∩ *I*
assumes *P* ∈ *Pring-set R I*
assumes 1 ≠**0**
shows *poly-eval R S g P* = *poly-eval R S*′ *g P*
apply(*rule indexed-pset.induct*[*of P I carrier R*])
apply (*simp add*: *assms*(*3*))
apply (*metis poly-eval-constant*)
apply (*metis assms*(*1*) *poly-eval-add*)
proof− **fix** *P i*
assume *A*: *P* ∈ *Pring-set R I poly-eval R S g P* = *poly-eval R S*′ *g P i* ∈ *I*
show *poly-eval R S g* (*P* ⨂ *i*) = *poly-eval R S*′ *g* (*P* ⨂ *i*)
using *assms poly-eval-index*[*of g - i*] *A*
by (*metis Diff-Diff-Int Diff-iff poly-eval-indexed-pmult*)
qed

lemma *poly-eval-trivial*:
assumes *closed-fun R g*
assumes *P* ∈ *Pring-set R* (*I* − *S*)
shows *poly-eval R S g P* = *P*
apply(*rule indexed-pset.induct*[*of P I* − *S carrier R*])
apply (*simp add*: *assms*(*2*))

using *poly-eval-constant* **apply** *blast*
apply (*metis assms*(*1*) *poly-eval-add*)
by (*metis Diff-iff assms*(*1*) *poly-eval-indexed-pmult*)

2.11.7 Total Evaluation of a Polynomial

lemma *zero-fun-closed*:
closed-fun R (*λn*. **0**)
by *blast*

lemma *deg-zero-cf-eval*:
shows *P* ∈ *Pring-set R I* ⟹ *poly-eval R I* (*λn*. **0**) *P* = *indexed-const* (*P* {#})
apply(*erule indexed-pset.induct*)
apply (*metis indexed-const-def poly-eval-constant*)
proof−
show ⋀*P Q*. *P* ∈ *Pring-set R I* ⟹
poly-eval R I (*λn*. **0**) *P* = *indexed-const* (*P* {#}) ⟹
Q ∈ *Pring-set R I* ⟹ *poly-eval R I* (*λn*. **0**) *Q* = *indexed-const* (*Q* {#})
⟹ *poly-eval R I* (*λn*. **0**) (*P* ⊕ *Q*) = *indexed-const* ((*P* ⊕ *Q*) {#})
proof−
fix *P*
fix *Q*
assume *A*: *P* ∈ *Pring-set R I*
assume *B*: *poly-eval R I* (*λn*. **0**) *P* = *indexed-const* (*P* {#})
assume *C*: *Q* ∈ *Pring-set R I*
assume *D*: *poly-eval R I* (*λn*. **0**) *Q* = *indexed-const* (*Q* {#})
have *indexed-const* ((*P* ⊕ *Q*) {#}) = *indexed-const* (*P* {#}) ⊕ *indexed-const*
(*Q* {#})
by (*metis indexed-padd-const indexed-padd-def*)
thus *poly-eval R I* (*λn*. **0**) (*P* ⊕ *Q*) = *indexed-const* ((*P* ⊕ *Q*) {#})
using *A B C D poly-eval-add*[*of P I Q λn*. **0** *I*]
by (*smt zero-fun-closed*)
qed
show ⋀*P i*. *P* ∈ *Pring-set R I* ⟹ *poly-eval R I* (*λn*. **0**) *P* = *indexed-const* (*P*
{#}) ⟹ *i* ∈ *I* ⟹ *poly-eval R I* (*λn*. **0**) (*P* ⊗ *i*) = *indexed-const* ((*P* ⊗ *i*)
{#})
proof−
fix *P*
fix *i*
assume *A*: *P* ∈ *Pring-set R I poly-eval R I* (*λn*. **0**) *P* = *indexed-const* (*P* {#})
i ∈ *I*
show *poly-eval R I* (*λn*. **0**) (*P* ⊗ *i*) = *indexed-const* ((*P* ⊗ *i*) {#})
proof−
have *poly-eval R I* (*λn*. **0**) (*P* ⊗ *i*) = *indexed-const* **0**
using *A*(*1*) *A*(*3*) *cring.poly-eval-constant is-cring poly-eval-indexed-pmult*
poly-eval-scalar-mult poly-scalar-mult-zero zero-closed
by (*metis Pi-I*)
have (*P* ⊗ *i*) {#} = **0**
using *indexed-pmult-def*

```
      by (metis empty-iff set-mset-empty)
    then show ?thesis
      using ‹poly-eval R I (λn. 0) (P ⨂ i) = indexed-const 0›
      by presburger
  qed
 qed
qed

lemma deg-zero-cf-mult:
 assumes P ∈ Pring-set R I
 assumes Q ∈ Pring-set R I
 shows (P ⨂p Q) {#} = P {#} ⊗ Q {#}
proof−
 have poly-eval R I (λn. 0) (P ⨂p Q) = poly-eval R I (λn. 0) P ⨂p poly-eval
R I (λn. 0) Q
   using zero-fun-closed assms
   by (metis poly-eval-mult)
 then have 0: indexed-const ((P ⨂p Q) {#}) = (indexed-const (P {#})) ⨂p
(indexed-const (Q {#}))
  by (metis P-ring-mult-closed' Pring-cfs-closed assms(1) assms(2) deg-zero-cf-eval
indexed-const-P-mult-eq indexed-const-def)
 have indexed-const ((P ⨂p Q) {#}) = (indexed-const ((P {#}) ⊗(Q {#})))
   apply(rule ccontr)
   using 0
   by (metis Pring-cfs-closed assms(1) assms(2) indexed-const-P-mult-eq)
 then show ?thesis
 proof −
   show ?thesis
     by (metis (no-types) ‹indexed-const ((P ⨂p Q) {#}) = indexed-const (P
{#} ⊗ Q {#})›
      local.ring-axioms ring.indexed-const-def)
 qed
qed

definition deg-zero-cf :: ('a, 'c) mvar-poly ⇒ 'a where
deg-zero-cf P = P {#}

lemma deg-zero-cf-ring-hom:
 shows ring-hom-ring (Pring R I) R (deg-zero-cf)
 apply(rule ring-hom-ringI)
 using Pring-is-ring apply blast
 apply (simp add: local.ring-axioms)
 apply (metis deg-zero-cf-def Pring-car Pring-cfs-closed)
 apply (metis deg-zero-cf-def Pring-car Pring-mult deg-zero-cf-mult)
 apply (metis deg-zero-cf-def Pring-add indexed-padd-def)
 by (metis deg-zero-cf-def Pring-one indexed-const-def)

end
```

definition *eval-in-ring* ::
('a,'b) *ring-scheme* ⇒ 'c *set* ⇒ ('c ⇒ 'a) ⇒ ('a, 'c) *mvar-poly* ⇒ 'a **where**
eval-in-ring R S g P = (*poly-eval R S g P*) {#}

definition *total-eval* ::
('a,'b) *ring-scheme* ⇒ ('c ⇒ 'a) ⇒ ('a, 'c) *mvar-poly* ⇒ 'a **where**
total-eval R g P = *eval-in-ring R UNIV g P*

context *cring*
begin

lemma *eval-in-ring-ring-hom*:
assumes *closed-fun R g*
shows *ring-hom-ring* (*Pring R I*) *R* (*eval-in-ring R S g*)
unfolding *eval-in-ring-def*
apply(*rule ring-hom-ringI*)
apply (*simp add*: *Pring-is-ring*)
apply (*simp add*: *local.ring-axioms*)
using *Pring-car Pring-carrier-coeff' assms poly-eval-closed*
apply (*metis*)
using *Pring-mult*[*of I*] *poly-eval-mult*[*of - I - g S*] *poly-eval-closed*[*of g - I S*] *deg-zero-cf-mult*[*of - I*] *assms*
Pring-car[*of I*]
apply (*metis deg-zero-cf-mult*)

using *Pring-add*[*of I*] *poly-eval-add*[*of - I - g S*] *Pring-car*[*of I*] *assms indexed-padd-def*
apply *metis*
unfolding *deg-zero-cf-def*
by (*metis Pring-one indexed-const-def one-closed poly-eval-constant*)

lemma *eval-in-ring-smult*:
assumes *P* ∈ *carrier* (*Pring R I*)
assumes *a* ∈ *carrier R*
assumes *closed-fun R g*
shows *eval-in-ring R S g* (*a* ⊙$_{Pring\ R\ I}$ *P*) = *a* ⊗ *eval-in-ring R S g P*
using *assms* **unfolding** *eval-in-ring-def*
by (*smt Pring-car Pring-smult poly-eval-scalar-mult poly-scalar-mult-def*)

lemma *total-eval-ring-hom*:
assumes *closed-fun R g*
shows *ring-hom-ring* (*Pring R I*) *R* (*total-eval R g*)
using *assms* **unfolding** *total-eval-def*
using *eval-in-ring-ring-hom* **by** *blast*

lemma *total-eval-smult*:
assumes *P* ∈ *carrier* (*Pring R I*)
assumes *a* ∈ *carrier R*

assumes *closed-fun R g*
shows *total-eval R g* ($a \odot_{Pring\ R\ I} P$) = $a \otimes$ *total-eval R g P*
using *assms* **unfolding** *total-eval-def*
using *eval-in-ring-smult* **by** *blast*

lemma *total-eval-const*:
assumes $k \in$ *carrier R*
shows *total-eval R g* (*indexed-const k*) = *k*
unfolding *total-eval-def eval-in-ring-def*
using *assms*
by (*metis indexed-const-def poly-eval-constant*)

lemma *total-eval-var*:
assumes *closed-fun R g*
shows (*total-eval R g* (*mset-to-IP R* {#*i*#})) = *g i*
unfolding *total-eval-def eval-in-ring-def*
using *UNIV-I assms indexed-const-def indexed-pmult-zero monom-add-mset one-closed one-mset-to-IP one-zeroD poly-eval-constant poly-eval-index singletonD*
by (*smt PiE iso-tuple-UNIV-I monom-eval-add monom-eval-car mset-to-IP-simp poly-scalar-mult-const r-one*)

lemma *total-eval-indexed-pmult*:
assumes *P* $\in$ *carrier* (*Pring R I*)
assumes *i* $\in$ *I*
assumes *closed-fun R g*
shows *total-eval R g* (*P* $\bigotimes$ *i*) = *total-eval R g P* $\otimes_R$ *g i*
proof−
have *P* $\bigotimes$ *i* = *P* $\bigotimes_p$ *mset-to-IP R* ((*add-mset i*) {#})
using *assms poly-index-mult*[*of P I i*] *Pring-car*
by *blast*
then have *0*: (*P* $\bigotimes$ *i*) = *P* $\otimes_{Pring\ R\ I}$ (*mset-to-IP R* {#*i*#})
by (*simp add*: *Pring-mult*)
then have *total-eval R g* (*P* $\bigotimes$ *i*) = (*total-eval R g P*) $\otimes_R$ (*total-eval R g* (*mset-to-IP R* {#*i*#}))
proof−
have (*mset-to-IP R* {#*i*#}) $\in$ *carrier* (*Pring R I*)
by (*metis Pring-car assms*(*2*) *mset-to-IP-closed set-mset-single singletonD subset-iff*)
then show *?thesis*
using *assms total-eval-ring-hom*[*of g I*] *ring-hom-mult*
by (*metis 0 ring-hom-ring.homh*)
qed
then show *?thesis*
by (*metis assms*(*3*) *cring.total-eval-var is-cring*)
qed

lemma *total-eval-mult*:
assumes *P* $\in$ *carrier* (*Pring R I*)
assumes *Q* $\in$ *carrier* (*Pring R I*)

assumes *closed-fun R g*
shows *total-eval R g* ($P \otimes_{Pring\ R\ I} Q$) = (*total-eval R g P*) $\otimes_R$(*total-eval R g Q*)
by (*metis assms*(*1*) *assms*(*2*) *assms*(*3*) *ring-hom-mult ring-hom-ring.homh total-eval-ring-hom*)

lemma *total-eval-add*:
assumes $P \in$ *carrier* (*Pring R I*)
assumes $Q \in$ *carrier* (*Pring R I*)
assumes *closed-fun R g*
shows *total-eval R g* ($P \oplus_{Pring\ R\ I} Q$) = (*total-eval R g P*) $\oplus_R$(*total-eval R g Q*)
by (*metis assms*(*1*) *assms*(*2*) *assms*(*3*) *ring-hom-add ring-hom-ring.homh total-eval-ring-hom*)

lemma *total-eval-one*:
assumes *closed-fun R g*
shows *total-eval R g* $\mathbf{1}_{Pring\ R\ I} = \mathbf{1}$
by (*metis Pring-one one-closed total-eval-const*)

lemma *total-eval-zero*:
assumes *closed-fun R g*
shows *total-eval R g* $\mathbf{0}_{Pring\ R\ I} = \mathbf{0}$
by (*metis Pring-zero total-eval-const zero-closed*)

lemma *total-eval-closed*:
assumes $P \in$ *carrier* (*Pring R I*)
assumes *closed-fun R g*
shows *total-eval R g P* $\in$ *carrier R*
using *assms total-eval-ring-hom*[*of g*]
by (*metis ring-hom-closed ring-hom-ring.homh*)

2.12 Constructing Homomorphisms from Indexed Polynomial Rings and a Universal Property

The inclusion of R into its polynomial ring

lemma *indexed-const-ring-hom*:
ring-hom-ring R (*Pring R I*) (*indexed-const*)
apply(*rule ring-hom-ringI*)
apply (*simp add*: *local.ring-axioms*)
apply (*simp add*: *Pring-is-ring*)
using *Pring-car indexed-pset.indexed-const* **apply** *blast*
apply (*metis Pring-mult indexed-const-P-mult-eq*)
apply (*metis indexed-padd-const local.ring-axioms ring.Pring-add*)
by (*metis Pring-one*)

lemma *indexed-const-inj-on*:
inj-on (*indexed-const*) (*carrier R*)

by (*metis cring.total-eval-const inj-onI is-cring*)

end

2.12.1 Mapping $R[x] \to S[x]$ along a homomorphism $R \to S$

definition *ring-hom-to-IP-ring-hom* ::
('*a*, '*e*) *ring-hom* ⇒ ('*a*, '*c*) *mvar-poly* ⇒ '*c monomial* ⇒ '*e* **where**
ring-hom-to-IP-ring-hom φ *P m* = φ (*P m*)

context *cring*
begin

lemma *ring-hom-to-IP-ring-hom-one*:
assumes *cring S*
assumes *ring-hom-ring R S* φ
shows *ring-hom-to-IP-ring-hom* φ $\mathbf{1}_{Pring\ R\ I}$ = $\mathbf{1}_{Pring\ S\ I}$
unfolding *ring-hom-to-IP-ring-hom-def*
proof
fix *m*
show φ ($\mathbf{1}_{Pring\ R\ I}$ *m*) = $\mathbf{1}_{Pring\ S\ I}$ *m*
proof(*cases m* = {#})
case *True*
then have ($\mathbf{1}_{Pring\ R\ I}$ *m*) = $\mathbf{1}_R$
by (*metis Pring-one indexed-const-def*)
then have φ ($\mathbf{1}_{Pring\ R\ I}$ *m*) = $\mathbf{1}_S$
using *assms*
unfolding *ring-hom-ring-def*
by (*metis assms*(*2*) *ring-hom-one ring-hom-ring.homh*)
then show *?thesis* **using** *assms True ring.indexed-const-def ring-hom-ring-def*
by (*metis ring.Pring-one*)
next
case *False*
then have ($\mathbf{1}_{Pring\ R\ I}$ *m*) = $\mathbf{0}_R$
by (*metis indexed-const-def local.ring-axioms ring.Pring-one*)

then have φ ($\mathbf{1}_{Pring\ R\ I}$ *m*) = $\mathbf{0}_S$
using *assms*
unfolding *ring-hom-ring-def cring-def*
by (*metis assms*(*2*) *ring-hom-zero ring-hom-ring.homh*)
then show *?thesis* **using** *assms False ring.indexed-const-def ring-hom-ring-def*
by (*metis ring.Pring-one*)
qed
qed

lemma *ring-hom-to-IP-ring-hom-constant*:
assumes *cring S*
assumes *ring-hom-ring R S* φ
assumes *a* ∈ *carrier R*

shows *ring-hom-to-IP-ring-hom* φ ((*indexed-const* a):: $'c$ *monomial* $\Rightarrow$ $'a$) = *ring.indexed-const* S (φ a)
unfolding *ring-hom-to-IP-ring-hom-def indexed-const-def*
proof
fix m:: $'c$ *monomial*
show φ (*if* $m = \{\#\}$ *then* a *else* $\mathbf{0}$) = *ring.indexed-const* S (φ a) m
apply(*cases* $m = \{\#\}$)
apply (*simp add*: *assms*(*1*) *cring.axioms*(*1*) *ring.indexed-const-def*)
proof−
assume $m \neq \{\#\}$
then have φ (*if* $m = \{\#\}$ *then* a *else* $\mathbf{0}$) = $\mathbf{0}_S$
by (*metis* (*full-types*) *assms*(*1*) *assms*(*2*) *cring-def local.ring-axioms ring-hom-ring.homh ring-hom-zero*)
then show φ (*if* $m = \{\#\}$ *then* a *else* $\mathbf{0}$) = *ring.indexed-const* S (φ a) m
using *assms*
by (*metis* ‹$m \neq \{\#\}$› *cring.axioms*(*1*) *ring.indexed-const-def*)
qed
qed

lemma *ring-hom-to-IP-ring-hom-add*:
assumes *cring* S
assumes *ring-hom-ring* R S φ
assumes $P \in$ *carrier* (*Pring* R I)
assumes $Q \in$ *carrier* (*Pring* R I)
shows *ring-hom-to-IP-ring-hom* φ ($P \oplus_{Pring\ R\ I}$ Q) =
(*ring-hom-to-IP-ring-hom* φ P) $\oplus_{Pring\ S\ I}$ (*ring-hom-to-IP-ring-hom* φ Q)
unfolding *ring-hom-to-IP-ring-hom-def*
proof
fix m
show φ (($P \oplus_{Pring\ R\ I}$ Q) m) = (($\lambda m.\ \varphi$ (P m)) $\oplus_{Pring\ S\ I}$ ($\lambda m.\ \varphi$ (Q m))) m
proof−
have *RHS*: (($\lambda m.\ \varphi$ (P m)) $\oplus_{Pring\ S\ I}$ ($\lambda m.\ \varphi$ (Q m))) m = φ (P m) $\oplus_S$ φ (Q m)
using *ring.Pring-add*[*of* S I - -] *assms*
by (*metis cring.axioms*(*1*) *ring.indexed-padd-def*)
have *LHS*: φ (($P \oplus_{Pring\ R\ I}$ Q) m) = φ ((P m)$\oplus_R$ (Q m))
by (*metis Pring-add indexed-padd-def*)
then show *?thesis*
using *assms* **unfolding** *ring-hom-ring-def*
using *ring-hom-add*[*of* φ R S P m Q m]
by (*metis Pring-carrier-coeff' RHS assms*(*2*) *ring-hom-ring.homh*)
qed
qed

lemma *ring-hom-to-IP-ring-hom-closed*:
assumes *cring* S
assumes *ring-hom-ring* R S φ
assumes $P \in$ *carrier* (*Pring* R I)

shows *ring-hom-to-IP-ring-hom* φ *P* $\in$ *carrier* (*Pring S I*)
apply(*rule indexed-pset.induct*[*of P I carrier R*])
using *Pring-car assms*(*3*) **apply** *blast*
proof−
show $\bigwedge k.$ *k* $\in$ *carrier R* $\Longrightarrow$ *ring-hom-to-IP-ring-hom* φ (*indexed-const k*) $\in$ *carrier* (*Pring S I*)
proof−
fix *k*
show *k* $\in$ *carrier R* $\Longrightarrow$ *ring-hom-to-IP-ring-hom* φ (*indexed-const k*) $\in$ *carrier* (*Pring S I*)
proof−
assume *A*: *k* $\in$ *carrier R*
have (φ *k*) $\in$ *carrier S*
by (*meson A assms*(*2*) *ring-hom-closed ring-hom-ring.homh*)
then have *0*: *ring.indexed-const S* (φ *k*) $\in$ *carrier* (*Pring S I*)
by (*metis assms*(*1*) *cring.axioms*(*1*) *ring.Pring-car ring.indexed-pset.indexed-const*)
then show *?thesis*
using *assms*(*2*)
by (*simp add*: *0 A ring-hom-to-IP-ring-hom-constant*[*of S* φ *k*] *assms*(*1*))
qed
qed
show $\bigwedge P\ Q.$ *P* $\in$ *Pring-set R I* $\Longrightarrow$
ring-hom-to-IP-ring-hom φ *P* $\in$ *carrier* (*Pring S I*) $\Longrightarrow$
Q $\in$ *Pring-set R I* $\Longrightarrow$
ring-hom-to-IP-ring-hom φ *Q* $\in$ *carrier* (*Pring S I*) $\Longrightarrow$ *ring-hom-to-IP-ring-hom* φ (*P* $\bigoplus$ *Q*) $\in$ *carrier* (*Pring S I*)
using *ring-hom-to-IP-ring-hom-add*[*of S* φ *- I*] *Pring-car*[*of I*] *ring.Pring-car*[*of S I*]
by (*metis Pring-add assms*(*1*) *assms*(*2*) *cring.axioms*(*1*) *ring.Pring-add-closed*)
show $\bigwedge P\ i.$ *P* $\in$ *Pring-set R I* $\Longrightarrow$
ring-hom-to-IP-ring-hom φ *P* $\in$ *carrier* (*Pring S I*) $\Longrightarrow$ *i* $\in$ *I* $\Longrightarrow$ *ring-hom-to-IP-ring-hom* φ (*P* $\bigotimes$ *i*) $\in$ *carrier* (*Pring S I*)
proof−
fix *P*
fix *i*
assume *A*: *P* $\in$ *Pring-set R I* *ring-hom-to-IP-ring-hom* φ *P* $\in$ *carrier* (*Pring S I*) *i* $\in$ *I*
have *0*: ($\lambda m.$ φ ((*P* $\bigotimes$ *i*) *m*)) = *ring.indexed-pmult S* (*ring-hom-to-IP-ring-hom* φ *P*) *i*
proof
fix *m*
show φ ((*P* $\bigotimes$ *i*) *m*) = *ring.indexed-pmult S* (*ring-hom-to-IP-ring-hom* φ *P*) *i m*
proof(*cases i* $\in\#$ *m*)
case *True*
then have *LHS*: φ ((*P* $\bigotimes$ *i*) *m*) = φ (*P* (*m* − {#*i*#}))
by (*metis indexed-pmult-def*)
then show *?thesis*

```
      using True
     by (metis assms(1) cring.axioms(1) ring.indexed-pmult-def ring-hom-to-IP-ring-hom-def)
    next
     case False
     then have φ ((P ⨂ i) m) = φ 0_R
      by (metis indexed-pmult-def)
     then have LHS: φ ((P ⨂ i) m) = 0_S
      by (metis assms(1) assms(2) cring.axioms(1) local.ring-axioms
          ring-hom-ring.homh ring-hom-zero)
     then show ?thesis
      using False assms ring.indexed-pmult-def
      by (metis cring.axioms(1))
    qed
   qed
   then show ring-hom-to-IP-ring-hom φ (P ⨂ i) ∈ carrier (Pring S I)
    using assms
    unfolding ring-hom-to-IP-ring-hom-def
   by (metis 0 A(2) A(3) cring.axioms(1) ring.Pring-car ring.indexed-pset.simps)
  qed
qed

lemma ring-hom-to-IP-ring-hom-monom:
  assumes cring S
  assumes ring-hom-ring R S φ
  shows ring-hom-to-IP-ring-hom φ (mset-to-IP R m) = mset-to-IP S m
proof
  fix x
  show ring-hom-to-IP-ring-hom φ (mset-to-IP R m) x = mset-to-IP S m x
   unfolding ring-hom-to-IP-ring-hom-def mset-to-IP-def apply( cases x = m )
   apply (metis (mono-tags, lifting) assms(2) ring-hom-one ring-hom-ring.homh)
   by (metis (full-types) assms(1) assms(2) cring.axioms(1) local.ring-axioms
       ring-hom-ring.homh ring-hom-zero)
qed

lemma Pring-morphism:
  assumes cring S
  assumes φ ∈ (carrier (Pring R I)) → (carrier S)
  assumes φ 1_Pring R I = 1_S
  assumes φ 0_Pring R I = 0_S
  assumes ⋀P Q. P ∈ carrier (Pring R I) ⟹ Q ∈ carrier (Pring R I) ⟹
          φ (P ⊕_Pring R I Q) = (φ P) ⊕_S (φ Q)
  assumes ⋀ i . ⋀ P. i ∈ I ⟹ P ∈ carrier (Pring R I) ⟹ φ (P ⨂ i) = (φ
P) ⊗_S (φ (mset-to-IP R {#i#}))
 assumes ⋀k Q. k ∈ carrier R ⟹ Q ∈ carrier (Pring R I) ⟹ φ (poly-scalar-mult
R k Q) =
                                  (φ (indexed-const k)) ⊗_S (φ Q)
  shows ring-hom-ring (Pring R I) S φ
  apply(rule ring-hom-ringI)
  apply (simp add: Pring-is-ring; fail)
```

```
  apply (simp add: assms(1) cring.axioms(1); fail)
  using assms(2) apply blast
proof−

  show ⋀x y. x ∈ carrier (Pring R I) ⟹ y ∈ carrier (Pring R I) ⟹ φ (x
⊗Pring R I y) = φ x ⊗S φ y
  proof−
    fix P Q
    assume A0: P ∈ carrier (Pring R I)
    assume A1: Q ∈ carrier (Pring R I)
    show φ (P ⊗Pring R I Q) = φ P ⊗S φ Q
    proof(rule mpoly-induct'[of ])
      show φ (P ⊗Pring R I indexed-const 0) = φ P ⊗S φ (indexed-const 0)
      proof−
        have 0: (P ⊗Pring R I indexed-const 0) = 0Pring R I
          using assms
         by (metis A0 Pring-is-cring Pring-zero cring.cring-simprules(27) is-cring)
        have 1:  φ P ⊗S φ (indexed-const 0) = 0S
        proof−
          have φ P ∈ carrier S
            using assms(2) A0
            by blast
          then show ?thesis
            using assms(4) Pring-zero[of I]
            by (metis assms(1) cring.cring-simprules(27))
        qed
        then show ?thesis using 0 1
          using assms(4) by presburger
      qed
      show ⋀n Q. (⋀Q. Q ∈ Pring-set R I ∧ card (monomials-of R Q) ≤ n ⟹
φ (P ⊗Pring R I Q) = φ P ⊗S φ Q) ⟹
            Q ∈ Pring-set R I ∧ card (monomials-of R Q) = Suc n ⟹ φ (P
⊗Pring R I Q) = φ P ⊗S φ Q
      proof− fix n fix Q
        assume IH:  (⋀Q. Q ∈ Pring-set R I ∧ card (monomials-of R Q) ≤ n ⟹
φ (P ⊗Pring R I Q) = φ P ⊗S φ Q)
        assume A: Q ∈ Pring-set R I ∧ card (monomials-of R Q) = Suc n
        show φ (P ⊗Pring R I Q) = φ P ⊗S φ Q
        proof−
         obtain m M where m-M-def: monomials-of R Q = insert m M ∧ m ∉ M
            using A
         by (metis card-0-eq ex-in-conv finite.emptyI mk-disjoint-insert nat.distinct(1))
         have Q = (restrict-poly-to-monom-set R Q M) ⊕Pring R I (poly-scalar-mult
R (Q m) (mset-to-IP R m))
         by (metis A Pring-add m-M-def remove-monom-eq remove-monom-restrict-poly-to-monom-set)
          obtain Q' where Q'-def: Q' = (restrict-poly-to-monom-set R Q M)
            by simp
          have Q'-fact:  Q' ∈ Pring-set R I ∧ card (monomials-of R Q') ≤ n
            proof−
```

have *0*: $Q' \in$ *Pring-set R I*
using *Q'-def A restriction-closed*
by *blast*
have *monomials-of R Q'* $= M$
using *A m-M-def Q'-def*
by (*metis restrict-poly-to-monom-set-monoms subset-insertI*)
then have *card* (*monomials-of R Q'*) $= n$ **using** *A m-M-def*
by (*metis 0 card-insert-disjoint diff-Suc-1 monomials-finite*)
then show *?thesis*
by (*simp add*: *0*)
qed
have *0*:($P \otimes_{Pring\ R\ I} Q$) = ($P \otimes_{Pring\ R\ I}$ ($Q' \oplus_{Pring\ R\ I}$ (*poly-scalar-mult R* (*Q m*) (*mset-to-IP R m*))))
using *Q'-def* ‹*Q* = *restrict-poly-to-monom-set R Q M* $\oplus_{Pring\ R\ I}$ *Mt* (*Q m*) *m*› **by** *presburger*
have *1*: ($P \otimes_{Pring\ R\ I} Q$) = ($P \otimes_{Pring\ R\ I}\ Q'$) $\oplus_{Pring\ R\ I}$ ($P \otimes_{Pring\ R\ I}$ (*poly-scalar-mult R* (*Q m*) (*mset-to-IP R m*)))
proof−
have *10*: $P \in$ *carrier* (*Pring R I*)
by (*simp add*: *A0*)
have *11*: $Q \in$ *carrier* (*Pring R I*)
by (*simp add*: *A Pring-car*)
have *12*: $Q' \in$ *carrier* (*Pring R I*)
using *A Pring-car Q'-def restriction-closed* **by** *blast*
have *13*: (*poly-scalar-mult R* (*Q m*) (*mset-to-IP R m*)) $\in$ *carrier* (*Pring R I*)
by (*metis A Pring-car Pring-carrier-coeff' insert-iff m-M-def mset-to-IP-closed' poly-scalar-mult-closed*)
then show *?thesis*
using *0 10 11 12 13*
by (*metis Pring-mult-rdistr*)
qed
then have φ ($P \otimes_{Pring\ R\ I} Q$) = ($\varphi$ ($P \otimes_{Pring\ R\ I}\ Q'$)) $\oplus_S$ φ ($P \otimes_{Pring\ R\ I}$ (*poly-scalar-mult R* (*Q m*) (*mset-to-IP R m*)))
by (*metis A A0 Pring-car Pring-mult-closed Pring-cfs-closed Q'-def assms*(*5*) *insert-iff*
local.ring-axioms m-M-def poly-scalar-mult-closed ring.mset-to-IP-closed' ring.restriction-closed)
then have φ ($P \otimes_{Pring\ R\ I} Q$) = ($\varphi$ P) $\otimes_S$ (φ Q') $\oplus_S$ φ ($P \otimes_{Pring\ R\ I}$ (*poly-scalar-mult R* (*Q m*) (*mset-to-IP R m*)))
using *IH*[*of Q'*] *Q'-fact*
by *presburger*
then have *2*: φ ($P \otimes_{Pring\ R\ I} Q$) = ($\varphi$ P) $\otimes_S$ (φ Q') $\oplus_S$ φ (*poly-scalar-mult R* (*Q m*) ($P \otimes_{Pring\ R\ I}$ (*mset-to-IP R m*)))
by (*metis A A0 Pring-car Pring-mult Pring-cfs-closed insert-iff m-M-def mset-to-IP-closed' times-poly-scalar-mult*)
have *3*: $\bigwedge k.\ k \in$ *carrier R* $\Longrightarrow$ *set-mset m* $\subseteq I \Longrightarrow \varphi$ (*poly-scalar-mult R k* ($P \otimes_{Pring\ R\ I}$ (*mset-to-IP R m*))) =

φ (*indexed-const k*) $\otimes_S$ (φ *P*) $\otimes_S$ φ (*mset-to-IP R m*)

proof(*induction m*)

case *empty*

assume *A*: *set-mset* {#} $\subseteq$ *I*

then have *E0*: (*P* $\otimes_{Pring\ R\ I}$ *mset-to-IP R* {#}) = *P*

by (*metis A0 Pring-mult-one Pring-one one-mset-to-IP*)

have *E1*: *k* $\in$ *carrier R*

using *A Pring-cfs-closed empty.prems*(*1*)

by *linarith*

have *E2*: φ (*mset-to-IP R* {#}) = $\mathbf{1}_S$

by (*metis Pring-one assms*(*3*) *one-mset-to-IP*)

have *E3*: φ (*poly-scalar-mult R k* (*P* $\otimes_{Pring\ R\ I}$ *mset-to-IP R* {#})) = φ (*indexed-const k*) $\otimes_S$ φ (*P* $\otimes_{Pring\ R\ I}$ *mset-to-IP R* {#})

using *E1 E2 E0 assms*(*7*)[*of k P* $\otimes_{Pring\ R\ I}$ *mset-to-IP R* {#}] *A0*

by (*simp add*: *A0 E1*)

have *E4*: φ (*P* $\otimes_{Pring\ R\ I}$ *mset-to-IP R* {#}) = (φ *P*)

using *E0*

by *auto*

have *E5*: φ (*poly-scalar-mult R k* (*P* $\otimes_{Pring\ R\ I}$ *mset-to-IP R* {#})) = φ (*indexed-const k*) $\otimes_S$ φ *P*

using *E0 E3* **by** *presburger*

have *E6*: φ (*indexed-const k*) $\otimes_S$ φ *P* = φ (*indexed-const k*) $\otimes_S$ φ *P* $\otimes_S$ $\mathbf{1}_S$

proof−

have *0*: φ *P* $\in$ *carrier S*

using *assms A0*

by *blast*

have *indexed-const k* $\in$ *carrier* (*Pring R I*)

using *assms*(*2*) *E1 Pring-car indexed-pset.indexed-const*

by *blast*

then have *1*: φ (*indexed-const k*) $\in$ *carrier S*

using *assms*(*2*)

by *blast*

show *?thesis*

using *assms*(*1*) *0 1*

by (*metis cring.cring-simprules*(*12*) *cring.cring-simprules*(*14*) *cring.cring-simprules*(*5*) *cring.cring-simprules*(*6*))

qed

then show *?case*

using *E0 E2 E3* **by** *presburger*

next

case (*add x m*)

assume *AA*: $\bigwedge$*k*. *k* $\in$ *carrier R* $\Longrightarrow$ *set-mset m* $\subseteq$ *I* $\Longrightarrow$ φ (*poly-scalar-mult R k* (*P* $\otimes_{Pring\ R\ I}$ *mset-to-IP R m*)) = φ (*indexed-const k*) $\otimes_S$ φ *P* $\otimes_S$ φ (*mset-to-IP R m*)

assume *P0*: *set-mset* (*add-mset x m*) $\subseteq$ *I*

then have *IA*: *set-mset m* $\subseteq$*I*

by (*metis insert-subset set-mset-add-mset-insert*)

have *IH*: φ (*poly-scalar-mult R k* (*P* $\otimes_{Pring\ R\ I}$ *mset-to-IP R m*)) =

φ (*indexed-const k*) $\otimes_S$ φ *P* $\otimes_S$ φ (*mset-to-IP R m*)
using *AA IA add.prems*(*1*)
by *blast*
then have *x-mem*: $x \in I$
by (*meson P0 subsetD union-single-eq-member*)
have *0*: *mset-to-IP R* (*add-mset x m*) = *mset-to-IP R m* $\bigotimes$ *x*
by (*simp add*: *monom-add-mset*)
then have *1*: *P* $\otimes_{Pring\ R\ I}$ *mset-to-IP R* (*add-mset x m*) = *P* $\otimes_{Pring\ R\ I}$ (*mset-to-IP R m* $\bigotimes$ *x*)
by *simp*
have *2*: *P* $\otimes_{Pring\ R\ I}$ *mset-to-IP R* (*add-mset x m*) =(*P* $\otimes_{Pring\ R\ I}$ (*mset-to-IP R m*)) $\bigotimes$ *x*
proof−
have *mset-to-IP R* (*add-mset x m*) = (*mset-to-IP R m*) $\bigotimes$ *x*
using *0* **by** *blast*
then have *P* $\otimes_{Pring\ R\ I}$ *mset-to-IP R* (*add-mset x m*) = *P* $\otimes_{Pring\ R\ I}$ ((*mset-to-IP R m*) $\bigotimes$ *x*)
by *simp*
then have *P* $\otimes_{Pring\ R\ I}$ *mset-to-IP R* (*add-mset x m*) = *P* $\otimes_{Pring\ R\ I}$ ((*mset-to-IP R m*) $\otimes_{Pring\ R\ I}$ *mset-to-IP R* {#*x*#})
by (*metis IA Pring-mult mset-to-IP-closed poly-index-mult x-mem*)
then have *P* $\otimes_{Pring\ R\ I}$ *mset-to-IP R* (*add-mset x m*) = (*P* $\otimes_{Pring\ R\ I}$ (*mset-to-IP R m*)) $\otimes_{Pring\ R\ I}$ *mset-to-IP R* {#*x*#}
by (*metis A0 IA Pring-car Pring-mult-assoc Pring-one Pring-one-closed indexed-pset.indexed-pmult monom-add-mset mset-to-IP-closed one-mset-to-IP x-mem*)
then show *?thesis*
by (*metis A0 IA Pring-car Pring-mult Pring-mult-closed mset-to-IP-closed poly-index-mult x-mem*)
qed
have *3*: *poly-scalar-mult R k* (*P* $\otimes_{Pring\ R\ I}$ *mset-to-IP R* (*add-mset x m*)) =
poly-scalar-mult R k (*P* $\otimes_{Pring\ R\ I}$ (*mset-to-IP R m*) $\bigotimes$ *x*)
using *2* **by** *presburger*
have *4*: *poly-scalar-mult R k* (*P* $\otimes_{Pring\ R\ I}$ *mset-to-IP R* (*add-mset x m*)) =
poly-scalar-mult R k (*P* $\otimes_{Pring\ R\ I}$ (*mset-to-IP R m*)) $\bigotimes$ *x*
proof−
have *poly-scalar-mult R k* (*P* $\otimes_{Pring\ R\ I}$ *mset-to-IP R* (*add-mset x m*)) =
poly-scalar-mult R k (*P* $\otimes_{Pring\ R\ I}$ (*mset-to-IP R m*) $\otimes_{Pring\ R\ I}$ *mset-to-IP R* {#*x*#})
by (*metis 2 A0 IA Pring-car Pring-mult Pring-mult-closed mset-to-IP-closed poly-index-mult x-mem*)
have *poly-scalar-mult R k* (*P* $\otimes_{Pring\ R\ I}$ *mset-to-IP R* (*add-mset x m*)) =
(*poly-scalar-mult R k* (*P* $\otimes_{Pring\ R\ I}$ *mset-to-IP R m*)) $\otimes_{Pring\ R\ I}$ *mset-to-IP R* {#*x*#}

by (*metis A0 IA Pring-car Pring-mult Pring-mult-closed Pring-one Pring-one-closed*
‹*poly-scalar-mult R k* (*P* $\otimes_{Pring\ R\ I}$ *mset-to-IP R* (*add-mset x m*)) = *poly-scalar-mult R k* (*P* $\otimes_{Pring\ R\ I}$ *mset-to-IP R m* $\otimes_{Pring\ R\ I}$ *mset-to-IP R* {#*x*#})›
add.prems(*1*) *indexed-pset.indexed-pmult monom-add-mset mset-to-IP-closed*
one-mset-to-IP poly-scalar-mult-times x-mem)
then show *?thesis*
by (*metis A0 IA Pring-car Pring-mult Pring-mult-closed add.prems*(*1*)
cring.poly-scalar-mult-closed is-cring mset-to-IP-closed
poly-index-mult x-mem)
qed
have *5*: φ (*poly-scalar-mult R k* (*P* $\otimes_{Pring\ R\ I}$ *mset-to-IP R* (*add-mset x m*))) =
φ (*poly-scalar-mult R k* (*P* $\otimes_{Pring\ R\ I}$ (*mset-to-IP R m*)) $\bigotimes$ *x*)
using *4* **by** *metis*
have *6*: φ (*poly-scalar-mult R k* (*P* $\otimes_{Pring\ R\ I}$ *mset-to-IP R* (*add-mset x m*))) =
φ (*poly-scalar-mult R k* (*P* $\otimes_{Pring\ R\ I}$ (*mset-to-IP R m*))) $\otimes_S$ φ (*mset-to-IP R* {#*x*#})
using *assms*
by (*metis 5 A0 IA Pring-car Pring-mult-closed add.prems*(*1*)
cring.poly-scalar-mult-closed is-cring mset-to-IP-closed x-mem)
have *7*: φ (*poly-scalar-mult R k* (*P* $\otimes_{Pring\ R\ I}$ *mset-to-IP R* (*add-mset x m*))) =
φ (*indexed-const k*) $\otimes_S$ φ *P* $\otimes_S$ φ (*mset-to-IP R m*) $\otimes_S$ φ (*mset-to-IP R* {#*x*#})
using *assms 6 IH*
by *presburger*
have *8*: φ (*mset-to-IP R m*) $\otimes_S$ φ (*mset-to-IP R* {#*x*#}) = φ (*mset-to-IP R* (*add-mset x m*))
proof−
have φ (*mset-to-IP R m*) $\otimes_S$ φ (*mset-to-IP R* {#*x*#}) =
φ ((*mset-to-IP R m*) $\otimes_{Pring\ R\ I}$ (*mset-to-IP R* {#*x*#}))
by (*metis IA Pring-car Pring-mult assms*(*6*) *mset-to-IP-closed poly-index-mult x-mem*)
then show *?thesis*
by (*metis 0 IA Pring-car assms*(*6*) *mset-to-IP-closed x-mem*)
qed
have *9*: φ (*poly-scalar-mult R k* (*P* $\otimes_{Pring\ R\ I}$ *mset-to-IP R* (*add-mset x m*))) =
φ (*indexed-const k*) $\otimes_S$ φ *P* $\otimes_S$ (φ (*mset-to-IP R m*) $\otimes_S$ φ (*mset-to-IP R* {#*x*#}))
proof−
have *0*: φ (*indexed-const k*) ∈ *carrier S*
proof−
have (*indexed-const k*) ∈ *carrier* (*Pring R I*)
using *A Pring-car Pring-cfs-closed indexed-pset.indexed-const*

add.prems(*1*)
by *blast*
then show *?thesis*
using *assms*(*2*)
by *blast*
qed
have *1*: $\varphi\ P \in$ *carrier S*
using *A0 assms*(*2*)
by *blast*
have *2*: φ (*mset-to-IP R m*) $\in$ *carrier S*
proof−
have (*mset-to-IP R m*) $\in$ *carrier* (*Pring R I*)
using *IA Pring-car mset-to-IP-closed* **by** *blast*
then show *?thesis* **using** *assms*(*2*)
by *blast*
qed
have *3*: φ (*mset-to-IP R* {#*x*#}) $\in$ *carrier S*
proof−
have (*mset-to-IP R* {#*x*#}) $\in$ *carrier* (*Pring R I*)
by (*metis Pring-car Pring-one Pring-one-closed indexed-pset.indexed-pmult monom-add-mset one-mset-to-IP x-mem*)
then show *?thesis*
using *assms*(*2*)
by *blast*
qed
have φ (*poly-scalar-mult R k* (*P* $\otimes_{Pring\ R\ I}$ *mset-to-IP R* (*add-mset x m*))) =
φ (*indexed-const k*) $\otimes_S$ ($\varphi\ P \otimes_S \varphi$ (*mset-to-IP R m*) $\otimes_S \varphi$ (*mset-to-IP R* {#*x*#}))
using *1 2 3 7*
by (*metis 0* ‹φ (*mset-to-IP R m*) $\in$ *carrier S*› *assms*(*1*) *cring.cring-simprules*(*11*) *cring.cring-simprules*(*5*))
then show *?thesis*
by (*metis 0 1 2 3 8 Pring-mult assms*(*1*) *cring.cring-simprules*(*11*) *cring.cring-simprules*(*5*))
qed
have *10*: φ (*poly-scalar-mult R k* (*P* $\otimes_{Pring\ R\ I}$ *mset-to-IP R* (*add-mset x m*))) =
φ (*indexed-const k*) $\otimes_S \varphi\ P \otimes_S \varphi$ (*mset-to-IP R* (*add-mset x m*))
using *8 9*
by *presburger*
then show *?case*
by *blast*
qed
have *4*: φ (*poly-scalar-mult R* (*Q m*) (*P* $\otimes_{Pring\ R\ I}$ (*mset-to-IP R m*))) =
φ (*indexed-const* (*Q m*)) $\otimes_S$ ($\varphi\ P$) $\otimes_S \varphi$ (*mset-to-IP R m*)
using *3*
by (*metis A Pring-mult insert-iff local.ring-axioms m-M-def mset-to-IP-indices*

ring.Pring-cfs-closed)
have *5*: $\varphi\ (P \otimes_{Pring\ R\ I} Q) = (\varphi\ P) \otimes_S (\varphi\ Q') \oplus_S \varphi\ (indexed\text{-}const\ (Q\ m)) \otimes_S (\varphi\ P) \otimes_S \varphi\ (mset\text{-}to\text{-}IP\ R\ m)$
using *2 4*
by *presburger*
have $\varphi\ (indexed\text{-}const\ (Q\ m)) \otimes_S (\varphi\ P) \otimes_S \varphi\ (mset\text{-}to\text{-}IP\ R\ m) = (\varphi\ P) \otimes_S \varphi\ (indexed\text{-}const\ (Q\ m)) \otimes_S \varphi\ (mset\text{-}to\text{-}IP\ R\ m)$
proof−
have *0*: $(\varphi\ P) \in carrier\ S$
using *A0 assms(2)*
by *blast*
have *1*: $\varphi\ (indexed\text{-}const\ (Q\ m)) \in carrier\ S$
proof−
have $indexed\text{-}const\ (Q\ m\ ) \in carrier\ (Pring\ R\ I)$
using *A Pring-car Pring-cfs-closed indexed-pset.indexed-const* **by** *blast*
then show *?thesis*
using *assms(2)*
by *blast*
qed
have *2*: $\varphi\ (mset\text{-}to\text{-}IP\ R\ m) \in carrier\ S$
proof−
have $(mset\text{-}to\text{-}IP\ R\ m) \in carrier\ (Pring\ R\ I)$
by (*metis A Pring-car insert-iff m-M-def mset-to-IP-closed'*)
then show *?thesis* **using** *assms(2)*
by *blast*
qed
show *?thesis* **using** *assms(1) 0 1 2*
by (*metis cring.cring-simprules(14)*)
qed
then have *6*: $\varphi\ (P \otimes_{Pring\ R\ I} Q) = (\varphi\ P) \otimes_S (\varphi\ Q') \oplus_S (\varphi\ P) \otimes_S \varphi\ (indexed\text{-}const\ (Q\ m)) \otimes_S \varphi\ (mset\text{-}to\text{-}IP\ R\ m)$
using *5*
by *presburger*
then have *7*: $\varphi\ (P \otimes_{Pring\ R\ I} Q) = (\varphi\ P) \otimes_S ((\varphi\ Q') \oplus_S \varphi\ (indexed\text{-}const\ (Q\ m)) \otimes_S \varphi\ (mset\text{-}to\text{-}IP\ R\ m))$
proof−
have *0*: $(\varphi\ P) \in carrier\ S$
using *A0 assms(2)*
by *blast*
have *1*: $\varphi\ (indexed\text{-}const\ (Q\ m)) \in carrier\ S$
proof−
have $indexed\text{-}const\ (Q\ m\ ) \in carrier\ (Pring\ R\ I)$
using *A Pring-car Pring-cfs-closed indexed-pset.indexed-const* **by** *blast*
then show *?thesis*
using *assms(2)*
by *blast*
qed
have *2*: $\varphi\ (mset\text{-}to\text{-}IP\ R\ m) \in carrier\ S$
proof−

```
        have (mset-to-IP R m) ∈ carrier (Pring R I)
          by (metis A Pring-car insert-iff m-M-def mset-to-IP-closed')
        then show ?thesis using assms(2)
          by blast
      qed
      have 3: φ Q' ∈ carrier S
      proof−
        have Q' ∈ carrier (Pring R I)
          using Q'-fact Pring-car
          by blast
        then show ?thesis
          using assms(2)
          by blast
      qed
      show ?thesis using 0 1 2 3 6
       by (metis assms(1) cring.cring-simprules(11) cring.cring-simprules(25)
cring.cring-simprules(5))
    qed
    then show ?thesis
      by (metis A Pring-car Pring-cfs-closed Q'-def Q'-fact
          ‹Q = restrict-poly-to-monom-set R Q M ⊕Pring R I Mt (Q m) m›
assms(5) assms(7)
        cring.poly-scalar-mult-closed insert-iff is-cring m-M-def mset-to-IP-closed')
  qed
 qed
 show Q ∈ Pring-set R I
   using A1 Pring-car
   by blast
 qed
qed
 show ⋀x y. x ∈ carrier (Pring R I) ⟹ y ∈ carrier (Pring R I) ⟹ φ (x
⊕Pring R I y) = φ x ⊕S φ y
  using assms(5)
  by blast
 show φ 1Pring R I = 1S
  by (simp add: assms(3))
qed

lemma(in cring) indexed-const-Pring-mult:
 assumes k ∈ carrier R
 assumes P ∈ carrier (Pring R I)
 shows (indexed-const k ⊗Pring R I P) m = k ⊗R (P m)
       (P ⊗Pring R I indexed-const k) m = k ⊗R (P m)
 apply (metis Pring-car Pring-mult assms(1) assms(2) poly-scalar-mult-def poly-scalar-mult-eq)
 by (metis Pring-car Pring-carrier-coeff' Pring-mult assms(1) assms(2) indexed-const-P-multE
m-comm)

lemma(in cring) ring-hom-to-IP-ring-hom-is-hom:
 assumes cring S
```

assumes *ring-hom-ring R S φ*
shows *ring-hom-ring* (*Pring R I*) (*Pring S I*) (*ring-hom-to-IP-ring-hom φ*)
proof(*rule Pring-morphism*)
show *0*: *cring* (*Pring S I*)
by (*simp add*: *assms*(*1*) *cring.axioms*(*1*) *ring.Pring-is-cring*)
show *1*: *ring-hom-to-IP-ring-hom φ* ∈ *carrier* (*Pring R I*) → *carrier* (*Pring S I*)
by (*meson Pi-I assms*(*1*) *assms*(*2*) *ring-hom-to-IP-ring-hom-closed*)
show *2*: *ring-hom-to-IP-ring-hom φ* $\mathbf{1}_{Pring\ R\ I} = \mathbf{1}_{Pring\ S\ I}$
by (*simp add*: *assms*(*1*) *assms*(*2*) *ring-hom-to-IP-ring-hom-one*)
show *3*: *ring-hom-to-IP-ring-hom φ* $\mathbf{0}_{Pring\ R\ I} = \mathbf{0}_{Pring\ S\ I}$
proof−
have ⋀*m*. *φ* ($\mathbf{0}_{Pring\ R\ I}$ *m*) = $\mathbf{0}_S$
using *assms*
by (*metis Pring-carrier-coeff′ Pring-zero Pring-zero-closed cring.axioms*(*1*) *cring.cring-simprules*(*26*) *indexed-const-Pring-mult*(*2*) *is-cring ring.Pring-is-cring ring-hom-ring.homh ring-hom-zero zero-closed*)
then have ⋀*m*. *φ* ($\mathbf{0}_{Pring\ R\ I}$ *m*) = $\mathbf{0}_{Pring\ S\ I}$ *m*
by (*metis assms*(*1*) *cring.axioms*(*1*) *ring.Pring-zero ring.indexed-const-def*)
then show *?thesis* **unfolding** *ring-hom-to-IP-ring-hom-def*
by *blast*
qed
show *4*: ⋀*P Q*. *P* ∈ *carrier* (*Pring R I*) ⟹
Q ∈ *carrier* (*Pring R I*) ⟹ *ring-hom-to-IP-ring-hom φ* (*P* $\oplus_{Pring\ R\ I}$ *Q*) = *ring-hom-to-IP-ring-hom φ P* $\oplus_{Pring\ S\ I}$ *ring-hom-to-IP-ring-hom φ Q*
using *assms*(*1*) *assms*(*2*) *ring-hom-to-IP-ring-hom-add* **by** *blast*
show *5*: ⋀*i P*. *i* ∈ *I* ⟹
P ∈ *carrier* (*Pring R I*) ⟹
ring-hom-to-IP-ring-hom φ (*P* ⨂ *i*) = *ring-hom-to-IP-ring-hom φ P* $\otimes_{Pring\ S\ I}$ *ring-hom-to-IP-ring-hom φ* (*mset-to-IP R* {#*i*#})
proof−
fix *i P*
assume *i0*: *i* ∈ *I*
assume *P0*: *P* ∈ *carrier* (*Pring R I*)
show *ring-hom-to-IP-ring-hom φ* (*P* ⨂ *i*) = *ring-hom-to-IP-ring-hom φ P* $\otimes_{Pring\ S\ I}$ *ring-hom-to-IP-ring-hom φ* (*mset-to-IP R* {#*i*#})
unfolding *ring-hom-to-IP-ring-hom-def*
proof
fix *m*
show *φ* ((*P* ⨂ *i*) *m*) = ((λ*m*. *φ* (*P m*)) $\otimes_{Pring\ S\ I}$ (λ*m*. *φ* (*mset-to-IP R* {#*i*#} *m*))) *m*
proof(*cases i* ∈# *m*)
case *True*
have (λ*m*. *φ* (*mset-to-IP R* {#*i*#} *m*)) = *mset-to-IP S* {#*i*#}
proof
fix *m*
show *φ* (*mset-to-IP R* {#*i*#} *m*) = *mset-to-IP S* {#*i*#} *m*
apply(*cases* {#*i*#} = *m*)
apply (*metis* (*mono-tags*, *lifting*) *assms*(*1*) *assms*(*2*) *cring.axioms*(*1*)

```
local.ring-axioms
                    mset-to-IP-def one-mset-to-IP ring.Pring-one ring.Pring-one
ring.one-mset-to-IP
              ring-hom-to-IP-ring-hom-def ring-hom-to-IP-ring-hom-one)
        proof−
          assume {#i#} ≠ m
          then have LHS: (mset-to-IP R {#i#} m) = 0_R
            by (metis mset-to-IP-simp′)
          have  RHS :mset-to-IP S {#i#} m = 0_S
            by (metis ‹{#i#} ≠ m› mset-to-IP-def)
          have φ 0_R = 0_S
            using assms(1) assms(2) cring.axioms(1) local.ring-axioms
              ring-hom-ring.homh ring-hom-zero by blast
          then show ?thesis
            using  LHS RHS
            by presburger
        qed
      qed
      then have RHS: ((λm. φ (P m)) ⊗_Pring S I (λm. φ (mset-to-IP R {#i#}
m))) m =
            ((λm. φ (P m)) ⊗_Pring S I mset-to-IP S {#i#}) m
        by presburger
      then have RHS′: ((λm. φ (P m)) ⊗_Pring S I (λm. φ (mset-to-IP R {#i#}
m))) m =
            (ring.indexed-pmult S (λm. φ (P m)) i) m
      proof−
        have 0: (λm. φ (P m)) ∈ Pring-set S I
         using ring-hom-to-IP-ring-hom-closed[of S φ P I] ring.Pring-car[of S I]
assms
          unfolding ring-hom-to-IP-ring-hom-def
          using P0 cring.axioms(1) by blast
        show ?thesis using ring.poly-index-mult[of S (λm. φ (P m)) I i]
         by (metis 0 ‹(λm. φ (mset-to-IP R {#i#} m)) = mset-to-IP S {#i#}›
             assms(1) cring.axioms(1) i0 ring.Pring-mult)
      qed
      then have RHS′′: ((λm. φ (P m)) ⊗_Pring S I (λm. φ (mset-to-IP R {#i#}
m))) m =
          (λm. φ (P m)) (m − {#i#}) using ring.indexed-pmult-def[of S (λm. φ
(P m)) i] True
        by (metis assms(1) cring.axioms(1))
      then show ?thesis
        by (metis True indexed-pmult-def)
    next
      case False
      have LHS: ((P ⨂ i) m) = 0_R
        using False
        by (meson indexed-pmult-def)
      have (λm. φ (mset-to-IP R {#i#} m)) = mset-to-IP S {#i#}
      proof
```

fix m
show φ (*mset-to-IP R* {#*i*#} *m*) = *mset-to-IP S* {#*i*#} *m*
apply(*cases* {#*i*#} = *m*)
apply (*metis* (*mono-tags*, *lifting*) *assms*(*1*) *assms*(*2*) *cring.axioms*(*1*) *local.ring-axioms*
mset-to-IP-def one-mset-to-IP ring.Pring-one ring.Pring-one ring.one-mset-to-IP
ring-hom-to-IP-ring-hom-def ring-hom-to-IP-ring-hom-one)
proof−
assume {#*i*#} ≠ *m*
then have *LHS*: (*mset-to-IP R* {#*i*#} *m*) = $\mathbf{0}_R$
by (*metis mset-to-IP-simp*′)
have *RHS* :*mset-to-IP S* {#*i*#} *m* = $\mathbf{0}_S$
by (*metis* ‹{#*i*#} ≠ *m*› *mset-to-IP-def*)
have φ $\mathbf{0}_R$ = $\mathbf{0}_S$
using *assms*(*1*) *assms*(*2*) *cring.axioms*(*1*) *local.ring-axioms*
ring-hom-ring.homh ring-hom-zero **by** *blast*
then show *?thesis*
using *LHS RHS*
by *presburger*
qed
qed
then have *RHS*: ((λ*m*. φ (*P m*)) $\otimes_{Pring\ S\ I}$ (λ*m*. φ (*mset-to-IP R* {#*i*#} *m*))) *m* =
((λ*m*. φ (*P m*)) $\otimes_{Pring\ S\ I}$ *mset-to-IP S* {#*i*#}) *m*
by *presburger*
then have *RHS*′: ((λ*m*. φ (*P m*)) $\otimes_{Pring\ S\ I}$ (λ*m*. φ (*mset-to-IP R* {#*i*#} *m*))) *m* =
(*ring.indexed-pmult S* (λ*m*. φ (*P m*)) *i*) *m*
proof−
have *0*: (λ*m*. φ (*P m*)) ∈ *Pring-set S I*
using *ring-hom-to-IP-ring-hom-closed*[*of S* φ *P I*] *ring.Pring-car*[*of S I*] *assms*
unfolding *ring-hom-to-IP-ring-hom-def*
using *P0 cring.axioms*(*1*) **by** *blast*
show *?thesis* **using** *ring.poly-index-mult*[*of S* (λ*m*. φ (*P m*)) *I i*]
by (*metis 0* ‹(λ*m*. φ (*mset-to-IP R* {#*i*#} *m*)) = *mset-to-IP S* {#*i*#}›
assms(*1*) *cring.axioms*(*1*) *i0 ring.Pring-mult*)
qed
then have *RHS*′′: ((λ*m*. φ (*P m*)) $\otimes_{Pring\ S\ I}$ (λ*m*. φ (*mset-to-IP R* {#*i*#} *m*))) *m* =
$\mathbf{0}_S$
using *False*
by (*metis assms*(*1*) *cring.axioms*(*1*) *ring.indexed-pmult-def*)
then show *?thesis*
using *LHS False assms ring-hom-zero*[*of* φ *R S*]
by (*metis cring.axioms*(*1*) *local.ring-axioms ring-hom-ring.homh*)
qed
qed

qed
show $\bigwedge k\ Q.\ k \in carrier\ R \Longrightarrow$
$Q \in carrier\ (Pring\ R\ I) \Longrightarrow$
$ring\text{-}hom\text{-}to\text{-}IP\text{-}ring\text{-}hom\ \varphi\ (poly\text{-}scalar\text{-}mult\ R\ k\ Q) = ring\text{-}hom\text{-}to\text{-}IP\text{-}ring\text{-}hom\ \varphi\ (indexed\text{-}const\ k) \otimes_{Pring\ S\ I} ring\text{-}hom\text{-}to\text{-}IP\text{-}ring\text{-}hom\ \varphi\ Q$
proof−
fix $k\ Q$
assume $A0$: $k \in carrier\ R$
assume $A1$: $Q \in carrier\ (Pring\ R\ I)$
show $ring\text{-}hom\text{-}to\text{-}IP\text{-}ring\text{-}hom\ \varphi\ (poly\text{-}scalar\text{-}mult\ R\ k\ Q) = ring\text{-}hom\text{-}to\text{-}IP\text{-}ring\text{-}hom\ \varphi\ (indexed\text{-}const\ k) \otimes_{Pring\ S\ I} ring\text{-}hom\text{-}to\text{-}IP\text{-}ring\text{-}hom\ \varphi\ Q$
proof
fix x
show $ring\text{-}hom\text{-}to\text{-}IP\text{-}ring\text{-}hom\ \varphi\ (poly\text{-}scalar\text{-}mult\ R\ k\ Q)\ x = (ring\text{-}hom\text{-}to\text{-}IP\text{-}ring\text{-}hom\ \varphi\ (indexed\text{-}const\ k) \otimes_{Pring\ S\ I} ring\text{-}hom\text{-}to\text{-}IP\text{-}ring\text{-}hom\ \varphi\ Q)\ x$
proof−
have LHS: $ring\text{-}hom\text{-}to\text{-}IP\text{-}ring\text{-}hom\ \varphi\ (poly\text{-}scalar\text{-}mult\ R\ k\ Q)\ x = \varphi\ (k \otimes_R Q\ x)$
by $(metis\ poly\text{-}scalar\text{-}mult\text{-}def\ ring\text{-}hom\text{-}to\text{-}IP\text{-}ring\text{-}hom\text{-}def)$
then have LHS': $ring\text{-}hom\text{-}to\text{-}IP\text{-}ring\text{-}hom\ \varphi\ (poly\text{-}scalar\text{-}mult\ R\ k\ Q)\ x = (\varphi\ k) \otimes_S \varphi\ (Q\ x)$
proof−
have $Q\ x \in carrier\ \ R$
using $A1\ Pring\text{-}car\ local.ring\text{-}axioms\ ring.Pring\text{-}cfs\text{-}closed$ **by** $blast$
then show $?thesis$
using $LHS\ assms\ ring\text{-}hom\text{-}mult[of\ \varphi\ R\ S\ k\ Q\ x]$
by $(metis\ A0\ ring\text{-}hom\text{-}ring.homh)$
qed
have RHS: $(ring\text{-}hom\text{-}to\text{-}IP\text{-}ring\text{-}hom\ \varphi\ (indexed\text{-}const\ k) \otimes_{Pring\ S\ I} ring\text{-}hom\text{-}to\text{-}IP\text{-}ring\text{-}hom\ \varphi\ Q)\ x = (\varphi\ k) \otimes_S (\varphi\ (Q\ x))$
proof−
have 0: $ring\text{-}hom\text{-}to\text{-}IP\text{-}ring\text{-}hom\ \varphi\ (indexed\text{-}const\ k) = ring.indexed\text{-}const\ S\ (\varphi\ k)$
by $(simp\ add{:}\ A0\ assms(1)\ assms(2)\ ring\text{-}hom\text{-}to\text{-}IP\text{-}ring\text{-}hom\text{-}constant)$
have 1: $(\varphi\ k) \in carrier\ S$
by $(meson\ A0\ assms(2)\ ring\text{-}hom\text{-}closed\ ring\text{-}hom\text{-}ring.homh)$
have 2: $ring\text{-}hom\text{-}to\text{-}IP\text{-}ring\text{-}hom\ \varphi\ Q \in carrier\ (Pring\ S\ I)$
using $A1\ assms\ ring\text{-}hom\text{-}to\text{-}IP\text{-}ring\text{-}hom\text{-}closed$
by $blast$
have 3: $(ring.indexed\text{-}const\ S\ (\varphi\ k) \otimes_{Pring\ S\ I} ring\text{-}hom\text{-}to\text{-}IP\text{-}ring\text{-}hom\ \varphi\ Q)\ x = (\varphi\ k) \otimes_S (\varphi\ (Q\ x))$
using $assms(1)\ 1\ 2$
$cring.indexed\text{-}const\text{-}Pring\text{-}mult(1)[of\ S\ \varphi\ k\ ring\text{-}hom\text{-}to\text{-}IP\text{-}ring\text{-}hom\ \varphi\ Q\ I\ x]$
$ring\text{-}hom\text{-}to\text{-}IP\text{-}ring\text{-}hom\text{-}def[of\ \varphi\ Q\ x]$
by $presburger$
then show $?thesis$

```
        by (metis 0)
      qed
      then show ?thesis
        using LHS'
        by metis
    qed
  qed
 qed
qed

lemma ring-hom-to-IP-ring-hom-smult:
  assumes cring S
  assumes ring-hom-ring R S φ
  assumes P ∈ carrier (Pring R I)
  assumes a ∈ carrier R
  shows ring-hom-to-IP-ring-hom φ (a ⊙_Pring R I P) =
              φ a ⊙_Pring S I (ring-hom-to-IP-ring-hom φ P)
proof fix m
  have 0: φ ((a ⊙_Pring R I P) m) = φ (a ⊗ (P m))
    using assms
    by (metis Pring-smult-cfs)
  hence 1: φ ((a ⊙_Pring R I P) m) = φ a ⊗_S φ (P m)
    using assms ring-hom-mult[of φ R S]
    by (metis Pring-carrier-coeff' ring-hom-ring.homh)
  have 2: (φ a ⊙_Pring S I (λm. φ (P m))) m = φ a ⊗_S φ (P m)
    using assms ring.Pring-smult[of S I]
    unfolding poly-scalar-mult-def cring-def
    by presburger
  show ring-hom-to-IP-ring-hom φ (a ⊙_Pring R I P) m =
       (φ a ⊙_Pring S I ring-hom-to-IP-ring-hom φ P) m
    unfolding ring-hom-to-IP-ring-hom-def using assms 1 2
    by presburger
qed
```

2.12.2 A Universal Property for Indexed Polynomial Rings

```
lemma Pring-universal-prop-0:
  assumes a-cring: cring S
  assumes index-map: closed-fun S g
  assumes ring-hom: ring-hom-ring R S φ
  assumes ψ = (total-eval S g) ∘ (ring-hom-to-IP-ring-hom φ)
  shows (ring-hom-ring (Pring R I) S ψ)
        (∀ i ∈ I. ψ (mset-to-IP R {#i#}) = g i)
        (∀ a ∈ carrier R. ψ (indexed-const a) = φ a)
        ∀ ϱ. (ring-hom-ring (Pring R I) S ϱ) ∧
          (∀ i ∈ I. ϱ (mset-to-IP R {#i#}) = g i) ∧
          (∀ a ∈ carrier R. ϱ (indexed-const a) = φ a) ⟶
          (∀ x ∈ carrier (Pring R I). ϱ x = ψ x)
proof−
```

have *0*: (*ring-hom-to-IP-ring-hom* φ) $\in$ *ring-hom* (*Pring R I*) (*Pring S I*)
using *a-cring ring-hom ring-hom-ring.homh ring-hom-to-IP-ring-hom-is-hom*
by *blast*
have *1*: (*total-eval S g*) $\in$ *ring-hom* (*Pring S I*) *S*
using *a-cring cring.total-eval-ring-hom index-map ring-hom-ring.homh*
by *blast*
show *P0*: *ring-hom-ring* (*Pring R I*) *S* ψ
using *ring-hom-trans 0 1 Pring-is-ring a-cring assms*(*4*) *cring.axioms*(*1*) *ring-hom-ringI2*
by *blast*
show *P1*: $\forall\, i{\in}I.\ \psi$ (*mset-to-IP R* {#*i*#}) = *g i*
proof
fix *i* **assume** *Pi*: *i* $\in$ *I*
show ψ (*mset-to-IP R* {#*i*#}) = *g i*
proof−
have *0*: ψ (*mset-to-IP R* {#*i*#}) = (*total-eval S g*) ((*ring-hom-to-IP-ring-hom* φ) (*mset-to-IP R* {#*i*#}))
by (*simp add*: *assms*(*4*))
have ((*ring-hom-to-IP-ring-hom* φ) (*mset-to-IP R* {#*i*#})) = (*mset-to-IP S* {#*i*#})
by (*simp add*: *a-cring ring-hom ring-hom-to-IP-ring-hom-monom*)
then have ψ (*mset-to-IP R* {#*i*#}) = (*total-eval S g*) (*mset-to-IP S* {#*i*#})
by (*simp add*: *0*)
then show *?thesis*
by (*simp add*: *a-cring cring.total-eval-var index-map*)
qed
qed
show *P2*: $\forall\, a{\in}$*carrier R*. ψ (*indexed-const a*) = φ *a*
proof
fix *a* **assume** *A*: *a* $\in$ *carrier R*
show ψ (*indexed-const a*) = φ *a*
proof−
have *0*: *ring-hom-to-IP-ring-hom* φ (*indexed-const a*) = *ring.indexed-const S* (φ *a*)
by (*simp add*: *A a-cring ring-hom ring-hom-to-IP-ring-hom-constant*)
have *1*: *total-eval S g* (*ring.indexed-const S* (φ *a*)) = φ *a*
by (*meson A a-cring cring.total-eval-const ring-hom ring-hom-closed ring-hom-ring.homh*)
show *?thesis*
using *assms 0 1*
by (*simp add*: *0 index-map*)
qed
qed
show $\forall$ ϱ. (*ring-hom-ring* (*Pring R I*) *S* ϱ) $\wedge$
($\forall\, i \in I.\ \varrho$ (*mset-to-IP R* {#*i*#}) = *g i*) $\wedge$
($\forall\, a \in$ *carrier R*. ϱ (*indexed-const a*) = φ *a*) $\longrightarrow$
($\forall\, x \in$ *carrier* (*Pring R I*). ϱ *x* = ψ *x*)
proof
fix ϱ
show *ring-hom-ring* (*Pring R I*) *S* ϱ $\wedge$ ($\forall\, i{\in}I.\ \varrho$ (*mset-to-IP R* {#*i*#}) = *g i*) $\wedge$ ($\forall\, a{\in}$*carrier R*. ϱ (*indexed-const a*) = φ *a*) $\longrightarrow$

```
      (∀ x∈carrier (Pring R I). ϱ x = ψ x)
  proof
    assume A: (ring-hom-ring (Pring R I) S ϱ) ∧
      (∀ i ∈ I. ϱ (mset-to-IP R {#i#}) = g i) ∧
      (∀ a ∈ carrier R. ϱ (indexed-const a) = φ a)
    show (∀ x ∈ carrier (Pring R I). ϱ x = ψ x)
    proof
      fix x assume B: x ∈ carrier (Pring R I)
      show ϱ x = ψ x
        apply(rule indexed-pset.induct[of x I carrier R])
        using B Pring-car apply blast
          apply (metis A P2)
      proof−
        show ⋀P Q. P ∈ Pring-set R I ⟹ ϱ P = ψ P ⟹ Q ∈ Pring-set R I
⟹ ϱ Q = ψ Q ⟹ ϱ (P ⊕ Q) = ψ (P ⊕ Q)
        proof−
          fix P Q
          assume A0: P ∈ Pring-set R I  ϱ P = ψ P  Q ∈ Pring-set R I  ϱ Q
= ψ Q
          show ϱ (P ⊕ Q) = ψ (P ⊕ Q)
            using A A0 ring-hom-add[of ψ Pring R I S P Q] ring-hom-add[of ϱ
Pring R I S P Q] P0
            by (metis Pring-add Pring-car ring-hom-ring.homh)
          qed
          show ⋀P i. P ∈ Pring-set R I ⟹ ϱ P = ψ P ⟹ i ∈ I ⟹ ϱ (P ⊗
i) = ψ (P ⊗ i)
          proof−
            fix P i
            assume A0:  P ∈ Pring-set R I  ϱ P = ψ P i ∈ I
            show ϱ (P ⊗ i) = ψ (P ⊗ i)
            proof−
              have ϱ (P ⊗Pring R I (mset-to-IP R {#i#})) = ψ (P ⊗Pring R I
(mset-to-IP R {#i#}))
                using A A0 ring-hom-mult[of ψ Pring R I S P (mset-to-IP R
{#i#})]
                  ring-hom-mult[of ϱ Pring R I S P (mset-to-IP R {#i#})] P0
            by (metis P1 Pring-car Pring-one Pring-one-closed indexed-pset.indexed-pmult
                monom-add-mset one-mset-to-IP ring-hom-ring.homh)
              then show ?thesis
                by (metis A0(1) A0(3) Pring-mult poly-index-mult)
            qed
          qed
        qed
      qed
    qed
  qed
qed

end
```

definition *close-fun* :: *'c set* ⇒ (*'e*, *'f*) *ring-scheme* ⇒ (*'c* ⇒ *'e*) ⇒ (*'c* ⇒ *'e*)
where
close-fun I S g = (λ*i*. (*if i* ∈ *I then g i else* $\mathbf{0}_S$))

context *cring*
begin

lemma *close-funE*:
assumes *cring S*
assumes *g* ∈ *I* → *carrier S*
shows *closed-fun S* (*close-fun I S g*)
apply(*rule cring.closed-funI*)
apply (*simp add*: *assms*(*1*))
by (*metis close-fun-def PiE assms*(*1*) *assms*(*2*) *cring.cring-simprules*(*2*))

end

definition *indexed-poly-induced-morphism* ::
'c set ⇒ (*'e*, *'f*) *ring-scheme* ⇒ (*'a*, *'e*) *ring-hom* ⇒ (*'c* ⇒ *'e*) ⇒ ((*'a*,*'c*) *mvar-poly*,
'e) *ring-hom* **where**
indexed-poly-induced-morphism I S φ *g* = (*total-eval S* (*close-fun I S g*)) ∘ (*ring-hom-to-IP-ring-hom* φ)

context *cring*
begin

lemma *Pring-universal-prop*:
assumes *a-cring*: *cring S*
assumes *index-map*: *g* ∈ *I* → *carrier S*
assumes *ring-hom*: *ring-hom-ring R S* φ
assumes ψ = *indexed-poly-induced-morphism I S* φ *g*
shows (*ring-hom-ring* (*Pring R I*) *S* ψ)
(∀ *i* ∈ *I*. ψ (*mset-to-IP R* {#*i*#}) = *g i*)
(∀ *a* ∈ *carrier R*. ψ (*indexed-const a*) = φ *a*)
∀ ϱ. (*ring-hom-ring* (*Pring R I*) *S* ϱ) ∧
(∀ *i* ∈ *I*. ϱ (*mset-to-IP R* {#*i*#}) = *g i*) ∧
(∀ *a* ∈ *carrier R*. ϱ (*indexed-const a*) = φ *a*) ⟶
(∀ *x* ∈ *carrier* (*Pring R I*). ϱ *x* = ψ *x*)
proof−
obtain *g'* **where** *g'-def*: *g'* = (*close-fun I S g*)
by *simp*
have *0*: *closed-fun S g'*
using *close-funE a-cring g'-def index-map*
by *blast*
show (*ring-hom-ring* (*Pring R I*) *S* ψ) **using** *assms 0*
using *cring.Pring-universal-prop-0*(*1*) *indexed-poly-induced-morphism-def g'-def is-cring*
by *blast*

show $(\forall\, i \in I.\ \psi\ (\textit{mset-to-IP}\ R\ \{\#i\#\}) = g\ i)$
proof −
have $(\forall\, i \in I.\ \psi\ (\textit{mset-to-IP}\ R\ \{\#i\#\}) = g'\ i)$
apply(*intro Pring-universal-prop-0*[*of S - φ*] *assms*)
unfolding *assms indexed-poly-induced-morphism-def g′-def*
using *assms 0 g′-def* **apply** *fastforce*
by *auto*
thus *?thesis* **unfolding** *g′-def* **using** *assms*
by (*simp add*: *close-fun-def*)
qed
show $\forall\, a{\in}\textit{carrier}\ R.\ \psi\ (\textit{indexed-const}\ a) = \varphi\ a$
using *0 indexed-poly-induced-morphism-def Pring-universal-prop-0*(*3*) *a-cring assms*(*4*) *g′-def ring-hom*
by *blast*
show $\forall\, \varrho.\ \textit{ring-hom-ring}\ (\textit{Pring}\ R\ I)\ S\ \varrho \wedge (\forall\, i{\in}I.\ \varrho\ (\textit{mset-to-IP}\ R\ \{\#i\#\}) = g\ i) \wedge (\forall\, a{\in}\textit{carrier}\ R.\ \varrho\ (\textit{indexed-const}\ a) = \varphi\ a) \longrightarrow (\forall\, x{\in}\textit{carrier}\ (\textit{Pring}\ R\ I).\ \varrho\ x = \psi\ x)$
proof −
have $\forall\, \varrho.\ \textit{ring-hom-ring}\ (\textit{Pring}\ R\ I)\ S\ \varrho \wedge (\forall\, i{\in}I.\ \varrho\ (\textit{mset-to-IP}\ R\ \{\#i\#\}) = g'\ i) \wedge (\forall\, a{\in}\textit{carrier}\ R.\ \varrho\ (\textit{indexed-const}\ a) = \varphi\ a) \longrightarrow (\forall\, x{\in}\textit{carrier}\ (\textit{Pring}\ R\ I).\ \varrho\ x = \psi\ x)$
using *assms 0 Pring-universal-prop-0*(*4*) *g′-def*
unfolding *indexed-poly-induced-morphism-def*
by *blast*
then show *?thesis*
using *g′-def*
unfolding *close-fun-def*
by *meson*
qed
qed

2.13 Mapping Mulitvariate Polynomials over a Single Variable to Univariate Polynomials

Constructor for multisets which have one distinct element

definition $\textit{nat-to-mset} :: 'c \Rightarrow \textit{nat} \Rightarrow 'c\ \textit{monomial}$ **where**
$\textit{nat-to-mset}\ i\ n = \textit{Abs-multiset}\ (\lambda j.\ \textit{if}\ (j = i)\ \textit{then}\ n\ \textit{else}\ 0)$

lemma *nat-to-msetE*: $\textit{count}\ (\textit{nat-to-mset}\ i\ n)\ i = n$
unfolding *nat-to-mset-def* **by** *simp*

lemma *nat-to-msetE′*:
assumes $j \neq i$
shows $\textit{count}\ (\textit{nat-to-mset}\ i\ n)\ j = 0$
unfolding *nat-to-mset-def* **using** *assms* **by** *simp*

lemma *nat-to-mset-add*: $\textit{nat-to-mset}\ i\ (n + m) = (\textit{nat-to-mset}\ i\ n) + (\textit{nat-to-mset}\ i\ m)$
apply(*rule multiset-eqI*)

by (*metis add.right-neutral nat-to-msetE nat-to-msetE′ count-union*)

lemma *nat-to-mset-inj*:
assumes $n \neq m$
shows (*nat-to-mset i n*) ≠ (*nat-to-mset i m*)
using *assms*
by (*metis nat-to-msetE*)

lemma *nat-to-mset-zero*: *nat-to-mset i 0* = {#}
by (*metis add.right-neutral add-cancel-right-right nat-to-mset-add*)

lemma *nat-to-mset-Suc*: *nat-to-mset i* (*Suc n*) = *add-mset i* (*nat-to-mset i n*)
using *nat-to-mset-add*[*of i n 1*]
by (*simp add*: *multiset-eqI nat-to-msetE nat-to-msetE′*)

lemma *nat-to-mset-Pring-singleton*:
assumes *cring R*
assumes *P* ∈ *carrier* (*Pring R* {*i*})
assumes *m* ∈ *monomials-of R P*
shows *m* = *nat-to-mset i* (*count m i*)
proof−
have ⋀ *j*. *count m j* = *count* (*nat-to-mset i* (*count m i*)) *j*
proof−
fix *j*
show *count m j* = *count* (*nat-to-mset i* (*count m i*)) *j*
apply(*cases j* = *i*)
apply (*simp add*: *nat-to-msetE*; *fail*)
proof−
assume *A*: *j* ≠*i*
have *P0*: *set-mset m* ⊆ {*i*}
using *assms*
by (*metis cring.axioms*(*1*) *ring.Pring-car ring.mset-to-IP-indices*)
then have *count m j* = *0*
using *A assms*
by (*metis count-inI empty-iff singletonD subset-singletonD*)
then show *count m j* = *count* (*nat-to-mset i* (*count m i*)) *j*
by (*simp add*: *A nat-to-msetE′*)
qed
qed
then show *?thesis*
using *multiset-eqI* **by** *blast*
qed

definition *IP-to-UP* :: *′d* ⇒ (*′e*, *′d*) *mvar-poly* ⇒ *′e u-poly* **where**
IP-to-UP i P = (λ (*n*::*nat*). *P* (*nat-to-mset i n*))

lemma *IP-to-UP-closed*:
assumes *cring R*
assumes *P* ∈ *carrier* (*Pring R* {*i*::*′c*})

shows *IP-to-UP i P* ∈ *carrier* (*UP R*)
proof−
have *IP-to-UP i P* ∈ *up R*
apply(*rule mem-upI*)
using *assms*
apply (*metis cring-def IP-to-UP-def ring.Pring-carrier-coeff'*)
unfolding *bound-def IP-to-UP-def*
apply(*rule ccontr*)
proof−
assume *A*: ∄ *n*. ∀ *m*>*n*. *P* (*nat-to-mset i m*) = $\mathbf{0}_R$
then have *0*: ∀ *n*. ∃ *m*> *n*. (*nat-to-mset i m*) ∈ *monomials-of R P*
by (*meson assms*(*1*) *cring-def ring.complement-of-monomials-of*)
have ¬ *finite* {*m*. (*nat-to-mset i m*) ∈ *monomials-of R P* }
proof
assume *finite* {*m*. *nat-to-mset i m* ∈ *monomials-of R P*}
then have ∃ *n*. ∀ *m*>*n*. *m* ∉ {*m*. *nat-to-mset i m* ∈ *monomials-of R P*}
by (*meson finite-nat-set-iff-bounded nat-less-le order.strict-trans*)
then have ∃ *n*. ∀ *m*>*n*. *nat-to-mset i m* ∉ *monomials-of R P*
by (*simp add*: *monomials-of-def*)
then show *False* **using** *0*
by *blast*
qed
then obtain *S* **where** *S-def*: *infinite* (*S*::*nat set*) ∧ (∀ *m* ∈ *S*. (*nat-to-mset i m*) ∈ *monomials-of R P*)
by *blast*
have *inj-on* (*nat-to-mset i*) *S*
using *inj-onI*[*of S nat-to-mset i*]
by (*meson nat-to-mset-inj*)
then have *1*: *infinite* (*nat-to-mset i* ‘ *S*)
using *S-def finite-imageD*
by *blast*
have *2*: (*nat-to-mset i* ‘ *S*) ⊆ (*monomials-of R P*)
using *S-def*
by *blast*
then have *infinite* (*monomials-of R P*)
using *S-def 1 finite-subset*
by *blast*
then show *False* **using** *assms*
by (*metis Pring-def cring-def partial-object.select-convs*(*1*) *ring.monomials-finite*)
qed
then show *?thesis*
by (*metis* (*no-types*, *lifting*) *UP-def partial-object.select-convs*(*1*))
qed

lemma *IP-to-UP-var*:
shows *IP-to-UP i* (*mset-to-IP R* {#*i*#}) = *X-poly R*
proof
have *UP*: *UP-cring R*
by (*simp add*: *UP-cring-def cring-axioms*)

fix *x*
show *IP-to-UP i* (*mset-to-IP R* {#*i*#}) *x* = *X-poly R x*
proof(*cases x* = *1*)
case *True*
then have *RHS*: *X-poly R x* = $\mathbf{1}_R$
unfolding *X-poly-def* **using** *UP UP-ring.cfs-monom*[*of R*]
unfolding *UP-ring-def*
using *local.ring-axioms one-closed* **by** *presburger*
have *LHS*: *IP-to-UP i* (*mset-to-IP R* ((*add-mset i*) {#})) *x*
= *mset-to-IP R* ((*add-mset i*) {#}) (*nat-to-mset i x*)
unfolding *IP-to-UP-def*
by *blast*
have (*nat-to-mset i x*) = {#*i*#}
proof−
have ⋀*n*. *count* (*nat-to-mset i x*) *n* = *count* {#*i*#} *n*
by (*simp add*: *True nat-to-msetE nat-to-msetE'*)
then show *?thesis*
using *multi-count-eq*
by *blast*
qed
then have *LHS*: *IP-to-UP i* (*mset-to-IP R* ((*add-mset i*) {#})) *x*
= *mset-to-IP R* ((*add-mset i*) {#}) {#*i*#}
using *LHS* **by** *presburger*
then show *?thesis*
unfolding *deg-zero-cf-def*
by (*metis RHS mset-to-IP-simp*)
next
case *False*
then have *RHS*: *X-poly R x* = $\mathbf{0}_R$
unfolding *X-poly-def*
using *UP UP-ring.cfs-monom*[*of R*]
unfolding *UP-ring-def*
using *local.ring-axioms one-closed* **by** *presburger*
have *LHS*: *IP-to-UP i* (*mset-to-IP R* ((*add-mset i*) {#})) *x*
= *mset-to-IP R* ((*add-mset i*) {#}) (*nat-to-mset i x*)
unfolding *IP-to-UP-def*
by *blast*
then show *?thesis*
unfolding *deg-zero-cf-def*
by (*metis False RHS count-single mset-to-IP-def nat-to-msetE*)
qed
qed

end

context *UP-cring*
begin

lemma *IP-to-UP-monom*:

shows *IP-to-UP i (mset-to-IP R (nat-to-mset i n)) = ((X-poly R)[^]$_{UP\ R}$n)*
proof
fix *x*
show *IP-to-UP i (mset-to-IP R (nat-to-mset i n)) x = (X-poly R [^]$_{UP\ R}$ n) x*
proof(*cases x = n*)
case *True*
have *RHS*: *(X-poly R [^]$_{UP\ R}$ n) x =* $\mathbf{1}_R$
unfolding *X-poly-def*
by (*metis P.nat-pow-closed P.nat-pow-eone P-def R.one-closed True UP-cring.X-closed UP-cring.monom-coeff UP-one-closed UP-r-one deg-one is-UP-cring monom-one monom-rep-X-pow to-poly-inverse to-poly-mult-simp*(*2*))
have *LHS*: *IP-to-UP i (mset-to-IP R (nat-to-mset i n)) x =* $\mathbf{1}_R$
by (*metis R.mset-to-IP-simp True IP-to-UP-def*)
then show *?thesis*
using *RHS* **by** *presburger*
next
case *False*
have *0*: *⋀x y::nat. nat-to-mset x = nat-to-mset y ⟹ x = y*
proof−
fix *a b::nat* **assume** *A*: *nat-to-mset a = nat-to-mset b*
then show *a = b* **unfolding** *nat-to-mset-def*
by (*metis A nat-to-msetE nat-to-msetE' zero-neq-one*)
qed
have *1*: *IP-to-UP i (mset-to-IP R (nat-to-mset i n)) x = (if nat-to-mset i x = nat-to-mset i n then* $\mathbf{1}$ *else* $\mathbf{0}$)
unfolding *IP-to-UP-def mset-to-IP-def*
by *blast*
hence *2*: *IP-to-UP i (mset-to-IP R (nat-to-mset i n)) x = (if x = n then* $\mathbf{1}$ *else* $\mathbf{0}$)
using *0*
by (*meson False nat-to-mset-inj*)
have *3*: *(X-poly R [^]$_{UP\ R}$ n) x =* $\mathbf{0}$
unfolding *X-poly-def* **using** *False*
by (*smt ctrm-degree P.nat-pow-closed P.nat-pow-eone P.r-null P-def R.one-closed UP-cring.ltrm-of-X UP-cring.ltrm-rep-X-pow UP-cring.X-closed UP-cring.monom-coeff UP-r-one UP-zero-closed X-mult-cf cfs-closed cfs-monom deg-nzero-nzero is-UP-cring monom-closed monom-one to-poly-inverse to-poly-mult-simp*(*2*))
thus *?thesis* **using** *2 1*
using *False* **by** *presburger*
qed
qed

lemma *IP-to-UP-one*:
IP-to-UP i $\mathbf{1}_{Pring\ R\ \{i\}} = \mathbf{1}_{UP\ R}$
proof
fix *x*
show *IP-to-UP i* $\mathbf{1}_{Pring\ R\ \{i\}}$ *x* $= \mathbf{1}_{UP\ R}$ *x*

```
  proof(cases x = 0)
    case True
    have RHS: 1_(UP R) x = 1_R
      using P-def True cfs-one by presburger
    have 1_(Pring R {i}) = (λ m. if m = {#} then 1_R else 0_R)
      by (metis R.Pring-one R.indexed-const-def)
    then have IP-to-UP i 1_(Pring R {i}) = IP-to-UP i (λ m. if m = {#} then 1_R
else 0_R)
      by presburger
    then have LHS: IP-to-UP i 1_(Pring R {i}) x = 1_R
    by (smt True count-empty IP-to-UP-def multi-count-eq nat-to-msetE nat-to-msetE')
    then show ?thesis
      using RHS by presburger
  next
    case False
    have RHS: 1_(UP R) x = 0_R
      by (smt False UP-def monoid.simps(2))
    show ?thesis
      using False count-empty
            nat-to-msetE
            ring.indexed-const-def
      unfolding IP-to-UP-def
      by (metis R.Pring-one R.ring-axioms RHS)
  qed
qed

lemma IP-to-UP-zero:
 IP-to-UP i 0_(Pring R {i}) = 0_(UP R)
proof
  fix x
  show IP-to-UP i 0_(Pring R {i}) x = 0_(UP R) x
    unfolding IP-to-UP-def using R.Pring-zero
    by (metis P-def R.indexed-zero-def cfs-zero)
qed

lemma IP-to-UP-add:
  assumes  x ∈ carrier (Pring R {i})
  assumes  y ∈ carrier (Pring R {i})
  shows  IP-to-UP i (x ⊕_(Pring R {i}) y) =
         IP-to-UP i x ⊕_(UP R) IP-to-UP i y
proof
  fix n
  have LHS: IP-to-UP i (x ⊕_(Pring R {i}) y) n = (x ⊕_(Pring R {i}) y) (nat-to-mset
i n)
    by (meson IP-to-UP-def)
  then have LHS: IP-to-UP i (x ⊕_(Pring R {i}) y) n = x (nat-to-mset i n) ⊕_R y
(nat-to-mset i n)
    using assms  unfolding IP-to-UP-def
```

by (*metis R.Pring-add R.indexed-padd-def*)
have *RHS*: (*IP-to-UP i x* $\oplus_{UP\ R}$ *IP-to-UP i y*) *n* =
(*IP-to-UP i x*) *n* $\oplus_R$ (*IP-to-UP i y*) *n*
using *assms UP-ring.cfs-add IP-to-UP-closed*
by (*simp add*: *UP-ring.cfs-add R-cring cring.IP-to-UP-closed is-UP-ring*)
then show *IP-to-UP i* (*x* $\oplus_{Pring\ R\ \{i\}}$ *y*) *n* = (*IP-to-UP i x* $\oplus_{UP\ R}$ *IP-to-UP i y*) *n*
using *assms*
by (*metis LHS IP-to-UP-def*)
qed

lemma *IP-to-UP-indexed-const*:
assumes *k* ∈ *carrier R*
shows *IP-to-UP i* (*ring.indexed-const R k*) = *to-polynomial R k*
proof
fix *x*
show *IP-to-UP i* (*ring.indexed-const R k*) *x* = *to-polynomial R k x*
proof(*cases x* = *0*)
case *True*
have *LHS*: *IP-to-UP i* (*ring.indexed-const R k*) *x* = *k*
using *True* **unfolding** *IP-to-UP-def*
by (*metis R.indexed-const-def nat-to-mset-zero*)
then show *?thesis*
using *assms*
unfolding *to-polynomial-def*
using *True to-polynomial-def*
by (*metis UP-ring.cfs-monom is-UP-ring*)
next
case *False*
have *LHS*: *IP-to-UP i* (*ring.indexed-const R k*) *x* = $\mathbf{0}_R$
using *False* **unfolding** *IP-to-UP-def*
by (*metis R.indexed-const-def nat-to-mset-inj nat-to-mset-zero*)
then show *?thesis*
using *assms*
unfolding *to-polynomial-def*
using *False UP-cring.intro UP-cring.monom-coeff UP-cring.monom-rep-X-pow*
using *P-def cfs-monom* **by** *presburger*
qed
qed

lemma *IP-to-UP-indexed-pmult*:
assumes *p* ∈ *carrier* (*Pring R* {*i*})
shows *IP-to-UP i* (*ring.indexed-pmult R p i*) = (*IP-to-UP i p*) $\otimes_{UP\ R}$ (*X-poly R*)
proof
fix *n*
have *0*: *IP-to-UP i p* ∈ *carrier* (*UP R*)
by (*simp add*: *R-cring assms cring.IP-to-UP-closed*)
show *IP-to-UP i* (*ring.indexed-pmult R p i*) *n* = (*IP-to-UP i p* $\otimes_{UP\ R}$ *X-poly*

R) *n*
proof(*cases n = 0*)
case *True*
then have *RHS*: (*IP-to-UP i p* $\otimes_{UP\ R}$ *X-poly R*) *n* = $\mathbf{0}_R$
by (*metis* (*no-types*, *lifting*) *0 lcf-closed One-nat-def P.r-null P-def R.r-null UP-cring.ltrm-of-X UP-cring.cfs-monom-mult UP-cring.cfs-monom-mult-l UP-zero-closed X-closed cfs-times-X deg-leE deg-nzero-nzero is-UP-cring lessI neq0-conv plus-1-eq-Suc to-poly-inverse*)
have *LHS*: *IP-to-UP i* (*ring.indexed-pmult R p i*) *n* = *ring.indexed-pmult R p i* (*nat-to-mset i n*)
unfolding *IP-to-UP-def*
by *blast*
then have *LHS'*: *IP-to-UP i* (*ring.indexed-pmult R p i*) *n* = (*p* $\otimes_{Pring\ R\ \{i\}}$ (*mset-to-IP R* {#*i*#})) (*nat-to-mset i n*)
using *assms*(*1*) *ring.Pring-car ring.Pring-mult ring.poly-index-mult singletonI*
by (*metis R.ring-axioms*)
then have *LHS'*: *IP-to-UP i* (*ring.indexed-pmult R p i*) *n* = (*p* $\otimes_{Pring\ R\ \{i\}}$ (*mset-to-IP R* {#*i*#})) {#}
using *True*
by (*metis nat-to-mset-zero*)
then show *?thesis* **using** *RHS LHS True assms*(*1*) *nat-to-mset-zero ring.indexed-pmult-def*
by (*metis R.ring-axioms empty-iff set-mset-empty*)
next
case *False*
then have *RHS*: (*IP-to-UP i p* $\otimes_{UP\ R}$ *X-poly R*) *n* = (*IP-to-UP i p*) (*n −1*)
using *0 Suc-diff-1 Suc-eq-plus1 assms*(*1*) *bot-nat-def IP-to-UP-def nat-neq-iff not-less0*
by (*metis* (*no-types*, *lifting*) *P-def UP-cring X-closed cfs-times-X cring.cring-simprules*(*14*))
have *LHS*: *IP-to-UP i* (*ring.indexed-pmult R p i*) *n* = *ring.indexed-pmult R p i* (*nat-to-mset i n*)
unfolding *IP-to-UP-def*
by *blast*
then have *LHS'*: *IP-to-UP i* (*ring.indexed-pmult R p i*) *n* = (*p* $\otimes_{Pring\ R\ \{i\}}$ (*mset-to-IP R* {#*i*#})) (*nat-to-mset i n*)
using *assms*(*1*) *ring.Pring-car ring.Pring-mult ring.poly-index-mult singletonI*
by (*metis R.ring-axioms*)
then have *LHS''*: *IP-to-UP i* (*ring.indexed-pmult R p i*) *n* = (*p* $\otimes_{Pring\ R\ \{i\}}$ (*mset-to-IP R* {#*i*#})) (*add-mset i* (*nat-to-mset i* (*n−1*)))
by (*metis False Suc-diff-1 nat-to-mset-Suc neq0-conv*)
then show *?thesis* **using** *RHS* **unfolding** *IP-to-UP-def*
by (*metis* (*no-types*, *lifting*) *False R.indexed-pmult-def Suc-diff-1 add-mset-remove-trivial add-mset-remove-trivial-If multi-self-add-other-not-self nat-to-mset-Suc neq0-conv*)
qed
qed

lemma *IP-to-UP-ring-hom*:
shows *ring-hom-ring* (*Pring R* {*i*}) (*UP R*) (*IP-to-UP i*)
apply(*rule cring.Pring-morphism*)
apply (*simp add*: *R-cring*; *fail*)
using *P-def UP-cring* **apply** *blast*
apply (*simp add*: *R.IP-to-UP-closed R-cring*; *fail*)
apply (*meson IP-to-UP-one*)
apply (*meson IP-to-UP-zero*)
apply (*meson IP-to-UP-add*)
apply (*metis R.IP-to-UP-var IP-to-UP-indexed-pmult singletonD*)
proof−
fix *k Q*
assume *A0*: *k* ∈ *carrier R*
assume *A1*: *Q* ∈ *carrier* (*Pring R* {*i*})
show *IP-to-UP i* (*poly-scalar-mult R k Q*) =
IP-to-UP i (*ring.indexed-const R k*) ⊗$_{UP\ R}$ *IP-to-UP i Q*
unfolding *poly-scalar-mult-def*
proof
fix *x*
show *IP-to-UP i* (λ*m*. *k* ⊗$_R$ *Q m*) *x* =
(*IP-to-UP i* (*ring.indexed-const R k*) ⊗$_{UP\ R}$ *IP-to-UP i Q*) *x*
proof−
have *LHS*: *IP-to-UP i* (λ*m*. *k* ⊗$_R$ *Q m*) *x* = *k* ⊗$_R$ *Q* (*nat-to-mset i x*)
unfolding *IP-to-UP-def*
by *blast*
have *RHS*: (*IP-to-UP i* (*ring.indexed-const R k*) ⊗$_{UP\ R}$ *IP-to-UP i Q*) *x* =
(*to-polynomial R k* ⊗$_{UP\ R}$ *IP-to-UP i Q*) *x*
by (*metis A0 IP-to-UP-indexed-const*)
have *RHS'*: (*IP-to-UP i* (*ring.indexed-const R k*) ⊗$_{UP\ R}$ *IP-to-UP i Q*) *x* =
k ⊗$_R$ ((*IP-to-UP i Q*) *x*)
proof−
have *0*: *deg R* (*to-polynomial R k*) = *0*
using *A0 degree-to-poly* **by** *blast*
have *1*: (*IP-to-UP i Q*) ∈ *carrier* (*UP R*)
using *IP-to-UP-closed* **unfolding** *P-def*
by (*simp add*: *A1 R.IP-to-UP-closed R-cring*)
then show *?thesis*
proof −
have *UP-cring R* ∧ *IP-to-UP i Q* ∈ *carrier* (*UP R*)
using *1 is-UP-cring* **by** *blast*
then show *?thesis*
by (*metis A0 UP-cring.to-poly-mult-simp*(*1*) *UP-ring.UP-mult-closed UP-ring.coeff-simp UP-ring.coeff-smult UP-ring.monom-closed IP-to-UP-indexed-const is-UP-ring to-polynomial-def*)
qed
qed
then show *?thesis*
by (*metis IP-to-UP-def*)
qed

qed
qed

lemma *IP-to-UP-ring-hom-inj*:
shows *inj-on* (*IP-to-UP i*) (*carrier* (*Pring R* {*i*}))
proof
fix *x y*
assume *A*: *x* ∈ *carrier* (*Pring R* {*i*}) *y* ∈ *carrier* (*Pring R* {*i*})
assume *B*: *IP-to-UP i x* = *IP-to-UP i y*
show *x* = *y*
proof
fix *a*
show *x a* = *y a*
proof(*cases set-mset a* ⊆ {*i*})
case *True*
then obtain *n* **where** *a* = (*nat-to-mset i n*)
by (*metis count-eq-zero-iff insert-subset multiset-eqI nat-to-msetE nat-to-msetE′*
set-eq-subset singletonD singleton-insert-inj-eq′ subset-insertI subset-refl)
then have *LHS*: *x a* = *IP-to-UP i x n*
by (*metis IP-to-UP-def*)
then show *?thesis*
by (*metis B* ‹*a* = *nat-to-mset i n*› *IP-to-UP-def*)
next
case *False*
then show *?thesis*
using *ring.Pring-set-zero*[*of R y* {*i*} *a*] *ring.Pring-set-zero*[*of R x* {*i*} *a*] *A*
by (*metis R.Pring-car R.ring-axioms*)
qed
qed
qed

lemma *IP-to-UP-scalar-mult*:
assumes *a* ∈ *carrier R*
assumes *p* ∈ *carrier* (*Pring R* {*i*})
shows (*IP-to-UP i* (*a* ⊙$_{Pring\ R\ \{i\}}$ *p*)) = *a*⊙$_{UP\ R}$ (*IP-to-UP i p*)
apply(*rule ring.indexed-pset.induct*[*of R p* {*i*} *carrier R*])
apply (*simp add*: *R.ring-axioms*; *fail*)
using *R.Pring-car assms*(*2*) **apply** *blast*
apply (*metis IP-to-UP-indexed-const P-def R.m-closed R.poly-scalar-mult-const*
R.ring-axioms assms(*1*) *ring.Pring-smult to-poly-closed to-poly-mult to-poly-mult-simp*(*1*))
proof−
show ⋀*P Q*. *P* ∈ *Pring-set R* {*i*} ⟹
IP-to-UP i (*a* ⊙$_{Pring\ R\ \{i\}}$ *P*) = *a* ⊙$_{UP\ R}$ *IP-to-UP i P* ⟹
Q ∈ *Pring-set R* {*i*} ⟹ *IP-to-UP i* (*a* ⊙$_{Pring\ R\ \{i\}}$ *Q*) = *a* ⊙$_{UP\ R}$
IP-to-UP i Q ⟹ *IP-to-UP i* (*a* ⊙$_{Pring\ R\ \{i\}}$ (*P* ⨁ *Q*)) = *a* ⊙$_{UP\ R}$ *IP-to-UP i*
(*P* ⨁ *Q*)
proof−
fix *p Q*
assume *A0*: *p* ∈ *Pring-set R* {*i*}

IP-to-UP i ($a \odot_{Pring\ R\ \{i\}} p$) = $a \odot_{UP\ R}$ *IP-to-UP i p*
Q ∈ *Pring-set R* {*i*}
IP-to-UP i ($a \odot_{Pring\ R\ \{i\}} Q$) = $a \odot_{UP\ R}$ *IP-to-UP i Q*
show *IP-to-UP i* ($a \odot_{Pring\ R\ \{i\}}$ ($p \bigoplus Q$)) = $a \odot_{UP\ R}$ *IP-to-UP i* ($p \bigoplus Q$)
proof −
have ($a \odot_{Pring\ R\ \{i\}}$ ($p \bigoplus Q$)) = $a \odot_{Pring\ R\ \{i\}}$ $p \oplus_{Pring\ R\ \{i\}}$ $a \odot_{Pring\ R\ \{i\}}$ Q
by (*metis A0(1) A0(3) R.Pring-add R.Pring-car R.Pring-smult-r-distr assms(1)*)
then show *?thesis* **using** *A0*
by (*metis IP-to-UP-add P-def R.IP-to-UP-closed R.Pring-add R.Pring-car R.Pring-smult-closed R-cring UP-smult-r-distr assms(1)*)
qed
qed
show $\bigwedge P\ ia.\ P \in$ *Pring-set R* {*i*} $\Longrightarrow$ *IP-to-UP i* ($a \odot_{Pring\ R\ \{i\}} P$) = $a \odot_{UP\ R}$ *IP-to-UP i P* $\Longrightarrow$ *ia* ∈ {*i*} $\Longrightarrow$ *IP-to-UP i* ($a \odot_{Pring\ R\ \{i\}}$ ($P \bigotimes ia$)) = $a \odot_{UP\ R}$ *IP-to-UP i* ($P \bigotimes ia$)
proof
fix *P j x*
assume *A0*: *P* ∈ *Pring-set R* {*i*}
assume *A1*: *IP-to-UP i* ($a \odot_{Pring\ R\ \{i\}} P$) = $a \odot_{UP\ R}$ *IP-to-UP i P*
assume *A2*: *j* ∈ {*i*}
then have *A3*: *j* = *i*
by *blast*
have *IP-to-UP i* (*ring.indexed-pmult R P j*) ∈ *carrier* (*UP R*)
by (*simp add*: *A0 A3 R.Pring-car R.indexed-pset.indexed-pmult R-cring cring.IP-to-UP-closed*)
then have ($a \odot_{UP\ R}$ (λ*n*. *ring.indexed-pmult R P j* (*nat-to-mset i n*))) *x* = $a \otimes_R$((λ*n*. *ring.indexed-pmult R P j* (*nat-to-mset i n*))) *x*
using *A0 A1 A3 assms* **unfolding** *IP-to-UP-def*
using *P-def cfs-smult* **by** *blast*
then show *IP-to-UP i* ($a \odot_{Pring\ R\ \{i\}}$ ($P \bigotimes j$)) *x* = ($a \odot_{UP\ R}$ *IP-to-UP i* ($P \bigotimes j$)) *x*
by (*metis A0 A2 IP-to-UP-def P-def R.Pring-car R.Pring-smult-cfs R.indexed-pset.indexed-pmult* ‹*IP-to-UP i* ($P \bigotimes j$) ∈ *carrier* (*UP R*)› *assms(1) cfs-smult*)
qed
qed

end

Evaluation of indexed polynomials commutes with evaluation of univariate polynomials:

lemma *pvar-closed*:
assumes *cring R*
assumes *i* ∈ *I*
shows (*pvar R i*) ∈ *carrier* (*Pring R I*)
by (*meson assms(1) assms(2) cring.axioms(1) ring.Pring-var-closed*)

context *UP-cring*
begin

lemma *pvar-mult*:
assumes $i \in I$
assumes $j \in I$
shows $(pvar\ R\ i) \otimes_{Pring\ R\ I} (pvar\ R\ j) = mset\text{-}to\text{-}IP\ R\ \{\#i, j\#\}$
proof−
have $\{\#i\#\} + \{\#j\#\} = \{\#\ i, j\#\}$
by *auto*
then show *?thesis*
unfolding *var-to-IP-def*
by (*metis R.Pring-mult R.monom-mult*)
qed

lemma *pvar-pow*:

assumes $i \in I$
shows $(pvar\ R\ i)[\hat{}]_{Pring\ R\ I}(n::nat) = mset\text{-}to\text{-}IP\ R\ (nat\text{-}to\text{-}mset\ i\ n)$
apply(*induction n*)
apply (*metis Group.nat-pow-0 R.one-mset-to-IP R.ring-axioms nat-to-mset-zero ring.Pring-one*)
proof−
fix *n*
assume *IH*: $pvar\ R\ i\ [\hat{}]_{Pring\ R\ I}\ n = mset\text{-}to\text{-}IP\ R\ (nat\text{-}to\text{-}mset\ i\ n)$
show $pvar\ R\ i\ [\hat{}]_{Pring\ R\ I}\ Suc\ n = mset\text{-}to\text{-}IP\ R\ (nat\text{-}to\text{-}mset\ i\ (Suc\ n))$
proof−
have $mset\text{-}to\text{-}IP\ R\ (nat\text{-}to\text{-}mset\ i\ (Suc\ n)) = mset\text{-}to\text{-}IP\ R\ (nat\text{-}to\text{-}mset\ i\ n) \otimes_{Pring\ R\ I} pvar\ R\ i$
using *R.monom-mult*[*of nat-to-mset i n nat-to-mset i 1*]
by (*metis One-nat-def R.Pring-mult Suc-eq-plus1 nat-to-mset-Suc nat-to-mset-add nat-to-mset-zero var-to-IP-def*)
then show *?thesis*
using *IH*
by *simp*
qed
qed

lemma *IP-to-UP-poly-eval*:
assumes $p \in Pring\text{-}set\ R\ \{i\}$
assumes *closed-fun R g*
shows $total\text{-}eval\ R\ g\ p = to\text{-}function\ R\ (IP\text{-}to\text{-}UP\ i\ p)\ (g\ i)$
apply(*rule R.indexed-pset.induct*[*of p* $\{i\}$ *carrier R*])
apply (*simp add*: *assms*(*1*); *fail*)
proof−
show $\bigwedge k.\ k \in carrier\ R \Longrightarrow total\text{-}eval\ R\ g\ (R.indexed\text{-}const\ k) = to\text{-}function\ R\ (IP\text{-}to\text{-}UP\ i\ (R.indexed\text{-}const\ k))\ (g\ i)$
proof−
fix *k*

assume *A*: $k \in$ *carrier R*
have *P0*: *total-eval R g* (*ring.indexed-const R k*) = *k*
unfolding *total-eval-def eval-in-ring-def*
using *cring.poly-eval-constant*[*of R k UNIV g*]
by (*metis A R.indexed-const-def R-cring*)
have *P1*: (*IP-to-UP i* (*ring.indexed-const R k*)) = *to-polynomial R k*
by (*meson A IP-to-UP-indexed-const*)
have *P2*: *to-function R* (*IP-to-UP i* (*ring.indexed-const R k*)) (*g i*) =
to-function R (*to-polynomial R k*) (*g i*)
using *P1* **by** *presburger*
have *P3*: *to-function R* (*to-polynomial R k*) (*g i*) = *k*
using *A assms*(*2*) *to-fun-to-poly*[*of k g i*] **unfolding** *to-fun-def* **by** *blast*
then show *total-eval R g* (*R.indexed-const k*) = *to-function R* (*IP-to-UP i* (*R.indexed-const k*)) (*g i*)
using *P0 P2* **by** *presburger*
qed
show $\bigwedge P\ Q.\ P \in$ *Pring-set R* $\{i\} \Longrightarrow$
total-eval R g P = *to-function R* (*IP-to-UP i P*) (*g i*) $\Longrightarrow$
$Q \in$ *Pring-set R* $\{i\} \Longrightarrow$
total-eval R g Q = *to-function R* (*IP-to-UP i Q*) (*g i*) $\Longrightarrow$ *total-eval R g* ($P \bigoplus Q$) = *to-function R* (*IP-to-UP i* ($P \bigoplus Q$)) (*g i*)
proof−
fix *p Q*
assume *A0*: $p \in$ *Pring-set R* $\{i\}$
assume *A1*: *total-eval R g p* = *to-function R* (*IP-to-UP i p*) (*g i*)
assume *A2*: $Q \in$ *Pring-set R* $\{i\}$
assume *A3*: *total-eval R g Q* = *to-function R* (*IP-to-UP i Q*) (*g i*)
have *total-eval R g* (*R.indexed-padd p Q*) = (*total-eval R g p*) $\oplus_R$ (*total-eval R g Q*)
using *R.total-eval-add*[*of p* $\{i\}$ *Q g*] *A0 A1*
by (*metis A2 R.Pring-add R.Pring-car assms*(*2*))
then
have *0*: *total-eval R g* ($p \bigoplus Q$) = *total-eval R g p* $\oplus$ *total-eval R g Q*
by *blast*
have *1*: *IP-to-UP i* ($p \bigoplus Q$) = *IP-to-UP i p* $\oplus_{UP\ R}$ *IP-to-UP i Q*
using *A0 A1 A3 assms A2 R.ring-axioms R-cring IP-to-UP-add*
by (*metis R.Pring-add R.Pring-car*)
have *g i* $\in$ *carrier R*
using *assms* **by** *blast*
hence *2*: *to-function R* (*IP-to-UP i* ($p \bigoplus Q$)) (*g i*) = *to-function R* (*IP-to-UP i p*) (*g i*) $\oplus$ *to-function R* (*IP-to-UP i Q*) (*g i*)
using *A0 A1 A3 assms A2 R.ring-axioms R-cring to-fun-plus*[*of IP-to-UP i p IP-to-UP i Q g i*]
IP-to-UP-closed is-UP-cring UP-cring.to-fun-def
to-fun-def 0 1
unfolding *to-fun-def P-def*
by (*smt* (*z3*) *P-def R.IP-to-UP-closed R.Pring-car to-fun-plus*)
show *total-eval R g* (*R.indexed-padd p Q*) = *to-function R* (*IP-to-UP i* (*ring.indexed-padd R p Q*)) (*g i*)

using *A0 A1 A3 assms A2 R.ring-axioms R-cring is-UP-cring to-fun-def 0 1 2*
unfolding *to-fun-def* **by** *metis*
qed
show $\bigwedge P$ *ia*.
$P \in$ *Pring-set R* $\{i\} \Longrightarrow$
total-eval R g P = to-function R (*IP-to-UP i P*) (*g i*) $\Longrightarrow$ *ia* $\in \{i\} \Longrightarrow$ *total-eval R g* ($P \bigotimes$ *ia*) = *to-function R* (*IP-to-UP i* ($P \bigotimes$ *ia*)) (*g i*)
proof−
fix *P*
fix *j*
assume *A0*: $P \in$ *Pring-set R* $\{i\}$
assume *A1*: *total-eval R g P = to-function R* (*IP-to-UP i P*) (*g i*)
assume *A2*: $j \in \{i\}$
then have *A3*: $j = i$
by *blast*
show *total-eval R g* ($P \bigotimes$ *j*) = *to-function R* (*IP-to-UP i* ($P \bigotimes$ *j*)) (*g i*)
proof−
have *LHS*: *total-eval R g* ($P \bigotimes$ *j*) = (*total-eval R g P*) $\otimes_R$ (*g i*)
using *assms A0 A3*
by (*metis R.Pring-car R-cring cring.total-eval-indexed-pmult insertI1*)
have *RHS*: *IP-to-UP i* ($P \bigotimes$ *j*)= *IP-to-UP i P* $\otimes_{UP\ R}$ *X-poly R*
by (*metis A0 A3 IP-to-UP-indexed-pmult R.Pring-car*)
have *g i* $\in$ *carrier R*
using *assms* **by** *blast*
then show *?thesis*
using *A0 A1 A3 X-closed to-fun-X*[*of g i*] *to-fun-mult*[*of IP-to-UP i P X-poly R g i*] *LHS RHS*
assms cring.axioms(*1*) *domain.axioms*(*1*)
IP-to-UP-indexed-pmult IP-to-UP-closed
Pring-car **unfolding** *to-fun-def P-def*
by (*smt* (*z3*) *P.m-comm P-def R.m-comm R-cring cring.IP-to-UP-closed ring.Pring-car to-fun-closed to-fun-def*)
qed
qed
qed
end

2.14 Mapping Univariate Polynomials to Multivariate Polynomials over a Singleton Variable Set

definition *UP-to-IP* :: ('*a*,'*b*) *ring-scheme* ⇒ '*c* ⇒ '*a u-poly* ⇒ ('*a*, '*c*) *mvar-poly*
where
UP-to-IP R i P = (λ *m*. *if* (*set-mset m*) ⊆ $\{i\}$ *then P* (*count m i*) *else* $\mathbf{0}_R$)

context *UP-cring*
begin

lemma *UP-to-IP-inv*:

```
  assumes p ∈ Pring-set R {i}
  shows UP-to-IP R i (IP-to-UP i p) = p
proof
  fix x
  show UP-to-IP R i (IP-to-UP i p) x = p x
  proof(cases (set-mset x) = {i})
    case True
    have {a. 0 < (λj. if j = i then count x i else 0) a} = {i}
     by (smt Collect-cong True count-eq-zero-iff neq0-conv singletonI singleton-conv)
  then have finite {j. (if j = i then count x i else 0) ≠ 0}
      by auto
    have (λj. if j = i then count x i else 0) = count x
    proof
      fix j
      show (if j = i then count x i else 0) = count x j
        apply(cases j = i)
        using True
        apply (simp; fail)
        using True
        by (metis count-inI singletonD)
    qed
    then have (Abs-multiset (λj. if j = i then count x i else 0)) = x
      using count-inverse
      by simp
    then show ?thesis
      unfolding UP-to-IP-def IP-to-UP-def nat-to-mset-def
      by (metis True set-eq-subset)
  next
    case False
    then show ?thesis
      apply(cases x = {#})
          apply (metis count-empty empty-subsetI IP-to-UP-def nat-to-mset-zero
set-mset-empty UP-to-IP-def)
     unfolding UP-to-IP-def IP-to-UP-def nat-to-mset-def
     using False  assms
     by (metis R.Pring-set-zero set-mset-eq-empty-iff subset-singletonD)
  qed
qed

lemma UP-to-IP-const:
  assumes a ∈ carrier R
  shows UP-to-IP R i (to-polynomial R a) = ring.indexed-const R a
proof
  fix x
  show UP-to-IP R i (to-polynomial R a) x = ring.indexed-const R a x
    apply(cases x = {#})
    unfolding UP-to-IP-def
     apply (metis R.indexed-const-def UP-ring.cfs-monom assms count-eq-zero-iff
insert-absorb insert-not-empty is-UP-ring set-mset-empty subset-insert subset-refl
```

to-polynomial-def)
by (*metis R.indexed-const-def UP-ring.cfs-monom assms count-eq-zero-iff is-UP-ring set-mset-eq-empty-iff subset-empty subset-insert to-polynomial-def*)
qed

lemma *UP-to-IP-add*:
assumes $p \in$ *carrier* (*UP R*)
assumes $Q \in$ *carrier* (*UP R*)
shows *UP-to-IP R i* ($p \oplus_{UP\ R} Q$) =
UP-to-IP R i p $\oplus_{Pring\ R\ \{i\}}$ *UP-to-IP R i Q*
proof
fix *x*
show *UP-to-IP R i* ($p \oplus_{UP\ R} Q$) *x* = (*UP-to-IP R i p* $\oplus_{Pring\ R\ \{i\}}$ *UP-to-IP R i Q*) *x*
proof(*cases set-mset x* $\subseteq \{i\}$)
case *True*
have (*UP-to-IP R i p* $\oplus_{Pring\ R\ \{i\}}$ *UP-to-IP R i Q*) *x* =
(*UP-to-IP R i p*) *x* $\oplus_R$ (*UP-to-IP R i Q*) *x*
using *True assms*
by (*metis R.Pring-add R.indexed-padd-def*)
then show *?thesis* **using** *assms True*
unfolding *UP-to-IP-def UP-def*
by (*smt partial-object.select-convs*(*1*) *restrict-def ring-record-simps*(*12*))
next
case *False*
have (*UP-to-IP R i p* $\oplus_{Pring\ R\ \{i\}}$ *UP-to-IP R i Q*) *x* =
(*UP-to-IP R i p*) *x* $\oplus_R$ (*UP-to-IP R i Q*) *x*
using *False assms*
by (*metis R.Pring-add R.indexed-padd-def*)
then show *?thesis* **using** *False assms*
unfolding *UP-to-IP-def UP-def*
using *R.l-zero R.zero-closed* **by** *presburger*
qed
qed

lemma *UP-to-IP-var*:
shows *UP-to-IP R i* (*X-poly R*) = *pvar R i*
proof
have *0*: (*count* {#*i*#} *i*) = *1*
by *simp*
have *1*: *set-mset* {#*i*#} $\subseteq \{i\}$
by *simp*
have *2*: *pvar R i* {#*i*#} = **1**
by (*metis R.mset-to-IP-simp var-to-IP-def*)
fix *x*
show *UP-to-IP R i* (*X-poly R*) *x* = *pvar R i x*
apply(*cases x* = {#*i*#})
using *X-poly-def*[*of R*] *cfs-monom*[*of* **1** *1 count x i*] *0 1 2*
unfolding *UP-to-IP-def P-def*

```
    using R.one-closed apply presburger
  proof−
    assume A: x ≠{#i#}
    then show (if set-mset x ⊆ {i} then X-poly R (count x i) else 0_R) = pvar R i
x
    proof(cases set-mset x ⊆ {i})
      case True
      have count x i ≠ 1 using True A
        by (metis One-nat-def count-empty count-inI count-single empty-iff multi-
set-eqI
          set-mset-add-mset-insert set-mset-empty set-mset-eq-empty-iff singletonD
singletonI subset-singletonD)
      then have 0: X-poly R (count x i) = 0_R
        using A UP-cring.X-closed UP-cring.degree-X UP-cring.intro True
        unfolding X-poly-def
        using P-def R.one-closed ‹1 ∈ carrier R ⟹ up-ring.monom P 1 1 (count
x i) = (if 1 = count x i then 1 else 0)› by presburger
      have pvar R i x = 0_R
        using A var-to-IP-def
        by (metis R.mset-to-IP-simp′)
      then show ?thesis
        using A 0 by presburger
    next
      case False
      have pvar R i x = 0_R using A var-to-IP-def False
        by (metis 1 R.Pring-set-zero R.mset-to-IP-closed)
      then show ?thesis
        unfolding UP-to-IP-def
        using False by presburger
    qed
  qed
qed

lemma UP-to-IP-var-pow:
  shows UP-to-IP R i ((X-poly R)[^]_UP R (n::nat)) = (pvar R i)[^]_Pring R {i}^n
proof
  fix x
  show UP-to-IP R i (X-poly R [^]_UP R n) x = (pvar R i [^]_Pring R {i} n) x
  proof(cases set-mset x ⊆ {i})
    case True
    show ?thesis
    proof(cases count x i = n)
      case T: True
      then have 0: x = nat-to-mset i n
        using True
        by (metis count-inI emptyE insert-iff multiset-eqI nat-to-msetE
            nat-to-msetE′ subsetD)
      have 1: x = nat-to-mset i (count x i)
        using 0 T by auto
```

```
    then have LHS: (pvar R i [^]_Pring R {i} n) x = 1_R
      using T True 0 1
      by (metis R.mset-to-IP-simp insertI1 pvar-pow)
    have 2: UP-to-IP R i (X-poly R [^]_UP R n) x = (up-ring.monom (UP R) 1
n) (count x i)
      unfolding UP-to-IP-def X-poly-def using True
      by (metis ctrm-degree P.nat-pow-closed P.nat-pow-eone P-def R.one-closed
UP-cring.monom-coeff
       UP-one-closed UP-r-one X-closed is-UP-cring monom-one monom-rep-X-pow
to-poly-inverse
          to-poly-mult-simp(2))
    then show ?thesis
      using True T LHS P-def R.one-closed cfs-monom
      by presburger
  next
    case False
    have (pvar R i [^]_Pring R {i} n) = mset-to-IP R (nat-to-mset i n)
      by (simp add: pvar-pow)
    hence 0: (pvar R i [^]_Pring R {i} n) x = 0
      by (metis False R.mset-to-IP-simp' nat-to-msetE)
    have 1: UP-to-IP R i (X-poly R [^]_UP R n) x = (up-ring.monom (UP R) 1
n) (count x i)
      unfolding UP-to-IP-def X-poly-def using False True
      by (metis ctrm-degree P.nat-pow-closed P.nat-pow-eone P-def R.one-closed
UP-one-closed
       UP-r-one X-closed cfs-monom monom-one monom-rep-X-pow to-poly-inverse
to-poly-mult-simp(2))
    thus ?thesis using True False
      unfolding UP-to-IP-def X-poly-def 0
      by (metis P-def R.one-closed cfs-monom)
  qed
 next
  case False
  then have 0: UP-to-IP R i (X-poly R [^]_UP R n) x = 0
    unfolding UP-to-IP-def
    by meson
  have (pvar R i [^]_Pring R {i} n) = mset-to-IP R (nat-to-mset i n)
    by (simp add: pvar-pow)
  hence (pvar R i [^]_Pring R {i} n) x = 0
   by (metis False R.mset-to-IP-simp' count-eq-zero-iff nat-to-msetE' singleton-iff
subsetI)
  then show ?thesis using 0
    by presburger
 qed
qed

lemma one-var-indexed-poly-monom-simp:
 assumes a ∈ carrier R
```

shows $(a \odot_{Pring\ R\ \{i\}} ((pvar\ R\ i)\ [\widehat{}]_{Pring\ R\ \{i\}}\ n))\ x = (if\ x = (nat\text{-}to\text{-}mset\ i\ n)\ then\ a\ else\ \mathbf{0})$
proof−
have *0*: $(a \odot_{Pring\ R\ \{i\}} ((pvar\ R\ i)\ [\widehat{}]_{Pring\ R\ \{i\}}\ n))\ x =$
$a \otimes (((pvar\ R\ i)\ [\widehat{}]_{Pring\ R\ \{i\}}\ n)\ x)$
using *Pring-smult-cfs Pring-var-closed assms cring-def is-cring monoid.nat-pow-closed ring.Pring-is-monoid singletonI*
by (*simp add*: *monoid.nat-pow-closed ring.Pring-is-monoid R.Pring-smult-cfs R.Pring-var-closed R.ring-axioms*)
have *1*: $(pvar\ R\ i)\ [\widehat{}]_{Pring\ R\ \{i\}}\ n = mset\text{-}to\text{-}IP\ R\ (nat\text{-}to\text{-}mset\ i\ n)$
using *insertI1*
by (*simp add*: *pvar-pow*)
then have *1*: $(a \odot_{Pring\ R\ \{i\}} ((pvar\ R\ i)\ [\widehat{}]_{Pring\ R\ \{i\}}\ n))\ x =$
$a \otimes (mset\text{-}to\text{-}IP\ R\ (nat\text{-}to\text{-}mset\ i\ n)\ x)$
using *0* **by** *presburger*
show *?thesis*
using *assms 1* **unfolding** *mset-to-IP-def*
using *r-null r-one* **by** *simp*
qed

lemma *UP-to-IP-monom*:
assumes $a \in carrier\ R$
shows $UP\text{-}to\text{-}IP\ R\ i\ (up\text{-}ring.monom\ (UP\ R)\ a\ n) = a \odot_{Pring\ R\ \{i\}} ((pvar\ R\ i)[\widehat{}]_{Pring\ R\ \{i\}} n)$
proof
fix *x*
show $UP\text{-}to\text{-}IP\ R\ i\ (up\text{-}ring.monom\ (UP\ R)\ a\ n)\ x = (a \odot_{Pring\ R\ \{i\}} ((pvar\ R\ i)\ [\widehat{}]_{Pring\ R\ \{i\}}\ n))\ x$
proof(*cases set-mset x* $\subseteq \{i\}$)
case *True*
then show *?thesis*
proof(*cases count x i* = *n*)
case *T*: *True*
then have *x* = *nat-to-mset i n*
using *True*
by (*metis count-inI emptyE insert-iff multiset-eqI nat-to-msetE nat-to-msetE' subsetD*)
then have *LHS*: $(a \odot_{Pring\ R\ \{i\}} ((pvar\ R\ i)\ [\widehat{}]_{Pring\ R\ \{i\}}\ n))\ x = a$
using *assms*
by (*simp add*: *one-var-indexed-poly-monom-simp*)
then show *?thesis*
unfolding *UP-to-IP-def*
using *T True assms*(*1*)
by (*metis UP-ring.cfs-monom is-UP-ring*)
next
case *False*
then show *?thesis* **using** *True*

```
      unfolding UP-to-IP-def
    by (metis INTEG.R.nat-to-msetE P-def assms cfs-monom one-var-indexed-poly-monom-simp)
  qed
 next
  case False
  then show ?thesis
   unfolding UP-to-IP-def
   by (metis (no-types, opaque-lifting) one-var-indexed-poly-monom-simp assms
       count-eq-zero-iff equalityD2 insert-subset nat-to-msetE' subsetI subset-eq)
 qed
qed

lemma UP-to-IP-monom':
 assumes a ∈ carrier R
 shows UP-to-IP R i (up-ring.monom (UP R) a n) = a ⊙_Pring R {i} ((pvar R
i)[^]_Pring R {i} n)
 by (metis R.Pring-smult UP-to-IP-monom assms)

lemma UP-to-IP-closed:
 assumes p ∈ carrier P
 shows (UP-to-IP R i p) ∈ carrier (Pring R {i})
 apply(rule poly-induct3[of ])
 using assms apply blast
 apply (metis P-def R.Pring-add-closed UP-to-IP-add)
proof−
 fix a fix n::nat
 assume A0: a ∈ carrier R
 have (pvar R i [^]_Pring R {i} n) ∈ carrier (Pring R {i})
  using pvar-closed[of R ] monoid.nat-pow-closed[of Pring R {i}]
 proof −
  show ?thesis
   by (meson R.Pring-is-monoid R.Pring-var-closed monoid.nat-pow-closed sin-
gleton-iff)
 qed
 then show a ∈ carrier R ⟹
       UP-to-IP R i (up-ring.monom P a n) ∈ carrier (Pring R {i})
  using A0 assms(1) UP-to-IP-monom[of a i n] cring.poly-scalar-mult-closed [of
R a - {i}]
  by (metis P-def R.Pring-smult-closed)
qed

lemma IP-to-UP-inv:
 assumes p ∈ carrier P
 shows IP-to-UP i (UP-to-IP R i p) = p
 apply(rule poly-induct3[of ])
 using assms apply linarith
proof−
 show ⋀p q. q ∈ carrier P ⟹
       p ∈ carrier P ⟹
```

```
        IP-to-UP i (UP-to-IP R i p) = p ⟹
        IP-to-UP i (UP-to-IP R i q) = q ⟹
        IP-to-UP i (UP-to-IP R i (p ⊕_P q)) = p ⊕_P q
  proof−
    fix p q assume A:
        q ∈ carrier P
        p ∈ carrier P
        IP-to-UP i (UP-to-IP R i p) = p
        IP-to-UP i (UP-to-IP R i q) = q
    show IP-to-UP i (UP-to-IP R i (p ⊕_P q)) = p ⊕_P q
      using A UP-to-IP-add[of p q i]
            UP-to-IP-closed
            IP-to-UP-add
      unfolding P-def
      by metis
  qed
  show ⋀a n. a ∈ carrier R ⟹
        IP-to-UP i (UP-to-IP R i (up-ring.monom P a n)) =
        up-ring.monom P a n
  proof−
    fix a fix n::nat
    assume A0: a ∈ carrier R
    have A1: pvar R i [^]_Pring R {i} n ∈ carrier (Pring R {i})
      using pvar-closed monoid.nat-pow-closed
      by (metis R.Pring-is-monoid R-cring singletonI)
    have UP-to-IP R i (up-ring.monom (UP R) a n) = a ⊙_Pring R {i} (pvar R i
[^]_Pring R {i} n)
      by (meson A0 UP-to-IP-monom')
    then have A2: IP-to-UP i (UP-to-IP R i (up-ring.monom (UP R) a n)) =
          IP-to-UP i (a ⊙_Pring R {i} (pvar R i [^]_Pring R {i} n))
      by presburger
    have A3: IP-to-UP i (pvar R i [^]_Pring R {i} n) = (up-ring.monom P 1 n)
    proof(induction n)
      case 0
      then show ?case
        by (metis Group.nat-pow-0 IP-to-UP-one P-def monom-one)
    next
      case (Suc n)

      then show ?case
      using IP-to-UP-ring-hom[of i]
            ring-hom-mult[of IP-to-UP i Pring R {i} UP R pvar R i pvar R i
[^]_Pring R {i} n]
            ring-hom-ring.homh[of Pring R {i} UP R IP-to-UP i]
        by (metis IP-to-UP-monom P.l-one P.nat-pow-closed P-def R.one-closed
UP-cring.ctrm-degree UP-cring.monom-rep-X-pow UP-one-closed X-closed cfs-monom
is-UP-cring monom-one pvar-pow singletonI to-poly-inverse to-poly-mult-simp(1))
    qed
```

```
    then show IP-to-UP i (UP-to-IP R i (up-ring.monom P a n)) =
          up-ring.monom P a n
      using A2 IP-to-UP-scalar-mult[of a pvar R i [^]Pring R {i} n i]
          A0 A1 P-def monic-monom-smult by presburger
  qed
qed

lemma UP-to-IP-mult:
  assumes p ∈ carrier (UP R)
  assumes Q ∈ carrier (UP R)
  shows UP-to-IP R i (p ⊗UP R Q) =
        UP-to-IP R i p ⊗Pring R {i} UP-to-IP R i Q
proof−
  have 0: IP-to-UP i (UP-to-IP R i (p ⊗UP R Q)) = (p ⊗UP R Q)
    by (meson UP-cring.IP-to-UP-inv UP-ring.UP-mult-closed assms(1) assms(2)
is-UP-cring is-UP-ring)
  have 1: IP-to-UP i (UP-to-IP R i p ⊗Pring R {i} UP-to-IP R i Q) =
       IP-to-UP i (UP-to-IP R i p) ⊗UP R IP-to-UP i ( UP-to-IP R i Q)
    using IP-to-UP-ring-hom[of i]
         ring-hom-mult[of IP-to-UP i]
         UP-to-IP-closed assms
    by (smt P-def ring-hom-ring.homh)
  have 2: IP-to-UP i (UP-to-IP R i (p ⊗UP R Q)) =
       IP-to-UP i (UP-to-IP R i p ⊗Pring R {i} UP-to-IP R i Q)
    using 0 1 assms
    by (metis UP-cring.IP-to-UP-inv is-UP-cring)
  then show ?thesis
   by (metis 0 P-def R.Pring-mult-closed R.ring-axioms assms(1) assms(2) ring.Pring-car
UP-to-IP-closed UP-to-IP-inv)
qed

lemma UP-to-IP-ring-hom:
shows ring-hom-ring (UP R) (Pring R {i}) (UP-to-IP R i)
  apply(rule ring-hom-ringI)
  using P-def UP-ring apply force
  apply (simp add: R.Pring-is-ring; fail)
  apply (metis P-def UP-to-IP-closed)
  apply (meson UP-to-IP-mult)
  apply (meson UP-to-IP-add)
  by (metis IP-to-UP-one R.Pring-car R.Pring-one-closed UP-to-IP-inv)

end
```

2.14.1 The isomorphism $R[I \cup J] \sim R[I][J]$, where I and J are disjoint variable sets

Given a ring R and variable sets I and J, we'd like to construct the canonical (iso)morphism $R[I \cup J] \to R[I][J]$. This can be done with the univeral prop-

erty of the previous section. Let $\phi : R \to R[J]$ be the inclusion of constants, and $f : J \to R[I]$ be the map which sends the variable i to the polynomial variable i over the ring $R[I][J]$. Then these are the two basic pieces of input required to give us a canonical homomoprhism $R[I \cup J] \to R[I][J]$ with the universal property. The first map ϕ will be "`dist_varset_morpshim`" below, and the second map will be "`dist_varset_var_ass`". The desired induced isomorphism will be called "`var_factor`".

definition(**in** *ring*) *dist-varset-morphism*
:: *'d set* ⇒ *'d set* ⇒
('*a*, (('*a*, '*d*) *mvar-poly*, '*d*) *mvar-poly*) *ring-hom* **where**
dist-varset-morphism (*I*:: '*d set*) (*J*:: '*d set*) =
(*ring.indexed-const* (*Pring R J*) :: ('*d multiset* ⇒ '*a*) ⇒ '*d multiset* ⇒ ('*d multiset* ⇒ '*a*))∘ (*ring.indexed-const R* ::'*a* ⇒ '*d multiset* ⇒ '*a*)

definition(**in** *ring*) *dist-varset-var-ass*
:: *'d set* ⇒ *'d set* ⇒ *'d* ⇒ (('*a*, '*d*) *mvar-poly*, '*d*) *mvar-poly*
where
dist-varset-var-ass (*I*:: '*d set*) (*J*:: '*d set*) = (λ*i*. *if i* ∈ *J then ring.indexed-const* (*Pring R J*) (*pvar R i*) *else*
pvar (*Pring R J*) *i*)

context *cring*
begin

lemma *dist-varset-morphism-is-morphism*:
assumes (*I*:: '*d set*) ⊆ *J0* ∪ *J1*
assumes *J1* ⊆ *I*
assumes φ = *dist-varset-morphism I J0*
shows *ring-hom-ring R* (*Pring* (*Pring R J0*) *J1*) φ
proof−
have *0*:*ring-hom-ring R* (*Pring R J0*) *indexed-const*
by (*simp add*: *indexed-const-ring-hom*)
have *1*:*ring-hom-ring* (*Pring R J0*) (*Pring* (*Pring R J0*) *J1*) (*ring.indexed-const* (*Pring R J0*))
by (*simp add*: *cring.indexed-const-ring-hom is-cring local.ring-axioms ring.Pring-is-cring*)
show *?thesis* **using** *0 1 assms ring-hom-trans*[*of indexed-const R Pring R J0 ring.indexed-const* (*Pring R J0*)
(*Pring* (*Pring R J0*) *J1*)]
unfolding *dist-varset-morphism-def*
by (*meson ring-hom-ring.axioms*(*1*) *ring-hom-ring.axioms*(*2*) *ring-hom-ring.homh ring-hom-ringI2*)
qed

definition *var-factor* ::
'd set ⇒ *'d set* ⇒ *'d set* ⇒
(('*a*, '*d*) *mvar-poly*, (('*a*, '*d*) *mvar-poly*, '*d*) *mvar-poly*) *ring-hom* **where**
var-factor (*I*:: '*d set*) (*J0*:: '*d set*) (*J1*:: '*d set*) = *indexed-poly-induced-morphism I* (*Pring* (*Pring R J0*) *J1*)

(*dist-varset-morphism I J0*)
(*dist-varset-var-ass I J0*)

lemma *indexed-const-closed*:
assumes $x \in$ *carrier R*
shows *indexed-const x* $\in$ *carrier* (*Pring R I*)
using *Pring-car assms indexed-pset.indexed-const* **by** *blast*

lemma *var-factor-morphism*:
assumes (*I*:: *'d set*) $\subseteq$ *J0* $\cup$ *J1*
assumes *J1* $\subseteq$ *I*
assumes *J1* $\cap$ *J0* = {}
assumes *g* = *dist-varset-var-ass I J0*
assumes φ = *dist-varset-morphism I J0*
assumes ψ = (*var-factor I J0 J1*)
shows *ring-hom-ring* (*Pring R I*) (*Pring* (*Pring R J0*) *J1*) ψ
$\bigwedge i.\ i \in J0 \cap I \Longrightarrow \psi$ (*pvar R i*) = *ring.indexed-const* (*Pring R J0*) (*pvar R i*)
$\bigwedge i.\ i \in J1 \Longrightarrow \psi$ (*pvar R i*) = *pvar* (*Pring R J0*) *i*
$\bigwedge a.\ a \in$ *carrier* (*Pring R* (*J0* $\cap$ *I*)) $\Longrightarrow \psi$ *a* = *ring.indexed-const* (*Pring R J0*) *a*
proof−
have *0*: *g* $\in$ *I* $\rightarrow$ *carrier* (*Pring* (*Pring R J0*) *J1*)
proof
fix *x*
assume *A0*: *x* $\in$ *I*
then have *A1*: *x* $\in$ *J0* $\vee$ *x* $\in$ *J1*
by (*meson UnE assms*(*1*) *subsetD*)
have *A2*: *x* $\notin$ *J0* $\Longrightarrow$ *x* $\in$ *J1*
using *A1* **by** *blast*
show *g x* $\in$ *carrier* (*Pring* (*Pring R J0*) *J1*)
apply(*cases x* $\in$ *J0*)
using *assms A0 A1 A2 pvar-closed*[*of R x J0*]
pvar-closed[*of Pring R J0 x J1*]
cring.indexed-const-closed[*of Pring R J0*]
unfolding *dist-varset-var-ass-def*
apply (*smt Pring-is-cring is-cring*)
using *assms A0 A1 A2 pvar-closed*[*of R x J0*]
pvar-closed[*of Pring R J0 x J1*]
cring.indexed-const-closed[*of Pring R J0*]
unfolding *dist-varset-var-ass-def*
by (*smt Pring-is-cring is-cring*)
qed
have *1*: *cring* (*Pring R J0*)
by (*simp add*: *Pring-is-cring is-cring*)
have *2*: *cring* (*Pring* (*Pring R J0*) *J1*)
by (*simp add*: *1 Pring-is-ring ring.Pring-is-cring*)
show *C0*: *ring-hom-ring* (*Pring R I*) (*Pring* (*Pring R J0*) *J1*) ψ
using *0 assms Pring-universal-prop*[*of Pring* (*Pring R J0*) *J1 g I* φ ψ]

dist-varset-morphism-is-morphism[*of I J0 J1*]
unfolding *var-factor-def*
by (*meson Pring-is-cring Pring-is-ring is-cring ring.Pring-is-cring*)
show *C1*: $\bigwedge i.\ i \in J0 \cap I \Longrightarrow \psi\ (pvar\ R\ i) = ring.indexed\text{-}const\ (Pring\ R\ J0)\ (pvar\ R\ i)$
using *0 1 2 assms Pring-universal-prop*(*2*)[*of Pring* (*Pring R J0*) *J1 g I* φ ψ]
dist-varset-morphism-is-morphism[*of I J0 J1* φ] *dist-varset-var-ass-def*
dist-varset-morphism-def **unfolding** *var-factor-def var-to-IP-def*
by (*smt IntE dist-varset-var-ass-def inf-commute inf-le2 ring.Pring-is-cring subsetD var-to-IP-def*)
have *3*: $\bigwedge i.\ i \in J1 \Longrightarrow g\ i = mset\text{-}to\text{-}IP\ (Pring\ R\ J0)\ \{\#i\#\}$
using *assms* **unfolding** *dist-varset-var-ass-def var-to-IP-def*
by (*meson disjoint-iff-not-equal*)
show *C2*: $\bigwedge i.\ i \in J1 \Longrightarrow \psi\ (pvar\ R\ i) = pvar\ (Pring\ R\ J0)\ i$
using *0 1 2 3 assms Pring-universal-prop*(*2*)[*of Pring* (*Pring R J0*) *J1 g I* φ ψ]
dist-varset-morphism-is-morphism[*of I J0 J1* φ]
unfolding *var-factor-def var-to-IP-def*
by (*metis subsetD*)
have *4*: $\bigwedge k.\ k \in carrier\ R \Longrightarrow \psi\ (indexed\text{-}const\ k) = ring.indexed\text{-}const\ (Pring\ R\ J0)\ (indexed\text{-}const\ k)$
using *0 1 2 3 assms Pring-universal-prop*(*3*)[*of Pring* (*Pring R J0*) *J1 g I* φ ψ]
dist-varset-morphism-is-morphism[*of I J0 J1* φ] *comp-apply*
unfolding *var-factor-def var-to-IP-def dist-varset-morphism-def*
by *metis*
show *C3*: $\bigwedge a.\ a \in carrier\ (Pring\ R\ (J0 \cap I)) \Longrightarrow \psi\ a = ring.indexed\text{-}const\ (Pring\ R\ J0)\ a$
proof− **fix** *a* **assume** *A*: $a \in carrier\ (Pring\ R\ (J0 \cap I))$
show $\psi\ a = ring.indexed\text{-}const\ (Pring\ R\ J0)\ a$
apply(*rule indexed-pset.induct*[*of a J0* $\cap$ *I carrier R*])
using *A Pring-car* **apply** *blast*
using *4*
apply *blast*
proof−
show $\bigwedge P\ Q.\ P \in Pring\text{-}set\ R\ (J0 \cap I) \Longrightarrow \psi\ P = ring.indexed\text{-}const\ (Pring\ R\ J0)\ P \Longrightarrow Q \in Pring\text{-}set\ R\ (J0 \cap I) \Longrightarrow \psi\ Q = ring.indexed\text{-}const\ (Pring\ R\ J0)\ Q \Longrightarrow \psi\ (P \bigoplus Q) = ring.indexed\text{-}const\ (Pring\ R\ J0)\ (P \bigoplus Q)$
proof− **fix** *P Q*
assume *A0*: $P \in Pring\text{-}set\ R\ (J0 \cap I)$
assume *A1*: $\psi\ P = ring.indexed\text{-}const\ (Pring\ R\ J0)\ P$
assume *A2*: $Q \in Pring\text{-}set\ R\ (J0 \cap I)$
assume *A3*: $\psi\ Q = ring.indexed\text{-}const\ (Pring\ R\ J0)\ Q$
have *A0'*: $P \in Pring\text{-}set\ R\ I$
using *A0*
by (*meson Int-lower2 Pring-carrier-subset subsetD*)
have *A1'*: $Q \in Pring\text{-}set\ R\ I$
by (*meson A2 Int-lower2 Pring-carrier-subset subsetD*)

have B: $\psi\ (P \bigoplus Q) = \psi\ P \oplus_{Pring\ (Pring\ R\ J0)\ J1} \psi\ Q$
using $A0'\ A1'\ A2\ A3\ C0$ *assms ring-hom-add*
by (*metis* (*no-types*, *lifting*) *Pring-add local.ring-axioms ring.Pring-car ring-hom-ring.homh*)
have *ring.indexed-const* (*Pring R J0*) ($P \bigoplus Q$) = *ring.indexed-const* (*Pring R J0*) P
$\oplus_{Pring\ (Pring\ R\ J0)\ J1}$ *ring.indexed-const* (*Pring R J0*) Q
by (*simp add*: *Pring-add Pring-is-ring ring.Pring-add ring.indexed-padd-const*)
then show $\psi\ (P \bigoplus Q)$ = *ring.indexed-const* (*Pring R J0*) ($P \bigoplus Q$)
using B
by (*simp add*: *A1 A3*)
qed
show $\bigwedge P\ i.\ P \in$ *Pring-set R* (*J0* $\cap$ *I*) $\Longrightarrow \psi\ P =$ *ring.indexed-const* (*Pring R J0*) $P \Longrightarrow$
$i \in J0 \cap I \Longrightarrow \psi\ (P \bigotimes i) =$ *ring.indexed-const* (*Pring R J0*) ($P \bigotimes i$)
proof− **fix** $P\ i$ **assume** $A0$: $P \in$ *Pring-set R* (*J0* $\cap$ *I*)
assume $A1$: $\psi\ P =$ *ring.indexed-const* (*Pring R J0*) P
assume $A2$: $i \in J0 \cap I$
have $A0'$: $P \in$ *carrier* (*Pring R I*)
using *A0 Pring-carrier-subset*
by (*metis* (*no-types*, *opaque-lifting*) *Pring-car in-mono inf-commute le-inf-iff subset-refl*)
have $A0''$: $P \in$ *carrier* (*Pring R J0*)
using *A0 Pring-carrier-subset*
by (*metis* (*no-types*, *opaque-lifting*) *Pring-car in-mono inf-commute le-inf-iff subset-refl*)
have $\psi\ (P \bigotimes i) = \psi\ P \otimes_{Pring\ (Pring\ R\ J0)\ J1}$ *ring.indexed-const* (*Pring R J0*) (*pvar R i*)
proof−
have ($P \bigotimes i$) = $P \otimes_{Pring\ R\ I}$ *pvar R i*
using $A0'\ A2$ **unfolding** *var-to-IP-def*
by (*metis A0 Pring-mult poly-index-mult*)
then show *?thesis*
using $A0'\ C0\ A2\ C1$[*of i*] *ring-hom-ring.homh*
ring-hom-mult[*of* ψ *Pring R I Pring* (*Pring R J0*) *J1 P pvar R i*]
by (*metis IntE Pring-var-closed*)
qed
then have $\psi\ (P \bigotimes i) =$ *ring.indexed-const* (*Pring R J0*) $P \otimes_{Pring\ (Pring\ R\ J0)\ J1}$ *ring.indexed-const* (*Pring R J0*) (*pvar R i*)
by (*simp add*: *A1*)
then have $\psi\ (P \bigotimes i) =$ *ring.indexed-const* (*Pring R J0*) ($P \otimes_{Pring\ R\ J0}$ (*pvar R i*))
using $A0''\ A2$ *cring.indexed-const-ring-hom*[*of Pring R J0 J1*] *ring-hom-ring.homh*
ring-hom-mult[*of ring.indexed-const* (*Pring R J0*) *Pring R J0* - *P pvar R i*]
by (*smt 1 IntE Pring-var-closed*)
then show $\psi\ (P \bigotimes i) =$ *ring.indexed-const* (*Pring R J0*) ($P \bigotimes i$)

using *poly-index-mult*[*of P J0 i*] **unfolding** *var-to-IP-def*
by (*metis A0'' A2 IntE Pring-car Pring-mult*)
qed
qed
qed
qed

lemma *var-factor-morphism'*:
assumes $I = J0 \cup J1$
assumes $J1 \subseteq I$
assumes $J1 \cap J0 = \{\}$
assumes $\psi = (\textit{var-factor } I\ J0\ J1)$
shows *ring-hom-ring* (*Pring R I*) (*Pring* (*Pring R J0*) *J1*) ψ
$\bigwedge i.\ i \in J1 \Longrightarrow \psi\ (\textit{pvar } R\ i) = \textit{pvar}\ (\textit{Pring } R\ J0)\ i$
$\bigwedge a.\ a \in \textit{carrier}\ (\textit{Pring } R\ (J0 \cap I)) \Longrightarrow \psi\ a = \textit{ring.indexed-const}\ (\textit{Pring } R\ J0)\ a$
using *assms var-factor-morphism*
apply *blast*
using *assms var-factor-morphism*(*3*)
apply (*metis subset-refl*)
using *assms var-factor-morphism*(*4*)
by (*metis Un-subset-iff Un-upper1*)

Constructing the inverse morphism for `var_factor_morphism`

lemma *pvar-ass-closed*:
assumes $J1 \subseteq I$
shows $\textit{pvar } R \in J1 \to \textit{carrier}\ (\textit{Pring } R\ I)$
by (*meson Pi-I Pring-var-closed assms subsetD*)

The following function gives us the inverse morphism $R[I][J] \to R[I \cup J]$:

definition *var-factor-inv* :: *'d set* ⇒ *'d set* ⇒ *'d set* ⇒
((('*a*, *'d*) *mvar-poly*, *'d*) *mvar-poly*, (*'a*, *'d*) *mvar-poly*) *ring-hom* **where**
var-factor-inv (*I*:: *'d set*) (*J0*:: *'d set*) (*J1*:: *'d set*) = *indexed-poly-induced-morphism J1* (*Pring R I*)
(*id*:: (*'d multiset* ⇒ *'a*) ⇒ *'d multiset* ⇒ *'a*)
(*pvar R*)

lemma *var-factor-inv-morphism*:
assumes $I = J0 \cup J1$
assumes $J1 \subseteq I$
assumes $J1 \cap J0 = \{\}$
assumes $\psi = (\textit{var-factor-inv } I\ J0\ J1)$
shows *ring-hom-ring* (*Pring* (*Pring R J0*) *J1*) (*Pring R I*) ψ
$\bigwedge i.\ i \in J1 \Longrightarrow \psi\ (\textit{pvar}\ (\textit{Pring } R\ J0)\ i) = \textit{pvar } R\ i$
$\bigwedge a.\ a \in \textit{carrier}\ (\textit{Pring } R\ J0) \Longrightarrow \psi\ (\textit{ring.indexed-const}\ (\textit{Pring } R\ J0)\ a) = a$
proof−
have *0*: *ring-hom-ring* (*Pring R J0*) (*Pring R I*) *id*

apply(*rule ring-hom-ringI*)
apply (*simp add*: *Pring-is-ring*; *fail*)
apply (*simp add*: *Pring-is-ring*;*fail*)
apply (*metis Pring-car Pring-carrier-subset Un-upper1 assms*(*1*) *id-apply subsetD*)
apply (*metis Pring-mult-eq id-apply*)
apply (*metis Pring-add-eq id-apply*)
by (*simp add*: *Pring-one-eq*)
then show *ring-hom-ring* (*Pring* (*Pring R J0*) *J1*) (*Pring R I*) ψ
using *cring.Pring-universal-prop*(*1*)[*of Pring R J0 Pring R I pvar R J1 id* ψ] *pvar-ass-closed*[*of J0 I*]
by (*metis Pring-is-cring assms*(*2*) *assms*(*4*) *is-cring pvar-ass-closed var-factor-inv-def*)
show $\bigwedge i.\ i \in J1 \Longrightarrow \psi$ (*pvar* (*Pring R J0*) *i*) = *pvar R i*
using *0 assms pvar-ass-closed*[*of J0 I*] *cring.Pring-universal-prop*(*2*)[*of Pring R J0 Pring R I pvar R J1 id* ψ]
by (*metis Pring-is-cring is-cring pvar-ass-closed var-factor-inv-def var-to-IP-def*)
show $\bigwedge a.\ a \in$ *carrier* (*Pring R J0*) $\Longrightarrow \psi$ (*ring.indexed-const* (*Pring R J0*) *a*) = *a*
using *0 assms pvar-ass-closed*[*of J0 I*] *cring.Pring-universal-prop*(*3*)[*of Pring R J0 Pring R I pvar R J1 id* ψ]
by (*smt Pi-I Pring-is-cring Pring-var-closed id-def is-cring subsetD var-factor-inv-def*)
qed

lemma *var-factor-inv-inverse*:
assumes $I = J0 \cup J1$
assumes $J1 \subseteq I$
assumes $J1 \cap J0 = \{\}$
assumes $\psi 1$ = (*var-factor-inv I J0 J1*)
assumes $\psi 0$ = (*var-factor I J0 J1*)
assumes $P \in$ *carrier* (*Pring R I*)
shows $\psi 1\ (\psi 0\ P) = P$
apply(*rule indexed-pset.induct*[*of P I carrier R*])
using *Pring-car assms*(*6*) **apply** *blast*
using *var-factor-inv-morphism*(*3*)[*of I J0 J1* $\psi 1$] *var-factor-morphism'*(*3*)[*of I J0 J1* $\psi 0$] *assms*
apply (*metis indexed-const-closed inf-sup-absorb*)
proof−
have *0*: *ring-hom-ring* (*Pring* (*Pring R J0*) *J1*) (*Pring R I*) $\psi 1$
by (*simp add*: *assms*(*1*) *assms*(*3*) *assms*(*4*) *var-factor-inv-morphism*(*1*))
have *1*: *ring-hom-ring* (*Pring R I*) (*Pring* (*Pring R J0*) *J1*) $\psi 0$
by (*simp add*: *assms*(*1*) *assms*(*3*) *assms*(*5*) *var-factor-morphism'*(*1*))
have *2*: $\psi 1 \circ \psi 0 \in$ *ring-hom* (*Pring R I*) (*Pring R I*)
using *0 1 ring-hom-trans*[*of* $\psi 0$ *Pring R I Pring* (*Pring R J0*) *J1* $\psi 1$ *Pring R I*] *ring-hom-ring.homh*[*of Pring R I Pring* (*Pring R J0*) *J1* $\psi 0$] *ring-hom-ring.homh*[*of Pring* (*Pring R J0*) *J1 Pring R I* $\psi 1$]
by *blast*
show $\bigwedge P\ Q.\ P \in$ *Pring-set R I* $\Longrightarrow$ $\psi 1\ (\psi 0\ P) = P \Longrightarrow Q \in$ *Pring-set R I* $\Longrightarrow \psi 1\ (\psi 0\ Q) = Q \Longrightarrow \psi 1\ (\psi 0$

$(P \bigoplus Q)) = P \bigoplus Q$
proof− **fix** P Q **assume** $A0$: $P \in$ *Pring-set* R I $\psi 1$ $(\psi 0\ P) = P$
assume $A1$: $Q \in$ *Pring-set* R I $\psi 1$ $(\psi 0\ Q) = Q$
show $\psi 1$ $(\psi 0$ $(P \bigoplus Q)) = P \bigoplus Q$
using $A0$ $A1$ 2 *ring-hom-add*[*of* $\psi 1 \circ \psi 0$ *Pring* R I *Pring* R I P Q] *comp-apply*[*of* $\psi 1$ $\psi 0$]
by (*simp add*: 2 ‹$P \in$ *Pring-set* R I› ‹$Q \in$ *Pring-set* R I› *Pring-add Pring-car*)
qed
show $\bigwedge P$ i. $P \in$ *Pring-set* R $I \Longrightarrow \psi 1$ $(\psi 0\ P) = P \Longrightarrow i \in I \Longrightarrow \psi 1$ $(\psi 0$ $(P \bigotimes i)) = P \bigotimes i$
proof−
fix P i **assume** A: $P \in$ *Pring-set* R I $\psi 1$ $(\psi 0\ P) = P$ $i \in I$
show $\psi 1$ $(\psi 0$ $(P \bigotimes i)) = P \bigotimes i$
proof−
have $A0$: $P \bigotimes i = P \otimes_{Pring\ R\ I}$ *pvar* R i
by (*metis* $A(1)$ $A(3)$ *Pring-mult local.ring-axioms ring.poly-index-mult var-to-IP-def*)
have $A1$: $\psi 1$ $(\psi 0$ $(pvar\ R\ i)) = pvar\ R\ i$
by (*metis* $A(3)$ *Int-iff Pring-var-closed UnE Un-subset-iff Un-upper1* $assms(1)$ $assms(2)$ $assms(3)$ $assms(4)$ $assms(5)$ $cring.var\text{-}factor\text{-}morphism(2)$ *is-cring* $var\text{-}factor\text{-}inv\text{-}morphism(2)$ $var\text{-}factor\text{-}inv\text{-}morphism(3)$ $var\text{-}factor\text{-}morphism'(2)$)
then show *?thesis*
using 2 *A0 A ring-hom-mult*[*of* $\psi 1 \circ \psi 0$ *Pring* R I - P *pvar* R i] *Pring-car Pring-var-closed comp-apply*[*of* $\psi 1$ $\psi 0$]
by *smt*
qed
qed
qed

lemma *var-factor-total-eval*:
assumes $I = J0 \cup J1$
assumes $J1 \subseteq I$
assumes $J1 \cap J0 = \{\}$
assumes $\psi = (var\text{-}factor\ I\ J0\ J1)$
assumes *closed-fun* R g
assumes $P \in carrier\ (Pring\ R\ I)$
shows *total-eval* R g P = *total-eval* R g (*total-eval* (*Pring* R *J0*) (*indexed-const* $\circ$ g) (ψ P))
apply(*rule indexed-pset.induct*[*of* P I *carrier* R])
using *Pring-car assms* **apply** *blast*
apply (*metis Pring-is-cring* $assms(1)$ $assms(2)$ $assms(3)$ $assms(4)$ $cring.total\text{-}eval\text{-}const$ *indexed-const-closed is-cring* $var\text{-}factor\text{-}morphism'(3)$)
proof−
show $\bigwedge P$ Q. $P \in$ *Pring-set* R $I \Longrightarrow$
total-eval R g P = *total-eval* R g (*total-eval* (*Pring* R *J0*) (*indexed-const* $\circ$ g) (ψ P)) $\Longrightarrow$
$Q \in$ *Pring-set* R $I \Longrightarrow$
total-eval R g Q = *total-eval* R g (*total-eval* (*Pring* R *J0*) (*indexed-const*

$\circ$ g) (ψ Q)) $\Longrightarrow$
total-eval R g ($P \bigoplus Q$) = *total-eval R g* (*total-eval* (*Pring R J0*) (*indexed-const* $\circ$ *g*) (ψ ($P \bigoplus Q$)))
proof− **fix** *P Q*
assume *A*: *P* ∈ *Pring-set R I*
total-eval R g P = *total-eval R g* (*total-eval* (*Pring R J0*) (*indexed-const* $\circ$ *g*) (ψ *P*))
assume *B*: *Q* ∈ *Pring-set R I*
total-eval R g Q = *total-eval R g* (*total-eval* (*Pring R J0*) (*indexed-const* $\circ$ *g*) (ψ *Q*))
have *0*: ψ ($P \bigoplus Q$) = ψ *P* $\oplus_{Pring\ (Pring\ R\ J0)\ J1}$ ψ *Q*
using *A B assms var-factor-morphism'*(*1*)[*of I J0 J1* ψ]
Pring-add[*of I P Q*] *Pring-car*[*of I*]
ring-hom-ring.homh[*of Pring R I Pring* (*Pring R J0*) *J1* ψ]
ring-hom-add[*of* ψ *Pring R I Pring* (*Pring R J0*) *J1 P Q*]
by *metis*
have *1*: *closed-fun* (*Pring R J0*) (*indexed-const* $\circ$ *g*)
using *assms comp-apply*
by (*smt Pi-I closed-fun-simp indexed-const-closed*)
have *2*: ψ *P* ∈ *carrier* (*Pring* (*Pring R J0*) *J1*)
using *assms A var-factor-morphism'*(*1*)[*of I J0 J1* ψ]
ring-hom-ring.homh ring-hom-closed Pring-car
by *metis*
have *3*: ψ *Q* ∈ *carrier* (*Pring* (*Pring R J0*) *J1*)
using *assms B var-factor-morphism'*(*1*)[*of I J0 J1* ψ]
ring-hom-ring.homh ring-hom-closed Pring-car
by *metis*
have *4*: (*total-eval* (*Pring R J0*) (*indexed-const* $\circ$ *g*) (ψ ($P \bigoplus Q$))) =
(*total-eval* (*Pring R J0*) (*indexed-const* $\circ$ *g*) (ψ *P*)) $\oplus_{Pring\ R\ J0}$
(*total-eval* (*Pring R J0*) (*indexed-const* $\circ$ *g*) (ψ *Q*))
using *0 1 2 3 A B assms cring.total-eval-add*[*of Pring R J0* ψ *P J1* ψ *Q indexed-const* $\circ$ *g*]
by (*metis Pring-car Pring-is-cring is-cring*)
have *5*: (*total-eval* (*Pring R J0*) (*indexed-const* $\circ$ *g*) (ψ *P*)) ∈ *carrier* (*Pring R J0*)
using *3 assms cring.total-eval-closed 1 2 Pring-is-cring is-cring*
by *blast*
have *6*: (*total-eval* (*Pring R J0*) (*indexed-const* $\circ$ *g*) (ψ *Q*)) ∈ *carrier* (*Pring R J0*)
using *4 assms 1 3 Pring-is-cring cring.total-eval-closed is-cring*
by *blast*
show *total-eval R g* ($P \bigoplus Q$) = *total-eval R g* (*total-eval* (*Pring R J0*) (*indexed-const* $\circ$ *g*) (ψ ($P \bigoplus Q$)))
using *5 6 4 assms*
total-eval-add[*of* (*total-eval* (*Pring R J0*) (*indexed-const* $\circ$ *g*) (ψ *p*)) *J0*
(*total-eval* (*Pring R J0*) (*indexed-const* $\circ$ *g*) (ψ *Q*))]
by (*smt A*(*1*) *A*(*2*) *B*(*1*) *B*(*2*) *Pring-add Pring-car in-mono indexed-pset-mono order-refl*
subsetD subset-iff total-eval-add)

qed
show $\bigwedge P\ i.\ P \in \mathit{Pring\text{-}set}\ R\ I \Longrightarrow$
$\mathit{total\text{-}eval}\ R\ g\ P = \mathit{total\text{-}eval}\ R\ g\ (\mathit{total\text{-}eval}\ (\mathit{Pring}\ R\ J0)\ (\mathit{indexed\text{-}const} \circ g)\ (\psi\ P)) \Longrightarrow$
$i \in I \Longrightarrow \mathit{total\text{-}eval}\ R\ g\ (P \bigotimes i) = \mathit{total\text{-}eval}\ R\ g\ (\mathit{total\text{-}eval}\ (\mathit{Pring}\ R\ J0)\ (\mathit{indexed\text{-}const} \circ g)\ (\psi\ (P \bigotimes i)))$
proof− **fix** $P\ i$
assume A: $P \in \mathit{Pring\text{-}set}\ R\ I$
$\mathit{total\text{-}eval}\ R\ g\ P = \mathit{total\text{-}eval}\ R\ g\ (\mathit{total\text{-}eval}\ (\mathit{Pring}\ R\ J0)\ (\mathit{indexed\text{-}const} \circ g)\ (\psi\ P))$
$i \in I$
have *0*: $(P \bigotimes i) = P \otimes_{\mathit{Pring}\ R\ I} (\mathit{pvar}\ R\ i)$
using *A poly-index-mult*
by (*metis Pring-mult var-to-IP-def*)
have *1*: $\psi\ (P \bigotimes i) = \psi\ P \otimes_{\mathit{Pring}\ (\mathit{Pring}\ R\ J0)\ J1} \psi\ (\mathit{pvar}\ R\ i)$
using *0 A assms var-factor-morphism′(1)[of I J0 J1 ψ] pvar-closed[of R i] ring-hom-mult ring-hom-ring.homh*
by (*smt Pring-car Pring-var-closed*)
have *2*: $\psi\ P \in \mathit{carrier}\ (\mathit{Pring}\ (\mathit{Pring}\ R\ J0)\ J1)$
using *assms A var-factor-morphism′(1)[of I J0 J1 ψ] ring-hom-ring.homh ring-hom-closed Pring-car*
by *metis*
have *3*: $\psi\ (\mathit{pvar}\ R\ i) \in \mathit{carrier}\ (\mathit{Pring}\ (\mathit{Pring}\ R\ J0)\ J1)$
using *assms A var-factor-morphism′(1)[of I J0 J1 ψ] ring-hom-ring.homh ring-hom-closed Pring-car pvar-closed[of R i I]*
by (*metis is-cring*)
have *4*: *closed-fun* $(\mathit{Pring}\ R\ J0)\ (\mathit{indexed\text{-}const} \circ g)$
apply(*rule cring.closed-funI*)
using *Pring-is-cring is-cring* **apply** *blast*
using *assms indexed-const-closed closed-fun-simp[of g] comp-apply[of indexed-const g]*
proof −
fix $x :: 'c$
show $(\mathit{indexed\text{-}const} \circ g)\ x \in \mathit{carrier}\ (\mathit{Pring}\ R\ J0)$
by (*metis* (*no-types*) ‹$\bigwedge n.$ *closed-fun R g* $\Longrightarrow$ *g n* $\in$ *carrier R*› ‹$\bigwedge x.$ $(\mathit{indexed\text{-}const} \circ g)\ x = \mathit{indexed\text{-}const}\ (g\ x)$› ‹*closed-fun R g*› *indexed-const-closed*)
qed
have *5*: $(\mathit{total\text{-}eval}\ (\mathit{Pring}\ R\ J0)\ (\mathit{indexed\text{-}const} \circ g)\ (\psi\ (P \bigotimes i))) =$
$(\mathit{total\text{-}eval}\ (\mathit{Pring}\ R\ J0)\ (\mathit{indexed\text{-}const} \circ g)\ (\psi\ P)) \otimes_{\mathit{Pring}\ R\ J0}$
$(\mathit{total\text{-}eval}\ (\mathit{Pring}\ R\ J0)\ (\mathit{indexed\text{-}const} \circ g)\ (\psi\ (\mathit{pvar}\ R\ i)))$
using *2 3 4 cring.total-eval-ring-hom[of Pring R J0 indexed-const ∘ g J1]*
by (*metis 1 Pring-is-cring cring.total-eval-mult is-cring*)
have *6*: $(\mathit{total\text{-}eval}\ (\mathit{Pring}\ R\ J0)\ (\mathit{indexed\text{-}const} \circ g)\ (\psi\ P)) \in \mathit{carrier}\ (\mathit{Pring}\ R\ J0)$
using *total-eval-closed 2 4 Pring-is-cring cring.total-eval-closed is-cring*
by *blast*
have *7*: $(\mathit{total\text{-}eval}\ (\mathit{Pring}\ R\ J0)\ (\mathit{indexed\text{-}const} \circ g)\ (\psi\ (\mathit{pvar}\ R\ i))) \in \mathit{carrier}\ (\mathit{Pring}\ R\ J0)$
using *3 4 Pring-is-cring cring.total-eval-closed is-cring*

by *blast*
have *8*: *total-eval R g* (*total-eval* (*Pring R J0*) (*indexed-const* ∘ *g*) (*ψ* (*P* ⨂ *i*))) =
total-eval R g (*total-eval* (*Pring R J0*) (*indexed-const* ∘ *g*) (*ψ P*)) ⊗$_R$
total-eval R g (*total-eval* (*Pring R J0*) (*indexed-const* ∘ *g*) (*ψ* (*pvar R i*)))
using *6 7*
by (*metis 5 assms*(*5*) *cring.total-eval-mult is-cring*)
have *9*: *total-eval R g* (*total-eval* (*Pring R J0*) (*indexed-const* ∘ *g*) (*ψ* (*P* ⨂ *i*))) =
total-eval R g P ⊗$_R$
total-eval R g (*total-eval* (*Pring R J0*) (*indexed-const* ∘ *g*) (*ψ* (*pvar R i*)))
using *8 A*(*2*)
by *presburger*
have *10*: *total-eval R g* (*total-eval* (*Pring R J0*) (*indexed-const* ∘ *g*) (*ψ* (*pvar R i*))) =
g i
proof(*cases i* ∈ *J0*)
case *True*
then have *ψ* (*pvar R i*) = *ring.indexed-const* (*Pring R J0*) (*pvar R i*)
using *assms*
by (*metis Pring-var-closed inf-sup-absorb var-factor-morphism'*(*3*))
then have (*total-eval* (*Pring R J0*) (*indexed-const* ∘ *g*) (*ψ* (*pvar R i*))) = (*pvar R i*)
by (*metis Pring-is-cring Pring-var-closed True cring.total-eval-const is-cring*)
then show *?thesis*
using *total-eval-var*[*of g i*] *assms var-to-IP-def*
by *metis*
next
case *False*
then have *ψ* (*pvar R i*) = (*pvar* (*Pring R J0*) *i*)
by (*metis A*(*3*) *UnE assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *cring.var-factor-morphism'*(*2*) *is-cring*)
then have *total-eval* (*Pring R J0*) (*indexed-const* ∘ *g*) (*ψ* (*pvar R i*)) =
indexed-const (*g i*)
using *cring.total-eval-var*[*of Pring R J0 indexed-const* ∘ *g i*] *comp-apply*
by (*metis 4 Pring-is-cring is-cring var-to-IP-def*)
then show *?thesis*
using *total-eval-const*[*of g i g*] *assms closed-fun-simp*[*of g i*]
by *metis*
qed
show *total-eval R g* (*P* ⨂ *i*) = *total-eval R g* (*total-eval* (*Pring R J0*) (*indexed-const* ∘ *g*) (*ψ* (*P* ⨂ *i*)))
using *9 10*
by (*metis A*(*1*) *A*(*3*) *Pring-car assms*(*5*) *total-eval-indexed-pmult*)
qed
qed

lemma *var-factor-closed*:
assumes $I = J0 \cup J1$
assumes $J1 \subseteq I$
assumes $J1 \cap J0 = \{\}$
assumes $P \in carrier\ (Pring\ R\ I)$
shows *var-factor I J0 J1 P* $\in$ *carrier* (*Pring* (*Pring R J0*) *J1*)
using *assms var-factor-morphism'*[*of I J0 J1*] *ring-hom-ring.homh*
by (*metis ring-hom-closed*)

lemma *var-factor-add*:
assumes $I = J0 \cup J1$
assumes $J1 \subseteq I$
assumes $J1 \cap J0 = \{\}$
assumes $P \in carrier\ (Pring\ R\ I)$
assumes $Q \in carrier\ (Pring\ R\ I)$
shows *var-factor I J0 J1* $(P \oplus_{Pring\ R\ I} Q) =$ *var-factor I J0 J1 P* $\oplus_{Pring\ (Pring\ R\ J0)\ J1}$ *var-factor I J0 J1 Q*
using *assms var-factor-morphism'*[*of I J0 J1*] *ring-hom-ring.homh*
by (*metis* (*no-types*, *lifting*) *ring-hom-add*)

lemma *var-factor-mult*:
assumes $I = J0 \cup J1$
assumes $J1 \subseteq I$
assumes $J1 \cap J0 = \{\}$
assumes $P \in carrier\ (Pring\ R\ I)$
assumes $Q \in carrier\ (Pring\ R\ I)$
shows *var-factor I J0 J1* $(P \otimes_{Pring\ R\ I} Q) =$ *var-factor I J0 J1 P* $\otimes_{Pring\ (Pring\ R\ J0)\ J1}$ *var-factor I J0 J1 Q*
using *assms var-factor-morphism'*[*of I J0 J1*] *ring-hom-ring.homh*
by (*metis* (*no-types*, *lifting*) *ring-hom-mult*)

2.14.2 Viewing a Mulitvariable Polynomial as a Univariate Polynomial over a Multivariate Polynomial Base Ring

definition *multivar-poly-to-univ-poly* ::
$'c\ set \Rightarrow 'c \Rightarrow ('a,'c)\ mvar\text{-}poly \Rightarrow$
$(('a,'c)\ mvar\text{-}poly)\ u\text{-}poly$ **where**
multivar-poly-to-univ-poly I i P = ((*IP-to-UP i*) $\circ$ (*var-factor I* $(I - \{i\})\ \{i\}$)) *P*

definition *univ-poly-to-multivar-poly* ::
$'c\ set \Rightarrow 'c \Rightarrow (('a,'c)\ mvar\text{-}poly)\ u\text{-}poly \Rightarrow$
$('a,'c)\ mvar\text{-}poly$ **where**
univ-poly-to-multivar-poly I i P =((*var-factor-inv I* $(I - \{i\})\ \{i\}$) $\circ$ (*UP-to-IP* (*Pring R* $(I - \{i\})$) *i*)) *P*

lemma *multivar-poly-to-univ-poly-is-hom*:
assumes $i \in I$
shows *multivar-poly-to-univ-poly I i* $\in$ *ring-hom* (*Pring R I*) (*UP* (*Pring R* ($I - \{i\}$)))

unfolding *multivar-poly-to-univ-poly-def comp-apply*
apply(*rule ring-hom-compose*[*of - Pring* (*Pring R* (*I* − {*i*})) {*i*} *- var-factor I* (*I* − {*i*}) {*i*} *IP-to-UP i*])
apply (*simp add*: *Pring-is-ring*; *fail*)
apply (*simp add*: *local.ring-axioms ring.Pring-is-ring*; *fail*)
apply(*rule UP-ring.UP-ring*) **unfolding** *UP-ring-def*
apply (*simp add*: *Pring-is-ring*; *fail*)
using *assms var-factor-morphism'*[*of I I* − {*i*} {*i*}] **unfolding** *ring-hom-ring-def ring-hom-ring-axioms-def*
apply *blast*
using *UP-cring.IP-to-UP-ring-hom*[*of Pring R* (*I* − {*i*}) *i*]
unfolding *ring-hom-ring-def ring-hom-ring-axioms-def UP-cring-def*
using *Pring-is-cring is-cring* **apply** *blast*
by *blast*

lemma *multivar-poly-to-univ-poly-inverse*:
assumes *i* ∈ *I*
assumes *ψ0* = *multivar-poly-to-univ-poly I i*
assumes *ψ1* = *univ-poly-to-multivar-poly I i*
assumes *P* ∈ *carrier* (*Pring R I*)
shows *ψ1* (*ψ0 P*) = *P*
proof−
have *closed*: *var-factor I* (*I* − {*i*}) {*i*} *P* ∈ *carrier* (*Pring* (*Pring R* (*I* − {*i*})) {*i*})
using *assms var-factor-closed*[*of I I* − {*i*} {*i*} *P*]
by *blast*
have *cancel-1*: *UP-to-IP* (*Pring R* (*I* − {*i*})) *i*
(*IP-to-UP i* (*var-factor I* (*I* − {*i*}) {*i*} *P*)) =
var-factor I (*I* − {*i*}) {*i*} *P*
using *closed assms ring.Pring-car*
UP-cring.UP-to-IP-inv[*of Pring R* (*I* − {*i*}) *var-factor I* (*I* − {*i*}) {*i*} *P i*]
Pring-is-ring **unfolding** *UP-cring-def*
using *is-cring local.ring-axioms ring.Pring-is-cring*
by *blast*
have *univ-poly-to-multivar-poly I i* (*multivar-poly-to-univ-poly I i P*) =
univ-poly-to-multivar-poly I i ((*IP-to-UP i*) (*var-factor I* (*I* − {*i*}) {*i*} *P*))
by (*metis comp-apply multivar-poly-to-univ-poly-def*)
then have *univ-poly-to-multivar-poly I i* (*multivar-poly-to-univ-poly I i P*) =
((*var-factor-inv I* (*I* − {*i*}) {*i*}) ((*UP-to-IP* (*Pring R* (*I* − {*i*})) *i*)
((*IP-to-UP i*) (*var-factor I* (*I* − {*i*}) {*i*} *P*))))
using *comp-apply univ-poly-to-multivar-poly-def*
by *metis*
then have *univ-poly-to-multivar-poly I i* (*multivar-poly-to-univ-poly I i P*) =
((*var-factor-inv I* (*I* − {*i*}) {*i*}) (*var-factor I* (*I* − {*i*}) {*i*} *P*))
using *cancel-1*
by *presburger*
then show *?thesis* **using** *assms var-factor-inv-inverse*[*of I I* − {*i*} {*i*} *- - P*]
by (*metis Diff-cancel Diff-disjoint Diff-eq-empty-iff Diff-partition Un-Diff-cancel*

Un-Diff-cancel2 Un-insert-right empty-Diff empty-subsetI insert-Diff-if insert-absorb)
qed

lemma *multivar-poly-to-univ-poly-total-eval*:
assumes $i \in I$
assumes ψ = *multivar-poly-to-univ-poly I i*
assumes $P \in$ *carrier* (*Pring R I*)
assumes *closed-fun R g*
shows *total-eval R g P* = *total-eval R g* (*to-function* (*Pring R* ($I - \{i\}$)) (ψ *P*) (*indexed-const* (*g i*)))
proof−
have *0*: *var-factor I* ($I - \{i\}$) $\{i\}$ *P* ∈ *Pring-set* (*Pring R* ($I - \{i\}$)) $\{i\}$
proof−
have *00*: *ring* (*Pring R* ($I - \{i\}$))
using *Pring-is-ring*
by *blast*
then show *?thesis*
using *assms*(*1*) *assms*(*3*) *var-factor-closed*[*of I I* − $\{i\}$ $\{i\}$ *P*] *ring.Pring-car*[*of Pring R* ($I - \{i\}$) $\{i\}$]
by *blast*
qed
have *1*: *closed-fun* (*Pring R* ($I - \{i\}$)) (*indexed-const* ∘ *g*)
using *assms comp-apply indexed-const-closed cring.closed-funI*[*of Pring R* ($I - \{i\}$)]
by (*metis Pring-is-cring closed-fun-simp is-cring*)
have *2*: *cring* (*Pring R* ($I - \{i\}$))
using *Pring-is-cring is-cring* **by** *blast*
have (*to-function* (*Pring R* ($I - \{i\}$)) (ψ *P*) (*indexed-const* (*g i*)))
= *total-eval* (*Pring R* ($I - \{i\}$)) (*indexed-const* ∘ *g*) ((*var-factor I* ($I - \{i\}$) $\{i\}$) *P*)
using *assms 0 1 2 comp-apply UP-cring.IP-to-UP-poly-eval*[*of Pring R* ($I - \{i\}$)
(*var-factor I* ($I - \{i\}$) $\{i\}$) *P i indexed-const* ∘ *g*]
unfolding *UP-cring-def multivar-poly-to-univ-poly-def*
by *metis*
then have *3*: *total-eval R g* ((*to-function* (*Pring R* ($I - \{i\}$)) (ψ *P*) (*indexed-const* (*g i*))))
= *total-eval R g* (*total-eval* (*Pring R* ($I - \{i\}$)) (*indexed-const* ∘ *g*) ((*var-factor I* ($I - \{i\}$) $\{i\}$) *P*))
by *presburger*
then have *total-eval R g P* = *total-eval R g* (*total-eval* (*Pring R* ($I - \{i\}$)) (*indexed-const* ∘ *g*) ((*var-factor I* ($I - \{i\}$) $\{i\}$) *P*))
using *assms var-factor-total-eval*[*of I I* − $\{i\}$ $\{i\}$ *var-factor I* ($I - \{i\}$) $\{i\}$ *g P*]
by *blast*
then show *?thesis*
using *3*

by *presburger*
qed

Induction for polynomial rings. Basically just `indexed_pset.induct` with some boilerplate translations removed

lemma(**in** *ring*) *Pring-car-induct''*:
assumes $Q \in$ *carrier* (*Pring R I*)
assumes $\bigwedge c.\ c \in$ *carrier* $R \Longrightarrow P$ (*indexed-const c*)
assumes $\bigwedge p\ q.\ p \in$ *carrier* (*Pring R I*) $\Longrightarrow q \in$ *carrier* (*Pring R I*) $\Longrightarrow P\ p \Longrightarrow P\ q \Longrightarrow P\ (p \oplus_{Pring\ R\ I} q)$
assumes $\bigwedge p\ i.\ p \in$ *carrier* (*Pring R I*) $\Longrightarrow i \in I \Longrightarrow P\ p \Longrightarrow P\ (p \otimes_{Pring\ R\ I}$ *pvar R i*)
shows *P Q*
apply(*rule indexed-pset.induct*[*of Q I carrier R*])
using *Pring-car assms*(*1*) **apply** *blast*
using *assms*(*2*) **apply** *blast*
apply (*metis* (*full-types*) *Pring-add Pring-car assms*(*3*))
proof−
fix *Pa i* **assume** *A*: *Pa* ∈ *Pring-set R I P Pa i* ∈ *I*
then have *0*: *Pa* ∈ *carrier* (*Pring R I*)
using *assms A Pring-car*
by *blast*
have *Pa* $\bigotimes$ *i* = *Pa* $\otimes_{Pring\ R\ I}$ *pvar R i*
using *0 poly-index-mult assms A*
by (*metis Pring-mult var-to-IP-def*)
then show *P* (*Pa* $\bigotimes$ *i*)
by (*simp add*: *0 A*(*2*) *A*(*3*) *assms*)
qed

2.14.3 Application: A Polynomial Ring over a Domain is a Domain

In this section, we use the fact the UP R is a domain when R is a domain to show the analogous result for indexed polynomial rings. We first prove it inductively for rings with a finite variable set, and then generalize to infinite variable sets using the fact that any two multivariable polynomials over an indexed polynomial ring must also lie in a finitely indexed polynomial subring.

lemma *indexed-const-mult*:
assumes *a* ∈ *carrier R*
assumes *b* ∈ *carrier R*
shows *indexed-const a* ⊗ $_{Pring\ R\ I}$ *indexed-const b* = *indexed-const* (*a* ⊗ *b*)
by (*metis Pring-mult assms*(*1*) *assms*(*2*) *indexed-const-P-mult-eq*)

lemma(**in** *domain*) *Pring-fin-vars-is-domain*:
assumes *finite* (*I* ::*'c set*)
shows *domain* (*Pring R I*)

proof(*rule finite-induct*, *rule assms*)
show *domain* (*Pring R* ({}::'*c set*))
proof(*rule domainI*)
show *cring* (*Pring R* {})
by (*simp add*: *Pring-is-cring is-cring*)
show $\mathbf{1}_{Pring\ R\ \{\}} \neq \mathbf{0}_{Pring\ R\ \{\}}$
proof−
have $\mathbf{1}_{Pring\ R\ \{\}}\ \{\#\} \neq \mathbf{0}_{Pring\ R\ \{\}}\ \{\#\}$
by (*metis Pring-one Pring-zero local.ring-axioms ring.indexed-const-def zero-not-one*)
thus *?thesis*
by *metis*
qed
show $\bigwedge a\ b.\ a \otimes_{Pring\ R\ (\{\}::'c\ set)}\ b = \mathbf{0}_{Pring\ R\ \{\}} \Longrightarrow$
$a \in carrier\ (Pring\ R\ \{\}) \Longrightarrow b \in carrier\ (Pring\ R\ \{\}) \Longrightarrow a = \mathbf{0}_{Pring\ R\ \{\}} \vee b = \mathbf{0}_{Pring\ R\ \{\}}$
proof−
fix *a b* **assume** *A*: $a \otimes_{Pring\ R\ (\{\}::'c\ set)}\ b = \mathbf{0}_{Pring\ R\ \{\}}$
$a \in carrier\ (Pring\ R\ \{\})$ $b \in carrier\ (Pring\ R\ \{\})$
have *0*: *monomials-of R a* ⊆ {{#}}
using *A Pring-set-zero* **unfolding** *monomials-of-def Pring-car*
by *blast*
have *1*: *monomials-of R b* ⊆ {{#}}
using *A Pring-set-zero* **unfolding** *monomials-of-def Pring-car*
by *blast*
obtain *A* **where** *A-def*: *A* = *a* {#}
by *blast*
obtain *B* **where** *B-def*: *B* = *b* {#}
by *blast*
have *2*: *a* = *indexed-const A*
unfolding *A-def*
apply(*rule ext*)
by (*metis 0 complement-of-monomials-of empty-iff in-mono indexed-const-def insert-iff*)
have *3*: *b* = *indexed-const B*
unfolding *B-def*
apply(*rule ext*)
by (*metis 1 complement-of-monomials-of empty-iff in-mono indexed-const-def insert-iff*)
have *4*: $a \otimes_{Pring\ R\ \{\}}\ b$ = *indexed-const* (*A* ⊗ *B*)
using *A* **unfolding** *2 3*
by (*metis 2 3 A-def B-def Pring-carrier-coeff' indexed-const-mult*)
have *5*: *A* ⊗ *B* = **0**
using *A* **unfolding** *4* **by** (*metis Pring-zero indexed-const-def*)
have *6*: *A* = **0** ∨ *B* = **0**
using *5 A-def B-def A*
by (*simp add*: *domain.integral-iff domain-axioms*)
show $a = \mathbf{0}_{Pring\ R\ \{\}} \vee b = \mathbf{0}_{Pring\ R\ \{\}}$

```
      unfolding 2 3 using 6
      by (metis Pring-zero)
   qed
  qed
  show ⋀x F. finite (F:: 'c set) ⟹ x ∉ F ⟹ domain (Pring R F) ⟹ domain
(Pring R (insert x F))
  proof− fix S:: 'c set  fix s assume A: finite S s ∉ S domain (Pring R S)
   show domain (Pring R (insert s S))
   proof−
     have ring-hom: multivar-poly-to-univ-poly (insert s S) s ∈ ring-hom (Pring
R (insert s S)) (UP (Pring R S))
      using multivar-poly-to-univ-poly-is-hom
      by (metis A(2) Diff-insert-absorb insertI1)
    have domain: domain (UP (Pring R S))
     apply(rule UP-domain.UP-domain)
     unfolding UP-domain-def by(rule A)
    show domain (Pring R (insert s S))
    apply(rule ring-hom-ring.inj-on-domain[of - UP (Pring R S) multivar-poly-to-univ-poly
(insert s S) s])
       apply(rule ring-hom-ring.intro)
        apply (simp add: Pring-is-ring; fail)
       apply(rule UP-ring.UP-ring)
      unfolding UP-ring-def
      apply (simp add: Pring-is-ring; fail)
      unfolding ring-hom-ring-axioms-def apply(rule ring-hom)
    proof(rule inj-onI)
      fix x y assume A: x ∈ carrier (Pring R (insert s S))
        y ∈ carrier (Pring R (insert s S))
         multivar-poly-to-univ-poly (insert s S) s x = multivar-poly-to-univ-poly
(insert s S) s y
    then have 0: univ-poly-to-multivar-poly (insert s S) s (multivar-poly-to-univ-poly
(insert s S) s x)
     = univ-poly-to-multivar-poly (insert s S) s (multivar-poly-to-univ-poly (insert
s S) s y)
       by auto
      have 1: univ-poly-to-multivar-poly (insert s S) s (multivar-poly-to-univ-poly
(insert s S) s x) = x
          using A by (meson cring.multivar-poly-to-univ-poly-inverse insertI1
is-cring)
      have 2: univ-poly-to-multivar-poly (insert s S) s (multivar-poly-to-univ-poly
(insert s S) s y) = y
       using A  by (meson insertI1 multivar-poly-to-univ-poly-inverse)
      show x = y
       using 0 unfolding 1 2 by auto
    next
      show domain (UP (Pring R S))
       apply(rule UP-domain.UP-domain)
       unfolding UP-domain-def by(rule A)
    qed
```

qed
qed
qed

lemma *locally-finite*:
assumes $a \in carrier\ (Pring\ R\ I)$
shows $\exists J.\ J \subseteq I \wedge finite\ J \wedge a \in carrier\ (Pring\ R\ J)$
proof(*rule Pring-car-induct*[*of* - *I*], *rule assms*)
have *0*: $\mathbf{0}_{Pring\ R\ I} = \mathbf{0}_{Pring\ R\ \{\}}$
by (*simp add*: *Pring-zero-eq*)
show $\exists J{\subseteq}I.\ finite\ J \wedge \mathbf{0}_{Pring\ R\ I} \in carrier\ (Pring\ R\ J)$
unfolding *0*
by (*meson Pring-zero-closed empty-subsetI finite.emptyI*)
next
fix *m k* **assume** *A*: *set-mset m* $\subseteq I \wedge k \in carrier\ R$
obtain *J* **where** *J-def*: $J = set\text{-}mset\ m$
by *blast*
have *0*: $k \odot_{Pring\ R\ I} mset\text{-}to\text{-}IP\ R\ m = k \odot_{Pring\ R\ J} mset\text{-}to\text{-}IP\ R\ m$
unfolding *J-def* **by** (*simp add*: *Pring-smult*)
have *1*: $k \odot_{Pring\ R\ I} mset\text{-}to\text{-}IP\ R\ m \in carrier\ (Pring\ R\ J)$
unfolding *0 J-def* **using** *A*
by (*simp add*: *Pring-car Pring-smult mset-to-IP-closed poly-scalar-mult-closed*)
show $\exists J{\subseteq}I.\ finite\ J \wedge k \odot_{Pring\ R\ I} mset\text{-}to\text{-}IP\ R\ m \in carrier\ (Pring\ R\ J)$
using *J-def 1 A* **by** *blast*
next
fix *Q m k*
assume *A*: $Q \in carrier\ (Pring\ R\ I) \wedge (\exists J{\subseteq}I.\ finite\ J \wedge Q \in carrier\ (Pring\ R\ J)) \wedge set\text{-}mset\ m \subseteq I \wedge k \in carrier\ R$
then obtain *J* **where** *J-def*: $J{\subseteq}I \wedge finite\ J \wedge Q \in carrier\ (Pring\ R\ J)$
by *blast*
obtain J' **where** J'*-def*: $J' = J \cup set\text{-}mset\ m$
by *blast*
have *0*: *finite* J'
unfolding J'*-def* **using** *J-def* **by** *blast*
have *1*: $k \odot_{Pring\ R\ I} mset\text{-}to\text{-}IP\ R\ m = k \odot_{Pring\ R\ J'} mset\text{-}to\text{-}IP\ R\ m$
unfolding J'*-def* **using** *A* **by** (*simp add*: *Pring-smult*)
have *2*: $Q \in carrier\ (Pring\ R\ J')$
using *J-def* **unfolding** J'*-def*
by (*metis Pring-car Pring-carrier-subset Un-upper1 subsetD*)
have *3*: $Q \bigoplus k \odot_{Pring\ R\ I} mset\text{-}to\text{-}IP\ R\ m \in carrier\ (Pring\ R\ J')$
using *0 1 2* J'*-def A*
by (*metis Pring-car Pring-smult-closed indexed-pset.indexed-padd mset-to-IP-closed sup.cobounded2*)
show $\exists J'{\subseteq}I.\ finite\ J' \wedge Q \bigoplus k \odot_{Pring\ R\ I} mset\text{-}to\text{-}IP\ R\ m \in carrier\ (Pring\ R\ J')$
using *3 0*
by (*metis A* J'*-def J-def Un-subset-iff*)
qed

lemma(**in** *domain*) *Pring-is-domain*:
  domain (*Pring R I*)
proof(*rule domainI*, *simp add*: *Pring-is-cring is-cring*)
  have *0*: $\mathbf{1}_{Pring\ R\ I}\ \{\#\} = \mathbf{1}$
    by (*simp add*: *Pring-one indexed-const-def*)
  have *1*: $\mathbf{0}_{Pring\ R\ I}\ \{\#\} = \mathbf{0}$
    by (*simp add*: *Pring-zero indexed-const-def*)
  have *2*: $\mathbf{1} \neq \mathbf{0}$
    by *simp*
  show $\mathbf{1}_{Pring\ R\ I} \neq \mathbf{0}_{Pring\ R\ I}$
    using *0 1 2* **by** *auto*
next
  fix *a b* **assume** *A*: $a \otimes_{Pring\ R\ I} b = \mathbf{0}_{Pring\ R\ I}$
                $a \in carrier\ (Pring\ R\ I)$
                $b \in carrier\ (Pring\ R\ I)$
  obtain *J0* **where** *J0-def*: $J0 \subseteq I \wedge finite\ J0 \wedge a \in carrier\ (Pring\ R\ J0)$
    using *A locally-finite* **by** *blast*
  obtain *J1* **where** *J1-def*: $J1 \subseteq I \wedge finite\ J1 \wedge b \in carrier\ (Pring\ R\ J1)$
    using *A locally-finite* **by** *blast*
  obtain *J* **where** *J-def*: $J = J0 \cup J1$
    by *blast*
  have *J-finite*: *finite J*
    using *J-def J0-def J1-def* **by** *blast*
  have *0*: $a \in carrier\ (Pring\ R\ J)$
    using *J0-def* **unfolding** *J-def*
   by (*metis* (*no-types*, *lifting*) *Pring-car Pring-carrier-subset Un-upper1 subset-eq*)
  have *1*: $b \in carrier\ (Pring\ R\ J)$
    using *J1-def* **unfolding** *J-def*
    by (*metis Pring-car Pring-carrier-subset in-mono sup.cobounded2*)
  have *2*: $a \otimes_{Pring\ R\ J} b = \mathbf{0}_{Pring\ R\ J}$
    using *A J-def 0 1* **by** (*metis Pring-mult-eq Pring-zero-eq*)
  have *3*: *domain* (*Pring R J*)
    using *J-finite Pring-fin-vars-is-domain*[*of J*] **by** *blast*
  have *4*: $a = \mathbf{0}_{Pring\ R\ J} \vee b = \mathbf{0}_{Pring\ R\ J}$
    using *3 2 0 1* **by** (*simp add*: *domain.integral*)
  thus $a = \mathbf{0}_{Pring\ R\ I} \vee b = \mathbf{0}_{Pring\ R\ I}$
    using *Pring-zero-eq* **by** *blast*
qed

2.14.4 Relabelling of Variables for Indexed Polynomial Rings

definition *relabel-vars* :: *'d set* ⇒ *'e set* ⇒ (*'d* ⇒ *'e*) ⇒
      (*'a*, *'d*) *mvar-poly* ⇒ (*'a*, *'e*) *mvar-poly* **where**
relabel-vars I J g = *indexed-poly-induced-morphism I* (*Pring R J*) *indexed-const* (*λi*. *pvar R* (*g i*))

lemma *relabel-vars-is-morphism*:
  assumes $g \in I \to J$
  shows *ring-hom-ring* (*Pring R I*) (*Pring R J*) (*relabel-vars I J g*)

$\bigwedge i.\ i \in I \Longrightarrow$ *relabel-vars I J g* (*pvar R i*) = *pvar R* (*g i*)
$\bigwedge c.\ c \in$ *carrier R* $\Longrightarrow$ *relabel-vars I J g* (*indexed-const c*) = *indexed-const c*
using *Pring-universal-prop*(*1*)[*of Pring R J* $\lambda i.$ *pvar R* (*g i*) *I indexed-const*]
assms **unfolding** *relabel-vars-def*
apply (*meson Pi-iff Pring-is-cring Pring-var-closed indexed-const-ring-hom is-cring*)
proof−
have *0*: *cring* (*Pring R J*) (*λi. mset-to-IP R* {#*g i*#}) ∈ *I* → *carrier* (*Pring R J*)
ring-hom-ring R (*Pring R J*) *indexed-const*
using *assms Pring-is-cring is-cring* **apply** *blast*
apply (*smt Pi-iff Pring-var-closed assms var-to-IP-def*)
by (*simp add*: *indexed-const-ring-hom*)
show $\bigwedge i.\ i \in I \Longrightarrow$ *indexed-poly-induced-morphism I* (*Pring R J*)
indexed-const (*λi. pvar R* (*g i*)) (*pvar R i*) = *pvar R* (*g i*)
using *0 assms Pring-universal-prop*(*2*)[*of Pring R J* $\lambda i.$ *pvar R* (*g i*) *I indexed-const*]
unfolding *relabel-vars-def var-to-IP-def*
by *blast*
show $\bigwedge c.\ c \in$ *carrier R* $\Longrightarrow$ *indexed-poly-induced-morphism I* (*Pring R J*)
indexed-const (*λi. pvar R* (*g i*)) (*indexed-const c*) = *indexed-const c*
using *0 assms Pring-universal-prop*(*3*)[*of Pring R J* $\lambda i.$ *pvar R* (*g i*) *I indexed-const*]
unfolding *var-to-IP-def*
by *blast*
qed

lemma *relabel-vars-add*:
assumes *g* ∈ *I* → *J*
assumes *P* ∈ *carrier* (*Pring R I*)
assumes *Q* ∈ *carrier* (*Pring R I*)
shows *relabel-vars I J g* (*P* $\oplus_{Pring\ R\ I}$ *Q*) = *relabel-vars I J g P* $\oplus_{Pring\ R\ J}$ *relabel-vars I J g Q*
using *assms relabel-vars-is-morphism*(*1*)[*of g I J*] *ring-hom-ring.homh ring-hom-add*
by (*metis* (*no-types*, *lifting*))

lemma *relabel-vars-mult*:
assumes *g* ∈ *I* → *J*
assumes *P* ∈ *carrier* (*Pring R I*)
assumes *Q* ∈ *carrier* (*Pring R I*)
shows *relabel-vars I J g* (*P* $\otimes_{Pring\ R\ I}$ *Q*) = *relabel-vars I J g P* $\otimes_{Pring\ R\ J}$ *relabel-vars I J g Q*
using *assms relabel-vars-is-morphism*(*1*)[*of g I J*] *ring-hom-ring.homh ring-hom-mult*
by (*metis* (*no-types*, *lifting*))

lemma *relabel-vars-closed*:
assumes *g* ∈ *I* → *J*
assumes *P* ∈ *carrier* (*Pring R I*)

shows *relabel-vars I J g P* ∈ *carrier* (*Pring R J*)
using *assms relabel-vars-is-morphism*(*1*)[*of g I J*] *ring-hom-ring.homh*
by (*metis ring-hom-closed*)

lemma *relabel-vars-smult*:
assumes *g* ∈ *I* → *J*
assumes *P* ∈ *carrier* (*Pring R I*)
assumes *a* ∈ *carrier R*
shows *relabel-vars I J g* (*a* ⊙$_{Pring\ R\ I}$ *P*) = *a* ⊙$_{Pring\ R\ J}$*relabel-vars I J g P*
proof−
have *0*: *a* ⊙$_{Pring\ R\ I}$*P* = *indexed-const a* ⊗$_{Pring\ R\ I}$ *P*
by (*metis Pring-car Pring-mult Pring-smult assms*(*2*) *assms*(*3*) *poly-scalar-mult-eq*)
have *1*: *a* ⊙$_{Pring\ R\ J}$*relabel-vars I J g P* = *indexed-const a* ⊗$_{Pring\ R\ J}$ *relabel-vars I J g P*
by (*metis Pring-car Pring-mult Pring-smult assms*(*1*) *assms*(*2*) *assms*(*3*) *poly-scalar-mult-eq relabel-vars-closed*)
show *?thesis* **using** *0 1 assms relabel-vars-mult relabel-vars-is-morphism*(*3*)[*of g I J a*]
by (*metis indexed-const-closed*)
qed

lemma *relabel-vars-inverse*:
assumes *g* ∈ *I* → *J*
assumes *h* ∈ *J* → *I*
assumes ⋀*i*. *i* ∈ *I* ⟹ *h* (*g i*) = *i*
assumes *P* ∈ *carrier* (*Pring R I*)
shows *relabel-vars J I h* (*relabel-vars I J g P*) = *P*
apply(*rule Pring-car-induct''*[*of - I*])
using *assms*(*4*) **apply** *blast*
using *assms*
apply (*metis relabel-vars-is-morphism*(*3*))
proof−
show ⋀*p q*. *p* ∈ *carrier* (*Pring R I*) ⟹
q ∈ *carrier* (*Pring R I*) ⟹
relabel-vars J I h (*relabel-vars I J g p*) = *p* ⟹
relabel-vars J I h (*relabel-vars I J g q*) = *q* ⟹
relabel-vars J I h (*relabel-vars I J g* (*p* ⊕$_{Pring\ R\ I}$ *q*)) = *p* ⊕$_{Pring\ R\ I}$ *q*
proof− **fix** *p q* **assume** *A*: *p* ∈ *carrier* (*Pring R I*) *q* ∈ *carrier* (*Pring R I*)
relabel-vars J I h (*relabel-vars I J g p*) = *p*
relabel-vars J I h (*relabel-vars I J g q*) = *q*
show *relabel-vars J I h* (*relabel-vars I J g* (*p* ⊕$_{Pring\ R\ I}$ *q*)) = *p* ⊕$_{Pring\ R\ I}$ *q*
proof−
have *0*: *relabel-vars I J g* (*p* ⊕$_{Pring\ R\ I}$ *q*) = *relabel-vars I J g p* ⊕$_{Pring\ R\ J}$ *relabel-vars I J g q*
using *A*(*1*) *A*(*2*) *assms*(*1*) *relabel-vars-add* **by** *blast*
then show *?thesis*
using *assms A relabel-vars-is-morphism*(*3*)[*of g I J*] *relabel-vars-is-morphism*(*3*)[*of h J I*]
relabel-vars-closed[*of g I J p*] *relabel-vars-closed*[*of g I J q*]

relabel-vars-add[*of h J I relabel-vars I J g p relabel-vars I J g q*]
by *presburger*
qed
qed
show $\bigwedge p\ i.\ p \in carrier\ (Pring\ R\ I) \Longrightarrow$
$i \in I \Longrightarrow$
relabel-vars J I h (*relabel-vars I J g p*) = *p* $\Longrightarrow$
relabel-vars J I h (*relabel-vars I J g* ($p \otimes_{Pring\ R\ I}$ *pvar R i*)) = $p \otimes_{Pring\ R\ I}$ *pvar R i*
proof− **fix** *p i* **assume** *A*: $p \in carrier\ (Pring\ R\ I)$
relabel-vars J I h (*relabel-vars I J g p*) = *p*
$i \in I$
show *relabel-vars J I h* (*relabel-vars I J g* ($p \otimes_{Pring\ R\ I}$ *pvar R i*)) = $p \otimes_{Pring\ R\ I}$ *pvar R i*
proof−
have *relabel-vars J I h* (*relabel-vars I J g* (*pvar R i*)) = *pvar R i*
using *assms relabel-vars-is-morphism*[*of g I J*] *relabel-vars-is-morphism*[*of h J I*]
by (*metis A*(*3*) *funcset-mem*)
then show *?thesis*
using *assms relabel-vars-is-morphism*[*of g I J*] *relabel-vars-is-morphism*[*of h J I*]
relabel-vars-closed[*of g I J p*] *relabel-vars-closed*[*of g I J pvar R i*]
relabel-vars-mult[*of g I J p pvar R i*]
relabel-vars-mult[*of h J I relabel-vars I J g p relabel-vars I J g* (*pvar R i*)]
by (*metis A*(*1*) *A*(*2*) *A*(*3*) *Pring-var-closed*)
qed
qed
qed

lemma *relabel-vars-total-eval*:
assumes $g \in I \to J$
assumes $P \in carrier\ (Pring\ R\ I)$
assumes *closed-fun R f*
shows *total-eval R* (*f* ∘ *g*) *P* = *total-eval R f* (*relabel-vars I J g P*)
proof(*rule Pring-car-induct''*[*of P I*])
show $P \in carrier\ (Pring\ R\ I)$
using *assms*(*2*) **by** *blast*
show $\bigwedge c.\ c \in carrier\ R \Longrightarrow$ *total-eval R* (*f* ∘ *g*) (*indexed-const c*) = *total-eval R f* (*relabel-vars I J g* (*indexed-const c*))
by (*metis assms*(*1*) *relabel-vars-is-morphism*(*3*) *total-eval-const*)
show $\bigwedge p\ q.\ p \in carrier\ (Pring\ R\ I) \Longrightarrow$
$q \in carrier\ (Pring\ R\ I) \Longrightarrow$
total-eval R (*f* ∘ *g*) *p* = *total-eval R f* (*relabel-vars I J g p*) $\Longrightarrow$
total-eval R (*f* ∘ *g*) *q* = *total-eval R f* (*relabel-vars I J g q*) $\Longrightarrow$
total-eval R (*f* ∘ *g*) ($p \oplus_{Pring\ R\ I} q$) = *total-eval R f* (*relabel-vars I J g* ($p \oplus_{Pring\ R\ I} q$))

proof− **fix** p q **assume** A: $p \in carrier\ (Pring\ R\ I)$ $q \in carrier\ (Pring\ R\ I)$
$total\text{-}eval\ R\ (f \circ g)\ p = total\text{-}eval\ R\ f\ (relabel\text{-}vars\ I\ J\ g\ p)$
$total\text{-}eval\ R\ (f \circ g)\ q = total\text{-}eval\ R\ f\ (relabel\text{-}vars\ I\ J\ g\ q)$
have *0*: $closed\text{-}fun\ R\ (f \circ g)$
apply(*rule closed-funI*)
using *comp-apply*[*of f g*] *closed-fun-simp*[*of f*] *assms*(*3*) **by** *presburger*
have *1*: $(relabel\text{-}vars\ I\ J\ g\ (p \oplus_{Pring\ R\ I} q)) =$
$(relabel\text{-}vars\ I\ J\ g\ p) \oplus_{Pring\ R\ J} (relabel\text{-}vars\ I\ J\ g\ q)$
using *A*(*1*) *A*(*2*) *assms*(*1*) *relabel-vars-add* **by** *blast*
have *2*: $total\text{-}eval\ R\ f\ (relabel\text{-}vars\ I\ J\ g\ (p \oplus_{Pring\ R\ I} q)) =$
$(total\text{-}eval\ R\ f\ (relabel\text{-}vars\ I\ J\ g\ p)) \oplus_R (total\text{-}eval\ R\ f\ (relabel\text{-}vars\ I\ J\ g\ q))$
using *total-eval-add*[*of - J - f*]
by (*metis 1 A*(*1*) *A*(*2*) *A*(*3*) *A*(*4*) *assms*(*1*) *assms*(*3*) *relabel-vars-closed*)
show $total\text{-}eval\ R\ (f \circ g)\ (p \oplus_{Pring\ R\ I} q) = total\text{-}eval\ R\ f\ (relabel\text{-}vars\ I\ J\ g\ (p \oplus_{Pring\ R\ I} q))$
using *A 0 1 2*
by (*metis total-eval-add*)
qed
show $\bigwedge p\ i.\ p \in carrier\ (Pring\ R\ I) \Longrightarrow i \in I \Longrightarrow total\text{-}eval\ R\ (f \circ g)\ p = total\text{-}eval\ R\ f\ (relabel\text{-}vars\ I\ J\ g\ p) \Longrightarrow total\text{-}eval\ R\ (f \circ g)\ (p \otimes_{Pring\ R\ I} pvar\ R\ i) = total\text{-}eval\ R\ f\ (relabel\text{-}vars\ I\ J\ g\ (p \otimes_{Pring\ R\ I} pvar\ R\ i))$
proof− **fix** p i **assume** A: $p \in carrier\ (Pring\ R\ I)$ $i \in I$
$total\text{-}eval\ R\ (f \circ g)\ p = total\text{-}eval\ R\ f\ (relabel\text{-}vars\ I\ J\ g\ p)$
have *0*: $closed\text{-}fun\ R\ (f \circ g)$
apply(*rule closed-funI*)
using *comp-apply*[*of f g*] *closed-fun-simp*[*of f*] *assms*(*3*) **by** *presburger*
have *1*: $(relabel\text{-}vars\ I\ J\ g\ (p \otimes_{Pring\ R\ I} (pvar\ R\ i))) =$
$(relabel\text{-}vars\ I\ J\ g\ p) \otimes_{Pring\ R\ J} (relabel\text{-}vars\ I\ J\ g\ (pvar\ R\ i))$
by (*meson A*(*1*) *A*(*2*) *assms*(*1*) *local.ring-axioms relabel-vars-mult ring.Pring-var-closed*)
have *2*: $total\text{-}eval\ R\ f\ (relabel\text{-}vars\ I\ J\ g\ (p \otimes_{Pring\ R\ I} (pvar\ R\ i))) =$
$(total\text{-}eval\ R\ f\ (relabel\text{-}vars\ I\ J\ g\ p)) \otimes_R (total\text{-}eval\ R\ f\ (relabel\text{-}vars\ I\ J\ g\ (pvar\ R\ i)))$
using *total-eval-mult*[*of relabel-vars I J g p J relabel-vars I J g* (*pvar R i*)]
by (*metis 1 A*(*1*) *A*(*2*) *A*(*3*) *assms*(*1*) *assms*(*3*) *is-cring pvar-closed relabel-vars-closed*)
have *3*: $total\text{-}eval\ R\ (f \circ g)\ (p \otimes_{Pring\ R\ I} pvar\ R\ i) =$
$total\text{-}eval\ R\ (f \circ g)\ p \otimes_R total\text{-}eval\ R\ (f \circ g)(\ pvar\ R\ i)$
by (*meson 0 A*(*1*) *A*(*2*) *Pring-var-closed total-eval-mult*)
have *4*: $total\text{-}eval\ R\ (f \circ g)(\ pvar\ R\ i) = (total\text{-}eval\ R\ f\ (relabel\text{-}vars\ I\ J\ g\ (pvar\ R\ i)))$
proof−
have *40*: $total\text{-}eval\ R\ (f \circ g)(\ pvar\ R\ i) = (f \circ g)\ i$
using *total-eval-var*[*of f ∘g i*]
by (*metis 0 var-to-IP-def*)
have *41*: $relabel\text{-}vars\ I\ J\ g\ (pvar\ R\ i) = pvar\ R\ (g\ i)$
by (*simp add*: *A*(*2*) *assms*(*1*) *relabel-vars-is-morphism*(*2*))

have *42*: *total-eval R f* (*relabel-vars I J g* (*pvar R i*)) = *f* (*g i*)
using *total-eval-var*
by (*metis 41 assms*(*3*) *var-to-IP-def*)
show *?thesis*
by (*metis 40 42 comp-apply*)
qed
show *total-eval R* (*f* ∘ *g*) (*p* ⊗$_{Pring\ R\ I}$ *pvar R i*) =
total-eval R f (*relabel-vars I J g* (*p* ⊗$_{Pring\ R\ I}$ *pvar R i*))
using *A 0 1 2 3 4*
by *presburger*
qed
qed

end

end
theory *Indices*
imports *Main*
begin

3 Basic Lemmas for Manipulating Indices and Lists

fun *index-list* **where**
index-list 0 = []|
index-list (*Suc n*) = *index-list n* @ [*n*]

lemma *index-list-length*:
length (*index-list n*) = *n*
by(*induction n*, *simp*, *auto*)

lemma *index-list-indices*:
k < *n* ⟹ (*index-list n*)!*k* = *k*
apply(*induction n*)
apply (*simp*; *fail*)
by (*simp add*: *index-list-length nth-append*)

lemma *index-list-set*:
set (*index-list n*) = {..<*n*}
apply(*induction n*)
apply *force*
by (*metis Zero-not-Suc atLeastLessThan-empty atLeastLessThan-singleton atLeast-LessThan-upt*
diff-Suc-1 index-list.elims ivl-disj-un-singleton(*2*) *lessI lessThan-Suc-atMost less-Suc-eq-le*
set-append sorted-list-of-set-empty sorted-list-of-set-range upt-rec)

fun *flat-map* :: ('*a* => '*b list*) => '*a list* => '*b list* **where**
flat-map f [] = []

|*flat-map f* (*h*#*t*) = (*f h*)@(*flat-map f t*)

abbreviation(*input*) *project-at-indices* (‹π_›) **where**
project-at-indices S as ≡ *nths as S*

fun *insert-at-index* :: *'a list* ⇒*'a* ⇒ *nat* ⇒ *'a list* **where**
insert-at-index as a n= (*take n as*) @ (*a*#(*drop n as*))

lemma *insert-at-index-length*:
shows *length* (*insert-at-index as a n*) = *length as* + *1*
by(*induction n*, *auto*)

lemma *insert-at-index-eq*[*simp*]:
assumes *n* ≤ *length as*
shows (*insert-at-index as a n*)!*n* = *a*
by (*metis assms insert-at-index.elims length-take min.absorb2 nth-append-length*)

lemma *insert-at-index-eq'*[*simp*]:
assumes *n* ≤ *length as*
assumes *k* < *n*
shows (*insert-at-index as a n*)!*k* = *as* ! *k*
using *assms*
by (*simp add*: *nth-append*)

lemma *insert-at-index-eq''*[*simp*]:
assumes *n* < *length as*
assumes *k* ≤ *n*
shows (*insert-at-index as a k*)!(*Suc n*) = *as* ! *n*
using *assms insert-at-index.simps*[*of as a k*]
by (*smt Suc-diff-Suc append-take-drop-id diff-Suc-Suc dual-order.order-iff-strict*
le-imp-less-Suc length-take less-trans min.absorb2 not-le nth-Cons-Suc nth-append)

Correctness of project_at_indices

definition *indices-of* :: *'a list* ⇒ *nat set* **where**
indices-of as = {..<(*length as*)}

lemma *proj-at-index-list-length*[*simp*]:
assumes *S* ⊆ *indices-of as*
shows *length* (*project-at-indices S as*) = *card S*
proof−
have *S* = {*i*. *i* < *length as* ∧ *i* ∈ *S*}
using *assms* **unfolding** *indices-of-def*
by *blast*
thus *?thesis*
using *length-nths*[*of as S*] **by** *auto*
qed

A function which enumerates finite sets

abbreviation(*input*) *set-to-list* :: *nat set* ⇒ *nat list* **where**

set-to-list S ≡ *sorted-list-of-set S*

lemma *set-to-list-set*:
assumes *finite S*
shows *set* (*set-to-list S*) = *S*
by (*simp add*: *assms*)

lemma *set-to-list-length*:
assumes *finite S*
shows *length* (*set-to-list S*) = *card S*
by (*metis assms length-remdups-card-conv length-sort set-sorted-list-of-set sorted-list-of-set-sort-remdups*)

lemma *set-to-list-empty*:
assumes *card S* = *0*
shows *set-to-list S* = []
by (*metis assms length-0-conv length-sorted-list-of-set*)

lemma *set-to-list-first*:
assumes *card S* > *0*
shows *Min S* = *set-to-list S* ! *0*
proof−
have *0*: *set* (*set-to-list S*) = *S*
using *assms card-ge-0-finite set-sorted-list-of-set* **by** *blast*
have *1*: *sorted* (*set-to-list S*)
by *simp*
show *?thesis* **apply**(*rule Min-eqI*)
using *assms card-ge-0-finite* **apply** *blast*
apply (*metis 0 1 in-set-conv-nth less-Suc0 less-or-eq-imp-le not-less-eq sorted-iff-nth-mono-less*)
by (*metis 0 Max-in assms card-0-eq card-ge-0-finite gr-zeroI in-set-conv-nth*
not-less0)
qed

lemma *set-to-list-last*:
assumes *card S* > *0*
shows *Max S* = *last* (*set-to-list S*)
proof−
have *0*: *set* (*set-to-list S*) = *S*
using *assms card-ge-0-finite set-sorted-list-of-set* **by** *blast*
have *1*: *sorted* (*set-to-list S*)
by *simp*
show *?thesis* **apply**(*rule Max-eqI*)
using *assms card-ge-0-finite* **apply** *blast*
apply (*smt 0 1 Suc-diff-1 in-set-conv-nth last-conv-nth le-simps*(*2*) *length-greater-0-conv*

less-or-eq-imp-le nat-neq-iff neq0-conv not-less-eq sorted-iff-nth-mono-less)
by (*metis 0 assms card.empty empty-set last-in-set less-numeral-extra*(*3*))
qed

lemma *set-to-list-insert-Max*:

assumes *finite S*
assumes $\bigwedge s.\ s \in S \Longrightarrow a > s$
shows *set-to-list* (*insert a S*) = *set-to-list S* @[*a*]
by (*metis assms*(*1*) *assms*(*2*) *card-0-eq card-insert-if finite.insertI infinite-growing*

insert-not-empty less-imp-le-nat sorted-insort-is-snoc sorted-list-of-set(*1*) *sorted-list-of-set*(*2*)

sorted-list-of-set-insert)

lemma *set-to-list-insert-Min*:
assumes *finite S*
assumes $\bigwedge s.\ s \in S \Longrightarrow a < s$
shows *set-to-list* (*insert a S*) = *a*#*set-to-list S*
by (*metis assms*(*1*) *assms*(*2*) *insort-is-Cons nat-less-le sorted-list-of-set*(*1*) *sorted-list-of-set-insert*)

fun *nth-elem* **where**
nth-elem S n = *set-to-list S* ! *n*

lemma *nth-elem-closed*:
assumes *i* < *card S*
shows *nth-elem S i* ∈ *S*
by (*metis assms card.infinite not-less0 nth-elem.elims nth-mem set-to-list-length sorted-list-of-set*(*1*))

lemma *nth-elem-Min*:
assumes *card S* > *0*
shows *nth-elem S 0* = *Min S*
by (*simp add*: *assms set-to-list-first*)

lemma *nth-elem-Max*:
assumes *card S* > *0*
shows *nth-elem S* (*card S* − *1*) = *Max S*
proof−
have *last* (*set-to-list S*) = *set-to-list S* ! (*card S* − *1*)
by (*metis assms card-0-eq card-ge-0-finite last-conv-nth neq0-conv set-to-list-length sorted-list-of-set-eq-Nil-iff*)
thus *?thesis*
using *assms set-to-list-last set-to-list-length*
by *simp*
qed

lemma *nth-elem-Suc*:
assumes *card S* > *Suc n*
shows *nth-elem S* (*Suc n*) > *nth-elem S n*
using *assms sorted-sorted-list-of-set*[*of S*] *set-to-list-length*[*of S*]
by (*metis Suc-lessD card.infinite distinct-sorted-list-of-set lessI nat-less-le not-less0 nth-elem.elims nth-eq-iff-index-eq sorted-iff-nth-mono-less*)

lemma *nth-elem-insert-Min*:

assumes *card S > 0*
assumes *a < Min S*
shows *nth-elem (insert a S) (Suc i) = nth-elem S i*
using *assms*
by (*metis Min-gr-iff card-0-eq card-ge-0-finite neq0-conv nth-Cons-Suc nth-elem.elims set-to-list-insert-Min*)

lemma *set-to-list-Suc-map*:
assumes *finite S*
shows *set-to-list (Suc ‘ S) = map Suc (set-to-list S)*
proof−
obtain *n* **where** *n-def*: *n = card S*
by *blast*
have ⋀*S*. *card S = n* ⟹ *set-to-list (Suc ‘ S) = map Suc (set-to-list S)*
proof(*induction n*)
case *0*
then show *?case*
by (*metis card-eq-0-iff finite-imageD image-is-empty inj-Suc list.simps*(*8*) *set-to-list-empty*)
next
case (*Suc n*)
have *0*: *S = insert (Min S) (S − {Min S})*
by (*metis Min-in Suc.prems card-gt-0-iff insert-Diff zero-less-Suc*)
have *1*: *sorted-list-of-set (Suc ‘ (S − {Min S})) = map Suc (sorted-list-of-set (S − {Min S}))*
by (*metis 0 Suc.IH Suc.prems card-Diff-singleton card.infinite diff-Suc-1 insertI1 nat.simps*(*3*))
have *2*: *set-to-list S = (Min S)#(set-to-list (S − {Min S}))*
by (*metis 0 DiffD1 Min-le Suc.prems card-Diff-singleton card.infinite card-insert-if diff-Suc-1 finite-Diff n-not-Suc-n nat.simps*(*3*) *nat-less-le set-to-list-insert-Min*)
have *3*: *sorted-list-of-set (Suc ‘ S) = (Min (Suc ‘ S))#(set-to-list ((Suc ‘ S) − {Min (Suc ‘ S)}))*
by (*metis DiffD1 Diff-idemp Min-in Min-le Suc.prems card-Diff1-less card-eq-0-iff finite-Diff finite-imageI image-is-empty insert-Diff nat.simps*(*3*) *nat-less-le set-to-list-insert-Min*)
have *4*: *(Min (Suc ‘ S)) = Suc (Min S)*
by (*metis Min.hom-commute Suc.prems Suc-le-mono card-eq-0-iff min-def nat.simps*(*3*))
have *5*: *sorted-list-of-set (Suc ‘ S) = Suc (Min S)#(set-to-list ((Suc ‘ S) − {Suc (Min S)}))*
using *3 4* **by** *auto*
have *6*: *sorted-list-of-set (Suc ‘ S) = Suc (Min S)#(set-to-list (Suc ‘ (S − {Min S})))*
by (*metis* (*no-types*, *lifting*) *0 5 Diff-insert-absorb image-insert inj-Suc inj-on-insert*)
show *?case*
using *6*
by (*simp add*: *1 2*)

qed
thus *?thesis*
using *n-def* **by** *blast*
qed

lemma *nth-elem-Suc-im*:
assumes $i < card\ S$
shows *nth-elem* (*Suc* ‘ *S*) *i* = *Suc* (*nth-elem S i*)
using *set-to-list-Suc-map*
by (*metis assms card-ge-0-finite dual-order.strict-trans not-gr0 nth-elem.elims nth-map set-to-list-length*)

lemma *set-to-list-upto*:
set-to-list $\{..<n\} = [0..<n]$
by (*simp add*: *lessThan-atLeast0*)

lemma *nth-elem-upto*:
assumes $i < n$
shows *nth-elem* $\{..<n\}$ *i* = *i*
using *set-to-list-upto*
by (*simp add*: *assms*)

Characterizing the entries of project_at_indices

lemma *project-at-indices-append*:
project-at-indices S (*as*@*bs*) = *project-at-indices S as* @ *project-at-indices* {*j*. *j* + *length as* ∈ *S*} *bs*
using *nths-append*[*of as bs S*] **by** *auto*

lemma *project-at-indices-nth*:
assumes $S \subseteq$ *indices-of as*
assumes *card S* > *i*
shows *project-at-indices S as* ! *i* = *as* ! (*nth-elem S i*)
proof−
have $\bigwedge$ *S i*. $S \subseteq$ *indices-of as* ∧ *card S* > *i* ⟹ *project-at-indices S as* ! *i* = *as* ! (*nth-elem S i*)
proof(*induction as*)
case *Nil*
then show *?case*
by (*metis list.size*(*3*) *not-less0 nths-nil proj-at-index-list-length*)
next
case (*Cons a as*)
assume *A*: $S \subseteq$ *indices-of* (*a* # *as*) ∧ *i* < *card S*
have *0*: *nths* (*a* # *as*) *S* = (*if* 0 ∈ *S then* [*a*] *else* []) @ *nths as* {*j*. *Suc j* ∈ *S*}
using *nths-Cons*[*of a as S*] **by** *simp*
show *nths* (*a* # *as*) *S* ! *i* = (*a* # *as*) ! *nth-elem S i*
proof(*cases* 0 ∈ *S*)
case *True*
show *?thesis*
proof(*cases S* = {*0*})

```
      case True
      then show ?thesis
        using 0 Cons.prems  by auto
    next
      case False
      have T0: nths (a # as) S = a#nths as {j. Suc j ∈ S}
        using 0
        by (simp add: True)
      have T1: {j. Suc j ∈ S} ⊆ indices-of as
      proof fix x assume A: x ∈ {j. Suc j ∈ S}
        then have Suc x < length (a#as)
        using Cons.prems indices-of-def by blast
       then show x ∈ indices-of as
        by (simp add: indices-of-def)
      qed
      have T2: ⋀i.  i < card {j. Suc j ∈ S} ⟹ nths as {j. Suc j ∈ S} ! i = as
! nth-elem {j. Suc j ∈ S} i
        using Cons.IH T1 by blast
       have T3: ⋀i.  i < card {j. Suc j ∈ S} ⟹ nth-elem {j. j > 0 ∧ j∈ S} i
= nth-elem S (Suc i)
      proof−
        have 0:  0 < card {j. Suc j ∈ S}
       by (smt Cons.prems Diff-iff Diff-subset False T0 T1 True add-diff-cancel-left'

            card.insert card-0-eq card.infinite finite-subset gr-zeroI insert-Diff
            length-Cons n-not-Suc-n plus-1-eq-Suc proj-at-index-list-length single-
tonI)
        have 1: (insert 0 {j. 0 < j ∧ j ∈ S}) = S
          apply(rule set-eqI) using True  gr0I by blast
        have 2: 0 < Min {j. 0 < j ∧ j ∈ S} using False
            by (metis (mono-tags, lifting) 1 Cons.prems Min-in finite-insert fi-
nite-lessThan
          finite-subset indices-of-def less-Suc-eq less-Suc-eq-0-disj mem-Collect-eq
singleton-conv)
         show ⋀i. i < card {j. Suc j ∈ S} ⟹ nth-elem {j. 0 < j ∧ j ∈ S} i =
nth-elem S (Suc i)
          using 0 1 2 nth-elem-insert-Min[of {j. 0 < j ∧ j ∈ S} 0] True False
          by (metis (no-types, lifting) Cons.prems T0 T1 card-gt-0-iff finite-insert
length-Cons less-SucI proj-at-index-list-length)
      qed
      show nths (a # as) S ! i = (a # as) ! nth-elem S i
        apply(cases i = 0)
           apply (metis Cons.prems Min-le T0 True card-ge-0-finite le-zero-eq
nth-Cons' nth-elem-Min)
      proof−
        assume i ≠ 0
        then have i = Suc (i − 1)
          using Suc-pred' by blast
        hence nths (a # as) S ! i = nths as {j. Suc j ∈ S} ! (i−1)
```

```
          using A by (simp add: T0)
        thus nths (a # as) S ! i = (a # as) ! nth-elem S i
        proof−
          have i − 1 < card {j. Suc j ∈ S}
                by (metis Cons.prems Suc-less-SucD T0 T1 ‹i = Suc (i − 1)›
length-Cons proj-at-index-list-length)
          hence 0: nth-elem {j. 0 < j ∧ j ∈ S} (i − 1) = nth-elem S i
            using T3[of i−1] ‹i = Suc (i − 1)› by auto

          have 1: nths as {j. Suc j ∈ S} ! (i−1) = as ! nth-elem {j. Suc j ∈ S}
(i−1)
            using T2 ‹i − 1 < card {j. Suc j ∈ S}› by blast
          have 2: (a # as) ! nth-elem S i = as! ((nth-elem S i) − 1)
        by (metis Cons.prems ‹i = Suc (i − 1)› not-less0 nth-Cons′ nth-elem-Suc)
          have 3: (nth-elem S i) − 1 = nth-elem {j. Suc j ∈ S} (i−1)
          proof−
            have Suc ‘ {j. Suc j ∈ S} = {j. 0 < j ∧ j ∈ S}
            proof
              show Suc ‘ {j. Suc j ∈ S} ⊆ {j. 0 < j ∧ j ∈ S}
                by blast
              show {j. 0 < j ∧ j ∈ S} ⊆ Suc ‘ {j. Suc j ∈ S}
                using Suc-pred gr0-conv-Suc by auto
            qed
            thus ?thesis
              using 0 ‹i − 1 < card {j. Suc j ∈ S}› nth-elem-Suc-im by fastforce
          qed
          show nths (a # as) S ! i = (a # as) ! nth-elem S i
            using 1 2 3 ‹nths (a # as) S ! i = nths as {j. Suc j ∈ S} ! (i − 1)›
by auto
        qed
      qed
    qed
  next
    case False
    have F0: nths (a # as) S = nths as {j. Suc j ∈ S}
      by (simp add: 0 False)
    have F1: Suc ‘{j. Suc j ∈ S} = S
    proof show Suc ‘ {j. Suc j ∈ S} ⊆ S by auto
          show S ⊆ Suc ‘ {j. Suc j ∈ S}  using False Suc-pred
             by (smt image-iff mem-Collect-eq neq0-conv subsetI)
    qed
    have F2: {j. Suc j ∈ S} ⊆ indices-of as ∧ i < card {j. Suc j ∈ S}
      using F1
     by (metis (mono-tags, lifting) A F0 Suc-less-SucD indices-of-def length-Cons
lessThan-iff
          mem-Collect-eq proj-at-index-list-length subset-iff)
    have F3: project-at-indices {j. Suc j ∈ S} as ! i = as ! (nth-elem {j. Suc j ∈
S} i)
      using F2 Cons(1)[of {j. Suc j ∈ S}] Cons(2)
```

```
      by blast
    then show ?thesis
      using F0 F1 F2 nth-elem-Suc-im by fastforce
  qed
 qed
 then show ?thesis
  using assms(1) assms(2) by blast
qed
```

An inverse for nth_elem

```
definition set-rank where
set-rank S x = (THE i. i < card S ∧ x = nth-elem S i)

lemma set-rank-exist:
 assumes finite S
 assumes x ∈ S
 shows ∃ i. i < card S ∧ x = nth-elem S i
 using assms nth-elem.simps[of S]
 by (metis in-set-conv-nth set-to-list-length sorted-list-of-set(1))

lemma set-rank-unique:
 assumes finite S
 assumes x ∈ S
 assumes i < card S ∧ x = nth-elem S i
 assumes j < card S ∧ x = nth-elem S j
 shows i = j
 using assms nth-elem.simps[of S]
 by (simp add: ‹i < card S ∧ x = nth-elem S i› ‹j < card S ∧ x = nth-elem S j›
     nth-eq-iff-index-eq set-to-list-length)

lemma nth-elem-set-rank-inv:
 assumes finite S
 assumes x ∈ S
 shows nth-elem S (set-rank S x) = x
 using the-equality  set-rank-unique set-rank-exist assms
 unfolding set-rank-def
 by smt

lemma set-rank-nth-elem-inv:
 assumes finite S
 assumes i < card S
 shows set-rank S (nth-elem S i) = i
 using the-equality  set-rank-unique set-rank-exist assms
 unfolding set-rank-def
proof −
 show (THE n. n < card S ∧ nth-elem S i = nth-elem S n) = i
  using assms(1) assms(2) nth-elem-closed set-rank-unique by blast
qed
```

lemma *set-rank-range*:
assumes *finite S*
assumes $x \in S$
shows *set-rank S x* < *card S*
using *assms*(*1*) *assms*(*2*) *set-rank-exist set-rank-nth-elem-inv* **by** *fastforce*

lemma *project-at-indices-nth'*:
assumes $S \subseteq$ *indices-of as*
assumes $i \in S$
shows *as* ! *i* = *project-at-indices S as* ! (*set-rank S i*)
by (*metis assms*(*1*) *assms*(*2*) *finite-lessThan finite-subset indices-of-def nth-elem-set-rank-inv*

project-at-indices-nth set-rank-range)

fun *proj-away-from-index* :: *nat* ⇒ *'a list* ⇒ *'a list* (‹$\pi_{\neq \_}$›)**where**
proj-away-from-index n as = (*take n as*)@(*drop* (*Suc n*) *as*)

proj_away_from_index is an inverse to insert_at_index

lemma *insert-at-index-project-away*[*simp*]:
assumes *k* < *length as*
assumes *bs* = (*insert-at-index as a k*)
shows $\pi_{\neq k}$ *bs* = *as*
using *assms insert-at-index.simps*[*of as a k*] *proj-away-from-index.simps*[*of k bs*]
by (*simp add*: ‹*k* < *length as*› *less-imp-le-nat min.absorb2*)

definition *fibred-cell* :: *'a list set* ⇒ (*'a list* ⇒ *'a* ⇒ *bool*) ⇒ *'a list set* **where**
fibred-cell C P = {*as* . ∃ *x t*. *as* = (*t*#*x*) ∧ *x* ∈ *C* ∧ (*P x t*)}

definition *fibred-cell-at-ind* :: *nat* ⇒ *'a list set* ⇒ (*'a list* ⇒ *'a* ⇒ *bool*) ⇒ *'a list set* **where**
fibred-cell-at-ind n C P = {*as* . ∃ *x t*. *as* = (*insert-at-index x t n*) ∧ *x* ∈ *C* ∧ (*P x t*)}

lemma *fibred-cell-lengths*:
assumes ⋀*k*. *k* ∈ *C* ⟹ *length k* = *n*
shows *k* ∈ (*fibred-cell C P*) ⟹ *length k* = *Suc n*
proof−
assume *k* ∈ (*fibred-cell C P*)
obtain *x t* **where** *k* = (*t*#*x*) ∧ *x* ∈ *C* ∧ *P x t*
proof −
assume *a1*: ⋀*t x*. *k* = *t* # *x* ∧ *x* ∈ *C* ∧ *P x t* ⟹ *thesis*
have ∃ *as a*. *k* = *a* # *as* ∧ *as* ∈ *C* ∧ *P as a*
using ‹*k* ∈ *fibred-cell C P*› *fibred-cell-def* **by** *blast*
then show *?thesis*
using *a1* **by** *blast*
qed
then show *?thesis*
by (*simp add*: *assms*)
qed

lemma *fibred-cell-at-ind-lengths*:
assumes $\bigwedge k.\ k \in C \Longrightarrow \mathit{length}\ k = n$
assumes $k \leq n$
shows $c \in (\mathit{fibred\text{-}cell\text{-}at\text{-}ind}\ k\ C\ P) \Longrightarrow \mathit{length}\ c = \mathit{Suc}\ n$
proof−
assume $c \in (\mathit{fibred\text{-}cell\text{-}at\text{-}ind}\ k\ C\ P)$
then obtain $x\ t$ **where** $c = (\mathit{insert\text{-}at\text{-}index}\ x\ t\ k) \wedge x \in C \wedge (P\ x\ t)$
using *assms*
unfolding *fibred-cell-at-ind-def*
by *blast*
then show *?thesis*
by (*simp add*: *assms*(*1*))
qed

lemma *project-fibred-cell*:
assumes $\bigwedge k.\ k \in C \Longrightarrow \mathit{length}\ k = n$
assumes $k < n$
assumes $\forall\, x \in C.\ \exists\, t.\ P\ x\ t$
shows $\pi_{\neq\ k}$ ‘ $(\mathit{fibred\text{-}cell\text{-}at\text{-}ind}\ k\ C\ P) = C$
proof
show $\pi_{\neq k}$ ‘ *fibred-cell-at-ind k C P* $\subseteq C$
proof
fix x
assume *x-def*: $x \in \pi_{\neq k}$ ‘ *fibred-cell-at-ind k C P*
then obtain c **where** *c-def*: $x = \pi_{\neq k}\ c \wedge c \in$ *fibred-cell-at-ind k C P*
by *blast*
then obtain $y\ t$ **where** *yt-def*: $c = (\mathit{insert\text{-}at\text{-}index}\ y\ t\ k) \wedge y \in C \wedge (P\ y\ t)$
using *assms*
unfolding *fibred-cell-at-ind-def*
by *blast*
have *0*: $x = \pi_{\neq k}\ c$
by (*simp add*: *c-def*)
have *1*: $y = \pi_{\neq k}\ c$
using *yt-def assms*(*1*) *assms*(*2*)
by (*metis insert-at-index-project-away*)
have *2*: $x = y$ **using** *0 1* **by** *auto*
then show $x \in C$
by (*simp add*: *yt-def*)
qed
show $C \subseteq \pi_{\neq k}$ ‘ *fibred-cell-at-ind k C P*
proof fix x
assume *A*: $x \in C$
obtain t **where** *t-def*: $P\ x\ t$
using *assms A* **by** *auto*
then show $x \in \pi_{\neq k}$ ‘ *fibred-cell-at-ind k C P*
proof −
have *f1*: $\forall\, a\ n\ A\ as.\ \mathit{take}\ n\ as\ @\ (a{::}'a)\ \#\ \mathit{drop}\ n\ as \notin A \vee as \in \pi_{\neq n}$ ‘ $A \vee$
$\neg\ n < \mathit{length}\ as$

by (*metis insert-at-index.simps insert-at-index-project-away rev-image-eqI*)

have $\forall\, n.\ \exists\, as\ a.\ take\ n\ x\ @\ t\ \#\ drop\ n\ x = insert\text{-}at\text{-}index\ as\ a\ n \wedge as \in C \wedge P\ as\ a$
using *A t-def* **by** *auto*
then have $\forall\, n.\ take\ n\ x\ @\ t\ \#\ drop\ n\ x \in \{insert\text{-}at\text{-}index\ as\ a\ n\ |as\ a.\ as \in C \wedge P\ as\ a\}$
by *blast*
then have $x \in \pi_{\neq k}\ `\ \{insert\text{-}at\text{-}index\ as\ a\ k\ |as\ a.\ as \in C \wedge P\ as\ a\}$
using *f1* **by** (*metis* (*lifting*) *A assms*(*1*) *assms*(*2*))
then show *?thesis*
by (*simp add*: *fibred-cell-at-ind-def*)
qed
qed
qed

definition *list-segment* **where**
list-segment i j as = *map* (*nth as*) [*i*..<*j*]

lemma *list-segment-length*:
assumes $i \leq j$
assumes $j \leq$ *length as*
shows *length* (*list-segment i j as*) = $j - i$
using *assms*
unfolding *list-segment-def*
by (*metis length-map length-upt*)

lemma *list-segment-drop*:
assumes $i <$ *length as*
shows (*list-segment i* (*length as*) *as*) = *drop i as*
by (*metis One-nat-def Suc-diff-Suc add-diff-inverse-nat drop0 drop-map drop-upt less-Suc-eq list-segment-def map-nth neq0-conv not-less0 plus-1-eq-Suc*)

lemma *list-segment-concat*:
assumes $j \leq k$
assumes $i \leq j$
shows (*list-segment i j as*) @ (*list-segment j k as*) = (*list-segment i k as*)
using *assms* **unfolding** *list-segment-def*
using *le-Suc-ex upt-add-eq-append*
by *fastforce*

lemma *list-segment-subset*:
assumes $j \leq k$
shows *set* (*list-segment i j as*) $\subseteq$ *set* (*list-segment i k as*)
apply(*cases* $i > j$)
unfolding *list-segment-def*
apply (*metis in-set-conv-nth length-map list.size*(*3*) *order.asym subsetI upt-rec zero-order*(*3*))
proof−

assume $\neg\ j < i$
then have $i \leq j$
using *not-le*
by *blast*
then have *list-segment i j as @ list-segment j k as = list-segment i k as*
using *assms list-segment-concat*[*of j k i as*] **by** *auto*
then show *set* (*map* ((!) *as*) [*i*..<*j*]) ⊆ *set* (*map* ((!) *as*) [*i*..<*k*])
using *set-append* **unfolding** *list-segment-def*
by (*metis Un-upper1*)
qed

lemma *list-segment-subset-list-set*:
assumes $j \leq$ *length as*
shows *set* (*list-segment i j as*) ⊆ *set as*
apply(*cases* $i \geq j$)
apply (*simp add*: *list-segment-def*)
proof−
assume *A*: ¬ $j \leq i$
then have *B*: $i < j$
by *auto*
have *0*: *list-segment i* (*length as*) *as* = *drop i as*
using *B assms list-segment-drop*[*of i as*] *less-le-trans*
by *blast*
have *1*: *set* (*list-segment i j as*) ⊆ *set* (*list-segment i* (*length as*) *as*)
using *B assms list-segment-subset*[*of j length as i as*]
by *blast*
then show *?thesis*
using *assms 0 dual-order.trans set-drop-subset*[*of i as*]
by *metis*
qed

definition *fun-inv* **where**
fun-inv = *inv*

end
theory *Ring-Powers*
imports *HOL*−*Algebra.Chinese-Remainder HOL*−*Combinatorics.List-Permutation*
Padic-Ints.Function-Ring HOL−*Algebra.Generated-Rings Cring-Multivariable-Poly*
Indices
begin

type-synonym *arity* = *nat*
type-synonym *'a tuple* = *'a list*

4 Cartesian Powers of a Ring

4.1 Constructing the Cartesian Power of a Ring

Powers of a ring

`R_list n R` produces the list $[R, ..., R]$ of length n

fun *R-list* :: *nat* ⇒ ('*a*, '*b*) *ring-scheme* ⇒ (('*a*, '*b*) *ring-scheme*) *list* **where**
R-list n R = *map* (λ-. *R*) (*index-list n*)

Cartesian powers of a ring

definition *cartesian-power* :: ('*a*, '*b*) *ring-scheme* ⇒ *nat* ⇒ ('*a list*) *ring* (‹-⁻› *80*)
where
R^n ≡ *RDirProd-list* (*R-list n R*)

lemma *R-list-length*:
length (*R-list n R*) = *n*
apply(*induction n*) **by** *auto*

lemma *R-list-nth*:
i < *n* ⟹ *R-list n R* ! *i* = *R*
by (*simp add*: *index-list-length*)

lemma *cartesian-power-car-memI*:
assumes *length as* = *n*
assumes *set as* ⊆ *carrier R*
shows *as* ∈ *carrier* (R^n)
unfolding *cartesian-power-def*
apply(*rule RDirProd-list-carrier-memI*)
using *R-list-length assms*(*1*) **apply** *auto*[*1*]
by (*metis R-list-length R-list-nth assms*(*1*) *assms*(*2*) *nth-mem subsetD*)

lemma *cartesian-power-car-memI'*:
assumes *length as* = *n*
assumes ⋀*i*. *i* < *n* ⟹ *as* ! *i* ∈ *carrier R*
shows *as* ∈ *carrier* (R^n)
unfolding *cartesian-power-def*
apply(*rule RDirProd-list-carrier-memI*)
using *R-list-length assms*(*1*) **apply** *auto*[*1*]
by (*metis R-list-length R-list-nth assms*(*2*))

lemma *cartesian-power-car-memE*:
assumes *as* ∈ *carrier* (R^n)
shows *length as* = *n*
using *RDirProd-list-carrier-mem*(*1*)
by (*metis R-list-length assms cartesian-power-def*)

lemma *cartesian-power-car-memE'*:
assumes *as* ∈ *carrier* (R^n)

assumes $i < n$
shows *as* ! $i \in$ *carrier* R
using *assms RDirProd-list-carrier-mem*(2)
by (*metis* (*no-types*, *lifting*) *R-list-length R-list-nth cartesian-power-def*)

lemma *cartesian-power-car-memE''*:
assumes *as* $\in$ *carrier* (R^n)
shows *set as* $\subseteq$ *carrier R*
using *cartesian-power-car-memE'*
by (*metis assms cartesian-power-car-memE in-set-conv-nth subsetI*)

lemma *cartesian-power-car-memI''*:
assumes *length as* = $n + k$
assumes *take n as* $\in$ *carrier* (R^n)
assumes *drop n as* $\in$ *carrier* (R^k)
shows *as* $\in$ *carrier* (R^{n+k})
apply(*rule cartesian-power-car-memI'*)
apply (*simp add*: *assms*(1))
proof− **fix** *i* **assume** *A*: $i < n + k$
show *as* ! $i \in$ *carrier R*
apply(*cases* $i < n$)
apply (*metis assms*(2) *cartesian-power-car-memE' nth-take*)
by (*metis A add-diff-inverse-nat add-less-imp-less-left*
append-take-drop-id assms(2) *assms*(3) *cartesian-power-car-memE*
cartesian-power-car-memE' nth-append-length-plus)
qed

lemma *cartesian-power-cons*:
assumes *as* $\in$ *carrier* (R^n)
assumes *a* $\in$ *carrier R*
shows *a*#*as* $\in$ *carrier* (R^{n+1})
apply(*rule cartesian-power-car-memI*)
apply (*metis One-nat-def assms*(1) *cartesian-power-car-memE list.size*(4))
by (*metis assms*(1) *assms*(2) *cartesian-power-car-memE cartesian-power-car-memE'*
in-set-conv-nth set-ConsD subsetI)

lemma *cartesian-power-append*:
assumes *as* $\in$ *carrier* (R^n)
assumes *a* $\in$ *carrier R*
shows *as*@[*a*] $\in$ *carrier* (R^{n+1})
apply(*rule cartesian-power-car-memI''*)
apply (*metis add.commute assms*(1) *cartesian-power-car-memE length-append-singleton*
plus-1-eq-Suc)
apply (*metis append-eq-append-conv-if assms*(1) *butlast-snoc cartesian-power-car-memE*
length-append-singleton lessI take-butlast)
by (*metis add.commute add.right-neutral append-eq-conv-conj assms*(1) *assms*(2)
bot-least
cartesian-power-car-memE cartesian-power-car-memI cartesian-power-cons
list.set(1) *list.size*(3))

lemma *cartesian-power-head*:
assumes *as* ∈ *carrier* ($R^{Suc\ n}$)
shows *hd as* ∈ *carrier R*
by (*metis assms cartesian-power-car-memE cartesian-power-car-memE'' list.set-sel*(*1*)
list.size(*3*) *old.nat.distinct*(*1*) *subsetD*)

lemma *cartesian-power-tail*:
assumes *as* ∈ *carrier* ($R^{Suc\ n}$)
shows *tl as* ∈ *carrier* (R^n)
apply(*rule cartesian-power-car-memI*)
apply (*metis add-diff-cancel-left' assms cartesian-power-car-memE length-tl plus-1-eq-Suc*)
by (*metis assms cartesian-power-car-memE cartesian-power-car-memE'' list.set-sel*(*2*)
list.size(*3*) *nat.simps*(*3*) *subsetD subsetI*)

lemma *insert-at-index-closed*:
assumes *length as* = *n*
assumes *as* ∈ *carrier* (R^n)
assumes *a* ∈ *carrier R*
assumes *k* ≤ *n*
shows (*insert-at-index as a k*) ∈ *carrier* ($R^{Suc\ n}$)
apply(*rule cartesian-power-car-memI'*)
apply (*metis Groups.add-ac*(*2*) *assms*(*1*) *insert-at-index-length plus-1-eq-Suc*)
by (*smt R-list-length Suc-le-eq assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*)
cartesian-power-car-memE' insert-at-index-eq insert-at-index-eq'
insert-at-index-eq'' less-Suc-eq less-Suc-eq-0-disj not-less-eq-eq)

lemma *insert-at-index-pow-not-car*:
assumes *k* ≤*n*
assumes *length x* = *n*
assumes (*insert-at-index x a k*) ∈ *carrier* ($R^{Suc\ n}$)
shows *x* ∈ *carrier* (R^n)
apply(*rule cartesian-power-car-memI'*)
apply (*simp add*: *assms*(*2*))
by (*metis Suc-mono assms*(*1*) *assms*(*2*) *assms*(*3*)
cartesian-power-car-memE' insert-at-index-eq'
insert-at-index-eq'' leI less-SucI)

lemma *insert-at-index-pow-not-car'*:
assumes *k* ≤*n*
assumes *length x* = *n*
assumes *x* ∉ *carrier* (R^n)
shows (*insert-at-index x a n*) ∉ *carrier* ($R^{Suc\ n}$)
by (*metis assms*(*2*) *assms*(*3*) *insert-at-index-pow-not-car lessI less-Suc-eq-le*)

lemma *take-closed*:
assumes *k* ≤*n*
assumes *x* ∈ *carrier* (R^n)
shows *take k x* ∈ *carrier* (R^k)

apply(*rule cartesian-power-car-memI*)
apply (*metis assms*(*1*) *assms*(*2*) *cartesian-power-car-memE length-take min.absorb-iff2*)
by (*meson assms*(*2*) *cartesian-power-car-memE''* *set-take-subset subset-trans*)

lemma *drop-closed*:
assumes $k < n$
assumes $x \in carrier\ (R^n)$
shows $drop\ k\ x \in carrier\ (R^{n\ -\ k})$
apply(*rule cartesian-power-car-memI*[*of drop k x n* − *k*])
using *assms*(*2*) *cartesian-power-car-memE length-drop* **apply** *blast*
by (*metis add-diff-inverse-nat assms*(*1*) *assms*(*2*) *cartesian-power-car-memE*
cartesian-power-car-memE' *in-set-conv-nth length-drop less-imp-le-nat*
nat-add-left-cancel-less nth-drop order.asym subsetI)

lemma *last-closed*:
assumes $n > 0$
assumes $x \in carrier\ (R^n)$
shows $last\ x \in carrier\ R$
using *assms*
by (*metis Suc-diff-1 cartesian-power-car-memE cartesian-power-car-memE'*
last-conv-nth lessI list.size(*3*) *neq0-conv*)

lemma *cartesian-power-concat*:
assumes $a \in carrier\ (R^n)$
assumes $b \in carrier\ (R^k)$
shows $a@b \in carrier\ (R^{n+k})$
$b@a \in carrier\ (R^{n+k})$
apply (*metis* (*no-types*, *lifting*) *append-eq-conv-conj assms*(*1*) *assms*(*2*)
cartesian-power-car-memE cartesian-power-car-memI'' *length-append*)
by (*metis* (*no-types*, *lifting*) *add.commute append-eq-conv-conj assms*(*1*) *assms*(*2*)

cartesian-power-car-memE cartesian-power-car-memI'' *length-append*)

lemma *cartesian-power-decomp*:
assumes $a \in carrier\ (R^{n+k})$
obtains *a0 a1* **where** $a0 \in carrier\ (R^n) \wedge a1 \in carrier\ (R^k) \wedge a0@a1 = a$
using *assms*
by (*metis* (*no-types*, *lifting*) *add-diff-cancel-left' append.assoc append-eq-append-conv*

append-take-drop-id cartesian-power-car-memE drop-closed le-add1
le-neq-implies-less length-append take-closed)

lemma *list-segment-pow*:
assumes $as \in carrier\ (R^n)$
assumes $j \leq n$
assumes $i \leq\ j$
shows $list\text{-}segment\ i\ j\ as \in carrier\ (R^{j\ -\ i})$
apply(*rule cartesian-power-car-memI*)
using *list-segment-length assms cartesian-power-car-memE*

apply *blast*
using *assms*
by (*metis cartesian-power-car-memE cartesian-power-car-memE''*
dual-order.trans list-segment-subset-list-set)

lemma *nth-list-segment*:
assumes $as \in carrier\ (R^n)$
assumes $j \leq n$
assumes $i \leq j$
assumes $k < j - i$
shows $(list\text{-}segment\ i\ j\ as)\ !\ k = as\ !\ (i + k)$
unfolding *list-segment-def*
using *assms nth-map-upt*[*of k j i* ((!) *as*)]
by *blast*

4.2 Mapping the Carrier of a Ring to its 1-Dimensional Cartesian Power.

context *cring*
begin

lemma *R1-carI*:
assumes *length as* = *1*
assumes *as*!*0* ∈ *carrier R*
shows $as \in carrier\ (R^1)$
apply(*rule cartesian-power-car-memI*)
using *assms*
apply *blast*
using *assms*
by (*metis in-set-conv-nth less-one subsetI*)

abbreviation(*input*) *to-R1* **where**
to-R1 a ≡ [*a*]

abbreviation(*input*) *to-R* :: *'a list* ⇒ *'a* **where**
to-R as ≡ *as*!*0*

lemma *to-R1-to-R*:
assumes $a \in carrier\ (R^1)$
shows *to-R1* (*to-R a*) = *a*
proof−
have *length a* = *1*
using *assms cartesian-power-car-memE* **by** *blast*
then obtain *b* **where** *a* = [*b*]
by (*metis One-nat-def length-0-conv length-Suc-conv*)
then show *?thesis*
using *assms*
by (*metis nth-Cons-0*)
qed

lemma *to-R-to-R1*:
shows *to-R* (*to-R1 a*) = *a*
by (*meson nth-Cons-0*)

lemma *to-R1-closed*:
assumes *a* ∈ *carrier R*
shows *to-R1 a* ∈ *carrier* (R^1)
proof(*rule R1-carI*)
show *length* [*a*] = *1*
by *simp*
show [*a*] ! *0* ∈ *carrier R*
using *assms to-R-to-R1* **by** *presburger*
qed

lemma *to-R-pow-closed*:
assumes *a* ∈ *carrier* (R^1)
shows *to-R a* ∈ *carrier R*
using *assms cartesian-power-car-memE'* **by** *blast*

lemma *to-R1-intersection*:
assumes *A* ⊆ *carrier R*
assumes *B* ⊆ *carrier R*
shows *to-R1* ‘ (*A* ∩ *B*) = *to-R1*‘ *A* ∩ *to-R1* ‘ *B*
proof
show (λ*a*. [*a*]) ‘ (*A* ∩ *B*) ⊆ (λ*a*. [*a*]) ‘ *A* ∩ (λ*a*. [*a*]) ‘ *B*
by *blast*
show (λ*a*. [*a*]) ‘ *A* ∩ (λ*a*. [*a*]) ‘ *B* ⊆ (λ*a*. [*a*]) ‘ (*A* ∩ *B*)
using *assms*
by *blast*
qed

lemma *to-R1-finite*:
assumes *finite A*
shows *finite* (*to-R1*‘ *A*)
card A = *card* (*to-R1*‘ *A*)
using *assms*
apply *blast*
apply(*rule finite.induct*[*of A*])
apply (*simp add*: *assms*(*1*))
apply *simp*
by (*smt card-insert-if finite-imageI image-iff image-insert list.inject*)

lemma *to-R1-carrier*:
to-R1‘ (*carrier R*)= *carrier* (R^1)
proof
show (λ*a*. [*a*]) ‘ *carrier R* ⊆ *carrier* (R^1)
proof fix *x*
assume *x* ∈ (λ*a*. [*a*]) ‘ *carrier R*

then show $x \in carrier\ (R^1)$
using *cartesian-power-car-memI*[*of x 1 R*]
by (*metis* (*no-types*, *lifting*) *image-iff to-R1-closed*)
qed
show $carrier\ (R^1) \subseteq (\lambda a.\ [a])\ `\ carrier\ R$
proof fix x
assume $x \in carrier\ (R^1)$
then obtain a **where** *a-def*: $a \in carrier\ R \wedge x = [a]$
using *cartesian-power-car-memE'*[*of x R 1*] *cartesian-power-car-memE*[*of x R 1*]
by (*metis less-numeral-extra*(*1*) *to-R1-to-R*)
then show $x \in (\lambda a.\ [a])\ `\ carrier\ R$
by *blast*
qed
qed

lemma *to-R1-diff*:
$to\text{-}R1`\ (A - B) = to\text{-}R1`\ A - to\text{-}R1`\ B$
proof
show $(\lambda a.\ [a])\ `\ (A - B) \subseteq (\lambda a.\ [a])\ `\ A - (\lambda a.\ [a])\ `\ B$
by *blast*
show $(\lambda a.\ [a])\ `\ A - (\lambda a.\ [a])\ `\ B \subseteq (\lambda a.\ [a])\ `\ (A - B)$
by *blast*
qed

lemma *to-R1-complement*:
shows $to\text{-}R1`\ (carrier\ R - A) = carrier\ (R^1) - to\text{-}R1`\ A$
by (*metis to-R1-carrier to-R1-diff*)

lemma *to-R1-subset*:
assumes $A \subseteq B$
shows $to\text{-}R1`\ A \subseteq to\text{-}R1`\ B$
using *assms*
by *blast*

lemma *to-R1-car-subset*:
assumes $A \subseteq carrier\ R$
shows $to\text{-}R1`\ A \subseteq carrier\ (R^1)$
using *assms to-R1-carrier*
by *blast*
end

4.3 Simple Cartesian Products

definition $cartesian\text{-}product :: ('a\ list)\ set \Rightarrow ('a\ list)\ set \Rightarrow ('a\ list)\ set$ **where**
$cartesian\text{-}product\ A\ B \equiv \{xs.\ \exists as \in A.\ \exists bs \in B.\ xs = as@bs\}$

lemma *cartesian-product-closed*:
assumes $A \subseteq carrier\ (R^n)$

assumes $B \subseteq$ *carrier* (R^m)
shows *cartesian-product A B* $\subseteq$ *carrier* (R^{n+m})
proof
fix x
assume A: $x \in$ *cartesian-product A B*
then obtain *as bs* **where** *as-bs-def*: $x = as@bs \wedge as \in A \wedge bs \in B$
unfolding *cartesian-product-def* **by** *blast*
show $x \in$ *carrier* (R^{n+m})
apply(*rule cartesian-power-car-memI'*)
apply (*metis as-bs-def assms cartesian-power-car-memE length-append subsetD*)
using *A* **unfolding** *cartesian-product-def*
by (*metis* (*no-types*, *lifting*) *add-diff-inverse-nat as-bs-def assms*(*1*) *assms*(*2*) *cartesian-power-car-memE cartesian-power-car-memE'* *nat-add-left-cancel-less nth-append subsetD*)
qed

lemma *cartesian-product-closed'*:
assumes $a \in$ *carrier* (R^n)
assumes $b \in$ *carrier* (R^m)
shows $(a@b) \in$ *carrier* (R^{n+m})
proof−
have $a@b \in$ *cartesian-product* $\{a\}$ $\{b\}$
using *cartesian-product-def* **by** *blast*
then show *?thesis*
using *cartesian-product-closed*[*of* $\{a\}$ *R n* $\{b\}$ *m*]
assms
by *blast*
qed

lemma *cartesian-product-carrier*:
cartesian-product (*carrier* (R^n)) (*carrier* (R^m)) = *carrier* (R^{n+m})
proof
show *cartesian-product* (*carrier* (R^n)) (*carrier* (R^m)) $\subseteq$ *carrier* (R^{n+m})
using *cartesian-product-closed*[*of* (*carrier* (R^n)) *R n* (*carrier* (R^m)) *m*]
by *blast*
show *carrier* $(R^{n+m}) \subseteq$ *cartesian-product* (*carrier* (R^n)) (*carrier* (R^m))
proof
fix x
assume A: $x \in$ *carrier* (R^{n+m})
have *0*: *take n x* $\in$ *carrier* (R^n)
apply(*rule cartesian-power-car-memI'*)
apply (*metis A cartesian-power-car-memE le-add1 length-take min.absorb2*)
by (*metis A add.commute cartesian-power-car-memE'* *nth-take trans-less-add2*)
have *1*: *drop n x* $\in$ *carrier* (R^m)
apply(*rule cartesian-power-car-memI'*)
apply (*metis A add-diff-cancel-left' cartesian-power-car-memE length-drop*)
by (*metis A cartesian-power-car-memE cartesian-power-car-memE' le-add1*

nat-add-left-cancel-less nth-drop)
show $x \in$ *cartesian-product* (*carrier* (R^n)) (*carrier* (R^m))
using *0 1*
by (*smt A cartesian-power-decomp cartesian-product-def mem-Collect-eq*)
qed

Higher function rings

qed

lemma *cartesian-product-memI*:
assumes $A \subseteq$ *carrier* (R^n)
assumes $B \subseteq$ *carrier* (R^m)
assumes *take n a* $\in A$
assumes *drop n a* $\in B$
shows $a \in$ *cartesian-product A B*
proof−
have $a =$ (*take n a*) @ (*drop n a*)
by (*metis append-take-drop-id*)
then show *?thesis*
using *assms*(*3*) *assms*(*4*) *cartesian-product-def* **by** *blast*
qed

lemma *cartesian-product-memI'*:
assumes $A \subseteq$ *carrier* (R^n)
assumes $B \subseteq$ *carrier* (R^m)
assumes $a \in A$
assumes $b \in B$
shows $a@b \in$ *cartesian-product A B*
using *assms* **unfolding** *cartesian-product-def*
by *blast*

lemma *cartesian-product-memE*:
assumes $a \in$ *cartesian-product A B*
assumes $A \subseteq$ *carrier* (R^n)
shows *take n a* $\in A$
drop n a $\in B$
using *assms* **unfolding** *cartesian-product-def*
apply (*smt append-eq-conv-conj cartesian-power-car-memE in-mono mem-Collect-eq*)
using *assms* **unfolding** *cartesian-product-def*
by (*smt append-eq-conv-conj cartesian-power-car-memE in-mono mem-Collect-eq*)

lemma *cartesian-product-intersection*:
assumes $A \subseteq$ *carrier* (R^n)
assumes $B \subseteq$ *carrier* (R^m)
assumes $C \subseteq$ *carrier* (R^n)
assumes $D \subseteq$ *carrier* (R^m)
shows *cartesian-product A B* $\cap$ *cartesian-product C D* = *cartesian-product* ($A \cap C$) ($B \cap D$)
proof

show *cartesian-product A B* ∩ *cartesian-product C D* ⊆ *cartesian-product* (*A* ∩ *C*) (*B* ∩ *D*)
proof fix *x*
assume *x* ∈ *cartesian-product A B* ∩ *cartesian-product C D*
then show *x* ∈ *cartesian-product* (*A* ∩ *C*) (*B* ∩ *D*)
using *assms cartesian-product-memE*[*of x C D*] *cartesian-product-memE*[*of x A B*]
cartesian-product-memI[*of A* ∩ *C R n B* ∩ *D m x*]
by (*smt Int-iff inf.coboundedI1*)
qed
show *cartesian-product* (*A* ∩ *C*) (*B* ∩ *D*) ⊆ *cartesian-product A B* ∩ *cartesian-product C D*
proof fix *x*
assume *x* ∈ *cartesian-product* (*A* ∩ *C*) (*B* ∩ *D*)
then show *x* ∈ *cartesian-product A B* ∩ *cartesian-product C D*
using *assms cartesian-product-memI*[*of C R n D m*] *cartesian-product-memI*[*of A R n B m*]
cartesian-product-memE[*of x A* ∩ *B C* ∩ *D R n*]
by (*metis* (*no-types*, *lifting*) *Int-iff cartesian-product-memE*(*1*) *cartesian-product-memE*(*2*) *inf-le1 subset-trans*)
qed
qed

lemma *cartesian-product-subsetI*:
assumes *C* ⊆ *A*
assumes *D* ⊆ *B*
shows *cartesian-product C D* ⊆ *cartesian-product A B*
using *assms* **unfolding** *cartesian-product-def*
by *blast*

lemma *cartesian-product-binary-union-right*:
assumes *C* ⊆ *carrier* (*R*^*n*)
assumes *D* ⊆ *carrier* (*R*^*n*)
shows *cartesian-product A* (*C* ∪ *D*) = (*cartesian-product A C*) ∪ (*cartesian-product A D*)
proof
show *cartesian-product A* (*C* ∪ *D*) ⊆ *cartesian-product A C* ∪ *cartesian-product A D*
unfolding *cartesian-product-def* **by** *blast*
show *cartesian-product A C* ∪ *cartesian-product A D* ⊆ *cartesian-product A* (*C* ∪ *D*)
unfolding *cartesian-product-def* **by** *blast*
qed

lemma *cartesian-product-binary-union-left*:
assumes *C* ⊆ *carrier* (*R*^*n*)
assumes *D* ⊆ *carrier* (*R*^*n*)
shows *cartesian-product* (*C* ∪ *D*) *A* = (*cartesian-product C A*) ∪ (*cartesian-product D A*)

proof
show *cartesian-product* $(C \cup D)$ $A \subseteq$ *cartesian-product* C $A \cup$ *cartesian-product* D A
unfolding *cartesian-product-def* **by** *blast*
show *cartesian-product* C $A \cup$ *cartesian-product* D $A \subseteq$ *cartesian-product* $(C \cup D)$ A
unfolding *cartesian-product-def* **by** *blast*
qed

lemma *cartesian-product-binary-intersection-right*:
assumes $C \subseteq$ *carrier* (R^n)
assumes $D \subseteq$ *carrier* (R^n)
assumes $A \subseteq$ *carrier* (R^m)
shows *cartesian-product* A $(C \cap D) =$ (*cartesian-product* A C) $\cap$ (*cartesian-product* A D)
proof
show *cartesian-product* A $(C \cap D) \subseteq$ *cartesian-product* A $C \cap$ *cartesian-product* A D
unfolding *cartesian-product-def* **by** *blast*
show *cartesian-product* A $C \cap$ *cartesian-product* A $D \subseteq$ *cartesian-product* A $(C \cap D)$
proof fix x **assume** A: $x \in$ *cartesian-product* A $C \cap$ *cartesian-product* A D
show $x \in$ *cartesian-product* A $(C \cap D)$ **apply**(*rule cartesian-product-memI*[*of* A R m - n])
apply (*simp add*: *assms*(*3*))
apply (*simp add*: *assms*(*1*) *inf.coboundedI1*)
apply (*meson A IntD1 assms*(*3*) *cartesian-product-memE*(*1*))
by (*meson A Int-iff assms*(*3*) *cartesian-product-memE*(*2*))
qed
qed

lemma *cartesian-product-binary-intersection-left*:
assumes $C \subseteq$ *carrier* (R^n)
assumes $D \subseteq$ *carrier* (R^n)
assumes $A \subseteq$ *carrier* (R^m)
shows *cartesian-product* $(C \cap D)$ $A =$ (*cartesian-product* C A) $\cap$ (*cartesian-product* D A)
proof
show *cartesian-product* $(C \cap D)$ $A \subseteq$ *cartesian-product* C $A \cap$ *cartesian-product* D A
unfolding *cartesian-product-def* **by** *blast*
show *cartesian-product* C $A \cap$ *cartesian-product* D $A \subseteq$ *cartesian-product* $(C \cap D)$ A
proof fix x **assume** A: $x \in$ *cartesian-product* C $A \cap$ *cartesian-product* D A
show $x \in$ *cartesian-product* $(C \cap D)$ A **apply**(*rule cartesian-product-memI*[*of* - R n - m])
apply (*simp add*: *assms*(*2*) *inf.coboundedI2*)
apply (*simp add*: *assms*(*3*))
apply (*meson A Int-iff assms*(*1*) *assms*(*2*) *cartesian-product-memE*(*1*))

```
        by (meson A IntD1 assms(1) cartesian-product-memE(2))
  qed
qed

lemma cartesian-product-car-complement-right:
  assumes A ⊆ carrier (R^m)
  shows carrier (R^n + m) − cartesian-product (carrier (R^n)) A =
        cartesian-product (carrier (R^n)) ((carrier (R^m)) − A)
proof
  show carrier (R^n + m) − cartesian-product (carrier (R^n)) A ⊆ cartesian-product
(carrier (R^n)) ((carrier (R^m)) − A)
   proof fix x assume A: x ∈ (carrier (R^n + m) − cartesian-product (carrier
(R^n)) A)
  show x ∈ cartesian-product (carrier (R^n)) ((carrier (R^m)) − A)
  apply(rule cartesian-product-memI[of - R n - m])
    apply simp
      apply simp
        apply (meson A DiffE le-add1 take-closed)
    apply(rule ccontr)
    proof−
      assume A': drop n x ∉ (carrier (R^m) − A)
      have drop n x ∈ A
      proof−
      have x ∈ cartesian-product (carrier (R^n)) (carrier (R^m))
        using A
        by (metis (mono-tags, lifting) DiffD1 cartesian-product-carrier)
      then show ?thesis
        using A' cartesian-product-memE[of x (carrier (R^n)) (carrier (R^m)) R n]
        by blast
      qed
      then show False
        using A cartesian-product-memI[of (carrier (R^n)) R n A m x]
        by (meson DiffD1 DiffD2 assms le-add1 order-refl take-closed)
    qed
  qed
  show cartesian-product (carrier (R^n)) ((carrier (R^m)) − A) ⊆ carrier (R^n + m)
− cartesian-product (carrier (R^n)) A
  proof fix x assume A: x ∈ cartesian-product (carrier (R^n)) ((carrier (R^m)) −
A)
    show x ∈ carrier (R^n + m) − cartesian-product (carrier (R^n)) A
      apply(rule ccontr)
      using A cartesian-product-memE[of x carrier (R^n) A R n]
      using A cartesian-product-memE[of x (carrier (R^n)) (carrier (R^m)) − A R
n]
      by (metis (no-types, lifting) DiffD1 DiffD2 DiffI
          append-take-drop-id cartesian-product-closed' order-refl)
  qed
qed
```

lemma *cartesian-product-car-complement-left*:
assumes $A \subseteq$ *carrier* (R^n)
shows *carrier* $(R^{n+m}) -$ *cartesian-product* A (*carrier* (R^m)) =
cartesian-product ((*carrier* $(R^n)) - A$) (*carrier* (R^m))
proof
show *carrier* $(R^{n+m}) -$ *cartesian-product* A (*carrier* $(R^m)) \subseteq$
cartesian-product ((*carrier* $(R^n)) - A$) (*carrier* (R^m))
proof fix x **assume** A: $x \in$ *carrier* $(R^{n+m}) -$ *cartesian-product* A (*carrier* (R^m))
show $x \in$ *cartesian-product* ((*carrier* $(R^n)) - A$) (*carrier* (R^m))
proof(*rule cartesian-product-memI*[*of* - R n - m])
show *carrier* $(R^n) - A \subseteq$ *carrier* (R^n)
by *simp*
show *carrier* $(R^m) \subseteq$ *carrier* (R^m)
by *simp*
show *take* n $x \in$ *carrier* $(R^n) - A$
by (*metis* (*no-types*, *lifting*) A *DiffD1 DiffD2 DiffI assms*
cartesian-product-carrier cartesian-product-memE(2) *cartesian-product-memI*
le-add1 order-refl take-closed)
show *drop* n $x \in$ *carrier* (R^m)
by (*metis* A *DiffD1 cartesian-product-carrier cartesian-product-memE*(2) *order-refl*)
qed
qed
show *cartesian-product* ((*carrier* $(R^n)) - A$) (*carrier* $(R^m)) \subseteq$
carrier $(R^{n+m}) -$ *cartesian-product* A (*carrier* (R^m))
proof fix x **assume** A: $x \in$ *cartesian-product* ((*carrier* $(R^n)) - A$) (*carrier* (R^m))
show $x \in$ *carrier* $(R^{n+m}) -$ *cartesian-product* A (*carrier* (R^m))
apply(*rule ccontr*)
using A *cartesian-product-memE*[*of* x ((*carrier* $(R^n)) - A$) (*carrier* $(R^m))$]
cartesian-product-memE[*of* x A (*carrier* $(R^m))$]
by (*smt DiffD1 DiffD2 DiffI Diff-subset append-take-drop-id assms cartesian-product-closed'*)
qed
qed

lemma *cartesian-product-complement-right*:
assumes $B \subseteq$ *carrier* (R^m)
assumes $A \subseteq$ *carrier* (R^n)
shows *cartesian-product* A (*carrier* $(R^m)) -$ (*cartesian-product* A B) =
cartesian-product A ((*carrier* $(R^m)) - B$)
proof
show *cartesian-product* A (*carrier* $(R^m)) -$ *cartesian-product* A $B \subseteq$ *cartesian-product* A ((*carrier* $(R^m)) - B$)
unfolding *cartesian-product-def* **by** *blast*
show *cartesian-product* A ((*carrier* $(R^m)) - B) \subseteq$ *cartesian-product* A ((*carrier* $(R^m))) -$ *cartesian-product* A B

proof fix x **assume** A: $x \in$ *cartesian-product* A ((*carrier* (R^m)) $-$ B)
have *0*: $x \in$ *cartesian-product* A (*carrier* (R^m))
using A **unfolding** *cartesian-product-def* **by** *blast*
show $x \in$ *cartesian-product* A (*carrier* (R^m)) $-$ *cartesian-product* A B
apply(*rule ccontr*)
using *assms 0 A cartesian-product-memE*[*of x A* ((*carrier* (R^m)) $-$ B) R n]
cartesian-product-memE[*of x A B R n*]
by *blast*
qed
qed

lemma *cartesian-product-complement-left*:
assumes $B \subseteq$ *carrier* (R^m)
assumes $A \subseteq$ *carrier* (R^n)
shows *cartesian-product* (*carrier* (R^m)) A $-$ (*cartesian-product* B A) =
cartesian-product ((*carrier* (R^m)) $-$ B) A
proof
show *cartesian-product* (*carrier* (R^m)) A $-$ *cartesian-product* B $A \subseteq$ *cartesian-product* ((*carrier* (R^m)) $-$ B) A
unfolding *cartesian-product-def* **by** *blast*
show *cartesian-product* ((*carrier* (R^m)) $-$ B) $A \subseteq$ *cartesian-product* (*carrier* (R^m)) A $-$ *cartesian-product* B A
proof fix x **assume** A: $x \in$ *cartesian-product* ((*carrier* (R^m)) $-$ B) A
have *0*: $x \in$ *cartesian-product* (*carrier* (R^m)) A
using A **unfolding** *cartesian-product-def* **by** *blast*
have *1*: *take* m $x \in$ (*carrier* (R^m)) $-$ B
using A *cartesian-product-memE*[*of x* ((*carrier* (R^m)) $-$ B) A R m]
by *blast*
have *2*: *drop* m $x \in A$
using *cartesian-product-memE*[*of x* ((*carrier* (R^m)) $-$ B) A R m]
by (*metis A Diff-subset*)
show $x \in$ *cartesian-product* (*carrier* (R^m)) A $-$ *cartesian-product* B A
apply(*rule ccontr*)
using *A 0 1 2 cartesian-product-memE*[*of x B A R m*] *assms*
by *blast*
qed
qed

lemma *cartesian-product-empty-right*:
assumes $A \subseteq$ *carrier* (R^n)
assumes $B = \{[]\}$
shows *cartesian-product* A $B = A$
proof
show *cartesian-product* A $B \subseteq A$
using *assms* **unfolding** *cartesian-product-def*
by (*smt append-Nil2 mem-Collect-eq singletonD subsetI*)
show $A \subseteq$ *cartesian-product* A B
using *assms* **unfolding** *cartesian-product-def*
by *blast*

qed

lemma *cartesian-product-empty-left*:
assumes $B \subseteq$ *carrier* (R^n)
assumes $A = \{[]\}$
shows *cartesian-product A B* $= B$
proof
show *cartesian-product A B* $\subseteq B$
using *assms* **unfolding** *cartesian-product-def*
by (*smt append.simps*(*1*) *mem-Collect-eq singletonD subsetI*)
show $B \subseteq$ *cartesian-product A B*
using *assms* **unfolding** *cartesian-product-def*
by *blast*
qed

4.4 Cartesian Products at Arbitrary Indices

definition(**in** *ring*) *ring-pow-proj* :: *nat* $\Rightarrow$ (*nat set*) $\Rightarrow$ ('*a list*) $\Rightarrow$ ('*a list*) ($\langle\pi_{\text{-}},\ \text{-}\rangle$) **where**
ring-pow-proj n S $\equiv$ *restrict* (*project-at-indices S*) (*carrier* (R^n))

The projection at an arbitrary index set

lemma *project-at-indices-closed*:
assumes $a \in$ *carrier* (R^n)
assumes $S \subseteq$ *indices-of a*
shows $\pi_S\ a \in$ *carrier* $(R^{card\ S})$
apply(*rule cartesian-power-car-memI'*)
using *assms proj-at-index-list-length* **apply** *blast*
using *assms project-at-indices-nth*[*of S*]
by (*smt cartesian-power-car-memE cartesian-power-car-memE' indices-of-def lessThan-iff nth-elem-closed subsetD*)

lemma(**in** *ring*) *ring-pow-proj-is-map*:
assumes $S \subseteq \{..<n\}$
shows $\pi_{n,S} \in$ *struct-maps* (R^n) $(R^{card\ S})$
proof(*rule struct-maps-memI*)
show $\bigwedge x.\ x \in$ *carrier* $(R^n) \Longrightarrow \pi_{n,S}\ x \in$ *carrier* $(R^{card\ S})$
using *project-at-indices-closed* **unfolding** *ring-pow-proj-def*
by (*metis assms cartesian-power-car-memE indices-of-def restrict-apply'*)
show $\bigwedge x.\ x \notin$ *carrier* $(R^n) \Longrightarrow \pi_{n,\ S}\ x =$ *undefined*
by (*metis restrict-apply ring-pow-proj-def*)
qed

lemma(**in** *ring*) *project-at-indices-ring-pow-proj*:
assumes $x \in$ *carrier* (R^n)
shows $\pi_S\ x = \pi_{n,S}\ x$
unfolding *ring-pow-proj-def*
by (*metis assms restrict-apply'*)

Cartesian products where the first factor A occurs at the entries of some arbitrary index set. Note that this product isn't completely arbitrary because the entries of the factor of A still occurs in ascending order.

definition *twisted-cartesian-product* (‹*Prod*$_{\_,\ \_}$›) **where**
twisted-cartesian-product S S′ A B = {*a* . *length a* = *card S* + *card S′* ∧ π_S *a* ∈ *A* ∧ $\pi_{S'}$ *a* ∈ *B*}

lemma *twisted-cartesian-product-mem-length*:
assumes *card S* = *n*
assumes *card S′* = *m*
assumes *a* ∈ $Prod_{S,S'}$ *A B*
shows *length a* = *n* + *m*
using *assms* **unfolding** *twisted-cartesian-product-def*
by *blast*

lemma *twisted-cartesian-product-closed*:
assumes *A* ⊆ *carrier* (R^n)
assumes *B* ⊆ *carrier* (R^m)
assumes *card S* = *n*
assumes *card S′* = *m*
assumes *S* ∪ *S′* = {..<*n* + *m*}
shows *twisted-cartesian-product S S′ A B* ⊆ *carrier* ($R^{n\ +\ m}$)
proof(*rule subsetI*)
fix *x* **assume** *A*: *x* ∈ *twisted-cartesian-product S S′ A B*
show *x* ∈ *carrier* ($R^{n\ +\ m}$)
proof(*rule cartesian-power-car-memI′*)
show *length x* = *n* + *m*
using *twisted-cartesian-product-mem-length* ‹*x* ∈ *twisted-cartesian-product S S′ A B*› *assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *assms*(*5*) **by** *blast*
fix *i* **assume** *A′*: *i* < *n* + *m*
have *0*: *indices-of x* = {..<*n*+*m*}
by (*simp add*: ‹*length x* = *n* + *m*› *indices-of-def*)
show *x* ! *i* ∈ *carrier R*
proof(*cases i* ∈ *S*)
case *True*
have *x*!*i* = π_S*x* ! (*set-rank S i*)
using *A′ 0 assms*
by (*metis True Un-upper1 project-at-indices-nth′*)
then show *?thesis*
using *project-at-indices-closed*[*of x R n* + *m S*] *A A′*
cartesian-power-car-memE′[*of* π_S *x R card S*]
by (*metis* (*no-types*, *lifting*) *True UnI2 Un-upper1 assms*(*1*) *assms*(*3*) *assms*(*5*)
finite-lessThan finite-subset mem-Collect-eq set-rank-range sup.absorb-iff1
twisted-cartesian-product-def)
next
case *False*
have *x*!*i* = $\pi_{S'}$*x* ! (*set-rank S′ i*)

using A' *0 assms*
by (*metis False UnE lessThan-iff project-at-indices-nth' sup.absorb-iff1 sup.right-idem*)
then show *?thesis*
using *project-at-indices-closed*[*of x R n + m S'*] *A A'*
cartesian-power-car-memE'[*of* $\pi_{S'}$ *x R card S'*]
by (*metis* (*no-types*, *lifting*) *False UnE UnI2 Un-upper2 assms*(*2*) *assms*(*4*) *assms*(*5*)
finite-lessThan finite-subset lessThan-iff mem-Collect-eq set-rank-range sup.absorb-iff1
twisted-cartesian-product-def)
qed
qed
qed

lemma *twisted-cartesian-product-memE*:
assumes $a \in$ *twisted-cartesian-product S S' A B*
shows $\pi_S\ a \in A\ \pi_{S'}\ a \in B$
using *assms*(*1*) **unfolding** *twisted-cartesian-product-def* **apply** *blast*
using *assms*(*1*) **unfolding** *twisted-cartesian-product-def* **by** *blast*

lemma *twisted-cartesian-product-memI*:
assumes $\pi_S\ a \in A$
assumes $\pi_{S'}\ a \in B$
assumes *length a* = *card S* + *card S'*
shows $a \in$ *twisted-cartesian-product S S' A B*
by (*metis* (*mono-tags*, *lifting*) *assms*(*1*) *assms*(*2*) *assms*(*3*) *mem-Collect-eq twisted-cartesian-product-def*)

lemma *twisted-cartesian-product-empty-left-factor*:
assumes $A = \{\}$
shows *twisted-cartesian-product S S' A B* = $\{\}$
by (*metis assms emptyE equals0I twisted-cartesian-product-memE*(*1*))

lemma *twisted-cartesian-product-empty-right-factor*:
assumes $B = \{\}$
shows *twisted-cartesian-product S S' A B* = $\{\}$
by (*metis assms emptyE equals0I twisted-cartesian-product-memE*(*2*))

lemma *twisted-cartesian-project-left*:
assumes $A \subseteq$ *carrier* (R^n)
assumes $B \subseteq$ *carrier* (R^m)
assumes $A \neq \{\}$
assumes $B \neq \{\}$
assumes *card S* = *n*
assumes *card S'* = *m*
assumes $S \cup S' = \{..<n + m\}$
shows π_S ‘ ($Prod_{S,S'}$ *A B*) = *A*
proof
have *f0*: $S \cap S' = \{\}$

proof−
have *card* $(S \cup S') = card\ S + card\ S'$
by (*simp add*: *assms*(5) *assms*(6) *assms*(7))
thus *?thesis*
by (*metis Nat.add-diff-assoc2 add.right-neutral add-diff-cancel-left' assms*(6) *assms*(7) *card-0-eq card-Un-Int finite-Int finite-Un finite-lessThan le-add1*)
qed
show π_S ‘ $(Prod_{S,S'}\ A\ B) \subseteq A$
unfolding *twisted-cartesian-product-def*
by *blast*
show $A \subseteq \pi_S$ ‘ $(Prod_{S,S'}\ A\ B)$
proof fix x **assume** *A*: $x \in A$
obtain y **where** *y-def*: $y \in B$
using *assms*(4) **by** *blast*
obtain a **where** *a-def*:
$a = map\ (\lambda i.\ if\ i \in S\ then\ (x\ !\ set\text{-}rank\ S\ i)\ else\ (y\ !\ set\text{-}rank\ S'\ i))\ [0..<n+m]$
by *blast*
have *0*: $S \subseteq$ *indices-of a*
by (*metis* (*no-types*, *lifting*) *Un-upper1 a-def assms*(7) *diff-zero indices-of-def length-map length-upt*)
have *1*: $S' \subseteq$ *indices-of a*
by (*metis* (*no-types*, *lifting*) *Un-upper2 a-def assms*(7) *diff-zero indices-of-def length-map length-upt*)
have *2*: $\pi_S\ a = x$
proof−
have *20*: *length* $(\pi_S\ a) = n$
by (*metis* (*no-types*, *lifting*) *Un-upper1 a-def assms*(5) *assms*(7) *diff-zero indices-of-def length-map length-upt proj-at-index-list-length*)
have $\bigwedge i.\ i < n \Longrightarrow \pi_S\ a\ !\ i = x\ !\ i$
proof− **fix** *i* **assume** *A*: $i < n$ **show** $\pi_S\ a\ !\ i = x\ !\ i$
using *0 assms a-def project-at-indices-nth'*[*of S a nth-elem S i*] *set-rank-nth-elem-inv*[*of S i*]
nth-map[*of i* [*0..<n+m*]]
by (*smt* (*z3*) *A add.left-neutral card.infinite diff-zero indices-of-def length-map length-upt lessThan-iff not-less-zero nth-elem-closed nth-map nth-upt subsetD*)
qed
thus *?thesis* **using** *20*
by (*metis A assms*(1) *cartesian-power-car-memE nth-equalityI subsetD*)
qed
have *3*: $\pi_{S'}\ a = y$
proof−
have *20*: *length* $(\pi_{S'}\ a) = m$
using *1 assms*(6) *proj-at-index-list-length* **by** *blast*
have $\bigwedge i.\ i < m \Longrightarrow \pi_{S'}\ a\ !\ i = y\ !\ i$
proof− **fix** *i* **assume** *A*: $i < m$
have *nth-elem* $S'\ i \notin S$
using *nth-elem-closed*[*of i S'*] *f0 A assms*(6) **by** *blast*
thus $\pi_{S'}\ a\ !\ i = y\ !\ i$

using *0 assms a-def project-at-indices-nth'[of S' a nth-elem S' i] set-rank-nth-elem-inv[of S' i]*
nth-map[of i [0..<n+m]]
by (*smt 1 A add.left-neutral card.infinite diff-zero indices-of-def length-map length-upt lessThan-iff not-less0 nth-elem-closed nth-map nth-upt subsetD*)
qed
thus *?thesis*
by (*metis 20 assms(2) cartesian-power-car-memE nth-equalityI subsetD y-def*)
qed
have $a \in (Prod_{S,S'}\ A\ B)$
apply(*rule twisted-cartesian-product-memI*)
apply (*simp add: 2 A*)
apply (*simp add: 3 y-def*)
by (*metis* (*no-types, lifting*) *a-def assms(5) assms(6) diff-zero length-map length-upt*)
thus $x \in \pi_S\ `\ (Prod_{S,S'}\ A\ B)$
using *2* **by** *blast*
qed
qed

lemma *twisted-cartesian-product-swap*:
shows $(Prod_{S,S'}\ A\ B) = (Prod_{S',S}\ B\ A)$
unfolding *twisted-cartesian-product-def*
by (*metis add.commute*)

lemma *twisted-cartesian-project-right*:
assumes $A \subseteq carrier\ (R^n)$
assumes $B \subseteq carrier\ (R^m)$
assumes $A \neq \{\}$
assumes $B \neq \{\}$
assumes $card\ S = n$
assumes $card\ S' = m$
assumes $S \cup S' = \{..<n + m\}$
shows $\pi_{S'}\ `\ (Prod_{S,S'}\ A\ B) = B$
using *assms twisted-cartesian-project-left[of B R m A n S' S] twisted-cartesian-product-swap*
by (*metis add.commute sup-commute*)

Cartesian products which send points $a = (a_1, \ldots, a_m)$ and $b = (b_1, \ldots, b_n)$ to the point $(a_1, \ldots, a_i, b_1, \ldots, b_n, a_{i+1}, \ldots, a_m)$

definition *splitting-permutation* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat* **where**
splitting-permutation l1 l2 i j = (*if j < i then j else*
(*if i ≤ j ∧ j < l1 then* (*l2 + j*) *else*
(*if j < l1 + l2 then j − l1 + i else j*)))

lemma *splitting-permutation-case-1-unique*:
assumes $i \leq l1$
assumes $y < i$

assumes *splitting-permutation l1 l2 i j* = *y*
shows *j* = *y*
unfolding *splitting-permutation-def*
using *assms*(*2*) *assms*(*3*) *splitting-permutation-def* **by** *auto*

lemma *splitting-permutation-case-1-exists*:
assumes *i* ≤ *l1*
assumes *y* < *i*
shows *splitting-permutation l1 l2 i y* = *y*
unfolding *splitting-permutation-def*
by (*simp add*: *assms*(*2*))

lemma *splitting-permutation-case-2-unique*:
assumes *i* ≤ *l1*
assumes *i* ≤ *y* ∧ *y* < *l2* + *i*
assumes *splitting-permutation l1 l2 i j* = *y*
shows *j* = *y* + *l1* − *i*
unfolding *splitting-permutation-def*
using *assms*(*1*) *assms*(*2*) *assms*(*3*) *le-add-diff-inverse2 not-less-iff-gr-or-eq*
splitting-permutation-def trans-less-add2 **by** *auto*

lemma *splitting-permutation-case-2-exists*:
assumes *i* ≤ *l1*
assumes *i* ≤ *y* ∧ *y* < *l2* + *i*
shows *splitting-permutation l1 l2 i* (*y* + *l1* − *i*) = *y*
unfolding *splitting-permutation-def*
using *assms*(*1*) *assms*(*2*) *less-diff-conv2* **by** *auto*

lemma *splitting-permutation-case-3-unique*:
assumes *i* ≤ *l1*
assumes *l2* + *i* ≤ *y* ∧ *y* < *l1* + *l2*
assumes *splitting-permutation l1 l2 i j* = *y*
shows *j* = *y* − *l2*
unfolding *splitting-permutation-def*
by (*smt Nat.le-diff-conv2 add-diff-cancel-left' add-diff-cancel-right' add-leD2*
assms(*2*) *assms*(*3*) *le-add1 le-diff-iff not-le splitting-permutation-def*)

lemma *splitting-permutation-case-3-exists*:
assumes *i* ≤ *l1*
assumes *l2* + *i* ≤ *y* ∧ *y* < *l1* + *l2*
shows *splitting-permutation l1 l2 i* (*y* − *l2*) = *y*
unfolding *splitting-permutation-def*
by (*metis Nat.le-diff-conv2 add.commute add-leD1 assms*(*2*) *leD le-add-diff-inverse*
less-diff-conv2)

lemma *splitting-permutation-case-4-unique*:
assumes *i* ≤ *l1*
assumes *l1* + *l2* ≤ *y*
assumes *splitting-permutation l1 l2 i j* = *y*

shows $j = y$
using *assms(1) assms(2) assms(3) le-add-diff-inverse2 less-le-trans splitting-permutation-def* **by** *auto*

lemma *splitting-permutation-case-4-exists*:
assumes $i \leq l1$
assumes $l1 + l2 \leq y$
shows *splitting-permutation l1 l2 i y* $= y$
unfolding *splitting-permutation-def*
using *assms(2)* **by** *auto*

lemma *splitting-permutation-permutes*:
assumes $i \leq l1$
shows (*splitting-permutation l1 l2 i*) *permutes* $\{..< l1 + l2\}$
proof−
have *0*: $(\forall x.\ x \notin \{..<l1 + l2\} \longrightarrow$ *splitting-permutation l1 l2 i x* $= x)$
proof fix *x* **show** $x \notin \{..<l1 + l2\} \longrightarrow$ *splitting-permutation l1 l2 i x* $= x$
proof assume *A*: $x \notin \{..<l1 + l2\}$
then show *splitting-permutation l1 l2 i x* $= x$
using *assms* **unfolding** *splitting-permutation-def*
by *simp*
qed
qed
have *1*: $(\forall y.\ \exists!x.$ *splitting-permutation l1 l2 i x* $= y)$
proof fix *y*
show $\exists!x.$ *splitting-permutation l1 l2 i x* $= y$
proof(*cases* $y < i$)
case *True*
then show *?thesis*
using *splitting-permutation-case-1-exists splitting-permutation-case-1-unique assms*
by (*metis splitting-permutation-def*)
next
case *F0*: *False*
show *?thesis*
proof(*cases* $i \leq y \wedge y < l2 + i$)
case *True*
then show *?thesis*
using *F0 splitting-permutation-case-2-exists splitting-permutation-case-2-unique assms*
by *metis*
next
case *F1*: *False*
show *?thesis*
proof(*cases* $l2 + i \leq y \wedge y < l1 + l2$)
case *True*
then show *?thesis*
using *F0 F1 splitting-permutation-case-3-exists splitting-permutation-case-3-unique assms*

by *metis*
next
case *F2*: *False*
show *?thesis*
using *F0 F1 F2 splitting-permutation-case-4-exists splitting-permutation-case-4-unique assms*
by (*metis leI not-less*)
qed
qed
qed
qed
show *?thesis*
using *0 1 permutes-def*
by *blast*
qed

lemma *splitting-permutation-action*:
assumes $i \leq l1$
assumes *length a1* = *l1*
assumes *length a2* = *l2*
shows *permute-list* (*splitting-permutation l1 l2 i*) ((*take i a1*) @ *a2* @ (*drop i a1*)) =
a1@*a2*
proof−
obtain *x* **where** *x-def*: *x* = *permute-list* (*splitting-permutation l1 l2 i*) ((*take i a1*) @ *a2* @ (*drop i a1*))
by *blast*
obtain *y* **where** *y-def*: *y* = *a1* @ *a2*
by *blast*
have *0*: *length x* = *length y*
using *x-def y-def assms splitting-permutation-permutes*[*of i l1 l2*]
by (*smt add.commute add.left-commute le-add-diff-inverse length-append length-drop length-permute-list length-take min.absorb2*)
have *1*: $\bigwedge i.\ i < l1 + l2 \Longrightarrow x\ !\ i = y\ !\ i$
proof− **fix** *j* **assume** *A*: *j* < *l1* + *l2*
show *x* ! *j* = *y* ! *j*
apply(*cases j* < *i*)
apply (*smt 0 A append-take-drop-id assms*(*1*) *assms*(*2*) *assms*(*3*) *length-append length-permute-list length-take less-le-trans min.absorb2 nth-append permute-list-nth splitting-permutation-case-1-exists splitting-permutation-permutes x-def y-def*)
apply(*cases i* ≤ *j* ∧ *j* < *l1*)
apply (*smt 0 A add.left-commute append-take-drop-id assms*(*1*) *assms*(*2*) *assms*(*3*) *le-add-diff-inverse length-append length-permute-list length-take min.absorb2 nth-append nth-append-length-plus permute-list-nth splitting-permutation-def splitting-permutation-permutes x-def y-def*)
using *x-def y-def assms*
by (*smt 0 A add.commute add-diff-cancel-left' add-diff-inverse-nat length-append length-permute-list length-take less-diff-conv min.absorb2 not-le nth-append permute-list-nth splitting-permutation-case-1-unique splitting-permutation-def splitting-permutation-permutes*)

qed
have *2*: *length x = l1 + l2*
by (*simp add*: *x-def assms*(*2*) *assms*(*3*))
have *3*: *x = y*
using *0 1 2*
by (*metis nth-equalityI*)
then show *?thesis*
using *x-def y-def*
by *blast*
qed

definition *scp-permutation* **where**
scp-permutation l1 l2 i = fun-inv (*splitting-permutation l1 l2 i*)

lemma *scp-permutation-action*:
assumes *i* ≤*l1*
assumes *length a1 = l1*
assumes *length a2 = l2*
shows *permute-list* (*scp-permutation l1 l2 i*) (*a1*@*a2*) = ((*take i a1*) @ *a2* @ (*drop i a1*))
proof−
have (*scp-permutation l1 l2 i*) ∘ (*splitting-permutation l1 l2 i*) = *id*
by (*metis assms*(*1*) *fun-inv-def permutes-inv-o*(*2*) *scp-permutation-def splitting-permutation-permutes*)
then have *permute-list* ((*scp-permutation l1 l2 i*) ∘ (*splitting-permutation l1 l2 i*)) ((*take i a1*) @ *a2* @ (*drop i a1*)) =
((*take i a1*) @ *a2* @ (*drop i a1*))
by (*metis permute-list-id*)
then show *?thesis* **using** *splitting-permutation-action permute-list-compose*
by (*smt* ‹*scp-permutation l1 l2 i* ∘ *splitting-permutation l1 l2 i = id*› *assms*(*1*) *assms*(*2*) *assms*(*3*) *fun-inv-def length-append length-permute-list permutes-inv permutes-inv-o*(*1*) *scp-permutation-def splitting-permutation-permutes*)
qed

lemma *scp-permutes*:
assumes *i* ≤*l1*
shows (*scp-permutation l1 l2 i*) *permutes* {..<*l1 + l2*}
by (*simp add*: *assms*(*1*) *fun-inv-def permutes-inv scp-permutation-def splitting-permutation-permutes*)

definition *split-cartesian-product* **where**
split-cartesian-product l1 l2 i A B = permute-list (*scp-permutation l1 l2 i*) ‘ (*cartesian-product A B*)

lemma *split-cartesian-product-memI*:
assumes *a1*@*a2* ∈ *A*
assumes *b* ∈ *B*
assumes *A* ⊆ *carrier* (R^{l1})
assumes *B* ⊆ *carrier* (R^{l2})

assumes *length a1 = i*
shows *a1@b@a2 ∈ split-cartesian-product l1 l2 i A B*
proof−
have *P*: *a1@a2@b ∈ cartesian-product A B*
by (*metis append.assoc assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *cartesian-product-memI'*)

have *0*: *i ≤ l1*
using *assms*
by (*metis cartesian-power-car-memE le-add1 length-append subset-iff*)
have *1*: *length* (*a1@a2*) = *l1*
using *assms*(*1*) *assms*(*3*) *cartesian-power-car-memE*
by *blast*
have *2*: *length b = l2*
using *assms*(*2*) *assms*(*4*) *cartesian-power-car-memE*
by *blast*
have *3*: *take i* (*a1 @ a2*) = *a1*
by (*simp add*: *assms*(*5*))
have *4*: *drop i* (*a1 @ a2*) = *a2*
by (*simp add*: *assms*(*5*))
have *permute-list* (*scp-permutation l1 l2 i*) ((*a1 @ a2*) *@ b*) = *take i* (*a1 @ a2*)
@ b @ drop i (*a1 @ a2*)
using *0 1 2 scp-permutation-action*[*of i l1 a1@a2 b l2*]
by *blast*
then have *permute-list* (*scp-permutation l1 l2 i*) ((*a1@a2*)*@b*) = *a1@b@a2*
by(*simp only*: *3 4*)
then have *permute-list* (*scp-permutation l1 l2 i*) (*a1@a2@b*) = *a1@b@a2*
by *simp*
then show *?thesis*
using *P* **unfolding** *split-cartesian-product-def*
by (*metis* (*mono-tags*, *lifting*) *image-eqI*)
qed

lemma *split-cartesian-product-memI'*:
assumes *a ∈ A*
assumes *b ∈ B*
assumes *A ⊆ carrier* (R^{l1})
assumes *B ⊆ carrier* (R^{l2})
assumes *i ≤ l1*
shows (*take i a*)*@b@*(*drop i a*) ∈ *split-cartesian-product l1 l2 i A B*
using *assms split-cartesian-product-memI*[*of take i a drop i a A b B R l1 l2 i*]
by (*metis append-take-drop-id cartesian-power-car-memE length-take min.absorb2 subset-iff*)

lemma *split-cartesian-product-memE*:
assumes *a ∈ split-cartesian-product l1 l2 i A B*
assumes *A ⊆ carrier* (R^{l1})
assumes *B ⊆ carrier* (R^{l2})
assumes *i ≤ l1*
shows (*take i a*)*@*(*drop* (*i + l2*) *a*) ∈ *A*

```
      (drop i (take (i + l2) a)) ∈ B
proof−
 obtain b where b-def: b ∈ cartesian-product A B ∧ a = permute-list (scp-permutation
l1 l2 i) b
   using assms split-cartesian-product-def
   by (metis (mono-tags, lifting) image-iff)
 then have 0: (take l1 b) ∈ A ∧ (drop l1 b) ∈ B
  using assms(2) cartesian-product-memE(1)[of b A B R l1] cartesian-product-memE(2)[of
b A B R l1]
   by metis
 have 1: a = (take i (take l1 b))@(drop l1 b)@(drop i (take l1 b))
   using  0 append-take-drop-id assms(2) assms(3) assms(4) b-def
     cartesian-power-car-memE scp-permutation-action subsetD
   by smt
 have 2: (take i a) = (take i (take l1 b))
   using 0 1
   by (metis (no-types, lifting) append-eq-append-conv append-take-drop-id
       assms(4) b-def length-permute-list length-take min.absorb1 take-take)
 have drop (i + l2) a = drop i (take l1 b)
 proof−
   have drop (i + l2) ( (take i (take l1 b))@(drop l1 b)@(drop i (take l1 b))) =
drop i (take l1 b)
     using assms
     by (metis 0 1 2 add.commute append-eq-conv-conj append-take-drop-id
         cartesian-power-car-memE drop-drop subsetD)
   then show ?thesis
     using 1
     by blast
 qed
 then show take i a @ drop (i + l2) a ∈ A
   by (metis 0 2 append-take-drop-id)
 have 3: length b = l1 + l2
   by (metis 0 append-take-drop-id assms(2) assms(3) cartesian-power-car-memE
length-append subsetD)
 then have (drop i (take (i + l2) ((take i (take l1 b))@(drop l1 b)@(drop i (take
l1 b))))) = (drop l1 b)
 proof−
   have 0: take (i + l2) ((take i (take l1 b))@(drop l1 b)@(drop i (take l1 b))) =
         take (i + l2) ((take i b)@(drop l1 b)@(drop i (take l1 b)))
     using assms(4)
     by (metis min.absorb1 take-take)
   have 1: length ((take i b)@(drop l1 b)) = i  + l2
     using 3  assms
      by (metis (no-types, opaque-lifting) add-diff-cancel-left' b-def length-append
length-drop
        length-permute-list length-take  min.absorb2 trans-le-add1)
   have 2: take (i + l2) (((take i b)@(drop l1 b))@(drop i (take l1 b))) = (take i
b)@(drop l1 b)
     using 1
```

by (*metis append-eq-conv-conj*)
have *3*: *take* (*i* + *l2*) ((*take i b*)@(*drop l1 b*)@(*drop i* (*take l1 b*))) = (*take i b*)@(*drop l1 b*)
using *2*
by (*metis append.assoc*)
have *4*: *take* (*i* + *l2*) ((*take i* (*take l1 b*))@(*drop l1 b*)@(*drop i* (*take l1 b*))) = (*take i b*)@(*drop l1 b*)
using *0 3*
by *presburger*
then have *5*: (*drop i* (*take* (*i* + *l2*) ((*take i* (*take l1 b*))@(*drop l1 b*)@(*drop i* (*take l1 b*))))) =
drop i ((*take i b*)@(*drop l1 b*))
by *presburger*
have *length* (*take i b*) = *i*
by (*metis 1 append-take-drop-id assms*(*4*) *le-add1 length-take min.absorb2 min.bounded-iff nat-le-linear take-all*)
then show *?thesis* **using** *5*
by (*metis append-eq-conv-conj*)
qed
then have *drop i* (*take* (*i* + *l2*) *a*) = *drop l1 b*
using *1* **by** *blast*
then show (*drop i* (*take* (*i* + *l2*) *a*)) ∈ *B*
using *0*
by *presburger*
qed

lemma *split-cartesian-product-mem-length*:
assumes *a* ∈ *split-cartesian-product l1 l2 i A B*
assumes *A* ⊆ *carrier* (R^{l1})
assumes *B* ⊆ *carrier* (R^{l2})
assumes *i* ≤ *l1*
shows *length a* = *l1* + *l2*
using *assms* **unfolding** *split-cartesian-product-def*
using *cartesian-product-closed*[*of A R l1 B l2*] *scp-permutes*[*of i l1 l2*]
by (*smt cartesian-power-car-memE imageE in-mono length-permute-list scp-permutation-def*)

lemma *split-cartesian-product-memE′*:
assumes *a1*@*b*@*a2* ∈ *split-cartesian-product l1 l2 i A B*
assumes *A* ⊆ *carrier* (R^{l1})
assumes *B* ⊆ *carrier* (R^{l2})
assumes *i* ≤ *l1*
assumes *length a1* = *i*
assumes *length b* = *l2*
assumes *length as* = (*l1* − *i*)
shows *a1*@*a2* ∈ *A*
b ∈ *B*
using *assms split-cartesian-product-memE*(*1*)[*of a1*@*b*@*a2 l1 l2 i A B R*]
apply (*metis append.assoc append-eq-conv-conj length-append*)
using *assms split-cartesian-product-memE*(*2*)[*of a1*@*b*@*a2 l1 l2 i A B R*]

by (*metis add-diff-cancel-left′ append-eq-conv-conj drop-take*)

lemma *split-cartesian-product-closed*:
assumes $A \subseteq$ *carrier* (R^{l1})
assumes $B \subseteq$ *carrier* (R^{l2})
assumes $i \leq l1$
shows *split-cartesian-product l1 l2 i A B* $\subseteq$ *carrier* ($R^{l1 + l2}$)
proof fix x
assume *A*: $x \in$ *split-cartesian-product l1 l2 i A B*
show $x \in$ *carrier* ($R^{l1 + l2}$)
apply(*rule cartesian-power-car-memI*)
apply (*meson ‹x ∈ split-cartesian-product l1 l2 i A B› assms(1) assms(2) assms(3) split-cartesian-product-mem-length*)
using *assms A* **unfolding** *split-cartesian-product-def*
using *cartesian-product-closed*[*of A R l1 B l2*]
by (*smt A cartesian-power-car-memE′′ image-iff length-permute-list scp-permutes set-permute-list split-cartesian-product-mem-length subsetD*)
qed

General function for permuting the elements of a simple cartesian product:

definition *intersperse* :: (*nat* ⇒ *nat*) ⇒ *′a tuple* ⇒ *′a tuple* ⇒ *′a tuple* **where**
intersperse σ as bs = *permute-list σ* (*as*@*bs*)

lemma *intersperseE*:
assumes *σ permutes* ({..<*n*})
assumes *length as* + *length bs* = *n*
shows *length* (*intersperse σ as bs*) = *n*
by (*metis assms(2) intersperse-def length-append length-permute-list*)

lemma *intersperseE′*:
assumes *σ permutes* ({..<*n*})
assumes *length as* + *length bs* = *n*
assumes *length as* = *k*
assumes *σ i* < *k*
shows (*intersperse σ as bs*)! *i* = *as* ! *σ i*
proof−
have *permute-list σ* (*as* @ *bs*) ! *i* = (*as* @ *bs*) ! *σ i*
using *assms permute-list-nth*[*of σ* (*as*@*bs*) *i*]
unfolding *intersperse-def*
by (*metis length-append lessThan-iff permutes-not-in trans-less-add1*)
then show *?thesis* **using** *assms*
by (*metis intersperse-def nth-append*)
qed

lemma *intersperseE′′*:
assumes *σ permutes* ({..<*n*})
assumes *length as* + *length bs* = *n*
assumes *length as* = *k*
assumes *i* < *n*

assumes $\sigma\ i \geq k$
shows $(intersperse\ \sigma\ as\ bs)!\ i\ =\ bs\ !\ ((\sigma\ i)\ -\ k)$
proof−
have *0*: $permute\text{-}list\ \sigma\ (as\ @\ bs)\ !\ i = (as\ @\ bs)\ !\ \sigma\ i$
using *assms permute-list-nth*[*of* $\sigma\ (as@bs)\ i$]
unfolding *intersperse-def*
proof −
have $(as\ @\ bs)\ !\ \sigma\ i = (as\ @\ bs)\ !\ \sigma\ ([0..<n]\ !\ i)$
by (*simp add*: ‹$i < n$›)
then show *?thesis*
by (*metis* (*no-types*) ‹$i < n$› ‹*length as* + *length bs* = n› *diff-zero length-append*

length-upt nth-map permute-list-def)
qed
have *1*: $\sigma\ i < n$
using *assms*
by (*meson lessThan-iff permutes-in-image*)
have *2*: $(\sigma\ i)\ -\ k < length\ bs$
using *1 assms*(*2*) *assms*(*3*) *assms*(*5*) **by** *linarith*
have $(as\ @\ bs)\ !\ (\sigma\ i)\ =\ bs\ !\ (\sigma\ i\ -\ length\ as)$
using *assms 1 2 nth-append*[*of as bs* $(\sigma\ i)$]
by (*meson not-le*)
then have *3*: $(as\ @\ bs)\ !\ (\sigma\ i)\ =\ bs\ !\ (\sigma\ i\ -\ k)$
using *assms*
by *blast*
have *4*: $permute\text{-}list\ \sigma\ (as\ @\ bs)\ !\ i = (as\ @\ bs)\ !\ (\sigma\ i)$
using *0* **by** *blast*
show *?thesis* **using** *4 3* **unfolding** *intersperse-def*
by *auto*
qed

Some more lemmas about the project_at_indices function.

lemma *project-at-indices-consecutive-ind-length*:
assumes $(i::nat) < j$
assumes $j \leq n$
assumes $length\ a = n$
shows $length\ (project\text{-}at\text{-}indices\ \{i..<j\}\ a) = j - i$
using *assms proj-at-index-list-length*[*of* $\{i..<j\}\ a$]
unfolding *indices-of-def*
by (*metis card-atLeastLessThan ivl-subset le-less-linear lessThan-atLeast0 not-less0*)

lemma *project-at-indices-consecutive-ind-length′*:
assumes $(i::nat) < j$
assumes $j \leq n$
assumes $a \in carrier\ (R^n)$
shows $length\ (project\text{-}at\text{-}indices\ \{i..<j\}\ a) = j - i$
using *assms*(*1*) *assms*(*2*) *assms*(*3*) *cartesian-power-car-memE project-at-indices-consecutive-ind-length*
by *blast*

lemma *sorted-list-of-set-from-up-to*:
assumes $(i::nat) < j$
assumes $k < j - i$
shows *sorted-list-of-set* $\{i..<j\}$! $k = i + k$
using *assms* **apply**(*induction k*)
apply *simp* **by** *simp*

lemma *nth-elem-consecutive-indices*:
assumes $(i::nat) < j$
assumes $k < j - i$
shows *nth-elem* $\{i..<j\}$ $k = i + k$
using *nth-elem.simps*[*of* $\{i..<j\}$ k] *sorted-list-of-set-from-up-to assms*(*2*)
by *auto*

lemma *project-at-indices-consecutive-indices*:
assumes $(i::nat) < j$
assumes $j \leq n$
assumes *length* $a = n$
assumes $k < j - i$
shows (*project-at-indices* $\{i..<j\}$ a) ! $k = a!$ $(i + k)$
using *assms nth-elem-consecutive-indices*[*of i j k*]
by (*metis atLeast0LessThan card-atLeastLessThan indices-of-def ivl-subset linorder-le-less-linear not-less0 project-at-indices-nth*)

lemma *project-at-indices-consecutive-indices'*:
assumes $(i::nat) < j$
assumes $j \leq n$
assumes $a \in$ *carrier* (R^n)
assumes $k < j - i$
shows (*project-at-indices* $\{i..<j\}$ a) ! $k = a!$ $(i + k)$
using *assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *cartesian-power-car-memE project-at-indices-consecutive-indice*
by *blast*

lemma *tl-as-projection*:
assumes $a \in$ *carrier* (R^n)
shows *tl* a = *project-at-indices* $\{1::nat..<n\}$ a
proof−
have *0*: *indices-of* $a = \{..<n\}$
using *assms cartesian-power-car-memE indices-of-def*
by *blast*
have *1*: *length* (*tl* a) $= n - 1$
using *assms cartesian-power-car-memE length-tl*
by *blast*
have *2*: *length* (*tl* a) = *length* (*project-at-indices* $\{1::nat..<n\}$ a)
using *0 assms cartesian-power-car-memE*[*of a R n*] *proj-at-index-list-length*[*of* $\{1::nat..<n\}$ a]
by (*metis 1 atLeastLessThan-iff card-atLeastLessThan lessThan-iff subsetI*)
have $\bigwedge i.\ i < n - 1 \Longrightarrow$ (*tl* a) ! i = (*project-at-indices* $\{1::nat..<n\}$ a) ! i

using *project-at-indices-consecutive-indices'*[*of 1 n n a R*] *assms*
by (*metis 1 One-nat-def Suc-leI le-add-diff-inverse2 le-numeral-extra*(*4*)
linorder-neqE-nat nat-add-left-cancel-le nat-diff-split-asm not-less0 nth-tl plus-1-eq-Suc)
then show *?thesis*
by (*metis 1 2 nth-equalityI*)
qed

4.5 Function Rings on Cartesian Powers

Complement operator

definition *ring-pow-comp* :: ('*a*, '*b*) *ring-scheme* ⇒ *arity* ⇒ '*a tuple set* ⇒ '*a tuple set* **where**
ring-pow-comp R n S ≡ *carrier* (R^n) − *S*

lemma *ring-pow-comp-closed*:
ring-pow-comp R n S ⊆ *carrier* (R^n)
by (*simp add*: *ring-pow-comp-def*)

lemma *ring-pow-comp-disjoint*:
ring-pow-comp R n S ∩ *S* = {}
by (*simp add*: *ring-pow-comp-def inf-sup-aci*(*1*))

lemma *ring-pow-comp-union*:
assumes *S* ⊆ *carrier* (R^n)
shows (*ring-pow-comp R n S*) ∪ *S* = *carrier* (R^n)
by (*metis ring-pow-comp-def Un-Diff-cancel2 assms sup.absorb-iff1*)

lemma *ring-pow-comp-carrier*:
ring-pow-comp R n (*carrier* (R^n)) = {}
by (*simp add*: *ring-pow-comp-def*)

lemma *ring-pow-comp-empty*:
ring-pow-comp R n {} = (*carrier* (R^n))
by (*simp add*: *ring-pow-comp-def*)

lemma *ring-pow-comp-demorgans*:
assumes *A* ⊆ *carrier* (R^n)
assumes *B* ⊆ *carrier* (R^n)
shows *ring-pow-comp R n* (*A* ∪ *B*) = (*ring-pow-comp R n A*) ∩ (*ring-pow-comp R n B*)
by (*simp add*: *ring-pow-comp-def Diff-Un*)

lemma *ring-pow-comp-demorgans'*:
assumes *A* ⊆ *carrier* (R^n)
assumes *B* ⊆ *carrier* (R^n)
shows *ring-pow-comp R n* (*A* ∩ *B*) = (*ring-pow-comp R n A*) ∪ (*ring-pow-comp R n B*)
by (*simp add*: *ring-pow-comp-def Diff-Int*)

lemma *ring-pow-comp-inv*:
assumes $A \subseteq$ *carrier* (R^n)
shows *ring-pow-comp R n* (*ring-pow-comp R n A*) $= A$
by (*simp add*: *ring-pow-comp-def assms double-diff*)

The function ring defined on the powers of a ring:

abbreviation(*input*) *ring-pow-function-ring* (‹*Fun*$_-$ -›) **where**
ring-pow-function-ring n R $\equiv$ *function-ring* (*carrier* (R^n)) *R*

Partial function application. Given a function $f(x_1, \ldots, x_{n+1})$, an index i and a point $a \in$ carrier R returns the function $(x_1, .., x_n) \mapsto f(x_1, \ldots, x_{i-1}, a, x_i, \ldots, x_n)$

lemma *ring-pow-function-ring-car-memE*:
assumes $f \in$ *carrier* $(Fun_n\ R)$
shows $f \in$ *extensional* (*carrier* (R^n))
$f \in$ *carrier* $(R^n) \to$ *carrier R*
using *ring-functions.function-ring-car-memE*[*of R f carrier* (R^n)] *assms*
unfolding *ring-functions-def*
using *function-ring-def partial-object.select-convs*(*1*) **apply** (*metis PiE-iff*)
using *Int-iff assms PiE-iff function-ring-def partial-object.select-convs*(*1*)
by (*simp add*: *PiE-iff function-ring-def*)

definition *partial-eval* :: ('*a*, '*b*) *ring-scheme* $\Rightarrow$ *arity* $\Rightarrow$ *nat* $\Rightarrow$ ('*a list* $\Rightarrow$ '*a*) $\Rightarrow$
'*a* $\Rightarrow$ ('*a list* $\Rightarrow$ '*a*) **where**
partial-eval R m n f c = *restrict* (λ *as*. *f* (*insert-at-index as c n*)) (*carrier* (R^m))

context *ring*
begin

lemma *function-ring-car-mem-closed*:
assumes $f \in$ *carrier* (*function-ring S R*)
assumes $s \in S$
shows *f s* $\in$ *carrier R*
using *assms* **unfolding** *function-ring-def ring-record-simps* **by** *blast*

lemma *function-ring-car-mem-closed'*:
assumes $f \in$ *carrier* $(Fun_{Suc\ k}\ R)$
assumes $s \in$ *carrier* $(R^{Suc\ k})$
shows *f s* $\in$ *carrier R*
using *assms* **unfolding** *function-ring-def ring-record-simps* **by** *blast*

lemma(**in** *ring*) *partial-eval-domain*:
assumes $f \in$ *carrier* $(Fun_{Suc\ k}\ R)$
assumes $a \in$ *carrier R*
assumes $n \leq k$
shows (*partial-eval R k n f a*) $\in$ *carrier* $(Fun_k\ R)$
apply(*rule ring-functions.function-ring-car-memI*)
proof−
show $\bigwedge x.\ x \in$ *carrier* $(R^k) \Longrightarrow$ (*partial-eval R k n f a*) $x \in$ (*carrier R*)

proof−
fix x
assume A: $x \in carrier\ (R^k)$
show $(partial\text{-}eval\ \ R\ k\ n\ f\ a)\ \ x \in (carrier\ R)$
proof$(cases\ n = k)$
case *True*
then have $(partial\text{-}eval\ \ R\ k\ n\ f\ a)\ \ x = f\ (insert\text{-}at\text{-}index\ x\ a\ n)$
by $(metis\ (no\text{-}types,\ lifting)\ A\ restrict\text{-}apply'\ partial\text{-}eval\text{-}def)$
then show $(partial\text{-}eval\ \ R\ k\ n\ f\ a)\ x \in carrier\ R$
using *insert-at-index-closed*[*of x k R a n*] *assms ring-functions.function-ring-car-memE*[*of R*]
unfolding *ring-functions-def*
by (*smt A cartesian-power-car-memE funcset-carrier ring-pow-function-ring-car-memE*(*2*))

next
case *False*
then have *F0*: $(partial\text{-}eval\ \ R\ k\ n\ f\ a)\ \ x = f\ (insert\text{-}at\text{-}index\ x\ a\ n)$
unfolding *partial-eval-def*
using *assms*
by $(meson\ A\ restrict\text{-}apply')$
have *F1*: $(insert\text{-}at\text{-}index\ x\ a\ n) \in carrier\ (R^{Suc\ k})$
using *A assms insert-at-index-closed*[*of x k R a n*] *cartesian-power-car-memE*

by *blast*
show $(partial\text{-}eval\ \ R\ k\ n\ f\ a)\ x \in carrier\ R$
unfolding *F0* **apply**(*rule function-ring-car-mem-closed*[*of f carrier* $(R^{Suc\ k})$])

apply (*simp add*: *assms*(*1*))
by(*rule F1*)
qed
qed
show $\bigwedge x.\ x \notin carrier\ (R^k) \Longrightarrow (partial\text{-}eval\ \ R\ k\ n\ f\ a)\ \ x = undefined$
proof−
fix x
assume $x \notin carrier\ (R^k)$
show $(partial\text{-}eval\ \ R\ k\ n\ f\ a)\ x = undefined$
unfolding *partial-eval-def*
by $(meson\ \langle x \notin carrier\ (R^k)\rangle\ restrict\text{-}apply)$
qed
show *ring-functions R*
unfolding *ring-functions-def*
by (*simp add*: *ring-axioms*)
qed

Pullbacks preserve ring power functions

lemma *fun-struct-maps*:
$struct\text{-}maps\ (R^n)\ R = carrier\ (Fun_n\ R)$
proof
show $struct\text{-}maps\ (R^n)\ R \subseteq carrier\ Fun_n\ R$

by (*smt function-ring-car-memI struct-maps-memE*(*1*) *struct-maps-memE*(*2*) *subsetI*)
show *carrier* (*Fun*$_n$ *R*) ⊆ *struct-maps* (*R*^{n}) *R*
using *struct-maps-memI ring-functions.function-ring-car-memE*
by (*smt function-ring-car-mem-closed ring-axioms ring-functions.function-ring-not-car ring-functions.intro subsetI*)
qed

lemma *pullback-fun-closed*:
assumes *f* ∈ *struct-maps* (*R*^{n}) (*R*^{m})
assumes *g* ∈ *carrier* (*Fun*$_m$ *R*)
shows *pullback* (*R*^{n}) *f g* ∈ *carrier* (*Fun*$_n$ *R*)
using *assms*(*1*) *assms*(*2*) *fun-struct-maps pullback-closed* **by** *blast*

end

Includes $R^{|S|}$ into R^n by pulling back along the projection $R^n \mapsto R^{|S|}$ at indices S

context *ring*
begin

definition(**in** *ring*) *ring-pow-inc* :: (*nat set*) ⇒ *arity* ⇒ ('*a tuple* ⇒ '*a*) => ('*a tuple* ⇒ '*a*) **where**
ring-pow-inc S n f = *pullback* (*R*^{n}) ($\pi_{n,S}$) *f*

lemma *ring-pow-inc-is-fun*:
assumes *S* ⊆ {..<*n*}
assumes *f* ∈ *carrier* (*Fun*$_{card\ S}$ *R*)
shows *ring-pow-inc S n f* ∈ *carrier* (*Fun*$_n$ *R*)
by (*metis assms*(*1*) *assms*(*2*) *ring-pow-proj-is-map pullback-fun-closed ring-pow-inc-def*)

The "standard" inclusion of powers of function rings into one another

abbreviation(*input*) *std-proj*:: *nat* ⇒ *nat* ⇒ ('*a list*) ⇒ ('*a list*) **where**
std-proj n m ≡ *ring-pow-proj n* ({..<*m*})

lemma *std-proj-id*:
assumes *m* ≤ *n*
assumes *as* ∈ *carrier* (*R*^{n})
assumes *i* < *m*
shows *std-proj n m as* ! *i* = *as* ! *i*
proof−
have {..<*m*} ⊆ *indices-of as*
using *assms cartesian-power-car-memE* **unfolding** *indices-of-def*
by *blast*
thus *?thesis*
unfolding *ring-pow-proj-def*
using *assms nth-elem-upto*[*of i m*]
project-at-indices-nth[*of* {..<*m*} *as i*]
by (*metis card-lessThan restrict-apply*)

qed

abbreviation(*input*) *std-inc*:: *nat* ⇒ *nat* ⇒ (*'a list* ⇒ *'a*) => (*'a list* ⇒ *'a*)
where
std-inc n m f ≡ *ring-pow-inc* ({..<*m*}) *n f*

lemma *std-proj-is-map*[*simp*]:
assumes $m \leq n$
shows *std-proj n m* ∈ *struct-maps* (R^n) (R^m)
by (*metis assms card-lessThan lessThan-subset-iff ring-pow-proj-is-map*)

end

4.6 Coordinate Functions

definition *var* :: (*'a*, *'b*) *ring-scheme* ⇒ *nat* ⇒ *nat* ⇒ (*'a list* ⇒ *'a*) **where**
var R n i = *restrict* (λ*x*. *x*!*i*) (*carrier* (R^n))

context *ring*
begin

lemma *var-in-car*:
assumes $i < n$
shows *var R n i* ∈ *carrier* (Fun_n *R*)
apply(*rule function-ring-car-memI*)
unfolding *var-def*
apply (*metis assms cartesian-power-car-memE' restrict-apply'*)
by (*meson restrict-apply*)

lemma *varE*[*simp*]:
assumes $i < n$
assumes $x \in$ *carrier* (R^n)
shows *var R n i x* = *x* ! *i*
unfolding *var-def*
using *assms*(*2*)
by (*meson restrict-apply'*)

lemma *std-inc-of-var*:
assumes $i < n$
assumes $n \leq m$
shows *std-inc m n* (*var R n i*) = (*var R m i*)
apply(*rule ext*)
proof−
fix *x*
show *std-inc m n* (*var R n i*) *x* = *var R m i x*
apply(*cases x* ∈ *carrier* (R^m))
proof−
show *x* ∈ *carrier* (R^m) ⟹ *std-inc m n* (*var R n i*) *x* = *var R m i x*

proof −
assume *A*: $x \in$ *carrier* (R^m)
have (*restrict* (*project-at-indices* ({..<*n*})) (*carrier* (R^m))) *x* = ((*project-at-indices* ({..<*n*})) *x*)
by (*meson A restrict-apply'*)
then have *B*: *std-inc m n* (*var R n i*) *x* = (*var R n i*) ((*project-at-indices* ({..<*n*})) *x*)
unfolding *ring-pow-inc-def ring-pow-proj-def pullback-def*
by (*metis A compose-eq*)
have *C*: *var R m i x* = *x* ! *i*
by (*metis A assms*(*1*) *assms*(*2*) *le-iff-add trans-less-add1 varE*)
show *std-inc m n* (*var R n i*) *x* = *var R m i x*
by (*metis A B C assms*(*1*) *assms*(*2*) *project-at-indices-ring-pow-proj std-proj-id std-proj-is-map struct-maps-memE*(*1*) *varE*)
qed
show *x* ∉ *carrier* $(R^m) \Longrightarrow$ *std-inc m n* (*var R n i*) *x* = *var R m i x*
by (*metis* (*mono-tags*, *lifting*) *assms*(*1*) *assms*(*2*) *card-lessThan lessThan-subset-iff less-SucI ring-axioms nat-induct-at-least ring.fun-struct-maps ring-pow-inc-is-fun struct-maps-memE*(*2*) *var-in-car*)
qed
qed

abbreviation *variable* (‹𝔳_›) **where**
variable n i ≡ *var R n i*

end

definition *var-set* :: ('*a*, '*b*) *ring-scheme* ⇒ *nat* ⇒ ('*a list* ⇒ '*a*) *set* **where**
var-set R n = *var R n* ‘ {..<*n*}

lemma *var-setE*:
assumes *f* ∈ *var-set R n*
obtains *i* **where** *f* = *var R n i* ∧ *i* ∈ {..<*n*}
by (*metis assms imageE that var-set-def*)

lemma *var-setI*:
assumes *i* ∈ {..<*n*}
assumes *f* = *var R n i*
shows *f* ∈ *var-set R n*
using *assms*(*1*) *assms*(*2*) *var-set-def*
by *blast*

context *ring*
begin

lemma *var-set-in-fun-ring-car*:
shows *var-set R n* ⊆ *carrier Fun*$_n$ *R*
proof
fix *x*

assume $x \in$ *var-set R n*
then obtain i **where** *i-def*: $i \in \{..<n\} \wedge x =$ *var R n i*
unfolding *var-set-def*
by *blast*
have $i < n$**using** *i-def*
using *atLeastLessThan-iff* **by** *blast*
then show $x \in$ *carrier* Fun_n *R*
using *i-def var-in-car* **by** *blast*
qed

end

4.7 Graphs of functions

definition *fun-graph*:: ('*a*, '*b*) *ring-scheme* ⇒ *nat* ⇒ ('*a list* ⇒ '*a*) ⇒ '*a list set*
where
fun-graph R n f = {*as*. (∃ *x* ∈ *carrier* (R^n). *as* = *x* @ [*f x*])}

context *ring*
begin

lemma *function-ring-car-memE*:
assumes *f* ∈ *carrier* (Fun_n *R*)
assumes *a* ∈ *carrier* (R^n)
shows *f a* ∈ *carrier R*
using *ring-functions.function-ring-car-memE*(*1*)[*of R f*]
unfolding *ring-functions-def*
by (*meson assms*(*1*) *assms*(*2*) *ring-axioms function-ring-car-mem-closed ring-functions-def*)

lemma *graph-range*:
assumes *f* ∈ *carrier* (Fun_n *R*)
shows *fun-graph R n f* ⊆ *carrier* ($R^{Suc\ n}$)
proof
fix *x*
assume *x-def*: *x* ∈ *fun-graph R n f*
obtain *a* **where** *a-def*: *a* ∈ *carrier* (R^n) ∧ *x* = *a*@[*f a*]
using *x-def fun-graph-def*
by (*smt mem-Collect-eq*)
have *f-closed*: *f a* ∈ *carrier R*
using *assms function-ring-car-memE a-def*
by *blast*
show *x* ∈ *carrier* ($R^{Suc\ n}$)
proof(*rule cartesian-power-car-memI*)
show *length x* = *Suc n*
using *x-def a-def cartesian-power-car-memE*[*of a R n*]
by (*metis length-append-singleton*)
have *set x* = *insert* (*f a*) (*set a*)

using *a-def*
by (*metis Un-insert-right append-Nil2 list.simps*(15) *set-append*)
thus *set x* ⊆ *carrier R*
using *a-def*
by (*metis cartesian-power-car-memE''* *f-closed insert-subset*)
qed
qed

lemma *fun-graph-memE*:
assumes *f* ∈ *carrier* (*Fun*$_n$ *R*)
assumes *p* ∈ *fun-graph R n f*
shows (*take n p*) ∈ *carrier* (*R*^{n})
using *assms* **unfolding** *fun-graph-def*
by (*metis* (*no-types*, *lifting*) *assms*(2) *graph-range le-add2 plus-1-eq-Suc subsetD take-closed*)

lemma *fun-graph-memE'*:
assumes *f* ∈ *carrier* (*Fun*$_n$ *R*)
assumes *p* ∈ *fun-graph R n f*
shows *f* (*take n p*) = *p*!*n*
using *assms*
unfolding *fun-graph-def*
by (*smt Cons-nth-drop-Suc append-take-drop-id assms*(2) *butlast-snoc cartesian-power-car-memE*

drop-all graph-range last-snoc le-Suc-eq lessI mem-Collect-eq subsetD)

apply a function f to the tuple consisting of the first n indices, leaving the remaining indices unchanged

definition *partial-image* :: *arity* ⇒ (*'c list* ⇒ *'c*) ⇒ *'c list* ⇒ *'c list* **where**
partial-image n f as = (*f* (*take n as*)) # (*drop n as*)

lemma *partial-image-range*:
assumes *f* ∈ *carrier* (*Fun*$_n$ *R*)
assumes *m* ≥ *n*
assumes *as* ∈ *carrier* (*R*^{m})
shows *partial-image n f as* ∈ *carrier* (*R*$^{m - n + 1}$)
proof(*cases m* = *n*)
case *True*
then have *f* (*take n as*) = *f as*
by (*metis assms*(2) *assms*(3) *cartesian-power-car-memE take-all*)
then have *0*: *f* (*take n as*) ∈ *carrier R*
using *True assms*(1) *assms*(3) *function-ring-car-memE* **by** *presburger*
have *1*: (*drop n as*) = []
using *True assms*(3) *cartesian-power-car-memE drop-all* **by** *blast*
then show *?thesis*
unfolding *partial-image-def*
using *0 1*
by (*metis* (*no-types*, *lifting*) *One-nat-def assms*(3) *cartesian-power-car-memE*
cartesian-power-car-memI empty-iff insert-iff length-drop list.set(1)

list.set(2) list.size(4) subsetI)
next
case *False*
then have *0*: (*drop n as*) ∈ *carrier* ($R^{m - n}$)
using *assms drop-closed*[*of n m as R*] *le-neq-implies-less*
by *blast*
have *1*: *f* (*take n as*) ∈ *carrier R*
using *assms*(*1*) *assms*(*2*) *assms*(*3*) *function-ring-car-memE take-closed* **by** *blast*
show *?thesis*
apply(*rule cartesian-power-car-memI*)
apply (*metis 0 One-nat-def cartesian-power-car-memE list.size*(*4*) *partial-image-def*)
by (*smt 1 assms*(*3*) *cartesian-power-car-memE cartesian-power-car-memE'* *in-set-conv-nth*
partial-image-def set-ConsD set-drop-subset subsetD subsetI)
qed

end

5 Coordinate Rings on Cartesian Powers

5.1 Basic Facts and Definitions

locale *cring-coord-rings* = *UP-cring* +
assumes *one-neq-zero*: **1** ≠ **0**

coordinate polynomial ring in n variables over a commutative ring

definition *coord-ring* :: ('*a*, '*b*) *ring-scheme* ⇒ *nat* ⇒ ('*a*, ('*a*, *nat*) *mvar-poly*) *module*
(‹- [$\mathcal{X}$_]› *80*) **where** $R[\mathcal{X}_n]$ ≡ *Pring R* {..< *n*::*nat*}

sublocale *cring-coord-rings* < *cring-functions R carrier* (R^n) Fun_n *R*
unfolding *cring-functions-def ring-functions-def*
apply (*simp add*: *R.ring-axioms R-cring*)
by *simp*

sublocale *cring-coord-rings* < *MP?*: *cring* $R[\mathcal{X}_n]$
by (*simp add*: *R.Pring-is-cring R-cring coord-ring-def*)

sublocale *cring-coord-rings* < *F?*: *cring* Fun_n *R*
by (*simp add*: *function-ring-is-cring*)

context *cring-coord-rings*
begin

lemma *coord-cring-cring*:
cring ($R[\mathcal{X}_n]$) **unfolding** *coord-ring-def*
by (*simp add*: *R.Pring-is-cring R-cring*)

coordinate constant functions

abbreviation(*input*) *coord-const* :: *'a* ⇒ (*'a*, *nat*) *mvar-poly* **where**
coord-const k ≡ *ring.indexed-const R k*

lemma *coord-const-ring-hom*:
ring-hom-ring R (*R*[$\mathcal{X}_n$]) *coord-const*
unfolding *coord-ring-def*
apply(*rule ring-hom-ringI*)
apply (*simp add*: *R.ring-axioms*)
apply (*simp add*: *R.Pring-is-ring*)
apply (*simp add*: *R.indexed-const-closed*)
apply (*simp add*: *R.indexed-const-mult*)
apply (*simp add*: *R.Pring-add R.indexed-padd-const*)
by (*simp add*: *R.Pring-one*)

coordinate functions

lemma *pvar-closed*:
assumes *i* < *n*
shows *pvar R i* ∈ *carrier* (*R*[$\mathcal{X}_n$])
unfolding *var-to-IP-def*
proof−
have *set-mset* {#*i*#} ⊆ {..<*n*}
using *assms*
by *simp*
then show *mset-to-IP R* {#*i*#} ∈ *carrier* (*R*[$\mathcal{X}_n$])
by (*simp add*: *R.ring-axioms coord-ring-def R.Pring-car ring.mset-to-IP-closed*)
qed

relationship between multiplciation by a variable and index multiplcation

lemma *pvar-indexed-pmult*:
assumes *p* ∈ *carrier* (*R*[$\mathcal{X}_n$])
shows (*p* ⨂ *i*) = *p* ⊗$_{R[\mathcal{X}_n]}$ *pvar R i*
proof−
have *p* ∈ *Pring-set R* {..<(*n*::*nat*)}
using *R.Pring-car assms*
by (*metis coord-ring-def*)
then have *p* ∈ *Pring-set R* (*UNIV*::*nat set*)
using *R.Pring-set-restrict*
by *blast*
then show *?thesis*
using *assms R.poly-index-mult*[*of p UNIV i*] **unfolding** *var-to-IP-def*
by (*metis R.Pring-mult UNIV-I coord-ring-def*)
qed

lemma *coord-ring-cfs-closed*:
assumes *p* ∈ *carrier* (*R*[$\mathcal{X}_n$])
shows *p m* ∈ *carrier R*
using *assms* **unfolding** *coord-ring-def*

using *R.Pring-carrier-coeff'* **by** *blast*

lemma *coord-ring-plus*:
assumes $p \in$ *carrier* $(R[\mathcal{X}_n])$
assumes $Q \in$ *carrier* $(R[\mathcal{X}_n])$
shows $(p \oplus_{R[\mathcal{X}_n]} Q)\ m = p\ m \oplus Q\ m$
using *assms* **unfolding** *coord-ring-def*
by (*metis R.Pring-add R.indexed-padd-def*)

lemma *coord-ring-uminus*:
assumes $p \in$ *carrier* $(R[\mathcal{X}_n])$
shows $(\ominus_{R[\mathcal{X}_n]} p)\ m = \ominus\ (p\ m)$
using *assms* **unfolding** *coord-ring-def*
using *MP.add.inv-closed MP.minus-minus coord-ring-cfs-closed coord-ring-def coord-ring-plus is-abelian-group R.is-cring R.ring-axioms*
by (*metis P-ring-uminus-def R.Pring-a-inv assms*)

lemma *coord-ring-minus*:
assumes $p \in$ *carrier* $(R[\mathcal{X}_n])$
assumes $Q \in$ *carrier* $(R[\mathcal{X}_n])$
shows $(p \ominus_{R[\mathcal{X}_n]} Q)\ m = p\ m \ominus Q\ m$
using *assms R.Pring-add*[*of - p Q*] *coord-ring-cfs-closed*
unfolding *indexed-padd-def coord-ring-def a-minus-def*
by (*metis* (*no-types*, *lifting*) *MP.add.inv-closed coord-ring-def coord-ring-plus cring-coord-rings.coord-ring-uminus cring-coord-rings-axioms*)

lemma *coord-ring-one*:
$\mathbf{1}_{R[\mathcal{X}_n]}\ m = (\mathit{if}\ m = \{\#\}\ \mathit{then}\ \mathbf{1}\ \mathit{else}\ \mathbf{0})$
by (*metis R.Pring-one coord-ring-def R.indexed-const-def*)

lemma *coord-ring-zero*:
$\mathbf{0}_{R[\mathcal{X}_n]}\ m = \mathbf{0}$
by (*metis MP.minus-zero MP.r-zero MP.zero-closed R-cring coord-ring-cfs-closed coord-ring-plus coord-ring-uminus cring.cring-simprules*(*17*))

Evaluation of a polynomial at a point

end

abbreviation(*input*) *point-to-eval-map* **where**
point-to-eval-map R as $\equiv (\lambda i.\ (\mathit{if}\ i < \mathit{length\ as\ then\ as}\ !\ i\ \mathit{else}\ \mathbf{0}_R))$

definition *eval-at-point* :: ('*a*, '*b*) *ring-scheme* ⇒ '*a list* ⇒ ('*a*, *nat*) *mvar-poly* ⇒ '*a* **where**
eval-at-point R as p ≡ *total-eval R* $(\lambda i.\ (\mathit{if}\ i < \mathit{length\ as\ then\ as}\ !\ i\ \mathit{else}\ \mathbf{0}_R))\ p$

lemma(**in** *cring-coord-rings*) *eval-at-point-factored*:

eval-at-point R as p = total-eval R (point-to-eval-map R as) p
using *eval-at-point-def* **by** *blast*

5.2 Total Evaluation of a Polynomial

abbreviation(*input*) *eval-at-poly* **where**
eval-at-poly R p as ≡ eval-at-point R as p

evaluation of coordinate polynomials

context *cring-coord-rings*
begin

lemma *eval-at-point-closed*:
assumes $a \in$ *carrier* (R^n)
assumes $p \in$ *carrier* $(R[\mathcal{X}_n])$
shows *eval-at-point R a p ∈ carrier R*
proof−
have *0*: *R.indexed-pset* ({..<*n*}− *UNIV*) (*carrier R*) ⊆ *carrier* $(R[\mathcal{X}_n])$
unfolding *coord-ring-def*
by (*simp add*: *R.Pring-car R.Pring-carrier-subset*)
have *1* : *poly-eval R UNIV* (λ*i*. *if i < length a then a* ! *i else* **0**) *p* ∈ *R.indexed-pset* ({..<*n*}− *UNIV*) (*carrier R*)
by (*smt R.Pring-car R.closed-funI R.poly-eval-closed R.zero-closed assms*(*1*) *assms*(*2*) *cartesian-power-car-memE cartesian-power-car-memE′ coord-ring-def*)
hence *2*: *poly-eval R UNIV* (λ*i*. *if i < length a then a* ! *i else* **0**) *p* ∈*carrier* $(R[\mathcal{X}_n])$
using *0* **by** *blast*
show *?thesis*
unfolding *eval-at-point-def total-eval-def eval-in-ring-def*
using *1 R.Pring-car R.Pring-cfs-closed cartesian-power-car-memE cartesian-power-car-memE′ R.closed-funI coord-ring-def R.zero-closed*
by *blast*
qed

lemma *eval-pvar*:
assumes *i* < (*n*::*nat*)
assumes $a \in$ *carrier* (R^n)
shows *eval-at-point R a* (*pvar R i*) = *a*!*i*
unfolding *eval-at-point-def*
proof−
have *pvar R i* = $\mathbf{1}_{R[\mathcal{X}_n]} \bigotimes i$
unfolding *var-to-IP-def*
by (*metis R.Pring-one coord-ring-def R.monom-add-mset R.one-mset-to-IP*)
then show *total-eval R* (λ*i*. *if i < length a then a* ! *i else* **0**) (*pvar R i*) = *a* ! *i*
using *assms R.total-eval-var*[*of* (λ*i*. (*if i< length a then a* ! *i else* $\mathbf{0}_R$)) *i*]
by (*smt cartesian-power-car-memE cartesian-power-car-memE′ R.closed-funI var-to-IP-def R.zero-closed*)
qed

lemma *eval-at-point-const*:
assumes $k \in$ *carrier* R
assumes $a \in$ *carrier* (R^n)
shows *eval-at-point* R a $(R.$*indexed-const* $k) = k$
unfolding *eval-at-point-def*
using *assms*(*1*) *R.total-eval-const* **by** *blast*

lemma *eval-at-point-add*:
assumes $a \in$ *carrier* (R^n)
assumes $A \in$ *carrier* $(R[\mathcal{X}_n])$
assumes $B \in$ *carrier* (*coord-ring* R n)
shows *eval-at-point* R a $(A \oplus_{\textit{coord-ring } R \ n} B) =$
eval-at-point R a $A \oplus_R$ *eval-at-point* R a B
unfolding *eval-at-point-def*
using *R.total-eval-add*[*of* A $\{..<n\}$ B]
by (*smt assms*(*1*) *assms*(*2*) *assms*(*3*) *cartesian-power-car-memE cartesian-power-car-memE′ R.closed-funI coord-ring-def R.zero-closed*)

lemma *eval-at-point-mult*:
assumes $a \in$ *carrier* (R^n)
assumes $A \in$ *carrier* $(R[\mathcal{X}_n])$
assumes $B \in$ *carrier* $((R[\mathcal{X}_n]))$
shows *eval-at-point* R a $(A \otimes_{R[\mathcal{X}_n]} B) =$
eval-at-point R a $A \otimes_R$ *eval-at-point* R a B
unfolding *eval-at-point-def*
using *R.total-eval-mult*[*of* A $\{..<n\}$ B]
by (*smt assms*(*1*) *assms*(*2*) *assms*(*3*) *cartesian-power-car-memE cartesian-power-car-memE′ R.closed-funI coord-ring-def R.zero-closed*)

lemma *eval-at-point-indexed-pmult*:
assumes $a \in$ *carrier* (R^n)
assumes $A \in$ *carrier* $(R[\mathcal{X}_n])$
assumes $i < n$
shows *eval-at-point* R a $(A \bigotimes i) =$
eval-at-point R a $A \otimes_R$ $(a!i)$
proof−
have $(A \bigotimes i) = A \otimes_{R[\mathcal{X}_n]}$ $(pvar\ R\ i)$
using *assms*(*2*) *pvar-indexed-pmult* **by** *blast*
then show *?thesis*
using *assms eval-at-point-mult eval-pvar pvar-closed*
by *presburger*
qed

lemma *eval-at-point-ring-hom*:
assumes $a \in$ *carrier* (R^n)
shows *ring-hom-ring* (*coord-ring* R I) R (*eval-at-point* R a)
unfolding *eval-at-point-def*
using *R.total-eval-ring-hom*
by (*smt assms cartesian-power-car-memE cartesian-power-car-memE′ R.closed-funI*

coord-ring-def R.zero-closed)

lemma *eval-at-point-scalar-mult*:
assumes $a \in carrier\ (R^n)$
assumes $A \in carrier\ (R[\mathcal{X}_n])$
assumes $k \in carrier\ R$
shows *eval-at-point R a* (*poly-scalar-mult R k A*) = $k \otimes_R$ (*eval-at-point R a A*)
using *assms* **unfolding** *eval-at-point-def total-eval-def eval-in-ring-def*
using *R.poly-eval-scalar-mult*[*of k* (λi. *if i* < *length a then a* ! *i else* **0**) *A* {..<*n*} *UNIV*]
poly-scalar-mult-def
by (*smt R.Pring-car cartesian-power-car-memE cartesian-power-car-memE' R.closed-funI coord-ring-def R.zero-closed*)

lemma *eval-at-point-smult*:
assumes $a \in carrier\ (R^n)$
assumes $A \in carrier\ (R[\mathcal{X}_n])$
assumes $k \in carrier\ R$
shows *eval-at-point R a* ($k \odot_{R[\mathcal{X}_n]} A$) = $k \otimes_R$ (*eval-at-point R a A*)
by (*metis R.Pring-smult assms*(*1*) *assms*(*2*) *assms*(*3*) *coord-ring-def eval-at-point-scalar-mult*)

lemma *eval-at-point-subtract*:
assumes $a \in carrier\ (R^n)$
assumes $A \in carrier\ (R[\mathcal{X}_n])$
assumes $B \in carrier\ (coord\text{-}ring\ R\ \ n)$
shows *eval-at-point R a* ($A \ominus_{coord\text{-}ring\ R\ \ n} B$) =
eval-at-point R a A $\ominus_R$ *eval-at-point R a B*
using *assms eval-at-point-add*[*of a n A* $\ominus_{coord\text{-}ring\ R\ \ n}$ *B*]
abelian-group.a-inv-closed[*of* $R[\mathcal{X}_n]$ *B*]
unfolding *a-minus-def*
abelian-group.a-inv-closed abelian-group.minus-minus abelian-group.r-neg1 abelian-groupE(*1*) *abelian-groupE*(*4*) *coord-cring-cring cring-def eval-at-point-add eval-at-point-closed is-abelian-group ring-def*
by (*smt MP.add.inv-closed MP.l-neg MP.r-zero MP.zero-closed R.add.inv-closed R.add.m-assoc R.l-neg R.r-zero R.zero-closed eval-at-point-add eval-at-point-closed*)

lemma *eval-at-point-a-inv*:
assumes $a \in carrier\ (R^n)$
assumes $B \in carrier\ (coord\text{-}ring\ R\ \ n)$
shows *eval-at-point R a* ($\ominus_{R[\mathcal{X}_n]} B$) = $\ominus_R$ *eval-at-point R a B*
using *assms eval-at-point-subtract*[*of a n* $\mathbf{0}_{R[\mathcal{X}_n]}$ *B*]
by (*smt MP.add.inv-eq-1-iff MP.l-zero MP.minus-add MP.zero-closed R.is-abelian-group R.r-neg R.r-neg2 a-minus-def abelian-group.a-inv-closed abelian-groupE*(*4*) *eval-at-point-add eval-at-point-closed*)

lemma *eval-at-point-nat-pow*:
assumes $a \in carrier\ (R^n)$
assumes $A \in carrier\ (R[\mathcal{X}_n])$
shows *eval-at-point R a* ($A[\widehat{}]_{R[\mathcal{X}_n]}(k::nat)$) = (*eval-at-point R a A*)[$\widehat{}$]*k*

apply(*induction k*)
apply (*metis Group.nat-pow-0 R.Pring-one assms*(*1*) *coord-ring-def eval-at-point-const R.one-closed*)
proof− **fix** *k*::*nat* **assume** *IH*: *eval-at-poly R* (*A* [^]$_{R[\mathcal{X}_n]}$ *k*) *a* = *eval-at-poly R A a* [^] *k*
have *A* [^]$_{R[\mathcal{X}_n]}$ *Suc k* = *A* [^]$_{R[\mathcal{X}_n]}$ *k* ⊗$_{(R[\mathcal{X}_n])}$ *A*
using *MP.nat-pow-Suc* **by** *blast*
then have *eval-at-poly R* (*A* [^]$_{R[\mathcal{X}_n]}$ *Suc k*) *a* =
eval-at-poly R (*A* [^]$_{R[\mathcal{X}_n]}$ *k*) *a* ⊗ *eval-at-poly R A a*
using *monoid.nat-pow-closed*[*of* (*R*[$\mathcal{X}_n$]) *A k*] *eval-at-point-mult*[*of a n A* [^]$_{R[\mathcal{X}_n]}$ *k A*] *assms*
by (*metis R.Pring-is-monoid coord-ring-def*)
then show *eval-at-poly R* (*A* [^]$_{R[\mathcal{X}_n]}$ *Suc k*) *a* = *eval-at-poly R A a* [^] *Suc k*
using *IH R.nat-pow-Suc*
by *auto*
qed

end

5.3 Partial Evaluation of a Polynomial

definition *coord-partial-eval* ::
('*a*, '*b*) *ring-scheme* ⇒ *nat set* ⇒ '*a list* ⇒ ('*a*, *nat*) *mvar-poly* ⇒ ('*a*, *nat*) *mvar-poly* **where**
coord-partial-eval R S as = *poly-eval R S* (*point-to-eval-map R as*)

context *cring-coord-rings*
begin

lemma *point-to-eval-map-closed*:
assumes *as* ∈ *carrier* (*R*^{n})
shows *closed-fun R* (*point-to-eval-map R as*)
using *assms*
by (*smt cartesian-power-car-memE cartesian-power-car-memE' R.closed-funI R.zero-closed*)

lemma *coord-partial-eval-hom*:
assumes *as* ∈ *carrier* (*R*^{n})
shows *coord-partial-eval R S as* ∈ *ring-hom* (*R*[$\mathcal{X}_n$]) (*R*[$\mathcal{X}_n$])
unfolding *coord-partial-eval-def*
using *point-to-eval-map-closed*[*of as n*] *assms*
R.poly-eval-ring-hom[*of* {..<*n*} {..<*n*} *point-to-eval-map R as S*]
by (*metis* (*mono-tags*, *lifting*) *Diff-subset coord-ring-def order-refl ring-hom-ring.homh*)

lemma *coord-partial-eval-hom'*:
assumes *as* ∈ *carrier* (*R*^{n})
shows *coord-partial-eval R S as* ∈ *ring-hom* (*R*[$\mathcal{X}_n$]) (*Pring R* ({..<*n*} − *S*))

unfolding *coord-partial-eval-def*
using *point-to-eval-map-closed*[*of as n*] *assms*
R.poly-eval-ring-hom[*of* {..<*n*} − *S* {..<*n*} *point-to-eval-map R as S*]
by (*metis* (*no-types*, *lifting*) *Diff-subset coord-ring-def order-refl ring-hom-ring.homh*)

lemma *coord-partial-eval-closed*:
assumes $S \subseteq \{..<n\}$
assumes $\{..<n\} - S \subseteq I$
assumes *as* ∈ *carrier* (R^n)
assumes $p \in$ *carrier* $(R[\mathcal{X}_n])$
shows *coord-partial-eval R S as p* ∈ *carrier* (*Pring R I*)
unfolding *coord-partial-eval-def*
using *R.poly-eval-closed*[*of point-to-eval-map R as p* {..<*n*} *S*] *R.Pring-car*[*of I*] *R.Pring-carrier-subset*
by (*smt R.Pring-car assms*(*2*) *assms*(*3*) *assms*(*4*) *cartesian-power-car-memE cartesian-power-car-memE′ R.closed-funI coord-ring-def subsetD R.zero-closed*)

lemma *coord-partial-eval-add*:
assumes *as* ∈ *carrier* (R^n)
assumes $p \in$ *carrier* $(R[\mathcal{X}_n])$
assumes $Q \in$ *carrier* $(R[\mathcal{X}_n])$
shows *coord-partial-eval R S as* $(p \oplus_{(R[\mathcal{X}_n])} Q) =$
(*coord-partial-eval R S as p*) $\oplus_{(R[\mathcal{X}_n])}$ (*coord-partial-eval R S as Q*)
using *assms R.poly-eval-add*[*of p* {..<*n*} *Q* (*point-to-eval-map R as*) *S*] *Pring-def*[*of R* {..<*n*}]
point-to-eval-map-closed[*of as n*]
unfolding *coord-partial-eval-def*
by (*metis R.Pring-add R.Pring-car coord-ring-def*)

lemma *coord-partial-eval-mult*:
assumes *as* ∈ *carrier* (R^n)
assumes $p \in$ *carrier* $(R[\mathcal{X}_n])$
assumes $Q \in$ *carrier* $(R[\mathcal{X}_n])$
shows *coord-partial-eval R S as* $(p \otimes_{(R[\mathcal{X}_n])} Q) =$
(*coord-partial-eval R S as p*) $\otimes_{(R[\mathcal{X}_n])}$ (*coord-partial-eval R S as Q*)
using *assms R.poly-eval-mult*[*of p* {..<*n*} *Q* (*point-to-eval-map R as*) *S*] *Pring-def*[*of R* {..<*n*}]
point-to-eval-map-closed[*of as n*]
unfolding *coord-partial-eval-def*
by (*metis R.Pring-car R.Pring-mult coord-ring-def*)

lemma *coord-partial-eval-pvar*:
assumes $\mathbf{1} \neq \mathbf{0}$
assumes *as* ∈ *carrier* (R^n)
assumes $i \in S \cap \{..<n\}$
shows *coord-partial-eval R S as* (*pvar R i*) = *coord-const* (*as*!*i*)
proof−
have *0*: $i \in S$ **using** *assms*

by *blast*
have $i < length\ as$
by (*metis IntD2 assms*(*2*) *assms*(*3*) *cartesian-power-car-memE lessThan-iff*)
then have (*point-to-eval-map R as i*) = *as*!*i*
by *presburger*
then show *?thesis*
unfolding *coord-partial-eval-def var-to-IP-def*
using *0 assms point-to-eval-map-closed*[*of as n*]
R.poly-eval-index[*of point-to-eval-map R as S i*]
by *presburger*
qed

lemma *coord-partial-eval-pvar'*:
assumes $\mathbf{1} \neq \mathbf{0}$
assumes $as \in carrier\ (R^n)$
assumes $i \notin S$
shows *coord-partial-eval R S as* (*pvar R i*) = (*pvar R i*)
unfolding *coord-partial-eval-def*
using *R.poly-eval-index*[*of point-to-eval-map R as S i*]
by (*smt assms*(*1*) *assms*(*2*) *assms*(*3*) *cartesian-power-car-memE cartesian-power-car-memE' R.closed-funI var-to-IP-def R.zero-closed*)

5.4 An induction rule for coordinate rings

lemma *coord-ring-induct*:
assumes $A \in carrier\ (R[\mathcal{X}_n])$
assumes $\bigwedge a.\ a \in carrier\ R \Longrightarrow p\ (coord\text{-}const\ a)$
assumes $\bigwedge i\ Q.\ Q \in carrier\ (R[\mathcal{X}_n]) \Longrightarrow p\ Q \Longrightarrow\ i < n \Longrightarrow p\ (Q \otimes_{R[\mathcal{X}_n]} pvar\ R\ i)$
assumes $\bigwedge Q0\ Q1.\ Q0 \in carrier\ (R[\mathcal{X}_n]) \Longrightarrow Q1 \in carrier\ (R[\mathcal{X}_n]) \Longrightarrow p\ Q0 \Longrightarrow p\ Q1 \Longrightarrow p\ (Q0 \oplus_{R[\mathcal{X}_n]}\ Q1)$
shows *p A*
apply(*rule R.indexed-pset.induct*[*of A* {..<*n*} *carrier R*])
using *R.Pring-car assms*(*1*)
apply (*metis coord-ring-def*)
using *assms*(*2*) **apply** *blast*
apply (*metis* (*full-types*) *R.Pring-add R.Pring-car assms*(*4*) *coord-ring-def*)
proof−
fix *a i*
assume *a* ∈ *Pring-set R* {..<*n*}
then have *0*: *a* ∈ *carrier* (*R*[$\mathcal{X}_n$])
using *R.Pring-car*
by (*simp add*: ‹⋀*I*. *carrier* (*Pring R I*) = *Pring-set R I*› ‹*a* ∈ *Pring-set R* {..<*n*}› *coord-ring-def*)
assume *1*: *p a*
assume *i* ∈ {..< *n*}
then have *2*: *i* < *n*
using *assms*
by *blast*

have $p\ (a \otimes_{R[\mathcal{X}_n]} pvar\ R\ i)$
using *0 1 2 assms*(*3*) **by** *blast*
then show $p\ (a \bigotimes i)$
using *0 pvar-indexed-pmult*
by *presburger*
qed

end

5.5 Algebraic Sets in Cartesian Powers

5.5.1 The Zero Set of a Single Polynomial

definition *zero-set* :: ('*a*, '*b*) *ring-scheme* ⇒ *nat* ⇒ ('*a*, *nat*) *mvar-poly* ⇒ '*a list set*
(‹*V*1›) **where**
zero-set R n p = $\{a \in carrier\ (R^n).\ eval\text{-}at\text{-}point\ R\ a\ p = \mathbf{0}_R\}$

context *cring-coord-rings*
begin

lemma *zero-setI*:
assumes $a \in carrier\ (R^n)$
assumes *eval-at-point R a p* = $\mathbf{0}_R$
shows $a \in$ *zero-set R n p*
using *assms*
by (*metis* (*mono-tags*, *lifting*) *mem-Collect-eq zero-set-def*)

lemma *zero-setE*:
assumes $a \in$ *zero-set R n p*
shows $a \in carrier\ (R^n)$
eval-at-point R a p = $\mathbf{0}_R$
using *assms zero-set-def*
apply *blast*
by (*metis* (*mono-tags*, *lifting*) *assms mem-Collect-eq zero-set-def*)

lemma *zero-set-closed*:
zero-set R n p ⊆ $carrier\ (R^n)$
unfolding *zero-set-def*
by *blast*

end

5.5.2 The Zero Set of a Collection of Polynomials

definition *affine-alg-set* :: ('*a*, '*b*) *ring-scheme* ⇒ *nat* ⇒ ('*a*, *nat*) *mvar-poly set* ⇒ '*a list set*
where *affine-alg-set R n as* = $\{a \in carrier\ (R^n).\ \forall\ b \in as.\ a \in$ (*zero-set R n b*)$\}$

context *cring-coord-rings*
begin

lemma *affine-alg-set-empty*:
affine-alg-set R n {} = *carrier* (R^n)
unfolding *affine-alg-set-def* **by** *blast*

lemma *affine-alg-set-subset-zero-set*:
assumes *b* ∈ *as*
shows *affine-alg-set R n as* ⊆ (*zero-set R n b*)
using *assms affine-alg-set-def*
by *blast*

lemma(**in** *cring-coord-rings*) *affine-alg-set-memE*:
assumes *b* ∈ *as*
assumes *a* ∈ *affine-alg-set R n as*
shows *eval-at-poly R b a* = **0**
using *affine-alg-set-subset-zero-set zero-set-def assms*(*1*) *assms*(*2*)
by *blast*

lemma *affine-alg-set-subset*:
assumes *as* ⊆ *bs*
shows *affine-alg-set R n bs* ⊆ *affine-alg-set R n as*
using *assms affine-alg-set-def*
by *blast*

lemma *affine-alg-set-empty-set*:
assumes *as* = {}
shows *affine-alg-set R n as* = *carrier* (R^n)
unfolding *affine-alg-set-def*
using *assms* **by** *blast*

lemma *affine-alg-set-closed*:
shows *affine-alg-set R n as* ⊆ *carrier* (R^n)
unfolding *affine-alg-set-def*
by *blast*

lemma *affine-alg-set-singleton*:
affine-alg-set R n {*a*} = *zero-set R n a*
unfolding *affine-alg-set-def* **using** *zero-set-closed*
by *blast*

lemma *affine-alg-set-insert*:
affine-alg-set R n (*insert a A*) = *zero-set R n a* ∩ (*affine-alg-set R n A*)
proof
show *affine-alg-set R n* (*insert a A*) ⊆ V_R *n a* ∩ *affine-alg-set R n A*
using *affine-alg-set-subset*
by (*metis Int-greatest affine-alg-set-subset-zero-set insertI1 subset-insertI*)
show V_R *n a* ∩ *affine-alg-set R n A* ⊆ *affine-alg-set R n* (*insert a A*)

unfolding *affine-alg-set-def*
by *blast*
qed

lemma *affine-alg-set-intersect*:
affine-alg-set R n $(A \cup B)$ = (*affine-alg-set R n A*) $\cap$ (*affine-alg-set R n B*)
unfolding *affine-alg-set-def* **by** *blast*

lemma *affine-alg-set-memI*:
assumes $a \in$ *carrier* (R^n)
assumes $\bigwedge p.\ p \in B \Longrightarrow$ *eval-at-point R a p* = **0**
shows $a \in$ (*affine-alg-set R n B*)
unfolding *affine-alg-set-def zero-set-def*
using *assms*
by *blast*

lemma *affine-alg-set-not-memE*:
assumes $a \in$ *carrier* (R^n)
assumes $a \notin$ (*affine-alg-set R n B*)
shows $\exists\, b \in B.$ *eval-at-poly R b a* $\neq$ **0**
using *assms affine-alg-set-memI* **by** *blast*

5.5.3 Finite Unions and Intersections of Algebraic Sets are Algebraic

The product set of two sets in an arbitrary ring. That is, the set $\{xy \mid x \in A \wedge y \in B\}$ for two sets A, B.

definition(**in** *ring*) *prod-set* :: *'a set* $\Rightarrow$ *'a set* $\Rightarrow$ *'a set* **where**
prod-set as bs = $(\lambda x.$ *fst x* $\otimes$ *snd x*) ‘ (*as* $\times$ *bs*)

lemma(**in** *ring*) *prod-setI*:
assumes $c \in$ *prod-set as bs*
shows $\exists\, a \in as.\ \exists\, b \in bs.\ c = a \otimes b$
proof−
obtain *p* **where** *p-def*: $p \in (as \times bs) \wedge c =$ *fst p* $\otimes$ *snd p*
using *assms prod-set-def*[*of as bs*]
by (*metis* (*no-types*, *lifting*) *image-iff*)
then show *?thesis*
using *mem-Times-iff* **by** *blast*
qed

lemma(**in** *ring*) *prod-set-closed*:
assumes *as* $\subseteq$ *carrier R*
assumes *bs* $\subseteq$ *carrier R*
shows *prod-set as bs* $\subseteq$ *carrier R*
proof
fix *x*
assume $x \in$ *prod-set as bs*
then obtain *a b* **where** $a \in as \wedge b \in bs \wedge x = a \otimes b$

by (*meson ring-axioms ring.prod-setI*)
then have $a \in$ *carrier* $R \wedge b \in$ *carrier* $R \wedge x = a \otimes b$
using *assms*
by *blast*
then show $x \in$ *carrier* R
by *blast*
qed

The set of products of elements from two finite sets is again finite.

lemma(**in** *ring*) *prod-set-finite*:
assumes *finite as*
assumes *finite bs*
shows *finite* (*prod-set as bs*) *card* (*prod-set as bs*) $\leq$ *card as* $*$ *card bs*
proof−
have *finite* (*as* $\times$ *bs*)
using *assms*
by *blast*
then show *finite* (*prod-set as bs*)
using *prod-set-def*
by (*metis* (*no-types*, *lifting*) *finite-imageI*)
have *card* (*prod-set as bs*) $\leq$ *card* (*as* $\times$ *bs*)
using *assms*
unfolding *prod-set-def*
using ‹*finite* (*as* $\times$ *bs*)› *card-image-le* **by** *blast*
then show *card* (*prod-set as bs*) $\leq$ *card as* $*$ *card bs*
by (*simp add*: *card-cartesian-product*)
qed

definition *poly-prod-set* **where**
poly-prod-set n as bs = *ring.prod-set* ($R[\mathcal{X}_n]$) *as bs*

lemma *poly-prod-setE*:
assumes $c \in$ *poly-prod-set n as bs*
shows $\exists\, a \in as.\ \exists\, b \in bs.\ c = a \otimes_{R[\mathcal{X}_n]} b$
using *ring.prod-setI*[*of* $R[\mathcal{X}_n]$] *R.Pring-is-ring assms poly-prod-set-def coord-cring-cring cring.axioms*(*1*)
by *blast*

lemma *poly-prod-setI*:
assumes $a \in as$
assumes $b \in bs$
shows $a \otimes_{R[\mathcal{X}_n]} b \in$ *poly-prod-set n as bs*
proof−
have *0*: (*a*,*b*) $\in$ (*as* $\times$ *bs*)
using *assms* **by** *blast*
have *1*: ($\lambda x.$ *fst x* $\otimes_{R[\mathcal{X}_n]}$ *snd x*) (*a*, *b*) = $a \otimes_{R[\mathcal{X}_n]} b$
by (*metis fst-conv snd-conv*)
have *2*: ($\lambda x.$ *fst x* $\otimes_{R[\mathcal{X}_n]}$ *snd x*) (*a*, *b*) $\in$ (($\lambda x.$ *fst x* $\otimes_{R[\mathcal{X}_n]}$ *snd x*) ‘ (*as* $\times$ *bs*))
using *0* **by** *blast*

have *3*: *ring* $(R[\mathcal{X}_n])$
by (*simp add*: *R.Pring-is-ring coord-ring-def*)
then show *?thesis*
unfolding *poly-prod-set-def* **using** *0 1 2 3 ring.prod-set-def*[*of* $R[\mathcal{X}_n]$ *as bs*]
by *presburger*
qed

lemma *poly-prod-set-closed*:
assumes *as* $\subseteq$ *carrier* $(R[\mathcal{X}_n])$
assumes *bs* $\subseteq$ *carrier* $(R[\mathcal{X}_n])$
shows *poly-prod-set n as bs* $\subseteq$ *carrier* $(R[\mathcal{X}_n])$
using *ring.prod-set-closed*[*of* $R[\mathcal{X}_n]$] *R.Pring-is-ring assms*(*1*) *assms*(*2*) *poly-prod-set-def*

by (*simp add*: *coord-cring-cring cring.axioms*(*1*))

lemma *poly-prod-set-finite*:
assumes *finite as*
assumes *finite bs*
shows *finite* (*poly-prod-set n as bs*) *card* (*poly-prod-set n as bs*) $\leq$ *card as* $*$ *card bs*

using *ring.prod-set-finite*[*of* $R[\mathcal{X}_n]$]
apply (*simp add*: *R.Pring-is-ring assms*(*1*) *assms*(*2*) *poly-prod-set-def*)
using *ring.prod-set-finite*[*of* $R[\mathcal{X}_n]$]
apply (*simp add*: *assms*(*1*) *assms*(*2*) *coord-cring-cring cring.axioms*(*1*))
by (*simp add*: *assms*(*1*) *assms*(*2*) *coord-cring-cring cring.axioms*(*1*) *poly-prod-set-def ring.prod-set-finite*(*2*))

end

locale *domain-coord-rings* = *cring-coord-rings* + *domain*

lemma(**in** *domain-coord-rings*) *poly-prod-set-algebraic-set*:
assumes *as* $\subseteq$ *carrier* $(R[\mathcal{X}_n])$
assumes *bs* $\subseteq$ *carrier* $(R[\mathcal{X}_n])$
shows *affine-alg-set R n as* $\cup$ *affine-alg-set R n bs* = *affine-alg-set R n* (*poly-prod-set n as bs*)
proof
show *affine-alg-set R n as* $\cup$ *affine-alg-set R n bs* $\subseteq$ *affine-alg-set R n* (*poly-prod-set n as bs*)
proof fix *x*
assume *A*: *x* $\in$ *affine-alg-set R n as* $\cup$ *affine-alg-set R n bs*
show *x* $\in$ *affine-alg-set R n* (*poly-prod-set n as bs*)
proof(*rule affine-alg-set-memI*)
show *x* $\in$ *carrier* (R^n)
using *A affine-alg-set-closed*
by *blast*
show $\bigwedge p$. *p* $\in$ *poly-prod-set n as bs* $\Longrightarrow$ *eval-at-poly R p x* = **0**
proof− **fix** *p*

assume *B*: $p \in$ *poly-prod-set n as bs*
show *eval-at-poly R p x* $= \mathbf{0}$
proof−
obtain *p0 p1* **where** *C*: *p0* $\in$ *as* $\wedge$ *p1* $\in$ *bs* $\wedge$ $p = p0 \otimes_{R[\mathcal{X}_n]} p1$
using *B poly-prod-setE* **by** *blast*
then have *D*: *eval-at-poly R p x* = (*eval-at-poly R p0 x*) $\otimes$ (*eval-at-poly R p1 x*)
using ‹*x* $\in$ *carrier* (R^n)› *assms*(*1*) *assms*(*2*) *eval-at-point-mult*
by *blast*
show *?thesis* **proof**(*cases x* $\in$ *affine-alg-set R n as*)
case *True*
then have (*eval-at-poly R p0 x*) $= \mathbf{0}$
using *C affine-alg-set-memE* **by** *blast*
then show *?thesis*
by (*smt C D* ‹*x* $\in$ *carrier* (R^n)› *assms*(*2*) *eval-at-point-closed R.semiring-axioms semiring.l-null subsetD*)
next
case *False*
then have *x* $\in$ *affine-alg-set R n bs*
using *A*
by *blast*
then have (*eval-at-poly R p1 x*) $= \mathbf{0}$
using *C affine-alg-set-memE* **by** *blast*
then show *?thesis*
using *C A False*
by (*smt D* ‹*x* $\in$ *carrier* (R^n)› *assms*(*1*) *eval-at-point-closed R.r-null subsetD*)
qed
qed
qed
qed
qed
show *affine-alg-set R n* (*poly-prod-set n as bs*) $\subseteq$ *affine-alg-set R n as* $\cup$ *affine-alg-set R n bs*
proof fix *x*
assume *A*: *x* $\in$ *affine-alg-set R n* (*poly-prod-set n as bs*)
have *x-car*: *x* $\in$ *carrier* (R^n)
using *A affine-alg-set-closed*
by *blast*
show *x* $\in$ *affine-alg-set R n as* $\cup$ *affine-alg-set R n bs*
proof(*cases x* $\in$ *affine-alg-set R n as*)
case *True*
then show *?thesis* **by** *blast*
next
case *False*
have *x* $\in$ *affine-alg-set R n bs*
proof(*rule affine-alg-set-memI*)
show *x* $\in$ *carrier* (R^n)
using *A affine-alg-set-closed* **by** *blast*

```
        show ⋀p. p ∈ bs ⟹ eval-at-poly R p x = 0
        proof−
          fix p assume p-def: p ∈ bs
          obtain a where a-def: a ∈ as ∧ eval-at-poly R a x ≠ 0
            using False affine-alg-set-not-memE ‹x ∈ carrier (R^n)›
            by blast
          then have a ⊗_{R[𝒳_n]} p ∈ (poly-prod-set n as bs)
            using poly-prod-setI[of a as p bs] p-def
            by blast
          then have eval-at-poly R (a ⊗_{R[𝒳_n]} p) x = 0
            using A affine-alg-set-memE
            by blast

          then have eval-at-poly R a x ⊗ eval-at-poly R p  x = 0
            using eval-at-point-mult[of x n a p]
           by (metis (no-types, lifting) ‹x ∈ carrier (R^n)› a-def assms(1) assms(2)
p-def subsetD)
          then show eval-at-poly R p  x  = 0
            using a-def p-def
              by (meson assms(1) assms(2) eval-at-point-closed integral-iff subsetD
x-car)
        qed
      qed
      then show ?thesis
        by blast
    qed
  qed
qed

definition is-algebraic :: ('a, 'b) ring-scheme ⇒ nat ⇒ 'a list set ⇒ bool where
is-algebraic R n S = (∃ps. finite ps ∧ ps ⊆ carrier (R[𝒳_n]) ∧ S = affine-alg-set
R n ps)

context cring-coord-rings
begin

lemma is-algebraicE:
  assumes is-algebraic R n S
  obtains ps where  finite ps ps ⊆ carrier (R[𝒳_n]) S = affine-alg-set R n ps
  using assms
  by (meson is-algebraic-def)

lemma is-algebraicI:
  assumes finite ps
  assumes ps ⊆ carrier (R[𝒳_n])
  assumes S = affine-alg-set R n ps
  shows is-algebraic R n S
  using is-algebraic-def assms
  by blast
```

lemma *is-algebraicI'*:
assumes $p \in$ *carrier* $(R[\mathcal{X}_n])$
assumes $S =$ *zero-set* R n p
shows *is-algebraic* R n S
by (*metis affine-alg-set-singleton assms*(*1*) *assms*(*2*) *empty-subsetI finite.emptyI finite.intros*(*2*) *insert-subset is-algebraic-def*)

end

definition *alg-sets* :: *arity* ⇒ ('*a*, '*b*) *ring-scheme* ⇒ ('*a list set*) *set* **where**
alg-sets n R = {*S*. *is-algebraic R n S*}

context *cring-coord-rings*
begin

lemma *intersection-is-alg*:
assumes *is-algebraic R n A*
assumes *is-algebraic R n B*
shows *is-algebraic R n* (*A* ∩ *B*)
proof−
obtain *as* **where** *as-def*: *finite as* ∧ *as* ⊆ *carrier* $(R[\mathcal{X}_n])$ ∧ *A* = *affine-alg-set R n as*
by (*meson assms*(*1*) *is-algebraicE*)
obtain *bs* **where** *bs-def*: *finite bs* ∧ *bs* ⊆ *carrier* $(R[\mathcal{X}_n])$ ∧ *B* = *affine-alg-set R n bs*
by (*meson assms*(*2*) *is-algebraicE*)
show *?thesis* **apply**(*rule is-algebraicI*[*of as* ∪ *bs*])
using *as-def bs-def* **apply** *blast*
using *as-def bs-def* **apply** *blast*
by (*simp add*: *affine-alg-set-intersect as-def bs-def*)
qed

lemma(**in** *domain-coord-rings*) *union-is-alg*:
assumes *is-algebraic R n A*
assumes *is-algebraic R n B*
shows *is-algebraic R n* (*A* ∪ *B*)
proof−
obtain *as* **where** *as-def*: *finite as* ∧ *as* ⊆ *carrier* $(R[\mathcal{X}_n])$ ∧ *A* = *affine-alg-set R n as*
by (*meson assms*(*1*) *is-algebraicE*)
obtain *bs* **where** *bs-def*: *finite bs* ∧ *bs* ⊆ *carrier* $(R[\mathcal{X}_n])$ ∧ *B* = *affine-alg-set R n bs*
by (*meson assms*(*2*) *is-algebraicE*)
show *?thesis* **apply**(*rule is-algebraicI*[*of poly-prod-set n as bs*])
using *as-def bs-def*
apply (*simp add*: *poly-prod-set-finite*(*1*))
using *as-def bs-def poly-prod-set-closed* **apply** *blast*
using *as-def bs-def poly-prod-set-algebraic-set*

by *simp*
qed

lemma *zero-set-zero*:
zero-set R n $\mathbf{0}_{R[\mathcal{X}_n]}$ = *carrier* (R^n)
by (*metis R.add.r-cancel-one cring.cring-simprules*(2) *cring.cring-simprules*(8)
coord-cring-cring cring-coord-rings.eval-at-point-add cring-coord-rings.eval-at-point-closed

cring-coord-rings-axioms subsetI subset-antisym zero-setI zero-set-closed)

lemma *affine-alg-set-set*:
affine-alg-set R n {$\mathbf{0}_{R[\mathcal{X}_n]}$} = *carrier* ($R^n$)
using *affine-alg-set-singleton zero-set-zero*
by *blast*

lemma *car-is-alg*:
is-algebraic R n (*carrier* (R^n))
apply(*rule is-algebraicI*[*of* {$\mathbf{0}_{R[\mathcal{X}_n]}$}])
apply *blast*
using *R.Pring-zero-closed*
apply *blast*
using *affine-alg-set-set* **by** *blast*

lemma *zero-set-nonzero-constant*:
assumes $a \neq \mathbf{0}$
assumes $a \in$ *carrier R*
shows *zero-set R n* (*coord-const a*) = {}
proof(*rule ccontr*)
assume *V n* (*coord-const a*) ≠ {}
then obtain *x* **where** $x \in$ *V n* (*coord-const a*)
by *blast*
then show *False*
by (*metis assms*(1) *assms*(2) *cring-coord-rings.eval-at-point-const cring-coord-rings.zero-setE*(1)
cring-coord-rings.zero-setE(2) *cring-coord-rings-axioms*)
qed

lemma *zero-set-one*:
assumes $a \neq \mathbf{0}$
assumes $a \in$ *carrier R*
shows *zero-set R n* $\mathbf{1}_{R[\mathcal{X}_n]}$ = {}
using *zero-set-nonzero-constant*
by (*metis R.Pring-one coord-ring-def one-neq-zero R.one-closed*)

lemma *empty-set-as-affine-alg-set*:
affine-alg-set R n {$\mathbf{1}_{R[\mathcal{X}_n]}$} = {}
using *affine-alg-set-singleton local.one-neq-zero zero-set-one* **by** *blast*

lemma *empty-is-alg*:
is-algebraic R n {}

apply(*rule is-algebraicI'*[*of* $\mathbf{1}_{R[\mathcal{X}_n]}$])
apply *blast*
using *local.one-neq-zero zero-set-one* **by** *blast*

5.5.4 Finite Sets Are Algebraic

the function mapping a point in R^n to the unique linear polynomial vanishing exclusively at that point

definition *pvar-trans* :: *nat* ⇒ *nat* ⇒ *'a* ⇒ (*'a*, *nat*) *mvar-poly* **where**
pvar-trans n i a = (*pvar R i*) $\ominus_{R[\mathcal{X}_n]}$ *coord-const a*

lemma *pvar-trans-closed*:
assumes *a* ∈ *carrier R*
assumes *i* < *n*
shows *pvar-trans n i a* ∈ *carrier* (*R*[$\mathcal{X}_n$])
unfolding *pvar-trans-def* **using** *assms*
by (*metis MP.minus-closed coord-ring-def R.indexed-const-closed local.pvar-closed*)

lemma *pvar-trans-eval*:
assumes *a* ∈ *carrier R*
assumes *b* ∈ *carrier* (*R*^*n*)
assumes *i* < *n*
shows *eval-at-point R b* (*pvar-trans n i a*) = (*b*!*i*) ⊖ *a*
proof−
have *eval-at-point R b* (*pvar-trans n i a*) =
(*eval-at-point R b* (*pvar R i*)) ⊕ (*eval-at-point R b* ($\ominus_{R[\mathcal{X}_n]}$ (*coord-const a*)))
unfolding *pvar-trans-def a-minus-def* **using** *assms*
by (*metis MP.add.inv-closed coord-ring-def eval-at-point-add R.indexed-const-closed local.pvar-closed*)

then show *?thesis*
by (*metis a-minus-def assms*(*1*) *assms*(*2*) *assms*(*3*) *coord-ring-def eval-at-point-a-inv eval-at-point-const eval-pvar R.indexed-const-closed*)
qed

definition *point-to-polys* :: *'a list* ⇒ (*'a*, *nat*) *mvar-poly list* **where**
point-to-polys as = *map* (λ *x*. *pvar-trans* (*length as*) (*snd x*) (*fst x*)) (*zip as* (*index-list* (*length as*)))

lemma *point-to-polys-length*:
length (*point-to-polys as*) = *length as*
unfolding *point-to-polys-def*
by (*smt index-list-length length-map list.map-ident map-eq-imp-length-eq zip-eq-conv*)

lemma *point-to-polysE*:
assumes *i* < *length as*
shows (*point-to-polys as*) ! *i* = (*pvar-trans* (*length as*) *i* (*as* ! *i*))
proof−

have (*zip as* (*index-list* (*length as*)))!*i* = ((*as*!*i*), *i*)
by (*metis assms index-list-indices index-list-length nth-zip*)
then have *0*: (*point-to-polys as*) ! *i* = (λ *x*. *pvar-trans* (*length as*) (*snd x*) (*fst x*)) ((*as*!*i*), *i*)
unfolding *point-to-polys-def*
using *assms nth-map*[*of i* (*zip as* (*index-list* (*length as*))) (λ*x*. *pvar-trans* (*length as*) (*snd x*) (*fst x*))]
by (*metis index-list-length length-map map-fst-zip*)
then show *?thesis*
by (*metis fst-conv snd-conv*)
qed

lemma *point-to-polysE′*:
assumes *as* ∈ *carrier* (R^n)
assumes *i* < *n*
shows *eval-at-point R as* ((*point-to-polys as*) ! *i*) = **0**
by (*metis assms*(*1*) *assms*(*2*) *cartesian-power-car-memE cartesian-power-car-memE′ point-to-polysE pvar-trans-eval R.r-right-minus-eq*)

lemma *point-to-polysE″*:
assumes *as* ∈ *carrier* (R^n)
assumes *b* ∈ *set* (*point-to-polys as*)
shows *eval-at-point R as b* = **0**
using *point-to-polysE′*
by (*metis assms*(*1*) *assms*(*2*) *cartesian-power-car-memE in-set-conv-nth point-to-polys-length*)

lemma *point-to-polys-zero-set*:
assumes *as* ∈ *carrier* (R^n)
assumes *b* ∈ *set* (*point-to-polys as*)
shows *as* ∈ *zero-set R n b*
using *assms*(*1*) *assms*(*2*) *point-to-polysE″ zero-setI* **by** *blast*

lemma *point-to-polys-closed*:
assumes *as* ∈ *carrier* (R^n)
shows *set* (*point-to-polys as*) ⊆ *carrier* ($R[\mathcal{X}_n]$)
using *assms point-to-polysE pvar-trans-closed*
by (*smt cartesian-power-car-memE cartesian-power-car-memE′ in-set-conv-nth point-to-polys-length subsetI*)

lemma *point-to-polys-affine-alg-set*:
assumes *as* ∈ *carrier* (R^n)
shows *affine-alg-set R n* (*set* (*point-to-polys as*)) = {*as*}
proof
show *affine-alg-set R n* (*set* (*point-to-polys as*)) ⊆ {*as*}
proof
fix *x*
assume *A0*: *x* ∈ *affine-alg-set R n* (*set* (*point-to-polys as*))
then have *0*: *length x* = *n* **using** *affine-alg-set-closed*[*of n* (*set* (*point-to-polys as*))]

```
    using cartesian-power-car-memE by blast
  have ⋀i. i < n ⟹ x!i = as!i
  proof−
    fix i
    assume A1: i < n
    show  x!i = as!i
      using A0
    by (smt A1 affine-alg-set-closed affine-alg-set-memE assms cartesian-power-car-memE

        cartesian-power-car-memE' nth-mem point-to-polysE point-to-polys-length
         pvar-trans-eval R.r-right-minus-eq subsetD)
  qed
  then have x = as
    by (metis 0 assms cartesian-power-car-memE nth-equalityI)
  then show x ∈ {as}
    by blast
 qed
 show {as} ⊆ affine-alg-set R n (set (point-to-polys as))
 proof−
   have as ∈  affine-alg-set R n (set (point-to-polys as))
     using affine-alg-set-not-memE assms point-to-polysE''
     by blast
   then show ?thesis
     by blast
 qed
qed

lemma singleton-is-algebraic:
 assumes as ∈ carrier (R^n)
 shows is-algebraic R n {as}
 apply(rule is-algebraicI[of (set (point-to-polys as))])
 apply blast
 using point-to-polys-affine-alg-set
 using assms point-to-polys-closed apply blast
 by (simp add: assms point-to-polys-affine-alg-set)

lemma(in domain-coord-rings) finite-sets-are-algebraic:
 assumes finite F
 shows F ⊆ carrier (R^n) ⟶ is-algebraic R n F
 apply(rule finite.induct)
 apply (simp add: assms)
 using empty-is-alg apply blast
 using singleton-is-algebraic
 by (metis union-is-alg insert-is-Un insert-subset)
```

5.6 Polynomial Maps

5.7 The Action of Index Permutations on Polynomials

definition *permute-poly-args* ::

nat ⇒ (*nat* ⇒ *nat*) ⇒ ('*a*, *nat*) *mvar-poly* ⇒ ('*a*, *nat*) *mvar-poly* **where**
permute-poly-args (*n*::*nat*) *σ p* = *indexed-poly-induced-morphism* {..<*n*} (*R*[$\mathcal{X}_n$]) *coord-const* (λ*i*. *pvar R* (*σ i*)) *p*

lemma *permute-poly-args-characterization*:
assumes *σ permutes* {..< *n*}
shows (*ring-hom-ring* (*R*[$\mathcal{X}_n$]) (*R*[$\mathcal{X}_n$]) (*permute-poly-args* (*n*::*nat*) *σ*))
(∀ *i* ∈ {..<*n*}. (*permute-poly-args* (*n*::*nat*) *σ*) (*pvar R i*) = *pvar R* (*σ i*))
(∀ *a* ∈ *carrier R*. *permute-poly-args* (*n*::*nat*) *σ* (*coord-const a*) = (*coord-const a*))
proof−
have *0*: *cring* (*R*[$\mathcal{X}_n$])
by (*simp add*: *MP.is-cring*)
have *1*: (λ*i*. *pvar R* (*σ i*)) ∈ {..<*n*} → *carrier* (*R*[$\mathcal{X}_n$])
proof
fix *x*
assume *A*: *x* ∈ {..<*n*}
then have *0*: *σ x* ∈ {..<*n*}
using *assms*
by (*meson permutes-in-image*)
then have *σ x* < *n*
using *assms*
by *auto*
then show *pvar R* (*σ x*) ∈ *carrier* (*R*[$\mathcal{X}_n$])
using *pvar-closed*[*of σ x n*]
by *blast*
qed
have *2*: *ring-hom-ring R* (*R*[$\mathcal{X}_n$]) *coord-const*
using *R.indexed-const-ring-hom* **unfolding** *coord-ring-def*
by *blast*
have *3*: *indexed-poly-induced-morphism* {..<*n*} (*R*[$\mathcal{X}_n$]) *coord-const* (λ*i*. *pvar R* (*σ i*)) =
indexed-poly-induced-morphism {..<*n*} (*R*[$\mathcal{X}_n$]) *coord-const* (λ*i*. *pvar R* (*σ i*))
by *blast*
show *ring-hom-ring* (*R*[$\mathcal{X}_n$]) (*R*[$\mathcal{X}_n$]) (*permute-poly-args n σ*)
using *0 1 2 3*
R.Pring-universal-prop[*of* (*R*[$\mathcal{X}_n$]) (λ*i*. *pvar R* (*σ i*)) {..<*n*} *coord-const permute-poly-args* (*n*::*nat*) *σ*]
unfolding *permute-poly-args-def*
by (*metis coord-ring-def*)
show ∀ *i*∈{..<*n*}. *permute-poly-args n σ* (*pvar R i*) = *pvar R* (*σ i*)
using *0 1 2 3*
R.Pring-universal-prop(*2*)[*of* (*R*[$\mathcal{X}_n$]) (λ*i*. *pvar R* (*σ i*)) {..<*n*} *coord-const permute-poly-args* (*n*::*nat*) *σ*]
unfolding *permute-poly-args-def var-to-IP-def*
by *blast*
show ∀ *a*∈*carrier R*. *permute-poly-args n σ* (*coord-const a*) = *coord-const a*
using *0 1 2 3*
R.Pring-universal-prop[*of* (*R*[$\mathcal{X}_n$]) (λ*i*. *pvar R* (*σ i*)) {..<*n*} *coord-const*

permute-poly-args (*n*::*nat*) σ]
unfolding *permute-poly-args-def var-to-IP-def*
by *blast*
qed

lemma *permute-poly-args-closed*:
assumes σ *permutes* {..<*n*}
assumes $p \in$ *carrier* ($R[\mathcal{X}_n]$)
shows *permute-poly-args n* σ $p \in$ *carrier* ($R[\mathcal{X}_n]$)
proof−
have (*ring-hom-ring* ($R[\mathcal{X}_n]$) ($R[\mathcal{X}_n]$) (*permute-poly-args* (*n*::*nat*) σ))
using *assms permute-poly-args-characterization*(*1*)
by *blast*
then have (*permute-poly-args* (*n*::*nat*) σ) $\in$ *carrier* ($R[\mathcal{X}_n]$) $\rightarrow$ *carrier* ($R[\mathcal{X}_n]$)
unfolding *ring-hom-ring-def ring-hom-ring-axioms-def ring-hom-def*
by *blast*
then show *?thesis*
using *assms*
by *blast*
qed

lemma *permute-poly-args-constant*:
assumes $a \in$ *carrier R*
assumes σ *permutes* {..<*n*}
shows *permute-poly-args n* σ (*coord-const a*) = (*coord-const a*)
using *assms permute-poly-args-characterization*(*3*)
by *blast*

lemma *permute-poly-args-add*:
assumes σ *permutes* {..<*n*}
assumes $p \in$ *carrier* ($R[\mathcal{X}_n]$)
assumes $q \in$ *carrier* ($R[\mathcal{X}_n]$)
shows *permute-poly-args n* σ ($p \oplus_{R[\mathcal{X}_n]} q$) = (*permute-poly-args n* σ p) $\oplus_{R[\mathcal{X}_n]}$ (*permute-poly-args n* σ q)
using *permute-poly-args-characterization*(*1*)[*of* σ] *assms*
unfolding *ring-hom-ring-def ring-hom-ring-axioms-def*
by (*metis* (*no-types*, *lifting*) *ring-hom-add*)

lemma *permute-poly-args-mult*:
assumes σ *permutes* {..<*n*}
assumes $p \in$ *carrier* ($R[\mathcal{X}_n]$)
assumes $q \in$ *carrier* ($R[\mathcal{X}_n]$)
shows *permute-poly-args n* σ ($p \otimes_{R[\mathcal{X}_n]} q$) = (*permute-poly-args n* σ p) $\otimes_{R[\mathcal{X}_n]}$ (*permute-poly-args n* σ q)
using *permute-poly-args-characterization*(*1*)[*of* σ] *assms*
unfolding *ring-hom-ring-def ring-hom-ring-axioms-def*
using *ring-hom-mult*
by (*metis* (*mono-tags*, *lifting*))

lemma *permute-poly-args-indexed-pmult*:
assumes σ *permutes* $\{..<n\}$
assumes $p \in$ *carrier* $(R[\mathcal{X}_n])$
assumes $i \in \{..<n\}$
shows (*permute-poly-args n* σ $(p \bigotimes i)) = ($*permute-poly-args n* σ *p*$) \bigotimes (\sigma\ i)$
proof
fix x
show *permute-poly-args n* σ $(p \bigotimes i)\ x = ($*permute-poly-args n* σ *p* $\bigotimes\ \sigma\ i)\ x$
proof−
have *0*: $(p \bigotimes i) = (p \otimes_{R[\mathcal{X}_n]}$ *pvar R i*$)$
using *assms pvar-indexed-pmult*
by *blast*
have *1*: (*permute-poly-args n* σ *p*$) \bigotimes (\sigma\ i) = ($*permute-poly-args n* σ *p*$) \otimes_{R[\mathcal{X}_n]}$ *pvar R* $(\sigma\ i)$
using *assms permute-poly-args-closed pvar-indexed-pmult* **by** *blast*
have *2*: *permute-poly-args n* σ $(p \bigotimes i)\ x =$ *permute-poly-args n* σ $(p \otimes_{R[\mathcal{X}_n]}$ *pvar R i*$)\ x$
using ‹$p \bigotimes i = p \otimes_{R[\mathcal{X}_n]}$ *pvar R i*› **by** *presburger*
then show *?thesis* **using** *1 R.Pring-var-closed assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*
permute-poly-args-mult R.is-cring permute-poly-args-characterization(*2*) *R.zero-closed*
by (*metis coord-ring-def*)
qed
qed

lemma *permute-list-closed*:
assumes $a \in$ *carrier* (R^n)
assumes σ *permutes* $\{..<n\}$
shows (*permute-list* σ *a*$) \in$ *carrier* (R^n)
apply(*rule cartesian-power-car-memI*)
using *assms cartesian-power-car-memE length-permute-list* **apply** *blast*
proof−
have *0*: *set a* $\subseteq$ *carrier R*
using *assms*(*1*) *cartesian-power-car-memE''* **by** *blast*
have σ *permutes* $\{..<$*length a*$\}$
proof−
have *0*: *length a* $= n$
using *assms cartesian-power-car-memE* **by** *blast*
have $\{..<n\} = \{..<$*length a*$\}$
using *0* **by** *blast*
then show *?thesis*
using *assms* **by** *presburger*
qed
have *1*: *set* (*permute-list* σ *a*$) =$ *set a*
using *assms set-permute-list*[*of* σ *a*] ‹σ *permutes* $\{..<$*length a*$\}$›
by *blast*
then show *set* (*permute-list* σ *a*$) \subseteq$ *carrier R*

by (*simp add*: *1 0*)
qed

lemma *permute-list-set*:
assumes $a \in$ *carrier* (R^n)
assumes σ *permutes* $\{..<n\}$
shows *set* (*permute-list* σ *a*) = *set a*
proof−
have σ *permutes* $\{..<$*length a*$\}$
proof−
have *0*: *length a* = *n*
using *assms cartesian-power-car-memE* **by** *blast*
have $\{..<n\} = \{..<$*length a*$\}$
using *0* **by** *blast*
then show *?thesis*
using *assms* **by** *presburger*
qed
then show *1*: *set* (*permute-list* σ *a*) = *set a*
using *assms set-permute-list*[*of* σ *a*]
by *blast*
qed

end

definition *perm-map* :: ('*a*, '*b*) *ring-scheme* ⇒ *nat* ⇒ (*nat* ⇒ *nat*) ⇒ '*a list* ⇒ '*a list* **where**
perm-map R n σ = *restrict* (*permute-list* σ) (*carrier* (R^n))

context *cring-coord-rings*
begin

lemma *perm-map-is-struct-map*:
assumes σ *permutes* $\{..<n\}$
shows *perm-map R n* $\sigma \in$ *struct-maps* (R^n) (R^n)
apply(*rule struct-maps-memI*)
unfolding *perm-map-def restrict-def* **using** *assms permute-list-closed*[*of - n* σ]
apply *metis*
by *metis*

lemma *permute-poly-args-eval*:
assumes $a \in$ *carrier* (R^n)
assumes σ *permutes* $\{..<n\}$
assumes $p \in$ *carrier* $(R[\mathcal{X}_n])$
shows *eval-at-point R a* (*permute-poly-args n* σ *p*) = *eval-at-point R* (*permute-list* σ *a*) *p*
apply(*rule R.indexed-pset.induct*[*of p* $\{..<n\}$ *carrier R*])
using *R.Pring-car assms* **apply** (*metis coord-ring-def*)
apply (*metis assms*(*1*) *assms*(*2*) *eval-at-point-const permute-list-closed permute-poly-args-constant*)

proof−
show $\bigwedge p\ Q.\ p \in$ *Pring-set* $R\ \{..<n\} \Longrightarrow$
eval-at-point $R\ a$ (*permute-poly-args* $n\ \sigma\ p$) = *eval-at-point* R (*permute-list* $\sigma\ a$) $p \Longrightarrow$
$Q \in$ *Pring-set* $R\ \{..<n\} \Longrightarrow$
eval-at-point $R\ a$ (*permute-poly-args* $n\ \sigma\ Q$) = *eval-at-point* R (*permute-list* $\sigma\ a$) $Q \Longrightarrow$
eval-at-point $R\ a$ (*permute-poly-args* $n\ \sigma$ ($p \bigoplus Q$)) = *eval-at-point* R (*permute-list* $\sigma\ a$) ($p \bigoplus Q$)
proof−
fix $p\ Q$ **assume** *A0*: $p \in$ *Pring-set* $R\ \{..<n\}$
assume *A1*: *eval-at-point* $R\ a$ (*permute-poly-args* $n\ \sigma\ p$) = *eval-at-point* R (*permute-list* $\sigma\ a$) p
assume *A2*: $Q \in$ *Pring-set* $R\ \{..<n\}$
assume *A3*: *eval-at-point* $R\ a$ (*permute-poly-args* $n\ \sigma\ Q$) = *eval-at-point* R (*permute-list* $\sigma\ a$) Q
have *A0'*: $p \in$ *carrier* ($R[\mathcal{X}_n]$)
using *A0 R.Pring-car* **unfolding** *coord-ring-def* **by** *blast*
have *A2'*: $Q \in$ *carrier* ($R[\mathcal{X}_n]$)
using *A2 R.Pring-car* **unfolding** *coord-ring-def* **by** *blast*
have *A4*: (*permute-poly-args* $n\ \sigma$ ($p \bigoplus Q$)) = (*permute-poly-args* $n\ \sigma\ p$) $\bigoplus$ (*permute-poly-args* $n\ \sigma\ Q$)
proof−
have (*permute-poly-args* $n\ \sigma$ ($p \oplus_{R[\mathcal{X}_n]} Q$)) = (*permute-poly-args* $n\ \sigma\ p$) $\oplus_{R[\mathcal{X}_n]}$ (*permute-poly-args* $n\ \sigma\ Q$)
using *A0' A2' assms permute-poly-args-add* **by** *blast*
then show *?thesis*
unfolding *coord-ring-def*
by (*metis R.Pring-add*)
qed
show *A5*: *eval-at-point* $R\ a$ (*permute-poly-args* $n\ \sigma$ ($p \bigoplus Q$)) = *eval-at-point* R (*permute-list* $\sigma\ a$) ($p \bigoplus Q$)
using *eval-at-point-add*[*of a n permute-poly-args n σ p permute-poly-args n σ Q*]
permute-poly-args-add[*of σ n p Q*] *A0' A1 A2' A3 A4 permute-poly-args-closed assms*
by (*metis R.Pring-add cartesian-power-car-memE cartesian-power-car-memE''*
cartesian-power-car-memI coord-ring-def eval-at-point-add length-permute-list permute-list-set)
qed
show $\bigwedge p\ i.\ p \in$ *Pring-set* $R\ \{..<n\} \Longrightarrow$
eval-at-point $R\ a$ (*permute-poly-args* $n\ \sigma\ p$) = *eval-at-point* R (*permute-list* $\sigma\ a$) $p \Longrightarrow$
$i \in \{..<n\} \Longrightarrow$ *eval-at-point* $R\ a$ (*permute-poly-args* $n\ \sigma$ ($p \bigotimes i$)) = *eval-at-point* R (*permute-list* $\sigma\ a$) ($p \bigotimes i$)
proof−
fix $p\ i$
assume *A0*: $p \in$ *Pring-set* $R\ \{..<n\}$

assume *A1*: *eval-at-point R a* (*permute-poly-args n σ p*) = *eval-at-point R* (*permute-list σ a*) *p*
assume *A2*: $i \in \{..<n\}$
have *LHS*: *eval-at-point R a* (*permute-poly-args n σ* ($p \bigotimes i$)) = *eval-at-point R a* (*permute-poly-args n σ p* $\bigotimes$ *σ i*)
using *permute-poly-args-indexed-pmult*[*of σ n p i*] *A0 A1 A2 assms*
by (*metis R.Pring-car coord-ring-def*)
then have *LHS′* : *eval-at-point R a* (*permute-poly-args n σ* ($p \bigotimes i$)) = *eval-at-point R a* (*permute-poly-args n σ p* $\otimes_{R[\mathcal{X}_n]}$ *pvar R* (*σ i*))
using *A0 R.Pring-car assms*(*1*) *assms permute-poly-args-closed pvar-indexed-pmult*
by (*metis coord-ring-def*)
have *eval-at-point R a* (*permute-poly-args n σ p* $\otimes_{R[\mathcal{X}_n]}$ *pvar R* (*σ i*)) = *eval-at-point R a* (*permute-poly-args n σ p*) $\otimes$ *eval-at-point R a* (*pvar R* (*σ i*))
proof−
have *1*: *permute-poly-args n σ p* ∈ *carrier* ($R[\mathcal{X}_n]$)
using *A0 R.Pring-car assms*(*1*) *assms permute-poly-args-closed*
by (*metis coord-ring-def*)
have *pvar R* (*σ i*) ∈ *carrier* ($R[\mathcal{X}_n]$)
proof−
have $\sigma\ i \in \{..<n\}$
using *A2 assms*
by (*meson permutes-in-image*)
then have $(\sigma\ i) < n$
by *blast*
then show *?thesis*
using *pvar-closed*[*of σ i n*]
by *blast*
qed
then have *LHS″* : *eval-at-point R a* (*permute-poly-args n σ* ($p \bigotimes i$)) = (*eval-at-point R a* (*permute-poly-args n σ p*)) $\otimes_R$ *eval-at-point R a* (*pvar R* (*σ i*))
using *LHS′ 1 eval-at-point-mult assms*
by *presburger*
then show *?thesis*
using *LHS′* **by** *presburger*
qed
then have *LHS″*: *eval-at-point R a* (*permute-poly-args n σ* ($p \bigotimes i$)) = *eval-at-point R a* (*permute-poly-args n σ p*) $\otimes$ *eval-at-point R a* (*pvar R* (*σ i*))
using *LHS′* **by** *presburger*
have *0*: *eval-at-point R a* (*pvar R* (*σ i*)) = *a*! (*σ i*)
proof−
have $\sigma\ i \in \{..<n\}$
using *A2 assms*
by (*meson permutes-in-image*)
then have *0*: $\sigma\ i < n$
by *blast*
have *1*: *permute-list σ a* ∈ *carrier* (R^n)

using *assms(1) assms(2) assms(3) permute-list-closed* **by** *blast*
show *?thesis*
using *0 1 eval-pvar[of σ i n a] assms*
by *blast*
qed
have *1*: *(permute-list σ a)! i = a! σ i*
proof−
have *length a = n*
using *assms cartesian-power-car-memE*
by *blast*
then have *{..<length a} = {..<n}*
by *blast*
then have *0*: *σ permutes {..<length a}*
using *assms*
by *presburger*
have *1*: *i < length a*
using *A2 ‹{..<length a} = {..<n}›*
by *blast*
show *?thesis* **using** *0 1 permute-list-nth[of σ a i]*
by *blast*
qed
have *LHS'''*: *eval-at-point R a (permute-poly-args n σ (p ⨂ i)) =*
eval-at-point R (permute-list σ a) p ⊗ a! (σ i)
using *0 LHS'' A1*
by *presburger*
have *RHS*: *eval-at-point R (permute-list σ a) (p ⨂ i) =*
(eval-at-point R (permute-list σ a) p) ⊗$_R$ *(eval-at-point R (permute-list σ a) (pvar R i))*
proof−
have *(p ⨂ i) = p ⊗*$_{R[\mathcal{X}_n]}$ *(pvar R i)*
using *A0 R.Pring-car pvar-indexed-pmult* **unfolding** *coord-ring-def*
by *blast*
then show *?thesis*
using *eval-at-point-mult[of (permute-list σ a) n p (pvar R i)]*
A0 A2 R.Pring-car R.Pring-var-closed assms(1) assms(2) assms(3) permute-list-closed
by *(metis coord-ring-def)*
qed
then have *RHS'*: *eval-at-point R (permute-list σ a) (p ⨂ i) =*
(eval-at-point R (permute-list σ a) p) ⊗$_R$ *(permute-list σ a)! i*
proof−
have *0*: *i < n*
using *A2 assms*
by *blast*
have *1*: *permute-list σ a ∈ carrier* (R^n)
using *assms permute-list-closed*
by *blast*
show *?thesis*
using *0 1 eval-pvar[of i n (permute-list σ a)] RHS*

by *presburger*
qed
then show *eval-at-point R a* (*permute-poly-args n σ* (*p* ⨂ *i*)) = *eval-at-point R* (*permute-list σ a*) (*p* ⨂ *i*)
using *LHS''' A1 1*
by *presburger*
qed
qed

5.8 Inverse Images of Sets by Tuples of Polynomials

definition *is-poly-tuple* :: *nat* ⇒ ('*a*, *nat*) *mvar-poly list* ⇒ *bool* **where**
is-poly-tuple (*n*::*nat*) *fs* = (*set* (*fs*) ⊆ *carrier* (*R*[$\mathcal{X}_n$]))

lemma *is-poly-tupleE*:
assumes *is-poly-tuple n fs*
assumes *j* < *length fs*
shows *fs* ! *j* ∈ *carrier* (*R*[$\mathcal{X}_n$])
using *assms is-poly-tuple-def nth-mem*
by *blast*

lemma *is-poly-tuple-Cons*:
assumes *is-poly-tuple n fs*
assumes *f* ∈ *carrier* (*R*[$\mathcal{X}_n$])
shows *is-poly-tuple n* (*f*#*fs*)
using *assms* **unfolding** *is-poly-tuple-def*
by (*metis* (*no-types*, *lifting*) *set-ConsD subset-iff*)

lemma *is-poly-tuple-append*:
assumes *is-poly-tuple n fs*
assumes *f* ∈ *carrier* (*R*[$\mathcal{X}_n$])
shows *is-poly-tuple n* (*fs*@[*f*])
using *assms set-append* **unfolding** *is-poly-tuple-def*
by (*metis* (*no-types*, *lifting*) *Un-subset-iff append-Nil2 set-ConsD subset-code*(*1*))

definition *poly-tuple-eval* :: ('*a*, *nat*) *mvar-poly list* ⇒ '*a list* ⇒ '*a list* **where**
poly-tuple-eval fs as = *map* (λ *f*. *eval-at-poly R f as*) *fs*

lemma *poly-tuple-evalE*:
assumes *is-poly-tuple n fs*
assumes *length fs* = *m*
assumes *as* ∈ *carrier* (R^n)
assumes *j* < *m*
shows (*poly-tuple-eval fs as*)!*j* ∈ *carrier R*
proof−
have *0*: (*poly-tuple-eval fs as*)!*j* = (*eval-at-poly R* (*fs*!*j*) *as*)
using *poly-tuple-eval-def*
by (*metis assms*(*2*) *assms*(*4*) *nth-map*)
have *1*: (*fs*!*j*) ∈ *carrier* (*R*[$\mathcal{X}_n$])

using *assms is-poly-tupleE*
by *blast*
show *?thesis*
using *assms 0 1 eval-at-point-closed*
by *presburger*
qed

lemma *poly-tuple-evalE′*:
shows *length* (*poly-tuple-eval fs as*) = *length fs*
unfolding *poly-tuple-eval-def*
using *length-map* **by** *blast*

lemma *poly-tuple-evalE′′*:
assumes *is-poly-tuple n fs*
assumes *length fs* = *m*
assumes *as* ∈ *carrier* (R^n)
assumes *j* < *m*
shows (*poly-tuple-eval fs as*)!*j* = (*eval-at-poly R* (*fs*!*j*) *as*)
using *assms*
unfolding *poly-tuple-eval-def*
using *nth-map*
by *blast*

lemma *poly-tuple-eval-closed*:
assumes *is-poly-tuple n fs*
assumes *length fs* = *m*
assumes *as* ∈ *carrier* (R^n)
shows (*poly-tuple-eval fs as*) ∈ *carrier* (R^m)
proof(*rule cartesian-power-car-memI*)
show *length* (*poly-tuple-eval fs as*) = *m*
using *assms*
by (*simp add*: *assms poly-tuple-evalE′*)
show *set* (*poly-tuple-eval fs as*) ⊆ *carrier R*
proof fix *x*
assume *x* ∈ *set* (*poly-tuple-eval fs as*)
then obtain *j* **where** *j-def*: *j*< *m* ∧ *x* = (*poly-tuple-eval fs as*)!*j*
using *assms*
by (*metis* ‹*length* (*poly-tuple-eval fs as*) = *m*› *in-set-conv-nth*)
then show *x* ∈ *carrier R*
using *assms*(*1*) *assms*(*2*) *assms*(*3*) *poly-tuple-evalE assms* **by** *blast*
qed
qed

lemma *poly-tuple-eval-Cons*:
assumes *is-poly-tuple n fs*
assumes *length fs* = *m*
assumes *as* ∈ *carrier* (R^n)
assumes *f* ∈ *carrier* ($R[\mathcal{X}_n]$)
shows (*poly-tuple-eval* (*f*#*fs*) *as*) = (*eval-at-point R as f*)#(*poly-tuple-eval fs as*)

using *assms poly-tuple-eval-def*
by (*metis list.simps*(9))

definition *poly-tuple-pullback* ::
nat ⇒ *'a list set* ⇒ (*'a*, *nat*) *mvar-poly list* ⇒ *'a list set* **where**
poly-tuple-pullback n S fs = ((*poly-tuple-eval fs*) −‘ *S*) ∩ (*carrier* (*R*^*n*))

lemma *poly-pullbackE*:
assumes *is-poly-tuple n fs*
assumes *length fs* = *m*
assumes *S* ⊆ *carrier* (*R*^*m*)
shows *poly-tuple-pullback n S fs* ⊆ *carrier* (*R*^*n*)
using *poly-tuple-pullback-def assms*
by *blast*

lemma *poly-pullbackE'*:
assumes *is-poly-tuple n fs*
assumes *length fs* = *m*
assumes *S* ⊆ *carrier* (*R*^*m*)
assumes *as* ∈ *poly-tuple-pullback n S fs*
shows *as* ∈ *carrier* (*R*^*n*)
poly-tuple-eval fs as ∈ *S*
using *assms*
apply (*meson poly-pullbackE subsetD*)
proof−
have *as* ∈ *poly-tuple-eval fs* −‘ *S*
using *assms* **unfolding** *poly-tuple-pullback-def*
by *blast*
then show *poly-tuple-eval fs as* ∈ *S*
by *blast*
qed

lemma *poly-pullbackI*:
assumes *is-poly-tuple n fs*
assumes *length fs* = *m*
assumes *S* ⊆ *carrier* (*R*^*m*)
assumes *as* ∈ *carrier* (*R*^*n*)
assumes *poly-tuple-eval fs as* ∈ *S*
shows *as* ∈ *poly-tuple-pullback n S fs*
using *assms*
unfolding *poly-tuple-pullback-def*
by *blast*

end

coordinate permutations as pullbacks. The point here is to realize that permutations of indices are just pullbacks (or pushforwards) by particular

polynomial maps

abbreviation *pvar-list* **where**
pvar-list R n ≡ *map* (*pvar R*) (*index-list n*)

lemma *pvar-list-elements*:
assumes *i* < *n*
shows *pvar-list R n* ! *i* = *pvar R i*
by (*simp add*: *assms index-list-indices index-list-length*)

lemma *pvar-list-length*:
length (*pvar-list R n*) = *n*
by (*simp add*: *index-list-length*)

context *cring-coord-rings*
begin

lemma *pvar-list-is-poly-tuple*:
shows *is-poly-tuple n* (*pvar-list R n*)
unfolding *is-poly-tuple-def*
proof fix *x*
assume *A*: *x* ∈ *set* (*pvar-list R n*)
have *set* (*index-list n*) = {..<*n*}
by (*simp add*: *index-list-set*)
obtain *i* **where** *i* < *n* ∧ *x* = *pvar R i*
using *A pvar-list-elements*[*of - n R*] *pvar-list-length*[*of R n*]
by (*metis in-set-conv-nth*)
then show *x* ∈ *carrier* (*R*[$\mathcal{X}_n$])
using *pvar-closed*
by *blast*
qed

lemma *permutation-of-poly-list-is-poly-list*:
assumes *is-poly-tuple k fs*
assumes *σ permutes* {..< *length fs*}
shows *is-poly-tuple k* (*permute-list σ fs*)
unfolding *is-poly-tuple-def*
proof−
show *set* (*permute-list σ fs*) ⊆ *carrier* (*coord-ring R k*)
using *assms is-poly-tuple-def set-permute-list*
by *blast*
qed

lemma *permutation-of-poly-listE*:
assumes *is-poly-tuple k fs*
assumes *σ permutes* {..< *length fs*}
assumes *i* < *length fs*
shows (*permute-list σ fs*) ! *i* = *fs* ! (*σ i*)
using *assms permute-list-nth*
by *blast*

lemma *pushforward-by-permutation-of-poly-list*:
assumes *is-poly-tuple k fs*
assumes σ *permutes* $\{..< length\ fs\}$
assumes *as* $\in$ *carrier* (R^k)
shows *poly-tuple-eval* (*permute-list* σ *fs*) *as* = *permute-list* σ (*poly-tuple-eval fs as*)
using *assms* **unfolding** *poly-tuple-eval-def*
by (*metis permute-list-map*)

lemma *pushforward-by-pvar-list*:
assumes *as* $\in$ *carrier* (R^n)
shows *poly-tuple-eval* (*pvar-list R n*) *as* = *as*
using *assms pvar-list-elements*[*of - n R*] **unfolding** *poly-tuple-eval-def* **using** *eval-pvar*[*of - n as*]
by (*metis* (*mono-tags*, *lifting*) *cartesian-power-car-memE length-map nth-equalityI nth-map pvar-list-length*)

lemma *pushforward-by-permuted-pvar-list*:
assumes σ *permutes* $\{..< n\}$
assumes *as* $\in$ *carrier* (R^n)
shows *poly-tuple-eval* (*permute-list* σ (*pvar-list R n*)) *as* = *permute-list* σ *as*
by (*metis assms pushforward-by-permutation-of-poly-list pushforward-by-pvar-list pvar-list-is-poly-tuple pvar-list-length*)

lemma *pullback-by-permutation-of-poly-list*:
assumes σ *permutes* $\{..< n\}$
assumes $S \subseteq$ *carrier* (R^n)
shows *poly-tuple-pullback n S* (*permute-list* σ (*pvar-list R n*)) =
permute-list (*fun-inv* σ) ‘ *S*
proof
show *poly-tuple-pullback n S* (*permute-list* σ (*pvar-list R n*)) $\subseteq$ *permute-list* (*fun-inv* σ) ‘ *S*
proof fix *x*
assume *A*: *x* $\in$ *poly-tuple-pullback n S* (*permute-list* σ (*pvar-list R n*))
then obtain *y* **where** *y-def*: *y* $\in$ *S* $\wedge$ *poly-tuple-eval* (*permute-list* σ (*pvar-list R n*)) *x* = *y*
by (*metis assms length-permute-list permutation-of-poly-list-is-poly-list poly-pullbackE'*(*2*) *pvar-list-is-poly-tuple pvar-list-length*)
then have *0*: *y* = *permute-list* σ *x*
by (*metis A assms length-permute-list permutation-of-poly-list-is-poly-list poly-pullbackE'*(*1*) *pushforward-by-permuted-pvar-list pvar-list-is-poly-tuple pvar-list-length*)
have *1*: *length x* = *n*
using *A*
by (*metis 0 length-permute-list poly-tuple-evalE' pvar-list-length y-def*)
then have $\{..<length\ x\} = \{..<n\}$

```
      by blast
    then have permute-list (fun-inv σ) y = x
     using 0 permutes-inv-o(1)[of σ {..< n}] permute-list-id[of x] permutes-inv[of
σ {..<n}]
           assms permute-list-compose[of (fun-inv σ) x σ ]
      unfolding fun-inv-def
      by metis
    then show  x ∈ permute-list (fun-inv σ) ‘ S
      using y-def by blast
  qed
   show permute-list (fun-inv σ) ‘ S ⊆ poly-tuple-pullback n S (permute-list σ
(pvar-list R n))
  proof fix x assume A: x ∈ permute-list (fun-inv σ) ‘ S
    then obtain y where y-def: y ∈ S ∧ x = permute-list (fun-inv σ) y
      by blast
    have 0: (fun-inv σ) permutes {..<n}
      using assms unfolding fun-inv-def
      by (simp add: permutes-inv)
    have 1: permute-list σ x = permute-list σ (permute-list (fun-inv σ) y)
      by (simp add: y-def)
    have 2: length y = n
      using y-def A assms cartesian-power-car-memE
      by blast
    have 3: σ permutes {..<length y}
      by (simp add: 2 assms)
    have 4: permute-list σ x = y
     using assms(2) permute-list-id[of y]  permute-list-compose[of σ y (fun-inv
σ) ]
          3 2 1 0 permutes-inv-o(2)[of σ {..< n}]
      unfolding fun-inv-def
      by metis
    have 5: x ∈ carrier (R^n)
     apply(rule cartesian-power-car-memI)
     using A 0 assms
      apply (metis 2 4 length-permute-list)
     using A 0 assms
     by (smt 2 in-set-conv-nth neq0-conv poly-tuple-evalE pushforward-by-pvar-list

       pvar-list-is-poly-tuple pvar-list-length set-permute-list subset-iff y-def)
    then have 6: poly-tuple-eval (permute-list σ (pvar-list R n)) x = y
     using 4  assms pushforward-by-permuted-pvar-list[of σ n x]
     by blast
    then show x ∈ poly-tuple-pullback n S (permute-list σ (pvar-list R n))
     using 5 y-def unfolding poly-tuple-pullback-def
     by blast
  qed
qed

lemma pullback-by-permutation-of-poly-list':
```

assumes σ *permutes* $\{..< n\}$
assumes $S \subseteq$ *carrier* (R^n)
shows *poly-tuple-pullback n S* (*permute-list* (*fun-inv* σ) (*pvar-list R n*)) =
permute-list σ ' *S*
proof−
have *0*: (*fun-inv* (*fun-inv* σ)) = σ
using *assms* **unfolding** *fun-inv-def*
using *permutes-inv-inv*
by *blast*
have *1*: *fun-inv* σ *permutes* $\{..<n\}$
unfolding *fun-inv-def*
using *assms permutes-inv* **by** *blast*
then show *?thesis* **using** *0 assms pullback-by-permutation-of-poly-list*[*of fun-inv* σ *n S*]
by *presburger*
qed

5.9 Composing Polynomial Tuples With Polynomials

composition of a multivaribale polynomial by a list of polynomials

definition *poly-compose* ::
nat ⇒ *nat* ⇒ ('*a*, *nat*) *mvar-poly list* ⇒ ('*a*, *nat*) *mvar-poly* ⇒ ('*a*, *nat*) *mvar-poly*
where
poly-compose n m fs = *indexed-poly-induced-morphism* $\{..<n\}$ (*coord-ring R m*) (λ *s. R.indexed-const s*) (λi. *fs*!*i*)

lemma *poly-compose-var*:
assumes *is-poly-tuple m fs*
assumes *length fs* = *n*
assumes *j* < *n*
shows *poly-compose n m fs* (*pvar R j*) = (*fs*!*j*)
proof−
have *0*: *cring* (*coord-ring R m*)
using *R.Pring-is-cring R.is-cring*
unfolding *coord-ring-def* **by** *blast*
have *1*: (!) *fs* ∈ $\{..<n\}$ → *carrier* (*coord-ring R m*)
using *assms is-poly-tuple-def*
by *auto*
have *2*: *ring-hom-ring R* (*coord-ring R m*) *coord-const*
using *indexed-const-ring-hom coord-const-ring-hom* **by** *blast*
have $\forall i \in \{..<n\}$. *indexed-poly-induced-morphism* $\{..<n\}$ (*coord-ring R m*) *coord-const* ((!) *fs*) (*mset-to-IP R* {#*i*#}) = *fs* ! *i*
using *assms 0 1 2 R.Pring-universal-prop*(*2*)[*of* (*coord-ring R m*) (λi. *fs*!*i*) $\{..<n\}$ (λ *s. R.indexed-const s*) *poly-compose n m fs*]
poly-compose-def
by (*metis var-to-IP-def*)
then show *?thesis*
using *assms*
unfolding *poly-compose-def var-to-IP-def*

by *blast*
qed

lemma *Pring-universal-prop-assms*:
assumes *is-poly-tuple m fs*
assumes *length fs = n*
shows $(\lambda i.\ fs!i) \in \{..<n\} \to$ *carrier* (*coord-ring R m*)
ring-hom-ring R (*coord-ring R m*) *coord-const*
proof
show $\bigwedge x.\ x \in \{..<n\} \Longrightarrow$ *fs* ! *x* ∈ *carrier* (*coord-ring R m*)
using *assms is-poly-tupleE* **by** *blast*
show *ring-hom-ring R* (*coord-ring R m*) *coord-const*
using *R.indexed-const-ring-hom coord-const-ring-hom* **by** *blast*
qed

lemma *poly-compose-ring-hom*:
assumes *is-poly-tuple m fs*
assumes *length fs = n*
shows (*ring-hom-ring* ($R[\mathcal{X}_n]$) (*coord-ring R m*) (*poly-compose n m fs*))
using *Pring-universal-prop-assms*[*of n fs*] *assms*
R.Pring-universal-prop(*1*)[*of* (*coord-ring R m*) ($\lambda i.\ fs!i$) {..<*n*} *coord-const*
(*poly-compose n m fs*)]
unfolding *poly-compose-def*
using *R.Pring-is-cring R.is-cring*
by (*metis Pi-I Pring-universal-prop-assms*(*2*) *coord-ring-def is-poly-tupleE lessThan-iff*)

lemma *poly-compose-closed*:
assumes *is-poly-tuple m fs*
assumes *length fs = n*
assumes *f* ∈ *carrier* ($R[\mathcal{X}_n]$)
shows (*poly-compose n m fs f*) ∈ *carrier* (*coord-ring R m*)
proof−
have *poly-compose n m fs* ∈ *carrier* ($R[\mathcal{X}_n]$) → *carrier* ($R\ [\mathcal{X}_m]$)
using *poly-compose-ring-hom*[*of m fs n*] *assms*
unfolding *ring-hom-ring-def ring-hom-ring-axioms-def ring-hom-def*
by *blast*
then show *?thesis* **using** *assms* **by** *blast*
qed

lemma *poly-compose-add*:
assumes *is-poly-tuple m fs*
assumes *length fs = n*
assumes *f* ∈ *carrier* ($R[\mathcal{X}_n]$)
assumes *g* ∈ *carrier* ($R[\mathcal{X}_n]$)
shows *poly-compose n m fs* (*f* $\oplus_{R[\mathcal{X}_n]}$ *g*) = (*poly-compose n m fs f*) $\oplus_{\textit{coord-ring R m}}$
(*poly-compose n m fs g*)
using *assms poly-compose-ring-hom ring-hom-add*
by (*metis* (*mono-tags*, *lifting*) *ring-hom-ring.homh*)

lemma *poly-compose-mult*:
assumes *is-poly-tuple m fs*
assumes *length fs* = *n*
assumes *f* ∈ *carrier* (*R*[$\mathcal{X}_n$])
assumes *g* ∈ *carrier* (*R*[$\mathcal{X}_n$])
shows *poly-compose n m fs* (*f* ⊗$_{R[\mathcal{X}_n]}$ *g*) = (*poly-compose n m fs f*) ⊗$_{coord\text{-}ring\ R\ m}$ (*poly-compose n m fs g*)
using *assms poly-compose-ring-hom ring-hom-mult*
by (*metis* (*mono-tags*, *lifting*) *ring-hom-ring.homh*)

lemma *poly-compose-indexed-pmult*:
assumes *is-poly-tuple m fs*
assumes *length fs* = *n*
assumes *f* ∈ *carrier* (*R*[$\mathcal{X}_n$])
assumes *i* < *n*
shows *poly-compose n m fs* (*f* ⨂ *i*) = (*poly-compose n m fs f*) ⊗$_{coord\text{-}ring\ R\ m}$ (*fs*!*i*)
proof−
have (*f* ⨂ *i*) = *f* ⊗$_{R[\mathcal{X}_n]}$ *pvar R i*
using *assms pvar-indexed-pmult*
by *blast*
then show *?thesis* **using** *poly-compose-mult poly-compose-var assms*
by (*metis pvar-closed*)
qed

lemma *poly-compose-const*:
assumes *is-poly-tuple m fs*
assumes *length fs* = *n*
assumes *a* ∈ *carrier R*
shows *poly-compose n m fs* (*coord-const a*) = *coord-const a*
using *R.Pring-universal-prop*(*3*)[*of* (*coord-ring R m*) (λ*i*. *fs*!*i*) {..<*n*} *coord-const* (*poly-compose n m fs*)]
Pring-universal-prop-assms assms
unfolding *poly-compose-def*
using *R.Pring-is-cring coord-cring-cring* **by** *blast*

evaluating polynomial compositions

lemma *poly-compose-eval*:
assumes *is-poly-tuple m fs*
assumes *length fs* = *n*
assumes *f* ∈ *carrier* (*R*[$\mathcal{X}_n$])
assumes *as* ∈ *carrier* (R^m)
shows *eval-at-point R* (*poly-tuple-eval fs as*) *f* = *eval-at-point R as* (*poly-compose n m fs f*)
apply(*rule R.indexed-pset.induct*[*of f* {..<*n*} *carrier R*])
using *R.Pring-car assms*
apply (*metis coord-ring-def*)
proof−
show ⋀*k*. *k* ∈ *carrier R* ⟹ *eval-at-poly R* (*coord-const k*) (*poly-tuple-eval fs*

as) = *eval-at-poly R* (*poly-compose n m fs* (*coord-const k*)) *as*
using *assms*
by (*metis* (*no-types*, *lifting*) *eval-at-point-factored poly-compose-const R.total-eval-const*)

show $\bigwedge p\ Q.\ p \in$ *Pring-set R* {..<*n*} $\Longrightarrow$
eval-at-poly R p (*poly-tuple-eval fs as*) = *eval-at-poly R* (*poly-compose n m fs p*) *as* $\Longrightarrow$
$Q \in$ *Pring-set R* {..<*n*} $\Longrightarrow$
eval-at-poly R Q (*poly-tuple-eval fs as*) = *eval-at-poly R* (*poly-compose n m fs Q*) *as* $\Longrightarrow$
eval-at-poly R ($p \bigoplus Q$) (*poly-tuple-eval fs as*) = *eval-at-poly R* (*poly-compose n m fs* ($p \bigoplus Q$)) *as*
proof−
fix *p Q*
assume *A0*: $p \in$ *Pring-set R* {..<*n*}
assume *A1*: *eval-at-poly R p* (*poly-tuple-eval fs as*) = *eval-at-poly R* (*poly-compose n m fs p*) *as*
assume *A2*: $Q \in$ *Pring-set R* {..<*n*}
assume *A3*: *eval-at-poly R Q* (*poly-tuple-eval fs as*) = *eval-at-poly R* (*poly-compose n m fs Q*) *as*
have *A4*: *eval-at-poly R* ($p \bigoplus Q$) (*poly-tuple-eval fs as*) = *eval-at-poly R p* (*poly-tuple-eval fs as*) $\oplus$ *eval-at-poly R Q* (*poly-tuple-eval fs as*)
using *A0 A1 A2 A3*
eval-at-point-add[*of* (*poly-tuple-eval fs as*) *n p Q*]
by (*metis R.Pring-add R.Pring-car assms*(2) *assms*(3) *assms*(4) *assms coord-ring-def neq0-conv poly-tuple-eval-closed*)
have *A5*: *poly-compose n m fs* ($p \bigoplus Q$) = *poly-compose n m fs p* $\oplus_{\textit{coord-ring R m}}$ *poly-compose n m fs Q*
using *assms poly-compose-add*
by (*metis A0 A2 R.Pring-add R.Pring-car coord-ring-def*)
have *A6*: *eval-at-poly R* (*poly-compose n m fs* ($p \bigoplus Q$)) *as* = *eval-at-poly R* (*poly-compose n m fs p*) *as* $\oplus$ *eval-at-poly R* (*poly-compose n m fs Q*) *as*
proof−
have *0*: *as* $\in$ *carrier* (R^m)
by (*simp add*: *assms*)
have *1*: *poly-compose n m fs p* $\in$ *carrier* (*coord-ring R m*)
using *A0 R.Pring-car assms*(1) *assms*(2) *assms*(3) *assms*(4) *poly-compose-closed*
by (*metis coord-ring-def*)
have *2*: *poly-compose n m fs Q* $\in$ *carrier* (*coord-ring R m*)
using *A2 R.Pring-car assms*(1) *assms*(2) *assms*(3) *assms*(4) *poly-compose-closed*
by (*metis coord-ring-def*)
show *?thesis*
using *0 1 2 eval-at-point-add*[*of as m* (*poly-compose n m fs p*) (*poly-compose n m fs Q*)]
A5
by *presburger*
qed

show *eval-at-poly R* (*p* $\bigoplus$ *Q*) (*poly-tuple-eval fs as*) = *eval-at-poly R* (*poly-compose n m fs* (*p* $\bigoplus$ *Q*)) *as*
using *A5 A6 A3 A1 A4*
by *presburger*
qed
show $\bigwedge$*p i. p* ∈ *Pring-set R* {..<*n*} ⟹
eval-at-poly R p (*poly-tuple-eval fs as*) = *eval-at-poly R* (*poly-compose n m fs p*) *as* ⟹
i ∈ {..<*n*} ⟹ *eval-at-poly R* (*p* $\bigotimes$ *i*) (*poly-tuple-eval fs as*) = *eval-at-poly R* (*poly-compose n m fs* (*p* $\bigotimes$ *i*)) *as*
using *assms poly-compose-indexed-pmult eval-at-point-indexed-pmult*
by (*smt R.Pring-car coord-ring-def eval-at-point-mult is-poly-tupleE lessThan-iff neq0-conv poly-compose-closed poly-tuple-evalE'' poly-tuple-eval-closed*)
qed

5.10 Extensional Polynomial Maps

Polynomial Maps between powers of a ring

definition *poly-map* :: *nat* ⇒ ('*a*, *nat*) *mvar-poly list* ⇒ '*a list* ⇒ '*a list* **where**
poly-map n fs = (λ*a* ∈ *carrier* (R^n). *poly-tuple-eval fs a*)

lemma *poly-map-is-struct-map*:
assumes *is-poly-tuple n fs*
assumes *length fs* = *m*
shows *poly-map n fs* ∈ *struct-maps* (R^n) (R^m)
apply(*rule struct-maps-memI*)
unfolding *poly-map-def* **using** *assms*
apply (*metis poly-tuple-eval-closed restrict-apply'*)
by (*meson restrict-apply*)

lemma *poly-map-closed*:
assumes *is-poly-tuple n fs*
assumes *length fs* = *m*
assumes *as* ∈ *carrier* (R^n)
shows *poly-map n fs as* ∈ *carrier* (R^m)
using *assms*
by (*meson poly-map-is-struct-map struct-maps-memE*(*1*))

definition *poly-maps* :: *nat* ⇒ *nat* ⇒ ('*a list* ⇒ '*a list*) *set* **where**
poly-maps n m = {*F*. (∃ *fs*. *length fs* = *m* ∧ *is-poly-tuple n fs* ∧ *F* = *poly-map n fs*)}

lemma *poly-maps-memE*:
assumes *F* ∈ *poly-maps n m*
obtains *fs* **where** *length fs* = *m* ∧ *is-poly-tuple n fs* ∧ *F* = *poly-map n fs*
using *assms* **unfolding** *poly-maps-def* **by** *blast*

lemma *poly-maps-memI*:
assumes *is-poly-tuple n fs*

assumes *length fs* = *m*
assumes *F* = *poly-map n fs*
shows *F* ∈ *poly-maps n m*
using *assms* **unfolding** *poly-maps-def* **by** *blast*

lemma *poly-map-compose-closed*:
assumes *is-poly-tuple n fs*
assumes *length fs* = *m*
assumes *is-poly-tuple k gs*
assumes *length gs* = *n*
shows *is-poly-tuple k* (*map* (*poly-compose n k gs*) *fs*)
unfolding *is-poly-tuple-def*
proof fix *y* **assume** *A*: *y* ∈ *set* (*map* (*poly-compose n k gs*) *fs*)
then obtain *f* **where** *f-def*: *f* ∈ *set fs* ∧ *y* = *poly-compose n k gs f*
by (*smt in-set-conv-nth length-map nth-map*)
then show *y* ∈ *carrier* (*coord-ring R k*)
using *assms poly-compose-closed*
by (*metis in-set-conv-nth is-poly-tupleE*)
qed

lemma *poly-map-compose-closed′*:
assumes *is-poly-tuple n fs*
assumes *length fs* = *m*
assumes *is-poly-tuple k gs*
assumes *length gs* = *n*
shows *poly-map k* (*map* (*poly-compose n k gs*) *fs*) ∈ *poly-maps k m*
apply(*rule poly-maps-memI*[*of - map* (*poly-compose n k gs*) *fs*])
using *poly-map-compose-closed*[*of n fs m k gs*] *assms* **apply** *blast*
apply (*simp add*: *assms*)
by *auto*

lemma *poly-map-pullback-char*:
assumes *is-poly-tuple n fs*
assumes *length fs* = *m*
assumes *is-poly-tuple k gs*
assumes *length gs* = *n*
shows (*pullback* (R^k) (*poly-map k gs*) (*poly-map n fs*)) =
poly-map k (*map* (*poly-compose n k gs*) *fs*)
proof(*rule ext*)
fix *x*
show *pullback* (R^k) (*poly-map k gs*) (*poly-map n fs*) *x* =
poly-map k (*map* (*poly-compose n k gs*) *fs*) *x*
proof(*cases x* ∈ *carrier* (R^k))
case *True*
have *0*: *length* (*pullback* (R^k) (*poly-map k gs*) (*poly-map n fs*) *x*) = *m*
using *True assms poly-map-closed cartesian-power-car-memE*
unfolding *pullback-def*
by (*metis* (*mono-tags*, *lifting*) *compose-eq*)
have *1*: *is-poly-tuple k* (*map* (*poly-compose n k gs*) *fs*)

by (*simp add*: *assms poly-map-compose-closed*)
have *2*: *length* (*map* (*poly-compose n k gs*) *fs*) = *m*
using *assms length-map* **by** *auto*
have *3*: $\bigwedge i.\ i < m \Longrightarrow$
(*pullback* (R^k) (*poly-map k gs*) (*poly-map n fs*) *x*)! *i* =
eval-at-point R (*poly-map k gs x*) (*fs* ! *i*)
unfolding *pullback-def poly-map-def poly-tuple-eval-def*
using *assms True*
by (*smt compose-eq nth-map poly-tuple-eval-closed poly-tuple-eval-def restrict-apply'*)
have *4*: $\bigwedge i.\ i < m \Longrightarrow$
poly-map k (*map* (*poly-compose n k gs*) *fs*) *x* ! *i* =
eval-at-point R (*poly-map k gs x*) (*fs* ! *i*)
unfolding *poly-map-def poly-tuple-eval-def* **using** *True assms*
by (*smt 2 cring-coord-rings.is-poly-tuple-def cring-coord-rings-axioms neq0-conv*

nth-map nth-mem poly-compose-eval poly-tuple-eval-def restrict-apply' subset-code(*1*))
show *?thesis* **using** *0 1 2 3 4 assms True*
by (*metis cartesian-power-car-memE nth-equalityI poly-map-closed*)
next
case *False*
then show *?thesis*
unfolding *poly-map-def pullback-def*
by (*metis affine-alg-set-empty compose-extensional extensional-restrict poly-map-def restrict-def*)
qed
qed

lemma *poly-map-pullback-closed*:
assumes *F* ∈ *poly-maps n m*
assumes *G* ∈ *poly-maps k n*
shows (*pullback* (R^k) *G F*) ∈ *poly-maps k m*
by (*metis assms poly-map-compose-closed'*
poly-map-pullback-char poly-maps-memE)

lemma *poly-map-cons*:
assumes *a* ∈ *carrier* (R^n)
shows *poly-map n* (*f*#*fs*) *a* = (*eval-at-point R a f*)#*poly-map n fs a*
unfolding *poly-map-def poly-tuple-eval-def*
by (*metis* (*mono-tags*, *lifting*) *assms list.simps*(*9*) *restrict-apply'*)

lemma *poly-map-append*:
assumes *a* ∈ *carrier* (R^n)
shows *poly-map n* (*fs*@*gs*) *a* = (*poly-map n fs a*) @ (*poly-map n gs a*)
proof(*induction fs*)
case *Nil*
then show *?case*
using *assms* **unfolding** *poly-map-def poly-tuple-eval-def*

by (*metis* (*no-types*, *lifting*) *map-append restrict-apply'*)
next
case (*Cons f fs*)
have *poly-map n* ((*f # fs*) @ *gs*) *a* = (*eval-at-point R a f*)#(*poly-map n* (*fs*@*gs*) *a*)
using *poly-map-cons*
by (*metis append-Cons assms*)
hence *poly-map n* ((*f # fs*) @ *gs*) *a* = (*eval-at-point R a f*)#(*poly-map n fs a*)@(*poly-map n gs a*)
using *Cons.IH* **by** *metis*
thus *?case*
by (*metis Cons-eq-appendI assms poly-map-cons*)
qed

6 Nesting of Polynomial Rings

lemma *poly-ring-car-mono*:
assumes $n \leq m$
shows *carrier* ($R[\mathcal{X}_n]$) ⊆ *carrier* (*coord-ring R m*)
using *R.Pring-carrier-subset*
unfolding *coord-ring-def*
by (*simp add*: *R.Pring-car R.Pring-carrier-subset assms*)

lemma *poly-ring-car-mono'*[*simp*]:
shows *carrier* ($R[\mathcal{X}_n]$) ⊆ *carrier* ($R[\mathcal{X}_{Suc\ n}]$)
carrier ($R[\mathcal{X}_n]$) ⊆ *carrier* ($R[\mathcal{X}_{n+m}]$)
using *poly-ring-car-mono*
apply *simp*
using *poly-ring-car-mono*
by *simp*

lemma *poly-ring-add-mono*:
assumes $n \leq m$
assumes $A \in$ *carrier* ($R[\mathcal{X}_n]$)
assumes $B \in$ *carrier* ($R[\mathcal{X}_n]$)
shows $A \oplus_{R[\mathcal{X}_n]} B = A \oplus_{coord\text{-}ring\ R\ m} B$
using *assms* **unfolding** *coord-ring-def*
by (*metis R.Pring-add-eq*)

lemma *poly-ring-add-mono'*:
assumes $A \in$ *carrier* ($R[\mathcal{X}_n]$)
assumes $B \in$ *carrier* ($R[\mathcal{X}_n]$)
shows $A \oplus_{R[\mathcal{X}_n]} B = A \oplus_{R[\mathcal{X}_{Suc\ n}]} B$
$A \oplus_{R[\mathcal{X}_n]} B = A \oplus_{R[\mathcal{X}_{n+m}]} B$
using *assms* **unfolding** *coord-ring-def*
apply (*metis R.Pring-add-eq*)
by (*metis R.Pring-add-eq*)

lemma *poly-ring-times-mono*:
assumes $n \leq m$
assumes $A \in$ *carrier* $(R[\mathcal{X}_n])$
assumes $B \in$ *carrier* $(R[\mathcal{X}_n])$
shows $A \otimes_{R[\mathcal{X}_n]} B = A \otimes_{\textit{coord-ring } R\ m} B$
using *assms* **unfolding** *coord-ring-def*
by (*metis R.Pring-mult-eq*)

lemma *poly-ring-times-mono'*:
assumes $A \in$ *carrier* $(R[\mathcal{X}_n])$
assumes $B \in$ *carrier* $(R[\mathcal{X}_n])$
shows $A \otimes_{R[\mathcal{X}_n]} B = A \otimes_{R[\mathcal{X}_{Suc\ n}]} B$
$A \otimes_{R[\mathcal{X}_n]} B = A \otimes_{R[\mathcal{X}_{n+m}]} B$
using *assms* **unfolding** *coord-ring-def*
apply (*metis R.Pring-mult-eq*)
by (*metis R.Pring-mult-eq*)

lemma *poly-ring-one-mono*:
assumes $n \leq m$
shows $\mathbf{1}_{R[\mathcal{X}_n]} = \mathbf{1}_{\textit{coord-ring } R\ m}$
by (*metis R.Pring-one coord-ring-def*)

lemma *poly-ring-zero-mono*:
assumes $n \leq m$
shows $\mathbf{0}_{R[\mathcal{X}_n]} = \mathbf{0}_{\textit{coord-ring } R\ m}$
using *R.Pring-zero-eq*
by (*metis coord-ring-def*)

replacing the variables in a polynomial with new variables

definition *shift-vars* :: *nat* ⇒ *nat* ⇒ (*'a*, *nat*) *mvar-poly* ⇒ (*'a*, *nat*) *mvar-poly*
where
shift-vars n m p = *indexed-poly-induced-morphism* {..<*n*} $(R[\mathcal{X}_{n+m}])$ *coord-const* (λ*i*. *pvar R* (*i* + *m*)) *p*

lemma *shift-vars-char*:
shows (*ring-hom-ring* $(R[\mathcal{X}_n])$ $(R[\mathcal{X}_{n+m}])$ (*shift-vars n m*))
(∀ *i* ∈ {..<*n*}. (*shift-vars n m*) (*pvar R i*) = *pvar R* (*i* + *m*))
(∀ *a* ∈ *carrier R*. (*shift-vars n m*) (*R.indexed-const a*) = (*coord-const a*))
proof−
have *1*: (λ*i*. *pvar R* (*i* + *m*)) ∈ {..<*n*} → *carrier* $(R[\mathcal{X}_{n+m}])$
proof fix *x*
assume *x* ∈ {..<*n*}
then have *x* + *m* < *n* + *m*
using *add-less-mono1* **by** *blast*
then show *pvar R* (*x* + *m*) ∈ *carrier* $(R[\mathcal{X}_{n+m}])$
using *pvar-closed* **by** *blast*
qed
have *2*: *ring-hom-ring R* $(R[\mathcal{X}_{n+m}])$ *coord-const*

using *R.indexed-const-ring-hom* **unfolding** *coord-ring-def*
by *blast*
have *3*: *shift-vars n m = indexed-poly-induced-morphism* {..<*n*} (*R*[$\mathcal{X}_{n+m}$]) *coord-const* (λ*i*. *pvar R* (*i* + *m*))
unfolding *shift-vars-def*
by *blast*
show (*ring-hom-ring* (*R*[$\mathcal{X}_n$]) (*R*[$\mathcal{X}_{n+m}$]) (*shift-vars n m*))
using *1 2 3 R.Pring-universal-prop*[*of* (*R*[$\mathcal{X}_{n+m}$]) (λ*i*. *pvar R* (*i* + *m*)) {..<*n*} *coord-const* (*shift-vars n m*)]
using *MP.is-cring* **by** (*metis coord-ring-def*)
show (∀ *i* ∈ {..<*n*}. (*shift-vars n m*) (*pvar R i*) = *pvar R* (*i* + *m*))
using *1 2 3 R.Pring-universal-prop*[*of* (*R*[$\mathcal{X}_{n+m}$]) (λ*i*. *pvar R* (*i* + *m*)) {..<*n*} *coord-const* (*shift-vars n m*)]
by (*metis R.Pring-is-cring MP.is-cring var-to-IP-def*)
show (∀ *a* ∈ *carrier R*. (*shift-vars n m*) (*R.indexed-const a*) = (*coord-const a*))
using *1 2 3 R.Pring-universal-prop*[*of* (*R*[$\mathcal{X}_{n+m}$]) (λ*i*. *pvar R* (*i* + *m*)) {..<*n*} *coord-const* (*shift-vars n m*)]
by (*meson MP.is-cring*)
qed

lemma *shift-vars-constant*:
assumes *a* ∈ *carrier R*
shows *shift-vars n m* (*coord-const a*) = *coord-const a*
using *assms*(*1*) *shift-vars-char*(*3*) **by** *blast*

lemma *shift-vars-pvar*:
assumes *i* ∈ {..<*n*}
shows *shift-vars n m* (*pvar R i*) = *pvar R* (*i* + *m*)
using *assms shift-vars-char*(*2*) **by** *blast*

lemma *shift-vars-add*:
assumes *p* ∈ *carrier* (*R*[$\mathcal{X}_n$])
assumes *Q* ∈ *carrier* (*R*[$\mathcal{X}_n$])
shows *shift-vars n m* (*p* $\oplus_{R[\mathcal{X}_n]}$ *Q*) = *shift-vars n m p* $\oplus_{R[\mathcal{X}_{n+m}]}$ *shift-vars n m Q*
using *assms shift-vars-char*(*1*)[*of n m*]
unfolding *ring-hom-ring-def ring-hom-ring-axioms-def*
using *ring-hom-add*
by (*metis* (*no-types*, *lifting*))

lemma *shift-vars-mult*:
assumes *p* ∈ *carrier* (*R*[$\mathcal{X}_n$])
assumes *Q* ∈ *carrier* (*R*[$\mathcal{X}_n$])
shows *shift-vars n m* (*p* $\otimes_{R[\mathcal{X}_n]}$ *Q*) = *shift-vars n m p* $\otimes_{R[\mathcal{X}_{n+m}]}$ *shift-vars n m Q*
using *assms shift-vars-char*(*1*)[*of n m*]
unfolding *ring-hom-ring-def ring-hom-ring-axioms-def* **unfolding** *coord-ring-def*

using *ring-hom-mult*

by *metis*

lemma *shift-vars-indexed-pmult*:
assumes $p \in$ *carrier* $(R[\mathcal{X}_n])$
assumes $i \in \{..<n\}$
shows *shift-vars n m* $(p \bigotimes i) = (\textit{shift-vars } n\ m\ p) \otimes_{R[\mathcal{X}_{n+m}]} (\textit{pvar } R\ (i + m))$
proof−
have $(p \bigotimes i) = p \otimes_{R[\mathcal{X}_n]} (\textit{pvar } R\ i)$
using *assms pvar-indexed-pmult* **by** *blast*
then show *?thesis*
using *shift-vars-mult shift-vars-pvar assms* **unfolding** *coord-ring-def*
by (*metis R.Pring-var-closed*)
qed

lemma *shift-vars-closed*:
assumes $p \in$ *carrier* $(R[\mathcal{X}_n])$
shows *shift-vars n m p* $\in$ *carrier* $(R[\mathcal{X}_{n+m}])$
using *assms shift-vars-char*(*1*)[*of n m*] *ring-hom-closed*[*of shift-vars n m*]
unfolding *ring-hom-ring-def ring-hom-ring-axioms-def*
by *blast*

lemma *shift-vars-eval*:
assumes $p \in$ *carrier* $(R[\mathcal{X}_n])$
assumes $a \in$ *carrier* (R^m)
assumes $b \in$ *carrier* (R^n)
shows *eval-at-point R* $(a@b)$ (*shift-vars n m p*) = *eval-at-point R b p*
apply(*rule R.indexed-pset.induct*[*of p* $\{..<n\}$ *carrier R*])
using *R.Pring-car assms* **apply** (*metis coord-ring-def*)
proof−
show $\bigwedge k.\ k \in \textit{carrier } R \Longrightarrow \textit{eval-at-poly } R\ (\textit{shift-vars } n\ m\ (\textit{coord-const } k))\ (a\ @\ b) = \textit{eval-at-poly } R\ (\textit{coord-const } k)\ b$
proof−
fix *k*
have *0*: $(a\ @\ b) \in$ *carrier* (R^{n+m})
using *assms*
by (*metis add.commute cartesian-product-closed'*)
assume *A*: $k \in$ *carrier R*
then show *eval-at-poly R* (*shift-vars n m* (*coord-const k*)) $(a\ @\ b) =$ *eval-at-poly R* (*coord-const k*) *b*
using *assms shift-vars-constant*
eval-at-point-const[*of k* $(a\ @\ b)$ $m + n$]
eval-at-point-const[*of k b n*] *0*
by (*metis eval-at-point-const*)
qed
show $\bigwedge p\ Q.\ p \in \textit{Pring-set } R\ \{..<n\} \Longrightarrow$
eval-at-poly R (*shift-vars n m p*) $(a\ @\ b) =$ *eval-at-poly R p b* $\Longrightarrow$
$Q \in \textit{Pring-set } R\ \{..<n\} \Longrightarrow$
eval-at-poly R (*shift-vars n m Q*) $(a\ @\ b) =$ *eval-at-poly R Q b* $\Longrightarrow$
eval-at-poly R (*shift-vars n m* $(p \bigoplus Q)$) $(a\ @\ b) =$ *eval-at-poly R* $(p \bigoplus$

Q) b
proof− **fix** p Q
assume *A0*: $p \in$ *Pring-set* R $\{..<n\}$
assume *A1*: *eval-at-poly* R (*shift-vars* n m p) (a @ b) = *eval-at-poly* R p b
assume *A2*: $Q \in$ *Pring-set* R $\{..<n\}$
assume *A3*: *eval-at-poly* R (*shift-vars* n m Q) (a @ b) = *eval-at-poly* R Q b
have *A4*: *eval-at-poly* R ($p \bigoplus Q$) b = *eval-at-poly* R p $b \oplus$ *eval-at-poly* R Q b
using *A0 A2 assms eval-at-point-add*[*of b n p Q*]
by (*metis R.Pring-add R.Pring-car coord-ring-def*)
have *A5*: (*shift-vars* n m ($p \bigoplus Q$)) = (*shift-vars* n m p) $\oplus_{R[\mathcal{X}_n]}$ (*shift-vars* n m Q)
using *A0 A2 R.Pring-add R.Pring-car assms*(*1*) *shift-vars-add* **unfolding** *coord-ring-def*
by *metis*
have *A6*: *eval-at-poly* R (*shift-vars* n m ($p \bigoplus Q$)) (a @ b) =
eval-at-poly R (*shift-vars* n m p) (a @ b) $\oplus$ *eval-at-poly* R (*shift-vars* n m Q) (a @ b)
using *A5 eval-at-point-add shift-vars-closed A0 A2 R.Pring-car add.commute*
assms **unfolding** *coord-ring-def*
by (*metis R.Pring-add cartesian-power-concat*(*1*))
have *A7*: *eval-at-poly* R (*shift-vars* n m ($p \bigoplus Q$)) (a @ b) =
eval-at-poly R p $b \oplus$ *eval-at-poly* R Q b
using *A6 A1 A3* **by** *presburger*
then show *eval-at-poly* R (*shift-vars* n m ($p \bigoplus Q$)) (a @ b) = *eval-at-poly* R ($p \bigoplus Q$) b
using *A4*
by *presburger*
qed
show $\bigwedge p\ i.\ p \in$ *Pring-set* R $\{..<n\} \Longrightarrow$
eval-at-poly R (*shift-vars* n m p) (a @ b) = *eval-at-poly* R p $b \Longrightarrow$
$i \in \{..<n\} \Longrightarrow$ *eval-at-poly* R (*shift-vars* n m ($p \bigotimes i$)) (a @ b) = *eval-at-poly* R ($p \bigotimes i$) b
proof− **fix** p i
assume *A0*: $p \in$ *Pring-set* R $\{..<n\}$
then have *A0'*: $p \in$ *carrier* ($R[\mathcal{X}_n]$)
using *R.Pring-car* **unfolding** *coord-ring-def*
by *blast*
assume *A1*: *eval-at-poly* R (*shift-vars* n m p) (a @ b) = *eval-at-poly* R p b
assume *A2*: $i \in \{..<n\}$
have *A3*: (*shift-vars* n m ($p \bigotimes i$)) = (*shift-vars* n m p) $\otimes_{R[\mathcal{X}_{n+m}]}$ (*pvar* R (i + m))
using *A0'* *shift-vars-indexed-pmult A2 assms*(*1*)
by *blast*
have *A4*: *eval-at-poly* R (*shift-vars* n m ($p \bigotimes i$)) (a @ b) =
eval-at-poly R ((*shift-vars* n m p) $\otimes_{R[\mathcal{X}_{n+m}]}$ (*pvar* R (i + m))) (a@b)
using *A3*
by *presburger*
have *A5*: a@$b \in$ *carrier* (R^{n+m})

using *assms(2) assms(3) cartesian-power-concat(2)* **by** *blast*
have *A6*: *eval-at-poly R (shift-vars n m (p ⨂ i)) (a @ b) =*
eval-at-poly R p b ⊗ eval-at-poly R (pvar R (i + m)) (a @ b)
using *A5 A0′ eval-at-point-mult*[*of a@b n+m shift-vars n m p pvar R (i + m)*]
unfolding *A4* **by** (*metis A1 A2 Groups.add-ac(2) lessThan-iff local.pvar-closed nat-add-left-cancel-less shift-vars-closed*)
have *A7*: *eval-at-poly R (pvar R (i + m)) (a @ b) = (a@b)!(i+m)*
proof−
have *i < n*
using *assms A2* **by** *blast*
then have *i + m < n + m*
using *add-less-cancel-right*
by *blast*
then show *?thesis*
using *A5 eval-pvar*[*of i+m n+m a@b*]
by *blast*
qed
then have *A8*: *eval-at-poly R (shift-vars n m (p ⨂ i)) (a @ b) = eval-at-poly R p b ⊗ ((a @ b)!(i+m))*
using *A6* **by** *presburger*
have *A9*: *eval-at-poly R (shift-vars n m (p ⨂ i)) (a @ b) = eval-at-poly R p b ⊗ (b!i)*
proof−
have *length a = m*
using *assms cartesian-power-car-memE* **by** *blast*
then have *(a @ b)!(i+m) = b!i*
by (*metis add.commute nth-append-length-plus*)
then show *?thesis*
using *A8*
by *presburger*
qed
show *eval-at-poly R (shift-vars n m (p ⨂ i)) (a @ b) = eval-at-poly R (p ⨂ i) b*
proof−
have *i < n*
using *A2 assms*
by *blast*
then have *eval-at-poly R (p ⨂ i) b = eval-at-poly R p b ⊗ (b!i)*
using *assms A0′ eval-at-point-indexed-pmult*
by *blast*
then show *?thesis* **using** *A9*
by *presburger*
qed
qed
qed

Evaluating a polynomial from a lower poly ring in a higher power:

lemma *poly-eval-cartesian-prod*:

assumes $a \in$ *carrier* (R^n)
assumes $b \in$ *carrier* (R^m)
assumes $p \in$ *carrier* $(R[\mathcal{X}_n])$
shows *eval-at-point R a p* = *eval-at-point R* $(a@b)$ *p*
apply(*rule coord-ring-induct*[*of p n*])
using *assms* **apply** *blast*
proof−
have *0*: $a@b \in$ *carrier* (R^{n+m})
using *assms cartesian-product-closed'* **by** *blast*
show $\bigwedge aa.$ *aa* $\in$ *carrier R* $\Longrightarrow$ *eval-at-poly R* (*coord-const aa*) *a* = *eval-at-poly R* (*coord-const aa*) (*a* @ *b*)
proof− **fix** *c* **assume** *c* $\in$ *carrier R*
show *eval-at-poly R* (*coord-const c*) *a* = *eval-at-poly R* (*coord-const c*) (*a* @ *b*)
using *eval-at-point-const*[*of c a n*] *eval-at-point-const*[*of c a@b n+m*] *0*
‹*c* $\in$ *carrier R*› *assms*(*2*) *assms*(*1*) **by** *presburger*
qed
show $\bigwedge i\ Q.$ *Q* $\in$ *carrier* $(R[\mathcal{X}_n]) \Longrightarrow$
eval-at-poly R Q a = *eval-at-poly R Q* (*a* @ *b*) $\Longrightarrow$
i < *n* $\Longrightarrow$ *eval-at-poly R* (*Q* $\otimes_{R[\mathcal{X}_n]}$ *pvar R i*) *a* = *eval-at-poly R* (*Q* $\otimes_{R[\mathcal{X}_n]}$ *pvar R i*) (*a* @ *b*)
proof−
fix *i Q*
assume *A0*: *Q* $\in$ *carrier* $(R[\mathcal{X}_n])$
assume *A1*: *eval-at-poly R Q a* = *eval-at-poly R Q* (*a* @ *b*)
assume *A2*: *i* < *n*
have *LHS*: *eval-at-poly R* (*Q* $\otimes_{R[\mathcal{X}_n]}$ *pvar R i*) *a* = *eval-at-poly R Q a* $\otimes$ (*a*!*i*)
by (*metis A0 A2 assms eval-at-point-indexed-pmult pvar-indexed-pmult*)
have *RHS*: *eval-at-poly R* (*Q* $\otimes_{R[\mathcal{X}_n]}$ *pvar R i*) (*a* @ *b*) = *eval-at-poly R Q* (*a*@*b*) $\otimes$ ((*a*@*b*)!*i*)
by (*smt 0 A0 A2 add.commute eval-at-point-indexed-pmult le-add1 poly-ring-car-mono pvar-indexed-pmult subsetD trans-less-add2*)
show *eval-at-poly R* (*Q* $\otimes_{R[\mathcal{X}_n]}$ *pvar R i*) *a* = *eval-at-poly R* (*Q* $\otimes_{R[\mathcal{X}_n]}$ *pvar R i*) (*a* @ *b*)
proof−
have *length a* > *i* **using** *A2 assms*
using *cartesian-power-car-memE* **by** *blast*
then have *a*!*i* = (*a*@*b*)!*i*
by (*metis nth-append*)
then show *?thesis*
using *LHS RHS A1*
by *presburger*
qed
qed
show $\bigwedge Q0\ Q1.$
Q0 $\in$ *carrier* $(R[\mathcal{X}_n]) \Longrightarrow$
Q1 $\in$ *carrier* $(R[\mathcal{X}_n]) \Longrightarrow$
eval-at-poly R Q0 a = *eval-at-poly R Q0* (*a* @ *b*) $\Longrightarrow$

eval-at-poly R Q1 a = eval-at-poly R Q1 (a @ b) $\Longrightarrow$
eval-at-poly R (Q0 $\oplus_{R[\mathcal{X}_n]}$ *Q1) a = eval-at-poly R (Q0* $\oplus_{R[\mathcal{X}_n]}$ *Q1) (a @ b)*
proof−
fix *Q0 Q1*
assume *A0*: *eval-at-poly R Q0 a = eval-at-poly R Q0 (a @ b)*
assume *A1*: *eval-at-poly R Q1 a = eval-at-poly R Q1 (a @ b)*
assume *A2*: *Q0 ∈ carrier (R*$[\mathcal{X}_n]$*)*
assume *A3*: *Q1 ∈ carrier (R*$[\mathcal{X}_n]$*)*
show *eval-at-poly R (Q0* $\oplus_{R[\mathcal{X}_n]}$ *Q1) a = eval-at-poly R (Q0* $\oplus_{R[\mathcal{X}_n]}$ *Q1) (a @ b)*
using *A0 A1 A2 A3 assms eval-at-point-add*[*of - n Q0 Q1*] *0* **unfolding** *coord-ring-def*
by (*metis* (*no-types*, *lifting*) *R.Pring-add-eq basic-trans-rules*(*31*) *coord-ring-def eval-at-point-add le-add1 poly-ring-car-mono*)
qed
qed

Evaluating polynomials at points in higher powers:

lemma *eval-at-points-higher-pow*:
assumes *p ∈ carrier (R*$[\mathcal{X}_n]$*)*
assumes *k ≥ n*
assumes *a ∈ carrier* (R^k)
shows *eval-at-point R a p = eval-at-point R (take n a) p*
using *poly-eval-cartesian-prod*[*of take n a n drop n a k − n p*] *assms*
by (*metis* (*no-types*, *lifting*) *append-take-drop-id cartesian-power-car-memE cartesian-power-car-memE″*
cartesian-power-car-memI length-drop set-drop-subset subset-trans take-closed)

6.1 Diagonal sets in even powers of R

In this section, by a diagonal set in $R^(2m)$ we will mean the set of points (x, x), where $x \in R^m$. This is slightly different from the standard definition. Introducing these sets will be useful for reasoning about multiplicative inverses of functions later on.

definition *diagonal* :: *nat* ⇒ *'a list set* **where**
diagonal m = {*x ∈ carrier* (R^{m+m}). *take m x = drop m x*}

lemma *diagonalE*:
assumes *x ∈ diagonal m*
shows *x = (take m x)@(take m x)*
x ∈ carrier (R^{m+m})
take m x ∈ carrier (R^m)
$\bigwedge i.\ i < m \Longrightarrow x!i = x!(i + m)$
apply (*metis* (*mono-tags*, *lifting*) *append-take-drop-id assms*(*1*) *diagonal-def mem-Collect-eq*)
using *assms diagonal-def*
apply *blast*

apply(*rule cartesian-power-car-memI*)
using *assms* **unfolding** *diagonal-def*
apply (*metis* (*no-types*, *lifting*) *cartesian-power-car-memE le-add2 mem-Collect-eq take-closed*)
proof−
show *set* (*take m x*) ⊆ *carrier R*
proof fix *a*
assume *a* ∈ *set* (*take m x*)
then have *a* ∈ *set x*
by (*meson in-set-takeD*)
then show *a* ∈ *carrier R*
using *assms* **unfolding** *diagonal-def* **using** *cartesian-power-car-memE'*[*of x*]

by (*smt cartesian-power-car-memE in-set-conv-nth mem-Collect-eq*)
qed
show ⋀*i*. *i* < *m* ⟹ *x*!*i* = *x*!(*i* + *m*)
proof− **fix** *i*
assume *A*: *i* < *m*
have *0*: *x* = (*take m x*)@(*take m x*)
using *assms diagonal-def*[*of m*]
by (*metis* (*mono-tags*, *lifting*) *append-take-drop-id mem-Collect-eq*)
then have *1*: *x*!*i* = *take m x* ! *i*
by (*metis A nth-take*)
have *2*: *length x* = *m* + *m*
using *assms*(*1*) *cartesian-power-car-memE diagonal-def* **by** *blast*
have *3*: *take m x* = *drop m x*
by (*metis 0 append-take-drop-id same-append-eq*)
have *4*: *drop m x* ! *i* = *x* ! (*i* + *m*)
by (*metis 2 add.commute le-add1 nth-drop*)
then show *x*!*i* = *x*!(*i* + *m*)
using *1 3* **by** *presburger*
qed
qed

lemma *diagonalI*:
assumes *x* = (*take m x*)@(*take m x*)
assumes *take m x* ∈ *carrier* (R^m)
shows *x* ∈ *diagonal m*
unfolding *diagonal-def* **using** *assms*
by (*metis* (*mono-tags*, *lifting*) *append-eq-conv-conj cartesian-power-car-memE cartesian-power-car-memI'' length-append mem-Collect-eq*)

definition *diag-def-poly* :: *nat* ⇒ *nat* ⇒('*a*, *nat*) *mvar-poly* **where**
diag-def-poly n i = *pvar R i* ⊖$_{coord\text{-}ring\ R\ (n\ +\ n)}$ *pvar R* (*i* + *n*)

lemma *diag-def-poly-closed*:
assumes *i* < *n*
shows *diag-def-poly n i* ∈ *carrier* (*coord-ring R* (*n* + *n*))
using *assms* **unfolding** *diag-def-poly-def coord-ring-def*

by (*metis* (*no-types*, *lifting*) *MP.minus-closed add.assoc add-leD1 coord-ring-def less-add-eq-less local.pvar-closed nat-less-le not-add-less1*)

lemma *diag-def-poly-eval*:
assumes $i < n$
assumes $x \in$ *carrier* (R^{n+n})
shows *eval-at-point R x* (*diag-def-poly n i*) $= (x!i) \ominus (x!(i + n))$

using *assms diag-def-poly-def*[*of n i*]
eval-at-point-subtract[*of x n + n pvar R i pvar R* $(i + n)$] *eval-pvar*[*of i n + n*]
eval-pvar[*of i*+*n n + n*] *pvar-closed*[*of i n + n*] *pvar-closed*[*of i + n n + n*]
by (*metis add-less-cancel-right trans-less-add2*)

definition *diag-def-poly-set* :: *nat* ⇒ ('*a*, *nat*) *mvar-poly set* **where**
diag-def-poly-set n = *diag-def-poly n* ' {..<*n*}

lemma *diag-def-poly-set-in-coord-ring*:
shows *diag-def-poly-set n* ⊆ *carrier* (*coord-ring R* $(n + n)$)
proof fix *x*
assume $x \in$ *diag-def-poly-set n*
then obtain *i* **where** *i-def*: $i < n \wedge x =$ *diag-def-poly n i*
unfolding *diag-def-poly-set-def*
by *blast*
then show $x \in$ *carrier* (*coord-ring R* $(n + n)$)
using *diag-def-poly-closed*
by *blast*
qed

lemma *diag-def-poly-set-finite*:
finite (*diag-def-poly-set n*)
unfolding *diag-def-poly-set-def*
by *blast*

lemma *diag-def-poly-eval-at-diagonal*:
assumes $x \in$ *diagonal n*
assumes $i < n$
shows *eval-at-point R x* (*diag-def-poly n i*) $= \mathbf{0}$
proof−
have $x!i = x!(i + n)$
using *assms diagonalE*(*4*) **by** *blast*
then show *?thesis*
by (*metis assms*(*1*) *assms*(*2*) *cartesian-power-car-memE cartesian-power-car-memE*′ *cring-coord-rings.diag-def-poly-eval cring-coord-rings-axioms diagonalE*(*2*) *point-to-polysE point-to-polysE*′ *pvar-trans-eval trans-less-add2*)
qed

lemma *diagonal-as-affine-alg-set*:
shows *diagonal n* = *affine-alg-set R* $(n + n)$ (*diag-def-poly-set n*)

```
proof
  show diagonal n ⊆ affine-alg-set R (n + n) (diag-def-poly-set n)
  proof fix x assume A: x ∈ diagonal n
    show  x ∈ affine-alg-set R (n + n) (diag-def-poly-set n)
      apply(rule affine-alg-set-memI)
      using A  diagonalE(2) apply blast
      using diag-def-poly-eval-at-diagonal[of x] diag-def-poly-set-def[of n]
          atLeastAtMost-iff[of - 0 n−1]
      by (metis (no-types, lifting) A image-iff lessThan-iff)
  qed
  show affine-alg-set R (n + n) (diag-def-poly-set n) ⊆ diagonal n
  proof fix x
    assume A: x ∈ affine-alg-set R (n + n) (diag-def-poly-set n)
    show x ∈ diagonal n
    proof(rule diagonalI)
      show x = take n x @ take n x
      proof−
        have 0: x = take n x @ drop n x
          by (metis append-take-drop-id)
        have take n x = drop n x
        proof−
          have 0: length x = n + n
            using A unfolding affine-alg-set-def
            using cartesian-power-car-memE by blast
          then have 1: length (take n x) = length (drop n x)
            using A
            by (metis (no-types, lifting) ‹x = take n x @ drop n x›
           add.commute add-right-cancel affine-alg-set-closed cartesian-power-car-memE

              le-add1 length-append subsetD take-closed)
          have ⋀i::nat. i < n ⟹ (take n x)!i = (drop n x) ! i
          proof− fix i::nat assume A0: i < n
            then have i ∈ {..<n} using atLeastAtMost-iff[of i 0 n−1]
              by auto
            then have diag-def-poly n i ∈  (diag-def-poly-set n)
              using diag-def-poly-set-def by blast
            then have eval-at-point R x (diag-def-poly n i) = 0
              using A affine-alg-set-memE by blast
            then have x!i = x!(n + i)
              using A0 diag-def-poly-eval[of i n x]
              by (metis (no-types, lifting) A add.commute affine-alg-set-closed
               cartesian-power-car-memE′ nat-add-left-cancel-less R.r-right-minus-eq
subsetD trans-less-add2)
            then show take n x ! i =drop n x ! i
              by (metis 0 A0 le-add1 nth-drop nth-take)
          qed
          then show ?thesis using 0
            by (metis 1 ‹x = take n x @ drop n x› add-less-mono
               length-append less-not-refl linorder-neqE-nat nth-equalityI)
```

```
      qed
      then show ?thesis
        using 0 by metis
    qed
    show take n x ∈ carrier (R^n)
      using A unfolding affine-alg-set-def
      by (meson A affine-alg-set-closed le-add2 subset-eq take-closed)
  qed
 qed
qed

lemma diagonal-is-algebraic:
 shows is-algebraic R (n + n) (diagonal n)
 apply(rule is-algebraicI[of diag-def-poly-set n])
 apply (simp add: diag-def-poly-set-finite)
 using  diag-def-poly-set-in-coord-ring apply blast
 by (simp add:  diagonal-as-affine-alg-set)

end
```

6.2 Tuples of Functions

```
definition is-function-tuple :: ('a, 'b) ring-scheme ⇒ nat ⇒ ('a list ⇒ 'a) list ⇒
bool where
is-function-tuple R n fs = (set fs ⊆ carrier (R^n) → carrier R)

lemma is-function-tupleI:
 assumes (set fs ⊆ carrier (R^n) → carrier R)
 shows is-function-tuple R n fs
 by (simp add: assms is-function-tuple-def)

lemma is-function-tuple-append:
 assumes is-function-tuple R n fs
 assumes is-function-tuple R n gs
 shows is-function-tuple R n (fs@gs)
 using assms is-function-tupleI set-append
 by (simp add: is-function-tuple-def)

lemma is-function-tuple-Cons:
 assumes is-function-tuple R n fs
 assumes f ∈ carrier (R^n) → carrier R
 shows is-function-tuple R n (f#fs)
 using assms is-function-tupleI
 by (simp add: assms(2) is-function-tuple-def)

lemma is-function-tuple-snoc:
 assumes is-function-tuple R n fs
 assumes f ∈ carrier (R^n) → carrier R
 shows is-function-tuple R n (fs@[f])
```

apply(*rule is-function-tupleI*)
by (*metis* (*no-types*) *Un-subset-iff append-Nil assms*(*1*) *assms*(*2*) *is-function-tuple-Cons is-function-tuple-def set-append*)

lemma *is-function-tuple-list-update*:
assumes *is-function-tuple R n fs*
assumes $f \in$ *carrier* $(R^n) \to$ *carrier R*
assumes $i < n$
shows *is-function-tuple R n* $(fs[i := f])$
apply(*rule is-function-tupleI*)
by (*metis assms*(*1*) *assms*(*2*) *is-function-tuple-def set-update-subsetI*)

definition *function-tuple-eval* :: $'b \Rightarrow 'c \Rightarrow ('d \Rightarrow 'a)$ *list* $\Rightarrow 'd \Rightarrow 'a$ *list* **where** *function-tuple-eval R n fs x* = *map* $(\lambda f.\ f\ x)$ *fs*

lemma *function-tuple-eval-closed*:
assumes *is-function-tuple R n fs*
assumes $x \in$ *carrier* (R^n)
shows *function-tuple-eval R n fs x* $\in$ *carrier* $(R^{length\ fs})$
apply(*rule cartesian-power-car-memI'*)
apply (*metis function-tuple-eval-def length-map*)
proof− **fix** *i* **assume** $i <$ *length fs*
then show *function-tuple-eval R n fs x* ! $i \in$ *carrier R*
unfolding *function-tuple-eval-def* **using** *assms* **unfolding** *is-function-tuple-def*

by (*metis funcset-carrier nth-map nth-mem subsetD*)
qed

definition *coord-fun* ::
$('a, 'c)$ *ring-scheme* $\Rightarrow$ *nat* $\Rightarrow$ $('a$ *list* $\Rightarrow 'b$ *list*$) \Rightarrow$ *nat* $\Rightarrow 'a$ *list* $\Rightarrow 'b$ **where** *coord-fun R n g i* = $(\lambda x \in$ *carrier* $(R^n).\ (g\ x)$! $i)$

lemma(**in** *cring*) *map-is-coord-fun-tuple*:
assumes $g \in$ *carrier* $(R^n) \to_E$ *carrier* (R^m)
shows $g = (\lambda\ x \in$ *carrier* $(R^n).$ *function-tuple-eval R n* (*map* (*coord-fun R n g*) $[0..<m]$) $x)$
proof
fix *x*
show *g x* = *restrict* (*function-tuple-eval R n* (*map* (*coord-fun R n g*) $[0..<m]$)) (*carrier* (R^n)) *x*
proof(*cases x* $\in$ *carrier* (R^n))
case *True*
then have *T0*: *restrict* (*function-tuple-eval R n* (*map* (*coord-fun R n g*) $[0..<m]$)) (*carrier* (R^n)) *x* =
function-tuple-eval R n (*map* (*coord-fun R n g*) $[0..<m]$) *x*
by (*meson restrict-apply'*)
have *T1*: *length* $(g\ x) = m$
by (*metis PiE-mem True assms cartesian-power-car-memE*)
have *T2*: $\bigwedge i.\ i < m \Longrightarrow (g\ x)$! $i =$ (*function-tuple-eval R n* (*map* (*coord-fun*

R n g) $[0..<m]$) x) ! i
unfolding *function-tuple-eval-def coord-fun-def*
using *restrict-apply True T1 length-map map-nth nth-map* **by** *smt*
have *T3*: *length* (*function-tuple-eval R n* (*map* (*coord-fun R n g*) $[0..<m]$) x)
$= m$
unfolding *function-tuple-eval-def* **using** *length-map*
by (*metis T1 map-nth*)
show *?thesis* **using** *T1 T2 T3*
by (*metis T0 nth-equalityI*)
next
case *False*
then show *?thesis* **using** *assms* **unfolding** *restrict-def*
by (*meson PiE-E*)
qed
qed

definition *function-tuple-comp* ::
$'c \Rightarrow ('a \Rightarrow 'd)$ *list* $\Rightarrow$ ($'d$ *list* $\Rightarrow$ $'b$) $\Rightarrow$ $'a \Rightarrow 'b$ **where**
function-tuple-comp R fs f = $f \circ$ (*function-tuple-eval R* (*0*::*nat*) *fs*)

lemma *function-tuple-comp-closed*:
assumes $f \in$ *carrier* $(R^n) \to$ *carrier R*
assumes *length fs* = n
assumes *is-function-tuple R m fs*
shows *function-tuple-comp R fs f* $\in$ *carrier* $(R^m) \to$ *carrier R*
unfolding *function-tuple-comp-def*
using *assms*
by (*smt Pi-iff comp-apply function-tuple-eval-closed function-tuple-eval-def*)

fun *id-function-tuple* **where**
id-function-tuple (*R*::($'a$,$'b$) *partial-object-scheme*) *0* = []|
id-function-tuple R (*Suc n*) = *id-function-tuple R n* @ [(λ(x::$'a$ *list*). x! n)]

lemma *id-function-tuple-is-function-tuple*:
$\bigwedge k.\ k \geq n \Longrightarrow$ *is-function-tuple R k* (*id-function-tuple R n*)
apply(*induction n*)
apply (*simp add*: *is-function-tupleI*)
proof− **fix** n k
assume *IH*: ($\bigwedge k.\ n \leq k \Longrightarrow$ *is-function-tuple R k* (*id-function-tuple R n*))

assume *A*: *Suc n* $\leq k$
have *0*: ($\lambda a.\ a!n$) $\in$ *carrier* $(R^k) \to$ *carrier R*
using *A cartesian-power-car-memE′*
by (*metis Pi-I Suc-le-lessD*)
have *1*: *is-function-tuple R k* (*id-function-tuple R n*)
using *A IH Suc-leD* **by** *blast*
then show *is-function-tuple R k* (*id-function-tuple R* (*Suc n*))
using *A 0 id-function-tuple.simps*(*2*)[*of R n*]
is-function-tuple-snoc[*of R k id-function-tuple R n* $\lambda a.\ a!n$]

by (*simp add*: *0*)
qed

lemma *id-function-tuple-is-function-tuple'*:
is-function-tuple R n (*id-function-tuple R n*)
by (*simp add*: *id-function-tuple-is-function-tuple*)

lemma *id-function-tuple-eval-is-take*:
assumes $a \in$ *carrier* (R^n)
shows $k \leq n \Longrightarrow$ *function-tuple-eval R n* (*id-function-tuple R k*) *a* = *take k a*
apply(*induction k*)
using *assms*
apply (*simp add*: *assms function-tuple-eval-def*)
proof− **fix** *k*
assume *IH*: ($k \leq$ $n \Longrightarrow$ *function-tuple-eval R n* (*id-function-tuple R k*) *a* = *take k a*)
assume *A*: *Suc k* ≤ *n*
then have *0*: *function-tuple-eval R n* (*id-function-tuple R k*) *a* = *take k a*
using *IH Suc-leD*
by *blast*
have *function-tuple-eval R n* (*id-function-tuple R* (*Suc k*)) *a*
= *function-tuple-eval R n* (*id-function-tuple R k*) *a* @ [*a*!*k*]
using *id-function-tuple.simps*(*2*)[*of R k*]
by (*simp add*: *function-tuple-eval-def*)
then show *function-tuple-eval R n* (*id-function-tuple R* (*Suc k*)) *a* = *take* (*Suc k*) *a*
by (*metis 0 A Suc-le-lessD assms cartesian-power-car-memE take-Suc-conv-app-nth*)
qed

lemma *id-function-tuple-eval-is-id*:
assumes $a \in$ *carrier* (R^n)
shows *function-tuple-eval R n* (*id-function-tuple R n*) *a* = *a*
using *assms id-function-tuple-eval-is-take*[*of a R n n*]
by (*metis cartesian-power-car-memE order-refl take-all*)

Composing a function tuple with a polynomial

definition *poly-function-tuple-comp* ::
('*a*, '*b*) *ring-scheme* ⇒ *nat* ⇒ ('*a list* ⇒ '*a*) *list* ⇒ ('*a*, *nat*) *mvar-poly* ⇒ '*a list* ⇒ '*a* **where**
poly-function-tuple-comp R n fs f = *eval-at-poly R f* ∘ *function-tuple-eval R n fs*

context *cring-coord-rings*
begin

lemma *poly-function-tuple-comp-closed*:
assumes *is-function-tuple R n fs*
assumes *f* ∈ *carrier* (*coord-ring R* (*length fs*))
shows *poly-function-tuple-comp R n fs f* ∈ *carrier* (R^n) → *carrier R*
proof fix *x* **assume** *A*: $x \in$ *carrier* (R^n)

then show *poly-function-tuple-comp R n fs f x ∈ carrier R*
using *assms function-tuple-eval-closed eval-at-point-closed*
unfolding *poly-function-tuple-comp-def*
by (*metis comp-apply*)
qed

lemma *poly-function-tuple-comp-eq*:
assumes *is-function-tuple R n fs*
assumes *f ∈ carrier* (*coord-ring R* (*length fs*))
assumes *a ∈ carrier* (R^n)
shows *poly-function-tuple-comp R n fs f a = eval-at-poly R f* (*function-tuple-eval R n fs a*)
unfolding *poly-function-tuple-comp-def*
using *comp-apply*
by *metis*

lemma *poly-function-tuple-comp-constant*:
assumes *is-function-tuple R n fs*
assumes *a ∈ carrier R*
assumes *x ∈ carrier* (R^n)
shows *poly-function-tuple-comp R n fs* (*coord-const a*) *x = a*
unfolding *poly-function-tuple-comp-def*
using *assms comp-apply function-tuple-eval-closed*
by (*metis eval-at-point-const*)

lemma *poly-function-tuple-comp-add*:
assumes *is-function-tuple R n fs*
assumes *k* ≤*length fs*
assumes *p ∈ carrier* (*coord-ring R k*)
assumes *Q ∈ carrier* (*coord-ring R k*)
assumes *x ∈ carrier* (R^n)
shows *poly-function-tuple-comp R n fs* ($p \oplus_{R[\mathcal{X}_n]} Q$) *x* =
(*poly-function-tuple-comp R n fs p x*) ⊕ (*poly-function-tuple-comp R n fs Q x*)
proof−
have *0*: *p ∈ carrier* (*coord-ring R* (*length fs*))
using *assms poly-ring-car-mono*[*of k length fs*]
by *blast*
have *1*: *Q ∈ carrier* (*coord-ring R* (*length fs*))
using *assms poly-ring-car-mono*[*of k length fs*]
by *blast*
show *?thesis*
using *assms*(*1*) *assms*(*5*) *0 1 R.Pring-add-eq*[*of*]
poly-function-tuple-comp-eq
function-tuple-eval-closed[*of R n fs x*]
eval-at-point-add[*of function-tuple-eval R n fs x length fs p Q*]
unfolding *coord-ring-def* **by** (*metis R.Pring-add-closed*)
qed

lemma *poly-function-tuple-comp-mult*:
assumes *is-function-tuple R n fs*
assumes $k \leq$*length fs*
assumes $p \in$ *carrier* (*coord-ring R k*)
assumes $Q \in$ *carrier* (*coord-ring R k*)
assumes $x \in$ *carrier* (R^n)
shows *poly-function-tuple-comp R n fs* ($p \otimes_{R[\mathcal{X}_n]} Q$) x =
(*poly-function-tuple-comp R n fs p x*) $\otimes$ (*poly-function-tuple-comp R n fs Q x*)
proof−
have *0*: $p \in$ *carrier* (*coord-ring R* (*length fs*))
using *assms poly-ring-car-mono*[*of k length fs*]
by *blast*
have *1*: $Q \in$ *carrier* (*coord-ring R* (*length fs*))
using *assms poly-ring-car-mono*[*of k length fs*]
by *blast*
show *?thesis*
using *assms 0 1*
poly-function-tuple-comp-eq
function-tuple-eval-closed[*of R n fs x*]
eval-at-point-mult[*of function-tuple-eval R n fs x length fs p Q*]
unfolding *coord-ring-def*
by (*metis MP.m-closed R.Pring-mult-eq coord-ring-def*)
qed

lemma *poly-function-tuple-comp-pvar*:
assumes *is-function-tuple R n fs*
assumes $k <$ *length fs*
assumes $x \in$ *carrier* (R^n)
shows *poly-function-tuple-comp R n fs* (*pvar R k*) x = (*fs* ! *k*) x
proof−
have (*map* ($\lambda f.\ f\ x$) *fs*) $\in$ *carrier* ($R^{length\ fs}$)
using *function-tuple-eval-closed*[*of R n fs x*]
unfolding *function-tuple-eval-def*
using *assms*(*1*) *assms*(*3*) **by** *blast*
then have *eval-at-poly R* (*pvar R k*) (*map* ($\lambda f.\ f\ x$) *fs*) = (*fs*! *k*) x
using *eval-pvar*[*of k length fs* (*map* ($\lambda f.\ f\ x$) *fs*)]
by (*metis assms*(*2*) *nth-map*)
then show *?thesis*
by (*metis* (*mono-tags*, *lifting*) *assms*(*1*) *assms*(*2*) *assms*(*3*) *function-tuple-eval-def*

nth-map poly-function-tuple-comp-eq pvar-closed)
qed

end

The coordinate ring of polynomials indexed by natural numbers

definition *Coord-ring* :: ('*a*, '*b*) *ring-scheme* ⇒ ('*a*, ('*a*, *nat*) *mvar-poly*) *module*
where

Coord-ring R = Pring R (UNIV :: nat set)

Some general closure lemmas for coordinate rings

context *cring-coord-rings*
begin
lemma *coord-ring-monom-term-closed*:
assumes $a \in$ *carrier* $(R[\mathcal{X}_n])$
assumes $b \in$ *carrier* $(R[\mathcal{X}_n])$
shows $a \otimes_{(R[\mathcal{X}_n])} b[\hat{}]_{(R[\mathcal{X}_n])}(n::nat) \in$ *carrier* $(R[\mathcal{X}_n])$
using *assms monoid.nat-pow-closed*[*of* $(R[\mathcal{X}_n])$]
unfolding *coord-ring-def*
by (*meson R.Pring-is-monoid monoid.m-closed*)

lemma *coord-ring-monom-term-plus-closed*:
assumes $a \in$ *carrier* $(R[\mathcal{X}_n])$
assumes $b \in$ *carrier* $(R[\mathcal{X}_n])$
assumes $c \in$ *carrier* $(R[\mathcal{X}_n])$
shows $c \oplus_{(R[\mathcal{X}_n])} a \otimes_{(R[\mathcal{X}_n])} b[\hat{}]_{(R[\mathcal{X}_n])}(n::nat) \in$ *carrier* $(R[\mathcal{X}_n])$
using *assms coord-ring-monom-term-closed R.Pring-add-closed*
by *blast*

end

6.3 Generic Univariate Polynomials

By a generic univariate polynomial, we mean a polynomial in one variable whose coefficients are coordinate functions over a ring. That is, a polynomial of the form:

$$f(t) = x_0 + x_1 t + \cdots + x_n t^n$$

Such a polynomial can be construed as an element of $R[x_0, .., x_n](t)$, or as an element of $R[x_0, .., x_n, x_n n + 1]$. We will intially define the latter version, and show that it can easily be cast to the former using the function "`IP_to_UP`". Such a polynomial can be cast to a univariate polynomial over the ring R by substituting a tuple of ring elements for the coefficients.

definition *generic-poly-lt* :: $('a, 'b)$ *ring-scheme* $\Rightarrow$ *nat* $\Rightarrow$ $('a, nat)$ *mvar-poly*
where
generic-poly-lt R n = (*pvar R* (*Suc n*)) $\otimes_{coord\text{-}ring\ R\ (Suc\ (Suc\ n))}$ (*pvar R 0*)$[\hat{}]_{coord\text{-}ring\ R\ (Suc\ (Suc\ n))}$ *n*

fun *generic-poly* **where**
generic-poly R (*0*::*nat*) = *pvar R 1* |
generic-poly R (*Suc n*) = (*generic-poly R n*) $\oplus_{(coord\text{-}ring\ R\ (n+3))}$ *generic-poly-lt R* (*Suc n*)

context *cring-coord-rings*

begin

lemma *generic-poly-lt-closed*:
generic-poly-lt R n ∈ *carrier* (*coord-ring R* (*Suc* (*Suc n*)))
proof−
have *0*: (*pvar R* (*Suc n*)) ∈ *carrier* (*coord-ring R* (*Suc* (*Suc n*)))
using *pvar-closed*
by *blast*
have *1*: (*pvar R 0*) ∈ *carrier* (*coord-ring R* (*Suc* (*Suc n*)))
using *pvar-closed*
by *blast*
then have (*pvar R 0*)[^]$_{\textit{coord-ring R (Suc (Suc n))}}$ *n* ∈ *carrier* (*coord-ring R* (*Suc* (*Suc n*)))
using *monoid.nat-pow-closed*
unfolding *coord-ring-def* **by** (*metis R.Pring-is-monoid*)
then show *?thesis* **using** *0*
unfolding *coord-ring-def*
by (*metis R.Pring-mult-closed coord-ring-def generic-poly-lt-def*)
qed

lemma *generic-poly-lt-eval*:
assumes *a* ∈ *carrier* (R^{n+2})
shows *eval-at-point R a* (*generic-poly-lt R n*) = *a*!(*Suc n*) ⊗ (*a*!*0*)[^]*n*
proof−
have (*pvar R 0* [^]$_{\textit{coord-ring R (Suc (Suc n))}}$ *n*) ∈ *carrier* (*coord-ring R* (*n* + *2*))
using *monoid.nat-pow-closed pvar-closed* **unfolding** *coord-ring-def*
by (*metis R.Pring-is-monoid add-2-eq-Suc′ zero-less-Suc*)
then have *eval-at-point R a* (*generic-poly-lt R n*) =
eval-at-poly R (*pvar R* (*Suc n*)) *a* ⊗ *eval-at-poly R* (*pvar R 0* [^]$_{\textit{coord-ring R (Suc (Suc n))}}$ *n*) *a*
unfolding *generic-poly-lt-def*
using *assms pvar-closed*[*of* (*Suc n*) *n* + *2*] *eval-at-point-mult*[*of a n* + *2 pvar R* (*Suc n*) (*pvar R 0*)[^]$_{\textit{coord-ring R (Suc (Suc n))}}$ *n*]
by (*metis add-2-eq-Suc′ lessI*)
then show *?thesis* **using** *assms*
by (*metis add-2-eq-Suc′ add-gr-0 eval-at-point-nat-pow eval-pvar lessI pvar-closed zero-less-numeral*)
qed

lemma *generic-poly-closed*:
generic-poly R n ∈ *carrier* (*coord-ring R* (*Suc* (*Suc n*)))
apply(*induction n*)
using *pvar-closed*[*of 1 Suc* (*Suc n*)]
apply (*metis One-nat-def generic-poly.simps*(*1*) *lessI pvar-closed*)
proof−
fix *n* **assume** *IH*: *generic-poly R n* ∈ *carrier* (*coord-ring R* (*Suc* (*Suc n*)))
have *generic-poly R n* ∈ *carrier* (*coord-ring R* (*Suc* (*Suc* (*Suc n*))))
using *IH poly-ring-car-mono′*[*of Suc* (*Suc n*)]
by *blast*

then show *generic-poly R (Suc n) ∈ carrier (coord-ring R (Suc (Suc (Suc n))))*
unfolding *coord-ring-def*
using *generic-poly.simps[of R] generic-poly-lt-closed[of n]*
by (*metis MP.add.m-closed R.Pring-add-eq coord-ring-def generic-poly-lt-closed*)
qed

lemma *generic-poly-closed′*:
assumes $k \leq n$
shows *generic-poly R k ∈ carrier (coord-ring R (Suc (Suc n)))*
by (*meson Suc-le-mono assms generic-poly-closed poly-ring-car-mono subsetD*)

lemma *generic-poly-eval-at-point*:
assumes $a \in carrier\ (R^{n+3})$
shows *eval-at-point R a (generic-poly R (Suc n)) = (eval-at-point R a (generic-poly R n)) ⊕*
$(a!(n+2)) \otimes (a!0)[\hat{}](Suc\ n)$
proof−
have *0*: *(generic-poly R n) ∈ carrier (coord-ring R (n + 3))*
using *generic-poly-closed′*
by (*metis Suc3-eq-add-3 add.commute eq-imp-le le-SucI*)
then show *?thesis*
using *generic-poly.simps(2)*
generic-poly-closed′[of n n + 3]
generic-poly-lt-eval eval-at-point-add[of a (n + 3) generic-poly R n]
by (*metis* (*no-types*, *lifting*) *add.left-commute add-2-eq-Suc′ assms*
generic-poly-lt-closed numeral-2-eq-2 numeral-3-eq-3 plus-1-eq-Suc)
qed

end

We can turn points in R^{n+1} into univariate polynomials with the associated coefficients via partial evaluation of the generic polynomials of degree n.

definition *ring-cfs-to-poly* ::
('a, 'b) ring-scheme ⇒ nat ⇒ 'a list ⇒ ('a, nat) mvar-poly **where**
ring-cfs-to-poly R n as = coord-partial-eval R {1..<n+2} (0$_R$*#as) (generic-poly R n)*

context *cring-coord-rings*
begin

lemma *ring-cfs-to-poly-closed*:
assumes $as \in carrier\ (R^{Suc\ n})$
shows *ring-cfs-to-poly R n as ∈ carrier (coord-ring R 1)*
proof−
have *0*: $\mathbf{0}\ \#\ as \in carrier\ (R^{n+2})$
apply(*rule cartesian-power-car-memI*)
using *assms*
apply (*metis add-2-eq-Suc′ cartesian-power-car-memE length-Cons*)

using *assms*
by (*metis cartesian-power-car-memE'' insert-subset list.simps*(15) *R.zero-closed*)
then have *1*: *coord-partial-eval R* {*1*..<*n* + *2*} (**0** # *as*) ∈ *ring-hom* (*coord-ring R* (*n* + *2*)) (*Pring R* ({..<*n* + *2*} − {*1*..<*n* + *2*}))
using *coord-partial-eval-hom'* **by** *blast*
have ({..<*n* + *2*} − {*1*..<*n* + *2*}) = {..<*1*}
by *auto*
then have *2*: *coord-partial-eval R* {*1*..<*n* + *2*} (**0** # *as*) ∈ *ring-hom* (*coord-ring R* (*n* + *2*)) (*coord-ring R 1*)
using *1* **unfolding** *coord-ring-def*
by *presburger*
then show *?thesis*
unfolding *ring-cfs-to-poly-def coord-ring-def*
by (*metis 0 Diff-subset* ‹{..<*n* + *2*} − {*1*..<*n* + *2*} = {..<*1*}›
add-2-eq-Suc' coord-partial-eval-closed generic-poly-closed
le-numeral-extra(*4*) *lessThan-minus-lessThan lessThan-subset-iff*)
qed

Variant which maps to the univariate polynomial ring

definition *ring-cfs-to-univ-poly* :: *nat* ⇒ *'a list* ⇒ *nat* ⇒ *'a* **where**
ring-cfs-to-univ-poly n as = *IP-to-UP* (*0*::*nat*) (*ring-cfs-to-poly R n as*)

lemma *ring-cfs-to-univ-poly-closed*:
assumes *as* ∈ *carrier* ($R^{Suc\ n}$)
shows *ring-cfs-to-univ-poly n as* ∈ *carrier* (*UP R*)
unfolding *ring-cfs-to-univ-poly-def* **apply**(*rule R.IP-to-UP-closed*, *rule R.is-cring*)
using *ring-cfs-to-poly-closed* **unfolding** *coord-ring-def*
using *assms* **by** (*metis One-nat-def lessThan-0 lessThan-Suc*)

lemma *ring-cfs-to-poly-eq*:
assumes *as* ∈ *carrier* ($R^{Suc\ n}$)
assumes *k* ≤*n*
shows *ring-cfs-to-poly R k as* = *ring-cfs-to-poly R k* (*take* (*Suc k*) *as*)
unfolding *ring-cfs-to-poly-def coord-partial-eval-def*
apply(*rule R.poly-eval-eval-function-eq*[*of* (*point-to-eval-map R* (**0** # *as*)) (*point-to-eval-map R* (**0** # *take* (*Suc k*) *as*)) {*1*..<*k* + *2*} - {..<*k* + *2*}])
proof−
show *closed-fun R* (*point-to-eval-map R* (**0** # *as*))
apply(*rule R.closed-funI*)
using *assms cartesian-power-car-memE*[*of as R Suc n*]
by (*metis cartesian-power-car-memE'' nth-mem set-ConsD subset-code*(*1*) *R.zero-closed*)
show *closed-fun R* (λ*i*. *if i* < *length* (**0** # *take* (*Suc k*) *as*) *then* (**0** # *take* (*Suc k*) *as*) ! *i else* **0**)
apply(*rule R.closed-funI*)
using *assms*
by (*metis cartesian-power-car-memE'' in-set-takeD nth-mem set-ConsD subset-code*(*1*) *R.zero-closed*)
have *0*: *length* (**0** # *as*) ≥ *k* + *2*
using *assms*

by (*metis Suc-le-mono add-2-eq-Suc′ cartesian-power-car-memE length-Cons*)
have *1*: *length* (**0** # *take* (*Suc k*) *as*) ≥*k* + *2*
using *0*
by (*metis add-2-eq-Suc′ assms*(*1*) *cartesian-power-car-memE*
impossible-Cons length-Cons not-less-eq-eq take-closed)
show *restrict* (*point-to-eval-map R* (**0** # *as*)) {*1*..<*k* + *2*} = *restrict* (*point-to-eval-map R* (**0** # *take* (*Suc k*) *as*)) {*1*..<*k* + *2*}
proof fix *x*
show *restrict* (*point-to-eval-map R* (**0** # *as*)) {*1*..<*k* + *2*} *x* = *restrict* (*point-to-eval-map R* (**0** # *take* (*Suc k*) *as*)) {*1*..<*k* + *2*} *x*
proof(*cases x* ∈ {*1*..<*k* + *2*})
case *True*
have *00*: *restrict* (*point-to-eval-map R* (**0** # *as*)) {*1*..<*k* + *2*} *x* = (**0**#*as*)!*x*
unfolding *restrict-def*
by (*metis 0 True atLeastLessThan-iff le-Suc-ex trans-less-add1*)
have *11*: *restrict* (*point-to-eval-map R* (**0** # *take* (*Suc k*) *as*)) {*1*..<*k* + *2*} *x* = (**0** # *take* (*Suc k*) *as*)!*x*
unfolding *restrict-def*
by (*metis 1 True atLeastLessThan-iff le-Suc-ex trans-less-add1*)
have *2*: (**0** # *as*) ! *x* = (**0** # *take* (*Suc k*) *as*) ! *x*
proof−
obtain *l* **where** *l-def*: *Suc l* = *x*
using *True*
by (*metis One-nat-def Suc-le-D atLeastLessThan-iff*)
have *P1*: (**0** # *as*) ! *x* = *as* ! *l*
using *0 True l-def*
by (*meson nth-Cons-Suc*)
have *P0*: (**0** # *take* (*Suc k*) *as*) ! *x* = (*take* (*Suc k*) *as*) ! *l*
using *1 True l-def*
by (*meson nth-Cons-Suc*)
have *l* < *Suc k*
using *True l-def*
by (*metis Suc-1 Suc-eq-plus1 Suc-less-SucD add-Suc-right atLeastLessThan-iff*)
then have (**0** # *take* (*Suc k*) *as*) ! *x* = *as* ! *l*
using *P0*
by (*metis nth-take*)
then show *?thesis*
using *P1* **by** *metis*
qed
then show *?thesis* **using** *00 11 True*
by *presburger*
next
case *False*
then show *?thesis*
unfolding *restrict-def*
by *presburger*
qed
qed
show *generic-poly R k* ∈ *Pring-set R* {..<*k* + *2*}

by (*metis R.Pring-car add-2-eq-Suc′ coord-ring-def generic-poly-closed*)
qed

lemma *coord-partial-eval-generic-poly-lt*:
assumes *as* ∈ *carrier* ($R^{Suc\ n}$)
shows *coord-partial-eval R* {*1..<n+2*} (**0**$_R$#*as*) (*generic-poly-lt R n*) =
poly-scalar-mult R (*as*!*n*) ((*pvar R 0*)[^]$_{coord\text{-}ring\ R\ (n+2)}$ *n*)
proof−
have *0*: **0** # *as* ∈ *carrier* ($R^{Suc\ (Suc\ n)}$)
using *assms cartesian-power-cons*
by (*metis Suc-eq-plus1 R.zero-closed*)
have *1*: *pvar R* (*Suc n*) ∈ *Pring-set R* {*..<n* + *2*}
using *pvar-closed*
by (*metis R.Pring-car add-2-eq-Suc′ coord-ring-def lessI*)
have *2*: *pvar R 0* [^]$_{coord\text{-}ring\ R\ (Suc\ (Suc\ n))}$ *n* ∈ *Pring-set R* {*..<n* + *2*}
using *monoid.nat-pow-closed pvar-closed* **unfolding** *coord-ring-def*
using *R.Pring-car R.Pring-is-monoid add-2-eq-Suc′ zero-less-Suc*
by (*metis*)
have *3*: *poly-eval R* {*1..<n* + *2*} (*point-to-eval-map R* (**0** # *as*))
(*pvar R* (*Suc n*) ⊗$_{coord\text{-}ring\ R\ (Suc\ (Suc\ n))}$ *pvar R 0* [^]$_{coord\text{-}ring\ R\ (Suc\ (Suc\ n))}$
n) =
(*poly-eval R* {*1..<n* + *2*} (*point-to-eval-map R* (**0** # *as*)) (*pvar R* (*Suc n*)))
⊗$_{coord\text{-}ring\ R\ (Suc\ (Suc\ n))}$
(*poly-eval R* {*1..<n* + *2*} (*point-to-eval-map R* (**0** # *as*))
(*pvar R 0* [^]$_{coord\text{-}ring\ R\ (Suc\ (Suc\ n))}$ *n*))
using *0 1 2 R.poly-eval-mult*[*of pvar R* (*Suc n*) {*..<n+2*} *pvar R 0* [^]$_{coord\text{-}ring\ R\ (Suc\ (Suc\ n))}$
n
(*point-to-eval-map R* (**0** # *as*)) {*1..<n* + *2*}] **unfolding** *coord-ring-def*
by (*smt R.Pring-mult cartesian-power-car-memE cartesian-power-car-memE′*
R.closed-funI R.zero-closed)
have *4*: *poly-eval R* {*1..<n* + *2*} (*point-to-eval-map R* (**0** # *as*))
(*pvar R* (*Suc n*) ⊗$_{coord\text{-}ring\ R\ (Suc\ (Suc\ n))}$ *pvar R 0* [^]$_{coord\text{-}ring\ R\ (Suc\ (Suc\ n))}$
n) =
(*coord-const* ((**0** # *as*)! (*Suc n*))) ⊗$_{coord\text{-}ring\ R\ (Suc\ (Suc\ n))}$
(*poly-eval R* {*1..<n* + *2*} (*point-to-eval-map R* (**0** # *as*))
(*pvar R 0* [^]$_{coord\text{-}ring\ R\ (Suc\ (Suc\ n))}$ *n*))
using *0 3 point-to-eval-map-closed*[*of* (**0** # *as*) *Suc* (*Suc n*)]
R.poly-eval-index[*of* (*point-to-eval-map R* (**0** # *as*)) {*1..<n* + *2*} *Suc n*]
add-2-eq-Suc′ atLeastLessThan-iff cartesian-power-car-memE le-neq-implies-less

less-Suc-eq not-less-eq-eq not-less-zero numeral-1-eq-Suc-0 numeral-One
var-to-IP-def
by (*smt local.one-neq-zero*)
have *5*: *pvar R 0* [^]$_{coord\text{-}ring\ R\ (Suc\ (Suc\ n))}$ *n* ∈ *Pring-set R* ({*..<n* + *2*} −
{*1..<n* + *2*})
proof−
have *0* ∈ {*..<n* + *2*} − {*1..<n* + *2*} **by** *auto*
then have *pvar R 0* [^]$_{Pring\ R\ (\{..<n\ +\ 2\}\ -\ \{1..<n\ +\ 2\})}$ *n* ∈ *carrier* (*Pring*

```
R ({..<n + 2} - {1..<n + 2}))
    using R.Pring-var-closed[of 0 {..<n + 2} - {1..<n + 2}] R.Pring-is-monoid[of
{..<n + 2} - {1..<n + 2}]
          monoid.nat-pow-closed[of Pring R ({..<n + 2} - {1..<n + 2}) pvar R 0
n ]
      by blast
   have ⋀k::nat. (pvar R 0 [^]coord-ring R (Suc (Suc n)) k) = (pvar R 0 [^]Pring R ({..<n + 2} - {1..<n + 2})
    proof− fix k::nat show (pvar R 0 [^]coord-ring R (Suc (Suc n)) k) = (pvar R
0 [^]Pring R ({..<n + 2} - {1..<n + 2})k)
        apply(induction k)
         using R.Pring-var-closed[of 0 {..<(Suc (Suc n))}] R.Pring-var-closed[of 0
{..<n + 2} - {1..<n + 2}]
        unfolding coord-ring-def
        apply (metis Group.nat-pow-0 R.ring-axioms R.Pring-one)
         using R.Pring-var-closed[of 0 {..<(Suc (Suc n))}] R.Pring-var-closed[of 0
{..<n + 2} - {1..<n + 2}]
        nat-pow-def
        by (metis R.Pring-mult-eq R.Pring-one-eq add-2-eq-Suc')
    qed
    then show ?thesis
      by (metis R.Pring-car ‹pvar R 0 [^]Pring R ({..<n + 2} - {1..<n + 2}) n ∈
carrier (Pring R ({..<n + 2} - {1..<n + 2}))›)
  qed
  have 6: (poly-eval R {1..<n + 2} (point-to-eval-map R (0 # as))
          (pvar R 0 [^]coord-ring R (Suc (Suc n)) n)) = (pvar R 0 [^]coord-ring R (Suc (Suc n))
n)
    using 5 0 point-to-eval-map-closed[of (0 # as) Suc (Suc n)]
       R.poly-eval-trivial[of (point-to-eval-map R (0 # as)) pvar R 0 [^]coord-ring R (Suc (Suc n))
n {..<n + 2} {1..<n + 2} ]
    by blast
  have 7: poly-eval R {1..<n + 2} (point-to-eval-map R (0 # as))
    (pvar R (Suc n) ⊗coord-ring R (Suc (Suc n)) pvar R 0 [^]coord-ring R (Suc (Suc n))
n) =
  (coord-const ((0 # as)! (Suc n))) ⊗coord-ring R (Suc (Suc n))
     (pvar R 0 [^]coord-ring R (Suc (Suc n)) n)
    using 4 6
    by presburger
  have 8: (0 # as) ! Suc n = as! n
    by (meson nth-Cons-Suc)
  have 88: (0 # as) ! Suc n ∈ carrier R
    by (metis 8 assms cartesian-power-car-memE' less-Suc-eq)
  have 9: poly-eval R {1..<n + 2} (point-to-eval-map R (0 # as))
    (pvar R (Suc n) ⊗coord-ring R (Suc (Suc n)) pvar R 0 [^]coord-ring R (Suc (Suc n))
n) =
    coord-const ((0 # as) ! Suc n) ⨂p pvar R 0 [^]coord-ring R (Suc (Suc n)) n
   using R.poly-scalar-mult-eq[of (0 # as) ! Suc n pvar R 0 [^]coord-ring R (Suc (Suc n))
n]
```

unfolding *coord-ring-def*
by (*metis* (*no-types*, *lifting*) *7 R.Pring-mult coord-ring-def*)
have *10*: *poly-scalar-mult R* ((**0** # *as*) ! *Suc n*) (*pvar R 0* [^]$_{\textit{coord-ring R (Suc (Suc n))}}$ *n*) =
coord-const ((**0** # *as*) ! *Suc n*) $\bigotimes_p$ *pvar R 0* [^]$_{\textit{coord-ring R (Suc (Suc n))}}$ *n*
using *9 8 88 0 5 R.poly-scalar-mult-eq*[*of* (**0** # *as*) ! *Suc n pvar R 0* [^]$_{\textit{coord-ring R (Suc (Suc n))}}$ *n* ({..<*n* + *2*} − {*1*..<*n* + *2*})]
by *blast*
have *11*: *poly-scalar-mult R* (*as*! *n*) (*pvar R 0* [^]$_{\textit{coord-ring R (Suc (Suc n))}}$ *n*) =
coord-const ((**0** # *as*) ! *Suc n*) $\bigotimes_p$ *pvar R 0* [^]$_{\textit{coord-ring R (Suc (Suc n))}}$ *n*
using *10 8*
by *metis*
have *12*: *poly-eval R* {*1*..<*n* + *2*} (*point-to-eval-map R* (**0** # *as*))
(*pvar R* (*Suc n*) $\otimes_{\textit{coord-ring R (Suc (Suc n))}}$ *pvar R 0* [^]$_{\textit{coord-ring R (Suc (Suc n))}}$ *n*) =
poly-scalar-mult R (*as* ! *n*) ((*pvar R 0*) [^]$_{\textit{coord-ring R (n + 2)}}$ *n*)
using *11 9*
by (*metis add-2-eq-Suc'*)
then show *?thesis*
unfolding *coord-partial-eval-def generic-poly-lt-def*
by *blast*
qed

lemma *coord-partial-eval-generic-poly-lt'*:
assumes *as* ∈ *carrier* ($R^{Suc\ n}$)
shows *coord-partial-eval R* {*1*..<*n*+*2*} ($\mathbf{0}_R$#*as*) (*generic-poly-lt R n*) =
poly-scalar-mult R (*as*!*n*) ((*pvar R 0*)[^]$_{\textit{coord-ring R 1}}$ *n*)
proof−
have *0*: *coord-partial-eval R* {*1*..<*n*+*2*} ($\mathbf{0}_R$#*as*) (*generic-poly-lt R n*) =
poly-scalar-mult R (*as*!*n*) ((*pvar R 0*)[^]$_{\textit{coord-ring R (n+2)}}$ *n*)
using *assms coord-partial-eval-generic-poly-lt* **by** *blast*
have *1*: ⋀*k*::*nat*. (*pvar R 0*)[^]$_{\textit{coord-ring R (n+2)}}$ *k* = (*pvar R 0*)[^]$_{\textit{coord-ring R 1}}$ *k*
proof− **fix** *k*::*nat* **show** (*pvar R 0*)[^]$_{\textit{coord-ring R (n+2)}}$ *k* = (*pvar R 0*)[^]$_{\textit{coord-ring R 1}}$ *k*
apply(*induction k*)
unfolding *coord-ring-def*
apply (*metis Group.nat-pow-0 R.Pring-one-eq*)
using *nat-pow-def*
by (*metis R.Pring-mult-eq R.Pring-one add-2-eq-Suc'*)
qed
then show *?thesis*
using *0* **by** *presburger*
qed

lemma *ring-cfs-to-poly-decomp*:
assumes *as* ∈ *carrier* ($R^{Suc\ (Suc\ n)}$)
shows *ring-cfs-to-poly R* (*Suc n*) *as* = *ring-cfs-to-poly R n as* $\oplus_{\textit{coord-ring R 1}}$

poly-scalar-mult R (as!(Suc n)) ((pvar R 0)[^]$_{coord\text{-}ring\ R\ 1}$ (Suc n))
proof−
have *LHS*: *ring-cfs-to-poly R (Suc n) as =*
*coord-partial-eval R {1..<Suc n + 2} (**0** # as) (generic-poly R n*
$\oplus_{coord\text{-}ring\ R\ (Suc\ (Suc\ (Suc\ n)))}$ *generic-poly-lt R (Suc n))*
by (*smt add-2-eq-Suc' add-Suc-right generic-poly.simps(2) numeral-2-eq-2 numeral-3-eq-3 ring-cfs-to-poly-def*)
have *LHS'*: *ring-cfs-to-poly R (Suc n) as =*
*coord-partial-eval R {1..<Suc n + 2} (**0** # as) (generic-poly R n)*
$\oplus_{coord\text{-}ring\ R\ (Suc\ (Suc\ (Suc\ n)))}$
*coord-partial-eval R {1..<Suc n + 2} (**0** # as) (generic-poly-lt R (Suc n))*
using *coord-partial-eval-add*[*of as Suc n*]
by (*metis LHS add-2-eq-Suc' add-Suc-shift assms cartesian-power-cons coord-partial-eval-add generic-poly-closed' generic-poly-lt-closed le-add2 plus-1-eq-Suc R.zero-closed*)
have *LHS''*: *ring-cfs-to-poly R (Suc n) as =*
*coord-partial-eval R {1..<Suc n + 2} (**0** # as) (generic-poly R n)*
$\oplus_{coord\text{-}ring\ R\ (Suc\ (Suc\ (Suc\ n)))}$
*coord-partial-eval R {1..<Suc n + 2} (**0** # as) (generic-poly-lt R (Suc n))*
using *coord-partial-eval-add*[*of as Suc n*]
by (*metis LHS add-2-eq-Suc' add-Suc-shift assms cartesian-power-cons coord-partial-eval-add generic-poly-closed' generic-poly-lt-closed le-add2 plus-1-eq-Suc R.zero-closed*)
have *LHS'''*: *ring-cfs-to-poly R (Suc n) as =*
*coord-partial-eval R {1..<Suc n + 2} (**0** # as) (generic-poly R n)*
$\oplus_{coord\text{-}ring\ R\ (Suc\ (Suc\ (Suc\ n)))}$
poly-scalar-mult R (as! (Suc n)) ((pvar R 0)[^]$_{coord\text{-}ring\ R\ 1}$ (Suc n))
using *LHS'' coord-partial-eval-generic-poly-lt'*[*of as Suc n*] *assms*
by *presburger*
have *0*: *coord-partial-eval R {1..<Suc n + 2} (**0** # as) (generic-poly R n) = ring-cfs-to-poly R n as*
proof−
have *00*: *(generic-poly R n) ∈ carrier (coord-ring R (n + 2))*
using *add-2-eq-Suc' generic-poly-closed* **by** *presburger*
have *01*: **1** ≠ **0**
using *one-neq-zero*
by *presburger*
have *02*: *(**0** # as) ∈ carrier* ($R^{Suc\ (Suc\ n)\ +\ 1}$)
using *cartesian-power-cons*[*of as R Suc (Suc n) **0***] *assms*
by *blast*
have *03*: *closed-fun R (point-to-eval-map R (**0** # as))*
using *point-to-eval-map-closed*[*of* **0**#*as Suc (Suc (Suc n))*]
by (*metis 02 Suc-eq-plus1*)
have *04*: *{1..<Suc n + 2} ∩ {..<n + 2} = {1..<n + 2} ∩ {..<n + 2}*
by *auto*
show *?thesis*

unfolding *ring-cfs-to-poly-def coord-partial-eval-def*
using *04 03 02 01 00 R.Pring-car*[*of* {..<*n* + *2*}] *assms*
R.poly-eval-eval-set-eq[*of point-to-eval-map R* (**0** # *as*) {*1*..<*Suc n* + *2*}
{..<*n* + *2*} {*1*..<*n* + *2*} (*generic-poly R n*)]
by (*metis coord-ring-def*)
qed
show *?thesis*
using *generic-poly.simps*(*2*)[*of R n*] *coord-partial-eval-add LHS''' 0*
unfolding *ring-cfs-to-poly-def*
by (*metis R.Pring-add-eq coord-ring-def*)
qed

lemma *ring-cfs-to-poly-decomp'*:
assumes *as* ∈ *carrier* ($R^{Suc\ (Suc\ n)}$)
shows *ring-cfs-to-poly R* (*Suc n*) *as* =
ring-cfs-to-poly R n (*take* (*Suc n*) *as*) $\oplus_{coord\text{-}ring\ R\ 1}$
poly-scalar-mult R (*as*!(*Suc n*)) ((*pvar R 0*)[^]$_{coord\text{-}ring\ R\ 1}$ (*Suc n*))
using *assms ring-cfs-to-poly-decomp*[*of as n*]
ring-cfs-to-poly-eq[*of as Suc n n*] *le-eq-less-or-eq less-Suc-eq*
by *presburger*

lemma *ring-cfs-to-univ-poly-decomp'*:
assumes *as* ∈ *carrier* ($R^{Suc\ (Suc\ n)}$)
shows *ring-cfs-to-univ-poly* (*Suc n*) *as* =
ring-cfs-to-univ-poly n (*take* (*Suc n*) *as*) $\oplus_{UP\ R}$
(*as*!(*Suc n*))$\odot_{UP\ R}$ (*X-poly R* [^]$_{UP\ R}$ (*Suc n*))
proof−
have *00*: (*pvar R 0* [^]$_{coord\text{-}ring\ R\ 1}$ *Suc n*) ∈ *carrier* (*Pring R* {*0*})
using *pvar-closed*[*of 0 1*] *monoid.nat-pow-closed*[*of coord-ring R 1* - *n*]
unfolding *coord-ring-def*
by (*metis One-nat-def R.Pring-is-monoid lessThan-0 lessThan-Suc less-one monoid.nat-pow-closed*)
have *LHS*: *ring-cfs-to-univ-poly* (*Suc n*) *as* =
IP-to-UP 0 (*ring-cfs-to-poly R n* (*take* (*Suc n*) *as*) $\oplus_{coord\text{-}ring\ R\ 1}$
poly-scalar-mult R (*as*!(*Suc n*)) ((*pvar R 0*)[^]$_{coord\text{-}ring\ R\ 1}$ (*Suc n*)))
using *assms ring-cfs-to-poly-decomp'*
unfolding *ring-cfs-to-univ-poly-def*
by *presburger*
have *LHS'*: *ring-cfs-to-univ-poly* (*Suc n*) *as* =
IP-to-UP 0 (*ring-cfs-to-poly R n* (*take* (*Suc n*) *as*)) $\oplus_{UP\ R}$
IP-to-UP 0 (*poly-scalar-mult R* (*as*!(*Suc n*)) ((*pvar R 0*)[^]$_{coord\text{-}ring\ R\ 1}$ (*Suc n*)))
proof−
have *0*: *ring-cfs-to-poly R n* (*take* (*Suc n*) *as*) ∈ *carrier* (*Pring R* {*0*})
by (*metis One-nat-def assms coord-ring-def le-add2 lessThan-0 lessThan-Suc plus-1-eq-Suc ring-cfs-to-poly-closed take-closed*)
have *1*: *as* ! *Suc n* ∈ *carrier R*
using *assms cartesian-power-car-memE'*[*of as R Suc* (*Suc n*)]
by *blast*

have *2*: *poly-scalar-mult R* (*as* ! *Suc n*) (*pvar R 0* [^]$_{coord\text{-}ring\ R\ 1}$ *Suc n*) ∈ *carrier* (*Pring R* {*0*})
using *1 00 R.Pring-car R.poly-scalar-mult-closed*[*of* (*as* ! *Suc n*) (*pvar R 0* [^]$_{coord\text{-}ring\ R\ 1}$ *Suc n*) {*0*}]
by *blast*
then show *?thesis*
using *0 1 2 UP-cring.IP-to-UP-add*[*of R* (*ring-cfs-to-poly R n* (*take* (*Suc n*) *as*)) *0*
poly-scalar-mult R (*as*!(*Suc n*)) ((*pvar R 0*)[^]$_{coord\text{-}ring\ R\ 1}$ (*Suc n*))]
by (*metis LHS One-nat-def UP-cring-def coord-ring-def R.is-cring lessThan-0 lessThan-Suc*)
qed
have *0*: *IP-to-UP 0* (*ring-cfs-to-poly R n* (*take* (*Suc n*) *as*)) =
ring-cfs-to-univ-poly n (*take* (*Suc n*) *as*)
using *ring-cfs-to-univ-poly-def*
by *presburger*
have *1*: (*mset-to-IP R* (*nat-to-mset 0* (*Suc n*))) = (*pvar R 0*)[^]$_{coord\text{-}ring\ R\ 1}$ (*Suc n*)
unfolding *coord-ring-def* **using** *lessThan-iff less-one*
by (*metis UP-cring.intro UP-cring.pvar-pow R.is-cring*)
have *2*: *as* ! *Suc n* ∈ *carrier R*
using *cartesian-power-car-memE′ assms*
by *blast*
have *3*: *IP-to-UP 0* (*poly-scalar-mult R* (*as* ! *Suc n*) (*pvar R 0* [^]$_{coord\text{-}ring\ R\ 1}$ *Suc n*)) =
as ! *Suc n* ⊙$_{UP\ R}$ *IP-to-UP 0* (*pvar R 0* [^]$_{coord\text{-}ring\ R\ 1}$ *Suc n*)
using *UP-cring.IP-to-UP-scalar-mult*[*of R as*!(*Suc n*) ((*pvar R 0*)[^]$_{coord\text{-}ring\ R\ 1}$ (*Suc n*)) *0*]
00 2 **unfolding** *coord-ring-def*
by (*metis R.Pring-smult UP-cring.intro R.is-cring*)
have *4*: *IP-to-UP 0* (*poly-scalar-mult R* (*as*!(*Suc n*)) ((*pvar R 0*)[^]$_{coord\text{-}ring\ R\ 1}$ (*Suc n*)))
= (*as*!(*Suc n*))⊙$_{UP\ R}$ (*X-poly R* [^]$_{UP\ R}$ (*Suc n*))
proof −
have *as* ! *Suc n* ⊙$_{UP\ R}$ *X-poly R* [^]$_{UP\ R}$ *Suc n* = *IP-to-UP* (*0*::*nat*) (*Mt* (*as* ! *Suc n*) (*nat-to-mset 0* (*Suc n*)))
using *3 1 UP-cring.IP-to-UP-monom*
by (*metis UP-cring.intro R.is-cring*)
then show *?thesis*
using ‹*mset-to-IP R* (*nat-to-mset 0* (*Suc n*)) = *pvar R 0* [^]$_{coord\text{-}ring\ R\ 1}$ *Suc n*›
by *presburger*
qed
then show *?thesis*
using *0 LHS′*
by *presburger*
qed

lemma *ring-cfs-to-univ-poly-decomp*:

assumes *as* ∈ *carrier* ($R^{Suc\ n}$)
assumes $k < n$
shows *ring-cfs-to-univ-poly* (*Suc k*) (*take* (*Suc* (*Suc k*)) *as*) = *ring-cfs-to-univ-poly k* (*take* (*Suc k*) *as*)
$\oplus_{UP\ R}$ (*as*!(*Suc k*)) $\odot_{UP\ R}$ (*X-poly R*)[^]$_{UP\ R}$ (*Suc k*)
proof−
have *0*: (*take* (*Suc* (*Suc k*)) *as*) ∈ *carrier* ($R^{Suc\ (Suc\ k)}$)
using *assms*
by (*meson Suc-leI Suc-mono take-closed*)
then show *?thesis* **using** *ring-cfs-to-univ-poly-decomp'*[*of take* (*Suc* (*Suc k*)) *as k*]
by (*metis* (*no-types*, *lifting*) *Suc-leI assms*(*1*) *assms*(*2*) *cartesian-power-car-memE*
lessI less-SucI nth-take nth-take-lemma)
qed

lemma *ring-cfs-to-univ-poly-degree*:
assumes *as* ∈ *carrier* ($R^{Suc\ n}$)
shows *deg R* (*ring-cfs-to-univ-poly n as*) ≤ *n*
as!*n* ≠ **0** ⟹ *deg R* (*ring-cfs-to-univ-poly n as*) = *n*
proof−
have *0*:⋀*as*. *as* ∈ *carrier* ($R^{Suc\ n}$) ⟹
deg R (*ring-cfs-to-univ-poly n as*) ≤ *n* ∧ (*as*!*n* ≠ **0** ⟶ *deg R* (*ring-cfs-to-univ-poly n as*) = *n*)
proof(*induction n*)
case *0*
show ⋀*as*. *as* ∈ *carrier* ($R^{Suc\ 0}$) ⟹
deg R (*ring-cfs-to-univ-poly 0 as*) ≤ *0* ∧
(*as* ! *0* ≠ **0** ⟶ *deg R* (*ring-cfs-to-univ-poly 0 as*) = *0*)
proof−
fix *as* **assume** *A*: *as* ∈ *carrier* ($R^{Suc\ 0}$)
have *0*:*cring R*
by (*simp add*: *R.is-cring*)
have *1*:**0** # *as* ∈ *carrier* (R^2)
using *A cartesian-power-cons*[*of as R Suc 0* **0**]
by (*metis numeral-1-eq-Suc-0 numeral-One one-add-one R.zero-closed*)
have *2*: (**0** # *as*) ! *1* = *as*!*0*
using *A*
by (*metis One-nat-def nth-Cons-Suc*)
have *3*: *1* ∈ {(*1*::*nat*)..<*0* + *2*} ∩ {..<*2*}
by *auto*
have *4*: *coord-partial-eval R* {*1*::*nat*..<*0* + *2*} (**0** # *as*) (*pvar R* (*1*::*nat*)) =
R.indexed-const (*as*!*0*)
unfolding *ring-cfs-to-univ-poly-def ring-cfs-to-poly-def*
using *0 1 2 one-neq-zero UP-cring.IP-to-UP-indexed-const*[*of R as*!*0 0*] *generic-poly.simps*(*1*)[*of R*] *coord-partial-eval-pvar*[*of* **0**#*as 2 1*::*nat* {*1*..<*0*+*2*}]
unfolding *UP-cring-def*
using *3* **by** *presburger*
have *5*: *ring-cfs-to-univ-poly 0 as* = *IP-to-UP* (*0*::*nat*) (*R.indexed-const* (*as*

```
! 0))
      unfolding ring-cfs-to-univ-poly-def ring-cfs-to-poly-def
      using 4 generic-poly.simps(1)[of R]
      by presburger
    hence ring-cfs-to-univ-poly 0 as = to-polynomial R (as!0)
      by (metis A UP-cring.IP-to-UP-indexed-const UP-cring.intro
          cartesian-power-car-memE' R.is-cring lessI)
  assume B: as ∈ carrier (R^Suc 0)
  have 0: (point-to-eval-map R (0 # as) 1) = as!0
  by (metis B One-nat-def cartesian-power-car-memE impossible-Cons le-numeral-extra(4)

      linorder-neqE-nat nat-less-le nth-Cons-Suc)
  have 1: closed-fun R ((point-to-eval-map R (0 # as)))
    apply(rule R.closed-funI)
  by (metis 0 B One-nat-def cartesian-power-car-memE cartesian-power-car-memE'

      length-Suc-conv less-Suc0 less-SucE nth-Cons-0 R.zero-closed)
  have 2: (1::nat) ∈ ({1..<0 + 2}::nat set)
    by simp
  have 3: poly-eval R {1..<0 + 2} (point-to-eval-map R (0 # as)) (mset-to-IP
R {#1#}) =
          coord-const (point-to-eval-map R (0 # as) 1)
    using generic-poly.simps(1)[of R] one-neq-zero
    unfolding ring-cfs-to-poly-def  coord-partial-eval-def var-to-IP-def
     using 0 1 2 R.poly-eval-index[of (point-to-eval-map R (0 # as)) {1..<0 +
2} 1]
    by (metis (no-types, lifting))
  have 4: (ring-cfs-to-poly R 0 as) = coord-const (as! 0)
    using 3 0 generic-poly.simps(1)[of R]
    unfolding ring-cfs-to-poly-def  coord-partial-eval-def var-to-IP-def
    by presburger
  have 5: as! 0 ∈ carrier R
    using assms B cartesian-power-car-memE' by blast
  have 6: (ring-cfs-to-univ-poly 0 as) = to-polynomial R (as! 0)
    unfolding ring-cfs-to-univ-poly-def ring-cfs-to-poly-def
    using 3 4 5 UP-cring.IP-to-UP-indexed-const[of R as!0 0::nat]
    unfolding coord-partial-eval-def
  by (smt 0 ‹ring-cfs-to-univ-poly 0 as = to-polynomial R (as ! 0)› generic-poly.simps(1)
ring-cfs-to-univ-poly-def var-to-IP-def)
  then show  deg R (ring-cfs-to-univ-poly 0 as) ≤ 0 ∧ (as ! 0 ≠ 0 ⟶ deg R
(ring-cfs-to-univ-poly 0 as) = 0)
    using  UP-cring.degree-to-poly[of R as! 0] 5  UP-cring-def[of R]
    using R.is-cring by presburger
 qed
 next
 case (Suc n)
  have 0: (ring-cfs-to-univ-poly (Suc n) as) = ring-cfs-to-univ-poly n (take (Suc
n) as) ⊕_UP R
            (as!(Suc n))⊙_UP R (X-poly R [^]_UP R (Suc n))
```

using *ring-cfs-to-univ-poly-decomp′ Suc.prems* **by** *blast*
have *1*: (*take* (*Suc n*) *as*) ∈ *carrier* ($R^{Suc\ n}$)
using *Suc.prems*
by (*meson le-Suc-eq take-closed*)
have *2*: *deg R* (*ring-cfs-to-univ-poly n* (*take* (*Suc n*) *as*)) ≤ *n*
using *1 Suc.IH*
by *blast*
have *3*: *deg R* ((*as*!(*Suc n*))⊙$_{UP\ R}$ (*X-poly R* [^]$_{UP\ R}$ (*Suc n*))) ≤ *Suc n*
using *UP-cring.degree-monom*[*of R as*!(*Suc n*) *Suc n*] *UP-cring-def*[*of R*]
Suc.prems cartesian-power-car-memE′ le-Suc-eq lessI less-imp-le-nat zero-less-Suc
by (*metis R.is-cring*)
have *4*: (*X-poly R* [^]$_{UP\ R}$ (*Suc n*)) ∈ *carrier* (*UP R*)
proof−
have *40*: *Group.monoid* (*UP R*)
using *UP-cring-def*[*of R*] *UP-domain-def cring.axioms*(*1*) *ring.is-monoid*
using *UP-cring.UP-cring R.is-cring* **by** *blast*
have *41*: *X-poly R* ∈ *carrier* (*UP R*)
using *UP-cring.X-closed*[*of R*] *UP-cring-def*[*of R*] *R.is-cring*
by *blast*
show *?thesis*
using *monoid.nat-pow-closed*[*of UP R X-poly R Suc n*] *40 41*
by *blast*
qed
have *5*: *deg R* (*ring-cfs-to-univ-poly* (*Suc n*) *as*) ≤*Suc n*
proof(*cases as*!(*Suc n*) = **0**)
case *True*
then have *T0*: (*as*!(*Suc n*))⊙$_{UP\ R}$ (*X-poly R* [^]$_{UP\ R}$ (*Suc n*)) = **0**$_{UP\ R}$
using *4 UP-ring.UP-smult-zero*[*of R X-poly R* [^]$_{UP\ R}$ (*Suc n*)] *UP-ring-def*[*of R*] *R.ring-axioms*
by *presburger*
then show *?thesis*
using *UP-ring.deg-zero*[*of R*] *UP-ring-def*[*of R*]
by (*metis 0 1 2 3 UP-ring.UP-zero-closed UP-ring.bound-deg-sum le-SucI R.ring-axioms ring-cfs-to-univ-poly-closed*)
next
case *False*
have *F0* : *as*!(*Suc n*) ∈ *carrier R*
by (*metis Suc.prems cartesian-power-car-memE le-simps*(*1*) *lessI not-less-eq-eq poly-tuple-evalE poly-tuple-evalE′ pushforward-by-pvar-list pvar-list-is-poly-tuple zero-less-Suc*)

have *F1*: (*as*!(*Suc n*))⊙$_{UP\ R}$ (*X-poly R* [^]$_{UP\ R}$ (*Suc n*)) ∈ *carrier* (*UP R*)
using *F0 4 UP-ring.UP-smult-closed*[*of R as*!(*Suc n*) *X-poly R* [^]$_{UP\ R}$ *Suc n*]
UP-ring-def[*of R*] *assms R.ring-axioms*
by *blast*
have *deg R* ((*as*!(*Suc n*))⊙$_{UP\ R}$ (*X-poly R* [^]$_{UP\ R}$ (*Suc n*))) = *Suc n*
using *False UP-cring.degree-monom*[*of R as*!(*Suc n*) *Suc n*] *UP-cring-def*[*of R*]
cartesian-power-car-memE′ lessI

using *F0 R.is-cring*
by *presburger*
then show *?thesis*
using *UP-ring.degree-of-sum-diff-degree*[*of R* (*as*!(*Suc n*))⊙$_{UP\ R}$ (*X-poly R* [↑]$_{UP\ R}$ (*Suc n*))
ring-cfs-to-univ-poly n (*take* (*Suc n*) *as*)] *1 2 4 UP-domain-def*[*of R*] *F1*
ring-cfs-to-univ-poly-closed[*of take* (*Suc n*) *as Suc n*] *0 3*
UP-ring-def[*of R*] *UP-cring-def*[*of R*]
UP-ring.equal-deg-sum less-Suc-eq-le ring-cfs-to-univ-poly-closed
by (*metis R.ring-axioms*)
qed
have *6*: (*as* ! (*Suc n*) ≠ **0** ⟶ *deg R* (*ring-cfs-to-univ-poly* (*Suc n*) *as*) = *Suc n*)
proof
assume *F*: *as* ! (*Suc n*) ≠ **0**
have *F0* : *as*!(*Suc n*) ∈ *carrier R*
by (*metis Suc.prems cartesian-power-car-memE le-simps*(*1*) *lessI not-less-eq-eq poly-tuple-evalE poly-tuple-evalE' pushforward-by-pvar-list pvar-list-is-poly-tuple zero-less-Suc*)

have *F1*: (*as*!(*Suc n*))⊙$_{UP\ R}$ (*X-poly R* [↑]$_{UP\ R}$ (*Suc n*)) ∈ *carrier* (*UP R*)
using *F0 4 UP-ring.UP-smult-closed*[*of R as*!(*Suc n*) *X-poly R* [↑]$_{UP\ R}$ *Suc n*]
UP-ring-def[*of R*] *assms R.ring-axioms*
by *blast*
then have *F2*: *deg R* ((*as*!(*Suc n*))⊙$_{UP\ R}$ (*X-poly R* [↑]$_{UP\ R}$ (*Suc n*))) = *Suc n*
using *F0 F UP-cring.degree-monom*[*of R as*!(*Suc n*) *Suc n*] *UP-cring-def*[*of R*] *R.is-cring*
by *presburger*
have *F3*: *ring-cfs-to-univ-poly n* (*take* (*Suc n*) *as*) ∈ *carrier* (*UP R*)
using *1 ring-cfs-to-univ-poly-closed*
by *blast*
show *deg R* (*ring-cfs-to-univ-poly* (*Suc n*) *as*) = *Suc n*
using *UP-domain-def*[*of R*] *0 F1 F2 F3 1 2*
UP-ring.degree-of-sum-diff-degree[*of R ring-cfs-to-univ-poly n* (*take* (*Suc n*) *as*)
as ! *Suc n* ⊙$_{UP\ R}$ *X-poly R* [↑]$_{UP\ R}$ *Suc n*]
UP-ring.equal-deg-sum le-imp-less-Suc UP-ring-def[*of R*] *UP-cring-def*[*of R*]
by (*metis R.ring-axioms*)
qed
show *?case*
using *5 6* **by** *blast*
qed
show *deg R* (*ring-cfs-to-univ-poly n as*) ≤ *n*
using *0 assms*
by *blast*
show *as* ! *n* ≠ **0** ⟹ *deg R* (*ring-cfs-to-univ-poly n as*) = *n*
using *0 assms*
by *blast*

qed

lemma *ring-cfs-to-univ-poly-constant*:
assumes *as* ∈ *carrier* (R^1)
shows *ring-cfs-to-univ-poly 0 as* = *to-polynomial R* (*as*!*0*)
proof−
have *0*: (*1*::*nat*) ∈ {*1*..<*0* + *2*}
by *simp*
have *1*: *closed-fun R* (*point-to-eval-map R* (**0** # *as*))
using *assms*
by (*smt cartesian-power-car-memE'' R.closed-funI nth-mem set-ConsD subset-code*(*1*) *R.zero-closed*)
have *2*: (*point-to-eval-map R* (**0** # *as*) (*1*::*nat*)) = *as*!*0*
by (*metis One-nat-def assms cartesian-power-car-memE impossible-Cons le-numeral-extra*(*4*) *linorder-neqE-nat nat-less-le nth-Cons-Suc*)
have *3*: *as*!*0* ∈ *carrier R*
using *assms cartesian-power-car-memE'*
by *blast*
have (*poly-eval R* {*1*::*nat*..<*0* + *2*} (*point-to-eval-map R* (**0** # *as*)) (*generic-poly R 0*)) = *coord-const* (*point-to-eval-map R* (**0** # *as*) *1*)
using *generic-poly.simps*(*1*)[*of R*] *0 1 one-not-zero cring.poly-eval-index*[*of R point-to-eval-map R* (**0** # *as*) {*1*..<*0* + *2*} *1*]
unfolding *var-to-IP-def*
using *R.is-cring local.one-neq-zero* **by** *presburger*
then have (*poly-eval R* {*1*..<*0* + *2*} (*point-to-eval-map R* (**0** # *as*)) (*generic-poly R 0*)) = *coord-const* (*as*!*0*)
using *2*
by *presburger*
then show *?thesis*
using *3*
unfolding *ring-cfs-to-univ-poly-def ring-cfs-to-poly-def coord-partial-eval-def*
by (*metis UP-cring.IP-to-UP-indexed-const UP-cring.intro R.is-cring*)
qed

lemma *ring-cfs-to-univ-poly-top-coeff*:
assumes *as* ∈ *carrier* ($R^{Suc\ n}$)
shows (*ring-cfs-to-univ-poly n as*) *n* = *as* ! *n*
proof(*cases n* = *0*)
case *True*
have *0*: *as* ! *0* ∈ *carrier R*
using *assms cartesian-power-car-memE'*
by *blast*
have *1*: *to-polynomial R* (*as* ! *0*) *0* = *as* ! *0*
using *assms cartesian-power-car-memE'*[*of as R Suc n*] *UP-ring.cfs-monom*[*of R*]
unfolding *to-polynomial-def UP-ring-def*
using *0 R.ring-axioms* **by** *presburger*
have *ring-cfs-to-univ-poly 0 as* = *to-polynomial R* (*as* ! *0*)
using *One-nat-def True assms ring-cfs-to-univ-poly-constant* **by** *presburger*

then show *?thesis*
using *True 1*
by *presburger*
next
case *False*
obtain *k* **where** *k-def*: *Suc k = n*
using *False*
by (*metis lessI less-Suc-eq-0-disj*)
have *ring-cfs-to-univ-poly (Suc k) as (Suc k) = as ! (Suc k)*
proof−
have *0*: *ring-cfs-to-univ-poly (Suc k) as n = ring-cfs-to-univ-poly (Suc k) (take (Suc (Suc k))as) n*
by (*metis assms(1) k-def le-Suc-eq ring-cfs-to-poly-eq ring-cfs-to-univ-poly-def*)
have *1*: *take (Suc (Suc k)) as ∈ carrier* $(R^{Suc\ (Suc\ k)})$
using *assms k-def take-closed*
by *blast*
have *2*: *ring-cfs-to-univ-poly (Suc k) (take (Suc (Suc k))as) =*
ring-cfs-to-univ-poly k (take (Suc k) (take (Suc (Suc k)) as)) ⊕$_{UP\ R}$ *(as!(Suc k))⊙*$_{UP\ R}$ *(X-poly R [^]*$_{UP\ R}$ *(Suc k))*
using *1 ring-cfs-to-univ-poly-decomp'[of take (Suc (Suc k))as k] assms*
by (*metis cartesian-power-car-memE k-def nat-le-linear take-all*)
have *3*: *ring-cfs-to-univ-poly (Suc k) (take (Suc (Suc k))as) =*
ring-cfs-to-univ-poly k (take (Suc k) as) ⊕$_{UP\ R}$ *(as!(Suc k))⊙*$_{UP\ R}$ *(X-poly R [^]*$_{UP\ R}$ *(Suc k))*
using *2*
by (*metis assms(1) k-def le-Suc-eq ring-cfs-to-poly-eq ring-cfs-to-univ-poly-decomp' ring-cfs-to-univ-poly-def*)
have *4*: *deg R (ring-cfs-to-univ-poly k (take (Suc k) as)) ≤ k*
by (*metis assms(1) dual-order.refl k-def le-SucI ring-cfs-to-univ-poly-degree(1) take-closed*)
have *5*: *(ring-cfs-to-univ-poly k (take (Suc k) as)) ∈ carrier (UP R)*
by (*metis assms(1) k-def le-Suc-eq le-refl ring-cfs-to-univ-poly-closed take-closed*)
have *6*: *X-poly R [^]*$_{UP\ R}$ *Suc k ∈ carrier (UP R)*
using *monoid.nat-pow-closed[of UP R X-poly R Suc k] domain-def ring.is-monoid[of UP R]*
UP-cring.X-closed[of R] UP-domain-def[of R] UP-cring-def[of R]
cring.axioms(1) UP-cring.UP-cring
using *R.is-cring* **by** *blast*
have *7*: *(as!(Suc k))⊙*$_{UP\ R}$ *(X-poly R [^]*$_{UP\ R}$ *(Suc k)) ∈ carrier (UP R)*
using *UP-ring.UP-smult-closed[of R as!(Suc k) (X-poly R [^]*$_{UP\ R}$ *(Suc k))]*
UP-ring-def[of R] domain-def 6 cartesian-power-car-memE'[of as R - Suc k]
assms(1) k-def R.ring-axioms **by** *blast*
have *8*: *ring-cfs-to-univ-poly (Suc k) as (Suc k) = ((as!(Suc k))⊙*$_{UP\ R}$ *(X-poly R [^]*$_{UP\ R}$ *(Suc k))) (Suc k)*
using *3 4 k-def*
5 7 UP-cring-def[of R] UP-ring-def[of R] add.r-cancel-one' assms(1)

cartesian-power-car-memE le-eq-less-or-eq
le-imp-less-Suc take-all R.zero-closed UP-ring.UP-a-comm UP-ring.coeff-of-sum-diff-degree0 R.ring-axioms
by (*metis* (*no-types*, *lifting*))
then show *?thesis* **using** *UP-cring-def*[*of R*] *UP-cring.monom-coeff assms*(*1*) *cartesian-power-car-memE*
k-def lessI point-to-eval-map-closed
by (*metis* (*no-types*, *lifting*) *cartesian-power-car-memE'* *R.is-cring*)
qed
then show *?thesis*
using *k-def False*
by *blast*
qed

lemma(**in** *UP-cring*) *monom-plus-lower-degree-top-coeff*:
assumes *degree* $p < n$
assumes $p \in$ *carrier* (*UP R*)
assumes $a \in$ *carrier R*
shows $(p \oplus_{UP\ R} (a \odot_{UP\ R} (X\text{-}poly\ R)[\hat{}]_{UP\ R}\ n))\ n = a$
proof−
have *0*: $(a \odot_{UP\ R} (X\text{-}poly\ R\ [\hat{}]_{UP\ R}\ n)) \in$ *carrier* (*UP R*)
using *P.nat-pow-closed P-def X-closed assms*(*3*) *smult-closed*
by *blast*
have *1*: $(\ (a \odot_{UP\ R} (X\text{-}poly\ R)[\hat{}]_{UP\ R}\ n) \oplus_{UP\ R} p)\ n = (a \odot_{UP\ R} (X\text{-}poly\ R)[\hat{}]_{UP\ R}\ n)\ n$
using *0 UP-ring.coeff-of-sum-diff-degree0*[*of R*] *UP-cring-def*[*of R*] *assms*(*1*) *assms*(*2*)
using *is-UP-ring* **by** *blast*
then show *?thesis*
using *0 assms P-def UP-a-comm UP-cring.monom-coeff UP-cring-def*[*of R*]
by (*metis R-cring*)
qed

lemma(**in** *UP-cring*) *monom-closed*:
assumes $a \in$ *carrier R*
shows $a \odot_{UP\ R} ((X\text{-}poly\ R)[\hat{}]_{UP\ R}\ (n::nat)) \in$ *carrier* (*UP R*)
using *P.nat-pow-closed P-def assms X-closed carrier-is-submodule submoduleE*(*4*)
by *blast*

lemma(**in** *UP-cring*) *monom-bottom-coeff*:
assumes $a \in$ *carrier R*
assumes $n > 0$
shows $(a \odot_{UP\ R} ((X\text{-}poly\ R)[\hat{}]_{UP\ R}\ (n::nat)))\ 0 = \mathbf{0}$
using *assms monom-coeff*[*of a n*] *P-def local.monom-coeff*
by *presburger*

lemma(**in** *UP-cring*) *monom-plus-lower-degree-bottom-coeff*:
assumes $0 < n$
assumes $p \in$ *carrier* (*UP R*)

assumes $a \in$ *carrier* R
shows $(p \oplus_{UP\ R} (a \odot_{UP\ R} (X\text{-}poly\ R)[\hat{}]_{UP\ R}\ (n::nat)))\ 0 = p\ 0$
proof−
have *0*: $p\ 0 \in$ *carrier* R
using *assms*(*2*) *UP-ring-def is-UP-ring P-def cfs-closed* **by** *blast*
have *1*: $(p \oplus_{UP\ R} (a \odot_{UP\ R} (X\text{-}poly\ R)[\hat{}]_{UP\ R}\ (n::nat)))\ 0 = p\ 0 \oplus (a \odot_{UP\ R} (X\text{-}poly\ R)[\hat{}]_{UP\ R}\ n)\ 0$
using *assms monom-closed*[*of a n*] *cfs-add*[*of p* $(a \odot_{UP\ R} (X\text{-}poly\ R)[\hat{}]_{UP\ R}\ (n::nat))\ 0$]
unfolding *P-def*
by *blast*
then have $(a \odot_{UP\ R} ((X\text{-}poly\ R)\ [\hat{}]_{UP\ R}\ n))\ 0 = \mathbf{0}$
using *monom-bottom-coeff*[*of a n*] *P-def assms*(*1*) *assms*(*3*) *local.monom-coeff*
by *blast*
then have *2*: $(p \oplus_{UP\ R} (a \odot_{UP\ R} (X\text{-}poly\ R)[\hat{}]_{UP\ R}\ (n::nat)))\ 0 = p\ 0 \oplus \mathbf{0}$
using *1* **by** *metis*
then show *?thesis*
using *0 R.add.l-cancel-one*[*of p 0*] *R.zero-closed*
by *presburger*
qed

lemma *ring-cfs-to-univ-poly-bottom-coeff*:
assumes *as* $\in$ *carrier* $(R^{Suc\ n})$
shows $(ring\text{-}cfs\text{-}to\text{-}univ\text{-}poly\ n\ as)\ 0 = as\ !\ 0$
proof−
have $\bigwedge as.\ as \in carrier\ (R^{Suc\ n}) \Longrightarrow (ring\text{-}cfs\text{-}to\text{-}univ\text{-}poly\ n\ as)\ 0 = as\ !\ 0$
apply(*induction n*)
using *ring-cfs-to-univ-poly-top-coeff* **apply** *blast*
proof−
fix *n as*
assume *IH*: $\bigwedge as.\ as \in carrier\ (R^{Suc\ n}) \Longrightarrow (ring\text{-}cfs\text{-}to\text{-}univ\text{-}poly\ n\ as)\ 0 = as\ !\ 0$
assume *A*: $as \in carrier\ (R^{Suc\ (Suc\ n)})$
show $ring\text{-}cfs\text{-}to\text{-}univ\text{-}poly\ (Suc\ n)\ as\ 0 = as\ !\ 0$
proof−
have *0*: $ring\text{-}cfs\text{-}to\text{-}univ\text{-}poly\ (Suc\ n)\ as = ring\text{-}cfs\text{-}to\text{-}univ\text{-}poly\ n\ (take\ (Suc\ n)\ as) \oplus_{UP\ R} (as!(Suc\ n)) \odot_{UP\ R} (X\text{-}poly\ R)[\hat{}]_{UP\ R}\ (Suc\ n)$
using *A ring-cfs-to-univ-poly-decomp′*[*of as n*]
by *blast*
have *1*: $ring\text{-}cfs\text{-}to\text{-}univ\text{-}poly\ n\ (take\ (Suc\ n)\ as) \in carrier\ (UP\ R)$
by (*meson A ring-cfs-to-univ-poly-closed R.is-cring le-Suc-eq take-closed*)
have *2*:$as\ !\ Suc\ n \in carrier\ R$
using *assms cartesian-power-car-memE′ A*
by *blast*
have *3*: $(ring\text{-}cfs\text{-}to\text{-}univ\text{-}poly\ n\ (take\ (Suc\ n)\ as) \oplus_{UP\ R} as\ !\ Suc\ n \odot_{UP\ R} X\text{-}poly\ R\ [\hat{}]_{UP\ R}\ (Suc\ n))\ 0 =$
$ring\text{-}cfs\text{-}to\text{-}univ\text{-}poly\ n\ (take\ (Suc\ n)\ as)\ 0$
proof−

have *as* ! *Suc n* $\odot_{UP\ R}$ *X-poly R* $[\widehat{\ }]_{UP\ R}$ *Suc n* $\in$ *carrier* (*UP R*)
by (*meson 2 UP-cring.monom-closed UP-cring-def R.is-cring*)
hence *30*: (*ring-cfs-to-univ-poly n* (*take* (*Suc n*) *as*) $\oplus_{UP\ R}$ *as* ! *Suc n* $\odot_{UP\ R}$ *X-poly R* $[\widehat{\ }]_{UP\ R}$ (*Suc n*)) *0* =
(*ring-cfs-to-univ-poly n* (*take* (*Suc n*) *as*)) *0* $\oplus$ (*as* ! *Suc n* $\odot_{UP\ R}$ *X-poly R* $[\widehat{\ }]_{UP\ R}$ (*Suc n*)) *0*
using *A ring-cfs-to-univ-poly-closed*[*of take* (*Suc n*) *as n*] *take-closed*[*of Suc n Suc* (*Suc n*) *as R*]
UP-ring.cfs-add[*of R ring-cfs-to-univ-poly n* (*take* (*Suc n*) *as*) *as* ! *Suc n* $\odot_{UP\ R}$ *X-poly R* $[\widehat{\ }]_{UP\ R}$ (*Suc n*) *0*]
unfolding *UP-ring-def*
using *1 R.ring-axioms* **by** *blast*
have *31*: (*as* ! *Suc n* $\odot_{UP\ R}$ *X-poly R* $[\widehat{\ }]_{UP\ R}$ *Suc n*) *0* = **0**
by (*metis* (*no-types*, *lifting*) *2 Suc-neq-Zero UP-cring.monom-coeff UP-cring-def R.is-cring*)
thus *?thesis* **using** *30 2*
by (*simp add*: *1 UP-car-memE*(*1*))
qed
have *4*: (*take* (*Suc n*) *as*) $\in$ *carrier* ($R^{Suc\ n}$)
by (*meson A le-Suc-eq take-closed*)
have *5*: *ring-cfs-to-univ-poly n* (*take* (*Suc n*) *as*) *0* = *as*!*0*
using *IH*[*of* (*take* (*Suc n*) *as*)] *4 nth-take*[*of 0 Suc n as*] *less-Suc-eq-0-disj*
by *presburger*
then show *?thesis*
using *0 3*
by *presburger*
qed
qed
then show *?thesis*
using *assms*
by *blast*
qed

lemma *ring-cfs-to-univ-poly-chain*:
assumes *as* $\in$ *carrier* ($R^{Suc\ n}$)
assumes $l \leq n$
shows $l \leq k \wedge k \leq n \implies$ (*ring-cfs-to-univ-poly k* (*take* (*Suc k*) *as*)) *l* = (*ring-cfs-to-univ-poly l* (*take* (*Suc l*) *as*)) *l*
apply(*induction k*)
apply *blast*
proof−
fix *k*
assume *IH*: ($l \leq k \wedge k \leq n \implies$ *ring-cfs-to-univ-poly k* (*take* (*Suc k*) *as*) *l* = *ring-cfs-to-univ-poly l* (*take* (*Suc l*) *as*) *l*)
assume *A*: $l \leq$ *Suc k* $\wedge$ *Suc k* $\leq n$
show *ring-cfs-to-univ-poly* (*Suc k*) (*take* (*Suc* (*Suc k*)) *as*) *l* = *ring-cfs-to-univ-poly l* (*take* (*Suc l*) *as*) *l*
proof(*cases l* = *Suc k*)
case *True*

then show *?thesis*
by *blast*
next
case *False*
then have $l \leq k \wedge k \leq n$
using *A le-Suc-eq*
by *blast*
then have *0*: *ring-cfs-to-univ-poly k (take (Suc k) as) l = ring-cfs-to-univ-poly l (take (Suc l) as) l*
using *IH*
by *blast*
have *1*: *ring-cfs-to-univ-poly (Suc k) (take (Suc (Suc k)) as) = ring-cfs-to-univ-poly k (take (Suc k) as)*
$\oplus_{UP\ R}$ *(as!(Suc k))* $\odot_{UP\ R}$ *(X-poly R)*$[\widehat{}]_{UP\ R}$ *(Suc k)*
using *assms A ring-cfs-to-univ-poly-decomp[of as n k] Suc-le-lessD*
by *blast*
have *2*: *ring-cfs-to-univ-poly (Suc k) (take (Suc (Suc k)) as) l = ring-cfs-to-univ-poly k (take (Suc k) as) l*
$\oplus$*((as!(Suc k))* $\odot_{UP\ R}$ *(X-poly R)*$[\widehat{}]_{UP\ R}$ *(Suc k)) l*
proof −
have *21*: *ring-cfs-to-univ-poly k (take (Suc k) as) ∈ carrier (UP R)*
by *(meson A assms(1) le-SucI ring-cfs-to-univ-poly-closed take-closed)*
have *22*: *as ! Suc k* $\odot_{UP\ R}$ *X-poly R* $[\widehat{}]_{UP\ R}$ *Suc k ∈ carrier (UP R)*
using *UP-ring-def[of R] A UP-ring.monom-closed assms(1) cartesian-power-car-memE′ less-Suc-eq-le*
monoid.nat-pow-closed[of UP R X-poly R Suc k]
unfolding *X-poly-def*
by *(metis UP-ring.UP-ring UP-ring.UP-smult-closed R.ring-axioms R.one-closed ring.is-monoid)*
show *?thesis*
using *1 21 22 UP-ring.cfs-add[of R ring-cfs-to-univ-poly k (take (Suc k) as) ((as!(Suc k))* $\odot_{UP\ R}$ *(X-poly R)*$[\widehat{}]_{UP\ R}$ *(Suc k)) l] UP-ring-def[of R] R.ring-axioms* **by** *presburger*
qed
have *3*: *(* $\odot_{UP\ R}$ *(X-poly R)*$[\widehat{}]_{UP\ R}$ *(Suc k)) l =* $\mathbf{0}$
using *UP-cring.monom-coeff[of R as!(Suc k)] A False UP-cring-def assms(1) cartesian-power-car-memE′*
by *(metis R.is-cring le-imp-less-Suc)*
then show *?thesis*
using *2*
by *(metis 0 Suc-le-mono assms(1) assms(2) cartesian-power-car-memE′ lessI R.r-zero ring-cfs-to-univ-poly-top-coeff take-closed)*
qed
qed

lemma *ring-cfs-to-univ-poly-coeffs*:
assumes *as* $\in$ *carrier* $(R^{Suc\ n})$
assumes $l \leq n$

shows (*ring-cfs-to-univ-poly n as*) *l* = (*ring-cfs-to-univ-poly l* (*take* (*Suc l*) *as*)) *l*
proof−
have (*take* (*Suc n*) *as*) = *as*
using *assms*
by (*metis cartesian-power-car-memE le-refl take-all*)
then show *?thesis*
using *ring-cfs-to-univ-poly-chain*[*of as n l n*]
by (*metis assms*(*1*) *assms*(*2*) *order-refl*)
qed

lemma *ring-cfs-to-univ-poly-coeffs′*:
assumes *as* ∈ *carrier* ($R^{Suc\ n}$)
assumes *l* ≤ *n*
shows (*ring-cfs-to-univ-poly n as*) *l* = *as*! *l*
proof−
have *0*: (*ring-cfs-to-univ-poly l* (*take* (*Suc l*) *as*)) *l* = (*take* (*Suc l*) *as*) ! *l*
by (*meson Suc-le-mono assms*(*1*) *assms*(*2*) *ring-cfs-to-univ-poly-top-coeff take-closed*)
have *1*: (*take* (*Suc l*) *as*) ! *l* = *as*! *l*
using *nth-take*[*of l Suc l as*]
by *blast*
then show *?thesis*
using *0 assms ring-cfs-to-univ-poly-coeffs*[*of as n l*]
by *presburger*
qed

lemma *ring-cfs-to-univ-poly-coeffs′′*:
assumes *as* ∈ *carrier* ($R^{Suc\ n}$)
shows (*ring-cfs-to-univ-poly n as*) *l* = (*if l* ≤ *n then as*! *l else* **0**)
apply(*cases l* ≤*n*)
apply (*meson assms ring-cfs-to-univ-poly-coeffs′*)
proof− **assume** ¬ *l* ≤ *n* **then**
have *A*: *n* < *l*
by *auto*
have *deg R* (*ring-cfs-to-univ-poly n as*) ≤ *n*
using *assms ring-cfs-to-univ-poly-degree*(*1*) **by** *blast*
then show *?thesis*
using *A domain-def*[*of R*] *deg-leE assms le-less-trans ring-cfs-to-univ-poly-closed UP-car-memE*(*2*)
by *auto*
qed
end

definition *fun-tuple-to-univ-poly* **where**
fun-tuple-to-univ-poly R n m fs x = *cring-coord-rings.ring-cfs-to-univ-poly R m* (*function-tuple-eval R n fs x*)

context *cring-coord-rings*
begin

lemma *fun-tuple-to-univ-poly-closed*:
assumes *is-function-tuple R n fs*
assumes $x \in$ *carrier* (R^n)
assumes *length fs* $=$ *Suc m*
shows *fun-tuple-to-univ-poly R n m fs x* $\in$ *carrier* (*UP R*)
unfolding *fun-tuple-to-univ-poly-def*
using *assms*
ring-cfs-to-univ-poly-closed[*of function-tuple-eval R n fs x m*]
function-tuple-eval-closed[*of R n fs x*]
by *metis*

lemma *fun-tuple-to-univ-poly-degree-bound*:
assumes *is-function-tuple R n fs*
assumes $x \in$ *carrier* (R^n)
assumes *length fs* $=$ *Suc m*
shows *deg R* (*fun-tuple-to-univ-poly R n m fs x*) $\leq m$
unfolding *fun-tuple-to-univ-poly-def*
using *ring-cfs-to-univ-poly-degree assms*
by (*metis function-tuple-eval-closed*)

lemma *fun-tuple-to-univ-poly-degree*:
assumes *is-function-tuple R n fs*
assumes $x \in$ *carrier* (R^n)
assumes *length fs* $=$ *Suc m*
assumes (*fs*!*m*) *x* $\neq \mathbf{0}$
shows *deg R* (*fun-tuple-to-univ-poly R n m fs x*) $= m$
unfolding *fun-tuple-to-univ-poly-def*
using *ring-cfs-to-univ-poly-degree*[*of function-tuple-eval R n fs x m*]
assms
function-tuple-eval-def
function-tuple-eval-closed[*of R n fs x*]
by (*metis lessI nth-map*)

6.4 Factoring a Polynomial as a Univariate Polynomial over a Multivariable Polynomial Ring

definition *pre-to-univ-poly-hom* :: *nat* $\Rightarrow$ *nat* $\Rightarrow$ ('*a*, (('*a*, *nat*) *mvar-poly*, *nat*) *mvar-poly*) *ring-hom* **where**
pre-to-univ-poly-hom n i= *MP.indexed-const* (*n*−*1*) ∘
R.indexed-const

lemma *pre-to-univ-poly-hom-is-hom*:
assumes $i < n$
shows *ring-hom-ring R* (*Pring* (*coord-ring R* (*n*−*1*)) {*i*}) (*pre-to-univ-poly-hom n i*)
using *ring-hom-trans*[*of R.indexed-const R coord-ring R* (*n*−*1*)
ring.indexed-const(*Pring R* ({..<*n*−*1*}))
Pring (*coord-ring R* (*n*−*1*)) {*i*}]
R.indexed-const-ring-hom[*of* {..<*n*−*1*}]

MP.indexed-const-ring-hom[*of n* {..<*n*−*1*}]
ring-hom-ring.homh[*of R coord-ring R* (*n* − *1*) *coord-const*]
unfolding *ring-hom-ring-def*[*of R*]
by (*smt MP.Pring-is-ring MP.indexed-const-ring-hom coord-ring-def pre-to-univ-poly-hom-def ring-hom-ring.homh ring-hom-ring-axioms-def*)

definition *pre-to-univ-poly-var-ass* ::
nat ⇒ *nat* ⇒ *nat* ⇒ (('*a*, *nat*) *mvar-poly*, *nat*) *mvar-poly* **where**
pre-to-univ-poly-var-ass n i j =(*if j* < *i then MP.indexed-const* (*n*−*1*) (*pvar R j*) *else*
(*if j* = *i then pvar* (*coord-ring R* (*n*−*1*)) *i else*
(*if j* < *n then MP.indexed-const* (*n*−*1*) (*pvar R* (*j* − *1*)) *else*
0$_{Pring\ (coord\text{-}ring\ R\ (n-1))\ \{i\}}$)))

lemma *pre-to-univ-poly-var-ass-closed*:
assumes *i* < *n*
shows *closed-fun* (*Pring* (*coord-ring R* (*n*−*1*)) {*i*}) (*pre-to-univ-poly-var-ass n i*)
proof fix *j*
show *pre-to-univ-poly-var-ass n i j* ∈ *carrier* (*Pring* (*coord-ring R* (*n* − *1*)) {*i*})
unfolding *pre-to-univ-poly-var-ass-def*
apply(*cases j* < *i*)
using *pvar-closed*[*of j n*] *assms cring.indexed-const-closed*
apply (*metis* (*no-types*, *lifting*) *R.Pring-is-cring Suc-diff-1 Suc-le-eq coord-ring-def diff-diff-cancel R.is-cring less-imp-diff-less local.pvar-closed not-less0 not-less-eq-eq*)

apply(*cases j* = *i*)
using *assms* **apply** (*meson pvar-closed R.Pring-is-cring R.is-cring singletonI*)

apply(*cases j* < *n*)
using *pvar-closed*[*of j*−*1 n*] *assms MP.indexed-const-closed R.Pring-is-cring Suc-diff-1 Suc-le-eq coord-ring-def R.is-cring pvar-closed neq0-conv not-le*
apply (*metis MP.Pring-var-closed singletonI*)
using *MP.Pring-is-ring*[*of n*−*1* {*i*}] **apply** *blast*
by (*smt MP.Pring-zero-closed MP.indexed-const-closed Suc-diff-1 Suc-le-eq le-eq-less-or-eq less-Suc-eq local.pvar-closed nat-induct*)
qed

lemma *pre-to-univ-poly-var-ass-closed′*:
assumes *i* < *n*
shows (*pre-to-univ-poly-var-ass n i*) ∈ {..<*n*} → *carrier* (*Pring* (*coord-ring R* (*n*−*1*)) {*i*})
by (*metis* (*no-types*, *lifting*) *Pi-iff UNIV-I assms pre-to-univ-poly-var-ass-closed*)

definition *pre-to-univ-poly* ::
nat ⇒ *nat* ⇒ (('*a*, *nat*) *mvar-poly*, (('*a*, *nat*) *mvar-poly*, *nat*) *mvar-poly*) *ring-hom*
where
pre-to-univ-poly (*n*::*nat*) (*i*::*nat*) = *indexed-poly-induced-morphism* {..<*n*} (*Pring*

(*coord-ring R (n−1)*) {*i*})
(*pre-to-univ-poly-hom n i*)
(*pre-to-univ-poly-var-ass n i*)

lemma *pre-to-univ-poly-is-hom*:
assumes $i < n$
assumes ψ = *pre-to-univ-poly n i*
shows *ring-hom-ring* ($R[\mathcal{X}_n]$) (*Pring* (*coord-ring R* (*n*−1)) {*i*}) ψ
$\bigwedge j.\ j < i \Longrightarrow \psi$ (*pvar R j*) = *MP.indexed-const* (*n*−1) (*pvar R j*)
ψ (*pvar R i*) = *pvar* (*coord-ring R* (*n*−1)) *i*
$\bigwedge j.\ i < j \wedge j < n \Longrightarrow \psi$ (*pvar R j*) = *MP.indexed-const* (*n*−1) (*pvar R* (*j* − 1))
$\bigwedge a.\ a \in$ *carrier R* $\Longrightarrow \psi$ (*coord-const a*) = *MP.indexed-const* (*n*−1) (*coord-const a*)
$\bigwedge p.\ p \in$ *carrier* ($R[\mathcal{X}_n]$) $\Longrightarrow$ *pre-to-univ-poly n i p* ∈ *carrier* (*Pring* (*coord-ring R* (*n*−1)) {*i*})
proof−
have *0*: *cring* (*Pring* (*coord-ring R* (*n* − 1)) {*i*})
using *MP.Pring-is-cring coord-cring-cring* **by** *blast*
have *1*: *pre-to-univ-poly-var-ass n i* ∈ {..<*n*} → *carrier* (*Pring* (*coord-ring R* (*n* − 1)) {*i*})
using *Pi-iff assms*(*1*) *pre-to-univ-poly-var-ass-closed*[*of i n*]
by *blast*
have *2*: *ring-hom-ring R* (*Pring* (*coord-ring R* (*n* − 1)) {*i*}) (*pre-to-univ-poly-hom n i*)
using *assms*(*1*) *pre-to-univ-poly-hom-is-hom* **by** *auto*

show *3*:*ring-hom-ring* ($R[\mathcal{X}_n]$) (*Pring* (*coord-ring R* (*n*−1)) {*i*}) ψ
using *R.Pring-universal-prop*(*1*)[*of* (*Pring* (*coord-ring R* (*n*−1)) {*i*}) *pre-to-univ-poly-var-ass n i*
{..<*n*} *pre-to-univ-poly-hom n i* ψ] *assms 0 1 2*
unfolding *pre-to-univ-poly-def*
by (*metis coord-ring-def*)

show $\bigwedge j.\ j < i \Longrightarrow$
ψ (*pvar R j*) = *MP.indexed-const* (*n*−1) (*pvar R j*)
proof−
fix *j* **assume** *A*: *j* < *i*
then have *00*: *MP.indexed-const* (*n* − 1) (*pvar R j*) = *pre-to-univ-poly-var-ass n i j*
unfolding *pre-to-univ-poly-var-ass-def* **by** *auto*
have *01*: *j* ∈ {..<*n*}
using *assms A* **by** *auto*
show ψ (*pvar R j*) = *MP.indexed-const* (*n*−1) (*pvar R j*)
using *R.Pring-universal-prop*(*2*)[*of* (*Pring* (*coord-ring R* (*n*−1)) {*i*}) *pre-to-univ-poly-var-ass n i*
{..<*n*} *pre-to-univ-poly-hom n i* ψ] *assms 0 1 2 01*
MP.is-cring
unfolding *pre-to-univ-poly-def 00* **unfolding** *coord-ring-def var-to-IP-def*

```
    by blast
  qed
  show ψ (pvar R i) = pvar (coord-ring R (n − 1)) i
   using R.Pring-universal-prop[of (Pring (coord-ring R (n−1)) {i}) pre-to-univ-poly-var-ass
n i
                                {..<n} pre-to-univ-poly-hom n i ψ] assms 0 1 2
    unfolding pre-to-univ-poly-def  coord-ring-def
    using lessThan-iff less-not-refl pre-to-univ-poly-var-ass-def var-to-IP-def
    by (metis coord-ring-def)
  show ⋀j. i < j ∧ j < n ⟹ ψ (pvar R j) = MP.indexed-const (n − 1) (pvar R
(j − 1))
   using R.Pring-universal-prop[of (Pring (coord-ring R (n−1)) {i}) pre-to-univ-poly-var-ass
n i
                                {..<n} pre-to-univ-poly-hom n i ψ] assms 0 1 2
    unfolding pre-to-univ-poly-def
    using add-diff-inverse-nat lessThan-iff less-diff-conv less-imp-add-positive
       not-add-less1 pre-to-univ-poly-var-ass-def trans-less-add2 var-to-IP-def
    by (metis (no-types, lifting) coord-ring-def)
  show ⋀a. a ∈ carrier R ⟹ ψ (R.indexed-const a) = MP.indexed-const (n −
1) (R.indexed-const a)
   using R.Pring-universal-prop(3)[of (Pring (coord-ring R (n−1)) {i}) pre-to-univ-poly-var-ass
n i
                               {..<n} pre-to-univ-poly-hom n i ψ] assms 0 1 2 comp-apply
    unfolding pre-to-univ-poly-def pre-to-univ-poly-hom-def
    by metis
  show ⋀p. p ∈ carrier (R[X_n]) ⟹ pre-to-univ-poly n i p ∈ carrier (Pring
(coord-ring R (n − 1)) {i})
  proof−
    fix p assume A: p ∈ carrier (R[X_n])
    have ψ ∈ carrier (R[X_n]) → carrier (Pring (coord-ring R (n − 1)) {i})
      using 3 unfolding ring-hom-ring-def ring-hom-ring-axioms-def ring-hom-def
by blast
    then show  pre-to-univ-poly n i p ∈ carrier (Pring (coord-ring R (n − 1)) {i})
      using A assms
      by blast
  qed
qed

lemma insert-at-index-closed:
  assumes a ∈ carrier (R^n)
  assumes x ∈ carrier R
  assumes i ≤ n
  shows insert-at-index a x i ∈ carrier (R^Suc n)
  apply(rule cartesian-power-car-memI')
proof−
  have 0: length (take i a) = i
    using assms(1) assms(3) cartesian-power-car-memE take-closed by blast
  have 1: length (drop i a) = (n − i)
    using assms cartesian-power-car-memE length-drop
```

by *blast*
then have *length* (*x* # *drop i a*) = *Suc* (*n* − *i*)
by (*metis length-Cons*)
then show *length* (*insert-at-index a x i*) = *Suc n*
using *0 1 assms*
by (*metis Suc-eq-plus1 cartesian-power-car-memE insert-at-index-length*)
show ⋀*ia*. *ia* < *Suc n* ⟹ *insert-at-index a x i* ! *ia* ∈ *carrier R*
proof− **fix** *j* **assume** *A*: *j* < *Suc n*
show *insert-at-index a x i* ! *j* ∈ *carrier R*
apply(*cases j* < *i*)
apply (*metis A assms*(*1*) *assms*(*3*) *cartesian-power-car-memE cartesian-power-car-memE′ insert-at-index-eq′ le-imp-less-Suc less-Suc-eq not-less-eq*)
apply(*cases j* = *i*)
apply (*metis assms*(*1*) *assms*(*2*) *assms*(*3*) *cartesian-power-car-memE insert-at-index-eq*)
proof− **assume** *A1*: ¬ *j* < *i* *j* ≠*i*
then have *i* < *j* **by** *auto*
then have (*take i a* @ *x* # *drop i a*) ! *j* = *drop i a* ! (*j* − (*Suc i*))
by (*metis 0 A1*(*1*) *Suc-diff-Suc nth-Cons-Suc nth-append*)
then show *insert-at-index a x i* ! *j* ∈ *carrier R*
by (*metis A* ‹*i* < *j*› *assms*(*1*) *cartesian-power-car-memE cartesian-power-car-memE′ insert-at-index-eq″ less-Suc-eq-0-disj less-Suc-eq-le not-less0*)
qed
qed
qed

lemma *pre-to-univ-poly-eval*:
assumes *i* < *Suc n*
assumes *p* ∈ *carrier* (*R*[$\mathcal{X}_{Suc\ n}$])
assumes *a* ∈ *carrier* (R^n)
assumes *x* ∈ *carrier R*
assumes *as* = *insert-at-index a x i*
shows *eval-at-point R as p* = *eval-at-point R a* (*total-eval* (*R*[$\mathcal{X}_n$]) (λ *i*. *coord-const x*) (*pre-to-univ-poly* (*Suc n*) *i p*))
apply(*rule R.Pring-car-induct″*[*of p* {..<*Suc n*}])
unfolding *coord-ring-def*
apply (*metis assms*(*2*) *coord-ring-def*)
proof−
have *0*: *as* ∈ *carrier* ($R^{Suc\ n}$)
using *assms insert-at-index-closed*
by (*meson less-Suc-eq-le*)
show ⋀*c*. *c* ∈ *carrier R* ⟹
eval-at-point R as (*R.indexed-const c*) =
eval-at-point R a (*total-eval* (*Pring R* {..<*n*}) (λ*i*. *R.indexed-const x*) (*pre-to-univ-poly* (*Suc n*) *i* (*R.indexed-const c*)))
proof− **fix** *c* **assume** *c* ∈ *carrier R*
have *00*: *eval-at-poly R* (*coord-const c*) *as* = *c*
using *assms eval-at-point-const*[*of c as Suc n*] *0* ‹*c* ∈ *carrier R*›
by *blast*

have *01*: *closed-fun* ($R[\mathcal{X}_n]$) ($\lambda n.$ *coord-const x*)
using *assms*(*4*) *R.indexed-const-closed*
by (*metis Pi-I coord-ring-def*)
have *02*: (*pre-to-univ-poly* (*Suc n*) *i* (*coord-const c*)) = *ring.indexed-const* ($R[\mathcal{X}_n]$) (*coord-const c*)
using *pre-to-univ-poly-is-hom*(*5*)[*of i Suc n - c*] ‹*c* ∈ *carrier R*› *assms*(*1*) *diff-Suc-1*
by (*metis coord-ring-def*)
have *03*: (*total-eval* ($R[\mathcal{X}_n]$) (λ *i. coord-const x*) (*pre-to-univ-poly* (*Suc n*) *i* (*coord-const c*))) =
coord-const c
using *01 cring.total-eval-const*[*of* $R[\mathcal{X}_n]$ *coord-const c*]
by (*smt 02 MP.total-eval-const* ‹*c* ∈ *carrier R*› *coord-ring-def cring.indexed-const-closed R.is-cring*)
show *eval-at-point R as* (*R.indexed-const c*) =
eval-at-point R a (*total-eval* (*Pring R* {..<*n*}) ($\lambda i.$ *R.indexed-const x*) (*pre-to-univ-poly* (*Suc n*) *i* (*R.indexed-const c*)))
using *assms 00 02 03*
by (*metis* ‹*c* ∈ *carrier R*› *coord-ring-def eval-at-point-const*)
qed
have *01*: *closed-fun* ($R[\mathcal{X}_n]$) ($\lambda n.$ *coord-const x*)
using *assms*(*4*) *R.indexed-const-closed*
by (*metis Pi-I coord-ring-def*)
have *02*: *ring-hom-ring* ($R[\mathcal{X}_{Suc\ n}]$) (*Pring* ($R[\mathcal{X}_n]$) {*i*}) (*pre-to-univ-poly* (*Suc n*) *i*)
using *pre-to-univ-poly-is-hom*(*1*)[*of i Suc n*]
by (*simp add*: *assms*)
show $\bigwedge p\ q.$ *p* ∈ *carrier* (*Pring R* {..<*Suc n*}) ⟹
q ∈ *carrier* (*Pring R* {..<*Suc n*}) ⟹
eval-at-point R as p = *eval-at-point R a* (*total-eval* (*Pring R* {..<*n*}) ($\lambda i.$ *R.indexed-const x*) (*pre-to-univ-poly* (*Suc n*) *i p*)) ⟹
eval-at-point R as q = *eval-at-point R a* (*total-eval* (*Pring R* {..<*n*}) ($\lambda i.$ *R.indexed-const x*) (*pre-to-univ-poly* (*Suc n*) *i q*)) ⟹
eval-at-point R as (*p* $\oplus_{Pring\ R\ \{..<Suc\ n\}}$ *q*) =
eval-at-point R a (*total-eval* (*Pring R* {..<*n*}) ($\lambda i.$ *R.indexed-const x*) (*pre-to-univ-poly* (*Suc n*) *i* (*p* $\oplus_{Pring\ R\ \{..<Suc\ n\}}$ *q*)))
proof− **fix** *p q* **assume** *A*: *p* ∈ *carrier* (*Pring R* {..<*Suc n*})
q ∈ *carrier* (*Pring R* {..<*Suc n*})
eval-at-point R as p = *eval-at-point R a* (*total-eval* (*Pring R* {..<*n*}) ($\lambda i.$ *R.indexed-const x*) (*pre-to-univ-poly* (*Suc n*) *i p*))
eval-at-point R as q = *eval-at-point R a* (*total-eval* (*Pring R* {..<*n*}) ($\lambda i.$ *R.indexed-const x*) (*pre-to-univ-poly* (*Suc n*) *i q*))
have *0*: *eval-at-poly R* (*p* $\oplus_{R[\mathcal{X}_{Suc\ n}]}$ *q*) *as* =
eval-at-poly R p as $\oplus_R$ *eval-at-poly R q as*
using *0 A*(*1*) *A*(*2*) *eval-at-point-add* **unfolding** *coord-ring-def*
by *blast*
have *1*: (*total-eval* ($R[\mathcal{X}_n]$) ($\lambda i.$ *coord-const x*) (*pre-to-univ-poly* (*Suc n*) *i* (*p* $\oplus_{R[\mathcal{X}_{Suc\ n}]}$ *q*))) =

(*total-eval* ($R[\mathcal{X}_n]$) (λ*i*. *coord-const x*) (*pre-to-univ-poly* (*Suc n*) *i p*))
$\oplus_{R[\mathcal{X}_n]}$
(*total-eval* ($R[\mathcal{X}_n]$) (λ*i*. *coord-const x*) (*pre-to-univ-poly* (*Suc n*) *i q*))
proof−
have *10*: *pre-to-univ-poly* (*Suc n*) *i p* ∈ *carrier* (*Pring* ($R[\mathcal{X}_n]$) {*i*})
using *pre-to-univ-poly-is-hom*(*6*)[*of i Suc n - p*]
unfolding *coord-ring-def*
by (*metis A*(*1*) *assms*(*1*) *diff-Suc-1*)
have *11*: *pre-to-univ-poly* (*Suc n*) *i q* ∈ *carrier* (*Pring* ($R[\mathcal{X}_n]$) {*i*})
using *pre-to-univ-poly-is-hom*(*6*)[*of i Suc n - q*]
unfolding *coord-ring-def*

by (*metis A*(*2*) *assms*(*1*) *diff-Suc-1*)
have *12*: (*pre-to-univ-poly* (*Suc n*) *i* (*p* $\oplus_{R[\mathcal{X}_{Suc\ n}]}$ *q*)) =
(*pre-to-univ-poly* (*Suc n*) *i p* $\oplus_{Pring}$ ($R[\mathcal{X}_n]$) {*i*} *pre-to-univ-poly* (*Suc n*) *i q*)
using *ring-hom-ring.homh A 02 ring-hom-add*[*of pre-to-univ-poly* (*Suc n*) *i* $R[\mathcal{X}_{Suc\ n}]$ *Pring* ($R[\mathcal{X}_n]$) {*i*}
p q]
unfolding *coord-ring-def*

by *blast*

show *?thesis*

using *01 10 11 12 A cring.total-eval-add*[*of* $R[\mathcal{X}_n]$ *pre-to-univ-poly* (*Suc n*) *i p* {*i*}
pre-to-univ-poly (*Suc n*) *i q* λ*i*. *coord-const x*]
coord-cring-cring
unfolding *coord-ring-def*

by *smt*
qed
have *2*: *eval-at-poly R* (*total-eval* ($R[\mathcal{X}_n]$) (λ*i*. *coord-const x*) (*pre-to-univ-poly* (*Suc n*) *i* (*p* $\oplus_{R[\mathcal{X}_{Suc\ n}]}$ *q*))) *a* =
eval-at-poly R (*total-eval* ($R[\mathcal{X}_n]$) (λ*i*. *coord-const x*) (*pre-to-univ-poly* (*Suc n*) *i p*)) *a* ⊕
eval-at-poly R (*total-eval* ($R[\mathcal{X}_n]$) (λ*i*. *coord-const x*) (*pre-to-univ-poly* (*Suc n*) *i q*)) *a*
proof−
have *20*: *pre-to-univ-poly* (*Suc n*) *i p* ∈ *carrier* (*Pring* ($R[\mathcal{X}_n]$) {*i*})
using *A*(*1*) *02* **unfolding** *ring-hom-ring-def ring-hom-ring-axioms-def ring-hom-def* **unfolding** *coord-ring-def*
by *blast*
have *21*: *pre-to-univ-poly* (*Suc n*) *i q* ∈ *carrier* (*Pring* ($R[\mathcal{X}_n]$) {*i*})
using *A*(*2*) *02* **unfolding** *ring-hom-ring-def ring-hom-ring-axioms-def ring-hom-def* **unfolding** *coord-ring-def*

by *blast*
have *22*: (*total-eval* ($R[\mathcal{X}_n]$) (λ*i*. *coord-const x*) (*pre-to-univ-poly* (*Suc n*) *i p*)) ∈
carrier ($R[\mathcal{X}_n]$)
using *21 01 A cring.total-eval-closed*[*of* $R[\mathcal{X}_n]$ *pre-to-univ-poly* (*Suc n*) *i p* {*i*} λ*i*. *coord-const x*] *20 coord-cring-cring*
by *metis*
have *23*: (*total-eval* ($R[\mathcal{X}_n]$) (λ*i*. *coord-const x*) (*pre-to-univ-poly* (*Suc n*) *i q*)) ∈
carrier ($R[\mathcal{X}_n]$)
using *cring.total-eval-closed*[*of* $R[\mathcal{X}_n]$ *pre-to-univ-poly* (*Suc n*) *i q* {*i*} λ*i*. *coord-const x*]
by (*metis 01 21 coord-cring-cring*)

show *?thesis*
using *1 22 23 assms*(*3*) *eval-at-point-add* **by** *presburger*
qed
show *eval-at-point R as* ($p \oplus_{Pring\ R\ \{..<Suc\ n\}}\ q$) =
eval-at-point R a (*total-eval* (*Pring R* {..<*n*}) (λ*i*. *R.indexed-const x*) (*pre-to-univ-poly* (*Suc n*) *i* ($p \oplus_{Pring\ R\ \{..<Suc\ n\}}\ q$)))
using *eval-at-point-add A 0 1 2*
unfolding *coord-ring-def*

by *presburger*
qed
fix *p j*
assume *A*: *p* ∈ *carrier* (*Pring R* {..<*Suc n*}) *j* ∈ {..<*Suc n*}
eval-at-point R as p = *eval-at-point R a* (*total-eval* (*Pring R* {..<*n*}) (λ*i*. *R.indexed-const x*) (*pre-to-univ-poly* (*Suc n*) *i p*))
show *eval-at-point R as* ($p \otimes_{Pring\ R\ \{..<Suc\ n\}}\ pvar\ R\ j$) =
eval-at-point R a
(*total-eval* (*Pring R* {..<*n*}) (λ*i*. *R.indexed-const x*) (*pre-to-univ-poly* (*Suc n*) *i* ($p \otimes_{Pring\ R\ \{..<Suc\ n\}}\ pvar\ R\ j$)))
proof−
have *A0*: *eval-at-poly R* ($p \otimes_{R[\mathcal{X}_{Suc\ n}]}\ pvar\ R\ j$) *as* =
eval-at-poly R p as ⊗ *as*!*j*
proof−
have *eval-at-poly R* (*pvar R j*) *as* = *as*!*j*
using *A*(*2*) *0 eval-pvar*
by *blast*
then show *?thesis* **using** *A eval-at-point-mult*[*of as Suc n p pvar R j*] *0*
by (*metis R.Pring-var-closed coord-ring-def*)
qed
have *A1*: (*pre-to-univ-poly* (*Suc n*) *i* ($p \otimes_{R[\mathcal{X}_{Suc\ n}]}\ pvar\ R\ j$)) =
(*pre-to-univ-poly* (*Suc n*) *i p*) $\otimes_{Pring\ (R[\mathcal{X}_n])}\ \{i\}$ *pre-to-univ-poly* (*Suc n*) *i* (*pvar R j*)
using *A 02 ring-hom-ring.homh ring-hom-mult*[*of* - $R[\mathcal{X}_{Suc\ n}]$ - *p pvar R j*] *R.Pring-var-closed*[*of j* {..< *Suc n*}]

unfolding *coord-ring-def*
by *blast*
have *A2*: (*total-eval* ($R[\mathcal{X}_n]$) (λ*i*. *coord-const x*) (*pre-to-univ-poly* (*Suc n*) *i* (*p* $\otimes_{R[\mathcal{X}_{Suc\ n}]}$ *pvar R j*))) =
(*total-eval* ($R[\mathcal{X}_n]$) (λ*i*. *coord-const x*) (*pre-to-univ-poly* (*Suc n*) *i p*))$\otimes_{R[\mathcal{X}_n]}$
(*total-eval* ($R[\mathcal{X}_n]$) (λ*i*. *coord-const x*) (*pre-to-univ-poly* (*Suc n*) *i* (*pvar R j*)))
proof−
have *A20*: *pre-to-univ-poly* (*Suc n*) *i p* ∈ *carrier* (*Pring* ($R[\mathcal{X}_n]$) {*i*})
using *02 A* **unfolding** *ring-hom-ring-def ring-hom-ring-axioms-def ring-hom-def*
unfolding *coord-ring-def*
by *blast*
have *A21*: *pre-to-univ-poly* (*Suc n*) *i* (*pvar R j*) ∈ *carrier* (*Pring* ($R[\mathcal{X}_n]$) {*i*})
using *02 A* **unfolding** *ring-hom-ring-def ring-hom-ring-axioms-def ring-hom-def*
using *R.Pring-var-closed*[*of j* {..< *Suc n*}]
unfolding *coord-ring-def*
by *blast*
show *?thesis* **using** *A1 cring.total-eval-mult*[*of - pre-to-univ-poly* (*Suc n*) *i p*]
by (*smt A20 A21 MP.closed-funI MP.total-eval-mult assms*(*4*) *coord-ring-def cring.indexed-const-closed R.is-cring*)
qed
have *A3*: *pre-to-univ-poly* (*Suc n*) *i p* ∈ *carrier* (*Pring* ($R[\mathcal{X}_n]$) {*i*})
using *02 A ring-hom-ring.homh* **unfolding** *ring-hom-def*
unfolding *coord-ring-def*
by *blast*
have *A4*: *pre-to-univ-poly* (*Suc n*) *i* (*pvar R j*) ∈ *carrier* (*Pring* ($R[\mathcal{X}_n]$) {*i*})
using *02 A ring-hom-ring.homh R.Pring-var-closed*[*of j* {..< *Suc n*}] **unfolding** *ring-hom-def*
unfolding *coord-ring-def*
by *blast*
have *A5*: *total-eval* ($R[\mathcal{X}_n]$) (λ*i*. *coord-const x*) (*pre-to-univ-poly* (*Suc n*) *i p*) ∈
carrier ($R[\mathcal{X}_n]$)
using *01 cring.total-eval-closed*[*of* $R[\mathcal{X}_n]$ *pre-to-univ-poly* (*Suc n*) *i p* {*i*}]
A3 coord-cring-cring
unfolding *coord-ring-def*
by *smt*
have *A6*: *total-eval* ($R[\mathcal{X}_n]$) (λ*i*. *coord-const x*) (*pre-to-univ-poly* (*Suc n*) *i* (*pvar R j*)) ∈
carrier ($R[\mathcal{X}_n]$)

using *01 cring.total-eval-closed*[*of* $R[\mathcal{X}_n]$ *pre-to-univ-poly* (*Suc n*) *i* (*pvar R j*) {*i*}]
A4 coord-cring-cring
unfolding *coord-ring-def*
by *smt*
have *A7*: *eval-at-poly R* (*total-eval* ($R[\mathcal{X}_n]$) (λ*i*. *coord-const x*) (*pre-to-univ-poly* (*Suc n*) *i* ($p \otimes_{R[\mathcal{X}_{Suc\ n}]}$ *pvar R j*))) *a*
= *eval-at-poly R* (*total-eval* ($R[\mathcal{X}_n]$) (λ*i*. *coord-const x*) (*pre-to-univ-poly* (*Suc n*) *i p*)) *a* ⊗
eval-at-poly R (*total-eval* ($R[\mathcal{X}_n]$) (λ*i*. *coord-const x*) (*pre-to-univ-poly* (*Suc n*) *i* (*pvar R j*))) *a*
using *eval-at-point-mult A5 A6 A2 assms*(*3*) **by** *presburger*
have *A8*: *eval-at-poly R* (*total-eval* ($R[\mathcal{X}_n]$) (λ*i*. *coord-const x*) (*pre-to-univ-poly* (*Suc n*) *i* (*pvar R j*))) *a* =
as!*j*
proof(*cases j* = *i*)
case *True*
then have *pre-to-univ-poly* (*Suc n*) *i* (*pvar R j*) = *pvar* ($R[\mathcal{X}_n]$) *i*
using *pre-to-univ-poly-is-hom*(*3*)[*of i Suc n*] *assms*(*1*) *diff-Suc-1* **by** *presburger*
then have *total-eval* ($R[\mathcal{X}_n]$) (λ*i*. *coord-const x*) (*pre-to-univ-poly* (*Suc n*) *i* (*pvar R j*)) =
coord-const x
using *cring.total-eval-var*[*of* $R[\mathcal{X}_n]$ λ*i*. *coord-const x*]
unfolding *coord-ring-def*
by (*smt 01* ‹$\bigwedge i$. ⟦*cring* (*R* [$\mathcal{X}_n$]); (λ*i*. *R.indexed-const x*) ∈ *UNIV* → *carrier* (*R* [$\mathcal{X}_n$])⟧ ⟹ *total-eval* (*R* [$\mathcal{X}_n$]) (λ*i*. *R.indexed-const x*) (*mset-to-IP* (*R* [$\mathcal{X}_n$]) {#*i*#}) = *R.indexed-const x*› *coord-ring-def cring-coord-rings.coord-cring-cring cring-coord-rings-axioms var-to-IP-def*)
then have *T0*: *eval-at-poly R* (*total-eval* ($R[\mathcal{X}_n]$) (λ*i*. *coord-const x*) (*pre-to-univ-poly* (*Suc n*) *i* (*pvar R j*))) *a*
= *x*
using *eval-at-point-const*
by (*metis assms*(*3*) *assms*(*4*))
have *T1*: *as*!*j* = *x*
using *assms*
by (*metis True assms*(*5*) *cartesian-power-car-memE insert-at-index-eq le-eq-less-or-eq nat-le-linear not-less-eq*)
then show *?thesis*
using *T0* **by** *blast*
next
case *False*
then show *?thesis*
proof(*cases j* < *i*)
case *True*
then have *pre-to-univ-poly* (*Suc n*) *i* (*pvar R j*) = *ring.indexed-const* ($R[\mathcal{X}_n]$) (*pvar R j*)
using *pre-to-univ-poly-is-hom*(*2*)[*of i Suc n*] *assms*(*1*) *diff-Suc-1*

unfolding *coord-ring-def*
by *presburger*
then have *total-eval* ($R[\mathcal{X}_n]$) ($\lambda i.$ *coord-const x*) (*pre-to-univ-poly* (*Suc n*) *i* (*pvar R j*)) =
pvar R j
using *cring.total-eval-const*[*of* $R[\mathcal{X}_n]$]
by (*smt Suc-less-eq True assms*(*1*) *coord-cring-cring less-trans-Suc local.pvar-closed*)
then have *T0*: *eval-at-poly R* (*total-eval* ($R[\mathcal{X}_n]$) ($\lambda i.$ *coord-const x*) (*pre-to-univ-poly* (*Suc n*) *i* (*pvar R j*))) *a*
= *a*!*j*
using *eval-pvar*
by (*metis Suc-less-eq True assms*(*1*) *assms*(*3*) *less-trans-Suc*)
have *T1*: *as*!*j* = *a*!*j*
using *assms*
by (*metis True assms*(*5*) *cartesian-power-car-memE insert-at-index-eq*′ *less-Suc-eq-le*)
then show *?thesis*
using *T0* **by** *presburger*
next
case *F*: *False*
then have *pre-to-univ-poly* (*Suc n*) *i* (*pvar R j*) = *ring.indexed-const* ($R[\mathcal{X}_n]$) (*pvar R* (*j*−*1*))
using *pre-to-univ-poly-is-hom*(*4*)[*of i Suc n*] *assms*(*1*) *diff-Suc-1*
unfolding *coord-ring-def*
by (*metis A*(*2*) *False lessThan-iff linorder-neqE-nat*)
then have *total-eval* ($R[\mathcal{X}_n]$) ($\lambda i.$ *coord-const x*) (*pre-to-univ-poly* (*Suc n*) *i* (*pvar R j*)) =
pvar R (*j*−*1*)
using *cring.total-eval-const*[*of* $R[\mathcal{X}_n]$]
by (*smt A*(*2*) *F False Suc-less-SucD add-diff-inverse-nat coord-cring-cring lessThan-iff less-one linorder-neqE-nat local.pvar-closed not-less0 plus-1-eq-Suc*)

then have *T0*: *eval-at-poly R* (*total-eval* ($R[\mathcal{X}_n]$) ($\lambda i.$ *coord-const x*) (*pre-to-univ-poly* (*Suc n*) *i* (*pvar R j*))) *a*
= *a*!(*j*−*1*)
using *eval-pvar*[*of j*−*1 n a*]
by (*metis A*(*2*) *F False One-nat-def Suc-diff-Suc Suc-less-eq assms*(*3*) *lessThan-iff linorder-neqE-nat minus-nat.diff-0 not-less0*)
have *T1*: *as*!*j* = *a*!(*j*−*1*)
proof−
obtain *k* **where** *k-def*: *j* = *i* + *1* + *k*
using *False F*
by (*metis Nat.add-0-right less-imp-add-positive less-one nat-neq-iff semiring-normalization-rules*(*25*))
show *as*!*j* = *a*!(*j*−*1*)
proof−
have *length* (*take i a*) = *i*
using *assms*

by (*meson cartesian-power-car-memE less-Suc-eq-le take-closed*)
then have *as!j* = (*x* # *drop i a*)!(*k*+*1*)
using *k-def assms*
unfolding *coord-ring-def*
by (*metis Suc-eq-plus1 add.assoc insert-at-index.simps nth-append-length-plus plus-1-eq-Suc*)
have *length* (*drop i a*) ≥ *k*
proof−
have *length* (*drop i a*) = *n* − *i*
using *assms cartesian-power-car-memE length-drop*
by *blast*
then show *?thesis*
using *assms k-def A*(*2*)
by (*metis Suc-eq-plus1 add.commute diff-Suc-Suc lessThan-iff less-diff-conv less-imp-le-nat*)
qed
then have *as!j* = (*drop i a*)! *k*
using *assms k-def*
by (*metis Nat.add-0-right One-nat-def* ‹*as* ! *j* = (*x* # *drop i a*) ! (*k* + *1*)› *add-Suc-right nth-Cons-Suc*)
then show *?thesis* **using** *k-def assms*
by (*metis Nat.add-diff-assoc2 add-diff-cancel-right′ cartesian-power-car-memE le-add2 less-Suc-eq-le nth-drop*)
qed
qed
then show *?thesis*
using *T0* **by** *presburger*
qed
qed
then show *eval-at-point R as* (*p* ⊗$_{Pring\ R\ \{..<Suc\ n\}}$ *pvar R j*) = *eval-at-point R a* (*total-eval* (*Pring R* {..<*n*}) (λ*i*. *R.indexed-const x*) (*pre-to-univ-poly* (*Suc n*) *i* (*p* ⊗$_{Pring\ R\ \{..<Suc\ n\}}$ *pvar R j*)))
using *A*(*3*) *A0 A7*
unfolding *coord-ring-def*
by *presburger*
qed
qed

definition *pre-to-univ-poly-inv-hom* ::
nat ⇒ *nat* ⇒ (('*a*, *nat*) *mvar-poly*,('*a*, *nat*) *mvar-poly*) *ring-hom* **where**
pre-to-univ-poly-inv-hom n i = *R.relabel-vars* {..<(*n*−*1*)} {..<*n*} (λ*j*. *if j* < *i then j else j* + *1*)

lemma *pre-to-univ-poly-inv-hom-is-hom*:
assumes *i* < *Suc n*
shows *ring-hom-ring* (*R*[$\mathcal{X}_n$]) (*R*[$\mathcal{X}_{Suc\ n}$]) (*pre-to-univ-poly-inv-hom* (*Suc n*) *i*)
proof−
have *0*: *ring-hom-ring* (*R*[$\mathcal{X}_n$]) (*R*[$\mathcal{X}_{Suc\ n}$]) (*R.relabel-vars* {..<*n*} {..<*Suc n*} (λ*j*. *if j* < *i then j else j* + *1*))

unfolding *coord-ring-def*
apply(*rule R.relabel-vars-is-morphism*)
using *assms*
by (*smt Pi-I Suc-eq-plus1 add-less-cancel-right lessThan-iff less-Suc-eq*)
then show *?thesis*
unfolding *pre-to-univ-poly-inv-hom-def*
by *simp*
qed

lemma *pre-to-univ-poly-inv-hom-const*:
assumes $i < Suc\ n$
assumes $k \in carrier\ R$
shows (*pre-to-univ-poly-inv-hom* (*Suc n*) *i*) (*R.indexed-const k*) = *R.indexed-const k*
proof−
have *0*: (*R.relabel-vars* {..<*n*} {..<*Suc n*} (λ*j*. *if j < i then j else j + 1*)) (*R.indexed-const k*) = *R.indexed-const k*
unfolding *coord-ring-def*
apply(*rule R.relabel-vars-is-morphism*)
using *assms*
apply (*smt Pi-I Suc-eq-plus1 add-less-cancel-right lessThan-iff less-Suc-eq*)
using *assms*(*2*) **by** *blast*
then show *?thesis*
unfolding *pre-to-univ-poly-inv-hom-def*
using *diff-Suc-1* **by** *presburger*
qed

lemma *pre-to-univ-poly-inv-hom-pvar-0*:
assumes $i < Suc\ n$
assumes $j < i$
shows *pre-to-univ-poly-inv-hom* (*Suc n*) *i* (*pvar R j*) =
pvar R j
unfolding *pre-to-univ-poly-inv-hom-def coord-ring-def*
using *R.relabel-vars-is-morphism*(*2*)[*of* λ*j*. *if j < i then j else j + 1* {..<*n*} {..<*Suc n*} *j*]
by (*smt Pi-I add.commute add-diff-cancel-left' assms*(*1*) *assms*(*2*) *lessThan-iff less-Suc-eq less-trans-Suc not-less-eq plus-1-eq-Suc*)

lemma *pre-to-univ-poly-inv-hom-pvar-1*:
assumes $i < Suc\ n$
assumes $i \leq j$
assumes $j < n$
shows *pre-to-univ-poly-inv-hom* (*Suc n*) *i* (*pvar R j*) =
pvar R (*j + 1*)
unfolding *pre-to-univ-poly-inv-hom-def*
using *assms R.relabel-vars-is-morphism*(*2*)[*of* λ*j*. *if j < i then j else j + 1* {..<*n*} {..< *Suc n*} *j*]
by (*smt Pi-I add.commute add-less-cancel-right diff-Suc-1 lessThan-iff less-Suc-eq not-le plus-1-eq-Suc*)

definition *pre-to-univ-poly-inv-var-ass* ::
nat ⇒ *nat* ⇒ *nat* ⇒ ('*a*, *nat*) *mvar-poly* **where**
pre-to-univ-poly-inv-var-ass n i j = *pvar R i*

lemma *pre-to-univ-poly-inv-var-ass-closed*:
assumes *i* < *Suc n*
shows *pre-to-univ-poly-inv-var-ass* (*Suc n*) *i* ∈ {*i*} → *carrier* (*R*[$\mathcal{X}_{Suc\ n}$])
by (*metis Pi-I assms local.pvar-closed pre-to-univ-poly-inv-var-ass-def*)

definition *pre-to-univ-poly-inv* ::
nat ⇒ *nat* ⇒ ((('*a*, *nat*) *mvar-poly*, *nat*) *mvar-poly*,('*a*, *nat*) *mvar-poly*) *ring-hom*
where
pre-to-univ-poly-inv n i = *indexed-poly-induced-morphism* {*i*} (*R*[$\mathcal{X}_n$])
(*pre-to-univ-poly-inv-hom n i*) (*pre-to-univ-poly-inv-var-ass*
n i)

lemma *pre-to-univ-poly-inv-is-hom*:
assumes *i* < *Suc n*
shows *ring-hom-ring* (*Pring* (*R*[$\mathcal{X}_n$]) {*i*}) (*R*[$\mathcal{X}_{Suc\ n}$]) (*pre-to-univ-poly-inv* (*Suc*
n) *i*)
apply(*rule cring.Pring-universal-prop*[*of - - pre-to-univ-poly-inv-var-ass* (*Suc n*)
i {*i*} *pre-to-univ-poly-inv-hom* (*Suc n*) *i*])
unfolding *coord-ring-def*
apply (*simp add*: *R.Pring-is-cring R.is-cring*)
apply (*simp add*: *R.Pring-is-cring R.is-cring*)
apply (*metis Pi-I R.Pring-var-closed assms lessThan-iff pre-to-univ-poly-inv-var-ass-def*)
apply (*metis assms coord-ring-def pre-to-univ-poly-inv-hom-is-hom*)
by (*simp add*: *coord-ring-def pre-to-univ-poly-inv-def*)

lemma *pre-to-univ-poly-inv-pvar*:
assumes *i* < *Suc n*
shows (*pre-to-univ-poly-inv* (*Suc n*) *i*) (*pvar* (*R*[$\mathcal{X}_n$]) *i*) = *pvar R i*
using *assms cring.Pring-universal-prop*[*of R*[$\mathcal{X}_n$] *R*[$\mathcal{X}_{Suc\ n}$]
pre-to-univ-poly-inv-var-ass (*Suc n*) *i* {*i*} *pre-to-univ-poly-inv-hom*
(*Suc n*) *i*]
by (*metis Pi-I coord-cring-cring cring-coord-rings.pre-to-univ-poly-inv-var-ass-def*
cring-coord-rings-axioms local.pvar-closed pre-to-univ-poly-inv-def
pre-to-univ-poly-inv-hom-is-hom singletonI var-to-IP-def)

lemma *pre-to-univ-poly-inv-const*:
assumes *i* < *Suc n*
assumes *p* ∈ *carrier* (*R*[$\mathcal{X}_n$])
shows (*pre-to-univ-poly-inv* (*Suc n*) *i*) (*ring.indexed-const* (*R*[$\mathcal{X}_n$]) *p*) = *pre-to-univ-poly-inv-hom*
(*Suc n*) *i p*
using *assms cring.Pring-universal-prop*[*of R*[$\mathcal{X}_n$] *R*[$\mathcal{X}_{Suc\ n}$]
pre-to-univ-poly-inv-var-ass (*Suc n*) *i* {*i*} *pre-to-univ-poly-inv-hom*
(*Suc n*) *i*]
by (*metis Pi-I coord-cring-cring cring-coord-rings.pre-to-univ-poly-inv-var-ass-def*

cring-coord-rings-axioms local.pvar-closed pre-to-univ-poly-inv-def pre-to-univ-poly-inv-hom-is-hom)

lemma *pre-to-univ-poly-inverse*:
assumes $i < Suc\ n$
assumes $p \in carrier\ (R[\mathcal{X}_{Suc\ n}])$
shows *pre-to-univ-poly-inv* (*Suc n*) *i* (*pre-to-univ-poly* (*Suc n*) *i p*) = *p*
apply(*rule R.Pring-car-induct''*[*of p* {..<*Suc n*}])
using *assms coord-ring-def* **apply** *metis*
proof−
show *0*: $\bigwedge c.\ c \in carrier\ R \Longrightarrow$ *pre-to-univ-poly-inv* (*Suc n*) *i* (*pre-to-univ-poly* (*Suc n*) *i* (*coord-const c*)) = *coord-const c*
proof−
fix *c* **assume** *A*: *c* ∈ *carrier R*
have *0*: *pre-to-univ-poly* (*Suc n*) *i* (*coord-const c*) =
MP.indexed-const n (*coord-const c*)
using *A assms*(*1*) *diff-Suc-1 pre-to-univ-poly-is-hom*(*5*) **by** *presburger*
have *1*: (λ*j*. *if j* < *i then j else j* + *1*) ∈ {..<*n*} → {..<*Suc n*}
by (*smt Pi-I Suc-eq-plus1 add-less-cancel-right lessThan-iff less-Suc-eq*)
have *2*: *pre-to-univ-poly-inv-hom* (*Suc n*) *i* (*coord-const c*) = *coord-const c*
unfolding *pre-to-univ-poly-inv-hom-def*
using *1 R.relabel-vars-is-morphism*(*3*)[*of* (λ*j*. *if j* < *i then j else j* + *1*) {..<*n*} {..<*Suc n*} *c*]
unfolding *coord-ring-def*
using *A diff-Suc-1* **by** *presburger*
show *pre-to-univ-poly-inv* (*Suc n*) *i* (*pre-to-univ-poly* (*Suc n*) *i* (*coord-const c*)) = *coord-const c*
using *0 1 2*
by (*metis* (*no-types*, *lifting*) *A R.indexed-const-closed assms*(*1*) *coord-ring-def pre-to-univ-poly-inv-const*)
qed
show *1*: $\bigwedge p\ q.\ p \in carrier\ (Pring\ R\ \{..<Suc\ n\}) \Longrightarrow$
$q \in carrier\ (Pring\ R\ \{..<Suc\ n\}) \Longrightarrow$
pre-to-univ-poly-inv (*Suc n*) *i* (*pre-to-univ-poly* (*Suc n*) *i p*) =
p ⟹
pre-to-univ-poly-inv (*Suc n*) *i* (*pre-to-univ-poly* (*Suc n*) *i q*) =
q ⟹
pre-to-univ-poly-inv (*Suc n*) *i*
(*pre-to-univ-poly* (*Suc n*) *i* ($p \oplus_{Pring\ R\ \{..<Suc\ n\}}\ q$)) =
$p \oplus_{Pring\ R\ \{..<Suc\ n\}}\ q$
proof− **fix** *p q* **assume** *A*: *p* ∈ *carrier* (*Pring R* {..<*Suc n*})
q ∈ *carrier* (*Pring R* {..<*Suc n*})
pre-to-univ-poly-inv (*Suc n*) *i* (*pre-to-univ-poly* (*Suc n*) *i p*) = *p*
pre-to-univ-poly-inv (*Suc n*) *i* (*pre-to-univ-poly* (*Suc n*) *i q*) = *q*
have *0*: (*pre-to-univ-poly* (*Suc n*) *i* ($p \oplus_{R[\mathcal{X}_{Suc\ n}]}\ q$)) =
(*pre-to-univ-poly* (*Suc n*) *i p*) $\oplus_{Pring\ (R[\mathcal{X}n])\ \{i\}}$ *pre-to-univ-poly* (*Suc n*) *i q*
using *pre-to-univ-poly-is-hom*(*1*)[*of i Suc n*] *ring-hom-ring.homh ring-hom-add A*
unfolding *coord-ring-def*

by (*metis* (*mono-tags*, *lifting*) *assms*(1) *diff-Suc-1*)
have 1: *pre-to-univ-poly* (*Suc n*) *i p* ∈ *carrier* (*Pring* (*R*[$\mathcal{X}_n$]) {*i*})
using *pre-to-univ-poly-is-hom*(1)[*of i Suc n*] *A*
unfolding *coord-ring-def*
by (*metis assms*(1) *coord-ring-def diff-Suc-1 pre-to-univ-poly-is-hom*(6))
have 2: *pre-to-univ-poly* (*Suc n*) *i q* ∈ *carrier* (*Pring* (*R*[$\mathcal{X}_n$]) {*i*})
using *pre-to-univ-poly-is-hom*(1)[*of i Suc n*] *A*
unfolding *coord-ring-def*
by (*metis assms*(1) *coord-ring-def diff-Suc-1 pre-to-univ-poly-is-hom*(6))
show *pre-to-univ-poly-inv* (*Suc n*) *i*
(*pre-to-univ-poly* (*Suc n*) *i* (*p* $\oplus_{Pring\ R\ \{..<Suc\ n\}}$ *q*)) =
p $\oplus_{Pring\ R\ \{..<Suc\ n\}}$ *q*
using *0 1 2 A pre-to-univ-poly-inv-is-hom*[*of i n*] *ring-hom-ring.homh ring-hom-add*
unfolding *coord-ring-def*
by (*smt assms*(1))
qed
show ⋀*p ia*.
p ∈ *carrier* (*Pring R* {..<*Suc n*}) ⟹
ia ∈ {..<*Suc n*} ⟹
pre-to-univ-poly-inv (*Suc n*) *i* (*pre-to-univ-poly* (*Suc n*) *i p*) = *p* ⟹
pre-to-univ-poly-inv (*Suc n*) *i*
(*pre-to-univ-poly* (*Suc n*) *i* (*p* $\otimes_{Pring\ R\ \{..<Suc\ n\}}$ *pvar R ia*)) =
p $\otimes_{Pring\ R\ \{..<Suc\ n\}}$ *pvar R ia*
proof− **fix** *p j*
assume *A*: *p* ∈ *carrier* (*Pring R* {..<*Suc n*}) *j* ∈ {..<*Suc n*}
pre-to-univ-poly-inv (*Suc n*) *i* (*pre-to-univ-poly* (*Suc n*) *i p*) = *p*
have 0: (*pre-to-univ-poly* (*Suc n*) *i* (*p* $\otimes_{R[\mathcal{X}_{Suc\ n}]}$ *pvar R j*))
= (*pre-to-univ-poly* (*Suc n*) *i p*) $\otimes_{Pring\ (R[\mathcal{X}_n])\ \{i\}}$ *pre-to-univ-poly* (*Suc n*) *i* (*pvar R j*)
using *pre-to-univ-poly-is-hom*(1)[*of i Suc n*] *ring-hom-ring.homh ring-hom-mult A*
unfolding *coord-ring-def*
by (*metis R.Pring-var-closed assms*(1) *diff-Suc-1*)
have 1: *pre-to-univ-poly* (*Suc n*) *i p* ∈ *carrier* (*Pring* (*R*[$\mathcal{X}_n$]) {*i*})
using *pre-to-univ-poly-is-hom*(1)[*of i Suc n*] *A*
unfolding *coord-ring-def*
by (*metis assms*(1) *coord-ring-def diff-Suc-1 pre-to-univ-poly-is-hom*(6))
have 1: *pre-to-univ-poly* (*Suc n*) *i* (*pvar R j*) ∈ *carrier* (*Pring* (*R*[$\mathcal{X}_n$]) {*i*})
using *pre-to-univ-poly-is-hom*(1)[*of i Suc n*] *A*
unfolding *coord-ring-def*
by (*metis R.Pring-var-closed assms*(1) *coord-ring-def diff-Suc-1 pre-to-univ-poly-is-hom*(6))
have 2: *pre-to-univ-poly-inv* (*Suc n*) *i* (*pre-to-univ-poly* (*Suc n*) *i* (*pvar R j*))
= *pvar R j*
proof(*cases j* = *i*)
case *True*
then have (*pre-to-univ-poly* (*Suc n*) *i* (*pvar R j*)) = *pvar* (*R*[$\mathcal{X}_n$]) *j*
using *pre-to-univ-poly-is-hom*(3)[*of i Suc n*] *assms*(1) *diff-Suc-1* **by** *presburger*

```
      then show ?thesis
      unfolding coord-ring-def
     using True ‹pre-to-univ-poly (Suc n) i (pvar R j) = pvar (R[\<X>_n]) j› assms(1)
pre-to-univ-poly-inv-pvar by presburger
    next
      case False
      show ?thesis
      proof(cases j < i)
        case True
          then have (pre-to-univ-poly (Suc n) i (pvar R j)) = ring.indexed-const
(R[\<X>_n]) (pvar R j)
          using pre-to-univ-poly-is-hom(2) [of i Suc n] assms(1) diff-Suc-1
           unfolding coord-ring-def

          by presburger
        then show ?thesis
          using pre-to-univ-poly-inv-const[of i n (pvar R j)]
                pre-to-univ-poly-inv-hom-pvar-0[of i n j]
          by (metis Suc-less-eq True assms(1) less-trans-Suc local.pvar-closed)
      next
        case F: False
          then have (pre-to-univ-poly (Suc n) i (pvar R j)) = ring.indexed-const
(R[\<X>_n]) (pvar R (j-1))
          using pre-to-univ-poly-is-hom(4)[of i Suc n] assms(1) diff-Suc-1
              unfolding coord-ring-def
 by (metis A(2) False lessThan-iff linorder-neqE-nat)
        then show ?thesis
          using pre-to-univ-poly-inv-const[of i n (pvar R (j-1))]
                pre-to-univ-poly-inv-hom-pvar-0[of i n j-1]
             by (metis (no-types, lifting) A(2) F False One-nat-def Suc-eq-plus1
add-diff-inverse-nat
                  assms(1) le-neq-implies-less lessThan-iff less-one local.pvar-closed
nat-le-linear
            not-less-eq plus-1-eq-Suc pre-to-univ-poly-inv-hom-pvar-1)
      qed
    qed
    show pre-to-univ-poly-inv (Suc n) i
       (pre-to-univ-poly (Suc n) i (p ⊗_Pring R {..<Suc n} pvar R j)) =
      p ⊗_Pring R {..<Suc n} pvar R j
      using 0 1 2 A pre-to-univ-poly-inv-is-hom[of i n]
           ring-hom-ring.homh[of - - pre-to-univ-poly-inv (Suc n) i ]
           ring-hom-mult[of pre-to-univ-poly-inv (Suc n) i ]
      unfolding coord-ring-def
      by (smt assms(1) coord-ring-def diff-Suc-1 pre-to-univ-poly-is-hom(6))
  qed
qed

lemma coord-ring-car-induct:
  assumes Q ∈ carrier (R[\<X>_n])
```

assumes $\bigwedge c.\ c \in carrier\ R \Longrightarrow A\ (R.indexed\text{-}const\ c)$
assumes $\bigwedge p\ q.\ p \in carrier\ (R[\mathcal{X}_n]) \Longrightarrow q \in carrier\ (R[\mathcal{X}_n]) \Longrightarrow A\ p \Longrightarrow A\ q \Longrightarrow A\ (p \oplus_{R[\mathcal{X}_n]} q)$
assumes $\bigwedge p\ i.\ p \in carrier\ (R[\mathcal{X}_n]) \Longrightarrow i < n \Longrightarrow A\ p \Longrightarrow A\ (p \otimes_{R[\mathcal{X}_n]} pvar\ R\ i)$
shows $A\ Q$
unfolding *coord-ring-def* **apply**(*rule R.Pring-car-induct''*[*of* - {..<*n*}])
apply (*metis assms*(*1*) *coord-ring-def*)
using *assms*(*2*) **apply** *auto*[*1*]
apply (*metis assms*(*3*) *coord-ring-def*)
by (*metis assms*(*4*) *coord-ring-def lessThan-iff*)

lemma *pre-to-univ-poly-inverse'*:
assumes $i < Suc\ n$
assumes $p \in carrier\ (R[\mathcal{X}_n])$
shows *pre-to-univ-poly* (*Suc n*) *i* (*pre-to-univ-poly-inv* (*Suc n*) *i* (*MP.indexed-const n p*)) = *MP.indexed-const n p*
apply(*rule coord-ring-car-induct*[*of* - *n*])
using *assms*(*2*) **apply** *blast*
proof−
show $\bigwedge c.\ c \in carrier\ R \Longrightarrow$
pre-to-univ-poly (*Suc n*) *i* (*pre-to-univ-poly-inv* (*Suc n*) *i* (*MP.indexed-const n* (*R.indexed-const c*))) =
MP.indexed-const n (*R.indexed-const c*)
proof− **fix** *k* **assume** *A*: $k \in carrier\ R$
have *0*: *R.indexed-const k* $\in carrier\ (R\ [\mathcal{X}_n])$
using *A*
by (*metis coord-ring-def R.indexed-const-closed*)
have *1*: *pre-to-univ-poly-inv* (*Suc n*) *i* (*MP.indexed-const n* (*R.indexed-const k*)) = *pre-to-univ-poly-inv-hom* (*Suc n*) *i* (*R.indexed-const k*)
using *0 assms pre-to-univ-poly-inv-const*[*of i n R.indexed-const k*]
by *linarith*
have *pre-to-univ-poly-inv-hom* (*Suc n*) *i* (*R.indexed-const k*) = *R.indexed-const k*
using *A pre-to-univ-poly-inv-hom-const*[*of i n k*] *assms*
by *blast*
thus *pre-to-univ-poly* (*Suc n*) *i* (*pre-to-univ-poly-inv* (*Suc n*) *i* (*MP.indexed-const n* (*R.indexed-const k*))) = *MP.indexed-const n* (*R.indexed-const k*)
using *1*
by (*metis A assms*(*1*) *coord-ring-def diff-Suc-1 pre-to-univ-poly-is-hom*(*5*))
qed
show $\bigwedge p\ q.\ p \in carrier\ (R\ [\mathcal{X}_n]) \Longrightarrow$
$q \in carrier\ (R\ [\mathcal{X}_n]) \Longrightarrow$
pre-to-univ-poly (*Suc n*) *i* (*pre-to-univ-poly-inv* (*Suc n*) *i* (*MP.indexed-const n p*)) = *MP.indexed-const n p* ⟹
pre-to-univ-poly (*Suc n*) *i* (*pre-to-univ-poly-inv* (*Suc n*) *i* (*MP.indexed-const n q*)) = *MP.indexed-const n q* ⟹
pre-to-univ-poly (*Suc n*) *i* (*pre-to-univ-poly-inv* (*Suc n*) *i* (*MP.indexed-const n* ($p \oplus_{R\ [\mathcal{X}_n]} q$))) = *MP.indexed-const n* ($p \oplus_{R\ [\mathcal{X}_n]} q$)

proof− **fix** p Q
assume A: $p \in carrier\ (R\ [\mathcal{X}_n])$
$pre\text{-}to\text{-}univ\text{-}poly\ (Suc\ n)\ i\ (pre\text{-}to\text{-}univ\text{-}poly\text{-}inv\ (Suc\ n)\ i\ (MP.indexed\text{-}const\ n\ p)) = MP.indexed\text{-}const\ n\ p$
$Q \in carrier\ (R\ [\mathcal{X}_n])$
$pre\text{-}to\text{-}univ\text{-}poly\ (Suc\ n)\ i\ (pre\text{-}to\text{-}univ\text{-}poly\text{-}inv\ (Suc\ n)\ i\ (MP.indexed\text{-}const\ n\ Q)) = MP.indexed\text{-}const\ n\ Q$
have *0*: $p \bigoplus Q = p \oplus_{R[\mathcal{X}_n]} Q$
by (*metis R.Pring-add coord-ring-def*)
have *1*: $MP.indexed\text{-}const\ n\ (p \bigoplus Q) = (MP.indexed\text{-}const\ n\ p) \oplus_{Pring\ (R[\mathcal{X}_n])\ \{i\}} (MP.indexed\text{-}const\ n\ Q)$
by (*metis 0 MP.Pring-add MP.indexed-padd-const*)
have *2*: $MP.indexed\text{-}const\ n\ p \in carrier\ (Pring\ (R\ [\mathcal{X}_n])\ \{i\})$
using *A* **unfolding** *coord-ring-def*
by (*metis MP.indexed-const-closed R.Pring-car coord-ring-def*)
have *3*: $MP.indexed\text{-}const\ n\ Q \in carrier\ (Pring\ (R\ [\mathcal{X}_n])\ \{i\})$
using *A(3) MP.indexed-const-closed* **by** *blast*
have *4*: $(pre\text{-}to\text{-}univ\text{-}poly\text{-}inv\ (Suc\ n)\ i\ (MP.indexed\text{-}const\ n\ (p \oplus_{R\ [\mathcal{X}_n]} Q))) =$
$pre\text{-}to\text{-}univ\text{-}poly\text{-}inv\ (Suc\ n)\ i\ (MP.indexed\text{-}const\ n\ p) \oplus_{R[\mathcal{X}_{Suc\ n}]} pre\text{-}to\text{-}univ\text{-}poly\text{-}inv\ (Suc\ n)\ i\ (MP.indexed\text{-}const\ n\ Q)$
using *pre-to-univ-poly-is-hom(1)*[*of i Suc n*]
pre-to-univ-poly-inv-is-hom(1)[*of i n*]
ring-hom-add[*of pre-to-univ-poly-inv (Suc n) i (Pring (R [*$\mathcal{X}_n$*]) {i}) (R [*$\mathcal{X}_{Suc\ n}$*]) MP.indexed-const n p MP.indexed-const n Q*]
ring-hom-ring.homh
MP.indexed-const-closed[*of p n {i}*]
MP.indexed-const-closed[*of Q n {i}*] *A R.Pring-car*[*of {..<n}*] **unfolding** *coord-ring-def*
by (*metis 0 1 assms(1) coord-ring-def*)
have *5*: $pre\text{-}to\text{-}univ\text{-}poly\ (Suc\ n)\ i\ (pre\text{-}to\text{-}univ\text{-}poly\text{-}inv\ (Suc\ n)\ i\ (MP.indexed\text{-}const\ n\ (p \oplus_{R\ [\mathcal{X}_n]} Q))) =$
$pre\text{-}to\text{-}univ\text{-}poly\ (Suc\ n)\ i\ (pre\text{-}to\text{-}univ\text{-}poly\text{-}inv\ (Suc\ n)\ i\ (MP.indexed\text{-}const\ n\ p)) \oplus_{Pring\ (R[\mathcal{X}_n])\ \{i\}} pre\text{-}to\text{-}univ\text{-}poly\ (Suc\ n)\ i\ (pre\text{-}to\text{-}univ\text{-}poly\text{-}inv\ (Suc\ n)\ i\ (MP.indexed\text{-}const\ n\ Q))$
proof−
have *50*: $pre\text{-}to\text{-}univ\text{-}poly\ (Suc\ n)\ i \in ring\text{-}hom\ (R\ [\mathcal{X}_{Suc\ n}])\ (Pring\ (R\ [\mathcal{X}_{Suc\ n\ -\ 1}])\ \{i\})$
using *pre-to-univ-poly-is-hom(1)*[*of i Suc n*] *ring-hom-ring.homh*
by (*metis assms(1)*)
have *51*: $pre\text{-}to\text{-}univ\text{-}poly\text{-}inv\ (Suc\ n)\ i \in ring\text{-}hom\ (Pring\ (R\ [\mathcal{X}_{Suc\ n\ -\ 1}])\ \{i\})\ (R\ [\mathcal{X}_{Suc\ n}])$
using *pre-to-univ-poly-inv-is-hom ring-hom-ring.homh*
by (*metis assms(1) diff-Suc-1*)
have *52*: $pre\text{-}to\text{-}univ\text{-}poly\text{-}inv\ (Suc\ n)\ i\ (MP.indexed\text{-}const\ n\ p) \in carrier\ (R\ [\mathcal{X}_{Suc\ n}])$
using *51 ring-hom-closed*[*of pre-to-univ-poly-inv (Suc n) i*]

by (*smt 2 diff-Suc-1*)
have *53*: *pre-to-univ-poly-inv* (*Suc n*) *i* (*MP.indexed-const n Q*) ∈ *carrier* (*R* [$\mathcal{X}_{Suc\ n}$])
using *51 ring-hom-closed*[*of pre-to-univ-poly-inv* (*Suc n*) *i*]
by (*smt 3 diff-Suc-1*)
show *?thesis* **using** *50 51 52 53*
using *pre-to-univ-poly-is-hom*(*1*)[*of i Suc n*]
ring-hom-add[*of pre-to-univ-poly* (*Suc n*) *i R* [$\mathcal{X}_{Suc\ n}$] *Pring* (*R* [$\mathcal{X}_{Suc\ n\ -\ 1}$]) {*i*}
pre-to-univ-poly-inv (*Suc n*) *i* (*MP.indexed-const n p*)
pre-to-univ-poly-inv (*Suc n*) *i* (*MP.indexed-const n Q*)] *4*
by (*metis diff-Suc-1*)
qed
show *pre-to-univ-poly* (*Suc n*) *i* (*pre-to-univ-poly-inv* (*Suc n*) *i* (*MP.indexed-const n* (*p* ⊕$_{R\ [\mathcal{X}_n]}$ *Q*))) = *MP.indexed-const n* (*p* ⊕$_{R\ [\mathcal{X}_n]}$ *Q*)
using *5 A 0 1* **by** *metis*
qed
show ⋀*p ia*.
p ∈ *carrier* (*R* [$\mathcal{X}_n$]) ⟹
ia < *n* ⟹
pre-to-univ-poly (*Suc n*) *i* (*pre-to-univ-poly-inv* (*Suc n*) *i* (*MP.indexed-const n p*)) = *MP.indexed-const n p* ⟹
pre-to-univ-poly (*Suc n*) *i* (*pre-to-univ-poly-inv* (*Suc n*) *i* (*MP.indexed-const n* (*p* ⊗$_{R\ [\mathcal{X}_n]}$ *pvar R ia*))) = *MP.indexed-const n* (*p* ⊗$_{R\ [\mathcal{X}_n]}$ *pvar R ia*)
proof− **fix** *p j*
assume *A*: *p* ∈ *carrier* (*R* [$\mathcal{X}_n$]) *j* < *n*
pre-to-univ-poly (*Suc n*) *i* (*pre-to-univ-poly-inv* (*Suc n*) *i* (*MP.indexed-const n p*)) = *MP.indexed-const n p*
show *pre-to-univ-poly* (*Suc n*) *i* (*pre-to-univ-poly-inv* (*Suc n*) *i* (*MP.indexed-const n* (*p* ⊗$_{R\ [\mathcal{X}_n]}$ *pvar R j*))) = *MP.indexed-const n* (*p* ⊗$_{R\ [\mathcal{X}_n]}$ *pvar R j*)
proof−
have *0*: *pre-to-univ-poly-inv* (*Suc n*) *i* ∈ *ring-hom* (*Pring* (*R* [$\mathcal{X}_{Suc\ n\ -\ 1}$]) {*i*}) (*R* [$\mathcal{X}_{Suc\ n}$])
using *pre-to-univ-poly-inv-is-hom*(*1*)[*of i n*] *ring-hom-ring.homh*
by (*metis assms*(*1*) *diff-Suc-1*)
have *1*: *MP.indexed-const n* (*p* ⊗$_{R\ [\mathcal{X}_n]}$ (*pvar R j*)) = *MP.indexed-const n p* ⊗$_{Pring\ (R[\mathcal{X}_n])\ \{i\}}$ *MP.indexed-const n* (*pvar R j*)
by (*metis A*(*1*) *A*(*2*) *MP.indexed-const-mult local.pvar-closed*)
have *2*: (*pre-to-univ-poly-inv* (*Suc n*) *i* (*MP.indexed-const n* (*p* ⊗$_{R\ [\mathcal{X}_n]}$ *pvar R j*))) =
pre-to-univ-poly-inv (*Suc n*) *i* (*MP.indexed-const n p*) ⊗$_{(R\ [\mathcal{X}_{Suc\ n}])}$ *pre-to-univ-poly-inv* (*Suc n*) *i* (*MP.indexed-const n* (*pvar R j*))
using *0 1 ring-hom-mult A*
by (*metis* (*no-types*, *lifting*) *MP.indexed-const-closed diff-Suc-1 local.pvar-closed*)
have *3*: *pre-to-univ-poly*(*Suc n*) *i* ∈ *ring-hom* (*R* [$\mathcal{X}_{Suc\ n}$]) (*Pring* (*R* [$\mathcal{X}_{Suc\ n\ -\ 1}$]) {*i*})
using *assms*(*1*) *pre-to-univ-poly-is-hom*(*1*) *ring-hom-ring.homh* **by** *blast*

have *4*: *pre-to-univ-poly* (*Suc n*) *i* (*pre-to-univ-poly-inv* (*Suc n*) *i* (*MP.indexed-const n* (*p* $\otimes_{R\ [\mathcal{X}_n]}$ *pvar R j*))) =
pre-to-univ-poly (*Suc n*) *i* (*pre-to-univ-poly-inv* (*Suc n*) *i* (*MP.indexed-const n p*)) $\otimes_{Pring\ (R\ [\mathcal{X}_n])\ \{i\}}$ *pre-to-univ-poly* (*Suc n*) *i* (*pre-to-univ-poly-inv* (*Suc n*) *i* (*MP.indexed-const n* (*pvar R j*)))
using *2 3 ring-hom-mult*
by (*smt 0 A*(*1*) *A*(*2*) *MP.indexed-const-closed diff-Suc-1 local.pvar-closed ring-hom-closed*)
have *5*: *pre-to-univ-poly* (*Suc n*) *i* (*pre-to-univ-poly-inv* (*Suc n*) *i* (*MP.indexed-const n* (*p* $\otimes_{R\ [\mathcal{X}_n]}$ *pvar R j*))) =
MP.indexed-const n p $\otimes_{Pring\ (R\ [\mathcal{X}_n])\ \{i\}}$ *pre-to-univ-poly* (*Suc n*) *i* (*pre-to-univ-poly-inv* (*Suc n*) *i* (*MP.indexed-const n* (*pvar R j*)))
using *A 4* **by** *presburger*
have *6*: *pre-to-univ-poly* (*Suc n*) *i* (*pre-to-univ-poly-inv* (*Suc n*) *i* (*MP.indexed-const n* (*pvar R j*))) = (*MP.indexed-const n* (*pvar R j*))
proof−
have (*pre-to-univ-poly-inv* (*Suc n*) *i* (*MP.indexed-const n* (*pvar R j*))) = *pre-to-univ-poly-inv-hom* (*Suc n*) *i* (*pvar R j*)
using *A*(*2*) *assms*(*1*) *local.pvar-closed pre-to-univ-poly-inv-const* **by** *blast*
hence *pre-to-univ-poly* (*Suc n*) *i* (*pre-to-univ-poly-inv* (*Suc n*) *i* (*MP.indexed-const n* (*pvar R j*))) =
pre-to-univ-poly (*Suc n*) *i* (*pre-to-univ-poly-inv-hom* (*Suc n*) *i* (*pvar R j*))
by *presburger*
show *?thesis*
proof(*cases j* < *i*)
case *True*
then have (*pre-to-univ-poly-inv-hom* (*Suc n*) *i* (*pvar R j*)) = (*pvar R j*)
using *pre-to-univ-poly-inv-hom-pvar-0*[*of i n j*] *assms*(*1*) **by** *blast*
thus *?thesis* **using** *pre-to-univ-poly-is-hom*
by (*metis True* ‹*pre-to-univ-poly-inv* (*Suc n*) *i* (*MP.indexed-const n* (*pvar R j*)) = *pre-to-univ-poly-inv-hom* (*Suc n*) *i* (*pvar R j*)› *assms*(*1*) *coord-ring-def diff-Suc-1*)
next
case *False*
have *pre-to-univ-poly-inv-hom* (*Suc n*) *i* (*pvar R j*) = *pvar R* (*j* + *1*)
using *pre-to-univ-poly-inv-hom-pvar-1*[*of i n j*] *A*(*2*) *False assms*(*1*) *not-le*
by *blast*
thus *?thesis* **using** *pre-to-univ-poly-is-hom*
by (*metis A*(*2*) *False Suc-eq-plus1* ‹*pre-to-univ-poly-inv* (*Suc n*) *i* (*MP.indexed-const n* (*pvar R j*)) = *pre-to-univ-poly-inv-hom* (*Suc n*) *i* (*pvar R j*)› *assms*(*1*) *coord-ring-def diff-Suc-1 not-less-eq*)
qed
qed
show *?thesis* **using** *6 A*

```
      using 1 5 by presburger
   qed
 qed
qed

definition to-univ-poly :: nat ⇒ nat ⇒
  (('a, nat) mvar-poly , ('a, nat) mvar-poly u-poly) ring-hom where
to-univ-poly n i  = IP-to-UP i ∘ (pre-to-univ-poly n i)

definition from-univ-poly :: nat ⇒ nat ⇒
      (('a, nat) mvar-poly u-poly , ('a, nat) mvar-poly) ring-hom where
from-univ-poly n i  = pre-to-univ-poly-inv n i ∘ (UP-to-IP (coord-ring R (n−1))
i)

lemma to-univ-poly-is-hom:
  assumes i ≤ n
  shows (to-univ-poly (Suc n) i) ∈ ring-hom (R[𝒳 Suc n]) (UP (R[𝒳 n]))
  unfolding to-univ-poly-def
  apply(rule ring-hom-trans[of - - Pring (R[𝒳 n]) {i}])
  using assms pre-to-univ-poly-is-hom ring-hom-ring.homh
   apply (metis diff-Suc-1 le-imp-less-Suc)
 using UP-cring.IP-to-UP-ring-hom[of (Pring R {..<n}) i] assms  ring-hom-ring.homh
  unfolding coord-ring-def UP-cring-def
  using R.Pring-is-cring R.is-cring by blast

lemma from-univ-poly-is-hom:
  assumes i ≤ n
  shows (from-univ-poly (Suc n) i) ∈ ring-hom  (UP (R[𝒳 n])) (R[𝒳 Suc n])
  unfolding from-univ-poly-def
  apply(rule ring-hom-trans[of - - Pring (R[𝒳 n]) {i}])
  using assms UP-cring.UP-to-IP-ring-hom[of coord-ring R (Suc n − 1) i]
          ring-hom-ring.homh[of UP (coord-ring R (Suc n − 1)) Pring (coord-ring
R (Suc n − 1)) {i} UP-to-IP (coord-ring R (Suc n − 1)) i]
  unfolding coord-ring-def UP-cring-def
  apply (metis R.Pring-is-cring diff-Suc-1 R.is-cring)
  using assms ring-hom-ring.homh le-imp-less-Suc pre-to-univ-poly-inv-is-hom
   unfolding coord-ring-def UP-cring-def
  by blast

lemma to-univ-poly-inverse:
  assumes i ≤ n
  assumes p ∈ carrier (R[𝒳 Suc n])
  shows from-univ-poly (Suc n) i (to-univ-poly (Suc n) i p) = p
proof−
  have 0: pre-to-univ-poly (Suc n) i p ∈ Pring-set (R[𝒳 n]) {i}
    using pre-to-univ-poly-is-hom(6)[of i Suc n - p] assms ring.Pring-car
     unfolding coord-ring-def UP-domain-def
    by (metis R.Pring-is-ring diff-Suc-1 le-imp-less-Suc)
  have 1: UP-to-IP (R[𝒳 n]) i
```

```
        (IP-to-UP i (pre-to-univ-poly (Suc n) i p)) =
        pre-to-univ-poly (Suc n) i p
    using 0 UP-cring.UP-to-IP-inv[of R[X_n] pre-to-univ-poly (Suc n) i p i ]
        R.Pring-is-cring
    unfolding coord-ring-def UP-cring-def
    using R.is-cring by blast
  have 2: from-univ-poly (Suc n) i (to-univ-poly (Suc n) i p) =
        (pre-to-univ-poly-inv (Suc n) i (
        (UP-to-IP (coord-ring R (Suc n − 1)) i) (
        (IP-to-UP i (
        (pre-to-univ-poly (Suc n) i) p)))))
    unfolding from-univ-poly-def to-univ-poly-def
  unfolding coord-ring-def
    by (metis comp-eq-dest-lhs)
  have 3: from-univ-poly (Suc n) i (to-univ-poly (Suc n) i p) =
        (pre-to-univ-poly-inv (Suc n) i (
        pre-to-univ-poly (Suc n) i p))
    using 0 1 2
  unfolding coord-ring-def
    using diff-Suc-1 by presburger
  then show ?thesis
    using pre-to-univ-poly-inverse assms(1) assms(2) less-Suc-eq-le
    by presburger
qed

lemma to-univ-poly-closed:
  assumes i ≤ n
  assumes p ∈ carrier (R[X_(Suc n)])
  shows to-univ-poly (Suc n) i p ∈ carrier (UP (R[X_n]))
  using to-univ-poly-is-hom[of i n] assms unfolding ring-hom-def
  by blast

lemma to-univ-poly-add:
  assumes i ≤ n
  assumes p ∈ carrier (R[X_(Suc n)])
  assumes Q ∈ carrier (R[X_(Suc n)])
  shows to-univ-poly (Suc n) i (p ⊕_(R[X_(Suc n)]) Q) =
      to-univ-poly (Suc n) i p ⊕_(UP (R[X_n])) to-univ-poly (Suc n) i Q
  using to-univ-poly-is-hom ring-hom-add
  by (metis assms(1) assms(2) assms(3))

lemma to-univ-poly-mult:
  assumes i ≤ n
  assumes p ∈ carrier (R[X_(Suc n)])
  assumes Q ∈ carrier (R[X_(Suc n)])
  shows to-univ-poly (Suc n) i (p ⊗_(R[X_(Suc n)]) Q) =
      to-univ-poly (Suc n) i p ⊗_(UP (R[X_n])) to-univ-poly (Suc n) i Q
  using to-univ-poly-is-hom ring-hom-mult
```

by (*metis assms*(*1*) *assms*(*2*) *assms*(*3*))

lemma *from-univ-poly-closed*:
assumes $i \leq n$
assumes $p \in carrier\ (UP\ (R[\mathcal{X}_n]))$
shows *from-univ-poly* (*Suc n*) *i p* $\in carrier\ (R[\mathcal{X}_{Suc\ n}])$
using *from-univ-poly-is-hom*[*of i n*] *assms* **unfolding** *ring-hom-def*
by *blast*

lemma *from-univ-poly-add*:
assumes $i \leq n$
assumes $p \in carrier\ (UP\ (R[\mathcal{X}_n]))$
assumes $Q \in carrier\ (UP\ (R[\mathcal{X}_n]))$
shows *from-univ-poly* (*Suc n*) *i* $(p \oplus_{UP\ (R[\mathcal{X}_n])} Q) =$
from-univ-poly (*Suc n*) *i p* $\oplus_{R[\mathcal{X}_{Suc\ n}]}$ *from-univ-poly* (*Suc n*) *i Q*
using *from-univ-poly-is-hom ring-hom-add*
by (*metis assms*(*1*) *assms*(*2*) *assms*(*3*))

lemma *from-univ-poly-mult*:
assumes $i \leq n$
assumes $p \in carrier\ (UP\ (R[\mathcal{X}_n]))$
assumes $Q \in carrier\ (UP\ (R[\mathcal{X}_n]))$
shows *from-univ-poly* (*Suc n*) *i* $(p \otimes_{UP\ (R[\mathcal{X}_n])} Q) =$
from-univ-poly (*Suc n*) *i p* $\otimes_{R[\mathcal{X}_{Suc\ n}]}$ *from-univ-poly* (*Suc n*) *i Q*
using *from-univ-poly-is-hom ring-hom-mult*
by (*metis assms*(*1*) *assms*(*2*) *assms*(*3*))

lemma(**in** *UP-cring*) *monom-as-mult*:
assumes $a \in carrier\ R$
shows *up-ring.monom* (*UP R*) *a n* = *to-poly a* $\otimes_{UP\ R}$ *up-ring.monom* (*UP R*) **1** *n*
by (*metis One-nat-def P-def R.one-closed R.r-one UP-cring.poly-shift-monom add-Suc assms is-UP-cring local.monom-mult plus-1-eq-Suc to-polynomial-def*)

lemma *cring-coord-rings-coord-ring*:
cring-coord-rings $(R[\mathcal{X}_n])$
unfolding *cring-coord-rings-def*
cring-coord-rings-axioms-def coord-ring-def
apply(*rule conjI*)
unfolding *UP-cring-def*
apply (*metis coord-cring-cring coord-ring-def*)
using *cring-coord-rings-axioms*
unfolding *cring-coord-rings-def cring-coord-rings-axioms-def*
by (*metis coord-ring-def coord-ring-one coord-ring-zero*)

lemma *from-univ-poly-monom-inverse*:
assumes $i < Suc\ n$
assumes $a \in carrier\ (R[\mathcal{X}_n])$

shows *to-univ-poly* (*Suc n*) *i* (*from-univ-poly* (*Suc n*) *i* (*up-ring.monom* (*UP* (*R* [$\mathcal{X}_n$])) *a m*)) = *up-ring.monom* (*UP* (*R* [$\mathcal{X}_n$])) *a m*
proof−
have *0*: *up-ring.monom* (*UP* (*R* [$\mathcal{X}_n$])) *a m* = (*to-polynomial* (*R*[$\mathcal{X}_n$]) *a*) $\otimes_{UP}$ (*R*[$\mathcal{X}_n$]) (*up-ring.monom* (*UP* (*R* [$\mathcal{X}_n$])) $\mathbf{1}_{R[\mathcal{X}_n]}$ *m*)
using *UP-cring.monom-as-mult*[*of R*[$\mathcal{X}_n$] *a m*] **unfolding** *UP-ring-def*
using *UP-cring-def assms coord-cring-cring* **by** *blast*
have *1* : (*UP-to-IP* (*R* [$\mathcal{X}_{Suc\ n\ -\ 1}$]) *i*) (*to-polynomial* (*R*[$\mathcal{X}_n$]) *a*) = *ring.indexed-const* (*R*[$\mathcal{X}_n$]) *a*
using *UP-cring.UP-to-IP-const*[*of R* [$\mathcal{X}_{Suc\ n\ -\ 1}$] *a i*] **unfolding** *UP-cring-def*

by (*simp add*: *assms coord-cring-cring*)
have *2*: (*from-univ-poly* (*Suc n*) *i* (*to-polynomial* (*R*[$\mathcal{X}_n$]) *a*))
= *pre-to-univ-poly-inv* (*Suc n*) *i* (*ring.indexed-const* (*R*[$\mathcal{X}_n$]) *a*)
unfolding *from-univ-poly-def* **using** *1*
by (*metis comp-apply*)
have *3*: *from-univ-poly* (*Suc n*) *i* (*to-polynomial* (*R* [$\mathcal{X}_n$]) *a*) = *pre-to-univ-poly-inv-hom* (*Suc n*) *i a*
using *pre-to-univ-poly-inv-const*[*of i n a*] *assms 2*
by *presburger*
have *4*: *to-univ-poly* (*Suc n*) *i* (*from-univ-poly* (*Suc n*) *i* (*to-polynomial* (*R* [$\mathcal{X}_n$]) *a*)) =
IP-to-UP i ((*pre-to-univ-poly* (*Suc n*) *i*) (*pre-to-univ-poly-inv-hom* (*Suc n*) *i a*))
using *3* **unfolding** *to-univ-poly-def from-univ-poly-def*
by (*metis comp-apply*)
have *5*: (*pre-to-univ-poly* (*Suc n*) *i*) (*pre-to-univ-poly-inv* (*Suc n*) *i* (*ring.indexed-const* (*R*[$\mathcal{X}_n$]) *a*)) = (*ring.indexed-const* (*R*[$\mathcal{X}_n$]) *a*)
using *assms*(*1*) *assms*(*2*) *pre-to-univ-poly-inverse'* **by** *blast*
have (*to-univ-poly* (*Suc n*) *i*) (*from-univ-poly* (*Suc n*) *i* (*to-polynomial* (*R*[$\mathcal{X}_n$]) *a*)) = *IP-to-UP i* (*ring.indexed-const* (*R*[$\mathcal{X}_n$]) *a*)
unfolding *to-univ-poly-def*
by (*metis 2 5 comp-apply*)
hence *6*: (*to-univ-poly* (*Suc n*) *i*) (*from-univ-poly* (*Suc n*) *i* (*to-polynomial* (*R*[$\mathcal{X}_n$]) *a*)) = *to-polynomial* (*R*[$\mathcal{X}_n$]) *a*
using *UP-cring.IP-to-UP-indexed-const*[*of R*[$\mathcal{X}_n$]]
by (*smt UP-cring-def assms*(*2*) *coord-cring-cring*)
have *7*: (*to-univ-poly* (*Suc n*) *i*) (*from-univ-poly* (*Suc n*) *i* (*up-ring.monom* (*UP* (*R* [$\mathcal{X}_n$])) $\mathbf{1}_{R[\mathcal{X}_n]}$ *m*)) = *up-ring.monom* (*UP* (*R* [$\mathcal{X}_n$])) $\mathbf{1}_{R[\mathcal{X}_n]}$ *m*
proof−
have *70*: *pvar* (*R* [$\mathcal{X}_n$]) *i* [^]$_{Pring}$ (*R* [$\mathcal{X}_n$]) {*i*} *m* ∈ *carrier* (*Pring* (*R* [$\mathcal{X}_n$]) {*i*})
using *Cring-Multivariable-Poly.pvar-closed*[*of R*[$\mathcal{X}_n$] *i* {*i*}] *monoid.nat-pow-closed*[*of R*[$\mathcal{X}_n$]]
by (*meson MP.Pring-is-monoid coord-cring-cring equalityD2 insert-subset monoid.nat-pow-closed*)
have *71*: (*UP-to-IP* (*R* [$\mathcal{X}_{Suc\ n\ -\ 1}$]) *i*) (*up-ring.monom* (*UP* (*R* [$\mathcal{X}_n$])) $\mathbf{1}_{R\ [\mathcal{X}_n]}$ *m*) =

$$(pvar\ (R[\mathcal{X}_n])\ i)[\hat{}]_{Pring\ (R[\mathcal{X}_n])\ \{i\}} m$$

using *70 UP-cring.UP-to-IP-monom*[*of* $R[\mathcal{X}_n]$ $\mathbf{1}_{R[\mathcal{X}_n]}$ *i m*] *cring.Pring-smult-one*[*of* $R[\mathcal{X}_n]$ *pvar* (R [$\mathcal{X}_n$]) *i* [$\hat{}$]$_{Pring\ (R\ [\mathcal{X}_n])\ \{i\}}$ *m* {*i*}]

unfolding *UP-cring-def*

using *MP.one-closed coord-cring-cring diff-Suc-1* **by** *presburger*

hence *72*: *from-univ-poly* (*Suc n*) *i* (*up-ring.monom* (*UP* (R [$\mathcal{X}_n$])) $\mathbf{1}_{R[\mathcal{X}_n]}$ *m*) =

pre-to-univ-poly-inv (*Suc n*) *i* ((*pvar* ($R[\mathcal{X}_n]$) *i*)[$\hat{}$]$_{Pring\ (R[\mathcal{X}_n])\ \{i\}}$ *m*)

unfolding *from-univ-poly-def*

using *comp-apply*[*of pre-to-univ-poly-inv* (*Suc n*) *i UP-to-IP* (R [$\mathcal{X}_{Suc\ n\ -\ 1}$]) *i up-ring.monom* (*UP* (R [$\mathcal{X}_n$])) $\mathbf{1}_{R\ [\mathcal{X}_n]}$ *m*]

by *presburger*

have *73*: *pre-to-univ-poly-inv* (*Suc n*) *i* ∈ *ring-hom* (*Pring* (R [$\mathcal{X}_n$]) {*i*}) (R [$\mathcal{X}_{Suc\ n}$])

using *pre-to-univ-poly-inv-is-hom*[*of i n*] *assms*(*1*) *ring-hom-ring.homh* **by** *blast*

hence *74*: *from-univ-poly* (*Suc n*) *i* (*up-ring.monom* (*UP* (R [$\mathcal{X}_n$])) $\mathbf{1}_{R[\mathcal{X}_n]}$ *m*) = (*pvar R i*)[$\hat{}$]$_{R[\mathcal{X}_{Suc\ n}]}$*m*

unfolding *from-univ-poly-def*

using *70 71 72 pre-to-univ-poly-inv-pvar*[*of i n*]

ring-hom-nat-pow[*of* (*Pring* (R [$\mathcal{X}_n$]) {*i*}) R [$\mathcal{X}_{Suc\ n}$] *pre-to-univ-poly-inv* (*Suc n*) *i* (*pvar* ($R[\mathcal{X}_n]$) *i*) *m*]

by (*metis MP.Pring-is-ring MP.Pring-var-closed MP.ring-axioms assms*(*1*) *from-univ-poly-def singletonI*)

hence *75*: (*to-univ-poly* (*Suc n*) *i*) (*from-univ-poly* (*Suc n*) *i* (*up-ring.monom* (*UP* (R [$\mathcal{X}_n$])) $\mathbf{1}_{R[\mathcal{X}_n]}$ *m*))

= (*to-univ-poly* (*Suc n*) *i*) ((*pvar R i*)[$\hat{}$]$_{R[\mathcal{X}_{Suc\ n}]}$*m*)

by *metis*

have *76*: *pre-to-univ-poly* (*Suc n*) *i* (*pvar R i*) = *pvar* (R [$\mathcal{X}_{Suc\ n\ -\ 1}$]) *i*

using *pre-to-univ-poly-is-hom*(*3*)[*of i Suc n*] *assms*(*1*) **by** *blast*

have *pre-to-univ-poly* (*Suc n*) *i* ∈ *ring-hom* (R [$\mathcal{X}_{Suc\ n}$]) (*Pring* (R [$\mathcal{X}_{Suc\ n\ -\ 1}$]) {*i*})

apply(*rule ring-hom-ring.homh*)

using *pre-to-univ-poly-is-hom*(*1*)[*of i Suc n*]

using *assms*(*1*) **by** *blast*

hence *pre-to-univ-poly* (*Suc n*) *i* (*pvar R i* [$\hat{}$]$_{R\ [\mathcal{X}_{Suc\ n}]}$ *m*) = *pvar* (R [$\mathcal{X}_{Suc\ n\ -\ 1}$]) *i* [$\hat{}$]$_{Pring\ (R\ [\mathcal{X}_{Suc\ n\ -\ 1}])\ \{i\}}$ *m*

using *76 ring-hom-nat-pow*[*of* $R[\mathcal{X}_{Suc\ n}]$ *Pring* (R [$\mathcal{X}_{Suc\ n\ -\ 1}$]) {*i*} *pre-to-univ-poly* (*Suc n*) *i pvar R i m*]

by (*metis MP.Pring-is-ring MP.ring-axioms assms*(*1*) *local.pvar-closed*)

hence *77*: (*to-univ-poly* (*Suc n*) *i*) (*from-univ-poly* (*Suc n*) *i* (*up-ring.monom* (*UP* (R [$\mathcal{X}_n$])) $\mathbf{1}_{R[\mathcal{X}_n]}$ *m*))

=*IP-to-UP i* (*pvar* (R [$\mathcal{X}_{Suc\ n\ -\ 1}$]) *i* [$\hat{}$]$_{Pring\ (R\ [\mathcal{X}_{Suc\ n\ -\ 1}])\ \{i\}}$ *m*)

unfolding *to-univ-poly-def* **using** *comp-apply*[*of IP-to-UP i pre-to-univ-poly* (*Suc n*) *i*]

using *74* **by** *presburger*

have *78*: *IP-to-UP i (pvar (R [$\mathcal{X}_{Suc\ n\ -\ 1}$]) i) = X-poly (R[$\mathcal{X}_n$])*
using *cring.IP-to-UP-var[of R[$\mathcal{X}_n$]]*
by (*simp add*: *MP.IP-to-UP-var var-to-IP-def*)
have *79*: *IP-to-UP i ∈ ring-hom (Pring (R [$\mathcal{X}_n$]) {i}) (UP (R [$\mathcal{X}_n$]))*
using *UP-cring.IP-to-UP-ring-hom[of R[$\mathcal{X}_n$] i] ring-hom-ring.homh[of Pring (R [$\mathcal{X}_n$]) {i}]*
unfolding *UP-cring-def*
using *coord-cring-cring* **by** *blast*
have *80*: *pvar (R [$\mathcal{X}_{Suc\ n\ -\ 1}$]) i ∈ carrier (Pring (R [$\mathcal{X}_n$]) {i})*
by (*metis 76 assms(1) diff-Suc-1 local.pvar-closed pre-to-univ-poly-is-hom(6)*)
have *81*: *ring (UP (R[$\mathcal{X}_n$]))*
using *UP-ring.UP-ring[of R[$\mathcal{X}_n$]]* **unfolding** *UP-ring-def*
using *MP.ring-axioms* **by** *blast*
hence *82*: *IP-to-UP i (pvar (R [$\mathcal{X}_{Suc\ n\ -\ 1}$]) i [^]$_{Pring\ (R[\mathcal{X}_n])\ \{i\}}$ m) = X-poly (R[$\mathcal{X}_n$]) [^]$_{UP\ (R\ [\mathcal{X}_n])}$ m*

using *78 79 80 ring-hom-nat-pow[of Pring (R [$\mathcal{X}_n$]) {i} UP (R [$\mathcal{X}_n$]) IP-to-UP i pvar (R [$\mathcal{X}_{Suc\ n\ -\ 1}$]) i m]*
by (*metis MP.Pring-is-ring*)
have *83*: $\mathbf{1}_{R\ [\mathcal{X}_n]}$ $\odot_{UP\ (R\ [\mathcal{X}_n])}$ *X-poly (R [$\mathcal{X}_n$]) [^]$_{UP\ (R\ [\mathcal{X}_n])}$ m = X-poly (R [$\mathcal{X}_n$]) [^]$_{UP\ (R\ [\mathcal{X}_n])}$ m*
using *UP-ring.UP-smult-one[of R[$\mathcal{X}_n$] X-poly (R [$\mathcal{X}_n$]) [^]$_{UP\ (R\ [\mathcal{X}_n])}$ m] UP-cring.X-closed[of R[$\mathcal{X}_n$]] monoid.nat-pow-closed[of UP (R[$\mathcal{X}_n$]) X-poly (R[$\mathcal{X}_n$]) m]*
unfolding *UP-ring-def UP-cring-def*
using *81 MP.ring-axioms coord-cring-cring ring.is-monoid* **by** *blast*
have *84*: *X-poly (R[$\mathcal{X}_n$]) [^]$_{UP\ (R\ [\mathcal{X}_n])}$ m = up-ring.monom (UP (R [$\mathcal{X}_n$])) $\mathbf{1}_{R\ [\mathcal{X}_n]}$ m*
using *83 UP-cring.monom-rep-X-pow[of R[$\mathcal{X}_n$] $\mathbf{1}_{R[\mathcal{X}_n]}$ m] monoid.nat-pow-closed[of UP (R[$\mathcal{X}_n$]) X-poly (R[$\mathcal{X}_n$]) m] 81*
unfolding *UP-cring-def*
using *MP.one-closed coord-cring-cring* **by** *presburger*
thus *?thesis* **using** *77*
by (*metis 82 diff-Suc-1*)
qed
have *8*: *to-univ-poly (Suc n) i (from-univ-poly (Suc n) i (up-ring.monom (UP (R [$\mathcal{X}_n$])) a m)) = to-univ-poly (Suc n) i (from-univ-poly (Suc n) i (to-polynomial (R[$\mathcal{X}_n$]) a)) $\otimes_{UP\ (R[\mathcal{X}_n])}$ to-univ-poly (Suc n) i (from-univ-poly (Suc n) i (up-ring.monom (UP (R [$\mathcal{X}_n$])) $\mathbf{1}_{R[\mathcal{X}_n]}$ m))*
proof−
have *80*: *to-polynomial (R [$\mathcal{X}_n$]) a ∈ carrier (UP (R [$\mathcal{X}_n$]))*
using *UP-cring.to-poly-closed[of R[$\mathcal{X}_n$] a] UP-cring-def assms(2) coord-cring-cring*
by *blast*
have *81*: *up-ring.monom (UP (R [$\mathcal{X}_n$])) $\mathbf{1}_{R\ [\mathcal{X}_n]}$ m ∈ carrier (UP (R [$\mathcal{X}_n$]))*
apply(*rule UP-ring.monom-closed[of R[$\mathcal{X}_n$]]*) **unfolding** *UP-ring-def* **using**

MP.one-closed
apply (*simp add*: *MP.ring-axioms*)
using *MP.one-closed* **by** *blast*
have *82*: (*from-univ-poly* (*Suc n*) *i* (*up-ring.monom* (*UP* (*R* [$\mathcal{X}_n$])) *a m*)) =
(*from-univ-poly* (*Suc n*) *i* (*to-polynomial* (*R*[$\mathcal{X}_n$]) *a*)) $\otimes_{(R[\mathcal{X}_{Suc\ n}])}$
(*from-univ-poly* (*Suc n*) *i* (*up-ring.monom* (*UP* (*R* [$\mathcal{X}_n$])) $\mathbf{1}_{R[\mathcal{X}_n]}$
m))
using *80 81 from-univ-poly-mult*[*of i n to-polynomial* (*R* [$\mathcal{X}_n$]) *a* (*up-ring.monom*
(*UP* (*R* [$\mathcal{X}_n$])) $\mathbf{1}_{R[\mathcal{X}_n]}$ *m*)] *0*
by (*metis assms*(*1*) *less-Suc-eq-le*)
thus *?thesis* **using** *to-univ-poly-mult 80 81*
by (*metis assms*(*1*) *from-univ-poly-closed less-Suc-eq-le*)
qed
thus *?thesis*
using *0 6 7* **by** *metis*
qed

lemma *from-univ-poly-inverse*:
assumes $i \leq n$
assumes $p \in$ *carrier* (*UP* (*R*[$\mathcal{X}_n$]))
shows *to-univ-poly* (*Suc n*) *i* (*from-univ-poly* (*Suc n*) *i p*) = *p*
proof(*rule UP-ring.poly-induct3*[*of R*[$\mathcal{X}_n$]])
show *UP-ring* (*R* [$\mathcal{X}_n$])
unfolding *UP-ring-def*
by (*simp add*: *MP.ring-axioms*)
show $p \in$ *carrier* (*UP* (*R* [$\mathcal{X}_n$]))
using *assms* **by** *blast*
show $\bigwedge p\ q.\ q \in$ *carrier* (*UP* (*R* [$\mathcal{X}_n$])) $\Longrightarrow$
$p \in$ *carrier* (*UP* (*R* [$\mathcal{X}_n$])) $\Longrightarrow$
to-univ-poly (*Suc n*) *i* (*from-univ-poly* (*Suc n*) *i p*) = *p* $\Longrightarrow$
to-univ-poly (*Suc n*) *i* (*from-univ-poly* (*Suc n*) *i q*) = *q* $\Longrightarrow$
to-univ-poly (*Suc n*) *i* (*from-univ-poly* (*Suc n*) *i* (*p* $\oplus_{UP\ (R\ [\mathcal{X}_n])}$ *q*)) =
p $\oplus_{UP\ (R\ [\mathcal{X}_n])}$ *q*
proof− **fix** *p q*
assume *A*: *q* $\in$ *carrier* (*UP* (*R* [$\mathcal{X}_n$])) *p* $\in$ *carrier* (*UP* (*R* [$\mathcal{X}_n$]))
to-univ-poly (*Suc n*) *i* (*from-univ-poly* (*Suc n*) *i p*) = *p*
to-univ-poly (*Suc n*) *i* (*from-univ-poly* (*Suc n*) *i q*) = *q*
show *to-univ-poly* (*Suc n*) *i* (*from-univ-poly* (*Suc n*) *i* (*p* $\oplus_{UP\ (R\ [\mathcal{X}_n])}$ *q*)) =
p $\oplus_{UP\ (R\ [\mathcal{X}_n])}$ *q*
using *A assms*
from-univ-poly-add[*of i n p q*]
to-univ-poly-add[*of i n from-univ-poly* (*Suc n*) *i p from-univ-poly* (*Suc n*) *i*
q]
from-univ-poly-closed[*of i n p*] *from-univ-poly-closed*[*of i n q*]
by *presburger*
qed
show $\bigwedge a\ na.\ a \in$ *carrier* (*R* [$\mathcal{X}_n$]) $\Longrightarrow$
to-univ-poly (*Suc n*) *i* (*from-univ-poly* (*Suc n*) *i* (*up-ring.monom* (*UP* (*R*

$[\mathcal{X}_n])) \ a \ na)) = up\text{-}ring.monom \ (UP \ (R \ [\mathcal{X}_n])) \ a \ na$
using *from-univ-poly-monom-inverse*[*of i*] *assms*(*1*) *le-imp-less-Suc* **by** *presburger*
qed

lemma *to-univ-poly-eval*:
assumes $i < Suc \ n$
assumes $p \in carrier \ (R[\mathcal{X}_{Suc \ n}])$
assumes $a \in carrier \ (R^n)$
assumes $x \in carrier \ R$
assumes $as = insert\text{-}at\text{-}index \ a \ x \ i$
shows *eval-at-point R as p* = *eval-at-point R a* (*to-function* ($R[\mathcal{X}_n]$) (*to-univ-poly* (*Suc n*) *i p*) (*coord-const x*))
proof−
have *0*: *pre-to-univ-poly* (*Suc n*) *i p* ∈ *Pring-set* ($R[\mathcal{X}_n]$) {*i*}
using *assms pre-to-univ-poly-is-hom*(*1*)[*of i Suc n*] **unfolding** *ring-hom-ring-def*

unfolding *coord-ring-def UP-domain-def coord-ring-def UP-domain-def*
by (*metis MP.Pring-car coord-ring-def diff-Suc-1 pre-to-univ-poly-is-hom*(*6*))
have *1*: *closed-fun* ($R[\mathcal{X}_n]$) (λ*n*. *coord-const x*)
using *assms*(*4*) *R.indexed-const-closed*
unfolding *coord-ring-def UP-domain-def*
by *blast*
have (*to-function* ($R[\mathcal{X}_n]$) (*to-univ-poly* (*Suc n*) *i p*) (*coord-const x*)) =
to-function ($R[\mathcal{X}_n]$) (*IP-to-UP i* ((*pre-to-univ-poly* (*Suc n*) *i*) *p*)) (*coord-const x*)
unfolding *to-univ-poly-def*
unfolding *coord-ring-def UP-domain-def*
by (*metis comp-apply*)
then have *2*: (*to-function* ($R[\mathcal{X}_n]$) (*to-univ-poly* (*Suc n*) *i p*) (*coord-const x*)) =

(*total-eval* ($R[\mathcal{X}_n]$) (λ *i*. *coord-const x*) (*pre-to-univ-poly* (*Suc n*) *i p*))
using *0 1 UP-cring.IP-to-UP-poly-eval*[*of* $R[\mathcal{X}_n]$
(*pre-to-univ-poly* (*Suc n*) *i*) *p i* λ *i*. *coord-const x*]
unfolding *coord-ring-def UP-cring-def*
using *assms*(*4*) *cring.indexed-const-closed R.Pring-is-cring R.cring-axioms*
by *smt*
then show *?thesis* **using** *pre-to-univ-poly-eval*[*of i n p a x as*]
using *assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *assms*(*5*) **by** *presburger*
qed

The function `one_over_poly`, introduced in the theory `Cring_Poly`, maps a polynomial $p(x)$ to the unique polynomial $q(x)$ which satisfies the relation $q(x) = x^n p(1/x)$. This will be used later to show that the function $f(x) = 1/x$ is semialgebraic over the field $\mathbb{Q}_p$.

lemma *to-univ-poly-one-over-poly*:
assumes *field R*
assumes $i < (Suc \ n)$
assumes $p \in carrier \ (R[\mathcal{X}_{Suc \ n}])$

assumes $Q = from\text{-}univ\text{-}poly\ (Suc\ n)\ i\ (UP\text{-}cring.one\text{-}over\text{-}poly\ (R[\mathcal{X}_n])\ (to\text{-}univ\text{-}poly\ (Suc\ n)\ i\ p))$
assumes $a \in carrier\ (R^n)$
assumes $x \in carrier\ R$
assumes $x \neq \mathbf{0}$
assumes $b = insert\text{-}at\text{-}index\ a\ x\ i$
assumes $c = insert\text{-}at\text{-}index\ a\ (inv\ x)\ i$
assumes $N = UP\text{-}ring.degree\ (R[\mathcal{X}_n])\ (to\text{-}univ\text{-}poly\ (Suc\ n)\ i\ p)$
shows $Q \in carrier\ (R[\mathcal{X}_{Suc\ n}])$
$eval\text{-}at\text{-}point\ R\ b\ Q = (x[\widehat{}]N) \otimes (eval\text{-}at\text{-}point\ R\ c\ p)$
proof−
have *0*: $(to\text{-}univ\text{-}poly\ (Suc\ n)\ i\ p) \in carrier\ (UP\ (R[\mathcal{X}_n]))$
using *assms(2) assms(3) less-Suc-eq-le to-univ-poly-closed* **by** *blast*
have *1*: $(UP\text{-}cring.one\text{-}over\text{-}poly\ (R[\mathcal{X}_n])\ (to\text{-}univ\text{-}poly\ (Suc\ n)\ i\ p)) \in carrier\ (UP\ (R[\mathcal{X}_n]))$
using *0 assms UP-domain-def UP-cring.one-over-poly-closed UP-cring-def coord-cring-cring* **by** *blast*
show $Q \in carrier\ (R[\mathcal{X}_{Suc\ n}])$
using *1 assms from-univ-poly-closed[of i n] less-Suc-eq-le*
by *blast*
have *2*: $coord\text{-}const\ x \in Units\ (R[\mathcal{X}_n])$
proof−

have *20*: $inv\ x \in carrier\ R$
using *assms(1) assms(6) assms(7) field.field-Units* **by** *blast*
have *21*: $x \otimes (inv\ x) = \mathbf{1}$
using *assms field.field-Units R.Units-r-inv*
by *blast*
have *22*: $coord\text{-}const\ x \in carrier\ (R[\mathcal{X}_n])$
using *assms(6) R.indexed-const-closed*
unfolding *coord-ring-def*
by *blast*
have *23*: $coord\text{-}const\ (inv\ x) \in carrier\ (R[\mathcal{X}_n])$
using *20 R.indexed-const-closed*
unfolding *coord-ring-def*
by *blast*
have *24*: $coord\text{-}const\ x \otimes_{R[\mathcal{X}_n]} coord\text{-}const\ (inv\ x) = coord\text{-}const\ (x \otimes (inv\ x))$
using *assms(6) 20 R.indexed-const-mult* **unfolding** *coord-ring-def*
by *blast*
have *25*: $coord\text{-}const\ x \otimes_{R[\mathcal{X}_n]} coord\text{-}const\ (inv\ x) = \mathbf{1}_{coord\text{-}ring\ R\ n}$
unfolding *coord-ring-def*
by (*metis 20 21 R.Pring-one assms(6) R.indexed-const-mult*)
have *26*: $coord\text{-}const\ (inv\ x) \otimes_{R[\mathcal{X}_n]} coord\text{-}const\ x = \mathbf{1}_{coord\text{-}ring\ R\ n}$
unfolding *coord-ring-def*
by (*metis 21 22 23 24 MP.m-comm R.Pring-one coord-ring-def*)
then show *?thesis*
using *23 Units-def[of R[\mathcal{X}_n]] 22 25*
by *blast*

qed
have *3*: $inv_{R[\mathcal{X}_n]}$ (*coord-const x*) = *coord-const* (*inv x*)
proof−
have *20*: *inv x* ∈ *carrier R*
using *assms*(*1*) *assms*(*6*) *assms*(*7*) *field.field-Units* **by** *blast*
have *21*: *x* ⊗ (*inv x*) = **1**
using *assms field.field-Units R.Units-r-inv*
by *blast*
have *22*: *coord-const x* ∈ *carrier* (*R*[$\mathcal{X}_n$])
using *assms*(*6*) *R.indexed-const-closed*
unfolding *coord-ring-def*
by *blast*
have *23*: *coord-const* (*inv x*) ∈ *carrier* (*R*[$\mathcal{X}_n$])
using *20 R.indexed-const-closed* **unfolding** *coord-ring-def*
by *blast*
have *24*: *coord-const x* $\otimes_{R[\mathcal{X}_n]}$ *coord-const* (*inv x*) = *coord-const* (*x* ⊗ (*inv x*))
using *assms*(*6*) *20 R.indexed-const-mult* **unfolding** *coord-ring-def*
by *blast*
have *25*: *coord-const x* $\otimes_{R[\mathcal{X}_n]}$ *coord-const* (*inv x*) = $\mathbf{1}_{coord\text{-}ring\ \ R\ n}$
unfolding *coord-ring-def*
by (*metis 20 21 R.Pring-one assms*(*6*) *R.indexed-const-mult*)
show *?thesis*
using *22 23 25 R.Pring-is-cring*[*of* {..<*n*}]
monoid.inv-char[*of R*[$\mathcal{X}_n$]]
unfolding *coord-ring-def*

by (*metis R.Pring-is-monoid R.Pring-mult-comm R.is-cring*)
qed
have *4*: *to-function* (*R*[$\mathcal{X}_n$]) (*UP-cring.one-over-poly* (*R*[$\mathcal{X}_n$]) (*to-univ-poly* (*Suc n*) *i p*))
(*coord-const x*) = (*coord-const x*)[^]$_{R[\mathcal{X}_n]}N$ $\otimes_{R[\mathcal{X}_n]}$
(*to-function* (*R*[$\mathcal{X}_n$]) ((*to-univ-poly* (*Suc n*) *i p*)) (*coord-const* (inv_R *x*)))
using *3 assms UP-cring-def UP-cring.one-over-poly-eval*[*of R*[$\mathcal{X}_n$] (*to-univ-poly* (*Suc n*) *i p*) *coord-const x*]
unfolding *coord-ring-def*
by (*metis 0 2 MP.Units-closed R.Pring-is-cring UP-cring.to-fun-def coord-ring-def R.is-cring*)
have *5*: *eval-at-point R a* (*to-function* (*R*[$\mathcal{X}_n$]) (*UP-cring.one-over-poly* (*R*[$\mathcal{X}_n$]) (*to-univ-poly* (*Suc n*) *i p*))
(*coord-const x*))
= *eval-at-point R a* ((*coord-const x*)[^]$_{R[\mathcal{X}_n]}N$ $\otimes_{R[\mathcal{X}_n]}$
(*to-function* (*R*[$\mathcal{X}_n$]) ((*to-univ-poly* (*Suc n*) *i p*)) (*coord-const* (inv_R *x*))))
using *4*
by *presburger*
have *6*: *to-univ-poly* (*Suc n*) *i Q* = (*UP-cring.one-over-poly* (*R*[$\mathcal{X}_n$]) (*to-univ-poly*

(*Suc n*) *i p*))
using *assms from-univ-poly-inverse*
by (*meson 1 less-Suc-eq-le*)
have *7*: *eval-at-point R a* (*to-function* (*R*[$\mathcal{X}_n$]) (*UP-cring.one-over-poly* (*R*[$\mathcal{X}_n$])
(*to-univ-poly* (*Suc n*) *i p*))
(*coord-const x*)) = *eval-at-point R b Q*
using *6 to-univ-poly-eval*[*of i n Q a x b*] *assms* ‹*Q* ∈ *carrier* (*R*[$\mathcal{X}_{Suc\ n}$])›
by *smt*
have *8*: (*coord-const x*)[^]$_{R[\mathcal{X}_n]}$*N* ∈ *carrier* (*R*[$\mathcal{X}_n$])
using *monoid.nat-pow-closed*[*of R*[$\mathcal{X}_n$]]
unfolding *coord-ring-def*
using *R.Pring-is-monoid assms*(*6*) *R.indexed-const-closed* **by** *blast*
have *9*: *to-function* (*R*[$\mathcal{X}_n$]) ((*to-univ-poly* (*Suc n*) *i p*)) (*coord-const* (*inv*$_R$ *x*))
∈ *carrier* (*R*[$\mathcal{X}_n$])
proof−
have *91*: *to-univ-poly* (*Suc n*) *i p* ∈ *carrier* (*UP* (*R*[$\mathcal{X}_n$]))
by (*simp add*: *0*)
have *coord-const* (*inv x*) ∈ *carrier* (*R*[$\mathcal{X}_n$])
proof−
have *inv x* ∈ *carrier R*
using *assms*(*1*) *assms*(*6*) *assms*(*7*) *field.field-Units* **by** *blast*
then show *?thesis*
using *R.indexed-const-closed*[*of inv x*] *assms*
unfolding *coord-ring-def*

by *blast*
qed
then show *?thesis*
using *91 UP-cring-def*[*of R*[$\mathcal{X}_n$]] *UP-cring.to-fun-closed*[*of R*[$\mathcal{X}_n$] *to-univ-poly*
(*Suc n*) *i p coord-const* (*inv*$_R$ *x*)]
to-univ-poly-closed[*of i n p*] *UP-domain-def*[*of R*[$\mathcal{X}_n$]]
unfolding *coord-ring-def*
using *R.Pring-is-cring R.is-cring*
by (*metis UP-cring.to-fun-def*)
qed
have *10*: *eval-at-point R b Q* = (*eval-at-point R a* ((*coord-const x*)[^]$_{R[\mathcal{X}_n]}$*N*))
⊗
(*eval-at-point R a* (*to-function* (*R*[$\mathcal{X}_n$]) ((*to-univ-poly* (*Suc n*) *i p*))
(*coord-const* (*inv*$_R$ *x*))))
using *7 5 eval-at-point-mult*[*of a n* (*coord-const x*)[^]$_{R[\mathcal{X}_n]}$*N*
(*to-function* (*R*[$\mathcal{X}_n$]) ((*to-univ-poly* (*Suc n*) *i p*)) (*coord-const* (*inv*$_R$ *x*)))]
8 9 assms(*5*)
by *presburger*
have *11*: *inv x* ∈ *carrier R*
using *assms*(*1*) *assms*(*6*) *assms*(*7*) *field.field-Units* **by** *blast*
have *12*: *eval-at-point R b Q* = (*eval-at-point R a* ((*coord-const x*)[^]$_{R[\mathcal{X}_n]}$*N*))
⊗
(*eval-at-point R c p*)
using *10 11 to-univ-poly-eval*[*of i n p a inv x c*] *assms*(*2*) *assms*(*3*) *assms*(*5*)

assms(*9*)
by *presburger*
show *12*: *eval-at-point R b Q* = (*x*[^]*N*) ⊗
(*eval-at-point R c p*)
proof−
have *0*: (*coord-const x*)[^]$_{R[\mathcal{X}_n]}$*N* = *coord-const* (*x*[^]*N*)
proof(*induction N*)
case *0*
have *00*: *coord-const x* [^]$_{R[\mathcal{X}_n]}$ (*0*::*nat*) = $\mathbf{1}_{R[\mathcal{X}_n]}$
using *nat-pow-def*[*of R*[$\mathcal{X}_n$] - (*0*::*nat*)]
unfolding *coord-ring-def*
by (*meson Group.nat-pow-0*)
then show *?case*
unfolding *coord-ring-def*
by (*metis Group.nat-pow-0 R.Pring-one*)
next
case (*Suc N*) **fix** *N*::*nat* **assume** *IH*: *coord-const x* [^]$_{R[\mathcal{X}_n]}$ *N* = *coord-const*
(*x* [^] *N*)
then show *?case*
using *R.indexed-const-mult Group.nat-pow-Suc Suc.IH assms*(*6*) *R.nat-pow-closed*
unfolding *coord-ring-def*
by (*metis*)
qed
have *1*: (*eval-at-point R a* ((*coord-const x*)[^]$_{R[\mathcal{X}_n]}$*N*)) = *x*[^]*N*
using *0*
by (*metis assms*(*5*) *assms*(*6*) *eval-at-point-const R.nat-pow-closed*)
show *?thesis* **using** *0 1 12*
by *presburger*
qed
qed

lemma *to-univ-poly-one-over-poly′*:
assumes *field R*
assumes *i* < (*Suc n*)
assumes *p* ∈ *carrier* (*R*[$\mathcal{X}_{Suc\ n}$])
assumes *Q* = *from-univ-poly* (*Suc n*) *i* (*UP-cring.one-over-poly* (*R*[$\mathcal{X}_n$]) (*to-univ-poly*
(*Suc n*) *i p*))
assumes *a* ∈ *carrier* (R^n)
assumes *x* ∈ *carrier R*
assumes *x* ≠ **0**
assumes *b* = *insert-at-index a x i*
assumes *c* = *insert-at-index a* (*inv x*) *i*
assumes *N* = *UP-ring.degree* (*R*[$\mathcal{X}_n$]) (*to-univ-poly* (*Suc n*) *i p*)
assumes *q* = (*pvar R i*)[^]$_{R[\mathcal{X}_{Suc\ n}]}$(*k*::*nat*)⊗$_{R[\mathcal{X}_{Suc\ n}]}$ *Q*
shows *q* ∈ *carrier* (*R*[$\mathcal{X}_{Suc\ n}$])
eval-at-point R b q = (*x*[^](*N* + *k*)) ⊗ (*eval-at-point R c p*)
proof−
have *0*: (*pvar R i*)[^]$_{R[\mathcal{X}_{Suc\ n}]}$*k* ∈ *carrier* (*R*[$\mathcal{X}_{Suc\ n}$])

using *pvar-closed*[*of i Suc n*] *monoid.nat-pow-closed*[]
unfolding *coord-ring-def*
by (*metis R.Pring-is-monoid assms*(2))
have *1*: $b \in carrier\ (R^{Suc\ n})$
using *assms*(2) *assms*(5) *assms*(6) *assms*(8) *insert-at-index-closed less-Suc-eq-le*
by *blast*
have *11* : $c \in carrier\ (R^{Suc\ n})$
proof−
have *inv x* ∈ *carrier R*
using *assms field.field-Units*
by *blast*
then show *?thesis*
using *assms insert-at-index-closed less-Suc-eq-le*
by *blast*
qed
have *2*: *eval-at-point R b q* = *eval-at-point R b* ((*pvar R i*)[^]$_{R[\mathcal{X}_{Suc\ n}]}$(*k*::*nat*))
⊗ *eval-at-point R b Q*
using *assms 0 1* **unfolding** *coord-ring-def*
by (*metis R.Pring-mult coord-ring-def eval-at-point-mult to-univ-poly-one-over-poly*(1))
have *3*: *eval-at-point R b* ((*pvar R i*)[^]$_{R[\mathcal{X}_{Suc\ n}]}$(*k*::*nat*)) =
x[^](*k*::*nat*)
proof(*induction k*)
case *0*
have *T0*: *eval-at-point R b* ((*pvar R i*)[^]$_{R[\mathcal{X}_{Suc\ n}]}$(*0*::*nat*)) =
eval-at-point R b ($\mathbf{1}_{R[\mathcal{X}_{Suc\ n}]}$)
using *nat-pow-def*[*of R*[$\mathcal{X}_{Suc\ n}$] *pvar R i 0*::*nat*]
by (*metis Group.nat-pow-0*)
then show *?case*
by (*metis 1 assms*(2) *eval-at-point-nat-pow R.nat-pow-0 local.pvar-closed*)
next
case (*Suc k*) **fix** *k*::*nat*
assume *IH*: *eval-at-poly R* (*pvar R i* [^]$_{R[\mathcal{X}_{Suc\ n}]}$ *k*) *b* = *x* [^] *k*
have *0*: *eval-at-poly R* (*pvar R i*) *b* = *b*!*i*
using *eval-pvar*[*of i Suc n*] *assms 1*
by *blast*
have *length a* = *n*
using *assms*(5) *cartesian-power-car-memE* **by** *blast*
then have *eval-at-poly R* (*pvar R i*) *b* = *x*
using *0 assms*(8) *insert-at-index-eq*[*of i a x*]
by (*metis assms*(2) *less-Suc-eq-le*)
then show *?case*
using *1 assms*(2) *eval-at-point-nat-pow local.pvar-closed*
by *blast*
qed
have *4*: *eval-at-point R b Q* = (*x*[^]*N*) ⊗ (*eval-at-point R c p*)
using *to-univ-poly-one-over-poly*(2)[*of i n p Q a x b c N*] *assms*(1) *assms*(10)
assms(2)
assms(3) *assms*(4) *assms*(5) *assms*(6) *assms*(7) *assms*(8) *assms*(9)

by *blast*
have *5*: *eval-at-point R b q* = $x[\hat{}](k::nat) \otimes ((x[\hat{}]N) \otimes (eval\text{-}at\text{-}point\ R\ c\ p))$
using *4 3 2*
by *presburger*
show *6*: *eval-at-point R b q* = $x[\hat{}](N + k) \otimes (eval\text{-}at\text{-}point\ R\ c\ p)$
proof−

have *60*: $x[\hat{}](k::nat) \in carrier\ R$
using *assms*(*6*) **by** *blast*
have *61*: $x[\hat{}]N \in carrier\ R$
using *assms*(*6*) **by** *blast*
have *62*: *eval-at-point R c p* ∈ *carrier R*
using *eval-at-point-closed*[*of c Suc n p*] ‹*c* ∈ *carrier* ($R^{Suc\ n}$)› *assms*(*3*)
by *blast*
show *?thesis* **using** *5 60 61 62*
by (*metis assms*(*6*) *R.m-assoc R.m-comm R.nat-pow-mult*)
qed
show *q* ∈ *carrier* ($R[\mathcal{X}_{Suc\ n}]$)
using *assms*
unfolding *coord-ring-def*
using *0 R.Pring-mult-closed to-univ-poly-one-over-poly*(*1*)
by (*metis coord-ring-def*)

qed

lemma *to-univ-poly-one-over-poly''*:
assumes *field R*
assumes *i* < (*Suc n*)
assumes *p* ∈ *carrier* ($R[\mathcal{X}_{Suc\ n}]$)
assumes *N* ≥ *UP-ring.degree* ($R[\mathcal{X}_n]$) (*to-univ-poly* (*Suc n*) *i p*)
shows ∃ *q* ∈ *carrier* ($R[\mathcal{X}_{Suc\ n}]$). (∀ *x* ∈ *carrier R* − {**0**}. (∀ *a* ∈ *carrier* (R^n).
eval-at-point R (*insert-at-index a x i*) *q* = (*x*[^]*N*) ⊗ (*eval-at-point R* (*insert-at-index a* (*inv x*) *i*) *p*)))
proof−
obtain *Q* **where** *Q-def*:
Q = *from-univ-poly* (*Suc n*) *i* (*UP-cring.one-over-poly* ($R[\mathcal{X}_n]$) (*to-univ-poly* (*Suc n*) *i p*))
by *blast*
obtain *k* **where** *k-def*: *k* = (*N* − *UP-ring.degree* ($R[\mathcal{X}_n]$) (*to-univ-poly* (*Suc n*) *i p*))
by *blast*
obtain *q* **where** *q-def*: *q* = (*pvar R i*)$[\hat{}]_{R[\mathcal{X}_{Suc\ n}]}(k::nat) \otimes_{R[\mathcal{X}_{Suc\ n}]} Q$
by *blast*
have *0*: (∀ *x* ∈ *carrier R* − {**0**}.(∀ *a* ∈ *carrier* (R^n).
eval-at-point R (*insert-at-index a x i*) *q* = (*x*[^]*N*) ⊗ (*eval-at-point R* (*insert-at-index a* (*inv x*) *i*) *p*)))
proof fix *x*
assume *A0*: *x* ∈ *carrier R* − {**0**}

show $\forall a{\in}carrier\ (R^n).\ eval\text{-}at\text{-}poly\ R\ q\ (insert\text{-}at\text{-}index\ a\ x\ i) = x\ [\widehat{}]\ N \otimes eval\text{-}at\text{-}poly\ R\ p\ (insert\text{-}at\text{-}index\ a\ (inv\ x)\ i)$
proof fix *a* **assume** *A1*: $a \in carrier\ (R^n)$
obtain *l* **where** *l-def*: $l = UP\text{-}ring.degree\ (R[\mathcal{X}_n])\ (to\text{-}univ\text{-}poly\ (Suc\ n)\ i\ p)$
by *blast*
have $eval\text{-}at\text{-}poly\ R\ q\ (insert\text{-}at\text{-}index\ a\ x\ i) = x\ [\widehat{}]\ (l + k) \otimes eval\text{-}at\text{-}poly\ R\ p\ (insert\text{-}at\text{-}index\ a\ (inv\ x)\ i)$
using *assms A1 A0 to-univ-poly-one-over-poly'(2)*[*of i n p Q a x insert-at-index a x i insert-at-index a (inv x) i l q k*]
Q-def l-def q-def
by *blast*
then show $eval\text{-}at\text{-}poly\ R\ q\ (insert\text{-}at\text{-}index\ a\ x\ i) = x\ [\widehat{}]\ N \otimes eval\text{-}at\text{-}poly\ R\ p\ (insert\text{-}at\text{-}index\ a\ (inv\ x)\ i)$
using *k-def assms l-def add-diff-inverse-nat less-Suc-eq not-less-eq*
by (*metis diff-diff-cancel diff-less-Suc*)
qed
qed
have *1*: $q \in carrier\ (R[\mathcal{X}_{Suc\ n}])$
proof−
obtain *a* **where** *a-def*: $a = map\ (\lambda i.\ \mathbf{1})\ [(0{::}nat)..{<}n]$
by *blast*
have *a-car*: $a \in carrier\ (R^n)$
apply(*rule cartesian-power-car-memI'*)
using *a-def*
apply (*metis Ex-list-of-length coeff-list length-map length-rev*)
proof− **fix** *i* **assume** *A*: $i < n$
then have $a!i = \mathbf{1}$
using *a-def*
by (*metis R-list-length length-map map-nth nth-map*)
then show $a\ !\ i \in carrier\ R$
using *a-def R.one-closed*
by *metis*
qed
then show $q \in carrier\ (R[\mathcal{X}_{Suc\ n}])$
using *assms q-def k-def Q-def to-univ-poly-one-over-poly'(1)*[*of i n p Q a* $\mathbf{1}$ *-- deg* $(R[\mathcal{X}_n])$
$(to\text{-}univ\text{-}poly\ (Suc\ n)\ i\ p)\ q\ N\ -deg\ (R[\mathcal{X}_n])\ (to\text{-}univ\text{-}poly\ (Suc\ n)\ i\ p)$]
using *one-closed local.one-neq-zero* **by** *blast*
qed
show *?thesis*
using *0 1* **by** *blast*
qed

7 Restricted Inverse Images and Complements

This section introduces some versions of basic set operations for extensional functions and sets. We would like a version of the inverse image which intersects the inverse image of a function with the set `carrier` (R^n), and a

version of the complement of a set which takes the comeplement relative to `carrier` (R^n). These will have to be defined in parametrized families, with one such object for each natural number n.

definition *evimage* (**infixr** ‹$^{-1}_1$› *90*) **where**
evimage n f S = ((*f* −‘ *S*) ∩ *carrier* (R^n))

definition *euminus-set* :: *nat* ⇒ *'a list set* ⇒ *'a list set* (‹- $^c_1$› *70*) **where**
$S^c n$ = *carrier* (R^n) − *S*

lemma *extensional-vimage-closed*:
f ^{-1}n *S* ⊆ *carrier* (R^n)
unfolding *evimage-def* **by** *blast*

7.1 Inverse image of a function

lemma *evimage-eq* [*simp*]: *a* ∈ *f* ^{-1}n *B* ⟷ *a* ∈ *carrier* (R^n) ∧ *f a* ∈ *B*
unfolding *evimage-def*
by *blast*

lemma *evimage-singleton-eq*: *a* ∈ *f* ^{-1}n {*b*} ⟷ *a* ∈ *carrier* (R^n) ∧ *f a* = *b*
unfolding *evimage-def*
by *blast*

lemma *evimageI* [*intro*]: *a* ∈ *carrier* (R^n) ⟹ *f a* = *b* ⟹ *b* ∈ *B* ⟹ *a* ∈ *f* ^{-1}n *B*
unfolding *vimage-def*
using *evimage-eq* **by** *blast*

lemma *evimageI2*: *a* ∈ *carrier* (R^n) ⟹ *f a* ∈ *A* ⟹ *a* ∈ *f* ^{-1}n *A*
unfolding *vimage-def* **by** *fast*

lemma *evimageE* [*elim!*]: *a* ∈ *f* ^{-1}n *B* ⟹ (⋀*x*. *f a* = *x* ⟹ *x* ∈ *B* ⟹ *p*) ⟹ *p*
unfolding *evimage-def*
by *blast*

lemma *evimageD*: *a* ∈ $f^{-1}n$ *A* ⟹ *f a* ∈ *A*
unfolding *vimage-def* **by** *fast*

lemma *evimage-empty* [*simp*]: *f* ^{-1}n {} = {}
by *blast*

lemma *evimage-Compl*:
assumes *f* ∈ *carrier* (R^n) → *carrier* (R^m)
shows (*f* $^{-1}n(A^c m)$) = ((*f* −‘ *A*)$^c n$)
proof−
have (*f* $^{-1}n(A^c m)$) = ((*f* −‘ (*carrier* (R^m)) − (*f* −‘ *A*))) ∩ *carrier* (R^n)
unfolding *evimage-def euminus-set-def* **by** *blast*
hence *0*: (*f* $^{-1}n(A^c m)$) = (*f* −‘ (*carrier* (R^m)) ∩ *carrier* (R^n)) − (*f* −‘ *A*)
by (*simp add*: *Int-Diff Int-commute*)

have $(f -` (carrier\ (R^m)) \cap carrier\ (R^n)) = carrier\ (R^n)$
proof
show $f -` carrier\ (R^m) \cap carrier\ (R^n) \subseteq carrier\ (R^n)$
by *auto*
show $carrier\ (R^n) \subseteq f -` carrier\ (R^m) \cap carrier\ (R^n)$
using *assms* **by** *blast*
qed
thus *?thesis* **using** *0*
by (*simp add*: *euminus-set-def*)
qed

lemma *evimage-Un* [*simp*]: $f\ ^{-1}n\ (A \cup B) = (f\ ^{-1}n\ A) \cup (f\ ^{-1}n\ B)$
unfolding *evimage-def* **by** *blast*

lemma *evimage-Int* [*simp*]: $f\ ^{-1}n\ (A \cap B) = (f\ ^{-1}n\ A) \cap (f\ ^{-1}n\ B)$
unfolding *evimage-def* **by** *blast*

lemma *evimage-Collect-eq* [*simp*]: $f\ ^{-1}n\ Collect\ p = \{y \in carrier\ (R^n).\ p\ (f\ y)\}$
unfolding *evimage-def* **by** *blast*

lemma *evimage-Collect*: $(\bigwedge x.\ x \in carrier\ (R^n) \Longrightarrow p\ (f\ x) = Q\ x) \Longrightarrow f\ ^{-1}n$ $(Collect\ p) = Collect\ Q \cap carrier\ (R^n)$
unfolding *evimage-def* **by** *blast*

lemma *evimage-insert*: $f\ ^{-1}n\ (insert\ a\ B) = (f\ ^{-1}n\ \{a\}) \cup (f\ ^{-1}n\ B)$
— NOT suitable for rewriting because of the recurrence of $\{a\}$.
unfolding *evimage-def* **by** *blast*

lemma *evimage-Diff*: $f\ ^{-1}n\ (A - B) = (f\ ^{-1}n\ A) - (f\ ^{-1}n\ B)$
unfolding *evimage-def* **by** *blast*

lemma *evimage-UNIV* [*simp*]: $f\ ^{-1}n\ UNIV = carrier\ (R^n)$
unfolding *evimage-def* **by** *blast*

lemma *evimage-mono*: $A \subseteq B \Longrightarrow f\ ^{-1}n\ A \subseteq f\ ^{-1}n\ B$
— monotonicity
unfolding *evimage-def* **by** *blast*

lemma *evimage-image-eq*: $(f\ ^{-1}n\ (f\ `\ A)) = \{y \in carrier\ (R^n).\ \exists\, x{\in}A.\ f\,x = f\,y\}$
unfolding *evimage-def* **by** (*blast intro*: *sym*)

lemma *image-evimage-subset*: $f\ `\ (f\ ^{-1}n\ A) \subseteq A$
by *blast*

lemma *image-evimage-eq* [*simp*]: $f\ `\ (f\ ^{-1}n\ A) = A \cap (f\ `\ carrier\ (R^n))$
unfolding *evimage-def* **by** *blast*

lemma *image-subset-iff-subset-evimage*: $A \subseteq carrier\ (R^n) \Longrightarrow f\ `\ A \subseteq B \longleftrightarrow A$ $\subseteq f\ ^{-1}n\ B$

by *blast*

lemma *evimage-const* [*simp*]: $((\lambda x.\ c)\ ^{-1}_n\ A) = (\textit{if } c \in A \textit{ then carrier } (R^n) \textit{ else } \{\})$
unfolding *evimage-def* **using** *vimage-const*[*of c A*]
by (*smt Int-commute inf-bot-right inf-top.right-neutral*)

lemma *evimage-if* [*simp*]: $((\lambda x.\ \textit{if } x \in B \textit{ then } c \textit{ else } d)\ ^{-1}_n\ A) =$
$(\textit{if } c \in A \textit{ then } (\textit{if } d \in A \textit{ then carrier } (R^n) \textit{ else } B \cap \textit{carrier } (R^n)\ )$
$\textit{else if } d \in A \textit{ then } B^c{}_n \textit{ else } \{\})$
unfolding *evimage-def euminus-set-def* **using** *vimage-if*[*of B c d A*]
by (*metis Diff-Compl Diff-UNIV Diff-empty Int-commute double-compl*)

lemma *evimage-inter-cong*: $(\bigwedge w.\ w \in S \Longrightarrow f\ w = g\ w) \Longrightarrow f\ ^{-1}_n\ y \cap S = g\ ^{-1}_n\ y \cap S$
unfolding *evimage-def*
by (*smt Int-assoc Int-commute vimage-inter-cong*)

lemma *evimage-ident* [*simp*]: $(\lambda x.\ x)\ ^{-1}_n\ Y = Y \cap \textit{carrier } (R^n)$
unfolding *evimage-def*
by *blast*

end

end
theory *Padic-Fields*
imports *Fraction-Field Padic-Ints.Hensels-Lemma*

begin

8 Constructing the p-adic Valued Field

As a field, we can define the field $\mathbb{Q}_p$ immediately as the fraction field of $\mathbb{Z}_p$. The valuation can then be extended from to $\mathbb{Z}_p$ to $\mathbb{Q}_p$ by defining $\text{val}(a/b) = \text{val}\ a - \text{val}\ b$ where $a, b \in \mathbb{Z}_p$.

8.1 A Locale for p-adic Fields

This section builds a locale for reasoning about general p-adic fields for a fixed p. The locale fixes constants for the ring of p-adic integers ($\mathbb{Z}_p$) and the inclusion map $\iota : \mathbb{Z}_p \to \mathbb{Q}_p$.

type-synonym *padic-number* = $((nat \Rightarrow int) \times (nat \Rightarrow int))$ *set*
locale *padic-fields*=

fixes Q_p:: - *ring* (**structure**)
fixes Z_p:: - *ring* (**structure**)
fixes p
fixes ι
defines $Z_p \equiv$ *padic-int* p
defines $Q_p \equiv$ *Frac* Z_p
defines $\iota \equiv$ *domain-frac.inc* Z_p
assumes *prime*: *prime* p

sublocale *padic-fields* < *Zp?*: *domain-frac* Z_p
by (*simp add*: Z_p*-def domain-frac.intro padic-int-is-domain prime*)

sublocale *padic-fields* < *Qp?*: *ring* Q_p
unfolding Q_p*-def*
by (*simp add*: *Fraction-Field.domain-frac-def domain-axioms domain-frac.fraction-field-is-field field.is-ring*)

sublocale *padic-fields* < *Qp?*: *cring* Q_p
unfolding Q_p*-def*
by (*simp add*: *Fraction-Field.domain-frac-def domain.axioms*(*1*) *domain-axioms domain-frac.fraction-field-is-domain*)

sublocale *padic-fields* < *Qp?*: *field* Q_p
unfolding Q_p*-def*
by (*simp add*: *Fraction-Field.domain-frac-def domain-axioms domain-frac.fraction-field-is-field*)

sublocale *padic-fields* < *Qp?*: *domain* Q_p
by (*simp add*: *Qp.domain-axioms*)

sublocale *padic-fields* < *padic-integers* Z_p
apply (*simp add*: *padic-integers-def prime*)
using Z_p*-def* **by** *auto*

sublocale *padic-fields* < *UPQ?*: *UP-cring* Q_p *UP* Q_p
using *Qp.is-cring UP-cring-def* **apply** *blast*
by *auto*

8.2 The Valuation Ring in $\mathbb{Q}_p$

The valuation ring $\mathcal{O}_p$ is the subring of elements in $\mathbb{Q}_p$ with positive valuation. It is an isomorphic copy of $\mathbb{Z}_p$.

context *padic-fields*
begin

abbreviation $\mathcal{O}_p$ **where**
$\mathcal{O}_p \equiv \iota$ ' *carrier* Z_p

lemma *inc-closed*:
assumes $a \in$ *carrier* Z_p

shows $\iota\ a \in carrier\ Q_p$
using Q_p-*def* ι-*def assms Zp.inc-is-hom ring-hom-closed*
by *fastforce*

lemma *inc-is-hom*:
$\iota \in ring\text{-}hom\ Z_p\ Q_p$
unfolding Q_p-*def* ι-*def*
by(*rule Zp.inc-is-hom*)

An alternate formula of the map ι

lemma *inc-def*:
assumes $a \in carrier\ Z_p$
shows $\iota\ a = frac\ a\ \mathbf{1}_{Z_p}$
using *assms inc-equation*[*of a*] ι-*def* **by** *auto*

lemma *inc-of-nonzero*:
assumes $a \in nonzero\ Z_p$
shows $\iota\ a \in nonzero\ Q_p$
using ι-*def assms* Q_p-*def Qp.nonzero-memI*
Zp.nonzero-closed Zp.nonzero-one-closed frac-closed local.inc-def nonzero-fraction

by (*metis Zp.nonzero-memE*(*2*) *inc-closed inc-inj1*)

lemma *inc-of-one*:
$\iota\ \mathbf{1}_{Z_p} = \mathbf{1}$
by (*simp add*: *inc-is-hom ring-hom-one*)

lemma *inc-of-zero*:
$\iota\ \mathbf{0}_{Z_p} = \mathbf{0}$
apply(*rule ring-hom-zero*[*of* ι], *rule inc-is-hom*)
apply (*simp add*: *Zp.ring-axioms*)
by (*simp add*: *Qp.ring-axioms*)

lemma *inc-of-sum*:
assumes $a \in carrier\ Z_p$
assumes $b \in carrier\ Z_p$
shows $\iota\ (a \oplus_{Z_p} b) = (\iota\ a) \oplus (\iota\ b)$
by(*rule ring-hom-add*[*of - Z*$_p$], *rule inc-is-hom*, *rule assms*, *rule assms*)

lemma *inc-of-prod*:
assumes $a \in carrier\ Z_p$
assumes $b \in carrier\ Z_p$
shows $\iota\ (a \otimes_{Z_p} b) = (\iota\ a) \otimes (\iota\ b)$
by (*simp add*: *assms*(*1*) *assms*(*2*) *inc-is-hom ring-hom-mult*)

lemma *inc-pow*:
assumes $a \in nonzero\ Z_p$
shows $\iota\ (a[\hat{}]_{Z_p}(n::nat)) = (\iota\ a)[\hat{}]\ n$
apply(*induction n*)

apply (*simp add*: *inc-of-one*)
by (*simp add*: *assms inc-of-prod Zp.nonzero-closed*)

lemma *inc-of-diff*:
assumes $a \in$ *carrier* Z_p
assumes $b \in$ *carrier* Z_p
shows $\iota\ (a \ominus_{Z_p} b) = (\iota\ a) \ominus (\iota\ b)$
using *assms* **unfolding** *a-minus-def*
using *inc-is-hom Qp.ring-axioms* Q_p*-def Zp.ring-hom-a-inv inc-of-sum* **by** *fastforce*

lemma *Units-nonzero-Qp*:
assumes $u \in$ *Units* Q_p
shows $u \in$ *nonzero* Q_p
by (*simp add*: *Qp.Units-nonzero assms*)

lemma *Units-eq-nonzero*:
Units Q_p = *nonzero* Q_p
using *frac-nonzero-Units* **unfolding** Q_p*-def* Z_p*-def*
by *blast*

lemma *Units-inverse-Qp*:
assumes $u \in$ *Units* Q_p
shows $inv_{Q_p}\ u \in$ *Units* Q_p
using Q_p*-def Units-eq-nonzero assms frac-nonzero-inv-Unit* **by** *auto*

lemma *nonzero-inverse-Qp*:
assumes $u \in$ *nonzero* Q_p
shows $inv_{Q_p}\ u \in$ *nonzero* Q_p
using *Units-eq-nonzero Units-inverse-Qp assms* **by** *auto*

lemma *frac-add*:
assumes $a \in$ *carrier* Z_p
assumes $b \in$ *nonzero* Z_p
assumes $c \in$ *carrier* Z_p
assumes $d \in$ *nonzero* Z_p
shows (*frac a b*) $\oplus$ (*frac c d*) = (*frac* $((a \otimes_{Z_p} d) \oplus_{Z_p} (b \otimes_{Z_p} c))\ (b \otimes_{Z_p} d)$)
by (*simp add*: Q_p*-def assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *local.frac-add*)

lemma *frac-add-common-denom*:
assumes $a \in$ *carrier* Z_p
assumes $b \in$ *carrier* Z_p
assumes $c \in$ *nonzero* Z_p
shows (*frac a c*) $\oplus$ (*frac b c*) = *frac* $(a \oplus_{Z_p} b)\ c$
by (*simp add*: Q_p*-def assms*(*1*) *assms*(*2*) *assms*(*3*) *frac-add-common-denom*)

lemma *frac-mult*:
assumes $a \in$ *carrier* Z_p
assumes $b \in$ *nonzero* Z_p

assumes $c \in carrier\ Z_p$
assumes $d \in nonzero\ Z_p$
shows $(frac\ a\ b) \otimes (frac\ c\ d) = (frac\ (a \otimes_{Z_p} c)\ (b \otimes_{Z_p} d))$
by (*simp add*: Q_p*-def assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *frac-mult*)

lemma *frac-one*:
assumes $a \in nonzero\ Z_p$
shows $frac\ a\ a = \mathbf{1}$
by (*simp add*: Q_p*-def assms frac-one*)

lemma *frac-closed*:
assumes $a \in carrier\ Z_p$
assumes $b \in nonzero\ Z_p$
shows $frac\ a\ b \in carrier\ Q_p$
by (*simp add*: Q_p*-def assms*(*1*) *assms*(*2*) *frac-closed*)

lemma *inv-in-frac*:
assumes $a \in carrier\ Q_p$
assumes $a \neq \mathbf{0}$
shows $inv_{Q_p}\ a \in carrier\ Q_p$
$inv_{Q_p}\ a \neq \mathbf{0}$
$inv_{Q_p}\ a \in nonzero\ Q_p$
apply (*simp add*: *assms*(*1*) *assms*(*2*) *field-inv*(*3*))
using *assms*(*1*) *assms*(*2*) *field-inv*(*1*) **apply** *fastforce*
using *Qp.not-nonzero-memE assms*(*1*) *assms*(*2*) *nonzero-inverse-Qp* **by** *blast*

lemma *nonzero-numer-imp-nonzero-fraction*:
assumes $a \in nonzero\ Z_p$
assumes $b \in nonzero\ Z_p$
shows $frac\ a\ b \neq \mathbf{0}$
by (*simp add*: Q_p*-def assms*(*1*) *assms*(*2*) *nonzero-fraction*)

lemma *nonzero-fraction-imp-numer-not-zero*:
assumes $a \in carrier\ Z_p$
assumes $b \in nonzero\ Z_p$
assumes $frac\ a\ b \neq \mathbf{0}$
shows $a \neq \mathbf{0}_{Z_p}$
using *assms fraction-zero* Q_p*-def* **by** *blast*

lemma *nonzero-fraction-imp-nonzero-numer*:
assumes $a \in carrier\ Z_p$
assumes $b \in nonzero\ Z_p$
assumes $frac\ a\ b \neq \mathbf{0}$
shows $a \in nonzero\ Z_p$
using *assms*(*1*) *assms*(*2*) *assms*(*3*) *nonzero-fraction-imp-numer-not-zero not-nonzero-Zp*
by *blast*

lemma(**in** *padic-fields*) *frac-inv-id*:
assumes $a \in nonzero\ Z_p$

assumes $b \in nonzero\ Z_p$
assumes $c \in nonzero\ Z_p$
assumes $d \in nonzero\ Z_p$
assumes $frac\ a\ b = frac\ c\ d$
shows $frac\ b\ a = frac\ d\ c$
using *frac-inv assms*
by *metis*

lemma(**in** *padic-fields*) *frac-uminus*:
assumes $a \in carrier\ Z_p$
assumes $b \in nonzero\ Z_p$
shows $\ominus\ (frac\ a\ b) = frac\ (\ominus_{Z_p}\ a)\ b$
by (*simp add*: Q_p*-def assms*(*1*) *assms*(*2*) *frac-uminus*)

lemma(**in** *padic-fields*) *i-mult*:
assumes $a \in carrier\ Z_p$
assumes $c \in carrier\ Z_p$
assumes $d \in nonzero\ Z_p$
shows $(\iota\ a) \otimes (frac\ c\ d) = frac\ (a \otimes_{Z_p}\ c)\ d$
by (*simp add*: Q_p*-def* ι*-def assms*(*1*) *assms*(*2*) *assms*(*3*) *i-mult*)

lemma *numer-denom-facts*:
assumes $a \in carrier\ Q_p$
shows $(numer\ a) \in carrier\ Z_p$
$(denom\ a) \in nonzero\ Z_p$
$a \neq \mathbf{0} \Longrightarrow numer\ a \neq \mathbf{0}_{Z_p}$
$a \otimes (\iota\ (denom\ a)) = \iota\ (numer\ a)$
$a = frac\ (numer\ a)\ (denom\ a)$
unfolding Q_p*-def*
using Q_p*-def assms numer-denom-facts*(*2*) **apply** *auto*[*1*]
using Q_p*-def assms numer-denom-facts*(*3*) **apply** *blast*
using Q_p*-def assms numer-denom-facts*(*4*) **apply** *blast*
using Q_p*-def* ι*-def assms numer-denom-facts*(*5*) **apply** *auto*[*1*]
using Q_p*-def assms numer-denom-facts*(*1*) **by** *auto*

lemma *get-common-denominator*:
assumes $x \in carrier\ Q_p$
assumes $y \in carrier\ Q_p$
obtains *a b c* **where**
$a \in carrier\ Z_p$
$b \in carrier\ Z_p$
$c \in nonzero\ Z_p$
$x = frac\ a\ c$
$y = frac\ b\ c$
using Q_p*-def assms*(*1*) *assms*(*2*) *common-denominator*[*of x y*]
by *blast*

abbreviation *fract* :: - ⇒ - ⇒ - (**infixl** ‹÷› *50*) **where**
$(fract\ a\ b) \equiv (a \otimes (inv_{Q_p}\ b))$

fract generalizes frac

lemma *fract-frac*:
assumes $a \in$ *carrier* Z_p
assumes $b \in$ *nonzero* Z_p
shows (*frac* a b) = (ι a $\div$ ι b)
proof−
have *B*: $b \in$ *carrier* Z_p
using *Zp.nonzero-closed assms*(*2*) **by** *auto*
have *P0*:(*inv*$_{Q_p}$ (ι b)) = *frac* $\mathbf{1}_{Z_p}$ b
by (*simp add*: *B* Q_p*-def Zp.nonzero-one-closed assms*(*2*) *frac-inv local.inc-def*)
have *P1*: (*frac* a b) = (ι a) $\otimes$ (*frac* $\mathbf{1}_{Z_p}$ b)
by (*simp add*: *assms*(*1*) *assms*(*2*) *i-mult*)
show *?thesis*
by (*simp add*: *P0 P1*)
qed

lemma *frac-eq*:
assumes $a \in$ *nonzero* Z_p
assumes $b \in$ *nonzero* Z_p
assumes *frac* a b = $\mathbf{1}$
shows $a = b$
proof−
have *frac* a b = *frac* b b
by (*simp add*: *assms*(*2*) *assms*(*3*) *frac-one*)
then have *frac* a $\mathbf{1}_{Z_p}$ = *frac* b $\mathbf{1}_{Z_p}$
using *assms*
by (*metis* (*no-types*, *lifting*) *Zp.nonzero-closed*
Zp.nonzero-one-closed frac-eqE frac-eqI')
then show *?thesis*
using ι*-def assms*(*1*) *assms*(*2*) *inc-inj2 local.inc-def*
by (*simp add*: *Zp.nonzero-closed*)
qed

lemma *fract-cancel-right*:
assumes $a \in$ *carrier* Q_p
assumes $b \in$ *nonzero* Q_p
shows $b \otimes (a \div b) = a$
by (*simp add*: *Qp.Units-closed Qp.m-lcomm Units-eq-nonzero assms*(*1*)
assms(*2*))

lemma *fract-cancel-left*:
assumes $a \in$ *carrier* Q_p
assumes $b \in$ *nonzero* Q_p
shows $(a \div b) \otimes b = a$
by (*simp add*: *Qp.m-comm Qp.nonzero-closed assms*(*1*) *assms*(*2*)
local.fract-cancel-right nonzero-inverse-Qp)

lemma *fract-mult*:
assumes $a \in$ *carrier* Q_p

assumes $b \in$ *nonzero* Q_p
assumes $c \in$ *carrier* Q_p
assumes $d \in$ *nonzero* Q_p
shows $(a \div b) \otimes (c \div d) = ((a \otimes c) \div (b \otimes d))$
using Q_p*-def assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*)
by (*simp add*: *fract-mult*)

lemma *Qp-nat-pow-nonzero*:
assumes $x \in$ *nonzero* Q_p
shows $x[\hat{}](n::nat) \in$ *nonzero* Q_p
using *Qp.Units-pow-closed Units-eq-nonzero assms* **by** *auto*

lemma *Qp-nonzero-nat-pow*:
assumes $x \in$ *carrier* Q_p
assumes $n > 0$
assumes $x[\hat{}](n::nat) \in$ *nonzero* Q_p
shows $x \in$ *nonzero* Q_p
using *Frac-nonzero-nat-pow* Q_p*-def assms*(*1*) *assms*(*2*) *assms*(*3*) **by** *blast*

lemma *Qp-int-pow-nonzero*:
assumes $x \in$ *nonzero* Q_p
shows $x[\hat{}](n::int) \in$ *nonzero* Q_p
using *Frac-int-pow-nonzero* Q_p*-def assms* **by** *blast*

lemma *Qp-nonzero-int-pow*:
assumes $x \in$ *carrier* Q_p
assumes $n > 0$
assumes $x[\hat{}](n::int) \in$ *nonzero* Q_p
shows $x \in$ *nonzero* Q_p
using *Frac-nonzero-int-pow* Q_p*-def assms*
by *auto*

lemma *pow-p-frac-0*:
assumes $(m::int) \geq n$
assumes $n \geq 0$
shows $(frac\ (\mathrm{p}[\hat{}]_{Z_p} m)\ (\mathrm{p}[\hat{}]_{Z_p} n)) = \iota\ (\mathrm{p}[\hat{}]_{Z_p}(m-n))$
proof−
have *0*: $\mathrm{p} \in$ *carrier* Z_p
by (*simp add*: *Zp.nat-inc-closed*)
have *1*: $m - n \geq 0$
using *assms*(*1*) **by** *auto*
have *2*: $nat\ (m - n) + (nat\ n) = nat\ m$
using *1 assms*(*2*) **by** *linarith*
have *3*: $m \geq 0$
using *assms* **by** *auto*
then have $(\mathrm{p}[\hat{}]_{Z_p}(nat\ (m-n))) \otimes_{Z_p} (\mathrm{p}[\hat{}]_{Z_p}(nat\ n)) = (\mathrm{p}[\hat{}]_{Z_p}(nat\ m))$
by (*simp add*: *2 p-natpow-prod*)
then have $(\mathrm{p}[\hat{}]_{Z_p}(m-n)) \otimes_{Z_p} (\mathrm{p}[\hat{}]_{Z_p} n) = (\mathrm{p}[\hat{}]_{Z_p} m)$
using *int-pow-int 1 3 assms*(*2*) *int-nat-eq* **by** *metis*

then have *P0*: (*frac* (p[^]$_{Z_p}$ *m*) (p[^]$_{Z_p}$ *n*)) = *frac* ((p[^]$_{Z_p}$(*m*−*n*))⊗$_{Z_p}$ (p[^]$_{Z_p}$ *n*)) (p[^]$_{Z_p}$ *n*)
by *simp*
have p ∈*carrier* Z_p
by (*simp add*: *0*)
have (p[^]$_{Z_p}$(*nat n*)) = [(*p*^(*nat n*))] ·$_{Z_p}$ **1**$_{Z_p}$
by (*simp add*: *p-pow-rep0*)
then have (p[^]$_{Z_p}$(*nat n*)) ∈ *carrier* Z_p
by (*simp add*: *Zp-nat-inc-closed*)
then have (p[^]$_{Z_p}$ *n*) ∈ *carrier* Z_p
using *assms*(*2*) **by** (*metis int-nat-eq int-pow-int*)
then have *P1*: (*frac* (p[^]$_{Z_p}$ *m*) (p[^]$_{Z_p}$ *n*)) = *frac* ((p[^]$_{Z_p}$(*m*−*n*))⊗$_{Z_p}$ (p[^]$_{Z_p}$ *n*)) ((**1**$_{Z_p}$ ⊗$_{Z_p}$ (p[^]$_{Z_p}$ *n*)))
by (*simp add*: ‹[*p*] ·$_{Z_p}$ **1**$_{Z_p}$ [^]$_{Z_p}$ (*m* − *n*) ⊗$_{Z_p}$ [*p*] ·$_{Z_p}$ **1**$_{Z_p}$ [^]$_{Z_p}$ *n* = [*p*] ·$_{Z_p}$ **1**$_{Z_p}$ [^]$_{Z_p}$ *m*›)
have *P2*: (p[^]$_{Z_p}$(*m*−*n*)) ∈ *carrier* Z_p
using *1 p-pow-car* **by** *blast*
have *P3*: (p[^]$_{Z_p}$ *n*) ∈ *carrier* Z_p
using ‹p [^]$_{Z_p}$ *n* ∈ *carrier* Z_p›
by *blast*
have *P4*: (p[^]$_{Z_p}$ *n*) ∈ *nonzero* Z_p
using *assms*(*2*) *p-int-pow-nonzero*
by *blast*
have *P5*: **1**$_{Z_p}$ ∈ *nonzero* Z_p
using *nonzero-def*
by (*simp add*: *Zp.nonzero-one-closed*)
have (*frac* (p[^]$_{Z_p}$(*m*−*n*)) **1**$_{Z_p}$) ⊗ (*frac* (p[^]$_{Z_p}$ *n*) (p[^]$_{Z_p}$ *n*))
= *frac* ((p[^]$_{Z_p}$(*m*−*n*))⊗$_{Z_p}$ (p[^]$_{Z_p}$ *n*)) ((**1**$_{Z_p}$ ⊗$_{Z_p}$ (p[^]$_{Z_p}$ *n*)))
by (*simp add*: *P2 P3 P4 Zp.nonzero-one-closed local.frac-mult*)
then have *frac* ((p[^]$_{Z_p}$(*m*−*n*))⊗$_{Z_p}$ (p[^]$_{Z_p}$ *n*)) ((**1**$_{Z_p}$ ⊗$_{Z_p}$ (p[^]$_{Z_p}$ *n*))) = (*frac* (p[^]$_{Z_p}$(*m*−*n*)) **1**$_{Z_p}$)
by (*simp add*: *P2 P4 Zp.nonzero-one-closed frac-cancel-lr mult-comm*)
then have *P6*: (*frac* (p[^]$_{Z_p}$ *m*) (p[^]$_{Z_p}$ *n*)) = (*frac* (p[^]$_{Z_p}$(*m*−*n*)) **1**$_{Z_p}$)
using *P1* **by** *blast*
have (*frac* (p[^]$_{Z_p}$(*m*−*n*)) **1**$_{Z_p}$) = ι (p[^]$_{Z_p}$(*m*−*n*))
using *inc-def* **by** (*simp add*: *P2*)
then show *?thesis*
using *P6* **by** *blast*
qed

lemma *pow-p-frac*:
assumes (*m*::*int*) ≤ *n*
assumes *m* ≥*0*
shows (*frac* (p[^]$_{Z_p}$ *m*) (p[^]$_{Z_p}$ *n*)) = *frac* **1**$_{Z_p}$ (p[^]$_{Z_p}$(*n*−*m*))
proof−
have (*frac* (p[^]$_{Z_p}$ *n*) (p[^]$_{Z_p}$ *m*)) = ι (p[^]$_{Z_p}$(*n*−*m*))
by (*simp add*: *assms(1) assms(2) pow-p-frac-0*)

then have *P0*:(*frac* (p[⁀]$_{Z_p}$ *n*) (p[⁀]$_{Z_p}$ *m*)) = *frac* (p[⁀]$_{Z_p}$(*n*−*m*)) $\mathbf{1}_{Z_p}$
by (*simp add*: *assms*(*1*) *local.inc-def p-pow-car*)
have *P1*: $\mathbf{1}_{Z_p}$ ∈*nonzero* Z_p
by (*simp add*: *Zp.nonzero-one-closed*)
have *P2*: p[⁀]$_{Z_p}$ *n* ∈ *nonzero* Z_p
using *assms*(*1*) *assms*(*2*) *p-int-pow-nonzero* **by** *auto*
have *P3*: p[⁀]$_{Z_p}$ *m* ∈ *nonzero* Z_p
by (*simp add*: *assms*(*2*) *p-int-pow-nonzero*)
have *P4*: (p[⁀]$_{Z_p}$(*n*−*m*)) ∈ *nonzero* Z_p
by (*simp add*: *assms*(*1*) *p-int-pow-nonzero*)
show *frac* (p[⁀]$_{Z_p}$ *m*) (p[⁀]$_{Z_p}$ *n*) = *frac* $\mathbf{1}_{Z_p}$ (p[⁀]$_{Z_p}$(*n*−*m*))
using *P0 P1 P2 P3 P4 p-pow-nonzero*
by (*meson local.frac-inv-id*)
qed

The copy of the prime p living in $\mathbb{Q}_p$:

abbreviation $\mathfrak{p}$ **where**
$\mathfrak{p} \equiv [p] \cdot_{Q_p} \mathbf{1}$

lemma(**in** *domain-frac*) *frac-inc-of-nat*:
Frac-inc R ([(*n*::*nat*)]· $\mathbf{1}$) = [*n*]·$_{Frac\ R}\mathbf{1}_{Frac\ R}$
by (*simp add*: *inc-equation nat-inc-rep*)

lemma *inc-of-nat*:
(ι ([(*n*::*nat*)]·$_{Z_p}$ $\mathbf{1}_{Z_p}$)) = [*n*]·$_{Q_p}\mathbf{1}$
unfolding Q_p*-def* ι*-def*
using *frac-inc-of-nat*[*of n*]
by *auto*

lemma(**in** *domain-frac*) *frac-inc-of-int*:
Frac-inc R ([(*n*::*int*)]· $\mathbf{1}$) = [*n*]·$_{Frac\ R}\mathbf{1}_{Frac\ R}$
apply(*induction n*)
apply (*simp add*: *add-pow-int-ge inc-equation nat-inc-rep*)
by (*simp add*: *add-pow-int-lt frac-uminus inc-equation nat-inc-rep nonzero-one-closed*)

lemma *inc-of-int*:
(ι ([(*n*::*int*)]·$_{Z_p}$ $\mathbf{1}_{Z_p}$)) = [*n*]·$_{Q_p}\mathbf{1}$
unfolding Q_p*-def* ι*-def*
using *frac-inc-of-int*[*of n*]
by *auto*

lemma *p-inc*:
$\mathfrak{p}$ = ι p
by (*simp add*: *inc-of-int*)

lemma *p-nonzero*:
$\mathfrak{p}$ ∈ *nonzero* Q_p
using Z_p*-def Zp-nat-inc-closed inc-of-nonzero ord-Zp-p p-inc*
p-nonzero **by** *auto*

lemma *p-natpow-inc*:
fixes *n*::*nat*
shows $\mathfrak{p}[\widehat{\ }]n = \iota$ (p $[\widehat{\ }]_{Z_p}$ n)
by (*simp add*: *Qp.int-nat-pow-rep inc-of-int p-pow-rep0*)

lemma *p-intpow-inc*:
fixes *n*::*int*
assumes $n \geq 0$
shows $\mathfrak{p}[\widehat{\ }]n = \iota$ (p $[\widehat{\ }]_{Z_p}$ n)
using *p-natpow-inc*
by (*metis assms int-nat-eq int-pow-int*)

lemma *p-intpow*:
fixes *n*::*int*
assumes $n < 0$
shows $\mathfrak{p}[\widehat{\ }]n = (frac\ \mathbf{1}_{Z_p}$ (p $[\widehat{\ }]_{Z_p}$ $(-n)))$
proof−
have *U0*: $(\mathfrak{p}\ [\widehat{\ }]\ (nat\ (-n))) \in Units\ Q_p$
using *Qp.Units-pow-closed Units-eq-nonzero p-nonzero* **by** *blast*
have *E0*: $(\mathfrak{p}\ [\widehat{\ }]\ (nat\ (-n))) = (\mathfrak{p}\ [\widehat{\ }]\ (-n))$
using *assms* **by** (*simp add*: *int-pow-def nat-pow-def*)
then have *U1*: $(\mathfrak{p}\ [\widehat{\ }]\ \ (-n)) \in Units\ Q_p$ **using** *U0*
by *simp*
have $(\mathfrak{p}[\widehat{\ }]n) = inv_{\ Q_p}\ (\mathfrak{p}\ [\widehat{\ }]\ (nat\ (-n)))$
using *assms* **by** (*simp add*: *int-pow-def nat-pow-def*)
then have $(\mathfrak{p}[\widehat{\ }]n) = inv_{\ Q_p}\ (\mathfrak{p}\ [\widehat{\ }]\ (-n))$
using *E0* **by** *simp*
then have $(\mathfrak{p}[\widehat{\ }]n) = inv_{\ Q_p}\ \iota$ (p $[\widehat{\ }]_{Z_p}(-n)$)
using *assms p-intpow-inc* **by** *auto*
then have *E1*: $(\mathfrak{p}[\widehat{\ }]n) = inv_{\ Q_p}\ \ frac$ (p $[\widehat{\ }]_{Z_p}(-n)$) $\mathbf{1}_{Z_p}$
using *assms local.inc-def p-pow-car* **by** *auto*
have *A*: (p $[\widehat{\ }]_{Z_p}(-n)$) $\in$ *nonzero* Z_p
using *assms p-pow-nonzero p-int-pow-nonzero*
by *auto*
then show *?thesis*
using *A frac-inv inc-def E1*
by (*simp add*: Q_p*-def Zp.nonzero-one-closed*)
qed

lemma *p-natpow-closed*[*simp*]:
fixes *n*::*nat*
shows $(\mathfrak{p}[\widehat{\ }]n) \in (carrier\ Q_p)$
$(\mathfrak{p}[\widehat{\ }]n) \in (nonzero\ Q_p)$
apply *blast*
using *Qp-nat-pow-nonzero p-nonzero* **by** *blast*

lemma *nonzero-int-pow-distrib*:
assumes $a \in nonzero\ Q_p$

assumes $b \in nonzero\ Q_p$
shows $(a \otimes b)\ [\hat{}](k::int) = a[\hat{}]k \otimes b[\hat{}]k$
proof(*induction k*)
case (*nonneg n*)
then show *?case* **using** *pow-nat*[*of n - Q*$_p$]
by (*smt Qp.nat-pow-distrib Qp.nonzero-closed assms*(*1*) *assms*(*2*) *int-pow-int*)
next
case *N*: (*neg n*)
have $a \otimes b \in Units\ Q_p$
using *assms Units-eq-nonzero* **by** *blast*
hence $(a \otimes b)\ [\hat{}] - int\ (Suc\ n) = inv\ ((a \otimes b)\ [\hat{}]\ (Suc\ n))$
by (*metis Qp.int-pow-inv*′ *int-pow-int*)
then show *?case* **using**
Qp.int-pow-inv′ *Qp.int-pow-unit-closed Qp.inv-of-prod*[*of a*[^]*Suc n b*[^]*Suc n*]
Qp.nat-pow-distrib Qp.nonzero-closed Units-eq-nonzero assms(*1*) *assms*(*2*) *int-pow-int* **by** *metis*
qed

lemma *val-ring-subring*:
subring $\mathcal{O}_p\ Q_p$
using Q_p*-def* ι*-def inc-im-is-subring* **by** *blast*

lemma *val-ring-closed*:
$\mathcal{O}_p \subseteq carrier\ Q_p$
by (*simp add*: *subringE*(*1*) *val-ring-subring*)

lemma *p-pow-diff*:
fixes *n*::*int*
fixes *m*::*int*
assumes $n \geq 0$
assumes $m \geq 0$
shows $\mathfrak{p}\ [\hat{}]\ (n - m) = frac\ (\mathrm{p}[\hat{}]_{Z_p} n)\ (\mathrm{p}[\hat{}]_{Z_p} m)$
proof−
have *0*: *comm-monoid* Q_p
by (*simp add*: *Qp.comm-monoid-axioms*)
have *1*: $\mathfrak{p} \in Units\ Q_p$
using *Units-eq-nonzero p-nonzero*
by *blast*
have *2*: $\mathfrak{p}\ [\hat{}]\ (n - m) = (\mathfrak{p}[\hat{}]n) \otimes (\mathfrak{p}\ [\hat{}]\ -m)$
by (*metis 1 Qp.int-pow-add diff-conv-add-uminus*)
have *3*: $\mathfrak{p}\ [\hat{}]\ (n - m) = (\mathfrak{p}[\hat{}]n) \otimes inv_{Q_p}(\mathfrak{p}\ [\hat{}]\ m)$
by (*simp add*: *1 2 Qp.int-pow-inv*′)
then show *?thesis* **using** *assms*
using *fract-frac p-int-pow-nonzero p-intpow-inc p-pow-car* **by** *presburger*
qed

lemma *Qp-int-pow-add*:
fixes *n*::*int*

fixes m::*int*
assumes $a \in$ *nonzero* Q_p
shows $a\ [\widehat{}]\ (n + m) = (a\ [\widehat{}]\ n) \otimes (a\ [\widehat{}]\ m)$
using *monoid.int-pow-add*[*of* Q_p *a n m*] *Units-eq-nonzero assms*
by (*simp add*: *Qp.monoid-axioms*)

lemma *Qp-nat-pow-pow*:
fixes n::*nat*
fixes m::*nat*
assumes $a \in$ *carrier* Q_p
shows $(a[\widehat{}](n*m)) = ((a[\widehat{}]n)[\widehat{}]m)$
by (*simp add*: *Qp.nat-pow-pow assms*)

lemma *Qp-p-nat-pow-pow*:
fixes n::*nat*
fixes m::*nat*
shows $(\mathfrak{p}\ [\widehat{}]\ (n*m)) = ((\mathfrak{p}[\widehat{}]n)[\widehat{}]m)$
using *Qp-nat-pow-pow*
by *simp*

lemma *Qp-units-int-pow*:
fixes n::*int*
assumes $a \in$ *nonzero* Q_p
shows $a[\widehat{}]n = a[\widehat{}]_{\textit{units-of}\ Q_p} n$
apply(*cases* $n \geq 0$)
using *monoid.units-of-pow*[*of* Q_p]
apply (*metis int-pow-def2 mult-of-is-Units nat-pow-mult-of not-le*)
by (*metis Qp.Units-pow-closed Qp.units-of-inv Units-eq-nonzero assms int-pow-def2 mult-of-is-Units nat-pow-mult-of*)

lemma *Qp-int-pow-pow*:
fixes n::*int*
fixes m::*int*
assumes $a \in$ *nonzero* Q_p
shows $(a[\widehat{}](n*m)) = ((a[\widehat{}]n)[\widehat{}]m)$
proof−
have *0*: $a \in$ *carrier* (*units-of* Q_p)
by (*simp add*: *Units-eq-nonzero assms units-of-carrier*)
have *group* (*units-of* Q_p)
using *monoid.units-group Qp.units-group*
by *blast*
then show *?thesis*
using *0 group.int-pow-pow*[*of units-of* Q_p] *Qp-int-pow-nonzero Qp-units-int-pow assms*
by *auto*
qed

lemma *Qp-p-int-pow-pow*:
fixes n::*int*

fixes m::int
shows $(\mathfrak{p}\ [\widehat{}]\ (n*m)) = ((\mathfrak{p}[\widehat{}]n)[\widehat{}]m)$
using *Qp-int-pow-pow p-nonzero* **by** *blast*

lemma *Qp-int-nat-pow-pow*:
fixes n::int
fixes m::nat
assumes $a \in nonzero\ Q_p$
shows $(a[\widehat{}](n*m)) = ((a[\widehat{}]n)[\widehat{}]m)$
by (*simp add*: *Qp-int-pow-pow assms int-pow-int*)

lemma *Qp-p-int-nat-pow-pow*:
fixes n::int
fixes m::nat
shows $(\mathfrak{p}\ [\widehat{}]\ (n*m)) = ((\mathfrak{p}[\widehat{}]n)[\widehat{}]m)$
by (*simp add*: *Qp-int-nat-pow-pow p-nonzero*)

lemma *Qp-nat-int-pow-pow*:
fixes n::nat
fixes m::int
assumes $a \in nonzero\ Q_p$
shows $(a[\widehat{}](n*m)) = ((a[\widehat{}]n)[\widehat{}]m)$
by (*simp add*: *Qp-int-pow-pow assms int-pow-int*)

lemma *Qp-p-nat-int-pow-pow*:
fixes n::nat
fixes m::int
shows $(\mathfrak{p}\ [\widehat{}]\ (n*m)) = ((\mathfrak{p}[\widehat{}]n)[\widehat{}]m)$
by (*simp add*: *Qp-nat-int-pow-pow p-nonzero*)

lemma *p-intpow-closed*:
fixes n::int
shows $(\mathfrak{p}[\widehat{}]n) \in (carrier\ Q_p)$
$(\mathfrak{p}[\widehat{}]n) \in (nonzero\ Q_p)$
apply (*simp add*: *Qp.nonzero-closed Qp-int-pow-nonzero p-nonzero*)
by (*simp add*: *Qp-int-pow-nonzero p-nonzero*)

lemma *p-intpow-add*:
fixes n::int
fixes m::int
shows $\mathfrak{p}\ [\widehat{}]\ (n + m) = (\mathfrak{p}\ [\widehat{}]\ n) \otimes (\mathfrak{p}\ [\widehat{}]\ m)$
using *Qp-int-pow-add p-nonzero* **by** *blast*

lemma *p-intpow-inv*:
fixes n::int
shows $(\mathfrak{p}\ [\widehat{}]\ n) \otimes (\mathfrak{p}\ [\widehat{}]\ -n) = \mathbf{1}$
using *Units-eq-nonzero monoid.int-pow-inv'*[*of* Q_p $\mathfrak{p}$ n]
by (*metis add.right-inverse int-pow-0 p-intpow-add*)

lemma *p-intpow-inv'*:
fixes *n*::*int*
shows $(\mathfrak{p}\ [\widehat{}]\ -n) \otimes (\mathfrak{p}\ [\widehat{}]\ n) = \mathbf{1}$
using *p-intpow-inv*
by (*metis add.commute p-intpow-add*)

lemma *p-intpow-inv''*:
fixes *n*::*int*
shows $(\mathfrak{p}\ [\widehat{}]\ -n) = inv_{Q_p}\ (\mathfrak{p}\ [\widehat{}]\ n)$
by (*simp add*: *Qp.int-pow-inv' Units-eq-nonzero p-nonzero*)

lemma *p-int-pow-factor-int-pow*:
assumes $a \in nonzero\ Q_p$
shows $(\mathfrak{p}[\widehat{}](n::int) \otimes a)[\widehat{}](k::int) = \mathfrak{p}[\widehat{}](n*k) \otimes a[\widehat{}]k$
using *assms nonzero-int-pow-distrib p-intpow-closed*(*2*) *Qp-p-int-pow-pow* **by** *presburger*

lemma *p-nat-pow-factor-int-pow*:
assumes $a \in nonzero\ Q_p$
shows $(\mathfrak{p}[\widehat{}](n::nat) \otimes a)[\widehat{}](k::int) = \mathfrak{p}[\widehat{}](n*k) \otimes a[\widehat{}]k$
using *assms Qp-p-int-nat-pow-pow p-natpow-closed*(*1*)
by (*metis int-pow-int p-int-pow-factor-int-pow*)

lemma *p-pow-factor*:
$\mathfrak{p}[\widehat{}]((int\ N)*l\ +\ k) = (\mathfrak{p}[\widehat{}]l)[\widehat{}](N::nat) \otimes \mathfrak{p}[\widehat{}]\ k$
by (*metis Qp-p-int-nat-pow-pow mult-of-nat-commute p-intpow-add*)

lemma *p-nat-pow-factor-nat-pow*:
assumes $a \in carrier\ Q_p$
shows $(\mathfrak{p}[\widehat{}](n::nat) \otimes a)[\widehat{}](k::nat) = \mathfrak{p}[\widehat{}](n*k) \otimes a[\widehat{}]k$
using *Qp.nat-pow-distrib Qp-p-nat-pow-pow assms p-natpow-closed*(*1*) **by** *presburger*

lemma *p-int-pow-factor-nat-pow*:
assumes $a \in carrier\ Q_p$
shows $(\mathfrak{p}[\widehat{}](n::int) \otimes a)[\widehat{}](k::nat) = \mathfrak{p}[\widehat{}](n*k) \otimes a[\widehat{}]k$
using *assms Qp.nat-pow-distrib Qp-p-int-nat-pow-pow p-intpow-closed*(*1*) **by** *presburger*

lemma(**in** *ring*) *r-minus-distr*:
assumes $a \in carrier\ R$
assumes $b \in carrier\ R$
assumes $c \in carrier\ R$
shows $a \otimes b \ominus a \otimes c = a \otimes (b \ominus c)$
using *assms*
unfolding *a-minus-def*
by (*simp add*: *r-distr r-minus*)

8.3 The Valuation on $\mathbb{Q}_p$

8.3.1 Extending the Valuation from $\mathbb{Z}_p$ to $\mathbb{Q}_p$

The valuation of a p-adic number can be defined as the difference of the valuations of an arbitrary choice of numerator and denominator.

definition *ord* **where**
ord x = (*ord-Zp* (*numer x*)) − (*ord-Zp* (*denom x*))

definition *val* **where**
val x = (*if x* = **0** *then* (∞::*eint*) *else eint* (*ord x*))

lemma *val-ord*[*simp*]:
assumes *a* ∈ *nonzero* Q_p
shows *val a* = *ord a*
using *assms nonzero-def val-def* **by** *force*

8.3.2 Properties of the Valuation

lemma *ord-of-frac*:
assumes *a* ∈ *nonzero* Z_p
assumes *b* ∈ *nonzero* Z_p
shows *ord* (*frac a b*) = (*ord-Zp a*) − (*ord-Zp b*)
proof−
have *frac a b* = *frac* (*numer* (*frac a b*)) (*denom* (*frac a b*))
by (*simp add*: *assms*(*1*) *assms*(*2*))
then have *a* ⊗$_{Z_p}$ (*denom* (*frac a b*)) = *b* ⊗$_{Z_p}$ (*numer* (*frac a b*))
by (*simp add*: *assms*(*1*) *assms*(*2*) *numer-denom-swap*)
then have (*ord-Zp a*) − (*ord-Zp b*) = (*ord-Zp* (*numer* (*frac a b*))) − (*ord-Zp* (*denom* (*frac a b*)))
using *ord-Zp-eq-frac* Q_p*-def* Z_p*-def*
by (*metis Zp.frac-closed Zp.nonzero-closed Zp.numer-denom-facts*(*4*) *assms*(*1*) *assms*(*2*) *local.numer-denom-facts*(*1*) *local.numer-denom-facts*(*2*) *nonzero-fraction ord-of-nonzero*(*2*) *ord-pos*)
then show *?thesis*
using *ord-def*
by *presburger*
qed

lemma *val-zero*:
val **0** = ∞ **by** (*simp add*: *val-def*)

lemma *ord-one*[*simp*]:
ord **1** = *0*
using *Zp.nonzero-one-closed local.frac-one ord-of-frac* **by** *fastforce*

lemma *val-one*[*simp*]:
val (**1**) = *0*
using *ord-one*

by (*simp add*: *Qp.one-nonzero zero-eint-def*)

lemma *val-of-frac*:
assumes $a \in$ *carrier* Z_p
assumes $b \in$ *nonzero* Z_p
shows *val* (*frac a b*) = (*val-Zp a*) − (*val-Zp b*)
proof(*cases* $a = \mathbf{0}_{Z_p}$)
case *True*
then show *?thesis*
using *assms*(*1*) *assms*(*2*) *local.val-zero*
by (*simp add*: Q_p*-def val-Zp-def*)
next
case *False*
then have $a \in$ *nonzero* Z_p
by (*simp add*: *assms*(*1*) *nonzero-def*)
then show *?thesis*
using *ord-of-frac*[*of a b*] *assms*(*2*) *val-def val-ord-Zp*
nonzero-numer-imp-nonzero-fraction
by (*simp add*: *Zp.nonzero-memE*(*2*))
qed

lemma Z_p*-division-Qp-0*[*simp*]:
assumes $u \in$ *Units* Z_p
assumes $v \in$ *Units* Z_p
shows *frac* ($u \otimes_{Z_p}$ (inv_{Z_p} v)) $\mathbf{1}_{Z_p}$= *frac u v*
proof−
have *0*: *frac v v* = $\mathbf{1}$
using *frac-one*
by (*simp add*: Q_p*-def Zp.Units-nonzero assms*(*2*))
have *1*:(inv_{Z_p} v) ∈ *carrier* Z_p
using *assms* **by** *blast*
have *2*:*frac* ($u \otimes_{Z_p}$ (inv_{Z_p} v)) $\mathbf{1}_{Z_p}$ ∈ *carrier* Q_p
by (*simp add*: *1 Zp.Units-closed Zp.nonzero-one-closed assms*(*1*) *local.frac-closed*)
have *3*: *frac* ($u \otimes_{Z_p}$ (inv_{Z_p} v)) $\mathbf{1}_{Z_p}$ = (*frac* ($u \otimes_{Z_p}$ (inv_{Z_p} v)) $\mathbf{1}_{Z_p}$) ⊗ *frac v v*
by (*simp add*: *0 2*)
then have *4*: *frac* ($u \otimes_{Z_p}$ (inv_{Z_p} v)) $\mathbf{1}_{Z_p}$ = (*frac* (($u \otimes_{Z_p}$ (inv_{Z_p} v)) $\otimes_{Z_p}$ v) v)
by (*simp add*: *Zp.Units-nonzero Zp.nonzero-closed assms*(*1*) *assms*(*2*) *frac-eqI'*)

then have *4*: *frac* ($u \otimes_{Z_p}$ (inv_{Z_p} v)) $\mathbf{1}_{Z_p}$ = (*frac* ($u \otimes_{Z_p}$ ((inv_{Z_p} v) $\otimes_{Z_p}$ v)) v)
by (*simp add*: *mult-assoc*)
have *5*:(inv_{Z_p} v) $\otimes_{Z_p}$ v = $\mathbf{1}_{Z_p}$
by (*simp add*: *assms*(*2*))
then show *frac* ($u \otimes_{Z_p}$ (inv_{Z_p} v)) $\mathbf{1}_{Z_p}$ = (*frac u v*)
by (*simp add*: *4 Zp.Units-closed assms*(*1*))
qed

lemma Z_p*-division-Qp-1*:

assumes $u \in$ *Units* Z_p
assumes $v \in$ *Units* Z_p
obtains w **where** $w \in$ *Units* Z_p
$\iota\ w =$ *frac* $u\ v$
proof−
have $(inv_{Z_p}\ v) \in$ *Units* Z_p
by (*simp add*: *assms*(2))
then have $(u \otimes_{Z_p} (inv_{Z_p}\ v)) \in$ *Units* Z_p
using *assms*
by *blast*
then show *?thesis*
using Z_p*-division-Qp-0 Zp.Units-closed assms*(1) *assms*(2)
local.inc-def that **by** *presburger*
qed

lemma *val-ring-ord-criterion*:
assumes $a \in$ *carrier* Q_p
assumes $a \neq \mathbf{0}$
assumes *ord* $a \geq 0$
shows $a \in \mathcal{O}_p$
proof−
obtain $c\ d$ **where** *P0*: $a =$ *frac* $c\ d$ **and** *P1*: $c \in$ *nonzero* Z_p **and** *P2*: $d \in$ *nonzero* Z_p
by (*metis assms*(1) *assms*(2) *get-common-denominator nonzero-fraction-imp-nonzero-numer*)

obtain $m\ n$ **where** *P3*: $m =$ *ord-Zp* c **and** *P4*:$n =$ *ord-Zp* d
by *metis*
obtain u **where** $u =$ *ac-Zp* c
by *simp*
then have *P5*:$c = (\mathrm{p}[\hat{}]_{Z_p} m) \otimes_{Z_p} u$ **and** *P6*:$u \in$ *Units* Z_p
apply (*simp add*: *P1 P3* ‹$u =$ *ac-Zp* c› *ac-Zp-factors'*)
using *P1 Zp.nonzero-memE*
by (*simp add*: ‹$u =$ *ac-Zp* c› *ac-Zp-is-Unit*)
obtain v **where** $v =$ *ac-Zp* d **by** *simp*
have *P7*:$d = (\mathrm{p}[\hat{}]_{Z_p} n) \otimes_{Z_p} v$ **and** *P8*:$v \in$ *Units* Z_p
using *P2 P4* ‹$v =$ *ac-Zp* d› *ac-Zp-factors'* **apply** *blast*
using *P2 Zp.nonzero-memE*
by (*simp add*: ‹$v =$ *ac-Zp* d› *ac-Zp-is-Unit*)
have *P9*: $a =$ *frac* $((\mathrm{p}[\hat{}]_{Z_p} m) \otimes_{Z_p} u)\ ((\mathrm{p}[\hat{}]_{Z_p} n) \otimes_{Z_p} v)$
by (*simp add*: *P0 P5 P7*)
have *P10*: $(\mathrm{p}[\hat{}]_{Z_p} m) \in$ *carrier* Z_p
using *P1 P3* Z_p*-def ord-pos Zp.nonzero-closed Zp.nonzero-memE*(2) *p-pow-car*
by *auto*
have *P11*: $(\mathrm{p}[\hat{}]_{Z_p} n) \in$ *nonzero* Z_p
by (*simp add*: *P2 P4 Zp.nonzero-closed Zp.nonzero-memE*(2) *ord-pos p-int-pow-nonzero*)
have *P12*: $u \in$ *carrier* Z_p
using *P6 Units-def*
by *blast*
have *P13*: $v \in$ *nonzero* Z_p

using *P8 Units-def ord-of-nonzero(2)*
by (*simp add*: *Zp.Units-nonzero*)
have *P14*: $a = (frac\ (\mathrm{p}[\hat{}]_{Z_p} m)\ (\mathrm{p}[\hat{}]_{Z_p} n)) \otimes (frac\ u\ v)$
using *P12 P13 P10 P9 P11* Q_p*-def frac-mult* **by** *metis*
have *P15*: $m \geq n$
proof−
have *ord-Zp c* ≥ *ord-Zp d*
using *P0 P1 P2 assms(3) ord-of-frac*[*of c d*]
by (*metis P3 P4 antisym eq-iff eq-iff-diff-eq-0 le-cases le-iff-diff-le-0 ord-Zp-def*)

then show *?thesis*
using *P3 P4* **by** *blast*
qed
have *P16*: $n \geq 0$
by (*simp add*: *P2 P4 Zp.nonzero-closed Zp.nonzero-memE(2) ord-pos*)
have *P17*: $a = (frac\ (\mathrm{p}[\hat{}]_{Z_p}(m-n))\ \mathbf{1}_{Z_p}) \otimes (frac\ u\ v)$
using *P14 P15 P16 local.inc-def*[*of* $(\mathrm{p}\ [\hat{}]_{Z_p}\ (n - m))$] *pow-p-frac-0*[*of n m*]
by (*simp add*: *local.inc-def p-pow-car*)
obtain *w* **where** *P18*: $w \in Units\ Z_p$ **and** *P19*: $\iota\ w = frac\ u\ v$
using Z_p*-division-Qp-1 P6 P8* **by** *blast*
have *P20*: $w \in carrier\ Z_p$
using *P18 Units-def* **by** *blast*
have *P21*: $a = \iota\ (\mathrm{p}[\hat{}]_{Z_p}(m-n)) \otimes \iota\ w$
using *P15 P17 P19* ι*-def inc-equation p-pow-car* **by** *auto*
have *P22*: $a = (frac\ (\mathrm{p}[\hat{}]_{Z_p}(m-n))\ \mathbf{1}_{Z_p}) \otimes (frac\ w\ \mathbf{1}_{Z_p})$
using *P17 P19 P20 local.inc-def* **by** *auto*
have *P23*: $\mathrm{p}[\hat{}]_{Z_p}(m-n) \in carrier\ Z_p$
by (*simp add*: *P15 p-pow-car*)
have *P24*: $a = (frac\ ((\mathrm{p}[\hat{}]_{Z_p}(m-n)) \otimes_{Z_p} w)\ \mathbf{1}_{Z_p})$
using *P20 P22 P23 frac-mult*
by (*simp add*: *Zp.nonzero-one-closed*)
have *P24*: $a = \iota((\mathrm{p}[\hat{}]_{Z_p}(m-n)) \otimes_{Z_p} w)$
by (*simp add*: *P20 P23 P24 cring.cring-simprules(5) domain.axioms(1) local.inc-def*)
then show *?thesis*
using *P20 P23* **by** *blast*
qed

lemma *val-ring-val-criterion*:
assumes $a \in carrier\ Q_p$
assumes $val\ a \geq 0$
shows $a \in \mathcal{O}_p$
apply(*cases* $a = \mathbf{0}$)
using *Qp.int-inc-zero* Q_p*-def inc-of-int* **apply** *blast*
using *assms* **unfolding** *val-def*
by (*simp add*: *val-ring-ord-criterion zero-eint-def*)

lemma *ord-of-inv*:
assumes $a \in carrier\ Q_p$

assumes $a \neq \mathbf{0}$
shows $ord\ (inv_{Q_p}\ a) = -\ (ord\ a)$
proof−
obtain $b\ c$ **where**
Frac: $a = frac\ b\ c$ **and**
Car: $b \in carrier\ Z_p$ **and**
Nz-c: $c \in nonzero\ Z_p$
using *assms*(1) *local.numer-denom-facts*(1) *local.numer-denom-facts*(2)
local.numer-denom-facts(5) **by** *blast*
have *Nz-b*: $b \in nonzero\ Z_p$
using *Frac Car Nz-c assms*(2) *nonzero-fraction-imp-nonzero-numer* **by** *metis*

then have $(inv_{Q_p}\ a) = frac\ c\ b$
using *Frac Nz-c frac-inv* Q_p*-def*
by *auto*
then show *?thesis* **using** *Frac Nz-b Nz-c ord-of-frac*[*of b c*] *ord-of-frac*[*of c b*]
by (*simp add*: *nonzero-def ord-Zp-def*)
qed

lemma *val-of-inv*:
assumes $a \in carrier\ Q_p$
assumes $a \neq \mathbf{0}$
shows $val\ (inv_{Q_p}\ a) = -\ (val\ a)$
using *ord-of-inv* **unfolding** *uminus-eint-def*
by (*simp add*: *assms*(1) *assms*(2) *inv-in-frac*(2) *val-def*)

Zp is a valuation ring in Qp

lemma Z_p*-mem*:
assumes $a \in carrier\ Q_p$
shows $a \in \mathcal{O}_p \vee (inv_{Q_p}\ a \in \mathcal{O}_p)$
proof(*cases* $inv_{Q_p} a \in \mathcal{O}_p \vee a = \mathbf{0}$)
case *True*
then show *?thesis*
using *val-ring-subring subringE*(2) **by** *auto*
next
case *False*
then have *Nz*: $a \neq \mathbf{0}$ **by** *auto*
have $\neg\ (ord\ a < 0)$
proof
assume $ord\ a < 0$
then have $ord\ (inv_{Q_p}\ a) > 0$
by (*simp add*: *assms*(1) *Nz ord-of-inv*)
then have 0: $ord\ (inv_{Q_p}\ a) \geq 0$
by *auto*
have 1: $(inv_{Q_p}\ a) \in carrier\ Q_p$
by (*simp add*: *assms*(1) *Nz inv-in-frac*)
have 2: $(inv_{Q_p}\ a) \neq \mathbf{0}$
by (*simp add*: *assms*(1) *Nz inv-in-frac*(2))
then have $(inv_{Q_p}\ a) \in \mathcal{O}_p$

using *val-ring-ord-criterion* **by** (*simp add*: *0 1*)
then show *False*
using *False* **by** *blast*
qed
then show *?thesis*
using *val-ring-ord-criterion assms*(*1*) *Nz* **by** *auto*
qed

lemma *Qp-val-ringI*:
assumes $a \in$ *carrier* Q_p
assumes *val* $a \geq 0$
shows $a \in \mathcal{O}_p$
using *assms val-ring-val-criterion* **by** *blast*

Criterion for determining when an element in Qp is zero

lemma *val-nonzero*:
assumes $a \in$ *carrier* Q_p
assumes $s >$ *val* a
shows $a \in$ *nonzero* Q_p
proof−
have *val* $a \neq \infty$
by (*metis assms*(*2*) *eint-ord-simps*(*6*))
then show *?thesis*
using *assms*
by (*metis* (*mono-tags*, *opaque-lifting*) *local.val-zero not-nonzero-Qp*)
qed

lemma *val-ineq*:
assumes $a \in$ *carrier* Q_p
assumes *val* $\mathbf{0} \leq$ *val* a
shows $a = \mathbf{0}$
using *assms* **unfolding** *val-def*
by (*metis* (*mono-tags*, *lifting*) *eint-ord-code*(*5*))

lemma *ord-minus*:
assumes $a \in$ *nonzero* Q_p
shows *ord* $a =$ *ord* $(\ominus a)$
proof−
have $\ominus\ a = \ominus$ (*frac* (*numer a*) (*denom a*))
using *assms Qp.nonzero-closed local.numer-denom-facts*(*5*) **by** *auto*
then have $\ominus\ a =$ (*frac* $(\ominus_{Z_p}$ (*numer a*)) (*denom a*))
by (*simp add*: *Qp.nonzero-closed assms local.frac-uminus local.numer-denom-facts*(*1*)
local.numer-denom-facts(*2*))
then show *?thesis*
by (*metis* Q_p*-def Qp.add.inv-eq-1-iff Qp.nonzero-closed Zp.add.inv-closed assms*
local.numer-denom-facts(*1*) *local.numer-denom-facts*(*2*) *nonzero-fraction-imp-nonzero-numer*
numer-nonzero ord-Zp-of-a-inv ord-def ord-of-frac)

qed

lemma *val-minus*:
assumes $a \in$ *carrier* Q_p
shows *val* a = *val* $(\ominus a)$
proof(*cases* $a = \mathbf{0}$)
case *True*
then show *?thesis*
using *Qp.minus-zero* **by** *presburger*
next
case *False*
then show *?thesis* **using** *Qp.domain-axioms assms cring.cring-simprules*(*21*)
cring.cring-simprules(*22*) *domain.axioms*(*1*) *not-nonzero-Qp*
ord-minus val-def
by *metis*
qed

The valuation is multiplicative:

lemma *ord-mult*:
assumes $x \in$ *nonzero* Q_p
assumes $y \in$ *nonzero* Q_p
shows (*ord* $(x \otimes y)$) = (*ord* x) + (*ord* y)
proof−
have *0*:$x \in$ *carrier* Q_p **using** *assms* **by**(*simp add*:*nonzero-def*)
have *1*:$y \in$*carrier* Q_p **using** *assms* **by**(*simp add*:*nonzero-def*)
obtain *a b c* **where**
A: $a \in$ *carrier* Z_p **and**
B: $b \in$ *carrier* Z_p **and**
C: $c \in$ *nonzero* Z_p **and**
Fx: x = *frac* a c **and**
Fy: y = *frac* b c
using *get-common-denominator 0 1* **by** *blast*
have *An*: $a \in$ *nonzero* Z_p
using *A C Fx assms*(*1*) *nonzero-def nonzero-fraction-imp-nonzero-numer*
Qp.nonzero-memE(*2*) **by** *auto*
have *Bn*: $b \in$ *nonzero* Z_p
using *B C Fy assms*(*2*) *nonzero-def nonzero-fraction-imp-nonzero-numer*
Qp.nonzero-memE(*2*) **by** *auto*
have *Fxy*: $x \otimes y$ = *frac* $(a \otimes_{Z_p} b)$ $(c \otimes_{Z_p} c)$
by (*simp add*: *A B C Fx Fy frac-mult*)
have *Cn*: $c \otimes_{Z_p} c \in$ *nonzero* Z_p
using *C Localization.submonoid.m-closed Zp.domain-axioms domain.nonzero-is-submonoid*

by *metis*
have *Ordxy0*: *ord* $(x \otimes y)$ = *ord-Zp* $(a \otimes_{Z_p} b)$ − *ord-Zp* $(c \otimes_{Z_p} c)$
by (*metis 0 1 An Bn C Cn Fx Fxy Fy Qp.integral*
Zp.nonzero-mult-closed Zp.zero-closed fraction-zero
nonzero-fraction nonzero-fraction-imp-nonzero-numer ord-of-frac)
have *Ordxy1*: *ord* $(x \otimes y)$ = (*ord-Zp* a) + (*ord-Zp* b) − ((*ord-Zp* c) + (*ord-Zp*

c))
  using *An Bn C*
  by (*simp add*: *Ordxy0 ord-Zp-mult*)
 show *?thesis*
 proof−
  have *ord x* + *ord y* = (*ord-Zp a*) − (*ord-Zp c*) + ((*ord-Zp b*) − (*ord-Zp c*))
   using *An Bn C Fx Fy ord-of-frac*[*of a c*] *ord-of-frac*[*of b c*] **by** *presburger*
  then show *?thesis*
   using *Ordxy1*
   by *presburger*
 qed
qed

lemma *val-mult0*:
 assumes *x* ∈ *nonzero* Q_p
 assumes *y* ∈ *nonzero* Q_p
 shows (*val* (*x* ⊗ *y*)) = (*val x*) + (*val y*)
proof−
 have *0*: *val x* = *ord x*
  using *assms*(*1*) *val-ord* **by** *metis*
 have *1*: *val y* = *ord y*
  using *assms*(*2*) *val-ord* **by** *metis*
 have *x* ⊗ *y* ≠ **0**
  using *field-axioms assms*(*1*) *assms*(*2*) *integral Qp.integral-iff*
   Qp.nonzero-closed Qp.nonzero-memE(*2*) **by** *presburger*
 then have *2*: *val* (*x* ⊗ *y*) = *ord* (*x* ⊗ *y*)
  by (*simp add*: *val-def*)
 have *3*: *val x* + *val y* = *ord x* + *ord y*
  by (*simp add*: *0 1*)
 have *4*: *val* (*x* ⊗ *y*) = *ord* (*x* ⊗ *y*)
  using *2* **by** *auto*
 then show *?thesis* **using** *3 4 ord-mult assms nonzero-def*
  by (*simp add*: *nonzero-def*)
qed

val is multiplicative everywhere

lemma *val-mult*:
 assumes *x* ∈ *carrier* Q_p
 assumes *y* ∈ *carrier* Q_p
 shows (*val* (*x* ⊗ *y*)) = (*val x*) + (*val y*)
 apply(*cases x* = **0** ∨ *y* = **0**)
 using *assms local.val-zero* **apply** *auto*[*1*]
  by (*meson assms*(*1*) *assms*(*2*) *not-nonzero-Qp val-mult0*)

val and ord are compatible with inclusion

lemma *ord-of-inc*:
assumes *x* ∈ *nonzero* Z_p
shows *ord-Zp x* = *ord*(*ι x*)
proof−

have $\iota\ x = frac\ x\ \mathbf{1}_{Z_p}$
using *assms*(*1*)
by (*simp add*: *Zp.nonzero-closed local.inc-def*)
then have *ord* ($\iota\ x$) = *ord-Zp x* − *ord-Zp* $\mathbf{1}_{Z_p}$
using *assms*(*1*) *ord-of-frac*
by (*simp add*: *Zp.nonzero-one-closed*)
then show *?thesis*
using *ord-Zp-one*
by (*simp add*: *ord-Zp-def*)
qed

lemma *val-of-inc*:
assumes $x \in carrier\ Z_p$
shows *val-Zp x* = *val* ($\iota\ x$)
proof(*cases* $x \in nonzero\ Z_p$)
case *True*
then show *?thesis*
using *inc-of-nonzero nonzero-def ord-Zp-def ord-of-inc val-Zp-def val-ord*
by (*simp add*: *nonzero-def*)
next
case *False*
then show *?thesis*
by (*metis Zp.nonzero-memI Zp.nonzero-one-closed assms local.inc-def nonzero-fraction-imp-numer-not-zero*
val-Zp-def val-def)
qed

lemma *Qp-inc-id*:
assumes $a \in nonzero\ Q_p$
assumes *ord a* $\geq$*0*
obtains *b* **where** $b \in nonzero\ Z_p$ **and** $a = \iota\ b$
using *assms*
by (*metis* (*no-types*, *opaque-lifting*) *Qp.nonzero-closed Qp.nonzero-memE*(*2*)
Zp.nonzero-one-closed Zp.zero-closed Zp-defs(*2*) *val-ring-ord-criterion image-iff*
local.inc-def
nonzero-fraction-imp-numer-not-zero not-nonzero-Zp that)

lemma *val-ring-memI*:
assumes $a \in carrier\ Q_p$
assumes *val a* $\geq$ *0*
shows $a \in \mathcal{O}_p$
using *assms Qp-val-ringI* **by** *blast*

lemma *val-ring-memE*:
assumes $a \in \mathcal{O}_p$
shows *val a* $\geq$ *0* $a \in carrier\ Q_p$
using *assms val-of-inc val-pos* **apply** *auto*[*1*]
using *assms inc-closed* **by** *auto*

lemma *val-ring-add-closed*:

assumes $a \in \mathcal{O}_p$
assumes $b \in \mathcal{O}_p$
shows $a \oplus b \in \mathcal{O}_p$
using *val-ring-subring subringE*(7) **by** (*metis assms*(1) *assms*(2))

lemma *val-ring-times-closed*:
assumes $a \in \mathcal{O}_p$
assumes $b \in \mathcal{O}_p$
shows $a \otimes b \in \mathcal{O}_p$
using *val-ring-subring subringE*(6) **by** (*metis assms*(1) *assms*(2))

lemma *val-ring-ainv-closed*:
assumes $a \in \mathcal{O}_p$
shows $\ominus a \in \mathcal{O}_p$
using *val-ring-subring subringE*(5) **by** (*metis assms*)

lemma *val-ring-minus-closed*:
assumes $a \in \mathcal{O}_p$
assumes $b \in \mathcal{O}_p$
shows $a \ominus b \in \mathcal{O}_p$
using *assms val-ring-subring val-ring-ainv-closed val-ring-add-closed*
unfolding *a-minus-def* **by** *blast*

lemma *one-in-val-ring*:
$\mathbf{1} \in \mathcal{O}_p$
apply(*rule val-ring-memI*)
apply *blast*
unfolding *val-one* **by** *blast*

lemma *zero-in-val-ring*:
$\mathbf{0} \in \mathcal{O}_p$
apply(*rule val-ring-memI*)
apply *blast*
unfolding *val-zero*
by *simp*

lemma *ord-p*:
ord $\mathfrak{p} = 1$
using *p-nonzero ord-Zp-p ord-of-inc p-inc*
by (*smt Zp-int-inc-closed ord-of-nonzero*(2))

lemma *ord-p-pow-nat*:
ord $(\mathfrak{p}\ [\widehat{}]\ (n::nat)) = n$
using *p-pow-nonzero ord-Zp-p ord-of-inc p-inc ord-Zp-p-pow p-natpow-inc p-pow-nonzero'*

by *auto*

lemma *ord-p-pow-int*:
ord $(\mathfrak{p}\ [\widehat{}]\ (n::int)) = n$

proof(*cases* $n \geq 0$)
case *True*
then show *?thesis*
by (*metis int-nat-eq int-pow-int ord-p-pow-nat*)
next
case *False*
then have *Neg*: $n < 0$ **by** *auto*
then have *0*: $\mathfrak{p}[\widehat{\ }]n = frac\ \mathbf{1}_{Z_p}\ (\mathrm{p}[\widehat{\ }]_{Z_p}(-n))$
using *p-intpow* **by** *auto*
have $(\mathrm{p}[\widehat{\ }]_{Z_p}(-n)) \in nonzero\ Z_p$
using *False p-int-pow-nonzero*
by (*simp add*: *nonzero-def*)
then have *ord* $(\mathfrak{p}\ [\widehat{\ }]\ (n::int)) = ord\text{-}Zp\ \mathbf{1}_{Z_p} - ord\text{-}Zp\ (\mathrm{p}[\widehat{\ }]_{Z_p}(-n))$
using *0 ord-of-frac*
by (*simp add*: *Zp.nonzero-one-closed*)
then have *ord* $(\mathfrak{p}\ [\widehat{\ }]\ (n::int)) = -\ ord\text{-}Zp\ (\mathrm{p}[\widehat{\ }]_{Z_p}(-n))$
using *ord-Zp-one* **by** *linarith*
then have *ord* $(\mathfrak{p}\ [\widehat{\ }]\ (n::int)) = -(-n)$
using *Neg ord-Zp-p-int-pow*
by (*metis int.lless-eq neg-0-le-iff-le*)
then show *?thesis*
by *auto*
qed

lemma *ord-nonneg*:
assumes $x \in \mathcal{O}_p$
assumes $x \neq \mathbf{0}$
shows *ord* $x \geq 0$
proof−
obtain *a* **where** *A*: $x = \iota\ a \wedge a \in carrier\ Z_p$
using *assms*(*1*) **by** *blast*
then have $a \in nonzero\ Z_p$ **using** *assms*(*2*)
local.inc-def nonzero-fraction-imp-numer-not-zero not-nonzero-Zp
using *Zp.nonzero-one-closed* **by** *blast*
then have *ord-Zp a* ≥ 0
using *A*
by (*simp add*: *Zp.nonzero-memE*(*2*) *ord-pos*)
then show *?thesis*
using *A* ‹$a \in nonzero\ Z_p$› *ord-of-inc* **by** *metis*
qed

lemma *val-p*:
val $\mathfrak{p} = 1$
using *p-inc val-Zp-p val-of-inc val-def ord-p p-nonzero val-ord*
by *simp*

lemma *val-p-int-pow*:
val $(\mathfrak{p}[\widehat{\ }](k::int)) = k$
using *ord-p-pow-int p-intpow-closed*(*2*) *val-ord* **by** *presburger*

lemma *val-p-int-pow-neg*:
val (𝔭[^](−*k*::*int*)) = − *eint k*
by (*metis eint.distinct*(*2*) *local.val-zero p-intpow-closed*(*1*) *p-intpow-inv''* *val-of-inv* *val-p-int-pow*)

lemma *nonzero-nat-pow-ord*:
assumes *a* ∈ *nonzero* Q_p
shows *ord* (*a* [^] (*n*::*nat*)) = *n* ∗ *ord a*
apply(*induction n*)
apply *simp*
using *Qp-nat-pow-nonzero assms semiring-normalization-rules*(*2*)
by (*simp add*: *semiring-normalization-rules*(*2*) *ord-mult*)

lemma *add-cancel-eint-geq*:
assumes (*eint a*) + *x* ≥ (*eint a*) + *y*
shows *x* ≥*y*
using *assms eint-add-left-cancel-le* **by** *blast*

lemma(**in** *padic-fields*) *prod-equal-val-imp-equal-val*:
assumes *a* ∈ *nonzero* Q_p
assumes *b* ∈ *carrier* Q_p
assumes *c* ∈ *carrier* Q_p
assumes *val* (*a* ⊗ *b*) = *val* (*a* ⊗ *c*)
shows *val b* = *val c*
proof(*cases b* = **0**)
case *True*
then have *val* (*a* ⊗ *b*) = ∞
using *Qp.nonzero-closed Qp.r-null assms*(*1*) *local.val-zero* **by** *presburger*
then have *c* = **0**
using *assms True*
by (*metis Qp.integral Qp.nonzero-closed Qp.nonzero-mult-in-car Qp.not-nonzero-memI* *eint-ord-simps*(*3*) *not-nonzero-Qp val-ineq*)
then show *?thesis*
using *True* **by** *blast*
next
case *False*
obtain *n* **where** *n-def*: *val a* = *eint n*
using *assms val-ord* **by** *blast*
then show *?thesis* **using** *val-mult*[*of a b*] *val-mult*[*of a c*] **unfolding** *n-def*
by (*simp add*: *Qp.nonzero-closed assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*))
qed

lemma *two-times-eint*:
shows *2*∗(*x*::*eint*) = *x* + *x*
by (*metis eint-2-minus-1-mult eint-add-cancel-fact plus-eq-infty-iff-eint times-eint-simps*(*4*))

lemma *times-cfs-val-mono*:
assumes $u \in Units\ Q_p$
assumes $a \in carrier\ Q_p$
assumes $b \in carrier\ Q_p$
assumes $val\ (u \otimes a) \leq val\ (u \otimes b)$
shows $val\ a \leq val\ b$
proof−
have $eint\ (ord\ u) + val\ a \leq eint\ (ord\ u) + val\ b$
using *assms val-ord val-mult Qp.Units-nonzero Qp.nonzero-closed Units-eq-nonzero*
by *metis*
thus *?thesis*
apply(*induction val a*)
using *add-cancel-eint-geq* **apply** *blast*
using *add-cancel-eint-geq* **by** *blast*
qed

lemma *times-cfs-val-mono′*:
assumes $u \in Units\ Q_p$
assumes $a \in carrier\ Q_p$
assumes $b \in carrier\ Q_p$
assumes $val\ (u \otimes a) \leq val\ (u \otimes b) + \alpha$
shows $val\ a \leq val\ b + \alpha$
proof−
obtain *c* **where** *c-def*: $c \in carrier\ Q_p \wedge val\ c = \alpha$
by (*metis Qp.zero-closed eint.exhaust local.val-zero p-intpow-closed*(*1*) *val-p-int-pow*)

have *0*: $val\ (u \otimes a) \leq val\ (u \otimes (b \otimes c))$
using *assms val-mult*[*of u* ⊗ *b c*] *Qp.m-assoc*[*of u b c*] *c-def Qp.Units-closed Qp.cring-simprules*(*5*)
by *metis*
have *1*: $val\ a \leq val\ (b \otimes c)$
apply(*rule times-cfs-val-mono*[*of u*])
using *assms* **apply** *blast* **using** *assms* **apply** *blast*
apply(*rule Qp.ring-simprules*) **using** *assms* **apply** *blast* **using** *c-def* **apply** *blast*
by(*rule 0*)
show *?thesis*
using *1 val-mult*[*of b c*] *assms c-def* **by** *smt*
qed

lemma *times-cfs-val-mono″*:
assumes $u \in Units\ Q_p$
assumes $a \in carrier\ Q_p$
assumes $b \in carrier\ Q_p$
assumes $val\ a \leq val\ b + \alpha$
shows $val\ (u \otimes a) \leq val\ (u \otimes b) + \alpha$
apply(*rule times-cfs-val-mono′*[*of inv u u* ⊗ *a u* ⊗ *b* α])
using *assms* **apply** *blast*
apply(*rule Qp.ring-simprules*)

using *assms* **apply** *blast* **using** *assms* **apply** *blast*
apply(*rule Qp.ring-simprules*)
using *assms* **apply** *blast* **using** *assms* **apply** *blast*
using *m-assoc assms*
by (*metis Qp.Units-closed Qp.cring-simprules*(*5*) *Qp.inv-cancelR*(*1*))

lemma *val-ineq-cancel-leq*:
assumes $a \in nonzero\ Q_p$
assumes $b \in carrier\ Q_p$
assumes $c \in carrier\ Q_p$
assumes $val\ (a \otimes b) \leq val\ (a \otimes c)$
shows $val\ b \leq val\ c$
using *Units-eq-nonzero assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *times-cfs-val-mono*
by *blast*

lemma *val-ineq-cancel-leq'*:
assumes $a \in nonzero\ Q_p$
assumes $b \in carrier\ Q_p$
assumes $c \in carrier\ Q_p$
assumes $val\ b \leq val\ c$
shows $val\ (a \otimes b) \leq val\ (a \otimes c)$
apply(*rule val-ineq-cancel-leq*[*of inv a a* $\otimes$ *b a* $\otimes$ *c*])
using *assms*(*1*) *nonzero-inverse-Qp* **apply** *blast*
using *Qp.nonzero-closed assms*(*1*) *assms*(*2*) **apply** *blast*
using *Qp.nonzero-closed assms assms*(*2*) **apply** *blast*
by (*metis Qp.inv-cancelR*(*1*) *Qp.m-closed Qp.nonzero-closed Units-eq-nonzero assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*))

lemma *val-ineq-cancel-le*:
assumes $a \in nonzero\ Q_p$
assumes $b \in carrier\ Q_p$
assumes $c \in carrier\ Q_p$
assumes $val\ (a \otimes b) < val\ (a \otimes c)$
shows $val\ b < val\ c$
using *Qp.nonzero-closed assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *val-mult* **by** *auto*

lemma *val-ineq-cancel-le'*:
assumes $a \in nonzero\ Q_p$
assumes $b \in carrier\ Q_p$
assumes $c \in carrier\ Q_p$
assumes $val\ b < val\ c$
shows $val\ (a \otimes b) < val\ (a \otimes c)$
apply(*rule val-ineq-cancel-le*[*of inv a a* $\otimes$ *b a* $\otimes$ *c*])
using *assms*(*1*) *nonzero-inverse-Qp* **apply** *blast*
using *Qp.nonzero-closed assms*(*1*) *assms*(*2*) **apply** *blast*
using *Qp.nonzero-closed assms assms*(*2*) **apply** *blast*
by (*metis Qp.inv-cancelR*(*1*) *Qp.m-closed Qp.nonzero-closed Units-eq-nonzero assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*))

lemma *finite-val-imp-nonzero*:
assumes $a \in$ *carrier* Q_p
assumes *val* $a \neq \infty$
shows $a \in$ *nonzero* Q_p
using *assms* **unfolding** *val-def nonzero-def*
by (*metis* (*mono-tags*, *lifting*) *mem-Collect-eq*)

lemma *val-ineq-cancel-leq''*:
assumes $a \in$ *nonzero* Q_p
assumes $b \in$ *carrier* Q_p
assumes $c \in$ *carrier* Q_p
assumes *val* $b \leq$ *val* $c +$ *eint* N
shows *val* $(a \otimes b) \leq$ *val* $(a \otimes c) +$ *eint* N
proof−
obtain d **where** *d-def*: $d = \mathfrak{p}[\hat{}]N \otimes c$
by *blast*
show *?thesis*
proof(*cases* $c = \mathbf{0}$)
case *True*
then show *?thesis* **unfolding** *True*
using *True Units-eq-nonzero assms*(*1*) *assms*(*2*) *assms*(*4*) *times-cfs-val-mono''*
by *blast*
next
case *False*
have *F0*: $c \in$ *nonzero* Q_p
using *False assms Qp.not-nonzero-memE* **by** *blast*
have *F1*: *val* $(a \otimes c) < \infty$
using *F0 assms*
by (*metis* (*no-types*, *lifting*) *Qp.integral Qp.nonzero-closed Qp.nonzero-memE*(*2*)
eint.distinct(*1*) *eint-ord-simps*(*4*) *val-def*)
have *F2*: $b \in$ *nonzero* Q_p
using *F0 assms*
by (*metis False Groups.add-ac*(*2*) *add-cancel-eint-geq finite-val-imp-nonzero*
local.val-zero plus-eint-simps(*2*) *val-ineq*)
have *F3*: *ord* $b \leq$ *ord* $c + N$
using *assms F0*
by (*metis F2 eint-ord-simps*(*1*) *plus-eint-simps*(*1*) *val-ord*)
have *F4*: *ord* $a +$ *ord* $b \leq$ *ord* $a +$ *ord* $c+ N$
using *assms F0 F1 F2 ord-mult F3* **by** *presburger*
have *F5*: *ord* $(a \otimes b) \leq$ *ord* $(a \otimes c) + N$
using *F4 F2 F0 assms ord-mult* **by** *presburger*
have *F6*: $a \otimes b \in$ *nonzero* Q_p
by (*metis F2 Qp.integral Qp.nonzero-memE*(*1*) *Qp.nonzero-mult-closed Qp.not-nonzero-memE*
assms(*1*))
have *F7*: $a \otimes c \in$ *nonzero* Q_p
by (*metis False Qp.integral Qp.nonzero-closed Qp.nonzero-memI Qp.nonzero-mult-closed*
assms(*1*) *assms*(*3*))
show *val* $(a \otimes b) \leq$ *val* $(a \otimes c) +$ *eint* N

using *val-ord*[*of* $a \otimes b$] *val-ord*[*of* $a \otimes c$] *F7 F7 F4*
Units-eq-nonzero assms(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *times-cfs-val-mono''*
by *blast*
qed
qed

8.3.3 The Ultrametric Inequality on $\mathbb{Q}_p$

lemma *ord-ultrametric*:
assumes $x \in$ *nonzero* Q_p
assumes $y \in$ *nonzero* Q_p
assumes $x \oplus y \in$ *nonzero* Q_p
shows *ord* $(x \oplus y) \geq$ *min* (*ord x*) (*ord y*)
proof−
have *0*:$x \in$ *carrier* Q_p **using** *assms* **by**(*simp add*:*nonzero-def*)
have *1*:$y \in$*carrier* Q_p **using** *assms* **by**(*simp add*:*nonzero-def*)
obtain *a b c* **where**
A: $a \in$ *carrier* Z_p **and**
B: $b \in$ *carrier* Z_p **and**
C: $c \in$ *nonzero* Z_p **and**
Fx: $x =$ *frac a c* **and**
Fy: $y =$ *frac b c*
using *0 1 get-common-denominator* **by** *blast*
have *An*: $a \in$ *nonzero* Z_p
using *A C Fx assms*(*1*) *nonzero-fraction-imp-nonzero-numer*
Qp.nonzero-memE(*2*) **by** *auto*
have *Bn*: $b \in$ *nonzero* Z_p
using *B C Fy assms*(*2*) *nonzero-fraction-imp-nonzero-numer Qp.nonzero-memE*(*2*)
by *auto*
have *Fxy*: $x \oplus y =$ *frac* $(a \oplus_{Z_p} b)$ *c* **using** *0 1*
by (*simp add*: *A B C Fx Fy frac-add-common-denom*)
have *ABn*: $a \oplus_{Z_p} b \in$ *nonzero* Z_p
proof−
have $a \oplus_{Z_p} b \in$ *carrier* Z_p
using *A B* Z_p*-def padic-add-closed prime* **by** *blast*
then show *?thesis*
using *Fxy C assms*(*3*) *Qp.nonzero-memE*(*2*)
nonzero-fraction-imp-nonzero-numer **by** *blast*
qed
have *Ordx*: *ord x* = *ord-Zp a* − *ord-Zp c*
using *Fx An C ord-of-frac* **by** *metis*
have *Ordy*: *ord y* = *ord-Zp b* − *ord-Zp c*
using *Fy Bn C ord-of-frac* **by** *metis*
have *Ordxy*: *ord* $(x \oplus y) =$ *ord-Zp* $(a \oplus_{Z_p} b)$ − *ord-Zp c*
using *Fxy ABn C ord-of-frac* **by** *metis*
then show *?thesis*
using *Ordx Ordy Ordxy ord-Zp-ultrametric*[*of a b*] *ABn An Bn*
by *linarith*
qed

lemma *ord-ultrametric'*:
assumes $x \in$ *nonzero* Q_p
assumes $y \in$ *nonzero* Q_p
assumes $x \ominus_{Q_p} y \in$ *nonzero* Q_p
shows *ord* $(x \ominus_{Q_p} y) \geq$ *min* (*ord* x) (*ord* y)
proof−
have *ord* $y =$ *ord* $(\ominus y)$
using *assms*(*2*) *ord-minus* **by** *blast*
then show *?thesis*
using *assms ord-ultrametric*[*of* $x \ominus y$]
unfolding *a-minus-def*
using *Qp.add.inv-closed Qp.add.inv-eq-1-iff Qp.nonzero-closed Qp.nonzero-memE*(*2*) *Qp.nonzero-memI*
by *metis*
qed

lemma *val-ultrametric0*:
assumes $x \in$ *nonzero* Q_p
assumes $y \in$ *nonzero* Q_p
assumes $x \oplus y \in$ *nonzero* Q_p
shows *min* (*val* x) (*val* y) $\leq$ *val* $(x \oplus y)$
proof−
have *0*: *val* $(x \oplus y) =$ *ord* $(x \oplus y)$
using *assms*(*3*) *nonzero-def val-def*[*of* $(x \oplus y)$] **by** *fastforce*
have *1*: *val* $x =$ *ord* x
using *assms*(*1*) *nonzero-def val-def*
by (*simp add*: *nonzero-def*)
have *2*: *val* $y =$ *ord* y
using *assms*(*2*) *nonzero-def val-def val-ord* **by** *auto*
have *3*: *ord* $(x \oplus y) \geq$ *min* (*ord* x) (*ord* y)
by (*simp add*: *assms*(*1*) *assms*(*2*) *assms*(*3*) *ord-ultrametric*)
then show *?thesis*
by (*simp add*: *0 1 2*)
qed

lemma *val-ultrametric*:
assumes $x \in$ *carrier* Q_p
assumes $y \in$ *carrier* Q_p
shows *min* (*val* x) (*val* y) $\leq$ *val* $(x \oplus y)$
apply(*cases* $x = \mathbf{0} \vee y = \mathbf{0}$)
using *assms*(*1*) *assms*(*2*) **apply** *auto*[*1*]
by (*metis Qp.add.m-closed assms*(*1*) *assms*(*2*) *eint-ord-code*(*3*) *local.val-zero not-nonzero-Qp val-ultrametric0*)

lemma *val-ultrametric'*:
assumes $x \in$ *carrier* Q_p
assumes $y \in$ *carrier* Q_p
shows *min* (*val* x) (*val* y) $\leq$ *val* $(x \ominus y)$

using *val-ultrametric*[*of* $x \ominus y$]
val-minus[*of* y]
assms
by (*metis Qp.domain-axioms a-minus-def cring.cring-simprules*(3) *domain.axioms*(1))

lemma *diff-ord-nonzero*:
assumes $x \in$ *nonzero* Q_p
assumes $y \in$ *nonzero* Q_p
assumes *ord* $x \neq$ *ord* y
shows $x \oplus y \in$ *nonzero* Q_p
proof(*rule ccontr*)
assume $x \oplus y \notin$ *nonzero* Q_p
then have $x \oplus y = \mathbf{0}$
using *Qp.add.m-closed Qp.nonzero-closed Qp.nonzero-memI assms*(1) *assms*(2)
by *blast*
then have $x = \ominus y$
using *Qp.minus-equality Qp.nonzero-closed assms*(1) *assms*(2) **by** *blast*
then have *ord* $x =$ *ord* y
using *assms*(2) *ord-minus* **by** *presburger*
then show *False*
using *assms*(3) **by** *blast*
qed

lemma *ord-ultrametric-noteq*:
assumes $x \in$ *nonzero* Q_p
assumes $y \in$ *nonzero* Q_p
assumes *ord* $x >$ *ord* y
shows *ord* $(x \oplus y) = ($*ord* $y)$
proof(*rule ccontr*)
assume *ord* $(x \oplus y) \neq$ *ord* y
have *00*:$x \oplus y \in$ *nonzero* Q_p
proof(*rule ccontr*)
assume $x \oplus y \notin$ *nonzero* Q_p
then have $x \oplus y = \mathbf{0}$
using *Qp.add.m-closed Qp.nonzero-closed Qp.nonzero-memI assms*(1) *assms*(2)
by *blast*
then have $x = \ominus y$
by (*smt* ‹$x \oplus y \notin$ *nonzero* Q_p› *assms*(1) *assms*(2) *assms*(3) *diff-ord-nonzero*)

then have *ord* $x =$ *ord* y
using *assms*(2) *ord-minus* **by** *presburger*
then show *False*
using *assms*(3) **by** *linarith*
qed
then have *0*: *ord* $(x \oplus y) >$ *ord* y
using *ord-ultrametric*[*of* x y] ‹*ord* $(x \oplus y) \neq$ *ord* y› *assms*(1) *assms*(2) *assms*(3)

by *linarith*
have *1*: $((y \oplus x) \ominus x) = y$

using *00 Qp.add.inv-solve-right' Qp.add.m-comm Qp.minus-eq Qp.nonzero-closed assms(1) assms(2)* **by** *presburger*
have *2*: *ord* $((y \oplus x) \ominus x) \geq min\ (ord\ (y \oplus x))\ (ord\ x)$
using *1 assms ord-ultrametric'[of* $(y \oplus x)$ *x] diff-ord-nonzero* **by** *auto*
have *3*: *ord* $y \geq min\ (ord\ x)\ (ord\ (x \oplus y))$
using *2 1* Q_p*-def Qp.domain-axioms* Z_p*-def assms(1) assms(2) cring.cring-simprules(10)*

domain.axioms(1) Qp.nonzero-closed **by** *fastforce*
show *False*
using *3 0 assms*
by *linarith*
qed

lemma *ord-ultrametric-noteq'*:
assumes $x \in nonzero\ Q_p$
assumes $y \in nonzero\ Q_p$
assumes *ord* $x >$ *ord* y
shows *ord* $(x \ominus y) = (ord\ y)$
using *assms ord-ultrametric-noteq[of* $x \ominus y$]
by (*metis Qp.add.inv-closed Qp.minus-eq Qp.nonzero-closed Qp.nonzero-memI Qp.r-neg Qp.r-zero ord-minus*)

lemma *ord-ultrametric-noteq''*:
assumes $x \in nonzero\ Q_p$
assumes $y \in nonzero\ Q_p$
assumes *ord* $y >$ *ord* x
shows *ord* $(x \ominus y) = (ord\ x)$
using *assms ord-ultrametric-noteq'[of y x]*
by (*metis Qp.minus-a-inv Qp.nonzero-closed Qp.not-eq-diff-nonzero ord-minus*)

lemma *val-ultrametric-noteq*:
assumes $x \in carrier\ Q_p$
assumes $y \in carrier\ Q_p$
assumes *val* $x >$ *val* y
shows *val* $(x \oplus y) =$ *val* y
apply(*cases* $x = \mathbf{0}$)
apply (*simp add*: *assms(2)*)
using *assms* **unfolding** *val-def*
by (*smt Qp.not-nonzero-memI diff-ord-nonzero eint-ord-simps(2) not-nonzero-Qp ord-ultrametric-noteq val-def val-nonzero*)

lemma *val-ultrametric-noteq'*:
assumes $x \in carrier\ Q_p$
assumes $y \in carrier\ Q_p$
assumes *val* $x >$ *val* y
shows *val* $(x \ominus y) =$ *val* y
using *assms val-ultrametric-noteq[of* $x \ominus y$]
by (*metis Qp.domain-axioms a-minus-def cring.cring-simprules(3) domain.axioms(1) val-minus*)

lemma *ultrametric-equal-eq*:
assumes $x \in carrier\ Q_p$
assumes $y \in carrier\ Q_p$
assumes $val\ (y \ominus x) > val\ x$
shows $val\ x = val\ y$
using *assms*
by (*metis* (*no-types*, *lifting*) *Qp.add.inv-closed Qp.add.m-assoc Qp.l-neg Qp.minus-closed Qp.minus-eq Qp.r-zero val-ultrametric-noteq*)

lemma *ultrametric-equal-eq'*:
assumes $x \in carrier\ Q_p$
assumes $y \in carrier\ Q_p$
assumes $val\ (x \ominus y) > val\ x$
shows $val\ x = val\ y$
using *assms ultrametric-equal-eq*[*of x y*]
by (*metis Qp.minus-a-inv Qp.minus-closed val-minus*)

lemma *val-ultrametric-noteq''*:
assumes $x \in carrier\ Q_p$
assumes $y \in carrier\ Q_p$
assumes $val\ x > val\ y$
shows $val\ (y \ominus x) = val\ y$
by (*metis Qp.minus-a-inv Qp.minus-closed assms*(*1*) *assms*(*2*) *assms*(*3*) *val-minus val-ultrametric-noteq'*)

Ultrametric over finite sums:

lemma *Min-mono*:
assumes *finite A*
assumes $A \neq \{\}$
assumes $\bigwedge a.\ a \in A \Longrightarrow\ f\ a \leq a$
shows $Min\ (f`A) \leq Min\ A$
by (*meson Min-in Min-le-iff assms*(*1*) *assms*(*2*) *assms*(*3*) *finite-imageI image-eqI image-is-empty*)

lemma *Min-mono'*:
assumes *finite A*
assumes $\bigwedge (a::'a).\ a \in A \Longrightarrow (f::'a \Rightarrow eint)\ \ a \leq\ \ g\ a$
shows $Min\ (f`A) \leq Min\ (g\ `A)$
proof−
have $(\forall\, a \in\ \ A.\ f\ a \leq g\ a) \longrightarrow Min\ (f`A) \leq Min\ (g`\ A)$
apply(*rule finite.induct*[*of A*])
apply (*simp add*: *assms*(*1*))
apply *simp*
proof fix *A a* **assume** *F*: *finite A* $(\forall\, a{\in}A.\ f\ a \leq g\ a) \longrightarrow Min\ (f\ `\ A) \leq Min\ (g\ `\ A)$
$\forall\, a{\in}insert\ a\ A.\ f\ a \leq g\ a$
show $Min\ (f\ `\ insert\ a\ A) \leq Min\ (g\ `\ insert\ a\ A)$
proof−

obtain k **where** *k-def*: $k \in$ *insert* $a\ A \wedge g\ k =$ *Min* (*g* ‘ *insert* $a\ A$)
using *assms*
by (*metis* (*mono-tags*, *opaque-lifting*) *F*(*1*) *Min-eq-iff finite.insertI finite-imageI image-iff image-is-empty insert-not-empty*)
then have *0*: $f\ k \leq g\ k$
using *F*(*3*) **by** *blast*
thus *?thesis* **using** *k-def*
by (*metis F*(*1*) *Min-le dual-order.trans finite.insertI finite-imageI image-eqI*)
qed
qed
thus *?thesis* **using** *assms* **by** *blast*
qed

lemma *eint-ord-trans*:
assumes (*a*::*eint*) $\leq b$
assumes $b \leq c$
shows $a \leq c$
using *assms* **by** *auto*

lemma *eint-Min-geq*:
assumes *finite* (*A*::*eint set*)
assumes $\bigwedge x.\ x \in A \Longrightarrow x \geq c$
assumes $A \neq \{\}$
shows *Min* $A \geq c$
using *Min-in*[*of A*] *assms*(*2*)[*of Min A*] *assms* **by** *blast*

lemma *eint-Min-gr*:
assumes *finite* (*A*::*eint set*)
assumes $\bigwedge x.\ x \in A \Longrightarrow x > c$
assumes $A \neq \{\}$
shows *Min* $A > c$
using *Min-in*[*of A*] *assms*(*2*)[*of Min A*] *assms* **by** *blast*

lemma *finsum-val-ultrametric*:
assumes $g \in A \to$ *carrier* Q_p
assumes *finite* A
assumes $A \neq \{\}$
shows *val* (*finsum* $Q_p\ g\ A$) $\geq$ *Min* (*val* ‘ (*g*‘*A*))
proof−
have $A \neq \{\} \wedge g \in A \to$ *carrier* $Q_p \longrightarrow$ *val* (*finsum* $Q_p\ g\ A$) $\geq$ *Min* (*val* ‘ (*g*‘*A*))

apply(*rule finite.induct*[*of A*])
apply (*simp add*: *assms*(*2*))
apply *blast*
proof fix $A\ a$ **assume** *A*: *finite* $A\ A \neq \{\} \wedge g \in A \to$ *carrier* $Q_p \longrightarrow$ *val* (*finsum* $Q_p\ g\ A$) $\geq$ *Min* (*val* ‘ *g* ‘ *A*)
insert $a\ A \neq \{\} \wedge g \in$ *insert* $a\ A \to$ *carrier* Q_p
show *val* (*finsum* $Q_p\ g$ (*insert* $a\ A$)) $\geq$ *Min* (*val* ‘ *g* ‘ *insert* $a\ A$)
apply(*cases* $a \in A$)

apply (*metis A(2) A(3) insert-absorb*)
proof(*cases A = {}*)
have *g-closed*: $g \in A \to$ *carrier* Q_p
using *A(3)* **by** *blast*
have *g-closed'*: $g \in$ *insert a A* $\to$ *carrier* Q_p
using *A(3)* **by** *linarith*
then have *ga*: *g a* $\in$ *carrier* Q_p
by *blast*
assume *a-notin*: $a \notin A$
case *True*
have *g a* $\in$ *carrier* Q_p **using** *A(3)*
by *blast*
then have *0*: (*finsum* Q_p *g* (*insert a A*)) = *g a*
using *A True abelian-monoid.finsum-insert*[*of* Q_p *A a g*] *abelian-monoid.finsum-empty*[*of* Q_p *g*]
Qp.abelian-monoid-axioms Qp.add.l-cancel-one Qp.zero-closed a-notin g-closed
by *metis*
have *1*: *Min* (*val* ‘ *g* ‘ *insert a A*) = *val* (*g a*)
by (*metis A(1) True Min-in finite.insertI finite-imageI image-empty image-insert insert-not-empty singletonD*)
show *?thesis*
using *0 1*
by *simp*
next
case *False*
assume *a-notin*: $a \notin A$
have *g-closed*: $g \in A \to$ *carrier* Q_p
using *A(3)* **by** *blast*
have *g-closed'*: $g \in$ *insert a A* $\to$ *carrier* Q_p
using *A(3)* **by** *linarith*
then have *ga*: *g a* $\in$ *carrier* Q_p
by *blast*
have *0*: *finsum* Q_p *g* (*insert a A*) = *g a* $\oplus$ *finsum* Q_p *g A*
by (*simp add*: *A(1) a-notin g-closed ga*)
have *1*: *min* (*val* (*g a*)) (*Min* (*val* ‘ *g* ‘ *A*)) = *Min* (*insert* (*val* (*g a*)) (*val* ‘ *g* ‘ *A*))
proof−
have *10*: *finite* (*val* ‘ *g* ‘ *A*) **using** *A*
by *blast*
then have *11*: *val* ‘ *g* ‘ *A* $\neq$ {}
using *False*
by *blast*
show *?thesis*
using *10 11 Min-insert*
by *simp*
qed
have *2*: *val* (*g a* $\oplus$ *finsum* Q_p *g A*) $\geq$ *min* (*val* (*g a*)) (*val* (*finsum* Q_p *g A*))
using *val-ultrametric*[*of g a finsum* Q_p *g A*] *abelian-monoid.finsum-closed*[*of*

```
Q_p g A]
          g-closed ga  Qp.abelian-monoid-axioms by blast
    have 3: val (finsum Q_p g A) ≥ Min (val ‘ g ‘ A)
      using A False
      by blast
    have 4: val (finsum Q_p g (insert a A)) ≥ min (val (g a)) (val (finsum Q_p
g A))
      using 2 0
      by presburger
    have 5: val (finsum Q_p g A) ≥ Min (val ‘ g ‘ A)
      using False 3 by blast
    show val (finsum Q_p g (insert a A)) ≥ Min (val ‘ g ‘ insert a A)
      using 5 4 1
      by (smt image-insert min.cobounded1 min-def order-trans)
  qed
 qed
 then show ?thesis
  using assms
  by blast
qed

lemma (in padic-fields) finsum-val-ultrametric':
 assumes g ∈ A → carrier Q_p
 assumes finite A
 assumes ⋀i. i ∈ A ⟹ val (g i) ≥ c
 shows val (finsum Q_p g A) ≥ c
proof(cases A = {})
 case True
 then show ?thesis using assms
  unfolding True Qp.finsum-empty
  using eint-ord-code(3) local.val-zero by presburger
next
 case False
 show ?thesis
 proof(rule eint-ord-trans[of - Min (val ‘ g ‘ A)])
  have 0: ⋀s. s ∈ val ‘ g ‘ A ⟹ s ≥ c
  proof− fix s assume A: s ∈ val ‘ g ‘ A
    then obtain a where a-def: a ∈ A ∧ s = val (g a)
      by blast
    have s-eq: s = val (g a)
      using a-def by blast
    show c ≤ s
      using assms(3)[of a] A a-def unfolding s-eq by blast
  qed
  show c ≤ Min (val ‘ g ‘ A)
   apply(rule eint-Min-geq)
   using assms apply blast using assms apply blast
   using False by blast
  show Min (val ‘ g ‘ A) ≤ val (finsum Q_p g A)
```

```
    by(rule finsum-val-ultrametric[of g A], rule assms, rule assms, rule False)
  qed
qed

lemma (in padic-fields) finsum-val-ultrametric'':
  assumes g ∈ A → carrier Q_p
  assumes finite A
  assumes ⋀i. i ∈ A ⟹ val (g i) > c
  assumes c < ∞
  shows  val (finsum Q_p g A) > c
proof(cases A = {})
  case True
  then show ?thesis using assms
    unfolding True Qp.finsum-empty  val-zero
    using eint-ord-code(3) local.val-zero  by blast
next
  case False
  show ?thesis
  proof(rule less-le-trans[of - Min (val ‘ g ‘ A)])
    have 0: ⋀s. s ∈ val ‘ g ‘ A ⟹ s > c
    proof− fix s assume A: s ∈ val ‘ g ‘ A
      then obtain a where a-def: a ∈ A ∧ s = val (g a)
        by blast
      have s-eq: s = val (g a)
        using a-def by blast
      show c < s
        using assms(3)[of a] A a-def unfolding s-eq  by blast
    qed
    show c < Min (val ‘ g ‘ A)
      apply(rule eint-Min-gr)
      using assms apply blast using assms
      using 0 apply blast
      using False by blast
    show Min (val ‘ g ‘ A) ≤ val (finsum Q_p g A)
      by(rule finsum-val-ultrametric[of g A], rule assms, rule assms, rule False)
  qed
qed

lemma Qp-diff-diff:
  assumes x ∈ carrier Q_p
  assumes c ∈ carrier Q_p
  assumes d ∈ carrier Q_p
  shows (x ⊖ c) ⊖ (d ⊖ c) = x ⊖ d
 by (smt Qp.domain-axioms a-minus-def assms(1) assms(2) assms(3) cring.cring-simprules(10)

     cring.cring-simprules(19) cring.cring-simprules(3) cring.cring-simprules(4)
     cring.cring-simprules(7) domain.axioms(1))
```

This variant of the ultrametric identity formalizes the common saying that

"all triangles in $\mathbb{Q}_p$ are isosceles":

lemma *Qp-isosceles*:
assumes $x \in$ *carrier* Q_p
assumes $c \in$ *carrier* Q_p
assumes $d \in$ *carrier* Q_p
assumes *val* $(x \ominus c) \geq v$
assumes *val* $(d \ominus c) \geq v$
shows *val* $(x \ominus d) \geq v$
proof−
have *val* $(x \ominus d) \geq$ *min* (*val* $(x \ominus c)$) (*val* $(d \ominus c)$)
using *assms Qp-diff-diff*[*of x c d*]
by (*metis Qp.domain-axioms cring.cring-simprules*(*4*) *domain.axioms*(*1*) *val-ultrametric'*)
then show *?thesis* **using** *assms*
by (*meson dual-order.trans min-le-iff-disj*)
qed

More variants on the ultrametric inequality

lemma *MinE*:
assumes *finite* (*A*::*eint set*)
assumes $a =$ *Min A*
assumes $b \in A$
shows $a \leq b$
using *assms* **by** (*simp add*: *assms*(*3*))

lemma *MinE'*:
assumes *finite* (*A*::*eint set*)
assumes $a =$ *Min A*
assumes $b \in A - \{a\}$
shows $a < b$
proof−
have *0*: $a \leq b$
using *assms MinE* **by** *blast*
have *1*: $a \neq b$ **using** *assms* **by** *blast*
show *?thesis* **using** *0 1*
by *simp*
qed

lemma *MinE''*:
assumes *finite A*
assumes $f \in A \to$ (*UNIV* :: *eint set*)
assumes $a =$ *Min* (f ‘ A)
assumes $b \in A$
shows $a \leq f\ b$
apply(*rule MinE*[*of f' A*]) **using** *assms* **apply** *blast* **using** *assms* **apply** *blast*
using *assms* **by** *blast*

lemma *finsum-val-ultrametric-diff*:
assumes $g \in A \to$ *carrier* Q_p
assumes *finite A*

assumes $A \neq \{\}$
assumes $\bigwedge a\ b.\ a \in A \Longrightarrow b \in A \Longrightarrow a \neq b \Longrightarrow val\ (g\ a) \neq val\ (g\ b)$
shows $val\ (finsum\ Q_p\ g\ A) = Min\ (val\ `\ g`A)$
proof−
have $Min\ (val\ `\ g\ `\ A) \in val\ `\ g\ `\ A$
using *Min-in*[*of* $val\ `\ g\ `\ A$] *assms* **by** *blast*
then obtain a **where** *a-def*: $a \in A \wedge Min\ (val\ `\ g`A) = val\ (g\ a)$
by *blast*
have *0*: $\bigwedge b.\ b \in A \Longrightarrow b \neq a \Longrightarrow val\ (g\ b) > \ val\ (g\ a)$
apply(*rule MinE'*[*of* $val\ `\ g\ `\ A$]) **using** *assms* **apply** *blast*
using *a-def assms* **apply** *presburger*
using *assms*(*4*)[*of a*] *a-def*
by (*metis image-eqI insert-Diff insert-iff*)
have *1*: $g\ a \oplus finsum\ Q_p\ g\ (A - \{a\}) = finsum\ Q_p\ g\ (insert\ a\ (A - \{a\}))$
using *assms Qp.finsum-insert*[*of* $A - \{a\}$ *a g*] *a-def finite-subset*[*of* $A - \{a\}$ *A*] **by** *blast*
have *2*: $finsum\ Q_p\ g\ A = g\ a \oplus finsum\ Q_p\ g\ (A - \{a\})$
unfolding *1* **using** *a-def*
by (*metis insert-Diff-single insert-absorb*)
have *3*: $g\ a \in carrier\ Q_p$
using *a-def assms* **by** *blast*
show $val\ (finsum\ Q_p\ g\ A) = Min\ (val\ `\ g`A)$
proof(*cases* $A = \{a\}$)
case *True*
show *?thesis* **using** *3 Qp.finsum-empty*[*of g*] *a-def* **unfolding** *2* **unfolding** *True*
by *auto*
next
case *False*
then have *F0*: $A - \{a\} \neq \{\}$
using *assms* **by** *blast*
have *F1*: $finsum\ Q_p\ g\ (A - \{a\}) \in carrier\ Q_p$
apply(*rule Qp.finsum-closed*)
using *assms* **by** *blast*
have *F2*: $val\ (finsum\ Q_p\ g\ (A - \{a\})) \geq Min\ (val\ `\ g`\ (A- \{a\}))$
apply(*rule finsum-val-ultrametric*)
using *assms* **apply** *blast* **using** *assms* **apply** *blast* **using** *False a-def* **by** *blast*
have *F3*: $Min\ (val\ `\ g`\ (A- \{a\})) \in (val\ `\ g`\ (A- \{a\}))$
apply(*rule Min-in*)
using *assms* **apply** *blast* **using** *False a-def* **by** *blast*
obtain b **where** *b-def*: $b \in A - \{a\} \wedge Min\ (val\ `\ g`\ (A- \{a\})) = val\ (g\ b)$
using *F3* **by** *blast*
have *F4*: $Min\ (val\ `\ g`\ (A- \{a\})) > val\ (g\ a)$
using *a-def assms b-def* **by** (*metis 0 Diff-iff singletonI*)
have *F5*: $val\ (finsum\ Q_p\ g\ (A - \{a\})) > val\ (g\ a)$
using *F4 F2 less-le-trans* **by** *blast*
then show *?thesis* **using** *2 F1 3*
by (*metis Qp.a-ac*(*2*) *a-def val-ultrametric-noteq*)

qed
qed

lemma *finsum-val-ultrametric-diff'*:
assumes $g \in A \to$ *carrier* Q_p
assumes *finite* A
assumes $A \neq \{\}$
assumes $\bigwedge a\ b.\ a \in A \Longrightarrow b \in A \Longrightarrow a \neq b \Longrightarrow$ *val* $(g\ a) \neq$ *val* $(g\ b)$
shows *val* (*finsum* $Q_p\ g\ A$) = (*MIN* $a \in A$. (*val* $(g\ a)$))
proof−
have *0*: $(\lambda\ a.$ *val* $(g\ a))$ ' $A =$ (*val* ' g'A)
by *blast*
show *?thesis* **using** *assms finsum-val-ultrametric-diff*[*of g A*] **unfolding** *0* **by** *blast*
qed

8.4 Constructing the Angular Component Maps on $\mathbb{Q}_p$

8.4.1 Unreduced Angular Component Map

While one can compute the residue of a p-adic integer mod p^n, this operation does not generalize to the p-adic field unless we restrict our attention to the valuation ring. However, we can still define the angular component maps on the field $\mathbb{Q}_p$, which allows us to take a sort of residue for any element $x \in \mathbb{Q}_p$. Given a nonzero element $x \in \mathbb{Q}_p^\times$, we can normalize it to obtain $p^{-ord(x)}x$ which has of valuation zero, and then computes its residue (viewed as an element of $\mathbb{Z}_p$). The resulting map agrees with the standard residue map on elements of $\mathbb{Q}_p$ of valuation zero, but not on terms of positive or negative valuation. For example, the element p^2 has an order 1 residue of 0, but its order 1 angular component is 1. In the formalism below, we will use the term "`angular_component`" to refer to the unreduced normalization map $x \mapsto p^{-ord(x)}x$, and use the notation "`ac n`" to refer to the angular component which has been reduced mod p^n. This is line with the terminology used in [1].

definition *angular-component* **where**
angular-component a = (*ac-Zp* (*numer* a)) $\otimes_{Z_p}$ (inv_{Z_p} *ac-Zp* (*denom* a))

lemma *ac-fract*:
assumes $c \in$ *carrier* Q_p
assumes $a \in$ *nonzero* Z_p
assumes $b \in$ *nonzero* Z_p
assumes $c =$ *frac* $a\ b$
shows *angular-component* c = (*ac-Zp* a)$\otimes_{Z_p}$ *inv* $_{Z_p}$(*ac-Zp* b)
proof−
have (*numer* c) $\otimes_{Z_p}$ b = (*denom* c) $\otimes_{Z_p}$ a
by (*simp add*: *assms*(*2*) *assms*(*3*) *assms*(*4*) *mult-comm numer-denom-swap*)
then have *ac-Zp* ((*numer* c) $\otimes_{Z_p}$ b) = *ac-Zp* ((*denom* c) $\otimes_{Z_p}$ a)

by *simp*
then have (*ac-Zp* (*numer c*)) $\otimes_{Z_p}$ (*ac-Zp b*) = (*ac-Zp* (*denom c*)) $\otimes_{Z_p}$ (*ac-Zp a*)
by (*metis* Q_p*-def Zp.nonzero-closed Zp.numer-denom-facts*(3) *ac-Zp-mult assms*(2) *assms*(3) *assms*(4) *local.frac-closed local.numer-denom-facts*(1) *nonzero-fraction-imp-nonzero-numer nonzero-numer-imp-nonzero-fraction numer-denom-frac*)
then have (*inv* $_{Z_p}$ (*ac-Zp* (*denom c*))) $\otimes_{Z_p}$ (*ac-Zp* (*numer c*)) $\otimes_{Z_p}$ (*ac-Zp b*) = (*ac-Zp a*)
using *ac-Zp-is-Unit*[*of ac-Zp* (*denom c*)] *Zp.domain-axioms inv-cancelR*(1) Q_p*-def Zp.Units-closed Zp.inv-cancelR*(1) *Zp.m-closed Zp.nonzero-closed Zp.nonzero-memE*(2) Z_p*-def ac-Zp-is-Unit assms*(1) *assms*(2) *mult-assoc padic-fields.numer-denom-facts*(2) *padic-fields-axioms* **by** *auto*
then have (*inv* $_{Z_p}$ (*ac-Zp* (*denom c*))) $\otimes_{Z_p}$ (*ac-Zp* (*numer c*)) = (*ac-Zp a*) $\otimes_{Z_p}$ *inv* $_{Z_p}$ (*ac-Zp b*)
using *ac-Zp-is-Unit*[*of ac-Zp b*] *Zp.domain-axioms inv-cancelL*(2)
by (*smt* Q_p*-def Zp.Units-inv-closed Zp.Units-r-inv Zp.nonzero-closed Zp.nonzero-memE*(2) *Zp.r-one* Z_p*-def ac-Zp-is-Unit assms*(1) *assms*(3) *mult-assoc mult-comm padic-fields.numer-denom-facts*(2) *padic-fields-axioms*)
then show *?thesis*
by (*simp add*: *angular-component-def mult-comm*)
qed

lemma *angular-component-closed*:
assumes $a \in$ *nonzero* Q_p
shows *angular-component a* $\in$ *carrier* Z_p
using *ac-fract assms* Q_p*-def Qp.nonzero-closed Qp.nonzero-memE*(2) *Zp.Units-inv-closed Zp.m-closed*
Zp.nonzero-closed Zp.nonzero-memE(2) Z_p*-def ac-Zp-is-Unit angular-component-def local.numer-denom-facts*(3)
padic-fields.numer-denom-facts(1) *padic-fields.numer-denom-facts*(2) *padic-fields-axioms padic-integers.ac-Zp-in-Zp padic-integers-def prime* **by** *auto*

lemma *angular-component-unit*:
assumes $a \in$ *nonzero* Q_p
shows *angular-component a* $\in$ *Units* Z_p
using *ac-fract*[*of a numer a denom a*] Q_p*-def Qp.nonzero-closed*
Qp.nonzero-memE(2) *Zp.Units-inv-Units Zp.Units-m-closed*
Zp.nonzero-closed Zp.nonzero-memE(2) Z_p*-def ac-Zp-is-Unit*
angular-component-def assms local.numer-denom-facts(1)
local.numer-denom-facts(2) *padic-fields.numer-denom-facts*(3)
padic-fields-axioms **by** *auto*

lemma *angular-component-factors-x*:
assumes $x \in$ *nonzero* Q_p
shows $x = (\mathfrak{p}[\hat{}](ord\ x)) \otimes \iota$ (*angular-component x*)
proof−
have *0*: *angular-component x* = (*ac-Zp* (*numer x*)) $\otimes_{Z_p}$ (*inv*$_{Z_p}$ *ac-Zp* (*denom x*))
by (*simp add*: *angular-component-def*)

have *1*: $(numer\ x) = (\mathrm{p}[\hat{}]_{Z_p}(ord\text{-}Zp\ (numer\ x))) \otimes_{Z_p} (ac\text{-}Zp\ (numer\ x))$
proof−
have $numer\ x \in nonzero\ Z_p$
using *assms* **unfolding** Q_p*-def*
by (*simp add*: *numer-nonzero*)
then show *?thesis* **using** *ac-Zp-factors-x*[*of numer x*]
by (*simp add*: *ac-Zp-factors'*)
qed
have *2*: $(denom\ x) = (\mathrm{p}[\hat{}]_{Z_p}(ord\text{-}Zp\ (denom\ x))) \otimes_{Z_p} (ac\text{-}Zp\ (denom\ x))$
proof−
have $denom\ x \in nonzero\ Z_p$
using *nonzero-memE assms numer-denom-facts*(*2*)
by (*simp add*: *Qp.nonzero-closed*)
then show *?thesis*
using *ac-Zp-factors-x*[*of denom x*] *Zp.nonzero-closed Zp.nonzero-memE*(*2*)
by *auto*
qed
have *3*: $\iota\ (angular\text{-}component\ x) = frac\ (ac\text{-}Zp\ (numer\ x))\ (ac\text{-}Zp\ (denom\ x))$
by (*metis 0* Q_p*-def Qp.nonzero-closed Qp.nonzero-memE*(*2*) *Zp.nonzero-closed Zp.nonzero-memE*(*2*) *Zp.numer-denom-facts*(*2*) Z_p*-def* Z_p*-division-Qp-0 ac-Zp-is-Unit angular-component-closed assms local.inc-def padic-fields.numer-denom-facts*(*2*) *padic-fields.numer-denom-fact padic-fields-axioms*)
have *4*: $(\mathfrak{p}[\hat{}]((ord\ x))) \otimes \iota\ (angular\text{-}component\ x) = (\mathfrak{p}[\hat{}]((ord\text{-}Zp\ (numer\ x)) - (ord\text{-}Zp\ (denom\ x)))) \otimes frac\ (ac\text{-}Zp\ (numer\ x))\ (ac\text{-}Zp\ (denom\ x))$
using *3 ord-def* **by** *presburger*
have *5*: $(\mathfrak{p}[\hat{}]((ord\ x))) \otimes \iota\ (angular\text{-}component\ x) = frac\ (\mathrm{p}[\hat{}]_{Z_p}((ord\text{-}Zp\ (numer\ x))))\ (\mathrm{p}[\hat{}]_{Z_p}(ord\text{-}Zp\ (denom\ x))) \otimes frac\ (ac\text{-}Zp\ (numer\ x))\ (ac\text{-}Zp\ (denom\ x))$
proof−
have $(\mathfrak{p}[\hat{}]((ord\text{-}Zp\ (numer\ x)) - (ord\text{-}Zp\ (denom\ x))))$
$= frac\ (\mathrm{p}[\hat{}]_{Z_p}((ord\text{-}Zp\ (numer\ x))))\ (\mathrm{p}[\hat{}]_{Z_p}(ord\text{-}Zp\ (denom\ x)))$
by (*metis* Q_p*-def Qp.nonzero-closed* Z_p*-def assms domain.nonzero-memE*(*1*) *domain.nonzero-memE*(*2*) *numer-nonzero ord-pos p-pow-diff padic-fields.numer-denom-facts*(*2*) *padic-fields-axioms padic-int-is-domain prime*)
then show *?thesis* **using** *4* **by** *metis*
qed
have *6*: $(\mathfrak{p}[\hat{}]((ord\ x))) \otimes \iota\ (angular\text{-}component\ x) = frac\ (\mathrm{p}[\hat{}]_{Z_p}((ord\text{-}Zp\ (numer\ x))) \otimes_{Z_p} (ac\text{-}Zp\ (numer\ x)))\ (\mathrm{p}[\hat{}]_{Z_p}(ord\text{-}Zp\ (denom\ x)) \otimes_{Z_p} (ac\text{-}Zp\ (denom\ x)))$
using *5 frac-mult*[*of* $\mathrm{p}[\hat{}]_{Z_p}((ord\text{-}Zp\ (numer\ x)))\ (\mathrm{p}[\hat{}]_{Z_p}(ord\text{-}Zp\ (denom\ x)))\ (ac\text{-}Zp\ (numer\ x))\ (ac\text{-}Zp\ (denom\ x))$] *mult-comm*
by (*metis 2* Q_p*-def Qp.nonzero-closed Zp.integral-iff Zp.nonzero-closed Zp.nonzero-memE*(*2*) *ac-Zp-in-Zp assms local.numer-denom-facts*(*2*) *not-nonzero-Zp numer-nonzero ord-pos p-int-pow-nonzero*)
then show *?thesis*
using *1 2 nonzero-memE assms numer-denom-facts*(*5*) *Qp.nonzero-closed* **by** *auto*
qed

lemma *angular-component-mult*:
assumes $x \in nonzero\ Q_p$
assumes $y \in nonzero\ Q_p$
shows $angular\text{-}component\ (x \otimes y) = (angular\text{-}component\ x) \otimes_{Z_p} (angular\text{-}component\ y)$
proof−
obtain $a\ b$ **where** $a = numer\ x$ **and**
$b = denom\ x$ **and**
$a\text{-}nz$: $a \in nonzero\ Z_p$ **and**
$b\text{-}nz$: $b \in nonzero\ Z_p$ **and**
$x\text{-}frac$: $x = frac\ a\ b$
using *assms Qp.nonzero-memE*[*of x*]
by (*meson local.numer-denom-facts*(*1*) *local.numer-denom-facts*(*2*) *local.numer-denom-facts*(*3*) *local.numer-denom-facts*(*5*) *not-nonzero-Zp*)
obtain $c\ d$ **where** $c = numer\ y$ **and**
$d = denom\ y$ **and**
$c\text{-}nz$: $c \in nonzero\ Z_p$ **and**
$d\text{-}nz$: $d \in nonzero\ Z_p$ **and**
$y\text{-}frac$: $y = frac\ c\ d$
using *assms Qp.nonzero-memE*[*of y*]
by (*meson local.numer-denom-facts*(*1*) *local.numer-denom-facts*(*2*) *local.numer-denom-facts*(*3*) *local.numer-denom-facts*(*5*) *not-nonzero-Zp*)
have *0*: $(x \otimes y) = frac\ (a \otimes_{Z_p} c)\ (b \otimes_{Z_p} d)$
using *a-nz b-nz c-nz d-nz frac-mult x-frac y-frac*
by (*simp add*: *Zp.nonzero-closed*)
have *1*: $angular\text{-}component\ (x \otimes y) = (ac\text{-}Zp\ (a \otimes_{Z_p} c)) \otimes_{Z_p} inv_{Z_p}\ (ac\text{-}Zp\ (b \otimes_{Z_p} d))$
by (*metis* (*mono-tags*, *lifting*) *0 Qp.nonzero-mult-closed Zp.integral-iff Zp.nonzero-closed Zp.nonzero-mult-closed a-nz ac-fract assms*(*1*) *assms*(*2*) *b-nz c-nz d-nz not-nonzero-Zp*)

have *2*: $angular\text{-}component\ (x \otimes y) = (ac\text{-}Zp\ a) \otimes_{Z_p} (ac\text{-}Zp\ c) \otimes_{Z_p} inv_{Z_p}\ ((ac\text{-}Zp\ b) \otimes_{Z_p} (ac\text{-}Zp\ d))$
by (*simp add*: *1 a-nz ac-Zp-mult b-nz c-nz d-nz*)
have *3*: $angular\text{-}component\ (x \otimes y) = (ac\text{-}Zp\ a) \otimes_{Z_p} (ac\text{-}Zp\ c) \otimes_{Z_p} inv_{Z_p}\ (ac\text{-}Zp\ b) \otimes_{Z_p} inv_{Z_p}\ (ac\text{-}Zp\ d)$
using *2*
by (*simp add*: *Zp.inv-of-prod Zp.nonzero-closed Zp.nonzero-memE*(*2*) *ac-Zp-is-Unit b-nz d-nz mult-assoc*)

have *4*: $angular\text{-}component\ (x \otimes y) = (ac\text{-}Zp\ a) \otimes_{Z_p} inv_{Z_p}\ (ac\text{-}Zp\ b) \otimes_{Z_p} (ac\text{-}Zp\ c) \otimes_{Z_p} inv_{Z_p}\ (ac\text{-}Zp\ d)$
using *3 a-nz ac-Zp-in-Zp ac-Zp-is-Unit b-nz c-nz m-assoc mult-comm mult-assoc* **by** *auto*
have *5*: $angular\text{-}component\ (x \otimes y) = ((ac\text{-}Zp\ a) \otimes_{Z_p} inv_{Z_p}\ (ac\text{-}Zp\ b)) \otimes_{Z_p} ((ac\text{-}Zp\ c) \otimes_{Z_p} inv_{Z_p}\ (ac\text{-}Zp\ d))$
using *4* **by** (*simp add*: *mult-assoc*)
then show *?thesis*

by (*simp add*: Z_p*-def* ‹$a = numer\ x$› ‹$b = denom\ x$› ‹$c = numer\ y$› ‹$d = denom\ y$›
padic-fields.angular-component-def padic-fields-axioms)
qed

lemma *angular-component-inv*:
assumes $x \in nonzero\ Q_p$
shows *angular-component* $(inv_{Q_p}\ x) = inv_{Z_p}\ (angular\text{-}component\ x)$
by (*metis* Q_p*-def Qp.one-closed Zp.Units-r-inv Zp.inv-unique' Zp.nonzero-closed Zp.nonzero-one-closed Zp.zero-not-one* Z_p*-def ι-def ac-Zp-is-Unit angular-component-closed angular-component-mult assms frac-nonzero-inv*(*1*) *frac-nonzero-inv*(*2*) *inc-equation inc-of-one nonzero-inverse-Qp padic-fields.ac-fract padic-fields-axioms*)

lemma *angular-component-one*:
angular-component $\mathbf{1} = \mathbf{1}_{Z_p}$
using *ι-def ac-Zp-one ac-fract inc-equation inc-of-one Zp.nonzero-one-closed* **by** *fastforce*

lemma *angular-component-ord-zero*:
assumes $ord\ x = 0$
assumes $x \in nonzero\ Q_p$
shows $\iota\ (angular\text{-}component\ x) = x$
proof−
have *0*: $ord\ x \geq 0$
using *assms* **by** *auto*
have *1*: $x \in nonzero\ Q_p$
proof−
have $x \neq \mathbf{0}$
using *nonzero-def assms*(*2*) *Qp.nonzero-memE*(*2*) **by** *auto*
then show *?thesis*
by (*simp add*: *assms*(*2*))
qed
obtain *a* **where** *a-def*: $a \in nonzero\ Z_p \wedge \iota\ a = x$
using *0 1 assms Qp-inc-id*[*of x*]
by *metis*
then have $x = frac\ a\ \mathbf{1}_{Z_p}$
using *local.inc-def Zp.nonzero-closed* **by** *blast*
then have $angular\text{-}component\ x = ac\text{-}Zp\ a \otimes_{Z_p} inv_{Z_p}\ (ac\text{-}Zp\ \mathbf{1}_{Z_p})$
using *ac-fract*[*of x a*] *1 nonzero-memE* Q_p*-def Zp.nonzero-closed Zp.nonzero-one-closed* Z_p*-def a-def*
local.frac-closed padic-fields.ac-fract padic-fields-axioms **by** *auto*
then have $angular\text{-}component\ x = ac\text{-}Zp\ a \otimes_{Z_p} inv_{Z_p}\ \mathbf{1}_{Z_p}$
by (*simp add*: *ac-Zp-one*)
then have *0*: $angular\text{-}component\ x = ac\text{-}Zp\ a \otimes_{Z_p} \mathbf{1}_{Z_p}$
by *simp*
have $ac\text{-}Zp\ a \in nonzero\ Z_p$
using *Zp.Units-nonzero Zp.nonzero-closed Zp.not-nonzero-memI a-def ac-Zp-is-Unit*
by *auto*
thus *?thesis* **using** *0*

by (*metis Zp.l-one Zp.nonzero-closed Zp-defs(1) Zp-defs(4) a-def ac-Zp-factors'*

assms(1) int-pow-0 mult-comm ord-of-inc)
qed

lemma *angular-component-of-inclusion*:
assumes $x \in nonzero\ Z_p$
assumes $y = \iota\ x$
shows *angular-component* $y = ac\text{-}Zp\ x$
proof−
have $y = local.frac\ x\ \mathbf{1}_{Z_p}$
using *assms*
by (*simp add*: *Zp.nonzero-closed local.inc-def*)
then have *0*: *angular-component* $y = (ac\text{-}Zp\ x) \otimes_{Z_p} inv_{Z_p}\ ac\text{-}Zp\ (\mathbf{1}_{Z_p})$
using *assms ac-fract*[*of y x*]
by (*simp add*: ‹$y = \iota\ x$› *Zp.nonzero-closed Zp.nonzero-one-closed local.frac-closed*)
have *1*: $ac\text{-}Zp\ \mathbf{1}_{Z_p} = \mathbf{1}_{Z_p}$
using *ac-Zp-one* **by** *blast*
have *2*: $(ac\text{-}Zp\ x)\ \in carrier\ Z_p$
using *Zp.Units-closed Zp.nonzero-closed ac-Zp-is-Unit assms(1) nonzero-fraction-imp-numer-not-zero nonzero-numer-imp-nonzero-fraction* **by** *auto*
have *3*: $inv_{Z_p}\ (ac\text{-}Zp\ \mathbf{1}_{Z_p}) = \mathbf{1}_{Z_p}$
using *1*
by *simp*
have *4*: $(ac\text{-}Zp\ x) \otimes_{Z_p} inv_{Z_p}\ ac\text{-}Zp\ (\mathbf{1}_{Z_p}) = (ac\text{-}Zp\ x) \otimes_{Z_p} \mathbf{1}_{Z_p}$
using *3* **by** *auto*
have *5*: $(ac\text{-}Zp\ x) \otimes_{Z_p} inv_{Z_p}\ ac\text{-}Zp\ (\mathbf{1}_{Z_p}) = (ac\text{-}Zp\ x)$
using *4 2 Zp.domain-axioms* **by** *simp*
then show *?thesis*
by (*simp add*: *0*)
qed

lemma *res-uminus*:
assumes $k > 0$
assumes $f \in carrier\ Z_p$
assumes $c \in carrier\ (Zp\text{-}res\text{-}ring\ k)$
assumes $c = \ominus_{Zp\text{-}res\text{-}ring\ k}\ (f\ k)$
shows $c = ((\ominus_{Z_p}\ f)\ k)$
using $Z_p\text{-}def\ assms(2)\ assms(4)$ *prime Zp-residue-a-inv(1)* **by** *auto*

lemma *ord-fract*:
assumes $a \in nonzero\ Q_p$
assumes $b \in nonzero\ Q_p$
shows $ord\ (a \div b) = ord\ a - ord\ b$
using *assms nonzero-memE nonzero-def nonzero-inverse-Qp ord-mult ord-of-inv*

Qp.nonzero-closed Qp.nonzero-memE(2) **by** *presburger*

lemma *val-fract*:

assumes $a \in$ *carrier* Q_p
assumes $b \in$ *nonzero* Q_p
shows *val* $(a \div b) =$ *val* $a -$ *val* b
apply(*cases* $a = \mathbf{0}$)
apply (*simp add*: *Units-eq-nonzero assms*(*2*) *val-def*)
proof−
assume $a \neq \mathbf{0}$
then have *0*: $a \in$ *nonzero* Q_p
by (*simp add*: *Qp.nonzero-memI assms*(*1*))
have *1*: $(a \div b) \in$ *nonzero* Q_p
by (*simp add*: *0 Localization.submonoid.m-closed Qp.nonzero-is-submonoid assms*(*2*) *nonzero-inverse-Qp*)
show *?thesis* **using** *0 1 assms*
by (*simp add*: *ord-fract*)
qed

lemma *zero-fract*:
assumes $a \in$ *nonzero* Q_p
shows $\mathbf{0} \div a = \mathbf{0}$
by (*simp add*: *Units-eq-nonzero assms*)

lemma *fract-closed*:
assumes $a \in$ *carrier* Q_p
assumes $b \in$ *nonzero* Q_p
shows $(a \div b) \in$ *carrier* Q_p
by (*simp add*: *Units-eq-nonzero assms*(*1*) *assms*(*2*))

lemma *val-of-power*:
assumes $a \in$ *nonzero* Q_p
shows *val* $(a[\hat{}](n::nat)) = n*(val\ a)$
unfolding *val-def* **using** *assms*
by (*simp add*: *Qp.nonzero-memE*(*2*) *Qp-nat-pow-nonzero nonzero-nat-pow-ord*)

lemma *val-zero-imp-val-pow-zero*:
assumes $a \in$ *carrier* Q_p
assumes *val* $a = 0$
shows *val* $(a[\hat{}](n::nat)) = 0$
apply(*induction n*)
using *assms* **apply** *simp*
proof−
fix n ::*nat*
assume *A*: *val* $(a\ [\hat{}]\ n) = 0$
have *0*: $a\ [\hat{}]\ Suc\ n = a\ [\hat{}]\ n \otimes a$
by *simp*
show *val* $(a\ [\hat{}]\ Suc\ n) = 0$
unfolding *0* **using** *A assms*
by (*simp add*: *val-mult*)
qed

val and ord of powers of p

lemma *val-p-nat-pow*:
val (𝔭[^](*k*::*nat*)) = *eint k*
by (*simp add*: *ord-p-pow-nat*)

lemma *ord-p-int-pow*:
ord (𝔭[^](*k*::*int*)) = *k*
by (*simp add*: *ord-p-pow-int*)

lemma *ord-p-nat-pow*:
ord (𝔭[^](*k*::*nat*)) = *k*
by (*simp add*: *ord-p-pow-nat*)

lemma *val-nonzero-frac*:
assumes *a* ∈ *nonzero* Q_p
assumes *b* ∈ *nonzero* Q_p
assumes *val* (*a* ÷ *b*) = *c*
shows *val a* = *val b* + *c*
proof−
obtain *n* **where** *n-def*: *c* = *eint n*
using *assms val-ord* **by** (*metis* (*no-types*, *lifting*) *Qp.integral Qp.nonzero-closed inv-in-frac*(*1*) *inv-in-frac*(*2*) *val-def*)
have *0*: *ord* (*a* ÷ *b*) = *n*
by (*metis assms*(*3*) *eint.distinct*(*2*) *eint.inject n-def val-def*)
have *1*: *ord a* − *ord b* = *n*
using *0 assms* **by** (*metis ord-fract*)
have *2*: *ord a* = *ord b* + *n*
using *1* **by** *presburger*
show *val a* = *val b* + *c*
unfolding *n-def* **using** *2 assms*
by (*metis Qp.nonzero-memE*(*1*) *fract-closed local.fract-cancel-right n-def val-mult*)
qed

lemma *val-nonzero-frac*′:
assumes *a* ∈ *nonzero* Q_p
assumes *b* ∈ *nonzero* Q_p
assumes *val* (*a* ÷ *b*) = *0*
shows *val a* = *val b*
using *val-nonzero-frac*[*of a b 0*]
by (*metis arith-simps*(*50*) *assms*(*1*) *assms*(*2*) *assms*(*3*))

lemma *equal-val-imp-equal-ord*:
assumes *a* ∈ *nonzero* Q_p
assumes *b* ∈ *carrier* Q_p
assumes *val a* = *val b*
shows *ord a* = *ord b b* ∈ *nonzero* Q_p
using *assms*
apply (*metis eint.inject eint.simps*(*3*) *eint-ord-simps*(*4*) *val-nonzero val-ord*)
using *assms* **unfolding** *nonzero-def*
using *val-def* **by** *auto*

lemma *int-pow-ord*:
assumes $a \in nonzero\ Q_p$
shows $ord\ (a[\hat{}](i::int)) = i* (ord\ a)$
proof(*induction i*)
show $\bigwedge n.\ ord\ (a\ [\hat{}]\ int\ n) = int\ n * ord\ a$
using *assms*
by (*metis int-pow-int mult-of-nat-commute nonzero-nat-pow-ord*)
show $\bigwedge n.\ ord\ (a\ [\hat{}] - int\ (Suc\ n)) = - int\ (Suc\ n) * ord\ a$
proof− **fix** *n*
have $(a\ [\hat{}] - int\ (Suc\ n)) \otimes (a\ [\hat{}]\ \ int\ (Suc\ n)) = a\ [\hat{}]\ (- int\ (Suc\ n) + int\ (Suc\ n))$
using *assms Qp-int-pow-add* **by** *blast*
hence $(a\ [\hat{}] - int\ (Suc\ n)) \otimes (a\ [\hat{}]\ \ int\ (Suc\ n)) = \mathbf{1}$
using *assms*
by (*metis ab-group-add-class.ab-left-minus int-pow-0*)
hence $ord\ (a\ [\hat{}] - int\ (Suc\ n)) + ord\ (a\ [\hat{}]\ \ int\ (Suc\ n)) = 0$
by (*metis Qp-int-pow-nonzero assms ord-mult ord-one*)
thus $ord\ (a\ [\hat{}] - int\ (Suc\ n)) = - int\ (Suc\ n) * ord\ a$
using *nonzero-nat-pow-ord assms*
by (*metis* ‹$\bigwedge n.\ ord\ (a\ [\hat{}]\ int\ n) = int\ n * ord\ a$› *add.inverse-inverse add-left-cancel more-arith-simps*(4) *more-arith-simps*(7))
qed
qed

lemma *int-pow-val*:
assumes $a \in nonzero\ Q_p$
shows $val\ (a[\hat{}](i::int)) = i* (val\ a)$
using *int-pow-ord Qp-int-pow-nonzero assms times-eint-simps*(1) *val-ord* **by** *presburger*

lemma *neg-int-pow-val*:
assumes $a \in nonzero\ Q_p$
shows $val\ (a[\hat{}]-(i::int)) = - (val\ (a[\hat{}]i))$
using *int-pow-val*[*of a i*]
by (*metis Qp-int-pow-nonzero assms int-pow-ord mult-minus-left val-ord val-p-int-pow val-p-int-pow-neg*)

lemma *int-pow-sum-val*:
assumes $a \in nonzero\ Q_p$
shows $val\ (a[\hat{}]((i::int) + j)) = \ (val\ (a[\hat{}]i)) + val\ (a[\hat{}]j)$
using *assms Qp.int-pow-add Qp-int-pow-nonzero Units-eq-nonzero val-mult0* **by** *presburger*

lemma *int-pow-diff-val*:
assumes $a \in nonzero\ Q_p$
shows $val\ (a[\hat{}]((i::int) - j)) = \ (val\ (a[\hat{}]i)) - val\ (a[\hat{}]j)$
proof−
obtain *k* **where** *k-def*: $val\ a = eint\ k$

using *assms val-ord* **by** *blast*
have *0*: *val* (*a*[^](*i*::*int*) − *j*)) = *eint* ((*i*−*j*)∗*k*)
using *assms k-def int-pow-val times-eint-simps*(*1*) **by** *presburger*
show *?thesis* **unfolding** *0* **using** *k-def*
by (*metis 0 Qp-int-pow-nonzero Rings.ring-distribs*(*3*) *assms eint.simps*(*1*)
idiff-eint-eint int-pow-ord val-ord)
qed

lemma *nat-add-pow-mult-assoc*:
assumes *a* ∈ *carrier* Q_p
shows [(*n*::*nat*)]·*a* = [*n*]·**1** ⊗ *a*
using *assms Qp.add-pow-ldistr Qp.l-one Qp.one-closed* **by** *presburger*

lemma(**in** *padic-integers*) *equal-res-imp-equal-ord-Zp*:
assumes *N* > *0*
assumes *a* ∈ *carrier Zp*
assumes *b* ∈ *carrier Zp*
assumes *a N* = *b N*
assumes *a N* ≠ *0*
shows *ord-Zp a* = *ord-Zp b*
proof−
have (*a* ⊖ *b*) *N* = *0*
using *assms*(*2*) *assms*(*3*) *assms*(*4*) *res-diff-zero-fact*′′ **by** *blast*
have *ord-Zp a* < *N*
using *assms zero-below-ord* **by** *force*
then show *?thesis*
by (*smt R.minus-closed Zp-def* ‹(*a* ⊖ *b*) *N* = *0*› *above-ord-nonzero assms*(*2*)
assms(*3*) *assms*(*4*) *assms*(*5*) *ord-Zp-ultrametric-eq*′ *ord-Zp-ultrametric-eq*′′ *padic-integers.ord-Zp-not-equal-imp*
padic-integers.ord-of-nonzero(*2*) *padic-integers.ord-pos padic-integers-axioms residue-of-zero*(*2*))
qed

lemma(**in** *padic-integers*) *equal-res-mod*:
assumes *N* > *k*
assumes *a* ∈ *carrier Zp*
assumes *b* ∈ *carrier Zp*
assumes *a N* = *b N*
shows *a k* = *b k*
proof−
have *0*: *a k* = *a N mod p*^*k*
using *assms*
by (*metis Zp-int-inc-rep Zp-int-inc-res less-or-eq-imp-le p-residue-padic-int*)
have *1*: *b k* = *b N mod p*^*k*
using *assms*
by (*metis Zp-int-inc-rep Zp-int-inc-res less-or-eq-imp-le p-residue-padic-int*)
show *?thesis* **unfolding** *0 1 assms* **using** *assms* **by** *blast*
qed

lemma *Qp-char-0*:
assumes (*n*::*nat*) ≠ *0*

shows $[n]\cdot\mathbf{1} \neq \mathbf{0}$
proof−
have $[n]\cdot_{Z_p} \mathbf{1}_{Z_p} \neq \mathbf{0}_{Z_p}$
using *Zp-char-0′ assms* **by** *blast*
thus *?thesis* **using** *inc-of-nat*
by (*metis Qp.nat-inc-zero Zp.nat-inc-zero Zp-nat-inc-closed ι-def inc-inj2*)
qed

lemma *Qp-char-0-int*:
assumes (*n*::*int*) ≠ *0*
shows $[n]\cdot\mathbf{1} \neq \mathbf{0}$
proof−
have $[n]\cdot_{Z_p} \mathbf{1}_{Z_p} \neq \mathbf{0}_{Z_p}$
using *Zp-char-0 assms*
by (*metis int-inc-mult linorder-neqE-linordered-idom mult-zero-l zero-less-mult-iff*)

thus *?thesis* **using** *inc-of-int*
by (*metis* Q_p*-def Zp-int-inc-closed ι-def inc-inj1*)
qed

lemma *add-int-pow-inject*:
assumes $[(k::int)]\cdot\mathbf{1} = [(j::int)]\cdot\mathbf{1}$
shows $k = j$
proof(*rule ccontr*)
assume *A*: $k \neq j$
then have *0*: $k - j \neq 0$
by *presburger*
hence *1*: $[(k-j)]\cdot\mathbf{1} \neq \mathbf{0}$
using *Qp-char-0-int* **by** *blast*
hence *2*: $[k]\cdot\mathbf{1} \oplus [(-j)]\cdot\ \mathbf{1}\ \neq \mathbf{0}$
by (*metis* (*no-types*, *opaque-lifting*) *Qp.add.int-pow-mult Qp.one-closed diff-minus-eq-add diff-zero minus-diff-eq*)
hence *2*: $[k]\cdot\mathbf{1} \ominus [j]\cdot\ \mathbf{1}\ \neq \mathbf{0}$
by (*metis Qp.add.int-pow-mult Qp.int-inc-zero Qp.one-closed assms more-arith-simps*(*4*))
thus *False* **using** *assms*
using *Qp.int-inc-closed Qp.r-right-minus-eq* **by** *blast*
qed

lemma *val-ord-nat-inc*:
assumes (*n*::*nat*) > *0*
shows *ord* $([n]\cdot\mathbf{1}) = val([n]\cdot\mathbf{1})$
using *val-ord assms Qp-char-0*
by (*metis less-irrefl-nat val-def*)

lemma *val-ord-int-inc*:
assumes (*n*::*int*) ≠ *0*
shows *ord* $([n]\cdot\mathbf{1}) = val([n]\cdot\mathbf{1})$
using *val-ord assms Qp-char-0-int val-def* **by** *presburger*

8.4.2 Reduced Angular Component Maps

definition *ac* :: *nat* ⇒ *padic-number* ⇒ *int* **where**
ac n x = (*if x* = **0** *then 0 else* (*angular-component x*) *n*)

lemma *ac-in-res-ring*:
assumes *x* ∈ *nonzero* Q_p
shows *ac n x* ∈ *carrier* (*Zp-res-ring n*)
unfolding *ac-def*
using *assms angular-component-closed*[*of x*]
by (*simp add*: *Qp.nonzero-memE*(*2*) *residues-closed*)

lemma *ac-in-res-ring'*[*simp*]:
assumes *x* ∈ *carrier* Q_p
shows *ac n x* ∈ *carrier* (*Zp-res-ring n*)
apply(*cases x* ∈ *nonzero* Q_p)
using *ac-in-res-ring* **apply** *blast*
by (*metis Qp.nonzero-memI ac-def assms mod-0 mod-in-carrier p-res-ring-car p-residue-alt-def*)

lemma *ac-mult'*:
assumes *x* ∈ *nonzero* Q_p
assumes *y* ∈ *nonzero* Q_p
shows *ac n* (*x* ⊗ *y*) = (*ac n x*) ⊗$_{Zp\text{-}res\text{-}ring\ n}$ (*ac n y*)
unfolding *ac-def*
proof−
have *0*: *angular-component* (*x* ⊗ *y*) = *angular-component x* ⊗$_{Z_p}$ *angular-component y*
using *assms angular-component-mult*[*of x y*]
by *auto*
show (*if x* ⊗ *y* = **0** *then 0 else angular-component* (*x* ⊗ *y*) *n*) =
(*if x* = **0** *then 0 else angular-component x n*) ⊗$_{Zp\text{-}res\text{-}ring\ n}$ (*if y* = **0** *then 0 else angular-component y n*)
using *assms angular-component-closed*[*of x*] *angular-component-closed*[*of y*]
by (*simp add*: *0 Qp.integral-iff Qp.nonzero-closed Qp.nonzero-memE*(*2*) *residue-of-prod*)

qed

lemma *ac-mult*:
assumes *x* ∈ *carrier* Q_p
assumes *y* ∈ *carrier* Q_p
shows *ac n* (*x* ⊗ *y*) = (*ac n x*) ⊗$_{Zp\text{-}res\text{-}ring\ n}$ (*ac n y*)
apply(*cases x* ∈ *nonzero* Q_p ∧ *y* ∈ *nonzero* Q_p)
apply (*simp add*: *ac-mult'*)
using *assms* **unfolding** *ac-def*
by (*smt Qp.integral-iff Qp.nonzero-memI residue-times-zero-l residue-times-zero-r*)

lemma *ac-one*[*simp*]:
assumes *n* ≥ *1*
shows *ac n* **1** = *1*

proof−
  have *ac n* $\mathbf{1} = \mathbf{1}_{Z_p}$ *n*
    unfolding *ac-def*
    using *angular-component-one*
    by *simp*
  then show *?thesis*
    using *assms residue-of-one*(*2*) **by** *auto*
qed

lemma *ac-one'*:
  assumes *n* > *0*
  shows *ac n* $\mathbf{1} = \mathbf{1}_{Zp\text{-}res\text{-}ring\ n}$
  using *assms residue-ring-def*
  by *auto*

lemma *ac-units*:
  assumes *x* ∈ *nonzero* Q_p
  assumes *n* > *0*
  shows *ac n x* ∈ *Units* (*Zp-res-ring n*)
proof−
  obtain *y* **where** *y-def*: *y* = *inv x*
    by *simp*
  have *y-nz*: *y* ∈ *nonzero* Q_p
    using *assms*(*1*) *nonzero-inverse-Qp y-def*
    by *blast*
  have *0*: *ac n* (*x* ⊗ *y*) = (*ac n x*) $\otimes_{Zp\text{-}res\text{-}ring\ n}$ (*ac n y*)
    using *ac-mult' assms*(*1*) *y-nz* **by** *blast*
  have *1*: *x* ⊗ *y* = **1**
   by (*metis Qp.field-axioms Qp.nonzero-memE Qp.nonzero-memE assms*(*1*) *field.field-inv*(*2*) *y-def*)
  have *2*: (*ac n x*) $\otimes_{Zp\text{-}res\text{-}ring\ n}$ (*ac n y*) = $\mathbf{1}_{Zp\text{-}res\text{-}ring\ n}$
    using *0 1 ac-one' assms*(*2*)
    by *auto*
  show *?thesis*
   by (*metis 2 R-comm-monoid ac-in-res-ring assms*(*1*) *assms*(*2*) *comm-monoid.UnitsI*(*1*) *nonzero-inverse-Qp y-def*)

qed

lemma *ac-inv*:
  assumes *x* ∈ *nonzero* Q_p
  assumes *n* > *0*
  shows *ac n* (*inv x*) = $inv_{Zp\text{-}res\text{-}ring\ n}$ (*ac n x*)
proof−
  have *x* ⊗ *inv x* = **1**
   by (*simp add*: *Qp.monoid-axioms Units-eq-nonzero assms*(*1*) *monoid.Units-r-inv*)

  then have *ac n* (*x* ⊗ *inv x*) = $\mathbf{1}_{Zp\text{-}res\text{-}ring\ n}$
    by (*simp add*: ‹(*x* ÷ *x*) = **1**› *ac-one' assms*(*2*))

then have $ac\ n\ x \otimes_{Zp\text{-}res\text{-}ring\ n} ac\ n\ (inv\ x) = \mathbf{1}_{Zp\text{-}res\text{-}ring\ n}$
using *Units-eq-nonzero Units-inverse-Qp ac-mult' assms(1)*
by *metis*
then show *?thesis*
by (*metis R-comm-monoid ac-in-res-ring assms(1) assms(2) comm-monoid.comm-inv-char nonzero-inverse-Qp*)
qed

lemma *ac-inv'*:
assumes $x \in nonzero\ Q_p$
assumes $n > 0$
shows $ac\ n\ (inv\ x) \otimes_{Zp\text{-}res\text{-}ring\ n} (ac\ n\ x) = \mathbf{1}_{Zp\text{-}res\text{-}ring\ n}$
using *ac-inv[of x n] ac-units[of x n] assms*
by (*metis* (*no-types*, *opaque-lifting*) *Qp.field-axioms Qp.nonzero-memE Qp.nonzero-memE Units-eq-nonzero Units-inverse-Qp ac-mult ac-one' field.field-inv(1)*)

lemma *ac-inv''*:
assumes $x \in nonzero\ Q_p$
assumes $n > 0$
shows $(ac\ n\ x) \otimes_{Zp\text{-}res\text{-}ring\ n} ac\ n\ (inv\ x) = \mathbf{1}_{Zp\text{-}res\text{-}ring\ n}$
using *ac-inv' assms(1) assms(2) residue-mult-comm* **by** *auto*

lemma *ac-inv'''*:
assumes $x \in nonzero\ Q_p$
assumes $n > 0$
shows $(ac\ n\ x) \otimes_{Zp\text{-}res\text{-}ring\ n} ac\ n\ (inv\ x) = 1$
$ac\ n\ (inv\ x) \otimes_{Zp\text{-}res\text{-}ring\ n} (ac\ n\ x) = 1$
by (*auto simp add*: *ac-inv'' ac-inv' assms(1) assms(2) p-res-ring-one*)

lemma *ac-val*:
assumes $a \in nonzero\ Q_p$
assumes $b \in nonzero\ Q_p$
assumes $val\ a = val\ b$
assumes $val\ (a \ominus b) \geq val\ a + n$
shows $ac\ n\ a = ac\ n\ b$
proof−
obtain m **where** *m-def*: $m = ord\ a$
by *simp*
have *0*: $a = (\mathfrak{p}[\hat{}]m) \otimes \iota\ (angular\text{-}component\ a)$
by (*simp add*: *angular-component-factors-x assms(1) m-def*)
have *1*: $b = (\mathfrak{p}[\hat{}]m) \otimes \iota\ (angular\text{-}component\ b)$
proof−
have $ord\ b = ord\ a$
using *assms(1) assms(2) assms(3)* **by** *auto*
then show *?thesis*
by (*metis angular-component-factors-x assms(2) m-def*)
qed
have *2*: $(a \ominus b) = (\mathfrak{p}[\hat{}]m) \otimes \iota\ (angular\text{-}component\ a)$

$\ominus(\mathfrak{p}[\widehat{}]m) \otimes \iota$ (*angular-component b*)
using *0 1* **by** *auto*
have *3*: (*a* $\ominus$*b*) = ($\mathfrak{p}[\widehat{}]m$) $\otimes$(ι (*angular-component a*)
$\ominus$ ι (*angular-component b*))
using *2 assms angular-component-closed inc-closed Qp.cring-simprules Qp.r-minus-distr*
by (*metis* (*no-types*, *lifting*) *p-intpow-closed*(*1*))
have *4*: (*a* $\ominus$*b*) = ($\mathfrak{p}[\widehat{}]m$) $\otimes$(ι ((*angular-component a*)
$\ominus_{Z_p}$(*angular-component b*)))
by (*simp add*: *3 angular-component-closed assms*(*1*) *assms*(*2*) *inc-of-diff*)
have *5*: *val* (*a* $\ominus$*b*) = *m* + *val* ((ι ((*angular-component a*)
$\ominus_{Z_p}$(*angular-component b*))))
by (*metis 4 Zp.nonzero-one-closed angular-component-closed assms*(*1*) *assms*(*2*)
cring.cring-simprules(*4*) *frac-closed local.inc-def ord-p-pow-int p-intpow-closed*(*1*)
p-intpow-closed(*2*) *Zp.domain-axioms domain-def val-mult val-ord*)
have *6*: *m* = *val a*
using Q_p*-def assms*(*1*) *m-def* **by** *auto*
have *7*: *m* + *val* (ι (*angular-component a* $\ominus_{Z_p}$ *angular-component b*)) $\geq$ *val a*
+ *n*
using *5 assms*(*4*) **by** *presburger*
have *8*: *m* + *val* (ι (*angular-component a* $\ominus_{Z_p}$ *angular-component b*)) $\geq$ *m* + *n*
using *6 7*
by (*metis plus-eint-simps*(*1*))
have *9*: *val* (ι (*angular-component a* $\ominus_{Z_p}$ *angular-component b*)) $\geq$ *n*
using *8 add-le-cancel-left eint-ile eint-ord-simps*(*1*) *linear plus-eint-simps*(*1*)
by *metis*
have *10*: *val-Zp* (*angular-component a* $\ominus_{Z_p}$ *angular-component b*) $\geq$ *n*
using *9*
by (*metis angular-component-closed assms*(*1*) *assms*(*2*) *cring.cring-simprules*(*4*)
Zp.domain-axioms domain-def val-of-inc)
have *11*: (*angular-component a* $\ominus_{Z_p}$ *angular-component b*) *n* = $\mathbf{0}_{Zp\text{-}res\text{-}ring\ n}$
using *9*
by (*simp add*: *10 angular-component-closed assms*(*1*) *assms*(*2*) *p-res-ring-zero*
zero-below-val-Zp)
have *12*: (*angular-component a n*) $\ominus_{Zp\text{-}res\text{-}ring\ n}$ (*angular-component b n*) =
$\mathbf{0}_{Zp\text{-}res\text{-}ring\ n}$
using *11 angular-component-closed assms*(*2*) *residue-of-diff* **by** *auto*
have *13*: (*angular-component a n*) = (*angular-component b n*)
apply(*cases n* = *0*)
apply (*metis angular-component-closed assms*(*1*) *assms*(*2*) *p-res-ring-0' residues-closed*)

using *12 angular-component-closed residue-closed ring.ring-simprules*[*of Zp-res-ring*
n]
by (*meson assms*(*1*) *assms*(*2*) *cring-def prime residues.cring residues-n ring.r-right-minus-eq*)

then show *?thesis*
by (*simp add*: *Qp.nonzero-memE ac-def assms*(*1*) *assms*(*2*))
qed

lemma *angular-component-nat-pow*:

assumes $a \in$ *nonzero* Q_p
shows *angular-component* $(a\ [\hat{}]\ (k::nat)) = (angular\text{-}component\ a)\ [\hat{}]_{Z_p}\ k$
apply(*induction k*)
apply (*simp add*: *angular-component-one*)
by (*simp add*: *Qp-nat-pow-nonzero angular-component-mult assms*)

lemma *angular-component-int-pow*:
assumes $a \in$ *nonzero* Q_p
shows *angular-component* $(a\ [\hat{}]\ (k::int)) = (angular\text{-}component\ a)\ [\hat{}]_{Z_p}\ k$
apply(*cases* $k \geq 0$)
using *angular-component-nat-pow assms*
apply (*metis int-pow-int nonneg-int-cases*)
proof−
assume $\neg\ 0 \leq k$
show *angular-component* $(a\ [\hat{}]\ k) =$ *angular-component* $a\ [\hat{}]_{Z_p}\ k$
using *angular-component-nat-pow*[*of a nat* $(-k)$] *assms*
Qp-nat-pow-nonzero[*of a*] ‹$\neg\ 0 \leq k$› *angular-component-inv*[*of* $(a\ [\hat{}]\ k)$] *int-pow-def2*
by (*metis angular-component-nat-pow angular-component-inv*)
qed

lemma *ac-nat-pow*:
assumes $a \in$ *nonzero* Q_p
shows *ac n* $(a\ [\hat{}]\ (k::nat)) = (ac\ n\ a)\hat{}\ k\ mod\ (p\hat{}n)$
proof(*cases* $k = 0$)
case *True*
show *?thesis* **apply**(*cases* $n = 0$)
apply (*metis Group.nat-pow-0 Qp.one-closed True ac-in-res-ring' mod-self p-res-ring-0' power-0*)
using *True prime residues.one-cong residues.res-one-eq residues-n* **by** *auto*
next
case *False*
show *?thesis*
apply(*cases* $n = 0$)
apply (*metis Qp-nat-pow-nonzero ac-in-res-ring assms mod-by-1 p-res-ring-0' power-0*)
using *assms angular-component-nat-pow*
by (*metis Qp.nonzero-closed Qp.nonzero-pow-nonzero Qp.not-nonzero-memI ac-def angular-component-closed neq0-conv power-residue*)
qed

lemma *ac-nat-pow'*:
assumes $a \in$ *nonzero* Q_p
assumes $n \neq 0$
shows *ac n* $(a\ [\hat{}]\ (k::nat)) = (ac\ n\ a)[\hat{}]_{Zp\text{-}res\text{-}ring\ n}\ k$
proof−
have $(ac\ n\ a)\hat{}\ k\ mod\ (p\hat{}n) = (ac\ n\ a)[\hat{}]_{Zp\text{-}res\text{-}ring\ n}\ k$
apply(*induction k*)
apply (*metis Group.nat-pow-0 assms*(*2*) *prime residues.pow-cong residues-n*)

by (*metis Group.nat-pow-Suc Qp-nat-pow-nonzero ac-mult' ac-nat-pow assms(1)*)
then show *?thesis*
by (*simp add*: *ac-nat-pow assms(1)*)
qed

lemma *ac-int-pow*:
assumes $a \in$ *nonzero* Q_p
assumes $n > 0$
shows *ac n* (*a* [^] (*k*::*int*)) = (*ac n a*)[^]$_{Zp\text{-}res\text{-}ring\ n}$ *k*
apply(*cases k* $\geq$*0*)
using *assms ac-nat-pow'*
apply (*metis int.lless-eq int.nat-pow-0 p-residues pow-nat residues-def*)
using *assms ac-nat-pow' ac-inv*[*of a* [^] *k*] *ac-units*[*of a* [^] *k*]
by (*metis Qp-nat-pow-nonzero ac-inv int-pow-def2 neq0-conv*)

lemma *angular-component-p*:
angular-component $\mathfrak{p} = \mathbf{1}_{Z_p}$
proof−
have $\mathfrak{p}$ = *frac* p $\mathbf{1}_{Z_p}$
by (*simp add*: *Zp-nat-inc-closed local.inc-def p-inc*)
then have *0*: *angular-component* $\mathfrak{p}$ = (*ac-Zp* p) $\otimes_{Z_p}$ inv_{Z_p} (*ac-Zp* $\mathbf{1}_{Z_p}$)
using *ac-Zp-one* **unfolding** *angular-component-def*
by (*metis* Q_p*-def Qp.int-inc-closed Zp.one-nonzero Zp-int-inc-closed ac-fract*
local.numer-denom-facts(*2*) *local.numer-denom-facts*(*5*) *nonzero-fraction-imp-nonzero-numer*
nonzero-numer-imp-nonzero-fraction numer-nonzero p-nonzero)
then have *angular-component* $\mathfrak{p}$ = (*ac-Zp* p) $\otimes_{Z_p}$ inv_{Z_p} $\mathbf{1}_{Z_p}$
by (*simp add*: *ac-Zp-one*)
then have *angular-component* $\mathfrak{p}$ = (*ac-Zp* p)
by (*simp add*: *ac-Zp-p*)
then show *?thesis*
using *ac-Zp-p*
by *simp*
qed

lemma *angular-component-p-nat-pow*:
angular-component ($\mathfrak{p}$ [^] (*n*::*nat*)) = $\mathbf{1}_{Z_p}$
apply(*induction n*)
apply (*simp add*: *angular-component-one*)
using *angular-component-nat-pow angular-component-p nat-pow-one p-nonzero*
Zp.nat-pow-one
by *presburger*

lemma *angular-component-p-int-pow*:
angular-component ($\mathfrak{p}$ [^] (*n*::*int*)) = $\mathbf{1}_{Z_p}$
apply(*cases n* $\geq$ *0*)
apply (*metis angular-component-p-nat-pow int-pow-int nonneg-int-cases*)
using *angular-component-p-nat-pow*[*of nat* (−*n*)] *angular-component-inv*[*of* ($\mathfrak{p}$
[^] *n*)] *angular-component-inv*[*of* $\mathfrak{p}$ [^] (*nat* (−*n*))]
by (*metis* (*mono-tags*, *opaque-lifting*) *Group.nat-pow-0 Zp.inv-one add.inverse-neutral*

angular-component-one int-nat-eq
int-pow-def2 nat-0-iff nat-int nat-zminus-int p-natpow-closed(2))

lemma *ac-p-nat-pow*:
assumes $k > 0$
shows *ac k* (𝔭 [^] (*n*::*nat*)) = *1*
proof−
have ¬ (𝔭 [^] *n*) = **0**
by (*simp add*: *Qp.nonzero-memE*)
have *angular-component* (𝔭 [^] *n*) *k* = *1*
using *assms angular-component-p-nat-pow*[*of n*] *ac-def ac-one angular-component-one*

by *auto*
then show *?thesis*
unfolding *ac-def* **using** *angular-component-p-nat-pow*[*of n*]
by (*simp add*: ‹(𝔭 [^] *n*) ≠ **0**›)
qed

lemma *ac-p*:
assumes $k > 0$
shows *ac k* 𝔭 = *1*
by (*metis Qp.int-inc-closed Qp.nat-pow-eone ac-p-nat-pow assms*)

lemma *ac-p-int-pow*:
assumes $k > 0$
shows *ac k* (𝔭 [^] (*n*::*int*)) = *1*
using *Qp.nonzero-memE*(*2*) *ac-def ac-one′ angular-component-one angular-component-p-int-pow*
assms p-intpow-closed(*2*) *p-res-ring-one* **by** *auto*

lemma *angular-component-p-nat-pow-factor*:
assumes *a* ∈ *nonzero* Q_p
shows *angular-component* ((𝔭 [^] (*n*::*nat*)) ⊗ *a*) = *angular-component a*
proof−
have *0*: *angular-component* ((𝔭 [^] *n*) ⊗ *a*) = *angular-component* (𝔭 [^] *n*) $\otimes_{Z_p}$
angular-component a
using *assms angular-component-mult*[*of* (𝔭 [^] (*n*::*nat*)) *a*] *p-natpow-closed*(*2*)
by *blast*
have *1*: *angular-component* (𝔭 [^] *n*) = $\mathbf{1}_{Z_p}$
by (*simp add*: *angular-component-p-nat-pow*)
have *2*: *angular-component* ((𝔭 [^] *n*) ⊗ *a*) = $\mathbf{1}_{Z_p}$ $\otimes_{Z_p}$ *angular-component a*
by (*simp add*: *0 1*)
then show *?thesis* **using** *angular-component-closed*[*of a*] *assms Zp.domain-axioms*
by (*simp add*: *assms cring.cring-simprules*(*12*) *domain.axioms*(*1*))
qed

lemma *ac-p-nat-pow-factor*:
assumes $m > 0$
assumes *a* ∈ *nonzero* Q_p
shows *ac m* ((𝔭 [^] (*n*::*nat*)) ⊗ *a*) = *ac m a*

using *angular-component-p-nat-pow-factor assms ac-def*
by (*metis* (*no-types*, *lifting*) *Qp.field-axioms Qp-nat-pow-nonzero Qp.nonzero-memE*

Qp.nonzero-memE Ring.integral p-nonzero)

lemma *angular-component-p-nat-pow-factor-right*:
assumes $a \in$ *nonzero* Q_p
shows *angular-component* $(a \otimes (\mathfrak{p}\ [\hat{}]\ (n::nat))) =$ *angular-component a*
proof−
have $((\mathfrak{p}\ [\hat{}]\ (n::nat)) \otimes a) = (a \otimes (\mathfrak{p}\ [\hat{}]\ (n::nat)))$
using *assms Qp.domain-axioms domain-def*
by (*simp add*: *assms domain-def Qp.nonzero-memE cring.cring-simprules*(14))
then show *?thesis*
using *angular-component-p-nat-pow-factor*[*of a n*]
by (*simp add*: *assms*)
qed

lemma *ac-p-nat-pow-factor-right*:
assumes $m > 0$
assumes $a \in$ *carrier* Q_p
shows *ac m* $(a \otimes (\mathfrak{p}\ [\hat{}]\ (n::nat))) =$ *ac m a*
using *assms angular-component-p-nat-pow-factor-right*[*of a n*]
unfolding *ac-def*
by (*metis Qp.integral Qp.l-null Qp.not-nonzero-memE Qp.not-nonzero-memI*
angular-component-p-nat-pow-factor-right p-natpow-closed(1) *p-natpow-closed*(2))

lemma *angular-component-p-int-pow-factor*:
assumes $a \in$ *carrier* Q_p
shows *angular-component* $((\mathfrak{p}\ [\hat{}]\ (n::int)) \otimes a) =$ *angular-component a*
by (*metis Qp.integral-iff Qp.l-one Qp.nonzero-memI Qp.one-nonzero angular-component-mult*
angular-component-one angular-component-p-int-pow assms p-intpow-closed(1)
p-intpow-closed(2))

lemma *ac-p-int-pow-factor*:
assumes $a \in$ *nonzero* Q_p
shows *ac m* $((\mathfrak{p}\ [\hat{}]\ (n::int)) \otimes a) =$ *ac m a*
apply(*cases* $m = 0$)
apply (*metis* (*no-types*, *lifting*) *Qp.integral Qp.nonzero-closed Qp.nonzero-memE*(2)

ac-def angular-component-p-int-pow-factor assms p-intpow-closed(2))
using *assms angular-component-p-int-pow-factor*[*of a n*]
by (*metis* (*no-types*, *lifting*) *Qp.integral Qp.nonzero-closed Qp.not-nonzero-memI*
ac-def
p-intpow-closed(2))

lemma *angular-component-p-int-pow-factor-right*:
assumes $a \in$ *carrier* Q_p
shows *angular-component* $(a \otimes (\mathfrak{p}\ [\hat{}]\ (n::int))) =$ *angular-component a*
using *Qp.m-comm angular-component-p-int-pow-factor assms p-intpow-closed*(1)

by *auto*

lemma *ac-p-int-pow-factor-right*:
assumes $a \in$ *carrier* Q_p
shows *ac* m $(a \otimes (\mathfrak{p}\ [\widehat{}]\ (n::int))) =$ *ac* m a
using *assms angular-component-p-int-pow-factor-right* **unfolding** *ac-def*
using *Qp.integral-iff Qp.nonzero-memE*(2) *p-intpow-closed*(1) *p-intpow-closed*(2)
by *presburger*

8.5 An Inverse for the inclusion map ι

definition *to-Zp* **where**
to-Zp $a = (if\ (a \in \mathcal{O}_p)\ then\ (SOME\ x.\ x \in carrier\ Z_p \wedge \iota\ x = a)\ else\ \mathbf{0}_{Z_p})$

lemma *to-Zp-closed*:
assumes $a \in$ *carrier* Q_p
shows *to-Zp* $a \in$ *carrier* Z_p
apply(*cases* $a \in \mathcal{O}_p$)
using *assms* **unfolding** *to-Zp-def* ι*-def*
apply (*smt image-iff tfl-some*)
by *simp*

lemma *to-Zp-inc*:
assumes $a \in \mathcal{O}_p$
shows ι (*to-Zp* a) $= a$
proof−
obtain c **where** *c-def*: $c = (SOME\ x.\ x \in carrier\ Z_p \wedge \iota\ x = a)$
by *simp*
have $(\exists\ x.\ x \in carrier\ Z_p \wedge \iota\ x = a)$
using *assms*(1)
by *blast*
then have $c \in carrier\ Z_p \wedge \iota\ c = a$
using *c-def*
by (*metis* (*mono-tags*, *lifting*) *tfl-some*)
then show ι (*to-Zp* a) $= a$
using *to-Zp-def c-def assms*(1)
by *auto*
qed

lemma *inc-to-Zp*:
assumes $b \in$ *carrier* Z_p
shows *to-Zp* $(\iota\ b) = b$
proof−
have ι (*to-Zp* $(\iota\ b)) = (\iota\ b)$
using *assms to-Zp-inc*[*of* $\iota\ b$]
by *blast*
then show *?thesis*
using *inc-inj2*[*of b to-Zp* $(\iota\ b)$] *assms to-Zp-closed inc-closed*
unfolding ι*-def* Q_p*-def*

by *auto*
qed

lemma *to-Zp-add*:
assumes $a \in \mathcal{O}_p$
assumes $b \in \mathcal{O}_p$
shows *to-Zp* $(a \oplus b) =$ *to-Zp* $a \oplus_{Z_p}$ (*to-Zp* b)
by (*metis* (*no-types*, *lifting*) *Zp.domain-axioms assms*(*1*) *assms*(*2*)
cring.cring-simprules(*1*) *domain.axioms*(*1*) *imageE inc-of-sum inc-to-Zp*)

lemma *to-Zp-mult*:
assumes $a \in \mathcal{O}_p$
assumes $b \in \mathcal{O}_p$
shows *to-Zp* $(a \otimes b) =$ *to-Zp* $a \otimes_{Z_p}$ (*to-Zp* b)
proof−
have $(a \otimes b) \in \mathcal{O}_p$
by (*simp add*: *val-ring-subring assms*(*1*) *assms*(*2*) *subringE*(*6*))
then have
ι ((*to-Zp* a) $\otimes_{Z_p}$ (*to-Zp*) b) = ((ι (*to-Zp* a)) $\otimes$ (ι (*to-Zp* b)))
using *assms*(*1*) *assms*(*2*) *inc-of-prod inc-to-Zp*
by *auto*
then have ι ((*to-Zp* a) $\otimes_{Z_p}$ (*to-Zp*) b) = $a \otimes b$
by (*simp add*: *assms*(*1*) *assms*(*2*) *to-Zp-inc*)
then have *to-Zp* (ι ((*to-Zp* a) $\otimes_{Z_p}$ (*to-Zp*) b)) = *to-Zp* $(a \otimes b)$
by *simp*
then show *?thesis*
by (*metis* (*no-types*, *opaque-lifting*) *Zp.domain-axioms val-ring-subring assms*(*1*)
assms(*2*)
cring.cring-simprules(*5*) *domain.axioms*(*1*) *inc-to-Zp subringE*(*1*) *subset-iff*
to-Zp-closed)
qed

lemma *to-Zp-minus*:
assumes $a \in \mathcal{O}_p$
assumes $b \in \mathcal{O}_p$
shows *to-Zp* $(a \ominus b) =$ *to-Zp* $a \ominus_{Z_p}$ (*to-Zp* b)
by (*metis* (*no-types*, *lifting*) *Zp.domain-axioms assms*(*1*) *assms*(*2*) *cring-def* do-
main.axioms(*1*)
image-iff inc-of-diff inc-to-Zp ring.ring-simprules(*4*))

lemma *to-Zp-one*:
shows *to-Zp* $\mathbf{1} = \mathbf{1}_{Z_p}$
using *Zp-def Zp.one-closed* ι*-def inc-of-one inc-to-Zp padic-integers-axioms*
by *fastforce*

lemma *to-Zp-zero*:
shows *to-Zp* $\mathbf{0} = \mathbf{0}_{Z_p}$
using Q_p*-def Zp-def Zp.domain-axioms* ι*-def domain-frac.inc-inj1 inc-to-Zp*
padic-integers-axioms to-Zp-def domain-frac-axioms **by** *fastforce*

lemma *to-Zp-ominus*:
assumes $a \in \mathcal{O}_p$
shows *to-Zp* $(\ominus\ a) = \ominus_{Z_p}\ (to\text{-}Zp\ a)$
proof−
have $\ominus a \in \mathcal{O}_p$
by (*simp add*: *val-ring-subring assms subringE*(*5*))
then show *?thesis*
by (*metis* (*no-types*, *lifting*) *Zp.domain-axioms Zp.nonzero-one-closed assms cring.cring-simprules*(*3*) *domain.axioms*(*1*) *frac-uminus image-iff inc-to-Zp local.inc-def*)
qed

lemma *to-Zp-val*:
assumes $a \in \mathcal{O}_p$
shows *val-Zp* (*to-Zp a*) = *val a*
by (*metis assms imageE inc-to-Zp val-of-inc*)

lemma *val-of-nat-inc*:
val $([(k::nat)]\cdot\mathbf{1}) \geq 0$
proof−
have $[(k::nat)]\cdot\mathbf{1} \in \mathcal{O}_p$
by (*metis Zp.one-closed Zp-nat-mult-closed image-eqI inc-of-nat*)
thus *?thesis*
using *val-ring-memE*(*1*) **by** *blast*
qed

lemma *val-of-int-inc*:
val $([(k::int)]\cdot\mathbf{1}) \geq 0$
proof−
have $\iota\ ([k]\ \cdot_{Z_p}\ \mathbf{1}_{Z_p}) = [k]\cdot\mathbf{1}$
using *inc-of-int* **by** *blast*
hence $[k]\cdot\mathbf{1} \in \mathcal{O}_p$
using *Zp.one-closed Zp-nat-mult-closed image-eqI inc-of-int*
by *blast*
thus *?thesis*
using *val-ring-memE*(*1*) **by** *blast*
qed

lemma *to-Zp-nat-inc*:
to-Zp $([(a::nat)]\cdot\mathbf{1}) = [a]\cdot_{Z_p}\mathbf{1}_{Z_p}$
apply(*induction a*)
using *Qp.nat-inc-zero Zp.nat-inc-zero to-Zp-zero* **apply** *presburger*
proof−
fix *a*::*nat* **assume** *A*: *to-Zp* $([a] \cdot \mathbf{1}) = [a]\ \cdot_{Z_p}\ \mathbf{1}_{Z_p}$
have *0*: $[(Suc\ a)]\cdot\mathbf{1} = [a]\cdot\mathbf{1} \oplus \mathbf{1}$
using *Qp.add.nat-pow-Suc* **by** *blast*
have *1*: $[a]\cdot\mathbf{1} \in \mathcal{O}_p$
using *Qp.nat-inc-closed val-of-nat-inc val-ring-memI* **by** *blast*

have *2*: *to-Zp* ([*Suc a*] · **1**) = (*to-Zp* ([*a*]·**1**)) $\oplus_{Z_p}$ $\mathbf{1}_{Z_p}$
using *A 1* **unfolding** *0*
by (*metis Zp.cring-simprules*(*6*) *Zp.nat-inc-closed inc-of-nat inc-of-one inc-of-sum inc-to-Zp sum-closed*)
show *to-Zp* ([*Suc a*] · **1**) = [*Suc a*] $\cdot_{Z_p}$ $\mathbf{1}_{Z_p}$
unfolding *2 A*
using *add-comm nat-suc* **by** *presburger*
qed

lemma *to-Zp-int-neg*:
to-Zp ([(−*int* (*a*::*nat*))]·**1**) = $\ominus_{Z_p}$([*int a*]·$_{Z_p}$**1** $_{Z_p}$)
proof−
have *0*: [(−*int* (*a*::*nat*))]·**1** = $\ominus$ ([*int a*]·**1**)
using *Qp.add.int-pow-neg* **by** *blast*
have *1*: [*int a*]·**1** ∈ $\mathcal{O}_p$
using *Qp.int-inc-closed val-of-int-inc val-ring-memI* **by** *blast*
show *?thesis* **using** *1* **unfolding** *0*
by (*metis Zp.int-inc-closed inc-of-int inc-to-Zp to-Zp-ominus*)
qed

lemma(**in** *ring*) *int-add-pow*:
[*int n*] · **1** = [*n*]·**1**
unfolding *add-pow-def*
by (*simp add*: *int-pow-int*)

lemma *int-add-pow*:
[*int n*] · **1** = [*n*]·**1**
unfolding *add-pow-def*
by (*simp add*: *int-pow-int*)

lemma *Zp-int-add-pow*:
[*int n*] $\cdot_{Z_p}$ $\mathbf{1}_{Z_p}$ = [*n*]·$_{Z_p}$$\mathbf{1}_{Z_p}$
unfolding *add-pow-def*
by (*simp add*: *int-pow-int*)

lemma *to-Zp-int-inc*:
to-Zp ([(*a*::*int*)]·**1**) = ([*a*]·$_{Z_p}$**1** $_{Z_p}$)
apply(*induction a*)
unfolding *int-add-pow Zp-int-add-pow*
using *to-Zp-nat-inc* **apply** *blast*
unfolding *to-Zp-int-neg* **using** *Zp.add.int-pow-neg Zp.one-closed*
by *presburger*

lemma *to-Zp-nat-add-pow*:
assumes *a* ∈ $\mathcal{O}_p$
shows *to-Zp* ([(*n*::*nat*)]·*a*) = [*n*]·$_{Z_p}$ *to-Zp a*
apply(*induction n*)
using *Qp.nat-mult-zero Zp-nat-inc-zero to-Zp-zero* **apply** *presburger*
proof− **fix** *n*::*nat* **assume** *A*: *to-Zp* ([*n*] · *a*) = [*n*] $\cdot_{Z_p}$ *to-Zp a*

have *0*: $[Suc\ n] \cdot a = [n] \cdot a \oplus a$
using *Qp.add.nat-pow-Suc* **by** *blast*
have *1*: *to-Zp* $([n] \cdot a \oplus a) =$ *to-Zp* $([n] \cdot a) \oplus_{Z_p}$ *to-Zp a*
apply(*rule to-Zp-add*[*of* $[n] \cdot a$ *a*])
apply(*induction n*)
using *Qp.nat-mult-zero zero-in-val-ring* **apply** *blast*
unfolding *Qp.add.nat-pow-Suc* **by**(*rule val-ring-add-closed*, *blast*, *rule assms*, *rule assms*)
show *to-Zp* $([Suc\ n] \cdot a) = [Suc\ n] \cdot_{Z_p}$ *to-Zp a*
unfolding *Qp.add.nat-pow-Suc Zp.add.nat-pow-Suc 1 A* **by** *blast*
qed

lemma *val-ring-res*:
assumes $a \in \mathcal{O}_p$
assumes $b \in \mathcal{O}_p$
shows *to-Zp* $(a \ominus b)\ N =$ *to-Zp a N* $\ominus_{Zp\text{-}res\text{-}ring\ N}$ *to-Zp b N*
proof−
have *to-Zp* $(a \ominus b)\ N =$ (*to-Zp a* $\ominus_{Z_p}$ *to-Zp b*) *N*
using *assms to-Zp-minus* **by** *presburger*
then show *?thesis* **using** *assms residue-of-diff to-Zp-closed val-ring-memE*(*2*)
by (*simp add*: *val-ring-memE*(*1*))
qed

lemma *res-diff-in-val-ring-imp-in-val-ring*:
assumes $a \in \mathcal{O}_p$
assumes $b \in$ *carrier* Q_p
assumes $a \ominus b \in \mathcal{O}_p$
shows $b \in \mathcal{O}_p$
proof−
have $b = a \ominus (a \ominus b)$
unfolding *a-minus-def*
using *assms val-ring-memE*(*2*)[*of a*] *Qp.r-neg2* Q_p*-def Qp.minus-sum* **by** *auto*
thus *?thesis* **using** *assms val-ring-minus-closed*[*of a* $a \ominus b$]
by *presburger*
qed

lemma(**in** *padic-fields*) *equal-res-imp-res-diff-zero*:
assumes $a \in \mathcal{O}_p$
assumes $b \in \mathcal{O}_p$
assumes *to-Zp a N* = *to-Zp b N*
shows *to-Zp* $(a \ominus b)\ N = 0$
using *val-ring-res*[*of a b*] *assms*
by (*metis res-diff-zero-fact*′ *to-Zp-closed val-ring-memE*(*2*))

lemma(**in** *padic-fields*) *equal-res-imp-val-diff-bound*:
assumes $a \in \mathcal{O}_p$
assumes $b \in \mathcal{O}_p$
assumes *to-Zp a N* = *to-Zp b N*
shows *val* $(a \ominus b) \geq N$

using *assms equal-res-imp-res-diff-zero*[*of a b N*]
by (*metis to-Zp-closed to-Zp-minus to-Zp-val val-Zp-dist-def val-Zp-dist-res-eq2 val-ring-memE*(*2*) *val-ring-minus-closed*)

lemma(**in** *padic-fields*) *equal-res-equal-val*:
assumes $a \in \mathcal{O}_p$
assumes $b \in \mathcal{O}_p$
assumes *val a* $<$ *N*
assumes *to-Zp a N* $=$ *to-Zp b N*
shows *val a* $=$ *val b*
proof−
have *val* $(a \ominus b) \geq N$
using *assms equal-res-imp-val-diff-bound* **by** *blast*
then have *val* $(a \ominus b) >$ *val a*
using *assms less-le-trans* **by** *blast*
then show *val a* $=$ *val b*
using *assms* **by** (*meson ultrametric-equal-eq*′ *val-ring-memE*)
qed

lemma(**in** *padic-fields*) *val-ring-equal-res-imp-equal-val*:
assumes $a \in \mathcal{O}_p$
assumes $b \in \mathcal{O}_p$
assumes *val a* $<$ *eint N*
assumes *val b* $<$ *eint N*
assumes *to-Zp a N* $=$ *to-Zp b N*
shows *val a* $=$ *val b*
proof−
have *val-Zp* (*to-Zp* $(a \ominus b)) \geq N$
using *assms val-ring-memE to-Zp-closed to-Zp-minus val-Zp-dist-def val-Zp-dist-res-eq2*
by *presburger*
thus *?thesis*
by (*meson assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*5*) *equal-res-equal-val*)
qed

end
end
theory *Padic-Field-Polynomials*
imports *Padic-Fields*

begin

9 p-adic Univariate Polynomials and Hensel's Lemma

type-synonym *padic-field-poly* $=$ *nat* $\Rightarrow$ *padic-number*

type-synonym *padic-field-fun* $=$ *padic-number* $\Rightarrow$ *padic-number*

9.1 Gauss Norms of Polynomials

The Gauss norm of a polynomial is defined to be the minimum valuation of a coefficient of that polynomial. This induces a valuation on the ring of polynomials, and in particular it satisfies the ultrametric inequality. In addition, the Gauss norm of a polynomial $f(x)$ gives a lower bound for the value val $(f(a))$ in terms of val (a), for a point $a \in \mathbb{Q}_p$. We introduce Gauss norms here as a useful tool for stating and proving Hensel's Lemma for the field $\mathbb{Q}_p$. We are abusing terminology slightly in calling this the Gauss norm, rather than the Gauss valuation, but this is just to conform with our decision to work exclusively with the p-adic valuation and not discuss the equivalent real-valued p-adic norm. For a detailed treatment of Gauss norms one can see, for example [2].

context *padic-fields*
begin

no-notation *Zp.to-fun* (**infixl**‹•› *70*)

abbreviation(*input*) Q_p*-x* **where**
Q_p*-x* ≡ *UP* Q_p

definition *gauss-norm* **where**
gauss-norm g = *Min* (*val* ‘ *g* ‘ {..*degree g*})

lemma *gauss-normE*:
assumes *g* ∈ *carrier* Q_p*-x*
shows *gauss-norm g* ≤ *val* (*g k*)
apply(*cases k* ≤ *degree g*)
unfolding *gauss-norm-def*
using *assms* **apply** *auto*[*1*]
proof−
assume ¬ *k* ≤ *degree g*
then have *g k* = $\mathbf{0}_{Q_p}$
by (*simp add*: *UPQ.deg-leE assms*)
then show *Min* (*val* ‘ *g* ‘ {..*deg* Q_p *g*}) ≤ *val* (*g k*)
by (*simp add*: *local.val-zero*)
qed

lemma *gauss-norm-geqI*:
assumes *g* ∈ *carrier* (*UP* Q_p)
assumes ⋀*n*. *val* (*g n*) ≥ *α*
shows *gauss-norm g* ≥ *α*
unfolding *gauss-norm-def* **using** *assms*
by *simp*

lemma *gauss-norm-eqI*:
assumes *g* ∈ *carrier* (*UP* Q_p)
assumes ⋀*n*. *val* (*g n*) ≥ *α*

assumes *val* (*g i*) = α
shows *gauss-norm g* = α
proof−
have *0*: *gauss-norm g* ≤ α
using *assms gauss-normE gauss-norm-def* **by** *fastforce*
have *1*: *gauss-norm g* ≥ α
using *assms gauss-norm-geqI* **by** *auto*
show *?thesis* **using** *0 1* **by** *auto*
qed

lemma *nonzero-poly-nonzero-coeff*:
assumes *g* ∈ *carrier* Q_p-*x*
assumes *g* ≠ $\mathbf{0}_{Q_p}$-*x*
shows ∃ *k*. *k* ≤*degree g* ∧ *g k* ≠$\mathbf{0}_{Q_p}$
proof(*rule ccontr*)
assume ¬ (∃ *k*≤*degree g*. *g k* ≠ $\mathbf{0}_{Q_p}$)
then have *0*: ⋀*k*. *g k* = $\mathbf{0}_{Q_p}$
by (*meson UPQ.deg-leE assms*(*1*) *not-le-imp-less*)
then show *False*
using *assms UPQ.cfs-zero* **by** *blast*
qed

lemma *gauss-norm-prop*:
assumes *g* ∈ *carrier* Q_p-*x*
assumes *g* ≠ $\mathbf{0}_{Q_p}$-*x*
shows *gauss-norm g* ≠ ∞
proof−
obtain *k* **where** *k-def*: *k* ≤*degree g* ∧ *g k* ≠$\mathbf{0}_{Q_p}$
using *assms nonzero-poly-nonzero-coeff*
by *blast*
then have *0*: *gauss-norm g* ≤ *val* (*g k*)
using *assms*(*1*) *gauss-normE* **by** *blast*
have *g k* ∈ *carrier* Q_p
using *UPQ.cfs-closed assms*(*1*) **by** *blast*
hence *val* (*g k*) < ∞
using *k-def assms*
by (*metis eint-ord-code*(*3*) *eint-ord-simps*(*4*) *val-ineq*)
then show *?thesis*
using *0 not-le* **by** *fastforce*
qed

lemma *gauss-norm-coeff-norm*:
∃ *n* ≤ *degree g*. (*gauss-norm g*) = *val* (*g n*)
proof−
have *finite* (*val* ‘ *g* ‘ {..*deg* Q_p *g*})
by *blast*
hence ∃ *x* ∈ (*val* ‘ *g* ‘ {..*deg* Q_p *g*}). *gauss-norm g* = *x*
unfolding *gauss-norm-def*
by *auto*

thus *?thesis* **unfolding** *gauss-norm-def*
by *blast*
qed

lemma *gauss-norm-smult-cfs*:
assumes $g \in$ *carrier* Q_p*-x*
assumes $a \in$ *carrier* Q_p
assumes *gauss-norm* $g = val\ (g\ k)$
shows *gauss-norm* $(a \odot_{Q_p\text{-}x} g) = val\ a + val\ (g\ k)$
proof−
obtain l **where** *l-def*: *gauss-norm* $(a \odot_{Q_p\text{-}x} g) = val\ ((a \odot_{Q_p\text{-}x} g)\ l)$
using *gauss-norm-coeff-norm*
by *blast*
then have *gauss-norm* $(a \odot_{Q_p\text{-}x} g) = val\ (a \otimes_{Q_p} (g\ l))$
using *assms*
by *simp*
then have *gauss-norm* $(a \odot_{Q_p\text{-}x} g) = val\ a + val\ (g\ l)$
by (*simp add*: *UPQ.cfs-closed assms*(*1*) *assms*(*2*) *val-mult*)
then have *0*: *gauss-norm* $(a \odot_{Q_p\text{-}x} g) \leq val\ a + val\ (g\ k)$
using *assms gauss-normE*[*of g l*]
by (*metis UPQ.UP-smult-closed UPQ.cfs-closed UPQ.cfs-smult gauss-normE val-mult*)
have $val\ a + val\ (g\ k) = val\ ((a \odot_{Q_p\text{-}x} g)\ k)$
by (*simp add*: *UPQ.cfs-closed assms*(*1*) *assms*(*2*) *val-mult*)
then have *gauss-norm* $(a \odot_{Q_p\text{-}x} g) \geq val\ a + val\ (g\ k)$
by (*metis* ‹*gauss-norm* $(a \odot_{UP\ Q_p} g) = val\ a + val\ (g\ l)$› *add-left-mono assms*(*1*) *assms*(*3*) *gauss-normE*)
then show *?thesis*
using *0* **by** *auto*
qed

lemma *gauss-norm-smult*:
assumes $g \in$ *carrier* Q_p*-x*
assumes $a \in$ *carrier* Q_p
shows *gauss-norm* $(a \odot_{Q_p\text{-}x} g) = val\ a +$ *gauss-norm* g
using *gauss-norm-smult-cfs*[*of g a*] *gauss-norm-coeff-norm*[*of g*] *assms*
by *metis*

lemma *gauss-norm-ultrametric*:
assumes $g \in$ *carrier* Q_p*-x*
assumes $h \in$ *carrier* Q_p*-x*
shows *gauss-norm* $(g \oplus_{Q_p\text{-}x} h) \geq min\ (gauss\text{-}norm\ g)\ (gauss\text{-}norm\ h)$
proof−
obtain k **where** *gauss-norm* $(g \oplus_{Q_p\text{-}x} h) = val\ ((g \oplus_{Q_p\text{-}x} h)\ k)$
using *gauss-norm-coeff-norm*
by *blast*
then have *0*: *gauss-norm* $(g \oplus_{Q_p\text{-}x} h) = val\ (g\ k \oplus_{Q_p} h\ k)$
by (*simp add*: *assms*(*1*) *assms*(*2*))

have *min (val (g k)) (val (h k))≥ min (gauss-norm g) (gauss-norm h)*
using *gauss-normE[of g k] gauss-normE[of h k] assms(1) assms(2) min.mono*
by *blast*
then show *?thesis*
using *0 val-ultrametric[of g k h k] assms(1) assms(2) dual-order.trans*
by (*metis* (*no-types*, *lifting*) *UPQ.cfs-closed*)
qed

lemma *gauss-norm-a-inv*:
assumes $f \in carrier\ (UP\ Q_p)$
shows $gauss\text{-}norm\ (\ominus_{UP\ Q_p} f) = gauss\text{-}norm\ f$
proof−
have *0*: $\bigwedge n.\ ((\ominus_{UP\ Q_p} f)\ n) = \ominus\ (f\ n)$
using *assms* **by** *simp*
have *1*: $\bigwedge n.\ val\ ((\ominus_{UP\ Q_p} f)\ n) = val\ (f\ n)$
using *0 assms UPQ.UP-car-memE(1) val-minus* **by** *presburger*
obtain *i* **where** *i-def*: $gauss\text{-}norm\ f = val\ (f\ i)$
using *assms gauss-norm-coeff-norm* **by** *blast*
have *2*: $\bigwedge k.\ val\ ((\ominus_{UP\ Q_p} f)\ k) \geq val\ (f\ i)$
unfolding *1*
using *i-def assms gauss-normE* **by** *fastforce*
show *?thesis*
apply(*rule gauss-norm-eqI*[*of - - i*])
apply (*simp add*: *assms*; *fail*)
unfolding *1* **using** *assms gauss-normE* **apply** *blast*
unfolding *i-def* **by** *blast*
qed

lemma *gauss-norm-ultrametric′*:
assumes $f \in carrier\ (UP\ Q_p)$
assumes $g \in carrier\ (UP\ Q_p)$
shows $gauss\text{-}norm\ (f \ominus_{UP\ Q_p} g) \geq min\ (gauss\text{-}norm\ f)\ (gauss\text{-}norm\ g)$
unfolding *a-minus-def*
using *assms gauss-norm-a-inv*[*of g*] *gauss-norm-ultrametric*
by (*metis UPQ.UP-a-inv-closed*)

lemma *gauss-norm-finsum*:
assumes $f \in A \to carrier\ Q_p\text{-}x$
assumes *finite A*
assumes $A \neq \{\}$
shows $gauss\text{-}norm\ (\bigoplus_{Q_p\text{-}x} i{\in}A.\ f\ i) \geq Min\ (gauss\text{-}norm\ `\ (f`A))$
proof−
obtain *k* **where** *k-def*: $val\ ((\bigoplus_{Q_p\text{-}x} i{\in}A.\ f\ i)\ k) = gauss\text{-}norm\ (\bigoplus_{Q_p\text{-}x} i{\in}A.\ f\ i)$
by (*metis gauss-norm-coeff-norm*)
then have *0*: $val\ (\bigoplus_{Q_p} i{\in}A.\ f\ i\ k) \geq Min\ (val\ `\ (\lambda\ i.\ f\ i\ k)\ `\ A)$
using *finsum-val-ultrametric*[*of* $\lambda\ i.\ f\ i\ k\ A$] *assms*
by (*simp add*: ‹$[\![(\lambda i.\ f\ i\ k) \in A \to carrier\ Q_p; finite\ A; A \neq \{\}]\!] \Longrightarrow Min\ (val\ `\ (\lambda i.\ f\ i\ k)\ `\ A) \leq val\ (\bigoplus i{\in}A.\ f\ i\ k)$› *Pi-iff UPQ.cfs-closed*)

have ($\bigwedge a.\ a \in A \Longrightarrow (val \circ (\lambda i.\ f\ i\ k))\ a \geq gauss\text{-}norm\ (f\ a)$)
using *gauss-normE assms*
by (*metis* (*no-types*, *lifting*) *Pi-split-insert-domain Set.set-insert comp-apply*)
then have *Min* (*val* ‘ (λ *i*. *f i k*) ‘ *A*) $\geq$ *Min* ((λ *i*. *gauss-norm* (*f i*)) ‘ *A*)
using *Min-mono'*[*of A*]
by (*simp add*: *assms*(*2*) *image-comp*)
then have *1*: *Min* (*val* ‘ (λ *i*. *f i k*) ‘ *A*) $\geq$ *Min* (*gauss-norm* ‘ *f* ‘ *A*)
by (*metis image-image*)
have $f \in A \to carrier\ (UP\ Q_p) \longrightarrow ((\bigoplus_{Q_p\text{-}x} i \in A.\ f\ i) \in carrier\ Q_p\text{-}x \wedge ((\bigoplus_{Q_p\text{-}x} i \in A.\ f\ i)\ k) = (\bigoplus_{Q_p} i \in A.\ f\ i\ k))$
apply(*rule finite.induct*[*of A*])
apply (*simp add*: *assms*(*2*); *fail*)
apply (*metis* (*no-types*, *lifting*) *Pi-I Qp.add.finprod-one-eqI UPQ.P.finsum-closed UPQ.P.finsum-empty UPQ.cfs-zero empty-iff*)
proof−
fix *a A* **assume** *A*: *finite A f* $\in A \to carrier\ (UP\ Q_p) \longrightarrow$ (*finsum* (*UP Q*$_p$) *f A* $\in$ *carrier* (*UP Q*$_p$) $\wedge$ *finsum* (*UP Q*$_p$) *f A k* = ($\bigoplus i \in A.\ f\ i\ k$))
show *f* $\in$ *insert a A* $\to$ *carrier* (*UP Q*$_p$) $\longrightarrow$ *finsum* (*UP Q*$_p$) *f* (*insert a A*) $\in$ *carrier* (*UP Q*$_p$) $\wedge$ *finsum* (*UP Q*$_p$) *f* (*insert a A*) *k* = ($\bigoplus i \in$ *insert a A*. *f i k*)
apply(*cases a* $\in$ *A*)
using *A*
apply (*simp add*: *insert-absorb*; *fail*)
proof assume *B*: *a* $\notin$ *A f* $\in$ *insert a A* $\to$ *carrier* (*UP Q*$_p$)
then have *f-a*: *f a* $\in$ *carrier* (*UP Q*$_p$)
by *blast*
have *f-A*: *f* $\in$ *A* $\to$ *carrier* (*UP Q*$_p$)
using *B* **by** *blast*
have *finsum* (*UP Q*$_p$) *f* (*insert a A*) = *f a* $\oplus_{UP\ Q_p}$ *finsum* (*UP Q*$_p$) *f A*
using *assms A B f-a f-A finsum-insert* **by** *simp*
then have *0*: *finsum* (*UP Q*$_p$) *f* (*insert a A*) *k* = *f a k* $\oplus_{Q_p}$ (*finsum* (*UP Q*$_p$) *f A*) *k*
using *f-a f-A A B*
by *simp*
have (λ *a*. *f a k*) $\in$ *A* $\to$ *carrier Q*$_p$
proof fix *a* **assume** *a* $\in$ *A*
then have *f a* $\in$ *carrier* (*UP Q*$_p$)
using *f-A* **by** *blast*
then show *f a k* $\in$ *carrier Q*$_p$
using *A cfs-closed* **by** *blast*
qed
then have *0*: *finsum* (*UP Q*$_p$) *f* (*insert a A*) *k* = ($\bigoplus i \in$ *insert a A*. *f i k*)
using *A B Qp.finsum-insert*[*of A a* λ *a*. *f a k*]
by (*simp add*: *UPQ.cfs-closed*)
thus *finsum* (*UP Q*$_p$) *f* (*insert a A*) $\in$ *carrier* (*UP Q*$_p$) $\wedge$ *finsum* (*UP Q*$_p$) *f* (*insert a A*) *k* = ($\bigoplus i \in$ *insert a A*. *f i k*)
using *B*(*2*) *UPQ.P.finsum-closed* **by** *blast*
qed
qed
then have ($\bigoplus_{Q_p\text{-}x} i \in A.\ f\ i) \in carrier\ Q_p\text{-}x \wedge ((\bigoplus_{Q_p\text{-}x} i \in A.\ f\ i)\ k) = (\bigoplus_{Q_p} i \in A.$

f i k)
  using *assms* **by** *blast*
 hence *3*: *gauss-norm* ($\bigoplus_{Q_p\text{-}x} i{\in}A.\ f\ i$) ≥ *Min* (*val* ‘ (λ *i*. *f i k*) ‘ *A*)
  using *0 k-def* **by** *auto*
 thus *?thesis*
  using *1 le-trans* **by** *auto*
qed

lemma *gauss-norm-monom*:
 assumes *a* ∈ *carrier* Q_p
 shows *gauss-norm* (*monom* Q_p*-x a n*) = *val a*
proof−
 have *val* ((*monom* Q_p*-x a n*) *n*) ≥ *gauss-norm* (*monom* Q_p*-x a n*)
  using *assms gauss-normE*[*of monom* Q_p*-x a n n*] *UPQ.monom-closed*
  by *blast*
 then show *?thesis*
  using *gauss-norm-coeff-norm*[*of monom* Q_p*-x a n*] *assms val-ineq UPQ.cfs-monom*
by *fastforce*
qed

lemma *val-val-ring-prod*:
 assumes *a* ∈ $\mathcal{O}_p$
 assumes *b* ∈ *carrier* Q_p
 shows *val* (*a* ⊗$_{Q_p}$ *b*) ≥ *val b*
proof−
 have *0*: *val* (*a* ⊗$_{Q_p}$ *b*) = *val a* + *val b*
  using *assms val-ring-memE*[*of a*] *val-mult*
  by *blast*
 have *1*: *val a* ≥ *0*
  using *assms*
  by (*simp add*: *val-ring-memE*)
 then show *?thesis*
  using *assms 0*
  by *simp*
qed

lemma *val-val-ring-prod′*:
 assumes *a* ∈ $\mathcal{O}_p$
 assumes *b* ∈ *carrier* Q_p
 shows *val* (*b* ⊗$_{Q_p}$ *a*) ≥ *val b*
 using *val-val-ring-prod*[*of a b*]
 by (*simp add*: *Qp.m-comm val-ring-memE assms*(*1*) *assms*(*2*))

lemma *val-ring-nat-pow-closed*:
 assumes *a* ∈ $\mathcal{O}_p$
 shows (*a*[^](*n*::*nat*)) ∈ $\mathcal{O}_p$
 apply(*induction n*)
 apply *auto*[*1*]
 using *Qp.inv-one* Z_p*-mem* **apply** *blast*

by (*metis Qp.nat-pow-Suc Qp.nat-pow-closed val-ring-memE assms image-eqI inc-of-prod to-Zp-closed to-Zp-inc to-Zp-mult*)

lemma *val-ringI*:
assumes $a \in$ *carrier* Q_p
assumes *val* $a \geq 0$
shows $a \in \mathcal{O}_p$
apply(*rule val-ring-val-criterion*)
using *assms* **by** *auto*

notation *UPQ.to-fun* (**infixl**‹•› *70*)

lemma *val-gauss-norm-eval*:
assumes $g \in$ *carrier* Q_p*-x*
assumes $a \in \mathcal{O}_p$
shows *val* $(g \bullet a) \geq$ *gauss-norm g*
proof−
have *0*: $g \bullet a = (\bigoplus_{Q_p} i \in \{..degree\ g\}.\ (g\ i) \otimes_{Q_p} (a[\hat{}]i))$
using *val-ring-memE assms to-fun-formula*[*of g a*] **by** *auto*

have *1*: $(\lambda i.\ g\ i \otimes_{Q_p} (a[\hat{}]i)) \in \{..degree\ g\} \to$ *carrier* Q_p
using *assms*
by (*meson Pi-I val-ring-memE cfs-closed monom-term-car*)
then have *2*: *val* $(g \bullet a) \geq$ *Min* (*val* ‘ $(\lambda\ i.\ ((g\ i) \otimes_{Q_p} (a[\hat{}]i)))$ ‘ $\{..degree\ g\}$)
using *0 finsum-val-ultrametric*[*of* $\lambda\ i.\ ((g\ i) \otimes_{Q_p} (a[\hat{}]i))$ $\{..degree\ g\}$]
by (*metis finite-atMost not-empty-eq-Iic-eq-empty*)
have *3*: $\bigwedge i.$ *val* $((g\ i) \otimes_{Q_p} (a[\hat{}]i)) =$ *val* $(g\ i) +$ *val* $(a[\hat{}]i)$
using *assms val-mult*
by (*simp add*: *val-ring-memE UPQ.cfs-closed*)
have *4*: $\bigwedge i.$ *val* $((g\ i) \otimes_{Q_p} (a[\hat{}]i)) \geq$ *val* $(g\ i)$
proof−
fix *i*
show *val* $((g\ i) \otimes_{Q_p} (a[\hat{}]i)) \geq$ *val* $(g\ i)$
using *val-val-ring-prod'*[*of* $a[\hat{}]i\ g\ i$]
assms(*1*) *assms*(*2*) *val-ring-nat-pow-closed cfs-closed*
by *simp*
qed
have *Min* (*val* ‘ $(\lambda i.\ g\ i \otimes_{Q_p} (a[\hat{}]i))$ ‘ $\{..degree\ g\}) \geq$ *Min* $((\lambda i.$ *val* $(g\ i))$ ‘ $\{..degree\ g\}$)
using *Min-mono'*[*of* $\{..degree\ g\}$ $\lambda i.$ *val* $(g\ i)$ $\lambda i.$ *val* $(g\ i \otimes_{Q_p} (a[\hat{}]i))$] *4 2*
by (*metis finite-atMost image-image*)
then have *Min* (*val* ‘ $(\lambda i.\ g\ i \otimes_{Q_p} (a[\hat{}]i))$ ‘ $\{..degree\ g\}) \geq$ *Min* (*val* ‘ *g* ‘ $\{..degree\ g\}$)
by (*metis image-image*)
then have *val* $(g \bullet a) \geq$ *Min* (*val* ‘ *g* ‘ $\{..degree\ g\}$)
using *2*
by (*meson atMost-iff atMost-subset-iff in-mono*)
then show *?thesis*

by (*simp add*: ‹*val* (*g*⋅*a*) ≥ *Min* (*val* ‘ *g* ‘ {..*degree g*})› *gauss-norm-def*)
qed

lemma *positive-gauss-norm-eval*:
assumes $g \in$ *carrier* Q_p-*x*
assumes *gauss-norm g* ≥ *0*
assumes $a \in \mathcal{O}_p$
shows $(g \cdot a) \in \mathcal{O}_p$
apply(*rule val-ring-val-criterion*[*of g*⋅*a*])
using *assms val-ring-memE*
using *UPQ.to-fun-closed* **apply** *blast*
using *assms val-gauss-norm-eval*[*of g a*] **by** *auto*

lemma *positive-gauss-norm-valuation-ring-coeffs*:
assumes $g \in$ *carrier* Q_p-*x*
assumes *gauss-norm g* ≥ *0*
shows $g\ n \in \mathcal{O}_p$
apply(*rule val-ringI*)
using *cfs-closed assms*(*1*) **apply** *blast*
using *gauss-normE*[*of g n*] *assms* **by** *auto*

lemma *val-ring-cfs-imp-nonneg-gauss-norm*:
assumes $g \in$ *carrier* (*UP* Q_p)
assumes $\bigwedge n.\ g\ n \in \mathcal{O}_p$
shows *gauss-norm g* ≥ *0*
by(*rule gauss-norm-geqI*, *rule assms*, *rule val-ring-memE*, *rule assms*)

lemma *val-of-add-pow*:
assumes $a \in$ *carrier* Q_p
shows *val* ([(*n*::*nat*)]⋅*a*) ≥ *val a*
proof−
have *0*: [(*n*::*nat*)]⋅*a* = ([*n*]⋅**1**)⊗*a*
using *assms Qp.add-pow-ldistr Qp.cring-simprules*(*12*) *Qp.one-closed* **by** *presburger*
have *1*: *val* ([(*n*::*nat*)]⋅*a*) = *val* ([*n*]⋅**1**) + *val a*
unfolding *0* **by**(*rule val-mult*, *simp*, *rule assms*)
show *?thesis* **unfolding** *1* **using** *assms*
by (*simp add*: *val-of-nat-inc*)
qed

lemma *gauss-norm-pderiv*:
assumes $g \in$ *carrier* (*UP* Q_p)
shows *gauss-norm g* ≤ *gauss-norm* (*pderiv g*)
apply(*rule gauss-norm-geqI*)
using *UPQ.pderiv-closed assms* **apply** *blast*
using *gauss-normE pderiv-cfs val-of-add-pow*
by (*smt UPQ.cfs-closed assms dual-order.trans*)

9.2 Mapping Polynomials with Value Ring Coefficients to Polynomials over $\mathbb{Z}_p$

definition *to-Zp-poly* **where**
to-Zp-poly g = (λ*n*. *to-Zp* (*g n*))

lemma *to-Zp-poly-closed*:
assumes *g* ∈ *carrier* Q_p*-x*
assumes *gauss-norm g* ≥ *0*
shows *to-Zp-poly g* ∈ *carrier* (*UP* Z_p)
proof−
have *to-Zp-poly g* ∈ *up* Z_p
apply(*rule mem-upI*)
unfolding *to-Zp-poly-def*
using *cfs-closed*[*of g*] *assms*(*1*) *to-Zp-closed*[*of*] **apply** *blast*
proof−
have ∃ *n*. *bound* $\mathbf{0}_{Q_p}$ *n g*
using *UPQ.deg-leE assms*(*1*) **by** *auto*
then obtain *n* **where** *n-def*: *bound* $\mathbf{0}_{Q_p}$ *n g*
by *blast*
then have *bound* $\mathbf{0}_{Z_p}$ *n* (λ*n*. *to-Zp* (*g n*))
unfolding *bound-def*
by (*simp add*: *to-Zp-zero*)
then show ∃ *n*. *bound* $\mathbf{0}_{Z_p}$ *n* (λ*n*. *to-Zp* (*g n*))
by *blast*
qed
then show *?thesis* **using** *UP-def*[*of* Z_p]
by *simp*
qed

definition *poly-inc* **where**
poly-inc g = (λ*n*::*nat*. ι (*g n*))

lemma *poly-inc-closed*:
assumes *g* ∈ *carrier* (*UP* Z_p)
shows *poly-inc g* ∈ *carrier* Q_p*-x*
proof−
have *poly-inc g* ∈ *up* Q_p
proof(*rule mem-upI*)
show ⋀*n*. *poly-inc g n* ∈ *carrier* Q_p
proof− **fix** *n*
have *g n* ∈ *carrier* Z_p
using *assms UP-def*
by (*simp add*: *UP-def mem-upD*)
then show *poly-inc g n* ∈ *carrier* Q_p
using *assms poly-inc-def*[*of g*] *inc-def*[*of g n*] *inc-closed*
by *force*
qed
show ∃ *n*. *bound* $\mathbf{0}_{Q_p}$ *n* (*poly-inc g*)

```
    proof−
      obtain n where n-def: bound 0_{Z_p} n g
        using assms bound-def[of 0_{Z_p} - g]Zp.cring-axioms UP-cring.deg-leE[of Z_p
g]
        unfolding UP-cring-def
        by metis
      then have bound 0_{Q_p} n (poly-inc g)
        unfolding poly-inc-def bound-def
        by (metis Qp.nat-inc-zero Zp.nat-inc-zero inc-of-nat)
      then show ?thesis by blast
    qed
  qed
  then show ?thesis
    by (simp add: ‹poly-inc g ∈ up Q_p› UP-def)
qed

lemma poly-inc-inverse-right:
  assumes g ∈ carrier (UP Z_p)
  shows to-Zp-poly (poly-inc g) = g
proof−
  have 0: ⋀n. g n ∈ carrier Z_p
    by (simp add: Zp.cfs-closed assms)
  show ?thesis
    unfolding to-Zp-poly-def poly-inc-def
  proof
    fix n
    show to-Zp (ι (g n)) = g n
      using 0 inc-to-Zp
      by auto
  qed
qed

lemma poly-inc-inverse-left:
  assumes g ∈ carrier Q_p-x
  assumes gauss-norm g ≥0
  shows poly-inc (to-Zp-poly g) = g
proof
  fix x
  show poly-inc (to-Zp-poly g) x = g x
    using assms unfolding poly-inc-def to-Zp-poly-def
    by (simp add: positive-gauss-norm-valuation-ring-coeffs to-Zp-inc)
qed

lemma poly-inc-plus:
  assumes f ∈ carrier (UP Z_p)
  assumes g ∈ carrier (UP Z_p)
  shows poly-inc (f ⊕_{UP Z_p} g) = poly-inc f ⊕_{UP Q_p} poly-inc g
proof
  fix n
```

have *0*: *poly-inc* $(f \oplus_{UP\ Z_p} g)\ n = \iota\ (f\ n \oplus_{Z_p} g\ n)$
unfolding *poly-inc-def* **using** *assms* **by** *auto*
have *1*: $(poly\text{-}inc\ f \oplus_{UP\ Q_p} poly\text{-}inc\ g)\ n = poly\text{-}inc\ f\ n \oplus poly\text{-}inc\ g\ n$
by(*rule cfs-add*, *rule poly-inc-closed*, *rule assms*, *rule poly-inc-closed*, *rule assms*)
show $poly\text{-}inc\ (f \oplus_{UP\ Z_p} g)\ n = (poly\text{-}inc\ f \oplus_{UP\ Q_p} poly\text{-}inc\ g)\ n$
unfolding *0 1* **unfolding** *poly-inc-def*
apply(*rule inc-of-sum*)
using *assms* **apply** (*simp add*: *Zp.cfs-closed*; *fail*)
using *assms* **by** (*simp add*: *Zp.cfs-closed*)
qed

lemma *poly-inc-monom*:
assumes $a \in carrier\ Z_p$
shows $poly\text{-}inc\ (monom\ (UP\ Z_p)\ a\ m) = monom\ (UP\ Q_p)\ (\iota\ a)\ m$
proof fix *n*
show $poly\text{-}inc\ (monom\ (UP\ Z_p)\ a\ m)\ n = monom\ (UP\ Q_p)\ (\iota\ a)\ m\ n$
apply(*cases* $m = n$)
using *assms cfs-monom*[*of* $\iota\ a$] *Zp.cfs-monom*[*of a*] **unfolding** *poly-inc-def*
apply (*simp add*: *inc-closed*; *fail*)
using *assms cfs-monom*[*of* $\iota\ a$] *Zp.cfs-monom*[*of a*] **unfolding** *poly-inc-def*
by (*metis Qp.nat-mult-zero Zp-nat-inc-zero inc-closed inc-of-nat*)
qed

lemma *poly-inc-times*:
assumes $f \in carrier\ (UP\ Z_p)$
assumes $g \in carrier\ (UP\ Z_p)$
shows $poly\text{-}inc\ (f \otimes_{UP\ Z_p} g) = poly\text{-}inc\ f \otimes_{UP\ Q_p} poly\text{-}inc\ g$
apply(*rule UP-ring.poly-induct3*[*of* Z_p])
apply (*simp add*: *Zp.is-UP-ring*; *fail*)
using *assms* **apply** *blast*
proof−
fix *p q*
assume *A*: $q \in carrier\ (UP\ Z_p)\ \ p \in carrier\ (UP\ Z_p)$
$poly\text{-}inc\ (f \otimes_{UP\ Z_p} p) = poly\text{-}inc\ f \otimes_{UP\ Q_p} poly\text{-}inc\ p$
$poly\text{-}inc\ (f \otimes_{UP\ Z_p} q) = poly\text{-}inc\ f \otimes_{UP\ Q_p} poly\text{-}inc\ q$
have *0*: $(f \otimes_{UP\ Z_p} (p \oplus_{UP\ Z_p} q)) = (f \otimes_{UP\ Z_p} p) \oplus_{UP\ Z_p} (f \otimes_{UP\ Z_p} q)$
using *assms*(*1*) *A*
by (*simp add*: *Zp.P.r-distr*)
have *1*: $poly\text{-}inc\ (p \oplus_{UP\ Z_p} q) = poly\text{-}inc\ p \oplus_{UP\ Q_p} poly\text{-}inc\ q$
by(*rule poly-inc-plus*, *rule A*, *rule A*)
show $poly\text{-}inc\ (f \otimes_{UP\ Z_p} (p \oplus_{UP\ Z_p} q)) = poly\text{-}inc\ f \otimes_{UP\ Q_p} poly\text{-}inc\ (p \oplus_{UP\ Z_p} q)$
unfolding *0 1* **using** *A poly-inc-closed poly-inc-plus*
by (*simp add*: *UPQ.P.r-distr assms*(*1*))
next
fix *a* **fix** *n*::*nat*
assume *A*: $a \in carrier\ Z_p$
show $poly\text{-}inc\ (f \otimes_{UP\ Z_p} monom\ (UP\ Z_p)\ a\ n) =$

```
        poly-inc f ⊗_{UP Q_p} poly-inc (monom (UP Z_p) a n)
  proof
    fix m
    show poly-inc (f ⊗_{UP Z_p} monom (UP Z_p) a n) m =
        (poly-inc f ⊗_{UP Q_p} poly-inc (monom (UP Z_p) a n)) m
    proof(cases m < n)
      case True
      have T0: (f ⊗_{UP Z_p} monom (UP Z_p) a n) m = 0_{Z_p}
        using True Zp.cfs-monom-mult[of f a m n] A assms
        by blast
      have T1: poly-inc (monom (UP Z_p) a n) =  (monom (UP Q_p) (ι a) n)
        by(rule poly-inc-monom , rule A)
      show ?thesis
        unfolding T0 T1 using True
         by (metis A Q_p-def T0 UPQ.cfs-monom-mult Zp-def assms(1) inc-closed
padic-fields.to-Zp-zero padic-fields-axioms poly-inc-closed poly-inc-def to-Zp-inc zero-in-val-ring)
    next
      case False
      then have F0: m ≥ n
        using False by simp
      have F1: (f ⊗_{UP Z_p} monom (UP Z_p) a n) m = a ⊗_{Z_p} f (m − n)
        using Zp.cfs-monom-mult-l' F0 A assms by simp
      have F2: poly-inc (monom (UP Z_p) a n)  = monom (UP Q_p) (ι a) n
        by(rule poly-inc-monom, rule A)
      have F3: (poly-inc f ⊗_{UP Q_p} poly-inc (monom (UP Z_p) a n)) m
              = (ι a) ⊗ (poly-inc f (m −n))
        using UPQ.cfs-monom-mult-l' F0 A assms poly-inc-closed
        by (simp add: F2 inc-closed)
      show ?thesis
        unfolding F3 unfolding poly-inc-def F1
        apply(rule inc-of-prod, rule A)
        using assms Zp.cfs-closed by blast
    qed
  qed
qed

lemma poly-inc-one:
poly-inc (1_{UP Z_p}) = 1_{UP Q_p}
apply(rule ext)
  unfolding poly-inc-def
  using inc-of-one inc-of-zero
  by simp

lemma poly-inc-zero:
poly-inc (0_{UP Z_p}) = 0_{UP Q_p}
apply(rule ext)
  unfolding poly-inc-def
  using inc-of-one inc-of-zero
```

by *simp*

lemma *poly-inc-hom*:
poly-inc ∈ *ring-hom* (*UP* Z_p) (*UP* Q_p)
apply(*rule ring-hom-memI*)
apply(*rule poly-inc-closed*, *blast*)
apply(*rule poly-inc-times*, *blast*, *blast*)
apply(*rule poly-inc-plus*, *blast*, *blast*)
by(*rule poly-inc-one*)

lemma *poly-inc-as-poly-lift-hom*:
assumes *f* ∈ *carrier* (*UP* Z_p)
shows *poly-inc f* = *poly-lift-hom* Z_p Q_p ι *f*
apply(*rule ext*)
unfolding *poly-inc-def*
using *Zp.poly-lift-hom-cf*[*of* Q_p ι *f*] *assms UPQ.R-cring local.inc-is-hom*
by *blast*

lemma *poly-inc-eval*:
assumes *g* ∈ *carrier* (*UP* Z_p)
assumes *a* ∈ *carrier* Z_p
shows *to-function* Q_p (*poly-inc g*) (ι *a*) = ι (*to-function* Z_p *g a*)
proof−
have *0*: *poly-inc g* = *poly-lift-hom* Z_p Q_p ι *g*
using *assms poly-inc-as-poly-lift-hom*[*of g*] **by** *blast*
have *1*: *to-function* Q_p (*poly-lift-hom* Z_p Q_p ι *g*) (ι *a*) = ι (*to-function* Z_p *g a*)
using *Zp.poly-lift-hom-eval*[*of* Q_p ι *g a*] *assms inc-is-hom*
unfolding *to-fun-def Zp.to-fun-def*
using *UPQ.R-cring* **by** *blast*
show *?thesis* **unfolding** *0 1*
by *blast*
qed

lemma *val-ring-poly-eval*:
assumes *f* ∈ *carrier* (*UP* Q_p)
assumes ⋀ *i*. *f i* ∈ $\mathcal{O}_p$
shows ⋀*x*. *x* ∈ $\mathcal{O}_p$ ⟹ *f* ⋅ *x* ∈ $\mathcal{O}_p$
apply(*rule positive-gauss-norm-eval*, *rule assms*)
apply(*rule val-ring-cfs-imp-nonneg-gauss-norm*)
using *assms* **by** *auto*

lemma *Zp-res-of-pow*:
assumes *a* ∈ *carrier* Z_p
assumes *b* ∈ *carrier* Z_p
assumes *a n* = *b n*
shows (*a*[^]$_{Z_p}$(*k*::*nat*)) *n* = (*b*[^]$_{Z_p}$(*k*::*nat*)) *n*
apply(*induction k*)
using *assms Group.nat-pow-0 to-Zp-one* **apply** *metis*
using *Zp.geometric-series-id*[*of a b*] *Zp-residue-mult-zero*(*1*) *assms*(*1*) *assms*(*2*)

assms(*3*)
pow-closed res-diff-zero-fact'' res-diff-zero-fact(*1*) **by** *metis*

lemma *to-Zp-nat-pow*:
assumes $a \in \mathcal{O}_p$
shows *to-Zp* $(a[\hat{}](n::nat)) = (to\text{-}Zp\ a)[\hat{}]_{Z_p}(n::nat)$
apply(*induction n*)
using *assms Group.nat-pow-0 to-Zp-one* **apply** *metis*
using *assms to-Zp-mult*[*of a*] *Qp.m-comm Qp.nat-pow-Suc val-ring-memE pow-suc to-Zp-closed val-ring-nat-pow-closed*
by *metis*

lemma *to-Zp-res-of-pow*:
assumes $a \in \mathcal{O}_p$
assumes $b \in \mathcal{O}_p$
assumes *to-Zp a n* = *to-Zp b n*
shows *to-Zp* $(a[\hat{}](k::nat))\ n = to\text{-}Zp\ (b[\hat{}](k::nat))\ n$
using *assms val-ring-memE Zp-res-of-pow to-Zp-closed to-Zp-nat-pow* **by** *presburger*

lemma *poly-eval-cong*:
assumes $g \in carrier\ (UP\ Q_p)$
assumes $\bigwedge i.\ g\ i \in \mathcal{O}_p$
assumes $a \in \mathcal{O}_p$
assumes $b \in \mathcal{O}_p$
assumes *to-Zp a k* = *to-Zp b k*
shows *to-Zp* $(g \cdot a)\ k = to\text{-}Zp\ (g \cdot b)\ k$
proof−
have $(\forall i.\ g\ i \in \mathcal{O}_p) \longrightarrow to\text{-}Zp\ (g \cdot a)\ k = to\text{-}Zp\ (g \cdot b)\ k$
proof(*rule UPQ.poly-induct*[*of g*])
show $g \in carrier\ (UP\ Q_p)$
using *assms* **by** *blast*
show $\bigwedge p.\ p \in carrier\ (UP\ Q_p) \Longrightarrow deg\ Q_p\ p = 0 \Longrightarrow (\forall i.\ p\ i \in \mathcal{O}_p) \longrightarrow to\text{-}Zp\ (p \cdot a)\ k = to\text{-}Zp\ (p \cdot b)\ k$
proof fix *p* **assume** *A*: $p \in carrier\ (UP\ Q_p)\ deg\ Q_p\ p = 0\ \forall i.\ p\ i \in \mathcal{O}_p$
obtain *c* **where** *c-def*: $c \in carrier\ Q_p \wedge p = up\text{-}ring.monom\ (UP\ Q_p)\ c\ 0$
using *A*
by (*metis UPQ.zcf-degree-zero UPQ.cfs-closed UPQ.trms-of-deg-leq-0 UPQ.trms-of-deg-leq-degree-f*)
have *p-eq*: $p = up\text{-}ring.monom\ (UP\ Q_p)\ c\ 0$
using *c-def* **by** *blast*
have *p-cfs*: *p 0* = *c*
unfolding *p-eq* **using** *c-def UP-ring.cfs-monom*[*of* Q_p *c 0 0*] *UPQ.P-is-UP-ring*
by *presburger*
have *c-closed*: $c \in \mathcal{O}_p$
using *p-cfs A*(*3*) **by** *blast*
have *0*: $(p \cdot a) = c$
unfolding *p-eq* **using** *c-def assms* **by** (*meson UPQ.to-fun-const val-ring-memE*(*2*))
have *1*: $(p \cdot b) = c$
unfolding *p-eq* **using** *c-def assms UPQ.to-fun-const val-ring-memE*(*2*) **by**

presburger
show *to-Zp* $(p \cdot a)\ k =$ *to-Zp* $(p \cdot b)\ k$
unfolding *0 1* **by** *blast*
qed
show $\bigwedge p.\ (\bigwedge q.\ q \in$ *carrier* $(UP\ Q_p) \Longrightarrow$ *deg* $Q_p\ q <$ *deg* $Q_p\ p \Longrightarrow (\forall\, i.\ q\ i \in \mathcal{O}_p) \longrightarrow$ *to-Zp* $(q \cdot a)\ k =$ *to-Zp* $(q \cdot b)\ k) \Longrightarrow$
$p \in$ *carrier* $(UP\ Q_p) \Longrightarrow 0 <$ *deg* $Q_p\ p \Longrightarrow (\forall\, i.\ p\ i \in \mathcal{O}_p) \longrightarrow$ *to-Zp* $(p \cdot a)\ k =$ *to-Zp* $(p \cdot b)\ k$
proof
fix p **assume** A: $(\bigwedge q.\ q \in$ *carrier* $(UP\ Q_p) \Longrightarrow$ *deg* $Q_p\ q <$ *deg* $Q_p\ p \Longrightarrow (\forall\, i.\ q\ i \in \mathcal{O}_p) \longrightarrow$ *to-Zp* $(q \cdot a)\ k =$ *to-Zp* $(q \cdot b)\ k)$
$p \in$ *carrier* $(UP\ Q_p)\ 0 <$ *deg* $Q_p\ p \quad \forall\, i.\ p\ i \in \mathcal{O}_p$
obtain q **where** *q-def*: $q \in$ *carrier* $(UP\ Q_p) \wedge$ *deg* $Q_p\ q <$ *deg* $Q_p\ p \wedge p =$ *UPQ.ltrm* $p \oplus_{UP\ Q_p} q$
by (*metis A(2) A(3) UPQ.ltrm-closed UPQ.ltrm-decomp UPQ.UP-a-comm*)
have *0*: $\bigwedge i.\ \ p\ i = q\ i \oplus$ *UPQ.ltrm* $p\ i$
using *q-def A*
by (*metis Qp.a-ac(2) UPQ.ltrm-closed UPQ.UP-car-memE(1) UPQ.cfs-add*)
have *1*: $\forall\, i.\ q\ i \in \mathcal{O}_p$
proof fix i
show $q\ i \in \mathcal{O}_p$
apply(*cases* $i <$ *deg* $Q_p\ p$)
using *0*[*of i*] *A(4) A(2) q-def*
using *UPQ.ltrm-closed UPQ.P.a-ac(2) UPQ.trunc-cfs UPQ.trunc-closed UPQ.trunc-simps(1)*
apply (*metis Qp.r-zero UPQ.ltrm-cfs UPQ.cfs-closed UPQ.deg-leE*)
using *q-def*
by (*metis* (*no-types, opaque-lifting*) *A(2) A(4) UPQ.P.add.m-closed UPQ.coeff-of-sum-diff-degree0 UPQ.deg-leE UPQ.equal-deg-sum UPQ.equal-deg-sum'* ‹$\bigwedge$*thesis*. $(\bigwedge q.\ q \in$ *carrier* $(UP\ Q_p) \wedge$ *deg* $Q_p\ q <$ *deg* $Q_p\ p \wedge p =$ *up-ring.monom* $(UP\ Q_p)\ (p\ (deg\ Q_p\ p))\ (deg\ Q_p\ p) \oplus_{UP\ Q_p} q \Longrightarrow$ *thesis*) $\Longrightarrow$ *thesis*› *lessI linorder-neqE-nat*)
qed
have *2*: *UPQ.lcf* $p \in \mathcal{O}_p$
using *A(4)* **by** *blast*
have *3*: *UPQ.ltrm* $p \cdot a =$ *UPQ.lcf* $p \otimes a[\hat{}]$ *deg* $Q_p\ p$
apply(*rule UP-cring.to-fun-monom*) **unfolding** *UP-cring-def*
apply (*simp add*: *UPQ.R-cring*)
apply (*simp add*: *A(2) UPQ.cfs-closed*)
using *assms(3) val-ring-memE(2)* **by** *blast*
have *4*: *UPQ.ltrm* $p \cdot b =$ *UPQ.lcf* $p \otimes b[\hat{}]$ *deg* $Q_p\ p$
apply(*rule UP-cring.to-fun-monom*) **unfolding** *UP-cring-def*
apply (*simp add*: *UPQ.R-cring*)
apply (*simp add*: *A(2) UPQ.cfs-closed*)
using *assms val-ring-memE(2)* **by** *blast*
have *p-eq*: $p = q \oplus_{UP\ Q_p}$ *UPQ.ltrm* p
using *q-def* **by** (*metis A(2) UPQ.ltrm-closed UPQ.UP-a-comm*)
have *5*: $p \cdot a = q \cdot a \oplus$ *UPQ.lcf* $p \otimes a[\hat{}]$ *deg* $Q_p\ p$
using *assms val-ring-memE(2) p-eq q-def UPQ.to-fun-plus*[*of q UPQ.ltrm p*

a]
by (*metis 3 A(2) UPQ.ltrm-closed UPQ.to-fun-plus*)
have *6*: $p \cdot b = q \cdot b \oplus$ *UPQ.lcf p* $\otimes$ *b*[$\widehat{}$] *deg* Q_p *p*
using *assms val-ring-memE(2) p-eq q-def UPQ.to-fun-plus*[*of q UPQ.ltrm p a*]
by (*metis 4 A(2) UPQ.ltrm-closed UPQ.to-fun-plus*)
have *7*: *UPQ.lcf p* $\otimes$ *b*[$\widehat{}$] *deg* Q_p *p* $\in \mathcal{O}_p$
apply(*rule val-ring-times-closed*)
using *2* **apply** *linarith*
by(*rule val-ring-nat-pow-closed*, *rule assms*)
have *8*: *UPQ.lcf p* $\otimes$ *a*[$\widehat{}$] *deg* Q_p *p* $\in \mathcal{O}_p$
apply(*rule val-ring-times-closed*)
using *2* **apply** *linarith*
by(*rule val-ring-nat-pow-closed*, *rule assms*)
have *9*: $q \cdot a \in \mathcal{O}_p$
using *q-def 1 assms*(*3*) *val-ring-poly-eval* **by** *blast*
have *10*: $q \cdot b \in \mathcal{O}_p$
using *q-def 1 assms*(*4*) *val-ring-poly-eval* **by** *blast*
have *11*: *to-Zp* ($p \cdot a$) = *to-Zp* ($q \cdot a$) $\oplus_{Z_p}$ *to-Zp* (*UPQ.ltrm p* $\cdot$ *a*)
using *5 8 9 to-Zp-add 3* **by** *presburger*
have *12*: *to-Zp* ($p \cdot b$) = *to-Zp* ($q \cdot b$) $\oplus_{Z_p}$ *to-Zp* (*UPQ.ltrm p* $\cdot$ *b*)
using *6 10 7 to-Zp-add 4* **by** *presburger*
have *13*: *to-Zp* ($p \cdot a$) *k* = *to-Zp* ($q \cdot a$) *k* $\oplus_{Zp\text{-}res\text{-}ring\ k}$ *to-Zp* (*UPQ.ltrm p* $\cdot$ *a*) *k*
unfolding *11* **using** *residue-of-sum* **by** *blast*
have *14*: *to-Zp* ($p \cdot b$) *k* = *to-Zp* ($q \cdot b$) *k* $\oplus_{Zp\text{-}res\text{-}ring\ k}$ *to-Zp* (*UPQ.ltrm p* $\cdot$ *b*) *k*
unfolding *12* **using** *residue-of-sum* **by** *blast*
have *15*: *to-Zp* (*UPQ.ltrm p* $\cdot$ *a*) *k* = *to-Zp* (*UPQ.ltrm p* $\cdot$ *b*) *k*
proof(*cases k = 0*)
case *True*
have *T0*: *to-Zp* (*UPQ.ltrm p* $\cdot$ *a*) $\in$ *carrier* Z_p
unfolding *3* **using** *8 to-Zp-closed val-ring-memE(2)* **by** *blast*
have *T1*: *to-Zp* (*UPQ.ltrm p* $\cdot$ *b*) $\in$ *carrier* Z_p
unfolding *4* **using** *7 to-Zp-closed val-ring-memE(2)* **by** *blast*
show *?thesis* **unfolding** *True* **using** *T0 T1 padic-integers.p-res-ring-0*
by (*metis p-res-ring-0' residues-closed*)
next
case *False*
have *k-pos*: *k* > *0*
using *False* **by** *presburger*
have *150*: *to-Zp* (*p* (*deg* Q_p *p*) $\otimes$ *a* [$\widehat{}$] *deg* Q_p *p*) = *to-Zp* (*p* (*deg* Q_p *p*)) $\otimes_{Z_p}$ *to-Zp*(*a* [$\widehat{}$] *deg* Q_p *p*)
apply(*rule to-Zp-mult*)
using *2* **apply** *blast*
by(*rule val-ring-nat-pow-closed*, *rule assms*)
have *151*: *to-Zp* (*p* (*deg* Q_p *p*) $\otimes$ *b* [$\widehat{}$] *deg* Q_p *p*) = *to-Zp* (*p* (*deg* Q_p *p*)) $\otimes_{Z_p}$ *to-Zp*(*b* [$\widehat{}$] *deg* Q_p *p*)
apply(*rule to-Zp-mult*)

```
        using 2 apply blast
       by(rule val-ring-nat-pow-closed, rule assms)
      have 152: to-Zp (p (deg Q_p p) ⊗ a [^] deg Q_p p) k = to-Zp (p (deg Q_p p))
k ⊗_Zp-res-ring k to-Zp( a [^] deg Q_p p) k
       unfolding 150 using residue-of-prod by blast
      have 153: to-Zp (p (deg Q_p p) ⊗ b [^] deg Q_p p) k = to-Zp (p (deg Q_p p))
k ⊗_Zp-res-ring k to-Zp( b [^] deg Q_p p) k
       unfolding 151 using residue-of-prod by blast
      have 154: to-Zp( a [^] deg Q_p p) k = to-Zp a k [^]_Zp-res-ring k deg Q_p p
      proof−
      have 01: ⋀m::nat. to-Zp (a[^]m) k = to-Zp a k [^]_Zp-res-ring k m
      proof−
       fix m::nat show to-Zp (a [^] m) k = to-Zp a k [^]_Zp-res-ring k m
      proof−
       have 00: to-Zp (a[^]m) = to-Zp a [^]_Z_p m
       using assms to-Zp-nat-pow[of a m] by blast
      have 01: to-Zp a ∈ carrier Z_p
       using assms to-Zp-closed val-ring-memE(2) by blast
      have 02: to-Zp a k ∈ carrier (Zp-res-ring k)
       using 01 residues-closed by blast
      have 03: cring (Zp-res-ring k)
       using k-pos padic-integers.R-cring padic-integers-axioms by blast
      have 01: (to-Zp a [^]_Z_p m) k = (to-Zp a) k [^]_Zp-res-ring k m
       apply(induction m)
       using 01 02 apply (metis Group.nat-pow-0 k-pos residue-of-one(1))
       using residue-of-prod[of to-Zp a [^]_Z_p m to-Zp a k] 01 02 03
      proof −
       fix ma :: nat
       assume (to-Zp a [^]_Z_p ma) k = to-Zp a k [^]_Zp-res-ring k ma
       then show (to-Zp a [^]_Z_p Suc ma) k = to-Zp a k [^]_Zp-res-ring k Suc ma
         by (metis (no-types) Group.nat-pow-Suc residue-of-prod)
      qed
      show ?thesis unfolding 00 01 by blast
      qed
      qed
      thus ?thesis by blast
      qed
      have 155: to-Zp( b [^] deg Q_p p) k = to-Zp b k [^]_Zp-res-ring k deg Q_p p
       using assms by (metis 154 to-Zp-res-of-pow)
      show ?thesis
       unfolding 3 4 152 153 154 155 assms by blast
    qed
    show to-Zp (p · a) k = to-Zp (p · b) k
     unfolding 13 14 15 using A 1 q-def by presburger
  qed
 qed
 thus ?thesis using assms by blast
qed
```

lemma *to-Zp-poly-eval*:
assumes $g \in$ *carrier* Q_p-*x*
assumes *gauss-norm* $g \geq 0$
assumes $a \in \mathcal{O}_p$
shows *to-Zp* (*to-function* Q_p *g a*) = *to-function* Z_p (*to-Zp-poly g*) (*to-Zp a*)
proof−
obtain *h* **where** *h-def*: *h* = *to-Zp-poly g*
by *blast*
obtain *b* **where** *b-def*: *b* = *to-Zp a*
by *blast*
have *h-poly-inc*: *poly-inc h* = *g*
unfolding *h-def* **using** *assms*
by (*simp add*: *poly-inc-inverse-left*)
have *b-inc*: *ι b* = *a*
unfolding *b-def* **using** *assms*
by (*simp add*: *to-Zp-inc*)
have *h-closed*: *h* ∈ *carrier* (*UP* Z_p)
unfolding *h-def* **using** *assms*
by (*simp add*: *to-Zp-poly-closed*)
have *b-closed*: *b* ∈ *carrier* Z_p
unfolding *b-def* **using** *assms*
by (*simp add*: *to-Zp-closed val-ring-memE*)
have *0*: *to-function* Q_p (*poly-inc h*) (*ι b*) = *ι* (*to-function* Z_p *h b*)
apply(*rule poly-inc-eval*)
using *h-def assms* **apply** (*simp add*: *to-Zp-poly-closed*; *fail*)
unfolding *b-def* **using** *assms*
by (*simp add*: *to-Zp-closed val-ring-memE*)
have *1*: *to-Zp* (*to-function* Q_p (*poly-inc h*) (*ι b*)) = *to-function* Z_p *h b*
unfolding *0*
using *h-closed b-closed Zp.to-fun-closed Zp.to-fun-def inc-to-Zp* **by** *auto*
show *?thesis*
using *1* **unfolding** *h-poly-inc b-inc*
unfolding *h-def b-def* **by** *blast*
qed

lemma *poly-eval-equal-val*:
assumes $g \in$ *carrier* (*UP* Q_p)
assumes $\bigwedge x.\ g\ x \in \mathcal{O}_p$
assumes $a \in \mathcal{O}_p$
assumes $b \in \mathcal{O}_p$
assumes *val* (*g* ⋅ *a*) < *eint n*
assumes *to-Zp a n* = *to-Zp b n*
shows *val* (*g* ⋅ *b*) = *val* (*g* ⋅ *a*)
proof−
have (∀ *x*. *g x* ∈ $\mathcal{O}_p$) ⟶ *to-Zp* (*g* ⋅ *b*) *n* = *to-Zp* (*g* ⋅ *a*) *n*
proof(*rule poly-induct*[*of g*])
show *g* ∈ *carrier* (*UP* Q_p)
by (*simp add*: *assms*(*1*))
show $\bigwedge p.$ *p* ∈ *carrier* (*UP* Q_p) ⟹ *deg* Q_p *p* = *0* ⟹ (∀ *x*. *p x* ∈ $\mathcal{O}_p$) ⟶

to-Zp $(p \cdot b)$ n = *to-Zp* $(p \cdot a)$ n
proof fix p **assume** A: $p \in$ *carrier* $(UP\ Q_p)$ *deg* Q_p $p = 0$ $\forall x.\ p\ x \in \mathcal{O}_p$
show *to-Zp* $(p \cdot b)$ n = *to-Zp* $(p \cdot a)$ n
using A **by** (*metis val-ring-memE UPQ.to-fun-ctrm UPQ.trms-of-deg-leq-0 UPQ.trms-of-deg-leq-degree-f assms*(3) *assms*(4))
qed
show $\bigwedge p.$ $(\bigwedge q.\ q \in$ *carrier* $(UP\ Q_p) \Longrightarrow$ *deg* Q_p $q <$ *deg* Q_p $p \Longrightarrow (\forall x.\ q\ x \in \mathcal{O}_p) \longrightarrow$ *to-Zp* $(q \cdot b)$ n = *to-Zp* $(q \cdot a)$ $n) \Longrightarrow$
$p \in$ *carrier* $(UP\ Q_p) \Longrightarrow 0 <$ *deg* Q_p $p \Longrightarrow (\forall x.\ p\ x \in \mathcal{O}_p) \longrightarrow$ *to-Zp* $(p \cdot b)$ n = *to-Zp* $(p \cdot a)$ n
proof fix p **assume** *IH*: $(\bigwedge q.\ q \in$ *carrier* $(UP\ Q_p) \Longrightarrow$ *deg* Q_p $q <$ *deg* Q_p $p \Longrightarrow (\forall x.\ q\ x \in \mathcal{O}_p) \longrightarrow$ *to-Zp* $(q \cdot b)$ n = *to-Zp* $(q \cdot a)$ $n)$
assume A: $p \in$ *carrier* $(UP\ Q_p)$ $0 <$ *deg* Q_p p $\forall x.\ p\ x \in \mathcal{O}_p$
show *to-Zp* $(p \cdot b)$ n = *to-Zp* $(p \cdot a)$ n
proof−
obtain q **where** *q-def*: $q \in$ *carrier* $(UP\ Q_p) \wedge$ *deg* Q_p $q <$ *deg* Q_p $p \wedge$
$p = q \oplus_{UP\ Q_p}$ *ltrm* p
using A **by** (*meson UPQ.ltrm-decomp*)
have *p-eq*: $p = q \oplus_{UP\ Q_p}$ *ltrm* p
using *q-def* **by** *blast*
have $\forall x.\ q\ x \in \mathcal{O}_p$ **proof fix** x
have *px*: $p\ x = (q \oplus_{UP\ Q_p}$ *ltrm* $p)\ x$
using *p-eq* **by** *simp*
show $q\ x \in \mathcal{O}_p$
proof(*cases* $x \leq$ *deg* Q_p q)
case *True*
then have $p\ x = q\ x$
unfolding *px* **using** *q-def A*
by (*smt UPQ.ltrm-closed UPQ.P.add.right-cancel UPQ.coeff-of-sum-diff-degree0 UPQ.deg-ltrm UPQ.trunc-cfs UPQ.trunc-closed UPQ.trunc-simps*(1) *less-eq-Suc-le nat-neq-iff not-less-eq-eq*)
then show *?thesis* **using** A
by *blast*
next
case *False*
then show *?thesis*
using *q-def UPQ.deg-eqI eq-imp-le nat-le-linear zero-in-val-ring*
by (*metis* (*no-types*, *lifting*) *UPQ.coeff-simp UPQ.deg-belowI*)
qed
qed
then have *0*: *to-Zp* $(q \cdot b)$ n = *to-Zp* $(q \cdot a)$ n
using *IH q-def* **by** *blast*
have *1*: *to-Zp* $(ltrm\ p \cdot b)$ n = *to-Zp* $(ltrm\ p \cdot a)$ n
proof−
have *10*: $(ltrm\ p \cdot b) = (p\ (deg\ Q_p\ p)) \otimes b[\widehat{\ }]\ (deg\ Q_p\ p)$
using *assms A* **by** (*meson val-ring-memE UPQ.to-fun-monom*)
have *11*: $(ltrm\ p \cdot a) = (p\ (deg\ Q_p\ p)) \otimes a[\widehat{\ }]\ (deg\ Q_p\ p)$
using *assms A* **by** (*meson val-ring-memE UPQ.to-fun-monom*)
have *12*: *to-Zp* $(b[\widehat{\ }]\ (deg\ Q_p\ p))$ n = *to-Zp* $(a[\widehat{\ }]\ (deg\ Q_p\ p))$ n

using *to-Zp-res-of-pow assms* **by** *metis*
have *13*: $p\ (deg\ Q_p\ p) \in \mathcal{O}_p$
using *A(3)* **by** *blast*
have *14*: $b[\widehat{}]\ (deg\ Q_p\ p) \in \mathcal{O}_p$
using *assms(4) val-ring-nat-pow-closed* **by** *blast*
have *15*: $a[\widehat{}]\ (deg\ Q_p\ p) \in \mathcal{O}_p$
using *assms(3) val-ring-nat-pow-closed* **by** *blast*
have *16*: $(ltrm\ p \cdot b) \in \mathcal{O}_p$
by (*simp add*: *10 13 14 val-ring-times-closed*)
have *17*: *to-Zp* $(ltrm\ p \cdot b)\ n =$ *to-Zp* $(p\ (deg\ Q_p\ p))\ n \otimes_{Zp\text{-}res\text{-}ring\ n}$ *to-Zp* $(b[\widehat{}]\ (deg\ Q_p\ p))\ n$
using *10 13 14 15 16 assms residue-of-prod to-Zp-mult* **by** *presburger*
have *18*: $(ltrm\ p \cdot a) \in \mathcal{O}_p$
by (*simp add*: *11 15 A(3) val-ring-times-closed*)
have *19*: *to-Zp* $(ltrm\ p \cdot a)\ n =$ *to-Zp* $(p\ (deg\ Q_p\ p))\ n \otimes_{Zp\text{-}res\text{-}ring\ n}$ *to-Zp* $(a[\widehat{}]\ (deg\ Q_p\ p))\ n$
using *10 13 14 15 16 17 18 assms residue-of-prod to-Zp-mult 11* **by** *presburger*
show *?thesis* **using** *12 17 19* **by** *presburger*
qed
have *2*: $p\ (deg\ Q_p\ p) \in \mathcal{O}_p$
using *A(3)* **by** *blast*
have *3*: $(ltrm\ p \cdot b) \in \mathcal{O}_p$
using *2 assms*
by (*metis A(1) Q$_p$-def val-ring-memE val-ring-memE UPQ.ltrm-closed Zp-def ι-def gauss-norm-monom padic-fields.positive-gauss-norm-eval padic-fields-axioms*)
have *4*: $(ltrm\ p \cdot a) \in \mathcal{O}_p$
using *2 assms*
by (*metis A(1) Q$_p$-def val-ring-memE val-ring-memE UPQ.ltrm-closed Zp-def ι-def gauss-norm-monom padic-fields.positive-gauss-norm-eval padic-fields-axioms*)
have *5*: $(q \cdot b) \in \mathcal{O}_p$
using ‹$\forall x.\ q\ x \in \mathcal{O}_p$› *assms(4) q-def*
by (*metis gauss-norm-coeff-norm positive-gauss-norm-eval val-ring-memE(1)*)
have *6*: $(q \cdot a) \in \mathcal{O}_p$
using ‹$\forall x.\ q\ x \in \mathcal{O}_p$› *assms(3) q-def*
by (*metis gauss-norm-coeff-norm positive-gauss-norm-eval val-ring-memE(1)*)
have *7*: *to-Zp* $(p \cdot b) =$ *to-Zp* $(ltrm\ p \cdot b)\ \oplus_{Z_p}$ *to-Zp* $(q \cdot b)$
using *5 3 q-def* **by** (*metis* (*no-types*, *lifting*) *A(1) val-ring-memE UPQ.ltrm-closed UPQ.to-fun-plus add-comm assms(4) to-Zp-add*)
have *8*: *to-Zp* $(p \cdot a) =$ *to-Zp* $(ltrm\ p \cdot a)\ \oplus_{Z_p}$ *to-Zp* $(q \cdot a)$
using *4 6 q-def* **by** (*metis* (*no-types*, *lifting*) *A(1) val-ring-memE UPQ.ltrm-closed UPQ.to-fun-plus add-comm assms(3) to-Zp-add*)
have *9*: *to-Zp* $(p \cdot b) \in$ *carrier* Z_p
using *A assms* **by** (*meson val-ring-memE UPQ.to-fun-closed to-Zp-closed*)
have *10*: *to-Zp* $(p \cdot a) \in$ *carrier* Z_p
using *A assms val-ring-memE UPQ.to-fun-closed to-Zp-closed* **by** *presburger*
have *11*: *to-Zp* $(p \cdot b)\ n =$ *to-Zp* $(ltrm\ p \cdot b)\ n\ \oplus_{Zp\text{-}res\text{-}ring\ n}$ *to-Zp* $(q \cdot$

```
b) n
        using 7 9 5 3 residue-of-sum by presburger
      have 12: to-Zp (p • a) n = to-Zp (ltrm p • a) n ⊕_{Zp-res-ring n} to-Zp (q • a)
n
        using 8 6 4 residue-of-sum by presburger
       show ?thesis using 0 11 12 q-def assms
        using 1 by presburger
     qed
   qed
  qed
  have (∀ x. g x ∈ 𝒪_p)
   using assms by blast
  hence 0: to-Zp (g • b) n = to-Zp (g • a) n
   using ‹(∀ x. g x ∈ 𝒪_p) ⟶ to-Zp (g • b) n = to-Zp (g • a) n› by blast
  have 1: g • a ∈ 𝒪_p
   using  assms(1) assms(2) assms(3)
   by (metis gauss-norm-coeff-norm positive-gauss-norm-eval val-ring-memE(1))
  have 2: g • b ∈ 𝒪_p
   using  assms(1) assms(2) assms(4)
   by (metis gauss-norm-coeff-norm positive-gauss-norm-eval val-ring-memE(1))
  have 3: val (g • b) < eint n
  proof−
   have P0: to-Zp (g • a) ∈ carrier Z_p
    using 1 val-ring-memE to-Zp-closed by blast
   have P1: to-Zp (g • b) ∈ carrier Z_p
    using 2 val-ring-memE to-Zp-closed by blast
   have P2: val-Zp (to-Zp (g • a)) < n
    using 1 assms to-Zp-val by presburger
   have P3: to-Zp (g • a) ≠ 0_{Z_p}
    using P2 P0 unfolding val-Zp-def    by (metis P2 infinity-ilessE val-Zp-def)
   have P4: (to-Zp (g • a)) n ≠ 0
    using 1 P2 P3 above-ord-nonzero[of to-Zp (g • a) n]
    by (metis P0 eint.inject less-eintE val-ord-Zp)
   then have to-Zp (g • b) n ≠ 0
    using 0 by linarith
   then have val-Zp (to-Zp (g • b)) < n
    using P1 P0
    by (smt below-val-Zp-zero eint-ile eint-ord-simps(1) eint-ord-simps(2) nonzero-imp-ex-nonzero-res
residue-of-zero(2) zero-below-val-Zp)
   then show ?thesis using 2
    by (metis to-Zp-val)
  qed
  thus ?thesis using 0 1 2 assms val-ring-equal-res-imp-equal-val[of g • b g • a n]
by blast
qed

lemma to-Zp-poly-monom:
  assumes a ∈ 𝒪_p
  shows to-Zp-poly (monom (UP Q_p) a n) = monom (UP Z_p) (to-Zp a) n
```

unfolding *to-Zp-poly-def*
apply(*rule ext*)
using *assms cfs-monom*[*of a n*] *Zp.cfs-monom*[*of to-Zp a n*]
by (*simp add*: *to-Zp-closed to-Zp-zero val-ring-memE*(*2*))

lemma *to-Zp-poly-add*:
assumes $f \in carrier\ (UP\ Q_p)$
assumes $gauss\text{-}norm\ f \geq 0$
assumes $g \in carrier\ (UP\ Q_p)$
assumes $gauss\text{-}norm\ g \geq 0$
shows $to\text{-}Zp\text{-}poly\ (f \oplus_{UP\ Q_p} g) = to\text{-}Zp\text{-}poly\ f \oplus_{UP\ Z_p} to\text{-}Zp\text{-}poly\ g$
proof−
obtain *F* **where** *F-def*: $F = to\text{-}Zp\text{-}poly\ f$
by *blast*
obtain *G* **where** *G-def*: $G = to\text{-}Zp\text{-}poly\ g$
by *blast*
have *F-closed*: $F \in carrier\ (UP\ Z_p)$
unfolding *F-def* **using** *assms*
by (*simp add*: *to-Zp-poly-closed*)
have *G-closed*: $G \in carrier\ (UP\ Z_p)$
unfolding *G-def* **using** *assms*
by (*simp add*: *to-Zp-poly-closed*)
have *F-inc*: $poly\text{-}inc\ F = f$
using *assms* **unfolding** *F-def*
using *poly-inc-inverse-left* **by** *blast*
have *G-inc*: $poly\text{-}inc\ G = g$
using *assms* **unfolding** *G-def*
by (*simp add*: *poly-inc-inverse-left*)
have *0*: $poly\text{-}inc\ (F \oplus_{UP\ Z_p} G) = poly\text{-}inc\ F \oplus_{UP\ Q_p} poly\text{-}inc\ G$
using *F-closed G-closed*
by (*simp add*: *poly-inc-plus*)
have *1*: $to\text{-}Zp\text{-}poly\ (poly\text{-}inc\ (F \oplus_{UP\ Z_p} G)) = F \oplus_{UP\ Z_p} G$
using *G-closed F-closed*
by (*simp add*: *poly-inc-inverse-right*)
show *?thesis*
using *1* **unfolding** *F-inc G-inc 0* **unfolding** *F-def G-def*
by *blast*
qed

lemma *to-Zp-poly-zero*:
$to\text{-}Zp\text{-}poly\ (\mathbf{0}_{UP\ Q_p}) = \mathbf{0}_{UP\ Z_p}$
unfolding *to-Zp-poly-def*
apply(*rule ext*)
by (*simp add*: *to-Zp-zero*)

lemma *to-Zp-poly-one*:
$to\text{-}Zp\text{-}poly\ (\mathbf{1}_{UP\ Q_p}) = \mathbf{1}_{UP\ Z_p}$
unfolding *to-Zp-poly-def*
apply(*rule ext*)

by (*metis Zp.UP-one-closed poly-inc-inverse-right poly-inc-one to-Zp-poly-def*)

lemma *val-ring-add-pow*:
assumes $a \in$ *carrier* Q_p
assumes *val* $a \geq 0$
shows *val* $([(n::nat)]\cdot a) \geq 0$
proof−
have *0*: $[(n::nat)]\cdot a = ([n]\cdot \mathbf{1})\otimes a$
using *assms Qp.add-pow-ldistr Qp.cring-simprules*(*12*) *Qp.one-closed* **by** *presburger*
show *?thesis* **unfolding** *0* **using** *assms*
by (*meson Qp.nat-inc-closed val-ring-memE val-of-nat-inc val-ringI val-ring-times-closed*)
qed

lemma *to-Zp-poly-pderiv*:
assumes $g \in$ *carrier* (*UP* Q_p)
assumes *gauss-norm* $g \geq 0$
shows *to-Zp-poly* (*pderiv g*) = *Zp.pderiv* (*to-Zp-poly g*)
proof−
have *0*: *gauss-norm* $g \geq 0 \longrightarrow$ *to-Zp-poly* (*pderiv g*) = *Zp.pderiv* (*to-Zp-poly g*)
proof(*rule poly-induct*, *rule assms*, *rule*)
fix *p*
assume *A*: $p \in$ *carrier* (*UP* Q_p)
deg Q_p *p* = *0*
0 ≤ *gauss-norm p*
obtain *a* **where** *a-def*: $a \in \mathcal{O}_p \wedge p =$ *monom* (*UP* Q_p) *a 0*
using *A*
by (*metis UPQ.ltrm-deg-0 positive-gauss-norm-valuation-ring-coeffs*)
have *p-eq*: *p* = *monom* (*UP* Q_p) *a 0*
using *a-def* **by** *blast*
have *0*: *to-Zp-poly p* = *monom* (*UP* Z_p) (*to-Zp a*) *0*
unfolding *p-eq*
apply(*rule to-Zp-poly-monom*)
using *a-def* **by** *blast*
have *1*: *UPQ.pderiv* (*monom* (*UP* Q_p) *a 0*) = $\mathbf{0}_{UP\ Q_p}$
using *A*(*1*) *A*(*2*) *UPQ.pderiv-deg-0 p-eq* **by** *blast*
have *2*: *Zp.pderiv* (*monom* (*UP* Z_p) (*to-Zp a*) *0*) = $\mathbf{0}_{UP\ Z_p}$
apply(*rule Zp.pderiv-deg-0*)
apply(*rule Zp.monom-closed*, *rule to-Zp-closed*)
using *a-def*
apply (*simp add*: *val-ring-memE*(*2*); *fail*)
apply(*cases to-Zp a* = $\mathbf{0}_{Z_p}$)
apply (*simp*; *fail*)
apply(*rule Zp.deg-monom*, *blast*)
using *a-def*
by (*simp add*: *to-Zp-closed val-ring-memE*(*2*))
show *to-Zp-poly* (*UPQ.pderiv p*) = *Zp.pderiv* (*to-Zp-poly p*)
unfolding *0* **unfolding** *p-eq*
unfolding *1 2 to-Zp-poly-zero* **by** *blast*

next
fix p
assume A: $\bigwedge q.\ q \in carrier\ (UP\ Q_p) \Longrightarrow$
$deg\ Q_p\ q < deg\ Q_p\ p \Longrightarrow$
$0 \leq gauss\text{-}norm\ q \longrightarrow$
$to\text{-}Zp\text{-}poly\ (UPQ.pderiv\ q) = Zp.pderiv\ (to\text{-}Zp\text{-}poly\ q)$
$p \in carrier\ (UP\ Q_p)$
$0 < deg\ Q_p\ p$
show $0 \leq gauss\text{-}norm\ p \longrightarrow to\text{-}Zp\text{-}poly\ (UPQ.pderiv\ p) = Zp.pderiv\ (to\text{-}Zp\text{-}poly\ p)$
proof
assume B: $0 \leq gauss\text{-}norm\ p$
obtain q **where** $q\text{-}def$: $q = trunc\ p$
by $blast$
have $p\text{-}eq$: $p = q \oplus_{UP\ Q_p} ltrm\ p$
by $(simp\ add{:}\ A(2)\ UPQ.trunc\text{-}simps(1)\ q\text{-}def)$
have $q\text{-}gauss\text{-}norm$: $gauss\text{-}norm\ q \geq 0$
unfolding $q\text{-}def$
apply$(rule\ gauss\text{-}norm\text{-}geqI)$
using A **apply** $(simp\ add{:}\ UPQ.trunc\text{-}closed;\ fail)$
using $trunc\text{-}cfs[of\ p]\ A\ gauss\text{-}normE$
proof $-$
fix $n :: nat$
have $f1$: $\mathbf{0} = q\ (deg\ Q_p\ p)$
by $(simp\ add{:}\ UPQ.deg\text{-}leE\ UPQ.trunc\text{-}closed\ UPQ.trunc\text{-}degree\ \langle 0 < deg\ Q_p\ p\rangle\ \langle p \in carrier\ (UP\ Q_p)\rangle\ q\text{-}def)$
have $\forall n.\ 0 \leq val\ (p\ n)$
by $(meson\ B\ \langle p \in carrier\ (UP\ Q_p)\rangle\ eint\text{-}ord\text{-}trans\ gauss\text{-}normE)$
then show $0 \leq val\ (Cring\text{-}Poly.truncate\ Q_p\ p\ n)$
using $f1$ **by** $(metis\ (no\text{-}types)\ Qp.nat\text{-}mult\text{-}zero\ UPQ.ltrm\text{-}closed\ UPQ.coeff\text{-}of\text{-}sum\text{-}diff\text{-}degree0\ UPQ.deg\text{-}ltrm\ UPQ.trunc\text{-}closed\ \langle\bigwedge n.\ [\![p \in carrier\ (UP\ Q_p);\ n < deg\ Q_p\ p]\!] \Longrightarrow Cring\text{-}Poly.truncate\ Q_p\ p\ n = p\ n\rangle\ \langle p \in carrier\ (UP\ Q_p)\rangle\ nat\text{-}neq\text{-}iff\ p\text{-}eq\ q\text{-}def\ val\text{-}of\text{-}nat\text{-}inc)$
qed
have 0: $to\text{-}Zp\text{-}poly\ (UPQ.pderiv\ q) = Zp.pderiv\ (to\text{-}Zp\text{-}poly\ q)$
using $A\ q\text{-}def\ q\text{-}gauss\text{-}norm$
by $(simp\ add{:}\ UPQ.trunc\text{-}closed\ UPQ.trunc\text{-}degree)$
have 1: $UPQ.pderiv\ (monom\ (UP\ Q_p)\ (p\ (deg\ Q_p\ p))\ (deg\ Q_p\ p)) =$
$monom\ (UP\ Q_p)\ ([deg\ Q_p\ p] \cdot p\ (deg\ Q_p\ p))\ (deg\ Q_p\ p - 1)$
apply$(rule\ pderiv\text{-}monom)$
using A **by** $(simp\ add{:}\ UPQ.UP\text{-}car\text{-}memE(1))$
have 2: $Zp.pderiv\ (monom\ (UP\ Z_p)\ (to\text{-}Zp\ (p\ (deg\ Q_p\ p)))\ (deg\ Q_p\ p)) =$
$monom\ (UP\ Z_p)\ ([deg\ Q_p\ p] \cdot_{Z_p} to\text{-}Zp\ (\ p\ (deg\ Q_p\ p)))\ (deg\ Q_p\ p - 1)$
using $A\ \ Zp.pderiv\text{-}monom[of\ to\text{-}Zp\ (\ p\ (deg\ Q_p\ p))\ deg\ Q_p\ p]$
by $(simp\ add{:}\ UPQ.lcf\text{-}closed\ to\text{-}Zp\text{-}closed)$
have 3: $to\text{-}Zp\text{-}poly\ (UPQ.pderiv\ (monom\ (UP\ Q_p)\ (p\ (deg\ Q_p\ p))\ (deg\ Q_p\ p))) = monom\ (UP\ Z_p)\ (to\text{-}Zp\ ([deg\ Q_p\ p] \cdot p\ (deg\ Q_p\ p)))\ (deg\ Q_p\ p - 1)$
unfolding 1 **apply**$(rule\ to\text{-}Zp\text{-}poly\text{-}monom)$
apply$(rule\ val\text{-}ring\text{-}memI)$

apply (*simp add*: *A*(*2*) *UPQ.UP-car-memE*(*1*); *fail*)
apply(*rule val-ring-add-pow*)
using *A*
apply (*simp add*: *UPQ.lcf-closed*; *fail*)
using *B A*
by (*simp add*: *positive-gauss-norm-valuation-ring-coeffs val-ring-memE*(*1*))
have *4*: *to-Zp-poly* (*ltrm p*) = *monom* (*UP* Z_p) (*to-Zp* (*p* (*deg* Q_p *p*))) (*deg* Q_p *p*)
apply(*rule to-Zp-poly-monom*) **using** *A*
by (*simp add*: *B positive-gauss-norm-valuation-ring-coeffs*)
have *5*: *to-Zp-poly* (*UPQ.pderiv* (*ltrm p*)) = *Zp.pderiv* (*to-Zp-poly* (*ltrm p*))
unfolding *3 4 2*
by (*simp add*: *A*(*2*) *B positive-gauss-norm-valuation-ring-coeffs to-Zp-nat-add-pow*)
have *6*: *pderiv p* = *pderiv q* $\oplus_{UP\ Q_p}$ *pderiv* (*ltrm p*)
using *p-eq*
by (*metis A*(*2*) *UPQ.ltrm-closed UPQ.pderiv-add UPQ.trunc-closed p-eq q-def*)
have *7*: *to-Zp-poly p* = *to-Zp-poly q* $\oplus_{UP\ Z_p}$ *to-Zp-poly* (*ltrm p*)
using *p-eq*
by (*metis* (*no-types*, *lifting*) *A*(*2*) *B UPQ.ltrm-closed UPQ.cfs-closed UPQ.trunc-closed gauss-norm-monom positive-gauss-norm-valuation-ring-coeffs q-def q-gauss-norm to-Zp-poly-add val-ring-memE*(*1*))
have *8*: *to-Zp-poly* (*pderiv p*) =
to-Zp-poly (*UPQ.pderiv q*) $\oplus_{UP\ Z_p}$
to-Zp-poly (*UPQ.pderiv* (*monom* (*UP* Q_p) (*p* (*deg* Q_p *p*)) (*deg* Q_p *p*)))
unfolding *6* **apply**(*rule to-Zp-poly-add*)
apply (*simp add*: *A*(*2*) *UPQ.pderiv-closed UPQ.trunc-closed q-def*; *fail*)
apply (*metis A*(*2*) *UPQ.cfs-closed UPQ.pderiv-cfs UPQ.trunc-closed gauss-norm-coeff-norm positive-gauss-norm-valuation-ring-coeffs q-def q-gauss-norm val-ring-add-pow val-ring-memE*(*1*))
apply (*simp add*: *A*(*2*) *UPQ.UP-car-memE*(*1*) *UPQ.pderiv-closed*; *fail*)
apply(*rule eint-ord-trans*[*of - gauss-norm* (*monom* (*UP* Q_p) (*p* (*deg* Q_p *p*)) (*deg* Q_p *p*))])
apply (*simp add*: *A*(*2*) *B UPQ.cfs-closed gauss-norm-monom positive-gauss-norm-valuation-ring-coeffs val-ring-memE*(*1*); *fail*)
apply(*rule gauss-norm-pderiv*)
using *A*(*2*) *UPQ.ltrm-closed* **by** *blast*
have *9*: *Zp.pderiv* (*to-Zp-poly p*) = *Zp.pderiv* (*to-Zp-poly q*) $\oplus_{UP\ Z_p}$
Zp.pderiv (*to-Zp-poly* (*monom* (*UP* Q_p) (*p* (*deg* Q_p *p*)) (*deg* Q_p *p*)))
unfolding *7* **apply**(*rule Zp.pderiv-add*)
apply(*rule to-Zp-poly-closed*)
apply (*simp add*: *A*(*2*) *UPQ.trunc-closed q-def*; *fail*)
apply (*simp add*: *q-gauss-norm*; *fail*)
apply(*rule to-Zp-poly-closed*)
apply (*simp add*: *A*(*2*) *UPQ.UP-car-memE*(*1*); *fail*)
by (*simp add*: *A*(*2*) *B UPQ.cfs-closed gauss-norm-monom positive-gauss-norm-valuation-ring-coeffs val-ring-memE*(*1*))
show *to-Zp-poly* (*UPQ.pderiv p*) = *Zp.pderiv* (*to-Zp-poly p*)

unfolding *9 8 5 0* **by** *blast*
qed
qed
thus *?thesis* **using** *assms* **by** *blast*
qed

lemma *val-p-int-pow*:
val ($\mathfrak{p}[\widehat{}]k$) = *eint* (*k*)
by (*simp add*: *ord-p-pow-int p-intpow-closed*(*2*))

definition *int-gauss-norm* **where**
int-gauss-norm g = (*SOME n*::*int*. *eint n* = *gauss-norm g*)

lemma *int-gauss-norm-eq*:
assumes *g* ∈ *carrier* (*UP* Q_p)
assumes *g* ≠ $\mathbf{0}_{UP\ Q_p}$
shows *eint* (*int-gauss-norm g*) = *gauss-norm g*
proof−
have *0*: *gauss-norm g* < ∞
using *assms* **by** (*simp add*: *gauss-norm-prop*)
then show *?thesis* **unfolding** *int-gauss-norm-def*
using *assms*
by *fastforce*
qed

lemma *int-gauss-norm-smult*:
assumes *g* ∈ *carrier* (*UP* Q_p)
assumes *g* ≠ $\mathbf{0}_{UP\ Q_p}$
assumes *a* ∈ *nonzero* Q_p
shows *int-gauss-norm* (*a* $\odot_{UP\ Q_p}$ *g*) = *ord a* + *int-gauss-norm g*
using *gauss-norm-smult*[*of g a*] *int-gauss-norm-eq val-ord assms*
by (*metis* (*no-types*, *opaque-lifting*) *Qp.nonzero-closed UPQ.UP-smult-closed UPQ.cfs-zero eint.distinct*(*2*) *eint.inject gauss-norm-coeff-norm local.val-zero plus-eint-simps*(*1*))

definition *normalize-poly* **where**
normalize-poly g = (*if g* = $\mathbf{0}_{UP\ Q_p}$ *then g else* ($\mathfrak{p}[\widehat{}](-$ *int-gauss-norm g*)) $\odot_{Q_p}$-*x*
g)

lemma *normalize-poly-zero*:
normalize-poly $\mathbf{0}_{UP\ Q_p}$ = $\mathbf{0}_{UP\ Q_p}$
unfolding *normalize-poly-def* **by** *simp*

lemma *normalize-poly-nonzero-eq*:
assumes *g* ≠ $\mathbf{0}_{UP\ Q_p}$
assumes *g* ∈ *carrier* (*UP* Q_p)
shows *normalize-poly g* = ($\mathfrak{p}[\widehat{}](-$ *int-gauss-norm g*)) $\odot_{UP\ Q_p}$ *g*
using *assms* **unfolding** *normalize-poly-def* **by** *simp*

lemma *int-gauss-norm-normalize-poly*:

assumes $g \neq \mathbf{0}_{UP\ Q_p}$
assumes $g \in carrier\ (UP\ Q_p)$
shows *int-gauss-norm* (*normalize-poly g*) = *0*
using *normalize-poly-nonzero-eq int-gauss-norm-smult assms*
by (*simp add*: *ord-p-pow-int p-intpow-closed*(*2*))

lemma *normalize-poly-closed*:
assumes $g \in carrier\ (UP\ Q_p)$
shows *normalize-poly g* $\in carrier\ (UP\ Q_p)$
using *assms* **unfolding** *normalize-poly-def*
by (*simp add*: *p-intpow-closed*(*1*))

lemma *normalize-poly-nonzero*:
assumes $g \neq \mathbf{0}_{UP\ Q_p}$
assumes $g \in carrier\ (UP\ Q_p)$
shows *normalize-poly g* $\neq \mathbf{0}_{UP\ Q_p}$
using *assms normalize-poly-nonzero-eq*
by (*metis* (*no-types*, *lifting*) *UPQ.UP-smult-one UPQ.module-axioms UPQ.smult-r-null module.smult-assoc1 p-intpow-closed*(*1*) *p-intpow-inv'*)

lemma *gauss-norm-normalize-poly*:
assumes $g \neq \mathbf{0}_{UP\ Q_p}$
assumes $g \in carrier\ (UP\ Q_p)$
shows *gauss-norm* (*normalize-poly g*) = *0*
proof−
have *0*: *eint* (*int-gauss-norm* (*normalize-poly g*)) = *gauss-norm* (*normalize-poly g*)
by(*rule int-gauss-norm-eq*, *rule normalize-poly-closed*, *rule assms*,
rule normalize-poly-nonzero, *rule assms*, *rule assms*)
show *?thesis*
using *0 int-gauss-norm-normalize-poly assms*
by (*simp add*: *zero-eint-def*)
qed

lemma *taylor-term-eval-eq*:
assumes $f \in carrier\ (UP\ Q_p)$
assumes $x \in carrier\ Q_p$
assumes $t \in carrier\ Q_p$
assumes $\bigwedge j.\ i \neq j \Longrightarrow val\ (UPQ.taylor\text{-}term\ x\ f\ i \cdot t) < val\ (UPQ.taylor\text{-}term\ x\ f\ j \cdot t)$
shows *val* $(f \cdot t) = val\ (UPQ.taylor\text{-}term\ x\ f\ i \cdot t)$
proof−
have *0*: $f = finsum\ (UP\ Q_p)\ (UPQ.taylor\text{-}term\ x\ f)\ \{..deg\ Q_p\ f\}$
by(*rule UPQ.taylor-term-sum*[*of f deg* Q_p *f x*], *rule assms*, *blast*, *rule assms*)
show *?thesis*
proof(*cases* $i \in \{..deg\ Q_p\ f\}$)
case *True*
have *T0*: $finsum\ (UP\ Q_p)\ (UPQ.taylor\text{-}term\ x\ f)\ \{..deg\ Q_p\ f\} = UPQ.taylor\text{-}term\ x\ f\ i \oplus_{UP\ Q_p} finsum\ (UP\ Q_p)\ (UPQ.taylor\text{-}term\ x\ f)\ (\{..deg\ Q_p\ f\} - \{i\})$

apply(*rule UPQ.P.finsum-remove*[*of* {..*deg* Q_p *f*} *UPQ.taylor-term x f i*])
by(*rule UPQ.taylor-term-closed*, *rule assms*, *rule assms*, *blast*, *rule True*)
have *T1*: *f* = *UPQ.taylor-term x f i* $\oplus_{UP\ Q_p}$ *finsum* (*UP* Q_p) (*UPQ.taylor-term x f*) ({..*deg* Q_p *f*} − {*i*})
using *0 T0* **by** *metis*
have *T2*: *finsum* (*UP* Q_p) (*UPQ.taylor-term x f*) ({..*deg* Q_p *f*} − {*i*}) ∈ *carrier* (*UP* Q_p)
apply(*rule UPQ.P.finsum-closed*)
using *UPQ.taylor-term-closed assms*(*1*) *assms*(*2*) **by** *blast*
have *T3*: *UPQ.taylor-term x f i* ∈ *carrier* (*UP* Q_p)
by(*rule UPQ.taylor-term-closed*, *rule assms*, *rule assms*)
obtain *g* **where** *g-def*: *g* = *f*
by *blast*
have *T4*: *g* = *UPQ.taylor-term x f i* $\oplus_{UP\ Q_p}$ *finsum* (*UP* Q_p) (*UPQ.taylor-term x f*) ({..*deg* Q_p *f*} − {*i*})
unfolding *g-def* **by**(*rule T1*)
have *g-closed*: *g* ∈ *carrier* (*UP* Q_p)
unfolding *g-def* **by**(*rule assms*)
have *T5*: *g* ⋅ *t* = *UPQ.taylor-term x f i* ⋅ *t* ⊕ (*finsum* (*UP* Q_p) (*UPQ.taylor-term x f*) ({..*deg* Q_p *f*} − {*i*})) ⋅ *t*
unfolding *T4* **by**(*rule UPQ.to-fun-plus*, *rule T2*, *rule T3*, *rule assms*)
have *T6*: (*finsum* (*UP* Q_p) (*UPQ.taylor-term x f*) ({..*deg* Q_p *f*} − {*i*})) ⋅ *t* =
(*finsum* Q_p (λ*i*. *UPQ.taylor-term x f i* ⋅ *t*) ({..*deg* Q_p *f*} − {*i*}))
apply(*rule UPQ.to-fun-finsum*, *blast*)
using *assms UPQ.taylor-term-closed* **apply** *blast*
using *assms* **by** *blast*
have *T7*: ⋀*j*. *j* ∈ {..*deg* Q_p *f*} − {*i*} ⟹ *val* (*UPQ.taylor-term x f j* ⋅ *t*) > *val* (*UPQ.taylor-term x f i* ⋅ *t*)
using *assms* **by** (*metis Diff-iff singletonI*)
have *T8*: *val* ((*finsum* (*UP* Q_p) (*UPQ.taylor-term x f*) ({..*deg* Q_p *f*} − {*i*})) ⋅ *t*) > *val* (*UPQ.taylor-term x f i* ⋅ *t*)
unfolding *T6*
apply(*rule finsum-val-ultrametric''*)
using *UPQ.taylor-term-closed assms*
apply (*metis* (*no-types*, *lifting*) *Pi-I UPQ.to-fun-closed*)
apply *blast*
using *assms T7* **apply** *blast*
using *assms*(*4*)[*of Suc i*] **using** *eint-ord-simps*(*4*)
assms(*4*) *eint-ord-code*(*6*) *g-def gr-implies-not-zero less-one* **by** *smt*
have *T9*: *val* (*g* ⋅ *t*) = *val* (*UPQ.taylor-term x f i* ⋅ *t*)
unfolding *T5* **using** *T8 T2 T3*
by (*metis* (*no-types*, *lifting*) *Qp.add.m-comm UPQ.to-fun-closed assms*(*3*) *val-ultrametric-noteq*)
show *?thesis* **using** *T9* **unfolding** *g-def* **by** *blast*
next
case *False*
have *i* > *deg* Q_p *f*
using *False* **by** *simp*
hence *i* > *deg* Q_p (*UPQ.taylor x f*)

using *assms UPQ.taylor-deg* **by** *presburger*
hence *F0*: *UPQ.taylor x f i* = **0**
using *assms UPQ.taylor-closed UPQ.deg-leE* **by** *blast*
have *F1*: (*UPQ.taylor-term x f i* ⋅ *t*) = **0**
using *UPQ.to-fun-taylor-term*[*of f t x i*]
unfolding *F0*
using *assms Qp.cring-simprules*(*2*) *Qp.cring-simprules*(*4*) *Qp.integral-iff Qp.nat-pow-closed* **by** *presburger*
show *?thesis*
using *assms*(*4*)[*of Suc i*] **unfolding** *F1*
by (*metis eint-ord-code*(*6*) *local.val-zero n-not-Suc-n*)
qed
qed

9.3 Hensel's Lemma for p-adic fields

theorem *hensels-lemma*:
assumes *f* ∈ *carrier* (*UP Q*$_p$)
assumes *a* ∈ $\mathcal{O}_p$
assumes *gauss-norm f* ≥ *0*
assumes *val* (*f*⋅*a*) > *2*∗*val* ((*pderiv f*)⋅*a*)
shows ∃!α ∈ $\mathcal{O}_p$. *f*⋅α = **0** ∧ *val* (*a* ⊖ α) > *val* ((*pderiv f*)⋅*a*)
proof−
have *a-closed*: *a* ∈ *carrier Q*$_p$
using *assms val-ring-memE* **by** *auto*
have *f-nonzero*: *f* ≠ **0**$_{UP\ Q_p}$
proof(*rule ccontr*)
assume *N*: ¬ *f* ≠ **0**$_{UP\ Q_p}$
then have *0*: *pderiv f* = **0**$_{UP\ Q_p}$
using *UPQ.deg-zero UPQ.pderiv-deg-0* **by** *blast*
have *1*: *f* = **0**$_{UP\ Q_p}$
using *N* **by** *auto*
have *2*: *eint 2* ∗ *val* (*UPQ.pderiv* **0**$_{UP\ Q_p}$ ⋅ *a*) = ∞
by (*simp add*: *UPQ.to-fun-zero local.a-closed local.val-zero*)
show *False* **using** *assms a-closed*
unfolding *2 1*
using *eint-ord-simps*(*6*) **by** *blast*
qed
obtain *h* **where** *h-def*: *h* = *to-Zp-poly f*
by *blast*
have *h-closed*: *h* ∈ *carrier* (*UP Z*$_p$)
unfolding *h-def* **using** *assms*
by (*simp add*: *to-Zp-poly-closed*)
have *h-deriv*: *Zp.pderiv h* = *to-Zp-poly* (*pderiv f*)
unfolding *h-def*
using *to-Zp-poly-pderiv*[*of f*] *assms* **by** *auto*
have *0*: *to-Zp* (*f*⋅*a*) = *to-function Z*$_p$ *h* (*to-Zp a*)
unfolding *h-def*
using *assms a-closed*

by (*simp add*: *UPQ.to-fun-def to-Zp-poly-eval*)
have *1*: *to-Zp* ((*pderiv f*)•*a*) = *to-function* Z_p (*Zp.pderiv h*) (*to-Zp a*)
unfolding *h-deriv*
using *assms a-closed UPQ.pderiv-closed UPQ.to-fun-def eint-ord-trans gauss-norm-pderiv to-Zp-poly-eval*
by *presburger*
have *2*: *val* (*f*•*a*) = *val-Zp* (*to-function* Z_p *h* (*to-Zp a*))
proof−
have *20*: *f*•*a* ∈ $\mathcal{O}_p$
using *assms positive-gauss-norm-eval* **by** *blast*
have *21*: *val* (*f*•*a*) = *val-Zp* (*to-Zp* (*f*•*a*))
using *20* **by** (*simp add*: *to-Zp-val*)
show *?thesis* **unfolding** *21 0* **by** *blast*
qed
have *3*: *val* ((*pderiv f*)•*a*) = *val-Zp* (*to-function* Z_p (*Zp.pderiv h*) (*to-Zp a*))
proof−
have *30*: (*pderiv f*)•*a* ∈ $\mathcal{O}_p$
using *positive-gauss-norm-eval assms gauss-norm-pderiv*
by (*meson UPQ.pderiv-closed eint-ord-trans*)
have *31*: *val* ((*pderiv f*)•*a*) = *val-Zp* (*to-Zp* ((*pderiv f*)•*a*))
using *30* **by** (*simp add*: *to-Zp-val*)
show *?thesis* **unfolding** *31 1* **by** *blast*
qed
have *4*: ∃!α. α ∈ *carrier* Z_p ∧
Zp.to-fun (*to-Zp-poly f*) α = $\mathbf{0}_{Z_p}$ ∧
val-Zp (*Zp.to-fun* (*Zp.pderiv* (*to-Zp-poly f*)) (*to-Zp a*))
< *val-Zp* (*to-Zp a* ⊖$_{Z_p}$ α)
apply(*rule hensels-lemma'*)
using *h-closed h-def* **apply** *blast*
using *assms local.a-closed to-Zp-closed* **apply** *blast*
using *assms* **unfolding** *2 3 h-def Zp.to-fun-def* **by** *blast*
obtain α **where** α*-def*: α ∈ *carrier* Z_p ∧
Zp.to-fun (*to-Zp-poly f*) α = $\mathbf{0}_{Z_p}$ ∧
val-Zp (*Zp.to-fun* (*Zp.pderiv* (*to-Zp-poly f*)) (*to-Zp a*))
< *val-Zp* (*to-Zp a* ⊖$_{Z_p}$ α)
∧ (∀ *x*. *x* ∈ *carrier* Z_p ∧
Zp.to-fun (*to-Zp-poly f*) *x* = $\mathbf{0}_{Z_p}$ ∧
val-Zp (*Zp.to-fun* (*Zp.pderiv* (*to-Zp-poly f*)) (*to-Zp a*))
< *val-Zp* (*to-Zp a* ⊖$_{Z_p}$ *x*) ⟶ *x* = α)
using *4* **by** *blast*
obtain β **where** β*-def*: β = ι α
by *blast*
have β*-closed*: β ∈ $\mathcal{O}_p$
using α*-def* **unfolding** β*-def* **by** *simp*
have *5*: (*Zp.to-fun* (*to-Zp-poly f*) α) = *to-Zp* (*f*•β)
using β*-closed to-Zp-poly-eval*[*of f* β] *assms*
unfolding β*-def UPQ.to-fun-def*
by (*simp add*: *Zp.to-fun-def* α*-def inc-to-Zp*)
have *6*: *to-Zp* (*f*•β) = $\mathbf{0}_{Z_p}$

using *5* α*-def* **by** *auto*
have β*-closed*: $\beta \in \mathcal{O}_p$
unfolding β*-def* **using** α*-def* **by** *simp*
have *7*: $(f \cdot \beta) = \mathbf{0}$
using *6 assms* **unfolding** β*-def*
by (*metis* β*-closed* β*-def inc-of-zero positive-gauss-norm-eval to-Zp-inc*)
have *8*: $\alpha =$ *to-Zp* β
unfolding β*-def* **using** α*-def*
by (*simp add*: *inc-to-Zp*)
have *9*: *to-Zp* $a \ominus_{Z_p} \alpha =$ *to-Zp* $(a \ominus \beta)$
unfolding *8* **using** *assms*(*2*) β*-closed*
by (*simp add*: *to-Zp-minus*)
have *10*: *val* $(a \ominus \beta) =$ *val-Zp* (*to-Zp* $a \ominus_{Z_p} \alpha$)
unfolding *9* **using** β*-closed assms*(*2*)
to-Zp-val val-ring-minus-closed **by** *presburger*
have *11*: *val* $(a \ominus \beta) >$ *val* $((pderiv\ f) \cdot a)$
using α*-def* **unfolding** *9 10 3 h-def*
by (*simp add*: *Zp.to-fun-def*)
have *12*: $\beta \in \mathcal{O}_p \wedge f \cdot \beta = \mathbf{0} \wedge$ *val* (*UPQ.pderiv* $f \cdot a$) $<$ *val* $(a \ominus \beta)$
using *11 7* β*-closed* **by** *linarith*
have *13*: $\forall x.\ x \in \mathcal{O}_p \wedge f \cdot x = \mathbf{0} \wedge$ *val* (*UPQ.pderiv* $f \cdot a$) $<$ *val* $(a \ominus x)$
$\longrightarrow x = \beta$
proof(*rule*, *rule*)
fix x **assume** *A*: $x \in \mathcal{O}_p \wedge\ f \cdot x = \mathbf{0} \wedge$ *val* (*UPQ.pderiv* $f \cdot a$) $<$ *val* $(a \ominus x)$
obtain y **where** *y-def*: $y =$ *to-Zp* x
by *blast*
have *y-closed*: $y \in$ *carrier* Z_p
unfolding *y-def* **using** *A*
by (*simp add*: *to-Zp-closed val-ring-memE*(*2*))
have *eval*: *Zp.to-fun* (*to-Zp-poly f*) $y = \mathbf{0}_{Z_p}$
unfolding *y-def* **using** *A assms*
by (*metis UPQ.to-fun-def Zp.to-fun-def to-Zp-poly-eval to-Zp-zero*)
have *0*: *to-Zp* $a \ominus_{Z_p} y =$ *to-Zp* $(a \ominus x)$
unfolding *y-def* **using** *A assms*
by (*simp add*: *to-Zp-minus*)
have *q*: *val-Zp* (*Zp.to-fun* (*Zp.pderiv* (*to-Zp-poly f*)) (*to-Zp a*)) = *val* (*UPQ.pderiv* $f \cdot a$)
by (*simp add*: *3 Zp.to-fun-def h-def*)
have *1*: $y \in$ *carrier* $Z_p \wedge$
Zp.to-fun (*to-Zp-poly f*) $y = \mathbf{0}_{Z_p} \wedge$
val-Zp (*Zp.to-fun* (*Zp.pderiv* (*to-Zp-poly f*)) (*to-Zp a*))
$<$ *val-Zp* (*to-Zp* $a \ominus_{Z_p} y$)
unfolding *0 eval Zp.to-fun-def h-def*
apply(*intro conjI y-closed*)
using *eval Zp.to-fun-def* **apply** (*simp*; *fail*)
using *A* **unfolding** *0 eval Zp.to-fun-def h-def 3*
using *assms*(*2*) *to-Zp-val val-ring-minus-closed* **by** *presburger*
have *2*: $y = \alpha$
using *1* α*-def* **by** *blast*

show $x = \beta$
using *y-def* **unfolding** *2 8* **using** *A* β*-closed*
by (*metis to-Zp-inc*)
qed
show $\exists!\alpha.\ \alpha \in \mathcal{O}_p \wedge f \cdot \alpha = \mathbf{0} \wedge$ *val* (*UPQ.pderiv f* $\cdot$ *a*) $<$ *val* (*a* $\ominus$ α)
using *12 13* **by** *metis*
qed

lemma *nth-root-poly-root-fixed*:
assumes (*n*::*nat*) $>$ *1*
assumes $a \in \mathcal{O}_p$
assumes *val* (**1** $\ominus_{Q_p}$ *a*) $>$ *2*$*$ *val* ([*n*]$\cdot$**1**)
shows ($\exists!$ $b \in \mathcal{O}_p$. (*b*[^]*n*) $=$ *a* $\wedge$ *val* (*b* $\ominus$ **1**) $>$ *val* ([*n*]$\cdot$**1**))
proof−
obtain *f* **where** *f-def*: *f* $=$ *up-ring.monom* (*UP* Q_p) **1** *n* $\ominus_{UP\ Q_p}$ *up-ring.monom* (*UP* Q_p) *a 0*
by *blast*
have *f-closed*: *f* $\in$ *carrier* (*UP* Q_p)
unfolding *f-def* **apply**(*rule UPQ.P.ring-simprules*)
apply (*simp*; *fail*) **using** *assms*
by (*simp add*: *val-ring-memE*(*2*))
have *0*: *UPQ.pderiv* (*up-ring.monom* (*UP* Q_p) *a 0*) $=$ $\mathbf{0}_{UP\ Q_p}$
using *assms*
by (*simp add*: *val-ring-memE*(*2*))
have *1*: *UPQ.pderiv* (*up-ring.monom* (*UP* Q_p) (**1**) *n*) $=$ (*up-ring.monom* (*UP* Q_p) ([*n*]$\cdot$**1**) (*n*−*1*))
using *UPQ.pderiv-monom* **by** *blast*
have *2*: *up-ring.monom* (*UP* Q_p) **1** *n* $\in$ *carrier* (*UP* Q_p)
by *simp*
have *3*: *up-ring.monom* (*UP* Q_p) *a 0* $\in$ *carrier* (*UP* Q_p)
using *assms val-ring-memE* **by** *simp*
have *4*: *UPQ.pderiv f* $=$ *up-ring.monom* (*UP* Q_p) ([*n*] $\cdot$ **1**) (*n* $-$ *1*) $\ominus_{UP\ Q_p}$ $\mathbf{0}_{UP\ Q_p}$
using *2 3 assms val-ring-memE UPQ.pderiv-minus*[*of up-ring.monom* (*UP* Q_p) **1** *n up-ring.monom* (*UP* Q_p) *a 0*]
unfolding *f-def 0 1* **by** *blast*
have *5*: *UPQ.pderiv f* $=$ (*up-ring.monom* (*UP* Q_p) ([*n*]$\cdot$**1**) (*n*−*1*))
unfolding *4 a-minus-def* **by** *simp*
have *a-closed*: *a* $\in$ *carrier* Q_p
using *assms val-ring-memE* **by** *blast*
have *6*: *UPQ.pderiv f* $\cdot$ **1** $=$ [*n*]$\cdot$**1** $\otimes$ **1**[^](*n*−*1*)
unfolding *5* **using** *a-closed*
by (*simp add*: *UPQ.to-fun-monom*)
have *7*: *val* (**1** $\ominus_{Q_p}$ *a*) $>$ *val* **1**
proof−
have *eint 2* $*$ *val* ([*n*] $\cdot$ **1**) $\geq$ *0*
by (*meson eint-ord-trans eint-pos-int-times-ge val-of-nat-inc zero-less-numeral*)
thus *?thesis*
using *assms* **unfolding** *val-one*

by (*simp add*: Q_p-*def*)
qed
hence *8*: *val a* = *val* **1**
using *a-closed*
by (*metis Qp.cring-simprules*(*6*) *ultrametric-equal-eq'*)
have *9*:*val* (*a* [^] (*n* − *1*)) = *0*
by (*simp add*: *8 local.a-closed val-zero-imp-val-pow-zero*)
have *10*: *val* ([*n*]·**1** ⊗ **1**[^](*n*−*1*)) = *val* ([*n*]·**1**)
unfolding *val-one 9* **by** *simp*
have *11*: *0* ≤ *gauss-norm f*
proof−
have *p0*: *gauss-norm* (*up-ring.monom* (*UP* Q_p) **1** *n*) ≥ *0*
using *gauss-norm-monom* **by** *simp*
have *p1*: *gauss-norm* (*up-ring.monom* (*UP* Q_p) *a 0*) ≥ *0*
using *gauss-norm-monom assms val-ring-memE* **by** *simp*
have *p2*: *min* (*gauss-norm* (*up-ring.monom* (*UP* Q_p) **1** *n*)) (*gauss-norm* (*up-ring.monom* (*UP* Q_p) *a 0*)) ≥ *0*
using *p0 p1* **by** *simp*
have *p3*: *0* ≤ *gauss-norm*
(*up-ring.monom* (*UP* Q_p) **1** *n* ⊖$_{UP\ Q_p}$ *up-ring.monom* (*UP* Q_p) *a 0*)
using *gauss-norm-ultrametric'*[*of up-ring.monom* (*UP* Q_p) **1** *n up-ring.monom* (*UP* Q_p) *a 0*]
p2 2 3 eint-ord-trans **by** *blast*
show *?thesis* **using** *p3* **unfolding** *f-def* **by** *simp*
qed
have *12*: ⋀α. α ∈ $\mathcal{O}_p$ ⟹ *f* · α = α[^]*n* ⊖ *a*
unfolding *f-def* **using** *a-closed*
by (*simp add*: *UPQ.to-fun-const UPQ.to-fun-diff UPQ.to-fun-monic-monom val-ring-memE*(*2*))
have *13*: ∃!α. α ∈ $\mathcal{O}_p$ ∧ *f* · α = **0** ∧ *val* (*UPQ.pderiv f* · **1**) < *val* (**1** ⊖ α)
apply(*rule hensels-lemma*, *rule f-closed*, *rule one-in-val-ring*, *rule 11*)
unfolding *6 10*
using *a-closed assms 12*[*of* **1**] *assms*(*3*)
by (*simp add*: *one-in-val-ring*)
have *14*: ⋀α. α ∈ $\mathcal{O}_p$ ⟹ α[^]*n* = *a* ⟷ *f* · α = **0**
unfolding *f-def* **using** *a-closed 12 f-def val-ring-memE*(*2*) **by** *auto*
have *15*: *val* (*UPQ.pderiv f* · **1**) = *val* ([*n*]·**1**)
unfolding *6 10* **by** *auto*
have *16*: ⋀α. α ∈ $\mathcal{O}_p$ ⟹ *val* (**1** ⊖ α) = *val* (α ⊖ **1**)
proof−
have *17*: ⋀α. α ∈ $\mathcal{O}_p$ ⟹ (**1** ⊖ α) = ⊖ (α ⊖ **1**)
using *val-ring-memE*
by (*meson Qp.minus-a-inv Qp.one-closed*)
show ⋀α. α ∈ $\mathcal{O}_p$ ⟹ *val* (**1** ⊖ α) = *val* (α ⊖ **1**)
unfolding *17*
using *Qp.minus-closed Qp.one-closed val-minus val-ring-memE*(*2*) **by** *presburger*
qed
show *?thesis* **using** *13* **unfolding** *15* **using** *14 16 Qp.one-closed val-ring-memE*(*2*)

by *metis*
qed

lemma *mod-zeroE*:
assumes (*a*::*int*) *mod k* = *0*
shows ∃ *l*. *a* = *l*∗*k*
using *assms*
using *Groups.mult-ac*(*2*) **by** *blast*

lemma *to-Zp-poly-closed'*:
assumes *g* ∈ *carrier* (*UP* Q_p)
assumes ⋀*i*. *g i* ∈ $\mathcal{O}_p$
shows *to-Zp-poly g* ∈ *carrier* (*UP* Z_p)
proof(*rule to-Zp-poly-closed*)
show *g* ∈ *carrier* (*UP* Q_p)
using *assms*(*1*) **by** *blast*
show *0* ≤ *gauss-norm g*
proof−
have ⋀*i*. *val* (*g i*) ≥ *0*
using *assms val-ring-memE* **by** *blast*
thus *?thesis* **unfolding** *gauss-norm-def*
by (*metis gauss-norm-coeff-norm gauss-norm-def*)
qed
qed

lemma *to-Zp-poly-eval-to-Zp*:
assumes *g* ∈ *carrier* (*UP* Q_p)
assumes ⋀*i*. *g i* ∈ $\mathcal{O}_p$
assumes *a* ∈ $\mathcal{O}_p$
shows *to-function* Z_p (*to-Zp-poly g*) (*to-Zp a*) = *to-Zp* (*g* ⋅ *a*)
proof−
have (∀ *i*. *g i* ∈ $\mathcal{O}_p$) ⟶ *to-function* Z_p (*to-Zp-poly g*) (*to-Zp a*) = *to-Zp* (*g* ⋅ *a*)
apply(*rule UPQ.poly-induct*[*of g*]) **using** *assms* **apply** *blast*
proof
fix *p* **assume** *A*: *p* ∈ *carrier* (*UP* Q_p) *deg* Q_p *p* = *0* ∀ *i*. *p i* ∈ $\mathcal{O}_p$
obtain *c* **where** *c-def*: *c* ∈ *carrier* Q_p ∧ *p* = *up-ring.monom* (*UP* Q_p) *c* *0*
using *A* **by** (*metis UPQ.ltrm-deg-0 val-ring-memE*(*2*))
have *0*: *to-Zp-poly* (*up-ring.monom* (*UP* Q_p) *c* *0*) = *up-ring.monom* (*UP* Z_p) (*to-Zp c*) *0*
unfolding *to-Zp-poly-def* **proof fix** *n* **show** *to-Zp* (*up-ring.monom* (*UP* Q_p) *c 0 n*) = *up-ring.monom* (*UP* Z_p) (*to-Zp c*) *0 n*
using *UP-ring.cfs-monom*[*of* Z_p *to-Zp c 0 n*] *UP-ring.cfs-monom*[*of* Q_p *c 0 n*] *to-Zp-closed*[*of c*]
unfolding *UP-ring-def*
apply(*cases 0* = *n*)
using *UPQ.cfs-monom Zp.cfs-monom c-def* **apply** *presburger*
using *UPQ.cfs-monom Zp.cfs-monom c-def*
using *to-Zp-zero* **by** *presburger*
qed

have *p-eq*: $p = up\text{-}ring.monom\ (UP\ Q_p)\ c\ \ 0$
using *c-def* **by** *blast*
have *1*: $(up\text{-}ring.monom\ (UP\ Q_p)\ c\ 0 \cdot a) = c$
using *UPQ.to-fun-to-poly*[*of c a*] *c-def assms val-ring-memE*
unfolding *to-polynomial-def* **by** *blast*
show *to-function* Z_p (*to-Zp-poly p*) (*to-Zp a*) = *to-Zp* ($p \cdot a$)
using *c-def assms*(*3*) *val-ring-memE*(*2*)[*of a*]
UP-cring.to-fun-to-poly[*of* Z_p *to-Zp c to-Zp a*]
unfolding *p-eq 0 1 Zp.to-fun-def to-polynomial-def*
using *Zp.UP-cring-axioms to-Zp-closed* **by** *blast*
next
show $\bigwedge p.\ (\bigwedge q.\ q \in carrier\ (UP\ Q_p) \Longrightarrow$
deg Q_p q < *deg* Q_p $p \Longrightarrow (\forall i.\ q\ i \in \mathcal{O}_p) \longrightarrow$ *to-function* Z_p (*to-Zp-poly* q) (*to-Zp a*) = *to-Zp* ($q \cdot a$)) $\Longrightarrow$
$p \in carrier\ (UP\ Q_p) \Longrightarrow 0 < deg\ Q_p\ p \Longrightarrow (\forall i.\ p\ i \in \mathcal{O}_p) \longrightarrow$ *to-function* Z_p (*to-Zp-poly p*) (*to-Zp a*) = *to-Zp* ($p \cdot a$)
proof fix p
assume *A*: $(\bigwedge q.\ q \in carrier\ (UP\ Q_p) \Longrightarrow$
deg Q_p q < *deg* Q_p $p \Longrightarrow (\forall i.\ q\ i \in \mathcal{O}_p) \longrightarrow$ *to-function* Z_p (*to-Zp-poly* q) (*to-Zp a*) = *to-Zp* ($q \cdot a$))
$p \in carrier\ (UP\ Q_p)$ $0 < deg\ Q_p\ p\ \forall i.\ p\ i \in \mathcal{O}_p$
show *to-function* Z_p (*to-Zp-poly p*) (*to-Zp a*) = *to-Zp* ($p \cdot a$)
proof−
obtain q **where** *q-def*: $q = truncate\ Q_p\ \ p$
by *blast*
have *q-closed*: $q \in carrier\ (UP\ Q_p)$
unfolding *q-def* **by**(*rule UPQ.trunc-closed*, *rule A*)
obtain c **where** *c-def*: $c = UPQ.lcf\ p$
by *blast*
obtain n **where** *n-def*: $n = deg\ Q_p\ p$
by *blast*
have *0*: $p = q \oplus_{UP\ Q_p} up\text{-}ring.monom\ (UP\ Q_p)\ c\ n$
unfolding *c-def n-def q-def*
using *A*(*2*) *UPQ.trunc-simps*(*1*) **by** *blast*
have *1*: *up-ring.monom* (*UP* Q_p) *c n* ∈ *carrier* (*UP* Q_p)
using *A*(*2*) *UPQ.ltrm-closed c-def n-def* **by** *blast*
have *2*: $p \cdot a\ = q\ \cdot a \oplus (c \otimes a[\hat{}]n)$
unfolding *0* **using** *assms val-ring-memE*
by (*metis 1 A*(*4*) *UPQ.to-fun-monom UPQ.to-fun-plus c-def q-closed*)
have *3*: $\bigwedge i.\ i < n \Longrightarrow q\ i = p\ i$
unfolding *n-def q-def*
using *A*(*2*) *UPQ.trunc-cfs* **by** *blast*
have *4*: *deg* Q_p $q < n$
unfolding *n-def q-def* **using** *A*
using *UPQ.trunc-degree* **by** *presburger*
have *5*: $\bigwedge i.\ i \geq n \Longrightarrow i >\ deg\ Q_p\ q$
using *A*[*of*] *less-le-trans*[*of deg* Q_p *q deg* Q_p *p*] **unfolding** *q-def n-def*
using *4 n-def q-def* **by** *blast*
have *6*: $\bigwedge i.\ i \geq n \Longrightarrow q\ i = \mathbf{0}$

using *q-closed 5 UPQ.deg-leE* **by** *blast*
have *7*: $(\forall\, i.\ q\ i \in \mathcal{O}_p) \longrightarrow$ *to-function* Z_p (*to-Zp-poly q*) (*to-Zp a*) = *to-Zp* ($q \cdot a$)
apply(*rule A*) **unfolding** *q-def*
using *q-closed q-def* **apply** *blast*
using *4 n-def q-def* **by** *blast*
have *8*: $(\forall\, i.\ q\ i \in \mathcal{O}_p)$
proof fix *i* **show** *q i* $\in \mathcal{O}_p$ **apply**(*cases i* < *n*)
using *3 A*(*4*) **apply** *blast* **using** *6*[*of i*]
by (*metis less-or-eq-imp-le linorder-neqE-nat zero-in-val-ring*)
qed
have *9*: *to-function* Z_p (*to-Zp-poly q*) (*to-Zp a*) = *to-Zp* ($q \cdot a$)
using *7 8* **by** *blast*
have *10*: *to-Zp-poly p* = *to-Zp-poly q* $\oplus_{UP\ Z_p}$ *to-Zp-poly* (*up-ring.monom* (*UP* Q_p) *c n*)
proof fix *x*
have *100*: *to-Zp-poly* (*up-ring.monom* (*UP* Q_p) *c n*) = (*up-ring.monom* (*UP* Z_p) (*to-Zp c*) *n*)
using *to-Zp-poly-monom*[*of c*] *A*(*4*) *c-def* **by** *blast*
have *101*: *deg* Z_p (*to-Zp-poly q*) ≤ *n*−*1*
apply(*rule UP-cring.deg-leqI*)
unfolding *UP-cring-def* **using** *Zp.R-cring* **apply** *auto*[*1*]
using *to-Zp-poly-closed′ 8 q-closed* **apply** *blast*
unfolding *to-Zp-poly-def* **using** *4 6*
by (*simp add*: *to-Zp-zero*)
have *102*: (*to-Zp-poly q*) ∈ *carrier* (*UP* Z_p)
apply(*rule to-Zp-poly-closed′*, *rule q-closed*) **using** *8* **by** *blast*
have *103*: *deg* Z_p (*to-Zp-poly q*) < *n*
using *101 4* **by** *linarith*
have *T0*: (*to-Zp-poly q* $\oplus_{UP\ Z_p}$ *to-Zp-poly* (*up-ring.monom* (*UP* Q_p) *c n*)) *x* =
(*to-Zp-poly q x*) $\oplus_{Z_p}$ (*to-Zp-poly* (*up-ring.monom* (*UP* Q_p) *c n*) *x*)
apply(*rule UP-ring.cfs-add*)
apply (*simp add*: *Zp.is-UP-ring*)
apply (*simp add*: *102*)
using *100 A*(*2*) *UPQ.lcf-closed c-def to-Zp-closed* **by** *auto*
have *c-closed*: *c* $\in \mathcal{O}_p$
unfolding *c-def* **using** *A*(*4*) **by** *blast*
have *to-Zp-c-closed*: *to-Zp c* ∈ *carrier* Z_p
using *c-closed to-Zp-closed val-ring-memE*(*2*) **by** *blast*
show *to-Zp-poly p x* = (*to-Zp-poly q* $\oplus_{UP\ Z_p}$ *to-Zp-poly* (*up-ring.monom* (*UP* Q_p) *c n*)) *x*
proof(*cases x* < *n*)
case *True*
have *T1*: (*to-Zp-poly* (*up-ring.monom* (*UP* Q_p) *c n*) *x*) = $\mathbf{0}_{Z_p}$
using *True UP-ring.cfs-monom*[*of* Z_p] **unfolding** *UP-ring-def*
by (*simp add*: *A*(*2*) *UPQ.ltrm-cfs c-def n-def to-Zp-poly-def to-Zp-zero*)
have *T2*: *to-Zp* (*p x*) = *to-Zp* (*q x*) **using** *3*[*of x*] *True* **by** *smt*
have *T3*: *to-Zp* (*p x*) ∈ *carrier* Z_p

apply(*rule to-Zp-closed*) **using** *A*(*2*) *UPQ.UP-car-memE*(*1*) **by** *blast*
show *?thesis* **using** *T3*
unfolding *T0* **unfolding** *T1* **unfolding** *to-Zp-poly-def T2*
using *Zp.cring-simprules*(*8*) *add-comm* **by** *presburger*
next
case *False*
have *F*: *q x* = **0**
using *False*
by (*metis 6 less-or-eq-imp-le linorder-neqE-nat*)
have *F′*: (*to-Zp-poly q*) *x* = $\mathbf{0}_{Z_p}$
unfolding *to-Zp-poly-def F* **using** *to-Zp-zero* **by** *blast*
show *to-Zp-poly p x* = (*to-Zp-poly q* $\oplus_{UP\ Z_p}$ *to-Zp-poly* (*up-ring.monom* (*UP* Q_p) *c n*)) *x*
proof(*cases x* = *n*)
case *True*
have *T1*: *to-Zp* (*p x*) ∈ *carrier* Z_p
apply(*rule to-Zp-closed*)
using *A*(*2*) *UPQ.UP-car-memE*(*1*) **by** *blast*
have *T2*: (*to-Zp-poly* (*up-ring.monom* (*UP* Q_p) *c n*) *x*) = *to-Zp c*
unfolding *100* **using** *UP-ring.cfs-monom*[*of* Z_p *to-Zp c n n*] **unfolding** *UP-ring-def True*
using *Zp.ring-axioms to-Zp-c-closed* **by** *presburger*
show *?thesis* **using** *to-Zp-c-closed* **unfolding** *T0 F′ T2* **unfolding** *to-Zp-poly-def True c-def n-def*
using *Zp.cring-simprules*(*8*) **by** *presburger*
next
case *FF*: *False*
have *F0*: *p x* = **0**
using *FF False* **unfolding** *n-def*
using *A*(*2*) *UPQ.UP-car-memE*(*2*) *linorder-neqE-nat* **by** *blast*
have *F1*: *q x* = **0**
using *FF False F* **by** *linarith*
have *F2*: (*up-ring.monom* (*UP* Q_p) *c n*) *x* = **0**
using *FF False A*(*2*) *UPQ.cfs-closed UPQ.cfs-monom c-def* **by** *presburger*
show *?thesis* **unfolding** *T0* **unfolding** *to-Zp-poly-def F0 F1 F2*
using *Zp.r-zero Zp.zero-closed to-Zp-zero* **by** *presburger*
qed
qed
qed
have *11*: *deg* Z_p (*to-Zp-poly q*) ≤ *n*−*1*
apply(*rule UP-cring.deg-leqI*)
unfolding *UP-cring-def* **using** *Zp.R-cring* **apply** *auto*[*1*]
using *to-Zp-poly-closed′ 8 q-closed* **apply** *blast*
unfolding *to-Zp-poly-def* **using** *4 6*
by (*smt diff-commute diff-diff-cancel less-one less-or-eq-imp-le linorder-neqE-nat to-Zp-zero zero-less-diff*)
have *12*: (*to-Zp-poly q*) ∈ *carrier* (*UP* Z_p)
apply(*rule to-Zp-poly-closed′*, *rule q-closed*) **using** *8* **by** *blast*

have *13*: *deg* Z_p (*to-Zp-poly q*) < *n*
using *11 4* **by** *linarith*
have *14*: *to-Zp-poly* (*up-ring.monom* (*UP* Q_p) *c n*) = (*up-ring.monom* (*UP* Z_p) (*to-Zp c*) *n*)
using *to-Zp-poly-monom*[*of c*] *A*(*4*) *c-def* **by** *blast*
have *15*: *Zp.to-fun* (*to-Zp-poly q* $\oplus_{UP\ Z_p}$ *to-Zp-poly* (*up-ring.monom* (*UP* Q_p) *c n*)) (*to-Zp a*)=
Zp.to-fun (*to-Zp-poly q*) (*to-Zp a*) $\oplus_{Z_p}$ *Zp.to-fun* (*to-Zp-poly* (*up-ring.monom* (*UP* Q_p) *c n*)) (*to-Zp a*)
apply(*rule Zp.to-fun-plus*)
unfolding *14* **apply**(*rule UP-ring.monom-closed*)
unfolding *UP-ring-def*
apply (*simp add*: *Zp.ring-axioms*)
apply (*simp add*: *A*(*2*) *UPQ.cfs-closed c-def to-Zp-closed*)
using *12* **apply** *blast*
apply(*rule to-Zp-closed*) **using** *assms val-ring-memE* **by** *blast*
have *16*: *to-Zp* (*q* · *a* ⊕ *c* ⊗ *a* [^] *n*) = *to-Zp* (*q* · *a*) $\oplus_{Z_p}$ *to-Zp* (*c* ⊗ *a* [^] *n*)
apply(*rule to-Zp-add*)
apply(*rule val-ring-poly-eval*, *rule q-closed*)
using *8* **apply** *blast*
apply(*rule assms*)
apply(*rule val-ring-times-closed*)
unfolding *c-def* **using** *A*(*4*) **apply** *blast*
by(*rule val-ring-nat-pow-closed*, *rule assms*)
have *17*: *to-function* Z_p (*up-ring.monom* (*UP* Z_p) (*to-Zp c*) *n*) (*to-Zp a*) = *to-Zp* (*c* ⊗ *a* [^] *n*)
proof−
have *170*: *to-Zp* (*c* ⊗ *a* [^] *n*) = *to-Zp c* $\otimes_{Z_p}$ *to-Zp* (*a* [^] *n*)
apply(*rule to-Zp-mult*[*of c a*[^]*n*])
unfolding *c-def* **using** *A*(*4*) **apply** *blast*
by(*rule val-ring-nat-pow-closed*, *rule assms*)
have *171*: *to-Zp* (*a* [^] *n*) = (*to-Zp a* [^]$_{Z_p}$ *n*)
by(*rule to-Zp-nat-pow*, *rule assms*)
have *172*: *to-Zp c* ∈ *carrier* Z_p
apply(*rule to-Zp-closed*) **unfolding** *c-def*
using *A*(*2*) *UPQ.UP-car-memE*(*1*) **by** *blast*
have *173*: *to-Zp a* ∈ *carrier* Z_p
apply(*rule to-Zp-closed*) **using** *assms val-ring-memE* **by** *blast*
show *?thesis*
using *172 173 Zp.to-fun-monom*[*of to-Zp c to-Zp a n*] **unfolding** *Zp.to-fun-def 170 171*
by *blast*
qed
show *?thesis*
using *15* **unfolding** *Zp.to-fun-def 10 2 16 9* **unfolding** *14 17*
by *blast*
qed
qed

qed
thus *?thesis* **using** *assms* **by** *blast*
qed

lemma *inc-nat-pow*:
assumes $a \in$ *carrier* Z_p
shows ι $([(n::nat)] \cdot_{Z_p} a) = [n] \cdot (\iota\ a)$
apply(*induction n*)
apply (*metis* Q_p*-def Qp.int-inc-zero Qp.nat-mult-zero Zp.add.nat-pow-0 Zp-int-inc-zero′ ι-def frac-inc-of-int*)
unfolding *Qp.add.nat-pow-Suc Zp.add.nat-pow-Suc*
using *Zp-nat-mult-closed assms inc-of-sum* **by** *presburger*

lemma *poly-inc-pderiv*:
assumes $g \in$ *carrier* (*UP* Z_p)
shows *poly-inc* (*Zp.pderiv g*) = *UPQ.pderiv* (*poly-inc g*)
proof fix *x*
have *0*: *UPQ.pderiv* (*poly-inc g*) *x* = [*Suc x*] · *poly-inc g* (*Suc x*)
apply(*rule UPQ.pderiv-cfs*[*of poly-inc g x*])
by(*rule poly-inc-closed*, *rule assms*)
have *1*: *Zp.pderiv g x* = [*Suc x*] $\cdot_{Z_p}$ *g* (*Suc x*)
by(*rule Zp.pderiv-cfs*[*of g x*], *rule assms*)
show *poly-inc* (*Zp.pderiv g*) *x* = *UPQ.pderiv* (*poly-inc g*) *x*
unfolding *0* **unfolding** *poly-inc-def 1* **apply**(*rule inc-nat-pow*)
using *Zp.UP-car-memE*(*1*) *assms* **by** *blast*
qed

lemma *Zp-hensels-lemma*:
assumes *f* ∈ *carrier Zp-x*
assumes *a* ∈ *carrier* Z_p
assumes *Zp.to-fun* (*Zp.pderiv f*) *a* ≠ $\mathbf{0}_{Z_p}$
assumes *Zp.to-fun f a* ≠ $\mathbf{0}_{Z_p}$
assumes *val-Zp* (*Zp.to-fun f a*) > *eint 2* ∗ *val-Zp* (*Zp.to-fun* (*Zp.pderiv f*) *a*)
obtains α **where**
Zp.to-fun f α = $\mathbf{0}_{Z_p}$ **and** $\alpha \in$ *carrier* Z_p
val-Zp (*a* $\ominus_{Z_p}$ α) > *val-Zp* (*Zp.to-fun* (*Zp.pderiv f*) *a*)
val-Zp (*a* $\ominus_{Z_p}$ α) = *val-Zp* (*divide* (*Zp.to-fun f a*) (*Zp.to-fun* (*Zp.pderiv f*) *a*))
val-Zp (*Zp.to-fun* (*Zp.pderiv f*) α) = *val-Zp* (*Zp.to-fun* (*Zp.pderiv f*) *a*)
proof−
have *hensel p f a*
using *assms*
by (*simp add*: *Zp-def hensel.intro hensel-axioms.intro padic-integers-axioms*)
then show *?thesis*
using *hensel.full-hensels-lemma*[*of p f a*] *that*
unfolding *Zp-def*
by *blast*
qed

end
end
theory *Padic-Field-Topology*
imports *Padic-Fields*
begin

10 Topology of p-adic Fields

In this section we develop some basic properties of the topology on the p-adics. Open and closed sets are defined, convex subsets of the value group are characterized.

type-synonym *padic-univ-poly* = *nat* ⇒ *padic-number*

10.1 p-adic Balls

context *padic-fields*
begin

definition *c-ball* :: *int* ⇒ *padic-number* ⇒ *padic-number set* (‹B_{-}[-]›) **where**
c-ball n c = {$x \in$ *carrier* Q_p. *val* ($x \ominus c$) ≥ n}

lemma *c-ballI*:
assumes $x \in$ *carrier* Q_p
assumes *val* ($x \ominus c$) ≥ n
shows $x \in$ *c-ball n c*
using *assms c-ball-def*
by *blast*

lemma *c-ballE*:
assumes $x \in$ *c-ball n c*
shows $x \in$ *carrier* Q_p
val ($x \ominus c$) ≥ n
using *assms c-ball-def* **apply** *blast*
using *assms c-ball-def* **by** *blast*

lemma *c-ball-in-Qp*:
$B_n[c] \subseteq$ *carrier* Q_p
unfolding *c-ball-def*
by *blast*

definition
q-ball :: *nat* ⇒ *int* ⇒ *int* ⇒ *padic-number* ⇒ *padic-number set* **where**
q-ball n k m c = {$x \in$ *carrier* Q_p. (*ac n* ($x \ominus c$) = $k \wedge$ (*ord* ($x \ominus c$)) = m) }

lemma *q-ballI*:
assumes $x \in$ *carrier* Q_p
assumes *ac n* ($x \ominus c$) = k
assumes (*ord* ($x \ominus c$)) = m

shows $x \in$ *q-ball n k m c*
using *assms q-ball-def*
by *blast*

lemma *q-ballE*:
assumes $x \in$ *q-ball n k m c*
shows $x \in$ *carrier* Q_p

using *assms q-ball-def* **by** *blast*

lemma *q-ballE'*:
assumes $x \in$ *q-ball n k m c*
shows *ac n* $(x \ominus c) = k$
(ord $(x \ominus c)) = m$
using *assms q-ball-def* **apply** *blast*
using *assms q-ball-def* **by** *blast*

lemma *q-ball-in-Qp*:
q-ball n k m c $\subseteq$ *carrier* Q_p
unfolding *q-ball-def* **by** *blast*

lemma *ac-ord-prop*:
assumes $a \in$ *nonzero* Q_p
assumes $b \in$ *nonzero* Q_p
assumes *ord a* $=$ *ord b*
assumes *ord a* $= n$
assumes *ac m a* $=$ *ac m b*
assumes $m > 0$
shows *val* $(a \ominus b) \geq m + n$
proof−
have *0*: $a = (\mathfrak{p}[\hat{}]n) \otimes \iota$ *(angular-component a)*
using *angular-component-factors-x assms(1) assms(4)* **by** *blast*
have *1*: $b = (\mathfrak{p}[\hat{}]n) \otimes \iota$ *(angular-component b)*
using *angular-component-factors-x assms(4) assms(2) assms(3)*
by *presburger*
have *2*: $a \ominus b = (\mathfrak{p}[\hat{}]n) \otimes \iota$ *(angular-component a)* $\ominus$
$(\mathfrak{p}[\hat{}]n) \otimes \iota$ *(angular-component b)*
using *0 1* **by** *auto*
have *3*: $a \ominus b = (\mathfrak{p}[\hat{}]n) \otimes (\ \iota$ *(angular-component a)* $\ominus\ \iota$ *(angular-component b))*
proof−
have *30*: $(\mathfrak{p}[\hat{}]n) \in$ *carrier* Q_p
by *(simp add: p-intpow-closed(1))*
have *31*: ι *(angular-component a)* $\in$ *carrier* Q_p
using *Zp.nonzero-one-closed angular-component-closed assms(1) frac-closed local.inc-def*
by *presburger*
have *32*: ι *(angular-component b)* $\in$ *carrier* Q_p
using *Zp.nonzero-one-closed angular-component-closed assms(2) frac-closed local.inc-def*

by *presburger*
show *?thesis*
using *2 30 31 32 ring.ring-simprules(23)[of* Q_p $(\mathfrak{p}[\hat{}]n)]$
unfolding *a-minus-def*
by (*metis Qp.domain-axioms cring.cring-simprules(25) cring.cring-simprules(29)*

cring.cring-simprules(3) domain.axioms(1))
qed
have *4*: $a \ominus b = (\mathfrak{p}[\hat{}]n) \otimes (\ \iota\ ((\textit{angular-component}\ a) \ominus_{Z_p} (\textit{angular-component}\ b)))$
using *3*
by (*simp add*: *angular-component-closed assms(1) assms(2) inc-of-diff*)
have *5*: *val-Zp* $((\textit{angular-component}\ a) \ominus_{Z_p} (\textit{angular-component}\ b)) \geq m$
proof−
have $((\textit{angular-component}\ a) \ominus_{Z_p} (\textit{angular-component}\ b))\ m = 0$
using *assms(5)*
unfolding *ac-def*
using Q_p*-def Qp.nonzero-memE(2) angular-component-closed assms(1) assms(2) residue-of-diff'*
by *auto*
then show *?thesis*
using *Zp.minus-closed angular-component-closed assms(1) assms(2) ord-Zp-geq val-Zp-def val-ord-Zp*
by *auto*
qed
have *6*: $val\ (a \ominus b) \geq n + val\ (\ \iota\ (\textit{angular-component}\ a) \ominus\ \iota\ (\textit{angular-component}\ b))$
using *3 Qp.minus-closed angular-component-closed assms(1) assms(2) inc-closed*

ord-p-pow-int p-intpow-closed(1) p-intpow-closed(2) val-mult val-ord
by *simp*
have *7*: $n + val\ (\ \iota\ (\textit{angular-component}\ a) \ominus\ \iota\ (\textit{angular-component}\ b))$
$= n + \textit{val-Zp}\ ((\textit{angular-component}\ a) \ominus_{Z_p} (\textit{angular-component}\ b))$
using *Zp.minus-closed angular-component-closed assms(1) assms(2) inc-of-diff val-of-inc*
by *simp*
have *8*: $n + \textit{val-Zp}\ (\ (\textit{angular-component}\ a) \ominus_{Z_p} (\textit{angular-component}\ b))$
$\geq n + m$
using *5*
by (*metis add-mono-thms-linordered-semiring(2) plus-eint-simps(1)*)
then have *9*: $n + val\ (\ \iota\ (\textit{angular-component}\ a) \ominus\ \iota\ (\textit{angular-component}\ b))$
$\geq n + m$
using *7* **by** *presburger*
then show *?thesis*
by (*metis* (*no-types*, *opaque-lifting*) *6 add.commute order-trans plus-eint-simps(1)*)

qed

lemma *c-ball-q-ball*:

assumes $b \in nonzero\ Q_p$
assumes $n > 0$
assumes $k = ac\ n\ b$
assumes $c \in carrier\ Q_p$
assumes $d \in q\text{-}ball\ n\ k\ m\ c$
shows $q\text{-}ball\ n\ k\ m\ c = c\text{-}ball\ (m + n)\ d$
proof
show $q\text{-}ball\ n\ k\ m\ c \subseteq B_{m + int\ n}[d]$
proof
fix x
assume *A0*: $x \in q\text{-}ball\ n\ k\ m\ c$
show $x \in B_{m + int\ n}[d]$
proof−
have *A1*: $(ac\ n\ (x \ominus c) = k \wedge (ord\ (x \ominus c)) = m)$
using *A0 q-ball-def*
by *blast*
have $val\ (x \ominus d) \geq m + n$
proof−
have *A2*: $(ac\ n\ (d \ominus c) = k \wedge (ord\ (d \ominus c)) = m)$
using *assms*(*5*) *q-ball-def*
by *blast*
have *A3*: $(x \ominus c) \in nonzero\ Q_p$
proof−
have $k \neq 0$
using *A2 assms*(*1*) *assms*(*3*) *assms*(*5*) *ac-units*[*of b n*]
by (*metis One-nat-def Suc-le-eq assms*(*2*) *zero-not-in-residue-units*)

then show *?thesis*
by (*smt A0 Qp.domain-axioms ac-def assms*(*4*) *cring.cring-simprules*(*4*)
domain.axioms(*1*)
mem-Collect-eq not-nonzero-Qp q-ball-def)
qed
have *A4*: $(d \ominus c) \in nonzero\ Q_p$
proof−
have $k \neq 0$
using *A2 assms*(*1*) *assms*(*3*) *assms*(*5*) *ac-units*[*of b n*]
by (*metis One-nat-def Suc-le-eq assms*(*2*) *zero-not-in-residue-units*)
then show *?thesis*
by (*metis* (*no-types*, *lifting*) *A2 Qp.domain-axioms ac-def assms*(*4*)
assms(*5*)
cring.cring-simprules(*4*) *domain.axioms*(*1*) *mem-Collect-eq not-nonzero-Qp*
q-ball-def)
qed
then have $val\ ((x \ominus c) \ominus (d \ominus c)) \geq n + m$
using *ac-ord-prop*[*of* $(x \ominus c)$ $(d \ominus c)$ *m n*] *A1 A2 assms A3*
by *simp*
then show *?thesis*
by (*smt A0 Qp-diff-diff assms*(*4*) *assms*(*5*) *q-ballE*)
qed

```
    then show ?thesis
      by (metis (no-types, lifting) A0 c-ball-def mem-Collect-eq q-ball-def)
  qed
qed
show B_{m + int n}[d] ⊆ q-ball n k m c
proof
  fix x
  assume A: x ∈ B_{m + int n}[d]
  show x ∈ q-ball n k m c
  proof−
    have A0: val (x ⊖ d) ≥ m + n
      using A c-ball-def
      by blast
    have A1: ord (d ⊖ c) = m
      using assms(5) q-ball-def
      by blast
    have A2: ac n (d ⊖ c) = k
      using assms(5) q-ball-def
      by blast
    have A3: (d ⊖ c) ≠ 0
      using A2 assms
    by (metis ac-def ac-units le-eq-less-or-eq le-neq-implies-less less-one nat-le-linear

        padic-integers.zero-not-in-residue-units padic-integers-def prime zero-less-iff-neq-zero)

    have A4: val (d ⊖ c) =m
      by (simp add: A1 A3 val-def)
    have A5: val (x ⊖ d) > val (d ⊖ c)
    by (smt A0 A4 assms(2) eint-ord-code(4) eint-ord-simps(1) eint-ord-simps(2)
of-nat-0-less-iff val-def)
    have A6: val ((x ⊖ d) ⊕ (d ⊖ c)) = m
      using A4 A0 A5
     by (metis (mono-tags, opaque-lifting) A Qp.minus-closed assms(4) assms(5)

         c-ballE(1) q-ballE val-ultrametric-noteq)
    have A7: val (x ⊖ c) = m
    proof−
      have (x ⊖ d) ⊕ (d ⊖ c) = ((x ⊖ d) ⊕ d) ⊖ c
        by (metis (no-types, lifting) A Qp.domain-axioms a-minus-def assms(4)
assms(5)
            c-ball-def cring.cring-simprules(3) cring.cring-simprules(4)
            cring.cring-simprules(7) domain.axioms(1) mem-Collect-eq q-ball-def)
      have (x ⊖ d) ⊕ (d ⊖ c) = (x ⊕ (⊖ d ⊕ d)) ⊖ c
       by (metis (no-types, opaque-lifting) A Qp.add.l-cancel-one Qp.add.m-comm
Qp.l-neg
            Qp.minus-closed Qp.plus-diff-simp Qp.zero-closed assms(4) assms(5)
            c-ballE(1) q-ballE)
      then show ?thesis
       by (metis (no-types, lifting) A A6 Qp.domain-axioms assms(5) c-ball-def
```

cring.cring-simprules(16) cring.cring-simprules(9) domain.axioms(1) mem-Collect-eq q-ball-def)
qed
have *A8*: *ac n* (*x* ⊖ *c*) = *ac n* (*d* ⊖ *c*)
proof−
have *A80*: (*x* ⊖ *c*) ∈ *nonzero* Q_p
by (*metis* (*no-types*, *lifting*) *A A4 A5 A7 Qp.domain-axioms assms(4) cring.cring-simprules(4) domain.axioms(1) mem-Collect-eq c-ball-def val-nonzero*)
have *A81*: (*d* ⊖ *c*) ∈ *nonzero* Q_p
by (*metis* (*no-types*, *lifting*) *A3 Qp.domain-axioms assms(4) assms(5) cring.cring-simprules(4) domain.axioms(1) mem-Collect-eq not-nonzero-Qp q-ball-def*)
have *A82*: *n* + *m*= *val* (*x* ⊖ *c*) + *n*
by (*simp add*: *A7*)
show *?thesis*
using *A0 A4 A7 ac-val*[*of* (*x* ⊖ *c*) (*d* ⊖ *c*) *n*] *A A80 A81 Qp-diff-diff assms(4) assms(5) c-ballE(1) q-ballE*
by *auto*
qed
show *?thesis* **using** *A8 A3 A7 A2 q-ball-def*[*of n k m c*] *q-ballI*[*of x n c k m*]

by (*metis* (*no-types*, *lifting*) *A A4 A5 Qp.minus-closed assms(4) c-ballE(1) eint.inject val-nonzero val-ord*)
qed
qed
qed

definition *is-ball* :: *padic-number set* ⇒ *bool* **where**
is-ball B = (∃(*m*::*int*). ∃ *c* ∈ *carrier* Q_p. (*B* = B_m[*c*]))

lemma *is-ball-imp-in-Qp*:
assumes *is-ball B*
shows *B* ⊆ *carrier* Q_p
unfolding *is-ball-def*
using *assms c-ball-in-Qp is-ball-def*
by *auto*

lemma *c-ball-centers*:
assumes *is-ball B*
assumes *B* = B_n[*c*]
assumes *d* ∈ *B*
assumes *c* ∈ *carrier* Q_p
shows *B* = B_n[*d*]
proof
show *B* ⊆ B_n[*d*]
proof
fix *x*
assume *A0*: *x* ∈ *B*

```
    have val (x ⊖ d) ≥ n
    proof−
      have A00: val (x ⊖ c) ≥ n
        using A0 assms(2) c-ballE(2) by blast
      have A01: val (d ⊖ c) ≥ n
        using assms(2) assms(3) c-ballE(2) by blast
      then show ?thesis
        using Qp-isosceles[of x c d n] assms A0 A00 c-ballE(1)
        by blast
    qed
    then show x ∈ B_n[d]
      using A0 assms(1) c-ballI is-ball-imp-in-Qp
      by blast
  qed
  show B_n[d] ⊆ B
  proof
    fix x
    assume x ∈ B_n[d]
    show x ∈ B
      using Qp-isosceles[of x d c n]
            assms
      unfolding c-ball-def
      by (metis (no-types, lifting) Qp.domain-axioms Qp-isosceles ‹x ∈ B_n[d]›
          a-minus-def assms(2) c-ballE(2) c-ballI cring.cring-simprules(17) do-
main.axioms(1)
        c-ballE(1))
  qed
qed

lemma c-ball-center-in:
  assumes is-ball B
  assumes B = B_n[c]
  assumes c ∈ carrier Q_p
  shows c ∈ B
  using assms  unfolding c-ball-def
  by (metis (no-types, lifting) Qp.r-right-minus-eq assms(2) c-ballI eint-ord-code(3)
local.val-zero)
```

Every point a has a point b of distance exactly n away from it.

```
lemma dist-nonempty:
  assumes a ∈ carrier Q_p
  shows ∃ b ∈ carrier Q_p. val (b ⊖ a) = eint n
proof−
  obtain b where b-def: b = (𝔭 [^] n) ⊕ a
    by simp
  have val (b ⊖a) = n
    using b-def assms
   by (metis (no-types, lifting) Qp.domain-axioms a-minus-def cring.cring-simprules(16)
       cring.cring-simprules(17) cring.cring-simprules(3) cring.cring-simprules(7)
```

domain.axioms(*1*) *ord-p-pow-int* *p-intpow-closed*(*1*) *p-intpow-closed*(*2*)
val-ord)
then show *?thesis*
by (*metis Qp.domain-axioms assms b-def cring.cring-simprules*(*1*) *domain.axioms*(*1*)
p-intpow-closed(*1*))
qed

lemma *dist-nonempty′*:
assumes $a \in$ *carrier* Q_p
shows $\exists\, b \in$ *carrier* Q_p. *val* $(b \ominus a) = \alpha$
proof(*cases* $\alpha = \infty$)
case *True*
then have *val* $(a \ominus a) = \alpha$
using *assms*
by (*metis Qp.r-right-minus-eq local.val-zero*)
then show *?thesis*
using *assms*
by *blast*
next
case *False*
then obtain n **where** *n-def*: *eint* $n = \alpha$
by *blast*
then show *?thesis*
using *assms dist-nonempty*[*of a n*]
by *blast*
qed

lemma *ball-rad-0*:
assumes *is-ball B*
assumes $B_m[c] \subseteq B_n[c]$
assumes $c \in$ *carrier* Q_p
shows $n \leq m$
proof−
obtain b **where** *b-def*: $b \in$ *carrier* $Q_p \wedge$ *val* $(b \ominus c) = m$
by (*meson assms*(*3*) *dist-nonempty*)
then have $b \in B_n[c]$
using *assms c-ballI*
by *auto*

then have $m \geq n$
using Q_p*-def Zp-def b-def c-ballE*(*2*) *padic-integers-axioms*
by *force*
then show *?thesis*
by (*simp*)
qed

lemma *ball-rad*:
assumes *is-ball B*
assumes $B = B_n[c]$

assumes $B = B_m[c]$
assumes $c \in$ *carrier* Q_p
shows $n = m$
proof−
have *0*: $n \geq m$
using *assms ball-rad-0*
by (*metis order-refl*)
have *1*: $m \geq n$
using *assms ball-rad-0*
by (*metis order-refl*)
show *?thesis*
using *0 1*
by *auto*
qed

definition *radius* :: *padic-number set* ⇒ *int* (‹*rad*›) **where**
radius $B = (SOME\ n.\ (\exists\, c \in carrier\ Q_p\ .\ B = B_n[c]))$

lemma *radius-of-ball*:
assumes *is-ball B*
assumes $c \in B$
shows $B = B_{rad\ B}[c]$
proof−
obtain *d m* **where** *d-m-def*: $d \in carrier\ Q_p \wedge\ B = B_m[d]$
using *assms*(*1*) *is-ball-def*
by *blast*
then have $B = B_m[c]$
using *assms*(*1*) *assms*(*2*) *c-ball-centers* **by** *blast*
then have *rad B* = *m*
proof−
have $\exists\, n.\ (\exists\, c \in carrier\ Q_p\ .\ B = B_n[c])$
using *d-m-def* **by** *blast*
then have $(\exists\, c \in carrier\ Q_p\ .\ B = B_{rad\ B}[c])$
using *radius-def*[*of B*]
by (*smt someI-ex*)
then show *?thesis*
using *radius-def ball-rad*[*of B m*]
by (*metis* (*mono-tags*, *lifting*) ‹$B = B_m[c]$› *assms*(*1*) *assms*(*2*) *c-ballE*(*1*)
c-ball-centers)
qed
then show *?thesis*
using ‹$B = B_m[c]$› **by** *blast*
qed

lemma *ball-rad'*:
assumes *is-ball B*
assumes $B = B_n[c]$
assumes $B = B_m[d]$
assumes $c \in carrier\ Q_p$

assumes $d \in$ *carrier* Q_p
shows $n = m$
by (*metis assms*(1) *assms*(2) *assms*(3) *assms*(4) *assms*(5) *ball-rad c-ball-center-in c-ball-centers*)

lemma *nested-balls*:
assumes *is-ball* B
assumes $B = B_n[c]$
assumes $B' = B_m[c]$
assumes $c \in$ *carrier* Q_p
assumes $d \in$ *carrier* Q_p
shows $n \geq m \longleftrightarrow B \subseteq B'$
proof
show $m \leq n \Longrightarrow B \subseteq B'$
proof
assume *A0*: $m \leq n$
then have *A0'*: $m \leq n$
by (*simp add*:)
fix x
assume *A1*: $x \in B$
show $x \in B'$
using *assms c-ballI*[*of x m c*] *A0' A1 c-ballE*(2)[*of x n c*] *c-ball-in-Qp*
by (*meson c-ballE*(1) *dual-order.trans eint-ord-simps*(1))
qed
show $B \subseteq B' \Longrightarrow m \leq n$
using *assms*(1) *assms*(2) *assms*(3) *assms*(4) *ball-rad-0*
by *blast*
qed

lemma *nested-balls'*:
assumes *is-ball* B
assumes *is-ball* B'
assumes $B \cap B' \neq \{\}$
shows $B \subseteq B' \vee B' \subseteq B$
proof−
obtain b **where** *b-def*: $b \in B \cap B'$
using *assms*(3) **by** *blast*
show $B \subseteq B' \vee B' \subseteq B$
proof−
have $\neg\ B \subseteq B' \Longrightarrow B' \subseteq B$
proof−
assume *A*: $\neg\ B \subseteq B'$
have *0*: $B = B_{rad\ B}[b]$
using *assms*(1) *b-def radius-of-ball* **by** *auto*
have *1*: $B' = B_{rad\ B'}[b]$
using *assms*(2) *b-def radius-of-ball* **by** *auto*
show $B' \subseteq B$ **using** *0 1 A nested-balls*
by (*smt IntD2* Q_p*-def Zp-def assms*(1) *assms*(2) *b-def c-ballE*(1) *padic-integers-axioms*)

qed
then show *?thesis* **by** *blast*
qed
qed

definition *is-bounded*:: *padic-number set* ⇒ *bool* **where**
is-bounded S = (∃ *n*. ∃ *c* ∈ *carrier* Q_p. *S* ⊆ $B_n[c]$)

lemma *empty-is-bounded*:
is-bounded {}
unfolding *is-bounded-def*
by *blast*

10.2 p-adic Open Sets

definition *is-open*:: *padic-number set* ⇒ *bool* **where**
is-open U ≡ (*U* ⊆ *carrier* Q_p) ∧ (∀ *c* ∈ *U*. ∃ *n*. $B_n[c]$⊆ *U*)

lemma *is-openI*:
assumes *U* ⊆*carrier* Q_p
assumes ⋀*c*. *c* ∈ *U* ⟹ ∃ *n*. $B_n[c]$⊆ *U*
shows *is-open U*
by (*simp add*: *assms*(*1*) *assms*(*2*) *is-open-def*)

lemma *ball-is-open*:
assumes *is-ball B*
shows *is-open B*
by (*metis* (*mono-tags*, *lifting*) *assms is-ball-imp-in-Qp is-open-def radius-of-ball subset-iff*)

lemma *is-open-imp-in-Qp*:
assumes *is-open U*
shows *U* ⊆ *carrier* Q_p
using *assms* **unfolding** *is-open-def*
by *linarith*

lemma *is-open-imp-in-Qp'*:
assumes *is-open U*
assumes *x* ∈ *U*
shows *x* ∈ *carrier* Q_p
using *assms*(*1*) *assms*(*2*) *is-open-imp-in-Qp*
by *blast*

Owing to the total disconnectedness of the p-adic field, every open set can be decomposed into a disjoint union of balls which are maximal with respect to containment in that set. This unique decomposition is occasionally useful.

definition *is-max-ball-of* ::*padic-number set* ⇒ *padic-number set* ⇒ *bool* **where**
is-max-ball-of U B ≡ (*is-ball B*) ∧ (*B* ⊆ *U*) ∧ (∀ *B'*. ((*is-ball B'*) ∧ (*B'* ⊆ *U*) ∧ *B* ⊆ *B'*) ⟶ *B'* ⊆ *B*)

lemma *is-max-ball-ofI*:
assumes $U \subseteq$ *carrier* Q_p
assumes $(B_m[c]) \subseteq U$
assumes $c \in$ *carrier* Q_p
assumes $\forall\, m'.\ m' < m \longrightarrow \neg\ B_{m'}[c] \subseteq U$
shows *is-max-ball-of* U $(B_m[c])$
proof(*rule ccontr*)
assume $\neg$ *is-max-ball-of* $U\ B_m[c]$
then have $\neg\ (\forall\, B'.$ *is-ball* $B' \wedge B' \subseteq U \wedge B_m[c] \subseteq B' \longrightarrow B' \subseteq B_m[c])$
using *assms is-max-ball-of-def*[*of* $U\ B_m[c]$]
unfolding *is-ball-def*
by *blast*
then obtain B' **where** B'-*def*: *is-ball* $B' \wedge B' \subseteq U \wedge B_m[c] \subseteq B' \wedge \neg\ B' \subseteq B_m[c]$
by *auto*
obtain n **where** n-*def*: $B' = B_n[c]$
by (*meson* B'-*def assms*(3) *c-ball-center-in is-ball-def radius-of-ball subset-iff*)
then show *False*
using *assms*
by (*smt* B'-*def* Q_p-*def Zp-def ball-rad-0 padic-integers-axioms*)
qed

lemma *int-prop*:
fixes P:: *int* $\Rightarrow$ *bool*
assumes $P\ n$
assumes $\forall\, m.\ m \leq N \longrightarrow \neg\ P\ m$
shows $\exists\, n.\ P\ n \wedge (\forall\, n'.\ P\ n' \longrightarrow n' \geq n)$
proof−
have $n > N$
by (*meson assms*(1) *assms*(2) *not-less*)
obtain k::*nat* **where** k-*def*: $k = nat\ (n - N)$
by *blast*
obtain l::*nat* **where** l-*def*: $l = (LEAST\ M.\ \ P\ (N + int\ M))$
by *simp*
have *0*: $P\ (N + int\ l)$
by (*metis* (*full-types*) *LeastI* ‹$N < n$› *assms*(1) l-*def zless-iff-Suc-zadd*)
have *1*: $l > 0$
using *0 assms*(2) *of-nat-0-less-iff* **by** *fastforce*
have *2*: $\bigwedge M.\ M < l \longrightarrow \neg\ P\ (N + M)$
by (*metis Least-le l-def less-le-trans nat-less-le*)
obtain n **where** n-*def*: $n = (N + int\ l)$
by *simp*
have $P\ n \wedge (\forall\, n'.\ P\ n' \longrightarrow n' \geq n)$
proof−
have $P\ n$
by (*simp add*: *0* n-*def*)
have $(\forall\, n'.\ P\ n' \longrightarrow n' \geq n)$
proof

```
      fix n'
      show P n' ⟶ n ≤ n'
      proof
        assume P n'
        show  n ≤n'
        proof(rule ccontr)
          assume  ¬ n ≤ n'
          then have C: n' < n
            by auto
          show False
          proof(cases n' ≥ N)
            case True
            obtain M where M-def: nat (n' − N) = M
              by simp
            then have M0: n' = N + int M
              using True by linarith
            have M1: M < l
              using n-def C M0
              by linarith
            then show ?thesis using 2 M-def M0 M1
              using ‹P n'› by auto
          next
            case False
            then show ?thesis using assms
              using ‹P n'› by auto
          qed
        qed
      qed
      then show ?thesis
        using ‹P n› by blast
    qed
    then show ?thesis by blast
  qed

  lemma open-max-ball:
    assumes  is-open U
    assumes  U ≠ carrier Q_p
    assumes c ∈ U
    shows ∃ B. is-max-ball-of U B ∧ c ∈ B
  proof−
    obtain B where B-def: is-ball B ∧ B ⊆ U ∧ c ∈ B
      by (meson assms(1) assms(3) c-ball-center-in is-ball-def is-open-imp-in-Qp'
  is-open-def padic-integers-axioms)
    show P: ∃ B. is-max-ball-of U B ∧ c ∈ B
    proof(rule ccontr)
      assume C: ∄ B. is-max-ball-of U B ∧ c ∈ B
      show False
      proof−
```

```
    have C': ∀ B. c ∈ B ⟶ ¬  is-max-ball-of U B
      using C
      by auto
    have C'': ∀ N. ∃ m <N. B_m[c] ⊆ U
    proof
      fix N
      show ∃ m<N. B_m[c] ⊆ U
      proof(rule  ccontr)
        assume A: ¬ (∃ m<N. B_m[c] ⊆ U)
        obtain P where P-def: P = (λ n. ∃ m<n. B_m[c] ⊆ U)
          by simp
        have A0: ∃ n. P n
          by (metis B-def P-def gt-ex radius-of-ball)
        have A1: ∀ m. m ≤N ⟶ ¬ P m
          using A P-def by auto
        have A2: ∃ n. P n ∧ (∀ n'. P n'⟶ n' ≥n)
          using A0 A1 int-prop
          by auto
        obtain n where n-def:  P n ∧ (∀ n'. P n'⟶ n' ≥n)
          using A2 by blast
        have  B_n[c] ⊆ U
        by (smt B-def P-def c-ball-def is-ball-def mem-Collect-eq n-def nested-balls
order-trans)
        obtain m where m-def: m < n ∧B_m[c] ⊆ U
          using P-def n-def by blast
        have m = n−1
        proof−
          have P (m +1)
            using P-def m-def
            by auto
          then have m + 1 ≥ n
            using n-def by blast
          then show ?thesis using m-def by auto
        qed
        have ∀ m'. m' < m ⟶ ¬ B_m'[c] ⊆ U
        proof
          fix m'
          show  m' < m ⟶ ¬ B_m'[c] ⊆ U
          proof
            assume m' < m
            show ¬ B_m'[c] ⊆ U
            proof
              assume B_m'[c] ⊆ U
              then have P (m' + 1)
                using P-def by auto
              have m'+ 1 < n
                using ‹m = n − 1› ‹m' < m› by linarith
              then show False
                using n-def ‹P (m' + 1)› by auto
```

```
          qed
        qed
      qed
      then have is-max-ball-of U B_m[c]
     using is-max-ball-ofI assms(1) assms(3) is-open-imp-in-Qp is-open-imp-in-Qp'
m-def
        by presburger
      then show False
        using C assms(1) assms(3) c-ball-center-in is-open-imp-in-Qp'
          is-max-ball-of-def padic-integers-axioms
        by blast
    qed
  qed
  have U = carrier Q_p
  proof
    show carrier Q_p ⊆ U
    proof
      fix x
      assume x ∈ carrier Q_p
      show x ∈ U
      proof(cases x = c)
        case True
        then show ?thesis using assms by auto
      next
        case False
        obtain m where m-def: eint m = val(x ⊖ c)
          using False
           by (metis (no-types, opaque-lifting) Qp-diff-diff Qp.domain-axioms ‹x
∈ carrier Q_p› a-minus-def
          assms(1) assms(3) cring.cring-simprules(16) cring.cring-simprules(17)
                cring.cring-simprules(4) domain.axioms(1) is-open-imp-in-Qp'
val-def val-minus)
        obtain m' where m'-def: m' < m ∧  B_m'[c] ⊆ U
          using C''
          by blast
        have val (x ⊖ c) ≥ m'
          by (metis eint-ord-simps(1) less-imp-le m'-def m-def)
        then have x ∈ B_m'[c]
          using ‹x ∈ carrier Q_p› c-ballI by blast
        then show x ∈ U
          using m'-def by blast
      qed
    qed
    show U ⊆ carrier Q_p
      using assms
      by (simp add: is-open-imp-in-Qp)
  qed
  then show False using assms by auto
 qed
```

```
  qed
qed

definition interior where
  interior U = {a. ∃ B. is-open B ∧ B ⊆ U ∧ a ∈ B}

lemma interior-subset:
  assumes U ⊆ carrier Q_p
  shows interior U ⊆ U
proof
  fix x
  assume x ∈ interior U
  show x ∈ U
  proof−
    obtain B where B-def: is-open B ∧ B ⊆ U ∧ x ∈ B
      using ‹x ∈ interior U› interior-def
      by auto
    then show x ∈ U
      by blast
  qed
qed

lemma interior-open:
  assumes U ⊆ carrier Q_p
  shows is-open (interior U)
proof(rule is-openI)
  show interior U ⊆ carrier Q_p
    using assms interior-subset by blast
  show ⋀c. c ∈ interior U ⟹ ∃ n. B_n[c] ⊆ interior U
  proof−
    fix c
    assume c ∈ interior U
    show ∃ n. B_n[c] ⊆ interior U
    proof−
    obtain B where B-def: is-open B ∧ B ⊆ U ∧ c ∈ B
      using ‹c ∈ interior U› interior-def padic-integers-axioms
      by auto
    then show ?thesis
    proof −
     obtain ii :: ((nat ⇒ int) × (nat ⇒ int)) set ⇒ ((nat ⇒ int) × (nat ⇒ int))
set set ⇒ int
       where
       B_ii c B[c] ⊆ B
       by (meson B-def is-open-def)
     then show ?thesis
       using B-def interior-def padic-integers-axioms by auto
qed
  qed
  qed
```

qed

lemma *interiorI*:
assumes $W \subseteq U$
assumes *is-open W*
shows $W \subseteq$ *interior U*
using *assms*(*1*) *assms*(*2*) *interior-def* **by** *blast*

lemma *max-ball-interior*:
assumes $U \subseteq$ *carrier* Q_p
assumes *is-max-ball-of* (*interior U*) *B*
shows *is-max-ball-of U B*
proof(*rule ccontr*)
assume *C*: $\neg$ *is-max-ball-of U B*
then obtain B' **where** B'*-def*: *is-ball* $B' \wedge B' \subseteq U \wedge B \subseteq B' \wedge B \neq B'$
by (*metis* (*no-types*, *lifting*) *assms*(*1*) *assms*(*2*) *dual-order.trans*
interior-subset is-max-ball-of-def)
then have $B' \subseteq$ *interior U*
using *interior-def padic-integers-axioms ball-is-open*
by *auto*
then show *False* **using** *assms* B'*-def is-max-ball-of-def*[*of interior U B*]
by *blast*
qed

lemma *ball-in-max-ball*:
assumes $U \subseteq$ *carrier* Q_p
assumes $U \neq$ *carrier* Q_p
assumes $c \in U$
assumes $\exists B.\ B \subseteq U \wedge$ *is-ball* $B \wedge c \in B$
shows $\exists B'.$ *is-max-ball-of U* $B' \wedge c \in B'$
proof−
obtain *B* **where** $B \subseteq U \wedge$ *is-ball* $B \wedge c \in B$
using *assms*(*4*)
by *blast*
then have *0*: $B \subseteq$ *interior U*
using *ball-is-open interior-def* **by** *blast*
have *1*: $c \in$ *interior U*
using *0* ‹$B \subseteq U \wedge$ *is-ball* $B \wedge c \in B$› **by** *blast*
then have $\exists B'.$ *is-max-ball-of* (*interior U*) $B' \wedge c \in B'$
using *open-max-ball*[*of interior U c*] *assms*(*1*) *assms*(*2*) *interior-open interior-subset*
by *blast*
then show *?thesis*
using *assms*(*1*) *max-ball-interior*
by *blast*
qed

lemma *ball-in-max-ball'*:
assumes $U \subseteq$ *carrier* Q_p

assumes $U \neq$ *carrier* Q_p
assumes $B \subseteq U \wedge$ *is-ball* B
shows $\exists B'$. *is-max-ball-of* $U\ B' \wedge B \subseteq B'$
proof−
obtain c **where** *c-def*: $c \in B$
by (*metis assms*(3) *c-ball-center-in is-ball-def*)
obtain B' **where** B'*-def*: *is-max-ball-of* $U\ B' \wedge c \in B'$
using *assms ball-in-max-ball*[*of U c*] *c-def*
by *blast*
then show *?thesis*
by (*meson assms*(3) *c-def disjoint-iff-not-equal nested-balls'*
is-max-ball-of-def padic-integers-axioms)
qed

lemma *max-balls-disjoint*:
assumes $U \subseteq$ *carrier* Q_p
assumes *is-max-ball-of* $U\ B$
assumes *is-max-ball-of* $U\ B'$
assumes $B \neq B'$
shows $B \cap B' = \{\}$
by (*meson assms*(2) *assms*(3) *assms*(4) *nested-balls' is-max-ball-of-def*
padic-integers-axioms subset-antisym)

definition *max-balls* :: *padic-number set* ⇒ *padic-number set set* **where**
max-balls $U = \{B.$ *is-max-ball-of* $U\ B\ \}$

lemma *max-balls-interior*:
assumes $U \subseteq$ *carrier* Q_p
assumes $U \neq$ *carrier* Q_p
shows *interior* $U = \{x \in$ *carrier* $Q_p.\ (\exists B \in (max\text{-}balls\ U).\ x \in B)\}$
proof
show *interior* $U \subseteq \{x \in$ *carrier* $Q_p.\ \exists B \in max\text{-}balls\ U.\ x \in B\}$
proof
fix x
assume A: $x \in$ *interior* U
show $x \in \{x \in$ *carrier* $Q_p.\ \exists B \in max\text{-}balls\ U.\ x \in B\}$
by (*metis* (*mono-tags*, *lifting*) A *assms*(1) *assms*(2) *interior-open*
interior-subset is-open-imp-in-Qp' max-ball-interior max-balls-def
mem-Collect-eq open-max-ball subset-antisym)
qed
show $\{x \in$ *carrier* $Q_p.\ \exists B \in max\text{-}balls\ U.\ x \in B\} \subseteq$ *interior* U
proof
fix x
assume A: $x \in \{x \in$ *carrier* $Q_p.\ \exists B \in max\text{-}balls\ U.\ x \in B\}$
show $x \in$ *interior* U
proof−
obtain B **where** B*-def*: $B \in max\text{-}balls\ U \wedge x \in B$
using A **by** *blast*
then have $B \subseteq$ *interior* U

by (*metis* (*no-types*, *lifting*) *interior-def is-max-ball-of-def mem-Collect-eq ball-is-open max-balls-def subsetI*)
then show *?thesis*
using *B-def* **by** *blast*
qed
qed
qed

lemma *max-balls-interior'*:
assumes $U \subseteq$ *carrier* Q_p
assumes $U \neq$ *carrier* Q_p
assumes $B \in$ *max-balls* U
shows $B \subseteq$ *interior* U
using *assms*(*1*) *assms*(*2*) *assms*(*3*) *is-max-ball-of-def max-balls-interior max-balls-def padic-integers-axioms*
by *auto*

lemma *max-balls-interior''*:
assumes $U \subseteq$ *carrier* Q_p
assumes $U \neq$ *carrier* Q_p
assumes $a \in$ *interior* U
shows $\exists B \in$ *max-balls* U. $a \in B$
using *assms*(*1*) *assms*(*2*) *assms*(*3*) *max-balls-interior*
by *blast*

lemma *open-interior*:
assumes *is-open* U
shows *interior* $U = U$
unfolding *interior-def* **using** *assms*
by *blast*

lemma *interior-idempotent*:
assumes $U \subseteq$ *carrier* Q_p
shows *interior* (*interior* U) = *interior* U
using *assms interior-open*[*of* U] *open-interior*[*of interior* U]
by *auto*

10.3 Convex Subsets of the Value Group

The content of this section will be useful for defining and reasoning about p-adic cells in the proof of Macintyre's theorem. It is proved that every convex set in the extended integers is either an open ray, a closed ray, a closed interval, or a left-closed interval.

definition *is-convex* :: *eint set* ⇒ *bool* **where**
is-convex $A = (\forall\ x \in A.\ \forall\, y \in A.\ \forall\, c.\ x \leq c \wedge c \leq y \longrightarrow c \in A)$

lemma *is-convexI*:
assumes $\bigwedge x\ y\ c.\ x \in A \Longrightarrow y \in A \Longrightarrow x \leq c \wedge c \leq y \Longrightarrow c \in A$

shows *is-convex A*
unfolding *is-convex-def*
using *assms*
by *blast*

lemma *is-convexE*:
assumes *is-convex A*
assumes $x \in A$
assumes $y \in A$
assumes $x \leq a$
assumes $a \leq y$
shows $a \in A$
using *assms is-convex-def*
by *blast*

lemma *empty-convex*:
is-convex {}
apply(*rule is-convexI*)
by *blast*

lemma *UNIV-convex*:
is-convex UNIV
apply(*rule is-convexI*)
by *blast*

definition *closed-interval* (‹*I*[- -]›) **where**
closed-interval α β = $\{a$. $\alpha \leq a \wedge \ a \leq \beta\}$

lemma *closed-interval-is-convex*:
assumes A = *closed-interval* α β
shows *is-convex A*
apply(*rule is-convexI*)
using *assms* **unfolding** *closed-interval-def*
by *auto*

lemma *empty-closed-interval*:
{} = *closed-interval* ∞ (*eint 1*)
unfolding *closed-interval-def*
by *auto*

definition *left-closed-interval* **where**
left-closed-interval α β = $\{a$. $\alpha \leq \ a \wedge \ a < \beta\}$

lemma *left-closed-interval-is-convex*:
assumes A = *left-closed-interval* α β
shows *is-convex A*
apply(*rule is-convexI*)
using *assms* **unfolding** *left-closed-interval-def*
using *leD order.trans* **by** *auto*

definition *closed-ray* **where**
closed-ray α β $= \{a . \quad a \leq \beta\}$

lemma *closed-ray-is-convex*:
assumes $A =$ *closed-ray* α β
shows *is-convex* A
apply(*rule is-convexI*)
using *assms* **unfolding** *closed-ray-def*
by *auto*

lemma *UNIV-closed-ray*:
(*UNIV*::*eint set*)= *closed-ray* α ∞
unfolding *closed-ray-def*
by *simp*

definition *open-ray* :: *eint* $\Rightarrow$ *eint* $\Rightarrow$ *eint set* **where**
open-ray α β $= \{a . \quad a < \beta\}$

lemma *open-ray-is-convex*:
assumes $A =$ *open-ray* α β
shows *is-convex* A
apply(*rule is-convexI*)
using *assms* **unfolding** *open-ray-def*
using *leD* **by** *auto*

lemma *open-rayE*:
assumes $a < \beta$
shows $a \in$ *open-ray* α β
unfolding *open-ray-def* **using** *assms*
by *blast*

lemma *value-group-is-open-ray*:
$UNIV - \{\infty\} =$ *open-ray* α ∞
proof
show $UNIV - \{\infty\} \subseteq$ *open-ray* α ∞
using *open-rayE*[*of* - α ∞]
by (*simp add*: *open-rayE subset-eq*)
show *open-ray* α $\infty \subseteq UNIV - \{\infty\}$
unfolding *open-ray-def*
by *blast*
qed

This is a predicate which identifies a certain kind of set-valued function on the extended integers. Convex conditions will be important in the definition of p-adic cells later, and it will be proved that every convex set is induced by a convex condition.

definition *is-convex-condition* :: (*eint* $\Rightarrow$ *eint* $\Rightarrow$ *eint set*) $\Rightarrow$ *bool*
where *is-convex-condition* $I \equiv$

$I = \textit{closed-interval} \vee I = \textit{left-closed-interval} \vee I = \textit{closed-ray} \vee I = \textit{open-ray}$

lemma *convex-condition-imp-convex*:
assumes *is-convex-condition I*
shows *is-convex* $(I\ \alpha\ \beta)$
using *assms*
unfolding *is-convex-condition-def*
by (*meson closed-interval-is-convex closed-ray-is-convex left-closed-interval-is-convex open-ray-is-convex*)

lemma *bounded-order*:
assumes $(a::eint) < \infty$
assumes $b \leq a$
obtains $k::nat$ **where** $a = b + k$
proof−
obtain $m::int$ **where** *m-def*: $a = m$
using *assms*(*1*) *less-infinityE* **by** *blast*
obtain $n::int$ **where** *n-def*: $b = n$
using *assms*(*2*) *eint-ile m-def* **by** *blast*
have *0*: $a - b = \ eint\ (m - n)$
by (*simp add*: *m-def n-def*)
then have $a = b + nat\ (m - n)$
using *assms m-def n-def*
by *simp*
thus *?thesis*
using *that* **by** *blast*
qed

Every convex set is given by a convex condition

lemma *convex-imp-convex-condition*:
assumes *is-convex A*
shows $\exists\ I\ \alpha\ \beta.$ *is-convex-condition* $I \wedge A = (I\ \alpha\ \beta)$
proof(*cases* $\exists\ a \in A.\ \forall\ b \in A.\ a \leq b$)
case *True*
then obtain α **where** *alpha-def*: $\alpha \in A \wedge (\forall\ b \in A.\ \alpha \leq b)$
by *blast*
then show *?thesis*
proof(*cases* $\exists\ a \in A.\ \forall\ b \in A.\ b \leq a$)
case *True*
then obtain β **where** *beta-def*: $\beta \in A \wedge (\forall\ b \in A.\ b \leq \beta)$
by *blast*
have $A =$ *closed-interval* $\alpha\ \beta$
unfolding *closed-interval-def*
using *alpha-def beta-def assms is-convexE*[*of A* $\alpha\ \beta$]
by *blast*
then show *?thesis*
using *is-convex-condition-def*
by *blast*

```
next
  case False
  have F0: ∀ a. α ≤ a ∧ a < ∞ ⟶ a ∈ A
  proof(rule ccontr)
    assume A: ¬ (∀ a. a ≥ α ∧ a < ∞ ⟶ a ∈ A)
    then obtain a where a-def:  α ≤ a ∧ a < ∞ ∧ a ∉ A
      by blast
    obtain n where n-def: α = eint n
      using False alpha-def by force
    obtain m where m-def: a = eint m
      using a-def less-infinityE by blast
    have ∀ k::nat. ∃ c. (α + eint (int k)) < c ∧ c ∈ A
    proof fix k
      show ∃ c>α + eint (int k). c ∈ A
        apply(induction k)
         using False alpha-def le-less-linear zero-eint-def apply fastforce
      proof− fix k
        assume IH: ∃ c>α + eint (int k). c ∈ A
        then obtain c where c-def: c>α + eint (int k) ∧ c ∈ A
          by blast
        then obtain c' where c'-def: c' ∈ A ∧ c < c'
          using False
          by (meson le-less-linear)
        then have (α + eint (int (Suc k))) ≤ c'
        proof−
          have 0: eint (int (Suc k)) = eint (int k) + eint 1
            by simp
          have α + eint (int k) ≤c'
            by (meson c'-def c-def le-less le-less-trans)
          then have 1: (α + eint (int k)) < c'
            using c'-def c-def le-less-trans
            by auto
          have 2: α + eint (int k) + eint 1 = α + eint (int (Suc k))
            using 0
            by (simp add: n-def)
          then show (α + eint (int (Suc k))) ≤ c'
            by (metis 1 ileI1 one-eint-def)
        qed
        then show ∃ c>α + eint (int (Suc k)). c ∈ A
          using False c'-def not-less by fastforce
      qed
    qed
    obtain k where k-def: k = a − α
      using a-def n-def
      by blast
    hence k ≥ 0
      using a-def n-def
      by (metis alpha-def eint-minus-le le-less)
    hence 0: ∃ n::nat. k = n
```

```
      using a-def n-def  k-def
     by (metis eint.distinct(2) eint-0-iff(2) eint-add-cancel-fact eint-ord-simps(1)
fromeint.cases m-def nonneg-int-cases plus-eint-simps(3))
    have 1: a = α + k
      using k-def a-def n-def
      by simp
    then obtain c where c ∈ A ∧ a < c
      by (metis 0 ‹∀ k. ∃ c>α + eint (int k). c ∈ A› the-eint.simps)
    then have a ∈ A
      using is-convexE[of A α a c] a-def alpha-def assms is-convexE
      by (meson linear not-less)
    then show False
      using a-def by blast
  qed
  have A = closed-interval α  ∞ ∨ A = left-closed-interval α ∞
    apply(cases ∞ ∈ A)
    using False eint-ord-code(3) apply blast
  proof−
    assume A0: ∞ ∉ A
    have A = left-closed-interval α ∞
    proof
      show A ⊆ left-closed-interval α ∞
        unfolding left-closed-interval-def
        using alpha-def A0
     by (metis (mono-tags, lifting) False eint-ord-code(3) le-less-linear less-le-trans
mem-Collect-eq subsetI)
      show left-closed-interval α ∞ ⊆ A
        unfolding left-closed-interval-def
        using alpha-def A0 F0
        by blast
    qed
    then show ?thesis
      by blast
  qed
  then show ?thesis
    unfolding is-convex-condition-def
    by blast
 qed
next
 case False
 show ?thesis apply(cases A = {})
  using empty-closed-interval is-convex-condition-def apply blast
 proof−
  assume A0: A ≠ {}
  have A ≠ {∞}
    using False
    by blast
  then obtain α where alpha-def: α ∈ A ∧ α ≠∞
    using A0
```

```
  by fastforce
have A1: ⋀k::nat. ∃ b ∈ A. b + eint (int k) ≤ α
proof− fix k
  show ∃ b ∈ A. b + eint (int k) ≤ α
  proof(induction k)
    case 0
    then have α + eint (int 0) = α
      by (simp add: zero-eint-def)
    then show ?case
      using alpha-def by auto
  next
    case (Suc k) fix k
    assume IH: ∃ b∈A. α ≥ b + eint (int k)
    show ∃ b∈A. α ≥ b + eint (int (Suc k))
    proof−
      obtain b where b-def: b ∈ A ∧ α ≥ b + eint (int k)
        using IH by blast
      then obtain c where c-def: c ∈ A ∧ c < b
        using False le-less-linear by blast
      have 0: (c + eint (int (Suc k))) < (b + eint (int (Suc k)))
        using c-def
        by simp
      have 1: b + eint (int (Suc k)) = (b + eint (int k)) + eint 1
        by simp
      then show ?thesis
       by (metis 0 b-def c-def eSuc-ile-mono ileI1 le-less one-eint-def)
    qed
  qed
qed
show ?thesis
proof(cases ∃ a ∈ A. ∀ b ∈ A. b ≤ a)
  case True
  then obtain β where beta-def: β ∈ A ∧ (∀ b ∈ A. b ≤ β)
    by blast
  have A = closed-ray α β
    unfolding closed-ray-def
  proof
    show A ⊆ {a. β ≥ a}
      using assms beta-def
      by blast
    show {a. β ≥ a} ⊆ A
    proof fix x assume x ∈ {a. β ≥ a}
      then have 0: β ≥ x by blast
      show x ∈ A
      proof(cases x ≤ α)
        case True
        obtain n where n-def: α= eint n
          using alpha-def
          by blast
```

```
      obtain m where m-def: x = eint m
        using True eint-ile n-def by blast
      have 1: m ≤ n
        using  True m-def n-def
        by simp
      have 2: eint (int (nat (n − m))) = eint (n − m)
        by (simp add: 1)
      then obtain b where b-def: b ∈ A ∧ b + eint (n − m) ≤ α
        using  A1[of nat (n − m)]
        by (metis)
      then have b + eint (n − m) ≤ x + eint (n − m)
        using b-def
        by (simp add: m-def n-def)
      then have b ≤ x
        by auto
      then show ?thesis
        using 0 assms b-def beta-def is-convex-def
        by blast
    next
      case False
      then show ?thesis
        using 0 alpha-def assms beta-def is-convexE
        by (meson linear)
    qed
  qed
  then show ?thesis
    using is-convex-condition-def
    by blast
next
  case f: False
  have F0: ∀ a. α ≤ a ∧ a < ∞ ⟶ a ∈ A
  proof(rule ccontr)
  assume A: ¬ (∀ a. a ≥ α ∧ a < ∞ ⟶ a ∈ A)
  then obtain a where a-def:  α ≤ a ∧ a < ∞ ∧ a ∉ A
    by blast
  obtain n where n-def: α = eint n
    using alpha-def by blast
  obtain m where m-def: a = eint m
    using a-def less-infinityE by blast
  have 0: ∀ k::nat. ∃ c. (α + eint (int k)) < c ∧ c ∈ A
  proof fix k
    show ∃ c>α + eint (int k). c ∈ A
      apply(induction k)
       using alpha-def f le-less-linear apply fastforce
    proof− fix k
      assume IH: ∃ c>α + eint (int k). c ∈ A
      then obtain c where c-def: c>α + eint (int k) ∧ c ∈ A
        by blast
```

```
      then obtain c' where c'-def: c' ∈ A ∧ c < c'
        using False f le-less-linear by blast
      then have (α + eint (int (Suc k))) ≤ c'
      proof−
        have 0: eint (int (Suc k)) = eint (int k) + eint 1
          by simp
        have α + eint (int k) ≤c'
          using c-def c'-def dual-order.strict-trans le-less by blast
        then have 1: (α + eint (int k)) < c'
          using c'-def c-def le-less-trans by auto
        have 2: α + eint (int k) + eint 1 = α + eint (int (Suc k))
          using 0 by (simp add: n-def)
        then show (α + eint (int (Suc k))) ≤ c'
          by (metis 1 ileI1 one-eint-def)
      qed
      then show ∃ c>α + eint (int (Suc k)). c ∈ A
        using False c'-def
        by (smt c-def eSuc-eint iadd-Suc-right ileI1 le-less of-nat-Suc)
    qed
  qed
  obtain k::nat where a = α + eint (int k)
    using bounded-order a-def
    by blast
  then obtain c where c ∈ A ∧ a <c
    using 0 by blast
  then have a ∈ A
    using is-convexE[of A α a c] a-def alpha-def assms is-convexE
    by (meson linear not-less)
  then show False
    using a-def by blast
  qed
  have A = UNIV − {∞}
  proof
    show A ⊆ UNIV − {∞}
      using f by auto
    show UNIV − {∞} ⊆ A
    proof fix x ::eint
      assume A: x ∈ UNIV − {∞}
      show x ∈ A
      proof(cases x ≤ α)
        case True
        obtain k::nat where k-def: x + k = α
          by (metis True alpha-def bounded-order eint-ord-simps(4))
        obtain c where c-def: c ∈ A ∧ c + k = α
          by (metis A1 True add.commute alpha-def assms eint-add-left-cancel-le
is-convexE k-def not-eint-eq)
        have x = c
          using k-def c-def
          by auto
```

thus *?thesis*
by (*simp add*: *c-def*)
next
case *False*
thus *?thesis*
using *A F0* **by** *auto*
qed
qed
qed
then show *?thesis*
by (*meson is-convex-condition-def value-group-is-open-ray*)
qed
qed
qed

lemma *ex-val-less*:
shows $\exists$ (α::*eint*). $\alpha < \beta$
apply(*induction* β)
using *eint-ord-simps*(*2*) *lt-ex* **apply** *blast*
using *eint-ord-simps*(*4*) **by** *blast*

lemma *ex-dist-less*:
assumes $c \in$ *carrier* Q_p
shows $\exists$ $a \in$ *carrier* Q_p. *val* ($a \ominus c$) $< \beta$
using *ex-val-less*[*of* β] *assms*
by (*metis dist-nonempty*$'$ *ex-val-less*)
end
end
theory *Generated-Boolean-Algebra*
imports *Main*
begin

11 Generated Boolean Algebras of Sets

11.1 Definitions and Basic Lemmas

lemma *equalityI*$'$:
assumes $\bigwedge x.\ x \in A \Longrightarrow x \in B$
assumes $\bigwedge x.\ x \in B \Longrightarrow x \in A$
shows $A = B$
using *assms* **by** *blast*

lemma *equalityI*$''$:
assumes $\bigwedge x.\ A\ x \Longrightarrow B\ x$
assumes $\bigwedge x.\ B\ x \Longrightarrow A\ x$
shows $\{x.\ A\ x\} = \{x.\ B\ x\}$
using *assms* **by** *blast*

lemma *SomeE*:

assumes $a = (SOME\ x.\ P\ x)$
assumes $P\ c$
shows $P\ a$
using *assms* **by** (*meson verit-sko-ex*)

lemma *SomeE'*:
assumes $a = (SOME\ x.\ P\ x)$
assumes $\exists\ x.\ P\ x$
shows $P\ a$
using *assms* **by** (*meson verit-sko-ex*)

12 Basic notions about boolean algebras over a set S, generated by a set of generators B

Note that the generators B need not be subsets of the set S

inductive-set *gen-boolean-algebra*
for S **and** B **where**
universe: $S \in gen\text{-}boolean\text{-}algebra\ S\ B$
| *generator*: $A \in B \Longrightarrow A \cap S \in gen\text{-}boolean\text{-}algebra\ S\ B$
| *union*: $[\![A \in gen\text{-}boolean\text{-}algebra\ S\ B;\ C \in gen\text{-}boolean\text{-}algebra\ S\ B]\!] \Longrightarrow A \cup C \in gen\text{-}boolean\text{-}algebra\ S\ B$
| *complement*: $A \in gen\text{-}boolean\text{-}algebra\ S\ B \Longrightarrow S - A \in gen\text{-}boolean\text{-}algebra\ S\ B$

lemma *gen-boolean-algebra-subset*:
shows $A \in gen\text{-}boolean\text{-}algebra\ S\ B \Longrightarrow A \subseteq S$
apply(*induction A rule*: *gen-boolean-algebra.induct*)
apply *blast*
apply *blast*
apply *blast*
by *blast*

lemma *gen-boolean-algebra-intersect*:
assumes $A \in gen\text{-}boolean\text{-}algebra\ S\ B$
assumes $C \in gen\text{-}boolean\text{-}algebra\ S\ B$
shows $A \cap C \in gen\text{-}boolean\text{-}algebra\ S\ B$
proof−
have *0*: $S - A \in gen\text{-}boolean\text{-}algebra\ S\ B$
using *assms*(*1*) *gen-boolean-algebra.complement* **by** *blast*
have *1*: $S - C \in gen\text{-}boolean\text{-}algebra\ S\ B$
using *assms*(*2*) *gen-boolean-algebra.complement* **by** *blast*
have *2*: $(S - A) \cup (S - C) \in gen\text{-}boolean\text{-}algebra\ S\ B$
using *0 1 gen-boolean-algebra.union* **by** *blast*
have $S - (A \cap C) \in gen\text{-}boolean\text{-}algebra\ S\ B$
by (*simp add*: *2 Diff-Int*)
then have *3*: $S - (S - (A \cap C)) \in gen\text{-}boolean\text{-}algebra\ S\ B$
using *gen-boolean-algebra.complement*
by *blast*

have $A \cap C \subseteq S$
using *assms*(*1*) *gen-boolean-algebra-subset*
by *blast*
then show *?thesis*
using *3*
by (*metis 0 Diff-partition Un-subset-iff assms*(*1*) *double-diff gen-boolean-algebra-subset*)
qed

lemma *gen-boolean-algebra-diff*:
assumes $A \in$ *gen-boolean-algebra* S B
assumes $C \in$ *gen-boolean-algebra* S B
shows $A - C \in$ *gen-boolean-algebra* S B
proof−
have $A - C = A \cap (S - C)$
by (*metis Int-Diff assms*(*1*) *gen-boolean-algebra-subset inf-absorb1*)
then show *?thesis*
by (*metis assms*(*1*) *assms*(*2*) *gen-boolean-algebra.complement gen-boolean-algebra-intersect*)
qed

lemma *gen-boolean-algebra-diff-eq*:
assumes $A \in$ *gen-boolean-algebra* S B
assumes $C \in$ *gen-boolean-algebra* S B
shows $A - C = A \cap (S - C)$
by (*metis Int-Diff assms*(*1*) *gen-boolean-algebra-subset inf-absorb1*)

lemma *gen-boolean-algebra-finite-union*:
assumes $\bigwedge a.\ a \in A \Longrightarrow a \in$ *gen-boolean-algebra* S B
assumes *finite* A
shows $\bigcup A \in$ *gen-boolean-algebra* S B
proof−
have $(\forall\, a \in A.\ a \in$ *gen-boolean-algebra* S $B) \longrightarrow \bigcup A \in$ *gen-boolean-algebra* S B
apply(*rule finite.induct*[*of* A])
apply (*simp add*: *assms*(*2*); *fail*)
apply (*metis DiffE Union-empty ex-in-conv gen-boolean-algebra.simps*)
by (*metis Union-insert gen-boolean-algebra.simps insert-iff*)
then show *?thesis* **using** *assms* **by** *blast*
qed

lemma *gen-boolean-algebra-finite-intersection*:
assumes $\bigwedge a.\ a \in A \Longrightarrow a \in$ *gen-boolean-algebra* S B
assumes *finite* A
assumes $A \neq \{\}$
shows $\bigcap A \in$ *gen-boolean-algebra* S B
proof−
have $(\forall\, a \in A.\ a \in$ *gen-boolean-algebra* S $B) \wedge A \neq \{\} \longrightarrow \bigcap A \in$ *gen-boolean-algebra* S B
apply(*rule finite.induct*[*of* A])
apply (*simp add*: *assms*(*2*))
apply *force*

using *gen-boolean-algebra-intersect* **by** *auto*
then show *?thesis* **using** *assms* **by** *blast*
qed

lemma *gen-boolean-algebra-generators*:
assumes $\bigwedge b.\ b \in B \Longrightarrow b \subseteq S$
assumes $b \in B$
shows $b \in$ *gen-boolean-algebra S B*
unfolding *gen-boolean-algebra.simps*[*of b*] **using** *assms*(*1*)[*of b*] *assms*(*2*) **by** *blast*

lemma *gen-boolean-algebra-generator-subset*:
assumes $A \in$ *gen-boolean-algebra S As*
assumes $As \subseteq Bs$
shows $A \in$ *gen-boolean-algebra S Bs*
apply(*rule gen-boolean-algebra.induct*[*of A S As*])
using *assms*(*1*) **apply** *blast*
apply (*simp add*: *gen-boolean-algebra.intros*(*1*); *fail*)
apply (*meson Set.basic-monos*(*7*) *assms*(*2*) *gen-boolean-algebra.intros*(*2*))
using *gen-boolean-algebra.intros*(*3*) **apply** *blast*
using *gen-boolean-algebra.intros*(*4*) **by** *blast*

lemma *gen-boolean-algebra-generators-union*:
assumes $A \in$ *gen-boolean-algebra S As*
assumes $C \in$ *gen-boolean-algebra S Cs*
shows $A \cup C \in$ *gen-boolean-algebra S* $(As \cup Cs)$
apply(*rule gen-boolean-algebra.induct*[*of C S Cs*])
using *assms* **apply** *blast*
apply(*rule gen-boolean-algebra.union*)
apply(*rule gen-boolean-algebra-generator-subset*[*of - - As*], *rule assms*, *blast*)
apply(*rule gen-boolean-algebra.universe*)
apply(*rule gen-boolean-algebra.union*)
apply(*rule gen-boolean-algebra-generator-subset*[*of - - As*], *rule assms*, *blast*)
apply(*rule gen-boolean-algebra.generator*, *blast*)
apply(*rule gen-boolean-algebra.union*)
apply(*rule gen-boolean-algebra-generator-subset*[*of - - As*], *rule assms*, *blast*)
apply(*rule gen-boolean-algebra.union*)
apply(*rule gen-boolean-algebra-generator-subset*[*of - - Cs*], *blast*, *blast*)
apply(*rule gen-boolean-algebra-generator-subset*[*of - - Cs*], *blast*, *blast*)
apply(*rule gen-boolean-algebra.union*)
apply(*rule gen-boolean-algebra-generator-subset*[*of - - As*], *rule assms*, *blast*)
apply(*rule gen-boolean-algebra-generator-subset*[*of - - Cs*])
apply(*rule gen-boolean-algebra-diff*)
apply(*rule gen-boolean-algebra.universe*)
apply *blast*
by *blast*

lemma *gen-boolean-algebra-finite-gen-wits*:
assumes $A \in$ *gen-boolean-algebra S B*

shows $\exists$ *Bs. finite Bs* $\wedge$ *Bs* $\subseteq$ *B* $\wedge$ *A* $\in$ *gen-boolean-algebra S Bs*
proof(*rule gen-boolean-algebra.induct*[*of A S B*])
show *A* $\in$ *gen-boolean-algebra S B*
using *assms* **by** *blast*
show $\exists$ *Bs. finite Bs* $\wedge$ *Bs* $\subseteq$ *B* $\wedge$ *S* $\in$ *gen-boolean-algebra S Bs*
using *gen-boolean-algebra.universe*[*of S* {}]
by *blast*
show $\bigwedge$*A. A* $\in$ *B* $\Longrightarrow$ $\exists$ *Bs. finite Bs* $\wedge$ *Bs* $\subseteq$ *B* $\wedge$ *A* $\cap$ *S* $\in$ *gen-boolean-algebra S Bs*
proof− **fix** *A* **assume** *A*: *A* $\in$ *B*
have *0*: {*A*} $\subseteq$ *B*
using *A* **by** *blast*
show $\exists$ *Bs. finite Bs* $\wedge$ *Bs* $\subseteq$ *B* $\wedge$ *A* $\cap$ *S* $\in$ *gen-boolean-algebra S Bs*
using *gen-boolean-algebra.generator*[*of A* {*A*} *S*] *0*
by (*meson finite.emptyI finite.insertI singletonI*)
qed
show $\bigwedge$*A C. A* $\in$ *gen-boolean-algebra S B* $\Longrightarrow$
$\exists$ *Bs. finite Bs* $\wedge$ *Bs* $\subseteq$ *B* $\wedge$ *A* $\in$ *gen-boolean-algebra S Bs* $\Longrightarrow$
C $\in$ *gen-boolean-algebra S B* $\Longrightarrow$
$\exists$ *Bs. finite Bs* $\wedge$ *Bs* $\subseteq$ *B* $\wedge$ *C* $\in$ *gen-boolean-algebra S Bs* $\Longrightarrow$ $\exists$ *Bs. finite Bs* $\wedge$ *Bs* $\subseteq$ *B* $\wedge$ *A* $\cup$ *C* $\in$ *gen-boolean-algebra S Bs*
proof− **fix** *A C*
assume *A*: *A* $\in$ *gen-boolean-algebra S B*
$\exists$ *Bs. finite Bs* $\wedge$ *Bs* $\subseteq$ *B* $\wedge$ *A* $\in$ *gen-boolean-algebra S Bs*
C $\in$ *gen-boolean-algebra S B*
$\exists$ *Bs. finite Bs* $\wedge$ *Bs* $\subseteq$ *B* $\wedge$ *C* $\in$ *gen-boolean-algebra S Bs*
obtain *As* **where** *As-def*: *finite As* $\wedge$ *As* $\subseteq$ *B* $\wedge$ *A* $\in$ *gen-boolean-algebra S As*
using *A* **by** *blast*
obtain *Cs* **where** *Cs-def*: *finite Cs* $\wedge$ *Cs* $\subseteq$ *B* $\wedge$ *C* $\in$ *gen-boolean-algebra S Cs*
using *A* **by** *blast*
obtain *Bs* **where** *Bs-def*: *Bs* = *As* $\cup$ *Cs*
by *blast*
have *Bs-sub*: *Bs* $\subseteq$ *B*
unfolding *Bs-def* **using** *As-def Cs-def* **by** *blast*
have *0*: *A* $\cup$ *C* $\in$ *gen-boolean-algebra S Bs*
unfolding *Bs-def*
apply(*rule gen-boolean-algebra-generators-union*)
using *As-def* **apply** *blast*
using *Cs-def* **by** *blast*
have *1*: *finite Bs*
unfolding *Bs-def* **using** *As-def Cs-def* **by** *blast*
show $\exists$ *Bs. finite Bs* $\wedge$ *Bs* $\subseteq$ *B* $\wedge$ *A* $\cup$ *C* $\in$ *gen-boolean-algebra S Bs*
using *Bs-sub 0 1* **by** *blast*
qed
show $\bigwedge$*A. A* $\in$ *gen-boolean-algebra S B* $\Longrightarrow$
$\exists$ *Bs. finite Bs* $\wedge$ *Bs* $\subseteq$ *B* $\wedge$ *A* $\in$ *gen-boolean-algebra S Bs* $\Longrightarrow$ $\exists$ *Bs. finite Bs* $\wedge$ *Bs* $\subseteq$ *B* $\wedge$ *S* $-$ *A* $\in$ *gen-boolean-algebra S Bs*
using *gen-boolean-algebra.complement* **by** *blast*
qed

lemma *gen-boolean-algebra-univ-mono*:
assumes $A \in$ *gen-boolean-algebra S B*
shows *gen-boolean-algebra A B* $\subseteq$ *gen-boolean-algebra S B*
proof(*rule subsetI*) **fix** *x* **assume** *A*: $x \in$ *gen-boolean-algebra A B*
obtain *a* **where** *a-def*: $a = A$
by *blast*
have *0*: $a \in$ *gen-boolean-algebra S B*
unfolding *a-def* **using** *assms* **by** *blast*
have *1*: $a = A \cap S$
using *assms gen-boolean-algebra-subset* **unfolding** *a-def* **by** *blast*
show $x \in$ *gen-boolean-algebra S B*
apply(*rule gen-boolean-algebra.induct*[*of x a B*])
using *A a-def* **apply** *blast* **apply**(*rule 0*)
apply (*metis 1 Int-left-commute assms gen-boolean-algebra.intros*(*2*) *gen-boolean-algebra-intersect*)
apply(*rule gen-boolean-algebra.union*, *blast*, *blast*)
apply(*rule gen-boolean-algebra-diff*)
apply(*rule 0*)
by *blast*
qed

The boolean algebra generated by a collection of elements in another algebra is contained in the original algebra:

lemma *gen-boolean-algebra-subalgebra*:
assumes *Xs* $\subseteq$ *gen-boolean-algebra S B*
shows *gen-boolean-algebra S Xs* $\subseteq$ *gen-boolean-algebra S B*
proof fix *x* **assume** *A*: $x \in$ *gen-boolean-algebra S Xs*
show $x \in$ *gen-boolean-algebra S B*
apply(*rule gen-boolean-algebra.induct*[*of x S Xs*])
apply (*simp add*: *A*; *fail*)
apply (*simp add*: *gen-boolean-algebra.universe*; *fail*)
using *assms gen-boolean-algebra.universe gen-boolean-algebra-intersect* **apply** *blast*
apply (*simp add*: *gen-boolean-algebra.union*; *fail*)
by (*simp add*: *gen-boolean-algebra.complement*)
qed

lemma *gen-boolean-algebra-idempotent*:
assumes $S = \bigcup$ *Xs*
shows *gen-boolean-algebra S* (*gen-boolean-algebra S Xs*) = (*gen-boolean-algebra S Xs*)
apply(*rule equalityI*)
apply(*rule subsetI*)
apply (*meson equalityD2 gen-boolean-algebra-subalgebra in-mono*)
apply(*rule subsetI*)
by (*metis gen-boolean-algebra.simps gen-boolean-algebra-subset inf.absorb1*)

We can always replace the set of generators *Xs* with their intersections with the universe set *S*, and obtain the same algebra.

lemma *gen-boolean-algebra-restrict-generators*:
gen-boolean-algebra S Xs =*gen-boolean-algebra S* ((∩) *S* ‘ *Xs*)
proof(*rule equalityI'*)
fix *x* **assume** *A*: *x* ∈ *gen-boolean-algebra S Xs*
show *x* ∈ *gen-boolean-algebra S* ((∩) *S* ‘ *Xs*)
apply(*rule gen-boolean-algebra.induct*[*of x S Xs*], *rule A*, *rule gen-boolean-algebra.universe*)

apply (*metis gen-boolean-algebra.generator image-eqI inf.right-idem inf-commute*)
apply(*rule gen-boolean-algebra.union*, *blast*, *blast*)
by(*rule gen-boolean-algebra-diff*, *rule gen-boolean-algebra.universe*, *blast*)
next
fix *x* **assume** *A*: *x* ∈ *gen-boolean-algebra S* ((∩) *S* ‘ *Xs*)
show *x* ∈ *gen-boolean-algebra S Xs*
apply(*rule gen-boolean-algebra.induct*[*of x S* (∩) *S* ‘ *Xs*], *rule A*, *rule gen-boolean-algebra.universe*,
rule gen-boolean-algebra-intersect)
using *gen-boolean-algebra.generator*[*of - Xs S*]
apply (*metis* (*no-types*, *lifting*) *Int-commute image-iff*)
apply(*rule gen-boolean-algebra.universe*)
apply(*rule gen-boolean-algebra.union*, *blast*, *blast*)
by(*rule gen-boolean-algebra-diff*, *rule gen-boolean-algebra.universe*, *blast*)
qed

Adding a generator to a generated boolean algebra is redundant if the generator already lies in the algebra.

lemma *add-generators*:
assumes *A* ∈ *gen-boolean-algebra S Xs*
shows *gen-boolean-algebra S Xs* = *gen-boolean-algebra S* (*insert A Xs*)
proof(*rule equalityI'*)
fix *x* **assume** *A*: *x* ∈ *gen-boolean-algebra S Xs*
show *x* ∈ *gen-boolean-algebra S* (*insert A Xs*)
apply(*rule gen-boolean-algebra.induct*[*of x S Xs*], *rule A*, *rule gen-boolean-algebra.universe*)
apply(*rule gen-boolean-algebra.generator*, *blast*)
apply(*rule gen-boolean-algebra.union*, *blast*,*blast*)
by(*rule gen-boolean-algebra-diff*, *rule gen-boolean-algebra.universe*, *blast*)
next
fix *x* **assume** *A*: *x* ∈ *gen-boolean-algebra S* (*insert A Xs*)
show *x* ∈ *gen-boolean-algebra S Xs*
apply(*rule gen-boolean-algebra.induct*[*of x S insert A Xs*], *rule A*, *rule gen-boolean-algebra.universe*)
using *assms gen-boolean-algebra.generator*[*of - Xs S*]
using *gen-boolean-algebra.universe gen-boolean-algebra-intersect* **apply** *blast*
apply(*rule gen-boolean-algebra.union*, *blast*, *blast*)
by(*rule gen-boolean-algebra-diff*, *rule gen-boolean-algebra.universe*, *blast*)
qed

12.1 Turning a Family of Sets into a Family of Disjoint Sets

This section outlines the standard construction where sets $A_0, \ldots, A_n$ are replaced by sets $A_0, A_1 - A_0, A_2 - (A_0 \cup A_1), ..., A_n - (\bigcup_{i=0}^{n-1} A_i)$ to obtain a

disjoint family of the same cardinality.

fun *rec-disjointify* **where**
rec-disjointify 0 f = {}|
rec-disjointify (*Suc m*) *f* = *insert* (*f m* − ⋃ (*rec-disjointify m f*)) (*rec-disjointify m f*)

lemma *card-of-rec-disjointify*:
card (*rec-disjointify m f*) ≤ *m*
apply(*induction m*) **unfolding** *rec-disjointify.simps*
apply *simp*
by (*metis Suc-le-mono card.infinite card-insert-disjoint finite-insert insert-absorb le-SucI*)

lemma *rec-disjointify-finite*:
finite (*rec-disjointify m f*)
apply(*induction m*)
unfolding *rec-disjointify.simps* **by** *auto*

lemma *rec-disjointify-in-gen-boolean-algebra*:
assumes *f* ‘ {..<*m*} ⊆ *gen-boolean-algebra S B*
shows *rec-disjointify m f* ⊆ *gen-boolean-algebra S B*
proof−
have ⋀*k*. *k* ≤ *m* ⟶ *rec-disjointify k f* ⊆ *gen-boolean-algebra S B*
proof− **fix** *k* **show** *k* ≤ *m* ⟶ *rec-disjointify k f* ⊆ *gen-boolean-algebra S B*
apply(*induction k*) **unfolding** *rec-disjointify.simps*(*1*) **using** *assms* **apply** *blast*
proof fix *k*
assume *IH*: *k* ≤ *m* ⟶ *rec-disjointify k f* ⊆ *gen-boolean-algebra S B*
Suc k ≤ *m*
then have *0*: *rec-disjointify k f* ⊆ *gen-boolean-algebra S B*
by (*simp add*: *IH*(*2*))
have *1*: *finite* (*rec-disjointify k f*)
using *rec-disjointify-finite* **by** *blast*
have *2*: *f k* ∈ *gen-boolean-algebra S B*
using *IH*(*2*) *assms*
by (*simp add*: *image-subset-iff*)
show *rec-disjointify* (*Suc k*) *f* ⊆ *gen-boolean-algebra S B*
using *0 1 2* **unfolding** *rec-disjointify.simps*
by (*simp add*: *gen-boolean-algebra-diff gen-boolean-algebra-finite-union subset-iff*)
qed
qed
thus *?thesis* **by** *blast*
qed

lemma *rec-disjointify-union*:
⋃ (*rec-disjointify m f*) = (⋃ *i* ∈ {..<*m*}. *f i*)
apply(*induction m*)
apply *simp* **unfolding** *rec-disjointify.simps insert-def*

apply(*rule equalityI*, *rule subsetI*)
apply (*simp add*: *lessThan-Suc*; *fail*)
apply(*rule subsetI*)
by (*simp add*: *lessThan-Suc*)

definition *enum-rec-disjointify* **where**
enum-rec-disjointify f m = *f m* − ⋃ (*rec-disjointify m f*)

lemma *rec-disjointify-as-enum-rec-disjointify-image*:
rec-disjointify m f = *enum-rec-disjointify f* ‘ {..<*m*}
apply(*induction m*)
unfolding *rec-disjointify.simps*
apply (*simp*; *fail*)
unfolding *enum-rec-disjointify-def*
using *lessThan-Suc* **by** *auto*

lemma *enum-rec-disjointify-subset*:
enum-rec-disjointify f m ⊆ *f m*
unfolding *enum-rec-disjointify-def*
by *auto*

lemma *enum-rec-disjointify-disjoint*:
assumes *k* < *m*
shows *enum-rec-disjointify f m* ∩ *enum-rec-disjointify f k* = {}
proof−
have *enum-rec-disjointify f k* ⊆ ⋃ (*rec-disjointify m f*)
unfolding *rec-disjointify-union*
using *assms enum-rec-disjointify-subset* **by** *fastforce*
thus *?thesis*
unfolding *enum-rec-disjointify-def*
by *auto*
qed

lemma *enum-rec-disjointify-disjoint′*:
assumes *k* ≠ *m*
shows *enum-rec-disjointify f m* ∩ *enum-rec-disjointify f k* = {}
apply(*cases k* < *m*) **using** *enum-rec-disjointify-disjoint*[*of k m f*]
apply *simp*
using *assms enum-rec-disjointify-disjoint*[*of m k f*] **by** *auto*

lemma *rec-disjointify-is-disjoint*:
assumes *A* ∈ *rec-disjointify m f*
assumes *B* ∈ *rec-disjointify m f*
assumes *A* ≠ *B*
shows *A* ∩ *B* = {}
using *rec-disjointify-as-enum-rec-disjointify-image enum-rec-disjointify-disjoint′*
assms
by (*smt image-iff*)

definition *enumerates* **where**
enumerates A f ≡ *finite A* ∧ *A* = *f* ‘ {..< (*card A*)} ∧ *inj-on f* {..< (*card A*)}

lemma *finite-imp-exists-enumeration*:
assumes *finite A*
shows ∃*f*. *enumerates A f*
unfolding *enumerates-def*
using *assms finite-imp-nat-seg-image-inj-on*[*of A*]
by (*metis card-Collect-less-nat card-image lessThan-def*)

lemma *enumeratesE*:
assumes *enumerates A f*
shows *finite A A* = *f* ‘ {..< *card A*} *inj-on f* {..< *card A*}
using *assms* **unfolding** *enumerates-def* **apply** *blast*
using *assms* **unfolding** *enumerates-def* **apply** *blast*
using *assms* **unfolding** *enumerates-def* **by** *blast*

lemma *rec-disjointify-finite-set*:
assumes *enumerates A f*
shows ⋃ (*rec-disjointify* (*card A*) *f*) = ⋃ *A*
unfolding *rec-disjointify-union*[*of card A f*]
using *enumeratesE*[*of A f*] *assms* **by** *auto*

definition *enumerate* **where**
enumerate A = (*SOME f*. *enumerates A f*)

lemma *enumerate-enumerates*:
assumes *finite A*
shows *enumerates A* (*enumerate A*)
unfolding *enumerate-def* **using** *finite-imp-exists-enumeration assms*
by (*simp add*: *finite-imp-exists-enumeration some-eq-ex*)

lemma *enumerateE*:
assumes *finite A*
assumes *a* ∈ *A*
shows ∃ *i* < *card A*. *a* = (*enumerate A*) *i*
using *enumerate-enumerates*[*of A*] *enumeratesE*[*of A*] *assms* **by** *blast*

definition *disjointify* **where**
disjointify As = *rec-disjointify* (*card As*) (*enumerate As*)

lemma *disjointify-is-disjoint*:
assumes *finite As*
assumes *A* ∈ *disjointify As*
assumes *B* ∈ *disjointify As*
assumes *A* ≠ *B*
shows *A* ∩ *B* = {}
using *assms rec-disjointify-is-disjoint*[*of A - - B*] **unfolding** *disjointify-def*
by *simp*

lemma *disjointify-union*:
assumes *finite As*
shows $\bigcup$ (*disjointify As*) = $\bigcup$ *As*
using *assms*
by (*simp add*: *disjointify-def enumerate-enumerates rec-disjointify-finite-set*)

lemma *disjointify-gen-boolean-algebra*:
assumes *finite As*
assumes *As* ⊆ *gen-boolean-algebra S B*
shows *disjointify As* ⊆ *gen-boolean-algebra S B*
using *assms* **unfolding** *disjointify-def*
by (*metis enumerate-enumerates enumeratesE*(*2*) *rec-disjointify-in-gen-boolean-algebra*)

lemma *disjointify-finite*:
assumes *finite As*
shows *finite* (*disjointify As*)
using *assms* **unfolding** *disjointify-def*
by (*simp add*: *rec-disjointify-finite*)

lemma *disjointify-card*:
assumes *finite As*
shows*card* (*disjointify As*) ≤ *card As*
by (*simp add*: *card-of-rec-disjointify disjointify-def*)

lemma *disjointify-subset*:
assumes *finite As*
assumes *A* ∈ *disjointify As*
shows ∃ *B* ∈ *As*. *A* ⊆ *B*
using *assms enum-rec-disjointify-subset enumerate-enumerates enumeratesE*
unfolding *disjointify-def*
by (*smt image-iff rec-disjointify-as-enum-rec-disjointify-image*)

12.2 The Atoms Generated by Collections of Sets

We can also turn a family of sets into a disjoint family by taking the atoms of the boolean algebra generated by these sets. This will still yield a finite family if the initial family is finite, but in general will be much larger in size.

12.2.1 Defining the Atoms of a Family of Sets

Here we intend that *As* is a subset of the collection of sets *Xs*. This function associate to each subset *As* ⊆ *Xs* a set which is contained in each element of *As*, and is disjoint from each element of *Xs* − *As*. Note that in general this may yield the empty set, but we will ultimately be interested in the cases where the result is nonempty.

definition *subset-to-atom* **where**

subset-to-atom Xs As = $\bigcap$ *As* − $\bigcup$ (*Xs* − *As*)

lemma *subset-to-atom-memI*:
assumes $\bigwedge A.\ A \in As \Longrightarrow x \in A$
assumes $\bigwedge A.\ A \in Xs \Longrightarrow A \notin As \Longrightarrow x \notin A$
shows *x* ∈ *subset-to-atom Xs As*
using *assms* **unfolding** *subset-to-atom-def*
by *blast*

lemma *subset-to-atom-memE*:
assumes *x* ∈ *subset-to-atom Xs As*
shows $\bigwedge A.\ A \in As \Longrightarrow x \in A$
$\bigwedge A.\ A \in Xs \Longrightarrow A \notin As \Longrightarrow x \notin A$
using *assms* **unfolding** *subset-to-atom-def* **by** *auto*

lemma *subset-to-atom-closed*:
assumes *As* ≠ {}
assumes *As* ⊆ *Xs*
shows *subset-to-atom Xs As* ⊆ $\bigcup$ *Xs*
proof−
have *0*: $\bigcap$ *As* ⊆ $\bigcup$ *As*
apply(*rule subsetI*)
using *assms*(*1*) **by** *blast*
show *?thesis*
apply(*rule subsetI*)
using *assms 0* **unfolding** *subset-to-atom-def*
by (*meson DiffD1 Union-mono subsetD*)
qed

lemma *subset-to-atom-as-intersection*:
assumes *As* ≠ {}
assumes *As* ⊆ *Xs*
assumes *S* = $\bigcup$ *Xs*
shows *subset-to-atom Xs As* = $\bigcap$ *As* ∩ ($\bigcap$ *X* ∈ *Xs* − *As*. *S* − *X*)
unfolding *assms subset-to-atom-def*
apply(*rule equalityI'*)
apply(*rule IntI*, *blast*)
apply(*rule InterI*)
using *INT-I assms*(*1*) *assms*(*2*) **apply** *auto*[*1*]
apply(*rule DiffI*, *blast*)
by *blast*

definition *atoms-of* **where**
atoms-of Xs = (*subset-to-atom Xs* ‘ ((*Pow Xs*) − {{}})) − {{}}

lemma *atoms-nonempty*:
assumes *A* ∈ *atoms-of Xs*
shows *A* ≠ {}
using *assms* **unfolding** *atoms-of-def* **by** *blast*

lemma *atoms-of-disjoint*:
assumes $A \in$ *atoms-of Xs*
assumes $B \in$ *atoms-of Xs*
assumes $A \neq B$
shows $A \cap B = \{\}$
proof−
obtain *a* **where** *a-def*: $a \subseteq Xs \wedge A =$ *subset-to-atom Xs a*
using *assms* **unfolding** *atoms-of-def* **by** *blast*
obtain *b* **where** *b-def*: $b \subseteq Xs \wedge B =$ *subset-to-atom Xs b*
using *assms* **unfolding** *atoms-of-def* **by** *blast*
have *a-neq-b*: $a \neq b$
using *assms* *a-def b-def* **by** *blast*
have $A \cap B \subseteq \{\}$
proof fix *x* **assume** *A*: $x \in A \cap B$
show $x \in \{\}$
proof(*cases* $a \subseteq b$)
case *True*
then obtain *c* **where** *c-def*: $c \in b - a$
using *a-neq-b* **by** *blast*
have *c-in-Xs*: $c \in Xs$
using *c-def b-def* **by** *blast*
have *x-in-c*: $x \in c$
using *A* *b-def c-def subset-to-atom-memE*[*of x Xs b c*] **by** *blast*
have *x-notin-c*: $x \notin c$
using *A* *a-def c-in-Xs c-def subset-to-atom-memE*[*of x Xs a c*] **by** *blast*
then show *?thesis* **using** *x-in-c* **by** *blast*
next
case *False*
then obtain *c* **where** *c-def*: $c \in a - b$
using *a-neq-b* **by** *blast*
have *c-in-Xs*: $c \in Xs$
using *c-def a-def* **by** *blast*
have *x-in-c*: $x \in c$
using *A* *a-def c-def subset-to-atom-memE*[*of x Xs a c*] **by** *blast*
have *x-notin-c*: $x \notin c$
using *A* *b-def c-in-Xs c-def subset-to-atom-memE*[*of x Xs b c*] **by** *blast*
then show *?thesis* **using** *x-in-c* **by** *blast*
qed
qed
thus $A \cap B = \{\}$
by *blast*
qed

The atoms of a family of sets *Xs* are minimal in the sense that they are either contained in or disjoint from each element of *Xs*.

lemma *atoms-are-minimal*:
assumes $A \in$ *atoms-of Xs*
assumes $X \in Xs$

shows $X \cap A = \{\} \vee A \subseteq X$
proof(*cases* $X \cap A = \{\}$)
case *True*
then show *?thesis* **by** *blast*
next
case *False*
obtain *As* **where** *As-def*: $As \in Pow\ Xs - \{\{\}\} \wedge A = subset\text{-}to\text{-}atom\ Xs\ As$
using *assms* **unfolding** *atoms-of-def* **by** *blast*
have *A-simp*: $A = subset\text{-}to\text{-}atom\ Xs\ As$
using *As-def* **by** *blast*
then show *?thesis* **using** *assms* **unfolding** *atoms-of-def subset-to-atom-def A-simp*
using *DiffD1 subset-eq* **by** *auto*
qed

12.2.2 Atoms Induced by Types of Points

The set of sets in Xs which contain some point x. In the case where Xs is some collection of first order formulas, this is just the type of x over these formulas.

definition *point-to-type* **where**
$point\text{-}to\text{-}type\ Xs\ x = \{X \in Xs.\ x \in X\}$

The type of a point x induces the unique atom of Xs which contains x.

lemma *point-in-atom-of-type*:
assumes $x \in \bigcup Xs$
shows $x \in subset\text{-}to\text{-}atom\ Xs\ (point\text{-}to\text{-}type\ Xs\ x)$
using *assms* **unfolding** *subset-to-atom-def point-to-type-def*
by *blast*

lemma *point-to-type-nonempty*:
assumes $x \in \bigcup Xs$
shows $point\text{-}to\text{-}type\ Xs\ x \neq \{\}$
using *assms* **unfolding** *point-to-type-def*
by *blast*

lemma *point-to-type-closed*:
$point\text{-}to\text{-}type\ Xs\ x \subseteq Pow\ (\bigcup Xs)$
unfolding *point-to-type-def*
by *blast*

lemma *atoms-of-covers*:
assumes $X = \bigcup Xs$
shows $\bigcup\ (atoms\text{-}of\ Xs) = X$
proof
show $\bigcup\ (atoms\text{-}of\ Xs) \subseteq X$
proof fix x **assume** A: $x \in \bigcup\ (atoms\text{-}of\ Xs)$
then obtain *As* **where** *As-def*: $As \in Pow\ Xs - \{\{\}\} \wedge x \in subset\text{-}to\text{-}atom\ Xs\ As$

unfolding *atoms-of-def* **by** *blast*
have *subset-to-atom Xs As* $\subseteq \bigcup$ *Xs*
using *subset-to-atom-closed*[*of As Xs*] *As-def* **by** *blast*
then show $x \in X$ **unfolding** *assms*
using *As-def* **by** *blast*
qed
show $X \subseteq \bigcup$ (*atoms-of Xs*) **apply**(*rule subsetI*)
using *point-to-type-nonempty point-in-atom-of-type point-to-type-closed*
unfolding *assms point-to-type-def atoms-of-def*
by *fastforce*
qed

lemma *atoms-of-covers′*:
shows $\bigcup$ (*atoms-of Xs*) = $\bigcup$ *Xs*
using *atoms-of-covers*[*of* $\bigcup$ *Xs*] **by** *blast*

Every atom of a collection *Xs* of sets is realized as the atom generated by the type of an element in that atom.

lemma *nonemtpy-atom-from-point-to-type*:
assumes $A \in$ *atoms-of Xs*
assumes $a \in A$
shows A = *subset-to-atom Xs* (*point-to-type Xs a*)
proof−
obtain *As* **where** *As-def*: $As \in (Pow\ Xs) - \{\} \wedge A$ = *subset-to-atom Xs As*
using *assms* **unfolding** *atoms-of-def* **by** *blast*
have *A-simp*: A = *subset-to-atom Xs As*
using *As-def* **by** *blast*
have *0*: *As* = *point-to-type Xs a*
apply(*rule equalityI*)
apply(*rule subsetI*)
apply (*smt As-def Diff-empty UnionI Union-Pow-eq assms point-in-atom-of-type subset-to-atom-memE*(*1*) *subset-to-atom-memE*(*2*))
apply(*rule subsetI*)
using *As-def assms subset-to-atom-memE*(*2*)
by (*metis* (*no-types*, *lifting*) *mem-Collect-eq point-to-type-def*)
show *?thesis*
using *point-in-atom-of-type 0*
atoms-of-covers′[*of Xs*] *assms* **unfolding** *A-simp*
by *auto*
qed

In light of the previous theorem, a point a and a collection of sets *Xs* is enough to recover the the unique atom of *Xs* which contains *a*.

definition *point-to-atom* **where**
point-to-atom Xs a = *subset-to-atom Xs* (*point-to-type Xs a*)

lemma *point-to-atom-closed*:
assumes $x \in \bigcup$ *Xs*
shows *point-to-atom Xs x* $\in$ *atoms-of Xs*

using *assms* **unfolding** *atoms-of-def point-to-atom-def*
by (*metis* (*full-types*) *Union-iff atoms-of-covers atoms-of-def nonemtpy-atom-from-point-to-type*)

All atoms of *Xs* are the atom induced by some point in the union of *Xs*.

lemma *atoms-induced-by-points*:
atoms-of Xs = *point-to-atom Xs* ‘ (⋃ *Xs*)
apply(*rule equalityI*)
apply(*rule subsetI*)
using *nonemtpy-atom-from-point-to-type atoms-nonempty atoms-of-covers′*
unfolding *point-to-atom-def*
apply (*smt DiffE Pow-empty Pow-iff atoms-of-def image-iff subsetD subsetI subset-to-atom-closed*)
apply(*rule subsetI*)
by (*metis* (*no-types*, *lifting*) *imageE point-to-atom-closed point-to-atom-def*)

12.2.3 Atoms of Generated Boolean Algebras

lemma *atoms-of-gen-boolean-algebra*:
assumes *Xs* ⊆ *gen-boolean-algebra S B*
assumes *finite Xs*
shows *atoms-of Xs* ⊆ *gen-boolean-algebra S B*
proof
fix *x* **assume** *A*: *x* ∈ *atoms-of Xs*
then obtain *As* **where** *As-def*: *As* ∈ ((*Pow Xs*) − {{}}) ∧ *x* = *subset-to-atom Xs As*
unfolding *atoms-of-def* **by** *blast*
have *x-simp*: *x* = *subset-to-atom Xs As*
using *As-def* **by** *blast*
have *0*: *finite As*
using *As-def assms finite-subset* **by** *auto*
have *1*: *As* ⊆ *gen-boolean-algebra S B*
using *As-def assms* **by** *blast*
have *2*: ⋂ *As* ∈ *gen-boolean-algebra S B*
using *0 1 assms*
by (*metis As-def DiffE gen-boolean-algebra-finite-intersection singletonI subset-eq*)
show *x* ∈ *gen-boolean-algebra S B*
using *A 2* **unfolding** *atoms-of-def subset-to-atom-def x-simp*
by (*metis* (*no-types*, *lifting*) *As-def DiffD1 Diff-partition Pow-iff Un-subset-iff assms*(*1*) *assms*(*2*) *finite-subset gen-boolean-algebra-diff gen-boolean-algebra-finite-union order-refl subsetD*)
qed

If the generators of a boolean algebra are contained in the universe, the atoms induced by the generators alone are minimal elements of the entire algebra.

lemma *finite-algebra-atoms-are-minimal*:
assumes *finite Xs*
assumes ⋃ *Xs* ⊆ *S*

assumes $A \in$ *atoms-of Xs*
assumes $X \in$ *gen-boolean-algebra S Xs*
shows $X \cap A = \{\} \vee A \subseteq X$
apply(*rule gen-boolean-algebra.induct*[*of X S Xs*])
apply (*simp add*: *assms*(*4*); *fail*)
apply (*metis Union-upper assms*(*2*) *assms*(*3*) *atoms-of-covers dual-order.trans*)
using *assms*(*2*) *assms*(*3*) *atoms-are-minimal* **apply** *fastforce*
apply *blast*
using *assms*
by (*metis Diff-Int-distrib2 Diff-empty Diff-eq-empty-iff Sup-upper atoms-of-covers' equalityE inf.absorb-iff2 order-trans*)

lemma *finite-set-imp-finite-atoms*:
assumes *finite Xs*
shows *finite* (*atoms-of Xs*)
using *assms* **unfolding** *atoms-of-def*
by *blast*

Every element in the boolean algebra generated by Xs over S is a (disjoint) union of atoms of generators:

lemma *gen-boolean-algebra-elem-uni-of-atoms*:
assumes *finite Xs*
assumes $S = \bigcup$ *Xs*
assumes $X \in$ *gen-boolean-algebra S Xs*
shows $X = \bigcup \{a \in$ *atoms-of Xs*. $a \subseteq X\}$
proof
show $X \subseteq \bigcup \{a \in$ *atoms-of Xs*. $a \subseteq X\}$
proof fix x **assume** A: $x \in X$
then have *point-to-atom Xs x* $\in$ *atoms-of Xs*
using *assms* **by** (*meson gen-boolean-algebra-subset point-to-atom-closed subsetD*)
then show $x \in \bigcup \{a \in$ *atoms-of Xs*. $a \subseteq X\}$
by (*smt A IntI Union-iff assms*(*1*) *assms*(*2*) *assms*(*3*) *empty-iff finite-algebra-atoms-are-minimal gen-boolean-algebra.universe gen-boolean-algebra-subset mem-Collect-eq point-in-atom-of-type point-to-atom-def subsetD*)
qed
show $\bigcup \{a \in$ *atoms-of Xs*. $a \subseteq X\} \subseteq X$
by *blast*
qed

In fact, every generated boolean algebra is the power set of the atoms of its generators:

lemma *gen-boolean-algebra-generated-by-atoms*:
assumes *finite Xs*
assumes $S = \bigcup$ *Xs*
shows *gen-boolean-algebra S Xs* $= \bigcup$ ‘ (*Pow* (*atoms-of Xs*))
proof
show *gen-boolean-algebra S Xs* $\subseteq \bigcup$ ‘ *Pow* (*atoms-of Xs*)
apply(*rule subsetI*)

using *gen-boolean-algebra-elem-uni-of-atoms*[*of Xs S*] *assms*
by *fastforce*
show $\bigcup$ ' *Pow* (*atoms-of Xs*) $\subseteq$ *gen-boolean-algebra S Xs*
apply(*rule subsetI*)
using *atoms-of-gen-boolean-algebra*[*of Xs S Xs*]
finite-subset[*of - atoms-of Xs*] *assms*
finite-set-imp-finite-atoms[*of Xs*]
gen-boolean-algebra-finite-union[*of - S Xs*]
by (*smt Pow-iff Union-upper gen-boolean-algebra.intros*(*2*) *image-iff inf.absorb1 subsetD subsetI*)
qed

Finitely generated boolean algebras are finite

lemma *fin-gens-imp-fin-algebra*:
assumes *finite Xs*
assumes $S = \bigcup$ *Xs*
shows *finite* (*gen-boolean-algebra S Xs*)
using *finite-set-imp-finite-atoms*[*of Xs*] *assms gen-boolean-algebra-generated-by-atoms*[*of Xs S*]
by *simp*

lemma *point-to-atom-equal*:
assumes *finite Xs*
assumes $S = \bigcup$ *Xs*
assumes $x \in S$
shows *point-to-atom Xs x* = *point-to-atom* (*gen-boolean-algebra S Xs*) *x*
proof
show *P0*: *point-to-atom Xs x* $\subseteq$ *point-to-atom* (*gen-boolean-algebra S Xs*) *x*
proof−
have *0*: *point-to-atom Xs x* $\cap$ *point-to-atom* (*gen-boolean-algebra S Xs*) *x* $\neq$ {}
using *assms*
by (*metis IntI UnionI empty-iff gen-boolean-algebra.universe point-in-atom-of-type point-to-atom-def*)
have *1*: *point-to-atom* (*gen-boolean-algebra S Xs*) *x* $\in$ *gen-boolean-algebra S Xs*
using *assms fin-gens-imp-fin-algebra*[*of Xs S*]
by (*meson UnionI atoms-of-gen-boolean-algebra gen-boolean-algebra.simps point-to-atom-closed subset-eq subset-refl*)
then show *?thesis*
using *0 finite-algebra-atoms-are-minimal*[*of Xs S point-to-atom Xs x point-to-atom* (*gen-boolean-algebra S Xs*) *x*]
assms(*1*) *assms*(*2*) *assms*(*3*) *atoms-induced-by-points* **by** *auto*
qed
show *point-to-atom* (*gen-boolean-algebra S Xs*) *x* $\subseteq$ *point-to-atom Xs x*
proof−
have *0*: *point-to-atom* (*gen-boolean-algebra S Xs*) *x* $\cap$ *point-to-atom Xs x* $\neq${}
using *assms P0 point-in-atom-of-type point-to-atom-def* **by** *fastforce*
have *1*: *point-to-atom* (*gen-boolean-algebra S Xs*) *x* $\in$ (*gen-boolean-algebra S Xs*)

using *assms gen-boolean-algebra-idempotent*[*of S Xs*] *atoms-of-gen-boolean-algebra*

by (*metis UnionI fin-gens-imp-fin-algebra gen-boolean-algebra.universe point-to-atom-closed subset-eq*)
have *2*: $\bigcup$ (*gen-boolean-algebra S Xs*) $\subseteq$ *S*
using *assms*
by (*simp add*: *Sup-le-iff gen-boolean-algebra-subset*)
hence *3*: $\bigcup$ (*gen-boolean-algebra S Xs*) = *S*
by (*simp add*: *Union-upper gen-boolean-algebra.universe subset-antisym*)
have *4*: *gen-boolean-algebra S* (*gen-boolean-algebra S Xs*) = *gen-boolean-algebra S Xs*
using *assms gen-boolean-algebra-idempotent*[*of S Xs*] **by** *blast*
have *5*: *point-to-atom Xs x* ∈ *gen-boolean-algebra S* (*gen-boolean-algebra S Xs*)
unfolding *4* **using** *assms*
by (*metis* (*no-types*, *opaque-lifting*) *Int-absorb1 Int-commute Union-upper atoms-of-gen-boolean-algebra gen-boolean-algebra.generator point-to-atom-closed subsetD subsetI*)
show *?thesis*
using *2 5 finite-algebra-atoms-are-minimal*[*of gen-boolean-algebra S Xs S point-to-atom* (*gen-boolean-algebra S Xs*) *x point-to-atom Xs x*] *0 1 2*
unfolding *4*
by (*metis 3 Int-commute assms*(*1*) *assms*(*2*) *assms*(*3*) *fin-gens-imp-fin-algebra point-to-atom-closed*)
qed
qed

When the set *Xs* of generators covers the universe set *S*, the atoms of *Xs* in the above sense are the same as the atoms of the boolean algebra they generate over *S*.

lemma *atoms-of-sets-eq-atoms-of-algebra*:
assumes *finite Xs*
assumes *S* = $\bigcup$ *Xs*
shows *atoms-of Xs* = *atoms-of* (*gen-boolean-algebra S Xs*)
proof
show *atoms-of Xs* ⊆ *atoms-of* (*gen-boolean-algebra S Xs*)
proof fix *A* **assume** *A*: *A* ∈ *atoms-of Xs*
then obtain *x* **where** *x-def*: *x* ∈ *S* ∧ *A* = *point-to-atom Xs x*
using *assms*
by (*metis atoms-induced-by-points image-iff*)
have *0*: *A* = *point-to-atom* (*gen-boolean-algebra S Xs*) *x*
using *assms point-to-atom-equal x-def* **by** *fastforce*
show *A* ∈ *atoms-of* (*gen-boolean-algebra S Xs*)
unfolding *0* **using** *assms A*
by (*metis* (*full-types*) *0 UnionI gen-boolean-algebra.universe point-to-atom-closed x-def*)
qed
show *atoms-of* (*gen-boolean-algebra S Xs*) ⊆ *atoms-of Xs*
proof fix *A* **assume** *A*: *A* ∈ *atoms-of* (*gen-boolean-algebra S Xs*)
then obtain *x* **where** *x-def*: *x* ∈ *S* ∧ *A* = *point-to-atom* (*gen-boolean-algebra*

S Xs) *x*
by (*metis atoms-induced-by-points cSup-eq-maximum gen-boolean-algebra.universe gen-boolean-algebra-subset image-iff*)
then show *A* ∈ *atoms-of Xs*
using *assms*(*1*) *assms*(*2*) *point-to-atom-closed point-to-atom-equal* **by** *fastforce*
qed
qed

lemma *atoms-closed*:
assumes *finite Xs*
assumes *A* ∈ *atoms-of* (*gen-boolean-algebra S Xs*)
assumes *S* = ⋃ *Xs*
shows *A* ∈ (*gen-boolean-algebra S Xs*)
proof−
have *1*: *A* = ⋃ {*A*}
by *blast*
have *2*: *A* ∈ *atoms-of Xs*
using *assms atoms-of-sets-eq-atoms-of-algebra*
by *blast*
show *?thesis*
using *gen-boolean-algebra-generated-by-atoms*[*of Xs S*]
assms 1 2 **unfolding** *Pow-def* **by** *blast*
qed

lemma *atoms-finite*:
assumes *finite Xs*
shows *finite* ((*atoms-of* (*gen-boolean-algebra S Xs*)))
proof−
have *0*: *gen-boolean-algebra S Xs* =*gen-boolean-algebra S* ((∩) *S* ‘ *Xs*)
using *gen-boolean-algebra-restrict-generators* **by** *blast*
have *1*: *gen-boolean-algebra S Xs* = *gen-boolean-algebra S* (*insert S* ((∩) *S* ‘ *Xs*))
unfolding *0* **by**(*rule add-generators*, *rule gen-boolean-algebra.universe*)
obtain *Ys* **where** *Ys-def*: *Ys* = (*insert S* ((∩) *S* ‘ *Xs*))
by *blast*
have *Ys-finite*: *finite Ys*
unfolding *Ys-def* **using** *assms* **by** *blast*
have *2*: ⋃ *Ys* = *S*
unfolding *Ys-def*
by *blast*
have *3*: *atoms-of Ys* = *atoms-of* (*gen-boolean-algebra S Xs*)
unfolding *Ys-def 1*
apply(*rule atoms-of-sets-eq-atoms-of-algebra*)
using *Ys-finite* **unfolding** *Ys-def* **apply** *blast*
by *blast*
have *4*: *finite* (*atoms-of Ys*)
by(*rule finite-set-imp-finite-atoms*, *rule Ys-finite*)
show *?thesis* **using** *4* **unfolding** *3* **by** *blast*
qed

We can distinguish atoms of a set of generators *Cs* by finding some element

of *Cs* which includes one and excludes the other.

lemma *distinct-atoms*:
assumes $Cs \neq \{\}$
assumes $a \in$ *atoms-of Cs*
assumes $b \in$ *atoms-of Cs*
assumes $a \neq b$
shows $(\exists B \in Cs.\ b \subseteq B \wedge a \cap B = \{\}) \vee (\exists A \in Cs.\ a \subseteq A \wedge b \cap A = \{\})$
proof−
obtain x **where** *x-def*: $x \in \bigcup Cs \wedge a =$ *point-to-atom Cs x*
by (*metis assms*(2) *atoms-induced-by-points imageE*)
obtain y **where** *y-def*: $y \in \bigcup Cs \wedge b =$ *point-to-atom Cs y*
by (*metis assms*(3) *atoms-induced-by-points imageE*)
have *0*: *point-to-atom Cs x* $\neq$ *point-to-atom Cs y*
using *x-def y-def assms* **by** *simp*
hence *1*: *point-to-type Cs x* $\neq$ *point-to-type Cs y*
unfolding *point-to-atom-def subset-to-atom-def* **by** *blast*
then obtain B **where** *B-def*: $B \in Cs \wedge (B \in$ *point-to-type Cs x* − *point-to-type Cs y* $\vee$ $B \in$ *point-to-type Cs y* − *point-to-type Cs x*)
unfolding *point-to-type-def* **by** *blast*
have *2*: $B \in$ *point-to-type Cs x* − *point-to-type Cs y* $\Longrightarrow a \subseteq B$
using *x-def point-to-atom-def subset-to-atom-memE*(1) **by** *fastforce*
have *3*: $B \in$ *point-to-type Cs y* − *point-to-type Cs y* $\Longrightarrow b \subseteq B$
using *y-def* **by** *blast*
show *?thesis* **using** *B-def 2 3*
by (*smt Diff-iff disjoint-iff-not-equal point-to-atom-def subset-eq subset-to-atom-memE*(1) *subset-to-atom-memE*(2) *x-def y-def*)
qed

12.3 Partitions of a Set

definition *disjoint* :: *'a set set* ⇒ *bool* **where**
disjoint Ss = $(\forall\ A \in Ss.\ \forall B \in Ss.\ \ A \neq B \longrightarrow A \cap B = \{\})$

lemma *disjointE*:
assumes *disjoint Ss*
assumes $A \in Ss$
assumes $B \in Ss$
assumes $A \neq B$
shows $A \cap B = \{\}$
by (*meson assms*(1) *assms*(2) *assms*(3) *assms*(4) *disjoint-def*)

lemma *disjointI*:
assumes $\bigwedge A\ B.\ A \in Ss \Longrightarrow B \in Ss \Longrightarrow A \neq B \Longrightarrow A \cap B = \{\}$
shows *disjoint Ss*
by (*meson assms disjoint-def*)

definition *is-partition* :: *'a set set* ⇒ *'a set* ⇒ *bool* (**infixl** ‹*partitions*› 75) **where**
S partitions A = (*disjoint S* $\wedge \bigcup S = A$)

lemma *is-partitionE*:
assumes *S partitions A*
shows *disjoint S*
$\bigcup S = A$
using *assms is-partition-def* **apply** *blast*
using *assms*
by (*simp add*: *is-partition-def*)

lemma *is-partitionI*:
assumes *disjoint S*
assumes $\bigcup S = A$
shows *S partitions A*
using *assms is-partition-def* **by** *blast*

If we start with a finite partition of a set A, and each element in that partition has a finite partition with some property P, then A itself has a finite partition where each element has property P.

lemma *iter-partition*:
assumes *As partitions A*
assumes *finite As*
assumes $\bigwedge a.\ a \in As \Longrightarrow \exists\, Bs.\ finite\ Bs \wedge Bs\ partitions\ a \wedge (\forall\, b \in Bs.\ P\ b)$
shows $\exists\, Bs.\ finite\ Bs \wedge Bs\ partitions\ A \wedge (\forall\, b \in Bs.\ P\ b)$
proof−
obtain *F* **where** *F-def*: $F = (\lambda a.\ (SOME\ Bs.\ finite\ Bs \wedge Bs\ partitions\ a \wedge (\forall\, b \in Bs.\ P\ b)))$
by *blast*
have *FE*: $\bigwedge a.\ a \in As \Longrightarrow finite\ (F\ a) \wedge (F\ a)\ partitions\ a \wedge (\forall\, b \in (F\ a).\ P\ b)$
proof− **fix** *a* **assume** *A*: $a \in As$
show *finite* $(F\ a) \wedge (F\ a)$ *partitions* $a \wedge (\forall\, b \in (F\ a).\ P\ b)$
apply(*rule SomeE'*[*of* - $\lambda Bs.$ *finite Bs* $\wedge$ *Bs partitions a* $\wedge$ $(\forall\, b \in Bs.\ P\ b)$])
unfolding *F-def* **apply** *blast*
using *assms* **by** (*simp add*: *A*)
qed
obtain *Bs* **where** *Bs-def*: $Bs = (\bigcup\ a \in As.\ F\ a)$
by *blast*
have *0*: *finite Bs*
unfolding *Bs-def* **using** *FE assms* **by** *blast*
have *1*: *disjoint Bs*
proof(*rule disjointI*)
fix *a b* **assume** *A*: $a \in Bs\ b \in Bs\ a \neq b$
obtain *c* **where** *c-def*: $c \in As \wedge a \in F\ c$
using *Bs-def A* **by** *blast*
obtain *d* **where** *d-def*: $d \in As \wedge b \in F\ d$
using *Bs-def A* **by** *blast*
have *0*: $a \subseteq c$
using *c-def FE*[*of c*] *is-partitionE*(*2*)[*of F c c*] **by** *blast*
have *1*: $b \subseteq d$
using *d-def FE*[*of d*] *is-partitionE*(*2*)[*of F d d*] **by** *blast*
show $a \cap b = \{\}$

proof(*cases* $c = d$)
case *True*
show *?thesis* **apply**(*rule disjointE*[*of* F c])
unfolding *True* **using** *FE is-partitionE d-def* **apply** *blast*
using *c-def* **unfolding** *True* **apply** *blast*
using *d-def* **apply** *blast*
by(*rule A*)
next
case *False*
have $c \cap d = \{\}$
apply(*rule disjointE*[*of As*])
using *assms is-partitionE* **apply** *blast*
using *c-def* **apply** *blast*
using *d-def* **apply** *blast*
using *False* **by** *blast*
then show *?thesis* **using** *0 1* **by** *blast*
qed
qed
have *2*: $(\forall\, b \in Bs.\ P\ b)$
apply(*rule*)
unfolding *Bs-def* **using** *FE*
by *blast*
have *FE'*: $\bigwedge a.\ a \in As \Longrightarrow (\bigcup\ (F\ a)) = a$
apply(*rule is-partitionE*)
using *FE* **by** *blast*
have *3*: *Bs partitions A*
apply(*rule is-partitionI*, *rule 1*)
apply(*rule equalityI'*)
unfolding *Bs-def* **using** *assms is-partitionE*(*2*)[*of As A*]
FE' is-partitionE(*2*) **apply** *blast*
proof−
fix x **assume** *A*: $x \in A$
then obtain a **where** *a-def*: $a \in As \wedge x \in a$
using *assms is-partitionE* **by** *blast*
then have $x \in (\bigcup\ (F\ a))$
using *a-def FE'* **by** *blast*
thus $x \in \bigcup\ (\bigcup\ (F\ `\ As))$
using *a-def A* **by** *blast*
qed
show $\exists\, Bs.$ *finite Bs* $\wedge$ *Bs partitions A* $\wedge$ $(\forall\, b{\in}Bs.\ P\ b)$
using *0 2 3* **by** *blast*
qed

12.4 Intersections of Families of Sets

definition *pairwise-intersect* **where**
pairwise-intersect As Bs $= \{c.\ \exists\, a \in As.\ \exists\, b \in Bs.\ c = a \cap b\}$

lemma *partition-intersection*:

assumes *As partitions A*
assumes *Bs partitions B*
shows (*pairwise-intersect As Bs*) *partitions* ($A \cap B$)
proof(*rule is-partitionI*, *rule disjointI*)
fix *a b* **assume** *a0*: $a \in$ *pairwise-intersect As Bs* $b \in$ *pairwise-intersect As Bs a* $\neq b$
obtain *a1 b1* **where** *def1*: $a1 \in As \wedge b1 \in Bs \wedge a = a1 \cap b1$
using *a0* **unfolding** *pairwise-intersect-def* **by** *blast*
obtain *a2 b2* **where** *def2*: $a2 \in As \wedge b2 \in Bs \wedge b = a2 \cap b2$
using *a0* **unfolding** *pairwise-intersect-def* **by** *blast*
have *0*: $a \cap b = (a1 \cap a2) \cap (b1 \cap b2)$
using *def1 def2* **by** *blast*
show $a \cap b = \{\}$
proof(*cases* $a1 \neq a2$)
case *True*
have *T0*: $a1 \cap a2 = \{\}$
apply(*rule disjointE*[*of As a1 a2*])
using *def1 def2 assms*(*1*) *True is-partitionE*(*1*)[*of As A*] **apply** *blast*
using *def1* **apply** *blast* **using** *def2* **apply** *blast* **by**(*rule True*)
thus *?thesis* **unfolding** *0* **by** *blast*
next
case *False*
then have *F0*: $b1 \neq b2$
using *a0 def1 def2* **by** *blast*
have *F1*: $b1 \cap b2 = \{\}$
apply(*rule disjointE*[*of Bs b1 b2*])
using *def1 def2 assms*(*2*) *F0* *is-partitionE*(*1*)[*of Bs B*] **apply** *blast*
using *def1* **apply** *blast* **using** *def2* **apply** *blast* **by**(*rule F0*)
thus *?thesis* **unfolding** *0* **by** *blast*
qed
next
show $\bigcup$ (*pairwise-intersect As Bs*) $= A \cap B$
proof(*rule equalityI'*)
fix *x* **assume** *A*: $x \in \bigcup$ (*pairwise-intersect As Bs*)
then obtain *a b* **where** *def1*: $a \in As \wedge b \in Bs \wedge x \in a \cap b$
unfolding *pairwise-intersect-def* **by** *blast*
have *0*: $a \subseteq A$
using *def1 assms is-partitionE* **by** *blast*
have *1*: $b \subseteq B$
using *def1 assms is-partitionE* **by** *blast*
show $x \in A \cap B$
using *0 1 def1* **by** *blast*
next
fix *x* **assume** *A*: $x \in A \cap B$
obtain *a* **where** *a-def*: $a \in As \wedge x \in a$
using *A assms is-partitionE* **by** *blast*
obtain *b* **where** *b-def*: $b \in Bs \wedge x \in b$
using *A assms is-partitionE* **by** *blast*
have *0*: $x \in a \cap b$

```
    using a-def b-def by blast
  show x ∈ ⋃ (pairwise-intersect As Bs)
    using a-def b-def 0 unfolding pairwise-intersect-def
    by blast
 qed
qed

lemma pairwise-intersect-finite:
 assumes finite As
 assumes finite Bs
 shows finite (pairwise-intersect As Bs)
proof−
 have 0: (pairwise-intersect As Bs) = (⋃ a ∈ As. (∩) a ‘ Bs)
  unfolding pairwise-intersect-def
  apply(rule equalityI′)
  unfolding mem-Collect-eq apply blast
  by blast
 have 1: ⋀a. a ∈ As ⟹ finite ((∩) a ‘ Bs)
  using assms by blast
 show ?thesis unfolding 0 using assms(1) 1 by blast
qed

definition family-intersect where
family-intersect parts = atoms-of (⋃ parts)

lemma family-intersect-partitions:
 assumes ⋀Ps. Ps ∈ parts ⟹ Ps partitions A
 assumes ⋀Ps. Ps ∈ parts ⟹ finite Ps
 assumes finite parts
 assumes parts ≠ {}
 shows family-intersect parts partitions A
proof(rule is-partitionI)
 show disjoint (family-intersect parts)
  apply(rule disjointI)
  unfolding family-intersect-def apply(rule atoms-of-disjoint)
  apply blast
  apply blast
  by blast
 show ⋃ (family-intersect parts) = A
 proof−
  have 0: ⋃ (family-intersect parts) = ⋃ (⋃ parts)
   unfolding family-intersect-def
   apply(rule atoms-of-covers)
   by blast
  have 1: ⋀Ps. Ps ∈ parts ⟹ ⋃Ps = A
   by(rule is-partitionE, rule assms, blast)
  show ?thesis unfolding 0
   using 1 assms by blast
 qed
```

qed

lemma *family-intersect-memE*:
assumes $\bigwedge Ps.\ Ps \in parts \Longrightarrow Ps\ partitions\ A$
assumes $\bigwedge Ps.\ Ps \in parts \Longrightarrow finite\ Ps$
assumes *finite parts*
assumes $parts \neq \{\}$
shows $\bigwedge Ps\ a.\ a \in family\text{-}intersect\ parts \Longrightarrow Ps \in parts \Longrightarrow \exists P \in Ps.\ a \subseteq P$
proof−
fix *Ps a* **assume** *A*: $a \in family\text{-}intersect\ parts\ Ps \in parts$
have *0*: $\bigcup\ Ps = A$
apply(*rule is-partitionE*)
using *A assms* **by** *blast*
have *1*: $\bigcup\ (family\text{-}intersect\ parts) = A$
apply(*rule is-partitionE*)
using *family-intersect-partitions assms* **by** *blast*
have *2*: $a \neq \{\}$
using *A* **unfolding** *family-intersect-def atoms-of-def* **by** *blast*
obtain *P* **where** *P-def*: $P \in Ps \wedge a \cap P \neq \{\}$
using *0 1 A 2* **by** *blast*
have *P-in*: $P \in (\bigcup\ parts)$
using *P-def A* **by** *blast*
have *a-sub*: $a \subseteq P$
using *atoms-are-minimal P-def A P-in* **unfolding** *family-intersect-def* **by** *blast*

show $\exists P \in Ps.\ a \subseteq P$
using *a-sub P-def* **by** *blast*
qed

lemma *family-intersect-mem-inter*:
assumes $\bigwedge Ps.\ Ps \in (parts{::}\ 'a\ set\ set\ set) \Longrightarrow Ps\ partitions\ A$
assumes $\bigwedge Ps.\ Ps \in parts \Longrightarrow finite\ Ps$
assumes *finite parts*
assumes $parts \neq \{\}$
assumes $a \in family\text{-}intersect\ parts$
shows $\exists f.\ \forall\ Ps \in parts.\ f\ Ps \in Ps \wedge a = (\bigcap\ Ps \in parts.\ f\ Ps)$
proof−
obtain *f* **where** *f-def*: $f = (\lambda Ps{::}\ 'a\ set\ set.\ (SOME\ P.\ P \in Ps \wedge a \subseteq P))$
by *blast*
have *f-eval*: $\bigwedge Ps.\ Ps \in parts \Longrightarrow f\ Ps \in Ps \wedge a \subseteq (f\ Ps)$
proof−
fix *Ps* **assume** *A*: $Ps \in parts$
obtain *P* **where** *P-def*: $P \in Ps \wedge a \subseteq P$
using *assms family-intersect-memE A* **by** *blast*
show $f\ Ps \in Ps \wedge a \subseteq f\ Ps$
apply(*rule SomeE*[*of f Ps - P*])
unfolding *f-def* **using** *A* **apply** *simp*
by(*rule P-def*)
qed

```
  have 0: a ≠ {}
    using assms unfolding family-intersect-def
    using atoms-nonempty by blast
  have 1: a = (⋂ Ps ∈ parts. f Ps)
  proof(rule equalityI)
    show 10: a ⊆ ⋂ (f ‘ parts)
      using f-eval by blast
    show ⋂ (f ‘ parts) ⊆ a
    proof
      fix x assume A: x ∈ ⋂ (f ‘ parts)
      obtain b where b-def: b = point-to-atom (⋃ parts) x
        by blast
      have b-atom: b ∈ atoms-of (⋃ parts)
        unfolding b-def apply(rule point-to-atom-closed)
        using A f-eval assms by blast
      show x-in-a: x ∈ a
      proof(rule ccontr)
        assume x ∉ a
        then have ¬ b ⊆ a
       using b-def unfolding point-to-atom-def point-to-type-def subset-to-atom-def
by blast
        hence p0: a ≠ b
          by blast
        have p1: b ∩ a = {}
          apply(rule atoms-of-disjoint[of - (⋃ parts)] )
            apply(rule b-atom)
          using assms unfolding family-intersect-def apply blast
          using p0 by blast
        have p2:  (∃ B∈⋃ parts. b ⊆ B ∧ a ∩ B = {}) ∨ (∃ A∈⋃ parts. a ⊆ A ∧
b ∩ A = {})
          using distinct-atoms[of ⋃ parts a b] assms
            by (metis Sup-bot-conv(1) b-atom equalityI' f-eval family-intersect-def
mem-simps(2) p0)
        show False
        proof(cases (∃ B∈⋃ parts. b ⊆ B ∧ a ∩ B = {}))
          case True
          then obtain B where B-def: B∈⋃ parts ∧ b ⊆ B ∧ a ∩ B = {}
            by blast
          obtain Ps where Ps-def: B ∈ Ps ∧ Ps ∈ parts
            using B-def by blast
          have B-neq: B ≠ f Ps
            using Ps-def B-def 10 0 by blast
          have B-cap: B ∩ f Ps = {}
            apply(rule disjointE[of Ps])
               apply(rule is-partitionE[of Ps A])
            using Ps-def assms apply blast
            using Ps-def apply blast
            using f-eval Ps-def apply blast
            by(rule B-neq)
```

```
      have b-cap: b ∩ f Ps = {}
        using B-cap B-def by blast
      have x-in-b: x ∈ b
     using b-def unfolding point-to-atom-def point-to-type-def subset-to-atom-def

        by blast
      show False using x-in-b b-cap Ps-def A by blast
    next
      case False
      then obtain B where B-def: B∈⋃ parts ∧ a ⊆ B ∧ b ∩ B = {}
        using p2 by blast
      obtain Ps where Ps-def: B ∈ Ps ∧ Ps ∈ parts
        using B-def by blast
      have F0: B = f Ps
      proof(rule ccontr)
        assume not: B ≠ f Ps
        have F0: B ∩ f Ps = {}
         apply(rule disjointE[of Ps])
           apply(rule is-partitionE[of Ps A])
        using Ps-def assms apply blast
        using Ps-def apply blast
        using f-eval Ps-def apply blast
        by(rule not)
        have a-sub: a ⊆ f Ps
          using 10 Ps-def by blast
        show False using F0 B-def a-sub 0  by blast
      qed
      have x-in-B: x ∈ B
        unfolding F0 using A Ps-def by blast
      have x-in-b: x ∈ b
     using b-def unfolding point-to-atom-def point-to-type-def subset-to-atom-def

        by blast
      show False using x-in-b x-in-B B-def by blast
    qed
  qed
 qed
 show ?thesis using f-eval 1 by blast
qed
```

If we take a finite family of partitions in a particular generated boolean algebra, where each partition itself is finite, then their induced partition is also in the algebra.

```
lemma family-intersect-in-gen-boolean-algebra:
 assumes A ∈ gen-boolean-algebra S B
 assumes ⋀Ps. Ps ∈ parts ⟹ Ps partitions A
 assumes ⋀Ps. Ps ∈ parts ⟹ finite Ps
 assumes ⋀Ps P. Ps ∈ parts ⟹ P ∈ Ps ⟹  P ∈ gen-boolean-algebra S B
```

assumes *finite parts*
assumes $parts \neq \{\}$
shows $\bigwedge P.\ P \in$ *family-intersect parts* $\Longrightarrow P \in$ *gen-boolean-algebra S B*
proof−
fix P **assume** *A*: $P \in$ *family-intersect parts*
have *0*: $P \in$ *atoms-of* $(\bigcup$ *parts*$)$
using *A* **unfolding** *family-intersect-def* **by** *blast*
have *1*: *finite* $(\bigcup$ *parts*$)$
using *assms* **by** *blast*
have *2*: $\bigcup$ *parts* $\subseteq$ *gen-boolean-algebra S B*
using *assms* **by** *blast*
obtain *Ps* **where** *Ps-def*: $Ps \in$ *parts*
using *assms* **by** *blast*
have *3*: $\bigcup$ $(\bigcup$ *parts*$) =$ *A*
apply(*rule equalityI'*)
using *assms is-partitionE*(*2*)[*of* - *A*] **apply** *blast*
using *assms is-partitionE*(*2*)[*of Ps A*] *Ps-def* **by** *blast*
have *4*: *atoms-of* $(\bigcup$ *parts*$)$ = *atoms-of* (*gen-boolean-algebra A* $(\bigcup$ *parts*$)$)
apply(*rule atoms-of-sets-eq-atoms-of-algebra*[*of* $\bigcup$ *parts A*])
apply(*rule 1*)
unfolding *3* **by** *blast*
have *5*: *atoms-of* $(\bigcup$ *parts*$) \subseteq$ (*gen-boolean-algebra A* $(\bigcup$ *parts*$)$)
apply(*rule atoms-of-gen-boolean-algebra*)
using *3 gen-boolean-algebra.generator*[*of* - $\bigcup$ *parts A*]
apply (*meson Sup-upper gen-boolean-algebra-generators subsetI*)
by(*rule 1*)
have *6*: $A \subseteq S$
using *assms gen-boolean-algebra-subset* **by** *blast*
have *7*: (*gen-boolean-algebra A* $(\bigcup$ *parts*$)) \subseteq$ *gen-boolean-algebra* (S) $(\bigcup$ *parts*$)$
apply(*rule gen-boolean-algebra-univ-mono*)
using *3 gen-boolean-algebra-finite-union*[*of* $\bigcup$ *parts S* $\bigcup$ *parts*]
gen-boolean-algebra.generator[*of* - $\bigcup$ *parts S*] *6 1*
by (*meson Sup-le-iff gen-boolean-algebra-generators*)
have *8*: *gen-boolean-algebra* (S) $(\bigcup$ *parts*$) \subseteq$ *gen-boolean-algebra S B*
apply(*rule gen-boolean-algebra-subalgebra*)
using *2* **by** *blast*
show $P \in$ *gen-boolean-algebra S B*
using *0 5 6 7 8* **by** *blast*
qed

end
theory *Padic-Field-Powers*
imports *Ring-Powers Padic-Field-Polynomials Generated-Boolean-Algebra*
Padic-Field-Topology

begin

This theory is intended to develop the necessary background on subsets of powers of a p-adic field to prove Macintyre's quantifier elimination theorem. In particular, we define semi-algebraic subsets of $\mathbb{Q}_p^n$, semi-algebraic functions $\mathbb{Q}_p^n \to \mathbb{Q}_p$, and semi- algebraic mappings $\mathbb{Q}_p^n \to \mathbb{Q}_p^m$ for arbitrary $n, m \in \mathbb{N}$. In addition we prove that many common sets and functions are semi-algebraic. We are closely following the paper [1] by Denef, where an algebraic proof of Mactinyre's theorem is developed.

13 Cartesian Powers of p-adic Fields

lemma *list-tl*:
tl $(t\#x) = x$
using *List.list.sel(3)* **by** *auto*

lemma *list-hd*:
hd $(t\#x) = t$
unfolding *List.list.sel(1)* **by** *auto*

sublocale *padic-fields* < *cring-coord-rings* Q_p *UP* Q_p
unfolding *cring-coord-rings-axioms-def cring-coord-rings-def*
using *Qp.zero-not-one UPQ.R-cring*
apply (*simp add*: *UPQ.is-UP-cring*)
by *auto*

sublocale *padic-fields* < *Qp*: *domain-coord-rings* Q_p *UP* Q_p
unfolding *domain-coord-rings-def cring-coord-rings-axioms-def cring-coord-rings-def*
using *Qp.domain-axioms Qp.zero-not-one UPQ.R-cring*
apply (*simp add*: *UPQ.UP-cring-axioms*)
by *auto*

context *padic-fields*
begin
no-notation *Zp.to-fun* (**infixl**‹•› *70*)
no-notation *ideal-prod* (**infixl** ‹·₁› *80*)

notation
evimage (**infixr** ‹$^{-1}$₁› *90*) **and**
euminus-set (‹- ^{c}₁› *70*)

type-synonym *padic-tuple* = *padic-number list*
type-synonym *padic-function* = *padic-number* ⇒ *padic-number*
type-synonym *padic-nary-function* = *padic-tuple* ⇒ *padic-number*
type-synonym *padic-function-tuple* = *padic-nary-function list*
type-synonym *padic-nary-function-poly* = *nat* ⇒ *padic-nary-function*

13.1 Polynomials over $\mathbb{Q}_p$ and Polynomial Maps

lemma *last-closed′*:
assumes $x@[t] \in$ *carrier* $({Q_p}^n)$
shows $t \in$ *carrier* Q_p
using *assms last-closed*[*of n* $x@[t]$ Q_p]
by (*metis* (*full-types*) *cartesian-power-car-memE gr0I last-snoc length-append-singleton less-not-refl zero-less-Suc*)

lemma *segment-in-car′*:
assumes $x@[t] \in$ *carrier* $({Q_p}^{Suc\ n})$
shows $x \in$ *carrier* $({Q_p}^n)$
proof−
have *0*: *length* $x = n$
by (*metis Suc-inject assms cartesian-power-car-memE length-append-singleton*)
have *set* $x \subseteq$ *set* $(x@[t])$
by (*metis rotate1.simps*(*2*) *set-rotate1 set-subset-Cons*)
then have *1*: *set* $x \subseteq$ *carrier* Q_p
using *assms cartesian-power-car-memE′′*[*of* $x@[t]$ Q_p *Suc n*]
by *blast*
show *?thesis*
using *0 1 assms cartesian-power-car-memI*[*of x n* Q_p]
by *blast*
qed

lemma *Qp-zero*:
${Q_p}^0 =$ *nil-ring*
unfolding *cartesian-power-def*
by *simp*

lemma *Qp-zero-carrier*:
carrier $({Q_p}^0) = \{[]\}$
by (*simp add*: *Qp-zero*)

Abbreviation for constant polynomials

abbreviation(*input*) *Qp-to-IP* **where**
Qp-to-IP k ≡ *Qp.indexed-const k*

lemma *Qp-to-IP-car*:
assumes $k \in$ *carrier* Q_p
shows *Qp-to-IP k* ∈ *carrier* $(Q_p[\mathcal{X}_n])$
using *assms*
unfolding *coord-ring-def*
using *Qp.indexed-const-closed* **by** *blast*

lemma(**in** *cring-coord-rings*) *smult-closed*:
assumes $a \in$ *carrier* R
assumes $q \in$ *carrier* $(R[\mathcal{X}_n])$
shows $a \odot_{R[\mathcal{X}_n]} q \in$ *carrier* $(R[\mathcal{X}_n])$
using *assms* **unfolding** *coord-ring-def*

using *Pring-smult-closed*
by (*simp add*: *R.Pring-smult-closed*)

lemma *Qp-poly-smult-cfs*:
assumes $a \in$ *carrier* Q_p
assumes $P \in$ *carrier* $(Q_p[\mathcal{X}_n])$
shows $(a \odot_{Q_p[\mathcal{X}_n]} P)\ m = a \otimes (P\ m)$
using *assms* **unfolding** *coord-ring-def*
using *Qp.Pring-smult-cfs* **by** *blast*

lemma *Qp-smult-r-distr*:
assumes $a \in$ *carrier* Q_p
assumes $P \in$ *carrier* $(Q_p[\mathcal{X}_n])$
assumes $q \in$ *carrier* $(Q_p[\mathcal{X}_n])$
shows $a \odot_{Q_p[\mathcal{X}_n]} (P \oplus_{Q_p[\mathcal{X}_n]} q) = (a \odot_{Q_p[\mathcal{X}_n]} P) \oplus_{Q_p[\mathcal{X}_n]} (a \odot_{Q_p[\mathcal{X}_n]} q)$
using *assms* **unfolding** *coord-ring-def*
using *Qp.Pring-smult-r-distr* **by** *blast*

lemma *Qp-smult-l-distr*:
assumes $a \in$ *carrier* Q_p
assumes $b \in$ *carrier* Q_p
assumes $P \in$ *carrier* $(Q_p[\mathcal{X}_n])$
shows $(a \oplus b) \odot_{Q_p[\mathcal{X}_n]} P = (a \odot_{Q_p[\mathcal{X}_n]} P) \oplus_{Q_p[\mathcal{X}_n]} (b \odot_{Q_p[\mathcal{X}_n]} P)$
using *assms* **unfolding** *coord-ring-def*
using *Qp.Pring-smult-l-distr* **by** *blast*

abbreviation(*input*) *Qp-funs* **where**
Qp-funs n ≡ *Fun*$_n$ Q_p

13.2 Evaluation of Polynomials in $\mathbb{Q}_p$

abbreviation(*input*) *Qp-ev* **where**
Qp-ev P q ≡ (*eval-at-point* Q_p *q P*)

lemma *Qp-ev-one*:
assumes $a \in$ *carrier* $({Q_p}^n)$
shows *Qp-ev* $\mathbf{1}_{Q_p[\mathcal{X}_n]}$ $a = \mathbf{1}$ **unfolding** *coord-ring-def*
by (*metis Qp.Pring-one eval-at-point-const Qp.one-closed assms*)

lemma *Qp-ev-zero*:
assumes $a \in$ *carrier* $({Q_p}^n)$
shows *Qp-ev* $\mathbf{0}_{Q_p[\mathcal{X}_n]}$ $a = \mathbf{0}$**unfolding** *coord-ring-def*
by (*metis Qp.Pring-zero eval-at-point-const Qp.zero-closed assms*)

lemma *Qp-eval-pvar-pow*:
assumes $a \in$ *carrier* $({Q_p}^n)$
assumes $k < n$
assumes $(m::nat) \neq 0$

shows *Qp-ev* ((*pvar* Q_p *k*)[^]$_{Q_p[\mathcal{X}_n]}$ *m*) *a* = ((*a*!*k*)[^]*m*)
by (*metis eval-at-point-nat-pow eval-pvar assms*(*1*) *assms*(*2*) *pvar-closed*)

composition of polynomials over $\mathbb{Q}_p$

definition *Qp-poly-comp* **where**
Qp-poly-comp m fs = *poly-compose* (*length fs*) *m fs*

lemmas about polynomial maps

lemma *Qp-is-poly-tupleI*:
assumes ⋀*i*. *i* < *length fs*⟹ *fs*!*i* ∈ *carrier* ($Q_p[\mathcal{X}_m]$)
shows *is-poly-tuple m fs*
unfolding *is-poly-tuple-def* **using** *assms*
using *cartesian-power-car-memE''* *cartesian-power-car-memI'* **by** *blast*

lemma *Qp-is-poly-tuple-append*:
assumes *is-poly-tuple m fs*
assumes *is-poly-tuple m gs*
shows *is-poly-tuple m* (*fs*@*gs*)
proof(*rule Qp-is-poly-tupleI*)
show ⋀*i*. *i* < *length* (*fs* @ *gs*) ⟹ (*fs* @ *gs*) ! *i* ∈ *carrier* ($Q_p[\mathcal{X}_m]$)
proof− **fix** *i* **assume** *A*: *i* < *length* (*fs* @ *gs*)
show (*fs* @ *gs*) ! *i* ∈ *carrier* ($Q_p[\mathcal{X}_m]$)
apply(*cases i* < *length fs*)
using *assms is-poly-tupleE*[*of m fs i*] *nth-append*[*of fs gs i*]
apply *presburger*
proof−
assume *B*: ¬ *i* < *length fs*
then have *C*: *length fs* ≤ *i* ∧ *i* < *length* (*fs* @ *gs*)
using *A not-le*
by *blast*
then have *i* − *length fs* < *length gs*
using *length-append*[*of fs gs*]
by *linarith*
then show *?thesis*
using *A assms is-poly-tupleE*[*of m gs i* − *length fs*] *nth-append*[*of fs gs i*] *B*
by *presburger*
qed
qed
qed

lemma *Qp-poly-mapE*:
assumes *is-poly-tuple n fs*
assumes *length fs* = *m*
assumes *as* ∈ *carrier* ($Q_p{}^n$)
assumes *j* < *m*
shows (*poly-map n fs as*)!*j* ∈ *carrier* Q_p
using *assms poly-map-closed cartesian-power-car-memE'* **by** *blast*

lemma *Qp-poly-mapE'*:

assumes *as* ∈ *carrier* (${Q_p}^n$)
shows *length* (*poly-map n fs as*) = *length fs*
unfolding *poly-map-def*
using *Qp.cring-axioms poly-tuple-evalE′*
by (*metis assms restrict-def*)

lemma *Qp-poly-mapE′′*:
assumes *is-poly-tuple n fs*
assumes *length fs* = *m*
assumes *n* ≠ *0*
assumes *as* ∈ *carrier* (${Q_p}^n$)
assumes *j* < *m*
shows (*poly-map n fs as*)!*j* = (*Qp-ev* (*fs*!*j*) *as*)
using *assms*
unfolding *poly-map-def poly-tuple-eval-def*
by (*metis* (*no-types*, *lifting*) *nth-map restrict-apply′*)

lemma *poly-map-apply*:
assumes *as* ∈ *carrier* (${Q_p}^n$)
shows *poly-map n fs as* = *poly-tuple-eval fs as*
unfolding *poly-map-def restrict-def*
by (*simp add*: *assms*)

lemma *poly-map-pullbackI*:
assumes *is-poly-tuple n fs*
assumes *as* ∈ *carrier* (${Q_p}^n$)
assumes *poly-map n fs as* ∈ *S*
shows *as* ∈ *poly-map n fs* $^{-1}{}_n$ *S*
using *assms poly-map-apply*
by *blast*

lemma *poly-map-pullbackI′*:
assumes *is-poly-tuple n fs*
assumes *as* ∈ *carrier* (${Q_p}^n$)
assumes *poly-map n fs as* ∈ *S*
shows *as* ∈ ((*poly-map n fs*) −‘ *S*)
by (*simp add*: *assms*(*3*))

lemmas about polynomial composition

lemma *poly-compose-ring-hom*:
assumes *is-poly-tuple m fs*
assumes *length fs* = *n*
shows (*ring-hom-ring* (${Q_p}[\mathcal{X}_n]$) (${Q_p}[\mathcal{X}_m]$) (*Qp-poly-comp m fs*))
unfolding *Qp-poly-comp-def*
by (*simp add*: *assms*(*1*) *assms*(*2*) *poly-compose-ring-hom*)

lemma *poly-compose-closed*:
assumes *is-poly-tuple m fs*
assumes *length fs* = *n*

assumes $f \in$ *carrier* $(Q_p[\mathcal{X}_n])$
shows $(Qp\text{-}poly\text{-}comp\ m\ fs\ f) \in$ *carrier* $(Q_p[\mathcal{X}_m])$
using *Qp.cring-axioms assms*
unfolding *Qp-poly-comp-def*
using *poly-compose-closed* **by** *blast*

lemma *poly-compose-add*:
assumes *is-poly-tuple m fs*
assumes *length fs* $= n$
assumes $f \in$ *carrier* $(Q_p[\mathcal{X}_n])$
assumes $g \in$ *carrier* $(Q_p[\mathcal{X}_n])$
shows $Qp\text{-}poly\text{-}comp\ m\ fs\ (f \oplus_{Q_p[\mathcal{X}_n]} g) = (Qp\text{-}poly\text{-}comp\ m\ fs\ f) \oplus_{Q_p[\mathcal{X}_m]}$
$(Qp\text{-}poly\text{-}comp\ m\ fs\ g)$
using *Qp.cring-axioms assms poly-compose-add*
unfolding *is-poly-tuple-def Qp-poly-comp-def*
by *blast*

lemma *poly-compose-mult*:
assumes *is-poly-tuple m fs*
assumes *length fs* $= n$
assumes $f \in$ *carrier* $(Q_p[\mathcal{X}_n])$
assumes $g \in$ *carrier* $(Q_p[\mathcal{X}_n])$
shows $Qp\text{-}poly\text{-}comp\ m\ fs\ (f \otimes_{Q_p[\mathcal{X}_n]} g) = (Qp\text{-}poly\text{-}comp\ m\ fs\ f) \otimes_{Q_p[\mathcal{X}_m]}$
$(Qp\text{-}poly\text{-}comp\ m\ fs\ g)$
using *Qp.cring-axioms assms poly-compose-mult*
unfolding *is-poly-tuple-def Qp-poly-comp-def*
by *blast*

lemma *poly-compose-const*:
assumes *is-poly-tuple m fs*
assumes *length fs* $= n$
assumes $a \in$ *carrier* Q_p
shows $Qp\text{-}poly\text{-}comp\ m\ fs\ (Qp\text{-}to\text{-}IP\ a) = Qp\text{-}to\text{-}IP\ a$
using *Qp.cring-axioms assms poly-compose-const*
unfolding *is-poly-tuple-def Qp-poly-comp-def*
by *metis*

lemma *Qp-poly-comp-eval*:
assumes *is-poly-tuple m fs*
assumes *length fs* $= n$
assumes $f \in$ *carrier* $(Q_p[\mathcal{X}_n])$
assumes $as \in$ *carrier* $({Q_p}^m)$
shows $Qp\text{-}ev\ (Qp\text{-}poly\text{-}comp\ m\ fs\ f)\ as = Qp\text{-}ev\ f\ (poly\text{-}map\ m\ fs\ as)$
proof−
have $(restrict\ (poly\text{-}tuple\text{-}eval\ fs)\ (carrier\ ({Q_p}^m))\ as) = poly\text{-}tuple\text{-}eval\ fs\ as$
unfolding *restrict-def*
by (*simp add*: *assms*)
thus *?thesis*

using *assms Qp.cring-axioms poly-compose-eval*
unfolding *Qp-poly-comp-def poly-map-def*
by *presburger*
qed

13.3 Mapping Univariate Polynomials to Multivariable Polynomials in One Variable

abbreviation(*input*) *to-Qp-x* **where**
to-Qp-x ≡ (*IP-to-UP* (*0*::*nat*) :: (*nat multiset* ⇒ *padic-number*) ⇒ *nat* ⇒ *padic-number*)

abbreviation(*input*) *from-Qp-x* **where**
from-Qp-x ≡ *UP-to-IP* Q_p (*0*::*nat*)

lemma *from-Qp-x-closed*:
assumes *q* ∈ *carrier* Q_p*-x*
shows *from-Qp-x q* ∈ *carrier* ($Q_p[\mathcal{X}_1]$)
using *assms UP-to-IP-closed* **unfolding** *coord-ring-def*
by (*metis One-nat-def lessThan-0 lessThan-Suc*)

lemma *to-Qp-x-closed*:
assumes *q* ∈ *carrier* ($Q_p[\mathcal{X}_1]$)
shows *to-Qp-x q* ∈ *carrier* Q_p*-x*
using *assms Qp.IP-to-UP-closed*[*of q 0*::*nat*] *Qp.cring-axioms*
unfolding *coord-ring-def*
by (*metis One-nat-def lessThan-0 lessThan-Suc*)

lemma *to-Qp-x-from-Qp-x*:
assumes *q* ∈ *carrier* ($Q_p[\mathcal{X}_1]$)
shows *from-Qp-x* (*to-Qp-x q*) = *q*
using *assms UP-to-IP-inv*[*of q 0*::*nat*] *Qp.Pring-car*
unfolding *coord-ring-def*
by (*metis One-nat-def lessThan-0 lessThan-Suc*)

lemma *from-Qp-x-to-Qp-x*:
assumes *q* ∈ *carrier* Q_p*-x*
shows *to-Qp-x* (*from-Qp-x q*) = *q*
by (*meson UPQ.IP-to-UP-inv assms*)

ring hom properties of these maps

lemma *to-Qp-x-ring-hom*:
to-Qp-x ∈ *ring-hom* ($Q_p[\mathcal{X}_1]$) Q_p*-x*
using *IP-to-UP-ring-hom*[*of 0*::*nat*] *ring-hom-ring.homh*
unfolding *coord-ring-def*
by (*metis One-nat-def lessThan-0 lessThan-Suc*)

lemma *from-Qp-x-ring-hom*:
from-Qp-x ∈ *ring-hom* Q_p*-x* ($Q_p[\mathcal{X}_1]$)
using *UP-to-IP-ring-hom ring-hom-ring.homh*

unfolding *coord-ring-def*
by (*metis One-nat-def lessThan-0 lessThan-Suc*)

lemma *from-Qp-x-add*:
assumes $a \in$ *carrier* Q_p*-x*
assumes $b \in$ *carrier* Q_p*-x*
shows *from-Qp-x* ($a \oplus_{Q_p\text{-}x} b$) = *from-Qp-x a* $\oplus_{Q_p[\mathcal{X}_1]}$ *from-Qp-x b*
by (*metis* (*mono-tags*, *lifting*) *assms*(*1*) *assms*(*2*) *from-Qp-x-ring-hom ring-hom-add*)

lemma *from-Qp-x-mult*:
assumes $a \in$ *carrier* Q_p*-x*
assumes $b \in$ *carrier* Q_p*-x*
shows *from-Qp-x* ($a \otimes_{Q_p\text{-}x} b$) = *from-Qp-x a* $\otimes_{Q_p[\mathcal{X}_1]}$ *from-Qp-x b*
by (*metis assms*(*1*) *assms*(*2*) *from-Qp-x-ring-hom ring-hom-mult*)

equivalence of evaluation maps

lemma *Qp-poly-Qp-x-eval*:
assumes $P \in$ *carrier* ($Q_p[\mathcal{X}_1]$)
assumes $a \in$ *carrier* ($Q_p{}^1$)
shows *Qp-ev P a* = (*to-Qp-x P*)$\cdot$(*Qp.to-R a*)
proof−
have *0*: (*IP-to-UP 0 P*) $\cdot$ (*a* ! *0*) = ((*IP-to-UP 0 P*) $\cdot$ (*if 0* < *length a then a* ! *0 else* **0**))
using *assms*
by (*metis* (*mono-tags*, *lifting*) *cartesian-power-car-memE gr0I zero-neq-one*)
have *1*: *closed-fun* Q_p (λn. *if n* < *length a then a* ! *n else* **0**)
proof
fix *n*
show (*if n* < *length a then a* ! *n else* **0**) ∈ *carrier* Q_p
apply(*cases n* < *length a*)
using *assms*
apply (*metis cartesian-power-car-memE cartesian-power-car-memE'*)
by (*meson Qp.cring-axioms cring.cring-simprules*(*2*))
qed
have *2*: *P* ∈ *Pring-set* Q_p {*0*::*nat*}
using *assms* **unfolding** *coord-ring-def*
by (*metis Qp.Pring-car UPQ.UP-to-IP-closed assms*(*1*) *to-Qp-x-closed to-Qp-x-from-Qp-x*)
have *3*: *total-eval* Q_p (λi. *if i* < *length a then a* ! *i else* **0**) *P* = *IP-to-UP 0 P* $\cdot$ (*if 0* < *length a then a* ! *0 else* **0**)
using *1 2 assms IP-to-UP-poly-eval*[*of P 0*::*nat* (λi. *if i* < *length a then a* ! *i else* **0**)]
UPQ.to-fun-def **by** *presburger*
then show *?thesis*
using *0*
unfolding *eval-at-point-def*
by *blast*
qed

lemma *Qp-x-Qp-poly-eval*:
assumes $P \in$ *carrier* Q_p*-x*
assumes $a \in$ *carrier* Q_p
shows $P \cdot a =$ *Qp-ev* (*from-Qp-x P*) (*to-R1 a*)
proof−
have *Qp-ev* (*from-Qp-x P*) (*to-R1 a*) = (*to-Qp-x* (*from-Qp-x P*))·(*Qp.to-R* (*Qp.to-R1 a*))
using *Qp-poly-Qp-x-eval assms*(*1*) *assms*(*2*) *from-Qp-x-closed Qp.to-R1-closed*
by *blast*
then show *?thesis* **using** *assms*
by (*metis UPQ.IP-to-UP-inv Qp.to-R-to-R1*)
qed

13.4 n^{th}-Power Sets over $\mathbb{Q}_p$

definition *P-set* **where**
P-set (*n*::*nat*) = {$a \in$ *nonzero* Q_p. (∃ $y \in$ *carrier* Q_p . ($y[\hat{}]$ *n*) = *a*)}

lemma *P-set-carrier*:
P-set n ⊆ *carrier* Q_p
unfolding *P-set-def nonzero-def*
by *blast*

lemma *P-set-memI*:
assumes $a \in$ *carrier* Q_p
assumes $a \neq \mathbf{0}$
assumes $b \in$ *carrier* Q_p
assumes $b[\hat{}](n::nat) = a$
shows $a \in$ *P-set n*
unfolding *P-set-def*
using *assms*
by (*metis* (*mono-tags*, *lifting*) *mem-Collect-eq not-nonzero-Qp*)

lemma *P-set-nonzero*:
P-set n ⊆ *nonzero* Q_p
unfolding *P-set-def* **by** *blast*

lemma *P-set-nonzero′*:
assumes $a \in$ *P-set n*
shows $a \in$ *nonzero* Q_p
$a \in$ *carrier* Q_p
using *assms P-set-nonzero P-set-carrier*
apply *blast* **using** *assms P-set-carrier* **by** *blast*

lemma *P-set-one*:
assumes $n \neq 0$
shows **1** $\in$ *P-set* (*n*::*nat*)
proof−

```
  have 0: 1[^]n = 1
    using Qp.nat-pow-one by blast
  have 1: 1 ∈ carrier Q_p
    by blast
  then show ?thesis
    using one-nonzero unfolding P-set-def
    using 0 by blast
qed

lemma zeroth-P-set:
P-set 0 = {1}
proof
  show P-set 0 ⊆ {1}
  unfolding P-set-def
  proof
    fix x
    assume x ∈ {a ∈ nonzero Q_p. ∃ y∈carrier Q_p. (y[^](0::nat)) = a}
    then have ∃ y∈carrier Q_p. (y[^](0::nat)) = x
      by blast
    then obtain a where a-def: a ∈ carrier Q_p ∧ (a[^](0::nat)) = x
      by blast
    then show x ∈ {1}
      using Qp.nat-pow-0 by blast
  qed
  show {1} ⊆ P-set 0
    using P-set-memI[of 1 1 0] Qp.nat-pow-one Qp.one-closed local.one-neq-zero
    by blast
qed

lemma P-set-mult-closed:
  assumes n ≠ 0
  assumes a ∈ P-set n
  assumes b ∈ P-set n
  shows a ⊗ b ∈ P-set n
proof−
  obtain a0 where a0-def: a0 ∈ carrier Q_p ∧ (a0 [^] n = a)
    using assms(2)
    unfolding P-set-def
    by blast
  obtain b0 where b0-def: b0 ∈ carrier Q_p ∧ (b0 [^] n = b)
    using assms(3)
    unfolding P-set-def
    by blast
  have (a0 ⊗ b0) [^] n = a0 [^] n ⊗ b0 [^] n
    using a0-def b0-def Qp.nat-pow-distrib by blast
  then have 0: a ⊗ b  = (a0 ⊗ b0) [^] n
    using a0-def b0-def by blast
  have 1: a0 ⊗ b0 ∈ carrier Q_p
    by (meson Qp.cring-axioms a0-def b0-def cring.cring-simprules(5))
```

have *2*: $a \otimes b \in$ *nonzero* Q_p
using *assms nonzero-is-submonoid* **unfolding** *P-set-def*
by (*metis* (*no-types*, *lifting*) *0 1 Qp.integral Qp-nat-pow-nonzero a0-def b0-def mem-Collect-eq not-nonzero-Qp*)
then show *?thesis*
using *0 1 assms*
unfolding *P-set-def* **by** *blast*
qed

lemma *P-set-inv-closed*:
assumes $a \in$ *P-set n*
shows *inv* $a \in$ *P-set n*
proof(*cases* $n = 0$)
case *True*
then show *?thesis*
using *assms zeroth-P-set*
by (*metis Qp.inv-one singletonD*)
next
case *False*
then show *?thesis* **proof**−
obtain *a0* **where** *a0-def*: $a0 \in$ *carrier* $Q_p \wedge a0[\hat{}]n = a$
using *assms P-set-def*[*of n*] **by** *blast*
have $a0 \in$ *nonzero* Q_p
apply(*rule ccontr*)
proof−
assume $a0 \notin$ *nonzero* Q_p
then have $a0 = \mathbf{0}$
using *a0-def*
by (*meson not-nonzero-Qp*)
then show *False* **using** *a0-def assms*
by (*metis* (*mono-tags*, *lifting*) *False P-set-def Qp.cring-axioms* ‹*a0* $\notin$ *nonzero* Q_p›
cring-def mem-Collect-eq neq0-conv ring.pow-zero)
qed
then have (*inv a0*)[$\hat{}$]n = *inv a*
using *a0-def* ‹*a0* $\in$ *carrier* $Q_p \wedge$ ($a0[\hat{}]n$) = *a*› ‹*a0* $\in$ *nonzero* Q_p› *Units-nonzero*
monoid.nat-pow-of-inv[*of* Q_p *a n*] *Qp.nat-pow-of-inv Units-eq-nonzero* **by** *presburger*
then show *?thesis*
by (*metis P-set-memI Qp.nat-pow-closed Qp.nonzero-memE*(*2*) *Qp.nonzero-pow-nonzero* ‹*a0* $\in$ *nonzero* Q_p› *a0-def inv-in-frac*(*1*) *inv-in-frac*(*2*))
qed
qed

lemma *P-set-val*:
assumes $a \in$ *P-set* (n::*nat*)
shows (*ord a*) *mod n* = *0*
proof(*cases n* = *0*)
case *True*

```
  then show ?thesis
    using assms zeroth-P-set
    by (metis mod-by-0 of-nat-0 ord-one singletonD)
next
  case False
  then show ?thesis
  proof−
    obtain b where b-def: b ∈ carrier Q_p ∧ (b[^] n) = a
      using assms P-set-def  by blast
    have an: a ∈ nonzero Q_p
      using P-set-def assms by blast
    have bn: b ∈ nonzero Q_p
    proof(rule ccontr)
      assume b ∉ nonzero Q_p
      then have b = 0 _Q_p
        using b-def not-nonzero-Qp
        by metis
      then have (b[^] n) = 0
        using False Qp.cring-axioms cring-def ring.pow-zero
        by blast
      then show False
        using b-def an  Qp.not-nonzero-memI by blast
    qed
    then have ord a = n ∗ (ord b)
      using b-def an nonzero-nat-pow-ord
      by blast
    then show ?thesis
      by (metis mod-mult-self1-is-0)
  qed
qed


lemma P-set-pow:
  assumes n > 0
  assumes s ∈ P-set n
  shows s[^]k ∈ P-set (n∗k)
proof−
  obtain y where y-def: y ∈ carrier Q_p ∧ y[^]n = s
    using assms unfolding P-set-def by blast
  then have 0: y ∈ nonzero Q_p
    using assms P-set-nonzero'(1) Qp-nonzero-nat-pow by blast
  have 1: y[^](n∗k) = s[^] k
    using 0 y-def assms Qp.nat-pow-pow by blast
  hence 2: s[^]k ∈ nonzero Q_p
    using 0 by (metis Qp-nat-pow-nonzero)
  thus ?thesis unfolding P-set-def using 1 y-def by blast
qed
```

13.5 Semialgebraic Sets

In this section we introduce the notion of a p-adic semialgebraic set. Intuitively, these are the subsets of $\mathbb{Q}_p^n$ which are definable by first order quantifier-free formulas in the standard first-order language of rings, with an additional relation symbol included for the relation $\text{val}(x) \leq \text{val}(y)$, interpreted according to the definiton of the p-adic valuation on $\mathbb{Q}_p$. In fact, by Macintyre's quantifier elimination theorem for the first-order theory of $\mathbb{Q}_p$ in this language, one can equivalently remove the "quantifier-free" clause from the latter definition. The definition we give here is also equivalent, and due to Denef in [1]. The given definition here is desirable mainly for its utility in producing a proof of Macintyre's theorem, which is our overarching goal.

13.5.1 Defining Semialgebraic Sets

definition *basic-semialg-set* **where**
basic-semialg-set (*m*::*nat*) (*n*::*nat*) *P* = {*q* ∈ *carrier* (*Q*$_p$^*m*). ∃ *y* ∈ *carrier Q*$_p$. *Qp-ev P q* = (*y*[^]*n*)}

lemma *basic-semialg-set-zero-set*:
assumes *P* ∈ *carrier* (*Q*$_p$[$\mathcal{X}_m$])
assumes *q* ∈ *carrier* (*Q*$_p$^*m*)
assumes *Qp-ev P q* = **0**
assumes *n* ≠ *0*
shows *q* ∈ *basic-semialg-set* (*m*::*nat*) (*n*::*nat*) *P*
proof−
have 0 = (**0**[^]*n*)
using *assms*(*4*) *Qp.nat-pow-zero* **by** *blast*
then show *?thesis*
unfolding *basic-semialg-set-def*
using *assms Qp.cring-axioms cring.cring-simprules*(*2*)
by *blast*
qed

lemma *basic-semialg-set-def'*:
assumes *n* ≠ *0*
assumes *P* ∈ *carrier* (*Q*$_p$[$\mathcal{X}_m$])
shows *basic-semialg-set* (*m*::*nat*) (*n*::*nat*) *P* = {*q* ∈ *carrier* (*Q*$_p$^*m*). *Qp-ev P q* = **0** ∨ *Qp-ev P q* ∈ (*P-set n*)}
proof
show *basic-semialg-set m n P* ⊆ {*q* ∈ *carrier* (*Q*$_p$^*m*). *Qp-ev P q* = **0** ∨ *Qp-ev P q* ∈ *P-set n*}
proof
fix *x*
assume *A*: *x* ∈ *basic-semialg-set m n P*
show *x* ∈ {*q* ∈ *carrier* (*Q*$_p$^*m*). *Qp-ev P q* = **0** ∨ *Qp-ev P q* ∈ *P-set n*}
apply(*cases Qp-ev P x* = **0**)

using *A basic-semialg-set-def* **apply** *blast*
unfolding *basic-semialg-set-def P-set-def*
proof
assume *A0*: *Qp-ev P x* $\neq$ **0**
have *A1*: $\exists\, y{\in}$*carrier* Q_p. *Qp-ev P x* = $(y[\widehat{}]n)$
using *A basic-semialg-set-def*
by *blast*
have *A2*: *x* $\in$ *carrier* $({Q_p}^m)$
using *A basic-semialg-set-def*
by *blast*
show *x* $\in$ *carrier* $({Q_p}^m)$ $\wedge$ (*Qp-ev P x* = **0** $\vee$ *Qp-ev P x* $\in$ {*a* $\in$ *nonzero* Q_p. $\exists\, y{\in}$*carrier* Q_p. $(y[\widehat{}]n)$ = *a*}
by (*metis* (*mono-tags*, *lifting*) *A1 A2 Qp.nonzero-memI assms*(2) *eval-at-point-closed mem-Collect-eq*)
qed
qed
show {*q* $\in$ *carrier* $({Q_p}^m)$. *Qp-ev P q* = **0** $\vee$ *Qp-ev P q* $\in$ *P-set n*} $\subseteq$ *basic-semialg-set m n P*
proof
fix *x*
assume *A*: *x* $\in$ {*q* $\in$ *carrier* $({Q_p}^m)$. *Qp-ev P q* = **0** $\vee$ *Qp-ev P q* $\in$ *P-set n*}
then have *A'*:*x* $\in$ *carrier* $({Q_p}^m)$
by *blast*
show *x* $\in$ *basic-semialg-set m n P*
using *A A'*
apply(*cases Qp-ev P x* = **0**)
using *assms basic-semialg-set-zero-set*[*of P m x n*]
apply *blast*
proof−
assume *B*: *x* $\in$ {*q* $\in$ *carrier* $({Q_p}^m)$. *Qp-ev P q* = **0** $\vee$ *Qp-ev P q* $\in$ *P-set n*}
assume *B'*: *x* $\in$ *carrier* $({Q_p}^m)$
assume *B''*: *Qp-ev P x* $\neq$ **0**
show *x* $\in$ *basic-semialg-set m n P*
unfolding *basic-semialg-set-def P-set-def*
proof
have $\exists\, y{\in}$*carrier* Q_p. *Qp-ev P x* = $(y[\widehat{}]n)$
using *A nonzero-def* [*of* Q_p] **unfolding** *P-set-def*
proof −
assume *x* $\in$ {*q* $\in$ *carrier* $({Q_p}^m)$. *Qp-ev P q* = **0** $\vee$ *Qp-ev P q* $\in$ {*a* $\in$ *nonzero* Q_p. $\exists\, y{\in}$*carrier* Q_p. $(y[\widehat{}]n)$ = *a*}}
then have *Qp-ev P x* $\in$ *nonzero* Q_p $\wedge$ ($\exists\, r$. *r* $\in$ *carrier* Q_p $\wedge$ $(r[\widehat{}]n)$ = *Qp-ev P x*)
using *B''* **by** *blast*
then show *?thesis*
by *blast*
qed
then show *x* $\in$ *carrier* $({Q_p}^m)$ $\wedge$ ($\exists\, y{\in}$*carrier* Q_p. *Qp-ev P x* = $(y[\widehat{}]n)$)
using *B'*
by *blast*

qed
qed
qed
qed

lemma *basic-semialg-set-memI*:
assumes $q \in carrier\ ({Q_p}^m)$
assumes $y \in carrier\ Q_p$
assumes $Qp\text{-}ev\ P\ q = (y[\frown]n)$
shows $q \in basic\text{-}semialg\text{-}set\ m\ n\ P$
using *assms*(*1*) *assms*(*2*) *assms*(*3*) *basic-semialg-set-def*
by *blast*

lemma *basic-semialg-set-memE*:
assumes $q \in basic\text{-}semialg\text{-}set\ m\ n\ P$
shows $q \in carrier\ ({Q_p}^m)$
$\exists\, y \in carrier\ Q_p.\ Qp\text{-}ev\ P\ q = (y[\frown]n)$
using *assms basic-semialg-set-def* **apply** *blast*
using *assms basic-semialg-set-def* **by** *blast*

definition *is-basic-semialg* :: $nat \Rightarrow ((nat \Rightarrow int) \times (nat \Rightarrow int))\ set\ list\ set \Rightarrow bool$ **where**
$is\text{-}basic\text{-}semialg\ m\ S \equiv (\exists\ (n{::}nat) \neq 0.\ (\exists\ P \in carrier\ (Q_p[\mathcal{X}_m]).\ S = basic\text{-}semialg\text{-}set\ m\ n\ P))$

abbreviation(*input*) *basic-semialgs* **where**
$basic\text{-}semialgs\ m \equiv \{S.\ (is\text{-}basic\text{-}semialg\ m\ S)\}$

definition *semialg-sets* **where**
$semialg\text{-}sets\ n = gen\text{-}boolean\text{-}algebra\ (carrier\ ({Q_p}^n))\ (basic\text{-}semialgs\ n)$

lemma *carrier-is-semialg*:
$(carrier\ ({Q_p}^n)) \in semialg\text{-}sets\ n$
unfolding *semialg-sets-def*
using *gen-boolean-algebra.universe* **by** *blast*

lemma *empty-set-is-semialg*:
$\{\} \in semialg\text{-}sets\ n$
using *carrier-is-semialg*[*of n*]
unfolding *semialg-sets-def* **using** *gen-boolean-algebra.complement*
by (*metis Diff-cancel*)

lemma *semialg-intersect*:
assumes $A \in semialg\text{-}sets\ n$
assumes $B \in semialg\text{-}sets\ n$
shows $(A \cap B) \in semialg\text{-}sets\ n$
using *assms*(*1*) *assms*(*2*) *gen-boolean-algebra-intersect semialg-sets-def*
by *blast*

lemma *semialg-union*:
assumes $A \in$ *semialg-sets n*
assumes $B \in$ *semialg-sets n*
shows $(A \cup B) \in$ *semialg-sets n*
using *assms gen-boolean-algebra.union semialg-sets-def*
by *blast*

lemma *semialg-complement*:
assumes $A \in$ *semialg-sets n*
shows $(carrier\ ({Q_p}^n) - A) \in$ *semialg-sets n*
using *assms gen-boolean-algebra.complement semialg-sets-def*
by *blast*

lemma *semialg-zero*:
assumes $A \in$ *semialg-sets 0*
shows $A = \{[]\} \vee A = \{\}$
using *assms*
unfolding *semialg-sets-def cartesian-power-def*
proof−
assume *A0*: $A \in$ *gen-boolean-algebra* (*carrier* (*RDirProd-list* (*R-list 0* Q_p)))
(*basic-semialgs 0*)
show $A = \{[]\} \vee A = \{\}$
proof−
have $A \neq \{[]\} \longrightarrow A = \{\}$
proof
assume *A1*: $A \neq \{[]\}$
show $A = \{\}$
proof−
have (*R-list 0* Q_p) = []
by *simp*
then have (*carrier* (*RDirProd-list* (*R-list 0* Q_p))) = $\{[]\}$
using *RDirProd-list-nil*
by *simp*
then show *?thesis*
using *A0 A1*
by (*metis gen-boolean-algebra-subset subset-singletonD*)
qed
qed
then show *?thesis*
by *linarith*
qed
qed

lemma *basic-semialg-is-semialg*:
assumes *is-basic-semialg n A*
shows $A \in$ *semialg-sets n*
by (*metis* (*no-types*, *lifting*) *assms gen-boolean-algebra.simps inf-absorb1*
is-basic-semialg-def mem-Collect-eq basic-semialg-set-def
semialg-sets-def subsetI)

lemma *basic-semialg-is-semialg'*:
assumes $f \in$ *carrier* $(Q_p[\mathcal{X}_n])$
assumes $m \neq 0$
assumes $A =$ *basic-semialg-set* n m f
shows $A \in$ *semialg-sets* n
using *assms basic-semialg-is-semialg is-basic-semialg-def*
by *blast*

definition *is-semialgebraic* **where**
is-semialgebraic n $S = (S \in$ *semialg-sets* $n)$

lemma *is-semialgebraicE*:
assumes *is-semialgebraic* n S
shows $S \in$ *semialg-sets* n
using *assms is-semialgebraic-def* **by** *blast*

lemma *is-semialgebraic-closed*:
assumes *is-semialgebraic* n S
shows $S \subseteq$ *carrier* $({Q_p}^n)$
using *is-semialgebraicE*[*of* n S] **unfolding** *semialg-sets-def*
using *assms gen-boolean-algebra-subset is-semialgebraicE semialg-sets-def*
by *blast*

lemma *is-semialgebraicI*:
assumes $S \in$ *semialg-sets* n
shows *is-semialgebraic* n S
by (*simp add*: *assms is-semialgebraic-def*)

lemma *basic-semialg-is-semialgebraic*:
assumes *is-basic-semialg* n A
shows *is-semialgebraic* n A
using *assms basic-semialg-is-semialg is-semialgebraicI* **by** *blast*

lemma *basic-semialg-is-semialgebraic'*:
assumes $f \in$ *carrier* $(Q_p[\mathcal{X}_n])$
assumes $m \neq 0$
assumes $A =$ *basic-semialg-set* n m f
shows *is-semialgebraic* n A
using *assms*(*1*) *assms*(*2*) *assms*(*3*) *basic-semialg-is-semialg' is-semialgebraicI*
by *blast*

13.5.2 Algebraic Sets over p-adic Fields

lemma *p-times-square-not-square*:
assumes $a \in$ *nonzero* Q_p
shows $\mathfrak{p} \otimes (a\ [\widehat{}]\ (2::nat)) \notin$ *P-set* $(2::nat)$
proof
assume *A*: $\mathfrak{p} \otimes (a[\widehat{}](2::nat)) \in$ *P-set* $(2::nat)$

then have $\mathfrak{p} \otimes (a[\hat{}](2::nat)) \in nonzero\ Q_p$
unfolding *P-set-def*
by *blast*
then obtain *b* **where** *b-def*: $b \in carrier\ Q_p \wedge b[\hat{}](2::nat) = \mathfrak{p} \otimes (a[\hat{}](2::nat))$
using *A P-set-def* **by** *blast*
have $b \in nonzero\ Q_p$
apply(*rule ccontr*) **using** *b-def assms*
by (*metis A P-set-nonzero'*(*1*) *Qp.nat-pow-zero not-nonzero-Qp zero-neq-numeral*)
then have *LHS*: $ord\ (b[\hat{}](2::nat)) = 2 * (ord\ b)$
using *nonzero-nat-pow-ord*
by *presburger*
have $ord(\ \mathfrak{p} \otimes (a[\hat{}](2::nat))) = 1 + 2 * ord\ a$
using *assms nonzero-nat-pow-ord Qp-nat-pow-nonzero ord-mult ord-p p-nonzero*
by *presburger*
then show *False*
using *b-def LHS*
by *presburger*
qed

lemma *p-times-square-not-square'*:
assumes $a \in carrier\ Q_p$
shows $\mathfrak{p} \otimes (a\ [\hat{}]\ (2::nat)) = \mathbf{0} \Longrightarrow a = \mathbf{0}$
by (*metis Qp.integral Qp.nat-pow-closed Qp.nonzero-closed Qp.nonzero-memE*(*2*) *Qp.nonzero-pow-nonzero assms p-nonzero*)

lemma *zero-set-semialg-set*:
assumes $q \in carrier\ (Q_p[\mathcal{X}_n])$
assumes $a \in carrier\ ({Q_p}^n)$
shows $Qp\text{-}ev\ q\ a = \mathbf{0} \longleftrightarrow (\ \exists y \in carrier\ Q_p.\ \mathfrak{p} \otimes ((Qp\text{-}ev\ q\ a)\ [\hat{}]\ (2::nat)) = y[\hat{}](2::nat))$
proof
show $Qp\text{-}ev\ q\ a = \mathbf{0} \Longrightarrow \exists y \in carrier\ Q_p.\ \mathfrak{p} \otimes (Qp\text{-}ev\ q\ a[\hat{}]\ (2::nat)) = (y[\hat{}]\ (2::nat))$
proof−
assume $Qp\text{-}ev\ q\ a = \mathbf{0}$
then have $\mathfrak{p} \otimes (Qp\text{-}ev\ q\ a[\hat{}](2::nat)) = (\mathbf{0}[\hat{}](2::nat))$
by (*metis Qp.int-inc-closed Qp.nat-pow-zero Qp.r-null zero-neq-numeral*)
then have $\mathbf{0} \in carrier\ Q_p \wedge \mathfrak{p} \otimes (Qp\text{-}ev\ q\ a[\hat{}](2::nat)) = (\mathbf{0}[\hat{}](2::nat))$
using *Qp.cring-axioms cring.cring-simprules*(*2*)
by *blast*
then show $\exists y \in carrier\ Q_p.\ \mathfrak{p} \otimes (Qp\text{-}ev\ q\ a[\hat{}]\ (2::nat)) = (y[\hat{}]\ (2::nat))$
by *blast*
qed
show $\exists y \in carrier\ Q_p.\ \mathfrak{p} \otimes (Qp\text{-}ev\ q\ a[\hat{}](2::nat)) = (y[\hat{}](2::nat)) \Longrightarrow Qp\text{-}ev\ q\ a = \mathbf{0}$
proof−
assume *A*: $\exists y \in carrier\ Q_p.\ \mathfrak{p} \otimes (Qp\text{-}ev\ q\ a[\hat{}](2::nat)) = (y[\hat{}](2::nat))$
then obtain *b* **where** *b-def*: $b \in carrier\ Q_p \wedge \mathfrak{p} \otimes (Qp\text{-}ev\ q\ a[\hat{}](2::nat)) = (b[\hat{}](2::nat))$

```
    by blast
  show Qp-ev q a = 0
  proof(rule ccontr)
    assume  Qp-ev q a ≠ 0
    then have  Qp-ev q a ∈ nonzero Q_p using assms eval-at-point-closed[of a n
q]  nonzero-def
    proof −
      have Qp-ev q a ∈ carrier Q_p
        using ‹[[a ∈ carrier (Q_p^n); q ∈ carrier (Q_p[X_n])]] ⟹
              Qp-ev q a ∈ carrier Q_p› ‹a ∈ carrier (Q_p^n)› ‹q ∈ carrier (Q_p[X_n])›
        by fastforce
      then have Qp-ev q a ∈ {r ∈ carrier Q_p. r ≠ 0}
        using ‹Qp-ev q a ≠ 0› by force
      then show ?thesis
        by (metis nonzero-def )
    qed
    then have 𝔭 ⊗ (Qp-ev q a[^](2::nat)) ∈ nonzero Q_p
      by (metis Qp.nonzero-closed Qp.nonzero-mult-closed Qp-nat-pow-nonzero
not-nonzero-Qp p-nonzero p-times-square-not-square')
    then have 𝔭 ⊗ (Qp-ev q a[^](2::nat)) ∈ P-set (2::nat)
      using b-def
      unfolding P-set-def
      by blast
    then show False
      using ‹Qp-ev q a ∈ nonzero Q_p› p-times-square-not-square
      by blast
  qed
 qed
qed

lemma alg-as-semialg:
 assumes P ∈ carrier (Q_p[X_n])
 assumes q = 𝔭 ⊙_{Q_p[X_n]} (P[^]_{Q_p[X_n]} (2::nat))
 shows zero-set Q_p n P = basic-semialg-set n (2::nat) q
proof
 have 00: ⋀x. x ∈ carrier (Q_p^n) ⟹ Qp-ev q x = 𝔭 ⊗ (Qp-ev P x) [^] (2::nat)
  using assms eval-at-point-smult MP.nat-pow-closed Qp.int-inc-closed eval-at-point-nat-pow
   by presburger
 show V_{Q_p} n P ⊆ basic-semialg-set n 2 q
 proof
   fix x
   assume A: x ∈ V_{Q_p} n P
   show x ∈ basic-semialg-set n (2::nat) q
   proof−
     have P: Qp-ev P x = 0
       using A zero-setE(2)
       by blast
     have Qp-ev q x = 0
     proof−
```

```
      have Qp-ev q x = 𝔭 ⊗ (Qp-ev (P[↑]Q_p[𝒳_n] (2::nat)) x)
          using assms eval-at-point-smult[of x n (P[↑]Q_p[𝒳_n] (2::nat)) 𝔭] basic-semialg-set-def
        by (meson A MP.nat-pow-closed Qp.int-inc-closed zero-setE(1))
      then show ?thesis
     by (metis A P Qp.int-inc-closed Qp.integral-iff Qp.nat-pow-zero Qp.zero-closed assms(1)
          eval-at-point-nat-pow neq0-conv zero-less-numeral zero-setE(1))
    qed
    then have 0: Qp-ev q x = 𝟬 ∨ Qp-ev q x ∈ P-set (2::nat)
      by blast
    have 1: x ∈ carrier (Q_p^n)
      using A zero-setE(1)
      by blast
    then show ?thesis using 0 basic-semialg-set-def'
      by (metis (no-types, opaque-lifting) Qp.nat-pow-zero Qp.zero-closed
        ‹eval-at-point Q_p x q = 𝟬› basic-semialg-set-memI zero-neq-numeral)
  qed
 qed
 show basic-semialg-set n 2 q ⊆ V_Q_p n P
 proof
  fix x
  assume A: x ∈ basic-semialg-set n 2 q
  have 0: ¬ Qp-ev q x ∈ P-set 2
  proof
   assume Qp-ev q x ∈ P-set 2
   then have 0: Qp-ev q x ∈ nonzero Q_p ∧ (∃y ∈ carrier Q_p . (y[↑] (2::nat))
= Qp-ev q x)
    using P-set-def by blast
   have ( ∃y ∈ carrier Q_p. 𝔭 ⊗ ((Qp-ev P x) [↑] (2::nat)) = y[↑](2::nat))
   proof−
    obtain y where y-def: y ∈ carrier Q_p ∧ (y[↑] (2::nat)) = Qp-ev q x
     using 0 by blast
    have x ∈ carrier (Q_p^n)
     using A basic-semialg-set-memE(1) by blast
    then have Qp-ev q x = 𝔭 ⊗ ((Qp-ev P x) [↑] (2::nat))
     using assms eval-at-point-scalar-mult 00 by blast
    then have (y[↑] (2::nat)) = 𝔭 ⊗ ((Qp-ev P x) [↑] (2::nat))
     using y-def by blast
    then show ?thesis using y-def by blast
   qed
   then have Qp-ev P x = 𝟬
    by (metis (no-types, lifting) A assms(1) basic-semialg-set-def mem-Collect-eq
zero-set-semialg-set)
   then have Qp-ev q x = 𝟬
    using assms eval-at-point-smult
   by (metis 00 A Qp.int-inc-closed Qp.nat-pow-zero Qp.r-null basic-semialg-set-memE(1)
zero-neq-numeral)
   then show False
```

using *0 Qp.not-nonzero-memI* **by** *blast*
qed
show $x \in V_{Q_p}\ n\ P$
apply(*rule zero-setI*)
using *A basic-semialg-set-memE*(*1*) **apply** *blast*
using *A 0 00*[*of x*]
by (*metis assms*(*1*) *basic-semialg-set-memE*(*1*) *basic-semialg-set-memE*(*2*) *zero-set-semialg-set*)
qed
qed

lemma *is-zero-set-imp-basic-semialg*:
assumes $P \in$ *carrier* $(Q_p[\mathcal{X}_n])$
assumes $S =$ *zero-set* $Q_p\ n\ P$
shows *is-basic-semialg n S*
unfolding *is-basic-semialg-def*
proof−
obtain *q* **where** *q-def*: $q = \mathfrak{p} \odot_{Q_p[\mathcal{X}_n]} (P[\hat{}]_{Q_p[\mathcal{X}_n]}\ (2::nat))$
by *blast*
have *0*: *zero-set* $Q_p\ n\ P =$ *basic-semialg-set n* $(2::nat)$ *q*
using *alg-as-semialg*[*of P n q*] *q-def assms*(*1*) **by** *linarith*
have $(P\ [\hat{}]_{Q_p[\mathcal{X}_n]}\ (2::nat)) \in$ *carrier* $(Q_p[\mathcal{X}_n])$
using *assms*(*1*)
by *blast*
then have $\mathfrak{p} \odot_{Q_p[\mathcal{X}_n]} (P\ [\hat{}]_{Q_p[\mathcal{X}_n]}\ (2::nat)) \in$ *carrier* $(Q_p[\mathcal{X}_n])$
using *assms q-def Qp.int-inc-closed local.smult-closed* **by** *blast*
then have *1*: $q \in$ *carrier* $(Q_p[\mathcal{X}_n])$
by (*metis q-def*)
then show $\exists\, m.\ m \neq 0 \wedge (\exists\, P \in$ *carrier* $(Q_p[\mathcal{X}_n]).\ S =$ *basic-semialg-set n m P*)
using *0 assms*
by (*metis zero-neq-numeral*)
qed

lemma *is-zero-set-imp-semialg*:
assumes $P \in$ *carrier* $(Q_p[\mathcal{X}_n])$
assumes $S =$ *zero-set* $Q_p\ n\ P$
shows *is-semialgebraic n S*
using *assms*(*1*) *assms*(*2*) *basic-semialg-is-semialg is-semialgebraicI is-zero-set-imp-basic-semialg*
by *blast*

Algebraic sets are semialgebraic

lemma *is-algebraic-imp-is-semialg*:
assumes *is-algebraic* $Q_p\ n\ S$
shows *is-semialgebraic n S*
proof(*rule is-semialgebraicI*)
obtain *ps* **where** *ps-def*: *finite ps* $\wedge$ *ps* $\subseteq$ *carrier* $(Q_p[\mathcal{X}_n]) \wedge S =$ *affine-alg-set* $Q_p\ n\ ps$
using *is-algebraicE*
by (*metis assms*)

have *ps* ⊆ *carrier* (*Q*$_p$[$\mathcal{X}_n$]) ⟶ *affine-alg-set* *Q*$_p$ *n ps* ∈ *semialg-sets n*
apply(*rule finite.induct*[*of ps*])
apply (*simp add*: *ps-def*)
using *affine-alg-set-empty*[*of n*]
apply (*simp add*: *carrier-is-semialg*)
proof
fix *A a*
assume *IH*: *A* ⊆ *carrier* (*Q*$_p$[$\mathcal{X}_n$]) ⟶ *affine-alg-set* *Q*$_p$ *n A* ∈ *semialg-sets n*
assume *P*: *insert a A* ⊆ *carrier* (*Q*$_p$[$\mathcal{X}_n$])
have *A* ⊆ *carrier* (*Q*$_p$[$\mathcal{X}_n$])
using *P* **by** *blast*
then
show *affine-alg-set* *Q*$_p$ *n* (*insert a A*) ∈ *semialg-sets n*
using *IH P semialg-intersect*[*of affine-alg-set* *Q*$_p$ *n A n affine-alg-set* *Q*$_p$ *n* {*a*}]
is-zero-set-imp-semialg affine-alg-set-insert[*of n a A*]
by (*metis Int-commute affine-alg-set-singleton insert-subset is-semialgebraicE*)
qed
then show *S* ∈ *semialg-sets n*
using *ps-def* **by** *blast*
qed

13.5.3 Basic Lemmas about the Semialgebraic Predicate

Finite and cofinite sets are semialgebraic

lemma *finite-is-semialg*:
assumes *F* ⊆ *carrier* (*Q*$_p{}^n$)
assumes *finite F*
shows *is-semialgebraic n F*
using *Qp.finite-sets-are-algebraic is-algebraic-imp-is-semialg*[*of n F*]
assms(*1*) *assms*(*2*)
by *blast*

definition *is-cofinite* **where**
is-cofinite n F = *finite* (*ring-pow-comp* *Q*$_p$ *n F*)

lemma *is-cofiniteE*:
assumes *F* ⊆ *carrier* (*Q*$_p{}^n$)
assumes *is-cofinite n F*
shows *finite* (*carrier* (*Q*$_p{}^n$) − *F*)
using *assms*(*2*) *is-cofinite-def*
by (*simp add*: *ring-pow-comp-def*)

lemma *complement-is-semialg*:
assumes *is-semialgebraic n F*
shows *is-semialgebraic n* ((*carrier* (*Q*$_p{}^n$)) − *F*)
using *assms is-semialgebraic-def semialg-complement* **by** *blast*

lemma *cofinite-is-semialgebraic*:
assumes $F \subseteq$ *carrier* $({Q_p}^n)$
assumes *is-cofinite n F*
shows *is-semialgebraic n F*
using *assms ring-pow-comp-inv*[*of F* Q_p *n*] *complement-is-semialg*[*of n* (*carrier* $({Q_p}^n) - F$)]
finite-is-semialg[*of* (*carrier* $({Q_p}^n) - F$)] *is-cofiniteE*[*of F*]
by (*simp add*: *ring-pow-comp-def*)

lemma *diff-is-semialgebraic*:
assumes *is-semialgebraic n A*
assumes *is-semialgebraic n B*
shows *is-semialgebraic n* $(A - B)$
apply(*rule is-semialgebraicI*)
using *assms* **unfolding** *semialg-sets-def*
using *gen-boolean-algebra-diff is-semialgebraicE semialg-sets-def*
by *blast*

lemma *intersection-is-semialg*:
assumes *is-semialgebraic n A*
assumes *is-semialgebraic n B*
shows *is-semialgebraic n* $(A \cap B)$
using *assms*(*1*) *assms*(*2*) *is-semialgebraicE is-semialgebraicI semialg-intersect*
by *blast*

lemma *union-is-semialgebraic*:
assumes *is-semialgebraic n A*
assumes *is-semialgebraic n B*
shows *is-semialgebraic n* $(A \cup B)$
using *assms*(*1*) *assms*(*2*) *is-semialgebraicE is-semialgebraicI semialg-union* **by** *blast*

lemma *carrier-is-semialgebraic*:
is-semialgebraic n (*carrier* $({Q_p}^n)$)
using *carrier-is-semialg*
by (*simp add*: *carrier-is-semialg is-semialgebraic-def*)

lemma *empty-is-semialgebraic*:
is-semialgebraic n $\{\}$
by (*simp add*: *empty-set-is-semialg is-semialgebraic-def*)

13.5.4 One-Dimensional Semialgebraic Sets

definition *one-var-semialg* **where**
one-var-semialg S = $((to\text{-}R1\ `\ S) \in (semialg\text{-}sets\ 1))$

definition *univ-basic-semialg-set* **where**
univ-basic-semialg-set (*m*::*nat*) *P* = $\{a \in carrier\ Q_p.\ (\exists\, y \in carrier\ Q_p.\ (P \cdot a = (y[\hat{}]m)))\}$

Equivalence of univ_basic_semialg_sets and semialgebraic subsets of $\mathbb{Q}^1$

lemma *univ-basic-semialg-set-to-semialg-set*:
assumes $P \in$ *carrier* Q_p*-x*
assumes $m \neq 0$
shows *to-R1* ‘ (*univ-basic-semialg-set m P*) = *basic-semialg-set 1 m* (*from-Qp-x P*)
proof
show $(\lambda a.\ [a])$ ‘ *univ-basic-semialg-set m P* ⊆ *basic-semialg-set 1 m* (*from-Qp-x P*)
proof fix *x*
assume *A*: $x \in (\lambda a.\ [a])$ ‘ *univ-basic-semialg-set m P*
then obtain *b y* **where** *by-def*:$b \in$ *carrier* $Q_p \wedge y \in$ *carrier* $Q_p \wedge (P \cdot b) = (y[\widehat{}]m) \wedge x = [b]$
unfolding *univ-basic-semialg-set-def*
by *blast*
then have $x \in$ *carrier* $({Q_p}^1)$
using *A Qp.to-R1-closed*[*of b*]
unfolding *univ-basic-semialg-set-def*
by *blast*
then show $x \in$ *basic-semialg-set 1 m* (*from-Qp-x P*)
using *by-def Qp-x-Qp-poly-eval assms*
unfolding *basic-semialg-set-def*
by *blast*
qed
show *basic-semialg-set 1 m* (*from-Qp-x P*) ⊆ $(\lambda a.\ [a])$ ‘ *univ-basic-semialg-set m P*
proof
fix *x*
assume *A*: $x \in$ *basic-semialg-set 1 m* (*from-Qp-x P*)
then obtain *b* **where** *b-def*: $b \in$ *carrier* $Q_p \wedge x = [b]$
unfolding *basic-semialg-set-def*
by (*metis* (*mono-tags*, *lifting*) *mem-Collect-eq Qp.to-R1-to-R Qp.to-R-pow-closed*)
obtain *y* **where** *y-def*: $y \in$ *carrier* $Q_p \wedge$ (*Qp-ev* (*from-Qp-x P*) $[b]$ = $(y[\widehat{}]m)$)
using *A b-def*
unfolding *basic-semialg-set-def*
by *blast*
have $P \cdot b = (y[\widehat{}]m)$
using *assms y-def b-def Qp-x-Qp-poly-eval* **by** *blast*
then show $x \in (\lambda a.\ [a])$ ‘ *univ-basic-semialg-set m P*
using *y-def b-def*
unfolding *basic-semialg-set-def univ-basic-semialg-set-def*
by *blast*
qed
qed

definition *is-univ-semialgebraic* **where**
is-univ-semialgebraic S = (*S* ⊆ *carrier* $Q_p \wedge$ *is-semialgebraic 1* (*to-R1* ‘ *S*))

lemma *is-univ-semialgebraicE*:

assumes *is-univ-semialgebraic S*
shows *is-semialgebraic 1* (*to-R1* ‘ *S*)
using *assms is-univ-semialgebraic-def* **by** *blast*

lemma *is-univ-semialgebraicI*:
assumes *is-semialgebraic 1* (*to-R1* ‘ *S*)
shows *is-univ-semialgebraic S*
proof−
have *S* ⊆ *carrier* Q_p
proof fix *x* **assume** *x* ∈ *S*
then have (*to-R1 x*) ∈ *carrier* ($Q_p{}^1$)
using *assms*
by (*smt Collect-mono-iff gen-boolean-algebra-subset image-def is-semialgebraicE mem-Collect-eq semialg-sets-def Qp.to-R1-carrier*)
then show *x* ∈ *carrier* Q_p
using *assms*
by (*metis nth-Cons-0 Qp.to-R-pow-closed*)
qed
then show *?thesis*
using *assms*
unfolding *is-univ-semialgebraic-def*
by *blast*
qed

lemma *univ-basic-semialg-set-is-univ-semialgebraic*:
assumes *P* ∈ *carrier* Q_p*-x*
assumes *m* ≠ *0*
shows *is-univ-semialgebraic* (*univ-basic-semialg-set m P*)
using *assms*
by (*metis* (*mono-tags*, *lifting*) *basic-semialg-is-semialgebraic′ from-Qp-x-closed is-univ-semialgebraic-def mem-Collect-eq subsetI univ-basic-semialg-set-def univ-basic-semialg-set-to-semialg-set*)

lemma *intersection-is-univ-semialgebraic*:
assumes *is-univ-semialgebraic A*
assumes *is-univ-semialgebraic B*
shows *is-univ-semialgebraic* (*A* ∩ *B*)
using *assms intersection-is-semialg*[*of 1* ((λ*a*. [*a*]) ‘ *A*) ((λ*a*. [*a*]) ‘ *B*)]
unfolding *is-univ-semialgebraic-def*
by (*metis le-infI1 Qp.to-R1-intersection*)

lemma *union-is-univ-semialgebraic*:
assumes *is-univ-semialgebraic A*
assumes *is-univ-semialgebraic B*
shows *is-univ-semialgebraic* (*A* ∪ *B*)
using *assms union-is-semialgebraic*[*of 1* ((λ*a*. [*a*]) ‘ *A*) ((λ*a*. [*a*]) ‘ *B*)]
unfolding *is-univ-semialgebraic-def*
by (*metis Un-subset-iff image-Un*)

lemma *diff-is-univ-semialgebraic*:
assumes *is-univ-semialgebraic A*
assumes *is-univ-semialgebraic B*
shows *is-univ-semialgebraic* $(A - B)$
using *assms diff-is-semialgebraic*[*of 1* $((\lambda a.\ [a])\ `\ A)\ ((\lambda a.\ [a])\ `\ B)$]
unfolding *is-univ-semialgebraic-def*
by (*smt Diff-subset subset-trans Qp.to-R1-diff*)

lemma *finite-is-univ-semialgebraic*:
assumes $A \subseteq$ *carrier* Q_p
assumes *finite A*
shows *is-univ-semialgebraic A*
using *assms finite-is-semialg*[*of* $((\lambda a.\ [a])\ `\ A)\ ]$ *to-R1-finite*[*of A*]
unfolding *is-univ-semialgebraic-def*
by (*metis Qp.to-R1-carrier Qp.to-R1-subset*)

13.5.5 Defining the p-adic Valuation Semialgebraically

lemma *Qp-square-root-criterion0*:
assumes $p \neq 2$
assumes $a \in$ *carrier* Q_p
assumes $b \in$ *carrier* Q_p
assumes *val* $a \leq$ *val* b
assumes $a \neq \mathbf{0}$
assumes $b \neq \mathbf{0}$
assumes *val* $a \geq 0$
shows $\exists y \in$ *carrier* Q_p. $a[\hat{}](2::nat) \oplus_{Q_p} \mathfrak{p} \otimes b[\hat{}](2::nat) = (y\ [\hat{}]\ (2::nat))$
proof−
have *0*: (*to-Zp a*) $\in$ *carrier* Z_p
using *assms*(*2*) *to-Zp-closed*
by *blast*
have *1*: (*to-Zp b*) $\in$ *carrier* Z_p
using *assms*(*3*) *to-Zp-closed*
by *blast*
have *2*: $a \in \mathcal{O}_p$
using *val-ring-val-criterion assms*(*2*) *assms*(*5*) *assms*(*7*) **by** *blast*
have *3*: $b \in \mathcal{O}_p$
using *assms val-ring-val-criterion*[*of b*] *dual-order.trans* **by** *blast*
have *4*: *val-Zp* (*to-Zp b*) = *val b*
using *3 Zp-def ι-def padic-fields.to-Zp-val padic-fields-axioms* **by** *blast*
have *5*: *val-Zp* (*to-Zp a*) = *val a*
using Q_p*-def Zp-def assms*(*2*) *assms*(*7*) *padic-fields.Qp-val-ringI padic-fields.to-Zp-val padic-fields-axioms*
by *blast*
have $\exists y \in$ *carrier* Z_p. (*to-Zp a*)$[\hat{}]_{Z_p}(2::nat) \oplus_{Z_p}$ p $\otimes_{Z_p}$(*to-Zp b*)$[\hat{}]_{Z_p}(2::nat)$ = $(y\ [\hat{}]_{Z_p}\ (2::nat))$
using *0 1 2 4 5 assms Zp-square-root-criterion*[*of* (*to-Zp a*) (*to-Zp b*)]
by (*metis 3 to-Zp-inc to-Zp-zero zero-in-val-ring*)
then obtain *y* **where** *y-def*: $y \in$ *carrier* $Z_p \wedge$ (*to-Zp a*)$[\hat{}]_{Z_p}(2::nat) \oplus_{Z_p}$ p

$\otimes_{Z_p}$(*to-Zp b*)[^]$_{Z_p}$(*2::nat*) = (*y* [^]$_{Z_p}$ (*2::nat*))
by *blast*
have *6*: *a*[^](*2::nat*) $\oplus_{Q_p}$ 𝔭 $\otimes$*b*[^](*2::nat*) = ((*ι y*) [^] (*2::nat*))
proof−
have *0*: *ι* (*y* [^]$_{Z_p}$ (*2::nat*)) = ((*ι y*) [^] (*2::nat*))
using *Qp-nonzero-nat-pow nat-pow-closed inc-pow nat-inc-zero inc-is-hom ι-def y-def ring-hom-nat-pow*[*of* Z_p Q_p *ι y 2*]
Q_p*-def Qp.ring-axioms Zp.ring-axioms*
by *blast*
have *1*: *ι* (*y* [^]$_{Z_p}$ (*2::nat*)) = *ι* ((*to-Zp a*)[^]$_{Z_p}$(*2::nat*) $\oplus_{Z_p}$ p $\otimes_{Z_p}$(*to-Zp b*)[^]$_{Z_p}$(*2::nat*))
using *y-def* **by** *presburger*
have *2*: *ι* (*y* [^]$_{Z_p}$ (*2::nat*)) = *ι* ((*to-Zp a*)[^]$_{Z_p}$(*2::nat*)) $\oplus_{Q_p}$ *ι* (p $\otimes_{Z_p}$(*to-Zp b*)[^]$_{Z_p}$(*2::nat*))
using *1 Zp.m-closed Zp-int-inc-closed assms*(*2*) *assms*(*3*) *inc-of-sum pow-closed to-Zp-closed* **by** *presburger*
hence *3*: *ι* (*y* [^]$_{Z_p}$ (*2::nat*)) = (*ι* (*to-Zp a*))[^](*2::nat*) $\oplus$ (*ι* p) $\otimes$ *ι* ((*to-Zp b*)[^]$_{Z_p}$(*2::nat*))
using *Qp-nonzero-nat-pow nat-pow-closed inc-pow nat-inc-zero inc-is-hom ι-def y-def ring-hom-nat-pow*[*of* Z_p Q_p *ι - 2*]
Q_p*-def Qp.ring-axioms Zp.ring-axioms Zp-int-inc-closed assms*(*2*) *assms*(*3*) *inc-of-prod pow-closed to-Zp-closed*
by *metis*
then show *?thesis*
using *0 4 val-ring-ord-criterion assms*(*2*) *assms*(*3*) *assms*(*4*) *assms*(*5*) *assms*(*6*) *assms*(*7*) *inc-pow not-nonzero-Zp ord-of-nonzero*(*1*) *p-inc to-Zp-closed to-Zp-inc*
by (*metis to-Zp-zero val-pos val-ringI zero-in-val-ring*)
qed
have (*ι y*) ∈ *carrier* Q_p
using *frac-closed local.inc-def y-def inc-closed* **by** *blast*
then show *?thesis*
using *6*
by *blast*
qed

lemma *eint-minus-ineq′*:
assumes (*a*::*eint*) ≥ *b*
shows *a* − *b* ≥ *0*
by (*metis assms eint-minus-ineq eint-ord-simps*(*3*) *idiff-infinity idiff-self order-trans top.extremum-unique top-eint-def*)

lemma *Qp-square-root-criterion*:
assumes *p* ≠ *2*
assumes *a* ∈ *carrier* Q_p
assumes *b* ∈ *carrier* Q_p
assumes *ord b* ≥ *ord a*
assumes *a* ≠ **0**
assumes *b* ≠ **0**

shows $\exists y \in$ *carrier* Q_p. $a[\hat{}](2::nat) \oplus_{Q_p} \mathfrak{p}\otimes b[\hat{}](2::nat) = (y\ [\hat{}]\ (2::nat))$
proof−
have $\exists k::int.\ k \leq min\ (ord\ a)\ (ord\ b) \wedge k\ mod\ 2 = 0$
proof−
let *?k* = *if* $(min\ (ord\ a)\ (ord\ b))\ mod\ 2 = 0$ *then min* $(ord\ a)\ (ord\ b)$ *else* $(min\ (ord\ a)\ (ord\ b)) - 1$
have *?k* $\leq min\ (ord\ a)\ (ord\ b) \wedge$ *?k mod* $2 = 0$
apply(*cases* $(min\ (ord\ a)\ (ord\ b))\ mod\ 2 = 0$)
apply *presburger*
by *presburger*
then show *?thesis*
by *meson*
qed
then obtain *k* **where** *k-def*: $k \leq min\ (ord\ a)\ (ord\ b) \wedge k\ mod\ 2 = 0$
by *meson*
obtain *a0* **where** *a0-def*: $a0 = (\mathfrak{p}[\hat{}](-k)) \otimes a$
by *blast*
obtain *b0* **where** *b0-def*: $b0 = (\mathfrak{p}[\hat{}](-k)) \otimes b$
by *blast*
have *0*: $a0 \in nonzero\ Q_p$
using *Qp.cring-axioms Qp.field-axioms Ring.integral a0-def assms*(2) *assms*(5) *cring-simprules*(5) *not-nonzero-Qp p-intpow-closed*(1) *p-nonzero*
by (*metis Qp-int-pow-nonzero cring.cring-simprules*(5))
have *1*: $val\ a0 = val\ a - k$
using *a0-def assms*(2) *assms*(5) *val-mult p-nonzero p-intpow-closed*(1)
by (*metis Qp.m-comm Qp-int-pow-nonzero p-intpow-inv'' val-fract val-p-int-pow*)
have *11*: $val\ b0 = val\ b - k$
using *assms*(3) *assms*(6) *b0-def val-mult p-nonzero p-intpow-closed*(1)
by (*metis Qp.m-lcomm Qp.one-closed Qp.r-one Qp-int-pow-nonzero p-intpow-inv'' val-fract val-p-int-pow*)
have *A*: $val\ a \geq k$
using *k-def val-ord assms* **by** (*smt eint-ord-simps*(1) *not-nonzero-Qp*)
have *B*: $val\ b \geq k$
using *k-def val-ord assms* **by** (*smt eint-ord-simps*(1) *not-nonzero-Qp*)
then have *2*: $val\ a0 \geq 0$
using *A 1 assms k-def eint-minus-ineq eint-ord-code*(5) *local.eint-minus-ineq'* **by** *presburger*
have *3*: $val\ a0 \leq val\ b0$
using *1 11 assms*
by (*metis eint.distinct*(2) *eint-minus-ineq eint-ord-simps*(1) *val-def*)
have *4*: $a0 \neq \mathbf{0}$
using *a0-def 0 Qp.nonzero-memE*(2) **by** *blast*
have *5*: $b0 \neq \mathbf{0}$
using *b0-def*
by (*metis 4 Qp.integral-iff a0-def assms*(2) *assms*(3) *assms*(6) *p-intpow-closed*(1))
have $\exists y \in$ *carrier* Q_p. $a0[\hat{}](2::nat) \oplus_{Q_p} \mathfrak{p}\otimes b0[\hat{}](2::nat) = (y\ [\hat{}]\ (2::nat))$
using *Qp-square-root-criterion0*[*of a0 b0*] *assms 2 3 4 5 b0-def a0-def Qp.m-closed p-intpow-closed*(1)

by *metis*
then obtain *y* **where** *y-def*: $y \in carrier\ Q_p \wedge a0[\hat{}](2::nat) \oplus_{Q_p} \mathfrak{p} \otimes b0[\hat{}](2::nat) = (y\ [\hat{}]\ (2::nat))$
by *blast*
then have *6*: $(\mathfrak{p}[\hat{}]\ (2 * k)) \otimes\ (a0[\hat{}](2::nat) \oplus_{Q_p}\ \mathfrak{p} \otimes b0[\hat{}](2::nat)) =\ (\mathfrak{p}[\hat{}]\ (2 * k))\ \otimes (y\ [\hat{}]\ (2::nat))$
by *presburger*
then have *8*: $((\mathfrak{p}[\hat{}]\ (2 * k)) \otimes\ (a0[\hat{}](2::nat))) \oplus_{Q_p} ((\mathfrak{p}[\hat{}]\ (2 * k)) \otimes (\mathfrak{p} \otimes b0[\hat{}](2::nat))) =\ (\mathfrak{p}[\hat{}]\ (2 * k))\ \otimes (y\ [\hat{}]\ (2::nat))$
using *6 Qp.r-distr*[*of* $(a0[\hat{}](2::nat))\ (\mathfrak{p} \otimes b0[\hat{}](2::nat))\ (\mathfrak{p}[\hat{}]\ (2 * k))$]
by (*metis Qp.add.int-pow-closed Qp.m-closed Qp.nat-pow-closed Qp.one-closed a0-def assms*(*2*) *assms*(*3*) *b0-def p-inc p-intpow-closed*(*1*) *y-def*)
have *9*: $(\mathfrak{p}[\hat{}](int\ 2*k)) = (\mathfrak{p}[\hat{}]k)[\hat{}](2::nat)$
using *Qp-int-nat-pow-pow*[*of* $\mathfrak{p}\ k\ 2$]
by (*metis mult-of-nat-commute p-nonzero*)
then have $((\mathfrak{p}[\hat{}]k)[\hat{}](2::nat) \otimes\ (a0[\hat{}](2::nat))) \oplus_{Q_p} (\mathfrak{p}[\hat{}]k)[\hat{}](2::nat) \otimes (\mathfrak{p} \otimes b0[\hat{}](2::nat)) =\ (\mathfrak{p}[\hat{}]k)[\hat{}](2::nat) \otimes (y\ [\hat{}]\ (2::nat))$
by (*metis 8 int-eq-iff-numeral*)
then have $((\mathfrak{p}[\hat{}]k) \otimes a0)[\hat{}](2::nat) \oplus_{Q_p} ((\mathfrak{p}[\hat{}]k)[\hat{}](2::nat)) \otimes (\mathfrak{p} \otimes b0[\hat{}](2::nat)) = ((\mathfrak{p}[\hat{}]k)[\hat{}](2::nat))\ \otimes (y\ [\hat{}]\ (2::nat))$
by (*metis Qp.cring-axioms a0-def assms*(*2*) *comm-monoid.nat-pow-distrib cring.cring-simprules*(*5*) *cring-def p-intpow-closed*(*1*))
then have *10*: $((\mathfrak{p}[\hat{}]k) \otimes a0)[\hat{}](2::nat) \oplus_{Q_p} ((\mathfrak{p}[\hat{}]k)[\hat{}](2::nat)) \otimes (\mathfrak{p} \otimes b0[\hat{}](2::nat)) =\ ((\mathfrak{p}[\hat{}]k)\ \otimes y)\ [\hat{}]\ (2::nat)$
using *comm-monoid.nat-pow-distrib y-def*
by (*metis Qp.comm-monoid-axioms p-intpow-closed*(*1*))
then have $((\mathfrak{p}[\hat{}]k) \otimes a0)[\hat{}](2::nat) \oplus_{Q_p} ((((\mathfrak{p}[\hat{}]k)[\hat{}](2::nat)) \otimes \mathfrak{p}) \otimes b0[\hat{}](2::nat)) =\ ((\mathfrak{p}[\hat{}]k)\ \otimes y)\ [\hat{}]\ (2::nat)$
using *10 monoid.m-assoc*[*of* $Q_p\ ((\mathfrak{p}[\hat{}]k)[\hat{}](2::nat))\ \mathfrak{p}\ \ b0[\hat{}](2::nat)$]
by (*metis Qp.int-inc-closed Qp.m-assoc Qp.m-closed Qp.nat-pow-closed assms*(*3*) *b0-def p-intpow-closed*(*1*))
then have $((\mathfrak{p}[\hat{}]k) \otimes a0)[\hat{}](2::nat) \oplus_{Q_p} ((\mathfrak{p} \otimes ((\mathfrak{p}[\hat{}]k)[\hat{}](2::nat)))) \otimes b0[\hat{}](2::nat) =\ ((\mathfrak{p}[\hat{}]k)\ \otimes y)\ [\hat{}]\ (2::nat)$
by (*metis Qp.group-commutes-pow Qp.int-inc-closed Qp.m-comm p-intpow-closed*(*1*))
then have $((\mathfrak{p}[\hat{}]k) \otimes a0)[\hat{}](2::nat) \oplus_{Q_p} \mathfrak{p} \otimes (((\mathfrak{p}[\hat{}]k)[\hat{}](2::nat)) \otimes b0[\hat{}](2::nat)) =\ ((\mathfrak{p}[\hat{}]k)\ \otimes y)\ [\hat{}]\ (2::nat)$
by (*metis 10 Qp.int-inc-closed Qp.m-closed Qp.m-lcomm Qp.nat-pow-closed assms*(*3*) *b0-def p-intpow-closed*(*1*))
then have $((\mathfrak{p}[\hat{}]k) \otimes a0)[\hat{}](2::nat) \oplus_{Q_p} \mathfrak{p} \otimes ((\mathfrak{p}[\hat{}]k) \otimes b0)[\hat{}](2::nat) =\ ((\mathfrak{p}[\hat{}]k)\ \otimes y)\ [\hat{}]\ (2::nat)$
by (*metis Qp.m-closed Qp.nat-pow-distrib assms*(*3*) *b0-def p-intpow-closed*(*1*))
then have $a[\hat{}](2::nat) \oplus_{Q_p} \mathfrak{p} \otimes b[\hat{}](2::nat) =\ ((\mathfrak{p}[\hat{}]k)\ \otimes y)\ [\hat{}]\ (2::nat)$
by (*metis Qp.l-one Qp.m-assoc a0-def assms*(*2*) *assms*(*3*) *b0-def p-intpow-closed*(*1*) *p-intpow-inv*)
then show *?thesis*
by (*meson Qp.cring-axioms cring.cring-simprules*(*5*) *p-intpow-closed*(*1*) *y-def*)
qed

lemma *Qp-val-ring-alt-def0*:
assumes $a \in$ *nonzero* Q_p
assumes *ord* $a \geq 0$
shows $\exists y \in$ *carrier* Q_p. $\mathbf{1} \oplus_{Q_p} (\mathfrak{p}[\hat{}](3::nat)) \otimes (a[\hat{}](4::nat)) = (y[\hat{}](2::nat))$
proof−
have $\exists y \in$ *carrier* Z_p. $\mathbf{1}_{Z_p} \oplus_{Z_p} (\mathrm{p}\ [\hat{}]_{Z_p}\ (3::nat)) \otimes_{Z_p} ((to\text{-}Zp\ a)\ [\hat{}]_{Z_p}\ (4::nat)) = (y\ [\hat{}]_{Z_p}\ (2::nat))$
using *padic-integers.Zp-semialg-eq*[*of p to-Zp a*] *prime assms to-Zp-def*
by (*metis* (*no-types*, *lifting*) *Qp.nonzero-closed Qp.not-nonzero-memI Zp-def val-ring-ord-criterion not-nonzero-Zp padic-integers-axioms to-Zp-closed to-Zp-inc to-Zp-zero zero-in-val-ring*)
then obtain *y* **where** *y-def*: $y \in$ *carrier* $Z_p \wedge \mathbf{1}_{Z_p} \oplus_{Z_p} (\mathrm{p}\ [\hat{}]_{Z_p}\ (3::nat)) \otimes_{Z_p} ((to\text{-}Zp\ a)\ [\hat{}]_{Z_p}\ (4::nat)) = (y\ [\hat{}]_{Z_p}\ (2::nat))$
by *blast*
then have $\mathbf{1} \oplus_{Q_p} (\mathfrak{p}[\hat{}](3::nat)) \otimes (a[\hat{}](4::nat)) = ((\iota\ y)[\hat{}](2::nat))$
using *Group.nat-pow-0 Group.nat-pow-Suc nonzero-def val-ring-ord-criterion assms inc-of-nonzero inc-of-prod inc-of-sum inc-pow m-closed nat-inc-closed nat-pow-closed not-nonzero-Zp numeral-2-eq-2 p-natpow-inc to-Zp-closed to-Zp-inc*
by (*smt Qp.nonzero-closed Qp.nonzero-memE*(2) *Zp.monom-term-car p-pow-nonzero*(1) *pow-closed to-Zp-zero zero-in-val-ring*)
then have $(\iota\ y) \in$ *carrier* $Q_p \wedge \mathbf{1} \oplus_{Q_p} (\mathfrak{p}[\hat{}](3::nat)) \otimes (a[\hat{}](4::nat)) = ((\iota\ y)[\hat{}](2::nat))$
using *y-def inc-closed* **by** *blast*
then show *?thesis*
by *blast*
qed

Defining the valuation semialgebraically for odd primes

lemma *P-set-ord-semialg-odd-p*:
assumes $p \neq 2$
assumes $a \in$ *carrier* Q_p
assumes $b \in$ *carrier* Q_p
shows *val* $a \leq$ *val* $b \longleftrightarrow (\exists y \in$ *carrier* Q_p. $(a[\hat{}](2::nat)) \oplus_{Q_p} (\mathfrak{p} \otimes (b[\hat{}](2::nat))) = (y[\hat{}](2::nat)))$
proof(*cases* $a = \mathbf{0}$)
case *True*
show *val* $a \leq$ *val* $b \longleftrightarrow (\exists y \in$ *carrier* Q_p. $(a[\hat{}](2::nat)) \oplus_{Q_p} (\mathfrak{p} \otimes (b[\hat{}](2::nat))) = (y[\hat{}](2::nat)))$
proof
show *val* $b \geq$ *val* $a \Longrightarrow \exists y \in$*carrier* Q_p. $(a[\hat{}](2::nat)) \oplus_{Q_p} \mathfrak{p} \otimes (b[\hat{}](2::nat)) = (y[\hat{}](2::nat))$
proof−
assume *A*: *val* $b \geq$ *val* a
then have *val* $b \geq \infty$
by (*metis True local.val-zero*)
then have $b = \mathbf{0}$
using *assms*(3) *local.val-zero val-ineq* **by** *presburger*

then have $(a[\hat{}](2::nat)) \oplus_{Q_p} \mathfrak{p} \otimes (b[\hat{}](2::nat)) = (\mathbf{0}[\hat{}](2::nat))$
using *True*
by (*metis Qp.int-inc-zero Qp.int-nat-pow-rep Qp.nonzero-closed Qp.r-null Qp.r-zero assms*(3) *p-nonzero zero-power2*)
then show *?thesis*
using ‹$b = \mathbf{0}$› *assms*(3) **by** *blast*
qed
show $\exists\, y{\in}carrier\ Q_p.\ (a[\hat{}](2::nat)) \oplus_{Q_p} \mathfrak{p} \otimes (b[\hat{}](2::nat)) = (y[\hat{}](2::nat)) \Longrightarrow val\ b \geq val\ a$
proof −
assume $\exists\, y{\in}carrier\ Q_p.\ (a[\hat{}](2::nat)) \oplus_{Q_p} \mathfrak{p} \otimes (b[\hat{}](2::nat)) = (y[\hat{}](2::nat))$
then obtain *y* **where** *y-def*: $y \in carrier\ Q_p \wedge (a[\hat{}](2::nat)) \oplus_{Q_p} \mathfrak{p} \otimes (b[\hat{}](2::nat)) = (y[\hat{}](2::nat))$
by *blast*
then have *0*: $\mathfrak{p} \otimes (b[\hat{}](2::nat)) = (y[\hat{}](2::nat))$
by (*metis* (*no-types*, *lifting*) *Qp.add.r-cancel-one' Qp.int-inc-closed Qp.nat-pow-closed Qp.not-nonzero-memI Qp-nonzero-nat-pow True assms*(2) *assms*(3) *local.monom-term-car not-nonzero-Qp zero-less-numeral*)
have $b = \mathbf{0}$
apply(*rule ccontr*)
using *0 assms y-def p-times-square-not-square*[*of b*]
unfolding *P-set-def*
by (*metis* (*no-types*, *opaque-lifting*) *P-set-memI Qp.nat-pow-closed True* ‹$b \in nonzero\ Q_p \Longrightarrow \mathfrak{p} \otimes b\ [\hat{}]\ 2 \notin P\text{-}set\ 2$› *not-nonzero-Qp p-times-square-not-square'*)
then show *?thesis*
using *eint-ord-code*(3) *local.val-zero* **by** *presburger*
qed
qed
next
case *False*
then show *?thesis*
proof(*cases* $b = \mathbf{0}$)
case *True*
then have $(a[\hat{}](2::nat)) \oplus_{Q_p} (\mathfrak{p} \otimes (b[\hat{}](2::nat))) = (a[\hat{}](2::nat))$
by (*metis Qp.add.l-cancel-one' Qp.int-inc-zero Qp.int-nat-pow-rep Qp.nat-pow-closed Qp.nonzero-closed Qp.r-null assms*(2) *assms*(3) *p-nonzero zero-power2*)
then have *0*: $(\exists\, y \in carrier\ Q_p.\ (a[\hat{}](2::nat)) \oplus_{Q_p} (\mathfrak{p} \otimes (b[\hat{}](2::nat))) = (y[\hat{}](2::nat)))$
using *assms*(2)
by *blast*
have *1*: $val\ a \leq val\ b$
using *True assms local.val-zero eint-ord-code*(3) **by** *presburger*
show $val\ a \leq val\ b \longleftrightarrow (\exists\, y \in carrier\ Q_p.\ (a[\hat{}](2::nat)) \oplus_{Q_p} (\mathfrak{p} \otimes (b[\hat{}](2::nat))) = (y[\hat{}](2::nat)))$
using *0 1*
by *blast*
next
case *F*: *False*
show $val\ a \leq val\ b \longleftrightarrow (\exists\, y \in carrier\ Q_p.\ (a[\hat{}](2::nat)) \oplus_{Q_p} (\mathfrak{p} \otimes (b[\hat{}](2::nat)))$

$= (y[\widehat{}](2::nat)))$
proof
show *val b* $\geq$ *val a* $\Longrightarrow \exists\, y{\in}carrier\ Q_p.\ (a[\widehat{}](2::nat)) \oplus_{Q_p} \mathfrak{p} \otimes (b[\widehat{}](2::nat))$ $= (y[\widehat{}](2::nat))$
proof−
assume *val b* $\geq$ *val a*
then have *ord b* $\geq$ *ord a*
using *F False*
by (*metis eint-ord-simps*(*1*) *val-def*)
then show $\exists\, y{\in}carrier\ Q_p.\ (a[\widehat{}](2::nat)) \oplus_{Q_p} \mathfrak{p} \otimes (b[\widehat{}](2::nat)) = (y[\widehat{}](2::nat))$
using *assms Qp-square-root-criterion*[*of a b*] *False F*
by *blast*
qed
show $\exists\, y{\in}carrier\ Q_p.(a[\widehat{}](2::nat)) \oplus_{Q_p} \mathfrak{p} \otimes (b[\widehat{}](2::nat)) = (y[\widehat{}](2::nat))$ $\Longrightarrow$ *val b* $\geq$ *val a*
proof−
assume $\exists\, y{\in}carrier\ Q_p.\ (a[\widehat{}](2::nat)) \oplus_{Q_p} \mathfrak{p} \otimes (b[\widehat{}](2::nat)) = (y[\widehat{}](2::nat))$
then obtain *y* **where** *y-def*: $y \in carrier\ Q_p \wedge (a[\widehat{}](2::nat)) \oplus_{Q_p} \mathfrak{p} \otimes (b[\widehat{}](2::nat)) = (y[\widehat{}](2::nat))$
by *blast*
have *0*: *ord* $(a[\widehat{}](2::nat)) =$ *2* ord a*
by (*metis* (*mono-tags*, *opaque-lifting*) *False Suc-1 assms*(*2*) *int-eq-iff-numeral nat-numeral nonzero-nat-pow-ord not-nonzero-Qp*)
have *1*: *ord* $(\mathfrak{p} \otimes (b[\widehat{}](2::nat))) = 1 + 2*$ *ord b*
proof−
have *0*: *ord* $(\mathfrak{p} \otimes (b[\widehat{}](2::nat)))$ = *ord* $\mathfrak{p}$ + *ord* $(b[\widehat{}](2::nat))$
using *F Qp-nat-pow-nonzero assms*(*3*) *not-nonzero-Qp ord-mult p-nonzero*
by *metis*
have *1*: *ord* $(b[\widehat{}](2::nat)) = 2*$ *ord b*
using *F assms*
by (*metis* (*mono-tags*, *opaque-lifting*) *Suc-1 int-eq-iff-numeral nat-numeral nonzero-nat-pow-ord not-nonzero-Qp*)
then show *?thesis*
using *0 ord-p*
by *linarith*
qed
show *val b* $\geq$ *val a*
proof(*rule ccontr*)
assume $\neg$ *val b* $\geq$ *val a*
then have *val b* $\neq$ *val a* $\wedge$ *val a* $\geq$ *val b*
by (*metis linear*)
then have *ord a* $>$ *ord b*
using *F False assms*
by (*metis* ‹$\neg$ *val a* $\leq$ *val b*› *eint-ord-simps*(*1*) *le-less not-less-iff-gr-or-eq val-def*)
then have *ord* $(a[\widehat{}](2::nat))$ $>$ *ord* $(\mathfrak{p} \otimes (b[\widehat{}](2::nat)))$
using *0 1*

```
          by linarith
           then have ord ((a[^](2::nat)) ⊕_{Q_p} 𝔭 ⊗ (b[^](2::nat))) = ord (𝔭 ⊗
(b[^](2::nat)))
             by (meson F False Qp.int-inc-closed Qp-nat-pow-nonzero assms(2)
assms(3)
         local.monom-term-car not-nonzero-Qp ord-ultrametric-noteq p-times-square-not-square')
         then have A0: ord (y[^](2::nat)) = 1 + 2* ord b
          by (metis 1 ‹y ∈ carrier Q_p ∧ (a[^]2) ⊕_{Q_p} 𝔭 ⊗ (b[^]2) = (y[^]2)›)
         have A1: (y[^](2::nat)) ∈ nonzero Q_p
          using y-def 0 1
        by (smt F False Qp.nonzero-closed Qp-nat-pow-nonzero assms(2) assms(3)
diff-ord-nonzero
         local.monom-term-car not-nonzero-Qp p-nonzero p-times-square-not-square')
         have A2: y ∈ nonzero Q_p
          using A1 Qp-nonzero-nat-pow pos2 y-def by blast
         have A3: ord (y[^](2::nat)) = 2* ord y
          using A2 nonzero-nat-pow-ord
          by presburger
         then show False using A0
          by presburger
       qed
     qed
   qed
  qed
qed
```

Defining the valuation ring semialgebraically for all primes

```
lemma Qp-val-ring-alt-def:
  assumes a ∈ carrier Q_p
  shows a ∈ 𝒪_p ⟷ (∃ y ∈ carrier Q_p. 𝟭 ⊕_{Q_p} (𝔭[^](3::nat))⊗ (a[^](4::nat)) =
(y[^](2::nat)))
proof(cases a = 𝟬)
  case True
  then have 𝟭 ⊕_{Q_p} (𝔭[^](3::nat))⊗ (a[^](4::nat)) = 𝟭
   by (metis Qp.add.l-cancel-one' Qp.integral-iff Qp.nat-pow-closed Qp.not-nonzero-memI
      Qp.one-closed Qp-nonzero-nat-pow assms not-nonzero-Qp p-natpow-closed(1)
zero-less-numeral)
  then have 𝟭 ⊕_{Q_p} (𝔭[^](3::nat))⊗ (a[^](4::nat)) = (𝟭[^](2::nat))
    using Qp.nat-pow-one by blast
  then show ?thesis
    using True zero-in-val-ring by blast
next
  case False
  show a ∈ 𝒪_p ⟷ (∃ y ∈ carrier Q_p. 𝟭 ⊕_{Q_p} (𝔭[^](3::nat))⊗ (a[^](4::nat)) =
(y[^](2::nat)))
  proof
    show a ∈ 𝒪_p ⟹ (∃ y ∈ carrier Q_p. 𝟭 ⊕_{Q_p} (𝔭[^](3::nat))⊗ (a[^](4::nat)) =
(y[^](2::nat)))
     using assms Qp-val-ring-alt-def0[of a] False
```

by (*meson not-nonzero-Qp ord-nonneg*)
show ($\exists\, y \in$ *carrier* Q_p. $\mathbf{1} \oplus_{Q_p} (\mathfrak{p}[\hat{}](3::nat)) \otimes (a[\hat{}](4::nat)) = (y[\hat{}](2::nat))) \Longrightarrow a \in \mathcal{O}_p$
proof −
assume ($\exists\, y \in$ *carrier* Q_p. $\mathbf{1} \oplus_{Q_p} (\mathfrak{p}[\hat{}](3::nat)) \otimes (a[\hat{}](4::nat)) = (y[\hat{}](2::nat))$)
then obtain *y* **where** *y-def*: $y \in$ *carrier* $Q_p \wedge \mathbf{1} \oplus_{Q_p} (\mathfrak{p}[\hat{}](3::nat)) \otimes (a[\hat{}](4::nat)) = (y[\hat{}](2::nat))$
by *blast*
then have $(\mathfrak{p}[\hat{}](3::nat)) \otimes (a[\hat{}](4::nat)) = (y[\hat{}](2::nat)) \ominus_{Q_p} \mathbf{1}$
using *Qp.ring-simprules*
by (*smt Qp.nat-pow-closed assms p-natpow-closed*(*1*))
then have *ord* $((\mathfrak{p}[\hat{}](3::nat)) \otimes (a[\hat{}](4::nat))) =$ *ord* $((y[\hat{}](2::nat)) \ominus_{Q_p} \mathbf{1})$
by *presburger*
then have *3* + *ord* $(a[\hat{}](4::nat)) =$ *ord* $((y[\hat{}](2::nat)) \ominus_{Q_p} \mathbf{1})$
by (*metis False Qp-nat-pow-nonzero assms not-nonzero-Qp of-nat-numeral ord-mult ord-p-pow-nat p-nonzero*)
then have *0*: *3* + *4*∗ *ord a* = *ord* $((y[\hat{}](2::nat)) \ominus_{Q_p} \mathbf{1})$
using *assms False nonzero-nat-pow-ord*[*of a* (*4*::*nat*)]
by (*metis nonzero-nat-pow-ord not-nonzero-Qp of-nat-numeral*)
have *ord a* ≥ *0*
proof(*rule ccontr*)
assume ¬ *0* ≤ *ord a*
then have *00*: *ord* $((y[\hat{}](2::nat)) \ominus_{Q_p} \mathbf{1}) <$ *0*
using *0*
by *linarith*
have *yn*: $y \in$ *nonzero* Q_p
apply(*rule ccontr*)
using *y-def 0*
by (*metis 00 Qp.not-eq-diff-nonzero Qp.one-closed Qp.one-nonzero Qp.pow-zero* ‹$\mathfrak{p}$ [^] *3* ⊗ *a* [^] *4* = *y* [^] *2* ⊖ **1**› *diff-ord-nonzero less-numeral-extra*(*3*) *local.one-neq-zero not-nonzero-Qp ord-one zero-less-numeral*)
then have *ord* $((y[\hat{}](2::nat)) \ominus_{Q_p} \mathbf{1}) =$ *ord* $(y[\hat{}](2::nat))$
using *y-def ord-ultrametric-noteq*′′[*of* $(y[\hat{}](2::nat))$ **1**]
by (*metis 00 False Qp.integral Qp.nat-pow-closed Qp.nonzero-closed Qp.nonzero-pow-nonzero Qp.not-eq-diff-nonzero Qp.one-nonzero Qp.r-right-minus-eq* ‹$\mathfrak{p}$ [^] *3* ⊗ *a* [^] *4* = *y* [^] *2* ⊖ **1**› *assms ord-one ord-ultrametric-noteq p-nonzero*)
then have *ord* $((y[\hat{}](2::nat)) \ominus_{Q_p} \mathbf{1}) =$ *2*∗ *ord y*
using *y-def Qp-nat-pow-nonzero Qp-nonzero-nat-pow nonzero-nat-pow-ord*[*of y* (*2*::*nat*)] *yn*
by *linarith*
then have *3* + (*4*∗ *ord a*) = *2*∗ *ord y*
using *00 0*
by *linarith*
then show *False*
by *presburger*
qed
then show $a \in \mathcal{O}_p$

using *False val-ring-ord-criterion assms* **by** *blast*
qed
qed
qed

lemma *Qp-val-alt-def*:
assumes $a \in$ *carrier* Q_p
assumes $b \in$ *carrier* Q_p
shows *val* $b \leq$ *val* $a \longleftrightarrow (\exists\, y \in$ *carrier* $Q_p.\ (b[\hat{}](4::nat)) \oplus_{Q_p} (\mathfrak{p}[\hat{}](3::nat)) \otimes (a[\hat{}](4::nat)) = (y[\hat{}](2::nat)))$
proof
show *val* $a \geq$ *val* $b \Longrightarrow \exists\, y \in$ *carrier* $Q_p.\ (b[\hat{}](4::nat)) \oplus_{Q_p} (\mathfrak{p}[\hat{}](3::nat)) \otimes (a[\hat{}](4::nat)) = (y[\hat{}](2::nat))$
proof−
assume *A*: *val* $a \geq$ *val* b
show $\exists\, y \in$ *carrier* $Q_p.\ (b[\hat{}](4::nat)) \oplus_{Q_p} (\mathfrak{p}[\hat{}](3::nat)) \otimes (a[\hat{}](4::nat)) = (y[\hat{}](2::nat))$
proof(*cases* $b = \mathbf{0}$)
case *True*
then have $a = \mathbf{0}$
using *A assms*(*1*) *val-ineq*
by *blast*
then have $(b[\hat{}](4::nat)) \oplus_{Q_p} (\mathfrak{p}[\hat{}](3::nat)) \otimes (a[\hat{}](4::nat)) = (\mathbf{0}[\hat{}](2::nat))$
by (*metis Qp.nat-pow-zero Qp.r-null Qp.r-zero True assms*(*2*) *p-natpow-closed*(*1*) *zero-neq-numeral*)
then show *?thesis*
using *True A assms*(*2*)
by *blast*
next
case *False*
assume *B*: $b \neq \mathbf{0}$
show $\exists\, y \in$ *carrier* $Q_p.\ (b[\hat{}](4::nat)) \oplus_{Q_p} (\mathfrak{p}[\hat{}](3::nat)) \otimes (a[\hat{}](4::nat)) = (y[\hat{}](2::nat))$
proof(*cases* $a = \mathbf{0}$)
case *True*
then have $(b[\hat{}](4::nat)) \oplus_{Q_p} (\mathfrak{p}[\hat{}](3::nat)) \otimes (a[\hat{}](4::nat)) = (b[\hat{}](4::nat))$
using *Qp.cring-axioms Qp.nat-pow-closed assms*(*2*) *cring-def p-natpow-closed*(*1*) *ring.pow-zero zero-less-numeral*
by (*metis Qp.add.l-cancel-one' Qp.integral-iff assms*(*1*))
then have $(b[\hat{}](4::nat)) \oplus_{Q_p} (\mathfrak{p}[\hat{}](3::nat)) \otimes (a[\hat{}](4::nat)) = ((b[\hat{}](2::nat))[\hat{}](2::nat))$
by (*metis Qp-nat-pow-pow assms*(*2*) *mult-2-right numeral-Bit0*)
then have $(b[\hat{}](2::nat)) \in$ *carrier* $Q_p \wedge (b[\hat{}](4::nat)) \oplus_{Q_p} (\mathfrak{p}[\hat{}](3::nat)) \otimes (a[\hat{}](4::nat)) = ((b[\hat{}](2::nat))[\hat{}]\ (2::nat))$
using *Qp.nat-pow-closed assms*(*2*)
by *blast*
then show *?thesis*
by *blast*

next
case *False*
have *F0*: $b \in nonzero\ Q_p$
using *B assms(2) not-nonzero-Qp*
by *metis*
have *F1*: $a \in nonzero\ Q_p$
using *False assms(1) not-nonzero-Qp*
by *metis*
then have $(a \div b) \in nonzero\ Q_p$
using *B*
by (*meson Localization.submonoid.m-closed Qp.nonzero-is-submonoid assms(2) inv-in-frac(3)*)
then have *val a* $\geq$ *val b*
using *F0 F1 A* **by** *blast*
then have *val* $(a \div b) \geq 0$
using *F0 F1 val-fract assms(1) local.eint-minus-ineq'* **by** *presburger*
obtain *y* **where** *y-def*: $y \in carrier\ Q_p \wedge \mathbf{1} \oplus_{Q_p} (\mathfrak{p}[\hat{}](3::nat)) \otimes ((a \div b)[\hat{}](4::nat)) = (y[\hat{}](2::nat))$
using *Qp-val-ring-alt-def0*
by (*meson B False Qp.integral Qp.nonzero-closed* ‹$(a \div b) \in nonzero\ Q_p$› ‹$0 \leq val\ (a \div b)$› *assms(1) assms(2) inv-in-frac(1) inv-in-frac(2) ord-nonneg val-ringI*)
then have $(b[\hat{}](4::nat)) \otimes (\mathbf{1} \oplus_{Q_p} (\mathfrak{p}[\hat{}](3::nat)) \otimes ((a \div b)[\hat{}](4::nat))) = (b[\hat{}](4::nat)) \otimes (y[\hat{}](2::nat))$
by *presburger*
then have *F2*: $(b[\hat{}](4::nat)) \otimes (\mathbf{1} \oplus_{Q_p} (\mathfrak{p}[\hat{}](3::nat)) \otimes ((a \div b)[\hat{}](4::nat))) = ((b[\hat{}](2::nat))\ [\hat{}]\ (2::nat)) \otimes (y[\hat{}](2::nat))$
by (*metis Qp.nat-pow-pow assms(2) mult-2-right numeral-Bit0*)
have *F3*: $((b[\hat{}](4::nat)) \otimes \mathbf{1}) \oplus_{Q_p} ((b[\hat{}](4::nat)) \otimes ((\mathfrak{p}[\hat{}](3::nat)) \otimes ((a \div b)[\hat{}](4::nat)))) = ((b[\hat{}](2::nat))[\hat{}]\ (2::nat)) \otimes (y[\hat{}](2::nat))$
proof−
have *0*: $(\mathfrak{p}[\hat{}](3::nat)) \otimes (a \div b[\hat{}](4::nat)) \in carrier\ Q_p$
proof−
have $(a \div b[\hat{}](4::nat)) \in carrier\ Q_p$
using *F0 Qp.nat-pow-closed assms(1) fract-closed Qp-nat-pow-nonzero* **by** *presburger*
then show *?thesis*
by (*meson Qp.cring-axioms cring.cring-simprules(5) p-natpow-closed(1)*)
qed
have *1*: $(b[\hat{}](4::nat)) \in carrier\ Q_p$
using *Qp.nat-pow-closed assms(2)*
by *blast*
then show *?thesis*
using *0 F2 ring.ring-simprules(23)*[*of* Q_p $\mathbf{1}$ $(\mathfrak{p}[\hat{}](3::nat)) \otimes ((a \div b)[\hat{}](4::nat))$ $(b[\hat{}](4::nat))$]
Qp.cring-axioms Qp.nonzero-mult-closed Qp.ring-axioms Qp-nat-pow-nonzero ‹$(a \div b) \in nonzero\ Q_p$› *p-nonzero*

by *blast*
qed
have *F4*: $(b[\hat{}](4::nat)) \in$ *carrier* Q_p
using *Qp.nat-pow-closed assms(2)*
by *blast*
then have $((b[\hat{}](4::nat)) \otimes \mathbf{1}) = (b[\hat{}](4::nat))$
using *Qp.r-one* **by** *blast*
then have *F5*: $(b[\hat{}](4::nat))\oplus_{Q_p} ((b[\hat{}](4::nat)) \otimes((\mathfrak{p}[\hat{}](3::nat))\otimes ((a \div b)[\hat{}](4::nat)))) =$
$((b[\hat{}](2::nat)) [\hat{}] (2::nat)) \otimes (y[\hat{}](2::nat))$
using *F3*
by *presburger*
have $((b[\hat{}](4::nat)) \otimes((\mathfrak{p}[\hat{}](3::nat))\otimes ((a \div b)[\hat{}](4::nat)))) = (\mathfrak{p}[\hat{}](3::nat))\otimes((b[\hat{}](4::nat)) \otimes ((a \div b)[\hat{}](4::nat)))$
proof−
have *0*: $(b[\hat{}](4::nat)) \in$ *carrier* Q_p
using *F4* **by** *blast*
have *1*: $(\mathfrak{p}[\hat{}](3::nat)) \in$ *carrier* Q_p
by *blast*
have *2*: $((a \div b)[\hat{}](4::nat)) \in$ *carrier* Q_p
using *F0 Qp.nat-pow-closed assms(1) fract-closed*
by *blast*
show *?thesis* **using** *0 1 2 monoid.m-assoc*[*of* Q_p] *comm-monoid.m-comm*[*of* Q_p]
using *Qp.m-lcomm* **by** *presburger*
qed
then have $(b[\hat{}](4::nat))\oplus_{Q_p} (\mathfrak{p}[\hat{}](3::nat))\otimes((b[\hat{}](4::nat)) \otimes ((a \div b)[\hat{}](4::nat))) =$
$((b[\hat{}](2::nat)) [\hat{}] (2::nat)) \otimes (y[\hat{}](2::nat))$
using *F5* **by** *presburger*
then have $(b[\hat{}](4::nat))\oplus_{Q_p} (\mathfrak{p}[\hat{}](3::nat))\otimes((b \otimes(a \div b))[\hat{}](4::nat)) =$
$((b[\hat{}](2::nat)) [\hat{}] (2::nat)) \otimes (y[\hat{}](2::nat))$
using *F0 Qp.nat-pow-distrib assms(1) assms(2) fract-closed* **by** *presburger*
then have $(b[\hat{}](4::nat))\oplus_{Q_p} (\mathfrak{p}[\hat{}](3::nat))\otimes(a[\hat{}](4::nat)) =$
$((b[\hat{}](2::nat)) [\hat{}] (2::nat)) \otimes (y[\hat{}](2::nat))$
by (*metis F0 assms(1) local.fract-cancel-right*)
then have $(b[\hat{}](4::nat))\oplus_{Q_p} (\mathfrak{p}[\hat{}](3::nat))\otimes(a[\hat{}](4::nat)) =$
$(((b[\hat{}](2::nat))\otimes y)[\hat{}](2::nat))$
using *Qp.nat-pow-closed Qp.nat-pow-distrib assms(2) y-def* **by** *blast*
then have $((b[\hat{}](2::nat))\otimes y) \in$ *carrier* $Q_p \wedge (b[\hat{}](4::nat))\oplus_{Q_p} (\mathfrak{p}[\hat{}](3::nat))\otimes(a[\hat{}](4::nat)) =$
$(((b[\hat{}](2::nat))\otimes y)[\hat{}](2::nat))$
by (*meson Qp.cring-axioms Qp.nat-pow-closed assms(2) cring.cring-simprules(5) y-def*)
then show *?thesis*
by *blast*
qed
qed
qed

show $\exists y \in$ *carrier* Q_p. $(b[\widehat{}](4::nat)) \oplus_{Q_p} (\mathfrak{p}[\widehat{}](3::nat)) \otimes (a[\widehat{}](4::nat)) = (y[\widehat{}](2::nat)) \Longrightarrow$ *val* $a \geq$ *val* b
proof −
assume *A*: $\exists y \in$ *carrier* Q_p. $(b[\widehat{}](4::nat)) \oplus_{Q_p} (\mathfrak{p}[\widehat{}](3::nat)) \otimes (a[\widehat{}](4::nat)) = (y[\widehat{}](2::nat))$
show *val* $a \geq$ *val* b
proof(*cases* $a = \mathbf{0}$)
case *True*
then show *?thesis*
using *eint-ord-code*(3) *local.val-zero* **by** *presburger*
next
case *False*
have $b \neq \mathbf{0}$
proof(*rule ccontr*)
assume $\neg\ b \neq \mathbf{0}$
then have $\exists y \in$ *carrier* Q_p. $(\mathfrak{p}[\widehat{}](3::nat)) \otimes (a[\widehat{}](4::nat)) = (y[\widehat{}](2::nat))$
using *A*
by (*metis* (*no-types*, *lifting*) *Qp.add.r-cancel-one'* *Qp.nat-pow-closed* *Qp.nonzero-memE*(2)
Qp-nonzero-nat-pow assms(1) *assms*(2) *local.monom-term-car not-nonzero-Qp p-natpow-closed*(1) *zero-less-numeral*)
then obtain *y* **where** *y-def*: $y \in$ *carrier* $Q_p \wedge (\mathfrak{p}[\widehat{}](3::nat)) \otimes (a[\widehat{}](4::nat)) = (y[\widehat{}](2::nat))$
by *blast*
have *0*: *ord* $((\mathfrak{p}[\widehat{}](3::nat)) \otimes (a[\widehat{}](4::nat))) = 3 + 4 *$ *ord a*
proof −
have *00*: $(\mathfrak{p}[\widehat{}](3::nat)) \in$ *nonzero* Q_p
using *Qp-nat-pow-nonzero p-nonzero* **by** *blast*
have *01*: $(a[\widehat{}](4::nat)) \in$ *nonzero* Q_p
using *False Qp-nat-pow-nonzero assms*(1) *not-nonzero-Qp Qp.nonzero-memI* **by** *presburger*
then show *?thesis* **using** *ord-mult*[*of* $\mathfrak{p}[\widehat{}](3::nat)$ $a[\widehat{}](4::nat)$]
by (*metis* (*no-types*, *lifting*) *00 False assms*(1) *nonzero-nat-pow-ord not-nonzero-Qp of-nat-numeral ord-p-pow-nat*)
qed
have *1*: *ord* $((\mathfrak{p}[\widehat{}](3::nat)) \otimes (a[\widehat{}](4::nat))) = 2 *$ (*ord y*)
proof −
have $y \neq \mathbf{0}$
proof(*rule ccontr*)
assume $\neg\ y \neq \mathbf{0}$
then have $(\mathfrak{p}[\widehat{}](3::nat)) \otimes (a[\widehat{}](4::nat)) = \mathbf{0}$
using *y-def* *Qp.cring-axioms cring-def pos2 ring.pow-zero* **by** *blast*
then show *False*
by (*metis False Qp.integral Qp.nat-pow-closed Qp.nonzero-pow-nonzero Qp.not-nonzero-memI Qp-nat-pow-nonzero assms*(1) *p-natpow-closed*(1) *p-nonzero*)
qed
then show *?thesis*
using *y-def*

by (*metis nonzero-nat-pow-ord not-nonzero-Qp of-nat-numeral*)
qed
then show *False*
using *0*
by *presburger*
qed
then have *F0*: $b \in nonzero\ Q_p$
using *assms*(*2*) *not-nonzero-Qp* **by** *metis*

have *F1*: $a \in nonzero\ Q_p$
using *False assms*(*1*) *not-nonzero-Qp* **by** *metis*
obtain *y* **where** *y-def*: $y \in carrier\ Q_p \wedge (b[\hat{}](4::nat)) \oplus_{Q_p} (\mathfrak{p}[\hat{}](3::nat)) \otimes (a[\hat{}](4::nat)) = (y[\hat{}](2::nat))$
using *A* **by** *blast*
show *?thesis*
proof(*rule ccontr*)
assume $\neg\ val\ a \geq val\ b$
then have *F2*: $ord\ a < ord\ b$
using *F0 F1 assms*
by (*metis False* ‹$b \neq \mathbf{0}$› *eint-ord-simps*(*1*) *leI val-def*)
have *0*: $ord\ ((\mathfrak{p}[\hat{}](3::nat)) \otimes (a[\hat{}](4::nat))) = 3 + 4 * ord\ a$
using *F0 ord-mult F1 Qp-nat-pow-nonzero nonzero-nat-pow-ord ord-p-pow-nat p-natpow-closed*(*2*)
by *presburger*
have *1*: $ord\ (b[\hat{}](4::nat)) = 4 * ord\ b$
using *F0 nonzero-nat-pow-ord*
by *presburger*
have *2*: $(4 * (ord\ b)) > 4 * (ord\ a)$
using *F2* **by** *linarith*
have *3*: $(4 * (ord\ b)) \leq 3 + 4 * ord\ a$
proof(*rule ccontr*)
assume $\neg\ (4 * (ord\ b)) \leq 3 + 4 * ord\ a$
then have $(4 * (ord\ b)) > 3 + 4 * ord\ a$
by *linarith*
then have *30*: $ord\ ((b[\hat{}](4::nat)) \oplus_{Q_p} (\mathfrak{p}[\hat{}](3::nat)) \otimes (a[\hat{}](4::nat))) = 3 + 4 * ord\ a$
using *0 1 F0 F1 Qp-nat-pow-nonzero Qp.nat-pow-closed assms*(*1*) *monom-term-car not-nonzero-Qp ord-ultrametric-noteq p-natpow-closed*(*1*) *p-nonzero*
by (*metis Qp.integral*)
have $y \in nonzero\ Q_p$
proof(*rule ccontr*)
assume *A*: $y \notin nonzero\ Q_p$
then have $y = \mathbf{0}$
using *y-def Qp.nonzero-memI* **by** *blast*
then have $b\ [\hat{}]\ 4 \oplus \mathfrak{p}\ [\hat{}]\ 3 \otimes a\ [\hat{}]\ 4 = \mathbf{0}$
by (*smt 0 1 A F0 False Qp.integral Qp.nat-pow-closed Qp.nonzero-closed Qp.nonzero-mult-closed Qp.nonzero-pow-nonzero Qp.pow-zero assms*(*1*) *diff-ord-nonzero not-nonzero-Qp p-nonzero pos2 y-def*)

```
      then show False
    by (smt 0 1 A F0 F1 Qp.integral Qp.nat-pow-closed Qp.nonzero-mult-closed
          Qp-nat-pow-nonzero assms(1) diff-ord-nonzero not-nonzero-Qp
p-natpow-closed(1) p-nonzero y-def)
    qed
    then have 31: ord ((b[^](4::nat)) ⊕_{Q_p} (𝔭[^](3::nat))⊗ (a[^](4::nat))) =
2∗ ord y
      using nonzero-nat-pow-ord y-def
      by presburger
    then show False using 30 by presburger
   qed
   show False
    using 2 3
    by presburger
  qed
 qed
qed
```

The polynomial in two variables which semialgebraically defines the valuation relation

```
definition Qp-val-poly where
Qp-val-poly = (pvar Q_p 1)[^]_{Q_p[X_2]}(4::nat) ⊕_{Q_p[X_2]} (𝔭[^](3::nat) ⊙_{Q_p[X_2]} ((pvar
Q_p 0)[^]_{Q_p[X_2]}(4::nat)))

lemma Qp-val-poly-closed:
Qp-val-poly ∈ carrier (Q_p[X_2])
proof−
  have (pvar Q_p 1) ∈ carrier (Q_p[X_2])
    using local.pvar-closed one-less-numeral-iff semiring-norm(76) by blast
  then have 0: (pvar Q_p 1)[^]_{Q_p[X_2]}(4::nat) ∈ carrier (Q_p[X_2])

    using  ring.Pring-is-ring[of Q_p {0::nat..2−1}]
       monoid.nat-pow-closed[of coord-ring Q_p 2] Qp.cring-axioms cring.axioms(1)
ring.Pring-is-monoid
    by blast
  have 1: (pvar Q_p 0)[^]_{Q_p[X_2]}(4::nat) ∈ carrier (Q_p[X_2])
    using local.pvar-closed pos2 by blast
  have 2: 𝔭[^](3::nat) ⊙_{Q_p[X_2]}(pvar Q_p 0)[^]_{Q_p[X_2]}(4::nat) ∈ carrier (Q_p[X_2])
    using 1 local.smult-closed p-natpow-closed(1) by blast
  then show ?thesis
    unfolding Qp-val-poly-def
    using 0 by blast
qed

lemma Qp-val-poly-eval:
  assumes a ∈ carrier Q_p
  assumes b ∈ carrier Q_p
 shows Qp-ev Qp-val-poly [a, b] = (b[^](4::nat)) ⊕_{Q_p} (𝔭[^](3::nat))⊗ (a[^](4::nat))
```

proof−
have *0*: $[a,b] \in$ *carrier* $(Q_p{}^2)$
proof(*rule cartesian-power-car-memI*)
show *length* $[a,\ b] = 2$
by *simp*
have *set* $[a,b] = \{a,b\}$
by *auto*
then show *set* $[a,\ b] \subseteq$ *carrier* Q_p
using *assms*
by (*simp add*: ‹$a \in$ *carrier* Q_p› ‹$b \in$ *carrier* Q_p›)
qed
obtain *f* **where** *f-def*: $f = ((pvar\ Q_p\ 1)[\hat{}]_{Q_p[\mathcal{X}_{\mathcal{2}}]}(4::nat))$
by *blast*
obtain *g* **where** *g-def*: $g = (\mathfrak{p}[\hat{}](3::nat) \odot_{Q_p[\mathcal{X}_{\mathcal{2}}]} ((pvar\ Q_p\ 0)[\hat{}]_{Q_p[\mathcal{X}_{\mathcal{2}}]}(4::nat)))$
by *blast*
have *1*: *Qp-val-poly* $= f \oplus_{Q_p[\mathcal{X}_{\mathcal{2}}]} g$
unfolding *Qp-val-poly-def*
using *f-def g-def* **by** *blast*
have *1*: *Qp-ev* $(pvar\ Q_p\ (0::nat))\ [a,b] = a$
using *eval-pvar*
by (*metis* ‹$[a,\ b] \in$ *carrier* $(Q_p{}^2)$› *nth-Cons-0 pos2*)
have *2*: *Qp-ev* $(pvar\ Q_p\ (1::nat))\ [a,b] = b$
using *eval-pvar*
by (*metis* (*no-types*, *lifting*) *0 One-nat-def add-diff-cancel-right′ assms*(*2*)
cartesian-power-car-memE gr-zeroI less-numeral-extra(*1*) *less-numeral-extra*(*4*)
list.size(*4*) *nth-Cons-pos Qp.to-R1-closed Qp.to-R-to-R1 zero-less-diff*)
have *3*: *Qp-ev* $((pvar\ Q_p\ 1)[\hat{}]_{Q_p[\mathcal{X}_{\mathcal{2}}]}(4::nat))\ [a,b] = (b[\hat{}](4::nat))$
by (*metis 0 2 eval-at-point-nat-pow local.pvar-closed one-less-numeral-iff semir-*
ing-norm(*76*))
have *4*: *Qp-ev* $((pvar\ Q_p\ 0)[\hat{}]_{Q_p[\mathcal{X}_{\mathcal{2}}]}(4::nat))\ [a,b] = (a[\hat{}](4::nat))$
using *0 1 eval-at-point-nat-pow local.pvar-closed pos2* **by** *presburger*
then have *5*: *Qp-ev* $(poly\text{-}scalar\text{-}mult\ Q_p\ (\mathfrak{p}[\hat{}](3::nat))\ ((pvar\ Q_p\ 0)[\hat{}]_{Q_p[\mathcal{X}_{\mathcal{2}}]}(4::nat)))$
$[a,b] = (\mathfrak{p}[\hat{}](3::nat)) \otimes (a[\hat{}](4::nat))$
using *eval-at-point-smult*[*of* $[a,b]$ *2* $(pvar\ Q_p\ 0)[\hat{}]_{Q_p[\mathcal{X}_{\mathcal{2}}]}(4::nat)\ \mathfrak{p}[\hat{}](3::nat)$
] *2*
by (*metis 0 MP.nat-pow-closed eval-at-point-scalar-mult local.pvar-closed p-natpow-closed*(*1*)
zero-less-numeral)
then show *?thesis*
proof−
have *00*: $[a,\ b] \in$ *carrier* $(Q_p{}^2)$
by (*simp add*: *0*)
have *01*: $pvar\ Q_p\ 1\ [\hat{}]_{Q_p[\mathcal{X}_{\mathcal{2}}]}\ (4::nat) \in$ *carrier* $(Q_p[\mathcal{X}_{\mathcal{2}}])$
by (*meson MP.nat-pow-closed local.pvar-closed one-less-numeral-iff semir-*
ing-norm(*76*))
have *02*: $\mathfrak{p}[\hat{}](3::nat) \odot_{Q_p[\mathcal{X}_{\mathcal{2}}]} (pvar\ Q_p\ 0\ [\hat{}]_{Q_p[\mathcal{X}_{\mathcal{2}}]}\ (4::nat)) \in$ *carrier*
$(Q_p[\mathcal{X}_{\mathcal{2}}])$
by (*meson MP.nat-pow-closed local.pvar-closed local.smult-closed p-natpow-closed*(*1*)
zero-less-numeral)

then show *?thesis*
unfolding *Qp-val-poly-def*
using *00 01 02*
by (*metis* (*no-types*, *lifting*) *3 4 MP.nat-pow-closed eval-at-point-add eval-at-point-smult local.pvar-closed p-natpow-closed*(*1*) *zero-less-numeral*)
qed
qed

lemma *Qp-2I*:
assumes *a* ∈ *carrier* Q_p
assumes *b* ∈ *carrier* Q_p
shows [*a*,*b*] ∈ *carrier* ($Q_p{}^2$)
apply(*rule cartesian-power-car-memI*)
using *assms*
apply (*simp add*: *assms*(*1*) *assms*(*2*))
using *assms*
by (*simp add*: *assms*(*1*) *assms*(*2*))

lemma *pair-id*:
assumes *length as* = *2*
shows *as* = [*as*!*0*, *as*!*1*]
using *assms*
by (*smt One-nat-def diff-Suc-1 length-Cons less-Suc0 less-SucE list.size*(*3*) *nth-Cons' nth-equalityI numeral-2-eq-2*)

lemma *Qp-val-semialg*:
assumes *a* ∈ *carrier* Q_p
assumes *b* ∈ *carrier* Q_p
shows *val b* ≤ *val a* ⟷ [*a*,*b*] ∈ *basic-semialg-set 2* (*2*::*nat*) *Qp-val-poly*
proof
show *val a* ≥ *val b* ⟹ [*a*, *b*] ∈ *basic-semialg-set 2 2 Qp-val-poly*
using *Qp-val-alt-def*[*of a b*] *Qp-2I*[*of a b*] *Qp-val-poly-eval*[*of a b*]
unfolding *basic-semialg-set-def*
by (*metis* (*mono-tags*, *lifting*) *assms*(*1*) *assms*(*2*) *mem-Collect-eq*)
show [*a*, *b*] ∈ *basic-semialg-set 2 2 Qp-val-poly* ⟹ *val a* ≥ *val b*
using *Qp-val-alt-def*[*of a b*] *Qp-2I*[*of a b*] *Qp-val-poly-eval*[*of a b*]
unfolding *basic-semialg-set-def*
using *assms*(*1*) *assms*(*2*)
by *blast*
qed

definition *val-relation-set* **where**
val-relation-set = {*as* ∈ *carrier* ($Q_p{}^2$). *val* (*as*!*1*) ≤ *val* (*as*!*0*)}

lemma *val-relation-setE*:
assumes *as* ∈ *val-relation-set*
shows *as*!*0* ∈ *carrier* Q_p ∧ *as*!*1* ∈ *carrier* Q_p ∧ *as* = [*as*!*0*,*as*!*1*] ∧ *val* (*as*!*1*) ≤ *val* (*as*!*0*)
using *assms* **unfolding** *val-relation-set-def*

by (*smt cartesian-power-car-memE cartesian-power-car-memE' mem-Collect-eq one-less-numeral-iff pair-id pos2 semiring-norm*(*76*))

lemma *val-relation-setI*:
assumes *as*!*0* ∈ *carrier* Q_p
assumes *as*!*1* ∈ *carrier* Q_p
assumes *length as* = *2*
assumes *val* (*as*!*1*) ≤ *val*(*as*!*0*)
shows *as* ∈ *val-relation-set*
unfolding *val-relation-set-def* **using** *assms Qp-2I*[*of as*!*0 as*!*1*]
by (*metis* (*no-types*, *lifting*) *mem-Collect-eq pair-id*)

lemma *val-relation-semialg*:
val-relation-set = *basic-semialg-set 2* (*2*::*nat*) *Qp-val-poly*
proof
show *val-relation-set* ⊆ *basic-semialg-set 2* (*2*::*nat*) *Qp-val-poly*
proof fix *as*
assume *A*: *as* ∈ *val-relation-set*
have *0*: *length as* = *2*
unfolding *val-relation-set-def*
by (*metis* (*no-types*, *lifting*) *A cartesian-power-car-memE mem-Collect-eq val-relation-set-def*)
have *1*: *as* = [*as* ! *0*, *as* ! *1*]
by (*metis* (*no-types*, *lifting*) *A cartesian-power-car-memE mem-Collect-eq pair-id val-relation-set-def*)
show *as* ∈ *basic-semialg-set 2* (*2*::*nat*) *Qp-val-poly*
using *A 1 val-relation-setE*[*of as*] *Qp-val-semialg*[*of as*!*0 as*!*1*]
by *presburger*
qed
show *basic-semialg-set 2* (*2*::*nat*) *Qp-val-poly* ⊆ *val-relation-set*
proof
fix *as*
assume *as* ∈ *basic-semialg-set 2* (*2*::*nat*) *Qp-val-poly*
then show *as* ∈ *val-relation-set*
using *val-relation-setI*[*of as*]
by (*smt cartesian-power-car-memE cartesian-power-car-memE' mem-Collect-eq one-less-numeral-iff Qp-val-semialg basic-semialg-set-def val-relation-set-def padic-fields-axioms pair-id pos2 semiring-norm*(*76*))
qed
qed

lemma *val-relation-is-semialgebraic*:
is-semialgebraic 2 val-relation-set
proof −
have {*rs* ∈ *carrier* ($Q_p{}^2$). *val* (*rs* ! *0*) ≥ *val* (*rs* ! *1*)} = *basic-semialg-set* (*Suc 1*) (*Suc 1*) *Qp-val-poly*
using *Suc-1 val-relation-semialg val-relation-set-def* **by** *presburger*
then show *?thesis*
by (*metis* (*no-types*) *Qp-val-poly-closed Suc-1 basic-semialg-is-semialgebraic'*

val-relation-set-def zero-neq-numeral)
qed

lemma *Qp-val-ring-is-semialg*:
obtains P **where** $P \in$ *carrier* Q_p*-x* $\wedge$ $\mathcal{O}_p$ = *univ-basic-semialg-set 2 P*
proof−
obtain P **where** *P-def*: $P = (\mathfrak{p}[\hat{}](3::nat)) \odot_{Q_p\text{-}x}$ (*X-poly* Q_p) $[\hat{}]_{Q_p\text{-}x}$ (4::*nat*) $\oplus_{Q_p\text{-}x}$ $\mathbf{1}_{Q_p\text{-}x}$
by *blast*
have *0*: $P \in$ *carrier* Q_p*-x*
proof−
have *0*: (*X-poly* Q_p) $\in$ *carrier* Q_p*-x*
using *UPQ.X-closed* **by** *blast*
then show *?thesis*
using *P-def UPQ.P.nat-pow-closed p-natpow-closed*(*1*) **by** *blast*
qed
have *1*: $\mathcal{O}_p$ = *univ-basic-semialg-set 2 P*
proof
show $\mathcal{O}_p \subseteq$ *univ-basic-semialg-set 2 P*
proof
fix x
assume *A*: $x \in \mathcal{O}_p$
show $x \in$ *univ-basic-semialg-set 2 P*
proof−
have *x-car*: $x \in$ *carrier* Q_p
using *A val-ring-memE* **by** *blast*
then have ($\exists y \in$ *carrier* Q_p. $\mathbf{1} \oplus_{Q_p} (\mathfrak{p}[\hat{}](3::nat)) \otimes (x[\hat{}](4::nat)) = (y[\hat{}](2::nat))$)
using *A Qp-val-ring-alt-def*[*of x*]
by *blast*
then obtain y **where** *y-def*: $y \in$ *carrier* $Q_p \wedge \mathbf{1} \oplus_{Q_p} (\mathfrak{p}[\hat{}](3::nat)) \otimes (x[\hat{}](4::nat)) = (y[\hat{}](2::nat))$
by *blast*
have $y \in$ *carrier* $Q_p \wedge P \cdot x = (y[\hat{}](2::nat))$
proof−
have $P \cdot x = \mathbf{1} \oplus_{Q_p} (\mathfrak{p}[\hat{}](3::nat)) \otimes (x[\hat{}](4::nat))$
proof−
have $((\mathfrak{p}[\hat{}](3::nat)) \odot_{Q_p\text{-}x}$ (*X-poly* Q_p) $[\hat{}]_{Q_p\text{-}x}$ (4::*nat*)) $\in$ *carrier* Q_p*-x*
using *UPQ.monom-closed p-natpow-closed*(*1*) **by** *blast*
then have $P \cdot x = (((\mathfrak{p}[\hat{}](3::nat)) \odot_{Q_p\text{-}x}$ (*X-poly* Q_p) $[\hat{}]_{Q_p\text{-}x}$ (4::*nat*))$\cdot x) \oplus_{Q_p} (\mathbf{1}_{Q_p\text{-}x} \cdot x)$
using *P-def x-car UPQ.to-fun-plus* **by** *blast*
then have *0*: $P \cdot x = (\mathfrak{p}[\hat{}](3::nat)) \otimes((\ (X\text{-}poly\ Q_p)\ [\hat{}]_{Q_p\text{-}x}\ (4::nat))\cdot x) \oplus_{Q_p} (\mathbf{1}_{Q_p\text{-}x} \cdot x)$
using *UPQ.P.nat-pow-closed UPQ.X-closed UPQ.to-fun-smult p-natpow-closed*(*1*) *x-car* **by** *presburger*
have $((\ (X\text{-}poly\ Q_p)\ [\hat{}]_{Q_p\text{-}x}\ (4::nat))\cdot x) = (x[\hat{}](4::nat))$
using *UPQ.to-fun-X-pow x-car* **by** *blast*
then have $P \cdot x = (\mathfrak{p}[\hat{}](3::nat)) \otimes(x[\hat{}](4::nat)) \oplus_{Q_p} \mathbf{1}$

```
        using 0 UPQ.to-fun-one x-car by presburger
      then show ?thesis
          using y-def Qp.add.m-comm Qp.one-closed local.monom-term-car
p-natpow-closed(1) x-car
        by presburger
    qed
    then show ?thesis
      using y-def
      by blast
  qed
  then show ?thesis
    unfolding univ-basic-semialg-set-def
    using x-car
    by blast
 qed
qed
show univ-basic-semialg-set 2 P ⊆ 𝒪_p
proof fix x
 assume A: x ∈ univ-basic-semialg-set (2::nat) P
 then obtain y where y-def: y ∈ carrier Q_p ∧ (P • x) = (y[^](2::nat))
  unfolding univ-basic-semialg-set-def
  by blast
 have x-car: x ∈ carrier Q_p
  using A
  by (metis (no-types, lifting) mem-Collect-eq univ-basic-semialg-set-def)
 have 0: (P • x) = (𝔭[^](3::nat)) ⊗ (x[^](4::nat)) ⊕_{Q_p} 1
 using P-def x-car UPQ.UP-one-closed UPQ.monom-closed UPQ.monom-rep-X-pow
UPQ.to-fun-monom
    UPQ.to-fun-one UPQ.to-fun-plus p-natpow-closed(1) by presburger
 have 1: y ∈ carrier Q_p ∧ (𝔭[^](3::nat)) ⊗ (x[^](4::nat)) ⊕_{Q_p} 1 = (y[^](2::nat))
  using 0 y-def
  by blast
 then show x ∈ 𝒪_p
  using x-car Qp-val-ring-alt-def[of x] y-def
 by (metis Qp.add.m-comm Qp.one-closed local.monom-term-car p-natpow-closed(1))
qed
qed
show ?thesis
 using 0 1 that
 by blast
qed

lemma Qp-val-ring-is-univ-semialgebraic:
is-univ-semialgebraic 𝒪_p
proof−
 obtain P where P ∈ carrier Q_p-x ∧ 𝒪_p = univ-basic-semialg-set 2 P
  using Qp-val-ring-is-semialg by blast
 then show ?thesis
  by (metis univ-basic-semialg-set-is-univ-semialgebraic zero-neq-numeral)
```

qed

lemma *Qp-val-ring-is-semialgebraic*:
is-semialgebraic 1 (*to-R1*‘ $\mathcal{O}_p$)
using *Qp-val-ring-is-univ-semialgebraic is-univ-semialgebraic-def* **by** *blast*

13.5.6 Inverse Images of Semialgebraic Sets by Polynomial Maps

lemma *basic-semialg-pullback*:
assumes $f \in$ *carrier* ($Q_p[\mathcal{X}_k]$)
assumes *is-poly-tuple n fs*
assumes *length fs* $= k$
assumes $S =$ *basic-semialg-set k m f*
assumes $m \neq 0$
shows *poly-map n fs* $^{-1}_n$ $S =$ *basic-semialg-set n m* (*Qp-poly-comp n fs f*)
proof
show *poly-map n fs* $^{-1}_n$ $S \subseteq$ *basic-semialg-set n m* (*Qp-poly-comp n fs f*)
proof
fix x
assume *A*: $x \in$ *poly-map n fs* $^{-1}_n$ S
then have *0*: *poly-map n fs* $x \in S$
proof −
have $\exists\, n\, f.\ \{rs.\ rs \in S\} \subseteq \{rs \in$ *carrier* ($Q_p{}^k$). $\exists\, r.\ r \in$ *carrier* $Q_p \wedge$ *Qp-ev f rs* $= (r[\widehat{}](n::nat))\}$
by (*metis* (*no-types*) *Collect-mem-eq* ‹$S =$ *basic-semialg-set k m f*› *basic-semialg-set-def eq-iff*)
then show *?thesis*
using *A* **by** *blast*
qed
have *1*: $x \in$ *carrier* ($Q_p{}^n$)
using *A assms*
by (*meson evimage-eq*)
have $\exists\, y \in$ (*carrier* Q_p). *Qp-ev f* (*poly-map n fs x*) $= (y[\widehat{}]m)$
using *A 0 assms basic-semialg-set-def*
by *blast*
then have $\exists\, y \in$ (*carrier* Q_p). *Qp-ev* (*Qp-poly-comp n fs f*) $x = (y[\widehat{}]m)$
using *1 assms Qp-poly-comp-eval*
by *blast*
then show $x \in$ *basic-semialg-set n m* (*Qp-poly-comp n fs f*)
using *1 basic-semialg-set-def*
by *blast*
qed
show *basic-semialg-set n m* (*Qp-poly-comp n fs f*) $\subseteq$ *poly-map n fs* $^{-1}_n$ S
proof fix x
assume *A*: $x \in$ *basic-semialg-set n m* (*Qp-poly-comp n fs f*)
have *0*: $x \in$ *carrier* ($Q_p{}^n$)
using *A basic-semialg-set-def*
by *blast*
have *1*: (*poly-map n fs x*) $\in$ *carrier* ($Q_p{}^k$)

using *0 poly-map-closed assms(2) assms(3)* **by** *blast*
show $x \in$ *poly-map n fs* $^{-1}_n$ *S*
proof−
have $\exists\, y \in$ *carrier* Q_p. *Qp-ev* (*Qp-poly-comp n fs f*) *x* = (*y*[^]*m*)
using *A basic-semialg-set-def*
by *blast*
then have *2*: $\exists\, y \in$ *carrier* Q_p. *Qp-ev f* (*poly-map n fs x*) = (*y*[^]*m*)
using *assms Qp-poly-comp-eval*
by (*metis* (*no-types*, *lifting*) *A basic-semialg-set-def mem-Collect-eq*)
have *3*: *poly-map n fs x* ∈ *S*
using *assms 0 1 basic-semialg-set-def*[*of k m f*] *2*
by *blast*
show *?thesis*
using *0 3* **by** *blast*
qed
qed
qed

lemma *basic-semialg-pullback′*:
assumes *is-poly-tuple n fs*
assumes *length fs* = *k*
assumes *A* ∈ *basic-semialgs k*
shows *poly-map n fs* $^{-1}_n$ *A* ∈ (*basic-semialgs n*)
proof−
obtain *f m* **where** *fm-def*: *m* ≠*0* ∧*f* ∈ *carrier* (*Q*$_p$[$\mathcal{X}_k$]) ∧ *A* = *basic-semialg-set k m f*
using *assms*
by (*metis is-basic-semialg-def mem-Collect-eq*)
then have *poly-map n fs* $^{-1}_n$ *A* = *basic-semialg-set n m* (*Qp-poly-comp n fs f*)
using *assms basic-semialg-pullback*[*of f k n fs A m*]
by *linarith*
then show *?thesis* **unfolding** *is-basic-semialg-def*
by (*metis* (*mono-tags*, *lifting*) *assms(1) assms(2) fm-def mem-Collect-eq poly-compose-closed*)
qed

lemma *semialg-pullback*:
assumes *is-poly-tuple n fs*
assumes *length fs* = *k*
assumes *S* ∈ *semialg-sets k*
shows *poly-map n fs* $^{-1}_n$ *S* ∈ *semialg-sets n*
unfolding *semialg-sets-def*
apply(*rule gen-boolean-algebra.induct*[*of S* (*carrier* (*Q*$_p{}^k$)) *basic-semialgs k*])
using *assms semialg-sets-def* **apply** *blast*
apply (*metis assms(1) assms(2) carrier-is-semialgebraic evimageI2 extensional-vimage-closed is-semialgebraicE poly-map-closed semialg-sets-def subsetI subset-antisym*)
apply (*metis Int-absorb2 assms(1) assms(2) basic-semialg-is-semialg basic-semialg-is-semialgebraic basic-semialg-pullback′ is-semialgebraic-closed mem-Collect-eq semialg-sets-def*)
apply (*metis evimage-Un semialg-sets-def semialg-union*)
by (*metis assms(1) assms(2) carrier-is-semialgebraic diff-is-semialgebraic evim-*

age-Diff extensional-vimage-closed is-semialgebraicE is-semialgebraicI poly-map-closed poly-map-pullbackI semialg-sets-def subsetI subset-antisym)

lemma *pullback-is-semialg*:
assumes *is-poly-tuple n fs*
assumes *length fs* = *k*
assumes $S \in$ *semialg-sets k*
shows *is-semialgebraic n* (*poly-map n fs* $^{-1}_n$ *S*)
using *assms*(*1*) *assms*(*2*) *assms*(*3*) *is-semialgebraicI padic-fields-axioms semialg-pullback*
by *blast*

Equality and inequality sets for a pair of polynomials

definition *val-ineq-set* **where**
val-ineq-set n f g = $\{x \in$ *carrier* $(Q_p{}^n)$. *val* (*Qp-ev f x*) $\leq$ *val* (*Qp-ev g x*)$\}$

lemma *poly-map-length* :
assumes *length fs* = *m*
assumes *as* $\in$ *carrier* $(Q_p{}^n)$
shows *length* (*poly-map n fs as*) = *m*
using *assms* **unfolding** *poly-map-def poly-tuple-eval-def*
by (*metis* (*no-types*, *lifting*) *length-map restrict-apply'*)

lemma *val-ineq-set-pullback*:
assumes $f \in$ *carrier* $(Q_p[\mathcal{X}_n])$
assumes $g \in$ *carrier* $(Q_p[\mathcal{X}_n])$
shows *val-ineq-set n f g* = *poly-map n* [*g*,*f*] $^{-1}_n$ *val-relation-set*
proof
show *val-ineq-set n f g* $\subseteq$ *poly-map n* [*g*,*f*] $^{-1}_n$ *val-relation-set*
proof
fix *x*
assume *x* $\in$ *val-ineq-set n f g*
then have *0*: *x* $\in$ *carrier* $(Q_p{}^n) \wedge$ *val* (*Qp-ev f x*) $\leq$ *val* (*Qp-ev g x*)
by (*metis* (*mono-tags*, *lifting*) *mem-Collect-eq val-ineq-set-def*)
have *1*: *poly-map n* [*g*,*f*] *x* = [*Qp-ev g x*, *Qp-ev f x*]
unfolding *poly-map-def poly-tuple-eval-def* **using** *0*
by (*metis* (*no-types*, *lifting*) *Cons-eq-map-conv list.simps*(*8*) *restrict-apply'*)
have *2*: *poly-map n* [*g*,*f*] *x* $\in$ *val-relation-set*
apply(*rule val-relation-setI*)
using *1 0 assms* **apply** (*metis eval-at-point-closed nth-Cons-0*)
using *1 0 assms* **apply** (*metis One-nat-def eval-at-point-closed diff-Suc-1 less-numeral-extra*(*1*) *nth-Cons-pos Qp.to-R-to-R1*)
using *poly-map-length assms 0* **apply** (*metis 1 Qp-2I cartesian-power-car-memE eval-at-point-closed*)
by (*metis 0 1 One-nat-def nth-Cons-0 nth-Cons-Suc*)
have *3*: *is-poly-tuple n* [*g*, *f*]
using *assms*
by (*smt One-nat-def diff-Suc-1 Qp-is-poly-tupleI length-Suc-conv less-SucE less-one list.size*(*3*) *nth-Cons'*)

then show $x \in$ *poly-map n* $[g,f]$ $^{-1}_n$ *val-relation-set*
using *0 1 2*
by *blast*
qed
show *poly-map n* $[g,f]$ $^{-1}_n$ *val-relation-set* $\subseteq$ *val-ineq-set n f g*
proof fix x
have *0*: *is-poly-tuple n* $[g, f]$
using *Qp-is-poly-tupleI assms*
by (*metis* (*no-types*, *lifting*) *diff-Suc-1 length-Cons less-Suc0 less-SucE list.size*(*3*) *nth-Cons'*)
assume *A*: $x \in$ *poly-map n* $[g,f]$ $^{-1}_n$ *val-relation-set*
then have *1*: $x \in$ *carrier* $({Q_p}^n) \wedge$ *poly-map n* $[g,f]$ $x \in$ *val-relation-set*
using *0*
by (*meson evimageD extensional-vimage-closed subsetD*)
have *2*: *poly-map n* $[g,f]$ $x = [$*Qp-ev g x*, *Qp-ev f x*$]$
by (*metis 1 Qp-poly-mapE' length-0-conv poly-map-cons*)
show $x \in$ *val-ineq-set n f g*
using *0 1 2* **unfolding** *val-ineq-set-def val-relation-set-def*
by (*metis* (*no-types*, *lifting*) *1 list.inject mem-Collect-eq nth-Cons-0 poly-map-apply val-relation-setE*)
qed
qed

lemma *val-ineq-set-is-semialg*:
assumes $f \in$ *carrier* $(Q_p[\mathcal{X}_n])$
assumes $g \in$ *carrier* $(Q_p[\mathcal{X}_n])$
shows *val-ineq-set n f g* $\in$ *semialg-sets n*
proof−
have *0*: *val-relation-set* $\in$ *semialg-sets 2*
using *val-relation-semialg basic-semialg-is-semialg'*
by (*metis Qp-val-poly-closed zero-neq-numeral*)
show *?thesis* **using** *val-ineq-set-pullback semialg-pullback*[*of n* $[g,f]$ *2 val-relation-set*]
by (*metis* (*no-types*, *lifting*) *0 assms*(*1*) *assms*(*2*) *diff-Suc-1 Qp-is-poly-tupleI length-Cons less-Suc0 less-SucE list.size*(*3*) *nth-Cons-0 nth-Cons-pos numeral-2-eq-2 zero-neq-numeral*)
qed

lemma *val-ineq-set-is-semialgebraic*:
assumes $f \in$ *carrier* $(Q_p[\mathcal{X}_n])$
assumes $g \in$ *carrier* $(Q_p[\mathcal{X}_n])$
shows *is-semialgebraic n* (*val-ineq-set n f g*)
using *assms*(*1*) *assms*(*2*) *is-semialgebraicI val-ineq-set-is-semialg* **by** *blast*

lemma *val-ineq-setI*:
assumes $f \in$ *carrier* $(Q_p[\mathcal{X}_n])$
assumes $g \in$ *carrier* $(Q_p[\mathcal{X}_n])$
assumes $x \in$ (*val-ineq-set n f g*)

shows $x \in$ *carrier* $({Q_p}^n)$
$\quad$ *val* $(Qp\text{-}ev\ f\ x) \leq$ *val* $(Qp\text{-}ev\ g\ x)$
using *assms* **unfolding** *val-ineq-set-def* **apply** *blast*
using *assms* **unfolding** *val-ineq-set-def* **by** *blast*

lemma *val-ineq-setE*:
assumes $f \in$ *carrier* $(Q_p[\mathcal{X}_n])$
assumes $g \in$ *carrier* $(Q_p[\mathcal{X}_n])$
assumes $x \in$ *carrier* $({Q_p}^n)$
assumes $\quad$ *val* $(Qp\text{-}ev\ f\ x) \leq$ *val* $(Qp\text{-}ev\ g\ x)$
shows $x \in$ (*val-ineq-set* $n\ f\ g$)
using *assms* **unfolding** *val-ineq-set-def*
by *blast*

lemma *val-ineq-set-is-semialgebraic'*:
assumes $f \in$ *carrier* $(Q_p[\mathcal{X}_n])$
assumes $g \in$ *carrier* $(Q_p[\mathcal{X}_n])$
shows *is-semialgebraic* n $\{x \in$ *carrier* $({Q_p}^n)$. *val* $(Qp\text{-}ev\ f\ x) \leq$ *val* $(Qp\text{-}ev\ g\ x)\}$
using *assms val-ineq-set-is-semialgebraic* **unfolding** *val-ineq-set-def* **by** *blast*

lemma *val-eq-set-is-semialgebraic*:
assumes $f \in$ *carrier* $(Q_p[\mathcal{X}_n])$
assumes $g \in$ *carrier* $(Q_p[\mathcal{X}_n])$
shows *is-semialgebraic* n $\{x \in$ *carrier* $({Q_p}^n)$. *val* $(Qp\text{-}ev\ f\ x) =$ *val* $(Qp\text{-}ev\ g\ x)\}$
proof−
have *0*: *is-semialgebraic* n $\{x \in$ *carrier* $({Q_p}^n)$. *val* $(Qp\text{-}ev\ f\ x) \leq$ *val* $(Qp\text{-}ev\ g\ x)\}$
using *assms val-ineq-set-is-semialgebraic* **unfolding** *val-ineq-set-def*
by *blast*
have *1*: *is-semialgebraic* n $\{x \in$ *carrier* $({Q_p}^n)$. *val* $(Qp\text{-}ev\ g\ x) \leq$ *val* $(Qp\text{-}ev\ f\ x)\}$
using *assms val-ineq-set-is-semialgebraic* **unfolding** *val-ineq-set-def*
by *blast*
have *2*: $\{x \in$ *carrier* $({Q_p}^n)$. *val* $(Qp\text{-}ev\ f\ x) =$ *val* $(Qp\text{-}ev\ g\ x)\} = \{x \in$ *carrier* $({Q_p}^n)$. *val* $(Qp\text{-}ev\ f\ x) \leq$ *val* $(Qp\text{-}ev\ g\ x)\} \cap$
$\{x \in$ *carrier* $({Q_p}^n)$. *val* $(Qp\text{-}ev\ g\ x) \leq$ *val* $(Qp\text{-}ev\ f\ x)\}$
apply(*rule equalityI*, *rule subsetI* , *rule IntI*) **unfolding** *mem-Collect-eq*
using *le-less* **apply** *blast* **apply** (*metis order-refl*)
apply(*rule subsetI*, *erule IntE*) **unfolding** *mem-Collect-eq*
by (*meson less-le-trans not-less-iff-gr-or-eq*)
show *?thesis* **unfolding** *2* **apply**(*rule intersection-is-semialg*)
using *0* **apply** *blast* **using** *1* **by** *blast*
qed

lemma *equalityI''*:
assumes $\bigwedge x.\ A\ x \Longrightarrow B\ x$
assumes $\bigwedge x.\ B\ x \Longrightarrow A\ x$
shows $\{x.\ A\ x\} = \{x.\ B\ x\}$

using *assms* **by** *blast*

lemma *val-strict-ineq-set-is-semialgebraic*:
assumes $f \in$ *carrier* $(Q_p[\mathcal{X}_n])$
assumes $g \in$ *carrier* $(Q_p[\mathcal{X}_n])$
shows *is-semialgebraic n* $\{x \in$ *carrier* $({Q_p}^n)$. *val* (*Qp-ev f x*) $<$ *val* (*Qp-ev g x*)$\}$
proof−
have *0*: $\{x \in$ *carrier* $({Q_p}^n)$. *val* (*Qp-ev f x*) $<$ *val* (*Qp-ev g x*)$\}$ =
$\{x \in$ *carrier* $({Q_p}^n)$. *val* (*Qp-ev f x*) $\leq$ *val* (*Qp-ev g x*)$\}$ − $\{x \in$ *carrier* $({Q_p}^n)$. *val* (*Qp-ev f x*) = *val* (*Qp-ev g x*)$\}$
apply(*rule equalityI'*, *rule DiffI*) **unfolding** *le-less mem-Collect-eq* **apply** *blast*
unfolding *mem-Collect-eq* **using** *neq-iff* **apply** *blast*
apply(*erule DiffE*) **unfolding** *mem-Collect-eq* **by** *blast*
show *?thesis* **unfolding** *0*
apply(*rule diff-is-semialgebraic*)
using *assms val-ineq-set-is-semialgebraic*[*of f n g*] **unfolding** *val-ineq-set-def* **apply** *blast*
using *assms val-eq-set-is-semialgebraic*[*of f n g*] **unfolding** *val-ineq-set-def* **by** *blast*
qed

lemma *constant-poly-val-exists*:
shows $\exists\, g \in$ *carrier* $(Q_p[\mathcal{X}_n])$. $(\forall\ x \in$ *carrier* $({Q_p}^n)$. *val* (*Qp-ev g x*) = *c*)
proof−
obtain *a* **where** *a-def*: $a \in$ *carrier* $Q_p \wedge$ *val a* = *c*
by (*meson Qp.minus-closed Qp.nonzero-closed dist-nonempty' p-nonzero*)
obtain *g* **where** *g-def*: *g* = *coord-const a*
by *blast*
show *?thesis* **using** *a-def g-def Qp-to-IP-car*
by (*metis* (*no-types*, *opaque-lifting*) *Qp-to-IP-car a-def eval-at-point-const g-def le-less subset-iff*)
qed

lemma *val-ineq-set-is-semialgebraic''*:
assumes $f \in$ *carrier* $(Q_p[\mathcal{X}_n])$
shows *is-semialgebraic n* $\{x \in$ *carrier* $({Q_p}^n)$. *val* (*Qp-ev f x*) $\leq$ *c*$\}$
proof−
obtain *g* **where** *g-def*: $g \in$ *carrier* $(Q_p[\mathcal{X}_n]) \wedge (\forall\ x \in$ *carrier* $({Q_p}^n)$. *val* (*Qp-ev g x*) = *c*)
using *constant-poly-val-exists* **by** *blast*
have *0*: *is-semialgebraic n* $\{x \in$ *carrier* $({Q_p}^n)$. *val* (*Qp-ev f x*) $\leq$ *val* (*Qp-ev g x*)$\}$
apply(*rule val-ineq-set-is-semialgebraic'*)
using *assms* **apply** *blast* **using** *g-def* **by** *blast*
have *1*: $\{x \in$ *carrier* $({Q_p}^n)$. *val* (*Qp-ev f x*) $\leq$ *val* (*Qp-ev g x*)$\}$ = $\{x \in$ *carrier* $({Q_p}^n)$. *val* (*Qp-ev f x*) $\leq$ *c*$\}$
apply(*rule equalityI''*) **using** *g-def* **apply** *fastforce* **using** *g-def* **by** *fastforce*
show *?thesis* **using** *0* **unfolding** *1* **by** *blast*
qed

lemma *val-ineq-set-is-semialgebraic'''*:
assumes $f \in$ *carrier* $(Q_p[\mathcal{X}_n])$
shows *is-semialgebraic n* $\{x \in$ *carrier* $({Q_p}^n).$ $c \leq$ *val* (*Qp-ev f x*)$\}$
proof−
obtain *g* **where** *g-def*: $g \in$ *carrier* $(Q_p[\mathcal{X}_n]) \wedge (\forall\ x \in$ *carrier* $({Q_p}^n).$ *val* (*Qp-ev g x*) $= c)$
using *constant-poly-val-exists* **by** *blast*
have *0*: *is-semialgebraic n* $\{x \in$ *carrier* $({Q_p}^n).$ *val* (*Qp-ev g x*) $\leq$ *val* (*Qp-ev f x*)$\}$
apply(*rule val-ineq-set-is-semialgebraic'*)
using *g-def* **apply** *blast* **using** *assms* **by** *blast*
have *1*: $\{x \in$ *carrier* $({Q_p}^n).$ *val* (*Qp-ev g x*) $\leq$ *val* (*Qp-ev f x*)$\} = \{x \in$ *carrier* $({Q_p}^n).$ $c \leq$ *val* (*Qp-ev f x*)$\}$
apply(*rule equalityI''*) **using** *g-def* **apply** *fastforce* **using** *g-def* **by** *fastforce*
show *?thesis* **using** *0* **unfolding** *1* **by** *blast*
qed

lemma *val-eq-set-is-semialgebraic'*:
assumes $f \in$ *carrier* $(Q_p[\mathcal{X}_n])$
shows *is-semialgebraic n* $\{x \in$ *carrier* $({Q_p}^n).$ *val* (*Qp-ev f x*) $= c\}$
proof−
obtain *g* **where** *g-def*: $g \in$ *carrier* $(Q_p[\mathcal{X}_n]) \wedge (\forall\ x \in$ *carrier* $({Q_p}^n).$ *val* (*Qp-ev g x*) $= c)$
using *constant-poly-val-exists* **by** *blast*
have *0*: *is-semialgebraic n* $\{x \in$ *carrier* $({Q_p}^n).$ *val* (*Qp-ev f x*) $=$ *val* (*Qp-ev g x*)$\}$
apply(*rule val-eq-set-is-semialgebraic*)
using *assms* **apply** *blast* **using** *g-def* **by** *blast*
have *1*: $\{x \in$ *carrier* $({Q_p}^n).$ *val* (*Qp-ev f x*) $=$ *val* (*Qp-ev g x*)$\} = \{x \in$ *carrier* $({Q_p}^n).$ *val* (*Qp-ev f x*) $= c\}$
apply(*rule equalityI''*) **using** *g-def* **apply** *fastforce* **using** *g-def* **by** *metis*
show *?thesis* **using** *0* **unfolding** *1* **by** *blast*
qed

lemma *val-strict-ineq-set-is-semialgebraic'*:
assumes $f \in$ *carrier* $(Q_p[\mathcal{X}_n])$
shows *is-semialgebraic n* $\{x \in$ *carrier* $({Q_p}^n).$ *val* (*Qp-ev f x*) $<$ $c\}$
proof−
obtain *g* **where** *g-def*: $g \in$ *carrier* $(Q_p[\mathcal{X}_n]) \wedge (\forall\ x \in$ *carrier* $({Q_p}^n).$ *val* (*Qp-ev g x*) $= c)$
using *constant-poly-val-exists* **by** *blast*
have *0*: *is-semialgebraic n* $\{x \in$ *carrier* $({Q_p}^n).$ *val* (*Qp-ev f x*) $<$ *val* (*Qp-ev g x*)$\}$
apply(*rule val-strict-ineq-set-is-semialgebraic*)
using *assms* **apply** *blast* **using** *g-def* **by** *blast*
have *1*: $\{x \in$ *carrier* $({Q_p}^n).$ *val* (*Qp-ev f x*) $<$ *val* (*Qp-ev g x*)$\} = \{x \in$ *carrier* $({Q_p}^n).$ *val* (*Qp-ev f x*) $<$ $c\}$
apply(*rule equalityI''*) **using** *g-def* **apply** *fastforce* **using** *g-def*

by *fastforce*
show *?thesis* **using** *0 g-def* **unfolding** *1*
by *blast*
qed

lemma *val-strict-ineq-set-is-semialgebraic''*:
assumes $f \in$ *carrier* $(Q_p[\mathcal{X}_n])$
shows *is-semialgebraic n* $\{x \in$ *carrier* $({Q_p}^n)$. $c <$ *val* (*Qp-ev f x*)$\}$
proof−
obtain *g* **where** *g-def*: $g \in$ *carrier* $(Q_p[\mathcal{X}_n]) \wedge (\forall\ x \in$ *carrier* $({Q_p}^n)$. *val* (*Qp-ev g x*) $= c)$
using *constant-poly-val-exists* **by** *blast*
have *0*: *is-semialgebraic n* $\{x \in$ *carrier* $({Q_p}^n)$. *val* (*Qp-ev g x*) $<$ *val* (*Qp-ev f x*)$\}$
apply(*rule val-strict-ineq-set-is-semialgebraic*)
using *g-def* **apply** *blast* **using** *assms* **by** *blast*
have *1*: $\{x \in$ *carrier* $({Q_p}^n)$. *val* (*Qp-ev g x*) $<$ *val* (*Qp-ev f x*)$\} = \{x \in$ *carrier* $({Q_p}^n)$. $c <$ *val* (*Qp-ev f x*)$\}$
apply(*rule equalityI''*) **using** *assms g-def* **apply** *fastforce* **using** *assms g-def* **by** *fastforce*
show *?thesis* **using** *0 g-def* **unfolding** *1*
by *blast*
qed

lemma(**in** *cring*) *R1-memE*:
assumes $x \in$ *carrier* (R^1)
shows $x = [(hd\ x)]$
using *assms cartesian-power-car-memE*
by (*metis diff-is-0-eq' hd-conv-nth le-eq-less-or-eq length-0-conv length-tl list.exhaust list.sel*(*3*) *normalize.cases nth-Cons-0 zero-neq-one*)

lemma(**in** *cring*) *R1-memE'*:
assumes $x \in$ *carrier* (R^1)
shows *hd x* $\in$ *carrier R*
using *R1-memE assms cartesian-power-car-memE*[*of x R 1*] *cartesian-power-car-memE'*[*of x R 1 0*]
by (*metis hd-conv-nth less-numeral-extra*(*1*) *list.size*(*3*) *zero-neq-one*)

lemma *univ-val-ineq-set-is-univ-semialgebraic*:
is-univ-semialgebraic $\{x \in$ *carrier* Q_p. *val* $x \leq c\}$
proof−
have *0*: *is-semialgebraic 1* $\{x \in$ *carrier* $({Q_p}^1)$. *val* (*Qp-ev* (*pvar* Q_p *0*) *x*) $\leq c\}$
apply(*rule val-ineq-set-is-semialgebraic''*)
using *pvar-closed* **by** *blast*
have *1*: $\{x \in$ *carrier* $({Q_p}^1)$. *val* (*Qp-ev* (*pvar* Q_p *0*) *x*) $\leq c\} =$ *to-R1* ‘ $\{x \in$ *carrier* Q_p. *val* $x \leq c\}$
proof(*rule equalityI'*)
show $\bigwedge x.\ x \in \{x \in$ *carrier* $({Q_p}^1)$. *val* (*eval-at-point* Q_p *x* (*pvar* Q_p *0*)) $\leq c\} \Longrightarrow x \in (\lambda a.\ [a])$ ‘ $\{x \in$ *carrier* Q_p. *val* $x \leq c\}$

proof− **fix** x **assume** A: $x \in \{x \in carrier\ (Q_p{}^1).\ val\ (eval\text{-}at\text{-}point\ Q_p\ x\ (pvar\ Q_p\ 0)) \leq c\}$
then have *0*: $x = [(hd\ x)] \wedge hd\ x \in carrier\ Q_p$
using *Qp.R1-memE Qp.R1-memE'* **by** *blast*
have *1*: $eval\text{-}at\text{-}point\ Q_p\ x\ (pvar\ Q_p\ 0) = hd\ x$
using *A 0*
by (*metis* (*no-types*, *lifting*) *One-nat-def eval-pvar lessI nth-Cons-0 Qp.to-R1-closed*)
then show $x \in (\lambda a.\ [a])\ `\ \{x \in carrier\ Q_p.\ val\ x \leq c\}$
using *A 0* **unfolding** *mem-Collect-eq*
by (*metis* (*no-types*, *lifting*) *image-iff mem-Collect-eq*)
qed
show $\bigwedge x.\ x \in (\lambda a.\ [a])\ `\ \{x \in carrier\ Q_p.\ val\ x \leq c\} \Longrightarrow x \in \{x \in carrier\ (Q_p{}^1).\ val\ (eval\text{-}at\text{-}point\ Q_p\ x\ (pvar\ Q_p\ 0)) \leq c\}$
proof fix x **assume** A: $x \in (\lambda a.\ [a])\ `\ \{x \in carrier\ Q_p.\ val\ x \leq c\}$
then obtain a **where** *a-def*: $x = [a] \wedge a \in carrier\ Q_p \wedge val\ a \leq c$
by *blast*
then have *0*: $x \in carrier\ (Q_p{}^1)$
using *cartesian-power-car-memI Qp.to-R1-closed* **by** *presburger*
then have *1*: $(eval\text{-}at\text{-}point\ Q_p\ x\ (pvar\ Q_p\ 0)) = a$
using *a-def* **by** (*metis eval-pvar less-one Qp.to-R-to-R1*)
show $x \in carrier\ (Q_p{}^1) \wedge val\ (eval\text{-}at\text{-}point\ Q_p\ x\ (pvar\ Q_p\ 0)) \leq c$
unfolding *1* **using** *a-def 0* **by** *blast*
qed
qed
show *?thesis* **using** *0* **unfolding** *1*
using *is-univ-semialgebraicI* **by** *blast*
qed

lemma *univ-val-strict-ineq-set-is-univ-semialgebraic*:
$is\text{-}univ\text{-}semialgebraic\ \{x \in carrier\ Q_p.\ val\ x < c\}$
proof−
have *0*: $is\text{-}semialgebraic\ 1\ \{x \in carrier\ (Q_p{}^1).\ val\ (Qp\text{-}ev\ (pvar\ Q_p\ 0)\ x) < c\}$
apply(*rule val-strict-ineq-set-is-semialgebraic'*)
using *pvar-closed* **by** *blast*
have *1*: $\{x \in carrier\ (Q_p{}^1).\ val\ (Qp\text{-}ev\ (pvar\ Q_p\ 0)\ x) < c\} = to\text{-}R1\ `\ \{x \in carrier\ Q_p.\ val\ x < c\}$
proof(*rule equalityI'*)
show $\bigwedge x.\ x \in \{x \in carrier\ (Q_p{}^1).\ val\ (eval\text{-}at\text{-}point\ Q_p\ x\ (pvar\ Q_p\ 0)) < c\} \Longrightarrow x \in (\lambda a.\ [a])\ `\ \{x \in carrier\ Q_p.\ val\ x < c\}$
proof− **fix** x **assume** A: $x \in \{x \in carrier\ (Q_p{}^1).\ val\ (eval\text{-}at\text{-}point\ Q_p\ x\ (pvar\ Q_p\ 0)) < c\}$
then have *0*: $x = [(hd\ x)] \wedge hd\ x \in carrier\ Q_p$
using *Qp.R1-memE Qp.R1-memE'* **by** *blast*
have *1*: $eval\text{-}at\text{-}point\ Q_p\ x\ (pvar\ Q_p\ 0) = hd\ x$
using *A 0*
by (*metis* (*no-types*, *lifting*) *One-nat-def eval-pvar lessI nth-Cons-0 Qp.to-R1-closed*)
then show $x \in (\lambda a.\ [a])\ `\ \{x \in carrier\ Q_p.\ val\ x < c\}$
using *A 0* **unfolding** *mem-Collect-eq*
by (*metis* (*no-types*, *lifting*) *image-iff mem-Collect-eq*)

qed
show $\bigwedge x.\ x \in (\lambda a.\ [a])\ `\ \{x \in carrier\ Q_p.\ val\ x < c\} \Longrightarrow x \in \{x \in carrier\ ({Q_p}^1).\ val\ (eval\text{-}at\text{-}point\ Q_p\ x\ (pvar\ Q_p\ 0)) < c\}$
proof fix x **assume** *A*: $x \in (\lambda a.\ [a])\ `\ \{x \in carrier\ Q_p.\ val\ x < c\}$
then obtain a **where** *a-def*: $x = [a] \wedge a \in carrier\ Q_p \wedge val\ a < c$
by *blast*
then have *0*: $x \in carrier\ ({Q_p}^1)$
using *cartesian-power-car-memI Qp.to-R1-closed* **by** *presburger*
then have *1*: $(eval\text{-}at\text{-}point\ Q_p\ x\ (pvar\ Q_p\ 0)) = a$
using *a-def* **by** (*metis eval-pvar less-one Qp.to-R-to-R1*)
show $x \in carrier\ ({Q_p}^1) \wedge val\ (eval\text{-}at\text{-}point\ Q_p\ x\ (pvar\ Q_p\ 0)) < c$
unfolding *1* **using** *a-def 0* **by** *blast*
qed
qed
show *?thesis* **using** *0* **unfolding** *1*
using *is-univ-semialgebraicI* **by** *blast*
qed

lemma *univ-val-eq-set-is-univ-semialgebraic*:
is-univ-semialgebraic $\{x \in carrier\ Q_p.\ val\ x = c\}$
proof−
have *0*: *is-semialgebraic 1* $\{x \in carrier\ ({Q_p}^1).\ val\ (Qp\text{-}ev\ (pvar\ Q_p\ 0)\ x) = c\}$
apply(*rule val-eq-set-is-semialgebraic′*)
using *pvar-closed* **by** *blast*
have *1*: $\{x \in carrier\ ({Q_p}^1).\ val\ (Qp\text{-}ev\ (pvar\ Q_p\ 0)\ x) = c\} = to\text{-}R1\ `\ \{x \in carrier\ Q_p.\ val\ x = c\}$
proof(*rule equalityI′*)
show $\bigwedge x.\ x \in \{x \in carrier\ ({Q_p}^1).\ val\ (eval\text{-}at\text{-}point\ Q_p\ x\ (pvar\ Q_p\ 0)) = c\} \Longrightarrow x \in (\lambda a.\ [a])\ `\ \{x \in carrier\ Q_p.\ val\ x = c\}$
proof− **fix** x **assume** *A*: $x \in \{x \in carrier\ ({Q_p}^1).\ val\ (eval\text{-}at\text{-}point\ Q_p\ x\ (pvar\ Q_p\ 0)) = c\}$
then have *0*: $x = [(hd\ x)] \wedge hd\ x \in carrier\ Q_p$
using *Qp.R1-memE Qp.R1-memE′* **by** *blast*
have *1*: $eval\text{-}at\text{-}point\ Q_p\ x\ (pvar\ Q_p\ 0) = hd\ x$
using *A 0*
by (*metis* (*no-types*, *lifting*) *One-nat-def eval-pvar lessI nth-Cons-0 Qp.to-R1-closed*)
show $x \in (\lambda a.\ [a])\ `\ \{x \in carrier\ Q_p.\ val\ x = c\}$
using *A 0* **unfolding** *mem-Collect-eq 1* **by** *blast*
qed
show $\bigwedge x.\ x \in (\lambda a.\ [a])\ `\ \{x \in carrier\ Q_p.\ val\ x = c\} \Longrightarrow x \in \{x \in carrier\ ({Q_p}^1).\ val\ (eval\text{-}at\text{-}point\ Q_p\ x\ (pvar\ Q_p\ 0)) = c\}$
proof fix x **assume** *A*: $x \in (\lambda a.\ [a])\ `\ \{x \in carrier\ Q_p.\ val\ x = c\}$
then obtain a **where** *a-def*: $x = [a] \wedge a \in carrier\ Q_p \wedge val\ a = c$
by *blast*
then have *0*: $x \in carrier\ ({Q_p}^1)$
using *cartesian-power-car-memI Qp.to-R1-closed* **by** *presburger*
then have *1*: $(eval\text{-}at\text{-}point\ Q_p\ x\ (pvar\ Q_p\ 0)) = a$
using *a-def* **by** (*metis eval-pvar less-one Qp.to-R-to-R1*)
show $x \in carrier\ ({Q_p}^1) \wedge val\ (eval\text{-}at\text{-}point\ Q_p\ x\ (pvar\ Q_p\ 0)) = c$

unfolding *1* **using** *a-def 0* **by** *blast*
qed
qed
show *?thesis* **using** *0* **unfolding** *1*
using *is-univ-semialgebraicI* **by** *blast*
qed

13.5.7 One Dimensional p-adic Balls are Semialgebraic

lemma *coord-ring-one-def*:
Pring Q_p $\{(0::nat)\} = (Q_p[\mathcal{X}_1])$
proof−
have $\{(0::nat)\} = \{..<1\}$
by *auto*
thus *?thesis*
unfolding *coord-ring-def*
by *auto*
qed

lemma *times-p-pow-val*:
assumes $a \in$ *carrier* Q_p
assumes $b = \mathfrak{p}[\hat{}]n \otimes a$
shows *val* $b =$ *val* $a + n$
using *val-mult*[*of* $\mathfrak{p}[\hat{}]n$ *a*] *assms* **unfolding** *assms*(*2*) *val-p-int-pow*
by (*metis add.commute p-intpow-closed*(*1*))

lemma *times-p-pow-neg-val*:
assumes $a \in$ *carrier* Q_p
assumes $b = \mathfrak{p}[\hat{}]-n \otimes a$
shows *val* $b =$ *val* $a - n$
by (*metis Qp.m-comm Qp-int-pow-nonzero assms*(*1*) *assms*(*2*) *p-intpow-closed*(*1*) *p-intpow-inv''* *p-nonzero val-fract val-p-int-pow*)

lemma *eint-minus-int-pos*:
assumes $a - eint\ n \geq 0$
shows $a \geq n$
using *assms* **apply**(*induction a*)
apply (*metis diff-ge-0-iff-ge eint-ord-simps*(*1*) *idiff-eint-eint zero-eint-def*)
by *simp*

p-adic balls as pullbacks of polynomial maps

lemma *balls-as-pullbacks*:
assumes $c \in$ *carrier* Q_p
shows $\exists P \in$ *carrier* $(Q_p[\mathcal{X}_1])$. *to-R1*‘ $B_n[c]$ = *poly-map 1* $[P]$ $^{-1}{}_1$ (*to-R1* ‘ $\mathcal{O}_p$)
proof−
obtain *P0* **where** *P0-def*: *P0* = (*to-poly* $(\mathfrak{p}[\hat{}](-n))) \otimes_{Q_p\text{-}x}((X\text{-}poly\ Q_p) \ominus_{Q_p\text{-}x}$ *to-poly c*)
by *blast*
have *0*: *P0* ∈ *carrier* Q_p-*x*

proof−
have *P0*: (*X-poly* Q_p) $\ominus_{Q_p\text{-}x}$ *to-poly* $c \in$ *carrier* Q_p-x
using *UPQ.X-closed UPQ.to-poly-closed assms* **by** *blast*
have *P1*: (*to-poly* ($\mathfrak{p}[\widehat{}](-n)$)) $\in$ *carrier* Q_p-x
using *UPQ.to-poly-closed p-intpow-closed*(*1*) **by** *blast*
then show *?thesis*
using *P0-def P0 P1*
by *blast*
qed
have *1*: $\bigwedge x.\ x \in$ *carrier* $Q_p \Longrightarrow$ *P0* $\cdot\ x = (\mathfrak{p}[\widehat{}](-n)) \otimes\ (x \ominus_{Q_p}\ c)$
proof− **fix** x **assume** *A*: $x \in$ *carrier* Q_p
have *P0*: (*to-poly* ($\mathfrak{p}[\widehat{}](-n)$)) $\cdot\ x = (\mathfrak{p}[\widehat{}](-n))$
using *A UPQ.to-fun-to-poly p-intpow-closed*(*1*) **by** *blast*
have *P1*: ((*X-poly* Q_p) $\ominus_{Q_p\text{-}x}$ *to-poly* c) $\cdot\ x =\ (x \ominus_{Q_p}\ c)$
by (*metis A UPQ.to-fun-X-minus X-poly-minus-def assms*)
have *P2*: *to-poly* ($\mathfrak{p}[\widehat{}](-n)$) $\in$ *carrier* Q_p-x
using *UPQ.to-poly-closed p-intpow-closed*(*1*) **by** *blast*
have *P3*: ((*X-poly* Q_p) $\ominus_{Q_p\text{-}x}$ *to-poly* c) $\in$ *carrier* Q_p-x
using *UPQ.X-closed UPQ.to-poly-closed assms* **by** *blast*
have *to-poly* ($\mathfrak{p}[\widehat{}]-\ n$) $\otimes_{Q_p\text{-}x}$ ((*X-poly* Q_p) $\ominus_{Q_p\text{-}x}$ *to-poly* c) $\cdot\ x =$ *to-poly* ($\mathfrak{p}[\widehat{}]-\ n$) $\cdot\ x \otimes$ (((*X-poly* Q_p) $\ominus_{Q_p\text{-}x}$ *to-poly* c) $\cdot\ x$)
using *A P0-def P0 P1 P2 P3 to-fun-mult*[*of to-poly* ($\mathfrak{p}[\widehat{}](-n)$) (*X-poly* Q_p) $\ominus_{Q_p\text{-}x}$ *to-poly* c x] *UPQ.to-fun-mult*
by *blast*
then have *to-poly* ($\mathfrak{p}[\widehat{}]-\ n$) $\otimes_{Q_p\text{-}x}$ ((*X-poly* Q_p) $\ominus_{Q_p\text{-}x}$ *to-poly* c) $\cdot\ x =$ ($\mathfrak{p}[\widehat{}](-n)$) $\otimes\ (x \ominus_{Q_p}\ c)$
by (*metis P0 P1*)
then show *P0* $\cdot\ x = (\mathfrak{p}[\widehat{}](-n)) \otimes\ (x \ominus_{Q_p}\ c)$
using *P0-def* **by** *metis*
qed
have *2*: $(\lambda a.\ [a])\ `\ B_n[c] =$ *poly-map 1* [*from-Qp-x P0*] $^{-1}$*1* $((\lambda a.\ [a])\ `\ \mathcal{O}_p)$
proof
show $(\lambda a.\ [a])\ `\ B_n[c] \subseteq$ *poly-map 1* [*from-Qp-x P0*] $^{-1}$*1* $((\lambda a.\ [a])\ `\ \mathcal{O}_p)$
proof
fix x
assume *A*: $x \in (\lambda a.\ [a])\ `\ B_n[c]$
then obtain a **where** *a-def*: $x = [a] \wedge a \in B_n[c]$
by *blast*
have *P0*: *P0* $\cdot\ a \in \mathcal{O}_p$
proof−
have $B_n[c] \subseteq$ *carrier* Q_p
using *c-ball-in-Qp* **by** *blast*
hence *a-closed*: $a \in$ *carrier* Q_p
using *a-def* **by** *blast*
have *P0*: *P0* $\cdot\ a = (\mathfrak{p}[\widehat{}](-n)) \otimes\ (a \ominus c)$
using *1 a-def c-ballE*(*1*)
by *blast*
then have *P1*: *val* (*P0* $\cdot\ a$) = *val* ($\mathfrak{p}[\widehat{}](-n)$) + *val* ($a \ominus c$)
using *val-mult*[*of* $\mathfrak{p}[\widehat{}]-n$ $a \ominus c$] *a-closed assms Qp.minus-closed p-intpow-closed*(*1*)

by *presburger*
then have *P2*: *val* $(P0 \cdot a) = val\ (a \ominus_{Q_p} c) - n$
by (*metis P0 Qp.m-comm Qp.minus-closed Qp-int-pow-nonzero assms local.a-closed*
p-intpow-closed(*1*) *p-intpow-inv''* *p-nonzero val-fract val-p-int-pow*)
have *P3*: *val* $(a \ominus_{Q_p} c) \geq n$
using *a-def c-ballE*(*2*)
by *blast*
then have *val* $(P0 \cdot a) \geq -n + n$
using *P2* **by** (*metis add.commute diff-conv-add-uminus diff-self local.eint-minus-ineq' zero-eint-def*)
then have *P4*: *val* $(P0 \cdot a) \geq 0$
by (*metis add.commute add.right-inverse zero-eint-def*)
have *P5*: $P0 \cdot a \in$ *carrier* Q_p
using *0 UPQ.to-fun-closed local.a-closed* **by** *blast*
then show *?thesis* **using** *P4*
using *val-ring-val-criterion*
by *blast*
qed
have *poly-map 1* [*from-Qp-x P0*] $x =$ [*Qp-ev* (*from-Qp-x P0*) [*a*]]
using *a-def poly-map-def*[*of 1* [*from-Qp-x P0*]] *poly-tuple-eval-def*[*of*]
by (*metis Qp-poly-mapE' c-ballE*(*1*) *length-0-conv poly-map-cons Qp.to-R1-closed*)
then have *poly-map 1* [*from-Qp-x P0*] $x =$ [$P0 \cdot a$]
using *Qp-x-Qp-poly-eval*[*of P0 a*]
by (*metis 0 a-def c-ballE*(*1*))
then have *P1*: *poly-map 1* [*from-Qp-x P0*] $x \in ((\lambda a.\ [a])\ `\ \mathcal{O}_p)$
using *P0*
by *blast*
have *P2*: $x \in$ *carrier* $({Q_p}^1)$
using *A c-ballE*(*1*) *Qp.to-R1-closed*
by *blast*
have *P3*: *is-poly-tuple 1* [*from-Qp-x P0*]
apply(*rule Qp-is-poly-tupleI*)
by (*metis 0 Qp-is-poly-tupleI from-Qp-x-closed gr-implies-not0 is-poly-tupleE is-poly-tuple-Cons list.size*(*3*) *zero-neq-one*)
show $x \in$ *poly-map 1* [*UP-to-IP* Q_p *0 P0*] $^{-1}{}_1$ $(\lambda a.\ [a])\ `\ \mathcal{O}_p$
using *P3 P2 P1* **unfolding** *evimage-def poly-map-def*
by *blast*
qed
have *20*: *is-poly-tuple 1* [*from-Qp-x P0*]
using *0 UP-to-IP-closed*[*of P0 0*::*nat*]
unfolding *is-poly-tuple-def*
by (*metis* (*no-types*, *lifting*) *empty-set from-Qp-x-closed list.simps*(*15*) *singletonD subset-code*(*1*))
show *poly-map 1* [*UP-to-IP* Q_p *0 P0*] $^{-1}{}_1$ $(\lambda a.\ [a])\ `\ \mathcal{O}_p \subseteq (\lambda a.\ [a])\ `\ B_n[c]$
proof fix x **assume** *A*: $x \in$ *poly-map 1* [*UP-to-IP* Q_p *0 P0*] $^{-1}{}_1$ $((\lambda a.\ [a])\ `\ \mathcal{O}_p)$
have *A0*: $(\lambda a.\ [a])\ `\ \mathcal{O}_p \subseteq$ *carrier* $({Q_p}^1)$
using *Qp-val-ring-is-univ-semialgebraic is-univ-semialgebraic-def Qp.to-R1-car-subset*

```
        Qp-val-ring-is-semialgebraic is-semialgebraic-closed by presburger
    have poly-map 1 [from-Qp-x P0] x ∈ ((λa. [a]) ‘ 𝒪_p)
      using A0 A 20 by blast
    then obtain a where a-def: a ∈ 𝒪_p ∧ (poly-map 1 [from-Qp-x P0] x) = [a]
      by blast
    have x-closed: x ∈ carrier (Q_p^1)
      using A
      by (meson evimage-eq)
    then obtain y where y-def: x = [y] ∧ y ∈ carrier Q_p
      using A
      by (metis Qp.to-R1-to-R Qp.to-R-pow-closed)
    have (poly-map 1 [from-Qp-x P0] x) = [(Qp-ev (from-Qp-x P0) [y])]
      unfolding poly-map-def poly-tuple-eval-def using x-closed
      by (smt 20 One-nat-def length-Suc-conv list.size(3) nth-Cons-0 nth-map
          poly-tuple-eval-closed poly-tuple-eval-def restrict-apply' Qp.to-R1-to-R
y-def zero-less-Suc)
    then have (poly-map 1 [from-Qp-x P0] x) = [P0 • y]
      by (metis 0 Qp-x-Qp-poly-eval y-def)
    then have [a] = [P0 • y]
      using a-def
      by presburger
    then have A1: a = (𝔭[^](−n)) ⊗ (y ⊖_{Q_p} c)
      using 1[of y] y-def
      by blast
    have y ∈ B_n[c]
    proof−
      have B0: val a = val (y ⊖_{Q_p} c) − n
        using A1 y-def Qp.minus-closed assms times-p-pow-neg-val by blast
      have B1: val a ≥ 0
        using a-def val-ring-memE by blast
      then have val (y ⊖_{Q_p} c) − n ≥ 0
        using B0
        by metis
      then have val (y ⊖_{Q_p} c) ≥ n
        using eint-minus-int-pos by blast
      then show y ∈ B_n[c]
        using c-ballI y-def by blast
    qed
    then show x ∈ (λa. [a]) ‘ B_n[c]
      using y-def by blast
  qed
 qed
 then show ?thesis
  by (meson 0 from-Qp-x-closed)
qed

lemma ball-is-semialgebraic:
 assumes c ∈ carrier Q_p
 shows is-semialgebraic 1 (to-R1‘ B_n[c])
```

proof−
obtain P **where** *P-def*: $P \in$ *carrier* $(Q_p[\mathcal{X}_1]) \wedge$ *to-R1*‘ $B_n[c] =$ *poly-map 1* $[P]$ $^{-1}{}_1$ $(to\text{-}R1\ `\ \mathcal{O}_p)$
using *assms balls-as-pullbacks*[*of c n*] **by** *meson*
have *is-poly-tuple 1* $[P]$
using *P-def* **unfolding** *is-poly-tuple-def*
by (*metis* (*no-types*, *opaque-lifting*) *list.inject list.set-cases neq-Nil-conv subset-code*(*1*))
then show *?thesis*
using *assms P-def pullback-is-semialg*[*of 1* $[P]$ *1* $((\lambda a.\ [a])\ `\ \mathcal{O}_p)$]
by (*metis* (*mono-tags*, *lifting*) *One-nat-def*
Qp-val-ring-is-semialgebraic is-semialgebraic-def length-Suc-conv
list.distinct(*1*) *list.size*(*3*))
qed

lemma *ball-is-univ-semialgebraic*:
assumes $c \in$ *carrier* Q_p
shows *is-univ-semialgebraic* $(B_n[c])$
using *assms ball-is-semialgebraic c-ball-in-Qp is-univ-semialgebraic-def*
by *presburger*

abbreviation *Qp-to-R1-set* **where**
Qp-to-R1-set $S \equiv$ *to-R1* ‘ S

13.5.8 Finite Unions and Intersections of Semialgebraic Sets

definition *are-semialgebraic* **where**
are-semialgebraic n Xs $= (\forall\ x.\ x \in Xs \longrightarrow$ *is-semialgebraic n x*$)$

lemma *are-semialgebraicI*:
assumes $\bigwedge x.\ x \in Xs \Longrightarrow$ *is-semialgebraic n x*
shows *are-semialgebraic n Xs*
using *are-semialgebraic-def assms* **by** *blast*

lemma *are-semialgebraicE*:
assumes *are-semialgebraic n Xs*
assumes $x \in Xs$
shows *is-semialgebraic n x*
using *are-semialgebraic-def assms*(*1*) *assms*(*2*) **by** *blast*

definition *are-univ-semialgebraic* **where**
are-univ-semialgebraic Xs $= (\forall\ x.\ x \in Xs \longrightarrow$ *is-univ-semialgebraic x*$)$

lemma *are-univ-semialgebraicI*:
assumes $\bigwedge x.\ x \in Xs \Longrightarrow$ *is-univ-semialgebraic x*
shows *are-univ-semialgebraic Xs*
using *are-univ-semialgebraic-def assms* **by** *blast*

lemma *are-univ-semialgebraicE*:

assumes *are-univ-semialgebraic Xs*
assumes $x \in Xs$
shows *is-univ-semialgebraic x*
using *are-univ-semialgebraic-def assms*(*1*) *assms*(*2*) **by** *blast*

lemma *are-univ-semialgebraic-semialgebraic*:
assumes *are-univ-semialgebraic Xs*
shows *are-semialgebraic 1* (*Qp-to-R1-set* ‘ *Xs*)
apply(*rule are-semialgebraicI*)
using *are-univ-semialgebraicE assms image-iff is-univ-semialgebraicE*
by (*metis* (*no-types*, *lifting*))

lemma *to-R1-set-union*:
to-R1 ‘ ($\bigcup$ *Xs*) = $\bigcup$ (*Qp-to-R1-set* ‘ *Xs*)
using *image-iff* **by** *blast*

lemma *to-R1-inter*:
assumes $Xs \neq \{\}$
shows *to-R1* ‘ ($\bigcap$ *Xs*) = $\bigcap$ (*Qp-to-R1-set* ‘ *Xs*)
proof
show *to-R1* ‘ ($\bigcap$ *Xs*) $\subseteq$ $\bigcap$ (*Qp-to-R1-set* ‘ *Xs*)
by *blast*
show $\bigcap$ (*Qp-to-R1-set* ‘ *Xs*) $\subseteq$ *to-R1* ‘ ($\bigcap$ *Xs*)
proof fix *x*
assume *A*: $x \in \bigcap$ (*Qp-to-R1-set* ‘ *Xs*)
then have *0*: $\bigwedge S.\ S \in Xs \Longrightarrow x \in$ (*Qp-to-R1-set S*)
by *blast*
obtain *S* **where** $S \in Xs \wedge x \in$ (*Qp-to-R1-set S*)
using *assms 0*
by *blast*
then obtain *b* **where** *b-def*: $b \in S \wedge x = [b]$
by *blast*
have $b \in$ ($\bigcap$ *Xs*)
using *0 b-def* **by** *blast*
then show $x \in$ *to-R1* ‘ ($\bigcap$ *Xs*)
using *b-def* **by** *blast*
qed
qed

lemma *finite-union-is-semialgebraic*:
assumes *finite Xs*
shows *Xs* $\subseteq$ *semialg-sets n* $\longrightarrow$ *is-semialgebraic n* ($\bigcup$ *Xs*)
apply(*rule finite.induct*[*of Xs*])
apply (*simp add*: *assms*)
apply (*simp add*: *empty-is-semialgebraic*)
by (*metis Sup-insert insert-subset is-semialgebraicI union-is-semialgebraic*)

lemma *finite-union-is-semialgebraic′*:
assumes *finite Xs*

assumes $Xs \subseteq$ *semialg-sets* n
shows *is-semialgebraic* n $(\bigcup\ Xs)$
using *assms*(*1*) *assms*(*2*) *finite-union-is-semialgebraic* **by** *blast*

lemma(**in** *padic-fields*) *finite-union-is-semialgebraic''*:
assumes *finite* S
assumes $\bigwedge x.\ x \in S \Longrightarrow$ *is-semialgebraic* m $(F\ x)$
shows *is-semialgebraic* m $(\bigcup\ x \in S.\ F\ x)$
using *assms finite-union-is-semialgebraic*[*of* F‘S m] **unfolding** *is-semialgebraic-def*
by *blast*

lemma *finite-union-is-univ-semialgebraic'*:
assumes *finite Xs*
assumes *are-univ-semialgebraic Xs*
shows *is-univ-semialgebraic* $(\bigcup\ Xs)$
proof−
have *is-semialgebraic 1* (*Qp-to-R1-set* $(\bigcup\ Xs)$)
using *assms finite-union-is-semialgebraic''*[*of* ((‘) $(\lambda a.\ [a])$ ‘ *Xs*)] *to-R1-set-union*[*of Xs*]
by (*metis* (*no-types*, *lifting*) *are-semialgebraicE are-univ-semialgebraic-semialgebraic finite-imageI is-semialgebraicE subsetI*)
then show *?thesis*
using *is-univ-semialgebraicI* **by** *blast*
qed

lemma *finite-intersection-is-semialgebraic*:
assumes *finite Xs*
shows $Xs \subseteq$ *semialg-sets* $n \wedge Xs \neq\{\} \longrightarrow$ *is-semialgebraic* n $(\bigcap\ Xs)$
apply(*rule finite.induct*[*of Xs*])
apply (*simp add*: *assms*)
apply *auto*[*1*]
proof fix A::$((nat \Rightarrow int) \times (nat \Rightarrow int))$ *set list set set* **fix** a
assume *0*: *finite A*
assume *1*: $A \subseteq$ *semialg-sets* $n \wedge A \neq \{\} \longrightarrow$ *is-semialgebraic* n $(\bigcap\ A)$
assume *2*: *insert a A* $\subseteq$ *semialg-sets* $n \wedge$ *insert a A* $\neq \{\}$
show *is-semialgebraic* n $(\bigcap\ (insert\ a\ A))$
proof(*cases* $A = \{\}$)
case *True*
then have *insert a A* $= \{a\}$
by *simp*
then show *?thesis*
by (*metis 2 cInf-singleton insert-subset is-semialgebraicI*)
next
case *False*
then have $A \subseteq$ *semialg-sets* $n \wedge A \neq \{\}$
using *2* **by** *blast*
then have *is-semialgebraic* n $(\bigcap\ A)$
using *1* **by** *linarith*
then show *?thesis*

using *0 1 2 intersection-is-semialg*
by (*metis Inf-insert insert-subset is-semialgebraicI*)
qed
qed

lemma *finite-intersection-is-semialgebraic'*:
assumes *finite Xs*
assumes $Xs \subseteq$ *semialg-sets n* $\wedge$ $Xs \neq \{\}$
shows *is-semialgebraic n* ($\bigcap$ *Xs*)
by (*simp add*: *assms*(*1*) *assms*(*2*) *finite-intersection-is-semialgebraic*)

lemma *finite-intersection-is-semialgebraic''*:
assumes *finite Xs*
assumes *are-semialgebraic n Xs* $\wedge$ $Xs \neq \{\}$
shows *is-semialgebraic n* ($\bigcap$ *Xs*)
by (*meson are-semialgebraicE assms*(*1*) *assms*(*2*)
finite-intersection-is-semialgebraic' is-semialgebraicE subsetI)

lemma *finite-intersection-is-univ-semialgebraic*:
assumes *finite Xs*
assumes *are-univ-semialgebraic Xs*
assumes $Xs \neq \{\}$
shows *is-univ-semialgebraic* ($\bigcap$ *Xs*)
proof−
have *are-semialgebraic 1* (*Qp-to-R1-set* ‘ *Xs*)
using *are-univ-semialgebraic-semialgebraic assms*(*2*) **by** *blast*
then have *is-semialgebraic 1* ($\bigcap$ (*Qp-to-R1-set* ‘ *Xs*))
using *assms finite-intersection-is-semialgebraic''*[*of Qp-to-R1-set* ‘ *Xs 1*]
by *blast*
then have *is-semialgebraic 1* (*Qp-to-R1-set* ($\bigcap$ *Xs*))
using *assms to-R1-inter*[*of Xs*]
by *presburger*
then show *?thesis*
using *is-univ-semialgebraicI* **by** *blast*
qed

13.6 Cartesian Products of Semialgebraic Sets

lemma *Qp-times-basic-semialg-right*:
assumes $a \in$ *carrier* ($Q_p[\mathcal{X}_n]$)
shows *cartesian-product* (*basic-semialg-set n k a*) (*carrier* (${Q_p}^m$)) = *basic-semialg-set* ($n+ m$) *k a*
proof
show *cartesian-product* (*basic-semialg-set n k a*) (*carrier* (${Q_p}^m$)) $\subseteq$ *basic-semialg-set* ($n + m$) *k a*
proof fix *x*
assume $x \in$ *cartesian-product* (*basic-semialg-set n k a*) (*carrier* (${Q_p}^m$))
then obtain *as bs* **where** *as-bs-def*: $as \in$ (*basic-semialg-set n k a*) $\wedge$ $bs \in$ (*carrier* (${Q_p}^m$)) $\wedge$ $x = as@bs$

using *cartesian-product-memE*[*of x basic-semialg-set n k a carrier* ($Q_p{}^m$) Q_p *n*]
basic-semialg-set-def
by (*metis* (*no-types*, *lifting*) *append-take-drop-id basic-semialg-set-memE*(*1*) *subsetI*)
have *0*: $x \in$ *carrier* ($Q_p{}^{n+m}$)
using *as-bs-def basic-semialg-set-memE*(*1*) *cartesian-product-closed'*
by *blast*
have *1*: (*Qp-ev a x* = *Qp-ev a as*)
using *as-bs-def poly-eval-cartesian-prod*[*of as n bs m a*] *assms basic-semialg-set-memE*(*1*) **by** *blast*
obtain *y* **where** *y-def*: $y \in$ *carrier* $Q_p \wedge$ (*Qp-ev a as* = (y[^]*k*))
using *as-bs-def* **using** *basic-semialg-set-memE*(*2*)[*of as n k a*] **by** *blast*
show $x \in$ *basic-semialg-set* (*n* + *m*) *k a*
apply(*rule basic-semialg-set-memI*[*of - - y*])
apply (*simp add*: *0*)
apply (*simp add*: *y-def*)
using *1 y-def* **by** *blast*
qed
show *basic-semialg-set* (*n* + *m*) *k a* ⊆ *cartesian-product* (*basic-semialg-set n k a*) (*carrier* ($Q_p{}^m$))
proof fix *x*
assume *A*: $x \in$ *basic-semialg-set* (*n* + *m*) *k a*
have *A0*: $x \in$ *carrier* ($Q_p{}^{n+m}$)
using *A basic-semialg-set-memE*(*1*) **by** *blast*
have *A1*: *set x* ⊆ *carrier* Q_p
using *A0*
by (*metis* (*no-types*, *lifting*) *cartesian-power-car-memE cartesian-power-car-memE' in-set-conv-nth subsetI*)
have *A2*: *length x* = *n* + *m*
using *A0 cartesian-power-car-memE*
by *blast*
obtain *as* **where** *as-def*: *as* = *take n x*
by *blast*
obtain *bs* **where** *bs-def*: *bs* = *drop n x*
by *blast*
have *0*: *x* = *as*@*bs*
using *A as-def bs-def*
by (*metis append-take-drop-id*)
have *1*: *as* ∈ *carrier* ($Q_p{}^n$)
apply(*rule cartesian-power-car-memI*)
using *as-def A2*
apply (*simp add*: *A2 min.absorb2*)
by (*metis* (*no-types*, *lifting*) *A1 as-def dual-order.trans set-take-subset*)
have *2*: *bs* ∈ *carrier* ($Q_p{}^m$)
apply(*rule cartesian-power-car-memI*)
using *bs-def A2*
apply (*simp add*: *A2*)
by (*metis A1 bs-def order.trans set-drop-subset*)

obtain *y* **where** *y-def*: $y \in$ *carrier* $Q_p \wedge$ *Qp-ev a x* $= (y[\widehat{}]k)$
using *basic-semialg-set-memE A* **by** *meson*
have *3*: *as* $\in$ *basic-semialg-set n k a*
apply(*rule basic-semialg-set-memI*[*of - - y*])
apply (*simp add*: *1*)
using ‹*y* $\in$ *carrier* $Q_p \wedge$ *Qp-ev a x* $= (y[\widehat{}]k)$› **apply** *blast*
using *y-def A 1 0 2 assms*(*1*) *poly-eval-cartesian-prod* **by** *blast*
show *x* $\in$ *cartesian-product* (*basic-semialg-set n k a*) (*carrier* $(Q_p{}^m)$)
using *3 2 0*
by (*metis* (*mono-tags*, *lifting*) *as-def basic-semialg-set-memE*(*1*) *bs-def cartesian-product-memI subsetI*)
qed
qed

lemma *Qp-times-basic-semialg-right-is-semialgebraic*:
assumes $k > 0$
assumes $a \in$ *carrier* $(Q_p[\mathcal{X}_n])$
shows *is-semialgebraic* $(n + m)$ (*cartesian-product* (*basic-semialg-set n k a*) (*carrier* $(Q_p{}^m)$))
proof−
have *0*: *cartesian-product* (*basic-semialg-set n k a*) (*carrier* $(Q_p{}^m)$) = *basic-semialg-set* $(n+ m)$ *k a*
using *Qp-times-basic-semialg-right assms*
by *presburger*
have *1*: $a \in$ *carrier* $(Q_p[\mathcal{X}_{n+m}])$
using *assms poly-ring-car-mono'*(*2*) **by** *blast*
have *2*: *is-semialgebraic* $(n + m)$ (*basic-semialg-set* $(n + m)$ *k a*)
using *assms basic-semialg-is-semialgebraic'*[*of a n+m k basic-semialg-set* $(n + m)$ *k a*]
1 **by** *blast*
show *?thesis*
using *0 2*
by *metis*
qed

lemma *Qp-times-basic-semialg-right-is-semialgebraic'*:
assumes $A \in$ *basic-semialgs n*
shows *is-semialgebraic* $(n + m)$ (*cartesian-product A* (*carrier* $(Q_p{}^m)$))
proof−
obtain *k P* **where** $k \neq 0 \wedge P \in$ *carrier* $(Q_p[\mathcal{X}_n]) \wedge A =$ *basic-semialg-set n k P*
using *assms is-basic-semialg-def*
by (*metis mem-Collect-eq*)
then show *?thesis* **using**
Qp-times-basic-semialg-right-is-semialgebraic[*of k P*]
using *assms*(*1*) **by** *blast*
qed

lemma *cartesian-product-memE'*:
assumes $x \in$ *cartesian-product A B*

obtains a b **where** $a \in A \wedge b \in B \wedge x = a@b$
using *assms* **unfolding** *cartesian-product-def* **by** *blast*

lemma *Qp-times-basic-semialg-left*:
assumes $a \in carrier\ (Q_p[\mathcal{X}_n])$
shows *cartesian-product* (*carrier* (${Q_p}^m$)) (*basic-semialg-set n k a*) = *basic-semialg-set* (*n*+*m*) *k* (*shift-vars n m a*)
proof
show *cartesian-product* (*carrier* (${Q_p}^m$)) (*basic-semialg-set n k a*) ⊆ *basic-semialg-set* (*n* + *m*) *k* (*shift-vars n m a*)
proof fix *x*
assume *A*: $x \in$ *cartesian-product* (*carrier* (${Q_p}^m$)) (*basic-semialg-set n k a*)
then obtain *as bs* **where** *as-bs-def*: *as* ∈ (*carrier* (${Q_p}^m$)) ∧ *bs* ∈ (*basic-semialg-set n k a*) ∧ *x* = *as*@*bs*
using *cartesian-product-memE'* **by** *blast*
have *0*: *Qp-ev* (*shift-vars n m a*) *x* = *Qp-ev a bs*
using *A as-bs-def assms shift-vars-eval*[*of a n as m bs*]
by (*metis* (*no-types*, *lifting*) *basic-semialg-set-memE*(*1*))
obtain *y* **where** *y-def*: *y* ∈ *carrier* Q_p ∧ *Qp-ev a bs* = (*y*[^]*k*)
using *as-bs-def basic-semialg-set-memE*(*2*)
by *blast*
have *1*: *x* ∈ *carrier* (${Q_p}^{n+m}$)
using *A as-bs-def*
by (*metis* (*no-types*, *lifting*) *add.commute basic-semialg-set-memE*(*1*) *cartesian-product-closed'*)
show *x* ∈ *basic-semialg-set* (*n* + *m*) *k* (*shift-vars n m a*)
apply(*rule basic-semialg-set-memI*[*of - - y*])
apply (*simp add*: *1*)
using *y-def* **apply** *blast*
using *0 y-def* **by** *blast*
qed
show *basic-semialg-set* (*n* + *m*) *k* (*shift-vars n m a*) ⊆ *cartesian-product* (*carrier* (${Q_p}^m$)) (*basic-semialg-set n k a*)
proof fix *x*
assume *A*: *x* ∈ *basic-semialg-set* (*n* + *m*) *k* (*shift-vars n m a*)
then obtain *y* **where** *y-def*: *y* ∈ *carrier* Q_p ∧ *Qp-ev* (*shift-vars n m a*) *x* = (*y*[^]*k*)
using *assms basic-semialg-set-memE*[*of x n* + *m k shift-vars n m a*]
shift-vars-closed[*of a m*] *Qp.cring-axioms*
by *blast*
have *x* ∈ *carrier* (${Q_p}^{m+n}$)
using *A basic-semialg-set-memE*(*1*)
by (*metis add.commute*)
then have *x* ∈ *cartesian-product* (*carrier* (${Q_p}^m$)) (*carrier* (${Q_p}^n$))
using *cartesian-product-carrier* **by** *blast*
then obtain *as bs* **where** *as-bs-def*: *x* = *as*@*bs* ∧ *as* ∈ *carrier* (${Q_p}^m$) ∧ *bs* ∈ *carrier* (${Q_p}^n$)
by (*meson cartesian-product-memE'*)
have *bs* ∈ (*basic-semialg-set n k a*)

apply(*rule basic-semialg-set-memI*[*of - - y*])
using *as-bs-def* **apply** *blast*
apply (*simp add*: *y-def*)
using *y-def shift-vars-eval*[*of a n as m bs*] *as-bs-def assms*(*1*)
by *metis*
then show $x \in$ *cartesian-product* (*carrier* ($Q_p{}^m$)) (*basic-semialg-set n k a*)
using *as-bs-def* **unfolding** *cartesian-product-def*
by *blast*
qed
qed

lemma *Qp-times-basic-semialg-left-is-semialgebraic*:
assumes $k > 0$
assumes $a \in$ *carrier* ($Q_p[\mathcal{X}_n]$)
shows *is-semialgebraic* ($n + m$) (*cartesian-product* (*carrier* ($Q_p{}^m$)) (*basic-semialg-set n k a*))
using *basic-semialg-is-semialgebraic'*[*of a n+m k*] *Qp-times-basic-semialg-left*
by (*metis assms*(*1*) *assms*(*2*) *basic-semialg-is-semialgebraic is-basic-semialg-def neq0-conv shift-vars-closed*)

lemma *Qp-times-basic-semialg-left-is-semialgebraic'*:
assumes $A \in$ *basic-semialgs n*
shows *is-semialgebraic* ($n + m$) (*cartesian-product* (*carrier* ($Q_p{}^m$)) *A*)
proof−
obtain *k P* **where** $k \neq 0 \wedge P \in$*carrier* ($Q_p[\mathcal{X}_n]$)$\wedge$ $A =$ *basic-semialg-set n k P*
using *assms is-basic-semialg-def*
by (*metis mem-Collect-eq*)
then show *?thesis* **using**
Qp-times-basic-semialg-left-is-semialgebraic[*of k P*]
using *assms*(*1*) **by** *blast*
qed

lemma *product-of-basic-semialgs-is-semialg*:
assumes $k > 0$
assumes $l > 0$
assumes $a \in$ *carrier* ($Q_p[\mathcal{X}_n]$)
assumes $b \in$ *carrier* ($Q_p[\mathcal{X}_m]$)
shows *is-semialgebraic* ($n + m$) (*cartesian-product* (*basic-semialg-set n k a*) (*basic-semialg-set m l b*))
proof−
have *0*: *cartesian-product* (*basic-semialg-set n k a*) (*carrier* ($Q_p{}^m$)) = *basic-semialg-set* ($n+ m$) *k a*
using *Qp-times-basic-semialg-right assms* **by** *presburger*
have *1*: *cartesian-product* (*carrier* ($Q_p{}^n$)) (*basic-semialg-set m l b*) = *basic-semialg-set* ($m + n$) *l* (*shift-vars m n b*)
using *Qp-times-basic-semialg-left assms* **by** *blast*
have *2*: (*cartesian-product* (*basic-semialg-set n k a*) (*basic-semialg-set m l b*)) =
cartesian-product (*basic-semialg-set n k a*) (*carrier* ($Q_p{}^m$)) ∩
cartesian-product (*carrier* ($Q_p{}^n$)) (*basic-semialg-set m l b*)

proof−
have *0*: *basic-semialg-set n k a* ⊆ *carrier* (${Q_p}^n$)
using *basic-semialg-set-memE*(*1*) **by** *blast*
have *1*: *carrier* (${Q_p}^m$) ⊆ *carrier* (${Q_p}^m$)
by *simp*
have *2*: *carrier* (${Q_p}^n$) ⊆ *carrier* (${Q_p}^n$)
by *simp*
have *3*: *basic-semialg-set m l b* ⊆ *carrier* (${Q_p}^m$)
using *basic-semialg-set-memE*(*1*) **by** *blast*
show *?thesis*
using *0 1 2 3 cartesian-product-intersection*[*of* (*basic-semialg-set n k a*) Q_p *n*
(*carrier* (${Q_p}^m$)) *m*
(*carrier* (${Q_p}^n$)) (*basic-semialg-set m l b*)]
by (*smt Collect-cong inf.absorb-iff1 inf.absorb-iff2*)
qed
then show *?thesis*
using *Qp-times-basic-semialg-left-is-semialgebraic*
Qp-times-basic-semialg-right-is-semialgebraic assms
by (*metis* (*no-types*, *lifting*) *add.commute intersection-is-semialg*)
qed

lemma *product-of-basic-semialgs-is-semialg′*:
assumes *A* ∈ (*basic-semialgs n*)
assumes *B* ∈ (*basic-semialgs m*)
shows *is-semialgebraic* (*n* + *m*) (*cartesian-product A B*)
proof−
obtain *k a* **where** *ka-def*: *k* > *0* ∧ *a* ∈ *carrier* ($Q_p[\mathcal{X}_n]$) ∧ *A* = (*basic-semialg-set n k a*)
using *assms*
by (*metis is-basic-semialg-def mem-Collect-eq neq0-conv*)
obtain *l b* **where** *lb-def*: *l* > *0* ∧ *b* ∈ *carrier* ($Q_p[\mathcal{X}_m]$) ∧ *B* = (*basic-semialg-set m l b*)
by (*metis assms*(*2*) *gr-zeroI is-basic-semialg-def mem-Collect-eq*)
show *?thesis* **using** *ka-def lb-def assms product-of-basic-semialgs-is-semialg*
by *blast*
qed

lemma *car-times-semialg-is-semialg*:
assumes *is-semialgebraic m B*
shows *is-semialgebraic* (*n* + *m*) (*cartesian-product* (*carrier* (${Q_p}^n$)) *B*)
apply(*rule gen-boolean-algebra.induct*[*of B carrier* (${Q_p}^m$)*basic-semialgs m*])
using *assms is-semialgebraic-def semialg-sets-def* **apply** *blast*
apply (*simp add*: *carrier-is-semialgebraic cartesian-product-carrier*)
proof−
show ⋀*A*. *A* ∈ *basic-semialgs m* ⟹ *is-semialgebraic* (*n* + *m*) (*cartesian-product* (*carrier* (${Q_p}^n$)) (*A* ∩ *carrier* (${Q_p}^m$)))
proof−
fix *A* **assume** *A*: *A* ∈ *basic-semialgs m*
then have (*A* ∩ *carrier* (${Q_p}^m$)) = *A*

by (*metis basic-semialg-set-memE*(*1*) *inf-absorb1 is-basic-semialg-def mem-Collect-eq subsetI*)
then show *is-semialgebraic* (*n* + *m*) (*cartesian-product* (*carrier* (*Q*$_p$^*n*)) (*A* ∩ *carrier* (*Q*$_p$^*m*)))
using *add.commute*[*of n m*] *assms A*
Qp-times-basic-semialg-left-is-semialgebraic′
by (*simp add*: ‹*n* + *m* = *m* + *n*›)
qed
show ⋀*A C*. *A* ∈ *gen-boolean-algebra* (*carrier* (*Q*$_p$^*m*)) (*basic-semialgs m*) ⟹
is-semialgebraic (*n* + *m*) (*cartesian-product* (*carrier* (*Q*$_p$^*n*)) *A*) ⟹
C ∈ *gen-boolean-algebra* (*carrier* (*Q*$_p$^*m*)) (*basic-semialgs m*) ⟹
is-semialgebraic (*n* + *m*) (*cartesian-product* (*carrier* (*Q*$_p$^*n*)) *C*) ⟹
is-semialgebraic (*n* + *m*) (*cartesian-product* (*carrier* (*Q*$_p$^*n*)) (*A* ∪ *C*))
proof− **fix** *A C* **assume** *A*: *A* ∈ *gen-boolean-algebra* (*carrier* (*Q*$_p$^*m*)) (*basic-semialgs m*)
is-semialgebraic (*n* + *m*) (*cartesian-product* (*carrier* (*Q*$_p$^*n*)) *A*)
C ∈ *gen-boolean-algebra* (*carrier* (*Q*$_p$^*m*)) (*basic-semialgs m*)
is-semialgebraic (*n* + *m*) (*cartesian-product* (*carrier* (*Q*$_p$^*n*)) *C*)
then have *B*: *is-semialgebraic* (*n* + *m*) (*cartesian-product* (*carrier* (*Q*$_p$^*n*)) *A* ∪ *cartesian-product* (*carrier* (*Q*$_p$^*n*)) *C*)
using *union-is-semialgebraic* **by** *blast*
show *is-semialgebraic* (*n* + *m*) (*cartesian-product* (*carrier* (*Q*$_p$^*n*)) (*A* ∪ *C*))
proof−
have *0*: *A* ⊆ *carrier* (*Q*$_p$^*m*)
using *A*(*1*) *gen-boolean-algebra-subset*
by *blast*
have *1*: *C* ⊆ *carrier* (*Q*$_p$^*m*)
using *A*(*3*) *gen-boolean-algebra-subset*
by *blast*
then show *?thesis* **using** *0 A B*
using *cartesian-product-binary-union-right*[*of A Q*$_p$ *m C* (*carrier* (*Q*$_p$^*n*))]
unfolding *is-semialgebraic-def semialg-sets-def*
by *presburger*
qed
qed
show ⋀*A*. *A* ∈ *gen-boolean-algebra* (*carrier* (*Q*$_p$^*m*)) (*basic-semialgs m*) ⟹
is-semialgebraic (*n* + *m*) (*cartesian-product* (*carrier* (*Q*$_p$^*n*)) *A*) ⟹
is-semialgebraic (*n* + *m*) (*cartesian-product* (*carrier* (*Q*$_p$^*n*)) (*carrier* (*Q*$_p$^*m*) − *A*))
proof− **fix** *A* **assume** *A*: *A* ∈ *gen-boolean-algebra* (*carrier* (*Q*$_p$^*m*)) (*basic-semialgs m*)
is-semialgebraic (*n* + *m*) (*cartesian-product* (*carrier* (*Q*$_p$^*n*)) *A*)
then have *A* ⊆ *carrier* (*Q*$_p$^*m*)
using *gen-boolean-algebra-subset*
by *blast*
then show *is-semialgebraic* (*n* + *m*) (*cartesian-product* (*carrier* (*Q*$_p$^*n*)) (*carrier* (*Q*$_p$^*m*) − *A*))
using *A cartesian-product-car-complement-right*[*of A Q*$_p$ *m n*]
unfolding *is-semialgebraic-def semialg-sets-def*

by (*metis* (*mono-tags*, *lifting*) *semialg-complement semialg-sets-def*)
qed
qed

lemma *basic-semialg-times-semialg-is-semialg*:
assumes $A \in$ *basic-semialgs n*
assumes *is-semialgebraic m B*
shows *is-semialgebraic* $(n + m)$ (*cartesian-product A B*)
apply(*rule gen-boolean-algebra.induct*[*of B carrier* $(Q_p{}^m)$*basic-semialgs m*])
using *assms*(*2*) *is-semialgebraic-def semialg-sets-def* **apply** *blast*
using *Qp-times-basic-semialg-right-is-semialgebraic′ assms*(*1*) **apply** *blast*
apply (*metis assms*(*1*) *basic-semialg-is-semialgebraic inf.absorb1 is-semialgebraic-closed mem-Collect-eq product-of-basic-semialgs-is-semialg′*)
apply (*metis* (*no-types*, *lifting*) *cartesian-product-binary-union-right is-semialgebraicI is-semialgebraic-closed semialg-sets-def union-is-semialgebraic*)
proof−
show $\bigwedge Aa.$ $Aa \in$ *gen-boolean-algebra* (*carrier* $(Q_p{}^m)$) (*basic-semialgs m*) $\Longrightarrow$ *is-semialgebraic* $(n + m)$ (*cartesian-product A Aa*) $\Longrightarrow$ *is-semialgebraic* $(n + m)$ (*cartesian-product A* (*carrier* $(Q_p{}^m)$ − *Aa*))
proof− **fix** *B* **assume** *A*: $B \in$ *gen-boolean-algebra* (*carrier* $(Q_p{}^m)$) (*basic-semialgs m*)
is-semialgebraic $(n + m)$ (*cartesian-product A B*)
show *is-semialgebraic* $(n + m)$ (*cartesian-product A* (*carrier* $(Q_p{}^m)$ − *B*))
using *A assms cartesian-product-complement-right*[*of B* Q_p *m A n*] *add.commute*[*of n m*]
proof −
have *f1*: $\forall n\ B.\ \neg$ *is-semialgebraic n B* $\vee$ $B \subseteq$ *carrier* $(Q_p{}^n)$
by (*meson is-semialgebraic-closed*)
have *is-basic-semialg n A*
using ‹$A \in \{S.$ *is-basic-semialg n S*$\}$› **by** *blast*
then have *f2*: *is-semialgebraic n A*
using *padic-fields.basic-semialg-is-semialgebraic padic-fields-axioms* **by** *blast*
have $B \in$ *semialg-sets m*
using ‹$B \in$ *gen-boolean-algebra* (*carrier* $(Q_p{}^m)$) $\{S.$ *is-basic-semialg m S*$\}$› *semialg-sets-def* **by** *blast*
then have *is-semialgebraic m B*
by (*meson padic-fields.is-semialgebraicI padic-fields-axioms*)
then show *?thesis*
using *f2 f1* **by** (*metis* (*no-types*) *Qp-times-basic-semialg-right-is-semialgebraic′* ‹$A \in \{S.$ *is-basic-semialg n S*$\}$› ‹$[\![B \subseteq$ *carrier* $(Q_p{}^m)$; $A \subseteq$ *carrier* $(Q_p{}^n)]\!] \Longrightarrow$ *cartesian-product A* (*carrier* $(Q_p{}^m)$) − *cartesian-product A B* = *cartesian-product A* (*carrier* $(Q_p{}^m)$ − *B*)› ‹*is-semialgebraic* $(n + m)$ (*cartesian-product A B*)› *diff-is-semialgebraic*)
qed
qed
qed

Semialgebraic sets are closed under cartesian products

lemma *cartesian-product-is-semialgebraic*:
assumes *is-semialgebraic n A*

assumes *is-semialgebraic m B*
shows *is-semialgebraic* (*n* + *m*) (*cartesian-product A B*)
apply(*rule gen-boolean-algebra.induct*[*of A carrier* (${Q_p}^n$)*basic-semialgs n*])
using *assms is-semialgebraicE semialg-sets-def* **apply** *blast*
using *assms car-times-semialg-is-semialg* **apply** *blast*
using *assms basic-semialg-times-semialg-is-semialg*
apply (*simp add*: *Int-absorb2 basic-semialg-is-semialgebraic is-semialgebraic-closed*)
proof−
show ⋀*A C*. *A* ∈ *gen-boolean-algebra* (*carrier* (${Q_p}^n$)) (*basic-semialgs n*) ⟹
is-semialgebraic (*n* + *m*) (*cartesian-product A B*) ⟹
C ∈ *gen-boolean-algebra* (*carrier* (${Q_p}^n$)) (*basic-semialgs n*) ⟹
is-semialgebraic (*n* + *m*) (*cartesian-product C B*) ⟹ *is-semialgebraic* (*n* + *m*) (*cartesian-product* (*A* ∪ *C*) *B*)
proof− **fix** *A C* **assume** *A*: *A* ∈ *gen-boolean-algebra* (*carrier* (${Q_p}^n$)) (*basic-semialgs n*)
is-semialgebraic (*n* + *m*) (*cartesian-product A B*)
C ∈ *gen-boolean-algebra* (*carrier* (${Q_p}^n$)) (*basic-semialgs n*)
is-semialgebraic (*n* + *m*) (*cartesian-product C B*)
show *is-semialgebraic* (*n* + *m*) (*cartesian-product* (*A* ∪ *C*) *B*)
using *A cartesian-product-binary-union-left*[*of A* Q_p *n C B*]
by (*metis* (*no-types*, *lifting*) *gen-boolean-algebra-subset union-is-semialgebraic*)
qed
show ⋀*A*. *A* ∈ *gen-boolean-algebra* (*carrier* (${Q_p}^n$)) (*basic-semialgs n*) ⟹
is-semialgebraic (*n* + *m*) (*cartesian-product A B*) ⟹ *is-semialgebraic* (*n* + *m*) (*cartesian-product* (*carrier* (${Q_p}^n$) − *A*) *B*)
proof− **fix** *A* **assume** *A*: *A* ∈ *gen-boolean-algebra* (*carrier* (${Q_p}^n$)) (*basic-semialgs n*)
is-semialgebraic (*n* + *m*) (*cartesian-product A B*)
show *is-semialgebraic* (*n* + *m*) (*cartesian-product* (*carrier* (${Q_p}^n$) − *A*) *B*)
using *assms A cartesian-product-complement-left*[*of A* Q_p *n B m*]
unfolding *is-semialgebraic-def semialg-sets-def*
by (*metis car-times-semialg-is-semialg diff-is-semialgebraic is-semialgebraicE is-semialgebraicI*
is-semialgebraic-closed semialg-sets-def)
qed
qed

13.7 N^{th} Power Residues

definition *nth-root-poly* **where**
nth-root-poly (*n*::*nat*) *a* = ((*X-poly* Q_p) [^]$_{Q_p\text{-}x}$ *n*) ⊖$_{Q_p\text{-}x}$ (*to-poly a*)

lemma *nth-root-poly-closed*:
assumes *a* ∈ *carrier* Q_p
shows *nth-root-poly n a* ∈ *carrier* Q_p-*x*
using *assms* **unfolding** *nth-root-poly-def*
by (*meson UPQ.P.minus-closed UPQ.P.nat-pow-closed UPQ.X-closed UPQ.to-poly-closed*)

lemma *nth-root-poly-eval*:

assumes $a \in$ *carrier* Q_p
assumes $b \in$ *carrier* Q_p
shows (*nth-root-poly n a*) $\cdot\ b = (b[\widehat{}]n) \ominus_{Q_p} a$
using *assms* **unfolding** *nth-root-poly-def*
using *UPQ.P.nat-pow-closed UPQ.X-closed UPQ.to-fun-X-pow UPQ.to-fun-diff UPQ.to-fun-to-poly UPQ.to-poly-closed* **by** *presburger*

Hensel's lemma gives us this criterion for the existence of n-th roots

lemma *nth-root-poly-root*:
assumes (*n*::*nat*) > *1*
assumes $a \in \mathcal{O}_p$
assumes $a \neq \mathbf{1}$
assumes *val* ($\mathbf{1} \ominus_{Q_p} a$) > *2* val* ([*n*]$\cdot\mathbf{1}$)
shows ($\exists\ b \in \mathcal{O}_p$. ($(b[\widehat{}]n) = a$))
proof−
obtain α **where** *alpha-def*: $\alpha \in$ *carrier* $Z_p \wedge \iota\ \alpha = a$
using *assms*(*2*) **by** *blast*
have *0*: $\alpha \in$ *carrier* Z_p
by (*simp add*: *alpha-def*)
have *1*: $\alpha \neq \mathbf{1}_{Z_p}$
using *assms alpha-def inc-of-one*
by *blast*
obtain *N* **where** *N-def*: $N = [n]\cdot_{Z_p} \mathbf{1}_{Z_p}$
by *blast*
have *N-closed*: $N \in$ *carrier* Z_p
using *N-def Zp-nat-mult-closed*
by *blast*
have *2*: $\iota\ ([n]\cdot_{Z_p} \mathbf{1}_{Z_p}) = ([n]\cdot\ \mathbf{1})$
proof(*induction n*)
case *0*
have *00*: $[(0::nat)] \cdot_{Z_p} \mathbf{1}_{Z_p} = \mathbf{0}_{Z_p}$
using *Zp-nat-inc-zero* **by** *blast*
have *01*: $[(0::nat)] \cdot_{Q_p} \mathbf{1} = \mathbf{0}$
using *Qp.nat-inc-zero* **by** *blast*
then show *?case*
using *00 inc-of-nat* **by** *blast*
next
case (*Suc n*)
then show *?case*
using *inc-of-nat* **by** *blast*
qed
have *3*: *val-Zp* ($\mathbf{1}_{Z_p} \ominus_{Z_p} \alpha$) = *val* ($\mathbf{1} \ominus_{Q_p} a$)
using *alpha-def Zp.one-closed ring-hom-one*[*of* $\iota\ Z_p\ Q_p$] *inc-is-hom Zp.ring-hom-minus*[*of* $Q_p\ \iota\ \mathbf{1}_{Z_p}\ \alpha$]
Qp.ring-axioms
unfolding ι-*def*
by (*metis* Q_p-*def Zp.minus-closed Zp-def padic-fields.val-of-inc padic-fields-axioms*)
have *4*: ([*n*]$\cdot_{Z_p} \mathbf{1}_{Z_p}) \in$ *nonzero* Z_p
proof−

have *40*: *int n* $\geq$ *0*
using *of-nat-0-le-iff* **by** *blast*
have *nat* (*int n*) = *n*
using *nat-int* **by** *blast*
hence [*n*]$\cdot_{Z_p}$ $\mathbf{1}_{Z_p}$ = [*int n*]$\cdot_{Z_p}$ $\mathbf{1}_{Z_p}$
using *40* **unfolding** *add-pow-def int-pow-def nat-pow-def*
proof −
have (*if int n < 0 then* $inv_{add\text{-}monoid\ Z_p}$ *rec-nat* $\mathbf{1}_{add\text{-}monoid\ Z_p}$ (λ*n f*. *f* $\otimes_{add\text{-}monoid\ Z_p}$ $\mathbf{1}_{Z_p}$) *0 else rec-nat* $\mathbf{1}_{add\text{-}monoid\ Z_p}$ (λ*n f*. *f* $\otimes_{add\text{-}monoid\ Z_p}$ $\mathbf{1}_{Z_p}$) *n*) = *rec-nat* $\mathbf{1}_{add\text{-}monoid\ Z_p}$ (λ*n f*. *f* $\otimes_{add\text{-}monoid\ Z_p}$ $\mathbf{1}_{Z_p}$) *n*
by (*meson of-nat-less-0-iff*)
then show *rec-nat* $\mathbf{1}_{add\text{-}monoid\ Z_p}$ (λ*n f*. *f* $\otimes_{add\text{-}monoid\ Z_p}$ $\mathbf{1}_{Z_p}$) *n* = (*let f* = *rec-nat* $\mathbf{1}_{add\text{-}monoid\ Z_p}$ (λ*n f*. *f* $\otimes_{add\text{-}monoid\ Z_p}$ $\mathbf{1}_{Z_p}$) *in if int n < 0 then* $inv_{add\text{-}monoid\ Z_p}$ *f* (*nat* (− *int n*)) *else f* (*nat* (*int n*)))
using ‹*nat* (*int n*) = *n*› **by** *presburger*
qed
thus *?thesis*
using *Zp-char-0*[*of n*] *Zp.not-nonzero-memE Zp-char-0' assms*(*1*) *gr-implies-not-zero*
by *blast*
qed
then have *5*: *val-Zp* ([*n*]$\cdot_{Z_p}$ $\mathbf{1}_{Z_p}$) = *val* ([*n*]$\cdot_{Q_p}$ (**1**))
using *2 ord-of-inc*
by (*metis N-closed N-def val-of-inc*)
then have *6*: (*val-Zp* ($\mathbf{1}_{Z_p}$ $\ominus_{Z_p}$ α)) > *2*∗(*val-Zp* ([*n*]$\cdot_{Z_p}$ $\mathbf{1}_{Z_p}$))
using *assms 3* **by** *presburger*
have ∃ *b* ∈ *carrier* Z_p. (*b*[^]$_{Z_p}$*n*= α)
using *Zp-nth-root-lemma*[*of* α *n*] *assms 0 1 6* **by** *blast*
then obtain *b* **where** *b-def*: *b* ∈ *carrier* Z_p ∧ (*b*[^]$_{Z_p}$*n*= α)
by *blast*
then have ι (*b* [^]$_{Z_p}$ *n*) = *a*
using *alpha-def* **by** *blast*
then have (ι *b*) [^] *n* = *a*
by (*metis Qp.nat-inc-zero* Q_p*-def Qp.nat-pow-zero Zp.nat-pow-0 Zp.nat-pow-zero Zp-nat-inc-zero* ι*-def alpha-def assms*(*3*) *b-def frac-inc-of-nat inc-of-one inc-pow not-nonzero-Qp*)
then show *?thesis*
using *b-def* **by** *blast*
qed

All points sufficiently close to 1 have nth roots

lemma *eint-nat-times-2*:
2∗(*n*::*nat*) = *2*∗*eint n*
using *times-eint-simps*(*1*)
by (*metis mult.commute mult-2-right of-nat-add*)

lemma *P-set-of-one*:
P-set 1 = *nonzero* Q_p
apply(*rule equalityI*) **apply**(*rule subsetI*)

unfolding *P-set-def nonzero-def mem-Collect-eq* **apply** *blast*
apply(*rule subsetI*) **unfolding** *P-set-def nonzero-def mem-Collect-eq*
using *Qp.nat-pow-eone* **by** *blast*

lemma *nth-power-fact*:
assumes (*n*::*nat*) ≥ *1*
shows ∃ (*m*::*nat*) > *0*. ∀ *u* ∈ *carrier* Q_p. *ac m u* = *1* ∧ *val u* = *0* ⟶ *u* ∈ *P-set n*
proof(*cases n* = *1*)
case *True*
have ∀ *u* ∈ *carrier* Q_p. *ac 1 u* = *1* ∧ *val u* = *0* ⟶ *u* ∈ *P-set n*
unfolding *True P-set-of-one*
by (*metis iless-Suc-eq padic-fields.val-ring-memE padic-fields.zero-in-val-ring padic-fields-axioms val-nonzero zero-eint-def*)
then show *?thesis* **by** *blast*
next
case *F*: *False*
obtain *m* **where** *m-def*: *m* = *1* + *nat* (*2*∗(*ord* ([*n*]·$_{Q_p}$ (**1**))))
by *blast*
then have *m-pos*: *m* > *0*
by *linarith*
have ∀ *u* ∈ *carrier* Q_p. *ac m u* = *1* ∧ *val u* = *0* ⟶ *u* ∈ *P-set n*
proof
fix *u*
assume *A*: *u* ∈ *carrier* Q_p
show *ac m u* = *1* ∧ *val u* = *0* ⟶ *u* ∈ *P-set n*
proof
assume *B*: *ac m u* = *1* ∧ *val u* = *0*
then have *0*: *val u* = *val* **1**
by (*smt A ac-def not-nonzero-Qp val-one val-ord zero-eint-def*)
have *1*: *ac m u* = *ac m* **1**
by (*metis B Qp.one-nonzero ac-p ac-p-int-pow-factor angular-component-factors-x angular-component-p inc-of-one m-pos p-nonzero*)
have *2*: *u* ∈ *nonzero* Q_p
proof−
have *ac m* **0** = *0*
by (*meson ac-def*)
then have *u* ≠ **0**
by (*metis B zero-neq-one*)
then show *?thesis*
using *A not-nonzero-Qp Qp.nonzero-memI* **by** *presburger*
qed
then have *3*: *val* (**1** ⊖$_{Q_p}$ *u*) ≥ *m* **using** *m-pos 0 1 ac-ord-prop*[*of* **1** *u 0*::*int m*]
by (*metis B Qp.one-nonzero add.right-neutral eint.inject val-ord zero-eint-def*)
show *u* ∈ *P-set n*
proof(*cases u* = **1**)
case *True*
then show *?thesis*

```
        by (metis P-set-one insert-iff zeroth-P-set)
      next
        case False
        have F0: u ∈ 𝒪_p
          apply(rule val-ring-memI, rule A)
          unfolding 0 val-one by auto
        have F1: val (1 ⊖_{Q_p} u) ≥ m
          using False 3 by blast
        have ord (1 ⊖ u) ≥ m
       by (metis A F1 False Qp.not-eq-diff-nonzero Qp.one-closed eint-ord-simps(1)
val-ord)
        hence F2: ord (1 ⊖_{Q_p} u) > 2*(ord ([n]· 1))
       using m-def F1 A False Qp.not-eq-diff-nonzero Qp.one-closed eint-ord-simps(1)
            int-nat-eq of-nat-1 of-nat-add val-ord[of 1 ⊖ u] eint-nat-times-2
          by linarith
        have val (1 ⊖_{Q_p} u) > 2*(val ([n]· 1))
        proof−
          have 0: val (1 ⊖_{Q_p} u) > 2*(ord ([n]· 1))
           using F2 val-ord[of 1 ⊖ u] A False Qp.not-eq-diff-nonzero Qp.one-closed
eint-ord-simps(2) by presburger
          have n > 0
            using assms  by linarith
          hence eint (ord ([n] · 1)) = val ([n] · 1)
            using val-ord-nat-inc[of n]
            by blast
          hence 2*ord ([n]· 1) = 2*val ([n]· 1)
            by (metis inc-of-nat times-eint-simps(1))
          thus ?thesis
            using 0 val-ord[of 1 ⊖ u] assms
            by presburger
        qed
        then have (∃  b ∈ 𝒪_p. ((b[^]n) = u))
          using m-def False nth-root-poly-root[of n u] F0 assms  F by linarith
        then have (∃  b ∈ carrier Q_p. ((b[^]n) = u))
          using val-ring-memE by blast
        then show u ∈ P-set n
          using P-set-def[of n] 2
          by blast
      qed
    qed
  qed
  then show ?thesis using m-pos by blast
qed

definition pow-res where
pow-res (n::nat) x = {a. a ∈ carrier Q_p ∧  (∃ y ∈ nonzero Q_p. (a = x ⊗ (y[^]n)))}

lemma nonzero-pow-res:
  assumes x ∈ nonzero Q_p
```

shows *pow-res* (*n*::*nat*) *x* ⊆ *nonzero* Q_p
proof
fix *a*
assume *a* ∈ *pow-res n x*
then obtain *y* **where** *y-def*: *y* ∈ *nonzero* Q_p ∧ (*a* = *x* ⊗ (*y*[^]*n*))
using *pow-res-def* **by** *blast*
then show *a* ∈ *nonzero* Q_p
using *assms Qp.Units-m-closed Qp-nat-pow-nonzero Units-eq-nonzero* **by** *blast*
qed

lemma *pow-res-of-zero*:
shows *pow-res n* **0** = {**0**}
unfolding *pow-res-def* **apply**(*rule equalityI*)
apply(*rule subsetI*)
unfolding *mem-Collect-eq*
apply (*metis Qp.integral-iff Qp.nat-pow-closed Qp.nonzero-closed Qp.zero-closed insertCI*)
apply(*rule subsetI*) **unfolding** *mem-Collect-eq*
by (*metis Qp.nat-pow-one Qp.one-nonzero Qp.r-one Qp.zero-closed equals0D insertE*)

lemma *equal-pow-resI*:
assumes *x* ∈ *carrier* Q_p
assumes *y* ∈ *pow-res n x*
shows *pow-res n x* = *pow-res n y*
proof
have *y-closed*: *y* ∈ *carrier* Q_p
using *assms* **unfolding** *pow-res-def* **by** *blast*
obtain *c* **where** *c-def*: *c* ∈ *nonzero* Q_p ∧ *y* = *x* ⊗ (*c*[^]*n*)
using *assms pow-res-def* **by** *blast*
have ((*inv c*)[^]*n*) = *inv* (*c*[^]*n*)
using *c-def Qp.field-axioms Qp.nat-pow-of-inv Units-eq-nonzero* **by** *blast*
then have *y* ⊗ ((*inv c*)[^]*n*) = *x*
using *y-closed c-def assms Qp.inv-cancelL*(*2*) *Qp.nonzero-closed Qp-nat-pow-nonzero Units-eq-nonzero*
by *presburger*
then have *P0*: (*inv c*) ∈ *nonzero* Q_p ∧ *x* =*y* ⊗ ((*inv c*)[^]*n*)
using *c-def nonzero-inverse-Qp* **by** *blast*
show *pow-res n x* ⊆ *pow-res n y*
proof
fix *a*
assume *A*: *a* ∈ *pow-res n x*
then have *a* ∈ *carrier* Q_p
by (*metis* (*no-types*, *lifting*) *mem-Collect-eq pow-res-def*)
obtain *b* **where** *b-def*: *b* ∈ *nonzero* Q_p ∧ *a* = *x* ⊗ (*b*[^]*n*)
using *A pow-res-def* **by** *blast*
then have *0*: *b* ∈ *nonzero* Q_p ∧ *a* = *y* ⊗ ((*inv c*)[^]*n*) ⊗ (*b*[^]*n*)
using ‹*y* ⊗ *inv c* [^] *n* = *x*› **by** *blast*
have *b* ∈ *nonzero* Q_p ∧ *a* = *y* ⊗ (((*inv c*) ⊗ *b*)[^]*n*)

proof −
have $(inv\ c)[\widehat{}]n \otimes (b[\widehat{}]n) = ((inv\ c)\ \otimes b)[\widehat{}]n$
using *c-def b-def assms P0 Qp.nat-pow-distrib Qp.nonzero-closed* **by** *presburger*
then have $y \otimes (((inv\ c)[\widehat{}]n) \otimes (b[\widehat{}]n)) = y \otimes (((inv\ c)\ \otimes b)[\widehat{}]n)$
by *presburger*
then show *?thesis*
using *y-closed 0 P0 Qp.m-assoc Qp.nat-pow-closed Qp.nonzero-closed assms(1)* **by** *presburger*
qed
then have $((inv\ c)\ \otimes b) \in nonzero\ Q_p \wedge a = y \otimes (((inv\ c)\ \otimes b)[\widehat{}]n)$
by (*metis P0 Qp.integral-iff Qp.nonzero-closed Qp.nonzero-mult-closed not-nonzero-Qp*)
then show $a \in pow\text{-}res\ n\ y$ **using** *pow-res-def* ‹$a \in carrier\ Q_p$› **by** *blast*
qed
show $pow\text{-}res\ n\ y \subseteq pow\text{-}res\ n\ x$
proof
fix a
assume *A*: $a \in pow\text{-}res\ n\ y$
then have *0*: $a \in carrier\ Q_p$
by (*metis* (*no-types*, *lifting*) *mem-Collect-eq pow-res-def*)
obtain b **where** *b-def*: $b \in nonzero\ Q_p \wedge a = y \otimes (b[\widehat{}]n)$
using *A pow-res-def* **by** *blast*
then have $a = (x \otimes (c[\widehat{}]n)) \otimes (b[\widehat{}]n)$
using *c-def* **by** *blast*
then have $a = x \otimes ((c[\widehat{}]n) \otimes (b[\widehat{}]n))$
by (*meson Qp.m-assoc Qp.nat-pow-closed Qp.nonzero-closed assms(1) b-def c-def*)
then have $a = x \otimes ((c \otimes b)[\widehat{}]n)$
using *Qp.nat-pow-distrib Qp.nonzero-closed b-def c-def* **by** *presburger*
then have $(c \otimes b) \in nonzero\ Q_p \wedge \quad a = x \otimes ((c \otimes b)[\widehat{}]n)$
by (*metis Qp.integral-iff Qp.nonzero-closed Qp.nonzero-mult-closed b-def c-def not-nonzero-Qp*)
then show $a \in pow\text{-}res\ n\ x$
using *pow-res-def 0* **by** *blast*
qed
qed

lemma *zeroth-pow-res*:
assumes $x \in carrier\ Q_p$
shows $pow\text{-}res\ 0\ x = \{x\}$
apply(*rule equalityI*)
apply(*rule subsetI*)
unfolding *pow-res-def mem-Collect-eq*
using *assms* **apply** (*metis Qp.nat-pow-0 Qp.r-one singletonI*)
apply(*rule subsetI*)
unfolding *pow-res-def mem-Collect-eq*
using *assms* **by** (*metis Qp.nat-pow-0 Qp.one-nonzero Qp.r-one equals0D insertE*)

lemma *Zp-car-zero-res*: **assumes** $x \in carrier\ Z_p$

shows $x\ 0 = 0$
using *assms* **unfolding** *Zp-def*
using *Zp-def Zp-defs(3) padic-set-zero-res prime* **by** *blast*

lemma *zeroth-ac*:
assumes $x \in carrier\ Q_p$
shows $ac\ 0\ x = 0$
apply(*cases* $x = \mathbf{0}$)
unfolding *ac-def* **apply** *presburger*
using *assms angular-component-closed*[*of x*] *Zp-car-zero-res* **unfolding** *nonzero-def mem-Collect-eq*
by *presburger*

lemma *nonzero-ac-imp-nonzero*:
assumes $x \in carrier\ Q_p$
assumes $ac\ m\ x \neq 0$
shows $x \in nonzero\ Q_p$
using *assms* **unfolding** *ac-def nonzero-def mem-Collect-eq*
by *presburger*

lemma *nonzero-ac-val-ord*:
assumes $x \in carrier\ Q_p$
assumes $ac\ m\ x \neq 0$
shows $val\ x = ord\ x$
using *nonzero-ac-imp-nonzero assms val-ord* **by** *blast*

lemma *pow-res-equal-ord*:
assumes $n > 0$
shows $\exists m > 0.\ \forall x\ y.\ x \in nonzero\ Q_p \wedge y \in nonzero\ Q_p \wedge ac\ m\ x = ac\ m\ y \wedge ord\ x = ord\ y \longrightarrow pow\text{-}res\ n\ x = pow\text{-}res\ n\ y$
proof−
obtain m **where** *m-def-0*: $m > 0 \wedge (\ \forall\ u \in carrier\ Q_p.\ ac\ m\ u = 1 \wedge val\ u = 0 \longrightarrow u \in P\text{-}set\ n)$
using *assms nth-power-fact*[*of n*]
by (*metis less-imp-le-nat less-one linorder-neqE-nat nat-le-linear zero-less-iff-neq-zero*)
then have *m-def*: $m > 0 \wedge (\ \forall\ u \in carrier\ Q_p.\ ac\ m\ u = 1 \wedge ord\ u = 0 \longrightarrow u \in P\text{-}set\ n)$
by (*smt nonzero-ac-val-ord zero-eint-def*)
have $\forall x\ y.\ x \in nonzero\ Q_p \wedge y \in nonzero\ Q_p \wedge ac\ m\ x = ac\ m\ y \wedge ord\ x = ord\ y \longrightarrow pow\text{-}res\ n\ x = pow\text{-}res\ n\ y$
proof
fix x
show $\forall y.\ x \in nonzero\ Q_p \wedge y \in nonzero\ Q_p \wedge ac\ m\ x = ac\ m\ y \wedge ord\ x = ord\ y \longrightarrow pow\text{-}res\ n\ x = pow\text{-}res\ n\ y$
proof fix y
show $x \in nonzero\ Q_p \wedge y \in nonzero\ Q_p \wedge ac\ m\ x = ac\ m\ y \wedge ord\ x = ord\ y \longrightarrow pow\text{-}res\ n\ x = pow\text{-}res\ n\ y$
proof
assume *A*: $x \in nonzero\ Q_p \wedge y \in nonzero\ Q_p \wedge ac\ m\ x = ac\ m\ y \wedge ord\ x$

$= ord\ y$
then have *0*: $ac\ m\ (x \div y) = 1$
using *ac-inv*[*of y m*] *ac-mult*
by (*metis ac-inv'''*(*1*) *ac-mult' m-def nonzero-inverse-Qp*)
have *1*: $ord\ (x \div y) = 0$
using *A ord-fract* **by** *presburger*
have *2*: $(x \div y) \in nonzero\ Q_p$
using *A*
by (*metis Qp.nonzero-closed Qp.nonzero-mult-closed local.fract-cancel-right nonzero-inverse-Qp not-nonzero-Qp zero-fract*)
have *3*: $(x \div y) \in P\text{-}set\ n$
using *m-def 0 1 2 nonzero-def*
by (*smt Qp.nonzero-closed*)
then obtain *b* **where** *b-def*: $b \in carrier\ Q_p \wedge (b[\frown]n) = (x \div y)$
using *P-set-def*
by *blast*
then have $(x \div y) \otimes y = (b[\frown]n) \otimes y$
by *presburger*
then have $x = (b[\frown]n) \otimes y$
using *A b-def*
by (*metis Qp.nonzero-closed local.fract-cancel-left*)
then have $x = y \otimes (b[\frown]n)$
using *A b-def*
by (*metis Qp.nonzero-closed local.fract-cancel-right*)
then have $x \in pow\text{-}res\ n\ y$
unfolding *pow-res-def* **using** *A b-def*
by (*metis* (*mono-tags*, *lifting*) *2 Qp.nat-pow-0 Qp.nonzero-closed Qp-nonzero-nat-pow mem-Collect-eq not-gr-zero*)
then show $pow\text{-}res\ n\ x = pow\text{-}res\ n\ y$
using *A equal-pow-resI*[*of x y n*] **unfolding** *nonzero-def*
by (*metis* (*mono-tags*, *lifting*) *A Qp.nonzero-closed equal-pow-resI*)
qed
qed
qed
then show *?thesis* **using** *m-def* **by** *blast*
qed

lemma *pow-res-equal*:
assumes $n > 0$
shows $\exists m{>}\ 0.\ \forall x\ y.\ x \in nonzero\ Q_p \wedge y \in nonzero\ Q_p \wedge ac\ m\ x = ac\ m\ y \wedge ord\ x = (ord\ y\ mod\ n) \longrightarrow pow\text{-}res\ n\ x = pow\text{-}res\ n\ y$
proof−
obtain *m* **where** *m-def*: $m > 0 \wedge (\forall x\ y.\ x \in nonzero\ Q_p \wedge y \in nonzero\ Q_p \wedge ac\ m\ x = ac\ m\ y \wedge ord\ x = ord\ y \longrightarrow pow\text{-}res\ n\ x = pow\text{-}res\ n\ y)$
using *assms pow-res-equal-ord*
by *meson*
have $\forall x\ y.\ x \in nonzero\ Q_p \wedge y \in nonzero\ Q_p \wedge ac\ m\ x = ac\ m\ y \wedge ord\ x = ord\ y\ mod\ int\ n \longrightarrow pow\text{-}res\ n\ x = pow\text{-}res\ n\ y$
proof fix *x*

show $\forall y.\ x \in nonzero\ Q_p \wedge y \in nonzero\ Q_p \wedge ac\ m\ x = ac\ m\ y \wedge ord\ x = ord\ y\ mod\ int\ n \longrightarrow pow\text{-}res\ n\ x = pow\text{-}res\ n\ y$
proof fix y
show $x \in nonzero\ Q_p \wedge y \in nonzero\ Q_p \wedge ac\ m\ x = ac\ m\ y \wedge ord\ x = ord\ y\ mod\ int\ n \longrightarrow pow\text{-}res\ n\ x = pow\text{-}res\ n\ y$
proof
assume *A*: $x \in nonzero\ Q_p \wedge y \in nonzero\ Q_p \wedge ac\ m\ x = ac\ m\ y \wedge ord\ x = ord\ y\ mod\ int\ n$
show $pow\text{-}res\ n\ x = pow\text{-}res\ n\ y$
proof−
have *A0*: $x \in nonzero\ Q_p \wedge y \in nonzero\ Q_p$
using *A* **by** *blast*
have *A1*: $ac\ m\ x = ac\ m\ y$
using *A* **by** *blast*
have *A2*: $ord\ x = ord\ y\ mod\ int\ n$
using *A* **by** *blast*
obtain *k* **where** *k-def*: $k = ord\ x$
by *blast*
obtain *l* **where** *l-def*: $ord\ y = ord\ x + (l::int)*(int\ n)$
using *assms A2*
by (*metis A k-def mod-eqE mod-mod-trivial mult-of-nat-commute*)
have *m-def'*: $\bigwedge x\ y.\ x \in nonzero\ Q_p \wedge y \in nonzero\ Q_p \wedge ac\ m\ x = ac\ m\ y \wedge ord\ x = ord\ y \Longrightarrow pow\text{-}res\ n\ x = pow\text{-}res\ n\ y$
using *m-def*
by *blast*
have *0*: $ord\ (y \otimes (\mathfrak{p}[\hat{}](-\ l*n))) = ord\ x$
proof−
have *0*: $ord\ (y \otimes (\mathfrak{p}[\hat{}](-\ l*n))) = ord\ y + (ord\ \ (\mathfrak{p}[\hat{}](-\ l*n)))$
using *ord-mult p-nonzero A0 Qp-int-pow-nonzero*
by *blast*
have *1*: $ord\ (\mathfrak{p}[\hat{}](-\ l*n)) = -\ l*n$
using *ord-p-pow-int*[*of* $-l*n$]
by *blast*
then have $ord\ (y \otimes (\mathfrak{p}[\hat{}](-\ l*n))) = ord\ y - l*n$
using *0*
by *linarith*
then show *?thesis*
using *k-def l-def* **by** *linarith*
qed
have *1*: $ac\ m\ (y \otimes (\mathfrak{p}[\hat{}](-\ l*n))) = ac\ m\ y$
using *assms ac-p-int-pow-factor-right*[*of*] *m-def A Qp.nonzero-closed* **by** *presburger*
have *2*: $y \otimes (\mathfrak{p}[\hat{}](-\ l*n)) \in nonzero\ Q_p$
using *A0 Qp-int-pow-nonzero*[*of* $\mathfrak{p}$ $-$ $l*n$] *Qp.cring-axioms nonzero-def cring.cring-simprules*(*5*)
fract-cancel-left not-nonzero-Qp p-intpow-inv'' p-nonzero zero-fract Qp.integral-iff
Qp.nonzero-closed Qp.nonzero-memE(*2*) *Qp.nonzero-memI Qp.nonzero-mult-closed minus-mult-commute mult-minus-right p-intpow-closed*(*1*) *p-intpow-closed*(*2*)

```
      by presburger
    then have 3: pow-res n (y ⊗ (𝔭[^](− l*n))) = pow-res n x
      using 2 A0 m-def'[of y ⊗ (𝔭[^](− l*n)) x] 0 1 A1
      by linarith
    have 4: (y ⊗ (𝔭[^](− l*n))) = (y ⊗ ((𝔭[^]− l)[^]n))
      using Qp-int-nat-pow-pow[of 𝔭 −l n] p-nonzero
      by presburger
    have y ⊗ (𝔭[^](− l*n))∈ pow-res n y
      using 2 4 Qp-int-pow-nonzero nonzero-def p-nonzero
      unfolding pow-res-def nonzero-def
    proof −
      assume a1: ⋀x n. x ∈ {a ∈ carrier Q_p. a ≠ 𝟬} ⟹ x [^] (n::int) ∈ {a
∈ carrier Q_p. a ≠ 𝟬}
      assume a2: 𝔭 ∈ {a ∈ carrier Q_p. a ≠ 𝟬}
      assume a3: y ⊗ 𝔭 [^] (− l * int n) ∈ {a ∈ carrier Q_p. a ≠ 𝟬}
      have f4: 𝔭 [^] (− 1 * l) ∈ {r ∈ carrier Q_p. r ≠ 𝟬}
        using a2 a1 by presburger
      have f5: − l = − 1 * l
        by linarith
      then have f6: y ⊗ 𝔭 [^] (− 1 * l * int n) = y ⊗ (𝔭 [^] (− 1 * l)) [^] n
        using ‹y ⊗ 𝔭 [^] (− l * int n) = y ⊗ (𝔭 [^] − l) [^] n› by presburger
      then have y ⊗ (𝔭 [^] (− 1 * l)) [^] n ∈ {r ∈ carrier Q_p. r ≠ 𝟬}
        using f5 a3 by presburger
      then have y ⊗ (𝔭 [^] (− 1 * l)) [^] n ∈ {r ∈ carrier Q_p. ∃ra. ra ∈ {r
∈ carrier Q_p. r ≠ 𝟬} ∧ r = y ⊗ ra [^] n}
        using f4 by blast
      then have y ⊗ 𝔭 [^] (− l * int n) ∈ {r ∈ carrier Q_p. ∃ra. ra ∈ {r ∈
carrier Q_p. r ≠ 𝟬} ∧ r = y ⊗ ra [^] n}
        using f6 f5 by presburger
      then show y ⊗ 𝔭 [^] (− l * int n) ∈ {r ∈ carrier Q_p. ∃ra∈{r ∈ carrier
Q_p. r ≠ 𝟬}. r = y ⊗ ra [^] n}
        by meson
    qed
    then have pow-res n (y ⊗ (𝔭[^](− l*n))) = pow-res n y
      using equal-pow-resI[of (y ⊗ (𝔭[^](− l*n))) y n] 2 A0 assms
          Qp.nonzero-mult-closed p-intpow-closed(2)
     by (metis (mono-tags, opaque-lifting) 3 A Qp.nonzero-closed equal-pow-resI)
    then show ?thesis using 3 by blast
  qed
 qed
qed
qed
then show ?thesis
  using m-def
  by blast
qed

definition pow-res-classes where
pow-res-classes n = {S. ∃x ∈ nonzero Q_p. S = pow-res n x }
```

lemma *pow-res-semialg-def*:
assumes $x \in$ *nonzero* Q_p
assumes $n \geq 1$
shows $\exists P \in$ *carrier* Q_p*-x. pow-res n x* = (*univ-basic-semialg-set n P*) − {**0**}
proof−
have *0*: *pow-res n x* = {$a \in$ *carrier* Q_p. $\exists y \in$*nonzero* Q_p. (*inv x*) ⊗ $a = (y[\widehat{}]n)$}
proof
show *pow-res n x* ⊆ {$a \in$ *carrier* Q_p. $\exists y \in$*nonzero* Q_p. *inv x* ⊗ $a = (y[\widehat{}]n)$}
proof
fix *a*
assume *A*: $a \in$ *pow-res n x*
then have $a \in$ *carrier* $Q_p \wedge$ ($\exists y \in$*nonzero* Q_p. $a = x \otimes (y[\widehat{}]n)$)
unfolding *pow-res-def*
by *blast*
then obtain *y* **where** *y-def*: $y \in$ *nonzero* $Q_p \wedge a = x \otimes (y[\widehat{}]n)$
by *blast*
then have $y \in$ *nonzero* $Q_p \wedge$ *inv x* ⊗ $a = (y[\widehat{}]n)$
proof −
show *?thesis*
by (*metis* (*no-types*, *opaque-lifting*) *Qp.m-assoc Qp.m-comm Qp.nat-pow-closed Qp.nonzero-closed*
‹$a \in$ *carrier* $Q_p \wedge$ ($\exists y \in$*nonzero* Q_p. $a = x \otimes y\ [\widehat{}]\ n$)› *assms*(*1*) *local.fract-cancel-right nonzero-inverse-Qp y-def*)
qed
then show $a \in$ {$a \in$ *carrier* Q_p. $\exists y \in$*nonzero* Q_p. *inv x* ⊗ $a = (y[\widehat{}]n)$}
using *assms* ‹$a \in$ *carrier* $Q_p \wedge$ ($\exists y \in$*nonzero* Q_p. $a = x \otimes (y[\widehat{}]n)$)›
by *blast*
qed

show {$a \in$ *carrier* Q_p. $\exists y \in$*nonzero* Q_p. *inv x* ⊗ $a = (y[\widehat{}]n)$} ⊆ *pow-res n x*
proof
fix *a*
assume *A*: $a \in$ {$a \in$ *carrier* Q_p. $\exists y \in$*nonzero* Q_p. *inv x* ⊗ $a = (y[\widehat{}]n)$}
show $a \in$ *pow-res n x*
proof−
have $a \in$ *carrier* $Q_p \wedge$ ($\exists y \in$*nonzero* Q_p. *inv x* ⊗ $a = (y[\widehat{}]n)$)
using *A* **by** *blast*
then obtain *y* **where** *y-def*: $y \in$*nonzero* $Q_p \wedge$ *inv x* ⊗ $a = (y[\widehat{}]n)$
by *blast*
then have $y \in$*nonzero* $Q_p \wedge$ $a = x \otimes (y[\widehat{}]n)$
by (*metis Qp.l-one Qp.m-assoc Qp.nonzero-closed Qp.not-nonzero-memI*
‹$a \in$ *carrier* $Q_p \wedge$ ($\exists y \in$*nonzero* Q_p. *inv x* ⊗ $a = y\ [\widehat{}]\ n$)› *assms*(*1*) *field-inv*(*2*) *inv-in-frac*(*1*))
then show *?thesis*
by (*metis* (*mono-tags*, *lifting*) ‹$a \in$ *carrier* $Q_p \wedge$ ($\exists y \in$*nonzero* Q_p. *inv x* ⊗ $a = (y[\widehat{}]n)$)› *mem-Collect-eq pow-res-def*)
qed
qed

qed
obtain P **where** *P-def*: $P = up\text{-}ring.monom\ Q_p\text{-}x\ (inv\ x)\ 1$
by *blast*
have *P-closed*: $P \in carrier\ Q_p\text{-}x$
using *P-def Qp.nonzero-closed Qp.nonzero-memE(2) UPQ.is-UP-monomE(1) UPQ.is-UP-monomI assms(1) inv-in-frac(1)* **by** *presburger*
have *P-eval*: $\bigwedge a.\ a \in carrier\ Q_p \Longrightarrow (P \cdot a) = (inv\ x) \otimes a$
using *P-def to-fun-monom*[*of*]
by (*metis Qp.nat-pow-eone Qp.nonzero-closed Qp.not-nonzero-memI assms(1) inv-in-frac(1)*)
have *0*: *pow-res n x* $= \{a \in carrier\ Q_p.\ \exists\, y{\in}nonzero\ Q_p.\ (P \cdot a) = (y[\widehat{}]n)\}$
proof
show *pow-res n x* $\subseteq \{a \in carrier\ Q_p.\ \exists\, y{\in}nonzero\ Q_p.\ P \cdot a = (y[\widehat{}]n)\}$
proof fix a
assume $a \in$ *pow-res n x*
then have $a \in carrier\ Q_p \wedge (\exists\, y{\in}nonzero\ Q_p.\ inv\ x \otimes a = (y[\widehat{}]n))$
using *0*
by *blast*
then show $a \in \{a \in carrier\ Q_p.\ \exists\, y{\in}nonzero\ Q_p.\ P \cdot a = (y[\widehat{}]n)\}$
using *P-eval*
by (*metis* (*mono-tags*, *lifting*) *mem-Collect-eq*)
qed
show $\{a \in carrier\ Q_p.\ \exists\, y{\in}nonzero\ Q_p.\ P \cdot a = (y[\widehat{}]n)\} \subseteq$ *pow-res n x*
proof fix a
assume $a \in \{a \in carrier\ Q_p.\ \exists\, y{\in}nonzero\ Q_p.\ P \cdot a = (y[\widehat{}]n)\}$
then obtain y **where** *y-def*: $y{\in}nonzero\ Q_p \wedge P \cdot a = (y[\widehat{}]n)$
by *blast*
then have $y{\in}nonzero\ Q_p \wedge inv\ x \otimes a = (y[\widehat{}]n)$
using *P-eval* ‹$a \in \{a \in carrier\ Q_p.\ \exists\, y{\in}nonzero\ Q_p.\ P \cdot a = (y[\widehat{}]n)\}$›
by *blast*
then have $a \in carrier\ Q_p \wedge (\exists\, y{\in}nonzero\ Q_p.\ inv\ x \otimes a = (y[\widehat{}]n))$
using ‹$a \in \{a \in carrier\ Q_p.\ \exists\, y{\in}nonzero\ Q_p.\ P \cdot a = (y[\widehat{}]n)\}$› **by** *blast*
then show $a \in$ *pow-res n x*
using *0*
by *blast*
qed
qed
have *1*: *univ-basic-semialg-set n P* $- \{\mathbf{0}\} = \{a \in carrier\ Q_p.\ \exists\, y{\in}nonzero\ Q_p.\ (P \cdot a) = (y[\widehat{}]n)\}$
proof
show *univ-basic-semialg-set n P* $- \{\mathbf{0}\} \subseteq \{a \in carrier\ Q_p.\ \exists\, y{\in}nonzero\ Q_p.\ P \cdot a = (y[\widehat{}]n)\}$
proof
fix a
assume *A*: $a \in$ *univ-basic-semialg-set n P* $- \{\mathbf{0}\}$
then have *A0*: $a \in carrier\ Q_p \wedge (\exists\, y{\in}carrier\ Q_p.\ P \cdot a = (y[\widehat{}]n))$
unfolding *univ-basic-semialg-set-def* **by** *blast*
then have *A0′*: $a \in nonzero\ Q_p \wedge (\exists\, y{\in}carrier\ Q_p.\ P \cdot a = (y[\widehat{}]n))$
using *A*

by (*metis DiffD2 not-nonzero-Qp singletonI*)
then obtain *y* **where** *y-def*: *y*∈*carrier* $Q_p \wedge P \cdot a = (y[\widehat{}]n)$
by *blast*
have *A1*: $(P \cdot a) \neq \mathbf{0}$
using *P-eval A0′ Qp.integral-iff Qp.nonzero-closed Qp.nonzero-memE*(*2*) *assms*(*1*) *inv-in-frac*(*1*) *inv-in-frac*(*2*) **by** *presburger*
have *A2*: $y \in$ *nonzero* Q_p
proof−
have *A20*: $(y[\widehat{}]n) \neq \mathbf{0}$
using *A1 y-def*
by *blast*
have $y \neq \mathbf{0}$
apply(*rule ccontr*) **using** *A20 assms*
by (*metis Qp.nat-pow-eone Qp.semiring-axioms Qp.zero-closed le-zero-eq semiring.nat-pow-zero*)
then show *?thesis*
using *y-def A1 not-nonzero-Qp Qp.not-nonzero-memE* **by** *blast*
qed
then have $y \in$ *nonzero* $Q_p \wedge P \cdot a = (y[\widehat{}]n)$ **using** *y-def*
by *blast*
then show $a \in \{a \in$ *carrier* Q_p. $\exists\, y \in$ *nonzero* Q_p. $P \cdot a = (y[\widehat{}]n)\}$
using *A0 nonzero-def*
by *blast*
qed
show $\{a \in$ *carrier* Q_p. $\exists\, y \in$ *nonzero* Q_p. $P \cdot a = (y[\widehat{}]n)\} \subseteq$ *univ-basic-semialg-set n P* − $\{\mathbf{0}\}$
proof
fix *a*
assume *A*: $a \in \{a \in$ *carrier* Q_p. $\exists\, y \in$ *nonzero* Q_p. $P \cdot a = (y[\widehat{}]n)\}$
then obtain *y* **where** *y-def*: *y*∈*nonzero* $Q_p \wedge P \cdot a = (y[\widehat{}]n)$
by *blast*
then have $y \neq \mathbf{0} \wedge y \in$ *carrier* $Q_p \wedge P \cdot a = (y[\widehat{}]n)$
by (*metis* (*mono-tags*, *opaque-lifting*) *Qp.nonzero-closed Qp.not-nonzero-memI*)
then have $a \neq \mathbf{0}$
using *P-eval*
by (*metis Qp.m-comm Qp.nonzero-closed Qp.nonzero-memE*(*2*) *Qp.nonzero-pow-nonzero Qp.zero-closed assms*(*1*) *inv-in-frac*(*1*) *zero-fract*)
then show $a \in$ *univ-basic-semialg-set n P* − $\{\mathbf{0}\}$
unfolding *univ-basic-semialg-set-def*
using *A* ‹$y \neq \mathbf{0} \wedge y \in$ *carrier* $Q_p \wedge P \cdot a = (y[\widehat{}]n)$›
by *blast*
qed
qed
show *?thesis* **using** *0 1*
by (*metis* (*no-types*, *lifting*) *P-closed*)
qed

lemma *pow-res-is-univ-semialgebraic*:
assumes $x \in$ *carrier* Q_p

shows *is-univ-semialgebraic* (*pow-res n x*)
proof(*cases n = 0*)
case *True*
have *T0*: *pow-res n x* = {*x*}
unfolding *True* **using** *assms*
by (*simp add*: *assms zeroth-pow-res*)
have [*x*] ∈ *carrier* ($Q_p{}^1$)
using *assms Qp.to-R1-closed* **by** *blast*
hence *is-semialgebraic 1* {[*x*]}
using *is-algebraic-imp-is-semialg singleton-is-algebraic* **by** *blast*
thus *?thesis* **unfolding** *T0* **using** *assms*
by (*simp add*: ‹*x* ∈ *carrier* Q_p› *finite-is-univ-semialgebraic*)
next
case *False*
show *?thesis*
proof(*cases x* = **0**)
case *True*
then show *?thesis* **using** *finite-is-univ-semialgebraic False pow-res-of-zero*
by (*metis Qp.zero-closed empty-subsetI finite.emptyI finite.insertI insert-subset*)
next
case *F*: *False*
then show *?thesis*
using *False pow-res-semialg-def*[*of x n*] *diff-is-univ-semialgebraic*[*of* - {**0**}]
finite-is-univ-semialgebraic[*of* {**0**}]
by (*metis Qp.zero-closed assms empty-subsetI finite.emptyI finite.insertI in-*
sert-subset less-one less-or-eq-imp-le linorder-neqE-nat not-nonzero-Qp univ-basic-semialg-set-is-univ-semialgeb
qed
qed

lemma *pow-res-is-semialg*:
assumes *x* ∈ *carrier* Q_p
shows *is-semialgebraic 1* (*to-R1* ‘ (*pow-res n x*))
using *assms pow-res-is-univ-semialgebraic is-univ-semialgebraicE*
by *blast*

lemma *pow-res-refl*:
assumes *x* ∈ *carrier* Q_p
shows *x* ∈ *pow-res n x*
proof−
have *x* = *x* ⊗ (**1** [^]*n*)
using *assms Qp.nat-pow-one Qp.r-one* **by** *presburger*
thus *?thesis*
using *assms* **unfolding** *pow-res-def mem-Collect-eq*
using *Qp.one-nonzero* **by** *blast*
qed

lemma *equal-pow-resE*:
assumes *a* ∈ *carrier* Q_p
assumes *b* ∈ *carrier* Q_p

assumes $n > 0$
assumes *pow-res n a* = *pow-res n b*
shows $\exists\ s \in$ *P-set n*. $a = b \otimes s$
proof−
have $a \in$ *pow-res n b*
using *assms pow-res-refl* **by** *blast*
then obtain y **where** *y-def*: $y \in$ *nonzero* $Q_p \wedge a = b \otimes y[\hat{}]n$
unfolding *pow-res-def* **by** *blast*
thus *?thesis* **unfolding** *P-set-def*
using *Qp.nonzero-closed Qp-nat-pow-nonzero* **by** *blast*
qed

lemma *pow-res-one*:
assumes $x \in$ *nonzero* Q_p
shows *pow-res 1 x* = *nonzero* Q_p
proof show *pow-res 1 x* $\subseteq$ *nonzero* Q_p
using *assms nonzero-pow-res*[*of x 1*] **by** *blast*
show *nonzero* $Q_p \subseteq$ *pow-res 1 x*
proof fix y **assume** *A*: $y \in$ *nonzero* Q_p
then have *0*: $\mathbf{1} \in$ *nonzero* $Q_p \wedge y = x \otimes ((inv\ x) \otimes y)[\hat{}](1::nat)$
using *assms Qp.m-comm Qp.nat-pow-eone Qp.nonzero-closed Qp.nonzero-mult-closed Qp.one-nonzero local.fract-cancel-right nonzero-inverse-Qp* **by** *presburger*
have *1*: $(inv\ x) \otimes y \in$ *nonzero* Q_p
using *A assms* **by** (*metis Qp.Units-m-closed Units-eq-nonzero nonzero-inverse-Qp*)
then show $y \in$ *pow-res 1 x*
unfolding *pow-res-def* **using** *0 1 A Qp.nonzero-closed* **by** *blast*
qed
qed

lemma *pow-res-zero*:
assumes $n > 0$
shows *pow-res n* $\mathbf{0} = \{\mathbf{0}\}$
proof
show *pow-res n* $\mathbf{0} \subseteq \{\mathbf{0}\}$
unfolding *pow-res-def*
using *Qp.l-null Qp.nat-pow-closed Qp.nonzero-closed* **by** *blast*
show $\{\mathbf{0}\} \subseteq$ *pow-res n* $\mathbf{0}$
using *assms* **unfolding** *pow-res-def*
using *Qp.l-null Qp.one-closed Qp.one-nonzero empty-subsetI insert-subset* **by** *blast*
qed

lemma *equal-pow-resI'*:
assumes $a \in$ *carrier* Q_p
assumes $b \in$ *carrier* Q_p
assumes $c \in$ *P-set n*
assumes $a = b \otimes c$

assumes $n > 0$
shows *pow-res n a = pow-res n b*
proof−
obtain *y* **where** *y-def*: $c = y[\widehat{}]n \wedge y \in carrier\ Q_p$
using *assms* **unfolding** *P-set-def* **by** *blast*
have *c-nonzero*: $c \in nonzero\ Q_p$
using *P-set-nonzero'*(*1*) *assms*(*3*) **by** *blast*
have *y-nonzero*: $y \in nonzero\ Q_p$
using *y-def c-nonzero Qp-nonzero-nat-pow assms*(*5*) **by** *blast*
have *0*: $a \in pow\text{-}res\ n\ b$
using *assms y-nonzero y-def* **unfolding** *pow-res-def*
by *blast*
show *?thesis*
apply(*cases* $b = \mathbf{0}$)
using *pow-res-zero Qp.l-null Qp.nonzero-closed assms*(*4*) *c-nonzero* **apply** *presburger*
by (*metis 0 assms*(*1*) *assms*(*2*) *assms*(*5*) *not-nonzero-Qp equal-pow-resI*)
qed

lemma *equal-pow-resI''*:
assumes $n > 0$
assumes $a \in nonzero\ Q_p$
assumes $b \in nonzero\ Q_p$
assumes $a \otimes inv\ b \in P\text{-}set\ n$
shows *pow-res n a = pow-res n b*
using *assms equal-pow-resI'*[*of a b a* $\otimes$ *inv b n*] *Qp.nonzero-closed local.fract-cancel-right*
by *blast*

lemma *equal-pow-resI'''*:
assumes $n > 0$
assumes $a \in nonzero\ Q_p$
assumes $b \in nonzero\ Q_p$
assumes $c \in nonzero\ Q_p$
assumes *pow-res n* $(c \otimes a)$ = *pow-res n* $(c \otimes b)$
shows *pow-res n a = pow-res n b*
proof−
have *0*: $c \otimes a \in nonzero\ Q_p$
by (*meson Localization.submonoid.m-closed Qp.nonzero-is-submonoid assms*(*2*) *assms*(*4*))
have *1*: $c \otimes b \in nonzero\ Q_p$
by (*meson Localization.submonoid.m-closed Qp.nonzero-is-submonoid assms*(*3*) *assms*(*4*))
have $c \otimes a \in pow\text{-}res\ n\ (c \otimes b)$
proof(*cases n = 1*)
case *True*
then show *?thesis*
using *assms 0 1 pow-res-one*[*of* $c \otimes b$] **by** *blast*
next
case *False*

```
    then have n ≥ 2
      using assms(1) by linarith
    then show ?thesis
      using assms 0 1 pow-res-refl[of c⊗a n] unfolding nonzero-def
      by blast
  qed
  then obtain y where y-def: y ∈ nonzero Q_p ∧ (c ⊗ a) = (c ⊗ b)⊗y[^]n
    using assms unfolding pow-res-def by blast
  then have a = b⊗y[^]n
    using assms
    by (metis Qp.m-assoc Qp.m-lcancel Qp.nonzero-closed Qp.nonzero-mult-closed
Qp.not-nonzero-memI Qp-nat-pow-nonzero)
  then show ?thesis
   by (metis P-set-memI Qp.nonzero-closed Qp.nonzero-mult-closed Qp.not-nonzero-memI
Qp-nat-pow-nonzero assms(1) assms(3) equal-pow-resI' y-def)
qed

lemma equal-pow-resI'''':
  assumes n > 0
  assumes a ∈ carrier Q_p
  assumes b ∈ carrier Q_p
  assumes a = b ⊗ u
  assumes u ∈ P-set n
  shows pow-res n a = pow-res n b
proof(cases a = 0)
  case True
  then have b = 0
    using assms unfolding P-set-def
   by (metis (no-types, lifting) Qp.integral Qp.nonzero-closed Qp.not-nonzero-memI
mem-Collect-eq)
  then show ?thesis using pow-res-zero
    using True by blast
next
  case False
  then have 0: a ∈ nonzero Q_p
    using Qp.not-nonzero-memE assms(2) by blast
  have 1: b ∈ nonzero Q_p
    using 0 assms unfolding P-set-def
     by (metis (no-types, lifting) Qp.integral-iff Qp.nonzero-closed mem-Collect-eq
not-nonzero-Qp)
  have 2: a ⊗ (inv b)∈ P-set n
    using assms 0 1
   by (metis P-set-nonzero'(2) Qp.inv-cancelR(1) Qp.m-comm Qp.nonzero-memE(2)
Units-eq-nonzero inv-in-frac(1))
  then show ?thesis using equal-pow-resI''
    by (meson 0 1 assms(1) equal-pow-resI)
qed
```

lemma *Zp-Units-ord-zero*:
assumes $a \in$ *Units* Z_p
shows *ord-Zp* $a = 0$
proof−
have $inv_{Z_p}\ a \in$ *nonzero* Z_p
apply(*rule Zp.nonzero-memI*, *rule Zp.Units-inv-closed*, *rule assms*)
using *assms Zp.Units-inverse in-Units-imp-not-zero* **by** *blast*
then have *ord-Zp* $(a \otimes_{Z_p} inv_{Z_p}\ a) =$ *ord-Zp* $a +$ *ord-Zp* $(inv_{Z_p}\ a)$
using *assms ord-Zp-mult Zp.Units-nonzero zero-not-one*
by (*metis Zp.zero-not-one*)
then show *?thesis*
by (*smt Zp.Units-closed Zp.Units-r-inv Zp.integral-iff Zp.nonzero-closed* ‹inv_{Z_p} $a \in$ *nonzero* Z_p› *assms ord-Zp-one ord-pos*)
qed

lemma *pow-res-nth-pow*:
assumes $a \in$ *nonzero* Q_p
assumes $n > 0$
shows *pow-res* n $(a[\widehat{}]n) =$ *pow-res* n **1**
proof
show *pow-res* n $(a\ [\widehat{}]\ n) \subseteq$ *pow-res* n **1**
proof fix x **assume** A: $x \in$ *pow-res* n $(a\ [\widehat{}]\ n)$
then show $x \in$ *pow-res* n **1**
by (*metis P-set-memI Qp.l-one Qp.nat-pow-closed Qp.nonzero-closed Qp.nonzero-memE*(2)
Qp.nonzero-pow-nonzero Qp.one-closed assms(1) *assms*(2) *equal-pow-resI'*)
qed
show *pow-res* n **1** $\subseteq$ *pow-res* n $(a\ [\widehat{}]\ n)$
proof fix x **assume** A: $x \in$ *pow-res* n **1**
then obtain y **where** *y-def*: $y \in$ *nonzero* $Q_p \wedge x = \mathbf{1} \otimes y[\widehat{}]n$
unfolding *pow-res-def* **by** *blast*
then have *0*: $x = y[\widehat{}]n$
using *Qp.l-one Qp.nonzero-closed* **by** *blast*
have $y[\widehat{}]n = a[\widehat{}]n \otimes (inv\ a \otimes y)[\widehat{}]n$
proof−
have $a[\widehat{}]n \otimes (inv\ a \otimes y)[\widehat{}]n = a[\widehat{}]n \otimes (inv\ a)[\widehat{}]n \otimes y[\widehat{}]n$
using *Qp.Units-inv-closed Qp.m-assoc Qp.nat-pow-closed Qp.nat-pow-distrib*
Qp.nonzero-closed Units-eq-nonzero assms(1) *y-def* **by** *presburger*
then show *?thesis*
by (*metis Qp.Units-inv-inv Qp.inv-cancelR*(1) *Qp.nat-pow-distrib Qp.nonzero-closed*
Qp.nonzero-mult-closed Units-eq-nonzero assms(1) *nonzero-inverse-Qp y-def*)
qed
then show $x \in$ *pow-res* n $(a\ [\widehat{}]\ n)$
using *y-def A assms* **unfolding** *pow-res-def mem-Collect-eq*
by (*metis 0 Qp.integral Qp.m-closed Qp.nonzero-closed Qp.not-nonzero-memI*
inv-in-frac(1) *inv-in-frac*(2) *not-nonzero-Qp*)
qed
qed

lemma *pow-res-of-p-pow*:

assumes $n > 0$
shows *pow-res n* (𝔭[^]((l::*int*)∗n)) = *pow-res n* **1**
proof−
have *0*: 𝔭[^]((l::*int*)∗n) = (𝔭[^]l)[^]n
using *Qp-p-int-nat-pow-pow* **by** *blast*
have 𝔭[^]((l::*int*)∗n) ∈ *P-set n*
using *P-set-memI*[*of* - 𝔭[^]l]
by (*metis 0 Qp.not-nonzero-memI Qp-int-pow-nonzero p-intpow-closed*(*1*) *p-nonzero*)
thus *?thesis*
using *0 assms p-intpow-closed*(*2*) *pow-res-nth-pow* **by** *presburger*
qed

lemma *pow-res-nonzero*:
assumes $n > 0$
assumes $a \in$ *nonzero* Q_p
assumes $b \in$ *carrier* Q_p
assumes *pow-res n a* = *pow-res n b*
shows $b \in$ *nonzero* Q_p
using *assms nonzero-pow-res*[*of a n*] *pow-res-zero*[*of n*]
by (*metis insert-subset not-nonzero-Qp*)

lemma *pow-res-mult*:
assumes $n > 0$
assumes $a \in$ *carrier* Q_p
assumes $b \in$ *carrier* Q_p
assumes $c \in$ *carrier* Q_p
assumes $d \in$ *carrier* Q_p
assumes *pow-res n a* = *pow-res n c*
assumes *pow-res n b* = *pow-res n d*
shows *pow-res n* ($a \otimes b$) = *pow-res n* ($c \otimes d$)
proof(*cases a* ∈ *nonzero* Q_p)
case *True*
then have $c \in$ *nonzero* Q_p
using *assms pow-res-nonzero* **by** *blast*
then obtain α **where** *alpha-def*: $\alpha \in$ *nonzero* $Q_p \wedge a = c \otimes \alpha$[^]$n$
using *assms True pow-res-refl*[*of a n*] **unfolding** *assms* **unfolding** *pow-res-def*
by *blast*
show *?thesis*
proof(*cases b* ∈ *nonzero* Q_p)
case *T*: *True*
then have $d \in$ *nonzero* Q_p
using *assms pow-res-nonzero* **by** *blast*
then obtain β **where** *beta-def*: $\beta \in$ *nonzero* $Q_p \wedge b = d \otimes \beta$[^]$n$
using *T pow-res-refl*[*of b n*] **unfolding** *assms* **unfolding** *pow-res-def*
using *assms* **by** *blast*
then have $a \otimes b = (c \otimes d) \otimes (\alpha$[^]$n \otimes \beta$[^] $n)$
using *Qp.m-assoc Qp.m-lcomm Qp.nonzero-closed Qp.nonzero-mult-closed*
Qp-nat-pow-nonzero alpha-def assms(*3*) *assms*(*4*) *assms*(*5*) **by** *presburger*
then have *0*: $a \otimes b = (c \otimes d) \otimes ((\alpha \otimes \beta)$[^] $n)$

```
    by (metis Qp.nat-pow-distrib Qp.nonzero-closed alpha-def beta-def)
  show ?thesis
    apply(intro equal-pow-resI'[of - -  (α ⊗ β)[^] n] Qp.ring-simprules assms
                P-set-memI[of - α ⊗ β] Qp.nat-pow-closed nonzero-memE 0
Qp-nat-pow-nonzero
                )
    using alpha-def beta-def apply auto
    apply(intro nonzero-memI Qp.nonzero-mult-closed)
    using alpha-def beta-def nonzero-memE  apply auto
    by (meson Qp.integral-iff)
 next
  case False
  then have b = 0
    by (meson assms not-nonzero-Qp)
  then have d = 0
    using assms by (metis False not-nonzero-Qp pow-res-nonzero)
  then show ?thesis
    using Qp.r-null ‹b = 0› assms  by presburger
 qed
next
 case False
 then have a = 0
    by (meson assms  not-nonzero-Qp)
  then have c = 0
    using assms by (metis False not-nonzero-Qp pow-res-nonzero)
  then show ?thesis
    using Qp.r-null ‹a = 0› assms Qp.l-null by presburger
 qed

lemma pow-res-p-pow-factor:
 assumes n > 0
 assumes a ∈ carrier Q_p
 shows pow-res n a = pow-res n (𝔭[^]((l::int)*n) ⊗ a)
proof(cases a  = 0)
 case True
 then show ?thesis
   using Qp.r-null p-intpow-closed(1) by presburger
next
 case False
 then show ?thesis using assms pow-res-of-p-pow
   using Qp.m-comm Qp.one-closed Qp.r-one p-intpow-closed(1) pow-res-mult by
presburger
qed

lemma pow-res-classes-finite:
 assumes n ≥ 1
 shows finite (pow-res-classes n)
proof(cases n = 1)
 case True
```

have *pow-res-classes n* = {(*nonzero* Q_p)}
using *True pow-res-one* **unfolding** *pow-res-classes-def*
using *Qp.one-nonzero* **by** *blast*
then show *?thesis* **by** *auto*
next
case *False*
then have *n-bound*: $n \geq 2$
using *assms* **by** *linarith*
obtain *m* **where** *m-def*: $m > 0 \wedge (\forall x\ y.\ x \in nonzero\ Q_p \wedge y \in nonzero\ Q_p \wedge ac\ m\ x = ac\ m\ y \wedge ord\ x = ord\ y \longrightarrow pow\text{-}res\ n\ x = pow\text{-}res\ n\ y)$
using *assms False pow-res-equal-ord n-bound*
by (*metis gr-zeroI le-numeral-extra*(*2*))
obtain *f* **where** *f-def*: $f = (\lambda\ \eta\ \nu.\ (SOME\ y.\ y \in (pow\text{-}res\text{-}classes\ n) \wedge (\exists\ x \in y.\ ac\ m\ x = \eta \wedge ord\ x = \nu)))$
by *blast*
have *0*: $\bigwedge x.\ x \in nonzero\ Q_p \Longrightarrow pow\text{-}res\ n\ x = f\ (ac\ m\ x)\ (ord\ x)$
proof− **fix** *x* **assume** *A*: $x \in nonzero\ Q_p$
obtain η **where** *eta-def*: $\eta = ac\ m\ x$
by *blast*
obtain ν **where** *nu-def*: $\nu = ord\ x$
by *blast*
have $\exists y \in$*pow-res n x. ac m y* = *ac m x* $\wedge$ *ord y* = *ord x*
using *pow-res-refl A assms neq0-conv Qp.nonzero-closed* **by** *blast*
hence *pow-res n x* $\in$ (*pow-res-classes n*) $\wedge$ ($\exists$ *y* $\in$ (*pow-res n x*). *ac m y* = η $\wedge$ *ord y* = ν)
unfolding *nu-def eta-def* **using** *assms* **unfolding** *pow-res-classes-def*
using *A* **by** *blast*
then have *0*: ($\exists$ *y*. *y* $\in$ (*pow-res-classes n*) $\wedge$ ($\exists$ *x* $\in$ *y*. *ac m x* = η $\wedge$ *ord x* = ν))
by *blast*
have *f* η ν = (*SOME y*. *y* $\in$ (*pow-res-classes n*) $\wedge$ ($\exists$ *x* $\in$ *y*. *ac m x* = η $\wedge$ *ord x* = ν))
using *f-def* **by** *blast*
then have *1*: *f* η ν $\in$ (*pow-res-classes n*) $\wedge$ (($\exists$ *y* $\in$ (*f* η ν). *ac m y* = η $\wedge$ *ord y* = ν))
using *0 someI-ex*[*of* λ *y*. *y* $\in$ (*pow-res-classes n*) $\wedge$ ($\exists$ *x* $\in$ *y*. *ac m x* = η $\wedge$ *ord x* = ν)]
unfolding *f-def* **by** *blast*
then obtain *y* **where** *y-def*: *y* $\in$ (*f* η ν) $\wedge$ *ac m y* = *ac m x* $\wedge$ *ord y* = *ord x*
unfolding *nu-def eta-def* **by** *blast*
obtain *a* **where** *a-def*: *a* $\in$ *nonzero* Q_p $\wedge$ (*f* η ν) = *pow-res n a*
using *1* **unfolding** *pow-res-classes-def* **by** *blast*
then have *2*: *y* $\in$ *pow-res n a*
using *y-def* **by** *blast*
have *3*: *y* $\in$ *nonzero* Q_p
using *y-def nonzero-pow-res*[*of a n*] *a-def* **by** *blast*
then have *4*: *pow-res n y* = *pow-res n a*
using *3 y-def a-def equal-pow-resI*[*of y a n*] *n-bound Qp.nonzero-closed*
by (*metis equal-pow-resI*)

have *5*: *pow-res n y = f η ν*
using *4 a-def* **by** *blast*
then show *pow-res n x = f (ac m x) (ord x)*
unfolding *eta-def nu-def*
using *3 A m-def y-def* **by** *blast*
qed
obtain *N* **where** *N-def*: *N > 0 ∧ (∀ u ∈ carrier Q_p. ac N u = 1 ∧ val u = 0 ⟶ u ∈ P-set n)*
using *n-bound nth-power-fact assms* **by** *blast*
have *1*: *⋀x. x ∈ nonzero Q_p ⟹ (∃ y ∈ nonzero Q_p. ord y ≥ 0 ∧ ord y < n ∧ pow-res n x = pow-res n y)*
proof− **fix** *x* **assume** *x-def*: *x ∈ nonzero Q_p*
then obtain *k* **where** *k-def*: *k = ord x mod n*
by *blast*
then obtain *l* **where** *l-def*: *ord x = (int n)∗l + k*
using *cancel-div-mod-rules(2)[of n ord x0]* **unfolding** *k-def*
by (*metis group-add-class.add.right-cancel*)
have *x = (𝔭[^](ord x)) ⊗ ι (angular-component x)*
using *x-def angular-component-factors-x* **by** *blast*
then have *x = (𝔭[^](n∗l + k)) ⊗ ι (angular-component x)*
unfolding *l-def* **by** *blast*
hence *x = 𝔭[^](int n∗l) ⊗ (𝔭[^] k) ⊗ ι (angular-component x)*
by (*metis p-intpow-add*)
hence *0*: *x = (𝔭[^]l)[^]n ⊗ (𝔭[^] k) ⊗ ι (angular-component x)*
using *p-pow-factor[of n l k]* ‹*x = 𝔭 [^] (int n ∗ l + k) ⊗ ι (angular-component x)*› **by** *presburger*
have *1*: *ι (angular-component x) ∈ carrier Q_p*
using *x-def angular-component-closed inc-closed* **by** *blast*
hence *2*: *x = (𝔭[^]l)[^]n ⊗ ((𝔭[^] k) ⊗ ι (angular-component x))*
using *0* **by** (*metis Qp.m-assoc Qp.nat-pow-closed p-intpow-closed(1)*)
obtain *a* **where** *a-def*: *a = (𝔭[^] k) ⊗ ι (angular-component x)*
by *blast*
have *30*: *angular-component x ∈ Units Z_p*
using *angular-component-unit x-def* **by** *blast*
then have *3*: *ι (angular-component x) ∈ Units Q_p*
by (*metis Units-eq-nonzero Zp.Units-closed in-Units-imp-not-zero inc-of-nonzero not-nonzero-Qp*)
have *4*: *ι (angular-component x) ∈ nonzero Q_p*
using *3 Units-nonzero-Qp* **by** *blast*
have *a-nonzero*: *a ∈ nonzero Q_p*
unfolding *a-def 4*
by (*meson 3 Qp.UnitsI(1) Qp.Units-m-closed Units-nonzero-Qp p-intpow-closed(1) p-intpow-inv*)
have *5*: *x = a ⊗(𝔭[^]l)[^]n*
using *2 a-nonzero* **unfolding** *a-def*
using *Qp.m-comm Qp.nat-pow-closed Qp.nonzero-closed p-intpow-closed(1)* **by** *presburger*
hence *x ∈ pow-res n a*
unfolding *pow-res-def*

using *Qp.nonzero-closed Qp-int-pow-nonzero p-nonzero x-def* **by** *blast*
hence *6*:*pow-res n a = pow-res n x*
using *x-def a-def equal-pow-resI*[*of x a n*] *a-nonzero n-bound Qp.nonzero-closed equal-pow-resI*
by *blast*
have *7*: *ord* (*ι* (*angular-component x*)) = *0*
proof−
have *ord-Zp* (*angular-component x*) = *0* **using** *30 Zp-Units-ord-zero* **by** *blast*
then have *val-Zp* (*angular-component x*) = *0*
using *30 unit-imp-val-Zp0* **by** *blast*
then have *val* (*ι* (*angular-component x*)) = *0*
by (*metis angular-component-closed val-of-inc x-def*)
then show *?thesis* **using** *angular-component-closed x-def*
by (*metis 30 Zp.Units-closed* ‹*ord-Zp* (*angular-component x*) = *0*› *in-Units-imp-not-zero not-nonzero-Qp ord-of-inc*)
qed
have *8*: *ord a* = *k*
unfolding *a-def* **using** *3 4 7 ord-mult*[*of* 𝔭 [^] *k ι* (*angular-component x*)] *ord-p-pow-int*[*of k*]
p-pow-nonzero
using *Qp-int-pow-nonzero p-nonzero* **by** *presburger*
have *9*: *k* < *n*
unfolding *k-def*
using *assms* **by** *auto*
from *6 8 9 assms* **have** ‹*0* ≤ *ord a*› ‹*ord a* < *int n*› ‹*pow-res n x = pow-res n a*›
by (*auto simp add*: *k-def*)
with *a-nonzero* **show** ∃ *y*∈*nonzero Q*$_p$. *0* ≤ *ord y* ∧ *ord y* < *int n* ∧ *pow-res n x = pow-res n y*
by *auto*
qed
have *2*: ⋀*x*. *x* ∈ (*pow-res-classes n*) ⟹ ∃ *η ν*. *η* ∈ *Units* (*Zp-res-ring m*) ∧ *ν* ∈ {*0*..<*int n*} ∧ *x* = *f η ν*
proof− **fix** *a* **assume** *A*: *a* ∈ (*pow-res-classes n*)
then obtain *x* **where** *x-def*: *x* ∈ *nonzero Q*$_p$ ∧ *a* = *pow-res n x*
unfolding *pow-res-classes-def* **by** *blast*
then obtain *x*′ **where** *x*′*-def*: *x*′ ∈ *nonzero Q*$_p$ ∧ *ord x*′ ≥ *0* ∧ *ord x*′ < *n* ∧ *pow-res n x*′ = *a*
using *1*[*of x*] **unfolding** *x-def* **by** *blast*
hence *20*: *f* (*ac m x*′) (*ord x*′) = *a*
using *0* **by** *blast*
have *21*: *ac m x*′ ∈ *Units* (*Zp-res-ring m*)
using *x*′*-def ac-units m-def* **by** *presburger*
then have *22*: *ac m x*′ ∈ *Units* (*Zp-res-ring m*) ∧ (*ord x*′) ∈ ({*0*..<*n*}::*int set*) ∧ *a* = *f* (*ac m x*′) (*ord x*′)
using *x*′*-def 20 atLeastLessThan-iff* **by** *blast*
then show ∃ *η ν*. *η* ∈ *Units* (*Zp-res-ring m*) ∧ *ν* ∈ {*0*..<*int n*} ∧ *a* = *f η ν* **by** *blast*
qed

obtain F **where** *F-def*: $F = (\lambda ps.\ f\ (fst\ ps)\ (snd\ ps))$
by *blast*
have *3*: $\bigwedge x.\ x \in (pow\text{-}res\text{-}classes\ n) \Longrightarrow \exists\ ps \in Units\ (Zp\text{-}res\text{-}ring\ m) \times \{0..<int\ n\}.\ x = f\ (fst\ ps)\ (snd\ ps)$
proof− **fix** x **assume** *A*: $x \in pow\text{-}res\text{-}classes\ n$
obtain $\eta\ \nu$ **where** *eta-nu-def*: $\eta \in Units\ (Zp\text{-}res\text{-}ring\ m) \wedge \nu \in \{0..<int\ n\} \wedge x = f\ \eta\ \nu$
using *2 A* **by** *blast*
then have $F\ (\eta, \nu) = x$
unfolding *F-def* **by** (*metis fst-conv snd-conv*)
then show $\exists\ ps \in Units\ (Zp\text{-}res\text{-}ring\ m) \times \{0..<int\ n\}.\ x = f\ (fst\ ps)\ (snd\ ps)$
using *eta-nu-def local.F-def* **by** *blast*
qed
have *4*: $pow\text{-}res\text{-}classes\ n \subseteq F\ `\ (Units\ (Zp\text{-}res\text{-}ring\ m) \times \{0..<int\ n\})$
unfolding *F-def* **using** *3*
by *blast*
have *finite* $(Units\ (Zp\text{-}res\text{-}ring\ m))$
using *m-def residues.finite-Units* **unfolding** *residues-def*
by (*metis Qp.one-nonzero ac-in-res-ring ac-one' p-res-ring-one p-residue-ring-car-memE*(*1*))
hence *finite* $(Units\ (Zp\text{-}res\text{-}ring\ m) \times \{0..<int\ n\})$
by *blast*
then show *finite* $(pow\text{-}res\text{-}classes\ n)$
using *4* **by** (*meson finite-surj*)
qed

lemma *pow-res-classes-univ-semialg*:
assumes $S \in pow\text{-}res\text{-}classes\ n$
shows *is-univ-semialgebraic S*
proof−
obtain x **where** *x-def*: $x \in nonzero\ Q_p \wedge S = pow\text{-}res\ n\ x$
using *assms* **unfolding** *pow-res-classes-def* **by** *blast*
then show *?thesis* **using** *pow-res-is-univ-semialgebraic*
using *Qp.nonzero-closed* **by** *blast*
qed

lemma *pow-res-classes-semialg*:
assumes $S \in pow\text{-}res\text{-}classes\ n$
shows *is-semialgebraic 1* $(to\text{-}R1\ `\ S)$
using *pow-res-classes-univ-semialg*
assms(*1*) *is-univ-semialgebraicE* **by** *blast*

definition *nth-pow-wits* **where**
$nth\text{-}pow\text{-}wits\ n = (\lambda\ S.\ (SOME\ x.\ x \in (S \cap \mathcal{O}_p)))\ `\ (pow\text{-}res\text{-}classes\ n)$

lemma *nth-pow-wits-finite*:
assumes $n > 0$
shows *finite* $(nth\text{-}pow\text{-}wits\ n)$
proof−

```
  have n ≥ 1
    by (simp add: assms leI)
  thus ?thesis
    unfolding nth-pow-wits-def using assms pow-res-classes-finite[of n] by blast
qed

lemma nth-pow-wits-covers:
  assumes n > 0
  assumes x ∈ nonzero Q_p
  shows ∃ y ∈ (nth-pow-wits n). y ∈ nonzero Q_p ∧ y ∈ 𝒪_p ∧ x ∈ pow-res n y
proof−
  have PP: (pow-res n x) ∈ pow-res-classes n
    unfolding pow-res-classes-def using assms by blast
  obtain k where k-def: val x = eint k
    using assms val-ord by blast
  obtain N::int where N-def: N = (if  k < 0 then −k else k) by blast
  then have N-nonneg: N ≥ 0
    unfolding N-def
    by presburger
  have 0: int n ≥ 1
    using assms by linarith
  have N∗(int n) + k ≥ 0
  proof(cases k<0)
    case True then have N = −k unfolding N-def
      by presburger
    then have N∗n + k = k∗(1− int n)
      using distrib-left[of k 1 −int n] mult-cancel-left2 mult-minus-left
     by (metis add.inverse-inverse diff-minus-eq-add minus-mult-minus neg-equal-iff-equal
uminus-add-conv-diff)
    then show ?thesis using 0 True  zero-less-mult-iff[of k 1 − int n]
   proof −
     have 0 ≤ N ∗ (int n − 1)
       by (meson 0 N-nonneg diff-ge-0-iff-ge zero-le-mult-iff)
     then show ?thesis
       by (metis (no-types) ‹N = − k› add.commute distrib-left minus-add-cancel
mult-minus1-right uminus-add-conv-diff)
   qed
  next
    case False
    then have N = k unfolding N-def
      by presburger
    then show ?thesis using 0 False
      by (metis N-nonneg add-increasing2 mult-nonneg-nonneg of-nat-0-le-iff)
  qed
  then have 1: ord (𝔭[⁻](N∗n)⊗x) ≥ 0
    using ord-mult k-def val-ord assms
    by (metis Qp-int-pow-nonzero eint.inject ord-p-pow-int p-nonzero)
  have 2: 𝔭[⁻](N∗n)⊗x ∈ pow-res n x
  proof−
```

have $\mathfrak{p}[\hat{}](N*n) = (\mathfrak{p}[\hat{}]N)[\hat{}]n$
using *Qp-p-int-nat-pow-pow* **by** *blast*
then have $\mathfrak{p}[\hat{}]N \in$ *nonzero* $Q_p \ \wedge \mathfrak{p}[\hat{}](N*n)\otimes x = x \otimes (\mathfrak{p}[\hat{}]N)[\hat{}]n$
by (*metis Qp.m-comm Qp.nonzero-closed Qp-int-pow-nonzero assms*(*2*) *p-nonzero*)
then show *?thesis* **unfolding** *pow-res-def*
by (*metis* (*mono-tags*, *lifting*) *Qp.m-closed Qp.nonzero-closed assms*(*2*) *mem-Collect-eq p-intpow-closed*(*1*))
qed
have *3*: $\mathfrak{p}[\hat{}](N*n)\otimes x \in \mathcal{O}_p$
using *1 assms*
by (*metis* Q_p*-def Qp.nonzero-mult-closed Qp-int-pow-nonzero* Z_p*-def val-ring-ord-criterion* ι*-def p-nonzero padic-fields.zero-in-val-ring padic-fields-axioms*)
have *4*: $x \in$ *pow-res n* $(\mathfrak{p}[\hat{}](N*n)\otimes x)$
using *2 equal-pow-resI*[*of x* $\mathfrak{p}[\hat{}](N*n)\otimes x$ *n*] *pow-res-refl*[*of* $\mathfrak{p}[\hat{}](N*n)\otimes x$ *n*] *assms*
Qp.nonzero-mult-closed p-intpow-closed(*2*) *pow-res-refl Qp.nonzero-closed*
by *metis*
have *5*: $\mathfrak{p}[\hat{}](N*n)\otimes x \in$ (*pow-res n x* $\cap \mathcal{O}_p$)
using *2 3* **by** *blast*
have *6*: (*SOME z*. $z \in$ (*pow-res n x*) $\cap \mathcal{O}_p$) $\in$ *pow-res n x* $\cap \mathcal{O}_p$ **using** *5*
by (*meson someI*)
obtain *y* **where** *y-def*: $y =$ (*SOME z*. $z \in$ (*pow-res n x*) $\cap \mathcal{O}_p$)
by *blast*
then have *A*: $y \in$ *pow-res n x*
using *6* **by** *blast*
then have *pow-res n x* = *pow-res n y*
using *equal-pow-resI*[*of x y n*] *assms y-def Qp.nonzero-closed nonzero-pow-res*
by *blast*
then have *7*: $x \in$ *pow-res n y*
using *pow-res-refl*[*of x n*] *assms* **unfolding** *nonzero-def* **by** *blast*
have *8*: $y \in$ *nonzero* Q_p
using *y-def PP 6 A nonzero-pow-res*[*of x n*] *assms*
by *blast*
have *9*: $y \in \mathcal{O}_p$
using *y-def 6* **by** *blast*
have $y \in (\lambda S.$ *SOME x*. $x \in S \cap \mathcal{O}_p)$ ‘ *pow-res-classes n* $\wedge$ $y \in$ *nonzero* $Q_p \wedge y \in \mathcal{O}_p \wedge x \in$ *pow-res n y*
using *y-def PP 6 7 8 9 A nonzero-pow-res*[*of x n*] *assms*
by *blast*
then show *?thesis* **unfolding** *nth-pow-wits-def* **by** *blast*
qed

lemma *nth-pow-wits-closed*:
assumes $n > 0$
assumes $x \in$ *nth-pow-wits n*
shows $x \in$ *carrier* Q_p $x \in \mathcal{O}_p$ $x \in$ *nonzero* Q_p $\exists$ $y \in$ *pow-res-classes n*. $y =$ *pow-res n x*
proof−
obtain *c* **where** *c-def*: $c \in$ *pow-res-classes n* $\wedge$ $x =$ (*SOME x*. $x \in (c \cap \mathcal{O}_p)$)

by (*metis* (*no-types*, *lifting*) *assms*(*2*) *image-iff nth-pow-wits-def*)
then obtain *y* **where** *y-def*: $y \in$ *nonzero* $Q_p \wedge c =$ *pow-res n y*
unfolding *pow-res-classes-def* **by** *blast*
then obtain *a* **where** *a-def*: $a \in$ (*nth-pow-wits n*) $\wedge$ $a \in$ *nonzero* $Q_p \wedge a \in \mathcal{O}_p$ $\wedge$ $y \in$ *pow-res n a*
using *nth-pow-wits-covers*[*of n y*] *assms*(*1*) **by** *blast*
have *00*: *pow-res n a* = *c*
using *equal-pow-resI*[*of a y n*] *y-def assms a-def* **unfolding** *nonzero-def* **by** *blast*
then have *P* :$a \in c \cap \mathcal{O}_p$
using *pow-res-refl*[*of a n*] *assms a-def* **unfolding** *00 nonzero-def* **by** *blast*
then show *0*: $x \in \mathcal{O}_p$ **using** *c-def*
by (*metis Collect-mem-eq Int-Collect tfl-some*)
then show $x \in$ *carrier* Q_p
using *val-ring-memE* **by** *blast*
have *1*: $c \subseteq$ *nonzero* Q_p
using *c-def nonzero-pow-res*[*of y n*] **unfolding** *pow-res-classes-def*
using *assms*(*1*) *y-def* **by** *blast*
have (*SOME x*. $x \in (c \cap \mathcal{O}_p)$) $\in (c \cap \mathcal{O}_p)$
using *P tfl-some*
by (*smt Int-def someI-ex*)
then have *2*: $x \in c$
using *c-def* **by** *blast*
thus $x \in$ *nonzero* Q_p
using *1* **by** *blast*
show $\exists$ $y \in$ *pow-res-classes n*. *y* = *pow-res n x*
using *00 2 c-def P a-def equal-pow-resI*[*of a x n*] *0 val-ring-memE assms*(*1*) **by** *blast*
qed

lemma *finite-extensional-funcset*:
assumes *finite A*
assumes *finite* (*B*::*'b set*)
shows *finite* (*A* $\rightarrow_E$ *B*)
using *finite-PiE*[*of A* λ-. *B*] *assms* **by** *blast*

lemma *nth-pow-wits-exists*:
assumes *m* > *0*
assumes *c* $\in$ *pow-res-classes m*
shows $\exists x.\ x \in c \cap \mathcal{O}_p$
proof−
obtain *x* **where** *x-def*: $x \in$ *nonzero* $Q_p \wedge$ *pow-res m x* = *c*
using *assms* **unfolding** *pow-res-classes-def* **by** *blast*
obtain *y* **where** *y-def*: $y \in$ (*nth-pow-wits m*) $\wedge$ $y \in$ *nonzero* $Q_p \wedge y \in \mathcal{O}_p \wedge x \in$ *pow-res m y*
using *nth-pow-wits-covers assms x-def*
by *blast*
have *0*: *pow-res m x* = *pow-res m y*
using *x-def y-def equal-pow-resI Qp.nonzero-closed assms*(*1*) **by** *blast*

then have *1*: $y \in$ *pow-res m x*
using *pow-res-refl*[*of y m*] *y-def assms* **unfolding** *nonzero-def* **by** *blast*
thus *?thesis* **using** *x-def y-def assms*
by *blast*
qed

lemma *pow-res-classes-mem-eq*:
assumes $m > 0$
assumes $a \in$ *pow-res-classes m*
assumes $x \in a$
shows $a =$ *pow-res m x*
proof−
obtain y **where** *y-def*: $y \in$ *nonzero* $Q_p \wedge a =$ *pow-res m y*
using *assms* **unfolding** *pow-res-classes-def* **by** *blast*
then show *?thesis* **using** *assms equal-pow-resI*[*of y x m*]
by (*meson Qp.nonzero-closed nonzero-pow-res equal-pow-resI subsetD*)
qed

lemma *nth-pow-wits-neq-pow-res*:
assumes $m > 0$
assumes $x \in$ *nth-pow-wits m*
assumes $y \in$ *nth-pow-wits m*
assumes $x \neq y$
shows *pow-res m x* $\neq$ *pow-res m y*
proof−
obtain a **where** *a-def*: $a \in$ *pow-res-classes m* $\wedge$ $x = (\lambda\ S.\ (SOME\ x.\ x \in (S \cap \mathcal{O}_p)))\ a$
using *assms* **unfolding** *nth-pow-wits-def* **by** *blast*
obtain b **where** *b-def*: $b \in$ *pow-res-classes m* $\wedge$ $y = (\lambda\ S.\ (SOME\ x.\ x \in (S \cap \mathcal{O}_p)))\ b$
using *assms* **unfolding** *nth-pow-wits-def* **by** *blast*
have *a-neq-b*: $a \neq b$
using *assms a-def b-def* **by** *blast*
have *0*: $x \in a \cap \mathcal{O}_p$
using *a-def nth-pow-wits-exists*[*of m a*] *assms*
by (*meson someI-ex*)
have *1*: $y \in b \cap \mathcal{O}_p$
using *b-def nth-pow-wits-exists*[*of m b*] *assms*
by (*meson someI-ex*)
have *2*: *pow-res m x* $= a$
using *a-def pow-res-classes-mem-eq*[*of m a x*] *assms 0*
by *blast*
have *3*: *pow-res m y* $= b$
using *b-def pow-res-classes-mem-eq*[*of m b y*] *assms 1*
by *blast*
show *?thesis*
by (*simp add*: *2 3 a-neq-b*)
qed

lemma *nth-pow-wits-disjoint-pow-res*:
assumes $m > 0$
assumes $x \in$ *nth-pow-wits* m
assumes $y \in$ *nth-pow-wits* m
assumes $x \neq y$
shows *pow-res* $m\ x \cap$ *pow-res* $m\ y = \{\}$
using *assms nth-pow-wits-neq-pow-res disjoint-iff-not-equal*
by (*metis* (*no-types*, *opaque-lifting*) *nth-pow-wits-closed*(*4*) *pow-res-classes-mem-eq*)

lemma *nth-power-fact'*:
assumes $0 <$ (n::*nat*)
shows $\exists m{>}0.\ \forall u{\in}$*carrier* $Q_p.$ *ac* $m\ u = 1 \wedge$ *val* $u = 0 \longrightarrow u \in$ *P-set* n
using *nth-power-fact*[*of n*] *assms*
by (*metis less-one less-or-eq-imp-le linorder-neqE-nat neq0-conv*)

lemma *equal-pow-res-criterion*:
assumes $N > 0$
assumes $n > 0$
assumes $\forall\ u \in$ *carrier* $Q_p.$ *ac* $N\ u = 1 \wedge$ *val* $u = 0 \longrightarrow u \in$ *P-set* n
assumes $a \in$ *carrier* Q_p
assumes $b \in$ *carrier* Q_p
assumes $c \in$ *carrier* Q_p
assumes $a = b \otimes (\mathbf{1} \oplus c)$
assumes *val* $c \geq N$
shows *pow-res* $n\ a =$ *pow-res* $n\ b$
proof(*cases* $b = \mathbf{0}$)
case *True*
then have $a = \mathbf{0}$
using *assms Qp.add.m-closed Qp.l-null Qp.one-closed* **by** *presburger*
then show *?thesis* **using** *True*
by *blast*
next
case *False*
then have *F0*: $a \div b = \mathbf{1} \oplus c$
by (*metis Qp.Units-one-closed Qp.add.m-closed Qp.inv-cancelR*(*2*) *Qp.one-closed Qp.unit-factor assms*(*4*) *assms*(*5*) *assms*(*6*) *assms*(*7*) *field-inv*(*2*) *inv-in-frac*(*1*))
have $0 <$ *eint* N
using *assms* **by** (*metis eint-ord-simps*(*2*) *of-nat-0-less-iff zero-eint-def*)
hence *F1*: *val* $\mathbf{1} <$ *val* c
using *assms less-le-trans*[*of 0 N val c*] **unfolding** *val-one*
by *blast*
hence *F2*: *val* $\mathbf{1} =$ *val* $(\mathbf{1} \oplus c)$
using *assms val-one one-nonzero Qp.add.m-comm Qp.one-closed val-ultrametric-noteq*
by *metis*
have *val* $\mathbf{1} +$ *eint* (*int* N) $\leq$ *val* $(\mathbf{1} \ominus (\mathbf{1} \oplus c))$
proof−
have *val* $(\mathbf{1} \ominus (\mathbf{1} \oplus c)) =$ *val* c
using *Qp.add.inv-closed Qp.minus-eq Qp.minus-sum Qp.one-closed Qp.r-neg2 assms*(*6*) *val-minus* **by** *presburger*

thus *?thesis*
unfolding *val-one* **using** *assms F1* **by** (*metis add.left-neutral*)
qed
hence *F3*: *ac N* $\mathbf{1}$ = *ac N* ($\mathbf{1} \oplus c$)
using *F2 F1 assms ac-val*[*of* $\mathbf{1}$ $\mathbf{1} \oplus c$ *N*]
by (*metis Qp.add.m-closed Qp.one-closed val-nonzero*)
have *F4*: $\mathbf{1} \oplus c \in$ *P-set n*
using *assms F1 F2 F3 val-one ac-one*
by (*metis Qp.add.m-closed Qp.one-closed Qp.one-nonzero ac-inv'' ac-inv'''*(*1*) *ac-one'*)
then show *?thesis*
using *assms*(*2*) *assms*(*4*) *assms*(*5*) *assms*(*7*) *equal-pow-resI'* **by** *blast*
qed

lemma *pow-res-nat-pow*:
assumes $n > 0$
assumes $a \in$ *carrier* Q_p
assumes $b \in$ *carrier* Q_p
assumes *pow-res n a* = *pow-res n b*
shows *pow-res n* (*a*[^](*k*::*nat*)) = *pow-res n* (*b*[^]*k*)
apply(*induction k*)
using *assms* **apply** (*metis Group.nat-pow-0*)
using *assms pow-res-mult* **by** (*smt Qp.nat-pow-Suc2 Qp.nat-pow-closed*)

lemma *pow-res-mult'*:
assumes $n > 0$
assumes $a \in$ *carrier* Q_p
assumes $b \in$ *carrier* Q_p
assumes $c \in$ *carrier* Q_p
assumes $d \in$ *carrier* Q_p
assumes $e \in$ *carrier* Q_p
assumes $f \in$ *carrier* Q_p
assumes *pow-res n a* = *pow-res n d*
assumes *pow-res n b* = *pow-res n e*
assumes *pow-res n c* = *pow-res n f*
shows *pow-res n* ($a \otimes b \otimes c$) = *pow-res n* ($d \otimes e \otimes f$)
proof−
have *pow-res n* ($a \otimes b$) = *pow-res n* ($d \otimes e$)
using *pow-res-mult assms* **by** *meson*
then show *?thesis* **using** *pow-res-mult assms*
by (*meson Qp.m-closed*)
qed

lemma *pow-res-disjoint*:
assumes $n > 0$
assumes $a \in$ *nonzero* Q_p
assumes $a \notin$ *pow-res n* $\mathbf{1}$

shows $\neg\ (\exists\, y \in \mathit{nonzero}\ Q_p.\ a = y[\hat{}]n)$
using *assms* **unfolding** *pow-res-def*
using *Qp.l-one Qp.nonzero-closed* **by** *blast*

lemma *pow-res-disjoint'*:
assumes $n > 0$
assumes $a \in \mathit{nonzero}\ Q_p$
assumes *pow-res n a* $\neq$ *pow-res n* **1**
shows $\neg\ (\exists\, y \in \mathit{nonzero}\ Q_p.\ a = y[\hat{}]n)$
using *assms pow-res-disjoint pow-res-refl*
by (*metis pow-res-nth-pow*)

lemma *pow-res-one-imp-nth-pow*:
assumes $n > 0$
assumes $a \in$ *pow-res n* **1**
shows $\exists\, y \in \mathit{nonzero}\ Q_p.\ a = y[\hat{}]n$
using *assms* **unfolding** *pow-res-def*
using *Qp.l-one Qp.nat-pow-closed Qp.nonzero-closed* **by** *blast*

lemma *pow-res-eq*:
assumes $n > 0$
assumes $a \in \mathit{carrier}\ Q_p$
assumes $b \in$ *pow-res n a*
shows *pow-res n b* = *pow-res n a*
proof(*cases a* = **0**)
case *True*
then show *?thesis* **using** *assms* **by** (*metis pow-res-zero singletonD*)
next
case *False*
then have *a-nonzero*: $a \in \mathit{nonzero}\ Q_p$ **using** *Qp.not-nonzero-memE assms*(*2*)
by *blast*
show *?thesis*
proof(*cases n* = *1*)
case *True*
then show *?thesis* **using** *a-nonzero assms*
using *pow-res-one* Q_p*-def Zp-def padic-fields-axioms* **by** *blast*
next
case *False*
then have $n \geq 2$
using *assms*(*1*) **by** *linarith*
then show *?thesis* **using** *False a-nonzero assms Qp.nonzero-closed nonzero-pow-res equal-pow-resI*
by *blast*
qed
qed

lemma *pow-res-classes-n-eq-one*:
pow-res-classes 1 = {*nonzero* Q_p}
unfolding *pow-res-classes-def* **using** *pow-res-one Qp.one-nonzero* **by** *blast*

lemma *nth-pow-wits-closed'*:
assumes $n > 0$
assumes $x \in$ *nth-pow-wits* n
shows $x \in \mathcal{O}_p \wedge x \in$ *nonzero* Q_p **using** *nth-pow-wits-closed assms* **by** *blast*

13.8 Semialgebraic Sets Defined by Congruences

13.8.1 p-adic ord Congruence Sets

lemma *carrier-is-univ-semialgebraic*:
is-univ-semialgebraic (*carrier* Q_p)
apply(*rule is-univ-semialgebraicI*)
using *Qp.to-R1-carrier carrier-is-semialgebraic*
by *presburger*

lemma *nonzero-is-univ-semialgebraic*:
is-univ-semialgebraic (*nonzero* Q_p)
proof−
have *nonzero* Q_p = *carrier* Q_p − {**0**}
unfolding *nonzero-def* **by** *blast*
then show *?thesis* **using** *diff-is-univ-semialgebraic*[*of carrier* Q_p {**0**}]
by (*metis Diff-empty Diff-insert0 carrier-is-univ-semialgebraic empty-subsetI finite.emptyI finite.insertI finite-is-univ-semialgebraic insert-subset*)
qed

definition *ord-congruence-set* **where**
ord-congruence-set n a = $\{x \in$ *nonzero* Q_p. *ord* x *mod* $n = a\}$

lemma *ord-congruence-set-nonzero*:
ord-congruence-set n $a \subseteq$ *nonzero* Q_p
by (*metis* (*no-types*, *lifting*) *mem-Collect-eq ord-congruence-set-def subsetI*)

lemma *ord-congruence-set-closed*:
ord-congruence-set n $a \subseteq$ *carrier* Q_p
using *nonzero-def ord-congruence-set-nonzero*
unfolding *nonzero-def*
by (*meson Qp.nonzero-closed ord-congruence-set-nonzero subset-iff*)

lemma *ord-congruence-set-memE*:
assumes $x \in$ *ord-congruence-set* n a
shows $x \in$ *nonzero* Q_p
ord x *mod* $n = a$
using *assms ord-congruence-set-nonzero* **apply** *blast*
by (*metis* (*mono-tags*, *lifting*) *assms mem-Collect-eq ord-congruence-set-def*)

lemma *ord-congruence-set-memI*:
assumes $x \in$ *nonzero* Q_p
assumes *ord* x *mod* $n = a$

shows $x \in$ *ord-congruence-set n a*
using *assms*
by (*metis* (*mono-tags*, *lifting*) *mem-Collect-eq ord-congruence-set-def*)

We want to prove that ord_congruence_set is a finite union of semialgebraic sets, hence is also semialgebraic.

lemma *pow-res-ord-cong*:
assumes $x \in$ *carrier* Q_p
assumes $x \in$ *ord-congruence-set n a*
shows *pow-res n x* $\subseteq$ *ord-congruence-set n a*
proof fix *y*
assume *A*: $y \in$ *pow-res n x*
show $y \in$ *ord-congruence-set* (*int n*) *a*
proof−
obtain *a* **where** *a-def*: $a \in$ *nonzero* $Q_p \wedge y = x \otimes (a[\hat{}]n)$
using *A pow-res-def*[*of n x*] **by** *blast*
have *0*: $x \in$ *nonzero* Q_p
using *assms*(*2*) *ord-congruence-set-memE*(*1*)
by *blast*
have *1*: $y \in$ *nonzero* Q_p
using *A*
by (*metis 0 Qp.integral Qp.nonzero-closed Qp.nonzero-mult-closed Qp-nat-pow-nonzero a-def not-nonzero-Qp*)
have *2*: *ord y* = *ord x* + *n*∗ *ord a*
using *a-def 0 1 Qp-nat-pow-nonzero nonzero-nat-pow-ord ord-mult*
by *presburger*
show *?thesis*
apply(*rule ord-congruence-set-memI*)
using *assms ord-congruence-set-memE 2 1*
apply *blast*
using *2 assms*(*2*) *ord-congruence-set-memE*(*2*)
by *presburger*
qed
qed

lemma *pow-res-classes-are-univ-semialgebraic*:
shows *are-univ-semialgebraic* (*pow-res-classes n*)
apply(*rule are-univ-semialgebraicI*)
using *pow-res-classes-univ-semialg* **by** *blast*

lemma *ord-congruence-set-univ-semialg*:
assumes $n \geq 0$
shows *is-univ-semialgebraic* (*ord-congruence-set n a*)
proof(*cases n* = *0*)
case *True*
have *T0*: *ord-congruence-set n a* = {$x \in$ *nonzero* Q_p. *ord x* = *a*}
unfolding *ord-congruence-set-def True* **by** *presburger*
have *T1*: {$x \in$ *nonzero* Q_p. *ord x* = *a*} = {$x \in$ *nonzero* Q_p. *val x* = *a*}
apply(*rule equalityI''*)

using *val-ord* **apply** *blast*
using *val-ord*
by (*metis eint.inject*)
have *T2*: $\{x \in nonzero\ Q_p.\ val\ x = a\} = \{x \in carrier\ Q_p.\ val\ x = a\}$
apply(*rule equalityI''*)
using *Qp.nonzero-closed* **apply** *blast*
by (*metis iless-Suc-eq val-nonzero val-val-ring-prod zero-in-val-ring*)
show *?thesis* **unfolding** *T0 T1 T2* **using** *univ-val-eq-set-is-univ-semialgebraic*
by *blast*
next
case *False*
obtain *F* **where** *F-def*: $F = \{S \in (pow\text{-}res\text{-}classes\ (nat\ n)).\ S \subseteq (ord\text{-}congruence\text{-}set\ n\ a)\ \}$
by *blast*
have *0*: $F \subseteq pow\text{-}res\text{-}classes\ (nat\ n)$
using *F-def* **by** *blast*
have *1*: *finite F*
using *0 False nat-mono*[*of 1 n*] *nat-numeral*[] *pow-res-classes-finite*[*of nat n*] *rev-finite-subset*
by (*smt assms nat-one-as-int*)
have *2*: *are-univ-semialgebraic F*
apply(*rule are-univ-semialgebraicI*) **using** *0 pow-res-classes-are-univ-semialgebraic*
by (*metis* (*mono-tags*) *are-univ-semialgebraicE are-univ-semialgebraic-def assms nat-mono nat-numeral subset-iff*)
have *3*: $\bigcup F = (ord\text{-}congruence\text{-}set\ n\ a)$
proof
show $\bigcup F \subseteq ord\text{-}congruence\text{-}set\ n\ a$
using *F-def*
by *blast*
show $ord\text{-}congruence\text{-}set\ n\ a \subseteq \bigcup F$
proof fix *x*
assume *A*: $x \in ord\text{-}congruence\text{-}set\ n\ a$
have *x-nonzero*: $x \in nonzero\ Q_p$
using *A ord-congruence-set-memE*(*1*) **by** *blast*
have *0*: $pow\text{-}res\ (nat\ n)\ x \in F$
using *A pow-res-classes-def F-def*
by (*smt nonzero-def assms mem-Collect-eq nat-0-le ord-congruence-set-memE*(*1*) *pow-res-ord-cong*)
have *1*: $x \in pow\text{-}res\ (nat\ n)\ x$ **using** *False x-nonzero assms pow-res-refl*[*of x nat n*]
using *Qp.nonzero-closed* **by** *blast*
show $x \in \bigcup F$
using *0 1*
by *blast*
qed
qed
then show *?thesis*
using *1 2 finite-union-is-univ-semialgebraic'*
by *fastforce*

qed

lemma *ord-congruence-set-is-semialg*:
assumes $n \geq 0$
shows *is-semialgebraic 1* (*Qp-to-R1-set* (*ord-congruence-set n a*))
using *assms is-univ-semialgebraicE ord-congruence-set-univ-semialg*
by *blast*

13.8.2 Congruence Sets for the order of the Evaluation of a Polynomial

lemma *poly-map-singleton*:
assumes $f \in$ *carrier* ($Q_p[\mathcal{X}_n]$)
assumes $x \in$ *carrier* (${Q_p}^n$)
shows *poly-map n* [*f*] *x* = [(*Qp-ev f x*)]
unfolding *poly-map-def poly-tuple-eval-def*
using *assms*
by (*metis* (*no-types*, *lifting*) *Cons-eq-map-conv list.simps*(*8*) *restrict-apply'*)

definition *poly-cong-set* **where**
poly-cong-set n f m a = {*x* ∈ *carrier* (${Q_p}^n$). (*Qp-ev f x*) ≠ **0** ∧ (*ord* (*Qp-ev f x*) *mod m* = *a*)}

lemma *poly-cong-set-as-pullback*:
assumes $f \in$ *carrier* ($Q_p[\mathcal{X}_n]$)
shows *poly-cong-set n f m a* = *poly-map n* [*f*] $^{-1}{}_n$(*Qp-to-R1-set* (*ord-congruence-set m a*))
proof
show *poly-cong-set n f m a* ⊆ *poly-map n* [*f*] $^{-1}{}_n$ ((λ*a*. [*a*]) ‘ *ord-congruence-set m a*)
proof fix *x*
assume *A*: *x* ∈ *poly-cong-set n f m a*
then have *0*: *x* ∈ *carrier* (${Q_p}^n$)
by (*metis* (*no-types*, *lifting*) *mem-Collect-eq poly-cong-set-def*)
have *1*: (*Qp-ev f x*) ≠ **0**
by (*metis* (*mono-tags*, *lifting*) *A mem-Collect-eq poly-cong-set-def*)
have *2*: (*ord* (*Qp-ev f x*) *mod m* = *a*)
by (*metis* (*mono-tags*, *lifting*) *A mem-Collect-eq poly-cong-set-def*)
have *3*: (*Qp-ev f x*) ∈ (*ord-congruence-set m a*)
using *0 1 2 eval-at-point-closed assms not-nonzero-Qp ord-congruence-set-memI*
by *metis*
show *x* ∈ *poly-map n* [*f*] $^{-1}{}_n$ ((λ*a*. [*a*]) ‘ *ord-congruence-set m a*)
proof−
have *00*: *poly-map n* [*f*] *x* = [(*Qp-ev f x*)]
using *0 assms poly-map-singleton* **by** *blast*
have *01*: [*eval-at-point* Q_p *x f*] ∈ *carrier* (${Q_p}^1$)
using *0 assms eval-at-point-closed Qp.to-R1-closed* **by** *blast*
hence *02*: *poly-map n* [*f*] *x* ∈ (λ*a*. [*a*]) ‘ *ord-congruence-set m a*
using *3 00* **by** *blast*

```
    then show x ∈ poly-map n [f] ^-1n ((λa. [a]) ‘ ord-congruence-set m a)
      using 0 unfolding evimage-def
      by blast
   qed
  qed
  show poly-map n [f] ^-1n (λa. [a]) ‘ ord-congruence-set m a
          ⊆ poly-cong-set n f m a
  proof fix x
   assume A: x ∈ poly-map n [f] ^-1n ((λa. [a]) ‘ (ord-congruence-set m a))
   have 0: ((λa. [a]) ‘ ord-congruence-set m a) ⊆ carrier (Q_p^1)
     using ord-congruence-set-closed Qp.to-R1-carrier by blast
   have is-poly-tuple n [f]
     using assms unfolding is-poly-tuple-def
     by (simp add: assms)
   then have 1:poly-map n [f] ^-1n((λa. [a]) ‘ ord-congruence-set m a) ⊆ carrier
(Q_p^n)
     using 0 A assms One-nat-def
     by (metis extensional-vimage-closed)
  then have 2: x ∈ carrier (Q_p^n)
    using A unfolding evimage-def by blast
   then have 3: poly-map n [f] x ∈ ((λa. [a]) ‘ ord-congruence-set m a)
     using A assms 0 One-nat-def
     by blast
   have poly-map n [f] x = [(Qp-ev f x)]
     using 2 assms poly-map-singleton by blast
   then have Qp-ev f x ∈ ord-congruence-set m a
     using 3
     by (metis (mono-tags, lifting) image-iff list.inject)
   then show x ∈ poly-cong-set n f m a
     unfolding poly-cong-set-def
     by (metis (mono-tags, lifting) 2 Qp.nonzero-memE(2)
       mem-Collect-eq ord-congruence-set-memE(1) ord-congruence-set-memE(2))
  qed
qed

lemma singleton-poly-tuple:
is-poly-tuple n [f] ⟷ f ∈ carrier (Q_p[𝒳_n])
  unfolding is-poly-tuple-def
 by (metis (no-types, lifting) list.distinct(1) list.set-cases list.set-intros(1) set-ConsD
subset-code(1))

lemma poly-cong-set-is-semialgebraic:
  assumes m ≥ 0
  assumes f ∈ carrier (Q_p[𝒳_n])
  shows is-semialgebraic n (poly-cong-set n f m a)
proof−
  have 0: (λa. [a]) ‘ ord-congruence-set m a ∈ semialg-sets 1
  using assms
        ord-congruence-set-is-semialg[of m a]
```

unfolding *is-semialgebraic-def*
by *blast*
have *1*: *length* [*f*] = *1*
by *simp*
hence *poly-map n* [*f*] $^{-1}_n$ (λ*a*. [*a*]) ‘ *ord-congruence-set m a* ∈ *semialg-sets n*
using *0 singleton-poly-tuple*[*of n f*] *zero-neq-one assms*
pullback-is-semialg[*of n* [*f*] *1* (λ*a*. [*a*]) ‘ *ord-congruence-set m a*]
unfolding *is-semialgebraic-def*
by *blast*
thus *?thesis* **using** *assms poly-cong-set-as-pullback*[*of f n m a*]
unfolding *is-semialgebraic-def*
by *presburger*
qed

13.8.3 Congruence Sets for Angular Components

If a set is a union of n-th power residues, then it is semialgebraic.

lemma *pow-res-union-imp-semialg*:
assumes *n* ≥ *1*
assumes *S* ⊆ *nonzero* Q_p
assumes ⋀*x*. *x* ∈ *S* ⟹ *pow-res n x* ⊆ *S*
shows *is-univ-semialgebraic S*
proof−
obtain *F* **where** *F-def*: *F* = {*T*. *T* ∈ *pow-res-classes n* ∧ *T* ⊆ *S*}
by *blast*
have *0*: *F* ⊆ *pow-res-classes n*
using *F-def* **by** *blast*
have *1*: *finite F*
using *0 pow-res-classes-finite*[*of n*] *assms*(*1*) *finite-subset*
by *auto*
have *2*: *are-univ-semialgebraic F*
using *0*
by (*meson are-univ-semialgebraicE are-univ-semialgebraicI assms*(*1*)
pow-res-classes-are-univ-semialgebraic padic-fields-axioms subsetD)
have *3*: *S* = ⋃ *F*
proof
show *S* ⊆ ⋃ *F*
proof fix *x*
assume *A*: *x* ∈ *S*
then have *pow-res n x* ⊆ *S*
using *assms*(*3*) **by** *blast*
then have *pow-res n x* ∈ *F*
using *A assms*(*2*) *F-def pow-res-classes-def*
by (*smt mem-Collect-eq subsetD*)
then have *pow-res n x* ⊆ ⋃ *F*
by *blast*
then show *x* ∈ ⋃ *F*
using *A assms*(*1*) *assms*(*2*) *pow-res-refl*[*of x n*] **unfolding** *nonzero-def* **by**
blast

```
    qed
    show ⋃ F ⊆ S
      using F-def
      by blast
  qed
  show ?thesis
    using 1 2 3 finite-union-is-univ-semialgebraic'
    by blast
qed

definition ac-cong-set1 where
ac-cong-set1 n y = {x ∈ carrier Q_p. x ≠ 0 ∧ ac n x = ac n y}

lemma ac-cong-set1-is-univ-semialg:
  assumes n > 0
  assumes b ∈ nonzero Q_p
  assumes b ∈ 𝒪_p
  shows is-univ-semialgebraic (ac-cong-set1 n b)
proof(cases n = 1 ∧ p = 2)
  case True
  have (ac-cong-set1 n b) = nonzero Q_p
  proof
    have 0: Units (Zp-res-ring n) = {1}
    proof show Units (Zp-res-ring n) ⊆ {1}
      proof fix x assume A: x ∈ Units (Zp-res-ring n)
        have 0: carrier (Zp-res-ring n) = {0..(int 2) − 1}
          using True
       by (metis assms(1) int-ops(3) p-residues power-one-right residues.res-carrier-eq)
        have 1: carrier (Zp-res-ring n) = {0..(1::int)}
        proof− have int 2 − 1 = (1::int)
                  by linarith
                then show ?thesis
                  using 0
                  by presburger
        qed
        have 15: {0..(1::int)} = {0, (1::int)}
          using atLeastAtMostPlus1-int-conv [of 0 0::int]
          by (smt atLeastAtMost-singleton insert-commute)
        have 2: carrier (Zp-res-ring n) = {0,(1::int)}
          using 1 15
          by blast
        have 3: 0 ∉ Units (Zp-res-ring n)
          using True zero-not-in-residue-units by blast
        have x ∈ carrier (Zp-res-ring n)
          using A unfolding Units-def by blast
        then have x = 1 using A  2 3
          by (metis 1 atLeastAtMost-iff atLeastatMost-empty
                    atLeastatMost-empty-iff2 linorder-neqE-linordered-idom mod-by-1
mod-pos-pos-trivial )
```

```
      then show x ∈ {1}
        by simp
    qed
    show {1} ⊆ Units (Zp-res-ring n)
      by (meson assms(1) empty-subsetI insert-subset residue-1-unit(1))
  qed
  show ac-cong-set1 n b ⊆ nonzero Q_p
   by (metis (mono-tags, lifting) ac-cong-set1-def mem-Collect-eq not-nonzero-Qp
subsetI)
  show nonzero Q_p ⊆ ac-cong-set1 n b
  proof fix x
    assume A: x ∈ nonzero Q_p
    then have P0: ac n x = 1
      using 0 ac-units assms(1) by blast
    have P1: ac n b = 1
      using assms 0 ac-units assms(1) by blast
    then have ac n x = ac n b
      using P0 by metis
    then show  x ∈ ac-cong-set1 n b
      unfolding ac-cong-set1-def using A
    proof −
      have x ∈ {r ∈ carrier Q_p. r ≠ 0}
        by (metis (no-types) ‹x ∈ nonzero Q_p› nonzero-def )
      then show x ∈ {r ∈ carrier Q_p. r ≠ 0 ∧ ac n r = ac n b}
        using ‹ac n x = ac n b› by force
    qed
  qed
 qed
 then show is-univ-semialgebraic (ac-cong-set1 n b)
  by (simp add: nonzero-is-univ-semialgebraic)
next
 case F: False
 have F0: 2 ≤ card (Units (Zp-res-ring n))
 proof(cases n = 1)
  case True
  then have field (Zp-res-ring n)
    using p-res-ring-1-field by blast
  then have F00: Units (Zp-res-ring n) = carrier (Zp-res-ring n) − {0_{Zp-res-ring n}}
    using field.field-Units by blast
  have F01: 0_{Zp-res-ring n} ∈ carrier (Zp-res-ring n)
   using assms(1) cring.cring-simprules(2) padic-integers.R-cring padic-integers-axioms
by blast
  have F02: card (carrier (Zp-res-ring n)) = p ∧ finite (carrier (Zp-res-ring n))
    by (smt F01 True nat-eq-iff2 p-res-ring-zero p-residue-ring-car-memE(1)
power-one-right residue-ring-card)
  have F03: 0_{residue-ring (p ^ n)} ∈ carrier (residue-ring (p ^ n))
    using F01 by blast
  have F04: int (card (carrier (residue-ring (p ^ n)))) ≥ int (card {0_{residue-ring (p ^ n)}})
```

by (*smt F02 F03 nat-int of-nat-0-le-iff of-nat-1 of-nat-power p-res-ring-0 p-res-ring-zero*
p-residue-ring-car-memE(*1*) *power-increasing power-one-right residue-ring-card*)
have *card* (*carrier* (*residue-ring* (*p* ^ *n*))) − *1* = *p* − *1*
using *F02 prime*
by (*metis Totient.of-nat-eq-1-iff True less-imp-le-nat less-one nat-int nat-less-eq-zless of-nat-1 of-nat-diff of-nat-zero-less-power-iff p-residues pos-int-cases power-0 power-one-right residue-ring-card residues.m-gt-one zero-le-one*)
hence *F05*: *card* (*carrier* (*residue-ring* (*p* ^ *n*)) − {$\mathbf{0}_{\textit{residue-ring}\ (p\ \hat{}\ n)}$}) = *p* − *1*
using *F02 F03 F04 card-Diff-singleton-if*[*of* (*carrier* (*Zp-res-ring n*)) $\mathbf{0}_{\textit{residue-ring}\ (p\hat{}n)}$]
True int-ops(*6*)[*of card* (*carrier* (*residue-ring* (*p* ^ *n*))) *card* {$\mathbf{0}_{\textit{residue-ring}\ (p\ \hat{}\ n)}$}]
p-res-ring-zero p-residue-ring-car-memE(*1*)
by (*metis*)
hence *F06*: *card* (*Units* (*Zp-res-ring n*)) = *p* −*1*
using *True F02 F01 F00*
by (*metis p-res-ring-zero*)
have *F04*: *p* − *1* ≥*2*
using *F prime*
by (*meson True linorder-cases not-less prime-ge-2-int zle-diff1-eq*)
then show *?thesis*
using *F03 F06*
by *linarith*
next
case *False*
then show *?thesis*
by (*metis assms*(*1*) *less-imp-le-nat mod2-gr-0 mod-less nat-le-linear nat-neq-iff residue-units-card-geq-2*)
qed
show *?thesis*
apply(*rule pow-res-union-imp-semialg*[*of card* (*Units* (*Zp-res-ring n*))])
using *F0 assms* **apply** *linarith*
apply (*metis* (*mono-tags*, *lifting*) *ac-cong-set1-def mem-Collect-eq not-nonzero-Qp subsetI*)
proof−
fix *x*
assume *AA*: *x* ∈ *ac-cong-set1 n b*
show *pow-res* (*card* (*Units* (*Zp-res-ring n*))) *x* ⊆ *ac-cong-set1 n b*
proof
fix *y*
assume *A*: *y* ∈ *pow-res* (*card* (*Units* (*Zp-res-ring n*))) *x*
show *y* ∈ *ac-cong-set1 n b*
proof−
obtain *k* **where** *k-def*: *k* = *card* (*Units* (*Zp-res-ring n*))
by *blast*
have *k* ≥*2*
using *assms k-def F F0* **by** *blast*
then obtain *a* **where** *a-def*: *a* ∈ *nonzero* Q_p ∧ *y* = *x* ⊗ (*a*[^]*k*)
using *k-def A pow-res-def*[*of k x*]

by *blast*
have *0*: $x \in$ *nonzero* Q_p
using *AA ac-cong-set1-def*
by (*metis* (*mono-tags*, *lifting*) *mem-Collect-eq not-nonzero-Qp*)
have *1*: $y \in$ *nonzero* Q_p
by (*metis 0 Qp.Units-m-closed Qp-nat-pow-nonzero Units-eq-nonzero* ‹$\bigwedge$*thesis*. ($\bigwedge$*a*. $a \in$ *nonzero* $Q_p \wedge y = x \otimes a\,[\widehat{}]\,k \Longrightarrow$ *thesis*) $\Longrightarrow$ *thesis*›)
have *ac n y* = *ac n x* $\otimes_{Zp\text{-}res\text{-}ring\ n}$ *ac n* (*a*[$\widehat{}$]*k*)
using *a-def 0 1 Qp-nat-pow-nonzero ac-mult'*
by *blast*
then have *2*: *ac n y* = *ac n x* $\otimes_{Zp\text{-}res\text{-}ring\ n}$ (*ac n a*)[$\widehat{}$]$_{Zp\text{-}res\text{-}ring\ n}$ *k*
proof−
have *ac n* (*a*[$\widehat{}$]*k*) = *ac n a* [$\widehat{}$]$_{Zp\text{-}res\text{-}ring\ n}$ *k*
using *a-def assms*(*1*) *ac-nat-pow'*[*of a n k*]
by *linarith*
then show *?thesis*
using ‹*ac n y* = *ac n x* $\otimes_{Zp\text{-}res\text{-}ring\ n}$ *ac n* (*a*[$\widehat{}$]*k*)›
by *presburger*
qed
then have *ac n y* = *ac n x*
proof−
have (*ac n a*) ∈ *Units* (*Zp-res-ring n*)
by (*metis* (*mono-tags*, *opaque-lifting*) *a-def ac-units assms*(*1*))
then have (*ac n a*)$\widehat{}$*k mod* (*p*$\widehat{}$*n*) = *1*
using *k-def a-def ac-nat-pow ac-nat-pow' assms*(*1*) *residue-units-nilpotent*
using *neq0-conv* **by** *presburger*
then have *00*: (*ac n a*)[$\widehat{}$]$_{Zp\text{-}res\text{-}ring\ n}$ *k* = *1*
by (*metis a-def ac-nat-pow ac-nat-pow' mod-by-1 power-0*
zero-neq-one)
have *ac n x* $\otimes_{residue\text{-}ring\ (p\ \widehat{}\ n)}$ *ac n a* [$\widehat{}$]$_{residue\text{-}ring\ (p\ \widehat{}\ n)}$ *k* = *ac n x* $\otimes_{Zp\text{-}res\text{-}ring\ n}$ $\mathbf{1}_{Zp\text{-}res\text{-}ring\ n}$
using *00 assms*(*1*) *p-res-ring-one* **by** *presburger*
hence *ac n x* $\otimes_{residue\text{-}ring\ (p\ \widehat{}\ n)}$ *ac n a* [$\widehat{}$]$_{residue\text{-}ring\ (p\ \widehat{}\ n)}$ *k* = *ac n x*
by (*metis 0 Qp.nonzero-closed Qp.one-nonzero Qp.r-one ac-mult' ac-one' assms*(*1*))
then show *?thesis*
using *2 0 00*
by *linarith*
qed
then show *?thesis*
using *1 AA nonzero-def*
ac-cong-set1-def[*of n b*] *mem-Collect-eq*
by *smt*
qed
qed
qed

definition *ac-cong-set* **where**

ac-cong-set n k = $\{x \in$ *carrier* Q_p. $x \neq \mathbf{0} \wedge$ *ac n x* = *k*$\}$

lemma *ac-cong-set-is-univ-semialg*:
assumes *n* >*0*
assumes *k* ∈ *Units* (*Zp-res-ring n*)
shows *is-univ-semialgebraic* (*ac-cong-set n k*)
proof−
have *k* ∈ *carrier* (*Zp-res-ring n*)
using *assms*(*2*) *Units-def*[*of Zp-res-ring n*]
by *blast*
then have *k-n*: ([*k*]·$_{Z_p}$**1**$_{Z_p}$) *n* = *k*
using *assms*
by (*metis Zp-int-inc-res mod-pos-pos-trivial p-residue-ring-car-memE*(*1*) *p-residue-ring-car-memE*(*2*))
obtain *b* **where** *b-def*: *b* = *ι* ([*k*]·$_{Z_p}$**1**$_{Z_p}$)
by *blast*
have *0*: *k mod p* ≠ *0*
using *assms residue-UnitsE*[*of n k*]
by (*metis le-eq-less-or-eq le-refl less-one nat-le-linear p-residues power-0*
power-one-right residues.mod-in-res-units residues-def zero-less-one
zero-neq-one zero-not-in-residue-units zero-power)
then have *val-Zp* ([*k*]·$_{Z_p}$**1**$_{Z_p}$) = *0*
using *val-Zp-p-int-unit* **by** *blast*
then have *1*: *val b* = *0*
by (*metis Zp-int-inc-closed b-def val-of-inc*)
have *2*: *b* ∈ $\mathcal{O}_p$
using *b-def Zp-int-mult-closed*
by *blast*
have *ord-Zp* ([*k*] ·$_{Z_p}$ **1**$_{Z_p}$) = *0*
using *0 ord-Zp-p-int-unit* **by** *blast*
have *ac-Zp* ([*k*]·$_{Z_p}$**1**$_{Z_p}$) = ([*k*]·$_{Z_p}$**1**$_{Z_p}$)
using *0 Zp-int-inc-closed ac-Zp-of-Unit ord-Zp-p-int-unit* ‹*val-Zp* ([*k*] ·$_{Z_p}$ **1**$_{Z_p}$)
= *0*›
by *blast*
then have (*angular-component b*) = ([*k*]·$_{Z_p}$**1**$_{Z_p}$)
using *b-def 1 2 angular-component-ord-zero*[*of b*]
by (*metis Qp.int-inc-zero Qp.one-closed val-ring-memE Zp.int-inc-zero Zp.one-closed*
Zp.one-nonzero Zp-int-inc-closed angular-component-of-inclusion inc-closed
inc-of-int
inc-of-one inc-to-Zp local.val-zero not-nonzero-Qp val-ineq val-one zero-in-val-ring)
then have *ac n b* = *k*
using *ac-def*[*of n b*] *k-n*
by (*metis Qp-char-0-int Zp-defs*(*1*) *ac-def b-def inc-of-int inc-of-one*)
then have *3*: (*ac-cong-set n k*) = (*ac-cong-set1 n b*)
unfolding *ac-cong-set-def ac-cong-set1-def*
by *meson*
have *4*: *b* ∈ *nonzero* Q_p
using *1 2 val-nonzero*
by (*metis Qp.one-closed val-ring-memE Zp-def ι-def local.one-neq-zero*
not-nonzero-Qp padic-fields.val-ring-memE padic-fields-axioms val-ineq val-one)

then show *?thesis*
using *1 2 3 assms ac-cong-set1-is-univ-semialg*[*of n b*] *val-nonzero*[*of b 1*]
by *presburger*
qed

definition *val-ring-constant-ac-set* **where**
val-ring-constant-ac-set n k = {$a \in \mathcal{O}_p$. *val a* = *0* ∧ *ac n a* = *k*}

lemma *val-nonzero′*:
assumes $a \in$ *carrier* Q_p
assumes *val a* = *eint k*
shows $a \in$ *nonzero* Q_p
using *val-nonzero*[*of a k* + *1*]
by (*metis Suc-ile-eq assms*(*1*) *assms*(*2*) *eint-ord-code*(*3*) *val-nonzero*)

lemma *val-ord′*:
assumes $a \in$ *carrier* Q_p
assumes $a \neq \mathbf{0}$
shows *val a* = *ord a*
by (*meson assms*(*1*) *assms*(*2*) *not-nonzero-Qp val-ord*)

lemma *val-ring-constant-ac-set-is-univ-semialgebraic*:
assumes *n* > *0*
assumes *k* ≠ *0*
shows *is-univ-semialgebraic* (*val-ring-constant-ac-set n k*)
proof(*cases val-ring-constant-ac-set n k* = {})
case *True*
then show *?thesis*
by (*metis equals0D order-refl pow-res-union-imp-semialg subsetI*)
next
case *False*
then obtain *b* **where** *b-def*: *b* ∈ *val-ring-constant-ac-set n k*
by *blast*
have *0*: *val-ring-constant-ac-set n k* = *q-ball n k 0* **0**
proof
show *val-ring-constant-ac-set n k* ⊆ *q-ball n k 0* **0**
proof fix *x* **assume** *A*: *x* ∈ *val-ring-constant-ac-set n k* **then**
show *x* ∈ *q-ball n k 0* **0**
proof−
have *0*: *x* ∈ $\mathcal{O}_p$ ∧ *val x* = *0* ∧ *ac n x* = *k*
using *A*
unfolding *val-ring-constant-ac-set-def*
by *blast*
then have *x-car*: *x* ∈ *carrier* Q_p
using *val-ring-memE*
by *blast*
then have *00*: *x* = *x* ⊖ **0**
using *Qp.ring-simprules* **by** *metis*
then have *1*: *ac n* (*x* $\ominus_{Q_p}$ **0**) = *k*

```
      using 0
      by presburger
    have 2: val (x ⊖_{Q_p} 0) = 0
      using 0 00
      by metis
    have 3: x ∈ nonzero Q_p
    proof(rule ccontr)
      assume  x ∉ nonzero Q_p
      then have x = 0
        using Qp.nonzero-memI x-car by blast
      then show False
        using 0 val-zero
        by (metis ac-def assms(2))
    qed
    have 4: ord (x ⊖_{Q_p} 0) = 0
    proof(rule ccontr)
      assume ord (x ⊖ 0) ≠ 0
      then have val (x ⊖ 0) ≠ 0
       by (metis 00 3 Qp.one-closed equal-val-imp-equal-ord(1) ord-one val-one)
      then show False
        using 2
        by blast
    qed
    show ?thesis
      using 0 1 4
      unfolding  q-ball-def
      using x-car by blast
  qed
qed
show q-ball n k 0 0 ⊆ val-ring-constant-ac-set n k
proof fix x
  assume A: x ∈ q-ball n k 0 0
  then have 0: ac n (x ⊖_{Q_p} 0) = k
    using q-ballE'(1) by blast
  have 1: ord (x ⊖_{Q_p} 0) = 0
    using q-ball-def A
    by blast
  have 2: x ∈ carrier Q_p
    using A q-ball-def by blast
  have 3: ord x = 0
    using 2 1 ring.ring-simprules[of Q_p]
    by (metis Qp.ring-axioms)
  have 4: ac n x = k
    using 0 2 1  cring.axioms(1)[of Q_p] ring.ring-simprules[of Q_p]
     by (metis Qp.ring-axioms)
   have 5: x ∈ 𝒪_p
     using Qp-val-ringI[of x] 2 3 val-ord val-nonzero'
  by (metis Qp.integral-iff val-ring-memE Zp.nonzero-closed angular-component-closed
            angular-component-ord-zero  image-eqI  local.numer-denom-facts(1)
```

local.numer-denom-facts(2)
local.numer-denom-facts(4) not-nonzero-Qp)
have *6*: $x \neq \mathbf{0}$
using *4 assms ac-def*[*of n x*]
by *meson*
have *7*: *val x = 0*
using *6 3 2 assms val-ord' zero-eint-def* **by** *presburger*
show $x \in$ *val-ring-constant-ac-set n k*
unfolding *val-ring-constant-ac-set-def*
using *7 6 5 4*
by *blast*
qed
qed
obtain *b* **where** *b-def*: $b \in$ *q-ball n k* (*0*::*int*) **0**
using *0 b-def* **by** *blast*
have *1*: $b \in$ *carrier* $Q_p \wedge$ *ac n b* $=$ *k*
using *b-def* **unfolding** *q-ball-def*
by (*metis* (*mono-tags*, *lifting*) *0 b-def mem-Collect-eq val-ring-constant-ac-set-def*)
then have *2*: $b \in$ *nonzero* Q_p
using *1 assms*
by (*metis ac-def not-nonzero-Qp*)
have *q-ball n k 0* **0** $= B_{0 +_{int} n}[b]$
using *1 b-def nonzero-def* [*of* Q_p] *assms 0 2 c-ball-q-ball*[*of b n k* **0** *b 0*]
by (*meson Qp.cring-axioms cring.cring-simprules(2)*)
then have *is-univ-semialgebraic* (*q-ball n k* (*0*::*int*) **0**)
using *1 ball-is-univ-semialgebraic*[*of b 0 + int n*]
by *metis*
then show *?thesis*
using *0* **by** *presburger*
qed

definition *val-ring-constant-ac-sets* **where**
val-ring-constant-ac-sets n $=$ *val-ring-constant-ac-set n* ‘ (*Units* (*Zp-res-ring n*))

lemma *val-ring-constant-ac-sets-are-univ-semialgebraic*:
assumes $n > 0$
shows *are-univ-semialgebraic* (*val-ring-constant-ac-sets n*)
proof(*rule are-univ-semialgebraicI*)
have *0*: $\neg$ *coprime 0 p*
using *coprime-0-right-iff*[*of p*] *coprime-commute*[*of p 0*] *coprime-int-iff*[*of nat p 0*]
nat-dvd-1-iff-1 prime-gt-1-nat zdvd1-eq
by (*metis not-prime-unit prime*)
have (*0*::*int*) $\notin$(*Units* (*Zp-res-ring n*))
apply(*rule ccontr*)
using *0 assms residues.cring*[*of p* $\hat{}$ *n*] **unfolding** *residues-def*
by (*smt less-one not-gr-zero power-le-imp-le-exp power-less-imp-less-exp residue-UnitsE*)
fix *x*
assume *A*: $x \in$ *val-ring-constant-ac-sets n*

then obtain k **where** *k-def*: $x = \textit{val-ring-constant-ac-set}\ n\ k \wedge k \in \textit{Units}\ (\textit{Zp-res-ring}\ n)$
by (*metis image-iff val-ring-constant-ac-sets-def*)
then show *is-univ-semialgebraic x*
using *assms*
by (*metis ‹0 ∉ Units (Zp-res-ring n)› val-ring-constant-ac-set-is-univ-semialgebraic*)
qed

definition *ac-cong-set3* **where**
$\textit{ac-cong-set3}\ n\ = \{as.\ \exists\ a\ b.\ a \in \textit{nonzero}\ Q_p \wedge b \in \mathcal{O}_p \wedge \textit{val}\ b = 0 \wedge (\textit{ac}\ n\ a = \textit{ac}\ n\ b) \wedge as = [a,\ b]\ \}$

definition *ac-cong-set2* **where**
$\textit{ac-cong-set2}\ n\ k = \{as.\ \exists\ a\ b.\ a \in \textit{nonzero}\ Q_p \wedge b \in \mathcal{O}_p \wedge \textit{val}\ b = 0 \wedge (\textit{ac}\ n\ a = k) \wedge (\textit{ac}\ n\ b) = k \wedge as = [a,\ b]\ \}$

lemma *ac-cong-set2-cartesian-product*:
assumes $k \in \textit{Units}\ (\textit{Zp-res-ring}\ n)$
assumes $n > 0$
shows $\textit{ac-cong-set2}\ n\ k = \textit{cartesian-product}\ (\textit{to-R1‘}\ (\textit{ac-cong-set}\ n\ k))\ (\textit{to-R1‘}\ (\textit{val-ring-constant-ac-set}\ n\ k))$
proof
show $\textit{ac-cong-set2}\ n\ k \subseteq \textit{cartesian-product}\ ((\lambda a.\ [a])\ ‘\ \textit{ac-cong-set}\ n\ k)\ ((\lambda a.\ [a])\ ‘\ \textit{val-ring-constant-ac-set}\ n\ k)$
proof fix x
assume *A*: $x \in \textit{ac-cong-set2}\ n\ k$
show $x \in (\textit{cartesian-product}\ ((\lambda a.\ [a])\ ‘\ \textit{ac-cong-set}\ n\ k)\ ((\lambda a.\ [a])\ ‘\ \textit{val-ring-constant-ac-set}\ n\ k))$
unfolding *ac-cong-set-def val-ring-constant-ac-set-def ac-cong-set2-def*
apply(*rule cartesian-product-memI*[*of - Q_p 1 - 1*])
apply (*metis* (*mono-tags*, *lifting*) *mem-Collect-eq subsetI Qp.to-R1-car-subset*)
apply (*metis* (*no-types*, *lifting*) *val-ring-memE mem-Collect-eq subsetI Qp.to-R1-car-subset*)
proof−
obtain *a b* **where** *ab-def*: $x = [a,b] \wedge a \in \textit{nonzero}\ Q_p \wedge b \in \mathcal{O}_p \wedge \textit{val}\ b = 0 \wedge (\textit{ac}\ n\ a = k) \wedge (\textit{ac}\ n\ b) = k$
using *A*
unfolding *ac-cong-set-def val-ring-constant-ac-set-def ac-cong-set2-def*
by *blast*
have *0*: *take 1 x* = [*a*]
by (*simp add*: *ab-def*)
have *1*: *drop 1 x* = [*b*]
by (*simp add*: *ab-def*)
have *2*: $a \in \{x \in \textit{carrier}\ Q_p.\ x \neq \mathbf{0} \wedge \textit{ac}\ n\ x = k\}$
using *ab-def nonzero-def*
by (*smt mem-Collect-eq*)
have *3*: $b \in \{a \in \mathcal{O}_p.\ \textit{val}\ a = 0 \wedge \textit{ac}\ n\ a = k\}$
using *ab-def*
by *blast*

show *take 1 x* ∈ (λ*a*. [*a*]) ‘ {*x* ∈ *carrier* Q_p. *x* ≠ **0** ∧ *ac n x* = *k*}
using *0 2* **by** *blast*
show *drop 1 x* ∈ (λ*a*. [*a*]) ‘ {*a* ∈ $\mathcal{O}_p$. *val a* = *0* ∧ *ac n a* = *k*}
using *1 3* **by** *blast*
qed
qed
show *cartesian-product* ((λ*a*. [*a*]) ‘ *ac-cong-set n k*) ((λ*a*. [*a*]) ‘ *val-ring-constant-ac-set n k*) ⊆ *ac-cong-set2 n k*
proof fix *x*
have *0*: (λ*a*. [*a*]) ‘ *ac-cong-set n k* ⊆ *carrier* ($Q_p{}^1$)
using *assms*
by (*metis* (*no-types*, *lifting*) *ac-cong-set-def mem-Collect-eq subsetI Qp.to-R1-car-subset*)
have *1*: ((λ*a*. [*a*]) ‘ *val-ring-constant-ac-set n k*) ⊆ *carrier* ($Q_p{}^1$)
by (*smt val-ring-memE mem-Collect-eq subsetI Qp.to-R1-carrier Qp.to-R1-subset val-ring-constant-ac-set-def*)
assume *A*: *x* ∈ *cartesian-product* ((λ*a*. [*a*]) ‘ *ac-cong-set n k*) ((λ*a*. [*a*]) ‘ *val-ring-constant-ac-set n k*)
then have *length x* = *2*
using *0 1 A cartesian-product-closed*[*of* ((λ*a*. [*a*]) ‘ *ac-cong-set n k*) Q_p *1* ((λ*a*. [*a*]) ‘ *val-ring-constant-ac-set n k*) *1*]
by (*metis* (*no-types*, *lifting*) *cartesian-power-car-memE one-add-one subset-iff*)
then obtain *a b* **where** *ab-def*: *take 1 x* = [*a*] ∧ *drop 1 x* = [*b*]
by (*metis One-nat-def add-diff-cancel-left′ drop0 drop-Cons-numeral numerals*(*1*) *pair-id plus-1-eq-Suc take0 take-Cons-numeral*)
have *2*: *a* ∈ (*ac-cong-set n k*) ∧ *b* ∈ (*val-ring-constant-ac-set n k*)
proof−
have *P0*: *take 1 x* ∈ (λ*a*. [*a*]) ‘ *ac-cong-set n k*
using *0 A cartesian-product-memE*[*of x* ((λ*a*. [*a*]) ‘ *ac-cong-set n k*) ((λ*a*. [*a*]) ‘ *val-ring-constant-ac-set n k*) Q_p *1*]
by *blast*
have *P1*: *drop 1 x* ∈ (λ*a*. [*a*]) ‘ *val-ring-constant-ac-set n k*
using *0 A cartesian-product-memE*[*of x* ((λ*a*. [*a*]) ‘ *ac-cong-set n k*) ((λ*a*. [*a*]) ‘ *val-ring-constant-ac-set n k*) Q_p *1*]
by *blast*
have *P2*: [*a*] ∈ (λ*a*. [*a*]) ‘ *ac-cong-set n k*
using *P0 ab-def*
by *metis*
have *P3*: [*b*] ∈ (λ*a*. [*a*]) ‘ *val-ring-constant-ac-set n k*
using *P1 ab-def* **by** *metis*
show *?thesis*
using *P2 P3*
by *blast*
qed
have *3*: *a* ∈ *nonzero* Q_p
using *2 assms nonzero-def* [*of* Q_p] *ac-cong-set-def*[*of n k*]
by *blast*
have *4*: *x* = [*a*,*b*]
by (*metis* (*no-types*, *lifting*) ‹*length x* = *2*› *ab-def less-numeral-extra*(*1*) *nth-Cons-0 nth-take nth-via-drop pair-id*)

then have $\exists\, a\ b.\ a \in \mathit{nonzero}\ Q_p \wedge b \in \mathcal{O}_p \wedge \mathit{val}\ b = 0 \wedge \mathit{ac}\ n\ a = k \wedge \mathit{ac}\ n\ b = k \wedge x = [a, b]$
using *2 3 ab-def* **unfolding** *val-ring-constant-ac-set-def ac-cong-set-def*
by *blast*
then show $x \in$ *ac-cong-set2 n k*
unfolding *ac-cong-set2-def val-ring-constant-ac-set-def ac-cong-set-def*
by *blast*
qed
qed

lemma *ac-cong-set2-is-semialg*:
assumes $k \in$ *Units* (*Zp-res-ring n*)
assumes $n > 0$
shows *is-semialgebraic 2* (*ac-cong-set2 n k*)
using *ac-cong-set-is-univ-semialg ac-cong-set2-cartesian-product*[*of k n*]
cartesian-product-is-semialgebraic[*of 1* ((λ*a.* [*a*]) ‘ *ac-cong-set n k*) *1* ((λ*a.* [*a*]) ‘ *val-ring-constant-ac-set n k*)]
by (*metis assms*(*1*) *assms*(*2*) *is-univ-semialgebraicE less-one less-or-eq-imp-le nat-neq-iff*
one-add-one val-ring-constant-ac-set-is-univ-semialgebraic zero-not-in-residue-units)

lemma *ac-cong-set3-as-union*:
assumes $n > 0$
shows *ac-cong-set3 n* = $\bigcup$ (*ac-cong-set2 n* ‘ (*Units* (*Zp-res-ring n*)))
proof
show *ac-cong-set3 n* ⊆ $\bigcup$ (*ac-cong-set2 n* ‘ *Units* (*Zp-res-ring n*))
proof fix *x* **assume** *A*: $x \in$ *ac-cong-set3 n*
then have *0*: $x \in$ (*ac-cong-set2 n* (*ac n* (*x*!*0*)))
unfolding *ac-cong-set2-def ac-cong-set3-def*
by (*smt mem-Collect-eq nth-Cons-0*)
have *1*: (*ac n* (*x*!*0*)) ∈ *Units* (*Zp-res-ring n*)
using *A* **unfolding** *ac-cong-set3-def*
by (*smt ac-units assms mem-Collect-eq nth-Cons-0*)
then show $x \in \bigcup$ (*ac-cong-set2 n* ‘ *Units* (*Zp-res-ring n*))
using *0*
by *blast*
qed
show $\bigcup$ (*ac-cong-set2 n* ‘ *Units* (*Zp-res-ring n*)) ⊆ *ac-cong-set3 n*
proof fix *x* **assume** *A*: $x \in \bigcup$ (*ac-cong-set2 n* ‘ *Units* (*Zp-res-ring n*))
obtain *k* **where** *k-def*: $x \in$ (*ac-cong-set2 n k*) $\wedge$ $k \in$ (*Units* (*Zp-res-ring n*))
using *A* **by** *blast*
have *0*: *k mod p* ≠ *0*
using *k-def One-nat-def Suc-le-eq assms less-numeral-extra*(*1*)
power-one-right residues.m-gt-one residues.mod-in-res-units
by (*metis p-residues residue-UnitsE zero-not-in-residue-units*)
obtain *b* **where** *b-def*: $b = ([k] \cdot_{Z_p} \mathbf{1}_{Z_p})$
by *blast*
have *k* ≠*0*
using *0 mod-0*

```
    by blast
  then have 1: b ∈ nonzero Z_p
    using 0 b-def int-unit
    by (metis Zp.Units-nonzero Zp.zero-not-one)
  have 10: ord-Zp b = 0 using 0 1
    using b-def ord-Zp-p-int-unit by blast
  have 2: ι b ∈ nonzero Q_p using k-def
    using 1 inc-of-nonzero by blast
  have 3: angular-component (ι b) = ac-Zp b
    using 1 angular-component-of-inclusion
    by blast
  have 4: ac-Zp b = b
    using 1 10
    by (metis 3 Zp.r-one ac-Zp-factors′ angular-component-closed inc-of-nonzero
int-pow-0 mult-comm ord-Zp-def)
  have 5: ac-Zp b n = k
  proof−
    have k ∈ carrier (Zp-res-ring n)
      using k-def unfolding Units-def by blast
    then show ?thesis
      using b-def k-def 4 Zp-int-inc-res  mod-pos-pos-trivial
      by (metis p-residue-ring-car-memE(1) p-residue-ring-car-memE(2))
  qed
  then have ac n (ι b) = k
    using 10 1 2 3 4 unfolding ac-def
    using Qp.not-nonzero-memI by metis
  then show x ∈ ac-cong-set3 n
    unfolding ac-cong-set3-def
    using k-def unfolding ac-cong-set2-def
    by (smt mem-Collect-eq)
 qed
qed

lemma ac-cong-set3-is-semialgebraic:
 assumes n > 0
 shows is-semialgebraic 2 (ac-cong-set3 n)
proof−
 have 0: finite (ac-cong-set2 n ‘ (Units (Zp-res-ring n)) )
   using assms residues.finite-Units[of p^n] unfolding residues-def
   using p-residues residues.finite-Units by blast
 have 1: are-semialgebraic 2 (ac-cong-set2 n ‘ (Units (Zp-res-ring n)) )
   apply(rule are-semialgebraicI)
   using ac-cong-set2-is-semialg assms by blast
 show ?thesis
   using 0 1 ac-cong-set3-as-union
  by (metis (no-types, lifting) are-semialgebraicE assms finite-union-is-semialgebraic′
is-semialgebraicE subsetI)
qed
```

13.9 Permutations of indices of semialgebraic sets

lemma *fun-inv-permute*:
assumes σ *permutes* $\{..<n\}$
shows *fun-inv* σ *permutes* $\{..<n\}$
$\sigma \circ (\textit{fun-inv}\ \sigma) = id$
$(\textit{fun-inv}\ \sigma) \circ \sigma = id$
using *assms* **unfolding** *fun-inv-def*
using *permutes-inv* **apply** *blast*
using *assms permutes-inv-o*(*1*) **apply** *blast*
using *assms permutes-inv-o*(*2*) **by** *blast*

lemma *poly-tuple-pullback-eq-poly-map-vimage*:
assumes *is-poly-tuple n fs*
assumes *length fs* $= m$
assumes $S \subseteq$ *carrir* $({Q_p}^m)$
shows *poly-map n fs* $^{-1}_n$ S = *poly-tuple-pullback n S fs*
unfolding *poly-map-def poly-tuple-pullback-def evimage-def restrict-def*
using *assms*
by (*smt vimage-inter-cong*)

lemma *permutation-is-semialgebraic*:
assumes *is-semialgebraic n S*
assumes σ *permutes* $\{..<n\}$
shows *is-semialgebraic n* (*permute-list* σ ‘ S)
proof−
have $S \subseteq$ *carrier* $({Q_p}^n)$
using *assms gen-boolean-algebra-subset is-semialgebraic-def semialg-sets-def*
by *blast*
then have (*permute-list* σ ‘ S) = *poly-tuple-pullback n S* (*permute-list* (*fun-inv* σ) (*pvar-list* Q_p *n*))
using *Qp.cring-axioms assms pullback-by-permutation-of-poly-list'*[*of* σ *n S*]
unfolding *poly-map-def*
by *blast*
then have *0*: (*permute-list* σ ‘ S) = *poly-tuple-pullback n S* (*permute-list* (*fun-inv* σ) (*pvar-list* Q_p *n*))
using *poly-tuple-pullback-def*
by *blast*
have *1*: (*fun-inv* σ) *permutes* $\{..<n\}$
using *assms* **unfolding** *fun-inv-def*
using *permutes-inv* **by** *blast*
then show *?thesis* **using** *1 pullback-is-semialg*[*of n* (*permute-list* (*fun-inv* σ) (*pvar-list* Q_p *n*))]
permutation-of-poly-list-is-poly-list[*of n* (*pvar-list* Q_p *n*) *fun-inv* σ]
pvar-list-is-poly-tuple[*of n*] *assms poly-tuple-pullback-eq-poly-map-vimage*
by (*metis 0* ‹$S \subseteq$ *carrier* $({Q_p}^n)$› *is-semialgebraic-def length-permute-list pvar-list-length*)
qed

lemma *permute-list-closed*:

assumes $a \in$ *carrier* $({Q_p}^n)$
assumes σ *permutes* $\{..<n\}$
shows *permute-list* σ $a \in$ *carrier* $({Q_p}^n)$
apply(*rule cartesian-power-car-memI*)
using *assms cartesian-power-car-memE length-permute-list* **apply** *blast*
using *assms cartesian-power-car-memE'' permute-list-set* **by** *blast*

lemma *permute-list-closed'*:
assumes σ *permutes* $\{..<n\}$
assumes *permute-list* σ $a \in$ *carrier* $({Q_p}^n)$
shows $a \in$ *carrier* $({Q_p}^n)$
apply(*rule cartesian-power-car-memI*)
apply (*metis assms*(*2*) *cartesian-power-car-memE length-permute-list*)
using *assms cartesian-power-car-memE'*[*of permute-list* σ a Q_p n]
by (*metis cartesian-power-car-memE in-set-conv-nth length-permute-list set-permute-list subsetI*)

lemma *permute-list-compose-inv*:
assumes σ *permutes* $\{..<n\}$
assumes $a \in$ *carrier* $({Q_p}^n)$
shows *permute-list* σ (*permute-list* (*fun-inv* σ) a) $= a$
permute-list (*fun-inv* σ) (*permute-list* σ a) $= a$
using *assms* **apply** (*metis cartesian-power-car-memE fun-inv-permute*(*3*) *permute-list-compose permute-list-id*)
using *assms* **by** (*metis cartesian-power-car-memE fun-inv-permute*(*2*) *fun-inv-permute*(*1*) *permute-list-compose permute-list-id*)

lemma *permutation-is-semialgebraic-imp-is-semialgebraic*:
assumes *is-semialgebraic n* (*permute-list* σ ‘ S)
assumes σ *permutes* $\{..<n\}$
shows *is-semialgebraic n S*
proof−
have *permute-list* (*fun-inv* σ) ‘ (*permute-list* σ ‘ S) $= S$
proof−
have *0*: (*permute-list* σ ‘ S) $\subseteq$ *carrier* $({Q_p}^n)$
using *assms* **unfolding** *is-semialgebraic-def semialg-sets-def*
using *gen-boolean-algebra-subset* **by** *blast*
have *1*: $S \subseteq$ *carrier* $({Q_p}^n)$
proof fix x **assume** $x \in S$ **then show** $x \in$ *carrier* $({Q_p}^n)$
using *0 assms*
by (*meson image-subset-iff permute-list-closed'*)
qed
show *?thesis*
proof show *permute-list* (*fun-inv* σ) ‘ *permute-list* σ ‘ $S \subseteq S$
using *0 assms permute-list-compose-inv*[*of* σ] *1 image-iff image-subset-iff subsetD*
by *smt*
show $S \subseteq$ *permute-list* (*fun-inv* σ) ‘ *permute-list* σ ‘ S
using *0 assms permute-list-compose-inv*[*of* σ]

by (*smt 1 image-iff subset-eq*)
qed
qed
then show *?thesis* **using** *permutation-is-semialgebraic*
by (*metis assms*(*1*) *assms*(*2*) *fun-inv-permute*(*1*))
qed

lemma *split-cartesian-product-is-semialgebraic*:
assumes $i \leq n$
assumes *is-semialgebraic n A*
assumes *is-semialgebraic m B*
shows *is-semialgebraic* (*n* + *m*) (*split-cartesian-product n m i A B*)
using *assms cartesian-product-is-semialgebraic scp-permutes*[*of i n m*]
permutation-is-semialgebraic[*of n* + *m cartesian-product A B* (*scp-permutation n m i*)]
unfolding *split-cartesian-product-def*
by *blast*

definition *reverse-val-relation-set* **where**
reverse-val-relation-set = {*as* ∈ *carrier* ($Q_p{}^2$). *val* (*as* ! *0*) ≤ *val* (*as* ! *1*)}

lemma *Qp-2-car-memE*:
assumes *x* ∈ *carrier* ($Q_p{}^2$)
shows *x* = [*x*!*0*, *x*!*1*]
proof−
have *length x* = *2*
using *assms cartesian-power-car-memE* **by** *blast*
then show *?thesis*
using *pair-id* **by** *blast*
qed

definition *flip* **where**
flip = (λ*i*::*nat*. (*if i* = *0 then 1 else* (*if i* = *1 then 0 else i*)))

lemma *flip-permutes*:
flip permutes {*0*,*1*}
unfolding *permutes-def flip-def*
by (*smt mem-simps*(*1*))

lemma *flip-eval*:
flip 0 = *1*
flip 1 = *0*
unfolding *flip-def*
by *auto*

lemma *flip-x*:
assumes *x* ∈ *carrier* ($Q_p{}^2$)
shows *permute-list flip x* = [*x*!*1*, *x*!*0*]
proof−

have *0*: $x = [x!0,\ x!1]$
using *assms Qp-2-car-memE* **by** *blast*
have *1*: *length* (*permute-list flip x*) = *length* $[x!1,\ x!0]$
using *0* **unfolding** *permute-list-def*
by (*metis length-Cons length-map map-nth*)
have *2*: $\bigwedge i.\ i < 2 \Longrightarrow$ *permute-list flip x* ! *i* = $[x!1,\ x!0]$! *i*
proof− **fix** *i*::*nat* **assume** *A*: *i* < *2*
show *permute-list flip x* ! *i* = $[x!1,\ x!0]$! *i*
using *0* **unfolding** *permute-list-def*
by (*smt flip-eval*(*1*) *flip-eval*(*2*) *length-Cons length-greater-0-conv list.simps*(*8*) *map-upt-Suc numeral-nat*(*7*) *upt-rec*)
qed
have $\bigwedge i.$ *i* < *length x* $\Longrightarrow$ *permute-list flip x* ! *i* = $[x!1,\ x!0]$! *i*
proof−
have *0*: *length x* = *2*
using *assms cartesian-power-car-memE* **by** *blast*
show $\bigwedge i.$ *i* < *length x* $\Longrightarrow$ *permute-list flip x* ! *i* = $[x!1,\ x!0]$! *i* **using** *2* **unfolding** *0*
by *blast*
qed
thus *?thesis* **using** *1*
by (*metis length-permute-list nth-equalityI*)
qed

lemma *permute-with-flip-closed*:
assumes *x* ∈ *carrier* (${Q_p}^{2::nat}$)
shows *permute-list flip x* ∈ *carrier* (${Q_p}^{2::nat}$)
apply(*rule permute-list-closed*)
using *assms* **apply** *blast*
proof−
have $\{0::nat,\ 1\} = \{..<2::nat\}$
by *auto*
thus *flip permutes* $\{..<2\}$
using *flip-permutes*
by *auto*
qed

lemma *reverse-val-relation-set-semialg*:
is-semialgebraic 2 reverse-val-relation-set
proof−
have *1*: *reverse-val-relation-set* = *permute-list flip* ‘ *val-relation-set*
apply(*rule equalityI'*)
proof−
show $\bigwedge x.$ *x* ∈ *reverse-val-relation-set* $\Longrightarrow$ *x* ∈ *permute-list flip* ‘ *val-relation-set*
proof− **fix** *x* **assume** *A*: *x* ∈ *reverse-val-relation-set*
have *0*: *permute-list flip x* = $[x\ !\ 1,\ x\ !\ 0]$
using *flip-x*[*of x*] *A* **unfolding** *reverse-val-relation-set-def*
by *blast*
have *1*: *permute-list flip x* ∈ *carrier* (${Q_p}^{2}$)

apply(*rule permute-with-flip-closed*) **using** *A* **unfolding** *reverse-val-relation-set-def*
by *blast*
have *2*: *permute-list flip x ∈ val-relation-set*
using *1 A* **unfolding** *0 reverse-val-relation-set-def val-relation-set-def mem-Collect-eq*
by (*metis Qp-2-car-memE list-hd list-tl*)
show *x ∈ permute-list flip ‘ val-relation-set*
using *flip-x*[*of x*] *A* **unfolding** *reverse-val-relation-set-def val-relation-set-def*
mem-Collect-eq
by (*metis* (*no-types*, *lifting*) *1 2 Qp-2-car-memE flip-x image-eqI list-tl*
nth-Cons-0 val-relation-set-def)
qed
show ⋀*x*. *x ∈ permute-list flip ‘ val-relation-set ⟹ x ∈ reverse-val-relation-set*
proof− **fix** *x* **assume** *a*: *x ∈ permute-list flip ‘ val-relation-set*
then obtain *y* **where** *y-def*: *y ∈ val-relation-set ∧x = permute-list flip y*
by *blast*
have *y-closed*: *y ∈ carrier* ($Q_p{}^2$)
using *y-def basic-semialg-set-memE*(*1*) *val-relation-semialg* **by** *blast*
have *y-length*: *length y = 2*
using *y-def basic-semialg-set-memE val-relation-semialg*
by (*metis cartesian-power-car-memE*)
obtain *a b* **where** *ab-def*: *y* = [*a*,*b*]
using *y-length pair-id* **by** *blast*
have *0*: *a = y*!*0*
using *ab-def*
by (*metis nth-Cons-0*)
have *1*: *b = y*!*1*
using *ab-def*
by (*metis cancel-comm-monoid-add-class.diff-cancel eq-numeral-extra*(*2*)
nth-Cons′)
have *a-closed*: *a ∈ carrier Q*$_p$
using *0 y-closed* **unfolding** *0*
by (*meson cartesian-power-car-memE′ rel-simps*(*75*) *zero-order*(*5*))
have *b-closed*: *b ∈ carrier Q*$_p$
proof−
have *1 <* (*2*::*nat*) **by** *linarith*
thus *?thesis*
using *y-closed* **unfolding** *1*
by (*meson cartesian-power-car-memE′*)
qed
have *2*: *x* = [*b*, *a*] **using** *flip-x*[*of y*] *y-def y-closed* **unfolding** *ab-def* **un-**
folding *0 1*
using ‹*y ∈ carrier* ($Q_p{}^2$) ⟹ *permute-list flip y* = [*y* ! *1*, *y* ! *0*]› *y-closed*
y-def **by** *presburger*
have *x-closed*: *x ∈ carrier* ($Q_p{}^2$)
using *y-def* **unfolding** *val-relation-set-def* **using** *permute-with-flip-closed*[*of*
y]
by *blast*
show *x ∈ reverse-val-relation-set*
using *x-closed y-def*

unfolding *val-relation-set-def reverse-val-relation-set-def mem-Collect-eq 2 0 1*
by (*metis Qp-2-car-memE list-hd list-tl*)
qed
qed
show *?thesis* **unfolding** *1*
apply(*rule permutation-is-semialgebraic*)
using *val-relation-is-semialgebraic* **apply** *blast*
using *flip-permutes*
by (*metis Suc-1 insert-commute lessThan-0 lessThan-Suc numeral-nat*(*7*))
qed

definition *strict-val-relation-set* **where**
strict-val-relation-set = {*as* ∈ *carrier* (*Q*$_p$^{2}). *val* (*as* ! *0*) < *val* (*as* ! *1*)}

definition *val-diag* **where**
val-diag = {*as* ∈ *carrier* (*Q*$_p$^{2}). *val* (*as* ! *0*) = *val* (*as* ! *1*)}

lemma *val-diag-semialg*:
is-semialgebraic 2 val-diag
proof−
have *val-diag* = *val-relation-set* ∩*reverse-val-relation-set*
apply(*rule equalityI'*)
apply(*rule IntI*)
unfolding *val-diag-def val-relation-set-def reverse-val-relation-set-def mem-Collect-eq*
apply *simp*
apply *simp*
apply(*erule IntE*) **unfolding** *mem-Collect-eq*
using *basic-trans-rules*(*24*) **by** *blast*
then show *?thesis* **using** *intersection-is-semialg*
by (*simp add*: *reverse-val-relation-set-semialg val-relation-is-semialgebraic*)
qed

lemma *strict-val-relation-set-is-semialg*:
is-semialgebraic 2 strict-val-relation-set
proof−
have *0*: *strict-val-relation-set* = *reverse-val-relation-set* − *val-diag*
apply(*rule equalityI'*)
apply(*rule DiffI*)
unfolding *strict-val-relation-set-def val-diag-def val-relation-set-def reverse-val-relation-set-def mem-Collect-eq*
using *order-le-less* **apply** *blast*
proof
show ⋀*x*. *x* ∈ *carrier* (*Q*$_p$^{2}) ∧ *val* (*x* ! *0*) < *val* (*x* ! *1*) ⟹ *x* ∈ *carrier* (*Q*$_p$^{2}) ∧ *val* (*x* ! *0*) = *val* (*x* ! *1*) ⟹ *False*
using *order-less-le* **by** *blast*
show ⋀*x*. *x* ∈ {*as* ∈ *carrier* (*Q*$_p$^{2}). *val* (*as* ! *0*) ≤ *val* (*as* ! *1*)} − {*as* ∈ *carrier* (*Q*$_p$^{2}). *val* (*as* ! *0*) = *val* (*as* ! *1*)} ⟹
x ∈ *carrier* (*Q*$_p$^{2}) ∧ *val* (*x* ! *0*) < *val* (*x* ! *1*)

```
    apply(erule DiffE) unfolding mem-Collect-eq using order-le-less by blast
  qed
  show ?thesis unfolding 0
   apply(rule diff-is-semialgebraic )
   using reverse-val-relation-set-semialg apply blast
      using val-diag-semialg by blast
qed

lemma singleton-length:
  length [a] = 1
  by auto

lemma take-closed':
  assumes m > 0
  assumes x ∈ carrier (Q_p^(m+l))
  shows take m x ∈ carrier (Q_p^m)
  apply(rule take-closed[of m m+l])
  apply simp using assms by blast

lemma triple-val-ineq-set-semialg:
  shows is-semialgebraic 3 {as ∈ carrier (Q_p^3). val (as!0) ≤ val (as!1) ∧ val
(as!1) ≤ val (as!2)}
proof−
  have 0: is-semialgebraic 3 {as ∈ carrier (Q_p^3). val (as!0) ≤ val (as!1)}
  proof−
    have 0: {as ∈ carrier (Q_p^3). val (as!0) ≤ val (as!1)} = cartesian-product
(reverse-val-relation-set) (carrier (Q_p^1))
   proof(rule equalityI')
     show ⋀x. x ∈ {as ∈ carrier (Q_p^3). val (as ! 0) ≤ val (as ! 1)} ⟹ x ∈
cartesian-product reverse-val-relation-set (carrier (Q_p^1))
    proof− fix x assume A: x ∈ {as ∈ carrier (Q_p^3). val (as ! 0) ≤ val (as !
1)}
      then have 0: length x = 3 unfolding mem-Collect-eq
       using cartesian-power-car-memE by blast
      obtain a where a-def: a = [x!0, x!1]
       by blast
      have a-length: length a = 2
      proof−
       have a = x!0 #[x!1]
         unfolding a-def
         by blast
       thus ?thesis using length-Cons[of x!0 [x!1]] unfolding singleton-length[of
x!1]
         by presburger
      qed
      obtain b where b-def: b = [x!2]
       by blast
      have b-length: length b = 1
       unfolding b-def singleton-length by auto
```

have *a-closed*: $a \in$ *reverse-val-relation-set*
proof−
have *0*: $a =$ *take 2 x*
apply(*rule nth-equalityI*)
unfolding *a-length 0 length-take*[*of 2 x*]
apply *linarith*
proof− **fix** *i*::*nat* **assume** *a*: $i < 2$ **show** $a\ !\ i =$ *take 2 x* $!\ i$
apply(*cases* $i = 0$)
apply (*metis a-def nth-Cons-0 nth-take zero-less-numeral*)
by (*smt 0* ‹*length* (*take 2 x*) = *min* (*length x*) *2*› *a-def linorder-neqE-nat min.commute min.strict-order-iff nth-take numeral-eq-iff one-less-numeral-iff pair-id pos2 rel-simps*(*22*) *rel-simps*(*48*) *rel-simps*(*9*) *semiring-norm*(*81*))
qed
have *1*: $a \in$ *carrier* $({Q_p}^2)$
apply(*rule cartesian-power-car-memI'*)
apply (*simp add*: *a-length*)
unfolding *0* **using** *A* **unfolding** *mem-Collect-eq*
using *cartesian-power-car-memE'* **by** *fastforce*
show *?thesis* **using** *1 A* **unfolding** *a-def reverse-val-relation-set-def A mem-Collect-eq*
by (*metis Qp-2-car-memE list-tl nth-Cons-0*)
qed
have *b-closed*: $b \in$ *carrier* $({Q_p}^1)$
apply(*rule cartesian-power-car-memI*)
unfolding *b-length* **apply** *blast*
apply(*rule subsetI*)
unfolding *b-def* **using** *A* **unfolding** *mem-Collect-eq* **using** *cartesian-power-car-memE'*[*of x* Q_p *3*::*nat 2*::*nat*]
by *simp*
have *2*: $x = a@b$
apply(*rule nth-equalityI*)
using *0* **unfolding** *a-length b-length length-append*[*of a b*] **apply** *presburger*
proof− **fix** *i* **assume** *A*: $i <$ *length x*
then have *A1*: $i < 3$
unfolding *0* **by** *blast*
show $x\ !\ i = (a\ @\ b)\ !\ i$
apply(*cases* $i = 0$)
apply (*metis a-def append.simps*(*2*) *nth-Cons-0*)
apply(*cases* (*i*:: *nat*) = *1*)
apply (*simp add*: *a-def*)

proof− **assume** *a*: $i \neq 0$ $i \neq 1$
then have $i = 2$
using *A1* **by** *presburger*
thus *?thesis*
by (*metis a-length b-def nth-append-length*)
qed
qed
have *3*: $a =$ *take 2 x*

apply(*rule nth-equalityI*)
unfolding *a-length 0 length-take*[*of 2 x*]
apply *linarith*
proof− **fix** *i*::*nat* **assume** *a*: *i* < *2* **show** *a* ! *i* = *take 2 x* ! *i*
apply(*cases i* = *0*)
apply (*metis a-def nth-Cons-0 nth-take zero-less-numeral*)
by (*smt 0* ‹*length* (*take 2 x*) = *min* (*length x*) *2*› *a-def linorder-neqE-nat min.commute min.strict-order-iff nth-take numeral-eq-iff one-less-numeral-iff pair-id pos2 rel-simps*(*22*) *rel-simps*(*48*) *rel-simps*(*9*) *semiring-norm*(*81*))
qed
show *x* ∈ *cartesian-product reverse-val-relation-set* (*carrier* (${Q_p}^1$))
apply(*rule cartesian-product-memI*[*of - Q_p 2 - 1*])
apply (*simp add*: *is-semialgebraic-closed reverse-val-relation-set-semialg*)
apply *blast*
using *3 a-closed* **apply** *blast*
proof−
have *drop 2 x* = *b*
unfolding *2* **unfolding** *3* **using** *0*
by *simp*
then show *drop 2 x* ∈ *carrier* (${Q_p}^1$)
using *b-closed* **by** *blast*
qed
qed
show ⋀*x*. *x* ∈ *cartesian-product reverse-val-relation-set* (*carrier* (${Q_p}^1$)) ⟹ *x* ∈ {*as* ∈ *carrier* (${Q_p}^3$). *val* (*as* ! *0*) ≤ *val* (*as* ! *1*)}
proof fix *x* **assume** *A*: *x* ∈ *cartesian-product reverse-val-relation-set* (*carrier* (${Q_p}^1$))
then obtain *a b* **where** *ab-def*: *a* ∈ *reverse-val-relation-set b* ∈ *carrier* (${Q_p}^1$) *x* = *a*@*b*
using *cartesian-product-memE*′[*of x reverse-val-relation-set carrier* (${Q_p}^1$)]
by *metis*
have *a-length*: *length a* = *2*
using *ab-def* **unfolding** *reverse-val-relation-set-def*
using *cartesian-power-car-memE* **by** *blast*
have (*0*::*nat*)< *2* **by** *presburger*
hence *0*: *x*!*0* = *a*!*0*
unfolding *ab-def* **using** *a-length*
by (*metis append.simps*(*2*) *nth-Cons-0 pair-id*)
have (*1*::*nat*)< *2* **by** *presburger*
hence *1*: *x*!*1* = *a*!*1*
unfolding *ab-def* **using** *a-length*
by (*metis append.simps*(*2*) *less-2-cases nth-Cons-0 nth-Cons-Suc pair-id*)
obtain *b*′ **where** *b*′*-def*: *b* = [*b*′]
using *ab-def cartesian-power-car-memE*
by (*metis* (*no-types*, *opaque-lifting*) *append-Cons append-Nil append-eq-append-conv min-list.cases singleton-length*)
have *b*′*-closed*: *b*′ ∈ *carrier Q_p*
using *b*′*-def ab-def cartesian-power-car-memE*
by (*metis Qp.R1-memE*′ *list-hd*)

have *x-closed*: $x \in carrier\ ({Q_p}^3)$
using *ab-def cartesian-power-append*[*of a* Q_p *2 b'*] *b'-def b'-closed*
unfolding *b'-def ab-def*(*3*) *reverse-val-relation-set-def mem-Collect-eq*
by *simp*
show $x \in carrier\ ({Q_p}^3) \wedge val\ (x\ !\ 0) \leq val\ (x\ !\ 1)$
using *x-closed ab-def* **unfolding** *reverse-val-relation-set-def mem-Collect-eq 0 1* **by** *blast*
qed
qed
show *?thesis* **unfolding** *0*
using *cartesian-product-is-semialgebraic*[*of 2 reverse-val-relation-set 1 carrier* $({Q_p}^1)$]
by (*simp add*: *carrier-is-semialgebraic reverse-val-relation-set-semialg*)
qed
have *1*: *is-semialgebraic 3* $\{as \in carrier\ ({Q_p}^3).\ val\ (as!1) \leq val\ (as!2)\}$
proof−
have *0*: $\{as \in carrier\ ({Q_p}^3).\ val\ (as!1) \leq val\ (as!2)\} = cartesian\text{-}product\ (carrier\ ({Q_p}^1))\ (reverse\text{-}val\text{-}relation\text{-}set)$
proof(*rule equalityI'*)
show $\bigwedge x.\ x \in \{as \in carrier\ ({Q_p}^3).\ val\ (as\ !\ 1) \leq val\ (as\ !\ 2)\} \Longrightarrow x \in cartesian\text{-}product\ (carrier\ ({Q_p}^1))\ reverse\text{-}val\text{-}relation\text{-}set$
proof−
fix *x* **assume** *A*: $x \in \{as \in carrier\ ({Q_p}^3).\ val\ (as\ !\ 1) \leq val\ (as\ !\ 2)\}$
then have *0*: *length x = 3* **unfolding** *mem-Collect-eq*
using *cartesian-power-car-memE* **by** *blast*
obtain *a* **where** *a-def*: *a* = [*x*!*1*, *x*!*2*]
by *blast*
have *a-length*: *length a = 2*
proof−
have *a* = *x*!*1* #[*x*!*2*]
unfolding *a-def*
by *blast*
thus *?thesis* **using** *length-Cons*[*of x*!*1* [*x*!*2*]] **unfolding** *singleton-length*[*of x*!*2*]
by *presburger*
qed
obtain *b* **where** *b-def*: *b* = [*x*!*0*]
by *blast*
have *b-length*: *length b = 1*
unfolding *b-def singleton-length* **by** *auto*
have *a-closed*: *a* ∈ *reverse-val-relation-set*
proof−
have *0*: *a = drop 1 x*
apply(*rule nth-equalityI*)
unfolding *a-length 0 length-drop*[*of 1 x*]
apply *linarith*
proof− **fix** *i*::*nat* **assume** *a*: *i < 2* **show** *a* ! *i* = *drop 1 x* ! *i*
apply(*cases i = 0*)
unfolding *a-def* **using** *nth-drop*[*of 1 x i*]

apply (*metis* (*no-types*, *opaque-lifting*) *0 a-def arith-extra-simps*(*6*) *diff-is-0-eq'' eq-imp-le eq-numeral-extra*(*1*) *flip-def flip-eval*(*1*) *less-numeral-extra*(*1*) *less-one less-or-eq-imp-le nat-add-left-cancel-le nat-le-linear nat-less-le nth-Cons-0 nth-drop numeral-neq-zero trans-less-add2 zero-less-diff*)
apply(*cases i = 1*)
using *nth-drop*[*of 1 x i*] **unfolding** *0*
apply (*metis 0 a-def a-length list.simps*(*1*) *nat-1-add-1 nth-drop one-le-numeral pair-id semiring-norm*(*3*))
using *a* **by** *presburger*
qed
have *1*: *a* ∈ *carrier* (*Q*$_p$^*2*)
using *a-def A drop-closed*[*of 1 3 x Q*$_p$] **unfolding** *0 mem-Collect-eq*
by (*metis One-nat-def Suc-1 diff-Suc-1 numeral-3-eq-3 rel-simps*(*49*) *semiring-norm*(*77*))
show *?thesis* **using** *1 A* **unfolding** *a-def reverse-val-relation-set-def A mem-Collect-eq*
by (*metis Qp-2-car-memE list-tl nth-Cons-0*)
qed
have *b-closed*: *b* ∈ *carrier* (*Q*$_p$^*1*)
apply(*rule cartesian-power-car-memI*)
unfolding *b-length* **apply** *blast*
apply(*rule subsetI*)
unfolding *b-def* **using** *A* **unfolding** *mem-Collect-eq* **using** *cartesian-power-car-memE'*[*of x Q*$_p$ *3*::*nat 0*::*nat*]
by (*metis b-def b-length in-set-conv-nth less-one Qp.to-R-to-R1 zero-less-numeral*)
have *2*: *x* = *b*@*a*
apply(*rule nth-equalityI*)
using *0* **unfolding** *a-length b-length length-append*[*of b a*] **apply** *presburger*
proof− **fix** *i* **assume** *A*: *i* < *length x*
then have *A1*: *i* < *3*
unfolding *0* **by** *blast*
show *x* ! *i* = (*b* @ *a*) ! *i*
apply(*cases i = 0*)
apply (*metis append.simps*(*2*) *b-def nth-Cons-0*)
apply(*cases* (*i*:: *nat*) = (*1*::*nat*))
using *append.simps a-def nth-Cons*
apply (*metis b-length nth-append-length*)
apply(*cases* (*i*:: *nat*) = (*2*::*nat*))
using *A* **unfolding** *0*
apply (*metis a-def a-length arith-special*(*3*) *b-length list.inject nth-append-length-plus pair-id*)
proof− **assume** *A0*: *i* ≠*0 i* ≠ *1 i* ≠*2*
then have *i* ≥ *3* **by** *presburger*
then show *x* ! *i* = (*b* @ *a*) ! *i*
using *A* **unfolding** *0* **by** *presburger*
qed
qed
have *3*: *a* = *drop 1 x*
apply(*rule nth-equalityI*)

unfolding *a-length 0 length-drop*[*of 1 x*]
apply *linarith*
proof− **fix** *i*::*nat* **assume** *a*: *i* < *2* **show** *a* ! *i* = *drop 1 x* ! *i*
apply(*cases i* = *0*)
unfolding *a-def* **using** *nth-drop*[*of 1 x i*]
apply (*metis* (*no-types*, *opaque-lifting*) *0 a-def arith-extra-simps*(*6*) *diff-is-0-eq*′ *eq-imp-le eq-numeral-extra*(*1*) *flip-def flip-eval*(*1*) *less-numeral-extra*(*1*) *less-one less-or-eq-imp-le nat-add-left-cancel-le nat-le-linear nat-less-le nth-Cons-0 nth-drop numeral-neq-zero trans-less-add2 zero-less-diff*)
apply(*cases i* = *1*)
using *nth-drop*[*of 1 x i*] **unfolding** *0*
apply (*metis 0 a-def a-length list.simps*(*1*) *nat-1-add-1 nth-drop one-le-numeral pair-id semiring-norm*(*3*))
using *a* **by** *presburger*
qed
show *x* ∈ *cartesian-product* (*carrier* ($Q_p{}^1$)) *reverse-val-relation-set*
apply(*rule cartesian-product-memI*[*of - Q*$_p$ *1 - 2*])
apply (*simp add*: *is-semialgebraic-closed reverse-val-relation-set-semialg*)
using *reverse-val-relation-set-def* **apply** *blast*
using *take-closed*[*of 1 3 x*] *A* **unfolding** *mem-Collect-eq* **apply** *auto*[*1*]
using *a-closed* **unfolding** *3* **by** *blast*
qed
show ⋀*x*. *x* ∈ *cartesian-product* (*carrier* ($Q_p{}^1$)) *reverse-val-relation-set* ⟹ *x* ∈ {*as* ∈ *carrier* ($Q_p{}^3$). *val* (*as* ! *1*) ≤ *val* (*as* ! *2*)}
proof fix *x* **assume** *A*: *x* ∈ *cartesian-product* (*carrier* ($Q_p{}^1$)) *reverse-val-relation-set*

then obtain *a b* **where** *ab-def*: *a* ∈ *reverse-val-relation-set b* ∈ *carrier* ($Q_p{}^1$) *x* = *b*@*a*
using *cartesian-product-memE*′[*of x carrier* ($Q_p{}^1$) *reverse-val-relation-set*]
by *metis*
have *a-length*: *length a* = *2*
using *ab-def* **unfolding** *reverse-val-relation-set-def*
using *cartesian-power-car-memE* **by** *blast*
obtain *b*′ **where** *b*′*-def*: *b* = [*b*′]
using *ab-def cartesian-power-car-memE*
by (*metis* (*no-types*, *opaque-lifting*) *append-Cons append-Nil append-eq-append-conv min-list.cases singleton-length*)
have *b*′*-closed*: *b*′ ∈ *carrier Q*$_p$
using *b*′*-def ab-def cartesian-power-car-memE*
by (*metis Qp.R1-memE*′ *list-hd*)
have *b-length*: *length b* = *1*
by (*simp add*: *b*′*-def*)
have *x-id*: *x* = *b*′#*a*
unfolding *ab-def b*′*-def* **by** *auto*
have (*1*::*nat*)< *2* **by** *presburger*
hence *0*: *x*!*1* = *a*!*0*
unfolding *ab-def b*′*-def* **using** *a-length*
by (*metis b*′*-def b-length nth-append-length pair-id*)
have *00*: *2* = *Suc 1*

by *auto*
have *1*: $x!2 = a!1$
using *a-length nth-Cons*[*of* b' *a 2*::*nat*]
unfolding *x-id 00*
by (*meson nth-Cons-Suc*)
have *x-closed*: $x \in carrier\ ({Q_p}^3)$
unfolding *x-id* b'*-def* **using** b'*-closed cartesian-power-cons*[*of a* Q_p *2* b'] *ab-def*
unfolding *reverse-val-relation-set-def mem-Collect-eq*
by *simp*

show $x \in carrier\ ({Q_p}^3) \wedge val\ (x\ !\ 1) \leq val\ (x\ !\ 2)$
using *x-closed ab-def* **unfolding** *reverse-val-relation-set-def mem-Collect-eq 0 1* **by** *blast*
qed
qed
show *?thesis* **unfolding** *0*
using *cartesian-product-is-semialgebraic*[*of 2 reverse-val-relation-set 1 carrier* $({Q_p}^1)$]
by (*metis add-num-simps*(*2*) *car-times-semialg-is-semialg one-plus-numeral reverse-val-relation-set-semialg*)
qed
have *2*: $\{as \in carrier\ ({Q_p}^3).\ val\ (as!0) \leq val\ (as!1) \wedge val\ (as!1) \leq val\ (as!2)\} =$
$\{as \in carrier\ ({Q_p}^3).\ val\ (as!0) \leq val\ (as!1)\} \cap \{as \in carrier\ ({Q_p}^3).\ val\ (as!1) \leq val\ (as!2)\}$
by *blast*
show *?thesis* **using** *intersection-is-semialg 0 1* **unfolding** *2* **by** *blast*
qed

lemma *triple-val-ineq-set-semialg'*:
shows *is-semialgebraic 3* $\{as \in carrier\ ({Q_p}^3).\ val\ (as!0) \leq val\ (as!1) \wedge val\ (as!1) < val\ (as!2)\}$
proof−
have *0*: *is-semialgebraic 3* $\{as \in carrier\ ({Q_p}^3).\ val\ (as!0) \leq val\ (as!1)\}$
proof−
have *0*: $\{as \in carrier\ ({Q_p}^3).\ val\ (as!0) \leq val\ (as!1)\} = cartesian\text{-}product\ (reverse\text{-}val\text{-}relation\text{-}set)\ (carrier\ ({Q_p}^1))$
proof(*rule equalityI'*)
show $\bigwedge x.\ x \in \{as \in carrier\ ({Q_p}^3).\ val\ (as\ !\ 0) \leq val\ (as\ !\ 1)\} \Longrightarrow x \in cartesian\text{-}product\ reverse\text{-}val\text{-}relation\text{-}set\ (carrier\ ({Q_p}^1))$
proof− **fix** *x* **assume** *A*: $x \in \{as \in carrier\ ({Q_p}^3).\ val\ (as\ !\ 0) \leq val\ (as\ !\ 1)\}$
then have *0*: *length x* = *3* **unfolding** *mem-Collect-eq*
using *cartesian-power-car-memE* **by** *blast*
obtain *a* **where** *a-def*: $a = [x!0,\ x!1]$
by *blast*
have *a-length*: *length a* = *2*
proof−
have $a = x!0\ \#[x!1]$

```
      unfolding a-def
      by blast
    thus ?thesis using length-Cons[of x!0 [x!1]] unfolding singleton-length[of
x!1]
      by presburger
  qed
  obtain b where b-def: b = [x!2]
    by blast
  have b-length: length b = 1
    unfolding b-def singleton-length by auto
  have a-closed: a ∈ reverse-val-relation-set
  proof−
    have 0: a = take 2 x
      apply(rule nth-equalityI)
      unfolding a-length 0 length-take[of 2 x]
       apply linarith
    proof− fix i::nat assume a: i < 2 show a ! i = take 2 x ! i
       apply(cases i = 0)
        apply (metis a-def nth-Cons-0 nth-take zero-less-numeral)
      by (smt 0 ‹length (take 2 x) = min (length x) 2› a-def linorder-neqE-nat
min.commute min.strict-order-iff nth-take numeral-eq-iff one-less-numeral-iff pair-id
pos2 rel-simps(22) rel-simps(48) rel-simps(9) semiring-norm(81))
    qed
    have 1: a ∈ carrier (Q_p^2)
      using a-def 0 A unfolding mem-Collect-eq
     by (meson Qp-2I cartesian-power-car-memE' rel-simps(49) rel-simps(51)
semiring-norm(77))
      show ?thesis using 1 A unfolding a-def reverse-val-relation-set-def A
mem-Collect-eq
       by (metis Qp-2-car-memE list-tl nth-Cons-0)
  qed
  have b-closed: b ∈ carrier (Q_p^1)
    apply(rule cartesian-power-car-memI)
    unfolding b-length apply blast
    apply(rule subsetI)
       unfolding b-def using A unfolding mem-Collect-eq using carte-
sian-power-car-memE'[of x Q_p 3::nat 2::nat]
    by simp
  have 2: x = a@b
    apply(rule nth-equalityI)
  using 0 unfolding a-length b-length length-append[of a b] apply presburger
  proof− fix i assume A: i < length x
    then have A1: i < 3
      unfolding 0 by blast
    show x ! i = (a @ b) ! i
      apply(cases i = 0)
       apply (metis a-def append.simps(2) nth-Cons-0)
      apply(cases (i:: nat) = 1)
       apply (simp add: a-def)
```

proof− **assume** *a*: $i \neq 0$ $i \neq 1$
then have $i = 2$
using *A1* **by** *presburger*
thus *?thesis*
by (*metis a-length b-def nth-append-length*)
qed
qed
have *3*: $a = take\ 2\ x$
apply(*rule nth-equalityI*)
unfolding *a-length 0 length-take*[*of 2 x*]
apply *linarith*
proof− **fix** *i*::*nat* **assume** *a*: $i < 2$ **show** $a\ !\ i = take\ 2\ x\ !\ i$
apply(*cases i = 0*)
apply (*metis a-def nth-Cons-0 nth-take zero-less-numeral*)
by (*smt 0* ‹*length* (*take 2 x*) = *min* (*length x*) *2*› *a-def linorder-neqE-nat min.commute min.strict-order-iff nth-take numeral-eq-iff one-less-numeral-iff pair-id pos2 rel-simps*(*22*) *rel-simps*(*48*) *rel-simps*(*9*) *semiring-norm*(*81*))
qed
show $x \in$ *cartesian-product reverse-val-relation-set* (*carrier* ($Q_p\ ^1$))
apply(*rule cartesian-product-memI*[*of - Q_p 2 - 1*])
apply (*simp add*: *is-semialgebraic-closed reverse-val-relation-set-semialg*)
apply *blast*
using *3 a-closed* **apply** *blast*
proof−
have *drop 2 x* = *b*
unfolding *2* **unfolding** *3* **using** *0*
by *simp*
then show *drop 2 x* ∈ *carrier* ($Q_p\ ^1$)
using *b-closed* **by** *blast*
qed
qed
show $\bigwedge x.\ x \in$ *cartesian-product reverse-val-relation-set* (*carrier* ($Q_p\ ^1$)) $\Longrightarrow$ $x \in \{as \in$ *carrier* ($Q_p\ ^3$). *val* (*as* ! *0*) ≤ *val* (*as* ! *1*)$\}$
proof fix *x* **assume** *A*: $x \in$ *cartesian-product reverse-val-relation-set* (*carrier* ($Q_p\ ^1$))
then obtain *a b* **where** *ab-def*: $a \in$ *reverse-val-relation-set* $b \in$ *carrier* ($Q_p\ ^1$) $x = a@b$
using *cartesian-product-memE'*[*of x reverse-val-relation-set carrier* ($Q_p\ ^1$)]
by *metis*
have *a-length*: *length a* = *2*
using *ab-def* **unfolding** *reverse-val-relation-set-def*
using *cartesian-power-car-memE* **by** *blast*
have (*0*::*nat*)< *2* **by** *presburger*
hence *0*: *x*!*0* = *a*!*0*
unfolding *ab-def* **using** *a-length*
by (*metis append.simps*(*2*) *nth-Cons-0 pair-id*)
have (*1*::*nat*)< *2* **by** *presburger*
hence *1*: *x*!*1* = *a*!*1*
unfolding *ab-def* **using** *a-length*

by (*metis append.simps*(*2*) *less-2-cases nth-Cons-0 nth-Cons-Suc pair-id*)
obtain b' **where** b'*-def*: $b = [b']$
using *ab-def cartesian-power-car-memE*
by (*metis* (*no-types*, *opaque-lifting*) *append-Cons append-Nil append-eq-append-conv min-list.cases singleton-length*)
have b'*-closed*: $b' \in$ *carrier* Q_p
using b'*-def ab-def cartesian-power-car-memE*
by (*metis Qp.R1-memE*$'$ *list-hd*)
have *x-closed*: $x \in$ *carrier* $({Q_p}^3)$
using *ab-def cartesian-power-append*[*of a* Q_p *2* b'] b'*-def* b'*-closed*
unfolding b'*-def ab-def*(*3*) *reverse-val-relation-set-def mem-Collect-eq*
by *simp*
show $x \in$ *carrier* $({Q_p}^3) \wedge$ *val* $(x\ !\ 0) \leq$ *val* $(x\ !\ 1)$
using *x-closed ab-def* **unfolding** *reverse-val-relation-set-def mem-Collect-eq 0 1* **by** *blast*
qed
qed
show *?thesis* **unfolding** *0*
using *cartesian-product-is-semialgebraic*[*of 2 reverse-val-relation-set 1 carrier* $({Q_p}^1)$]
by (*simp add*: *carrier-is-semialgebraic reverse-val-relation-set-semialg*)
qed
have *1*: *is-semialgebraic 3* $\{as \in$ *carrier* $({Q_p}^3)$. *val* $(as!1) <$ *val* $(as!2)\}$
proof−
have *0*: $\{as \in$ *carrier* $({Q_p}^3)$. *val* $(as!1) <$ *val* $(as!2)\} =$ *cartesian-product* (*carrier* $({Q_p}^1)$) (*strict-val-relation-set*)
proof(*rule equalityI*$'$)
show $\bigwedge x.\ x \in \{as \in$ *carrier* $({Q_p}^3)$. *val* $(as\ !\ 1) <$ *val* $(as\ !\ 2)\} \Longrightarrow x \in$ *cartesian-product* (*carrier* $({Q_p}^1)$) *strict-val-relation-set*
proof− **fix** x **assume** A: $x \in \{as \in$ *carrier* $({Q_p}^3)$. *val* $(as\ !\ 1) <$ *val* $(as\ !\ 2)\}$
then have *0*: *length* $x = 3$ **unfolding** *mem-Collect-eq*
using *cartesian-power-car-memE* **by** *blast*
obtain *a* **where** *a-def*: $a = [x!1, x!2]$
by *blast*
have *a-length*: *length* $a = 2$
proof−
have $a = x!1\ \#[x!2]$
unfolding *a-def*
by *blast*
thus *?thesis* **using** *length-Cons*[*of* $x!1$ $[x!2]$] **unfolding** *singleton-length*[*of* $x!2$]
by *presburger*
qed
obtain *b* **where** *b-def*: $b = [x!0]$
by *blast*
have *b-length*: *length* $b = 1$
unfolding *b-def singleton-length* **by** *auto*
have *a-closed*: $a \in$ *strict-val-relation-set*

proof−
have *0*: $a = drop\ 1\ x$
apply(*rule nth-equalityI*)
unfolding *a-length 0 length-drop*[*of 1 x*]
apply *linarith*
proof− **fix** *i*::*nat* **assume** *a*: $i < 2$ **show** $a\ !\ i = drop\ 1\ x\ !\ i$
apply(*cases* $i = 0$)
unfolding *a-def* **using** *nth-drop*[*of 1 x i*]
apply (*metis* (*no-types*, *opaque-lifting*) *0 a-def arith-extra-simps*(*6*) *diff-is-0-eq′ eq-imp-le eq-numeral-extra*(*1*) *flip-def flip-eval*(*1*) *less-numeral-extra*(*1*) *less-one less-or-eq-imp-le nat-add-left-cancel-le nat-le-linear nat-less-le nth-Cons-0 nth-drop numeral-neq-zero trans-less-add2 zero-less-diff*)
apply(*cases* $i = 1$)
using *nth-drop*[*of 1 x i*] **unfolding** *0*
apply (*metis 0 a-def a-length list.simps*(*1*) *nat-1-add-1 nth-drop one-le-numeral pair-id semiring-norm*(*3*))
using *a* **by** *presburger*
qed
have *1*: $a \in carrier\ ({Q_p}^2)$
using *a-def A drop-closed*[*of 1 3 x* Q_p] **unfolding** *0 mem-Collect-eq*
by (*metis One-nat-def Suc-1 diff-Suc-1 numeral-3-eq-3 rel-simps*(*49*) *semiring-norm*(*77*))
show *?thesis* **using** *1 A* **unfolding** *a-def strict-val-relation-set-def A mem-Collect-eq*
by (*metis Qp-2-car-memE list-tl nth-Cons-0*)
qed
have *b-closed*: $b \in carrier\ ({Q_p}^1)$
apply(*rule cartesian-power-car-memI*)
unfolding *b-length* **apply** *blast*
apply(*rule subsetI*)
unfolding *b-def* **using** *A* **unfolding** *mem-Collect-eq* **using** *cartesian-power-car-memE′*[*of x* Q_p *3*::*nat 0*::*nat*]
by (*metis b-def b-length in-set-conv-nth less-one Qp.to-R-to-R1 zero-less-numeral*)
have *2*: $x = b@a$
apply(*rule nth-equalityI*)
using *0* **unfolding** *a-length b-length length-append*[*of b a*] **apply** *presburger*
proof− **fix** *i* **assume** *A*: $i < length\ x$
then have *A1*: $i < 3$
unfolding *0* **by** *blast*
show $x\ !\ i = (b\ @\ a)\ !\ i$
apply(*cases* $i = 0$)
apply (*metis append.simps*(*2*) *b-def nth-Cons-0*)
apply(*cases* (i:: *nat*) = (*1*::*nat*))
using *append.simps a-def nth-Cons*
apply (*metis b-length nth-append-length*)
apply(*cases* (i:: *nat*) = (*2*::*nat*))
using *A* **unfolding** *0*
apply (*metis a-def a-length arith-special*(*3*) *b-length list.inject nth-append-length-plus pair-id*)

proof− **assume** *A0*: $i \neq 0$ $i \neq 1$ $i \neq 2$
then have $i \geq 3$ **by** *presburger*
then show $x\,!\,i = (b\ @\ a)\,!\,i$
using *A* **unfolding** *0* **by** *presburger*
qed
qed
have *3*: $a = drop\ 1\ x$
apply(*rule nth-equalityI*)
unfolding *a-length 0 length-drop*[*of 1 x*]
apply *linarith*
proof− **fix** *i*::*nat* **assume** *a*: $i < 2$ **show** $a\,!\,i = drop\ 1\ x\,!\,i$
apply(*cases* $i = 0$)
unfolding *a-def* **using** *nth-drop*[*of 1 x i*]
apply (*metis* (*no-types*, *opaque-lifting*) *0 a-def arith-extra-simps*(*6*) *diff-is-0-eq′ eq-imp-le eq-numeral-extra*(*1*) *flip-def flip-eval*(*1*) *less-numeral-extra*(*1*) *less-one less-or-eq-imp-le nat-add-left-cancel-le nat-le-linear nat-less-le nth-Cons-0 nth-drop numeral-neq-zero trans-less-add2 zero-less-diff*)
apply(*cases* $i = 1$)
using *nth-drop*[*of 1 x i*] **unfolding** *0*
apply (*metis 0 a-def a-length list.simps*(*1*) *nat-1-add-1 nth-drop one-le-numeral pair-id semiring-norm*(*3*))
using *a* **by** *presburger*
qed
show $x \in$ *cartesian-product* (*carrier* ($Q_p{}^1$)) *strict-val-relation-set*
apply(*rule cartesian-product-memI*[*of - Qp 1 - 2*])
apply (*simp add*: *is-semialgebraic-closed strict-val-relation-set-is-semialg*)
using *strict-val-relation-set-def* **apply** *blast*
using *take-closed*[*of 1 3 x Qp*] *A* **unfolding** *mem-Collect-eq* **apply** *auto*[*1*]
using *a-closed* **unfolding** *3* **by** *blast*
qed
show $\bigwedge x.\ x \in$ *cartesian-product* (*carrier* ($Q_p{}^1$)) *strict-val-relation-set* $\Longrightarrow x \in \{as \in$ *carrier* ($Q_p{}^3$). *val* (*as* ! *1*) < *val* (*as* ! *2*)$\}$
proof fix *x* **assume** *A*: $x \in$ *cartesian-product* (*carrier* ($Q_p{}^1$)) *strict-val-relation-set*

then obtain *a b* **where** *ab-def*: $a \in$ *strict-val-relation-set* $b \in$ *carrier* ($Q_p{}^1$) $x = b@a$
using *cartesian-product-memE′*[*of x carrier* ($Q_p{}^1$) *strict-val-relation-set*]
by *metis*
have *a-length*: *length a* = *2*
using *ab-def* **unfolding** *strict-val-relation-set-def*
using *cartesian-power-car-memE* **by** *blast*
obtain *b′* **where** *b′-def*: $b = [b']$
using *ab-def cartesian-power-car-memE*
by (*metis* (*no-types*, *opaque-lifting*) *append-Cons append-Nil append-eq-append-conv min-list.cases singleton-length*)
have *b′-closed*: $b' \in$ *carrier* Q_p
using *b′-def ab-def*
by (*metis Qp.R1-memE′ list-hd*)
have *b-length*: *length b* = *1*

by (*simp add*: *b′-def*)
have *x-id*: $x = b'\#a$
unfolding *ab-def b′-def* **by** *auto*
have (*1*::*nat*)< *2* **by** *presburger*
hence *0*: *x*!*1* = *a*!*0*
unfolding *ab-def b′-def* **using** *a-length*
by (*metis b′-def b-length nth-append-length pair-id*)
have *00*: *2* = *Suc 1*
by *auto*
have *1*: *x*!*2* = *a*!*1*
using *a-length nth-Cons*[*of b′ a 2*::*nat*]
unfolding *x-id 00*
by (*meson nth-Cons-Suc*)
have *x-closed*: *x* ∈ *carrier* ($Q_p{}^3$)
unfolding *x-id b′-def* **using** *b′-closed cartesian-power-cons*[*of a* Q_p *2 b′*] *ab-def*
unfolding *strict-val-relation-set-def mem-Collect-eq*
by *simp*
show *x* ∈ *carrier* ($Q_p{}^3$) ∧ *val* (*x* ! *1*) < *val* (*x* ! *2*)
using *x-closed ab-def* **unfolding** *strict-val-relation-set-def mem-Collect-eq 0 1* **by** *blast*
qed
qed
show *?thesis* **unfolding** *0*
using *cartesian-product-is-semialgebraic*[*of 2 reverse-val-relation-set 1 carrier* ($Q_p{}^1$)]
by (*metis add-num-simps*(*2*) *car-times-semialg-is-semialg one-plus-numeral strict-val-relation-set-is-semialg*)
qed
have *2*: {*as* ∈ *carrier* ($Q_p{}^3$). *val* (*as*!*0*) ≤ *val* (*as*!*1*) ∧ *val* (*as*!*1*) < *val* (*as*!*2*)}=
{*as* ∈ *carrier* ($Q_p{}^3$). *val* (*as*!*0*) ≤ *val* (*as*!*1*)} ∩ {*as* ∈ *carrier* ($Q_p{}^3$). *val* (*as*!*1*) < *val* (*as*!*2*)}
by *blast*
show *?thesis* **using** *intersection-is-semialg 0 1* **unfolding** *2* **by** *blast*
qed

lemma *triple-val-ineq-set-semialg′′*:
shows *is-semialgebraic 3* {*as* ∈ *carrier* ($Q_p{}^3$). *val* (*as*!*1*) < *val* (*as*!*2*)}
proof−
have *0*: {*as* ∈ *carrier* ($Q_p{}^3$). *val* (*as*!*1*) < *val* (*as*!*2*)} = *cartesian-product* (*carrier* ($Q_p{}^1$)) (*strict-val-relation-set*)
proof(*rule equalityI′*)
show ⋀*x*. *x* ∈ {*as* ∈ *carrier* ($Q_p{}^3$). *val* (*as* ! *1*) < *val* (*as* ! *2*)} ⟹ *x* ∈ *cartesian-product* (*carrier* ($Q_p{}^1$)) *strict-val-relation-set*
proof− **fix** *x* **assume** *A*: *x* ∈ {*as* ∈ *carrier* ($Q_p{}^3$). *val* (*as* ! *1*) < *val* (*as* ! *2*)}
then have *0*: *length x* = *3* **unfolding** *mem-Collect-eq*
using *cartesian-power-car-memE* **by** *blast*
obtain *a* **where** *a-def*: *a* = [*x*!*1*, *x*!*2*]

```
  by blast
 have a-length: length a = 2
 proof−
  have a = x!1 #[x!2]
   unfolding a-def
   by blast
 thus ?thesis using length-Cons[of x!1 [x!2]] unfolding singleton-length[of
x!2]
   by presburger
 qed
 obtain b where b-def: b = [x!0]
  by blast
 have b-length: length b = 1
  unfolding b-def singleton-length by auto
 have a-closed: a ∈ strict-val-relation-set
 proof−
  have 0: a = drop 1 x
   apply(rule nth-equalityI)
   unfolding a-length 0 length-drop[of 1 x]
    apply linarith
  proof− fix i::nat assume a: i < 2 show a ! i = drop 1 x ! i
     apply(cases i = 0)
     unfolding a-def using nth-drop[of 1 x i]
       apply (metis (no-types, opaque-lifting) 0 a-def arith-extra-simps(6)
diff-is-0-eq′ eq-imp-le eq-numeral-extra(1) flip-def flip-eval(1) less-numeral-extra(1)
less-one less-or-eq-imp-le nat-add-left-cancel-le nat-le-linear nat-less-le nth-Cons-0
nth-drop numeral-neq-zero trans-less-add2 zero-less-diff)
     apply(cases i = 1)
     using nth-drop[of 1 x i] unfolding 0
        apply (metis 0 a-def a-length list.simps(1) nat-1-add-1 nth-drop
one-le-numeral pair-id semiring-norm(3))
     using a by presburger
  qed
   have 1: a ∈ carrier (Q_p^2)
    using a-def A drop-closed[of 1 3 x Q_p] unfolding 0 mem-Collect-eq
     by (metis One-nat-def Suc-1 diff-Suc-1 numeral-3-eq-3 rel-simps(49)
semiring-norm(77))
    show ?thesis using 1 A unfolding a-def strict-val-relation-set-def A
mem-Collect-eq
    by (metis Qp-2-car-memE list-tl nth-Cons-0)
 qed
 have b-closed: b ∈ carrier (Q_p^1)
  apply(rule cartesian-power-car-memI)
  unfolding b-length apply blast
  apply(rule subsetI)
    unfolding b-def using A unfolding mem-Collect-eq using carte-
sian-power-car-memE′[of x Q_p 3::nat 0::nat]
 by (metis b-def b-length in-set-conv-nth less-one Qp.to-R-to-R1 zero-less-numeral)
 have 2: x = b@a
```

apply(*rule nth-equalityI*)
using *0* **unfolding** *a-length b-length length-append*[*of b a*] **apply** *presburger*
proof− **fix** *i* **assume** *A*: *i* < *length x*
then have *A1*: *i* < *3*
unfolding *0* **by** *blast*
show *x* ! *i* = (*b* @ *a*) ! *i*
apply(*cases i* = *0*)
apply (*metis append.simps*(*2*) *b-def nth-Cons-0*)
apply(*cases* (*i*:: *nat*) = (*1*::*nat*))
using *append.simps a-def nth-Cons*
apply (*metis b-length nth-append-length*)
apply(*cases* (*i*:: *nat*) = (*2*::*nat*))
using *A* **unfolding** *0*
apply (*metis a-def a-length arith-special*(*3*) *b-length list.inject nth-append-length-plus pair-id*)
proof− **assume** *A0*: *i* ≠*0 i* ≠ *1 i* ≠*2*
then have *i* ≥ *3* **by** *presburger*
then show *x* ! *i* = (*b* @ *a*) ! *i*
using *A* **unfolding** *0* **by** *presburger*
qed
qed
have *3*: *a* = *drop 1 x*
apply(*rule nth-equalityI*)
unfolding *a-length 0 length-drop*[*of 1 x*]
apply *linarith*
proof− **fix** *i*::*nat* **assume** *a*: *i* < *2* **show** *a* ! *i* = *drop 1 x* ! *i*
apply(*cases i* = *0*)
unfolding *a-def* **using** *nth-drop*[*of 1 x i*]
apply (*metis* (*no-types*, *opaque-lifting*) *0 a-def arith-extra-simps*(*6*) *diff-is-0-eq′ eq-imp-le eq-numeral-extra*(*1*) *flip-def flip-eval*(*1*) *less-numeral-extra*(*1*) *less-one less-or-eq-imp-le nat-add-left-cancel-le nat-le-linear nat-less-le nth-Cons-0 nth-drop numeral-neq-zero trans-less-add2 zero-less-diff*)
apply(*cases i* = *1*)
using *nth-drop*[*of 1 x i*] **unfolding** *0*
apply (*metis 0 a-def a-length list.simps*(*1*) *nat-1-add-1 nth-drop one-le-numeral pair-id semiring-norm*(*3*))
using *a* **by** *presburger*
qed
show *x* ∈ *cartesian-product* (*carrier* (*Q*$_p$ *1*)) *strict-val-relation-set*
apply(*rule cartesian-product-memI*[*of - Q*$_p$ *1 - 2*])
apply (*simp add*: *is-semialgebraic-closed strict-val-relation-set-is-semialg*)
using *strict-val-relation-set-def* **apply** *blast*
using *take-closed*[*of 1 3 x*] *A* **unfolding** *mem-Collect-eq*
using *one-le-numeral* **apply** *blast*
using *a-closed* **unfolding** *3* **by** *blast*
qed
show ⋀*x*. *x* ∈ *cartesian-product* (*carrier* (*Q*$_p$ *1*)) *strict-val-relation-set* ⟹ *x* ∈ {*as* ∈ *carrier* (*Q*$_p$ *3*). *val* (*as* ! *1*) < *val* (*as* ! *2*)}
proof fix *x* **assume** *A*: *x* ∈ *cartesian-product* (*carrier* (*Q*$_p$ *1*)) *strict-val-relation-set*

then obtain a b **where** *ab-def*: $a \in$ *strict-val-relation-set* $b \in$ *carrier* $(Q_p{}^1)$ $x = b@a$
using *cartesian-product-memE'*[*of* x *carrier* $(Q_p{}^1)$ *strict-val-relation-set*]
by *metis*
have *a-length*: *length* $a = 2$
using *ab-def* **unfolding** *strict-val-relation-set-def*
using *cartesian-power-car-memE* **by** *blast*
obtain b' **where** b'*-def*: $b = [b']$
using *ab-def cartesian-power-car-memE*
by (*metis* (*no-types*, *opaque-lifting*) *append-Cons append-Nil append-eq-append-conv min-list.cases singleton-length*)
have b'*-closed*: $b' \in$ *carrier* Q_p
using b'*-def ab-def cartesian-power-car-memE*
by (*metis Qp.R1-memE' list-hd*)
have *b-length*: *length* $b = 1$
by (*simp add*: b'*-def*)
have *x-id*: $x = b'\#a$
unfolding *ab-def* b'*-def* **by** *auto*
have (*1*::*nat*)< *2* **by** *presburger*
hence *0*: $x!1 = a!0$
unfolding *ab-def* b'*-def* **using** *a-length*
by (*metis* b'*-def b-length nth-append-length pair-id*)
have *00*: *2* = *Suc 1*
by *auto*
have *1*: $x!2 = a!1$
using *a-length nth-Cons*[*of* b' *a 2*::*nat*]
unfolding *x-id 00*
by (*meson nth-Cons-Suc*)
have *x-closed*: $x \in$ *carrier* $(Q_p{}^3)$
unfolding *x-id* b'*-def* **using** b'*-closed cartesian-power-cons*[*of a* Q_p *2* b'] *ab-def*
unfolding *strict-val-relation-set-def mem-Collect-eq*
by *simp*

show $x \in$ *carrier* $(Q_p{}^3) \wedge$ *val* $(x\ !\ 1) <$ *val* $(x\ !\ 2)$
using *x-closed ab-def* **unfolding** *strict-val-relation-set-def mem-Collect-eq 0 1* **by** *blast*
qed
qed
show *?thesis* **unfolding** *0*
using *cartesian-product-is-semialgebraic*[*of 2 reverse-val-relation-set 1 carrier* $(Q_p{}^1)$]
by (*metis add-num-simps*(*2*) *car-times-semialg-is-semialg one-plus-numeral strict-val-relation-set-is-semialg*)
qed

lemma *triple-val-ineq-set-semialg'''*:
shows *is-semialgebraic 3* $\{as \in$ *carrier* $(Q_p{}^3).$ *val* $(as!1) \leq$ *val* $(as!2)\}$

proof−
have *0*: $\{as \in carrier\ ({Q_p}^3).\ val\ (as!1) \leq val\ (as!2)\} = cartesian\text{-}product\ (carrier\ ({Q_p}^1))\ (reverse\text{-}val\text{-}relation\text{-}set)$
proof(*rule equalityI'*)
show $\bigwedge x.\ x \in \{as \in carrier\ ({Q_p}^3).\ val\ (as\ !\ 1) \leq val\ (as\ !\ 2)\} \Longrightarrow x \in cartesian\text{-}product\ (carrier\ ({Q_p}^1))\ reverse\text{-}val\text{-}relation\text{-}set$
proof− **fix** *x* **assume** *A*: $x \in \{as \in carrier\ ({Q_p}^3).\ val\ (as\ !\ 1) \leq val\ (as\ !\ 2)\}$
then have *0*: *length x = 3* **unfolding** *mem-Collect-eq*
using *cartesian-power-car-memE* **by** *blast*
obtain *a* **where** *a-def*: *a* = [*x*!*1*, *x*!*2*]
by *blast*
have *a-length*: *length a = 2*
proof−
have *a = x*!*1* #[*x*!*2*]
unfolding *a-def*
by *blast*
thus *?thesis* **using** *length-Cons*[*of x*!*1* [*x*!*2*]] **unfolding** *singleton-length*[*of x*!*2*]
by *presburger*
qed
obtain *b* **where** *b-def*: *b* = [*x*!*0*]
by *blast*
have *b-length*: *length b = 1*
unfolding *b-def singleton-length* **by** *auto*
have *a-closed*: *a* ∈ *reverse-val-relation-set*
proof−
have *0*: *a = drop 1 x*
apply(*rule nth-equalityI*)
unfolding *a-length 0 length-drop*[*of 1 x*]
apply *linarith*
proof− **fix** *i*::*nat* **assume** *a*: *i < 2* **show** *a ! i = drop 1 x ! i*
apply(*cases i = 0*)
unfolding *a-def* **using** *nth-drop*[*of 1 x i*]
apply (*metis* (*no-types*, *opaque-lifting*) *0 a-def arith-extra-simps*(*6*) *diff-is-0-eq' eq-imp-le eq-numeral-extra*(*1*) *flip-def flip-eval*(*1*) *less-numeral-extra*(*1*) *less-one less-or-eq-imp-le nat-add-left-cancel-le nat-le-linear nat-less-le nth-Cons-0 nth-drop numeral-neq-zero trans-less-add2 zero-less-diff*)
apply(*cases i = 1*)
using *nth-drop*[*of 1 x i*] **unfolding** *0*
apply (*metis 0 a-def a-length list.simps*(*1*) *nat-1-add-1 nth-drop one-le-numeral pair-id semiring-norm*(*3*))
using *a* **by** *presburger*
qed
have *1*: *a* ∈ *carrier* $({Q_p}^2)$
using *a-def A drop-closed*[*of 1 3 x* Q_p] **unfolding** *0 mem-Collect-eq*
by (*metis One-nat-def Suc-1 diff-Suc-1 numeral-3-eq-3 rel-simps*(*49*) *semiring-norm*(*77*))
show *?thesis* **using** *1 A* **unfolding** *a-def reverse-val-relation-set-def A*

mem-Collect-eq
by (*metis Qp-2-car-memE list-tl nth-Cons-0*)
qed
have *b-closed*: $b \in carrier\ (Q_p{}^1)$
apply(*rule cartesian-power-car-memI*)
unfolding *b-length* **apply** *blast*
apply(*rule subsetI*)
unfolding *b-def* **using** *A* **unfolding** *mem-Collect-eq* **using** *cartesian-power-car-memE'*[*of x* Q_p *3*::*nat* *0*::*nat*]
by (*metis b-def b-length in-set-conv-nth less-one Qp.to-R-to-R1 zero-less-numeral*)
have *2*: $x = b@a$
apply(*rule nth-equalityI*)
using *0* **unfolding** *a-length b-length length-append*[*of b a*] **apply** *presburger*
proof− **fix** *i* **assume** *A*: $i < length\ x$
then have *A1*: $i < 3$
unfolding *0* **by** *blast*
show $x\ !\ i = (b\ @\ a)\ !\ i$
apply(*cases* $i = 0$)
apply (*metis append.simps*(*2*) *b-def nth-Cons-0*)
apply(*cases* (i:: *nat*) = (*1*::*nat*))
using *append.simps a-def nth-Cons*
apply (*metis b-length nth-append-length*)
apply(*cases* (i:: *nat*) = (*2*::*nat*))
using *A* **unfolding** *0*
apply (*metis a-def a-length arith-special*(*3*) *b-length list.inject nth-append-length-plus pair-id*)
proof− **assume** *A0*: $i \neq 0\ i \neq 1\ i \neq 2$
then have $i \geq 3$ **by** *presburger*
then show $x\ !\ i = (b\ @\ a)\ !\ i$
using *A* **unfolding** *0* **by** *presburger*
qed
qed
have *3*: $a = drop\ 1\ x$
apply(*rule nth-equalityI*)
unfolding *a-length* *0 length-drop*[*of 1 x*]
apply *linarith*
proof− **fix** *i*::*nat* **assume** *a*: $i < 2$ **show** $a\ !\ i = drop\ 1\ x\ !\ i$
apply(*cases* $i = 0$)
unfolding *a-def* **using** *nth-drop*[*of 1 x i*]
apply (*metis* (*no-types*, *opaque-lifting*) *0 a-def arith-extra-simps*(*6*) *diff-is-0-eq'* *eq-imp-le eq-numeral-extra*(*1*) *flip-def flip-eval*(*1*) *less-numeral-extra*(*1*) *less-one less-or-eq-imp-le nat-add-left-cancel-le nat-le-linear nat-less-le nth-Cons-0 nth-drop numeral-neq-zero trans-less-add2 zero-less-diff*)
apply(*cases* $i = 1$)
using *nth-drop*[*of 1 x i*] **unfolding** *0*
apply (*metis 0 a-def a-length list.simps*(*1*) *nat-1-add-1 nth-drop one-le-numeral pair-id semiring-norm*(*3*))
using *a* **by** *presburger*
qed

show $x \in$ *cartesian-product* (*carrier* ($Q_p{}^1$)) *reverse-val-relation-set*
apply(*rule cartesian-product-memI*[*of - Q_p 1 - 2*])
apply (*simp add*: *is-semialgebraic-closed reverse-val-relation-set-semialg*)
using *reverse-val-relation-set-def* **apply** *blast*
using *take-closed*[*of 1 3 x*] *A* **unfolding** *mem-Collect-eq* **apply** *auto*[*1*]
using *a-closed* **unfolding** *3* **by** *blast*
qed
show $\bigwedge x.\ x \in$ *cartesian-product* (*carrier* ($Q_p{}^1$)) *reverse-val-relation-set* $\Longrightarrow$ $x \in \{as \in$ *carrier* ($Q_p{}^3$). *val* (*as* ! *1*) $\leq$ *val* (*as* ! *2*)$\}$
proof fix *x* **assume** *A*: $x \in$ *cartesian-product* (*carrier* ($Q_p{}^1$)) *reverse-val-relation-set*

then obtain *a b* **where** *ab-def*: $a \in$ *reverse-val-relation-set* $b \in$ *carrier* ($Q_p{}^1$) $x = b@a$
using *cartesian-product-memE'*[*of x carrier* ($Q_p{}^1$) *reverse-val-relation-set*]
by *metis*
have *a-length*: *length a* = *2*
using *ab-def* **unfolding** *reverse-val-relation-set-def*
using *cartesian-power-car-memE* **by** *blast*
obtain *b'* **where** *b'-def*: $b = [b']$
using *ab-def cartesian-power-car-memE*
by (*metis* (*no-types*, *opaque-lifting*) *append-Cons append-Nil append-eq-append-conv min-list.cases singleton-length*)
have *b'-closed*: $b' \in$ *carrier* Q_p
using *b'-def ab-def cartesian-power-car-memE*
by (*metis Qp.R1-memE' list-hd*)
have *b-length*: *length b* = *1*
by (*simp add*: *b'-def*)
have *x-id*: $x = b'\#a$
unfolding *ab-def b'-def* **by** *auto*
have (*1*::*nat*)< *2* **by** *presburger*
hence *0*: *x*!*1* = *a*!*0*
unfolding *ab-def b'-def* **using** *a-length*
by (*metis b'-def b-length nth-append-length pair-id*)
have *00*: *2* = *Suc 1*
by *auto*
have *1*: *x*!*2* = *a*!*1*
using *a-length nth-Cons*[*of b' a 2*::*nat*]
unfolding *x-id 00*
by (*meson nth-Cons-Suc*)
have *x-closed*: $x \in$ *carrier* ($Q_p{}^3$)
unfolding *x-id b'-def* **using** *b'-closed cartesian-power-cons*[*of a Q_p 2 b'*] *ab-def*
unfolding *reverse-val-relation-set-def mem-Collect-eq*
by *simp*

show $x \in$ *carrier* ($Q_p{}^3$) $\wedge$ *val* (*x* ! *1*) $\leq$ *val* (*x* ! *2*)
using *x-closed ab-def* **unfolding** *reverse-val-relation-set-def mem-Collect-eq 0 1* **by** *blast*
qed

qed
show *?thesis* **unfolding** *0*
using *cartesian-product-is-semialgebraic*[*of 2 reverse-val-relation-set 1 carrier* $(Q_p{}^1)$]
by (*metis add-num-simps*(*2*) *car-times-semialg-is-semialg one-plus-numeral reverse-val-relation-set-semialg*)
qed

13.10 Semialgebraic Functions

The most natural way to define a semialgebraic function $f : \mathbb{Q}_p^n \to \mathbb{Q}_p$ is a function whose graph is a semialgebraic subset of $\mathbb{Q}_p^{n+1}$. However, the definition given here is slightly different, and devised by Denef in [1] in order to prove Macintyre's theorem. As Denef notes, we can use Macintyre's theorem to deduce that the given definition perfectly aligns with the intuitive one.

13.10.1 Defining Semialgebraic Functions

Apply a function f to the tuple consisting of the first n indices, leaving the remaining indices unchanged

definition *partial-image* **where**
partial-image m f xs = (*f* (*take m xs*))#(*drop m xs*)

definition *partial-pullback* **where**
partial-pullback m f l S = (*partial-image m f*) $^{-1}{}_{m+l}$ *S*

lemma *partial-pullback-memE*:
assumes *as* ∈ *partial-pullback m f l S*
shows *as* ∈ *carrier* $(Q_p{}^{m\,+\,l})$ *partial-image m f as* ∈ *S*
using *assms* **apply** (*metis evimage-eq partial-pullback-def*)
using *assms* **unfolding** *partial-pullback-def*
by *blast*

lemma *partial-pullback-closed*:
partial-pullback m f l S ⊆ *carrier* $(Q_p{}^{m\,+\,l})$
using *partial-pullback-memE*(*1*) **by** *blast*

lemma *partial-pullback-memI*:
assumes *as* ∈ *carrier* $(Q_p{}^{m\,+\,k})$
assumes (*f* (*take m as*))#(*drop m as*) ∈ *S*
shows *as* ∈ *partial-pullback m f k S*
using *assms* **unfolding** *partial-pullback-def partial-image-def evimage-def*
by *blast*

lemma *partial-image-eq*:
assumes *as* ∈ *carrier* $(Q_p{}^n)$

assumes *bs* ∈ *carrier* ($Q_p{}^k$)
assumes *x* = *as* @ *bs*
shows *partial-image n f x* = (*f as*)#*bs*
proof−
have *0*: (*take n x*) = *as*
by (*metis append-eq-conv-conj assms*(*1*) *assms*(*3*) *cartesian-power-car-memE*)
have *1*: *drop n x* = *bs*
by (*metis 0 append-take-drop-id assms*(*3*) *same-append-eq*)
show *?thesis* **using** *0 1* **unfolding** *partial-image-def*
by *blast*
qed

lemma *partial-pullback-memE′*:
assumes *as* ∈ *carrier* ($Q_p{}^n$)
assumes *bs* ∈ *carrier* ($Q_p{}^k$)
assumes *x* = *as* @ *bs*
assumes *x* ∈ *partial-pullback n f k S*
shows (*f as*)#*bs* ∈ *S*
using *partial-pullback-memE*[*of x n f k S*] *partial-image-def*[*of n f x*]
by (*metis assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *partial-image-eq*)

Partial pullbacks have the same algebraic properties as pullbacks

lemma *partial-pullback-intersect*:
partial-pullback m f l (*S1* ∩ *S2*) = (*partial-pullback m f l S1*) ∩ (*partial-pullback m f l S2*)
unfolding *partial-pullback-def*
by *simp*

lemma *partial-pullback-union*:
partial-pullback m f l (*S1* ∪ *S2*) = (*partial-pullback m f l S1*) ∪ (*partial-pullback m f l S2*)
unfolding *partial-pullback-def*
by *simp*

lemma *cartesian-power-drop*:
assumes *x* ∈ *carrier* ($Q_p{}^{n+l}$)
shows *drop n x* ∈ *carrier* ($Q_p{}^l$)
apply(*rule cartesian-power-car-memI*)
using *assms cartesian-power-car-memE*
apply (*metis add-diff-cancel-left′ length-drop*)
using *assms cartesian-power-car-memE″*
by (*metis order.trans set-drop-subset*)

lemma *partial-pullback-complement*:
assumes *f* ∈ *carrier* ($Q_p{}^m$) → *carrier* Q_p
shows *partial-pullback m f l* (*carrier* ($Q_p{}^{Suc\ l}$) − *S*) = *carrier* ($Q_p{}^{m\ +\ l}$) − (*partial-pullback m f l S*)
apply(*rule equalityI*)
using *partial-pullback-def*[*of m f l* (*carrier* ($Q_p{}^{Suc\ l}$) − *S*)]

```
      partial-pullback-def[of m f l S]
   apply (smt Diff-iff evimage-Diff partial-pullback-memE(1) subsetI)
proof fix x assume A: x ∈ carrier (Q_p^(m + l)) − partial-pullback m f l S
  show x ∈ partial-pullback m f l (carrier (Q_p^(Suc l)) − S)
    apply(rule partial-pullback-memI)
    using A
     apply blast
  proof
    have 00: Suc l = l + 1
      by auto
    have 0: drop m x ∈ carrier (Q_p^l)
      by (meson A DiffD1 cartesian-power-drop)
    have 1: take m x ∈ carrier (Q_p^m)
      using A by (meson DiffD1 le-add1 take-closed)
    have f (take m x) # drop m x ∈ carrier (Q_p^(l+1))
      using assms 0 1 00 cartesian-power-cons[of drop m x Q_p l f (take m x)]
      by blast
    thus f (take m x) # drop m x ∈ carrier (Q_p^(Suc l))
      using 00 by metis
    show f (take m x) # drop m x ∉ S
      using A unfolding partial-pullback-def partial-image-def
      by blast
  qed
qed

lemma partial-pullback-carrier:
  assumes f ∈ carrier (Q_p^m) → carrier Q_p
  shows partial-pullback m f l (carrier (Q_p^(Suc l))) = carrier (Q_p^(m + l))
  apply(rule equalityI)
  using partial-pullback-memE(1) apply blast
proof fix x assume A: x ∈ carrier (Q_p^(m + l))
  show x ∈ partial-pullback m f l (carrier (Q_p^(Suc l)))
    apply(rule partial-pullback-memI)
  using A cartesian-power-drop[of x m l] assms
   apply blast
proof−
  have f (take m x) ∈ carrier Q_p
    using A  assms take-closed[of m m+l x Q_p]
    by (meson Pi-mem le-add1)
  then show f (take m x) # drop m x ∈ carrier (Q_p^(Suc l))
    using cartesian-power-drop[of x m l]
    by (metis A add.commute cartesian-power-cons plus-1-eq-Suc)
qed
qed
```

Definition 1.4 from Denef

```
definition is-semialg-function where
is-semialg-function m f = ((f ∈ carrier (Q_p^m) → carrier Q_p)  ∧
```

$(\forall\, l \geq 0.\ \forall\, S \in \textit{semialg-sets}\ (1 + l).\ \textit{is-semialgebraic}\ (m + l)$
$(\textit{partial-pullback}\ m\ f\ l\ S)))$

lemma *is-semialg-function-closed*:
assumes *is-semialg-function m f*
shows $f \in \textit{carrier}\ ({Q_p}^m) \to \textit{carrier}\ Q_p$
using *is-semialg-function-def assms* **by** *blast*

lemma *is-semialg-functionE*:
assumes *is-semialg-function m f*
assumes *is-semialgebraic* $(1 + k)$ *S*
shows *is-semialgebraic* $(m + k)$ $(\textit{partial-pullback}\ m\ f\ k\ S)$
using *is-semialg-function-def assms*
by (*meson is-semialgebraicE le0*)

lemma *is-semialg-functionI*:
assumes $f \in \textit{carrier}\ ({Q_p}^m) \to \textit{carrier}\ Q_p$
assumes $\bigwedge k\ S.\ S \in \textit{semialg-sets}\ (1 + k) \Longrightarrow \textit{is-semialgebraic}\ (m + k)\ (\textit{partial-pullback}$
$m\ f\ k\ S)$
shows *is-semialg-function m f*
using *assms* **unfolding** *is-semialg-function-def*
by *blast*

Semialgebraicity for functions can be verified on basic semialgebraic sets

lemma *is-semialg-functionI'*:
assumes $f \in \textit{carrier}\ ({Q_p}^m) \to \textit{carrier}\ Q_p$
assumes $\bigwedge k\ \ S.\ S \in \textit{basic-semialgs}\ (1 + k) \Longrightarrow \textit{is-semialgebraic}\ (m + k)$
$(\textit{partial-pullback}\ m\ f\ k\ S)$
shows *is-semialg-function m f*
apply(*rule is-semialg-functionI*)
using *assms*(*1*) **apply** *blast*
proof−
show $\bigwedge k\ S.\ S \in \textit{semialg-sets}\ (1 + k) \Longrightarrow \textit{is-semialgebraic}\ (m + k)\ (\textit{partial-pullback}$
$m\ f\ k\ S)$
proof− **fix** *k S* **assume** *A*: $S \in \textit{semialg-sets}\ (1 + k)$
show *is-semialgebraic* $(m + k)$ $(\textit{partial-pullback}\ m\ f\ k\ S)$
apply(*rule gen-boolean-algebra.induct*[*of S carrier* $({Q_p}^{1+k})$ *basic-semialgs* $(1$
$+ k)$])
using *A* **unfolding** *semialg-sets-def*
apply *blast*
using *partial-pullback-carrier assms carrier-is-semialgebraic plus-1-eq-Suc*
apply *presburger*
apply (*metis assms*(*1*) *assms*(*2*) *carrier-is-semialgebraic intersection-is-semialg*
partial-pullback-carrier partial-pullback-intersect plus-1-eq-Suc)
using *partial-pullback-union union-is-semialgebraic* **apply** *presburger*
using *assms*(*1*) *complement-is-semialg partial-pullback-complement plus-1-eq-Suc*
by *presburger*
qed
qed

Graphs of semialgebraic functions are semialgebraic

abbreviation *graph* **where**
graph ≡ *fun-graph* Q_p

lemma *graph-memE*:
assumes $f \in carrier\ ({Q_p}^m) \to carrier\ Q_p$
assumes $x \in graph\ m\ f$
shows $f\ (take\ m\ x) = x!m$
$x = (take\ m\ x)@[f\ (take\ m\ x)]$
$take\ m\ x \in carrier\ ({Q_p}^m)$
proof−
obtain *a* **where** *a-def*: $a \in carrier\ ({Q_p}^m) \wedge x = a\ @\ [f\ a]$
using *assms*
unfolding *fun-graph-def*
by *blast*
then have *0*: $a = take\ m\ x$
by (*metis append-eq-conv-conj cartesian-power-car-memE*)
then show $f\ (take\ m\ x) = x!m$
by (*metis a-def cartesian-power-car-memE nth-append-length*)
show $x = (take\ m\ x)@[f\ (take\ m\ x)]$
using *0 a-def*
by *blast*
show $take\ m\ x \in carrier\ ({Q_p}^m)$
using *0 a-def* **by** *blast*
qed

lemma *graph-memI*:
assumes $f \in carrier\ ({Q_p}^m) \to carrier\ Q_p$
assumes $f\ (take\ m\ x) = x!m$
assumes $x \in carrier\ ({Q_p}^{m+1})$
shows $x \in graph\ m\ f$
proof−
have *0*: $take\ m\ x \in carrier\ ({Q_p}^m)$
apply(*rule take-closed*[*of - m + 1*])
apply *simp*
using *assms*(*3*) **by** *blast*
have $x = (take\ m\ x)@[x!m]$
by (*metis* ‹$take\ m\ x \in carrier\ ({Q_p}^m)$› *add.commute*
assms(*3*) *cartesian-power-car-memE length-append-singleton lessI*
nth-equalityI nth-take plus-1-eq-Suc take-Suc-conv-app-nth)
then have $x = (take\ m\ x)@[f\ (take\ m\ x)]$
using *assms*(*2*)
by *presburger*
then show *?thesis*
using *assms 0*
unfolding *fun-graph-def*
by *blast*
qed

lemma *graph-mem-closed*:
assumes $f \in carrier\ ({Q_p}^m) \to carrier\ Q_p$
assumes $x \in graph\ m\ f$
shows $x \in carrier\ ({Q_p}^{m+1})$
proof(*rule cartesian-power-car-memI'*)
show $length\ x = m + 1$
using *assms graph-memE*[*of f m x*]
by (*smt Groups.add-ac*(*2*) *cartesian-power-car-memE fun-graph-def length-append-singleton mem-Collect-eq plus-1-eq-Suc*)
show $\bigwedge i.\ i < m + 1 \Longrightarrow x\ !\ i \in carrier\ Q_p$
proof− **fix** *i* **assume** *A*: $i < m + 1$
then show $x\ !\ i \in carrier\ Q_p$
proof(*cases* $i = m$)
case *True*
then show *?thesis* **using** *graph-memE*[*of f m x*]
by (*metis PiE assms*(*1*) *assms*(*2*))
next
case *False*
then show *?thesis* **using** *graph-memE*[*of f m x*]
by (*metis* ‹$i < m + 1$› *add.commute assms*(*1*) *assms*(*2*) *cartesian-power-car-memE'* *less-SucE nth-take plus-1-eq-Suc*)
qed
qed
qed

lemma *graph-closed*:
assumes $f \in carrier\ ({Q_p}^m) \to carrier\ Q_p$
shows $graph\ m\ f \subseteq carrier\ ({Q_p}^{m+1})$
using *assms graph-mem-closed*
by *blast*

The m-dimensional diagonal set is semialgebraic

notation *diagonal* (‹Δ ›)

lemma *diag-is-algebraic*:
shows *is-algebraic* $Q_p\ (n + n)\ (\Delta\ n)$
using *Qp.cring-axioms diagonal-is-algebraic*
by *blast*

lemma *diag-is-semialgebraic*:
shows *is-semialgebraic* $(n + n)\ (\Delta\ n)$
using *diag-is-algebraic is-algebraic-imp-is-semialg*
by *blast*

Transposition permutations

definition *transpose* **where**
$transpose\ i\ j = (Fun.swap\ i\ j\ id)$

lemma *transpose-permutes*:

```
  assumes i< n
  assumes j < n
  shows transpose i j permutes {..<n}
  unfolding permutes-def transpose-def
proof
  show ∀ x. x ∉ {..<n} ⟶ Fun.swap i j id x = x
    using assms by (auto simp: Transposition.transpose-def)
  show ∀ y. ∃ !x. Fun.swap i j id x = y
  proof  fix y show ∃ !x. Fun.swap i j id x = y
      using swap-id-eq[of i j y]
  by (metis eq-id-iff swap-apply(1) swap-apply(2) swap-id-eq swap-self)
qed
qed

lemma transpose-alt-def:
transpose a b x = (if x = a then b else if x = b then a else x)
  using swap-id-eq
  by (simp add: transpose-def)

definition last-to-first where
last-to-first n = (λi. if i = (n−1) then 0 else if i < n−1 then i + 1 else i)

definition first-to-last where
first-to-last n = fun-inv (last-to-first n)

lemma last-to-first-permutes:
  assumes (n::nat) > 0
  shows last-to-first n permutes {..<n}
  unfolding permutes-def
proof
  show ∀ x. x ∉ {..<n} ⟶ last-to-first n x = x
  proof fix x show  x ∉ {..<n} ⟶ last-to-first n x = x
    proof assume A: x ∉ {..<n} then have ¬ x < n
        by blast then have x ≥ n by linarith
      then show  last-to-first n x = x
        unfolding last-to-first-def using assms
        by auto
    qed
  qed
  show ∀ y. ∃ !x. last-to-first n x = y
  proof fix y
    show ∃ !x. last-to-first n x = y
    proof(cases y = 0)
      case True
      then have 0: last-to-first n (n−1) = y
        using last-to-first-def
        by (simp add: last-to-first-def)
      have 1: ⋀x. last-to-first n x = y ⟹ x = n−1
        unfolding last-to-first-def using True
```

```
      by (metis add-gr-0 less-numeral-extra(1) not-gr-zero)
    show ?thesis
      using 0 1
      by blast
  next
    case False
    then show ?thesis
    proof(cases y < n)
      case True
      then have 0: last-to-first n (y−1) = y
        using False True
        unfolding last-to-first-def
        using add.commute by auto
      have 1: ⋀x. last-to-first n x = y ⟹ x =(y−1)
        unfolding last-to-first-def
        using True False
        by auto
      show ?thesis using 0 1 by blast
    next
      case F: False
      then have 0: y ≥ n
        using not-less by blast
      then have 1: last-to-first n y = y
        by (simp add: ‹∀x. x ∉ {..<n} ⟶ last-to-first n x = x›)
      have 2: ⋀x. last-to-first n x = y ⟹ x =y
        using 0 unfolding last-to-first-def
        using False by presburger
      then show ?thesis using 1 2 by blast
    qed
  qed
qed

definition graph-swap where
graph-swap n f = permute-list ((first-to-last (n+1))) ‘ (graph n f)

lemma last-to-first-eq:
  assumes length as = n
  shows permute-list (last-to-first (n+1)) (a#as)  = (as@[a])
proof−
  have 0: ⋀i. i < (n+1) ⟹ permute-list (last-to-first (n + 1)) (a # as) ! i =
(as@[a]) ! i
  proof−
    fix i assume A: i < n+1
    show permute-list (last-to-first (n + 1)) (a # as) ! i = (as @ [a]) ! i
    proof(cases i = n)
      case True
      have 0: (as @ [a]) ! i = a
        by (metis True assms nth-append-length)
```

```
    have 1: length (a#as) = n + 1
      by (simp add: assms)
    have 2: i < length (a # as)
      using 1 A by linarith
    have 3: last-to-first (n + 1) permutes {..<length (a # as)}
      by (metis 1 add-gr-0 last-to-first-permutes less-numeral-extra(1))
   have 4: permute-list (last-to-first (n + 1)) (a # as) ! i = (a # as) ! last-to-first
(n + 1) i
      using  2 3  permute-list-nth[of last-to-first (n + 1) a#as i]
      by blast
    have 5: permute-list (last-to-first (n + 1)) (a # as) ! i = (a # as) ! 0
      using 4 unfolding last-to-first-def
      by (simp add: True)
    have 6: permute-list (last-to-first (n + 1)) (a # as) ! i = a
      using 5
      by simp
    then show ?thesis using 0   by auto
  next
    case False
    then show ?thesis
      by (smt A add.commute add.right-neutral add-diff-cancel-right' add-gr-0
       add-less-cancel-left append.simps(1) append.simps(2) assms last-to-first-def
       last-to-first-permutes less-SucE less-numeral-extra(1) list.size(3) list.size(4)
          nth-append permute-list-nth plus-1-eq-Suc)
  qed
 qed
 have 1: length (a#as) = n + 1
  by (simp add: assms)
 have 2: length (permute-list (last-to-first (n+1)) (a#as)) = n + 1
  by (metis 1 length-permute-list)
 have 3: length (as@[a]) = n + 1
  by (simp add: assms)
 then show ?thesis using 0 2
  by (metis nth-equalityI)
qed

lemma first-to-last-eq:
 assumes as ∈ carrier (Q_p^n)
 assumes a ∈ carrier Q_p
 shows permute-list (first-to-last (n+1)) (as@[a])  = (a#as)
proof−
 have length as = n
  using assms(1) cartesian-power-car-memE by blast
 then show ?thesis
 using last-to-first-eq last-to-first-permutes[of n]
      permute-list-compose-inv(2)[of  (last-to-first (n + 1)) n a # as]
 unfolding first-to-last-def
 by (metis add-gr-0 assms(1) assms(2) cartesian-power-append last-to-first-permutes
     less-one permute-list-closed' permute-list-compose-inv(2))
```

qed

lemma *graph-swapI*:
assumes $as \in carrier\ ({Q_p}^n)$
assumes $f \in carrier\ ({Q_p}^n) \to carrier\ Q_p$
shows $(f\ as)\#as \in graph\text{-}swap\ n\ f$
proof−
have *0*: $as@[f\ as] \in graph\ n\ f$
using *assms* **using** *graph-memI*[*of f n*] *fun-graph-def*
by *blast*
have *1*: $f\ as \in carrier\ Q_p$
using *assms*
by *blast*
then show *?thesis*
using *assms 0 first-to-last-eq*[*of as n f as*]
unfolding *graph-swap-def*
by (*metis image-eqI*)
qed

lemma *graph-swapE*:
assumes $x \in graph\text{-}swap\ n\ f$
assumes $f \in carrier\ ({Q_p}^n) \to carrier\ Q_p$
shows $hd\ x = f\ (tl\ x)$
proof−
obtain *y* **where** *y-def*: $y \in graph\ n\ f \wedge x = permute\text{-}list\ (first\text{-}to\text{-}last\ (n{+}1))\ y$
using *assms graph-swap-def*
by (*smt image-def mem-Collect-eq*)
then have $take\ n\ y \in carrier\ ({Q_p}^n)$
using *assms*(*2*) *graph-memE*(*3*)
by *blast*
then show $hd\ x = f\ (tl\ x)$
by (*metis* (*no-types*, *lifting*) *add.commute assms*(*2*) *cartesian-power-car-memE'*
first-to-last-eq graph-memE(*1*) *graph-memE*(*2*) *graph-mem-closed lessI*
list.sel(*1*)
list.sel(*3*) *plus-1-eq-Suc y-def*)
qed

Semialgebraic functions have semialgebraic graphs

lemma *graph-as-partial-pullback*:
assumes $f \in carrier\ ({Q_p}^n) \to carrier\ Q_p$
shows $partial\text{-}pullback\ n\ f\ 1\ (\Delta\ 1) = graph\ n\ f$
proof
show $partial\text{-}pullback\ n\ f\ 1\ (\Delta\ 1) \subseteq graph\ n\ f$
proof fix *x* **assume** *A*: $x \in partial\text{-}pullback\ n\ f\ 1\ (\Delta\ 1)$
then have *0*: $f\ (take\ n\ x)\ \#\ drop\ n\ x \in \Delta\ \ 1$
by (*metis local.partial-image-def partial-pullback-memE*(*2*))
then have *1*: $length\ (f\ (take\ n\ x)\ \#\ drop\ n\ x)\ = 2$
using *diagonal-def*
by (*metis* (*no-types*, *lifting*) *cartesian-power-car-memE mem-Collect-eq one-add-one*)

```
    then obtain b where b-def: [b] = drop n x
      by (metis list.inject pair-id)
    then have [f (take n x), b] ∈ Δ  1
      using 0
      by presburger
    then have b = f (take n x)
      using 0
        by (smt One-nat-def Qp.cring-axioms diagonal-def drop0 drop-Suc-Cons
list.inject mem-Collect-eq take-Suc-Cons)
    then have x = (take n x)@[f (take n x)]
      by (metis append-take-drop-id b-def)
    then show x ∈ graph n f using graph-memI[of f n x]
      by (metis (no-types, lifting) A ‹b = f (take n x)›
          assms b-def nth-via-drop partial-pullback-memE(1))
  qed
  show graph n f ⊆ partial-pullback n f 1 (Δ  1)
  proof fix x
    assume A: x ∈ graph n f
    then have 0: x ∈ carrier (Q_p^{n+1})
      using assms graph-mem-closed by blast
    have x = (take n x) @ [f (take n x)]
      using A graph-memE(2)[of f n x] assms
      by blast
    then have partial-image n f x =  [f (take n x), f (take n x)]
      by (metis append-take-drop-id local.partial-image-def same-append-eq)
    then have partial-image n f x ∈ Δ 1
     using assms 0 diagonal-def[of 1] Qp.cring-axioms diagonalI[of  partial-image
n f x]
       by (metis (no-types, lifting) A append-Cons append-eq-conv-conj
          cartesian-power-car-memE cartesian-power-car-memE' graph-memE(1)
          less-add-one self-append-conv2 Qp.to-R1-closed)
    then show x ∈ partial-pullback n f 1 (Δ  1)
      unfolding partial-pullback-def using 0
      by blast
  qed
qed

lemma semialg-graph:
  assumes is-semialg-function n f
  shows is-semialgebraic (n + 1) (graph n f)
  using assms graph-as-partial-pullback[of f n] unfolding is-semialg-function-def
 by (metis diag-is-semialgebraic is-semialgebraicE less-imp-le-nat less-numeral-extra(1))
```

Functions induced by polynomials are semialgebraic

```
definition var-list-segment where
var-list-segment i j = map (λi. pvar Q_p i) [i..< j]

lemma var-list-segment-length:
  assumes i ≤ j
```

shows *length* (*var-list-segment i j*) = *j* − *i*
using *assms var-list-segment-def*
by *fastforce*

lemma *var-list-segment-entry*:
assumes *k* < *j* − *i*
assumes *i* ≤ *j*
shows *var-list-segment i j* ! *k* = *pvar* Q_p (*i* + *k*)
using *assms var-list-segment-length*
unfolding *var-list-segment-def*
using *nth-map-upt* **by** *blast*

lemma *var-list-segment-is-poly-tuple*:
assumes *i* ≤*j*
assumes *j* ≤ *n*
shows *is-poly-tuple n* (*var-list-segment i j*)
apply(*rule Qp-is-poly-tupleI*)
using *assms var-list-segment-entry var-list-segment-length Qp.cring-axioms pvar-closed*[*of - n*]
by (*metis* (*no-types*, *opaque-lifting*) *add.commute add-lessD1 diff-add-inverse le-Suc-ex less-diff-conv*)

lemma *map-by-var-list-segment*:
assumes *as* ∈ *carrier* ($Q_p{}^n$)
assumes *j* ≤ *n*
assumes *i* ≤ *j*
shows *poly-map n* (*var-list-segment i j*) *as* = *list-segment i j as*
apply(*rule nth-equalityI*)
unfolding *poly-map-def var-list-segment-def list-segment-def restrict-def poly-tuple-eval-def*
apply (*metis* (*full-types*) *assms*(*1*) *length-map*)
using *assms eval-pvar*[*of - n as*] *Qp.cring-axioms length-map add.commute length-upt less-diff-conv less-imp-add-positive nth-map nth-upt trans-less-add2*
by (*smt le-add-diff-inverse2*)

lemma *map-by-var-list-segment-to-length*:
assumes *as* ∈ *carrier* ($Q_p{}^n$)
assumes *i* ≤ *n*
shows *poly-map n* (*var-list-segment i n*) *as* = *drop i as*
apply(*rule nth-equalityI*)
apply (*metis Qp-poly-mapE' assms*(*1*) *assms*(*2*) *cartesian-power-car-memE length-drop var-list-segment-length*)
using *assms map-by-var-list-segment*[*of as n n i*] *list-segment-drop*[*of i as*] *cartesian-power-car-memE*[*of as* Q_p *n*] *map-nth*[*of*] *nth-drop nth-map*[*of - [i..<n]* (*pvar* Q_p)] *nth-map*[*of - map* (*pvar* Q_p) *[i..<n] eval-at-point* Q_p *as*]
unfolding *poly-map-def poly-tuple-eval-def var-list-segment-def restrict-def list-segment-def*
by (*smt add.commute add-eq-self-zero drop-map drop-upt le-Suc-ex le-refl*)

lemma *map-tail-by-var-list-segment*:
assumes $as \in carrier\ ({Q_p}^n)$
assumes $a \in carrier\ Q_p$
assumes $i < n$
shows *poly-map* $(n+1)$ $(var\text{-}list\text{-}segment\ 1\ (n+1))$ $(a\#as) = as$
proof−
have *0*: $(a\#as) \in carrier\ ({Q_p}^{n+1})$
using *assms*
by (*meson cartesian-power-cons*)
have *1*: *length as* $= n$
using *assms cartesian-power-car-memE*
by *blast*
have *2*: *drop 1* $(a\ \#\ as) = as$
using *0 1* **using** *list-segment-drop*[*of 1 a*#*as*]
by (*metis One-nat-def drop0 drop-Suc-Cons*)
have *1* $\leq n + 1$ **by** *auto*
then show *?thesis*
using *0 2 map-by-var-list-segment-to-length*[*of a*#*as n*+*1 1*]
by *presburger*
qed

lemma *Qp-poly-tuple-Cons*:
assumes *is-poly-tuple n fs*
assumes $f \in carrier\ (Q_p[\mathcal{X}_k])$
assumes $k \leq n$
shows *is-poly-tuple n* $(f\#fs)$
using *is-poly-tuple-Cons*[*of n fs f*] *poly-ring-car-mono*[*of k n*] *assms*
by *blast*

lemma *poly-map-Cons*:
assumes *is-poly-tuple n fs*
assumes $f \in carrier\ (Q_p[\mathcal{X}_n])$
assumes $a \in carrier\ ({Q_p}^n)$
shows *poly-map n* $(f\#fs)$ $a = (Qp\text{-}ev\ f\ a)\#poly\text{-}map\ n\ fs\ a$
using *assms poly-map-cons* **by** *blast*

lemma *poly-map-append'*:
assumes *is-poly-tuple n fs*
assumes *is-poly-tuple n gs*
assumes $a \in carrier\ ({Q_p}^n)$
shows *poly-map n* $(fs@gs)$ $a = poly\text{-}map\ n\ fs\ a$ @ *poly-map n gs a*
using *assms*(*3*) *poly-map-append* **by** *blast*

lemma *partial-pullback-by-poly*:
assumes $f \in carrier\ (Q_p[\mathcal{X}_n])$
assumes $S \subseteq carrier\ ({Q_p}^{1+k})$
shows *partial-pullback n* $(Qp\text{-}ev\ f)$ $k\ S = poly\text{-}tuple\text{-}pullback\ (n+k)\ S\ (f\#$
$(var\text{-}list\text{-}segment\ n\ (n+k)))$

proof
show *partial-pullback n (Qp-ev f) k S ⊆ poly-tuple-pullback (n+k) S (f # var-list-segment n (n + k))*
proof fix *x* **assume** *A*: *x ∈ partial-pullback n (Qp-ev f) k S*
then obtain *as bs* **where** *as-bs-def*: *as ∈ carrier (Q_p^n) ∧ bs ∈ carrier (Q_p^k) ∧ x = as @ bs*
using *partial-pullback-memE(1)[of x n (Qp-ev f) k S] cartesian-power-decomp*
by *metis*
then have *0*: *(Qp-ev f as#bs) ∈ S*
using *A partial-pullback-memE'*
by *blast*
have *1*: *Qp-ev f as = Qp-ev f (as@bs)*
using *assms as-bs-def poly-eval-cartesian-prod[of as n bs k f] Qp.cring-axioms [of]*
by *metis*
then have *2*: *((Qp-ev f x) #bs) ∈ S*
using *0 as-bs-def*
by *presburger*
have *3*: *bs = list-segment n (n+k) x*
using *as-bs-def list-segment-drop[of n x]*
by *(metis (no-types, lifting) add-cancel-right-right add-diff-cancel-left' append-eq-append-conv append-take-drop-id cartesian-power-car-memE length-0-conv length-append length-map length-upt linorder-neqE-nat list-segment-def not-add-less1)*
have *4*: *is-poly-tuple (n+k) (f # var-list-segment n (n + k))*
using *Qp-poly-tuple-Cons var-list-segment-is-poly-tuple*
by *(metis add.commute assms(1) dual-order.refl le-add2)*
have *5*: *f ∈ carrier (Q_p [𝒳_{n + k}])*
using *poly-ring-car-mono[of n n + k] assms le-add1* **by** *blast*
have *6*: *is-poly-tuple (n + k) (var-list-segment n (n + k))*
by *(simp add: var-list-segment-is-poly-tuple)*
have *7*: *x ∈ carrier (Q_p^{n + k})*
using *as-bs-def cartesian-power-concat(1)* **by** *blast*
hence *8*: *poly-map (n+k) (f # var-list-segment n (n + k)) x = (Qp-ev f x)#poly-map (n+k) (var-list-segment n (n + k)) x*
using *5 6 7 A poly-map-Cons[of n + k var-list-segment n (n + k) f x] 4*
unfolding *partial-pullback-def evimage-def*
by *blast*
hence *6*: *poly-map (n+k) (f # var-list-segment n (n + k)) x = (Qp-ev f x)#bs*
using *3 7 le-add1 le-refl map-by-var-list-segment* **by** *presburger*
show *x ∈ poly-tuple-pullback (n+k) S (f # var-list-segment n (n + k))*
unfolding *poly-tuple-pullback-def* **using** *6*
by *(metis 2 7 IntI poly-map-apply vimage-eq)*
qed
show *poly-tuple-pullback (n + k) S (f # var-list-segment n (n + k)) ⊆ partial-pullback n (Qp-ev f) k S*
proof fix *x*
assume *A*: *x ∈ poly-tuple-pullback (n + k) S (f # var-list-segment n (n + k))*

have *0*: *is-poly-tuple* (*n*+*k*) (*f* # *var-list-segment n* (*n* + *k*))
using *Qp-poly-tuple-Cons assms*(*1*) *le-add1 var-list-segment-is-poly-tuple*
by *blast*
have *1*: *x* ∈ *carrier* (*Q*$_p$^*n*+*k*)
using *A* **unfolding** *poly-tuple-pullback-def*
by *blast*
have *2*: *poly-map* (*n*+*k*) (*f* # *var-list-segment n* (*n* + *k*)) *x* ∈ *S*
using *1 assms A* **unfolding** *poly-map-def poly-tuple-pullback-def restrict-def*
by (*metis* (*no-types*, *opaque-lifting*) *Int-commute add.commute evimage-def evimage-eq*)
have *3*: *poly-map* (*n*+*k*) (*f* # *var-list-segment n* (*n* + *k*)) *x* = (*Qp-ev f x*)#(*drop n x*)
using *poly-map-Cons*[*of n* + *k var-list-segment n* (*n* + *k*) *f x*] *1 assms*(*1*) *map-by-var-list-segment-to-length*
le-add1 poly-map-cons **by** *presburger*
have *4*: *poly-map* (*n*+*k*) (*f* # *var-list-segment n* (*n* + *k*)) *x* = (*Qp-ev f* (*take n x*))#(*drop n x*)
using *assms 1 3 eval-at-points-higher-pow*[*of f n n* + *k x*] *le-add1*
by (*metis nat-le-iff-add*)
show *x* ∈ *partial-pullback n* (*Qp-ev f*) *k S*
apply(*rule partial-pullback-memI*)
using *1* **apply** *blast*
using *2 3 4* **by** *metis*
qed
qed

lemma *poly-is-semialg*:
assumes *f* ∈ *carrier* (*Q*$_p$[$\mathcal{X}_n$])
shows *is-semialg-function n* (*Qp-ev f*)
proof(*rule is-semialg-functionI*)
show *Qp-ev f* ∈ *carrier* (*Q*$_p$^*n*) → *carrier Q*$_p$
using *assms*
by (*meson Pi-I eval-at-point-closed*)
show ⋀*k S*. *S* ∈ *semialg-sets* (*1* + *k*) ⟹ *is-semialgebraic* (*n* + *k*) (*partial-pullback n* (*Qp-ev f*) *k S*)
proof− **fix** *k*::*nat* **fix** *S*
assume *A*: *S* ∈ *semialg-sets* (*1* + *k*)
have *0*: *is-poly-tuple* (*n* + *k*) (*f* # *var-list-segment n* (*n* + *k*))
by (*metis add.commute assms le-add2 order-refl Qp-poly-tuple-Cons var-list-segment-is-poly-tuple*)
have *1*: *length* (*f* # *var-list-segment n* (*n* + *k*)) = *k* + *1*
by (*metis add.commute add-diff-cancel-left' le-add1 length-Cons plus-1-eq-Suc var-list-segment-length*)
have *2*: *partial-pullback n* (*Qp-ev f*) *k S* = *poly-tuple-pullback* (*n* + *k*) *S* (*f* # *var-list-segment n* (*n* + *k*))
using *A assms partial-pullback-by-poly*[*of f n S k*]
unfolding *semialg-sets-def*
using *gen-boolean-algebra-subset*
by *blast*

then show *is-semialgebraic* $(n + k)$ (*partial-pullback* n (*Qp-ev* f) k S)
using *add.commute*[*of* 1 k] 0 1 *assms*(1)
pullback-is-semialg[*of* n+k (f # *var-list-segment* n (n + k)) k+1 S]
by (*metis A is-semialgebraicI is-semialgebraic-closed poly-tuple-pullback-eq-poly-map-vimage*)
qed
qed

Families of polynomials defined by semialgebraic coefficient functions

lemma *semialg-function-on-carrier*:
assumes *is-semialg-function* n f
assumes *restrict* f (*carrier* (${Q_p}^n$)) = *restrict* g (*carrier* (${Q_p}^n$))
shows *is-semialg-function* n g
proof(*rule is-semialg-functionI*)
have 0: $f \in$ *carrier* (${Q_p}^n$) $\to$ *carrier* Q_p
using *assms*(1) *is-semialg-function-closed*
by *blast*
show $g \in$ *carrier* (${Q_p}^n$) $\to$ *carrier* Q_p
proof fix x **assume** A: $x \in$ *carrier* (${Q_p}^n$) **then show** $g\ x \in$ *carrier* Q_p
using *assms*(2) 0
by (*metis* (*no-types*, *lifting*) *PiE restrict-Pi-cancel*)
qed
show $\bigwedge k\ S.\ S \in$ *semialg-sets* $(1 + k) \Longrightarrow$ *is-semialgebraic* $(n + k)$ (*partial-pullback* n g k S)
proof− **fix** k S
assume A: $S \in$ *semialg-sets* $(1 + k)$
have 1: *is-semialgebraic* $(n + k)$ (*partial-pullback* n f k S)
using A *assms*(1) *is-semialg-functionE is-semialgebraicI*
by *blast*
have 2: (*partial-pullback* n f k S) = (*partial-pullback* n g k S)
unfolding *partial-pullback-def partial-image-def evimage-def*
proof
show ($\lambda xs.$ f (*take* n xs) # *drop* n xs) −‘ S ∩ *carrier* (${Q_p}^{n+k}$) ⊆ ($\lambda xs.$ g (*take* n xs) # *drop* n xs) −‘ S ∩ *carrier* (${Q_p}^{n+k}$)
proof fix x **assume** $x \in$ ($\lambda xs.$ f (*take* n xs) # *drop* n xs) −‘ S ∩ *carrier* (${Q_p}^{n+k}$)
have (*take* n x) ∈ *carrier* (${Q_p}^n$)
using *assms*
by (*meson* ‹$x \in$ ($\lambda xs.$ f (*take* n xs) # *drop* n xs) −‘ S ∩ *carrier* (${Q_p}^{n+k}$)› *inf-le2 le-add1 subset-iff take-closed*)
then have f (*take* n x) = g (*take* n x)
using *assms* **unfolding** *restrict-def*
by *meson*
then show $x \in$ ($\lambda xs.$ g (*take* n xs) # *drop* n xs) −‘ S ∩ *carrier* (${Q_p}^{n+k}$)
using *assms* ‹$x \in$ ($\lambda xs.$ f (*take* n xs) # *drop* n xs) −‘ S ∩ *carrier* (${Q_p}^{n+k}$)›
by *blast*
qed
show ($\lambda xs.$ g (*take* n xs) # *drop* n xs) −‘ S ∩ *carrier* (${Q_p}^{n+k}$) ⊆ ($\lambda xs.$ f (*take* n xs) # *drop* n xs) −‘ S ∩ *carrier* (${Q_p}^{n+k}$)
proof fix x **assume** A: $x \in$ ($\lambda xs.$ g (*take* n xs) # *drop* n xs) −‘ S ∩ *carrier*

```
(Q_p^{n+k})
      have (take n x) ∈ carrier (Q_p^n)
        using assms
        by (meson A  inf-le2 le-add1 subset-iff take-closed)
      then have f (take n x) = g (take n x)
        using assms unfolding  restrict-def
        by meson
      then show x ∈ (λxs. f (take n xs) # drop n xs) -` S ∩ carrier (Q_p^{n+k})
        using A by blast
    qed
  qed
  then show is-semialgebraic (n + k) (partial-pullback n g k S)
    using 1 by auto
 qed
qed

lemma semialg-function-on-carrier':
 assumes is-semialg-function n f
 assumes ⋀a. a ∈ carrier (Q_p^n) ⟹ f a = g a
 shows is-semialg-function n g
 using assms semialg-function-on-carrier unfolding restrict-def
 by (meson restrict-ext semialg-function-on-carrier)

lemma constant-function-is-semialg:
 assumes n > 0
 assumes x ∈ carrier Q_p
 assumes ⋀ a. a ∈ carrier (Q_p^n) ⟹ f a = x
 shows is-semialg-function n f
proof(rule semialg-function-on-carrier[of - Qp-ev (Qp-to-IP x)])
 show is-semialg-function n (Qp-ev (Qp-to-IP x))
   using assms poly-is-semialg[of (Qp-to-IP x)] Qp-to-IP-car
   by blast
 have 0: ⋀ a. a ∈ carrier (Q_p^n) ⟹ f a = Qp-ev (Qp-to-IP x) a
   using eval-at-point-const assms
   by blast
 then show restrict (Qp-ev (Qp-to-IP x)) (carrier (Q_p^n)) = restrict f (carrier
(Q_p^n))
   by (metis (no-types, lifting) restrict-ext)
qed

lemma cartesian-product-singleton-factor-projection-is-semialg:
 assumes A ⊆ carrier (Q_p^m)
 assumes b ∈ carrier (Q_p^n)
 assumes is-semialgebraic (m+n) (cartesian-product A {b})
 shows is-semialgebraic m A
proof−
 obtain f where f-def: f = map (pvar Q_p) [0..<m]
   by blast
 have 0: is-poly-tuple m f
```

using *assms var-list-segment-is-poly-tuple*[*of 0 m m*]
unfolding *var-list-segment-def f-def* **by** *blast*
have *4*: *length f* = *m*
unfolding *f-def* **using** *length-map*[*of pvar* Q_p [*0*..<*m*]] **by** *auto*
obtain *g* **where** *g-def*: (*g*::(*nat multiset* ⇒ ((*nat* ⇒ *int*) × (*nat* ⇒ *int*)) *set*) *list*) = *map* (λ*i*::*nat*. *Qp.indexed-const* (*b* ! *i*)) [(*0*::*nat*)..<*n*]
by *blast*
have *1*: *is-poly-tuple m g*
proof−
have *0*: *set* [*0*::*nat*..< *n*] = {..<*n*}
using *atLeast-upt* **by** *blast*
then have ⋀*i*. *i* ∈ *set* [*0*::*nat*..< *n*] ⟹ *b*!*i* ∈ *carrier* Q_p
using *assms*(*2*) *cartesian-power-car-memE'*[*of b* Q_p *n*] **by** *blast*
hence *1*: ⋀*i*. *i* ∈ *set* [*0*::*nat*..< *n*] ⟹ *Qp.indexed-const* (*b* ! *i*) ∈ *carrier* (Q_p[$\mathcal{X}_m$])
using *assms Qp-to-IP-car* **by** *blast*
show *?thesis*
unfolding *is-poly-tuple-def g-def*
apply(*rule subsetI*)
using *set-map*[*of* λ*i*. *Qp.indexed-const* (*b* ! *i*) [*0*..<*n*]] *1* **unfolding** *0*
by (*smt image-iff*)
qed
have *2*: *is-poly-tuple m* (*f*@*g*)
using *0 1 Qp-is-poly-tuple-append assms*(*3*) **by** *blast*
have *3*: ⋀*x*. *x* ∈ *carrier* ($Q_p{}^m$) ⟹ *poly-tuple-eval* (*f*@*g*) *x* = *x*@*b*
proof− **fix** *x* **assume** *A*: *x* ∈ *carrier* ($Q_p{}^m$)
have *30*: *poly-tuple-eval f x* = *x*
proof−
have *300*: *length* (*poly-tuple-eval f x*) = *length x*
unfolding *poly-tuple-eval-def* **using** *cartesian-power-car-memE*
by (*metis 4 A length-map*)
have ⋀*i*. *i* < *length x* ⟹ *poly-tuple-eval f x* ! *i* = *x* ! *i*
unfolding *f-def poly-tuple-eval-def* **using** *nth-map*
by (*metis 4 A add-cancel-right-left cartesian-power-car-memE eval-pvar f-def length-map nth-upt*)
thus *?thesis* **using** *300*
by (*metis nth-equalityI*)
qed
have *31*: *poly-tuple-eval g x* = *b*
proof−
have *310*: *length* (*poly-tuple-eval g x*) = *length b*
unfolding *poly-tuple-eval-def g-def* **using** *cartesian-power-car-memE*
by (*metis assms*(*2*) *length-map map-nth*)
have *311*: *length b* = *n* **using** *assms cartesian-power-car-memE* **by** *blast*
hence ⋀*i*. *i* < *n* ⟹ *poly-tuple-eval g x* ! *i* = *b* ! *i* **proof**− **fix** *i* **assume** *i* < *n*
thus *poly-tuple-eval g x* ! *i* = *b* ! *i*
unfolding *g-def poly-tuple-eval-def* **using** *eval-at-point-const*[*of b*!*i x m*] *310 nth-map*

by (*metis 311 A assms(2) cartesian-power-car-memE′ length-map map-nth*)
qed
thus *?thesis* **using** *311 310 nth-equalityI*
by (*metis list-eq-iff-nth-eq*)
qed
have *32*: *poly-tuple-eval* (*f* @ *g*) *x* = *poly-map* *m* (*f*@*g*) *x*
unfolding *poly-map-def restrict-def* **using** *A*
by (*simp add*: *A*)
have *33*: *poly-tuple-eval f x* = *poly-map* *m f x*
unfolding *poly-map-def restrict-def* **using** *A*
by (*simp add*: *A*)
have *34*: *poly-tuple-eval g x* = *poly-map* *m g x*
unfolding *poly-map-def restrict-def* **using** *A*
by (*simp add*: *A*)
show *poly-tuple-eval* (*f* @ *g*) *x* = *x* @ *b*
using *assms 1 2 30 31* *poly-map-append*[*of x m f g*] *A* **unfolding** *32 33 34*
by (*simp add*: *A* ‹*b* ∈ *carrier* (*Q*$_p$ ^ *n*)›)
qed
have *4*: *A* = (*poly-tuple-eval* (*f*@*g*) ^-1_m (*cartesian-product A* {*b*}))
proof
show *A* ⊆ *poly-tuple-eval* (*f* @ *g*) ^-1_m *cartesian-product A* {*b*}
proof(*rule subsetI*) **fix** *x* **assume** *A*: *x* ∈ *A*
then have *0*: *poly-tuple-eval* (*f*@*g*) *x* = *x*@*b*
using *3 assms* **by** *blast*
then show *x* ∈ *poly-tuple-eval* (*f* @ *g*) ^-1_m *cartesian-product A* {*b*}
using *A cartesian-product-memE*
by (*smt Un-upper1 assms(1) assms(2) cartesian-product-memI′ evimageI2 in-mono insert-is-Un mk-disjoint-insert singletonI*)
qed
show *poly-tuple-eval* (*f* @ *g*) ^-1_m *cartesian-product A* {*b*} ⊆ *A*
proof(*rule subsetI*) **fix** *x* **assume** *A*: *x* ∈ (*poly-tuple-eval* (*f* @ *g*) ^-1_m *cartesian-product A* {*b*})
then have *poly-tuple-eval* (*f* @ *g*) *x* ∈ *cartesian-product A* {*b*}
by *blast*
then have *x*@*b* ∈ *cartesian-product A* {*b*}
using *A 3* **by** (*metis evimage-eq*)
then show *x* ∈ *A*
using *A*
by (*metis append-same-eq cartesian-product-memE′ singletonD*)
qed
qed
have *5*: *A* = *poly-map m* (*f*@*g*) ^-1_m (*cartesian-product A* {*b*})
proof
show *A* ⊆ *poly-map m* (*f* @ *g*) ^-1_m *cartesian-product A* {*b*}
unfolding *poly-map-def evimage-def restrict-def* **using** *4*
by (*smt IntI assms(1) evimageD in-mono subsetI vimageI*)
show *poly-map m* (*f* @ *g*) ^-1_m *cartesian-product A* {*b*} ⊆ *A*
unfolding *poly-map-def evimage-def restrict-def* **using** *4*
by (*smt Int-iff evimageI2 subsetI vimage-eq*)

qed
have *6*: *length* (*f* @ *g*) = *m* + *n*
unfolding *f-def g-def* **by** (*metis index-list-length length-append length-map map-nth*)
show *?thesis* **using** *2 5 6 assms pullback-is-semialg*[*of m f*@*g m*+*n cartesian-product A* {*b*}]
by (*metis is-semialgebraicE zero-eq-add-iff-both-eq-0*)
qed

lemma *cartesian-product-factor-projection-is-semialg*:
assumes *A* ⊆ *carrier* (*Q*$_p$^*m*)
assumes *B* ⊆ *carrier* (*Q*$_p$^*n*)
assumes *B* ≠ {}
assumes *is-semialgebraic* (*m*+*n*) (*cartesian-product A B*)
shows *is-semialgebraic m A*
proof−
obtain *b* **where** *b-def*: *b* ∈ *B*
using *assms* **by** *blast*
have *is-semialgebraic n* {*b*}
using *assms b-def is-algebraic-imp-is-semialg singleton-is-algebraic* **by** *blast*
hence *0*: *is-semialgebraic* (*m*+*n*) (*cartesian-product* (*carrier* (*Q*$_p$^*m*)) {*b*})
using *car-times-semialg-is-semialg assms*(*4*) **by** *blast*
have (*cartesian-product* (*carrier* (*Q*$_p$^*m*)) {*b*}) ∩ (*cartesian-product A B*)
= (*cartesian-product A* {*b*})
using *assms b-def cartesian-product-intersection*[*of carrier* (*Q*$_p$^*m*) *Q*$_p$ *m* {*b*} *n A B*]
by (*metis* (*no-types*, *lifting*) *Int-absorb1 Int-empty-left Int-insert-left-if1* ‹*is-semialgebraic n* {*b*}› *is-semialgebraic-closed set-eq-subset*)
hence *is-semialgebraic* (*m*+*n*) (*cartesian-product A* {*b*})
using *assms 0 intersection-is-semialg* **by** *metis*
thus *?thesis* **using** *assms cartesian-product-singleton-factor-projection-is-semialg*
by (*meson* ‹*is-semialgebraic n* {*b*}› *insert-subset is-semialgebraic-closed*)
qed

lemma *partial-pullback-cartesian-product*:
assumes ξ ∈ *carrier* (*Q*$_p$^*m*) → *carrier Q*$_p$
assumes *S* ⊆ *carrier* (*Q*$_p$^1)
shows *cartesian-product* (*partial-pullback m* ξ *0 S*) (*carrier* (*Q*$_p$^1)) = *partial-pullback m* ξ *1* (*cartesian-product S* (*carrier* (*Q*$_p$^1)))
proof
show *cartesian-product* (*partial-pullback m* ξ *0 S*) (*carrier* (*Q*$_p$^1)) ⊆ *partial-pullback m* ξ *1* (*cartesian-product S* (*carrier* (*Q*$_p$^1)))
proof fix *x* **assume** *A*: *x* ∈ *cartesian-product* (*partial-pullback m* ξ *0 S*) (*carrier* (*Q*$_p$^1))
then obtain *y t* **where** *yt-def*: *x* = *y*@[*t*] ∧ *y* ∈ *partial-pullback m* ξ *0 S* ∧ *t* ∈ *carrier Q*$_p$
by (*metis cartesian-product-memE*′ *Qp.to-R1-to-R Qp.to-R-pow-closed*)
then have [ξ *y*] ∈ *S*
using *partial-pullback-memE* **unfolding** *partial-image-def*

by (*metis* (*no-types*, *lifting*) *add.right-neutral append.right-neutral cartesian-power-drop le-zero-eq take-closed partial-pullback-memE' take-eq-Nil*)
then have *0*: $[\xi\ y]@[t] \in$ *cartesian-product S* (*carrier* ($Q_p{}^1$))
using *cartesian-product-memI' yt-def*
by (*metis assms*(*2*) *carrier-is-semialgebraic is-semialgebraic-closed Qp.to-R1-closed*)
have *1*: $x \in$ *carrier* ($Q_p{}^{m+1}$)
using *A yt-def*
by (*metis add.right-neutral cartesian-power-append partial-pullback-memE*(*1*))
show $x \in$ *partial-pullback m* ξ *1* (*cartesian-product S* (*carrier* ($Q_p{}^1$)))
apply(*rule partial-pullback-memI*)
using *1* **apply** *blast*
using *yt-def 0*
by (*smt Cons-eq-appendI add.right-neutral local.partial-image-def partial-image-eq partial-pullback-memE*(*1*) *self-append-conv2 Qp.to-R1-closed*)
qed
show *partial-pullback m* ξ *1* (*cartesian-product S* (*carrier* ($Q_p{}^1$))) $\subseteq$ *cartesian-product* (*partial-pullback m* ξ *0 S*) (*carrier* ($Q_p{}^1$))
proof(*rule subsetI*) **fix** *x* **assume** *A*: $x \in$ *partial-pullback m* ξ *1* (*cartesian-product S* (*carrier* ($Q_p{}^1$)))
then have *0*: $x \in$ *carrier* ($Q_p{}^{m+1}$)
using *assms partial-pullback-memE*[*of x m* ξ *1 cartesian-product S* (*carrier* ($Q_p{}^1$))]
by *blast*
have *1*: ξ (*take m x*) # *drop m x* $\in$ *cartesian-product S* (*carrier* ($Q_p{}^1$))
using *A assms partial-pullback-memE*[*of x m* ξ *1 cartesian-product S* (*carrier* ($Q_p{}^1$))]
unfolding *partial-image-def*
by *blast*
have *2*: ξ (*take m* (*take m x*)) # *drop m* (*take m x*) = [ξ (*take m x*)]
using *0 1*
by (*metis add.commute add.right-neutral append.right-neutral append-take-drop-id take0 take-drop*)
show $x \in$ *cartesian-product* (*partial-pullback m* ξ *0 S*) (*carrier* ($Q_p{}^1$))
apply(*rule cartesian-product-memI*[*of - Q_p m - 1*])
apply (*metis add-cancel-right-right partial-pullback-closed*)
apply *blast*
apply(*rule partial-pullback-memI*[*of - m 0* ξ *S*]) **using** *0*
apply (*metis Nat.add-0-right le-iff-add take-closed*)
using *2* **apply** (*metis* (*no-types*, *lifting*) *1 add.commute add.right-neutral assms*(*2*) *cartesian-product-memE*(*1*) *list.inject plus-1-eq-Suc take-Suc-Cons take-drop*)
using *0 cartesian-power-drop* **by** *blast*
qed
qed

lemma *cartesian-product-swap*:
assumes $A \subseteq$ *carrier* ($Q_p{}^n$)
assumes $B \subseteq$ *carrier* ($Q_p{}^m$)
assumes *is-semialgebraic* (*m*+*n*) (*cartesian-product A B*)
shows *is-semialgebraic* (*m*+*n*) (*cartesian-product B A*)

proof−
obtain *f* **where** *f-def*: $f = (\lambda i.\ (\mathit{if}\ i < m\ \mathit{then}\ n + i\ \mathit{else}\ (\mathit{if}\ i < m{+}n\ \mathit{then}\ i - m\ \mathit{else}\ i)))$
by *blast*
have *0*: $\bigwedge i.\ i \in \{..{<}m\} \longrightarrow f\ i \in \{n..{<}m{+}n\}$
unfolding *f-def* **by** *simp*
have *1*: $\bigwedge i.\ i \in \{m..{<}m{+}n\} \longrightarrow f\ i \in \{..{<}n\}$
unfolding *f-def* **by** (*simp add*: *less-diff-conv2*)
have *2*: $\bigwedge i.\ i \notin \{..{<}m + n\} \longrightarrow f\ i \notin \{..{<}m + n\}$
unfolding *f-def* **by** *simp*
have *f-permutes*: *f permutes* $\{..{<}m{+}n\}$
unfolding *permutes-def*
proof
show $\forall x.\ x \notin \{..{<}m + n\} \longrightarrow f\ x = x$
unfolding *f-def* **by** *simp*
show $\forall y.\ \exists!x.\ f\ x = y$
proof fix *y*
show $\exists!x.\ f\ x = y$
proof(*cases* $y < n$)
case *True*
have *T0*: $f\ (y{+}m) = y$
unfolding *f-def* **using** *True*
by *simp*
have $\bigwedge i.\ f\ i = y \Longrightarrow i \in \{m..{<}m{+}n\}$
using *0 1 2 True f-def nat-neq-iff* **by** *fastforce*
hence $\bigwedge i.\ f\ i = y \Longrightarrow i = y{+}m$
using *T0* **unfolding** *f-def* **by** *auto*
thus *?thesis* **using** *T0* **by** *blast*
next
case *False*
show *?thesis*
proof(*cases* $y \in \{n..{<}m{+}n\}$)
case *True*
have *T0*: $f\ (y{-}n) = y$
using *True* **unfolding** *f-def* **by** *auto*
have $\bigwedge i.\ f\ i = y \Longrightarrow i \in \{..{<}m\}$
using *0 1 2 True f-def*
by (*metis False atLeastLessThan-iff diff-add-inverse2 diff-diff-cancel diff-le-self*
lessThan-iff less-imp-diff-less linordered-semidom-class.add-diff-inverse nat-neq-iff not-add-less1)
hence $\bigwedge i.\ f\ i = y \Longrightarrow i = y{-}\ n$
using *f-def* **by** *force*
then show *?thesis* **using** *T0* **by** *blast*
next
case *F*: *False*
then show *?thesis* **using** *0 1 2* **unfolding** *f-def*
using *False add-diff-inverse-nat lessThan-iff* **by** *auto*
qed

qed
qed
qed
have *permute-list f ‘ (cartesian-product A B) = (cartesian-product B A)*
proof
show *permute-list f ‘ cartesian-product A B ⊆ cartesian-product B A*
proof fix *x* **assume** *A*: *x ∈ permute-list f ‘ cartesian-product A B*
then obtain *a b* **where** *ab-def*: *a ∈ A ∧b ∈ B ∧ x = permute-list f (a@b)*
by (*metis* (*mono-tags*, *lifting*) *cartesian-product-memE′ image-iff*)
have *0*: *x = permute-list f (a@b)*
using *ab-def* **by** *blast*
have *1*: *length a = n*
using *ab-def assms cartesian-power-car-memE*[*of a* Q_p *n*] **by** *blast*
have *2*: *length b = m*
using *ab-def assms cartesian-power-car-memE*[*of b* Q_p *m*] **by** *blast*
have *3*: *length x = m + n*
using *1 2 0 f-permutes* **by** *simp*
have *4*: *⋀i. i < m ⟹ x ! i = (a@b) ! (f i)*
unfolding *0* **using** *permute-list-nth*
by (*metis 0 3 f-permutes length-permute-list trans-less-add1*)
hence *5*: *⋀i. i < m ⟹ x ! i = b!i*
unfolding *f-def* **using** *1 2*
by (*metis 4 f-def nth-append-length-plus*)
have *6*: *⋀i. i ∈ {m..<m+n} ⟹ x ! i = (a@b) ! (i − m)*
unfolding *0* **using** *f-def permute-list-nth f-permutes*
by (*metis* (*no-types*, *lifting*) *0 3 atLeastLessThan-iff length-permute-list not-add-less2*
ordered-cancel-comm-monoid-diff-class.diff-add)
have *7*: *x = b@a*
proof(*rule nth-equalityI*)
show *length x = length (b @ a)*
using *1 2 3* **by** *simp*
show *⋀i. i < length x ⟹ x ! i = (b @ a) ! i*
unfolding *3* **using** *1 2 4 5*
by (*smt 0 add.commute add-diff-inverse-nat f-def f-permutes length-append nat-add-left-cancel-less nth-append permute-list-nth*)
qed
show *x ∈ cartesian-product B A* **unfolding** *7* **using** *ab-def* **unfolding** *cartesian-product-def* **by** *blast*
qed
show *cartesian-product B A ⊆ permute-list f ‘ cartesian-product A B*
proof fix *y* **assume** *A*: *y ∈ cartesian-product B A*
then obtain *b a* **where** *ab-def*: *b ∈ B ∧ a ∈ A ∧ y = b@a*
using *cartesian-product-memE′* **by** *blast*
obtain *x* **where** *0*: *x = permute-list f (a@b)*
by *blast*
have *1*: *length a = n*
using *ab-def assms cartesian-power-car-memE*[*of a* Q_p *n*] **by** *blast*
have *2*: *length b = m*

using *ab-def assms cartesian-power-car-memE*[*of b* Q_p *m*] **by** *blast*
have *3*: *length x* = *m* + *n*
using *1 2 0 f-permutes* **by** *simp*
have *4*: $\bigwedge i.\ i < m \Longrightarrow x\ !\ i = (a@b)\ !\ (f\ i)$
unfolding *0* **using** *permute-list-nth*
by (*metis 0 3 f-permutes length-permute-list trans-less-add1*)
hence *5*: $\bigwedge i.\ i < m \Longrightarrow x\ !\ i = b!i$
unfolding *f-def* **using** *1 2*
by (*metis 4 f-def nth-append-length-plus*)
have *6*: $\bigwedge i.\ i \in \{m..<m+n\} \Longrightarrow x\ !\ i = (a@b)\ !\ (i - m)$
unfolding *0* **using** *f-def permute-list-nth f-permutes*
by (*metis* (*no-types*, *lifting*) *0 3 atLeastLessThan-iff length-permute-list not-add-less2*
ordered-cancel-comm-monoid-diff-class.diff-add)
have *7*: *x* = *b*@*a*
proof(*rule nth-equalityI*)
show *length x* = *length* (*b* @ *a*)
using *1 2 3* **by** *simp*
show $\bigwedge i.\ i <$ *length x* $\Longrightarrow x\ !\ i = (b\ @\ a)\ !\ i$
unfolding *3* **using** *1 2 4 5*
by (*smt 0 add.commute add-diff-inverse-nat f-def f-permutes length-append nat-add-left-cancel-less nth-append permute-list-nth*)
qed
show *y* ∈ *permute-list f* ‘ *cartesian-product A B*
using *ab-def 7 cartesian-product-memI*'[*of - Q*$_p$] **unfolding** *0*
by (*metis assms*(*1*) *assms*(*2*) *image-eqI*)
qed
qed
thus *?thesis* **using** *assms f-permutes permutation-is-semialgebraic*
by *metis*
qed

lemma *Qp-zero-subset-is-semialg*:
assumes *S* ⊆ *carrier* ($Q_p{}^0$)
shows *is-semialgebraic 0 S*
proof(*cases S* = {})
case *True*
then show *?thesis*
by (*simp add*: *empty-is-semialgebraic*)
next
case *False*
then have *S* = *carrier* ($Q_p{}^0$)
using *assms* **unfolding** *Qp-zero-carrier* **by** *blast*
then show *?thesis*
by (*simp add*: *carrier-is-semialgebraic*)
qed

lemma *cartesian-product-empty-list*:
cartesian-product A {[]} = *A*

cartesian-product {[]} *A* = *A*
proof
  show *cartesian-product A* {[]} ⊆ *A*
    apply(*rule subsetI*)
    unfolding *cartesian-product-def*
    by (*smt append-Nil2 empty-iff insert-iff mem-Collect-eq*)
  show *A* ⊆ *cartesian-product A* {[]}
    apply(*rule subsetI*)
    unfolding *cartesian-product-def*
    by (*smt append-Nil2 empty-iff insert-iff mem-Collect-eq*)
  show *cartesian-product* {[]} *A* = *A*
  proof
    show *cartesian-product* {[]} *A* ⊆ *A*
    apply(*rule subsetI*)
    unfolding *cartesian-product-def*
    by (*smt append-self-conv2 bex-empty insert-compr mem-Collect-eq*)
  show *A* ⊆ *cartesian-product* {[]} *A*
    apply(*rule subsetI*)
    unfolding *cartesian-product-def*
    by *blast*
  qed
qed

lemma *cartesian-product-singleton-factor-projection-is-semialg'*:
  assumes *A* ⊆ *carrier* (${Q_p}^m$)
  assumes *b* ∈ *carrier* (${Q_p}^n$)
  assumes *is-semialgebraic* (*m*+*n*) (*cartesian-product A* {*b*})
  shows *is-semialgebraic m A*
proof(*cases n* > *0*)
  case *True*
  show *?thesis*
  proof(*cases m* > *0*)
    case *T*: *True*
    then show *?thesis*
      using *assms True cartesian-product-singleton-factor-projection-is-semialg* **by** *blast*
  next
    case *False*
    then show *?thesis* **using** *Qp-zero-subset-is-semialg assms* **by** *blast*
  qed
next
  case *False*
  then have *F0*: *b* = []
    using *assms Qp-zero-carrier* **by** *blast*
  have *cartesian-product A* {*b*} = *A*
    unfolding *F0*
    by (*simp add*: *cartesian-product-empty-list*(*1*))
  then show *?thesis* **using** *assms False*
    by (*metis add.right-neutral gr0I*)

qed

13.11 More on graphs of functions

This section lays the groundwork for showing that semialgebraic functions are closed under various algebraic operations

The take and drop functions on lists are polynomial maps

lemma *function-restriction*:
assumes $g \in carrier\ ({Q_p}^n) \to S$
assumes $n \leq k$
shows $(g \circ (take\ n)) \in carrier\ ({Q_p}^k) \to S$
proof fix x
assume $x \in carrier\ ({Q_p}^k)$
then have *take n x* $\in carrier\ ({Q_p}^n)$
using *assms*(*2*) *take-closed*
by *blast*
then show $(g \circ take\ n)\ x \in S$
using *assms comp-apply*
by (*metis Pi-iff comp-def*)
qed

lemma *partial-pullback-restriction*:
assumes $g \in carrier\ ({Q_p}^n) \to carrier\ Q_p$
assumes $n < k$
shows *partial-pullback k* $(g \circ take\ n)$ *m S* =
split-cartesian-product $(n + m)\ (k - n)\ n$ (*partial-pullback n g m S*) (*carrier* $({Q_p}^{k-n})$)
proof(*rule equalityI*)
show *partial-pullback k* $(g \circ take\ n)$ *m S* $\subseteq$ *split-cartesian-product* $(n + m)\ (k - n)\ n$ (*partial-pullback n g m S*) (*carrier* $({Q_p}^{k-n})$)
proof fix *x* **assume** *A*: $x \in$ *partial-pullback k* $(g \circ take\ n)$ *m S*
obtain *as bs* **where** *asbs-def*: $x = as@bs \wedge as \in carrier\ ({Q_p}^k) \wedge bs \in carrier\ ({Q_p}^m)$
using *partial-pullback-memE*[*of x k g* $\circ$ *take n m S*] *A cartesian-power-decomp*[*of x Q*$_p$ *k m*]
by *metis*
have *0*: $((g \circ (take\ n))\ as)\#bs \in S$
using *asbs-def partial-pullback-memE'*[*of as k bs m x*] *A*
by *blast*
have *1*: $(g\ (take\ n\ as))\#bs \in S$
using *0*
by (*metis comp-apply*)
have *2*: *take n as* @ *bs* $\in carrier\ ({Q_p}^{n+m})$
by (*meson asbs-def assms*(*2*) *cartesian-power-concat*(*1*) *less-imp-le-nat take-closed*)
have *3*: $(take\ n\ as)@bs \in$ (*partial-pullback n g m S*)
using *1 2 partial-pullback-memI*[*of* $(take\ n\ as)@bs$ *n m g S*]
by (*metis* (*mono-tags*, *opaque-lifting*) *asbs-def assms*(*2*) *local.partial-image-def nat-less-le*

partial-image-eq subsetD subset-refl take-closed)
have *4*: *drop n as* ∈ (*carrier* ($Q_p{}^{k-n}$))
using *asbs-def assms*(*2*) *drop-closed*
by *blast*
show *x* ∈ *split-cartesian-product* (*n* + *m*) (*k* − *n*) *n* (*partial-pullback n g m S*) (*carrier* ($Q_p{}^{k-n}$))
using *split-cartesian-product-memI*[*of take n as bs*
partial-pullback n g m S drop n as
carrier ($Q_p{}^{k-n}$) Q_p *n* + *m k* − *n n*] *4*
by (*metis* (*no-types*, *lifting*) *3 append.assoc append-take-drop-id*
asbs-def assms(*2*) *cartesian-power-car-memE less-imp-le-nat partial-pullback-memE*(*1*)
subsetI take-closed)
qed
show *split-cartesian-product* (*n* + *m*) (*k* − *n*) *n* (*partial-pullback n g m S*) (*carrier* ($Q_p{}^{k-n}$)) ⊆ *partial-pullback k* (*g* ∘ *take n*) *m S*
proof fix *x* **assume** *A*: *x* ∈ *split-cartesian-product* (*n* + *m*) (*k* − *n*) *n* (*partial-pullback n g m S*) (*carrier* ($Q_p{}^{k-n}$))
show *x* ∈ *partial-pullback k* (*g* ∘ *take n*) *m S*
proof(*rule partial-pullback-memI*)
have *0*: (*partial-pullback n g m S*) ⊆ *carrier* ($Q_p{}^{n+m}$)
using *partial-pullback-closed* **by** *blast*
then have *split-cartesian-product* (*n* + *m*) (*k* − *n*) *n* (*partial-pullback n g m S*) (*carrier* ($Q_p{}^{k-n}$)) ⊆ *carrier* ($Q_p{}^{n+m+(k-n)}$)
using *assms A split-cartesian-product-closed*[*of partial-pullback n g m S* Q_p *n* + *m*
carrier ($Q_p{}^{k-n}$) *k* − *n n*]
using *le-add1* **by** *blast*
then show *P*: *x* ∈ *carrier* ($Q_p{}^{k+m}$)
by (*smt A Nat.add-diff-assoc2 add.commute add-diff-cancel-left*′ *assms*(*2*) *le-add1 less-imp-le-nat subsetD*)
have *take n x* @ *drop* (*n* + (*k* − *n*)) *x* ∈ *partial-pullback n g m S*
using *0 A split-cartesian-product-memE*[*of x n* + *m k* − *n n partial-pullback n g m S carrier* ($Q_p{}^{k-n}$) Q_p]
le-add1 **by** *blast*
have *1*: *g* (*take n x*) # *drop k x* ∈ *S*
using *partial-pullback-memE*
by (*metis* (*no-types*, *lifting*) ‹*take n x* @ *drop* (*n* + (*k* − *n*)) *x* ∈ *partial-pullback n g m S*›
‹*x* ∈ *carrier* ($Q_p{}^{k+m}$)› *add.assoc assms*(*2*) *cartesian-power-drop le-add1*
le-add-diff-inverse less-imp-le-nat partial-pullback-memE′ *take-closed*)
have *2*: *g* (*take n x*) = (*g* ∘ *take n*) (*take k x*)
using *assms P comp-apply*[*of g take n take k x*]
by (*metis add.commute append-same-eq append-take-drop-id less-imp-add-positive take-add take-drop*)
then show (*g* ∘ *take n*) (*take k x*) # *drop k x* ∈ *S*
using *1* **by** *presburger*
qed
qed
qed

lemma *comp-take-is-semialg*:
assumes *is-semialg-function n g*
assumes $n < k$
assumes $0 < n$
shows *is-semialg-function k* $(g \circ (take\ n))$
proof(*rule is-semialg-functionI*)
show $g \circ take\ n \in carrier\ ({Q_p}^k) \to carrier\ Q_p$
using *assms function-restriction*[*of g n carrier* Q_p *k*] *dual-order.strict-implies-order is-semialg-function-closed*
by *blast*
show $\bigwedge ka\ S.\ S \in semialg\text{-}sets\ (1 + ka) \Longrightarrow is\text{-}semialgebraic\ (k + ka)\ (partial\text{-}pullback\ k\ (g \circ take\ n)\ ka\ S)$
proof− **fix** *l S* **assume** *A*: $S \in semialg\text{-}sets\ (1 + l)$
have *0*: *is-semialgebraic* $(n + l)$ $(partial\text{-}pullback\ n\ g\ l\ S)$
using *assms A is-semialg-functionE is-semialgebraicI*
by *blast*
have *is-semialgebraic* $(n + l + (k - n))$ $(split\text{-}cartesian\text{-}product\ (n + l)\ (k - n)\ n\ (partial\text{-}pullback\ n\ g\ l\ S)\ (carrier\ ({Q_p}^{k - n})))$
using *A 0 split-cartesian-product-is-semialgebraic*[*of - - partial-pullback n g l S - carrier* $({Q_p}^{k - n})$] *add-gr-0 assms*(*2*) *assms*(*3*) *carrier-is-semialgebraic le-add1 zero-less-diff*
by *presburger*
then show *is-semialgebraic* $(k + l)$ $(partial\text{-}pullback\ k\ (g \circ take\ n)\ l\ S)$
using *partial-pullback-restriction*[*of g n k l S*]
by (*metis* (*no-types*, *lifting*) *add.assoc add.commute assms*(*1*) *assms*(*2*) *is-semialg-function-closed le-add-diff-inverse less-imp-le-nat*)
qed
qed

Restriction of a graph to a semialgebraic domain

lemma *graph-formula*:
assumes $g \in carrier\ ({Q_p}^n) \to carrier\ Q_p$
shows *graph n g* $= \{as \in carrier\ ({Q_p}^{Suc\ n}).\ g\ (take\ n\ as) = as!n\}$
using *assms graph-memI fun-graph-def*[*of* Q_p *n g*]
by (*smt Collect-cong Suc-eq-plus1 graph-memE*(*1*) *graph-mem-closed mem-Collect-eq*)

definition *restricted-graph* **where**
restricted-graph n g S $= \{as \in carrier\ ({Q_p}^{Suc\ n}).\ take\ n\ as \in S \wedge g\ (take\ n\ as) = as!n\ \}$

lemma *restricted-graph-closed*:
restricted-graph n g S $\subseteq carrier\ ({Q_p}^{Suc\ n})$
by (*metis* (*no-types*, *lifting*) *mem-Collect-eq restricted-graph-def subsetI*)

lemma *restricted-graph-memE*:
assumes $a \in restricted\text{-}graph\ n\ g\ S$
shows $a \in carrier\ ({Q_p}^{Suc\ n})\ take\ n\ a \in S\ g\ (take\ n\ a) = a!n$
using *assms*

using *restricted-graph-closed* **apply** *blast*
apply (*metis* (*no-types*, *lifting*) *assms mem-Collect-eq restricted-graph-def*)
using *assms* **unfolding** *restricted-graph-def*
by *blast*

lemma *restricted-graph-mem-formula*:
assumes $a \in$ *restricted-graph n g S*
shows $a =$ (*take n a*)@[*g* (*take n a*)]
proof−
have *length a* = *Suc n*
using *assms*
by (*metis* (*no-types*, *lifting*) *cartesian-power-car-memE mem-Collect-eq restricted-graph-def*)
then have $a =$ (*take n a*)@[*a*!*n*]
by (*metis append-eq-append-conv-if hd-drop-conv-nth lessI take-hd-drop*)
then show *?thesis*
by (*metis assms restricted-graph-memE*(*3*))
qed

lemma *restricted-graph-memI*:
assumes $a \in$ *carrier* $({Q_p}^{Suc\ n})$
assumes *take n a* $\in S$
assumes *g* (*take n a*) = *a*!*n*
shows $a \in$ *restricted-graph n g S*
using *assms restricted-graph-def*
by *blast*

lemma *restricted-graph-memI′*:
assumes $a \in S$
assumes $g \in$ *carrier* $({Q_p}^{n}) \to$ *carrier* Q_p
assumes $S \subseteq$ *carrier* $({Q_p}^{n})$
shows (*a*@[*g a*]) $\in$ *restricted-graph n g S*
proof−
have $a \in$ *carrier* $({Q_p}^{n})$
using *assms*(*1*) *assms*(*3*) **by** *blast*
then have *g a* $\in$ *carrier* Q_p
using *assms* **by** *blast*
then have *0*: *a* @ [*g a*] $\in$ *carrier* $({Q_p}^{Suc\ n})$
using *assms*
by (*metis* (*no-types*, *lifting*) *add.commute cartesian-power-append plus-1-eq-Suc subsetD*)
have *1*: *take n* (*a* @ [*g a*]) $\in S$
using *assms*
by (*metis* (*no-types*, *lifting*) *append-eq-conv-conj cartesian-power-car-memE subsetD*)
show *?thesis*
using *assms restricted-graph-memI*[*of a*@[*g a*] *n S g*]
by (*metis 0* ‹*a* $\in$ *carrier* $({Q_p}^{n})$› *append-eq-conv-conj cartesian-power-car-memE nth-append-length*)

qed

lemma *restricted-graph-subset*:
assumes $g \in carrier\ ({Q_p}^n) \to carrier\ Q_p$
assumes $S \subseteq carrier\ ({Q_p}^n)$
shows *restricted-graph n g S* $\subseteq$ *graph n g*
proof fix x **assume** A: $x \in$ *restricted-graph n g S*
show $x \in$ *graph n g*
apply(*rule graph-memI*)
using *assms*(*1*) **apply** *blast*
using A *restricted-graph-memE*(*3*) **apply** *blast*
by (*metis A add.commute plus-1-eq-Suc restricted-graph-memE*(*1*))
qed

lemma *restricted-graph-subset'*:
assumes $g \in carrier\ ({Q_p}^n) \to carrier\ Q_p$
assumes $S \subseteq carrier\ ({Q_p}^n)$
shows *restricted-graph n g S* $\subseteq$ *cartesian-product S* (*carrier* $({Q_p}^1)$)
proof fix a **assume** A: $a \in$ *restricted-graph n g S*
then have $a = (take\ n\ a)@[g\ (take\ n\ a)]$
using *restricted-graph-mem-formula* **by** *blast*
then show $a \in$ *cartesian-product S* (*carrier* $({Q_p}^1)$)
using *cartesian-product-memI' A* **unfolding** *restricted-graph-def*
by (*metis* (*mono-tags*, *lifting*) *assms*(*2*) *last-closed' mem-Collect-eq subsetI Qp.to-R1-closed*)
qed

lemma *restricted-graph-intersection*:
assumes $g \in carrier\ ({Q_p}^n) \to carrier\ Q_p$
assumes $S \subseteq carrier\ ({Q_p}^n)$
shows *restricted-graph n g S* = *graph n g* $\cap$ (*cartesian-product S* (*carrier* $({Q_p}^1)$))
proof
show *restricted-graph n g S* $\subseteq$ *graph n g* $\cap$ *cartesian-product S* (*carrier* $({Q_p}^1)$)
using *assms restricted-graph-subset restricted-graph-subset'*
by (*meson Int-subset-iff*)
show *graph n g* $\cap$ *cartesian-product S* (*carrier* $({Q_p}^1)$) $\subseteq$ *restricted-graph n g S*
proof fix x **assume** A: $x \in$ *graph n g* $\cap$ *cartesian-product S* (*carrier* $({Q_p}^1)$)
show $x \in$ *restricted-graph n g S*
apply(*rule restricted-graph-memI*)
using A *graph-memE*[*of g n x*]
apply (*metis* (*no-types*, *lifting*) *Int-iff add.commute assms*(*1*) *graph-mem-closed plus-1-eq-Suc*)
using A *graph-memE*[*of g n x*] *cartesian-product-memE*[*of x S carrier* $({Q_p}^1)$ Q_p *n*]
using *assms*(*2*) **apply** *blast*
using A *graph-memE*[*of g n x*] *cartesian-product-memE*[*of x S carrier* $({Q_p}^1)$ Q_p *n*]
using *assms*(*1*) **by** *blast*
qed

qed

lemma *restricted-graph-is-semialgebraic*:
assumes *is-semialg-function n g*
assumes *is-semialgebraic n S*
shows *is-semialgebraic* (*n+1*) (*restricted-graph n g S*)
proof−
have *0*: *restricted-graph n g S* = *graph n g* ∩ (*cartesian-product S* (*carrier* ($Q_p{}^1$)))
using *assms is-semialg-function-closed is-semialgebraic-closed*
restricted-graph-intersection **by** *presburger*
have *1*: *is-semialgebraic* (*n* + *1*) (*graph n g*)
using *assms semialg-graph*
by *blast*
have *2*: *is-semialgebraic* (*n* + *1*) (*cartesian-product S* (*carrier* ($Q_p{}^1$)))
using *cartesian-product-is-semialgebraic*[*of n S 1 carrier* ($Q_p{}^1$)] *assms*
carrier-is-semialgebraic less-one
by *presburger*
then show *?thesis*
using *0 1 2 intersection-is-semialg*[*of n+1 graph n g cartesian-product S* (*carrier* ($Q_p{}^1$))]
by *presburger*
qed

lemma *take-closed*:
assumes *n* ≤ *k*
assumes *x* ∈ *carrier* ($Q_p{}^k$)
shows *take n x* ∈ *carrier* ($Q_p{}^n$)
using *assms take-closed*
by *blast*

lemma *take-compose-closed*:
assumes *g* ∈ *carrier* ($Q_p{}^n$) → *carrier* Q_p
assumes *n* < *k*
shows *g* ∘ *take n* ∈ *carrier* ($Q_p{}^k$) → *carrier* Q_p
proof fix *x* **assume** *A*: *x* ∈ *carrier* ($Q_p{}^k$)
then have (*take n x*) ∈ *carrier* ($Q_p{}^n$)
using *assms less-imp-le-nat take-closed*
by *blast*
then have *g* (*take n x*) ∈ *carrier* Q_p
using *assms*(*1*) **by** *blast*
then show (*g* ∘ *take n*) *x* ∈ *carrier* Q_p
using *comp-apply*[*of g take n x*]
by *presburger*
qed

lemma *take-graph-formula*:
assumes *g* ∈ *carrier* ($Q_p{}^n$) → *carrier* Q_p
assumes *n* < *k*
assumes *0* < *n*

shows *graph k* $(g \circ (take\ n)) = \{as \in carrier\ ({Q_p}^{k+1}).\ g\ (take\ n\ as) = as!k\}$
proof−
have $\bigwedge as.\ as \in carrier\ ({Q_p}^{k+1}) \Longrightarrow (g \circ take\ n)\ (take\ k\ as) = g\ (take\ n\ as)$
using *assms comp-apply take-take*[*of n k*]
proof −
fix *as* :: $((nat \Rightarrow int) \times (nat \Rightarrow int))\ set\ list$
show $(g \circ take\ n)\ (take\ k\ as) = g\ (take\ n\ as)$
by (*metis* (*no-types*) ‹$n < k$› *comp-eq-dest-lhs min.strict-order-iff take-take*)
qed
then show *?thesis*
using *take-compose-closed*[*of g n k*] *assms comp-apply*[*of g take n*] *graph-formula*[*of* $g \circ (take\ n)$ *k*]
by (*smt Collect-cong Suc-eq-plus1*)
qed

lemma *graph-memI′*:
assumes $a \in carrier\ ({Q_p}^{Suc\ n})$
assumes $take\ n\ a \in carrier\ ({Q_p}^{n})$
assumes $g\ (take\ n\ a) = a!n$
shows $a \in graph\ n\ g$
using *assms fun-graph-def*[*of* Q_p *n g*]
by (*smt cartesian-power-car-memE eq-imp-le lessI mem-Collect-eq take-Suc-conv-app-nth take-all*)

lemma *graph-memI″*:
assumes $a \in carrier\ ({Q_p}^{n})$
assumes $g \in carrier\ ({Q_p}^{n}) \to carrier\ Q_p$
shows $(a@[g\ a]) \in graph\ n\ g$
using *assms fun-graph-def*
by *blast*

lemma *graph-as-restricted-graph*:
assumes $f \in carrier\ ({Q_p}^{n}) \to carrier\ Q_p$
shows $graph\ n\ f = restricted\text{-}graph\ n\ f\ (carrier\ ({Q_p}^{n}))$
apply(*rule equalityI*)
apply (*metis Suc-eq-plus1 assms graph-memE*(*1*) *graph-memE*(*3*) *graph-mem-closed restricted-graph-memI subsetI*)
by (*simp add*: *assms restricted-graph-subset*)

definition *double-graph* **where**
$double\text{-}graph\ n\ f\ g = \{as \in carrier\ ({Q_p}^{n+2}).\ f\ (take\ n\ as) = as!n \wedge g\ (take\ n\ as) = as!(n + 1)\}$

lemma *double-graph-rep*:
assumes $g \in carrier\ ({Q_p}^{n}) \to carrier\ Q_p$
assumes $f \in carrier\ ({Q_p}^{n}) \to carrier\ Q_p$
shows $double\text{-}graph\ n\ f\ g = restricted\text{-}graph\ (n + 1)\ (g \circ take\ n)\ (graph\ n\ f)$
proof
show $double\text{-}graph\ n\ f\ g \subseteq restricted\text{-}graph\ (n + 1)\ (g \circ take\ n)\ (graph\ n\ f)$

proof fix x **assume** A: $x \in$ *double-graph n f g*
then have *0*: $x \in$ *carrier* $({Q_p}^{n+2}) \wedge f\ (take\ n\ x) = x!n \wedge g\ (take\ n\ x) = x!(n + 1)$
using *double-graph-def* **by** *blast*
have *1*: *take* $(n+1)\ x \in$ *graph n f*
apply(*rule graph-memI*)
using *assms*(*2*) **apply** *blast*
apply (*metis 0 append-eq-conv-conj cartesian-power-car-memE le-add1 length-take*
less-add-same-cancel1 less-numeral-extra(*1*) *min.absorb2 nth-take take-add*)
by (*metis* (*no-types*, *opaque-lifting*) *0 Suc-eq-plus1 Suc-n-not-le-n add-cancel-right-right*
dual-order.antisym le-iff-add not-less-eq-eq one-add-one plus-1-eq-Suc take-closed)
show $x \in$ *restricted-graph* $(n + 1)\ (g \circ take\ n)\ (graph\ n\ f)$
apply(*rule restricted-graph-memI*)
apply (*metis 0 One-nat-def add-Suc-right numeral-2-eq-2*)
using *1* **apply** *blast*
using *0 take-take*[*of n n + 1 x*] *comp-apply*
by (*metis le-add1 min.absorb1*)
qed
show *restricted-graph* $(n + 1)\ (g \circ take\ n)\ (graph\ n\ f) \subseteq$ *double-graph n f g*
proof fix x
assume A: $x \in$ *restricted-graph* $(n + 1)\ (g \circ take\ n)\ (graph\ n\ f)$
then have *0*: $x \in$ *carrier* $({Q_p}^{Suc\ (n + 1)}) \wedge take\ (n + 1)\ x \in graph\ n\ f \wedge (g \circ take\ n)\ (take\ (n + 1)\ x) = x\ !\ (n + 1)$
using *restricted-graph-memE*[*of x n+1* $(g \circ take\ n)$ *graph n f*]
by *blast*
then have *1*: $x \in$ *carrier* $({Q_p}^{n+2})$
using *0*
by (*metis Suc-1 add-Suc-right*)
have *2*: $f\ (take\ n\ x) = x\ !\ n$
using *0 take-take*[*of n n + 1 x*] *graph-memE*[*of f n take* $(n + 1)\ x$]
by (*metis assms*(*2*) *le-add1 less-add-same-cancel1 less-numeral-extra*(*1*) *min.absorb1 nth-take*)
have *3*: $g\ (take\ n\ x) = x\ !\ (n + 1)$
using *0 comp-apply take-take*[*of n n + 1 x*]
by (*metis le-add1 min.absorb1*)
then show $x \in$ *double-graph n f g*
unfolding *double-graph-def* **using** *1 2 3*
by *blast*
qed
qed

lemma *double-graph-is-semialg*:
assumes $n > 0$
assumes *is-semialg-function n f*
assumes *is-semialg-function n g*
shows *is-semialgebraic* $(n+2)$ (*double-graph n f g*)
using *double-graph-rep*[*of g n f*] *assms restricted-graph-is-semialgebraic*[*of n g* $\circ$

take n graph n f]
by (*metis* (*no-types*, *lifting*) *Suc-eq-plus1 add-Suc-right is-semialg-function-closed less-add-same-cancel1 less-numeral-extra*(*1*) *one-add-one restricted-graph-is-semialgebraic comp-take-is-semialg semialg-graph*)

definition *add-vars* :: *nat* ⇒ *nat* ⇒ *padic-tuple* ⇒ *padic-number* **where**
add-vars i j as = *as*!*i* $\oplus_{Q_p}$ *as*!*j*

lemma *add-vars-rep*:
assumes *as* ∈ *carrier* ($Q_p{}^n$)
assumes *i* < *n*
assumes *j* < *n*
shows *add-vars i j as* = *Qp-ev* ((*pvar* Q_p *i*) $\oplus_{Q_p[\mathcal{X}_n]}$ (*pvar* Q_p *j*)) *as*
unfolding *add-vars-def*
using *assms eval-at-point-add*[*of as n pvar* Q_p *i pvar* Q_p *j*]
eval-pvar **by** (*metis pvar-closed*)

lemma *add-vars-is-semialg*:
assumes *i* < *n*
assumes *j* < *n*
assumes *a* ∈ *carrier* ($Q_p{}^n$)
shows *is-semialg-function n* (*add-vars i j*)
proof−
have *pvar* Q_p *i* $\oplus_{Q_p[\mathcal{X}_n]}$ *pvar* Q_p *j* ∈ *carrier* ($Q_p[\mathcal{X}_n]$)
using *assms pvar-closed*[*of*]
by *blast*
then have *is-semialg-function n* (*Qp-ev* (*pvar* Q_p *i* $\oplus_{Q_p[\mathcal{X}_n]}$ *pvar* Q_p *j*))
using *assms poly-is-semialg*[*of* (*pvar* Q_p *i*) $\oplus_{Q_p[\mathcal{X}_n]}$ (*pvar* Q_p *j*)]
by *blast*
then show *?thesis*
using *assms add-vars-rep*
semialg-function-on-carrier[*of n* *Qp-ev* ((*pvar* Q_p *i*) $\oplus_{Q_p[\mathcal{X}_n]}$ (*pvar* Q_p *j*))
add-vars i j]
by (*metis* (*no-types*, *lifting*) *restrict-ext*)
qed

definition *mult-vars* :: *nat* ⇒ *nat* ⇒ *padic-tuple* ⇒ *padic-number* **where**
mult-vars i j as = *as*!*i* ⊗ *as*!*j*

lemma *mult-vars-rep*:
assumes *as* ∈ *carrier* ($Q_p{}^n$)
assumes *i* < *n*
assumes *j* < *n*
shows *mult-vars i j as* = *Qp-ev* ((*pvar* Q_p *i*) $\otimes_{Q_p[\mathcal{X}_n]}$ (*pvar* Q_p *j*)) *as*
unfolding *mult-vars-def*
using *assms eval-at-point-mult*[*of as n pvar* Q_p *i pvar* Q_p *j*]
eval-pvar[*of i n as*] *eval-pvar*[*of j n as*]
by (*metis pvar-closed*)

lemma *mult-vars-is-semialg*:
assumes $i < n$
assumes $j < n$
assumes $a \in carrier\ ({Q_p}^n)$
shows *is-semialg-function n* (*mult-vars i j*)
proof−
have *pvar* Q_p $i \otimes_{Q_p[\mathcal{X}_n]}$ *pvar* Q_p $j \in carrier\ (Q_p[\mathcal{X}_n])$
using *assms pvar-closed*[*of*]
by *blast*
then have *is-semialg-function n* (*Qp-ev* (*pvar* Q_p $i \otimes_{Q_p[\mathcal{X}_n]}$ *pvar* Q_p j))
using *assms poly-is-semialg*[*of* (*pvar* Q_p i) $\otimes_{Q_p[\mathcal{X}_n]}$ (*pvar* Q_p j)]
by *blast*
then show *?thesis*
using *assms mult-vars-rep*
semialg-function-on-carrier[*of n Qp-ev* ((*pvar* Q_p i) $\otimes_{Q_p[\mathcal{X}_n]}$ (*pvar* Q_p j))
mult-vars i j]
by (*metis* (*no-types*, *lifting*) *restrict-ext*)
qed

definition *minus-vars* :: *nat* ⇒ *padic-tuple* ⇒ *padic-number* **where**
minus-vars i as = $\ominus_{Q_p}$ *as*!*i*

lemma *minus-vars-rep*:
assumes *as* ∈ *carrier* $({Q_p}^n)$
assumes $i < n$
shows *minus-vars i as* = *Qp-ev* ($\ominus_{Q_p[\mathcal{X}_n]}$(*pvar* Q_p i)) *as*
unfolding *minus-vars-def*
using *assms eval-pvar*[*of i n as*] *eval-at-point-a-inv*[*of as n pvar* Q_p i]
by (*metis pvar-closed*)

lemma *minus-vars-is-semialg*:
assumes $i < n$
assumes $a \in carrier\ ({Q_p}^n)$
shows *is-semialg-function n* (*minus-vars i*)
proof−
have *0*: *pvar* Q_p i ∈ *carrier* $(Q_p[\mathcal{X}_n])$
using *assms pvar-closed*[*of*] *Qp.cring-axioms* **by** *presburger*
have *is-semialg-function n* (*Qp-ev* ($\ominus_{Q_p[\mathcal{X}_n]}$(*pvar* Q_p i)))
apply(*rule poly-is-semialg*)
using *0* **by** *blast*
then show *?thesis*
using *assms minus-vars-rep*[*of a i n*]
semialg-function-on-carrier[*of n - minus-vars i*]
by (*metis* (*no-types*, *lifting*) *minus-vars-rep restrict-ext*)
qed

definition *extended-graph* **where**

extended-graph n f g h = {*as* ∈ *carrier* ($Q_p{}^{n+3}$).
f (*take n as*) = *as*!*n* ∧ *g* (*take n as*) = *as*! (*n* + *1*) ∧ *h* [(*f* (*take n as*)),(*g* (*take n as*))] = *as*! (*n* + *2*) }

lemma *extended-graph-rep*:
extended-graph n f g h = *restricted-graph* (*n* + *2*) (*h* ∘ (*drop n*)) (*double-graph n f g*)
proof
show *extended-graph n f g h* ⊆ *restricted-graph* (*n* + *2*) (*h* ∘ *drop n*) (*double-graph n f g*)
proof fix *x*
assume *x* ∈ *extended-graph n f g h*
then have *A*: *x* ∈ *carrier* ($Q_p{}^{n+3}$) ∧*f* (*take n x*) = *x*!*n* ∧ *g* (*take n x*) = *x*! (*n* + *1*) ∧
h [(*f* (*take n x*)),(*g* (*take n x*))] = *x*! (*n* + *2*)
unfolding *extended-graph-def* **by** *blast*
then have *0*: *take* (*n* + *2*) *x* ∈ *carrier* ($Q_p{}^{n+2}$)
proof −
have *Suc* (*Suc n*) ≤ *n* + *numeral* (*num.One* + *num.Bit0 num.One*)
by *simp*
then show *?thesis*
by (*metis* (*no-types*) ‹*x* ∈ *carrier* ($Q_p{}^{n+3}$) ∧ *f* (*take n x*) = *x* ! *n* ∧ *g* (*take n x*) = *x* ! (*n* + *1*) ∧ *h* [*f* (*take n x*), *g* (*take n x*)] = *x* ! (*n* + *2*)› *add-2-eq-Suc' add-One-commute semiring-norm*(*5*) *take-closed*)
qed
have *1*: *f* (*take n* (*take* (*n* + *2*) *x*)) = (*take* (*n* + *2*) *x*) ! *n*
using *A*
by (*metis Suc-1 add.commute append-same-eq append-take-drop-id less-add-same-cancel1 nth-take take-add take-drop zero-less-Suc*)
have *2*: *g* (*take n* (*take* (*n* + *2*) *x*)) = (*take* (*n* + *2*) *x*) ! (*n* + *1*)
using *A*
by (*smt add.assoc add.commute append-same-eq append-take-drop-id less-add-same-cancel1 less-numeral-extra*(*1*) *nth-take one-add-one take-add take-drop*)
then have *3*: *take* (*n* + *2*) *x* ∈ *double-graph n f g*
unfolding *double-graph-def*
using *0 1 2*
by *blast*
have *4*: *drop n* (*take* (*n* + *2*) *x*) = [(*f* (*take n x*)),(*g* (*take n x*))]
proof−
have *40*: *take* (*n* + *2*) *x* ! (*n* + *1*) = *x*! (*n* + *1*)
by (*metis add.commute add-2-eq-Suc' lessI nth-take plus-1-eq-Suc*)
have *41*: *take* (*n* + *2*) *x* ! *n* = *x*! *n*
by (*metis Suc-1 less-SucI less-add-same-cancel1 less-numeral-extra*(*1*) *nth-take*)
have *42*: *take* (*n* + *2*) *x* ! (*n* + *1*) = *g* (*take n x*)
using *40 A*
by *blast*
have *43*: *take* (*n* + *2*) *x* ! *n* = *f* (*take n x*)
using *41 A*

```
      by blast
    show ?thesis using A 42 43
    by (metis 0 add-cancel-right-right cartesian-power-car-memE cartesian-power-drop
        le-add-same-cancel1 nth-drop pair-id zero-le-numeral)
  qed
  then have 5: (h ∘ drop n) (take (n + 2) x) = x ! (n + 2)
    using 3 A
    by (metis add-2-eq-Suc' comp-eq-dest-lhs)
  show x ∈ restricted-graph (n + 2) (h ∘ drop n) (double-graph n f g)
    using restricted-graph-def A 3 5
    by (metis (no-types, lifting) One-nat-def Suc-1
        add-Suc-right numeral-3-eq-3 restricted-graph-memI)
 qed
 show restricted-graph (n + 2) (h ∘ drop n) (double-graph n f g) ⊆ extended-graph
n f g h
 proof fix x assume A: x ∈ restricted-graph (n + 2) (h ∘ drop n) (double-graph
n f g)
  then have 0: take (n+2) x ∈ double-graph n f g
    using restricted-graph-memE(2) by blast
  have 1: (h ∘ drop n) (take (n+2) x) = x! (n+2)
    by (meson A restricted-graph-memE(3) padic-fields-axioms)
  have 2: x ∈ carrier (Q_p^(n+3))
    using A
   by (metis (no-types, opaque-lifting) Suc3-eq-add-3 add.commute add-2-eq-Suc'
        restricted-graph-closed subsetD)
  have 3: length x = n + 3
    using 2 cartesian-power-car-memE by blast
  have 4: drop n (take (n+2) x) = [x!n, x!(n+1)]
  proof−
    have length (take (n+2) x) = n+2
      by (simp add: 3)
    then have 40:length (drop n (take (n+2) x)) = 2
      by (metis add-2-eq-Suc' add-diff-cancel-left' length-drop)
    have 41: (drop n (take (n+2) x))!0 = x!n
      using 3
     by (metis Nat.add-0-right ‹length (take (n + 2) x) = n + 2› add-gr-0 le-add1
less-add-same-cancel1 less-numeral-extra(1) nth-drop nth-take one-add-one)
    have 42: (drop n (take (n+2) x))!1 = x!(n+1)
      using 3 nth-take nth-drop A
     by (metis add.commute le-add1 less-add-same-cancel1 less-numeral-extra(1)
one-add-one take-drop)
    show ?thesis
      using 40 41 42
      by (metis pair-id)
  qed
  have (take n x) = take n (take (n+2) x)
    using take-take 3
    by (metis le-add1 min.absorb1)
  then have 5: f (take n x) = x ! n
```

```
    using 0 double-graph-def[of n f g] 3
    by (smt Suc-1 less-add-same-cancel1 mem-Collect-eq nth-take zero-less-Suc)
  have 6: g (take n x) = x ! (n + 1)
    using 0 double-graph-def[of n f g] 3 take-take[of n n+2 x]
     by (smt Suc-1 ‹take n x = take n (take (n + 2) x)› add-Suc-right lessI
mem-Collect-eq nth-take)
  have 7: h [f (take n x), g (take n x)] = x ! (n + 2)
    using 4 A comp-apply
    by (metis 1 5 6)
  show x ∈ extended-graph n f g h
    unfolding extended-graph-def
    using 2 5 6 7 A
    by blast
 qed
qed

lemma function-tuple-eval-closed:
 assumes is-function-tuple Q_p n fs
 assumes x ∈ carrier (Q_p^n)
 shows function-tuple-eval Q_p n fs x ∈ carrier (Q_p^length fs)
 using function-tuple-eval-closed[of Q_p n fs x] assms by blast

definition k-graph where
k-graph n fs = {x ∈ carrier (Q_p^n + length fs). x = (take n x)@ (function-tuple-eval
Q_p n fs (take n x)) }

lemma k-graph-memI:
 assumes is-function-tuple Q_p n fs
 assumes x = as@function-tuple-eval Q_p n fs as
 assumes as ∈ carrier (Q_p^n)
 shows x ∈ k-graph n fs
proof−
 have take n x = as
   using assms
   by (metis append-eq-conv-conj cartesian-power-car-memE)
 then show ?thesis unfolding k-graph-def using assms
  by (smt append-eq-conv-conj cartesian-power-car-memE cartesian-power-car-memI''
length-append local.function-tuple-eval-closed mem-Collect-eq)
qed
```

composing a function with a function tuple

```
lemma Qp-function-tuple-comp-closed:
 assumes f ∈ carrier (Q_p^n) → carrier Q_p
 assumes length fs = n
 assumes is-function-tuple Q_p m fs
 shows function-tuple-comp Q_p fs f ∈ carrier (Q_p^m) → carrier Q_p
 using assms function-tuple-comp-closed
 by blast
```

13.11.1 Tuples of Semialgebraic Functions

Predicate for a tuple of semialgebraic functions

definition *is-semialg-function-tuple* **where**
is-semialg-function-tuple n fs = (∀ *f* ∈ *set fs*. *is-semialg-function n f*)

lemma *is-semialg-function-tupleI*:
assumes ⋀ *f*. *f* ∈ *set fs* ⟹ *is-semialg-function n f*
shows *is-semialg-function-tuple n fs*
using *assms is-semialg-function-tuple-def*
by *blast*

lemma *is-semialg-function-tupleE*:
assumes *is-semialg-function-tuple n fs*
assumes *i* < *length fs*
shows *is-semialg-function n* (*fs* ! *i*)
by (*meson assms*(*1*) *assms*(*2*) *in-set-conv-nth is-semialg-function-tuple-def padic-fields-axioms*)

lemma *is-semialg-function-tupleE'*:
assumes *is-semialg-function-tuple n fs*
assumes *f* ∈ *set fs*
shows *is-semialg-function n f*
using *assms*(*1*) *assms*(*2*) *is-semialg-function-tuple-def*
by *blast*

lemma *semialg-function-tuple-is-function-tuple*:
assumes *is-semialg-function-tuple n fs*
shows *is-function-tuple* Q_p *n fs*
apply(*rule is-function-tupleI*)
using *assms is-semialg-function-closed is-semialg-function-tupleE'* **by** *blast*

lemma *const-poly-function-tuple-comp-is-semialg*:
assumes *n* > *0*
assumes *is-semialg-function-tuple n fs*
assumes *a* ∈ *carrier* Q_p
shows *is-semialg-function n* (*poly-function-tuple-comp* Q_p *n fs* (*Qp-to-IP a*))
apply(*rule semialg-function-on-carrier*[*of n Qp-ev* (*Qp-to-IP a*)])
using *poly-is-semialg*[*of* (*Qp-to-IP a*)]
using *assms*(*1*) *assms*(*3*) *Qp-to-IP-car* **apply** *blast*
using *poly-function-tuple-comp-eq*[*of n fs* (*Qp-to-IP a*)] *assms* **unfolding** *restrict-def*
by (*metis* (*no-types*, *opaque-lifting*) *eval-at-point-const poly-function-tuple-comp-constant semialg-function-tuple-is-function-tuple*)

lemma *pvar-poly-function-tuple-comp-is-semialg*:
assumes *n* > *0*
assumes *is-semialg-function-tuple n fs*
assumes *i* < *length fs*
shows *is-semialg-function n* (*poly-function-tuple-comp* Q_p *n fs* (*pvar* Q_p *i*))

apply(*rule semialg-function-on-carrier*[*of n fs*!*i*])
using *assms*(*2*) *assms*(*3*) *is-semialg-function-tupleE* **apply** *blast*
by (*metis assms*(*2*) *assms*(*3*) *poly-function-tuple-comp-pvar*
restrict-ext semialg-function-tuple-is-function-tuple)

Polynomial functions with semialgebraic coefficients

definition *point-to-univ-poly* :: *nat* ⇒ *padic-tuple* ⇒ *padic-univ-poly* **where**
point-to-univ-poly n a = *ring-cfs-to-univ-poly n a*

definition *tuple-partial-image* **where**
tuple-partial-image m fs x = (*function-tuple-eval* Q_p *m fs* (*take m x*))@(*drop m x*)

lemma *tuple-partial-image-closed*:
assumes *length fs* > *0*
assumes *is-function-tuple* Q_p *n fs*
assumes *x* ∈ *carrier* ($Q_p{}^{n+l}$)
shows *tuple-partial-image n fs x* ∈ *carrier* ($Q_p{}^{length\ fs\ +\ l}$)
using *assms* **unfolding** *tuple-partial-image-def*
by (*meson cartesian-power-concat*(*1*) *cartesian-power-drop*
function-tuple-eval-closed le-add1 take-closed)

lemma *tuple-partial-image-indices*:
assumes *length fs* > *0*
assumes *is-function-tuple* Q_p *n fs*
assumes *x* ∈ *carrier* ($Q_p{}^{n+l}$)
assumes *i* < *length fs*
shows (*tuple-partial-image n fs x*) ! *i* = (*fs*!*i*) (*take n x*)
proof−
have *0*: (*function-tuple-eval* Q_p *n fs* (*take n x*)) ! *i* = (*fs*!*i*) (*take n x*)
using *assms* **unfolding** *function-tuple-eval-def*
by (*meson nth-map*)
have *1*: *length* (*function-tuple-eval* Q_p *n fs* (*take n x*)) > *i*
by (*metis assms*(*4*) *function-tuple-eval-def length-map*)
show *?thesis*
using *0 1 assms* **unfolding** *tuple-partial-image-def*
by (*metis nth-append*)
qed

lemma *tuple-partial-image-indices'*:
assumes *length fs* > *0*
assumes *is-function-tuple* Q_p *n fs*
assumes *x* ∈ *carrier* ($Q_p{}^{n+l}$)
assumes *i* < *l*
shows (*tuple-partial-image n fs x*) ! (*length fs* + *i*) = *x*!(*n* + *i*)
using *assms* **unfolding** *tuple-partial-image-def*
by (*metis* (*no-types*, *lifting*) *cartesian-power-car-memE function-tuple-eval-closed*
le-add1
nth-append-length-plus nth-drop take-closed)

definition *tuple-partial-pullback* **where**
tuple-partial-pullback n fs l S = ((*tuple-partial-image n fs*)−‘*S*) ∩ *carrier* ($Q_p{}^{n+l}$)

lemma *tuple-partial-pullback-memE*:
assumes *as* ∈ *tuple-partial-pullback m fs l S*
shows *as* ∈ *carrier* ($Q_p{}^{m\ +\ l}$) *tuple-partial-image m fs as* ∈ *S*
using *assms*
apply (*metis* (*no-types*, *opaque-lifting*) *Int-iff add.commute tuple-partial-pullback-def*)
using *assms* **unfolding** *tuple-partial-pullback-def*
by *blast*

lemma *tuple-partial-pullback-closed*:
tuple-partial-pullback m fs l S ⊆ *carrier* ($Q_p{}^{m\ +\ l}$)
using *tuple-partial-pullback-memE* **by** *blast*

lemma *tuple-partial-pullback-memI*:
assumes *as* ∈ *carrier* ($Q_p{}^{m\ +\ k}$)
assumes *is-function-tuple* Q_p *m fs*
assumes ((*function-tuple-eval* Q_p *m fs*) (*take m as*))@(*drop m as*) ∈ *S*
shows *as* ∈ *tuple-partial-pullback m fs k S*
using *assms* **unfolding** *tuple-partial-pullback-def tuple-partial-image-def*
by *blast*

lemma *tuple-partial-image-eq*:
assumes *as* ∈ *carrier* ($Q_p{}^{n}$)
assumes *bs* ∈ *carrier* ($Q_p{}^{k}$)
assumes *x* = *as* @ *bs*
shows *tuple-partial-image n fs x* = ((*function-tuple-eval* Q_p *n fs*) *as*)@*bs*
proof−
have *0*: (*take n x*) = *as*
by (*metis append-eq-conv-conj assms*(*1*) *assms*(*3*) *cartesian-power-car-memE*)
have *1*: *drop n x* = *bs*
by (*metis 0 append-take-drop-id assms*(*3*) *same-append-eq*)
show *?thesis* **using** *assms 0 1* **unfolding** *tuple-partial-image-def*
by *presburger*
qed

lemma *tuple-partial-pullback-memE′*:
assumes *as* ∈ *carrier* ($Q_p{}^{n}$)
assumes *bs* ∈ *carrier* ($Q_p{}^{k}$)
assumes *x* = *as* @ *bs*
assumes *x* ∈ *tuple-partial-pullback n fs k S*
shows (*function-tuple-eval* Q_p *n fs as*)@*bs* ∈ *S*
using *tuple-partial-pullback-memE*[*of x n fs k S*] *tuple-partial-image-def*[*of n fs x*]
by (*metis assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *tuple-partial-image-eq*)

tuple partial pullbacks have the same algebraic properties as pullbacks

lemma *tuple-partial-pullback-intersect*:

tuple-partial-pullback m f l (S1 ∩ S2) = (tuple-partial-pullback m f l S1) ∩ (tuple-partial-pullback m f l S2)
unfolding *tuple-partial-pullback-def*
by *blast*

lemma *tuple-partial-pullback-union*:
tuple-partial-pullback m f l (S1 ∪ S2) = (tuple-partial-pullback m f l S1) ∪ (tuple-partial-pullback m f l S2)
unfolding *tuple-partial-pullback-def*
by *blast*

lemma *tuple-partial-pullback-complement*:
assumes *is-function-tuple* Q_p *m fs*
shows *tuple-partial-pullback m fs l* ((*carrier* (Q_p ^ *length fs + l*)) − *S*) = *carrier* (Q_p ^ *m + l*) − (*tuple-partial-pullback m fs l S*)
apply(*rule equalityI*)
using *tuple-partial-pullback-def*[*of m fs l* ((*carrier* (Q_p ^ *length fs + l*)) − *S*)]
tuple-partial-pullback-def[*of m fs l S*]
apply *blast*
proof fix *x* **assume** *A*: *x* ∈ *carrier* (Q_p ^ *m + l*) − *tuple-partial-pullback m fs l S*
show *x* ∈ *tuple-partial-pullback m fs l* (*carrier* (Q_p ^ *length fs + l*) − *S*)
apply(*rule tuple-partial-pullback-memI*)
using *A*
apply *blast*
using *assms*
apply *blast*
proof
have *0*: *drop m x* ∈ *carrier* (Q_p ^ *l*)
by (*meson A DiffD1 cartesian-power-drop*)
have *1*: *take m x* ∈ *carrier* (Q_p ^ *m*)
using *A*
by (*meson DiffD1 le-add1 take-closed*)
show *function-tuple-eval* Q_p *m fs* (*take m x*) @ *drop m x*
∈ *carrier* (Q_p ^ *length fs + l*)
using *0 1 assms*
using *cartesian-power-concat*(*1*) *function-tuple-eval-closed* **by** *blast*
show *function-tuple-eval* Q_p *m fs* (*take m x*) @ *drop m x* ∉ *S*
using *A* **unfolding** *tuple-partial-pullback-def tuple-partial-image-def*
by *blast*
qed
qed

lemma *tuple-partial-pullback-carrier*:
assumes *is-function-tuple* Q_p *m fs*
shows *tuple-partial-pullback m fs l* (*carrier* (Q_p ^ *length fs + l*)) = *carrier* (Q_p ^ *m + l*)
apply(*rule equalityI*)
using *tuple-partial-pullback-memE*(*1*) **apply** *blast*
proof fix *x* **assume** *A*: *x* ∈ *carrier* (Q_p ^ *m + l*)

show $x \in$ *tuple-partial-pullback m fs l* (*carrier* (Q_p ^ *length fs* + *l*))
apply(*rule tuple-partial-pullback-memI*)
using *A cartesian-power-drop*[*of x m l*] *take-closed assms*
apply *blast*
using *assms* **apply** *blast*
proof−
have *function-tuple-eval* Q_p *m fs* (*take m x*) $\in$ *carrier* (Q_p ^ *length fs*)
using *A take-closed assms*
function-tuple-eval-closed le-add1
by *blast*
then show *function-tuple-eval* Q_p *m fs* (*take m x*) @ *drop m x*
$\in$ *carrier* (Q_p ^ *length fs* + *l*)
using *cartesian-power-drop*[*of x m l*] *A cartesian-power-concat*(*1*)
by *blast*
qed
qed

definition *is-semialg-map-tuple* **where**
is-semialg-map-tuple m fs = (*is-function-tuple* Q_p *m fs* ∧
(∀ *l* ≥ *0*. ∀ *S* ∈ *semialg-sets* ((*length fs*) + *l*). *is-semialgebraic*
(*m* + *l*) (*tuple-partial-pullback m fs l S*)))

lemma *is-semialg-map-tuple-closed*:
assumes *is-semialg-map-tuple m fs*
shows *is-function-tuple* Q_p *m fs*
using *is-semialg-map-tuple-def assms* **by** *blast*

lemma *is-semialg-map-tupleE*:
assumes *is-semialg-map-tuple m fs*
assumes *is-semialgebraic* ((*length fs*) + *l*) *S*
shows *is-semialgebraic* (*m* + *l*) (*tuple-partial-pullback m fs l S*)
using *is-semialg-map-tuple-def*[*of m fs*] *assms is-semialgebraicE*[*of* ((*length fs*)
+ *l*) *S*]
by *blast*

lemma *is-semialg-map-tupleI*:
assumes *is-function-tuple* Q_p *m fs*
assumes ⋀*k S*. *S* ∈ *semialg-sets* ((*length fs*) + *k*) ⟹ *is-semialgebraic* (*m* +
k) (*tuple-partial-pullback m fs k S*)
shows *is-semialg-map-tuple m fs*
using *assms* **unfolding** *is-semialg-map-tuple-def*
by *blast*

Semialgebraicity for maps can be verified on basic semialgebraic sets

lemma *is-semialg-map-tupleI′*:
assumes *is-function-tuple* Q_p *m fs*
assumes ⋀*k S*. *S* ∈ *basic-semialgs* ((*length fs*) + *k*) ⟹ *is-semialgebraic* (*m* +
k) (*tuple-partial-pullback m fs k S*)
shows *is-semialg-map-tuple m fs*

apply(*rule is-semialg-map-tupleI*)
using *assms(1)* **apply** *blast*
proof−
show $\bigwedge k\ S.\ S \in$ *semialg-sets* ((*length fs*) + *k*) $\Longrightarrow$ *is-semialgebraic* (*m* + *k*) (*tuple-partial-pullback m fs k S*)
proof− **fix** *k S* **assume** *A*: $S \in$ *semialg-sets* ((*length fs*) + *k*)
show *is-semialgebraic* (*m* + *k*) (*tuple-partial-pullback m fs k S*)
apply(*rule gen-boolean-algebra.induct*[*of S carrier* ($Q_p{}^{length\ fs\ +\ k}$) *basic-semialgs* ((*length fs*) + *k*)])
using *A* **unfolding** *semialg-sets-def*
apply *blast*
using *tuple-partial-pullback-carrier assms carrier-is-semialgebraic plus-1-eq-Suc* **apply** *presburger*
using *assms(1) assms(2) carrier-is-semialgebraic intersection-is-semialg tuple-partial-pullback-carrier tuple-partial-pullback-intersect* **apply** *presburger*
using *tuple-partial-pullback-union union-is-semialgebraic* **apply** *presburger*
using *assms(1) complement-is-semialg tuple-partial-pullback-complement plus-1-eq-Suc* **by** *presburger*
qed
qed

The goal of this section is to show that tuples of semialgebraic functions are semialgebraic maps.

The function $(x_0, x, y) \mapsto (x_0, f(x), y)$

definition *twisted-partial-image* **where**
twisted-partial-image n m f xs = (*take n xs*)@ *partial-image m f* (*drop n xs*)

The set $(x_0, x, y) \mid (x_0, f(x), y) \in S$

Convention: a function which produces a subset of (Qp (i + j +k)) will receive the 3 arity parameters in sequence, at the very beginning of the function

definition *twisted-partial-pullback* **where**
twisted-partial-pullback n m l f S = ((*twisted-partial-image n m f*)−‘*S*) ∩ *carrier* ($Q_p{}^{n+m+l}$)

lemma *twisted-partial-pullback-memE*:
assumes *as* ∈ *twisted-partial-pullback n m l f S*
shows *as* ∈ *carrier* ($Q_p{}^{n+m+l}$) *twisted-partial-image n m f as* ∈ *S*
using *assms*
apply (*metis* (*no-types*, *opaque-lifting*) *Int-iff add.commute twisted-partial-pullback-def subset-iff*)
using *assms* **unfolding** *twisted-partial-pullback-def*
by *blast*

lemma *twisted-partial-pullback-closed*:
twisted-partial-pullback n m l f S ⊆ *carrier* ($Q_p{}^{n+m+l}$)

using *twisted-partial-pullback-memE*(*1*) **by** *blast*

lemma *twisted-partial-pullback-memI*:
assumes *as* ∈ *carrier* ($Q_p{}^{n+m+l}$)
assumes (*take n as*)@((*f* (*take m* (*drop n as*)))#(*drop* (*n* + *m*) *as*)) ∈ *S*
shows *as* ∈ *twisted-partial-pullback n m l f S*
using *assms* **unfolding** *twisted-partial-pullback-def twisted-partial-image-def*
by (*metis* (*no-types*, *lifting*) *IntI add.commute drop-drop local.partial-image-def vimageI*)

lemma *twisted-partial-image-eq*:
assumes *as* ∈ *carrier* ($Q_p{}^{n}$)
assumes *bs* ∈ *carrier* ($Q_p{}^{m}$)
assumes *cs* ∈ *carrier* ($Q_p{}^{l}$)
assumes *x* = *as* @ *bs* @ *cs*
shows *twisted-partial-image n m f x* = *as*@((*f bs*)#*cs*)
proof−
have *0*: (*take n x*) = *as*
by (*metis append-eq-conv-conj assms*(*1*) *assms*(*4*)
cartesian-power-car-memE)
have *1*: *twisted-partial-image n m f x* = *as*@(*partial-image m f* (*bs*@*cs*))
using *0 assms twisted-partial-image-def*
by (*metis append-eq-conv-conj cartesian-power-car-memE*)
have *2*: (*partial-image m f* (*bs*@*cs*)) = (*f bs*)#*cs*
using *partial-image-eq*[*of bs m cs l bs*@*cs f*] *assms*
by *blast*
show *?thesis* **using** *assms 0 1 2*
by (*metis*)
qed

lemma *twisted-partial-pullback-memE′*:
assumes *as* ∈ *carrier* ($Q_p{}^{n}$)
assumes *bs* ∈ *carrier* ($Q_p{}^{m}$)
assumes *cs* ∈ *carrier* ($Q_p{}^{l}$)
assumes *x* = *as* @ *bs* @ *cs*
assumes *x* ∈ *twisted-partial-pullback n m l f S*
shows *as*@((*f bs*)#*cs*) ∈ *S*
by (*metis* (*no-types*, *lifting*) *assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *assms*(*5*)
twisted-partial-image-eq twisted-partial-pullback-memE(*2*))

partial pullbacks have the same algebraic properties as pullbacks

permutation which moves the entry at index i to 0

definition *twisting-permutation* **where**
twisting-permutation (*i*::*nat*) = (λ*j*. *if j* < *i then j* + *1 else*
(*if j* = *i then 0 else j*))

lemma *twisting-permutation-permutes*:
assumes *i* < *n*

```
  shows twisting-permutation i permutes {..<n}
proof−
  have 0: ⋀x. x > i ⟹ twisting-permutation i x = x
    unfolding twisting-permutation-def
    by auto
  have 1: (∀ x. x ∉ {..<n} ⟶ twisting-permutation i x = x)
    using 0 assms
    by auto
  have 2: (∀ y. ∃!x. twisting-permutation i x = y)
  proof fix y
    show  ∃!x. twisting-permutation i x = y
    proof(cases y = 0)
      case True
      show ∃!x. twisting-permutation i x = y
        by (metis  Suc-eq-plus1 True  add-eq-0-iff-both-eq-0 less-nat-zero-code
            nat-neq-iff twisting-permutation-def zero-neq-one)
    next
      case False
      show ?thesis
      proof(cases y ≤i)
        case True
        show ?thesis
        proof
          show twisting-permutation i (y − 1) = y
            using True
                by (metis False add.commute add-diff-inverse-nat diff-less gr-zeroI
le-eq-less-or-eq
                    less-imp-diff-less less-one twisting-permutation-def)
          show ⋀x. twisting-permutation i x = y ⟹ x = y − 1
            using True False twisting-permutation-def by force
        qed
      next
        case False
        then show ?thesis
          by (auto simp add: twisting-permutation-def)
      qed
    qed
  qed
  show ?thesis
    using 1 2
    by (simp add: permutes-def)
qed

lemma twisting-permutation-action:
  assumes length as = i
  shows permute-list (twisting-permutation i) (b#(as@bs)) =  as@(b#bs)
proof−
   have 0: length (permute-list (twisting-permutation i) (b#(as@bs))) = length
(as@(b#bs))
```

```
    by (metis add.assoc length-append length-permute-list list.size(4))
  have ⋀j. j < length (as@(b#bs))
        ⟹ (permute-list (twisting-permutation i) (b#(as@bs))) ! j =  (as@(b#bs))
! j
  proof−
    fix j assume A: j < length (as@(b#bs))
    show (permute-list (twisting-permutation i) (b#(as@bs))) ! j =  (as@(b#bs))
! j
    proof(cases j < i)
      case True
      then have T0: twisting-permutation i j = j + 1
        using twisting-permutation-def by auto
      then have T1: (b # as @ bs) ! twisting-permutation i j = as!j
        using assms
        by (simp add: assms True nth-append)
      have T2: (permute-list (twisting-permutation i) (b # as @ bs)) ! j = as!j
      proof−
        have twisting-permutation i permutes {..<length (b # as @ bs)}
          by (metis (full-types) assms length-Cons length-append
              not-add-less1 not-less-eq twisting-permutation-permutes)
        then show ?thesis
          using True permute-list-nth[of twisting-permutation i b#(as@bs) j ]
              twisting-permutation-permutes[of i length (b#(as@bs))] assms
          by (metis T0 T1 add-cancel-right-right lessThan-iff
              permutes-not-in zero-neq-one)
      qed
      have T3: (as @ b # bs) ! j = as!j
        using assms True
        by (simp add: assms nth-append)
     show (permute-list (twisting-permutation i) (b #( as @ bs))) ! j = (as @ b #
bs) ! j
        using T3 T2
        by simp
    next
      case False
      show ?thesis
      proof(cases j = i)
        case True
      then have T0: twisting-permutation i j = 0
        using twisting-permutation-def by auto
      then have T1: (b # as @ bs) ! twisting-permutation i j = b
        using assms
        by (simp add: assms True nth-append)
      have T2: (permute-list (twisting-permutation i) (b # as @ bs)) ! j = b
      proof−
        have twisting-permutation i permutes {..<length (b # as @ bs)}
          by (metis (full-types) assms length-Cons length-append
              not-add-less1 not-less-eq twisting-permutation-permutes)
        then show ?thesis
```

```
      using True permute-list-nth[of twisting-permutation i b#(as@bs) j ]
        twisting-permutation-permutes[of i length (b#(as@bs))] assms
      by (metis 0 A T1 length-permute-list)
    qed
    have T3: (as @ b # bs) ! j = b
     by (metis True assms nth-append-length)
    show ?thesis
     by (simp add: T2 T3)
    next
     case F: False
     then have F0: twisting-permutation i j = j
      by (simp add: False twisting-permutation-def)
     then have F1: (b # as @ bs) ! twisting-permutation i j = bs! (j − i − 1)
      using assms
      by (metis (mono-tags, lifting) F False Suc-diff-1
          cancel-ab-semigroup-add-class.diff-right-commute linorder-neqE-nat
not-gr-zero
          not-less-eq nth-Cons' nth-append)
     have F2: (permute-list (twisting-permutation i) (b # as @ bs)) ! j = bs ! (j
− i − 1)
      using F1 assms permute-list-nth
      by (metis A add-cancel-right-right append.assoc last-to-first-eq le-add1
      le-eq-less-or-eq length-0-conv length-append length-permute-list list.distinct(1)
        twisting-permutation-permutes)
     have F3: (as @ b # bs) ! j = bs!(j − i − 1)
        by (metis F False assms linorder-neqE-nat nth-Cons-pos nth-append
zero-less-diff)
     then show ?thesis
      using F2 F3
      by presburger
    qed
   qed
  qed
  then show ?thesis
   using 0
   by (metis nth-equalityI)
qed

lemma twisting-permutation-action':
 assumes length as = i
 shows permute-list (fun-inv (twisting-permutation i)) (as@(b#bs)) = (b#(as@bs))

proof−
 obtain TI where TI-def: TI = twisting-permutation i
  by blast
 have 0: TI permutes {..<length (as@(b#bs))}
  using assms TI-def twisting-permutation-permutes[of i length (as@(b#bs))]
  by (metis add-diff-cancel-left' gr0I length-0-conv length-append list.distinct(1)
zero-less-diff)
```

have *1*: (*fun-inv TI*) *permutes* {..<*length* (*as*@(*b*#*bs*))}
by (*metis 0 Nil-is-append-conv fun-inv-permute*(*1*) *length-greater-0-conv list.distinct*(*1*))
have *permute-list* (*fun-inv* (*twisting-permutation i*)) (*as*@(*b*#*bs*)) =
permute-list (*fun-inv* (*twisting-permutation i*)) (*permute-list* (*twisting-permutation i*) (*b*#(*as*@*bs*)))
using *twisting-permutation-action*[*of as i b bs*] *assms*
by (*simp add*: ‹*length as* = *i*›)
then have *permute-list* (*fun-inv TI*) (*as*@(*b*#*bs*)) =
permute-list ((*fun-inv TI*) ∘ *TI*) (*b*#(*as*@*bs*))
using *0 1*
by (*metis TI-def fun-inv-permute*(*2*) *fun-inv-permute*(*3*) *length-greater-0-conv length-permute-list permute-list-compose*)
then show *?thesis*
by (*metis 0 Nil-is-append-conv TI-def fun-inv-permute*(*3*) *length-greater-0-conv list.distinct*(*1*) *permute-list-id*)
qed

lemma *twisting-semialg*:
assumes *is-semialgebraic n S*
assumes *n* > *i*
shows *is-semialgebraic n* ((*permute-list* ((*twisting-permutation i*)) ‘ *S*))
proof−
obtain *TI* **where** *TI-def*: *TI* = *twisting-permutation i*
by *blast*
have *0*: *TI permutes* {..<(*n*::*nat*)}
using *assms TI-def twisting-permutation-permutes*[*of i n*]
by *blast*
have (*TI*) *permutes* {..<*n*}
using *TI-def 0*
by *auto*
then show *?thesis*
using *assms permutation-is-semialgebraic*[*of n S TI*] *TI-def*
by *blast*
qed

lemma *twisting-semialg′*:
assumes *is-semialgebraic n S*
assumes *n* > *i*
shows *is-semialgebraic n* ((*permute-list* (*fun-inv* (*twisting-permutation i*)) ‘ *S*))
proof−
obtain *TI* **where** *TI-def*: *TI* = *twisting-permutation i*
by *blast*
have *0*: *TI permutes* {..<(*n*::*nat*)}
using *assms TI-def twisting-permutation-permutes*[*of i n*]
by *blast*
have (*fun-inv TI*) *permutes* {..<*n*} **using** *0 permutes-inv*[*of TI* {..<*n*}]
unfolding *fun-inv-def*
by *blast*
then show *?thesis*

using *assms permutation-is-semialgebraic*[*of n S fun-inv TI*] *TI-def*
by *blast*
qed

Defining a permutation that does: $(x0, x1, y) \mapsto (x_1, x0, y)$

definition *tp-1* **where**
tp-1 i j = (λ *n*. (*if n*<*i then j* + *n else*
(*if i* ≤ *n* ∧ *n* < *i* + *j then n* − *i else*
n)))
lemma *permutes-I*:
assumes ⋀*x*. *x* ∉ *S* ⟹ *f x* = *x*
assumes ⋀*y*. *y* ∈ *S* ⟹ ∃!*x* ∈ *S*. *f x* = *y*
assumes ⋀*x*. *x* ∈ *S* ⟹ *f x* ∈ *S*
shows *f permutes S*
proof−
have *0* : (∀ *x*. *x* ∉ *S* ⟶ *f x* = *x*)
using *assms*(*1*) **by** *blast*
have *1*: (∀ *y*. ∃!*x*. *f x* = *y*)
proof fix *y*
show ∃!*x*. *f x* = *y*
apply(*cases y* ∈ *S*)
apply (*metis 0 assms*(*2*))
proof−
assume *y* ∉ *S*
then show ∃!*x*. *f x* = *y*
by (*metis assms*(*1*) *assms*(*3*))
qed
qed
show *?thesis*
using *assms 1*
unfolding *permutes-def*
by *blast*
qed

lemma *tp-1-permutes*:
(*tp-1* (*i*::*nat*) *j*) *permutes* {..< *i* + *j*}
proof(*rule permutes-I*)
show ⋀*x*. *x* ∉ {..<*i* + *j*} ⟹ *tp-1 i j x* = *x*
proof− **fix** *x* **assume** *A*: *x* ∉ {..<*i* + *j*}
then show *tp-1 i j x* = *x*
unfolding *tp-1-def*
by *auto*
qed
show ⋀*y*. *y* ∈ {..<*i* + *j*} ⟹ ∃!*x*. *x* ∈ {..<*i* + *j*} ∧ *tp-1 i j x* = *y*
proof− **fix** *y* **assume** *A*: *y* ∈ {..<*i* + *j*}
show ∃!*x*. *x* ∈ {..<*i* + *j*} ∧ *tp-1 i j x* = *y*
proof(*cases y* < *j*)
case *True*
then have *0*:*tp-1 i j* (*y* + *i*) = *y*

```
      by (simp add: tp-1-def)
    have 1: ⋀x. x ≠ y + i ⟹ tp-1 i j x ≠ y
    proof− fix x assume A: x ≠ y + i
      show tp-1 i j x ≠ y
        apply(cases x < j)
          apply (metis A True add.commute le-add-diff-inverse le-eq-less-or-eq
nat-neq-iff not-add-less1 tp-1-def trans-less-add2)
         by (metis A True add.commute le-add-diff-inverse less-not-refl tp-1-def
trans-less-add1)
    qed
    show ?thesis using 0 1
      by (metis A ‹⋀x. x ∉ {..<i + j} ⟹ tp-1 i j x = x›)
  next
    case False
    then have y − j < i
      using A by auto
    then have tp-1 i j (y − j) = y
      using False tp-1-def
      by (simp add: tp-1-def)
    then show ?thesis
      by (smt A False ‹⋀x. x ∉ {..<i + j} ⟹ tp-1 i j x = x›
          add.commute add-diff-inverse-nat add-left-imp-eq
          less-diff-conv2 not-less tp-1-def
          padic-fields-axioms)
  qed
 qed
 show ⋀x. x ∈ {..<i + j} ⟹ tp-1 i j x ∈ {..<i + j}
 proof fix x assume A: x ∈ {..<i + j}
  show tp-1 i j x < i + j
    unfolding tp-1-def using A
    by (simp add: trans-less-add2)
 qed
qed

lemma tp-1-permutes':
(tp-1 (i::nat) j) permutes {..< i + j + k}
 using tp-1-permutes
 by (simp add: permutes-def)

lemma tp-1-permutation-action:
 assumes a ∈ carrier (Q_p^i)
 assumes b ∈ carrier (Q_p^j)
 assumes c ∈ carrier (Q_p^n)
 shows permute-list (tp-1 i j) (b@a@c)= a@b@c
proof−
 have 0:length (permute-list (tp-1 i j) (b@a@c))= length (a@b@c)
  by (metis add.commute append.assoc length-append length-permute-list)
 have ⋀x. x < length (a@b@c) ⟹ (permute-list (tp-1 i j) (b@a@c)) ! x= (a@b@c)
! x
```

proof− **fix** x **assume** A: $x < length\ (a@b@c)$
have B: $length\ (a\ @\ b\ @\ c) = i + j + length\ c$
using *add.assoc assms*(*1*) *assms*(*2*) *cartesian-power-car-memE length-append*
by *metis*
have C: *tp-1 i j permutes* {..<*length* (*a* @ *b* @ *c*)}
using *B assms tp-1-permutes'*[*of i j length b*] *tp-1-permutes'*
by *presburger*
have D: $length\ a = i$
using *assms*(*1*) *cartesian-power-car-memE* **by** *blast*
have E: $length\ b = j$
using *assms*(*2*) *cartesian-power-car-memE* **by** *blast*
show (*permute-list* (*tp-1 i j*) (*b*@*a*@*c*)) ! *x*= (*a*@*b*@*c*) ! *x*
proof(*cases x* < *i*)
case *True*
have *T0*: (*tp-1 i j x*) = *j* + *x*
using *tp-1-def*[*of i j*] *True*
by *auto*
then have (*b*@ *a* @ *c*) ! (*tp-1 i j x*) = *a*!*x*
using *D E assms*(*1*) *assms*(*2*) *assms*(*3*) *True nth-append*
by (*metis nth-append-length-plus*)
then show *?thesis*
using *A B C assms permute-list-nth*[*of tp-1 i j a*@*b*@*c*]
by (*metis D True* ‹*length* (*permute-list* (*tp-1 i j*) (*b* @ *a* @ *c*)) =
length (*a* @ *b* @ *c*)› *length-permute-list nth-append permute-list-nth*)
next
case *False*
show *?thesis*
proof(*cases x* < *i* + *j*)
case *True*
then have *T0*: (*tp-1 i j x*) = *x* − *i*
by (*meson False not-less tp-1-def*)
have *x* − *i* < *length b*
using *E False True* **by** *linarith*
then have *T1*: *permute-list* (*tp-1 i j*) (*b*@ *a* @ *c*) ! *x* = *b*!(*x*−*i*)
using *nth-append*
by (*metis A C T0* ‹*length* (*permute-list* (*tp-1 i j*) (*b* @ *a* @ *c*)) = *length*
(*a* @ *b* @ *c*)›
length-permute-list permute-list-nth)
then show *?thesis*
by (*metis D False* ‹*x* − *i* < *length b*› *nth-append*)
next
case *False*
then have (*tp-1 i j x*) = *x*
by (*meson tp-1-def trans-less-add1*)
then show *?thesis*
by (*smt A C D E False add.commute add-diff-inverse-nat append.assoc*
length-append nth-append-length-plus permute-list-nth)
qed
qed

qed
then show *?thesis*
using *0*
by (*metis list-eq-iff-nth-eq*)
qed

definition *tw* **where**
tw i j = *permute-list* (*tp-1 j i*)

lemma *tw-is-semialg*:
assumes $n > 0$
assumes *is-semialgebraic n S*
assumes $n \geq i + j$
shows *is-semialgebraic n* ((*tw i j*)‘*S*)
unfolding *tw-def*
using *assms tp-1-permutes'*[*of j i n* − (*j* + *i*)]
permutation-is-semialgebraic[*of n S*]
by (*metis add.commute le-add-diff-inverse*)

lemma *twisted-partial-pullback-factored*:
assumes $f \in (carrier\ ({Q_p}^m)) \to carrier\ Q_p$
assumes $S \subseteq carrier\ ({Q_p}^{n+1+\ l})$
assumes $Y = partial\text{-}pullback\ m\ f\ (n + l)$ (*permute-list* (*fun-inv* (*twisting-permutation n*)) ‘ *S*)
shows *twisted-partial-pullback n m l f S* = (*tw m n*) ‘ *Y*
proof
show *twisted-partial-pullback n m l f S* ⊆ *tw m n* ‘ *Y*
proof fix *x*
assume *A*: $x \in twisted\text{-}partial\text{-}pullback\ n\ m\ l\ f\ S$
then have *x-closed*: $x \in carrier\ ({Q_p}^{n+m+l})$
using *twisted-partial-pullback-memE*(*1*) **by** *blast*
obtain *a* **where** *a-def*: *a* = *take n x*
by *blast*
obtain *b* **where** *b-def*: *b* = *take m* (*drop n x*)
by *blast*
obtain *c* **where** *c-def*: *c* = (*drop* (*n* + *m*) *x*)
by *blast*
have *x-eq*:*x* = *a*@(*b*@*c*)
by (*metis a-def append.assoc append-take-drop-id b-def c-def take-add*)
have *a-closed*: $a \in carrier\ ({Q_p}^n)$
by (*metis* (*no-types*, *lifting*) *a-def dual-order.trans le-add1 take-closed x-closed*)
have *b-closed*: $b \in carrier\ ({Q_p}^m)$
proof−
have *drop n x* ∈ $carrier\ ({Q_p}^{m\ +\ l})$
by (*metis* (*no-types*, *lifting*) *add.assoc cartesian-power-drop x-closed*)
then show *?thesis*
using *b-def le-add1 take-closed* **by** *blast*
qed
have *c-closed*: $c \in carrier\ ({Q_p}^l)$

using *c-def cartesian-power-drop x-closed* **by** *blast*
have *B*: $a@((f\ b)\#c) \in S$
using *A twisted-partial-pullback-memE'*
by (*smt a-closed a-def add.commute append-take-drop-id b-closed b-def c-closed c-def drop-drop*)
have *permute-list* (*fun-inv* (*twisting-permutation n*)) ($a@((f\ b)\#c)$) = ($f\ b)\#(a@c)$
using *assms twisting-permutation-action'*[*of a n f b c*] *a-closed cartesian-power-car-memE*
by *blast*
then have *C*: $(f\ b)\#(a@c) \in$ (*permute-list* (*fun-inv* (*twisting-permutation n*)) ‘ *S*)
by (*metis B image-eqI*)
have *C*: $b@(a@c) \in$ *partial-pullback m f* ($n + l$) (*permute-list* (*fun-inv* (*twisting-permutation n*)) ‘ *S*)
proof(*rule partial-pullback-memI*)
show $b\ @\ a\ @\ c \in$ *carrier* ($Q_p{}^m + (n + l)$)
using *a-closed b-closed c-closed cartesian-power-concat*(*1*)
by *blast*
have *0*: (*take m* ($b\ @\ a\ @\ c$)) = *b*
by (*metis append.right-neutral b-closed cartesian-power-car-memE diff-is-0-eq diff-self-eq-0 take0 take-all take-append*)
have *1*: *drop m* ($b\ @\ a\ @\ c$) = $a@c$
by (*metis 0 append-take-drop-id same-append-eq*)
show *f* (*take m* ($b\ @\ a\ @\ c$)) # *drop m* ($b\ @\ a\ @\ c$) ∈ *permute-list* (*fun-inv* (*twisting-permutation n*)) ‘ *S*
using *0 1 C*
by *presburger*
qed
have *D*: *tw m n* ($b@(a@c)$) = $a@(b@c)$
using *assms tw-def a-closed b-closed c-closed*
by (*metis tp-1-permutation-action x-eq*)
then show $x \in$ *tw m n* ‘ *Y*
using *x-eq C assms*
by (*metis image-eqI*)
qed
show *tw m n* ‘ *Y* ⊆ *twisted-partial-pullback n m l f S*
proof fix *x*
assume *A*: $x \in$ *tw m n* ‘ *Y*
then obtain *y* **where** *y-def*: $x = tw\ m\ n\ y \wedge y \in Y$
by *blast*
obtain *as* **where** *as-def*: $as \in$ (*permute-list* (*fun-inv* (*twisting-permutation n*)) ‘ *S*) ∧
as = *partial-image m f y*
using *partial-pullback-memE*
by (*metis assms*(*3*) *y-def*)
obtain *s* **where** *s-def*: $s \in S \wedge$ *permute-list* (*fun-inv* (*twisting-permutation n*)) $s = as$
using *as-def* **by** *blast*

obtain b **where** $b\text{-}def$: $b = take\ m\ y$
by $blast$
obtain a **where** $a\text{-}def$: $a = take\ n\ (drop\ m\ y)$
by $blast$
obtain c **where** $c\text{-}def$: $c = drop\ (n + m)\ y$
by $blast$
have $y\text{-}closed$: $y \in carrier\ ({Q_p}^{m\ +\ n\ +\ l})$
by $(metis\ add.assoc\ assms(3)\ partial\text{-}pullback\text{-}memE(1)\ y\text{-}def)$
then have $y\text{-}eq$: $y = b@a@c$
using $a\text{-}def\ b\text{-}def\ c\text{-}def$
by $(metis\ append\text{-}take\text{-}drop\text{-}id\ drop\text{-}drop)$
have $a\text{-}closed$: $a \in carrier\ ({Q_p}^n)$
by $(metis\ a\text{-}def\ add.commute\ cartesian\text{-}power\text{-}drop\ le\text{-}add1\ take\text{-}closed\ take\text{-}drop\ y\text{-}closed)$
have $b\text{-}closed$: $b \in carrier\ ({Q_p}^m)$
using $add\text{-}leD2\ b\text{-}def\ le\text{-}add1\ take\text{-}closed\ y\text{-}closed$
by $(meson\ trans\text{-}le\text{-}add1)$
have $c\text{-}closed$: $c \in carrier\ ({Q_p}^l)$
using $c\text{-}def\ cartesian\text{-}power\text{-}drop\ y\text{-}closed$
by $(metis\ add.commute)$
have $ac\text{-}closed$: $a@c \in carrier\ ({Q_p}^{n+l})$
using $a\text{-}closed\ c\text{-}closed\ cartesian\text{-}power\text{-}concat(1)$ **by** $blast$
then have C: $local.partial\text{-}image\ m\ f\ y = f\ b\ \#\ a\ @\ c$
using $b\text{-}closed\ y\text{-}eq\ partial\text{-}image\text{-}eq[of\ b\ m\ a@c\ n\ +\ l\ y\ f]$
by $blast$
then have $as\text{-}eq$: $as = (f\ b)\#(a@c)$
using $as\text{-}def$
by $force$
have B: $(tw\ m\ n)\ y = a@b@c$ **using** $y\text{-}eq\ tw\text{-}def[of\ n\ m]\ tp\text{-}1\text{-}permutation\text{-}action$
by $(smt\ a\text{-}closed\ b\text{-}closed\ c\text{-}closed\ tw\text{-}def)$
then have $x = a@(b@c)$
by $(simp\ add:\ y\text{-}def)$
then have $twisted\text{-}partial\text{-}image\ n\ m\ f\ x = \ a@((f\ b)\#\ c)$
using $a\text{-}closed\ b\text{-}closed\ c\text{-}closed\ twisted\text{-}partial\text{-}image\text{-}eq$
by $blast$
then have D: $permute\text{-}list\ (twisting\text{-}permutation\ n)\ as = twisted\text{-}partial\text{-}image\ n\ m\ f\ x$
using $as\text{-}eq\ twisting\text{-}permutation\text{-}action[of\ a\ n\ f\ b\ c\ ]$
by $(metis\ a\text{-}closed\ cartesian\text{-}power\text{-}car\text{-}memE)$
have $permute\text{-}list\ (twisting\text{-}permutation\ n)\ as \in S$
proof−
have S: $length\ s > n$
using $s\text{-}def\ assms\ cartesian\text{-}power\text{-}car\text{-}memE\ le\text{-}add1\ le\text{-}neq\text{-}implies\text{-}less\ le\text{-}trans\ less\text{-}add\text{-}same\text{-}cancel1\ less\text{-}one\ not\text{-}add\text{-}less1$
by $(metis\ (no\text{-}types,\ lifting)\ subset\text{-}iff)$
have $permute\text{-}list\ (twisting\text{-}permutation\ n)\ as = permute\text{-}list\ (twisting\text{-}permutation\ n)\ (permute\text{-}list\ (fun\text{-}inv\ (twisting\text{-}permutation\ n))\ s)$
using $fun\text{-}inv\text{-}def\ \ s\text{-}def$ **by** $blast$
then have $permute\text{-}list\ (twisting\text{-}permutation\ n)\ as =$

```
        permute-list ((twisting-permutation n) ∘ (fun-inv (twisting-permutation
n))) s
      using fun-inv-permute(2) fun-inv-permute(3) length-greater-0-conv
          length-permute-list twisting-permutation-permutes[of n length s]
              permute-list-compose[of fun-inv (twisting-permutation n) s twist-
ing-permutation n]
      by (metis S permute-list-compose)
    then have permute-list (twisting-permutation n) as =
      permute-list (id) s
      by (metis S ‹permute-list (twisting-permutation n) as = permute-list
        (twisting-permutation n) (permute-list (fun-inv (twisting-permutation n))
s)›
       fun-inv-permute(3) length-greater-0-conv length-permute-list permute-list-compose
         twisting-permutation-permutes)
    then have permute-list (twisting-permutation n) as = s
      by simp
    then show ?thesis
      using s-def
      by (simp add: ‹s ∈ S ∧ permute-list (fun-inv (twisting-permutation n)) s =
as›)
  qed
  then show x ∈ twisted-partial-pullback n m l f S
    unfolding twisted-partial-pullback-def using D
    by (smt ‹x = a @ b @ c› a-closed append.assoc append-eq-conv-conj b-closed
      c-closed cartesian-power-car-memE cartesian-power-concat(1) length-append
      list.inject local.partial-image-def twisted-partial-image-def
      twisted-partial-pullback-def twisted-partial-pullback-memI)
 qed
qed

lemma twisted-partial-pullback-is-semialgebraic:
 assumes is-semialg-function m f
 assumes is-semialgebraic (n + 1 + l) S
 shows is-semialgebraic (n + m + l)(twisted-partial-pullback n m l f S)
proof−
 have (fun-inv (twisting-permutation n)) permutes {..<n + 1 + l}
   by (simp add: fun-inv-permute(1) twisting-permutation-permutes)
 then have is-semialgebraic (1 + n + l) (permute-list (fun-inv (twisting-permutation
n)) ‘ S)
   using add-gr-0 assms(2) permutation-is-semialgebraic
   by (metis add.commute)
 then have is-semialgebraic (n + m + l)
        (partial-pullback m f (n + l) (permute-list (fun-inv (twisting-permutation
n)) ‘ S))
  using assms is-semialg-functionE[of m f n + l (permute-list (fun-inv (twisting-permutation
n)) ‘ S)]
   by (metis (no-types, lifting) add.assoc add.commute)
 then have is-semialgebraic (n + m + l)
              ((tw m n) ‘(partial-pullback m f (n + l) (permute-list (fun-inv
```

(*twisting-permutation n*)) ‘ *S*)))
unfolding *tw-def*
using *tp-1-permutes'*[*of n m l*] *assms permutation-is-semialgebraic*[*of n + m + l*
partial-pullback m f (*n + l*) (*permute-list* (*fun-inv* (*twisting-permutation n*)) ‘ *S*)
tp-1 n m]
by *blast*
then show *?thesis*
using *twisted-partial-pullback-factored assms*(*1*) *assms*(*2*)
is-semialg-function-closed is-semialgebraic-closed
by *presburger*
qed

definition *augment* **where**
augment n x = *take n x* @ *take n x* @ *drop n x*

lemma *augment-closed*:
assumes *x* ∈ *carrier* ($Q_p{}^{n+l}$)
shows *augment n x* ∈ *carrier* ($Q_p{}^{n+n+\ l}$)
apply(*rule cartesian-power-car-memI*)
apply (*smt ab-semigroup-add-class.add-ac*(*1*) *add.commute append-take-drop-id assms augment-def cartesian-power-car-memE cartesian-power-drop length-append*)
using *assms cartesian-power-car-memE''* **unfolding** *augment-def*
by (*metis* (*no-types*, *opaque-lifting*) *append-take-drop-id cartesian-power-concat*(*2*) *nat-le-iff-add take-closed*)

lemma *tuple-partial-image-factor*:
assumes *is-function-tuple* Q_p *m fs*
assumes *f* ∈ *carrier* ($Q_p{}^m$) → *carrier* Q_p
assumes *length fs* = *n*
assumes *x* ∈ *carrier* ($Q_p{}^{m\ +\ l}$)
shows *tuple-partial-image m* (*fs*@[*f*]) *x* = *twisted-partial-image n m f* (*tuple-partial-image m fs* (*augment m x*))
proof−
obtain *a* **where** *a-def*: *a* = *take m x*
by *blast*
obtain *b* **where** *b-def*: *b* = *drop m x*
by *blast*
have *a-closed*: *a* ∈ *carrier* ($Q_p{}^m$)
using *a-def assms*(*4*) *le-add1 take-closed*
by (*meson dual-order.trans*)
have *b-closed*: *b* ∈ *carrier* ($Q_p{}^l$)
using *assms*(*4*) *b-def cartesian-power-drop*
by (*metis* (*no-types*, *lifting*))
have *A*: (*augment m x*) = *a* @ (*a* @ *b*)
using *a-def augment-def b-def*
by *blast*
have *0*: *tuple-partial-image m fs* (*augment m x*) = ((*function-tuple-eval* Q_p *m fs*)

```
a) @ a @ b
    using A a-closed b-closed tuple-partial-image-eq[of a m a@b m + l augment m
x fs]
      cartesian-power-concat(1)
    by blast
  have 1: tuple-partial-image m (fs@[f]) x = ((function-tuple-eval Q_p m fs a) @ [f
a])@b
   using 0 tuple-partial-image-eq[of a m b l x fs@[f]] unfolding function-tuple-eval-def
    by (metis (no-types, lifting) a-closed a-def append-take-drop-id b-closed b-def
       list.simps(8) list.simps(9) map-append)
  have 2: tuple-partial-image m (fs@[f]) x = (function-tuple-eval Q_p m fs a) @ ((f
a)#b)
    using 1
    by (metis (no-types, lifting) append-Cons append-Nil2 append-eq-append-conv2
same-append-eq)
  have 3: tuple-partial-image m fs x = (function-tuple-eval Q_p m fs a) @ b
   using a-def b-def 2 tuple-partial-image-eq[of a m b l x fs ] assms  tuple-partial-image-def
    by blast
  have 4: twisted-partial-image n m f (tuple-partial-image m fs (augment m x)) =
                (function-tuple-eval Q_p m fs a) @ ((f a)#b)
    using twisted-partial-image-eq[of - n - m - l] 0  assms(1) assms(3) b-closed
      local.a-closed local.function-tuple-eval-closed by blast
  show ?thesis using 2 4
    by presburger
qed

definition diagonalize where
diagonalize n m S = S ∩ cartesian-product (Δ n) (carrier (Q_p^m))

lemma diagaonlize-is-semiaglebraic:
  assumes is-semialgebraic (n + n + m) S
  shows is-semialgebraic (n + n + m) (diagonalize n m S)
proof(cases m = 0)
  case True
  then have 0: carrier (Q_p^m) = {[]}
    unfolding  cartesian-power-def
    by simp
  have 1: Δ n ⊆ carrier (Q_p^{n+n})
    using Qp.cring-axioms assms diagonalE(2)
    by blast
  then have cartesian-product (Δ n) (carrier (Q_p^m)) = Δ n
    using 0 cartesian-product-empty-right[of Δ n Q_p n + n  carrier (Q_p^m)]
    by linarith
  then have diagonalize n m S = S ∩ (Δ n)
    using diagonalize-def
    by presburger
  then show ?thesis
    using intersection-is-semialg True assms diag-is-semialgebraic
    by auto
```

next
case *False*
have *is-semialgebraic* $(n + n + m)$ (*cartesian-product* $(\Delta\ n)$ (*carrier* $({Q_p}^m)$))
using *carrier-is-semialgebraic*[*of m*]
cartesian-product-is-semialgebraic[*of* $n + n\ \Delta\ n\ m$ *carrier* $({Q_p}^m)$]
diag-is-semialgebraic[*of n*] *False*
by *blast*
then show *?thesis*
using *intersection-is-semialg assms*(*1*) *diagonalize-def*
by *presburger*
qed

lemma *list-segment-take*:
assumes *length a* $\geq n$
shows *list-segment 0 n a* = *take n a*
proof−
have *0*: *length* (*list-segment 0 n a*) = *length* (*take n a*)
using *assms* **unfolding** *list-segment-def*
by (*metis* (*no-types*, *lifting*) *Groups.add-ac*(*2*) *add-diff-cancel-left'*
append-take-drop-id le-Suc-ex length-append length-drop length-map map-nth)
have $\bigwedge i.\ i < n \Longrightarrow$ *list-segment 0 n a* !*i* = *take n a* ! *i*
unfolding *list-segment-def* **using** *assms*
by (*metis add.left-neutral diff-zero nth-map-upt nth-take*)
then show *?thesis* **using** *0*
by (*metis assms diff-zero le0 list-segment-length nth-equalityI*)
qed

lemma *augment-inverse-is-semialgebraic*:
assumes *is-semialgebraic* $(n{+}n{+}l)$ *S*
shows *is-semialgebraic* $(n{+}l)$ ((*augment n* −‘ *S*) ∩ *carrier* $({Q_p}^{n+l})$)
proof−
obtain *Ps* **where** *Ps-def*: *Ps* = (*var-list-segment 0 n*)
by *blast*
obtain *Qs* **where** *Qs-def*: *Qs* = (*var-list-segment n* $(n{+}l)$)
by *blast*
obtain *Fs* **where** *Fs-def*: *Fs* = *Ps*@*Ps*@*Qs*
by *blast*
have *0*: *is-poly-tuple* $(n{+}l)$ *Ps*
by (*simp add*: *Ps-def var-list-segment-is-poly-tuple*)
have *1*: *is-poly-tuple* $(n{+}l)$ *Qs*
by (*simp add*: *Qs-def var-list-segment-is-poly-tuple*)
have *2*: *is-poly-tuple* $(n{+}l)$ (*Ps*@*Qs*)
using *Qp-is-poly-tuple-append*[*of* $n{+}l$ *Ps Qs*]
by (*metis* (*no-types*, *opaque-lifting*) *0 1 add.commute*)
have *is-poly-tuple* $(n{+}l)$ *Fs*
using *0 2 Qp-is-poly-tuple-append*[*of* $n + l$ *Ps Ps*@*Qs*] *Fs-def assms*
by *blast*
have *3*: $\bigwedge x.\ x \in$ *carrier* $({Q_p}^{n+l}) \Longrightarrow$ *augment n x* = *poly-map* $(n + l)$ *Fs x*
proof− **fix** *x* **assume** *A*: *x* ∈ *carrier* $({Q_p}^{n+l})$

```
  have 30: poly-map (n+l) Ps x = take n x
    using Ps-def  map-by-var-list-segment[of x n + l n 0]
          list-segment-take[of n x] cartesian-power-car-memE[of x Q_p n+l]
    by (simp add: A)
  have 31: poly-map (n + l) Qs x = drop n x
    using Qs-def map-by-var-list-segment-to-length[of x n + l n] A le-add1
    by blast
  have 32: poly-map (n + l) (Ps@Qs) x = take n x @ drop n x
    using poly-map-append[of x n+l Ps Qs ]
    by (metis 30 31 A append-take-drop-id)
  show augment n x = poly-map (n + l) Fs x
    using 30 32 poly-map-append
    by (metis A Fs-def poly-map-append augment-def)
qed
have 4: (augment n -` S) ∩ carrier (Q_p^{n+l}) = poly-tuple-pullback (n + l) S Fs
proof
  show augment n -` S ∩ carrier (Q_p^{n+l}) ⊆ poly-tuple-pullback (n + l) S Fs
  proof fix x assume A: x ∈ augment n -` S ∩ carrier (Q_p^{n+l})
    then have 40: augment n x ∈ S
      by blast
    have 41: augment n x ∈ carrier (Q_p^{n+n+ l})
      using 40 assms unfolding augment-def
      using is-semialgebraic-closed
      by blast
    have x ∈ carrier (Q_p^{n+l})
    proof−
      have take n x @ x ∈ carrier (Q_p^{n+n+ l})
        using augment-def A
        by (metis 41 append-take-drop-id)
      then have 0: drop n (take n x @ x) ∈ carrier (Q_p^{n+l})
        by (metis (no-types, lifting) add.assoc cartesian-power-drop)
      have drop n (take n x @ x) = x
      proof−
        have length x ≥ n
          using A
          by (metis IntD2 cartesian-power-car-memE le-add1)
        then have length (take n x) = n
          by (metis add-right-cancel append-take-drop-id
              le-add-diff-inverse length-append length-drop)
        then show ?thesis
          by (metis append-eq-conv-conj)
      qed
      then show ?thesis
        using 0
        by presburger
    qed
    then show x ∈ poly-tuple-pullback (n + l) S Fs
      using 41 3 unfolding poly-tuple-pullback-def
    by (metis (no-types, opaque-lifting) 40 add.commute cartesian-power-car-memE
```

evimageI evimage-def poly-map-apply)
qed
show *poly-tuple-pullback* $(n + l)$ *S Fs* $\subseteq$ *augment n* $-‘$ *S* $\cap$ *carrier* $({Q_p}^{n+l})$
proof fix *x* **assume** *A*: $x \in$ *poly-tuple-pullback* $(n + l)$ *S Fs*
have $x \in$ *carrier* $({Q_p}^{n+l})$
using *A* **unfolding** *poly-tuple-pullback-def* **by** *blast*
then show $x \in$ *augment n* $-‘$ *S* $\cap$ *carrier* $({Q_p}^{n+l})$
using *3*
by (*metis* (*no-types*, *lifting*) *A poly-map-apply poly-tuple-pullback-def vimage-inter-cong*)
qed
qed
then show *?thesis* **using** *assms pullback-is-semialg*[*of n* + *l Fs*]
using *poly-tuple-pullback-eq-poly-map-vimage*
unfolding *restrict-def evimage-def Fs-def*
by (*smt 4 Ex-list-of-length Fs-def Ps-def Qs-def* ‹*is-poly-tuple* $(n + l)$ *Fs*› *add.commute*
add-diff-cancel-left′ append-assoc diff-zero is-semialgebraic-closed le-add2 length-append
not-add-less1 not-gr-zero padic-fields.is-semialgebraicE padic-fields-axioms var-list-segment-length zero-le)
qed

lemma *tuple-partial-pullback-is-semialg-map-tuple-induct*:
assumes *is-semialg-map-tuple m fs*
assumes *is-semialg-function m f*
assumes *length fs* $= n$
shows *is-semialg-map-tuple m* (*fs*@[*f*])
proof(*rule is-semialg-map-tupleI*)
have *0*: *is-function-tuple* Q_p *m fs*
using *assms is-semialg-map-tuple-def*
by *blast*
show *is-function-tuple* Q_p *m* (*fs* @ [*f*])
proof(*rule is-function-tupleI*)
have *A0*: *set* (*fs* @ [*f*]) = *insert f* (*set fs*)
by *simp*
have *A1*: *set fs* $\subseteq$ *carrier* $({Q_p}^m) \to$ *carrier* Q_p
using *0 is-function-tuple-def*
by *blast*
then show *set* (*fs* @ [*f*]) $\subseteq$ *carrier* $({Q_p}^m) \to$ *carrier* Q_p
using *assms 0*
by (*metis* (*no-types*, *lifting*) *A0 is-semialg-function-closed list.simps*(*15*) *set-ConsD subset-code*(*1*))
qed
show $\bigwedge k$ *S*. *S* $\in$ *semialg-sets* (*length* (*fs* @ [*f*]) + *k*) $\Longrightarrow$
is-semialgebraic $(m + k)$ (*tuple-partial-pullback m* (*fs* @ [*f*]) *k S*)
proof− **fix** *l S*
assume *A*: *S* $\in$ *semialg-sets* (*length* (*fs* @ [*f*]) + *l*)
then have *B*: *S* $\in$ *semialg-sets* $(n + l + 1)$

```
    using assms
    by (metis (no-types, lifting) add.commute add-Suc-right add-diff-cancel-left'
        append-Nil2 diff-Suc-1 length-Suc-conv length-append)
  show is-semialgebraic (m + l) (tuple-partial-pullback m (fs @ [f]) l S)
  proof−
    obtain S0 where S0-def: S0 = tuple-partial-pullback m fs (l+1) S
      by blast
    have 0: is-semialgebraic (m + l + 1) S0
      using B assms is-semialg-map-tupleE[of m fs l + 1 S]
      by (metis S0-def add.assoc is-semialgebraicI)
    obtain S1 where S1-def: S1 = twisted-partial-pullback m m l f S0
      by blast
    then have is-semialgebraic (m + m + l) S1
      using S1-def assms(1) 0 twisted-partial-pullback-is-semialgebraic[of m f m l
S0]
      by (simp add: assms(2))
    then have L: is-semialgebraic (m + m + l) (diagonalize m l S1)
      using assms  diagaonlize-is-semiaglebraic
      by blast
    have 1: (tuple-partial-pullback m (fs @ [f]) l S)
          = (augment m −‘ (diagonalize m l S1)) ∩ carrier (Q_p^(m + l))
    proof
      show tuple-partial-pullback m (fs @ [f]) l S ⊆
                augment m −‘ diagonalize m l S1 ∩ carrier (Q_p^(m + l))

      proof fix x assume P0: x ∈ tuple-partial-pullback m (fs @ [f]) l S
        show x ∈ augment m −‘ diagonalize m l S1 ∩ carrier (Q_p^(m + l))
        proof
          show x ∈ carrier (Q_p^(m + l))
            using tuple-partial-pullback-closed P0
            by blast
          show x ∈ augment m −‘ diagonalize m l S1
          proof−
            obtain a where a-def: a = take m x
              by blast
            then have a-closed: a ∈ carrier (Q_p^m)
              using ‹x ∈ carrier (Q_p^(m + l))› le-add1 take-closed
              by blast
            obtain b where b-def: b = drop m x
              by blast
            then have b-closed: b ∈ carrier (Q_p^l)
              using ‹x ∈ carrier (Q_p^(m + l))› cartesian-power-drop
              by blast
            have x-eq: x = a@b
              using a-def b-def
              by (metis append-take-drop-id)
            have X0: a @ a @ b = augment m x
              by (metis a-def augment-def b-def)
            have a @ a @ b ∈  diagonalize m l S1
```

proof−
have *length* (*a*@*a*) = *m* + *m*
using *a-closed cartesian-power-car-memE length-append*
by *blast*
then have *take* (*m* + *m*) (*a* @ *a* @ *b*) = *a*@*a*
by (*metis append.assoc append-eq-conv-conj*)
then have *X00*: *take* (*m* + *m*) (*a* @ *a* @ *b*) ∈ Δ *m*
using *diagonalI*[*of a*@*a*] *a-def a-closed*
by (*metis append-eq-conv-conj cartesian-power-car-memE*)
then have *X01*: *a* @ *a* @ *b* ∈ *cartesian-product* (Δ *m*) (*carrier* ($Q_p{}^l$))
using *a-closed b-closed cartesian-product-memI*[*of* Δ *m* Q_p *m*+*m carrier* ($Q_p{}^l$) *l a* @ *a* @ *b*]
unfolding *diagonal-def*
by (*metis* (*no-types*, *lifting*) *X0* ‹*x* ∈ *carrier* ($Q_p{}^{m\,+\,l}$)› *augment-closed cartesian-power-drop mem-Collect-eq subsetI*)
have *X02*: *twisted-partial-image m m f* (*a* @ *a* @ *b*) = *a* @ ((*f a*)# *b*)
using *twisted-partial-image-eq*[*of a m a m b l - f*] *a-closed b-closed*
by *blast*
have *a* @ *a* @ *b* ∈ *S1*
proof−
have *twisted-partial-image m m f* (*a* @ *a* @ *b*) ∈ *S0*
proof−
have *X020*:*tuple-partial-image m fs* (*a* @ ((*f a*)# *b*))
= (*function-tuple-eval* Q_p *m fs a*)@[*f a*]@*b*
using *tuple-partial-image-eq*[*of a m* (*f a*)# *b l* + *1* - *fs*]
by (*metis* (*no-types*, *lifting*) *a-closed append-Cons append-eq-conv-conj cartesian-power-car-memE self-append-conv2 tuple-partial-image-def*)
have *X021*: (*function-tuple-eval* Q_p *m fs a*)@[*f a*]@*b* ∈ *S*
proof−
have *X0210*: (*function-tuple-eval* Q_p *m fs a*)@[*f a*]@*b* =
(*function-tuple-eval* Q_p *m* (*fs*@[*f*]) *a*)@*b*
unfolding *function-tuple-eval-def*
by (*metis* (*mono-tags*, *lifting*) *append.assoc list.simps*(*8*) *list.simps*(*9*) *map-append*)
have *X0211*: (*function-tuple-eval* Q_p *m* (*fs*@[*f*]) *a*)@*b* =
tuple-partial-image m (*fs* @ [*f*]) *x*
using *x-eq tuple-partial-image-eq*[*of a m b l x fs*@[*f*]]
by (*simp add*: *a-closed b-closed*)
have *tuple-partial-image m* (*fs* @ [*f*]) *x* ∈ *S*
using *P0 tuple-partial-pullback-memE*(*2*)
by *blast*
then show *?thesis* **using** *X0211 X0210* **by** *presburger*
qed
have *X022*: *tuple-partial-image m fs* (*twisted-partial-image m m f* (*a* @ *a* @ *b*))
= (*function-tuple-eval* Q_p *m fs a*)@[*f a*]@*b*
proof−
have *X0220*: *tuple-partial-image m fs* (*twisted-partial-image m*

```
m f (a @ a @ b)) =
                        tuple-partial-image m fs (a @ ((f a)# b))
                  using X02 by presburger
                 have X0221: tuple-partial-image m fs (twisted-partial-image m
m f (a @ a @ b)) =
                        (function-tuple-eval Q_p m fs a) @ ((f a)# b)
                  using tuple-partial-image-eq
                  by (metis X02 X020 append-Cons self-append-conv2)
                then show ?thesis
                  unfolding function-tuple-eval-def
                  by (metis X02 X020 X0221 append-same-eq)
              qed
              have X023: tuple-partial-image m fs (twisted-partial-image m m f
(a @ a @ b)) ∈ S
                using X02 X020 X021 by presburger
                    have twisted-partial-image m m f (a @ a @ b) ∈ carrier
(Q_p^m + (l+1))
              proof−
                have a @ ((f a)# b) ∈ carrier (Q_p^m + (l+1))
                 apply(rule cartesian-power-car-memI)
                        apply (metis a-closed add.commute b-closed carte-
sian-power-car-memE
                     length-Cons length-append plus-1-eq-Suc)
                proof−
                  have f ∈ carrier (Q_p^m) → carrier Q_p
                    using assms(2) is-semialg-function-closed by blast
                  then have f a ∈ carrier Q_p
                    using a-closed assms
                    by blast
                  then show set (a @ f a # b) ⊆ carrier Q_p
                    using  assms a-closed b-closed
                by (meson cartesian-power-car-memE'' cartesian-power-concat(1)
cartesian-power-cons)
                qed
                then show ?thesis
                  using X02
                  by presburger
              qed
              then show ?thesis
                using S0-def X023 tuple-partial-pullback-def[of m fs l+1 S ]
                by blast
            qed
            then show ?thesis using X02 S1-def twisted-partial-pullback-def
                  by (metis (no-types, lifting) X0 ‹x ∈ carrier (Q_p^m + l)›
augment-closed
                drop-drop local.partial-image-def twisted-partial-image-def
                twisted-partial-pullback-memI)
          qed
          then show ?thesis using X01 diagonalize-def[of m l S1]
```

by *blast*
qed
then show *?thesis*
by (*metis X0 vimageI*)
qed
qed
qed
show *augment m −‘ diagonalize m l S1 ∩ carrier* $(Q_p{}^{m+l}) \subseteq$ *tuple-partial-pullback m* (*fs @* [*f*]) *l S*
proof
fix *x*
assume *A*: *x ∈ augment m −‘ diagonalize m l S1 ∩ carrier* $(Q_p{}^{m+l})$
then have *X0*: *x ∈ carrier* $(Q_p{}^{m+l})$
by *blast*
obtain *a* **where** *a-def*: *a = take m x*
by *blast*
then have *a-closed*: *a ∈ carrier* $(Q_p{}^{m})$
using *X0 le-add1 take-closed* **by** *blast*
obtain *b* **where** *b-def*: *b = drop m x*
by *blast*
then have *a-closed*: *b ∈ carrier* $(Q_p{}^{l})$
using *X0 cartesian-power-drop*
by *blast*
have *X1*: *augment m x = a@a@b*
using *a-def augment-def b-def*
by *blast*
have *X2*: *a@a@b ∈ diagonalize m l S1*
using *A X1*
by (*metis Int-iff vimage-eq*)
have *X3*: *a@a@b ∈ S1*
using *X2 diagonalize-def*
by *blast*
have *X4*: *twisted-partial-image m m f* (*a@a@b*) *∈ S0*
using *X3 S1-def twisted-partial-pullback-memE*(*2*)
by *blast*
have *X5*: *a@*((*f a*)#*b*) *∈ S0*
using *X4 twisted-partial-image-eq*[*of a m a m b l - f*]
by (*metis X0 a-closed a-def le-add1 take-closed*)
have *X6*: *tuple-partial-image m fs* (*a@*((*f a*)#*b*)) *∈ S*
using *S0-def X5 tuple-partial-pullback-memE*(*2*)
by *blast*
have *X7*: ((*function-tuple-eval* Q_p *m fs a*)*@*((*f a*)#*b*)) *∈ S*
using *X6* **using** *tuple-partial-image-eq*
by (*metis X0 a-def append-eq-conv-conj cartesian-power-car-memE le-add1 take-closed tuple-partial-image-def*)
have *X8*: ((*function-tuple-eval* Q_p *m fs a*)*@*((*f a*)#*b*)) =
tuple-partial-image m (*fs @* [*f*]) *x*
proof−
have *X80*: *tuple-partial-image m* (*fs @* [*f*]) *x* = (*function-tuple-eval* Q_p

m (*fs*@[*f*]) *a*)@*b*
using *tuple-partial-image-def a-def b-def*
by *blast*
then show *?thesis* **unfolding** *function-tuple-eval-def*
by (*metis* (*no-types*, *lifting*) *append-Cons append-eq-append-conv2 list.simps*(*9*) *map-append self-append-conv2*)
qed
show *x* ∈ *tuple-partial-pullback m* (*fs* @ [*f*]) *l S*
using *X8 X7 tuple-partial-pullback-def*
by (*metis X0* ‹*is-function-tuple* Q_p *m* (*fs* @ [*f*])›
tuple-partial-image-def tuple-partial-pullback-memI)
qed
qed
then show *?thesis*
using *augment-inverse-is-semialgebraic*
by (*simp add*: *L*)
qed
qed
qed

lemma *singleton-tuple-partial-pullback-is-semialg-map-tuple*:
assumes *is-semialg-function-tuple m fs*
assumes *length fs* = *1*
shows *is-semialg-map-tuple m fs*
proof(*rule is-semialg-map-tupleI*)
show *is-function-tuple* Q_p *m fs*
by (*simp add*: *assms*(*1*) *semialg-function-tuple-is-function-tuple*)
show ⋀*k S*. *S* ∈ *semialg-sets* (*length fs* + *k*) ⟹ *is-semialgebraic* (*m* + *k*) (*tuple-partial-pullback m fs k S*)
proof− **fix** *k S* **assume** *A*: *S* ∈ *semialg-sets* (*length fs* + *k*)
show *is-semialgebraic* (*m* + *k*) (*tuple-partial-pullback m fs k S*)
proof−
obtain *f* **where** *f-def*: *fs* = [*f*]
using *assms*
by (*metis One-nat-def length-0-conv length-Suc-conv*)
have *0*: *is-semialg-function m f*
using *f-def assms is-semialg-function-tupleE'*[*of m fs f*]
by *simp*
have *1*: ⋀*x*. *tuple-partial-image m fs x* = *partial-image m f x*
unfolding *function-tuple-eval-def tuple-partial-image-def partial-image-def*
by (*metis* (*no-types*, *lifting*) *append-Cons append-Nil2 append-eq-append-conv-if f-def list.simps*(*8*) *list.simps*(*9*))
have *2*: *tuple-partial-pullback m fs k S* = *partial-pullback m f k S*
proof
show *tuple-partial-pullback m fs k S* ⊆ *partial-pullback m f k S*
using *1* **unfolding** *tuple-partial-pullback-def partial-pullback-def evimage-def*
by (*metis* (*no-types*, *lifting*) *set-eq-subset vimage-inter-cong*)
show *partial-pullback m f k S* ⊆ *tuple-partial-pullback m fs k S*

using *1* **unfolding** *tuple-partial-pullback-def partial-pullback-def evimage-def*
by *blast*
qed
then show *?thesis*
by (*metis 0 A assms*(*2*) *is-semialg-functionE is-semialgebraicI*)
qed
qed
qed

lemma *empty-tuple-partial-pullback-is-semialg-map-tuple*:
assumes *is-semialg-function-tuple m fs*
assumes *length fs = 0*
shows *is-semialg-map-tuple m fs*
apply(*rule is-semialg-map-tupleI*)
using *assms*(*1*) *semialg-function-tuple-is-function-tuple* **apply** *blast*
proof−
fix *k S* **assume** *A*: *S* ∈ *semialg-sets* (*length fs* + *k*)
then have *0*: *is-semialgebraic k S*
by (*metis add.left-neutral assms*(*2*) *is-semialgebraicI*)
have *1*: *tuple-partial-pullback m fs k S* = *cartesian-product* (*carrier* (${Q_p}^m$)) *S*
proof
have *1*: ⋀*x*. *function-tuple-eval* Q_p *m fs* (*take m x*) = []
using *assms* **unfolding** *function-tuple-eval-def*
by *blast*
show *tuple-partial-pullback m fs k S* ⊆ *cartesian-product* (*carrier* (${Q_p}^m$)) *S*
apply(*rule subsetI*) **apply**(*rule cartesian-product-memI*[*of carrier* (${Q_p}^m$) Q_p *m S k*])
apply *blast* **using** *0 is-semialgebraic-closed* **apply** *blast*
using *0 assms* **unfolding** *1 tuple-partial-pullback-def tuple-partial-image-def*
apply (*meson IntD2 le-add1 take-closed*)
by (*metis append-Nil evimageD evimage-def*)
have *2*: *cartesian-product* (*carrier* (${Q_p}^m$)) *S* ⊆ *carrier* (${Q_p}^{m + k}$)
using *is-semialgebraic-closed*[*of k S*] *0 assms cartesian-product-closed*[*of carrier* (${Q_p}^m$) Q_p *m S k*] **by** *blast*
show *cartesian-product* (*carrier* (${Q_p}^m$)) *S* ⊆ *tuple-partial-pullback m fs k S*
apply(*rule subsetI*) **apply**(*rule tuple-partial-pullback-memI*)
using *2* **apply** *blast*
using *assms semialg-function-tuple-is-function-tuple* **apply** *blast*
unfolding *1*
by (*metis carrier-is-semialgebraic cartesian-product-memE*(*2*) *is-semialgebraic-closed self-append-conv2*)
qed
show *is-semialgebraic* (*m* + *k*) (*tuple-partial-pullback m fs k S*)
unfolding *1*
using *0 car-times-semialg-is-semialg* **by** *blast*
qed

lemma *tuple-partial-pullback-is-semialg-map-tuple*:

assumes *is-semialg-function-tuple m fs*
shows *is-semialg-map-tuple m fs*
proof−
have $\bigwedge n\ fs.$ *is-semialg-function-tuple m fs* $\wedge$ *length fs* $= n \Longrightarrow$ *is-semialg-map-tuple m fs*
proof− **fix** *n*
show $\bigwedge fs.$ *is-semialg-function-tuple m fs* $\wedge$ *length fs* $= n \Longrightarrow$ *is-semialg-map-tuple m fs*
apply(*induction n*)
using *singleton-tuple-partial-pullback-is-semialg-map-tuple empty-tuple-partial-pullback-is-semialg-map-tup*
apply *blast*
proof−
fix *n fs*
assume *IH*: ($\bigwedge fs.$ *is-semialg-function-tuple m fs* $\wedge$ *length fs* $= n \Longrightarrow$ *is-semialg-map-tuple m fs*)
assume *A*: *is-semialg-function-tuple m fs* $\wedge$ *length fs* $=$ *Suc n*
then obtain *gs f* **where** *gs-f-def*: *fs* $=$ *gs*@[*f*]
by (*metis length-Suc-conv list.discI rev-exhaust*)
have *gs-length*: *length gs* $= n$
using *gs-f-def*
by (*metis A length-append-singleton nat.inject*)
have *0*: *set gs* $\subseteq$ *set fs*
by (*simp add*: *gs-f-def subsetI*)
have *1*: *is-semialg-function-tuple m gs*
apply(*rule is-semialg-function-tupleI*)
using *0 A gs-f-def is-semialg-function-tupleE'*[*of m fs*]
by *blast*
then have *2*: *is-semialg-map-tuple m gs*
using *IH gs-length*
by *blast*
have *3*: *is-semialg-function m f*
using *gs-f-def A*
by (*metis gs-length is-semialg-function-tupleE lessI nth-append-length*)
then show *is-semialg-map-tuple m fs*
using *assms 2 gs-f-def tuple-partial-pullback-is-semialg-map-tuple-induct*
by *blast*
qed
qed
then show *?thesis*
using *assms* **by** *blast*
qed

13.11.2 Semialgebraic Functions are Closed under Composition with Semialgebraic Tuples

lemma *function-tuple-comp-partial-pullback*:
assumes *is-semialg-function-tuple m fs*
assumes *length fs* $= n$
assumes *is-semialg-function n f*

assumes $S \subseteq$ *carrier* $({Q_p}^{1+k})$
shows *partial-pullback m* (*function-tuple-comp* Q_p *fs f*) *k S* =
tuple-partial-pullback m fs k (*partial-pullback n f k S*)
proof−
have *0*: $\bigwedge x$. *partial-image m* (*function-tuple-comp* Q_p *fs f*) *x* =
partial-image n f (*tuple-partial-image m fs x*)
unfolding *partial-image-def function-tuple-comp-def tuple-partial-image-def*
using *comp-apply*[*of f function-tuple-eval* Q_p *0 fs*]
unfolding *function-tuple-eval-def*
proof −
fix *x* :: ((*nat* ⇒ *int*) × (*nat* ⇒ *int*)) *set list*
assume *a1*: $\bigwedge x$. (*f* ∘ (λ*x*. *map* (λ*f*. *f x*) *fs*)) *x* = *f* (*map* (λ*f*. *f x*) *fs*)
have *f2*: ∀*f rs*. *drop n* (*map f fs* @ (*rs*::((*nat* ⇒ *int*) × (*nat* ⇒ *int*)) *set list*))
= *rs*
by (*simp add*: *assms*(*2*))
have ∀*f rs*. *take n* (*map f fs* @ (*rs*::((*nat* ⇒ *int*) × (*nat* ⇒ *int*)) *set list*)) =
map f fs
by (*simp add*: *assms*(*2*))
then show (*f* ∘ (λ*rs*. *map* (λ*f*. *f rs*) *fs*)) (*take m x*) # *drop m x* =
f (*take n* (*map* (λ*f*. *f* (*take m x*)) *fs* @ *drop m x*)) # *drop n* (*map* (λ*f*.
f (*take m x*)) *fs* @ *drop m x*)
using *f2 a1* **by** *presburger*
qed
show *partial-pullback m* (*function-tuple-comp* Q_p *fs f*) *k S* =
tuple-partial-pullback m fs k (*partial-pullback n f k S*)
proof
show *partial-pullback m* (*function-tuple-comp* Q_p *fs f*) *k S* ⊆ *tuple-partial-pullback*
m fs k (*partial-pullback n f k S*)
proof fix *x* **assume** *A*: *x* ∈ *partial-pullback m* (*function-tuple-comp* Q_p *fs f*) *k*
S
then have *1*: *partial-image m* (*function-tuple-comp* Q_p *fs f*) *x* ∈ *S*
using *partial-pullback-memE*(*2*) **by** *blast*
have *2*: *partial-image n f* (*tuple-partial-image m fs x*) ∈ *S*
using *0 1*
by *presburger*
have *3*: *x* ∈ *carrier* $({Q_p}^{m} + k)$
using *A assms*
by (*metis partial-pullback-memE*(*1*))
have *4*: *tuple-partial-image m fs x* ∈ *partial-pullback n f k S*
apply(*rule partial-pullback-memI*)
apply (*metis 0 3 add-cancel-left-left assms*(*1*) *assms*(*2*) *cartesian-power-drop*
drop0
list.inject local.partial-image-def not-gr-zero semialg-function-tuple-is-function-tuple
tuple-partial-image-closed)
by (*metis 2 local.partial-image-def*)
show *x* ∈ *tuple-partial-pullback m fs k* (*partial-pullback n f k S*)
apply(*rule tuple-partial-pullback-memI*)
apply (*simp add*: *3*)
using *assms*(*1*) *semialg-function-tuple-is-function-tuple* **apply** *blast*

```
      by (metis 4 tuple-partial-image-def)
    qed
    show  tuple-partial-pullback m fs k (partial-pullback n f k S) ⊆ partial-pullback
m (function-tuple-comp Q_p fs f) k S
    proof fix x assume A: x ∈ tuple-partial-pullback m fs k (partial-pullback n f k
S)
      show x ∈ partial-pullback m (function-tuple-comp Q_p fs f) k S
      proof−
        have partial-image n f (tuple-partial-image m fs x) ∈ S
          using A partial-pullback-memE(2) tuple-partial-pullback-memE(2)
          by blast
        show ?thesis
          apply(rule partial-pullback-memI)
           apply (meson A subset-eq tuple-partial-pullback-closed)
          by (metis 0 ‹local.partial-image n f (tuple-partial-image m fs x) ∈ S›
              local.partial-image-def)
      qed
    qed
  qed
qed

lemma semialg-function-tuple-comp:
  assumes is-semialg-function-tuple m fs
  assumes length fs = n
  assumes is-semialg-function n f
  shows is-semialg-function m (function-tuple-comp Q_p fs f)
proof(rule is-semialg-functionI)
  show function-tuple-comp Q_p fs f ∈ carrier ((Q_p)^m) → carrier Q_p
    using function-tuple-comp-closed[of f Q_p n fs] assms(1) assms(2)
      assms(3) is-semialg-function-closed semialg-function-tuple-is-function-tuple
    by blast
 show ⋀k S. S ∈ semialg-sets (1 + k) ⟹ is-semialgebraic (m + k) (partial-pullback
m (function-tuple-comp Q_p fs f) k S)
  proof− fix k S
    assume A0: S ∈ semialg-sets (1 + k)
    show is-semialgebraic (m + k) (partial-pullback m (function-tuple-comp Q_p fs
f) k S)
    proof−
      have 0: partial-pullback m (function-tuple-comp Q_p fs f) k S =
            tuple-partial-pullback m fs k (partial-pullback n f k S)
        using  function-tuple-comp-partial-pullback[of m fs n f S k] assms
          ‹S ∈ semialg-sets (1 + k)› is-semialgebraicI is-semialgebraic-closed
        by blast
      have 1: is-semialgebraic (n + k) (partial-pullback n f k S)
        using assms A0 is-semialg-functionE is-semialgebraicI
        by blast
     have 2: is-semialgebraic (m + k) (tuple-partial-pullback m fs k (partial-pullback
n f k S))
        using 1 0 assms tuple-partial-pullback-is-semialg-map-tuple[of m fs]
```

is-semialg-map-tupleE[*of m fs k partial-pullback n f k S*]
by *blast*
then show *?thesis*
using *0*
by *simp*
qed
qed
qed

13.11.3 Algebraic Operations on Semialgebraic Functions

Defining the set of extensional semialgebraic functions

definition *Qp-add-fun* **where**
Qp-add-fun xs = *xs*!*0* $\oplus_{Q_p}$ *xs*!*1*

definition *Qp-mult-fun* **where**
Qp-mult-fun xs = *xs*!*0* $\otimes$ *xs*!*1*

Inversion function on first coordinates of Qp tuples. Arbitrarily redefined at 0 to map to 0

definition *Qp-invert* **where**
Qp-invert xs = (*if* ((*xs*!*0*) = **0**) *then* **0** *else* (*inv* (*xs*!*0*)))

Addition is semialgebraic

lemma *addition-is-semialg*:
is-semialg-function 2 Qp-add-fun
proof−
have *0*: $\bigwedge x.$ *x* ∈ *carrier* (${Q_p}^2$) ⟹ *Qp-add-fun x* = *Qp-ev* (*pvar* Q_p *0* $\oplus_{Q_p[\mathcal{X}_2]}$ *pvar* Q_p *1*) *x*
proof− **fix** *x* **assume** *A*: *x* ∈ *carrier* (${Q_p}^2$)
have *Qp-ev* (*pvar* Q_p *0* $\oplus_{Q_p[\mathcal{X}_2]}$ *pvar* Q_p *1*) *x* = (*Qp-ev* (*pvar* Q_p *0*) *x*) $\oplus_{Q_p}$ (*Qp-ev* (*pvar* Q_p *1*) *x*)
by (*metis A One-nat-def eval-at-point-add pvar-closed less-Suc-eq numeral-2-eq-2*)
then show *Qp-add-fun x* = *Qp-ev* (*pvar* Q_p *0* $\oplus_{Q_p[\mathcal{X}_2]}$ *pvar* Q_p *1*) *x*
by (*metis A Qp-add-fun-def add-vars-def add-vars-rep one-less-numeral-iff pos2 semiring-norm*(*76*))
qed
then have *1*: *restrict Qp-add-fun* (*carrier* (${Q_p}^2$)) =
restrict (*Qp-ev* (*pvar* Q_p *0* $\oplus_{Q_p[\mathcal{X}_2]}$ *pvar* Q_p *1*)) (*carrier* (${Q_p}^2$))
by (*meson restrict-ext*)
have *is-semialg-function 2* (*Qp-ev* (*pvar* Q_p *0* $\oplus_{Q_p[\mathcal{X}_2]}$ *pvar* Q_p *1*))
using *poly-is-semialg*[*of pvar* Q_p *0* $\oplus_{Q_p[\mathcal{X}_2]}$ *pvar* Q_p *1*]
by (*meson MP.add.m-closed local.pvar-closed one-less-numeral-iff pos2 semiring-norm*(*76*))
then show *?thesis*
using *1 semialg-function-on-carrier*[*of 2 Qp-add-fun Qp-ev* (*pvar* Q_p *0* $\oplus_{Q_p[\mathcal{X}_2]}$ *pvar* Q_p *1*)]

semialg-function-on-carrier
by *presburger*
qed

Multiplication is semialgebraic:

lemma *multiplication-is-semialg*:
is-semialg-function 2 Qp-mult-fun
proof−
have *0*: $\bigwedge x.\ x \in carrier\ ({Q_p}^2) \Longrightarrow Qp\text{-}mult\text{-}fun\ x = Qp\text{-}ev\ (pvar\ Q_p\ 0 \otimes_{Q_p[\mathcal{X}_2]} pvar\ Q_p\ 1)\ x$
proof− **fix** *x* **assume** *A*: $x \in carrier\ ({Q_p}^2)$
have $Qp\text{-}ev\ (pvar\ Q_p\ 0 \otimes_{Q_p[\mathcal{X}_2]} pvar\ Q_p\ 1)\ x =$
$(Qp\text{-}ev\ (pvar\ Q_p\ 0)\ x) \otimes (Qp\text{-}ev\ (pvar\ Q_p\ 1)\ x)$
by (*metis A One-nat-def eval-at-point-mult pvar-closed less-Suc-eq numeral-2-eq-2*)
then show $Qp\text{-}mult\text{-}fun\ x = Qp\text{-}ev\ (pvar\ Q_p\ 0 \otimes_{Q_p[\mathcal{X}_2]} pvar\ Q_p\ 1)\ x$
by (*metis A Qp-mult-fun-def mult-vars-def mult-vars-rep*
one-less-numeral-iff pos2 semiring-norm(*76*))
qed
then have *1*: $restrict\ Qp\text{-}mult\text{-}fun\ (carrier\ ({Q_p}^2)) =$
$restrict\ (Qp\text{-}ev\ (pvar\ Q_p\ 0 \otimes_{Q_p[\mathcal{X}_2]} pvar\ Q_p\ 1))\ (carrier\ ({Q_p}^2))$
by (*meson restrict-ext*)
have $is\text{-}semialg\text{-}function\ 2\ (Qp\text{-}ev\ (pvar\ Q_p\ 0 \otimes_{Q_p[\mathcal{X}_2]} pvar\ Q_p\ 1))$
using $poly\text{-}is\text{-}semialg[of\ pvar\ Q_p\ 0 \otimes_{Q_p[\mathcal{X}_2]} pvar\ Q_p\ 1]$
by (*meson MP.m-closed local.pvar-closed one-less-numeral-iff pos2 semiring-norm*(*76*))
thus *?thesis*
using $1\ semialg\text{-}function\text{-}on\text{-}carrier[of\ 2\ Qp\text{-}mult\text{-}fun\ Qp\text{-}ev\ (pvar\ Q_p\ 0 \otimes_{Q_p[\mathcal{X}_2]} pvar\ Q_p\ 1)]$
semialg-function-on-carrier
by *presburger*
qed

Inversion is semialgebraic:

lemma(**in** *field*) *field-nat-pow-inv*:
assumes $a \in carrier\ R$
assumes $a \neq \mathbf{0}$
shows $inv\ (a\ [\hat{}]\ (n::nat)) = (inv\ a)\ [\hat{}]\ (n :: nat)$
apply(*induction n*)
using *inv-one local.nat-pow-0* **apply** *presburger*
using *assms nat-pow-of-inv*
by (*metis Units-one-closed field-inv*(*2*) *field-inv*(*3*) *unit-factor*)

lemma *Qp-invert-basic-semialg*:
assumes *is-basic-semialg* $(1 + k)$ *S*
shows *is-semialgebraic* $(1 + k)$ (*partial-pullback 1 Qp-invert k S*)
proof−
obtain *P n* **where** *P-n-def*: $(n::nat) \neq 0 \wedge P \in carrier\ (Q_p[\mathcal{X}_{1+k}]) \wedge S = basic\text{-}semialg\text{-}set\ (1+k)\ n\ P \wedge P \in carrier\ (Q_p[\mathcal{X}_{1+k}])$

using *assms is-basic-semialg-def*
by *meson*
obtain *d*::*nat* **where** *d-def*: $d = deg\ (coord\text{-}ring\ Q_p\ k)\ (to\text{-}univ\text{-}poly\ (Suc\ k)\ 0\ P)$
by *auto*
obtain *l* **where** *l-def*: $l = ((-\ d)\ mod\ n)$
by *blast*
have *10*: $n > 0$
using *P-n-def*
by *blast*
have *11*: $l \geq 0$
using *10* **by** (*simp add*: *l-def*)
have *1*: *int n dvd l* + *int d*
by (*simp add*: *l-def ac-simps mod-0-imp-dvd mod-add-right-eq*)
then obtain *m*::*int* **where** *m-def*: *l* + *int d* = *int n* ∗ *m* **..**
with *10* **have** ‹*m* = (*l* + *d*) *div n*›
by *simp*
with *10 11* **have** *2*: $m \geq 0$
by (*simp add*: *div-int-pos-iff*)
obtain *N* **where** *N-def*: $N = m * n$
by *blast*
from *11* **have** *3*: $N \geq d$
using *m-def* **by** (*simp add*: *N-def ac-simps*)
have *4*: $deg\ (coord\text{-}ring\ Q_p\ k)\ (to\text{-}univ\text{-}poly\ (Suc\ k)\ 0\ P) \leq nat\ N$
using *d-def N-def 3*
by *linarith*
have *5*: $P \in carrier\ (coord\text{-}ring\ Q_p\ (Suc\ k))$
by (*metis P-n-def plus-1-eq-Suc*)
have *6*: $\exists q{\in}carrier\ (coord\text{-}ring\ Q_p\ (Suc\ k)).$
$\forall x{\in}carrier\ Q_p - \{\mathbf{0}\}.\ \forall a{\in}carrier\ ({Q_p}^k).\ Qp\text{-}ev\ q\ (insert\text{-}at\text{-}index\ a\ x\ 0) = (x[\widehat{}]nat\ N) \otimes Qp\text{-}ev\ P\ (insert\text{-}at\text{-}index\ a\ (inv\ x)\ 0)$
using *3 4 d-def to-univ-poly-one-over-poly''*[*of 0 k P nat N*] *5 Qp.field-axioms*
by *blast*
obtain *q* **where** *q-def*: $q \in carrier\ (coord\text{-}ring\ Q_p\ (Suc\ k)) \wedge (\forall\ x \in carrier\ Q_p - \{\mathbf{0}\}.\ (\forall\ a \in\ carrier\ ({Q_p}^k).$
$eval\text{-}at\text{-}point\ Q_p\ (insert\text{-}at\text{-}index\ a\ x\ 0)\ q = (x[\widehat{}]\ (nat\ N)) \otimes (eval\text{-}at\text{-}point\ Q_p\ (insert\text{-}at\text{-}index\ a\ (inv\ x)\ 0)\ P)))$
using *6*
by *blast*
obtain *T* **where** *T-def*: $T = basic\text{-}semialg\text{-}set\ (1{+}k)\ n\ q$
by *auto*
have *is-basic-semialg* (*1* + *k*) *T*
proof−
have $q \in carrier\ (\ Q_p[\mathcal{X}_{Suc\ k}])$
using *q-def*
by *presburger*
then show *?thesis*
using *T-def is-basic-semialg-def*
by (*metis P-n-def plus-1-eq-Suc*)

qed
then have *T-semialg*: *is-semialgebraic* (*1+k*) *T*
using *T-def basic-semialg-is-semialg*[*of 1+k T*] *is-semialgebraicI*
by *blast*
obtain *Nz* **where** *Nz-def*: $Nz = \{xs \in carrier\ ({Q_p}^{Suc\ k}).\ xs!0 \neq \mathbf{0}\}$
by *blast*
have *Nz-semialg*: *is-semialgebraic* (*1+k*) *Nz*
proof−
obtain *Nzc* **where** *Nzc-def*: $Nzc = \{xs \in carrier\ ({Q_p}^{Suc\ k}).\ xs!0 = \mathbf{0}\}$
by *blast*
have *0*: *Nzc* = *zero-set* Q_p (*Suc k*) (*pvar* Q_p *0*)
unfolding *zero-set-def*
using *Nzc-def*
by (*metis* (*no-types*, *lifting*) *Collect-cong eval-pvar zero-less-Suc*)
have *1*: *is-algebraic* Q_p (*1+k*) *Nzc*
using *0 pvar-closed*[*of*]
by (*metis is-algebraicI' plus-1-eq-Suc zero-less-Suc*)
then have *2*: *is-semialgebraic* (*1+k*) *Nzc*
using *is-algebraic-imp-is-semialg* **by** *blast*
have *3*: *Nz* = *carrier* $({Q_p}^{Suc\ k})$ − *Nzc*
using *Nz-def Nzc-def*
by *blast*
then show *?thesis*
using *2*
by (*simp add*: *complement-is-semialg*)
qed
have *7*: (*partial-pullback 1 Qp-invert k S*) ∩ *Nz* = *T* ∩ *Nz*
proof
show *partial-pullback 1 Qp-invert k S* ∩ *Nz* ⊆ *T* ∩ *Nz*
proof fix *c* **assume** *A*: *c* ∈ *partial-pullback 1 Qp-invert k S* ∩ *Nz*
show *c* ∈ *T* ∩ *Nz*
proof−
have *c-closed*: *c* ∈ *carrier* $({Q_p}^{1+k})$
using *A partial-pullback-closed*[*of 1 Qp-invert k S*]
by *blast*
obtain *x a* **where** *xa-def*: *c* = (*x*#*a*)
using *c-closed*
by (*metis Suc-eq-plus1 add.commute cartesian-power-car-memE length-Suc-conv*)
have *x-closed*: *x* ∈ *carrier* Q_p
using *xa-def c-closed*
by (*metis* (*no-types*, *lifting*) *append-Cons cartesian-power-decomp list.inject Qp.to-R1-to-R Qp.to-R-pow-closed*)
have *a-closed*: *a* ∈ *carrier* $({Q_p}^{k})$
using *xa-def c-closed*
by (*metis One-nat-def cartesian-power-drop drop0 drop-Suc-Cons*)
have *0*: *c* ∈ *Nz*
using *A* **by** *blast*
then have *x* ≠ **0**
using *Nz-def xa-def*

by (*metis* (*mono-tags*, *lifting*) *mem-Collect-eq nth-Cons-0*)
have *1*: *Qp-invert* [*x*] = *inv x*
unfolding *Qp-invert-def*
by (*metis* ‹*x* ≠ **0**› *nth-Cons-0*)
have *2*: *partial-image 1 Qp-invert c* ∈ *S*
using *A partial-pullback-memE*[*of c 1 Qp-invert k S*]
by *blast*
have *3*: *inv x* # *a* ∈ *S*
proof−
have *30*: [*x*] = *take 1 c*
by (*simp add*: *xa-def*)
have *31*: *a* = *drop 1 c*
by (*simp add*: *xa-def*)
show *?thesis*
using *1 30 31 partial-image-def*[*of 1 Qp-invert c*] *xa-def 2*
by *metis*
qed
obtain *y* **where** *y-def*: *y* ∈ *carrier* Q_p ∧ *eval-at-point* Q_p (*inv x* # *a*) *P* = *y* [^] *n*
using *3 P-n-def basic-semialg-set-memE*(*2*)
by *blast*
then have *4*: *x* [^] (*nat N*) ⊗ *eval-at-point* Q_p (*inv x* # *a*) *P*
= *x* [^] (*nat N*) ⊗ *y* [^] *n*
by *presburger*
have *5*: *x* [^] (*nat N*) ⊗ *y* [^] *n* = ((*x*[^]*m*)⊗*y*)[^]*n*
proof−
have *50*: *x* [^] (*N*) ⊗ *y* [^] *n* = *x* [^] (*m*∗*n*) ⊗ *y* [^] *n*
using *N-def* **by** *blast*
have *51*: *x* [^] (*m*∗*n*) = (*x*[^]*m*)[^]*n*
using *Qp-int-nat-pow-pow* ‹*x* ≠ **0**› *not-nonzero-Qp x-closed*
by *metis*
have *52*: *x* [^] (*m*∗*n*)⊗ *y* [^] *n* = ((*x*[^]*m*) ⊗ *y*) [^] *n*
proof−
have *0*: *x* [^] (*m*∗*n*)⊗ *y* [^] *n*= (*x*[^]*m*)[^]*n* ⊗ (*y*[^] *n*)
using *51* **by** *presburger*
have *1*: (*x*[^]*m*)[^]*n* ⊗ (*y*[^] *n*) = ((*x*[^]*m*) ⊗ *y*) [^] *n*
apply(*induction n*)
using *Qp.nat-pow-0 Qp.one-closed Qp.r-one* **apply** *presburger*
using *x-closed y-def*
by (*metis Qp.nat-pow-distrib Qp.nonzero-closed Qp-int-pow-nonzero* ‹*x* ≠ **0**› *not-nonzero-Qp*)
then show *?thesis*
using *0* **by** *blast*
qed
have *53*: *x* [^] *N* = *x* [^] (*nat N*)
using *11 m-def N-def* **by** (*simp add*: *ac-simps*)
then show *?thesis*
using *50 52*
by *presburger*

```
      qed
      have 6: x [^] (nat N) ⊗ eval-at-point Q_p (inv x # a) P = ((x[^]m)⊗y)[^]n
        using 4 5
        by blast
      have 7: eval-at-point Q_p c q = ((x[^]m)⊗y)[^]n
      proof−
        have 70: (insert-at-index a (inv x) 0) = inv x # a
          using insert-at-index.simps
            by (metis (no-types, lifting) append-eq-append-conv2 append-same-eq
append-take-drop-id drop0 same-append-eq)
        have 71: (insert-at-index a x) 0 =  x # a
          by simp
        then show ?thesis using 6 q-def
          by (metis 70 DiffI ‹x ≠ 0› a-closed empty-iff insert-iff x-closed xa-def)
      qed
      have 8: (x[^]m)⊗y ∈ carrier Q_p
      proof−
        have 80: x[^]m ∈ carrier Q_p
        using ‹x ≠ 0› x-closed Qp-int-pow-nonzero[of x m] unfolding nonzero-def
          by blast
        then show ?thesis
          using y-def by blast
      qed
      then have c ∈ T
        using T-def basic-semialg-set-def 7 c-closed by blast
      then show ?thesis
        by (simp add: ‹c ∈ T› 0)
    qed
  qed
  show T ∩ Nz ⊆ partial-pullback 1 Qp-invert k S ∩ Nz
  proof fix x assume A: x ∈ T ∩ Nz
    show  x ∈ partial-pullback 1 Qp-invert k S ∩ Nz
    proof−
      have  x ∈ partial-pullback 1 Qp-invert k S
      proof(rule partial-pullback-memI)
        show x-closed: x ∈ carrier (Q_p^(1+k))
          using T-def A
          by (meson IntD1 basic-semialg-set-memE(1))
        show Qp-invert (take 1 x) # drop 1 x ∈ S
        proof−
          have 00: x!0 ≠ 0
            using A Nz-def
            by blast
          then have 0: Qp-invert (take 1 x) # drop 1 x = inv (x!0) # drop 1 x
            unfolding Qp-invert-def
            by (smt One-nat-def lessI nth-take)
          have drop 1 x ∈ carrier (Q_p^k)
            using ‹x ∈ carrier (Q_p^(1+k))› cartesian-power-drop by blast
          obtain a where a-def: a = (x!0)
```

by *blast*
have *a-closed*: $a \in$ *carrier* Q_p
using *00 a-def A Nz-def cartesian-power-car-memE'*[*of x* Q_p *Suc k 0*] *inv-in-frac*(*1*)
by *blast*
have *a-nz*: $a \neq \mathbf{0}$
using *a-def Nz-def A*
by *blast*
obtain *b* **where** *b-def*: $b =$ *drop 1 x*
by *blast*
have *b-closed*: $b \in$ *carrier* $({Q_p}^k)$
using *b-def A Nz-def* ‹*drop 1 x* ∈ *carrier* $({Q_p}^k)$›
by *blast*
have *abx*: $x = a\#b$
using *a-def b-def x-closed*
by (*metis* (*no-types*, *lifting*) *One-nat-def append-Cons append-Nil append-eq-conv-conj cartesian-power-car-memE cartesian-power-decomp lessI nth-take Qp.to-R1-to-R*)
have *1*: *Qp-invert* (*take 1 x*) # *drop 1 x* = (*inv a*)#*b*
using *0 a-def b-def*
by *blast*
have *22*: *eval-at-point* Q_p (*insert-at-index b a 0*) *q* =
($a[\widehat{}]$ (*nat N*)) ⊗ (*eval-at-point* Q_p (*insert-at-index b* (*inv a*) *0*) *P*)
using *q-def a-closed a-nz b-closed*
by *blast*
obtain *c* **where** *c-def*: *c* ∈ *carrier* Q_p ∧ *Qp-ev q x* = ($c[\widehat{}]n$)
using *A T-def* **unfolding** *basic-semialg-set-def*
by *blast*
obtain *c'* **where** *c'-def*: *c'* = (*inv a*)$[\widehat{}]m$ ⊗ *c*
by *blast*
have *c'-closed*: *c'* ∈ *carrier* Q_p
using *c-def a-def a-closed a-nz Qp-int-pow-nonzero nonzero-def c'-def inv-in-frac*(*3*) *Qp.m-closed Qp.nonzero-closed* **by** *presburger*
have *3*: (*eval-at-point* Q_p ((*inv a*) # *b*) *P*) = ($c'[\widehat{}]n$)
proof−
have *30*: *x* = *insert-at-index b a 0*
using *abx*
by *simp*
have *31*: ($c[\widehat{}]n$) =
($a[\widehat{}]$ (*nat N*)) ⊗ (*eval-at-point* Q_p (*insert-at-index b* (*inv a*) *0*) *P*)
using *22 30 c-def*
by *blast*
have *32*: *insert-at-index b* (*inv a*) *0* = (*inv a*) # *b*
using *insert-at-index.simps*
by (*metis drop0 self-append-conv2 take0*)
have *33*: ($c[\widehat{}]n$) =
($a[\widehat{}]$ (*nat N*)) ⊗ (*eval-at-point* Q_p ((*inv a*) # *b*) *P*)
using *31 32* **by** *presburger*
have *34*: (*inv a*) # *b* ∈ *carrier* $({Q_p}^{1+k})$

apply(*rule cartesian-power-car-memI''*)
apply (*metis b-closed cartesian-power-car-memE length-Suc-conv plus-1-eq-Suc*)
using *a-closed a-nz b-closed*
apply (*metis One-nat-def inv-in-frac(1) take0 take-Suc-Cons Qp.to-R1-closed*)
by (*metis abx b-closed b-def drop-Cons' not-Cons-self2*)
have *35*: (*eval-at-point* Q_p ((*inv a*) # *b*) *P*) ∈ *carrier* Q_p
using *34 P-n-def eval-at-point-closed*
by *blast*
have *inv*(*a*[^] (*nat N*)) ⊗ (*c*[^]*n*) =
inv(*a*[^] (*nat N*)) ⊗ ((*a*[^] (*nat N*)) ⊗ (*eval-at-point* Q_p ((*inv a*) # *b*) *P*))
using *31 33* **by** *presburger*
then have *6*: *inv*(*a*[^] (*nat N*)) ⊗ (*c*[^]*n*) =
inv(*a*[^] (*nat N*)) ⊗ (*a*[^] (*nat N*)) ⊗ (*eval-at-point* Q_p ((*inv a*) # *b*) *P*)
using *35 monoid.m-assoc*[*of* Q_p] *Qp.monoid-axioms Qp.nat-pow-closed Qp.nonzero-pow-nonzero a-nz inv-in-frac(1) local.a-closed* **by** *presburger*
have *37*:*inv*(*a*[^] (*nat N*)) ⊗ (*c*[^]*n*) = (*eval-at-point* Q_p ((*inv a*) # *b*) *P*)
proof−
have *inv*(*a*[^] (*nat N*)) ⊗ (*a*[^] (*nat N*)) = **1**
using *a-closed a-nz Qp.nat-pow-closed Qp.nonzero-pow-nonzero field-inv(1)*
by *blast*
then have *inv*(*a*[^] (*nat N*)) ⊗ (*c*[^]*n*) =
1 ⊗ (*eval-at-point* Q_p ((*inv a*) # *b*) *P*)
using *6* **by** *presburger*
then show *?thesis* **using** *35 Qp.l-one* **by** *blast*
qed
have *38*:(*inv a*)[^] (*nat N*) ⊗ (*c*[^]*n*) = (*eval-at-point* Q_p ((*inv a*) # *b*) *P*)
using *37 group.nat-pow-inv*[*of* Q_p *a nat N*] *a-closed Qp.field-axioms field.field-nat-pow-inv*[*of* Q_p]
by (*metis a-nz*)
have *39*:((*inv a*)[^]*m*) [^] $_{Q_p}$ *n* ⊗ (*c*[^]*n*) = (*eval-at-point* Q_p ((*inv a*) # *b*) *P*)
using *2 38 monoid.nat-pow-pow*[*of* Q_p *inv a*] *N-def*
by (*smt 3 Qp-int-nat-pow-pow a-closed a-nz inv-in-frac(3) of-nat-0-le-iff pow-nat*)
have *310*:((((*inv a*)[^]*m*) ⊗ *c*)[^]*n*) = (*eval-at-point* Q_p ((*inv a*) # *b*) *P*)
proof−
have *AA*: (*inv a*)[^]*m* ∈ *carrier* Q_p
using *Qp-int-pow-nonzero nonzero-def a-closed a-nz inv-in-frac(3) Qp.nonzero-closed*
by *presburger*

```
            have ((inv a)[^]m) [^] Q_p n ⊗ (c[^]n) = ((((inv a)[^]m) ⊗ c)[^]n)
              using Qp.nat-pow-distrib[of (inv a)[^]m c n] a-closed a-def c-def
AA by blast
            then show ?thesis
              using 39 by blast
          qed
          then show ?thesis using c'-def
            by blast
        qed
        have 4: inv a # b ∈ carrier (Q_p^{1+k})
          by (metis a-closed a-nz add.commute b-closed cartesian-power-cons
inv-in-frac(1))
        then have 5: ((inv a) # b) ∈ S
          using 3 P-n-def c'-closed basic-semialg-set-memI[of (inv a) # b 1 +
k c' P n]
          by blast
        have 6: Qp-invert (take 1 x) # drop 1 x = inv a # b
          using a-def b-def unfolding Qp-invert-def using 1 Qp-invert-def
          by blast
        show ?thesis using 5 6
          by presburger
      qed
    qed
    then show ?thesis
      using A by blast
  qed
 qed
 have 8: is-semialgebraic (1+k) ((partial-pullback 1 Qp-invert k S) ∩ Nz)
  using 7 Nz-semialg T-semialg intersection-is-semialg
  by auto
 have 9: (partial-pullback 1 Qp-invert k S) − Nz = {xs ∈ carrier (Q_p^{Suc k}). xs!0
= 0} ∩S
 proof
  show partial-pullback 1 Qp-invert k S − Nz ⊆ {xs ∈ carrier (Q_p^{Suc k}). xs ! 0
= 0} ∩ S
  proof fix x assume A: x ∈ partial-pullback 1 Qp-invert k S − Nz
    have 0: x ∈ carrier (Q_p^{Suc k})
      using A
      by (metis DiffD1 partial-pullback-memE(1) plus-1-eq-Suc)
    have 1: take 1 x ∈ carrier (Q_p^1)
      by (metis 0 le-add1 plus-1-eq-Suc take-closed)
    have 2: drop 1 x ∈ carrier (Q_p^k)
      using 0 cartesian-power-drop plus-1-eq-Suc
      by presburger
    have 3: x = take 1 x @ drop 1 x
      using 0
      by (metis append-take-drop-id)
    have 4: Qp-invert (take 1 x) # drop 1 x ∈ S
```

```
      using A partial-pullback-memE'[of take 1 x 1 drop 1 x k x Qp-invert S] 1 2
3
      by blast
    have 5: x!0 = 0
      using A 0 Nz-def by blast
    have 6: Qp-invert (take 1 x) # drop 1 x = x
    proof−
      have (take 1 x) =[x!0]
        using 0
        by (metis 1 3 append-Cons nth-Cons-0 Qp.to-R1-to-R)
      then have Qp-invert (take 1 x) = 0
        unfolding Qp-invert-def using 5
        by (metis less-one nth-take)
      then show ?thesis using 0 5
        by (metis 3 Cons-eq-append-conv ‹take 1 x = [x ! 0]› self-append-conv2)
    qed
    have x ∈ S
      using 6 4
      by presburger
    then show x ∈ {xs ∈ carrier (Q_p^Suc k). xs ! 0 = 0} ∩  S
      using Nz-def A 0
      by blast
  qed
  show {xs ∈ carrier (Q_p^Suc k). xs ! 0 = 0} ∩ S ⊆ partial-pullback 1 Qp-invert
k S − Nz
  proof fix x assume A: x ∈ {xs ∈ carrier (Q_p^Suc k). xs ! 0 = 0} ∩ S
    have A0: x ∈ carrier (Q_p^Suc k)
      using A by blast
    have A1: x!0  = 0
      using A by blast
    have A2: x ∈ S
      using A by blast
    show  x ∈ partial-pullback 1 Qp-invert k S − Nz
    proof
      show x ∉ Nz
        using Nz-def A1 by blast
      show  x ∈ partial-pullback 1 Qp-invert k S
      proof(rule partial-pullback-memI)
        show x ∈ carrier (Q_p^1+k)
          using A0
          by (simp add: A0)
        show Qp-invert (take 1 x) # drop 1 x ∈ S
        proof−
          have Qp-invert (take 1 x) = 0
            unfolding Qp-invert-def  using A0 A1
            by (metis less-numeral-extra(1) nth-take)
          then have Qp-invert (take 1 x) # drop 1 x = x
            using A0 A1 A2
             by (metis (no-types, lifting) Cons-eq-append-conv Qp-invert-def ‹x ∈
```

```
carrier (Qp^(1+k))›
            append-take-drop-id inv-in-frac(2) le-add-same-cancel1 self-append-conv2
                take-closed Qp.to-R1-to-R Qp.to-R-pow-closed zero-le)
          then show ?thesis
            using A2 by presburger
        qed
      qed
    qed
  qed
  have 10: is-semialgebraic (1+k) {xs ∈ carrier (Qp^Suc k). xs!0 = 0}
  proof−
    have {xs ∈ carrier (Qp^Suc k). xs!0 = 0} = V_Qp (Suc k) (pvar Qp 0)
      unfolding zero-set-def using eval-pvar[of 0 Suc k] Qp.cring-axioms
      by blast
    then show ?thesis using
        is-zero-set-imp-basic-semialg pvar-closed[of 0 Suc k] Qp.cring-axioms
        is-zero-set-imp-semialg plus-1-eq-Suc zero-less-Suc
      by presburger
  qed
  have 11: is-semialgebraic (1+k) ({xs ∈ carrier (Qp^Suc k). xs!0 = 0} ∩S)
    using 10 assms basic-semialg-is-semialgebraic intersection-is-semialg
    by blast
  have 12: (partial-pullback 1 Qp-invert k S) = ((partial-pullback 1 Qp-invert k S)
∩ Nz) ∪
                                                ((partial-pullback 1 Qp-invert k S) − Nz)
    by blast
  have 13: is-semialgebraic (1+k) ((partial-pullback 1 Qp-invert k S) − Nz)
    using 11 9 by metis
  show ?thesis
    using 8 12 13
    by (metis 7 Int-Diff-Un Int-commute plus-1-eq-Suc union-is-semialgebraic)
qed

lemma Qp-invert-is-semialg:
is-semialg-function 1 Qp-invert
proof(rule is-semialg-functionI′)
  show 0: Qp-invert ∈ carrier (Qp^1) → carrier Qp
  proof fix x
    assume A: x ∈ carrier (Qp^1)
    then obtain a where a-def: x = [a]
      by (metis Qp.to-R1-to-R)
    have a-closed: a ∈ carrier Qp
      using a-def A cartesian-power-concat(1) last-closed′
      by blast
    show Qp-invert x ∈ carrier Qp
    apply(cases a = 0)
    unfolding Qp-invert-def using a-def a-closed
    apply (meson Qp.to-R-to-R1)
```

by (*metis a-closed a-def inv-in-frac*(*1*) *Qp.to-R-to-R1*)
qed
show $\bigwedge k\ S.\ S \in$ *basic-semialgs* $(1 + k) \Longrightarrow$ *is-semialgebraic* $(1 + k)$ (*partial-pullback 1 Qp-invert k S*)
using *Qp-invert-basic-semialg*
by *blast*
qed

lemma *Taylor-deg-1-expansion''*:
assumes $f \in$ *carrier* Q_p-*x*
assumes $\bigwedge n.\ f\ n \in \mathcal{O}_p$
assumes $a \in \mathcal{O}_p$
assumes $b \in \mathcal{O}_p$
shows $\exists\, c\ c'\ c''.\ c =$ *to-fun f a* $\wedge\ c' =$ *deriv f a* $\wedge\ c \in \mathcal{O}_p \wedge\ c' \in \mathcal{O}_p \wedge c'' \in \mathcal{O}_p \wedge$
to-fun f $(b) = c \oplus c' \otimes (b \ominus a) \oplus (c'' \otimes (b \ominus a)[\widehat{\ }](2::nat))$
proof−
obtain *S* **where** *S-def*: $S = (Q_p \ (\!|\ carrier := \mathcal{O}_p\ |\!))$
by *blast*
have *1*: $f \in$ *carrier* (*UP S*)
unfolding *S-def* **using** *val-ring-subring UPQ.poly-cfs-subring*[*of* $\mathcal{O}_p$ *f*] *assms*
by *blast*
have *2*: $f \in$ *carrier* (*UP* (Q_p(|*carrier* := $\mathcal{O}_p$|)))
using *val-ring-subring 1 assms poly-cfs-subring*[*of* $\mathcal{O}_p$]
by *blast*
have *3*: $\exists\, c \in \mathcal{O}_p.\ f \cdot b = f \cdot a \oplus$ *UPQ.deriv f a* $\otimes (b \ominus a) \oplus c \otimes (b \ominus a)\ [\widehat{\ }]$ (*2*::*nat*)
using *UP-subring-taylor-appr'*[*of* $\mathcal{O}_p$ *f b a*] *UP-subring-taylor-appr*[*of* $\mathcal{O}_p$ *f b a*] *val-ring-subring 1 2 assms*
by *blast*
then show *?thesis*
using *UP-subring-taylor-appr*[*of* $\mathcal{O}_p$ *f b a*] *assms UP-subring-deriv-closed*[*of* $\mathcal{O}_p$ *f a*]
UP-subring-eval-closed[*of* $\mathcal{O}_p$ *f a*] *2 val-ring-subring* **by** *blast*
qed

end

13.12 Sets Defined by Residues of Valuation Ring Elements

sublocale *padic-fields* < *Res*: *cring Zp-res-ring* (*Suc n*)
using *p-residues residues.cring*
by *blast*

context *padic-fields*
begin

definition *Qp-res* **where**
Qp-res x n = *to-Zp x n*

lemma *Qp-res-closed*:
assumes $x \in \mathcal{O}_p$
shows *Qp-res x n* $\in$ *carrier* (*Zp-res-ring n*)
unfolding *Qp-res-def* **using** *assms val-ring-memE residue-closed to-Zp-closed*
by *blast*

lemma *Qp-res-add*:
assumes $x \in \mathcal{O}_p$
assumes $y \in \mathcal{O}_p$
shows *Qp-res* $(x \oplus y)$ *n* = *Qp-res x n* $\oplus_{\text{Zp-res-ring } n}$ *Qp-res y n*
unfolding *Qp-res-def*
using *assms residue-of-sum to-Zp-add* **by** *presburger*

lemma *Qp-res-mult*:
assumes $x \in \mathcal{O}_p$
assumes $y \in \mathcal{O}_p$
shows *Qp-res* $(x \otimes y)$ *n* = *Qp-res x n* $\otimes_{\text{Zp-res-ring } n}$ *Qp-res y n*
unfolding *Qp-res-def*
using *assms residue-of-prod to-Zp-mult* **by** *presburger*

lemma *Qp-res-diff*:
assumes $x \in \mathcal{O}_p$
assumes $y \in \mathcal{O}_p$
shows *Qp-res* $(x \ominus y)$ *n* = *Qp-res x n* $\ominus_{\text{Zp-res-ring } n}$ *Qp-res y n*
unfolding *Qp-res-def*
using *assms residue-of-diff to-Zp-minus*
by (*meson val-ring-res*)

lemma *Qp-res-zero*:
shows *Qp-res* **0** *n* = *0*
unfolding *Qp-res-def to-Zp-zero*
using *residue-of-zero*(*2*) **by** *blast*

lemma *Qp-res-one*:
assumes *n* > *0*
shows *Qp-res* **1** *n* = (*1*::*int*)
using *assms*
unfolding *Qp-res-def to-Zp-one*
using *residue-of-one*(*2*) **by** *blast*

lemma *Qp-res-nat-inc*:
shows *Qp-res* ([(*n*::*nat*)]·**1**) *n* = *n mod p*^*n*
unfolding *Qp-res-def* **unfolding** *to-Zp-nat-inc*
using *Zp-nat-inc-res* **by** *blast*

lemma *Qp-res-int-inc*:
shows *Qp-res* ([(*k*::*int*)]·**1**) *n* = *k mod p*^*n*
unfolding *Qp-res-def* **unfolding** *to-Zp-int-inc*

using *Zp-int-inc-res* **by** *blast*

lemma *Qp-poly-res-monom*:
assumes $a \in \mathcal{O}_p$
assumes $x \in \mathcal{O}_p$
assumes *Qp-res a n = 0*
assumes $k > 0$
shows *Qp-res* (*up-ring.monom* (*UP* Q_p) *a k* $\cdot$ *x*) *n = 0*
proof−
have *0*: *up-ring.monom* (*UP* Q_p) *a k* $\cdot$ *x* = *a* $\otimes$ *x* [^] *k*
apply(*rule UPQ.to-fun-monom*[*of a x k*])
using *assms val-ring-memE* **apply** *blast*
using *assms val-ring-memE* **by** *blast*
have *1*: *x*[^]*k* $\in \mathcal{O}_p$
using *assms val-ring-nat-pow-closed* **by** *blast*
show *?thesis* **unfolding** *0*
using *Qp-res-mult*[*of a x*[^]*k n*] *assms*
using *1 residue-times-zero-r* **by** *presburger*
qed

lemma *Qp-poly-res-zero*:
assumes *q* $\in$ *carrier* (*UP* Q_p)
assumes $\bigwedge i.$ *q i* $\in \mathcal{O}_p$
assumes $\bigwedge i.$ *Qp-res* (*q i*) *n = 0*
assumes *x* $\in \mathcal{O}_p$
shows *Qp-res* (*q* $\cdot$ *x*) *n = 0*
proof−
have ($\forall i.$ *q i* $\in \mathcal{O}_p \wedge$ *Qp-res* (*q i*) *n = 0*) $\longrightarrow$ *Qp-res* (*q* $\cdot$ *x*) *n = 0*
proof(*rule UPQ.poly-induct*[*of q*], *rule assms*, *rule*)
fix *p* **assume** *A*: *p* $\in$ *carrier* (*UP* Q_p) *deg* Q_p *p = 0* $\forall i.$ *p i* $\in \mathcal{O}_p \wedge$ *Qp-res* (*p i*) *n = 0*
have *0*: *p* $\cdot$ *x* = *p 0*
using *assms*
by (*metis A*(*1*) *A*(*2*) *val-ring-memE UPQ.ltrm-deg-0 UPQ.to-fun-ctrm*)
show *Qp-res* (*p* $\cdot$ *x*) *n = 0*
unfolding *0* **using** *A* **by** *blast*
next
fix *p*
assume *A0*: ($\bigwedge q.$ *q* $\in$ *carrier* (*UP* Q_p) $\Longrightarrow$ *deg* Q_p *q* < *deg* Q_p *p* $\Longrightarrow$ ($\forall i.$ *q i* $\in \mathcal{O}_p \wedge$ *Qp-res* (*q i*) *n = 0*) $\longrightarrow$ *Qp-res* (*q* $\cdot$ *x*) *n = 0*)
p $\in$ *carrier* (*UP* Q_p) *0* < *deg* Q_p *p*
show ($\forall i.$ *p i* $\in \mathcal{O}_p \wedge$ *Qp-res* (*p i*) *n = 0*) $\longrightarrow$ *Qp-res* (*p* $\cdot$ *x*) *n = 0*
proof assume *A1*: $\forall i.$ *p i* $\in \mathcal{O}_p \wedge$ *Qp-res* (*p i*) *n = 0*
obtain *k* **where** *k-def*: *k = deg* Q_p *p*
by *blast*
obtain *q* **where** *q-def*: *q = UPQ.trunc p*
by *blast*
have *q-closed*: *q* $\in$ *carrier* (*UP* Q_p)
unfolding *q-def*

```
    using A0(2) UPQ.trunc-closed by blast
  have q-deg: deg Q_p q < deg Q_p p
    unfolding q-def
    using A0(2) A0(3) UPQ.trunc-degree by blast
  have 9: ⋀i. i < deg Q_p p ⟹ q i = p i
    unfolding q-def
    using A0(2) UPQ.trunc-cfs by blast
  have 90: ⋀i. ¬ i < deg Q_p p ⟹ q i = 0
    unfolding q-def
  proof −
    fix i :: nat
    assume ¬ i < deg Q_p p
    then have deg Q_p q < i
      using q-deg by linarith
    then show Cring-Poly.truncate Q_p p i = 0
      using UPQ.deg-gtE q-closed q-def by blast
  qed
  have 10: (∀ i. q i ∈ 𝒪_p ∧ Qp-res (q i) n = 0)
  proof fix i
    show q i ∈ 𝒪_p ∧ Qp-res (q i) n = 0
      apply(cases i < deg Q_p p)
       using A1 9[of i] apply presburger
       unfolding q-def using Qp-res-zero 90
       by (metis q-def zero-in-val-ring)
  qed
  have 11: Qp-res (q ⋅ x) n = 0
    using 10 A1 A0 q-closed q-deg by blast
  have 12: p = q ⊕_{UP Q_p} up-ring.monom (UP Q_p) (p k) k
    unfolding k-def q-def
    using A0(2) UPQ.trunc-simps(1) by blast
  have 13: p ⋅ x = q ⋅ x ⊕ (up-ring.monom (UP Q_p) (p k) k) ⋅ x
  proof−
     have 0: (q ⊕_{UP Q_p} up-ring.monom (UP Q_p) (p k) k) ⋅ x = q ⋅ x ⊕
up-ring.monom (UP Q_p) (p k) k ⋅ x
      apply(rule UPQ.to-fun-plus)
      using A0(2) UPQ.ltrm-closed k-def apply blast
      unfolding q-def apply(rule UPQ.trunc-closed, rule A0)
      using assms val-ring-memE by blast
    show ?thesis
      using 0 12 by metis
  qed
  have 14: (up-ring.monom (UP Q_p) (p k) k) ⋅ x ∈ 𝒪_p
    apply(rule val-ring-poly-eval)
      using A0(2) UPQ.ltrm-closed k-def apply blast
      using UPQ.cfs-monom[of p k k ] A1 zero-in-val-ring
      using A0(2) UPQ.ltrm-cfs k-def apply presburger
      using assms(4) by blast
  have 15: Qp-res ((up-ring.monom (UP Q_p) (p k) k) ⋅ x) n = 0
    apply(rule Qp-poly-res-monom)
```

using *A1* **apply** *blast* **using** *assms* **apply** *blast*
using *A1* **apply** *blast* **unfolding** *k-def* **using** *A0* **by** *blast*
have *16*: *Qp-res* $(q \cdot x)$ $n = 0$
using *A0 10 11* **by** *blast*
have *17*: $q \cdot x \in \mathcal{O}_p$
apply(*rule val-ring-poly-eval*, *rule q-closed*)
using *10* **apply** *blast* **by**(*rule assms*)
have *18*: *Qp-res* $(q \cdot x \oplus (up\text{-}ring.monom\ (UP\ Q_p)\ (p\ k)\ k) \cdot x)$ $n = 0$
using *Qp-res-add*[*of* $q \cdot x$ *up-ring.monom* $(UP\ Q_p)$ $(p\ k)$ $k \cdot x$ n] *14 17*
unfolding *15 16*
by (*metis 10 Qp-res-add UPQ.cfs-add UPQ.coeff-of-sum-diff-degree0 q-closed q-deg*)
show *Qp-res* $(p \cdot x)$ $n = 0$
using *13 18* **by** *metis*
qed
qed
thus *?thesis* **using** *assms* **by** *blast*
qed

lemma *Qp-poly-res-eval-0*:
assumes $f \in carrier\ (UP\ Q_p)$
assumes $g \in carrier\ (UP\ Q_p)$
assumes $x \in \mathcal{O}_p$
assumes $\bigwedge i.\ f\ i \in \mathcal{O}_p$
assumes $\bigwedge i.\ g\ i \in \mathcal{O}_p$
assumes $\bigwedge i.$ *Qp-res* $(f\ i)$ $n =$ *Qp-res* $(g\ i)$ n
shows *Qp-res* $(f \cdot x)$ $n =$ *Qp-res* $(g \cdot x)$ n
proof−
obtain F **where** *F-def*: $F = f \ominus_{UP\ Q_p} g$
by *blast*
have *F-closed*: $F \in carrier\ (UP\ Q_p)$
unfolding *F-def*
using *assms* **by** *blast*
have *F-cfs*: $\bigwedge i.\ F\ i = (f\ i) \ominus (g\ i)$
unfolding *F-def*
using *assms UPQ.cfs-minus* **by** *blast*
have *F-cfs-res*: $\bigwedge i.$ *Qp-res* $(F\ i)$ $n =$ *Qp-res* $(f\ i)$ $n \ominus_{Zp\text{-}res\text{-}ring\ n}$ *Qp-res* $(g\ i)$ n
unfolding *F-cfs* **apply**(*rule Qp-res-diff*)
using *assms* **apply** *blast* **using** *assms* **by** *blast*
have *0*: $\bigwedge i.$ *Qp-res* $(f\ i)$ $n =$ *Qp-res* $(g\ i)$ n
using *assms* **by** *blast*
have *F-cfs-res'*: $\bigwedge i.$ *Qp-res* $(F\ i)$ $n = 0$
unfolding *F-cfs-res 0*
by (*metis diff-self mod-0 residue-minus*)
have *1*: $\bigwedge i.\ F\ i \in \mathcal{O}_p$
unfolding *F-cfs* **using** *assms*
using *val-ring-minus-closed* **by** *blast*
have *2*: *Qp-res* $(F \cdot x)$ $n = 0$

by(*rule Qp-poly-res-zero*, *rule F-closed*, *rule 1*, *rule F-cfs-res'*, *rule assms*)
have *3*: $F \cdot x = f \cdot x \ominus g \cdot x$
unfolding *F-def* **using** *assms*
by (*meson assms UPQ.to-fun-diff val-ring-memE*)
have *4*: *Qp-res* $(F \cdot x)$ n = *Qp-res* $(f \cdot x)$ n $\ominus_{\text{Zp-res-ring } n}$ *Qp-res* $(g \cdot x)$ n
unfolding *3* **apply**(*rule Qp-res-diff*, *rule val-ring-poly-eval*, *rule assms*)
using *assms* **apply** *blast* **using** *assms* **apply** *blast*
apply(*rule val-ring-poly-eval*, *rule assms*)
using *assms* **apply** *blast* **by**(*rule assms*)
have *5*: $f \cdot x \in \mathcal{O}_p$
apply(*rule val-ring-poly-eval*, *rule assms*)
using *assms* **apply** *blast* **using** *assms* **by** *blast*
have *6*: $g \cdot x \in \mathcal{O}_p$
apply(*rule val-ring-poly-eval*, *rule assms*)
using *assms* **apply** *blast* **by**(*rule assms*)
show *Qp-res* $(f \cdot x)$ n = *Qp-res* $(g \cdot x)$ n
using *5 6 2 Qp-res-closed*[*of* $f \cdot x$ n] *Qp-res-closed*[*of* $g \cdot x$ n]
unfolding *4*
proof −
assume *Qp-res* $(f \cdot x)$ n $\ominus_{\text{Zp-res-ring } n}$ *Qp-res* $(g \cdot x)$ n = *0*
then show *?thesis*
by (*metis* (*no-types*) *Qp-res-def 5 6 res-diff-zero-fact*(*1*) *residue-of-diff to-Zp-closed val-ring-memE*)
qed
qed

lemma *Qp-poly-res-eval-1*:
assumes $f \in$ *carrier* (*UP* Q_p)
assumes $x \in \mathcal{O}_p$
assumes $y \in \mathcal{O}_p$
assumes $\bigwedge i.\ f\ i \in \mathcal{O}_p$
assumes *Qp-res* x n = *Qp-res* y n
shows *Qp-res* $(f \cdot x)$ n = *Qp-res* $(f \cdot y)$ n
proof−
have $(\forall i.\ f\ i \in \mathcal{O}_p) \longrightarrow$ *Qp-res* $(f \cdot x)$ n = *Qp-res* $(f \cdot y)$ n
apply(*rule UPQ.poly-induct*[*of f*], *rule assms*)
proof fix f **assume** *A*: $f \in$ *carrier* (*UP* Q_p) *deg* Q_p f = *0* $\forall i.\ f\ i \in \mathcal{O}_p$
show *Qp-res* $(f \cdot x)$ n = *Qp-res* $(f \cdot y)$ n
proof−
obtain *a* **where** *a-def*: $a \in$ *carrier* $Q_p \wedge f$ = *to-polynomial* Q_p a
using *assms*
by (*metis A*(*1*) *A*(*2*) *UPQ.lcf-closed UPQ.to-poly-inverse*)
have *a-eq*: f = *to-polynomial* Q_p a
using *a-def* **by** *blast*
have *0*: $f \cdot x = a$
using *a-def assms* **unfolding** *a-eq*
by (*meson UPQ.to-fun-to-poly val-ring-memE*)
have *1*: $f \cdot y = a$
using *a-def assms* **unfolding** *a-eq*

by (*meson UPQ.to-fun-to-poly val-ring-memE*)
show *Qp-res* (*f* ⋅ *x*) *n* = *Qp-res* (*f* ⋅ *y*) *n*
unfolding *0 1* **by** *blast*
qed
next
fix *f*
assume *A*: (⋀*q*. *q* ∈ *carrier* (*UP Q*$_p$) ⟹ *deg Q*$_p$ *q* < *deg Q*$_p$ *f* ⟹ (∀ *i*. *q i* ∈ $\mathcal{O}_p$) ⟶ *Qp-res* (*q* ⋅ *x*) *n* = *Qp-res* (*q* ⋅ *y*) *n*)
f ∈ *carrier* (*UP Q*$_p$) 0 < *deg Q*$_p$ *f*
show (∀ *i*. *f i* ∈ $\mathcal{O}_p$) ⟶ *Qp-res* (*f* ⋅ *x*) *n* = *Qp-res* (*f* ⋅ *y*) *n*
proof assume *A1*: ∀ *i*. *f i* ∈ $\mathcal{O}_p$
obtain *q* **where** *q-def*: *q* = *UPQ.trunc f*
by *blast*
have *q-closed*: *q* ∈ *carrier* (*UP Q*$_p$)
using *q-def A UPQ.trunc-closed* **by** *presburger*
have *q-deg*: *deg Q*$_p$ *q* < *deg Q*$_p$ *f*
using *q-def A UPQ.trunc-degree* **by** *blast*
have *q-cfs*: ∀ *i*. *q i* ∈ $\mathcal{O}_p$
proof fix *i* **show** *q i* ∈ $\mathcal{O}_p$
apply(*cases i* < *deg Q*$_p$ *f*)
unfolding *q-def* **using** *A A1 UPQ.trunc-cfs*
apply *presburger*
using *q-deg q-closed*
proof −
assume ¬ *i* < *deg Q*$_p$ *f*
then have *deg Q*$_p$ *f* ≤ *i*
by (*meson diff-is-0-eq neq0-conv zero-less-diff*)
then show *Cring-Poly.truncate Q*$_p$ *f i* ∈ $\mathcal{O}_p$
by (*metis* (*no-types*) *UPQ.deg-eqI diff-is-0-eq*′ *le-trans nat-le-linear neq0-conv q-closed q-def q-deg zero-in-val-ring zero-less-diff*)
qed
qed
hence *0*: *Qp-res* (*q* ⋅ *x*) *n* = *Qp-res* (*q* ⋅ *y*) *n*
using *A q-closed q-deg* **by** *blast*
have *1*: *Qp-res* (*UPQ.ltrm f* ⋅ *x*) *n* = *Qp-res* (*UPQ.ltrm f* ⋅ *y*) *n*
proof−
have *10*: *UPQ.ltrm f* ⋅ *x* = (*f* (*deg Q*$_p$ *f*)) ⊗ *x*[^](*deg Q*$_p$ *f*)
using *A assms A1 UPQ.to-fun-monom val-ring-memE* **by** *presburger*
have *11*: *UPQ.ltrm f* ⋅ *y* = (*f* (*deg Q*$_p$ *f*)) ⊗ *y*[^](*deg Q*$_p$ *f*)
using *A assms A1 UPQ.to-fun-monom val-ring-memE* **by** *presburger*
obtain *d* **where** *d-def*: *d* = *deg Q*$_p$ *f*
by *blast*
have *12*: *Qp-res* (*x*[^]*d*) *n* = *Qp-res* (*y*[^]*d*) *n*
apply(*induction d*)
using *Qp.nat-pow-0* **apply** *presburger*
using *Qp-res-mult assms Qp.nat-pow-Suc val-ring-nat-pow-closed* **by** *presburger*
hence *13*: *Qp-res* (*x* [^] *deg Q*$_p$ *f*) *n* = *Qp-res* (*y* [^] *deg Q*$_p$ *f*) *n*
unfolding *d-def* **by** *blast*

```
      have 14: x [^] deg Q_p f ∈ 𝒪_p
        using assms val-ring-nat-pow-closed by blast
      have 15: y [^] deg Q_p f ∈ 𝒪_p
        using assms val-ring-nat-pow-closed by blast
      have 16: Qp-res (f (deg Q_p f) ⊗ x [^] deg Q_p f) n = Qp-res (f (deg Q_p f))
n ⊗_residue-ring (p ^ n) Qp-res (x [^] deg Q_p f) n
        apply(rule Qp-res-mult[of f (deg Q_p f)  x[^](deg Q_p f) n])
        using A1 apply blast by(rule 14)
      have 17: Qp-res (f (deg Q_p f) ⊗ y [^] deg Q_p f) n = Qp-res (f (deg Q_p f))
n ⊗_residue-ring (p ^ n) Qp-res (y [^] deg Q_p f) n
        apply(rule Qp-res-mult[of f (deg Q_p f)  y[^](deg Q_p f) n])
        using A1 apply blast by(rule 15)
      show ?thesis
        unfolding 10 11 16 17 13 by blast
    qed
    have f-decomp: f = q ⊕_UP Q_p UPQ.ltrm f
      using A unfolding q-def
      using UPQ.trunc-simps(1) by blast
    have 2: f · x = q · x ⊕ (UPQ.ltrm f · x)
      using A f-decomp q-closed q-cfs
      by (metis val-ring-memE UPQ.ltrm-closed UPQ.to-fun-plus assms(2))
    have 3: f · y = q · y ⊕ (UPQ.ltrm f · y)
      using A f-decomp q-closed q-cfs
      by (metis val-ring-memE UPQ.ltrm-closed UPQ.to-fun-plus assms(3))
    show 4:  Qp-res (f · x) n = Qp-res (f · y) n
      unfolding 2 3 using assms q-cfs Qp-res-add 0 1
      by (metis (no-types, opaque-lifting) 2 3 A(2) A1 Qp-res-def poly-eval-cong)
  qed
 qed
 thus ?thesis using assms by blast
qed

lemma Qp-poly-res-eval-2:
 assumes f ∈ carrier (UP Q_p)
 assumes g ∈ carrier (UP Q_p)
 assumes x ∈ 𝒪_p
 assumes y ∈ 𝒪_p
 assumes ⋀i. f i ∈ 𝒪_p
 assumes ⋀i. g i ∈ 𝒪_p
 assumes ⋀i. Qp-res (f i) n = Qp-res (g i) n
 assumes Qp-res x n = Qp-res y n
 shows Qp-res (f · x) n = Qp-res (g · y) n
proof−
 have 0: Qp-res (f · x) n = Qp-res (g · x) n
   using Qp-poly-res-eval-0 assms by blast
 have 1: Qp-res (g · x) n = Qp-res (g · y) n
   using Qp-poly-res-eval-1 assms by blast
 show ?thesis unfolding 0 1 by blast
qed
```

definition *poly-res-class* **where**
poly-res-class n d f = {*q* ∈ *carrier* (*UP* Q_p). *deg* Q_p *q* ≤ *d* ∧ (∀ *i*. *q i* ∈ $\mathcal{O}_p$ ∧
Qp-res (*f i*) *n* = *Qp-res* (*q i*) *n*) }

lemma *poly-res-class-closed*:
assumes *f* ∈ *carrier* (*UP* Q_p)
assumes *g* ∈ *carrier* (*UP* Q_p)
assumes *deg* Q_p *f* ≤ *d*
assumes *deg* Q_p *g* ≤ *d*
assumes *g* ∈ *poly-res-class n d f*
shows *poly-res-class n d f* = *poly-res-class n d g*
unfolding *poly-res-class-def*
apply(*rule equalityI*)
apply(*rule subsetI*)
unfolding *mem-Collect-eq* **apply**(*rule conjI*, *blast*, *rule conjI*, *blast*)
using *assms* **unfolding** *poly-res-class-def mem-Collect-eq*
apply *presburger*
apply(*rule subsetI*) **unfolding** *mem-Collect-eq*
apply(*rule conjI*, *blast*, *rule conjI*, *blast*)
using *assms* **unfolding** *poly-res-class-def mem-Collect-eq*
by *presburger*

lemma *poly-res-class-memE*:
assumes *f* ∈ *poly-res-class n d g*
shows *f* ∈ *carrier* (*UP* Q_p)
deg Q_p *f* ≤ *d*
f i ∈ $\mathcal{O}_p$
Qp-res (*g i*) *n* = *Qp-res* (*f i*) *n*
using *assms* **unfolding** *poly-res-class-def mem-Collect-eq* **apply** *blast*
using *assms* **unfolding** *poly-res-class-def mem-Collect-eq* **apply** *blast*
using *assms* **unfolding** *poly-res-class-def mem-Collect-eq* **apply** *blast*
using *assms* **unfolding** *poly-res-class-def mem-Collect-eq* **by** *blast*

definition *val-ring-polys* **where**
val-ring-polys = {*f* ∈ *carrier* (*UP* Q_p). (∀ *i*. *f i* ∈ $\mathcal{O}_p$)}

lemma *val-ring-polys-closed*:
val-ring-polys ⊆ *carrier* (*UP* Q_p)
unfolding *val-ring-polys-def* **by** *blast*

lemma *val-ring-polys-memI*:
assumes *f* ∈ *carrier* (*UP* Q_p)
assumes ⋀*i*. *f i* ∈ $\mathcal{O}_p$
shows *f* ∈ *val-ring-polys*
using *assms* **unfolding** *val-ring-polys-def* **by** *blast*

lemma *val-ring-polys-memE*:
assumes *f* ∈ *val-ring-polys*

shows $f \in carrier\ (UP\ Q_p)$
$f\ i \in \mathcal{O}_p$
using *assms* **unfolding** *val-ring-polys-def* **apply** *blast*
using *assms* **unfolding** *val-ring-polys-def* **by** *blast*

definition *val-ring-polys-grad* **where**
$val\text{-}ring\text{-}polys\text{-}grad\ d = \{f \in val\text{-}ring\text{-}polys.\ deg\ Q_p\ f \leq d\}$

lemma *val-ring-polys-grad-closed*:
$val\text{-}ring\text{-}polys\text{-}grad\ d \subseteq val\text{-}ring\text{-}polys$
unfolding *val-ring-polys-grad-def* **by** *blast*

lemma *val-ring-polys-grad-closed'*:
$val\text{-}ring\text{-}polys\text{-}grad\ d \subseteq carrier\ (UP\ Q_p)$
unfolding *val-ring-polys-grad-def val-ring-polys-def* **by** *blast*

lemma *val-ring-polys-grad-memI*:
assumes $f \in carrier\ (UP\ Q_p)$
assumes $\bigwedge i.\ f\ i \in \mathcal{O}_p$
assumes $deg\ Q_p\ f \leq d$
shows $f \in val\text{-}ring\text{-}polys\text{-}grad\ d$
using *assms* **unfolding** *val-ring-polys-grad-def val-ring-polys-def* **by** *blast*

lemma *val-ring-polys-grad-memE*:
assumes $f \in val\text{-}ring\text{-}polys\text{-}grad\ d$
shows $f \in carrier\ (UP\ Q_p)$
$deg\ Q_p\ f \leq d$
$f\ i \in \mathcal{O}_p$
using *assms* **unfolding** *val-ring-polys-grad-def val-ring-polys-def* **apply** *blast*
using *assms* **unfolding** *val-ring-polys-grad-def val-ring-polys-def* **apply** *blast*
using *assms* **unfolding** *val-ring-polys-grad-def val-ring-polys-def* **by** *blast*

lemma *poly-res-classes-in-val-ring-polys-grad*:
assumes $f \in val\text{-}ring\text{-}polys\text{-}grad\ d$
shows $poly\text{-}res\text{-}class\ n\ d\ f \subseteq val\text{-}ring\text{-}polys\text{-}grad\ d$
apply(*rule subsetI*, *rule val-ring-polys-grad-memI*)
apply(*rule poly-res-class-memE*[*of - n d f*], *blast*)
apply(*rule poly-res-class-memE*[*of - n d f*], *blast*)
by(*rule poly-res-class-memE*[*of - n d f*], *blast*)

lemma *poly-res-class-disjoint*:
assumes $f \in val\text{-}ring\text{-}polys\text{-}grad\ d$
assumes $f \notin poly\text{-}res\text{-}class\ n\ d\ g$
shows $poly\text{-}res\text{-}class\ n\ d\ f \cap poly\text{-}res\text{-}class\ n\ d\ g = \{\}$
apply(*rule equalityI*)
apply(*rule subsetI*)
using *assms*
unfolding *poly-res-class-def mem-Collect-eq Int-iff*
apply (*metis val-ring-polys-grad-memE*(*1*) *val-ring-polys-grad-memE*(*2*) *val-ring-polys-grad-memE*(*3*))

by *blast*

lemma *poly-res-class-refl*:
assumes $f \in$ *val-ring-polys-grad* d
shows $f \in$ *poly-res-class* n d f
unfolding *poly-res-class-def mem-Collect-eq*
using *assms val-ring-polys-grad-memE*(*1*) *val-ring-polys-grad-memE*(*2*) *val-ring-polys-grad-memE*(*3*)
by *blast*

lemma *poly-res-class-memI*:
assumes $f \in$ *carrier* (*UP* Q_p)
assumes *deg* Q_p $f \leq d$
assumes $\bigwedge i$. $f\ i \in \mathcal{O}_p$
assumes $\bigwedge i$. *Qp-res* (*f i*) n = *Qp-res* (*g i*) n
shows $f \in$ *poly-res-class* n d g
unfolding *poly-res-class-def mem-Collect-eq* **using** *assms*
by *metis*

definition *poly-res-classes* **where**
poly-res-classes n d = *poly-res-class* n d ‘ *val-ring-polys-grad* d

lemma *poly-res-classes-disjoint*:
assumes $A \in$ *poly-res-classes* n d
assumes $B \in$ *poly-res-classes* n d
assumes $g \in A - B$
shows $A \cap B = \{\}$
proof−
obtain *a* **where** *a-def*: $a \in$ *val-ring-polys-grad* $d \wedge A =$ *poly-res-class* n d a
using *assms* **unfolding** *poly-res-classes-def* **by** *blast*
obtain *b* **where** *b-def*: $b \in$ *val-ring-polys-grad* $d \wedge B =$ *poly-res-class* n d b
using *assms* **unfolding** *poly-res-classes-def* **by** *blast*
have *0*: $\bigwedge f$. $f \in A \cap B \Longrightarrow$ *False*
proof−
fix *f* **assume** *A*: $f \in A \cap B$
have *1*: $\exists i$. *Qp-res* (*g i*) $n \neq$ *Qp-res* (*f i*) n
proof(*rule ccontr*)
assume *B*: $\nexists i$. *Qp-res* (*g i*) $n \neq$ *Qp-res* (*f i*) n
then have *2*: $\bigwedge i$. *Qp-res* (*g i*) n = *Qp-res* (*f i*) n
by *blast*
have *3*: $g \in$ *poly-res-class* n d a
using *a-def assms* **by** *blast*
have *4*: $\bigwedge i$. *Qp-res* (*b i*) n = *Qp-res* (*f i*) n
apply(*rule poly-res-class-memE*[*of* - n d])
using *assms A b-def* **by** *blast*
have *5*: $\bigwedge i$. *Qp-res* (*a i*) n = *Qp-res* (*g i*) n
apply(*rule poly-res-class-memE*[*of* - n d])
using *assms A a-def* **by** *blast*
have *6*: $g \in$ *poly-res-class* n d b
apply(*rule poly-res-class-memI*, *rule poly-res-class-memE*[*of* - n d a], *rule*

3,
        rule poly-res-class-memE[*of* - *n d a*], *rule 3*, *rule poly-res-class-memE*[*of* - *n d a*], *rule 3*)
      unfolding *2 4* **by** *blast*
    show *False* **using** *6 b-def assms* **by** *blast*
  qed
  then obtain *i* **where** *i-def*: *Qp-res* (*g i*) *n* ≠ *Qp-res* (*f i*) *n*
    by *blast*
  have *2*: ⋀*i*. *Qp-res* (*a i*) *n* = *Qp-res* (*f i*) *n*
    apply(*rule poly-res-class-memE*[*of* - *n d*])
    using *A a-def* **by** *blast*
  have *3*: ⋀*i*. *Qp-res* (*b i*) *n* = *Qp-res* (*f i*) *n*
    apply(*rule poly-res-class-memE*[*of* - *n d*])
    using *A b-def* **by** *blast*
  have *4*: ⋀*i*. *Qp-res* (*a i*) *n* = *Qp-res* (*g i*) *n*
    apply(*rule poly-res-class-memE*[*of* - *n d*])
    using *assms a-def* **by** *blast*
  show *False* **using** *i-def 2* **unfolding** *4 2* **by** *blast*
 qed
 show *?thesis* **using** *0* **by** *blast*
qed

definition *int-fun-to-poly* **where**
int-fun-to-poly (*f*::*nat* ⇒ *int*) *i* = [(*f i*)]·**1**

lemma *int-fun-to-poly-closed*:
 assumes ⋀*i*. *i* > *d* ⟹ *f i* = *0*
 shows *int-fun-to-poly f* ∈ *carrier* (*UP* Q_p)
 apply(*rule UPQ.UP-car-memI*[*of d*])
 using *assms* **unfolding** *int-fun-to-poly-def*
 using *Qp.int-inc-zero* **apply** *presburger*
 by(*rule Qp.int-inc-closed*)

lemma *int-fun-to-poly-deg*:
 assumes ⋀*i*. *i* > *d* ⟹ *f i* = *0*
 shows *deg* Q_p (*int-fun-to-poly f*) ≤ *d*
 apply(*rule UPQ.deg-leqI*, *rule int-fun-to-poly-closed*, *rule assms*, *blast*)
 unfolding *int-fun-to-poly-def* **using** *assms*
 using *Qp.int-inc-zero* **by** *presburger*

lemma *Qp-res-mod-triv*:
 assumes *a* ∈ $\mathcal{O}_p$
 shows *Qp-res a n mod p* ^ *n* = *Qp-res a n*
 using *assms Qp-res-closed*[*of a n*]
by (*meson mod-pos-pos-trivial p-residue-ring-car-memE*(*1*) *p-residue-ring-car-memE*(*2*))

lemma *int-fun-to-poly-is-class-wit*:
 assumes *f* ∈ *poly-res-class n d g*
 shows (*int-fun-to-poly* (λ*i*::*nat*. *Qp-res* (*f i*) *n*)) ∈ *poly-res-class n d g*

proof(*rule poly-res-class-memI*[*of*], *rule int-fun-to-poly-closed*[*of d*])
show *0*: $\bigwedge i.\ d < i \Longrightarrow$ *Qp-res* (*f i*) *n* = *0*
proof− **fix** *i* **assume** *A*: *d* < *i*
hence *0*: *deg* Q_p *f* < *i*
using *A* *assms poly-res-class-memE*(*2*)[*of f n d g*]
by *linarith*
have *1*: *f i* = **0**
using *0 assms poly-res-class-memE*[*of f n d g*]
using *UPQ.UP-car-memE*(*2*) **by** *blast*
show *Qp-res* (*f i*) *n* = *0*
unfolding *1 Qp-res-zero* **by** *blast*
qed
show *deg* Q_p (*int-fun-to-poly* (λ*i*. *Qp-res* (*f i*) *n*)) ≤ *d*
by(*rule int-fun-to-poly-deg*, *rule 0*, *blast*)
show $\bigwedge i.$ *int-fun-to-poly* (λ*i*. *Qp-res* (*f i*) *n*) *i* ∈ $\mathcal{O}_p$
unfolding *int-fun-to-poly-def*
using *Qp.int-mult-closed Qp-val-ringI val-of-int-inc* **by** *blast*
show $\bigwedge i.$ *Qp-res* (*int-fun-to-poly* (λ*i*. *Qp-res* (*f i*) *n*) *i*) *n* = *Qp-res* (*g i*) *n*
unfolding *int-fun-to-poly-def Qp-res-int-inc*
using *Qp-res-mod-triv assms poly-res-class-memE*(*4*) *Qp-res-closed UPQ.cfs-closed*
by (*metis poly-res-class-memE*(*3*))
qed

lemma *finite-support-funs-finite*:
finite (({..*d*} → *carrier* (*Zp-res-ring n*)) ∩ {(*f*::*nat* ⇒ *int*). ∀ *i* > *d*. *f i* = *0*})
proof−
have *0*: *finite* (Π_E *i* ∈ {..*d*}.*carrier* (*Zp-res-ring n*))
apply(*rule finite-PiE*, *blast*)
using *residue-ring-card*[*of n*] **by** *blast*
obtain *g* **where** *g-def*: *g* = (λ*f*. (λ*i*::*nat*. *if i* ∈ {..*d*} *then f i else* (*0*::*int*)))
by *blast*
have *1*: *g* ‘ (Π_E *i* ∈ {..*d*}.*carrier* (*Zp-res-ring n*)) = (({..*d*} → *carrier* (*Zp-res-ring n*)) ∩ {(*f*::*nat* ⇒ *int*). ∀ *i* > *d*. *f i* = *0*})
proof(*rule equalityI*, *rule subsetI*)
fix *x* **assume** *A*: *x* ∈ *g* ‘ ({..*d*} →$_E$ *carrier* (*residue-ring* (*p* ^ *n*)))
obtain *f* **where** *f-def*: *f* ∈ (Π_E *i* ∈ {..*d*}.*carrier* (*Zp-res-ring n*)) ∧ *x* = *g f*
using *A* **by** *blast*
have *x-eq*: *x* = *g f*
using *f-def* **by** *blast*
show *x* ∈ ({..*d*} → *carrier* (*residue-ring* (*p* ^ *n*))) ∩ {*f*. ∀ *i*>*d*. *f i* = *0*}
proof(*rule*, *rule*)
fix *i* **assume** *A*: *i* ∈ {..*d*}
show *x i* ∈ *carrier* (*Zp-res-ring n*)
proof(*cases i* ∈ {..*d*})
case *True*
then have *T0*: *f i* ∈ *carrier* (*Zp-res-ring n*)
using *f-def* **by** *blast*
have *x i* = *f i*
unfolding *x-eq g-def*

```
      using True by metis
    thus ?thesis using T0 by metis
  next
    case False
    then have F0: x i = 0
      unfolding x-eq g-def by metis
    show ?thesis
      unfolding F0
      by (metis residue-mult-closed residue-times-zero-r)
  qed
next
  show x ∈ {f. ∀ i>d. f i = 0}
  proof(rule, rule, rule)
    fix i assume A: d < i
    then have 0: i ∉ {..d}
      by simp
    thus x i = 0
      unfolding x-eq g-def
      by metis
  qed
qed
next
  show ({..d} → carrier (residue-ring (p ^ n))) ∩ {f. ∀ i>d. f i = 0}
  ⊆ g ` ({..d} →E carrier (residue-ring (p ^ n)))
  proof(rule subsetI)
  fix x
  assume A:  x ∈ ({..d} → carrier (residue-ring (p ^ n))) ∩ {f. ∀ i>d. f i = 0}
  show  x ∈ g ` ({..d} →E carrier (residue-ring (p ^ n)))
  proof−
    obtain h where h-def: h = restrict x {..d}
      by blast
    have 0: ⋀i. i ∈ {..d} ⟹ h i = x i
      unfolding h-def restrict-apply by metis
    have 1: ⋀i. i ∉ {..d} ⟹ h i = undefined
      unfolding h-def restrict-apply by metis
    have 2: ⋀i. i ∈ {..d} ⟹ h i ∈ carrier (Zp-res-ring n)
      using A 0 unfolding 0 by blast
    have 3: h ∈ {..d} →E carrier (residue-ring (p ^ n))
      by(rule, rule 2, blast, rule 1, blast)
    have 4: ⋀i. i ∉ {..d} ⟹ x i = 0
      using A unfolding Int-iff mem-Collect-eq
      by (metis atMost-iff eq-imp-le le-simps(1) linorder-neqE-nat)
    have 5: x = g h
    proof fix i
      show x i = g h i
        unfolding g-def
        apply(cases i ∈ {..d})
        using 0 apply metis unfolding 4
        by metis
```

```
    qed
    show ?thesis unfolding 5 using 3 by blast
   qed
  qed
 qed
 have 2: finite (g ‘ (Π_E i ∈ {..d}.carrier (Zp-res-ring n)))
  using 0 by blast
 show ?thesis using 2 unfolding 1 by blast
qed

lemma poly-res-classes-finite:
finite (poly-res-classes n d)
proof−
 have 0: poly-res-class n d ‘ int-fun-to-poly ‘ (({..d} → carrier (Zp-res-ring n)) ∩
{(f::nat ⇒ int). ∀ i > d. f i = 0}) = poly-res-classes n d
 proof(rule equalityI, rule subsetI)
  fix x assume A: x ∈ poly-res-class n d ‘ int-fun-to-poly ‘ (({..d} → carrier
(residue-ring (p ^ n))) ∩ {f. ∀ i>d. f i = 0})
  then obtain f where f-def: f ∈ ({..d} → carrier (residue-ring (p ^ n))) ∩ {f.
∀ i>d. f i = 0} ∧
                        x = poly-res-class n d (int-fun-to-poly f)
   by blast
  have x-eq: x = poly-res-class n d (int-fun-to-poly f)
   using f-def by blast
  show x ∈ poly-res-classes n d
  proof−
   have 0: int-fun-to-poly f ∈ val-ring-polys-grad d
    apply(rule val-ring-polys-grad-memI, rule int-fun-to-poly-closed[of d])
    using f-def apply blast
    using int-fun-to-poly-def
   apply (metis Qp.int-inc-closed padic-fields.int-fun-to-poly-def padic-fields.val-of-int-inc
padic-fields-axioms val-ring-memI)
    apply(rule int-fun-to-poly-deg)
    using f-def by blast
   show ?thesis unfolding poly-res-classes-def x-eq
    using 0 by blast
  qed
 next
  show poly-res-classes n d
  ⊆ poly-res-class n d ‘
    int-fun-to-poly ‘
    (({..d} → carrier (residue-ring (p ^ n))) ∩
     {f. ∀ i>d. f i = 0})
  proof(rule subsetI)
  fix x assume A: x ∈ poly-res-classes n d
  show x ∈ poly-res-class n d ‘ int-fun-to-poly ‘ (({..d} → carrier (residue-ring
(p ^ n))) ∩ {f. ∀ i>d. f i = 0})
  proof−
   obtain f where f-def: f ∈ val-ring-polys-grad d ∧ x = poly-res-class n d f
```

```
      using A unfolding poly-res-classes-def by blast
    have x-eq: x = poly-res-class n d f
      using f-def by blast
    obtain h where h-def: h = (λi::nat. Qp-res (f i) n)
      by blast
    have 0: ⋀i. i > d ⟹ f i = 0
    proof− fix i assume A: i > d
      have i > deg Q_p f
        apply(rule le-less-trans[of - d])
      using f-def unfolding val-ring-polys-grad-def val-ring-polys-def mem-Collect-eq
        apply blast
        by(rule A)
      then show f i = 0
     using f-def unfolding val-ring-polys-grad-def val-ring-polys-def mem-Collect-eq
       using UPQ.deg-leE by blast
    qed
    have 1: ⋀i. i > d ⟹ h i = 0
      unfolding h-def 0 Qp-res-zero by blast
    have 2: x = poly-res-class n d (int-fun-to-poly h)
      unfolding x-eq
      apply(rule poly-res-class-closed)
   using f-def unfolding val-ring-polys-grad-def val-ring-polys-def mem-Collect-eq
apply blast
        apply(rule int-fun-to-poly-closed[of d], rule 1, blast)
   using f-def unfolding val-ring-polys-grad-def val-ring-polys-def mem-Collect-eq
apply blast
       apply(rule int-fun-to-poly-deg, rule 1, blast)
      unfolding h-def
      apply(rule int-fun-to-poly-is-class-wit, rule poly-res-class-refl)
      using f-def by blast
    have 3: h ∈ ({..d} → carrier (residue-ring (p ^ n))) ∩ {f. ∀ i>d. f i = 0}
      apply(rule , rule )
     unfolding h-def apply(rule Qp-res-closed, rule val-ring-polys-grad-memE[of
- d])
      using f-def apply blast
      unfolding mem-Collect-eq apply(rule, rule)
      unfolding 0 Qp-res-zero by blast
    show ?thesis
      unfolding 2 using 3 by blast
    qed
  qed
 qed
 have 1: finite (poly-res-class n d ‘ int-fun-to-poly ‘ (({..d} → carrier (Zp-res-ring
n)) ∩ {(f::nat ⇒ int). ∀ i > d. f i = 0}))
  using finite-support-funs-finite by blast
 show ?thesis using 1 unfolding 0 by blast
qed

lemma Qp-res-eq-zeroI:
```

assumes $a \in \mathcal{O}_p$
assumes *val* $a \geq n$
shows *Qp-res* $a\ n = 0$
proof−
have *0*: *val-Zp* (*to-Zp* a) $\geq n$
using *assms to-Zp-val* **by** *presburger*
have *1*: *to-Zp* $a\ n = 0$
apply(*rule zero-below-val-Zp*, *rule to-Zp-closed*)
using *val-ring-closed assms* **apply** *blast*
by(*rule 0*)
thus *?thesis* **unfolding** *Qp-res-def* **by** *blast*
qed

lemma *Qp-res-eqI*:
assumes $a \in \mathcal{O}_p$
assumes $b \in \mathcal{O}_p$
assumes *Qp-res* $(a \ominus b)\ n = 0$
shows *Qp-res* $a\ n =$ *Qp-res* $b\ n$
using *assms* **by** (*metis Qp-res-def val-ring-memE res-diff-zero-fact*(*1*) *to-Zp-closed to-Zp-minus*)

lemma *Qp-res-eqI'*:
assumes $a \in \mathcal{O}_p$
assumes $b \in \mathcal{O}_p$
assumes *val* $(a \ominus b) \geq n$
shows *Qp-res* $a\ n =$ *Qp-res* $b\ n$
apply(*rule Qp-res-eqI*, *rule assms*, *rule assms*, *rule Qp-res-eq-zeroI*)
using *assms* Q_p*-def Zp-def* ι*-def padic-fields.val-ring-minus-closed padic-fields-axioms*
apply *blast*
by(*rule assms*)

lemma *Qp-res-eqE*:
assumes $a \in \mathcal{O}_p$
assumes $b \in \mathcal{O}_p$
assumes *Qp-res* $a\ n =$ *Qp-res* $b\ n$
shows *val* $(a \ominus b) \geq n$
proof−
have *0*: *val* $(a \ominus b) =$ *val-Zp* (*to-Zp* $a \ominus_{Z_p}$ *to-Zp* b)
using *assms*
by (*metis to-Zp-minus to-Zp-val val-ring-minus-closed*)
have *1*: (*to-Zp* $a \ominus_{Z_p}$ *to-Zp* b) $n = 0$
using *assms* **unfolding** *Qp-res-def*
by (*meson val-ring-memE res-diff-zero-fact'' to-Zp-closed*)
have *2*: *val-Zp* (*to-Zp* $a \ominus_{Z_p}$ *to-Zp* b) $\geq n$
apply(*cases to-Zp* $a \ominus_{Z_p}$ *to-Zp* $b = \mathbf{0}_{Z_p}$)
proof −
assume *a1*: *to-Zp* $a \ominus_{Z_p}$ *to-Zp* $b = \mathbf{0}_{Z_p}$
have $\forall n$. *eint* (*int* n) $\leq$ *val-Zp* $\mathbf{0}_{Z_p}$
by (*metis* (*no-types*) *Zp.r-right-minus-eq Zp.zero-closed val-Zp-dist-def val-Zp-dist-res-eq2*)

then show *?thesis*
using *a1* **by** *presburger*
next
assume *a1*: *to-Zp a* $\ominus_{Z_p}$ *to-Zp b* $\neq \mathbf{0}_{Z_p}$
have *00*: *to-Zp a* $\ominus_{Z_p}$ *to-Zp b* $\in$ *carrier* Z_p
using *assms*
by (*meson val-ring-memE Zp.cring-simprules*(*4*) *to-Zp-closed*)
show *?thesis*
using *1 a1 ord-Zp-geq*[*of to-Zp a* $\ominus_{Z_p}$ *to-Zp b n*] *00*
val-ord-Zp[*of to-Zp a* $\ominus_{Z_p}$ *to-Zp b*] *eint-ord-code* **by** *metis*
qed
thus *?thesis* **unfolding** *0* **by** *blast*
qed

lemma *notin-closed*:
$(\neg\ ((c::eint) \leq x \wedge x \leq d)) = (x < c \vee d < x)$
by *auto*

lemma *Qp-res-neqI*:
assumes $a \in \mathcal{O}_p$
assumes $b \in \mathcal{O}_p$
assumes *val* $(a \ominus b) < n$
shows *Qp-res a n* $\neq$ *Qp-res b n*
apply(*rule ccontr*)
using *Qp-res-eqE*[*of a b n*] *assms*
using *notin-closed* **by** *blast*

lemma *Qp-res-equal*:
assumes $a \in \mathcal{O}_p$
assumes $l =$ *Qp-res a n*
shows *Qp-res a n* $=$ *Qp-res* $([l]\cdot\mathbf{1})$ *n*
unfolding *Qp-res-int-inc assms* **using** *assms Qp-res-mod-triv* **by** *presburger*

definition *Qp-res-class* **where**
Qp-res-class n b $= \{a \in \mathcal{O}_p.$ *Qp-res a n* $=$ *Qp-res b n*$\}$

definition *Qp-res-classes* **where**
Qp-res-classes n $=$ *Qp-res-class n* ‘ $\mathcal{O}_p$

lemma *val-ring-int-inc-closed*:
$[(k::int)]\cdot\mathbf{1} \in \mathcal{O}_p$
proof−
have *0*: $[(k::int)]\cdot\mathbf{1} = \iota\ ([(k::int)]\cdot_{Z_p}\mathbf{1}_{Z_p})$
using *inc-of-int* **by** *blast*
thus *?thesis*
by *blast*
qed

lemma *val-ring-nat-inc-closed*:

$[(k::nat)]\cdot \mathbf{1} \in \mathcal{O}_p$
proof −
have *0*: $[k]\cdot \mathbf{1} = \iota\ ([k]\cdot_{Z_p} \mathbf{1}_{Z_p})$
using *inc-of-nat* **by** *blast*
thus *?thesis*
by *blast*
qed

lemma *Qp-res-classes-wits*:
Qp-res-classes n = $(\lambda l::int.$ *Qp-res-class n* $([l]\cdot \mathbf{1}))$ ‘ (*carrier* (*Zp-res-ring n*))
proof −
obtain *F* **where** *F-def*: $F = (\lambda l::int.$ *Qp-res-class n* $([l]\cdot \mathbf{1}))$
by *blast*
have *0*: *Qp-res-classes n* = *F* ‘ (*carrier* (*Zp-res-ring n*))
proof(*rule equalityI*, *rule subsetI*)
fix *x* **assume** *A*: $x \in$ *Qp-res-classes n*
then obtain *a* **where** *a-def*: $a \in \mathcal{O}_p \wedge x =$ *Qp-res-class n a*
unfolding *Qp-res-classes-def* **by** *blast*
have *1*: *Qp-res a n* = *Qp-res* $([($*Qp-res a n*$)]\cdot \mathbf{1})$ *n*
apply(*rule Qp-res-equal*)
using *a-def* **apply** *blast* **by** *blast*
have *2*: *Qp-res-class n a* = *Qp-res-class n* $([($*Qp-res a n*$)]\cdot \mathbf{1})$
unfolding *Qp-res-class-def* **using** *1* **by** *metis*
have *3*: *x* = *Qp-res-class n* $([($*Qp-res a n*$)]\cdot \mathbf{1})$
using *a-def* **unfolding** *2* **by** *blast*
have *4*: $a \in \mathcal{O}_p$
using *a-def* **by** *blast*
show $x \in F$ ‘ *carrier* (*Zp-res-ring n*)
unfolding *F-def 3*
using *Qp-res-closed*[*of a n*] *4* **by** *blast*
next
show *F* ‘ *carrier* (*residue-ring* $(p \hat{} n)) \subseteq$ *Qp-res-classes n*
proof(*rule subsetI*)
fix *x* **assume** *A*: $x \in F$ ‘ (*carrier* (*Zp-res-ring n*))
then obtain *l* **where** *l-def*: $l \in$ *carrier* (*Zp-res-ring n*) $\wedge\ x = F\ l$
using *A* **by** *blast*
have *0*: $x = F\ l$
using *l-def* **by** *blast*
show $x \in$ *Qp-res-classes n*
unfolding *0 F-def Qp-res-classes-def* **using** *val-ring-int-inc-closed* **by** *blast*
qed
qed
then show *?thesis* **unfolding** *F-def* **by** *blast*
qed

lemma *Qp-res-classes-finite*:
finite (*Qp-res-classes n*)
by (*metis Qp-res-classes-wits finite-atLeastLessThan-int finite-imageI p-res-ring-car*)

definition *Qp-cong-set* **where**
Qp-cong-set α a = $\{x \in \mathcal{O}_p.$ *to-Zp* x α = a $\alpha\}$

lemma *Qp-cong-set-as-ball*:
assumes $a \in$ *carrier* Z_p
assumes a α = *0*
shows *Qp-cong-set* α a = $B_\alpha[\mathbf{0}]$
proof−
have *0*: ι $a \in$ *carrier* Q_p
using *assms inc-closed*[*of a*] **by** *blast*
show *?thesis*
proof
show *Qp-cong-set* α $a \subseteq B_\alpha[\mathbf{0}]$
proof fix x **assume** *A*: $x \in$ *Qp-cong-set* α a
show $x \in B_\alpha$ [$\mathbf{0}$]
proof(*rule c-ballI*)
show *t0*: $x \in$ *carrier* Q_p
using *A* **unfolding** *Qp-cong-set-def*
using *val-ring-memE* **by** *blast*
show *eint* (*int* α) $\leq$ *val* ($x \ominus \mathbf{0}$)
proof−
have *t1*: *to-Zp* x α = *0*
using *A* **unfolding** *Qp-cong-set-def*
by (*metis* (*mono-tags*, *lifting*) *assms*(*2*) *mem-Collect-eq*)
have *t2*: *val-Zp* (*to-Zp* x) $\geq \alpha$
apply(*cases to-Zp* $x = \mathbf{0}_{Z_p}$)
apply (*metis Zp.r-right-minus-eq Zp.zero-closed val-Zp-dist-def val-Zp-dist-res-eq2*)
using *ord-Zp-geq*[*of to-Zp* x α] *A* **unfolding** *Qp-cong-set-def*
by (*metis* (*no-types*, *lifting*) *val-ring-memE eint-ord-simps*(*1*) *t1*
to-Zp-closed to-Zp-def val-ord-Zp)
then show *?thesis* **using** *A* **unfolding** *Qp-cong-set-def mem-Collect-eq*
using *val-ring-memE*
by (*metis Qp-res-eqE Qp-res-eq-zeroI Qp-res-zero to-Zp-val zero-in-val-ring*)
qed
qed
qed
show $B_{int\ \alpha}[\mathbf{0}] \subseteq$ *Qp-cong-set* α a
proof fix x **assume** *A*: $x \in B_{int\ \alpha}[\mathbf{0}]$
then have *0*: *val* $x \geq \alpha$
using *assms c-ballE*[*of* x α $\mathbf{0}$]
by (*smt Qp.minus-closed Qp.r-right-minus-eq Qp-diff-diff*)
have *1*: *to-Zp* $x \in$ *carrier* Z_p
using *A 0 assms c-ballE*(*1*) *to-Zp-closed* **by** *blast*
have *2*: $x \in \mathcal{O}_p$
using *0 A val-ringI c-ballE*
by (*smt* Q_p*-def Zp-def* ι*-def eint-ord-simps*(*1*) *of-nat-0 of-nat-le-0-iff*
val-ring-ord-criterion padic-fields-axioms val-ord' zero-in-val-ring)
then have *val-Zp* (*to-Zp* x) $\geq \alpha$
using *0 1 A assms c-ballE*[*of* x α $\mathbf{0}$] *to-Zp-val* **by** *presburger*

```
    then have to-Zp x α = 0
      using 1 zero-below-val-Zp by blast
    then show  x ∈ Qp-cong-set α a
      unfolding Qp-cong-set-def using assms(2) 2
      by (metis (mono-tags, lifting) mem-Collect-eq)
  qed
 qed
qed

lemma Qp-cong-set-as-ball':
 assumes a ∈ carrier Z_p
 assumes val-Zp a < eint (int α)
 shows Qp-cong-set α a = B_α[(ι a)]
proof
 show Qp-cong-set α a ⊆ B_α[ι a]
 proof fix x
  assume A:  x ∈ Qp-cong-set α a
  then have 0: to-Zp x α = a α
    unfolding Qp-cong-set-def  by blast
  have 1:  x ∈ 𝒪_p
    using A unfolding Qp-cong-set-def by blast
  have 2: to-Zp x ∈ carrier Z_p
    using 1 val-ring-memE to-Zp-closed by blast
  have 3: val-Zp (to-Zp x ⊖_{Z_p} a) ≥ α
    using 0 assms 2 val-Zp-dist-def val-Zp-dist-res-eq2 by presburger
  have 4: val-Zp (to-Zp x ⊖_{Z_p} a) > val-Zp a
     using 3 assms(2) less-le-trans[of val-Zp a eint (int α) val-Zp (to-Zp x ⊖_{Z_p}
a) ]
    by blast
  then have 5: val-Zp (to-Zp x) = val-Zp a
    using assms 2 equal-val-Zp by blast
  have 7: val (x ⊖ (ι a)) ≥ α
     using 3 5 1 by (metis 2 Zp.minus-closed assms(1) inc-of-diff to-Zp-inc
val-of-inc)
  then show x ∈ B_{int α}[ι a]
    using c-ballI[of x α ι a]  1 assms val-ring-memE by blast
 qed
 show B_{int α}[ι a] ⊆ Qp-cong-set α a
 proof fix x
  assume A: x ∈ B_{int α}[ι a]
  then have 0: val (x ⊖ ι a) ≥ α
    using c-ballE by blast
  have 1: val (ι a) = val-Zp a
    using assms  Zp-def ι-def padic-fields.val-of-inc padic-fields-axioms
    by metis
  then have 2: val (x ⊖ ι a) > val (ι a)
    using 0 assms less-le-trans[of val (ι a) eint (int α) val (x ⊖ ι a)]
    by metis
  have ι a ∈ carrier Q_p
```

```
    using assms(1) inc-closed by blast
  then have B: val x = val (ι a)
    using 2 A assms c-ballE(1)[of x α ι a]
    by (metis ultrametric-equal-eq)
  have 3: val-Zp (to-Zp x) = val-Zp a
    by (metis 1 A ‹val x = val (ι a)› assms(1) c-ballE(1) to-Zp-val val-pos
val-ringI)
  have 4: val-Zp (to-Zp x ⊖_{Z_p} a) ≥ α
    using 0 A 3
    by (metis B Zp.minus-closed assms(1) c-ballE(1) inc-of-diff to-Zp-closed
to-Zp-inc val-of-inc val-pos val-ring-val-criterion)
  then have 5: to-Zp x α = a α
  by (meson A Zp.minus-closed assms(1) c-ballE(1) res-diff-zero-fact(1) to-Zp-closed
zero-below-val-Zp)
  have 6: x ∈ 𝒪_p
  proof−
    have val x ≥ 0
      using B assms 1 val-pos by presburger
    then show ?thesis
      using A c-ballE(1) val-ringI by blast
  qed
  then show x ∈ Qp-cong-set α a unfolding Qp-cong-set-def
    using 5 by blast
 qed
qed

lemma Qp-cong-set-is-univ-semialgebraic:
 assumes a ∈ carrier Z_p
 shows is-univ-semialgebraic (Qp-cong-set α a)
proof(cases a α = 0)
 case True
 then show ?thesis
   using ball-is-univ-semialgebraic[of 0 α] Qp.zero-closed Qp-cong-set-as-ball
assms
  by metis
next
 case False
 then have α ≠ 0
  using assms residues-closed[of a 0]
  by (meson p-res-ring-0')
 then obtain n where n-def: Suc n = α
  by (metis lessI less-Suc-eq-0-disj)
 then have val-Zp a < eint (int α)
  using below-val-Zp-zero[of a n]
 by (smt False assms eint-ile eint-ord-simps(1) eint-ord-simps(2) zero-below-val-Zp)
 then show ?thesis
  using ball-is-univ-semialgebraic[of ι a α] Qp.zero-closed Qp-cong-set-as-ball'[of
a α] assms
      inc-closed by presburger
```

qed

lemma *constant-res-set-semialg*:
assumes $l \in$ *carrier* (*Zp-res-ring n*)
shows *is-univ-semialgebraic* $\{x \in \mathcal{O}_p.$ *Qp-res x n* $= l\}$
proof−
have *0*: $\{x \in \mathcal{O}_p.$ *Qp-res x n* $= l\}$ = *Qp-cong-set n* $([l]\cdot_{Z_p} \mathbf{1}_{Z_p})$
apply(*rule equalityI'*)
unfolding *mem-Collect-eq Qp-cong-set-def Qp-res-def*
apply (*metis val-ring-memE Zp-int-inc-rep nat-le-linear p-residue-padic-int to-Zp-closed*)
using *assms*
by (*metis Zp-int-inc-res mod-pos-pos-trivial p-residue-ring-car-memE*(*1*) *p-residue-ring-car-memE*(*2*))
show *?thesis* **unfolding** *0*
apply(*rule Qp-cong-set-is-univ-semialgebraic*)
by(*rule Zp.int-inc-closed*)
qed

end

end
theory *Padic-Semialgebraic-Function-Ring*
imports *Padic-Field-Powers*
begin

14 Rings of Semialgebraic Functions

In order to efficiently formalize Denef's proof of Macintyre's theorem, it is necessary to be able to reason about semialgebraic functions algebraically. For example, we need to consider polynomials in one variable whose coefficients are semialgebraic functions, and take their Taylor expansions centered at a semialgebraic function. To facilitate this kind of reasoning, it is necessary to construct, for each arity m, a ring `SA(m)` of semialgebraic functions in m variables. These functions must be extensional functions which are undefined outside of the carrier set of $\mathbb{Q}_p^m$.

The developments in this theory are mainly lemmas and defintitions which build the necessary theory to prove the cell decomposition theorems of [1], and finally Macintyre's Theorem, which says that semi-algebraic sets are closed under projections.

14.1 Some eint Arithmetic

context *padic-fields*
begin

lemma *eint-minus-ineq'*:

assumes $a \leq eint\ N$
assumes $b - a \leq c$
shows $b - eint\ N \leq c$
using *assms* **by**(*induction c*, *induction b*, *induction a*, *auto*)

lemma *eint-minus-plus*:
$a - (eint\ b + eint\ c) = a - eint\ b - eint\ c$
apply(*induction a*)
apply (*metis diff-add-eq-diff-diff-swap idiff-eint-eint plus-eint-simps*(*1*) *semiring-normalization-rules*(*24*))
using *idiff-infinity* **by** *presburger*

lemma *eint-minus-plus'*:
$a - (eint\ b + eint\ c) = a - eint\ c - eint\ b$
by (*metis add.commute eint-minus-plus*)

lemma *eint-minus-plus''*:
assumes $a - eint\ c - eint\ b = eint\ f$
shows $a - eint\ c - eint\ f = eint\ b$
using *assms* **apply**(*induction a*)
apply (*metis add.commute add-diff-cancel-eint eint.distinct*(*2*) *eint-add-cancel-fact*)
by *simp*

lemma *uminus-involutive*[*simp*]:
$-(-x::eint) = x$
apply(*induction x*)
unfolding *uminus-eint-def* **by** *auto*

lemma *eint-minus*:
$(a::eint) - (b::eint) = a + (-b)$
apply(*induction a*)
apply(*induction b*)
proof −
fix *int* :: *int* **and** *inta* :: *int*
have $\forall\ e\ ea.\ (ea::eint) + (e + - ea) = ea - ea + e$
by (*simp add*: *eint-uminus-eq*)
then have $\forall\ i\ ia.\ eint\ (ia + i) + - eint\ ia = eint\ i$
by (*metis ab-group-add-class.ab-diff-conv-add-uminus add.assoc add-minus-cancel idiff-eint-eint plus-eint-simps*(*1*))
then show *eint inta* − *eint int* = *eint inta* + − *eint int*
by (*metis ab-group-add-class.ab-diff-conv-add-uminus add.commute add-minus-cancel idiff-eint-eint*)
next
show $\bigwedge int.\ eint\ int - \infty = eint\ int + - \infty$
by (*metis eint-uminus-eq i0-ne-infinity idiff-infinity-right idiff-self plus-eq-infty-iff-eint uminus-involutive*)
show $\infty - b = \infty + - b$
apply(*induction b*)
apply *simp*

by *auto*
qed

lemma *eint-mult-Suc*:
eint (*Suc k*) ∗ *a* = *eint k* ∗ *a* + *a*
apply(*induction a*)
apply (*metis add.commute eSuc-eint mult-eSuc′ of-nat-Suc*)
using *plus-eint-simps*(*3*) *times-eint-simps*(*4*)
by *presburger*

lemma *eint-mult-Suc-mono*:
assumes *a* ≤ *eint b* ⟶ *eint* (*int k*) ∗ *a* ≤ *eint* (*int k*) ∗ *eint b*
shows *a* ≤ *eint b* ⟶ *eint* (*int* (*Suc k*)) ∗ *a* ≤ *eint* (*int* (*Suc k*)) ∗ *eint b*
using *assms eint-mult-Suc*
by (*metis add-mono-thms-linordered-semiring*(*1*))

lemma *eint-nat-mult-mono*:
assumes (*a*::*eint*) ≤ *b*
shows *eint* (*k*::*nat*)∗*a* ≤ *eint k*∗*b*
proof−
have (*a*::*eint*) ≤ *b* ⟶ *eint* (*k*::*nat*)∗*a* ≤ *eint k*∗*b*
apply(*induction k*) **apply**(*induction b*)
apply (*metis eint-ile eq-iff mult-not-zero of-nat-0 times-eint-simps*(*1*))
apply *simp*
apply(*induction b*)
using *eint-mult-Suc-mono* **apply** *blast*
using *eint-ord-simps*(*3*) *times-eint-simps*(*4*) **by** *presburger*
thus *?thesis* **using** *assms* **by** *blast*
qed

lemma *eint-Suc-zero*:
eint (*int* (*Suc 0*)) ∗ *a* = *a*
apply(*induction a*)
apply *simp*
by *simp*

lemma *eint-add-mono*:
assumes (*a*::*eint*) ≤ *b*
assumes (*c*::*eint*) ≤ *d*
shows *a* + *c* ≤ *b* + *d*
using *assms*
by (*simp add*: *add-mono*)

lemma *eint-nat-mult-mono-rev*:
assumes *k* > *0*
assumes *eint* (*k*::*nat*)∗*a* ≤ *eint k*∗*b*
shows (*a*::*eint*) ≤ *b*
proof(*rule ccontr*)
assume ¬ *a* ≤ *b*

then have *A*: $b < a$
using *leI* **by** *blast*
have $b < a \longrightarrow$ *eint* (k::*nat*)$*b <$ *eint* $k*a$
apply(*induction b*) **apply**(*induction a*)
using *A assms eint-ord-simps*(*2*) *times-eint-simps*(*1*) *zmult-zless-mono2-lemma*
apply *presburger*
using *eint-ord-simps*(*4*) *nat-mult-not-infty times-eint-simps*(*4*) **apply** *presburger*
using *eint-ord-simps*(*6*) **by** *blast*
then have *eint* (k::*nat*)$*b <$ *eint* $k*a$
using *A* **by** *blast*
hence ¬ *eint* (k::*nat*)$*a \leq$ *eint* $k*b$
by (*metis* ‹¬ $a \leq b$› *antisym eint-nat-mult-mono linear neq-iff*)
then show *False* **using** *assms* **by** *blast*
qed

14.2 Lemmas on Function Ring Operations

lemma *Qp-funs-is-cring*:
cring (*Fun*$_n$ Q_p)
using *F.M-cring* **by** *blast*

lemma *Qp-funs-is-monoid*:
monoid (*Fun*$_n$ Q_p)
using *F.is-monoid* **by** *blast*

lemma *Qp-funs-car-memE*:
assumes $f \in$ *carrier* (*Fun*$_n$ Q_p)
shows $f \in$ (*carrier* (${Q_p}^n$)) $\to$ (*carrier* Q_p)
by (*simp add*: *assms ring-pow-function-ring-car-memE*(*2*))

lemma *Qp-funs-car-memI*:
assumes $g \in$ *carrier* (${Q_p}^n$) $\to$ *carrier* Q_p
assumes $\bigwedge x.\ x \notin$ (*carrier* (${Q_p}^n$)) $\Longrightarrow g\ x =$ *undefined*
shows $g \in$ *carrier* (*Fun*$_n$ Q_p)
apply(*rule Qp.function-ring-car-memI*)
using *assms* **apply** *blast*
using *assms* **by** *blast*

lemma *Qp-funs-car-memI'*:
assumes $g \in$ *carrier* (${Q_p}^n$) $\to$ *carrier* Q_p
assumes *restrict* g (*carrier* (${Q_p}^n$)) $= g$
shows $g \in$ *carrier* (*Fun*$_n$ Q_p)
apply(*intro Qp-funs-car-memI assms*)
using *assms* **unfolding** *restrict-def*
by (*metis* (*mono-tags*, *lifting*))

lemma *Qp-funs-car-memI''*:
assumes $f \in$ *carrier* (${Q_p}^n$) $\to$ *carrier* Q_p

assumes $g = (\lambda\ x \in (carrier\ ({Q_p}^n)).\ f\ x)$
shows $g \in carrier\ (Fun_n\ Q_p)$
apply(*rule Qp-funs-car-memI*)
using *assms*
apply (*meson restrict-Pi-cancel*)
by (*metis assms*(*2*) *restrict-def*)

lemma *Qp-funs-one*:
$\mathbf{1}_{Fun_n\ Q_p} = (\lambda\ x \in carrier\ ({Q_p}^n).\ \mathbf{1})$
unfolding *function-ring-def function-one-def*
by (*meson monoid.select-convs*(*2*))

lemma *Qp-funs-zero*:
$\mathbf{0}_{Fun_n\ Q_p} = (\lambda\ x \in carrier\ ({Q_p}^n).\ \mathbf{0}_{Q_p})$
unfolding *function-ring-def function-zero-def*
by (*meson ring-record-simps*(*11*))

lemma *Qp-funs-add*:
assumes $x \in carrier\ ({Q_p}^n)$
assumes $f \in (carrier\ ({Q_p}^n)) \to carrier\ Q_p$
assumes $g \in (carrier\ ({Q_p}^n)) \to carrier\ Q_p$
shows $(f \oplus_{Fun_n\ Q_p} g)\ x = f\ x \oplus_{Q_p} g\ x$
using *assms function-ring-def*[*of carrier* $({Q_p}^n)$ Q_p]
unfolding *function-add-def*
by (*metis* (*mono-tags*, *lifting*) *restrict-apply*′ *ring-record-simps*(*12*))

lemma *Qp-funs-add*′:
assumes $x \in carrier\ ({Q_p}^n)$
assumes $f \in (carrier\ (Fun_n\ Q_p))$
assumes $g \in (carrier\ (Fun_n\ Q_p))$
shows $(f \oplus_{Fun_n\ Q_p} g)\ x = f\ x \oplus_{Q_p} g\ x$
using *assms Qp-funs-add Qp-funs-car-memE*
by *blast*

lemma *Qp-funs-add*′′:
assumes $f \in (carrier\ (Fun_n\ Q_p))$
assumes $g \in (carrier\ (Fun_n\ Q_p))$
shows $(f \oplus_{Fun_n\ Q_p} g) = (\lambda\ x \in carrier\ ({Q_p}^n).\ f\ x \oplus_{Q_p} g\ x)$
unfolding *function-ring-def function-add-def* **using** *ring-record-simps*(*12*)
by *metis*

lemma *Qp-funs-add*′′′:
assumes $x \in carrier\ ({Q_p}^n)$
shows $(f \oplus_{Fun_n\ Q_p} g)\ x = f\ x \oplus_{Q_p} g\ x$
using *assms function-ring-def*[*of carrier* $({Q_p}^n)$ Q_p]
unfolding *function-add-def*
by (*metis* (*mono-tags*, *lifting*) *restrict-apply*′ *ring-record-simps*(*12*))

lemma *Qp-funs-mult*:

assumes $x \in carrier\ ({Q_p}^n)$
assumes $f \in (carrier\ ({Q_p}^n)) \to carrier\ Q_p$
assumes $g \in (carrier\ ({Q_p}^n)) \to carrier\ Q_p$
shows $(f \otimes_{Fun_n\ Q_p} g)\ x = f\ x \otimes g\ x$
using *assms function-ring-def*[*of carrier* $({Q_p}^n)$ Q_p]
unfolding *function-mult-def*
by (*metis* (*no-types*, *lifting*) *monoid.select-convs*(*1*) *restrict-apply'*)

lemma *Qp-funs-mult'*:
assumes $x \in carrier\ ({Q_p}^n)$
assumes $f \in (carrier\ (Fun_n\ Q_p))$
assumes $g \in (carrier\ (Fun_n\ Q_p))$
shows $(f \otimes_{Fun_n\ Q_p} g)\ x = f\ x \otimes g\ x$
using *assms Qp-funs-mult Qp-funs-car-memE*
by *blast*

lemma *Qp-funs-mult''*:
assumes $f \in (carrier\ (Fun_n\ Q_p))$
assumes $g \in (carrier\ (Fun_n\ Q_p))$
shows $(f \otimes_{Fun_n\ Q_p} g) = (\lambda\ x \in carrier\ ({Q_p}^n).\ f\ x \otimes g\ x)$
unfolding *function-ring-def function-mult-def* **using** *ring-record-simps*(*5*)
by *metis*

lemma *Qp-funs-mult'''*:
assumes $x \in carrier\ ({Q_p}^n)$
shows $(f \otimes_{Fun_n\ Q_p} g)\ x = f\ x \otimes g\ x$
using *assms function-ring-def*[*of carrier* $({Q_p}^n)$ Q_p]
unfolding *function-mult-def*
by (*metis* (*mono-tags*, *lifting*) *monoid.select-convs*(*1*) *restrict-apply'*)

lemma *Qp-funs-a-inv*:
assumes $x \in carrier\ ({Q_p}^n)$
assumes $f \in (carrier\ (Fun_n\ Q_p))$
shows $(\ominus_{Fun_n\ Q_p} f)\ x = \ominus\ (f\ x)$
using *assms local.function-uminus-eval*
by (*simp add*: *local.function-uminus-eval''*)

lemma *Qp-funs-a-inv'*:
assumes $f \in (carrier\ (Fun_n\ Q_p))$
shows $(\ominus_{Fun_n\ Q_p} f) = (\lambda\ x \in carrier\ ({Q_p}^n).\ \ominus\ (f\ x))$
proof fix x
show $(\ominus_{Fun_n\ Q_p} f)\ x = (\lambda x \in carrier\ ({Q_p}^n).\ \ominus f\ x)\ x$
apply(*cases* $x \in carrier\ ({Q_p}^n)$)
apply (*metis* (*no-types*, *lifting*) *Qp-funs-a-inv assms restrict-apply'*)
by (*simp add*: *assms local.function-ring-not-car*)
qed

abbreviation(*input*) *Qp-const* (‹c_-›) **where**
Qp-const n c ≡ *constant-function* $(carrier\ ({Q_p}^n))$ *c*

lemma *Qp-constE*:
assumes $c \in carrier\ Q_p$
assumes $x \in carrier\ ({Q_p}^n)$
shows *Qp-const n c x* $= c$
using *assms* **unfolding** *constant-function-def*
by (*meson restrict-apply'*)

lemma *Qp-funs-Units-memI*:
assumes $f \in (carrier\ (Fun_n\ Q_p))$
assumes $\bigwedge x.\ x \in carrier\ ({Q_p}^n) \Longrightarrow\ f\ x \neq \mathbf{0}_{Q_p}$
shows $f \in (Units\ (Fun_n\ Q_p))$
$inv_{Fun_n\ Q_p}\ f = (\lambda\ x \in carrier\ ({Q_p}^n).\ inv_{Q_p}\ (f\ x))$
proof−
obtain *g* **where** *g-def*: $g = (\lambda\ x \in carrier\ ({Q_p}^n).\ inv_{Q_p}\ (f\ x))$
by *blast*
have *g-closed*: $g \in (carrier\ (Fun_n\ Q_p))$
by(*rule Qp-funs-car-memI*, *unfold g-def*, *auto*,
intro field-inv(*3*) *assms Qp.function-ring-car-memE*[*of - n*], *auto*)
have $\bigwedge x.\ x \in carrier\ ({Q_p}^n) \Longrightarrow f\ x \otimes g\ x = \mathbf{1}$
using *assms g-def*
by (*metis* (*no-types*, *lifting*) *Qp.function-ring-car-memE field-inv*(*2*) *restrict-apply*)
then have *0*: $f \otimes_{Fun_n\ Q_p}\ g = \mathbf{1}_{Fun_n\ Q_p}$
using *assms g-def Qp-funs-mult''*[*of f n g*] *Qp-funs-one*[*of n*] *g-closed*
by (*metis* (*no-types*, *lifting*) *restrict-ext*)
then show $f \in (Units\ (Fun_n\ Q_p))$
using *comm-monoid.UnitsI*[*of Fun*$_n$ Q_p] *assms*(*1*) *g-closed local.F.comm-monoid-axioms*
by *presburger*
have $inv_{Fun_n\ Q_p}\ f = g$
using *g-def g-closed 0 cring.invI*[*of Fun*$_n$ Q_p] *Qp-funs-is-cring assms*(*1*)
by *presburger*
show $inv_{Fun_n\ Q_p}\ f = (\lambda\ x \in carrier\ ({Q_p}^n).\ inv_{Q_p}\ (f\ x))$
using *assms g-def 0* ‹$inv_{Fun_n\ Q_p}\ f = g$›
by *blast*
qed

lemma *Qp-funs-Units-memE*:
assumes $f \in (Units\ (Fun_n\ Q_p))$
shows $f \otimes_{Fun_n\ Q_p}\ inv_{Fun_n\ Q_p}\ f = \mathbf{1}_{Fun_n\ Q_p}$
$inv_{Fun_n\ Q_p}\ f \otimes_{Fun_n\ Q_p}\ f = \mathbf{1}_{Fun_n\ Q_p}$
$\bigwedge x.\ x \in carrier\ ({Q_p}^n) \Longrightarrow\ f\ x \neq \mathbf{0}_{Q_p}$
using *monoid.Units-r-inv*[*of Fun*$_n$ Q_p *f*] *assms Qp-funs-is-monoid*
apply *blast*
using *monoid.Units-l-inv*[*of Fun*$_n$ Q_p *f*] *assms Qp-funs-is-monoid*
apply *blast*
proof−
obtain *g* **where** *g-def*: $g = inv_{Fun_n\ Q_p}\ f$
by *blast*
show $\bigwedge x.\ x \in carrier\ ({Q_p}^n) \Longrightarrow\ f\ x \neq \mathbf{0}_{Q_p}$

proof− **fix** x **assume** A: $x \in carrier\ ({Q_p}^n)$
have $f \otimes_{Fun_n\ Q_p} g = \mathbf{1}_{Fun_n\ Q_p}$
using *assms g-def Qp-funs-is-monoid*
‹⟦*Group.monoid* (*Fun*$_n$ Q_p); $f \in$ *Units* (*Fun*$_n$ Q_p)⟧ ⟹ $f \otimes_{Fun_n\ Q_p}$ $inv_{Fun_n\ Q_p}$ $f = \mathbf{1}_{Fun_n\ Q_p}$›
by *blast*
then have *0*: $f\ x \otimes g\ x = \mathbf{1}$
using *A assms g-def Qp-funs-mult'*[*of x n f g*] *Qp-funs-one*[*of n*]
by (*metis Qp-funs-is-monoid monoid.Units-closed monoid.Units-inv-closed restrict-apply*)
have *1*: $g\ x \in carrier\ Q_p$
using *g-def A assms local.function-ring-car-closed* **by** *auto*
then show $f\ x \neq \mathbf{0}_{Q_p}$
using *0*
by (*metis Qp.l-null local.one-neq-zero*)
qed
qed

lemma *Qp-funs-m-inv*:
assumes $x \in carrier\ ({Q_p}^n)$
assumes $f \in (Units\ (Fun_n\ Q_p))$
shows $(inv_{Fun_n\ Q_p}\ f)\ x = inv_{Q_p}\ (f\ x)$
using *Qp-funs-Units-memI*(*2*) *Qp-funs-Units-memE*(*3*) *assms*
by (*metis* (*no-types*, *lifting*) *Qp-funs-is-monoid monoid.Units-closed restrict-apply*)

14.3 Defining the Rings of Semialgebraic Functions

definition *semialg-functions* **where**
semialg-functions n = {$f \in (carrier\ ({Q_p}^n)) \to carrier\ Q_p$. *is-semialg-function n f* ∧ f = *restrict f* (*carrier* (${Q_p}^n$))}

lemma *semialg-functions-memE*:
assumes $f \in$ *semialg-functions n*
shows *is-semialg-function n f*
$f \in carrier\ (Fun_n\ Q_p)$
$f \in carrier\ ({Q_p}^n) \to carrier\ Q_p$
using *semialg-functions-def assms* **apply** *blast*
apply(*rule Qp-funs-car-memI'*)
using *assms*
apply (*metis* (*no-types*, *lifting*) *mem-Collect-eq semialg-functions-def*)
using *assms* **unfolding** *semialg-functions-def*
apply (*metis* (*mono-tags*, *lifting*) *mem-Collect-eq*)
by (*metis* (*no-types*, *lifting*) *assms mem-Collect-eq semialg-functions-def*)

lemma *semialg-functions-in-Qp-funs*:
semialg-functions n ⊆ *carrier* (*Fun*$_n$ Q_p)
using *semialg-functions-memE*
by *blast*

lemma *semialg-functions-memI*:
assumes $f \in$ *carrier* $(Fun_n\ Q_p)$
assumes *is-semialg-function n f*
shows $f \in$ *semialg-functions n*
using *assms* **unfolding** *semialg-functions-def*
by (*metis* (*mono-tags*, *lifting*) *Qp-funs-car-memI function-ring-car-eqI*
is-semialg-function-closed mem-Collect-eq restrict-Pi-cancel restrict-apply)

lemma *restrict-is-semialg*:
assumes *is-semialg-function n f*
shows *is-semialg-function n* (*restrict f* (*carrier* $({Q_p}^n)$))
proof(*rule is-semialg-functionI*)
show *0*: *restrict f* (*carrier* $({Q_p}^n)$) $\in$ *carrier* $({Q_p}^n) \to$ *carrier* Q_p
using *assms is-semialg-function-closed* **by** *blast*
show $\bigwedge k\ S.\ S \in$ *semialg-sets* $(1 + k) \Longrightarrow$ *is-semialgebraic* $(n + k)$ (*partial-pullback*
n (*restrict f* (*carrier* $({Q_p}^n)$)) *k S*)
proof− **fix** *k S* **assume** *A*: $S \in$ *semialg-sets* $(1 + k)$
have (*partial-pullback n* (*restrict f* (*carrier* $({Q_p}^n)$)) *k S*) = *partial-pullback n f*
k S
apply(*intro equalityI' partial-pullback-memI*, *meson partial-pullback-memE*)
unfolding *partial-pullback-def partial-image-def evimage-eq restrict-def*
apply (*metis* (*mono-tags*, *lifting*) *le-add1 local.take-closed*)
apply *blast*
by (*metis* (*mono-tags*, *lifting*) *le-add1 local.take-closed*)
then show *is-semialgebraic* $(n + k)$ (*partial-pullback n* (*restrict f* (*carrier*
$({Q_p}^n)$)) *k S*)
using *assms A is-semialg-functionE is-semialgebraicI*
by *presburger*
qed
qed

lemma *restrict-in-semialg-functions*:
assumes *is-semialg-function n f*
shows (*restrict f* (*carrier* $({Q_p}^n)$)) $\in$ *semialg-functions n*
using *assms restrict-is-semialg*
unfolding *semialg-functions-def*
by (*metis* (*mono-tags*, *lifting*) *is-semialg-function-closed mem-Collect-eq restrict-apply'*
restrict-ext)

lemma *constant-function-is-semialg*:
assumes $a \in$ *carrier* Q_p
shows *is-semialg-function n* (*constant-function* (*carrier* $({Q_p}^n)$) *a*)
proof−
have (*constant-function* (*carrier* $({Q_p}^n)$) *a*) = *restrict* (*Qp-ev* (*Qp.indexed-const*
a)) (*carrier* $({Q_p}^n)$)
apply(*rule ext*)
unfolding *constant-function-def*
using *eval-at-point-const assms* **by** *simp*
then show *?thesis* **using** *restrict-in-semialg-functions poly-is-semialg*[*of Qp.indexed-const*

a]
using *assms*(*1*) *Qp-to-IP-car restrict-is-semialg* **by** *presburger*
qed

lemma *constant-function-in-semialg-functions*:
assumes $a \in$ *carrier* Q_p
shows *Qp-const n a* $\in$ *semialg-functions n*
apply(*unfold semialg-functions-def constant-function-def mem-Collect-eq*, *intro conjI*, *auto simp*: *assms*)
using *assms constant-function-is-semialg*[*of a n*] **unfolding** *constant-function-def*
by *auto*

lemma *function-one-as-constant*:
$\mathbf{1}_{Fun_n\ Q_p}$ = *Qp-const n* **1**
unfolding *constant-function-def function-ring-def*[*of carrier* $({Q_p}^n)$ Q_p] *function-one-def*
by *simp*

lemma *function-zero-as-constant*:
$\mathbf{0}_{Fun_n\ Q_p}$ = *Qp-const n* $\mathbf{0}_{Q_p}$
unfolding *constant-function-def function-ring-def*[*of carrier* $({Q_p}^n)$ Q_p] *function-zero-def*
by *simp*

lemma *sum-in-semialg-functions*:
assumes *f* $\in$ *semialg-functions n*
assumes *g* $\in$ *semialg-functions n*
shows $f \oplus_{Fun_n\ Q_p} g \in$ *semialg-functions n*
proof−
have *0*:$f \oplus_{Fun_n\ Q_p} g$ = *restrict* (*function-tuple-comp* Q_p [*f*,*g*] *Qp-add-fun*) (*carrier* $({Q_p}^n)$)
proof fix *x*
show $(f \oplus_{Fun_n\ Q_p} g)\ x$ = *restrict* (*function-tuple-comp* Q_p [*f*, *g*] *Qp-add-fun*) (*carrier* $({Q_p}^n)$) *x*
proof(*cases x* $\in$ *carrier* $({Q_p}^n)$)
case *True*
have *restrict* (*function-tuple-comp* Q_p [*f*, *g*] *Qp-add-fun*) (*carrier* $({Q_p}^n)$) *x* = *Qp-add-fun* [*f x*, *g x*]
unfolding *function-tuple-comp-def function-tuple-eval-def restrict-def*
using *comp-apply*[*of Qp-add-fun* (λ*x*. *map* (λ*f*. *f x*) [*f*, *g*]) *x*]
by (*metis* (*no-types*, *lifting*) *True list.simps*(*8*) *list.simps*(*9*))
then have *restrict* (*function-tuple-comp* Q_p [*f*, *g*] *Qp-add-fun*) (*carrier* $({Q_p}^n)$) *x* = *f x* $\oplus_{Q_p}$ *g x*
unfolding *Qp-add-fun-def*
by (*metis One-nat-def nth-Cons-0 nth-Cons-Suc*)
then show *?thesis* **using** *True function-ring-def*[*of carrier* $({Q_p}^n)$ Q_p]
unfolding *function-add-def*
by (*metis* (*no-types*, *lifting*) *Qp-funs-add assms*(*1*) *assms*(*2*) *mem-Collect-eq semialg-functions-def*)

next
case *False*
have $(f \oplus_{Funn\ Q_p}\ g)\ x = undefined$
using *function-ring-def*[*of carrier* $(Q_p{}^n)$ Q_p] **unfolding** *function-add-def*
by (*metis* (*mono-tags*, *lifting*) *False restrict-apply ring-record-simps*(*12*))
then show *?thesis*
by (*metis False restrict-def*)
qed
qed
have *1*: *is-semialg-function-tuple n* [*f*, *g*]
using *assms is-semialg-function-tupleI*[*of* [*f*, *g*] *n*] *semialg-functions-memE*
by (*metis list.distinct*(*1*) *list.set-cases set-ConsD*)
have *2*: *is-semialg-function n* (*function-tuple-comp* Q_p [*f*,*g*] *Qp-add-fun*)
apply(*rule semialg-function-tuple-comp*[*of - - 2*])
apply (*simp add*: *1*)
apply *simp*
by (*simp add*: *addition-is-semialg*)
show *?thesis*
apply(*rule semialg-functions-memI*)
apply (*meson Qp-funs-is-cring assms*(*1*) *assms*(*2*) *cring.cring-simprules*(*1*) *semialg-functions-memE*(*2*))
using *0 2 restrict-is-semialg* **by** *presburger*
qed

lemma *prod-in-semialg-functions*:
assumes $f \in$ *semialg-functions n*
assumes $g \in$ *semialg-functions n*
shows $f \otimes_{Funn\ Q_p}\ g \in$ *semialg-functions n*
proof−
have *0*:$f \otimes_{Funn\ Q_p}\ g =$ *restrict* (*function-tuple-comp* Q_p [*f*,*g*] *Qp-mult-fun*) (*carrier* $(Q_p{}^n)$)
proof fix *x*
show $(f \otimes_{Funn\ Q_p}\ g)\ x =$ *restrict* (*function-tuple-comp* Q_p [*f*, *g*] *Qp-mult-fun*) (*carrier* $(Q_p{}^n)$) *x*
proof(*cases* $x \in$ *carrier* $(Q_p{}^n)$)
case *True*
have *restrict* (*function-tuple-comp* Q_p [*f*, *g*] *Qp-mult-fun*) (*carrier* $(Q_p{}^n)$) *x* = *Qp-mult-fun* [*f x*, *g x*]
unfolding *function-tuple-comp-def function-tuple-eval-def restrict-def*
using *comp-apply*[*of Qp-mult-fun* (λ*x*. *map* (λ*f*. *f x*) [*f*, *g*]) *x*]
by (*metis* (*no-types*, *lifting*) *True list.simps*(*8*) *list.simps*(*9*))
then have *restrict* (*function-tuple-comp* Q_p [*f*, *g*] *Qp-mult-fun*) (*carrier* $(Q_p{}^n)$) *x* = *f x* ⊗ *g x*
unfolding *Qp-mult-fun-def*
by (*metis One-nat-def nth-Cons-0 nth-Cons-Suc*)
then show *?thesis* **using** *True function-ring-def*[*of carrier* $(Q_p{}^n)$ Q_p]
unfolding *function-mult-def*
by (*metis* (*no-types*, *lifting*) *Qp-funs-mult assms*(*1*) *assms*(*2*) *mem-Collect-eq semialg-functions-def*)

```
    next
      case False
      have (f ⊗_Funn Q_p g) x = undefined
        using function-ring-def[of carrier (Q_p^n) Q_p] unfolding function-mult-def
        by (metis (mono-tags, lifting) False restrict-apply ring-record-simps(5))
      then show ?thesis
        by (metis False restrict-def)
    qed
  qed
  have 1: is-semialg-function-tuple n [f, g]
    using assms is-semialg-function-tupleI[of [f, g] n] semialg-functions-memE
    by (metis list.distinct(1) list.set-cases set-ConsD)
  have 2: is-semialg-function n (function-tuple-comp Q_p [f,g] Qp-mult-fun)
    apply(rule semialg-function-tuple-comp[of - - 2])
       apply (simp add: 1)
      apply simp
    by (simp add: multiplication-is-semialg)
  show ?thesis
    apply(rule semialg-functions-memI)
     apply (meson Qp-funs-is-cring assms(1) assms(2) cring.cring-simprules(5)
semialg-functions-memE(2))
    using 0 2 restrict-is-semialg by presburger
qed

lemma inv-in-semialg-functions:
  assumes f ∈ semialg-functions n
  assumes ⋀ x. x ∈ carrier (Q_p^n) ⟹ f x ≠ 0_{Q_p}
  shows inv_{Funn Q_p} f ∈ semialg-functions n
proof−
  have 0: inv_{Funn Q_p} f = restrict (function-tuple-comp Q_p [f] Qp-invert) (carrier
(Q_p^n))
  proof fix x
     show (inv_{Funn Q_p} f) x = restrict (function-tuple-comp Q_p [f] Qp-invert)
(carrier (Q_p^n)) x
    proof(cases x ∈ carrier (Q_p^n))
      case True
      have (function-tuple-comp Q_p [f] Qp-invert) x = Qp-invert [f x]
        unfolding function-tuple-comp-def function-tuple-eval-def
        using comp-apply  by (metis (no-types, lifting) list.simps(8) list.simps(9))
      then have (function-tuple-comp Q_p [f] Qp-invert) x = inv_{Q_p} (f x)
        unfolding Qp-invert-def
        using True assms(2) Qp.to-R-to-R1 by presburger
      then show ?thesis
        using True restrict-apply
        by (metis (mono-tags, opaque-lifting) Qp-funs-Units-memI(1)
            Qp-funs-m-inv assms(1) assms(2) semialg-functions-memE(2))
    next
      case False
      have inv_{Funn Q_p} f ∈ carrier (Fun_n Q_p)
```

```
      using assms
    by (meson Qp-funs-Units-memI(1) Qp-funs-is-monoid monoid.Units-inv-closed
semialg-functions-memE(2))
    then show ?thesis using False restrict-apply function-ring-not-car
      by auto
  qed
 qed
 have is-semialg-function n (function-tuple-comp Q_p [f] Qp-invert)
  apply(rule semialg-function-tuple-comp[of - - 1])
   apply (simp add: assms(1) is-semialg-function-tuple-def semialg-functions-memE(1))
    apply simp
    using Qp-invert-is-semialg by blast
 then show ?thesis
  using 0 restrict-in-semialg-functions
  by presburger
qed

lemma a-inv-in-semialg-functions:
 assumes f ∈ semialg-functions n
 shows ⊖_{Fun n Q_p} f ∈ semialg-functions n
proof−
 have ⊖_{Fun n Q_p} f = (Qp-const n (⊖ 1)) ⊗_{Fun n Q_p} f
 proof fix x
  show (⊖_{Fun n Q_p} f) x = (Qp-const n (⊖ 1) ⊗_{Fun n Q_p} f) x
  proof(cases x ∈ carrier (Q_p^n))
   case True
   have 0: (⊖_{Fun n Q_p} f) x = ⊖ (f x)
    using Qp-funs-a-inv semialg-functions-memE True assms(1) by blast
   have 1: (Qp-const n (⊖ 1) ⊗_{Fun n Q_p} f) x = (⊖ 1)⊗(f x)
    using Qp-funs-mult[of x n Qp-const n (⊖ 1) f] assms Qp-constE[of ⊖ 1 x
n]
      Qp-funs-mult' Qp.add.inv-closed Qp.one-closed Qp-funs-mult''' True by
presburger
   have 2: f x ∈ carrier Q_p
    using True semialg-functions-memE[of f n] assms by blast
   show ?thesis
    using True assms 0 1 2 Qp.l-minus Qp.l-one Qp.one-closed by presburger
  next
   case False
   have (⊖_{function-ring (carrier (Q_p^n)) Q_p} f) x = undefined
    using Qp-funs-a-inv'[of f n] False assms semialg-functions-memE
    by (metis (no-types, lifting) restrict-apply)
   then show ?thesis
    using False function-ring-defs(2)[of n] Qp-funs-a-inv'[of f n]
    unfolding function-mult-def restrict-def
    by presburger
  qed
 qed
 then show ?thesis
```

using *prod-in-semialg-functions*[*of Qp-const n* (⊖ **1**) *n f*] *assms*
constant-function-in-semialg-functions[*of* ⊖ **1** *n*] *Qp.add.inv-closed Qp.one-closed*
by *presburger*
qed

lemma *semialg-functions-subring*:
shows *subring* (*semialg-functions n*) (*Fun*$_n$ *Q*$_p$)
apply(*rule ring.subringI*)
using *Qp-funs-is-cring cring.axioms*(*1*) **apply** *blast*
apply (*simp add*: *semialg-functions-in-Qp-funs*)
using *Qp.one-closed constant-function-in-semialg-functions function-one-as-constant*
apply *presburger*
using *a-inv-in-semialg-functions* **apply** *blast*
using *prod-in-semialg-functions* **apply** *blast*
using *sum-in-semialg-functions* **by** *blast*

lemma *semialg-functions-subcring*:
shows *subcring* (*semialg-functions n*) (*Fun*$_n$ *Q*$_p$)
using *semialg-functions-subring cring.subcringI'*
using *Qp-funs-is-cring* **by** *blast*

definition *SA* **where**
SA n = (*Fun*$_n$ *Q*$_p$)(| *carrier* := *semialg-functions n*|)

lemma *SA-is-ring*:
shows *ring* (*SA n*)
proof−
have *ring* (*Fun*$_n$ *Q*$_p$)
by (*simp add*: *Qp-funs-is-cring cring.axioms*(*1*))
then show *?thesis*
unfolding *SA-def*
using *ring.subring-is-ring*[*of Fun*$_n$ *Q*$_p$ *semialg-functions n*] *semialg-functions-subring*[*of n*]
by *blast*
qed

lemma *SA-is-cring*:
shows *cring* (*SA n*)
using *ring.subcring-iff*[*of Fun*$_n$ *Q*$_p$ *semialg-functions n*] *semialg-functions-subcring*[*of n*]
Qp-funs-is-cring cring.axioms(*1*) *semialg-functions-in-Qp-funs*
unfolding *SA-def*
by *blast*

lemma *SA-is-monoid*:
shows *monoid* (*SA n*)
using *SA-is-ring*[*of n*] **unfolding** *ring-def*
by *linarith*

lemma *SA-is-abelian-monoid*:
shows *abelian-monoid* (*SA n*)
using *SA-is-ring*[*of n*] **unfolding** *ring-def abelian-group-def* **by** *blast*

lemma *SA-car*:
carrier (*SA n*) = *semialg-functions n*
unfolding *SA-def*
by *simp*

lemma *SA-car-in-Qp-funs-car*:
carrier (*SA n*) ⊆ *carrier* (*Fun*$_n$ Q_p)
by (*simp add*: *SA-car semialg-functions-in-Qp-funs*)

lemma *SA-car-memI*:
assumes *f* ∈ *carrier* (*Fun*$_n$ Q_p)
assumes *is-semialg-function n f*
shows *f* ∈ *carrier* (*SA n*)
using *assms semialg-functions-memI*[*of f n*] *SA-car*
by *blast*

lemma *SA-car-memE*:
assumes *f* ∈ *carrier* (*SA n*)
shows *is-semialg-function n f*
f ∈ *carrier* (*Fun*$_n$ Q_p)
f ∈ *carrier* ($Q_p{}^n$) → *carrier* Q_p
using *SA-car assms semialg-functions-memE*(*1*) **apply** *blast*
using *SA-car assms semialg-functions-memE*(*2*) **apply** *blast*
using *SA-car assms semialg-functions-memE*(*3*) **by** *blast*

lemma *SA-plus*:
($\oplus_{SA\ n}$) = ($\oplus_{Fun_n\ Q_p}$)
unfolding *SA-def*
by *simp*

lemma *SA-times*:
($\otimes_{SA\ n}$) = ($\otimes_{Fun_n\ Q_p}$)
unfolding *SA-def*
by *simp*

lemma *SA-one*:
($\mathbf{1}_{SA\ n}$) = ($\mathbf{1}_{Fun_n\ Q_p}$)
unfolding *SA-def*
by *simp*

lemma *SA-zero*:
($\mathbf{0}_{SA\ n}$) = ($\mathbf{0}_{Fun_n\ Q_p}$)
unfolding *SA-def*
by *simp*

lemma *SA-zero-is-function-ring*:
$(Fun_0\ Q_p) = SA\ 0$
proof−
have *0*: $carrier\ (Fun_0\ Q_p) = carrier\ (SA\ 0)$
proof
show $carrier\ (function\text{-}ring\ (carrier\ ({Q_p}^0))\ Q_p) \subseteq carrier\ (SA\ 0)$
proof fix f **assume** *A0*: $f \in carrier\ (function\text{-}ring\ (carrier\ ({Q_p}^0))\ Q_p)$
show $f \in carrier\ (SA\ 0)$
proof(*rule SA-car-memI*)
show $f \in carrier\ (function\text{-}ring\ (carrier\ ({Q_p}^0))\ Q_p)$
using *A0* **by** *blast*
show *is-semialg-function 0 f*
proof(*rule is-semialg-functionI*)
show $f \in carrier\ ({Q_p}^0) \to carrier\ Q_p$
using *A0 Qp.function-ring-car-memE* **by** *blast*
show $\bigwedge k\ S.\ S \in semialg\text{-}sets\ (1\ +\ k) \Longrightarrow is\text{-}semialgebraic\ (0\ +\ k)$
(*partial-pullback 0 f k S*)
proof− **fix** $k\ S$ **assume** *A*: $S \in semialg\text{-}sets\ (1{+}k)$
obtain *a* **where** *a-def*: $a = f\ []$
by *blast*
have *0*: $carrier\ ({Q_p}^0) = \{[]\}$
using *Qp-zero-carrier* **by** *blast*
have *1*: $(partial\text{-}pullback\ 0\ f\ k\ S) = \{x \in carrier\ ({Q_p}^k).\ a\#x \in S\}$
proof
show $partial\text{-}pullback\ 0\ f\ k\ S \subseteq \{x \in carrier\ ({Q_p}^k).\ a\ \#\ x \in S\}$
apply(*rule subsetI*)
unfolding *partial-pullback-def partial-image-def* **using** *a-def*
by (*metis* (*no-types*, *lifting*) *add.left-neutral drop0 evimage-eq mem-Collect-eq take-eq-Nil*)
show $\{x \in carrier\ ({Q_p}^k).\ a\ \#\ x \in S\} \subseteq partial\text{-}pullback\ 0\ f\ k\ S$
apply(*rule subsetI*)
unfolding *partial-pullback-def partial-image-def a-def*
by (*metis* (*no-types*, *lifting*) *add.left-neutral drop0 evimageI2 mem-Collect-eq take0*)
qed
have *2*: $cartesian\text{-}product\ \{[a]\}\ (partial\text{-}pullback\ 0\ f\ k\ S) = (cartesian\text{-}product\ \{[a]\}\ (carrier\ ({Q_p}^k))) \cap S$
proof
show $cartesian\text{-}product\ \{[a]\}\ (partial\text{-}pullback\ 0\ f\ k\ S) \subseteq cartesian\text{-}product\ \{[a]\}\ (carrier\ ({Q_p}^k)) \cap S$
proof(*rule subsetI*) **fix** x **assume** *A*: $x \in cartesian\text{-}product\ \{[a]\}\ (partial\text{-}pullback\ 0\ f\ k\ S)$
then obtain y **where** *y-def*: $y \in (partial\text{-}pullback\ 0\ f\ k\ S) \wedge x = a\#y$
using *cartesian-product-memE′*
by (*metis* (*no-types*, *lifting*) *Cons-eq-appendI self-append-conv2 singletonD*)
hence *20*: $x \in S$
unfolding *1* **by** *blast*
have *21*: $x = [a]@y$

using *y-def* **by** (*simp add*: *y-def*)
have $x \in$ *cartesian-product* $\{[a]\}$ (*carrier* $(Q_p{}^k)$)
unfolding *21* **apply**(*rule cartesian-product-memI'*[*of* - Q_p *1* - *k*])
using *a-def* **apply** (*metis function-ring-car-closed Qp-zero-carrier* ‹*f* ∈ *carrier* (*function-ring* (*carrier* $(Q_p{}^0)$) Q_p)› *empty-subsetI insert-subset singletonI Qp.to-R1-closed*)
apply *blast*
apply *blast*
using *y-def* **unfolding** *partial-pullback-def evimage-def*
by (*metis IntD2 add-cancel-right-left*)
thus $x \in$ *cartesian-product* $\{[a]\}$ (*carrier* $(Q_p{}^k)$) $\cap$ *S*
using *20* **by** *blast*
qed
show *cartesian-product* $\{[a]\}$ (*carrier* $(Q_p{}^k)$) $\cap$ *S* $\subseteq$ *cartesian-product* $\{[a]\}$ (*partial-pullback 0 f k S*)
proof fix *x* **assume** *A*: $x \in$ *cartesian-product* $\{[a]\}$ (*carrier* $(Q_p{}^k)$) $\cap$ *S*
then obtain *y* **where** *y-def*: $x = a\#y \wedge y \in$ *carrier* $(Q_p{}^k)$
using *cartesian-product-memE'*
by (*metis* (*no-types*, *lifting*) *Cons-eq-appendI IntD1 append-Nil singletonD*)
have *00*: $y \in$ *partial-pullback 0 f k S*
using *y-def* **unfolding** *1* **using** *A* **by** *blast*
have *01*: $x = [a]@y$
using *y-def* **by** (*simp add*: *y-def*)
have *02*: *partial-pullback 0 f k S* $\subseteq$ *carrier* $(Q_p{}^k)$
unfolding *partial-pullback-def*
by (*simp add*: *extensional-vimage-closed*)
show $x \in$ *cartesian-product* $\{[a]\}$ (*partial-pullback 0 f k S*)
unfolding *01* **apply**(*rule cartesian-product-memI'*[*of* $\{[a]\}$ Q_p *1 partial-pullback 0 f k S k* [*a*] *y*])
apply (*metis function-ring-car-closed Qp-zero-carrier* ‹*f* ∈ *carrier* (*function-ring* (*carrier* $(Q_p{}^0)$) Q_p)› *a-def empty-subsetI insert-subset singletonI Qp.to-R1-closed*)
using *02* **apply** *blast*
apply *blast*
using *00* **by** *blast*
qed
qed
have *3*:*is-semialgebraic 1* $\{[a]\}$
proof−
have $a \in$ *carrier* Q_p
using *a-def Qp.function-ring-car-memE 0* ‹*f* ∈ *carrier* (*function-ring* (*carrier* $(Q_p{}^0)$) Q_p)› **by** *blast*
hence [*a*] ∈ *carrier* $(Q_p{}^1)$
using *Qp.to-R1-closed* **by** *blast*
thus *?thesis*
using *is-algebraic-imp-is-semialg singleton-is-algebraic* **by** *blast*
qed
have *4*: *is-semialgebraic* (*1*+*k*) (*cartesian-product* $\{[a]\}$ (*carrier* $(Q_p{}^k)$))

```
                using 3 carrier-is-semialgebraic cartesian-product-is-semialgebraic
less-one by blast
          have  5: is-semialgebraic (1+k) (cartesian-product {[a]} (partial-pullback
0 f k S))
         unfolding 2 using 3 4 A intersection-is-semialg padic-fields.is-semialgebraicI
padic-fields-axioms by blast
           have 6: {[a]} ⊆ carrier (Q_p^1)
            using a-def A0 0 by (metis Qp.function-ring-car-memE empty-subsetI
insert-subset singletonI Qp.to-R1-closed)
           have  7: is-semialgebraic (k+1) (cartesian-product (partial-pullback 0 f
k S) {[a]})
            apply(rule cartesian-product-swap)
              using 6 apply blast
             apply (metis add-cancel-right-left partial-pullback-closed)
            using 5 by auto
          have 8: is-semialgebraic k (partial-pullback 0 f k S)
           apply(rule cartesian-product-singleton-factor-projection-is-semialg'[of -
- [a] 1])
                apply (metis add-cancel-right-left partial-pullback-closed)
                apply (metis A0 Qp.function-ring-car-memE Qp-zero-carrier a-def
singletonI Qp.to-R1-closed)
              using 7 by blast
          thus is-semialgebraic (0 + k) (partial-pullback 0 f k S)
             by simp
         qed
       qed
     qed
   qed
   show carrier (SA 0) ⊆ carrier (function-ring (carrier (Q_p^0)) Q_p)
     using SA-car-in-Qp-funs-car by blast
 qed
 then have 1: semialg-functions 0 = carrier (Fun_0 Q_p)
   unfolding 0 SA-def by auto
 show ?thesis unfolding SA-def 1 by auto
qed

lemma constant-fun-closed:
 assumes c ∈ carrier Q_p
 shows constant-function (carrier (Q_p^m)) c ∈ carrier (SA m)
 using constant-function-in-semialg-functions SA-car assms by blast

lemma SA-0-car-memI:
 assumes ξ ∈ carrier (Q_p^0) → carrier Q_p
 assumes ⋀x. x ∉ carrier (Q_p^0) ⟹ ξ x = undefined
 shows ξ ∈ carrier (SA 0)
proof−
 have 0: carrier (Q_p^0) = {[]}
   by (simp add: Qp-zero-carrier)
 obtain c where c-def: ξ [] = c
```

by *blast*
have *1*: ξ = *constant-function* (*carrier* ($Q_p{}^0$)) *c*
unfolding *constant-function-def restrict-def*
using *assms c-def* **unfolding** *0*
by (*metis empty-iff insert-iff*)
have *2*: *c* ∈ *carrier* Q_p
using *assms*(*1*) *c-def* **unfolding** *0* **by** *blast*
show *?thesis* **unfolding** *1*
using *2 constant-fun-closed* **by** *blast*
qed

lemma *car-SA-0-mem-imp-const*:
assumes *a* ∈ *carrier* (*SA 0*)
shows ∃ *c* ∈ *carrier* Q_p. *a* = *Qp-const 0 c*
proof−
obtain *c* **where** *c-def*: *c* = *a* []
by *blast*
have *car-zero*: *carrier* ($Q_p{}^0$) = {[]}
using *Qp-zero-carrier* **by** *blast*
have *0*: *a* = *constant-function* (*carrier* ($Q_p{}^0$)) *c*
proof fix *x*
show *a x* = *constant-function* (*carrier* ($Q_p{}^0$)) *c x*
apply(*cases x* ∈ *carrier* ($Q_p{}^0$))
using *assms SA-car-memE*[*of a 0*] *c-def*
unfolding *constant-function-def restrict-def car-zero*
apply (*metis empty-iff insert-iff*)
using *assms SA-car-memE*(*2*)[*of a 0*] *c-def*
unfolding *constant-function-def restrict-def car-zero*
by (*metis car-zero function-ring-not-car*)
qed
have *c-closed*: *c* ∈ *carrier* Q_p
using *assms SA-car-memE*(*3*)[*of a 0*] **unfolding** *c-def car-zero*
by *blast*
thus *?thesis* **using** *0* **by** *blast*
qed

lemma *SA-zeroE*:
assumes *a* ∈ *carrier* ($Q_p{}^n$)
shows 0 $_{SA\ n}$ *a* = **0**
using *function-zero-eval SA-zero assms* **by** *presburger*

lemma *SA-oneE*:
assumes *a* ∈ *carrier* ($Q_p{}^n$)
shows 1 $_{SA\ n}$ *a* = **1**
using *function-one-eval SA-one assms* **by** *presburger*
end

sublocale *padic-fields* < *UPSA?*: *UP-cring SA m UP* (*SA m*)
unfolding *UP-cring-def* **using** *SA-is-cring*[*of m*] **by** *auto*

context *padic-fields*
begin

lemma *SA-add*:
assumes $x \in$ *carrier* $(Q_p{}^n)$
shows $(f \oplus_{SA\ n} g)\ x = f\ x \oplus_{Q_p} g\ x$
using *Qp-funs-add'''* *SA-plus* *assms* **by** *presburger*

lemma *SA-add'*:
assumes $x \notin$ *carrier* $(Q_p{}^n)$
shows $(f \oplus_{SA\ n} g)\ x =$ *undefined*
proof−
have $(f \oplus_{SA\ n} g)\ x =$ *function-add* (*carrier* $(Q_p{}^n))\ Q_p\ f\ g\ x$
using *SA-plus*[*of n*] **unfolding** *function-ring-def*
by (*metis ring-record-simps*(*12*))
then show *?thesis*
unfolding *function-add-def* **using** *restrict-apply* *assms*
by (*metis* (*no-types*, *lifting*))
qed

lemma *SA-mult*:
assumes $x \in$ *carrier* $(Q_p{}^n)$
shows $(f \otimes_{SA\ n} g)\ x = f\ x \otimes g\ x$
using *Qp-funs-mult'''* *SA-times* *assms* **by** *presburger*

lemma *SA-mult'*:
assumes $x \notin$ *carrier* $(Q_p{}^n)$
shows $(f \otimes_{SA\ n} g)\ x =$ *undefined*
proof−
have $(f \otimes_{SA\ n} g)\ x =$ *function-mult* (*carrier* $(Q_p{}^n))\ Q_p\ f\ g\ x$
using *SA-times*[*of n*] **unfolding** *function-ring-def*
by (*metis ring-record-simps*(*5*))
then show *?thesis*
unfolding *function-mult-def* **using** *restrict-apply* *assms*
by (*metis* (*no-types*, *lifting*))
qed

lemma *SA-u-minus-eval*:
assumes $f \in$ *carrier* $(SA\ n)$
assumes $x \in$ *carrier* $(Q_p{}^n)$
shows $(\ominus_{SA\ n} f)\ x = \ominus\ (f\ x)$
proof−
have $f \oplus_{SA\ n} (\ominus_{SA\ n} f) = \mathbf{0}_{SA\ n}$
using *assms* *SA-is-cring* *cring.cring-simprules*(*17*) **by** *metis*
have $(f \oplus_{SA\ n} (\ominus_{SA\ n} f))\ x = \mathbf{0}_{SA\ n}\ x$
using *assms* ‹$f \oplus_{SA\ n} \ominus_{SA\ n} f = \mathbf{0}_{SA\ n}$› **by** *presburger*
then have $(f\ x) \oplus (\ominus_{SA\ n} f)\ x = \mathbf{0}$
using *assms* *function-zero-eval* *SA-add* **unfolding** *SA-zero* **by** *blast*

then show *?thesis*
using *assms SA-is-ring*
by (*meson Qp.add.inv-closed Qp.add.inv-comm Qp.function-ring-car-memE Qp.minus-unique Qp.r-neg SA-car-memE*(*2*) *ring.ring-simprules*(*3*))
qed

lemma *SA-a-inv-eval*:
assumes $f \in$ *carrier* (*SA n*)
assumes $x \in$ *carrier* ($Q_p{}^n$)
shows ($\ominus_{SA\ n}\ f$) $x = \ominus\ (f\ x)$
proof−
have $f \oplus_{SA\ n} (\ominus_{SA\ n}\ f) = \mathbf{0}_{SA\ n}$
using *assms SA-is-cring cring.cring-simprules*(*17*) **by** *metis*
have $(f \oplus_{SA\ n} (\ominus_{SA\ n}\ f))\ x = \mathbf{0}_{SA\ n}\ x$
using *assms* ‹$f \oplus_{SA\ n} \ominus_{SA\ n} f = \mathbf{0}_{SA\ n}$› **by** *presburger*
then have $(f\ x) \oplus (\ominus_{SA\ n}\ f)\ x = \mathbf{0}$
by (*metis function-zero-eval SA-add SA-zero assms*)
then show *?thesis*
by (*metis* (*no-types*, *lifting*) *PiE* Q_p*-def Qp.add.m-comm Qp.minus-equality SA-is-cring Zp-def assms*(*1*) *assms*(*2*) *cring.cring-simprules*(*3*) *padic-fields.SA-car-memE*(*3*) *padic-fields-axioms*)
qed

lemma *SA-nat-pow*:
assumes $x \in$ *carrier* ($Q_p{}^n$)
shows ($f\ [\widehat{}]_{SA\ n}$ (*k*::*nat*)) $x = (f\ x)\ [\widehat{}]_{Q_p}\ k$
apply(*induction k*)
using *assms nat-pow-def*
apply (*metis function-one-eval SA-one old.nat.simps*(*6*))
using *assms SA-mult*
by (*metis Group.nat-pow-Suc*)

lemma *SA-nat-pow′*:
assumes $x \notin$ *carrier* ($Q_p{}^n$)
shows ($f\ [\widehat{}]_{SA\ n}$ (*k*::*nat*)) $x =$ *undefined*
apply(*induction k*)
using *assms nat-pow-def*[*of SA n f*]
apply (*metis* (*no-types*, *lifting*) *Group.nat-pow-0 Qp-funs-one SA-one restrict-apply*)
by (*metis Group.nat-pow-Suc SA-mult′ assms*)

lemma *SA-add-closed-id*:
assumes *is-semialg-function n f*
assumes *is-semialg-function n g*
shows *restrict f* (*carrier* ($Q_p{}^n$)) $\oplus_{SA\ n}$ *restrict g* (*carrier* ($Q_p{}^n$)) = $f \oplus_{SA\ n} g$
proof fix *x*
show (*restrict f* (*carrier* ($Q_p{}^n$)) $\oplus_{SA\ n}$ *restrict g* (*carrier* ($Q_p{}^n$))) $x = (f \oplus_{SA\ n} g)\ x$
apply(*cases* $x \in$ *carrier* ($Q_p{}^n$))
using *assms restrict-apply*

apply (*metis SA-add*)
using *assms*
by (*metis SA-add'*)
qed

lemma *SA-mult-closed-id*:
assumes *is-semialg-function n f*
assumes *is-semialg-function n g*
shows *restrict f* (*carrier* ($Q_p{}^n$)) $\otimes_{SA\ n}$ *restrict g* (*carrier* ($Q_p{}^n$)) = *f* $\otimes_{SA\ n}$ *g*
proof fix *x*
show (*restrict f* (*carrier* ($Q_p{}^n$)) $\otimes_{SA\ n}$ *restrict g* (*carrier* ($Q_p{}^n$))) *x* = (*f* $\otimes_{SA\ n}$ *g*) *x*
apply(*cases x* ∈ *carrier* ($Q_p{}^n$))
using *assms restrict-apply*
apply (*metis SA-mult*)
using *assms*
by (*metis SA-mult'*)
qed

lemma *SA-add-closed*:
assumes *is-semialg-function n f*
assumes *is-semialg-function n g*
shows *f* $\oplus_{SA\ n}$ *g* ∈ *carrier* (*SA n*)
using *assms SA-add-closed-id*
by (*metis SA-car SA-plus restrict-in-semialg-functions sum-in-semialg-functions*)

lemma *SA-mult-closed*:
assumes *is-semialg-function n f*
assumes *is-semialg-function n g*
shows *f* $\otimes_{SA\ n}$ *g* ∈ *carrier* (*SA n*)
using *assms SA-mult-closed-id*
by (*metis SA-car SA-is-cring cring.cring-simprules*(*5*) *restrict-in-semialg-functions*)

lemma *SA-add-closed-right*:
assumes *is-semialg-function n f*
assumes *g* ∈ *carrier* (*SA n*)
shows *f* $\oplus_{SA\ n}$ *g* ∈ *carrier* (*SA n*)
using *SA-add-closed SA-car-memE*(*1*) *assms*(*1*) *assms*(*2*) **by** *blast*

lemma *SA-mult-closed-right*:
assumes *is-semialg-function n f*
assumes *g* ∈ *carrier* (*SA n*)
shows *f* $\otimes_{SA\ n}$ *g* ∈ *carrier* (*SA n*)
using *SA-car-memE*(*1*) *SA-mult-closed assms*(*1*) *assms*(*2*) **by** *blast*

lemma *SA-add-closed-left*:
assumes *f* ∈ *carrier* (*SA n*)
assumes *is-semialg-function n g*
shows *f* $\oplus_{SA\ n}$ *g* ∈ *carrier* (*SA n*)

using *SA-add-closed SA-car-memE*(*1*) *assms*(*1*) *assms*(*2*) **by** *blast*

lemma *SA-mult-closed-left*:
assumes $f \in$ *carrier* (*SA n*)
assumes *is-semialg-function n g*
shows $f \otimes_{SA\ n} g \in$ *carrier* (*SA n*)
using *SA-car-memE*(*1*) *SA-mult-closed assms*(*1*) *assms*(*2*) **by** *blast*

lemma *SA-nat-pow-closed*:
assumes *is-semialg-function n f*
shows $f\ [\widehat{}]_{SA\ n}\ (k{::}nat) \in$ *carrier* (*SA n*)
apply(*induction k*)
using *nat-pow-def*[*of SA n f*]
apply (*metis Group.nat-pow-0 monoid.one-closed SA-is-monoid*)
by (*metis Group.nat-pow-Suc SA-car assms*(*1*) *assms SA-mult-closed semialg-functions-memE*(*1*))

lemma *SA-imp-semialg*:
assumes $f \in$ *carrier* (*SA n*)
shows *is-semialg-function n f*
using *SA-car assms semialg-functions-memE*(*1*) **by** *blast*

lemma *SA-minus-closed*:
assumes $f \in$ *carrier* (*SA n*)
assumes $g \in$ *carrier* (*SA n*)
shows $(f\ \ominus_{SA\ n}\ g) \in$ *carrier* (*SA n*)
using *assms* **unfolding** *a-minus-def*
by (*meson SA-add-closed-left SA-imp-semialg SA-is-ring ring.ring-simprules*(*3*))

lemma(**in** *ring*) *add-pow-closed* :
assumes $b \in$ *carrier R*
shows $([(m{::}nat)]\cdot_R b) \in$ *carrier R*
by(*rule add.nat-pow-closed*, *rule assms*)

lemma(**in** *ring*) *add-pow-Suc*:
assumes $b \in$ *carrier R*
shows $[(Suc\ m)]\cdot b = [m]\cdot b \oplus b$
using *assms add.nat-pow-Suc* **by** *blast*

lemma(**in** *ring*) *add-pow-zero*:
assumes $b \in$ *carrier R*
shows $[(0{::}nat)]\cdot b = \mathbf{0}$
using *assms nat-mult-zero*
by *blast*

lemma *Fun-add-pow-apply*:
assumes $b \in$ *carrier* ($Fun_n\ Q_p$)
assumes $a \in$ *carrier* (${Q_p}^n$)
shows $([(m{::}nat)]\cdot_{Fun_n\ Q_p}\ b)\ a = [m]\cdot(\ b\ a)$
proof−

have *0*: *b a* ∈ *carrier* Q_p
using *Qp.function-ring-car-mem-closed assms* **by** *fastforce*
have *1*: *ring* (*Fun*$_n$ Q_p)
using *function-ring-is-ring* **by** *blast*
show *?thesis*
proof(*induction m*)
case *0*
have ([(*0*::*nat*)] ·$_{function\text{-}ring\ (carrier\ ({Q_p}^n))\ Q_p}$ *b*) = $\mathbf{0}_{Fun_n\ Q_p}$
using *1 ring.add-pow-zero*[*of Fun*$_n$ Q_p *b*] *assms* **by** *blast*
then show *?case*
using *function-zero-eval Qp.nat-mult-zero assms* **by** *presburger*
next
case (*Suc m*)
then show *?case* **using** *Suc ring.add-pow-Suc*[*of SA n b m*] *assms*
by (*metis* (*no-types*, *lifting*) *0 1 Qp.ring-axioms SA-add SA-plus ring.add-pow-Suc*)
qed
qed

lemma *SA-add-pow-apply*:
assumes *b* ∈ *carrier* (*SA n*)
assumes *a* ∈ *carrier* (${Q_p}^n$)
shows ([(*m*::*nat*)]·$_{SA\ n}$ *b*) *a* = [*m*]·(*b a*)
apply(*induction m*)
using *assms SA-is-ring*[*of n*] *Fun-add-pow-apply*
apply (*metis function-zero-eval Qp.nat-mult-zero SA-zero ring.add-pow-zero*)
using *assms SA-is-ring*[*of n*] *ring.add-pow-Suc*[*of Fun*$_n$ Q_p *b*] *ring.add-pow-Suc*[*of SA n b*] *SA-plus*[*of n*]
using *Fun-add-pow-apply*
by (*metis Qp.add.nat-pow-Suc SA-add*)

lemma *Qp-funs-Units-SA-Units*:
assumes *f* ∈ *Units* (*Fun*$_n$ Q_p)
assumes *is-semialg-function n f*
shows *f* ∈ *Units* (*SA n*)
proof−
have *0*: *f* ∈ *carrier* (*Fun*$_n$ Q_p)
by (*meson Qp-funs-is-monoid assms*(*1*) *monoid.Units-closed*)
have *1*: *inv*$_{Fun_n\ Q_p}$ *f* ∈ *semialg-functions n*
using *monoid.Units-closed*[*of Fun*$_n$ Q_p *f*]
assms inv-in-semialg-functions[*of f n*] *Qp-funs-Units-memE*(*3*)[*of f n*]
semialg-functions-memI[*of f n*] *Qp-funs-is-monoid* **by** *blast*
then have *2*: *f* ⊗$_{SA\ n}$ (*inv*$_{Fun_n\ Q_p}$ *f*) = $\mathbf{1}_{SA\ n}$
using *Qp-funs-Units-memE*(*1*)[*of f n*] *SA-one SA-times assms*(*1*)
by *presburger*
then have *3*: (*inv*$_{Fun_n\ Q_p}$ *f*) ⊗$_{SA\ n}$ *f* = $\mathbf{1}_{SA\ n}$
using *Qp-funs-Units-memE*(*2*)[*of f n*] *SA-one SA-times assms*(*1*)
by *presburger*
have *4*: *f* ∈ *carrier* (*SA n*)
using *0 SA-car assms*(*2*) *semialg-functions-memI* **by** *blast*

have *5*: $inv_{Fun_n\ Q_p}\ f \in carrier\ (SA\ n)$
using *SA-car-memI 1 SA-car* **by** *blast*
show *?thesis*
using *5 4 3 2* **unfolding** *Units-def*
by *blast*
qed

lemma *SA-Units-memE*:
assumes $f \in (Units\ (SA\ n))$
shows $f \otimes_{SA\ n} inv_{SA\ n}\ f = \mathbf{1}_{SA\ n}$
$inv_{SA\ n}\ f \otimes_{SA\ n} f = \mathbf{1}_{SA\ n}$
using *assms SA-is-monoid*[*of n*] *monoid.Units-r-inv*[*of SA n f*]
apply *blast*
using *assms SA-is-monoid*[*of n*] *monoid.Units-l-inv*[*of SA n f*]
by *blast*

lemma *SA-Units-closed*:
assumes $f \in (Units\ (SA\ n))$
shows $f \in carrier\ (SA\ n)$
using *assms* **unfolding** *Units-def* **by** *blast*

lemma *SA-Units-inv-closed*:
assumes $f \in (Units\ (SA\ n))$
shows $inv_{SA\ n}\ f \in carrier\ (SA\ n)$
using *assms SA-is-monoid*[*of n*] *monoid.Units-inv-closed*[*of SA n f*]
by *blast*

lemma *SA-Units-Qp-funs-Units*:
assumes $f \in (Units\ (SA\ n))$
shows $f \in (Units\ (Fun_n\ Q_p))$
proof−
have *0*: $f \otimes_{SA\ n} inv_{SA\ n}\ f = \mathbf{1}_{SA\ n}$
$inv_{SA\ n}\ f \otimes_{SA\ n} f = \mathbf{1}_{SA\ n}$
using *R.Units-r-inv assms* **apply** *blast*
using *R.Units-l-inv assms* **by** *blast*
have *1*: $f \in carrier\ (Fun_n\ Q_p)$
using *assms*
by (*metis SA-car SA-is-monoid monoid.Units-closed semialg-functions-memE*(*2*))
have *2*: $inv_{SA\ n}\ f \in carrier\ (Fun_n\ Q_p)$
using *SA-Units-inv-closed SA-car assms semialg-functions-memE*(*2*) **by** *blast*
then show *?thesis*
using *0 1 2 SA-one SA-times local.F.UnitsI*(*1*) **by** *auto*
qed

lemma *SA-Units-Qp-funs-inv*:
assumes $f \in (Units\ (SA\ n))$
shows $inv_{SA\ n}\ f = inv_{Fun_n\ Q_p}\ f$
using *assms SA-Units-Qp-funs-Units*
by (*metis* (*no-types*, *opaque-lifting*) *Qp-funs-is-cring Qp-funs-is-monoid SA-Units-memE*(*1*)

SA-is-monoid SA-one SA-times cring.invI(1) monoid.Units-closed monoid.Units-inv-Units)

lemma *SA-Units-memI*:
assumes $f \in (carrier\ (SA\ n))$
assumes $\bigwedge x.\ x \in carrier\ ({Q_p}^n) \Longrightarrow\ f\ x \neq \mathbf{0}_{Q_p}$
shows $f \in (Units\ (SA\ n))$
using *assms Qp-funs-Units-memI*[*of f n*] *Qp-funs-Units-SA-Units SA-car SA-imp-semialg semialg-functions-memE*(*2*) **by** *blast*

lemma *SA-Units-memE'*:
assumes $f \in (Units\ (SA\ n))$
shows $\bigwedge x.\ x \in carrier\ ({Q_p}^n) \Longrightarrow\ f\ x \neq \mathbf{0}_{Q_p}$
using *assms Qp-funs-Units-memE*[*of f n*] *SA-Units-Qp-funs-Units*
by *blast*

lemma *Qp-n-nonempty*:
shows $carrier\ ({Q_p}^n) \neq \{\}$
apply(*induction n*)
apply (*simp add*: *Qp-zero-carrier*)
using *cartesian-power-cons*[*of - Q_p - 1*] *Qp.one-closed*
by (*metis Suc-eq-plus1 all-not-in-conv cartesian-power-cons empty-iff*)

lemma *SA-one-not-zero*:
shows $\mathbf{1}_{SA\ n} \neq \mathbf{0}_{\ SA\ n}$
proof−
obtain *a* **where** *a-def*: $a \in carrier\ ({Q_p}^n)$
using *Qp-n-nonempty* **by** *blast*
have $\mathbf{1}_{SA\ n}\ a\ \neq \mathbf{0}_{\ SA\ n}\ a$
using *function-one-eval function-zero-eval SA-one SA-zero a-def local.one-neq-zero*
by *presburger*
then show *?thesis*
by *metis*
qed

lemma *SA-units-not-zero*:
assumes $f \in Units\ (SA\ n)$
shows $f \neq \mathbf{0}_{\ SA\ n}$
using *SA-one-not-zero*
by (*metis assms padic-fields.SA-is-ring padic-fields-axioms ring.ring-in-Units-imp-not-zero*)

lemma *SA-Units-nonzero*:
assumes $f \in Units\ (SA\ m)$
assumes $x \in carrier\ ({Q_p}^m)$
shows $f\ x \in nonzero\ Q_p$
unfolding *nonzero-def mem-Collect-eq*
apply(*rule conjI*)
using *assms SA-Units-closed SA-car-memE*(*3*)[*of f m*] **apply** *blast*
using *assms SA-Units-memE'* **by** *blast*

lemma *SA-car-closed*:
assumes $f \in$ *carrier* (*SA m*)
assumes $x \in$ *carrier* ($Q_p{}^m$)
shows $f\ x \in$ *carrier* Q_p
using *assms SA-car-memE*(*3*) **by** *blast*

lemma *SA-Units-closed-fun*:
assumes $f \in$ *Units* (*SA m*)
assumes $x \in$ *carrier* ($Q_p{}^m$)
shows $f\ x \in$ *carrier* Q_p
using *SA-Units-closed SA-car-closed assms* **by** *blast*

lemma *SA-inv-eval*:
assumes $f \in$ *Units* (*SA n*)
assumes $x \in$ *carrier* ($Q_p{}^n$)
shows ($inv_{SA\ n}\ f$) $x = inv\ (f\ x)$
proof−
have $f \otimes_{SA\ n} (inv_{SA\ n}\ f) = \mathbf{1}_{SA\ n}$
using *assms SA-is-cring SA-Units-memE*(*1*) **by** *blast*
hence $(f \otimes_{SA\ n} (inv_{SA\ n}\ f))\ x = \mathbf{1}_{SA\ n}\ x$
using *assms* **by** *presburger*
then have $(f\ x) \otimes (inv_{SA\ n}\ f)\ x = \mathbf{1}$
by (*metis function-one-eval SA-mult SA-one assms*)
then show *?thesis*
by (*metis* Q_p*-def Qp-funs-m-inv Zp-def assms*(*1*) *assms*(*2*) *padic-fields.SA-Units-Qp-funs-Units padic-fields.SA-Units-Qp-funs-inv padic-fields-axioms*)
qed

lemma *SA-div-eval*:
assumes $f \in$ *Units* (*SA n*)
assumes $h \in$ *carrier* (*SA n*)
assumes $x \in$ *carrier* ($Q_p{}^n$)
shows ($h \otimes_{SA\ n} (inv_{SA\ n}\ f)$) $x = h\ x \otimes inv\ (f\ x)$
using *assms SA-inv-eval SA-mult* **by** *presburger*

lemma *SA-unit-int-pow*:
assumes $f \in$ *Units* (*SA m*)
assumes $x \in$ *carrier* ($Q_p{}^m$)
shows ($f[\widehat{}]_{SA\ m}(i::int)$) $x = (f\ x)[\widehat{}]i$
proof(*induction i*)
case (*nonneg n*)
have *0*: $(f\ [\widehat{}]_{SA\ m}\ int\ n) = (f\ [\widehat{}]_{SA\ m}\ n)$
using *assms* **by** (*meson int-pow-int*)
then show *?case* **using** *SA-Units-closed*[*of f m*] *assms*
by (*metis SA-nat-pow int-pow-int*)
next
case (*neg n*)
have *0*: $(f\ [\widehat{}]_{SA\ m} - int\ (Suc\ n)) = inv_{\ SA\ m}(f\ [\widehat{}]_{SA\ m}\ (Suc\ n))$
using *assms* **by** (*metis R.int-pow-inv' int-pow-int*)

then show *?case* **unfolding** *0* **using** *assms*
by (*metis Qp.int-pow-inv' R.Units-pow-closed SA-Units-nonzero SA-inv-eval SA-nat-pow Units-eq-nonzero int-pow-int*)
qed

lemma *restrict-in-SA-car*:
assumes *is-semialg-function n f*
shows *restrict f* (*carrier* ($Q_p{}^n$)) ∈ *carrier* (*SA n*)
using *assms SA-car restrict-in-semialg-functions*
by *blast*

lemma *SA-smult*:
$(\odot_{SA\ n}) = (\odot_{Fun n\ Q_p})$
unfolding *SA-def* **by** *auto*

lemma *SA-smult-formula*:
assumes *h* ∈ *carrier* (*SA n*)
assumes *q* ∈ *carrier* Q_p
assumes *a* ∈ *carrier* ($Q_p{}^n$)
shows (*q* $\odot_{SA\ n}$ *h*) *a* = *q* ⊗(*h a*)
using *SA-smult assms function-smult-eval SA-car-memE*(*2*) **by** *presburger*

lemma *SA-smult-closed*:
assumes *h* ∈ *carrier* (*SA n*)
assumes *q* ∈ *carrier* Q_p
shows *q* $\odot_{SA\ n}$ *h* ∈ *carrier* (*SA n*)
proof−
obtain *g* **where** *g-def*: *g* = $\mathfrak{c}_n$ *q*
by *blast*
have *g-closed*: *g* ∈ *carrier* (*SA n*)
using *g-def assms constant-function-is-semialg*[*of q n*] *constant-function-closed SA-car-memI*
by *blast*
have *q* $\odot_{SA\ n}$ *h* = *g* $\otimes_{SA\ n}$ *h*
apply(*rule function-ring-car-eqI*[*of - n*])
using *function-smult-closed SA-car-memE*(*2*) *SA-smult assms* **apply** *presburger*
using *SA-car-memE*(*2*) *assms*(*1*) *assms*(*2*) *g-closed padic-fields.SA-imp-semialg padic-fields.SA-mult-closed-right padic-fields-axioms* **apply** *blast*
using *Qp-constE SA-mult SA-smult-formula assms g-def* **by** *presburger*
thus *?thesis*
using *SA-imp-semialg SA-mult-closed-right assms*(*1*) *assms*(*2*) *g-closed* **by** *presburger*
qed

lemma *p-mult-function-val*:
assumes *f* ∈ *carrier* (*SA m*)
assumes *x* ∈ *carrier* ($Q_p{}^m$)
shows *val* (($\mathfrak{p}$ $\odot_{SA\ m}$*f*) *x*) = *val* (*f x*) + *1*
proof−

```
  have 0: (𝔭⊙_{SA m}f) x = 𝔭⊗(f x)
    using Qp.int-inc-closed SA-smult-formula assms(1) assms(2) by blast
  show ?thesis  unfolding 0 using assms
     by (metis Qp.function-ring-car-memE Qp.int-inc-closed Qp.m-comm SA-car
semialg-functions-memE(2) val-mult val-p)
qed

lemma Qp-char-0'':
  assumes a ∈ carrier Q_p
  assumes a ≠ 𝟬
  assumes (k::nat) > 0
  shows [k]·a ≠ 𝟬
proof−
  have 0: [k]·𝟭 ≠𝟬
    using Qp-char-0 assms(3) by blast
  have [k]·a = [k]·𝟭 ⊗ a
    using Qp.add-pow-ldistr Qp.l-one Qp.one-closed assms(1) by presburger
  thus ?thesis using 0 assms
    using Qp.integral by blast
qed

lemma SA-char-zero:
  assumes f ∈ carrier (SA m)
  assumes f ≠ 𝟬_{SA m}
  assumes n > 0
  shows [(n::nat)]·_{SA m}f ≠ 𝟬_{SA m}
proof assume A: [n] ·_{SA m} f = 𝟬_{SA m}
  obtain x where x-def: x ∈ carrier ((Q_p)^m) ∧ f x ≠ 𝟬
    using assms
   by (metis function-ring-car-eqI R.cring-simprules(2) SA-car-memE(2) SA-zeroE)
  have 0: ([(n::nat)]·_{SA m}f) x = [n]·(f x)
    using SA-add-pow-apply assms(1) x-def by blast
  have 1: [n]·(f x) = 𝟬
    using 0 unfolding A  using SA-zeroE x-def by blast
  have 2: f x ∈ nonzero Q_p
    using x-def assms
   by (metis Qp.function-ring-car-memE SA-car not-nonzero-Qp semialg-functions-memE(2))
  then show False using x-def
    using 1 Qp.nonzero-memE(1) Qp-char-0'' assms(3) by blast
qed
```

14.4 Defining Semialgebraic Maps

We can define a semialgebraic map in essentially the same way that Denef defines semialgebraic functions. As for functions, we can define the partial pullback of a set $S \subseteq \mathbb{Q}_p^{n+l}$ by a map $g : \mathbb{Q}_p^m \to \mathbb{Q}_p^n$ to be the set

$$\{(x, y) \in \mathbb{Q}_p^m \times \mathbb{Q}_p^l \mid (f(x), y) \in S\}$$

and say that g is a semialgebraic map if for every l, and every semialgebraic $S \subseteq \mathbb{Q}_p^{n+l}$, the partial pullback of S by g is also semialgebraic. On this definition, it is immediate that the composition $f \circ g$ of a semialgebraic function $f : \mathbb{Q}_p^n \to \mathbb{Q}$ and a semialgebraic map $g : \mathbb{Q}_p^m \to \mathbb{Q}_p^n$ is semialgebraic. It is also not hard to show that a map is semialgebraic if and only if all of its coordinate functions are semialgebraic functions. This allows us to build new semialgebraic functions out of old ones via composition.

Generalizing the notion of partial image partial pullbacks from functions to maps:

definition *map-partial-image* **where**
map-partial-image m f xs = (*f* (*take m xs*))@(*drop m xs*)

definition *map-partial-pullback* **where**
map-partial-pullback m f l S = (*map-partial-image m f*) $^{-1}{}_{m+l}$ *S*

lemma *map-partial-pullback-memE*:
assumes *as* ∈ *map-partial-pullback m f l S*
shows *as* ∈ *carrier* (${Q_p}^{m+l}$) *map-partial-image m f as* ∈ *S*
using *assms* **unfolding** *map-partial-pullback-def evimage-def*
apply (*metis* (*no-types*, *opaque-lifting*) *Int-iff add.commute*)
using *assms* **unfolding** *map-partial-pullback-def*
by *blast*

lemma *map-partial-pullback-closed*:
map-partial-pullback m f l S ⊆ *carrier* (${Q_p}^{m+l}$)
using *map-partial-pullback-memE*(*1*) **by** *blast*

lemma *map-partial-pullback-memI*:
assumes *as* ∈ *carrier* (${Q_p}^{m+k}$)
assumes (*f* (*take m as*))@(*drop m as*) ∈ *S*
shows *as* ∈ *map-partial-pullback m f k S*
using *assms* **unfolding** *map-partial-pullback-def map-partial-image-def*
by *blast*

lemma *map-partial-image-eq*:
assumes *as* ∈ *carrier* (${Q_p}^{n}$)
assumes *bs* ∈ *carrier* (${Q_p}^{k}$)
assumes *x* = *as* @ *bs*
shows *map-partial-image n f x* = (*f as*)@*bs*
proof−
have *0*: (*take n x*) = *as*
by (*metis append-eq-conv-conj assms*(*1*) *assms*(*3*) *cartesian-power-car-memE*)
have *1*: *drop n x* = *bs*
by (*metis 0 append-take-drop-id assms*(*3*) *same-append-eq*)
show *?thesis* **using** *0 1* **unfolding** *map-partial-image-def*
by *blast*
qed

lemma *map-partial-pullback-memE'*:
assumes $as \in carrier\ ({Q_p}^n)$
assumes $bs \in carrier\ ({Q_p}^k)$
assumes $x = as\ @\ bs$
assumes $x \in map\text{-}partial\text{-}pullback\ n\ f\ k\ S$
shows $(f\ as)@bs \in S$
using *map-partial-pullback-memE*[*of x n f k S*] *map-partial-image-def*[*of n f x*]
by (*metis assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *map-partial-image-eq*)

Partial pullbacks have the same algebraic properties as pullbacks.

lemma *map-partial-pullback-intersect*:
$map\text{-}partial\text{-}pullback\ m\ f\ l\ (S1 \cap S2) = (map\text{-}partial\text{-}pullback\ m\ f\ l\ S1) \cap (map\text{-}partial\text{-}pullback\ m\ f\ l\ S2)$
unfolding *map-partial-pullback-def*
by *simp*

lemma *map-partial-pullback-union*:
$map\text{-}partial\text{-}pullback\ m\ f\ l\ (S1 \cup S2) = (map\text{-}partial\text{-}pullback\ m\ f\ l\ S1) \cup (map\text{-}partial\text{-}pullback\ m\ f\ l\ S2)$
unfolding *map-partial-pullback-def*
by *simp*

lemma *map-partial-pullback-complement*:
assumes $f \in carrier\ ({Q_p}^m) \to carrier\ ({Q_p}^n)$
shows $map\text{-}partial\text{-}pullback\ m\ f\ l\ (carrier\ ({Q_p}^{n+l}) - S) = carrier\ ({Q_p}^{m+l}) - (map\text{-}partial\text{-}pullback\ m\ f\ l\ S)$
apply(*rule equalityI*)
using *map-partial-pullback-def*[*of m f l* (*carrier* $({Q_p}^{n+l}) - S$)]
map-partial-pullback-def[*of m f l S*]
apply (*metis* (*no-types*, *lifting*) *DiffD2 DiffI evimage-Diff map-partial-pullback-memE*(*1*) *subsetI*)
proof fix *x* **assume** *A*: $x \in carrier\ ({Q_p}^{m+l}) - map\text{-}partial\text{-}pullback\ m\ f\ l\ S$
show $x \in map\text{-}partial\text{-}pullback\ m\ f\ l\ (carrier\ ({Q_p}^{n+l}) - S)$
apply(*rule map-partial-pullback-memI*)
using *A*
apply *blast*
proof
have *0*: $drop\ m\ x \in carrier\ ({Q_p}^l)$
by (*meson A DiffD1 cartesian-power-drop*)
have *1*: $take\ m\ x \in carrier\ ({Q_p}^m)$
using *A*
by (*meson DiffD1 take-closed le-add1*)
show $f\ (take\ m\ x)\ @\ drop\ m\ x \in carrier\ ({Q_p}^{n+l})$
using *0 1 assms*
by (*meson Pi-iff cartesian-power-concat*(*1*))
show $f\ (take\ m\ x)\ @\ drop\ m\ x \notin S$
using *A* **unfolding** *map-partial-pullback-def map-partial-image-def*

by *blast*
qed
qed

lemma *map-partial-pullback-carrier*:
assumes $f \in$ *carrier* $({Q_p}^m) \to$ *carrier* $({Q_p}^n)$
shows *map-partial-pullback* $m\ f\ l$ (*carrier* $({Q_p}^{n+l})) =$ *carrier* $({Q_p}^{m+l})$
apply(*rule equalityI*)
using *map-partial-pullback-memE*(*1*) **apply** *blast*
proof fix x **assume** A: $x \in$ *carrier* $({Q_p}^{m+l})$
show $x \in$ *map-partial-pullback* $m\ f\ l$ (*carrier* $({Q_p}^{n+l})$)
apply(*rule map-partial-pullback-memI*)
using A *cartesian-power-drop*[*of x m l*] *take-closed assms*
apply *blast*
proof−
have f (*take* $m\ x$) $\in$ *carrier* $({Q_p}^n)$
using A *take-closed assms*
by (*meson Pi-mem le-add1*)
then show f (*take* $m\ x$) @ *drop* $m\ x \in$ *carrier* $({Q_p}^{n+l})$
using *cartesian-power-drop*[*of x m l*] A *cartesian-power-concat*(*1*)[*of - Q_p n - l*]
by *blast*
qed
qed

definition *is-semialg-map* **where**
is-semialg-map $m\ n\ f = (f \in$ *carrier* $({Q_p}^m) \to$ *carrier* $({Q_p}^n)\ \wedge$
$(\forall\, l \geq 0.\ \forall\, S \in$ *semialg-sets* $(n + l).$ *is-semialgebraic* $(m + l)$
(*map-partial-pullback* $m\ f\ l\ S$)))

lemma *is-semialg-map-closed*:
assumes *is-semialg-map* $m\ n\ f$
shows $f \in$ *carrier* $({Q_p}^m) \to$ *carrier* $({Q_p}^n)$
using *is-semialg-map-def assms* **by** *blast*

lemma *is-semialg-map-closed′*:
assumes *is-semialg-map* $m\ n\ f\ x \in$ *carrier* $({Q_p}^m)$
shows $f\ x \in$ *carrier* $({Q_p}^n)$
using *is-semialg-map-def assms* **by** *blast*

lemma *is-semialg-mapE*:
assumes *is-semialg-map* $m\ n\ f$
assumes *is-semialgebraic* $(n + k)\ S$
shows *is-semialgebraic* $(m + k)$ (*map-partial-pullback* $m\ f\ k\ S$)
using *is-semialg-map-def assms*
by (*meson is-semialgebraicE le0*)

lemma *is-semialg-mapE′*:
assumes *is-semialg-map* $m\ n\ f$
assumes *is-semialgebraic* $(n + k)\ S$

shows *is-semialgebraic* $(m + k)$ (*map-partial-image* $m\ f\ {}^{-1}{}_{m+k}\ S$)
using *assms is-semialg-mapE* **unfolding** *map-partial-pullback-def*
by *blast*

lemma *is-semialg-mapI*:
assumes $f \in$ *carrier* $({Q_p}^m) \to$ *carrier* $({Q_p}^n)$
assumes $\bigwedge k\ S.\ S \in$ *semialg-sets* $(n + k) \Longrightarrow$ *is-semialgebraic* $(m + k)$ (*map-partial-pullback* $m\ f\ k\ S$)
shows *is-semialg-map* $m\ n\ f$
using *assms* **unfolding** *is-semialg-map-def*
by *blast*

lemma *is-semialg-mapI'*:
assumes $f \in$ *carrier* $({Q_p}^m) \to$ *carrier* $({Q_p}^n)$
assumes $\bigwedge k\ S.\ S \in$ *semialg-sets* $(n + k) \Longrightarrow$ *is-semialgebraic* $(m + k)$ (*map-partial-image* $m\ f\ {}^{-1}{}_{m+k}\ S$)
shows *is-semialg-map* $m\ n\ f$
using *assms is-semialg-mapI* **unfolding** *map-partial-pullback-def*
by *blast*

Semialgebraicity for functions can be verified on basic semialgebraic sets.

lemma *is-semialg-mapI''*:
assumes $f \in$ *carrier* $({Q_p}^m) \to$ *carrier* $({Q_p}^n)$
assumes $\bigwedge k\ \ S.\ S \in$ *basic-semialgs* $(n + k) \Longrightarrow$ *is-semialgebraic* $(m + k)$ (*map-partial-pullback* $m\ f\ k\ S$)
shows *is-semialg-map* $m\ n\ f$
apply(*rule is-semialg-mapI*)
using *assms*(*1*) **apply** *blast*
proof−
show $\bigwedge k\ S.\ S \in$ *semialg-sets* $(n + k) \Longrightarrow$ *is-semialgebraic* $(m + k)$ (*map-partial-pullback* $m\ f\ k\ S$)
proof− **fix** $k\ S$ **assume** *A*: $S \in$ *semialg-sets* $(n + k)$
show *is-semialgebraic* $(m + k)$ (*map-partial-pullback* $m\ f\ k\ S$)
apply(*rule gen-boolean-algebra.induct*[*of* S *carrier* $({Q_p}^{n+k})$ *basic-semialgs* $(n + k)$])
using *A* **unfolding** *semialg-sets-def*
apply *blast*
using *map-partial-pullback-carrier assms carrier-is-semialgebraic plus-1-eq-Suc* **apply** *presburger*
apply (*simp add*: *assms*(*1*) *assms*(*2*) *carrier-is-semialgebraic intersection-is-semialg map-partial-pullback-carrier map-partial-pullback-intersect*)
using *map-partial-pullback-union union-is-semialgebraic* **apply** *presburger*
using *assms*(*1*) *complement-is-semialg map-partial-pullback-complement plus-1-eq-Suc* **by** *presburger*
qed
qed

lemma *is-semialg-mapI'''*:
assumes $f \in$ *carrier* $({Q_p}^m) \to$ *carrier* $({Q_p}^n)$

assumes $\bigwedge k\ \ S.\ S \in \textit{basic-semialgs}\ (n + k) \Longrightarrow \textit{is-semialgebraic}\ (m + k)$
$(\textit{map-partial-image}\ m\ f\ {}^{-1}{}_{m+k}\ S)$
shows *is-semialg-map m n f*
using *is-semialg-mapI'' assms* **unfolding** *map-partial-pullback-def*
by *blast*

lemma *id-is-semialg-map*:
is-semialg-map n n (λ *x. x*)
proof−
have *0*: $\bigwedge k\ S.\ S \in \textit{semialg-sets}\ (n + k) \Longrightarrow (\lambda xs.\ \textit{take}\ n\ xs\ @\ \textit{drop}\ n\ xs) -` S \cap \textit{carrier}\ ({Q_p}^{n+k}) = S$
apply(*rule equalityI'*)
apply (*metis* (*no-types*, *lifting*) *Int-iff append-take-drop-id vimage-eq*)
by (*metis* (*no-types*, *lifting*) *IntI append-take-drop-id in-mono is-semialgebraicI is-semialgebraic-closed vimageI*)
show *?thesis*
by(*intro is-semialg-mapI*,
unfold map-partial-pullback-def map-partial-image-def evimage-def is-semialgebraic-def 0,
auto)
qed

lemma *map-partial-pullback-comp*:
assumes *is-semialg-map m n f*
assumes *is-semialg-map k m g*
shows (*map-partial-pullback k* ($f \circ g$) *l S*) = (*map-partial-pullback k g l* (*map-partial-pullback m f l S*))
proof
show *map-partial-pullback k* ($f \circ g$) *l S* $\subseteq$ *map-partial-pullback k g l* (*map-partial-pullback m f l S*)
proof fix *x* **assume** *A*: *x* $\in$ *map-partial-pullback k* ($f \circ g$) *l S*
show *x* $\in$ *map-partial-pullback k g l* (*map-partial-pullback m f l S*)
proof(*rule map-partial-pullback-memI*)
show *0*: *x* $\in$ *carrier* (${Q_p}^{k+l}$)
using *A map-partial-pullback-memE*(*1*) **by** *blast*
show *g* (*take k x*) @ *drop k x* $\in$ *map-partial-pullback m f l S*
proof(*rule map-partial-pullback-memI*)
show *g* (*take k x*) @ *drop k x* $\in$ *carrier* (${Q_p}^{m+l}$)
proof−
have *1*: *g* (*take k x*) $\in$ *carrier* (${Q_p}^{m}$)
using *0 assms*(*2*) *is-semialg-map-closed*[*of k m g*]
by (*meson Pi-iff le-add1 take-closed*)
then show *?thesis*
by (*metis 0 add.commute cartesian-power-concat*(*2*) *cartesian-power-drop*)
qed
show *f* (*take m* (*g* (*take k x*) @ *drop k x*)) @ *drop m* (*g* (*take k x*) @ *drop k x*) $\in$ *S*
using *map-partial-pullback-memE*[*of x k f* $\circ$ *g l S*]

```
        comp-apply[of f g] map-partial-image-eq[of take k x k drop k x l x f ∘ g]
      by (metis (no-types, lifting) A ‹g (take k x) @ drop k x ∈ carrier (Q_p^(m + l))›
          append-eq-append-conv append-take-drop-id cartesian-power-car-memE
          cartesian-power-drop map-partial-image-def)
    qed
  qed
 qed
 show map-partial-pullback k g l (map-partial-pullback m f l S) ⊆ map-partial-pullback
k (f ∘ g) l S
  proof fix x assume A: x ∈ map-partial-pullback k g l (map-partial-pullback m f
l S)
   have 0: (take m (map-partial-image k g x)) = g (take k x)
   proof−
    have take k x ∈ carrier (Q_p^k)
     using map-partial-pullback-memE[of x k g l] A le-add1 take-closed
     by blast
    then have length (g (take k x)) = m
     using assms is-semialg-map-closed[of k m g] cartesian-power-car-memE
     by blast
    then show ?thesis
     using assms unfolding map-partial-image-def
     by (metis append-eq-conv-conj)
   qed
   show  x ∈ map-partial-pullback k (f ∘ g) l S
    apply(rule map-partial-pullback-memI)
    using A map-partial-pullback-memE
     apply blast
   using 0 assms A comp-apply map-partial-pullback-memE[of x k g l map-partial-pullback
m f l S]
         map-partial-pullback-memE[of map-partial-image k g x m f l S]
  map-partial-image-eq[of take k x k drop k x l x g]
  map-partial-image-eq[of take m (map-partial-image k g x) m drop m (map-partial-image
k g x) l (map-partial-image k g x) f ]
  by (metis (no-types, lifting) cartesian-power-drop le-add1 map-partial-image-def
map-partial-pullback-memE' take-closed)
qed
qed

lemma semialg-map-comp-closed:
  assumes is-semialg-map m n f
  assumes is-semialg-map k m g
  shows is-semialg-map k n (f ∘ g)
  apply(intro is-semialg-mapI , unfold Pi-iff comp-def, intro ballI,
      intro is-semialg-map-closed'[of m n f] is-semialg-map-closed'[of k m g]  assms,
blast)
proof− fix l S assume A: S ∈ semialg-sets (n + l)
  have  is-semialgebraic (k + l) (map-partial-pullback k (f ∘ g) l S)
   using map-partial-pullback-comp is-semialg-mapE A assms(1) assms(2) is-semialgebraicI
    by presburger
```

```
  thus is-semialgebraic (k + l) (map-partial-pullback k (λx. f (g x)) l S)
    unfolding comp-def by auto
qed

lemma partial-pullback-comp:
  assumes is-semialg-function m f
  assumes is-semialg-map k m g
  shows (partial-pullback k (f ∘ g) l S) = (map-partial-pullback k g l (partial-pullback
m f l S))
proof
  show partial-pullback k (f ∘ g) l S ⊆ map-partial-pullback k g l (partial-pullback
m f l S)
  proof fix x assume A: x ∈ partial-pullback k (f ∘ g) l S
    show x ∈ map-partial-pullback k g l (partial-pullback m f l S)
    proof(rule map-partial-pullback-memI)
      show 0: x ∈ carrier (Q_p^{k+l})
        using A partial-pullback-memE(1) by blast
      show g (take k x) @ drop k x ∈ partial-pullback m f l S
      proof(rule partial-pullback-memI)
        show g (take k x) @ drop k x ∈ carrier (Q_p^{m+l})
        proof−
          have 1: g (take k x) ∈ carrier (Q_p^m)
            using 0 assms(2) is-semialg-map-closed[of k m g]
            by (meson Pi-iff le-add1 take-closed)
          then show ?thesis
          by (metis 0 add.commute cartesian-power-concat(2) cartesian-power-drop)
        qed
        show f (take m (g (take k x) @ drop k x)) # drop m (g (take k x) @ drop k
x) ∈ S
          using partial-pullback-memE[of x k f ∘ g l S]
            comp-apply[of f g] partial-image-eq[of take k x k drop k x l x f ∘ g]
          by (metis (no-types, lifting) A ‹g (take k x) @ drop k x ∈ carrier (Q_p^{m+l})›
              add-diff-cancel-left′ append-eq-append-conv append-take-drop-id
              cartesian-power-car-memE length-drop local.partial-image-def)
      qed
    qed
  qed
  show map-partial-pullback k g l (partial-pullback m f l S) ⊆ partial-pullback k (f
∘ g) l S
  proof fix x assume A: x ∈ map-partial-pullback k g l (partial-pullback m f l S)
    have 0: (take m (map-partial-image k g x)) = g (take k x)
    proof−
      have take k x ∈ carrier (Q_p^k)
        using map-partial-pullback-memE[of x k g l] A le-add1 take-closed
        by blast
      then have length (g (take k x)) = m
        using assms is-semialg-map-closed[of k m g] cartesian-power-car-memE
        by blast
      then show ?thesis
```

using *assms* **unfolding** *map-partial-image-def*
by (*metis append-eq-conv-conj*)
qed
show $x \in$ *partial-pullback k* $(f \circ g)$ *l S*
apply(*rule partial-pullback-memI*)
using *A map-partial-pullback-memE*
apply *blast*
using *0 assms A comp-apply map-partial-pullback-memE*[*of x k g l partial-pullback m f l S*]
partial-pullback-memE[*of map-partial-image k g x m f l S*]
map-partial-image-eq[*of take k x k drop k x l x g*]
partial-image-eq[*of take m* (*map-partial-image k g x*) *m drop m* (*map-partial-image k g x*) *l* (*map-partial-image k g x*) *f*]
by (*metis* (*no-types*, *lifting*) *cartesian-power-drop le-add1 map-partial-image-def partial-pullback-memE′ take-closed*)

qed
qed

lemma *semialg-function-comp-closed*:
assumes *is-semialg-function m f*
assumes *is-semialg-map k m g*
shows *is-semialg-function k* $(f \circ g)$
proof(*rule is-semialg-functionI*)
show $f \circ g \in$ *carrier* $({Q_p}^k) \to$ *carrier* Q_p
proof fix *x* **assume** *A*: $x \in$ *carrier* $({Q_p}^k)$
show $(f \circ g)\ x \in$ *carrier* Q_p
using *A assms is-semialg-map-closed*[*of k m g*] *is-semialg-function-closed*[*of m f*] *comp-apply*[*of f g x*]
by (*metis* (*no-types*, *lifting*) *Pi-mem*)
qed
show $\bigwedge ka\ S.\ S \in$ *semialg-sets* $(1 + ka) \Longrightarrow$ *is-semialgebraic* $(k + ka)$ (*partial-pullback k* $(f \circ g)$ *ka S*)
proof− **fix** *n S* **assume** *A*: $S \in$ *semialg-sets* $(1 + n)$
show *is-semialgebraic* $(k + n)$ (*partial-pullback k* $(f \circ g)$ *n S*)
using *A assms partial-pullback-comp is-semialg-functionE is-semialg-mapE is-semialgebraicI* **by** *presburger*
qed
qed

lemma *semialg-map-evimage-is-semialg*:
assumes *is-semialg-map k m g*
assumes *is-semialgebraic m S*
shows *is-semialgebraic k* $(g\ {}^{-1}{}_k\ S)$
proof−
have $g\ {}^{-1}{}_k\ S =$ *map-partial-pullback k g 0 S*
proof
show $g\ {}^{-1}{}_k\ S \subseteq$ *map-partial-pullback k g 0 S*
proof fix *x* **assume** *A*: $x \in g\ {}^{-1}{}_k\ S$

then have *0*: $x \in carrier\ ({Q_p}^k) \wedge g\ x \in S$
by (*meson evimage-eq*)
have $x = take\ k\ x\ @\ drop\ k\ x$
using *0* **by** (*simp add*: *0*)
then show $x \in map\text{-}partial\text{-}pullback\ k\ g\ 0\ S$
unfolding *map-partial-pullback-def map-partial-image-def*
by (*metis* (*no-types*, *lifting*) *0 Nat.add-0-right add.commute append-Nil2 append-same-eq append-take-drop-id evimageI2 map-partial-image-def map-partial-image-eq take0 take-drop*)
qed
show $map\text{-}partial\text{-}pullback\ k\ g\ 0\ S \subseteq {g\ ^{-1}}_k\ S$
proof fix *x* **assume** *A*: $x \in map\text{-}partial\text{-}pullback\ k\ g\ 0\ S$
then have *0*: $g\ (take\ k\ x)\ @\ (drop\ k\ x) \in S$
unfolding *map-partial-pullback-def map-partial-image-def*
by *blast*
have *1*: $x \in carrier\ ({Q_p}^k)$
using *A* **unfolding** *map-partial-pullback-def map-partial-image-def*
by (*metis* (*no-types*, *lifting*) *Nat.add-0-right evimage-eq*)
hence $take\ k\ x = x$
by (*metis cartesian-power-car-memE le-eq-less-or-eq take-all*)
then show $x \in {g\ ^{-1}}_k\ S$
using *0 1* **unfolding** *evimage-def*
by (*metis* (*no-types*, *lifting*) *IntI append.assoc append-same-eq append-take-drop-id same-append-eq vimageI*)
qed
qed
then show *?thesis* **using** *assms*
by (*metis add.right-neutral is-semialg-mapE' map-partial-pullback-def*)
qed

14.5 Examples of Semialgebraic Maps

lemma *semialg-map-on-carrier*:
assumes *is-semialg-map n m f*
assumes $restrict\ f\ (carrier\ ({Q_p}^n)) = restrict\ g\ (carrier\ ({Q_p}^n))$
shows *is-semialg-map n m g*
proof(*rule is-semialg-mapI*)
have *0*: $f \in carrier\ ({Q_p}^n) \to carrier\ ({Q_p}^m)$
using *assms*(*1*) *is-semialg-map-closed*
by *blast*
show $g \in carrier\ ({Q_p}^n) \to carrier\ ({Q_p}^m)$
proof fix *x* **assume** *A*: $x \in carrier\ ({Q_p}^n)$ **then show** $g\ x \in carrier\ ({Q_p}^m)$
using *assms*(*2*) *0*
by (*metis* (*no-types*, *lifting*) *PiE restrict-Pi-cancel*)
qed
show $\bigwedge k\ S.\ S \in semialg\text{-}sets\ (m + k) \Longrightarrow is\text{-}semialgebraic\ (n + k)\ (map\text{-}partial\text{-}pullback\ n\ g\ k\ S)$
proof− **fix** *k S*

assume *A*: $S \in$ *semialg-sets* $(m + k)$
have *1*: *is-semialgebraic* $(n + k)$ (*map-partial-pullback n f k S*)
using *A assms(1) is-semialg-mapE is-semialgebraicI*
by *blast*
have *2*: (*map-partial-pullback n f k S*) = (*map-partial-pullback n g k S*)
unfolding *map-partial-pullback-def map-partial-image-def evimage-def*
proof
show $(\lambda xs.\ f\ (take\ n\ xs)\ @\ drop\ n\ xs) -` S \cap carrier\ ({Q_p}^{n+k}) \subseteq (\lambda xs.\ g\ (take\ n\ xs)\ @\ drop\ n\ xs) -` S \cap carrier\ ({Q_p}^{n+k})$
proof fix *x* **assume** *A*: $x \in (\lambda xs.\ f\ (take\ n\ xs)\ @\ drop\ n\ xs) -` S \cap carrier\ ({Q_p}^{n+k})$
have $(take\ n\ x) \in carrier\ ({Q_p}^{n})$
using *assms A*
by (*meson Int-iff le-add1 take-closed*)
then have $f\ (take\ n\ x) = g\ (take\ n\ x)$
using *assms* **unfolding** *restrict-def*
by *meson*
then show $x \in (\lambda xs.\ g\ (take\ n\ xs)\ @\ drop\ n\ xs) -` S \cap carrier\ ({Q_p}^{n+k})$
using *assms*
by (*metis* (*no-types*, *lifting*) *A Int-iff vimageE vimageI*)
qed
show $(\lambda xs.\ g\ (take\ n\ xs)\ @\ drop\ n\ xs) -` S \cap carrier\ ({Q_p}^{n+k}) \subseteq (\lambda xs.\ f\ (take\ n\ xs)\ @\ drop\ n\ xs) -` S \cap carrier\ ({Q_p}^{n+k})$
proof fix *x* **assume** *A*: $x \in (\lambda xs.\ g\ (take\ n\ xs)\ @\ drop\ n\ xs) -` S \cap carrier\ ({Q_p}^{n+k})$
have $(take\ n\ x) \in carrier\ ({Q_p}^{n})$
using *assms*
by (*meson A inf-le2 le-add1 subset-iff take-closed*)
then have $f\ (take\ n\ x) = g\ (take\ n\ x)$
using *assms* **unfolding** *restrict-def*
by *meson*
then show $x \in (\lambda xs.\ f\ (take\ n\ xs)\ @\ drop\ n\ xs) -` S \cap carrier\ ({Q_p}^{n+k})$
using *A* **by** *blast*
qed
qed
then show *is-semialgebraic* $(n + k)$ (*map-partial-pullback n g k S*)
using *1* **by** *auto*
qed
qed

lemma *semialg-function-tuple-is-semialg-map*:
assumes *is-semialg-function-tuple m fs*
assumes *length fs* = *n*
shows *is-semialg-map m n* (*function-tuple-eval* Q_p *m fs*)
apply(*rule is-semialg-mapI*)
using *function-tuple-eval-closed*[*of m fs*]
apply (*metis Pi-I assms(1) assms(2) semialg-function-tuple-is-function-tuple*)
proof− **fix** *k S* **assume** *A*: $S \in$ *semialg-sets* $(n + k)$
show *is-semialgebraic* $(m + k)$ (*map-partial-pullback m* (*function-tuple-eval* Q_p

m fs) *k S*)
using *is-semialg-map-tupleE*[*of m fs k S*] *assms A tuple-partial-pullback-is-semialg-map-tuple*[*of m fs*]
unfolding *tuple-partial-pullback-def map-partial-pullback-def*
map-partial-image-def tuple-partial-image-def is-semialgebraic-def
by (*metis evimage-def*)
qed

lemma *index-is-semialg-function*:
assumes $n > k$
shows *is-semialg-function n* ($\lambda as.\ as!k$)
proof−
have *0*: *restrict* ($\lambda as.\ as!k$) (*carrier* ($Q_p{}^n$)) = *restrict* (*Qp-ev* (*pvar* Q_p *k*)) (*carrier* ($Q_p{}^n$))
using *assms* **by** (*metis* (*no-types*, *lifting*) *eval-pvar restrict-ext*)
have *1*: *is-semialg-function n* (*Qp-ev* (*pvar* Q_p *k*))
using *pvar-closed assms poly-is-semialg*[*of pvar* Q_p *k*] **by** *blast*
show *?thesis*
using *0 1 semialg-function-on-carrier*[*of n Qp-ev* (*pvar* Q_p *k*) ($\lambda as.\ as!k$)]
by *presburger*
qed

definition *Qp-ith* **where**
Qp-ith m i = ($\lambda\ x \in$ *carrier* ($Q_p{}^m$). *x*!*i*)

lemma *Qp-ith-closed*:
assumes $i < m$
shows *Qp-ith m i* ∈ *carrier* (*SA m*)
proof(*rule SA-car-memI*)
show *Qp-ith m i* ∈ *carrier* (*function-ring* (*carrier* ($Q_p{}^m$)) Q_p)
apply(*rule Qp.function-ring-car-memI*[*of carrier*($Q_p{}^m$)])
using *assms cartesian-power-car-memE'*[*of - Q*$_p$ *m i*] **unfolding** *Qp-ith-def*
apply (*metis restrict-apply*)
unfolding *restrict-def* **by** *meson*
have *0*: $\bigwedge x.\ x \in$ *carrier* ($Q_p{}^m$) $\Longrightarrow$ *Qp-ith m i x* = *eval-at-point* Q_p *x* (*pvar* Q_p *i*)
using *assms eval-pvar*[*of i m*] **unfolding** *Qp-ith-def restrict-def* **by** *presburger*
have *1*: *is-semialg-function m* (*eval-at-poly* Q_p (*pvar* Q_p *i*))
using *assms pvar-closed*[*of i m*] *poly-is-semialg* **by** *blast*
show *is-semialg-function m* (*local.Qp-ith m i*)
apply(*rule semialg-function-on-carrier'*[*of m eval-at-poly* Q_p (*pvar* Q_p *i*)])
using *1* **apply** *blast*
using *0* **by** *blast*
qed

lemma *take-is-semialg-map*:
assumes $n \geq k$
shows *is-semialg-map n k* (*take k*)
proof−

obtain *fs* **where** *fs-def*: $fs = map\ (\lambda i{::}nat.\ (\lambda as \in carrier\ ({Q_p}^n).\ as!i))\ [0{::}nat..<k]$
by *blast*
have *0*: *is-semialg-function-tuple n fs*
proof(*rule is-semialg-function-tupleI*)
fix *f* **assume** *A*: $f \in set\ fs$
then obtain *i* **where** *i-def*: $i \in set\ [0{::}nat..<k] \wedge f = (\lambda as \in carrier\ ({Q_p}^n).\ as!i)$
using *A fs-def*
by (*metis* (*no-types*, *lifting*) *in-set-conv-nth length-map nth-map*)
have *i-less*: $i < k$
proof−
have *set* $[0{::}nat..<k] = \{0..<k\}$
using *atLeastLessThan-upt* **by** *blast*
then show *?thesis* **using** *i-def*
using *atLeastLessThan-iff* **by** *blast*
qed
show *is-semialg-function n f*
apply(*rule semialg-function-on-carrier*[*of n* ($\lambda as.\ as\ !\ i$)],
rule index-is-semialg-function[*of i n*])
using *A i-def assms* **by** *auto*
qed
have *1*: *restrict* (*function-tuple-eval* Q_p *n fs*) (*carrier* $({Q_p}^n)$) = *restrict* (*take k*) (*carrier* $({Q_p}^n)$)
unfolding *function-tuple-eval-def*
proof fix *x*
show $(\lambda x{\in}carrier\ ({Q_p}^n).\ map\ (\lambda f.\ f\ x)\ fs)\ x = restrict\ (take\ k)\ (carrier\ ({Q_p}^n))\ x$
proof(*cases* $x \in carrier\ ({Q_p}^n)$)
case *True*
have $(map\ (\lambda f.\ f\ x)\ fs) = take\ k\ x$
proof−
have $\bigwedge i.\ i < k \Longrightarrow (map\ (\lambda f.\ f\ x)\ fs)\ !\ i = x\ !\ i$
proof− **fix** *i* **assume** *A*: $i < k$
have *length* $[0{::}nat..<k] = k$
using *assms* **by** *simp*
then have *length fs* = *k*
using *fs-def*
by (*metis length-map*)
then have *0*: $(map\ (\lambda f.\ f\ x)\ fs)\ !\ i = (fs!i)\ x$
using *A* **by** (*meson nth-map*)
have *1*: $(fs!i)\ x = x!i$
using *A nth-map*[*of i* $[0..<k]$ ($\lambda i.\ \lambda as{\in}carrier\ ({Q_p}^n).\ as\ !\ i$)] *True*
unfolding *fs-def restrict-def* **by** *auto*
then show $map\ (\lambda f.\ f\ x)\ fs\ !\ i = x\ !\ i$
using *0 assms A True fs-def nth-map*[*of i fs*] *cartesian-power-car-memE*[*of x* Q_p *n*]
by *blast*
qed

```
    then have 0: ⋀i. i < k ⟹ (map (λf. f x) fs) ! i = (take k x) ! i
      using assms True  nth-take by blast
    have 1: length (map (λf. f x) fs) = length (take k x)
      using fs-def True assms
      by (metis cartesian-power-car-memE length-map map-nth take-closed)
    have 2: length (take k x) = k
      using assms True cartesian-power-car-memE take-closed
      by blast
    show ?thesis using 0 1 2
      by (metis nth-equalityI)
   qed
   then show ?thesis using True unfolding restrict-def
    by presburger
  next
   case False
   then show ?thesis unfolding restrict-def
    by (simp add: False)
  qed
  qed
  have 2:  is-semialg-map n k (function-tuple-eval Q_p n fs)
   using 0 semialg-function-tuple-is-semialg-map[of n fs k] assms fs-def  length-map
   by (metis (no-types, lifting) diff-zero length-upt)
  show ?thesis using 1 2
   using semialg-map-on-carrier by blast
qed

lemma drop-is-semialg-map:
  shows is-semialg-map (k + n) n (drop k)
proof−
  obtain fs where fs-def: fs = map (λi::nat. (λas ∈ carrier (Q_p^{n+k}). as!i))
[k..<n+k]
   by blast
  have 0: is-semialg-function-tuple (k+n) fs
  proof(rule is-semialg-function-tupleI)
   fix f assume A: f ∈ set fs
   then obtain i where i-def: i ∈ set [k..<n+k] ∧ f = (λas ∈ carrier (Q_p^{n+k}).
as!i)
    using A fs-def
    by (metis (no-types, lifting) in-set-conv-nth length-map nth-map)
   have i-less: i ≥ k ∧ i < n + k
   proof−
    have set [k..<n+k] = {k..<n+k}
     using atLeastLessThan-upt by blast
    then show ?thesis using i-def
     using atLeastLessThan-iff by blast
   qed
   have is-semialg-function (n + k) f
    using A index-is-semialg-function[of i n+k ]
        i-less semialg-function-on-carrier[of n+k (λas. as ! i) f] i-def
```

restrict-is-semialg
by *blast*
then show *is-semialg-function* $(k + n)$ f
by (*simp add*: *add.commute*)
qed
have *1*: *restrict* (*function-tuple-eval* Q_p $(n+k)$ *fs*) (*carrier* $(Q_p{}^{n+k})$) = *restrict* (*drop k*) (*carrier* $(Q_p{}^{n+k})$)
unfolding *function-tuple-eval-def*
proof fix x
show $(\lambda x{\in}carrier\ (Q_p{}^{n+k}).\ map\ (\lambda f.\ f\ x)\ fs)\ x$ = *restrict* (*drop k*) (*carrier* $(Q_p{}^{n+k})$) x
proof(*cases* $x{\in}carrier\ (Q_p{}^{n+k})$)
case *True*
have (*map* $(\lambda f.\ f\ x)$ *fs*) = *drop k x*
proof−
have $\bigwedge i.\ i < n \Longrightarrow (map\ (\lambda f.\ f\ x)\ fs)\ !\ i = x\ !\ (i{+}k)$
proof− **fix** i **assume** *A*: $i < n$
have *00*: *length* $[k..{<}n{+}k] = n$
by *simp*
then have *length fs* = n
using *fs-def*
by (*metis length-map*)
then have *0*:$(map\ (\lambda f.\ f\ x)\ fs)\ !\ i = (fs!i)\ x$
using *A* **by** (*meson nth-map*)
have $[k..{<}n{+}k]!i = i + k$
by (*simp add*: *A*)
have (*map* $(\lambda i.\ \lambda as{\in}carrier\ (Q_p{}^{n+k}).\ as\ !\ i)\ [k..{<}n\ +\ k])\ !\ i = ((\lambda i.\ \lambda as{\in}carrier\ (Q_p{}^{n+k}).\ as\ !\ i)\ ([k..{<}n + k]\ !\ i))$
using *A 00 nth-map*[*of i* $[k..{<}\ n\ +\ k]$ $(\lambda i.\ \lambda as{\in}carrier\ (Q_p{}^{n+k}).\ as\ !\ i)$]
by *linarith*
then have *1*: *fs*!*i* = $(\lambda as \in carrier\ (Q_p{}^{n+k}).\ as!(i{+}k))$
using *fs-def A* ‹$[k..{<}n + k]\ !\ i = i + k$› **by** *presburger*
then show *map* $(\lambda f.\ f\ x)$ *fs* ! *i* = *x* ! $(i + k)$
using *True 0*
by (*metis* (*no-types*, *lifting*) *restrict-apply*)
qed
then have *0*: $\bigwedge i.\ i < n \Longrightarrow (map\ (\lambda f.\ f\ x)\ fs)\ !\ i = (drop\ k\ x)\ !\ i$
using *True nth-drop*
by (*metis add.commute cartesian-power-car-memE le-add2*)
have *1*: *length* (*map* $(\lambda f.\ f\ x)$ *fs*) = *length* (*drop k x*)
using *fs-def True length-drop*[*of k x*]
by (*metis cartesian-power-car-memE length-map length-upt*)
have *2*: *length* (*drop k x*) = n
using *True cartesian-power-car-memE*
by (*metis add-diff-cancel-right′ length-drop*)
show *?thesis* **using** *0 1 2*
by (*metis nth-equalityI*)
qed

then show *?thesis* **using** *True* **unfolding** *restrict-def*
by *presburger*
next
case *False*
then show *?thesis* **unfolding** *restrict-def*
by (*simp add*: *False*)
qed
qed
have *00*: *length* [*k*..<*n*+*k*] = *n*
by *simp*
then have *length fs* = *n*
using *fs-def*
by (*metis length-map*)
then have *2*: *is-semialg-map* (*k* + *n*) *n* (*function-tuple-eval* Q_p (*k* + *n*) *fs*)
using *0 semialg-function-tuple-is-semialg-map*[*of k* + *n fs n*]
by *blast*
show *?thesis* **using** *1 2*
using *semialg-map-on-carrier*[*of k* + *n n function-tuple-eval* Q_p (*k* + *n*) *fs drop k*]
by (*metis add.commute*)
qed

lemma *project-at-indices-is-semialg-map*:
assumes *S* ⊆ {..<*n*}
shows *is-semialg-map n* (*card S*) π_S
proof−
obtain *k* **where** *k-def*: *k* = *card S*
by *blast*
have *0*: *card* {..<*n*} = *n*
by *simp*
have *1*: *finite S*
using *assms finite-subset*
by *blast*
have *2*: *card S* ≤ *n*
using *assms 0 1*
by (*metis card-mono finite-lessThan*)
then have *k-size*: *k* ≤ *n*
using *k-def assms 0 1 2*
by *blast*
obtain *fs* **where** *fs-def*: *fs* = *map* (λ*i*::*nat*. (λ*as* ∈ *carrier* ($Q_p{}^n$). *as*!(*nth-elem S i*))) [*0*..<*k*]
by *blast*
have *4*: *length fs* = *k*
using *fs-def assms 1 k-def*
by (*metis Ex-list-of-length length-map map-nth*)
have *5*: *is-semialg-function-tuple n fs*
proof(*rule is-semialg-function-tupleI*)
fix *f* **assume** *A*: *f* ∈ *set fs*
then obtain *i* **where** *i-def*: *i* ∈ *set* [*0*..<*k*] ∧ *f* = (λ*as* ∈ *carrier* ($Q_p{}^n$).

```
as!(nth-elem S i))
      using A fs-def atLeast-upt 4  in-set-conv-nth  map-eq-conv map-nth
      by auto
    have i-le-k:i < k
    proof−
      have set [0..<k] = {..<k}
        using atLeast-upt by blast
      then show ?thesis
        using i-def
        by blast
    qed
    then have i-in-S: nth-elem S i ∈ S
      using 1 k-def nth-elem-closed by blast
    then have nth-elem S i < n
      using assms
      by blast
    show is-semialg-function n f
      using A index-is-semialg-function[of nth-elem S i n]
            semialg-function-on-carrier[of n (λas. as ! nth-elem S i)] i-def re-
strict-is-semialg
          ‹nth-elem S i < n› by blast
  qed
  have 6: restrict (function-tuple-eval Q_p n fs) (carrier (Q_p^n)) = restrict (π_S)
(carrier (Q_p^n))
  unfolding function-tuple-eval-def
  proof fix x
    show (λx∈carrier (Q_p^n). map (λf. f x) fs) x = restrict (π_S) (carrier (Q_p^n))
x
    proof(cases x ∈ carrier (Q_p^n))
    case True
    have (map (λf. f x) fs) = π_S x
    proof−
      have ⋀i. i < k ⟹ (map (λf. f x) fs) ! i = (π_S x) ! i
      proof− fix i assume A: i < k
        have T0:(map (λf. f x) fs) ! i = (fs!i) x
          using A nth-map by (metis 4)
        have T1: indices-of x = {..< n}
          using True cartesian-power-car-memE indices-of-def
          by blast
        have T2: set (set-to-list S) ⊆ indices-of x
          using assms True  by (simp add: True 1 T1)
        have T3: length x = n
          using True cartesian-power-car-memE by blast
        have T4: ([0..<k] ! i) =   i
          using A by simp
        have T5: nth-elem S i < n
          using assms 0 1 2 A k-def
          by (meson lessThan-iff nth-elem-closed subsetD)
        have T6: nth-elem S ([0..<k] ! i) = nth-elem S i
```

```
      by (simp add: T4)
    have T6: (λas∈carrier (Q_p^n). as ! nth-elem S ([0..<k] ! i)) x = x! (nth-elem
S i)
     proof−
      have (λas∈carrier (Q_p^n). as ! nth-elem S ([0..<k] ! i)) x = x! nth-elem S
([0..<k] ! i)
        using True restrict-def by metis
      then show ?thesis
        using A T3 T4 0 1 2 T5 T6  True
        by metis
     qed
     have T7:  map (λf. f x) fs ! i = x! (nth-elem S i)
         using fs-def T0 A nth-map[of i [0..<k] (λi. λas∈carrier (Q_p^n). as !
nth-elem S i)]
      by (metis 4 T6 length-map)
     show map (λf. f x) fs ! i = π_S x ! i
      using True T0 T1 T2 fs-def 5 unfolding T7
      by (metis A assms(1) k-def project-at-indices-nth)
   qed
   have 1: length (map (λf. f x) fs) = length (π_S x)
    using fs-def True assms proj-at-index-list-length[of S x] k-def
    by (metis 4 cartesian-power-car-memE indices-of-def length-map)
   have 2: length (π_S x) = k
    using assms True 0 1 2
    by (metis 4 length-map)
   show ?thesis using  1 2
    by (metis ‹⋀i. i < k ⟹ map (λf. f x) fs ! i = π_S x ! i› nth-equalityI)
  qed
  then show ?thesis using True unfolding restrict-def
   by presburger
  next
  case False
  then show ?thesis unfolding restrict-def
   by (simp add: False)
  qed
 qed
 have 7:  is-semialg-map n k (function-tuple-eval Q_p n fs)
  using 0 semialg-function-tuple-is-semialg-map[of n fs k] assms fs-def  length-map
4 5 k-size
  by blast
 show ?thesis using 6 7
  using semialg-map-on-carrier k-def
  by blast
qed

lemma tl-is-semialg-map:
 shows is-semialg-map (Suc n) n tl
proof−
 have 0: (card {1..<Suc n}) = n
```

proof−
have *Suc n − 1 = n*
using *diff-Suc-1* **by** *blast*
then show *?thesis*
by *simp*
qed
have *3*: $\{1..<Suc\ n\} \subseteq \{..<Suc\ n\}$
using *atLeastLessThan-iff* **by** *blast*
have *4*: *is-semialg-map* (*Suc n*) *n* (*project-at-indices* $\{1..<Suc\ n\}$)
using *0 project-at-indices-is-semialg-map*
by (*metis 3*)
show *?thesis*
using *0 3 4*
semialg-map-on-carrier[*of Suc n n* (*project-at-indices* $\{1..<Suc\ n\}$) *tl*]
unfolding *restrict-def*
by (*metis* (*no-types*, *lifting*) *tl-as-projection*)
qed

Coordinate functions are semialgebraic maps.

lemma *coord-fun-is-SA*:
assumes *is-semialg-map n m g*
assumes $i < m$
shows *coord-fun* Q_p *n g i* ∈ *carrier* (*SA n*)
proof(*rule SA-car-memI*)
show *coord-fun* Q_p *n g i* ∈ *carrier* (*function-ring* (*carrier* (${Q_p}^n$)) Q_p)
apply(*rule Qp.function-ring-car-memI*)
unfolding *coord-fun-def* **using** *assms*
apply (*metis* (*no-types*, *lifting*) *Pi-iff cartesian-power-car-memE′ is-semialg-map-closed restrict-apply′*)
by (*meson restrict-apply*)
show *is-semialg-function n* (*coord-fun* Q_p *n g i*)
proof−
have *0*: *is-semialg-function m* (λ *x*. *x* ! *i*)
using *assms gr-implies-not0 index-is-semialg-function* **by** *blast*
have *1*: (*coord-fun* Q_p *n g i*) = *restrict* ((λ*x*. *x* ! *i*) ∘ *g*) (*carrier* (${Q_p}^n$))
unfolding *coord-fun-def* **using** *comp-apply*
by *metis*
show *?thesis*
using *semialg-function-on-carrier*[*of n* ((λ*x*. *x* ! *i*) ∘ *g*) *coord-fun* Q_p *n g i*]
assms semialg-function-comp-closed[*of m* λ*x*. *x* ! *i n g*] *assms 0 1*
unfolding *coord-fun-def*
using *restrict-is-semialg* **by** *presburger*
qed
qed

lemma *coord-fun-map-is-semialg-tuple*:
assumes *is-semialg-map n m g*
shows *is-semialg-function-tuple n* (*map* (*coord-fun* Q_p *n g*) [*0*..<*m*])
proof(*rule is-semialg-function-tupleI*)

have *0*: *set* [*0*..<*m*] = {..<*m*}
using *atLeast-upt* **by** *blast*
fix *f* **assume** *A*: *f* ∈ *set* (*map* (*coord-fun* Q_p *n g*) [*0*..<*m*])
then obtain *i* **where** *i-def*: *i* < *m* ∧ *f* = *coord-fun* Q_p *n g i*
using *set-map*[*of coord-fun* Q_p *n g* [*0*..<*m*]] *0*
by (*metis image-iff lessThan-iff*)
show *is-semialg-function n f*
using *i-def A assms coord-fun-is-SA*[*of n m g i*] *SA-imp-semialg* **by** *blast*
qed

lemma *semialg-map-cons*:
assumes *is-semialg-map n m g*
assumes *f* ∈ *carrier* (*SA n*)
shows *is-semialg-map n* (*Suc m*) (λ *x* ∈ *carrier* ($Q_p{}^n$). *f x* # *g x*)
proof−
obtain *Fs* **where** *Fs-def*: *Fs* = *f* # (*map* (*coord-fun* Q_p *n g*) [*0*..<*m*])
by *blast*
have *0*: *is-semialg-function-tuple n Fs*
apply(*rule is-semialg-function-tupleI*)
using *is-semialg-function-tupleE'*[*of n map* (*coord-fun* Q_p *n g*) [*0*..<*m*]]
coord-fun-map-is-semialg-tuple[*of n m g*] *assms SA-car-memE*(*1*)[*of f n*]
set-ConsD[*of - f map* (*coord-fun* Q_p *n g*) [*0*..<*m*]] *assms*
unfolding *Fs-def* **by** *blast*
have *1*: (λ *x* ∈ *carrier* ($Q_p{}^n$). *f x* # *g x*) = (λ *x* ∈ *carrier* ($Q_p{}^n$). *function-tuple-eval* Q_p *n Fs x*)
proof(*rule ext*) **fix** *x* **show** (λ*x*∈*carrier* ($Q_p{}^n$). *f x* # *g x*) *x* = *restrict* (*function-tuple-eval* Q_p *n Fs*) (*carrier* ($Q_p{}^n$)) *x*
proof(*cases x* ∈ *carrier* ($Q_p{}^n$))
case *True* **then have** *T0*: (λ*x*∈*carrier* ($Q_p{}^n$). *f x* # *g x*) *x* = *f x* # *g x*
by (*meson restrict-apply'*)
have *T1*: *restrict* (*function-tuple-eval* Q_p *n Fs*) (*carrier* ($Q_p{}^n$)) *x* = *function-tuple-eval* Q_p *n Fs x*
using *True* **by** (*meson restrict-apply'*)
hence *T2*: *restrict* (*function-tuple-eval* Q_p *n Fs*) (*carrier* ($Q_p{}^n$)) *x* = *f x* # (*function-tuple-eval* Q_p *n* (*map* (*coord-fun* Q_p *n g*) [*0*..<*m*]) *x*)
unfolding *function-tuple-eval-def Fs-def* **by** (*metis* (*no-types*, *lifting*) *list.simps*(*9*))
have *T3*: *g x* ∈ *carrier* ($Q_p{}^m$)
using *assms True is-semialg-map-closed* **by** *blast*
have *length* [*0*..<*m*] = *m*
by *auto*
hence *T4*: *length* (*map* (*coord-fun* Q_p *n g*) [*0*..<*m*]) = *m*
using *length-map*[*of* (*coord-fun* Q_p *n g*) [*0*..<*m*]] **by** *metis*
hence *T5*: *length* (*function-tuple-eval* Q_p *n* (*map* (*coord-fun* Q_p *n g*) [*0*..<*m*]) *x*) = *m*
unfolding *function-tuple-eval-def* **using** *length-map* **by** *metis*
have *T6*: (*function-tuple-eval* Q_p *n* (*map* (*coord-fun* Q_p *n g*) [*0*..<*m*]) *x*) = *g x*
apply(*rule nth-equalityI*) **using** *T3 T5*

using *cartesian-power-car-memE* **apply** *blast*
using *cartesian-power-car-memE*[*of g x* Q_p *m*] *T5 T4 T3 True*
nth-map[*of* - (*map* (λ*i*. λ*x*∈*carrier* ($Q_p{}^n$). *g x* ! *i*) [*0*..<*m*]) (λ*f*. *f x*)]
nth-map[*of* - [*0*..<*m*] (λ*i*. λ*x*∈*carrier* ($Q_p{}^n$). *g x* ! *i*)]
unfolding *function-tuple-eval-def coord-fun-def restrict-def*
by (*metis* (*no-types*, *lifting*) ‹*length* [*0*..<*m*] = *m*› *map-nth nth-map*)
hence *T7*: *restrict* (*function-tuple-eval* Q_p *n Fs*) (*carrier* ($Q_p{}^n$)) *x* = *f x* # *g x*
using *T4 T2* **by** *presburger*
thus *?thesis* **using** *T0*
by *presburger*
next
case *False*
then show *?thesis* **unfolding** *restrict-def*
by *metis*
qed
qed
have *2*: *length Fs* = *Suc m*
unfolding *Fs-def* **using** *length-map*[*of* (*coord-fun* Q_p *n g*) [*0*..<*m*]] *length-Cons*[*of f map* (*coord-fun* Q_p *n g*) [*0*..<*m*]]
using *length-upt* **by** *presburger*
have *3*: *is-semialg-map n* (*Suc m*) (λ *x* ∈ *carrier* ($Q_p{}^n$). *function-tuple-eval* Q_p *n Fs x*)
apply(*rule semialg-map-on-carrier*[*of* - - *function-tuple-eval* Q_p *n Fs*],
intro semialg-function-tuple-is-semialg-map[*of n Fs Suc m*] *0 2*)
by *auto*
show *?thesis* **using** *1 3*
by *presburger*
qed

Extensional Semialgebraic Maps:

definition *semialg-maps* **where**
semialg-maps n m ≡ {*f*. *is-semialg-map n m f* ∧ *f* ∈ *struct-maps* ($Q_p{}^n$) ($Q_p{}^m$)}

lemma *semialg-mapsE*:
assumes *f* ∈ (*semialg-maps n m*)
shows *is-semialg-map n m f*
f ∈ *struct-maps* ($Q_p{}^n$) ($Q_p{}^m$)
f ∈ *carrier* ($Q_p{}^n$) → *carrier* ($Q_p{}^m$)
using *assms* **unfolding** *semialg-maps-def* **apply** *blast*
using *assms* **unfolding** *semialg-maps-def* **apply** *blast*
apply(*rule is-semialg-map-closed*)
using *assms* **unfolding** *semialg-maps-def* **by** *blast*

definition *to-semialg-map* **where**
to-semialg-map n m f = *restrict f* (*carrier* ($Q_p{}^n$))

lemma *to-semialg-map-is-semialg-map*:
assumes *is-semialg-map n m f*

shows *to-semialg-map n m f ∈ semialg-maps n m*
using *assms* **unfolding** *to-semialg-map-def semialg-maps-def struct-maps-def mem-Collect-eq*
using *is-semialg-map-closed′ semialg-map-on-carrier* **by** *force*

14.6 Application of Functions to Segments of Tuples

definition *take-apply* **where**
take-apply m n f = restrict (f ∘ take n) (carrier ($Q_p{}^m$*))*

definition *drop-apply* **where**
drop-apply m n f = restrict (f ∘ drop n) (carrier ($Q_p{}^m$*))*

lemma *take-apply-closed*:
assumes *f ∈ carrier (*$Fun_n\ Q_p$*)*
assumes *k ≥ n*
shows *take-apply k n f ∈ carrier (*$Fun_k\ Q_p$*)*
proof(*rule Qp.function-ring-car-memI*)
show *⋀a. a ∈ carrier (*$Q_p{}^k$*) ⟹ take-apply k n f a ∈ carrier* Q_p
proof− **fix** *a* **assume** *A*: *a ∈ carrier (*$Q_p{}^k$*)* **show** *take-apply k n f a ∈ carrier* Q_p
using *A assms comp-apply*[*of f take n a*] *Qp.function-ring-car-memE*[*of f n*] *take-closed*[*of n k a*]
unfolding *take-apply-def restrict-def*
by *metis*
qed
show *⋀a. a ∉ carrier (*$Q_p{}^k$*) ⟹ take-apply k n f a = undefined*
unfolding *take-apply-def restrict-def*
by *meson*
qed

lemma *take-apply-SA-closed*:
assumes *f ∈ carrier (SA n)*
assumes *k ≥ n*
shows *take-apply k n f ∈ carrier (SA k)*
apply(*rule SA-car-memI*)
using *SA-car-memE*(*2*) *assms*(*1*) *assms*(*2*) *take-apply-closed* **apply** *blast*
using *assms take-is-semialg-map*[*of n k*] **unfolding** *take-apply-def*
by (*metis padic-fields.SA-imp-semialg padic-fields-axioms restrict-is-semialg semialg-function-comp-closed*)

lemma *drop-apply-closed*:
assumes *f ∈ carrier (*$Fun_{k\ -\ n}\ Q_p$*)*
assumes *k ≥ n*
shows *drop-apply k n f ∈ carrier (*$Fun_k\ Q_p$*)*
proof(*rule Qp.function-ring-car-memI*)
show *⋀a. a ∈ carrier (*$Q_p{}^k$*) ⟹ drop-apply k n f a ∈ carrier* Q_p
proof− **fix** *a* **assume** *A*: *a ∈ carrier (*$Q_p{}^k$*)* **show** *drop-apply k n f a ∈ carrier* Q_p

using *A assms comp-apply*[*of f drop n a*] *Qp.function-ring-car-memE*[*of f*] *drop-closed*[*of n k a* Q_p]
unfolding *drop-apply-def restrict-def*
by (*metis* (*no-types*, *opaque-lifting*) *Qp.function-ring-car-memE add-diff-cancel-right' cartesian-power-drop dec-induct diff-diff-cancel diff-less-Suc diff-less-mono2 infinite-descent le-neq-implies-less less-antisym linorder-neqE-nat not-less0 not-less-eq-eq*)
qed
show $\bigwedge a.\ a \notin carrier\ ({Q_p}^k) \Longrightarrow$ *drop-apply k n f a* = *undefined*
unfolding *drop-apply-def restrict-def*
by *meson*
qed

lemma *drop-apply-SA-closed*:
assumes $f \in carrier\ (SA\ (k-n))$
assumes $k \geq n$
shows *drop-apply k n f* $\in carrier\ (SA\ k)$
apply(*rule SA-car-memI*)
using *SA-car-memE*(*2*) *assms*(*1*) *assms*(*2*) *drop-apply-closed less-imp-le-nat* **apply** *blast*
using *assms drop-is-semialg-map*[*of n k* − *n*] *semialg-function-comp-closed*[*of k* − *n f k drop n*] **unfolding** *drop-apply-def*
by (*metis* (*no-types*, *lifting*) *SA-imp-semialg le-add-diff-inverse restrict-is-semialg*)

lemma *take-apply-apply*:
assumes $f \in carrier\ (SA\ n)$
assumes $a \in carrier\ ({Q_p}^n)$
assumes $b \in carrier\ ({Q_p}^k)$
shows *take-apply* $(n+k)$ *n f* $(a@b) = f\ a$
proof−
have $a@b \in carrier\ ({Q_p}^{n+k})$
using *assms cartesian-power-concat*(*1*) **by** *blast*
thus *?thesis*
unfolding *take-apply-def restrict-def*
using *assms cartesian-power-car-memE comp-apply*[*of f take n*]
by (*metis append-eq-conv-conj*)
qed

lemma *drop-apply-apply*:
assumes $f \in carrier\ (SA\ k)$
assumes $a \in carrier\ ({Q_p}^n)$
assumes $b \in carrier\ ({Q_p}^k)$
shows *drop-apply* $(n+k)$ *n f* $(a@b) = f\ b$
proof−
have $a@b \in carrier\ ({Q_p}^{n+k})$
using *assms cartesian-power-concat*(*1*) **by** *blast*
thus *?thesis*
unfolding *drop-apply-def restrict-def*
using *assms cartesian-power-car-memE comp-apply*[*of f drop n*]

by (*metis append-eq-conv-conj*)
qed

lemma *drop-apply-add*:
assumes $f \in carrier\ (SA\ n)$
assumes $g \in carrier\ (SA\ n)$
shows $drop\text{-}apply\ (n+k)\ k\ (f\ \oplus_{SA\ n}\ g) = drop\text{-}apply\ (n+k)\ k\ f\ \oplus_{SA\ (n+k)}$ *drop-apply* $(n+k)\ k\ g$
apply(*rule function-ring-car-eqI*[*of - n + k*])
using *drop-apply-SA-closed assms fun-add-closed SA-car-memE*(*2*) *SA-plus diff-add-inverse2 drop-apply-closed le-add2* **apply** *presburger*
using *drop-apply-SA-closed assms fun-add-closed SA-car-memE*(*2*) *SA-plus diff-add-inverse2 drop-apply-closed le-add2* **apply** *presburger*
proof−
fix *a* **assume** *A*: $a \in carrier\ ({Q_p}^{n+k})$
then obtain *b c* **where** *bc-def*: $b \in carrier\ ({Q_p}^{k}) \wedge c \in carrier\ ({Q_p}^{n}) \wedge a = b@c$
by (*metis* (*no-types*, *lifting*) *add.commute cartesian-power-decomp*)
have *0*: $drop\text{-}apply\ (n+k)\ k\ (f\ \oplus_{SA\ n}\ g)\ a = f\ c \oplus g\ c$
using *assms bc-def drop-apply-apply*[*of f* $\oplus_{SA\ n}$ *g n b k c*]
by (*metis SA-add SA-imp-semialg add.commute padic-fields.SA-add-closed-left padic-fields-axioms*)
then show $drop\text{-}apply\ (n+k)\ k\ (f\ \oplus_{SA\ n}\ g)\ a = (drop\text{-}apply\ (n+k)\ k\ f\ \oplus_{SA\ (n+k)}\ drop\text{-}apply\ (n+k)\ k\ g)\ a$
using *bc-def drop-apply-apply assms*
by (*metis A SA-add add.commute*)
qed

lemma *drop-apply-mult*:
assumes $f \in carrier\ (SA\ n)$
assumes $g \in carrier\ (SA\ n)$
shows $drop\text{-}apply\ (n+k)\ k\ (f\ \otimes_{SA\ n}\ g) = drop\text{-}apply\ (n+k)\ k\ f\ \otimes_{SA\ (n+k)}$ *drop-apply* $(n+k)\ k\ g$
apply(*rule function-ring-car-eqI*[*of - n + k*])
using *drop-apply-SA-closed assms fun-mult-closed SA-car-memE*(*2*) *SA-times diff-add-inverse2 drop-apply-closed le-add2* **apply** *presburger*
using *drop-apply-SA-closed assms fun-mult-closed SA-car-memE*(*2*) *SA-times diff-add-inverse2 drop-apply-closed le-add2* **apply** *presburger*
proof−
fix *a* **assume** *A*: $a \in carrier\ ({Q_p}^{n+k})$
then obtain *b c* **where** *bc-def*: $b \in carrier\ ({Q_p}^{k}) \wedge c \in carrier\ ({Q_p}^{n}) \wedge a = b@c$
by (*metis* (*no-types*, *lifting*) *add.commute cartesian-power-decomp*)
have *0*: $drop\text{-}apply\ (n+k)\ k\ (f\ \otimes_{SA\ n}\ g)\ a = f\ c \otimes g\ c$
using *assms bc-def drop-apply-apply*[*of f* $\otimes_{SA\ n}$ *g n b k c*]
by (*metis SA-imp-semialg SA-mult SA-mult-closed-right add.commute*)
then show $drop\text{-}apply\ (n+k)\ k\ (f\ \otimes_{SA\ n}\ g)\ a = (drop\text{-}apply\ (n+k)\ k\ f\ \otimes_{SA\ (n+k)}\ drop\text{-}apply\ (n+k)\ k\ g)\ a$

using *bc-def drop-apply-apply assms* **by** (*metis A SA-mult add.commute*)
qed

lemma *drop-apply-one*:
shows *drop-apply* (*n*+*k*) *k* $\mathbf{1}_{SA\ n}$ = $\mathbf{1}_{SA\ (n+k)}$
apply(*rule function-ring-car-eqI*[*of* - *n* + *k*])
apply (*metis function-one-closed SA-one add-diff-cancel-right' drop-apply-closed le-add2*)
using *function-one-closed SA-one* **apply** *presburger*
proof−
fix *a* **assume** *A*: *a* ∈ *carrier* ($Q_p{}^{n+k}$)
show *drop-apply* (*n* + *k*) *k* $\mathbf{1}_{SA\ n}$ *a* = $\mathbf{1}_{SA\ (n+k)}$ *a*
unfolding *drop-apply-def restrict-def*
using *SA-one*[*of n*+*k*] *SA-one*[*of n*] *comp-apply*[*of* $\mathbf{1}_{SA\ n}$ *drop k a*] *drop-closed*[*of k n*+*k a* Q_p]
function-ring-defs(4)
unfolding *function-one-def*
by (*metis A function-one-eval add.commute cartesian-power-drop*)
qed

lemma *drop-apply-is-hom*:
shows *drop-apply* (*n* + *k*) *k* ∈ *ring-hom* (*SA n*) (*SA* (*n* + *k*))
apply(*rule ring-hom-memI*)
using *drop-apply-SA-closed*[*of* - *k*+*n k*]
apply (*metis add.commute add-diff-cancel-left' le-add1*)
using *drop-apply-mult* **apply** *blast*
using *drop-apply-add* **apply** *blast*
using *drop-apply-one* **by** *blast*

lemma *take-apply-add*:
assumes *f* ∈ *carrier* (*SA k*)
assumes *g* ∈ *carrier* (*SA k*)
shows *take-apply* (*n*+*k*) *k* (*f* $\oplus_{SA\ k}$ *g*) = *take-apply* (*n*+*k*) *k f* $\oplus_{SA\ (n+k)}$ *take-apply* (*n*+*k*) *k g*
apply(*rule function-ring-car-eqI*[*of* - *n* + *k*])
using *take-apply-SA-closed assms fun-add-closed SA-car-memE*(2) *SA-plus diff-add-inverse2 take-apply-closed le-add2* **apply** *presburger*
using *take-apply-SA-closed assms fun-add-closed SA-car-memE*(2) *SA-plus diff-add-inverse2 take-apply-closed le-add2* **apply** *presburger*
proof−
fix *a* **assume** *A*: *a* ∈ *carrier* ($Q_p{}^{n+k}$)
then obtain *b c* **where** *bc-def*: *b* ∈ *carrier* ($Q_p{}^{k}$) ∧ *c* ∈ *carrier* ($Q_p{}^{n}$) ∧ *a* = *b*@*c*
by (*metis* (*no-types*, *lifting*) *add.commute cartesian-power-decomp*)
hence *0*: *take-apply* (*n* + *k*) *k* (*f* $\oplus_{SA\ k}$ *g*) *a* = *f b* ⊕ *g b*
using *assms bc-def take-apply-apply*[*of f* $\oplus_{SA\ k}$ *g k b c*]
by (*metis SA-add SA-imp-semialg add.commute padic-fields.SA-add-closed-left padic-fields-axioms*)
then show *take-apply* (*n* + *k*) *k* (*f* $\oplus_{SA\ k}$ *g*) *a* = (*take-apply* (*n* + *k*) *k f*

$\oplus_{SA\ (n\ +\ k)}$ *take-apply* $(n\ +\ k)\ k\ g)\ a$
using *bc-def take-apply-apply assms*
by (*metis A SA-add add.commute*)
qed

lemma *take-apply-mult*:
assumes $f \in$ *carrier* $(SA\ k)$
assumes $g \in$ *carrier* $(SA\ k)$
shows *take-apply* $(n+k)\ k\ (f\ \otimes_{SA\ k}\ g) =$ *take-apply* $(n+k)\ k\ f\ \otimes_{SA\ (n\ +\ k)}$ *take-apply* $(n+k)\ k\ g$
apply(*rule function-ring-car-eqI*[*of - n + k*])
using *take-apply-SA-closed assms fun-mult-closed SA-car-memE*(*2*) *SA-times diff-add-inverse2 take-apply-closed le-add2* **apply** *presburger*
using *take-apply-SA-closed assms fun-mult-closed SA-car-memE*(*2*) *SA-times diff-add-inverse2 take-apply-closed le-add2* **apply** *presburger*
proof−
fix *a* **assume** *A*: $a \in$ *carrier* $({Q_p}^{n+k})$
then obtain *b c* **where** *bc-def*: $b \in$ *carrier* $({Q_p}^{k}) \wedge c \in$ *carrier* $({Q_p}^{n}) \wedge a = b@c$
by (*metis* (*no-types*, *lifting*) *add.commute cartesian-power-decomp*)
hence *0*: *take-apply* $(n\ +\ k)\ k\ (f\ \otimes_{SA\ k}\ g)\ a = f\ b\ \otimes\ g\ b$
using *assms bc-def take-apply-apply*[*of f* $\otimes_{SA\ k}$ *g k b c*]
by (*metis SA-mult SA-imp-semialg add.commute padic-fields.SA-mult-closed-left padic-fields-axioms*)
then show *take-apply* $(n\ +\ k)\ k\ (f\ \otimes_{SA\ k}\ g)\ a =$ (*take-apply* $(n\ +\ k)\ k\ f\ \otimes_{SA\ (n\ +\ k)}$ *take-apply* $(n\ +\ k)\ k\ g)\ a$
using *bc-def take-apply-apply assms*
by (*metis A SA-mult add.commute*)
qed

lemma *take-apply-one*:
shows *take-apply* $(n+k)\ k\ \mathbf{1}_{SA\ k} = \mathbf{1}_{SA\ (n+k)}$
apply(*rule function-ring-car-eqI*[*of - n + k*])
using *function-one-closed SA-one le-add2 take-apply-closed* **apply** *presburger*
using *function-one-closed SA-one* **apply** *presburger*
proof−
fix *a* **assume** *A*: $a \in$ *carrier* $({Q_p}^{n+k})$
show *take-apply* $(n\ +\ k)\ k\ \mathbf{1}_{SA\ k}\ a = \mathbf{1}_{SA\ (n\ +\ k)}\ a$
unfolding *take-apply-def restrict-def*
using *SA-one*[*of n+k*] *SA-one*[*of k*] *comp-apply*[*of* $\mathbf{1}_{SA\ k}$ *take k a*] *take-closed*[*of k n + k a*]
function-ring-defs(*4*)
unfolding *function-one-def*
using *A function-one-eval le-add2* **by** *metis*
qed

lemma *take-apply-is-hom*:
shows *take-apply* $(n\ +\ k)\ k \in$ *ring-hom* $(SA\ k)\ (SA\ (n\ +\ k))$

apply(*rule ring-hom-memI*)
using *take-apply-SA-closed*[*of - k n+k*] *le-add2* **apply** *blast*
using *take-apply-mult* **apply** *blast*
using *take-apply-add* **apply** *blast*
using *take-apply-one* **by** *blast*

lemma *drop-apply-units*:
assumes $f \in$ *Units* (*SA n*)
shows *drop-apply* (*n+k*) *k f* $\in$ *Units* (*SA* (*n+k*))
apply(*rule SA-Units-memI*)
using *drop-apply-SA-closed*[*of f n+k k*] *assms SA-Units-closed*
apply (*metis add-diff-cancel-right' le-add2*)
proof−
show $\bigwedge x.\ x \in$ *carrier* (${Q_p}^{n\ +\ k}$) $\Longrightarrow$ *drop-apply* (*n* + *k*) *k f x* $\neq$ **0**
proof− **fix** *x* **assume** *A*: *x* $\in$ *carrier* (${Q_p}^{n+k}$)
then have *drop-apply* (*n* + *k*) *k f x* = *f* (*drop k x*)
unfolding *drop-apply-def restrict-def* **by** (*meson comp-def*)
then show *drop-apply* (*n* + *k*) *k f x* $\neq$ **0**
using *SA-Units-memE'*[*of f n drop k x*]
by (*metis A add.commute assms cartesian-power-drop*)
qed
qed

lemma *take-apply-units*:
assumes *f* $\in$ *Units* (*SA k*)
shows *take-apply* (*n+k*) *k f* $\in$ *Units* (*SA* (*n+k*))
apply(*rule SA-Units-memI*)
using *take-apply-SA-closed*[*of f k n+k*] *assms SA-Units-closed le-add2* **apply** *blast*
proof−
show $\bigwedge x.\ x \in$ *carrier* (${Q_p}^{n\ +\ k}$) $\Longrightarrow$ *take-apply* (*n* + *k*) *k f x* $\neq$ **0**
proof− **fix** *x* **assume** *A*: *x* $\in$ *carrier* (${Q_p}^{n+k}$)
then have *take-apply* (*n* + *k*) *k f x* = *f* (*take k x*)
unfolding *take-apply-def restrict-def* **by** (*meson comp-def*)
then show *take-apply* (*n* + *k*) *k f x* $\neq$ **0**
using *SA-Units-memE'*[*of f k take k x*] *A assms le-add2 local.take-closed* **by** *blast*
qed
qed

14.7 Level Sets of Semialgebraic Functions

lemma *evimage-is-semialg*:
assumes *h* $\in$ *carrier* (*SA n*)
assumes *is-univ-semialgebraic S*
shows *is-semialgebraic n* (*h* $^{-1}_n$ *S*)
proof−
have *0*: *is-semialgebraic 1* (*to-R1* ‘ *S*)
using *assms is-univ-semialgebraicE* **by** *blast*

```
  have 1: h -1_n S = partial-pullback n h (0::nat) (to-R1 ` S)
  proof show h -1_n S ⊆ partial-pullback n h 0 ((λa. [a]) ` S)
    proof fix x assume A: x ∈ h -1_n S
      then have 0: h x ∈ S by blast
      have x-closed: x ∈ carrier (Q_p^n)
        by (meson A evimage-eq)
      have 1: drop n x = []
        using cartesian-power-car-memE[of x Q_p n] drop-all x-closed
        by blast
      have 2: take n x = x
        using cartesian-power-car-memE[of x Q_p n] x-closed take-all
        by blast
      then show x ∈ partial-pullback n h 0 ((λa. [a]) ` S)
        unfolding partial-pullback-def partial-image-def evimage-def
        using 0 1 2 x-closed
        by (metis (no-types, lifting) IntI Nat.add-0-right image-iff vimageI)
    qed
    show partial-pullback n h 0 ((λa. [a]) ` S) ⊆ h -1_n S
    proof fix x assume A: x ∈ partial-pullback n h 0 ((λa. [a]) ` S)
      have x-closed: x ∈ carrier (Q_p^n)
        using A unfolding partial-pullback-def evimage-def
        by (metis A Nat.add-0-right partial-pullback-memE(1))
      then have (partial-image n h) x = [h x]
        unfolding partial-image-def
     by (metis (no-types, opaque-lifting) One-nat-def Qp.zero-closed append.right-neutral
append-Nil
          local.partial-image-def partial-image-eq segment-in-car' Qp.to-R1-closed)
      then have h x ∈ S
        using A unfolding partial-pullback-def
      by (metis (no-types, lifting) A image-iff partial-pullback-memE(2) Qp.to-R-to-R1)
      thus x ∈ h -1_n S
        using x-closed by blast
    qed
  qed
  then show ?thesis
   using 0 is-semialg-functionE[of n h 0 ((λa. [a]) ` S)] assms SA-car-memE(1)[of
h n]
    by (metis Nat.add-0-right SA-car)
qed

lemma semialg-val-ineq-set-is-semialg:
  assumes g ∈ carrier (SA k)
  assumes f ∈ carrier (SA k)
  shows is-semialgebraic k {x ∈ carrier (Q_p^k). val (g x) ≤ val (f x)}
proof−
  obtain F where F-def: F = function-tuple-eval Q_p k [f, g]
    by blast
  have P0: is-semialg-function-tuple k [f, g]
    using is-semialg-function-tupleI[of [f, g] k]
```

by (*metis assms list.distinct*(*1*) *list.set-cases padic-fields.SA-imp-semialg padic-fields-axioms set-ConsD*)
hence *P1*: *is-semialg-map k 2 F*
using *assms semialg-function-tuple-is-semialg-map*[*of k* [*f*, *g*] *2*]
unfolding *F-def* **by** (*simp add*: ‹*f* ∈ *carrier* (*SA k*)› ‹*g* ∈ *carrier* (*SA k*)›)
have {*x* ∈ *carrier* (*Q*$_p$ ^ *k*). *val* (*g x*) ≤ *val* (*f x*)} = *F* ^-1$_k$ *val-relation-set*
proof
show {*x* ∈ *carrier* (*Q*$_p$ ^ *k*). *val* (*g x*) ≤ *val* (*f x*)} ⊆ *F* ^-1$_k$ *val-relation-set*
proof fix *x* **assume** *A*: *x* ∈ {*x* ∈ *carrier* (*Q*$_p$ ^ *k*). *val* (*g x*) ≤ *val* (*f x*)}
then have *0*: *x* ∈ *carrier* (*Q*$_p$ ^ *k*) ∧ *val* (*g x*) ≤ *val* (*f x*) **by** *blast*
have *1*: *F x* = [*f x*, *g x*]
unfolding *F-def* **using** *A* **unfolding** *function-tuple-eval-def*
by (*metis* (*no-types*, *lifting*) *list.simps*(*8*) *map-eq-Cons-conv*)
have *2*: *val* (*g x*) ≤ *val* (*f x*)
using *A*
by *blast*
have *3*: *F x* ∈ *carrier* (*Q*$_p$ ^ *2*)
using *assms A 1*
by (*metis* (*no-types*, *lifting*) *0 Qp.function-ring-car-mem-closed Qp-2I SA-car-memE*(*2*))
then have *4*: *x* ∈ *carrier* (*Q*$_p$ ^ *k*) ∧ *F x* ∈ *val-relation-set*
unfolding *val-relation-set-def F-def* **using** *0 1 2 3*
by (*metis* (*no-types*, *lifting*) *cartesian-power-car-memE cartesian-power-car-memE*′ *list.inject local.F-def one-less-numeral-iff pair-id semiring-norm*(*76*) *val-relation-setI*
val-relation-set-def zero-less-numeral)
then show *x* ∈ *F* ^-1$_k$ *val-relation-set*
by *blast*
qed
show *F* ^-1$_k$ *val-relation-set* ⊆ {*x* ∈ *carrier* (*Q*$_p$ ^ *k*). *val* (*g x*) ≤ *val* (*f x*)}
proof fix *x* **assume** *A*: *x* ∈ *F* ^-1$_k$ *val-relation-set*
then have *0*: *x* ∈ *carrier* (*Q*$_p$ ^ *k*) ∧ *F x* ∈ *val-relation-set*
by (*meson evimage-eq*)
then have *0*: *x* ∈ *carrier* (*Q*$_p$ ^ *k*) ∧ [*f x*, *g x*] ∈ *val-relation-set*
unfolding *F-def function-tuple-eval-def*
by (*metis* (*no-types*, *lifting*) *list.simps*(*8*) *list.simps*(*9*))
then have *x* ∈ *carrier* (*Q*$_p$ ^ *k*) ∧ *val* (*g x*) ≤ *val* (*f x*)
unfolding *F-def val-relation-set-def*
by (*metis* (*no-types*, *lifting*) *0 list.inject val-relation-setE*)
then show *x* ∈ {*x* ∈ *carrier* (*Q*$_p$ ^ *k*). *val* (*g x*) ≤ *val* (*f x*)}
by *blast*
qed
qed
then show *?thesis*
using *assms P0 P1 val-relation-is-semialgebraic semialg-map-evimage-is-semialg*[*of k 2 F val-relation-set*] *pos2*
by *presburger*
qed

lemma *semialg-val-ineq-set-is-semialg′*:
assumes $f \in carrier\ (SA\ k)$
shows *is-semialgebraic k* $\{x \in carrier\ ({Q_p}^k).\ val\ (f\ x) \leq C\}$
proof−
obtain *a* **where** *a-def*: $a \in carrier\ Q_p \wedge val\ a = C$
by (*meson Qp.carrier-not-empty Qp.minus-closed dist-nonempty′ equals0I*)
then obtain *g* **where** *g-def*: $g = constant\text{-}function\ (carrier\ ({Q_p}^k))\ a$
by *blast*
have *0*: $g \in carrier\ (SA\ k)$
using *g-def a-def SA-car assms*(*1*) *constant-function-in-semialg-functions* **by** *blast*
have *1*: $\{x \in carrier\ ({Q_p}^k).\ val\ (f\ x) \leq C\} = \{x \in carrier\ ({Q_p}^k).\ val\ (f\ x) \leq val\ (g\ x)\}$
using *g-def* **by** (*metis* (*no-types*, *lifting*) *Qp-constE a-def*)
then show *?thesis* **using** *assms 0 semialg-val-ineq-set-is-semialg*[*of f k g*]
by *presburger*
qed

lemma *semialg-val-ineq-set-is-semialg′′*:
assumes $f \in carrier\ (SA\ k)$
shows *is-semialgebraic k* $\{x \in carrier\ ({Q_p}^k).\ val\ (f\ x) \geq C\}$
proof−
obtain *a* **where** *a-def*: $a \in carrier\ Q_p \wedge val\ a = C$
by (*meson Qp.carrier-not-empty Qp.minus-closed dist-nonempty′ equals0I*)
then obtain *g* **where** *g-def*: $g = constant\text{-}function\ (carrier\ ({Q_p}^k))\ a$
by *blast*
have *0*: $g \in carrier\ (SA\ k)$
using *g-def a-def SA-car assms*(*1*) *constant-function-in-semialg-functions* **by** *blast*
have *1*: $\{x \in carrier\ ({Q_p}^k).\ val\ (f\ x) \geq C\} = \{x \in carrier\ ({Q_p}^k).\ val\ (f\ x) \geq val\ (g\ x)\}$
using *g-def* **by** (*metis* (*no-types*, *lifting*) *Qp-constE a-def*)
then show *?thesis* **using** *assms 0 semialg-val-ineq-set-is-semialg*[*of g k f*]
by *presburger*
qed

lemma *semialg-level-set-is-semialg*:
assumes $f \in carrier\ (SA\ k)$
assumes $c \in carrier\ Q_p$
shows *is-semialgebraic k* $\{x \in carrier\ ({Q_p}^k).\ f\ x = c\}$
proof−
have *0*: *is-univ-semialgebraic* $\{c\}$
apply(*rule finite-is-univ-semialgebraic*) **using** *assms* **apply** *blast* **by** *auto*
have *1*: $\{x \in carrier\ ({Q_p}^k).\ f\ x = c\} = f\ {}^{-1}_k\ \{c\}$
unfolding *evimage-def* **by** *auto*
then show *?thesis* **using** *0 assms evimage-is-semialg* **by** *presburger*
qed

lemma *semialg-val-eq-set-is-semialg'*:
assumes $f \in$ *carrier* $(SA\ k)$
shows *is-semialgebraic* k $\{x \in$ *carrier* $(Q_p{}^k).$ *val* $(f\ x) = C\}$
proof(*cases* $C = \infty$)
case *True*
then have $\{x \in$ *carrier* $(Q_p{}^k).$ *val* $(f\ x) = C\} = \{x \in$ *carrier* $(Q_p{}^k).\ f\ x = \mathbf{0}\}$
using *assms* **unfolding** *val-def* **by** (*meson eint.distinct*(*1*))
then have $\{x \in$ *carrier* $(Q_p{}^k).$ *val* $(f\ x) = C\} = f\ {}^{-1}{}_k\ \{\mathbf{0}\}$
unfolding *evimage-def* **by** *blast*
then show *?thesis*
using *assms semialg-level-set-is-semialg*[*of f k* **0**] *Qp.zero-closed* ‹$\{x \in$ *carrier* $(Q_p{}^k).$ *val* $(f\ x) = C\} = \{x \in$ *carrier* $(Q_p{}^k).\ f\ x = \mathbf{0}\}$› **by** *presburger*
next
case *False*
then obtain N::*int* **where** *N-def*: $C =$ *eint* N
by *blast*
have *0*: $\{x \in$ *carrier* $(Q_p{}^k).$ *val* $(f\ x) = C\} = \{x \in$ *carrier* $(Q_p{}^k).$ *val* $(f\ x) \leq N\} - \{x \in$ *carrier* $(Q_p{}^k).$ *val* $(f\ x) \leq$ (*eint* $(N-1)$)$\}$
proof
show $\{x \in$ *carrier* $(Q_p{}^k).$ *val* $(f\ x) = C\} \subseteq \{x \in$ *carrier* $(Q_p{}^k).$ *val* $(f\ x) \leq N\} - \{x \in$ *carrier* $(Q_p{}^k).$ *val* $(f\ x) \leq$ (*eint* $(N-1)$)$\}$
proof
fix x **assume** *A*: $x \in \{x \in$ *carrier* $(Q_p{}^k).$ *val* $(f\ x) = C\}$
then have *0*: $x \in$ *carrier* $(Q_p{}^k) \wedge$ *val* $(f\ x) = C$
by *blast*
have *1*: $\neg$ *val* $(f\ x) \leq$ (*eint* $(N-1)$)
using *A N-def assms 0 eint-ord-simps*(*1*) **by** *presburger*
have *2*: *val* $(f\ x) \leq$ (*eint* N)
using *0 N-def eint-ord-simps*(*1*) **by** *presburger*
show $x \in \{x \in$ *carrier* $(Q_p{}^k).$ *val* $(f\ x) \leq N\} - \{x \in$ *carrier* $(Q_p{}^k).$ *val* $(f\ x) \leq$ (*eint* $(N-1)$)$\}$
using *0 1 2*
by *blast*
qed
show $\{x \in$ *carrier* $(Q_p{}^k).$ *val* $(f\ x) \leq N\} - \{x \in$ *carrier* $(Q_p{}^k).$ *val* $(f\ x) \leq$ (*eint* $(N-1)$)$\} \subseteq \{x \in$ *carrier* $(Q_p{}^k).$ *val* $(f\ x) = C\}$
proof fix x **assume** *A*: $x \in \{x \in$ *carrier* $(Q_p{}^k).$ *val* $(f\ x) \leq N\} - \{x \in$ *carrier* $(Q_p{}^k).$ *val* $(f\ x) \leq$ (*eint* $(N-1)$)$\}$
have *0*: $x \in$ *carrier* $(Q_p{}^k) \wedge$ *val* $(f\ x) \leq C$
using *A N-def* **by** *blast*
have *1*: $\neg$ *val* $(f\ x) \leq$ (*eint* $(N-1)$)
using *A 0* **by** *blast*
have *2*: *val* $(f\ x) = C$
proof(*rule ccontr*)
assume *val* $(f\ x) \neq C$
then have *val* $(f\ x) < C$
using *0* **by** *auto*
then obtain M **where** *M-def*: *val* $(f\ x) =$ *eint* M

```
      using N-def eint-iless by blast
    then have M < N
      by (metis N-def ‹val (f x) < C› eint-ord-simps(2))
    then have val (f x) ≤ eint (N − 1)
      using M-def eint-ord-simps(1) by presburger
    then show False using 1 by blast
  qed
  show x ∈ {x ∈ carrier (Q_p^k). val (f x) = C}
    using 0 2 by blast
 qed
qed
have 1: is-semialgebraic k {x ∈ carrier (Q_p^k). val (f x) ≤ N}
 using assms semialg-val-ineq-set-is-semialg'[of f k N] by blast
have 2: is-semialgebraic k {x ∈ carrier (Q_p^k). val (f x) ≤ (eint (N−1))}
 using assms semialg-val-ineq-set-is-semialg' by blast
show ?thesis using 0 1 2
 using diff-is-semialgebraic by presburger
qed

lemma semialg-val-eq-set-is-semialg:
 assumes g ∈ carrier (SA k)
 assumes f ∈ carrier (SA k)
 shows is-semialgebraic k {x ∈ carrier (Q_p^k). val (g x) = val (f x)}
proof−
 have 0: is-semialgebraic k {x ∈ carrier (Q_p^k). val (g x) ≤ val (f x)}
  using assms semialg-val-ineq-set-is-semialg[of g k f] by blast
 have 1: is-semialgebraic k {x ∈ carrier (Q_p^k). val (g x) ≥ val (f x)}
  using assms semialg-val-ineq-set-is-semialg[of f k g] by blast
 have 2: {x ∈ carrier (Q_p^k). val (g x) = val (f x)} = {x ∈ carrier (Q_p^k). val
(g x) ≤ val (f x)} ∩ {x ∈ carrier (Q_p^k). val (g x) ≥ val (f x)}
  using eq-iff by blast
 show ?thesis using 0 1 2 intersection-is-semialg by presburger
qed

lemma semialg-val-strict-ineq-set-formula:
{x ∈ carrier (Q_p^k). val (g x) < val (f x)} = {x ∈ carrier (Q_p^k). val (g x) ≤ val
(f x)} − {x ∈ carrier (Q_p^k). val (g x) = val (f x)}
 using neq-iff le-less by blast

lemma semialg-val-ineq-set-complement:
carrier (Q_p^k) − {x ∈ carrier (Q_p^k). val (g x) ≤ val (f x)} = {x ∈ carrier (Q_p^k).
val (f x) < val (g x)}
 using not-le by blast

lemma semialg-val-strict-ineq-set-complement:
carrier (Q_p^k) − {x ∈ carrier (Q_p^k). val (g x) < val (f x)} = {x ∈ carrier (Q_p^k).
val (f x) ≤ val (g x)}
 using not-le by blast
```

lemma *semialg-val-strict-ineq-set-is-semialg*:
assumes $g \in carrier\ (SA\ k)$
assumes $f \in carrier\ (SA\ k)$
shows $is\text{-}semialgebraic\ k\ \{x \in carrier\ ({Q_p}^k).\ val\ (g\ x) < val\ (f\ x)\}$
using *semialg-val-ineq-set-complement*[*of k f g*] *assms diff-is-semialgebraic semialg-val-ineq-set-is-semialg*[*of f*]
by (*metis* (*no-types*, *lifting*) *complement-is-semialg*)

lemma *semialg-val-strict-ineq-set-is-semialg'*:
assumes $f \in carrier\ (SA\ k)$
shows $is\text{-}semialgebraic\ k\ \{x \in carrier\ ({Q_p}^k).\ val\ (f\ x) < C\}$
proof−
obtain *a* **where** *a-def*: $a \in carrier\ Q_p \wedge val\ a = C$
by (*meson Qp.carrier-not-empty Qp.minus-closed dist-nonempty' equals0I*)
then obtain *g* **where** *g-def*: $g = constant\text{-}function\ (carrier\ ({Q_p}^k))\ a$
by *blast*
have *0*: $g \in carrier\ (SA\ k)$
using *g-def a-def SA-car assms*(*1*) *constant-function-in-semialg-functions* **by** *blast*
have *1*: $\{x \in carrier\ ({Q_p}^k).\ val\ (f\ x) < C\} = \{x \in carrier\ ({Q_p}^k).\ val\ (f\ x) < val\ (g\ x)\}$
using *g-def* **by** (*metis* (*no-types*, *lifting*) *Qp-constE a-def*)
then show *?thesis* **using** *assms 0 semialg-val-strict-ineq-set-is-semialg*[*of f k g*]
by *presburger*
qed

lemma *semialg-val-strict-ineq-set-is-semialg''*:
assumes $f \in carrier\ (SA\ k)$
shows $is\text{-}semialgebraic\ k\ \{x \in carrier\ ({Q_p}^k).\ val\ (f\ x) > C\}$
proof−
obtain *a* **where** *a-def*: $a \in carrier\ Q_p \wedge val\ a = C$
by (*meson Qp.carrier-not-empty Qp.minus-closed dist-nonempty' equals0I*)
then obtain *g* **where** *g-def*: $g = constant\text{-}function\ (carrier\ ({Q_p}^k))\ a$
by *blast*
have *0*: $g \in carrier\ (SA\ k)$
using *g-def a-def SA-car assms*(*1*) *constant-function-in-semialg-functions* **by** *blast*
have *1*: $\{x \in carrier\ ({Q_p}^k).\ val\ (f\ x) > C\} = \{x \in carrier\ ({Q_p}^k).\ val\ (f\ x) > val\ (g\ x)\}$
using *g-def* **by** (*metis* (*no-types*, *lifting*) *Qp-constE a-def*)
then show *?thesis* **using** *assms 0 semialg-val-strict-ineq-set-is-semialg*[*of g k f*]
by *presburger*
qed

lemma *semialg-val-ineq-set-plus*:
assumes $N > 0$
assumes $c \in carrier\ (SA\ N)$
assumes $a \in carrier\ (SA\ N)$
shows $is\text{-}semialgebraic\ N\ \ \{x \in carrier\ ({Q_p}^N).\ val\ (c\ x) \leq val\ (a\ x) + eint\ n\}$

proof−
obtain *b* **where** *b-def*: $b = \mathfrak{p}[\widehat{}]n \odot_{SA\ N} a$
by *blast*
have *b-closed*: $b \in carrier\ (SA\ N)$
unfolding *b-def* **using** *assms SA-smult-closed p-intpow-closed(1)* **by** *blast*
have $\bigwedge x.\ x \in carrier\ ({Q_p}^N) \Longrightarrow val\ (b\ x) = val\ (a\ x) + eint\ n$
unfolding *b-def* **by** (*metis Qp.function-ring-car-memE SA-car-memE(2) SA-smult-formula assms(3) p-intpow-closed(1) times-p-pow-val*)
hence *0*: $\{x \in carrier\ ({Q_p}^N).\ val\ (c\ x) \leq val\ (a\ x) + eint\ n\} = \{x \in carrier\ ({Q_p}^N).\ val\ (c\ x) \leq val\ (b\ x)\}$
by (*metis* (*no-types*, *opaque-lifting*) *add.commute*)
show *?thesis* **unfolding** *0* **using** *assms b-def b-closed semialg-val-ineq-set-is-semialg*[*of c N b*] **by** *blast*
qed

lemma *semialg-val-eq-set-plus*:
assumes $N > 0$
assumes $c \in carrier\ (SA\ N)$
assumes $a \in carrier\ (SA\ N)$
shows *is-semialgebraic N* $\{x \in carrier\ ({Q_p}^N).\ val\ (c\ x) = val\ (a\ x) + eint\ n\}$
proof−
obtain *b* **where** *b-def*: $b = \mathfrak{p}[\widehat{}]n \odot_{SA\ N} a$
by *blast*
have *b-closed*: $b \in carrier\ (SA\ N)$
unfolding *b-def* **using** *assms SA-smult-closed p-intpow-closed(1)* **by** *blast*
have $\bigwedge x.\ x \in carrier\ ({Q_p}^N) \Longrightarrow val\ (b\ x) = val\ (a\ x) + eint\ n$
unfolding *b-def* **by** (*metis Qp.function-ring-car-memE SA-car-memE(2) SA-smult-formula assms(3) p-intpow-closed(1) times-p-pow-val*)
hence *0*: $\{x \in carrier\ ({Q_p}^N).\ val\ (c\ x) = val\ (a\ x) + eint\ n\} = \{x \in carrier\ ({Q_p}^N).\ val\ (c\ x) = val\ (b\ x)\}$
by (*metis* (*no-types*, *opaque-lifting*) *add.commute*)
show *?thesis* **unfolding** *0* **using** *assms b-def b-closed semialg-val-eq-set-is-semialg*[*of c N b*] **by** *blast*
qed

definition *SA-zero-set* **where**
$SA\text{-}zero\text{-}set\ n\ f = \{x \in carrier\ ({Q_p}^n).\ f\ x = \mathbf{0}\}$

lemma *SA-zero-set-is-semialgebraic*:
assumes $f \in carrier\ (SA\ n)$
shows *is-semialgebraic n* (*SA-zero-set n f*)
using *assms semialg-level-set-is-semialg*[*of f n* **0**] **unfolding** *SA-zero-set-def*
by *blast*

lemma *SA-zero-set-memE*:
assumes $f \in carrier\ (SA\ n)$
assumes $x \in SA\text{-}zero\text{-}set\ n\ f$
shows $f\ x = \mathbf{0}$

using *assms* **unfolding** *SA-zero-set-def* **by** *blast*

lemma *SA-zero-set-memI*:
assumes $f \in carrier\ (SA\ n)$
assumes $x \in carrier\ ({Q_p}^n)$
assumes $f\ x = \mathbf{0}$
shows $x \in SA\text{-}zero\text{-}set\ n\ f$
using *assms* **unfolding** *SA-zero-set-def* **by** *blast*

lemma *SA-zero-set-of-zero*:
$SA\text{-}zero\text{-}set\ m\ (\mathbf{0}_{SA\ m}) = carrier\ ({Q_p}^m)$
apply(*rule equalityI'*)
unfolding *SA-zero-set-def mem-Collect-eq*
apply *blast*
using *SA-zeroE* **by** *blast*

definition *SA-nonzero-set* **where**
$SA\text{-}nonzero\text{-}set\ n\ f = \{x \in carrier\ ({Q_p}^n).\ f\ x \neq \mathbf{0}\}$

lemma *nonzero-evimage-closed*:
assumes $f \in carrier\ (SA\ n)$
shows $is\text{-}semialgebraic\ n\ \{x \in carrier\ ({Q_p}^n).\ f\ x \neq \mathbf{0}\}$
proof−
have $\{x \in carrier\ ({Q_p}^n).\ f\ x \neq \mathbf{0}\} = f\ ^{-1}_n\ nonzero\ Q_p$
unfolding *nonzero-def evimage-def* **using** *SA-car-memE*[*of f n*] *assms* **by** *blast*
thus *?thesis* **using** *assms evimage-is-semialg*[*of f n nonzero* Q_p] *nonzero-is-univ-semialgebraic*
by *presburger*
qed

lemma *SA-nonzero-set-is-semialgebraic*:
assumes $f \in carrier\ (SA\ n)$
shows $is\text{-}semialgebraic\ n\ (SA\text{-}nonzero\text{-}set\ n\ f)$
using *assms semialg-level-set-is-semialg*[*of f n* **0**] **unfolding** *SA-nonzero-set-def*
using *nonzero-evimage-closed* **by** *blast*

lemma *SA-nonzero-set-memE*:
assumes $f \in carrier\ (SA\ n)$
assumes $x \in SA\text{-}nonzero\text{-}set\ n\ f$
shows $f\ x \neq \mathbf{0}$
using *assms* **unfolding** *SA-nonzero-set-def* **by** *blast*

lemma *SA-nonzero-set-memI*:
assumes $f \in carrier\ (SA\ n)$
assumes $x \in carrier\ ({Q_p}^n)$
assumes $f\ x \neq \mathbf{0}$
shows $x \in SA\text{-}nonzero\text{-}set\ n\ f$
using *assms* **unfolding** *SA-nonzero-set-def*
by *blast*

lemma *SA-nonzero-set-of-zero*:
SA-nonzero-set m ($\mathbf{0}_{SA\ m}$) = {}
apply(*rule equalityI'*)
unfolding *SA-nonzero-set-def mem-Collect-eq*
using *SA-zeroE* **apply** *blast*
by *blast*

lemma *SA-car-memI'*:
assumes $\bigwedge x.\ x \in carrier\ ({Q_p}^m) \Longrightarrow f\ x \in carrier\ Q_p$
assumes $\bigwedge x.\ x \notin carrier\ ({Q_p}^m) \Longrightarrow f\ x = undefined$
assumes $\bigwedge k\ n\ P.\ n > 0 \Longrightarrow P \in carrier\ (Q_p\ [\mathcal{X}_{1\ +\ k}]) \Longrightarrow$ *is-semialgebraic* (*m* + *k*) (*partial-pullback m f k* (*basic-semialg-set* (1 + *k*) *n P*))
shows *f* ∈ *carrier* (*SA m*)
apply(*rule SA-car-memI*)
apply(*rule Qp.function-ring-car-memI*)
using *assms*(*1*) **apply** *blast* **using** *assms*(*2*) **apply** *blast*
apply(*rule is-semialg-functionI'*)
using *assms*(*1*) **apply** *blast*
using *assms*(*3*) **unfolding** *is-basic-semialg-def*
by *blast*

lemma(**in** *padic-fields*) *SA-zero-set-is-semialg*:
assumes *a* ∈ *carrier* (*SA m*)
shows *is-semialgebraic m* {*x* ∈ *carrier* ($Q_p{}^m$). *a x* = **0**}
using *assms semialg-level-set-is-semialg*[*of a m* **0**] *Qp.zero-closed* **by** *blast*

lemma(**in** *padic-fields*) *SA-nonzero-set-is-semialg*:
assumes *a* ∈ *carrier* (*SA m*)
shows *is-semialgebraic m* {*x* ∈ *carrier* ($Q_p{}^m$). *a x* ≠ **0**}
proof−
have *0*: {*x* ∈ *carrier* ($Q_p{}^m$). *a x* ≠ **0**} = *carrier* ($Q_p{}^m$) − {*x* ∈ *carrier* ($Q_p{}^m$). *a x* = **0**}
by *blast*
show *?thesis* **using** *assms SA-zero-set-is-semialg*[*of a m*] *complement-is-semialg*[*of m* {*x* ∈ *carrier* ($Q_p{}^m$). *a x* = **0**}]
unfolding *0* **by** *blast*
qed

lemma *zero-set-nonzero-set-covers*:
carrier ($Q_p{}^n$) = *SA-zero-set n f* ∪ *SA-nonzero-set n f*
unfolding *SA-zero-set-def SA-nonzero-set-def*
apply(*rule equalityI'*)
unfolding *mem-Collect-eq*
apply *blast*
by *blast*

lemma *zero-set-nonzero-set-covers'*:
assumes *S* ⊆ *carrier* ($Q_p{}^n$)
shows *S* = (*S* ∩ *SA-zero-set n f*) ∪ (*S* ∩ *SA-nonzero-set n f*)

using *assms zero-set-nonzero-set-covers* **by** *blast*

lemma *zero-set-nonzero-set-covers-semialg-set*:
assumes *is-semialgebraic n S*
shows $S = (S \cap SA\text{-}zero\text{-}set\ n\ f) \cup (S \cap SA\text{-}nonzero\text{-}set\ n\ f)$
using *assms is-semialgebraic-closed zero-set-nonzero-set-covers'* **by** *blast*

14.8 Partitioning Semialgebraic Sets According to Valuations of Functions

Given a semialgebraic set A and a finite set of semialgebraic functions Fs, a common construction is to simplify one's understanding of the behaviour of the functions Fs on A by finitely paritioning A into subsets where the element $f \in F$ for which $val(fx)$ is minimal is constant as x ranges over each piece of the parititon. The function `Min_set` helps construct this by picking out the subset of a set A where the valuation of a particular element of Fs is minimal. Such a set will always be semialgebraic.

lemma *disjointify-semialg*:
assumes *finite As*
assumes $As \subseteq$ *semialg-sets n*
shows *disjointify As* $\subseteq$ *semialg-sets n*
using *assms* **unfolding** *semialg-sets-def*
by (*simp add*: *disjointify-gen-boolean-algebra*)

lemma *semialgebraic-subalgebra*:
assumes *finite Xs*
assumes $Xs \subseteq$ *semialg-sets n*
shows *atoms-of Xs* $\subseteq$ *semialg-sets n*
using *assms* **unfolding** *semialg-sets-def*
by (*simp add*: *atoms-of-gen-boolean-algebra*)

lemma(**in** *padic-fields*) *finite-intersection-is-semialg*:
assumes *finite Xs*
assumes $Xs \neq \{\}$
assumes $F\ `\ Xs \subseteq$ *semialg-sets m*
shows *is-semialgebraic m* $(\bigcap\ i \in Xs.\ F\ i)$
proof−
have *0*: $F\ `\ Xs \subseteq$ *semialg-sets m* $\wedge\ F\ `\ Xs \neq \{\}$
using *assms* **by** *blast*
thus *?thesis*
using *assms finite-intersection-is-semialgebraic*[*of F ` Xs m*]
by *blast*
qed

definition *Min-set* **where**
Min-set m As a = *carrier* $(Q_p{}^m) \cap (\bigcap\ f \in As.\ \{x \in carrier\ (Q_p{}^m).\ val\ (a\ x) \leq val\ (f\ x)\ \})$

lemma *Min-set-memE*:
assumes $x \in$ *Min-set m As a*
assumes $f \in$ *As*
shows *val* $(a\ x) \leq$ *val* $(f\ x)$
using *assms* **unfolding** *Min-set-def* **by** *blast*

lemma *Min-set-closed*:
Min-set m As a $\subseteq$ *carrier* $({Q_p}^m)$
unfolding *Min-set-def* **by** *blast*

lemma *Min-set-semialg0*:
assumes *As* $\subseteq$ *carrier* $(SA\ m)$
assumes *finite As*
assumes $a \in$ *As*
assumes *As* $\neq \{\}$
shows *is-semialgebraic m* (*Min-set m As a*)
unfolding *Min-set-def* **apply**(*rule intersection-is-semialg*)
using *carrier-is-semialgebraic* **apply** *blast*
apply(*rule finite-intersection-is-semialg*)
using *assms* **apply** *blast*
using *assms* **apply** *blast*
proof(*rule subsetI*) **fix** *x* **assume** *A*: $x \in (\lambda i.\ \{x \in$ *carrier* $({Q_p}^m).$ *val* $(a\ x) \leq$ *val* $(i\ x)\})$ ‘ *As*
then obtain *f* **where** *f-def*: $f \in$ *As* $\wedge\ x = \{x \in$ *carrier* $({Q_p}^m).$ *val* $(a\ x) \leq$ *val* $(f\ x)\}$
by *blast*
have *f-closed*: $f \in$ *carrier* $(SA\ m)$
using *f-def assms* **by** *blast*
have *a-closed*: $a \in$ *carrier* $(SA\ m)$
using *assms* **by** *blast*
have *is-semialgebraic m* $\{x \in$ *carrier* $({Q_p}^m).$ *val* $(a\ x) \leq$ *val* $(f\ x)\}$
using *a-closed f-closed semialg-val-ineq-set-is-semialg* **by** *blast*
thus $x \in$ *semialg-sets m*
using *f-def* **unfolding** *is-semialgebraic-def* **by** *blast*
qed

lemma *Min-set-semialg*:
assumes *As* $\subseteq$ *carrier* $(SA\ m)$
assumes *finite As*
assumes $a \in$ *As*
shows *is-semialgebraic m* (*Min-set m As a*)
apply(*cases As* $= \{\}$)
using *Min-set-def assms*(*3*) **apply** *blast*
using *assms Min-set-semialg0* **by** *blast*

lemma *Min-sets-cover*:
assumes *As* $\neq \{\}$
assumes *finite As*

shows *carrier* $(Q_p{}^m) = (\bigcup\ a \in As.\ Min\text{-}set\ m\ As\ a)$
proof
show *carrier* $(Q_p{}^m) \subseteq \bigcup\ (Min\text{-}set\ m\ As\ `\ As)$
proof fix x **assume** A: $x \in carrier\ (Q_p{}^m)$
obtain v **where** $v\text{-}def$: $v = Min\ ((\lambda f.\ val\ (f\ x))\ `\ As)$
by *blast*
obtain f **where** $f\text{-}def$: $f \in As \wedge v = val\ (f\ x)$
unfolding $v\text{-}def$ **using** *assms Min-in*[*of* $((\lambda f.\ val\ (f\ x))\ `\ As)$]
by *blast*
have $v\text{-}def'$: $v = val\ (f\ x)$
using $f\text{-}def$ **by** *blast*
have 0: $x \in Min\text{-}set\ m\ As\ f$
unfolding *Min-set-def*
apply(*rule IntI*)
using A **apply** *blast*
proof(*rule InterI*) **fix** s **assume** s: $s \in (\lambda fa.\ \{x \in carrier\ (Q_p{}^m).\ val\ (f\ x) \leq val\ (fa\ x)\})\ `\ As$
then obtain g **where** $g\text{-}def$: $g \in As \wedge s = \{x \in carrier\ (Q_p{}^m).\ val\ (f\ x) \leq val\ (g\ x)\}$
by *blast*
have $s\text{-}def$: $s = \{x \in carrier\ (Q_p{}^m).\ val\ (f\ x) \leq val\ (g\ x)\}$
using $g\text{-}def$ **by** *blast*
have 00: $val\ (g\ x) \in ((\lambda f.\ val\ (f\ x))\ `\ As)$
using $g\text{-}def$ **by** *blast*
show $x \in s$
unfolding $s\text{-}def$ *mem-Collect-eq* **using** 00 A *assms MinE*[*of* $((\lambda f.\ val\ (f\ x))\ `\ As)$ v $val\ (g\ x)$]
unfolding $v\text{-}def$ **by** (*metis f-def finite-imageI v-def*)
qed
thus $x \in \bigcup\ (Min\text{-}set\ m\ As\ `\ As)$
using $f\text{-}def$ **by** *blast*
qed
show $\bigcup\ (Min\text{-}set\ m\ As\ `\ As) \subseteq carrier\ (Q_p{}^m)$
unfolding *Min-set-def* **by** *blast*
qed

The sets defined by the function `Min_set` for a fixed set of functions may not all be disjoint, but we can easily refine then to obtain a finite partition via the function "disjointify".

definition *Min-set-partition* **where**
$Min\text{-}set\text{-}partition\ m\ As\ B = disjointify\ ((\cap)B\ `\ (Min\text{-}set\ m\ As\ `\ As))$

lemma *Min-set-partition-finite*:
assumes *finite As*
shows *finite* (*Min-set-partition m As B*)
unfolding *Min-set-partition-def*
by (*meson assms disjointify-finite finite-imageI*)

lemma *Min-set-partition-semialg0*:

assumes *finite As*
assumes *As* ⊆ *carrier* (*SA m*)
assumes *is-semialgebraic m B*
assumes *S* ∈ ((∩)*B* ‘ (*Min-set m As* ‘ *As*))
shows *is-semialgebraic m S*
using *Min-set-semialg*[*of As m*] *assms intersection-is-semialg*[*of m B*]
by *blast*

lemma *Min-set-partition-semialg*:
assumes *finite As*
assumes *As* ⊆ *carrier* (*SA m*)
assumes *is-semialgebraic m B*
assumes *S* ∈ (*Min-set-partition m As B*)
shows *is-semialgebraic m S*
proof−
have *0*: (∩) *B* ‘ *Min-set m As* ‘ *As* ⊆ *semialg-sets m*
apply(*rule subsetI*)
using *Min-set-partition-semialg0*[*of As m B*] *assms* **unfolding** *is-semialgebraic-def*
by *blast*
thus *?thesis*
unfolding *is-semialgebraic-def*
using *assms Min-set-partition-semialg0*[*of As m B*] *disjointify-semialg*[*of* ((∩) *B* ‘ *Min-set m As* ‘ *As*) *m*]
unfolding *Min-set-partition-def is-semialgebraic-def* **by** *blast*
qed

lemma *Min-set-partition-covers0*:
assumes *finite As*
assumes *As* ≠ {}
assumes *As* ⊆ *carrier* (*SA m*)
assumes *is-semialgebraic m B*
shows ⋃ ((∩)*B* ‘ (*Min-set m As* ‘ *As*)) = *B*
proof−
have *0*: ⋃ ((∩)*B* ‘ (*Min-set m As* ‘ *As*)) = *B* ∩ ⋃ (*Min-set m As* ‘ *As*)
by *blast*
have *1*: *B* ⊆ *carrier* ($Q_p{}^m$)
using *assms is-semialgebraic-closed* **by** *blast*
show *?thesis* **unfolding** *0* **using** *1 assms Min-sets-cover*[*of As m*] **by** *blast*
qed

lemma *Min-set-partition-covers*:
assumes *finite As*
assumes *As* ⊆ *carrier* (*SA m*)
assumes *As* ≠ {}
assumes *is-semialgebraic m B*
shows ⋃ (*Min-set-partition m As B*) = *B*
unfolding *Min-set-partition-def*
using *Min-set-partition-covers0*[*of As m B*] *assms disjointify-union*[*of* ((∩) *B* ‘ *Min-set m As* ‘ *As*)]

by (*metis finite-imageI*)

lemma *Min-set-partition-disjoint*:
assumes *finite As*
assumes $As \subseteq carrier\ (SA\ m)$
assumes $As \neq \{\}$
assumes *is-semialgebraic m B*
assumes $s \in$ *Min-set-partition m As B*
assumes $s' \in$ *Min-set-partition m As B*
assumes $s \neq s'$
shows $s \cap s' = \{\}$
apply(*rule disjointify-is-disjoint*[*of* $((\cap)\ B\ `$ *Min-set m As* $`\ As)\ s\ s'$])
using *assms finite-imageI* **apply** *blast*
using *assms* **unfolding** *Min-set-partition-def* **apply** *blast*
using *assms* **unfolding** *Min-set-partition-def* **apply** *blast*
using *assms* **by** *blast*

lemma *Min-set-partition-memE*:
assumes *finite As*
assumes $As \subseteq carrier\ (SA\ m)$
assumes $As \neq \{\}$
assumes *is-semialgebraic m B*
assumes $s \in$ *Min-set-partition m As B*
shows $\exists f \in As.\ (\forall x \in s.\ (\forall g \in As.\ val\ (f\ x) \leq val\ (g\ x)))$
proof−
obtain s' **where** s'-*def*: $s' \in ((\cap)\ B\ `$ *Min-set m As* $`\ As) \wedge s \subseteq s'$
using *finite-imageI assms disjointify-subset*[*of* $((\cap)\ B\ `$ *Min-set m As* $`\ As)\ s$]
unfolding *Min-set-partition-def* **by** *blast*
obtain f **where** *f-def*: $f \in As \wedge s' = B \cap$ *Min-set m As f*
using s'-*def* **by** *blast*
have *0*: $(\forall x \in s'.\ (\forall g \in As.\ val\ (f\ x) \leq val\ (g\ x)))$
using *f-def Min-set-memE*[*of - m As f*] **by** *blast*
thus *?thesis*
using s'-*def* **by** (*meson f-def subset-iff*)
qed

14.9 Valuative Congruence Sets for Semialgebraic Functions

The set of points x where the values $ord\ f(x)$ satisfy a congruence are important basic examples of semialgebraic sets, and will be vital in the proof of Macintyre's Theorem. The lemma below is essentially the content of Denef's Lemma 2.1.3 from his cell decomposition paper.

lemma *pre-SA-unit-cong-set-is-semialg*:
assumes $k \geq 0$
assumes $f \in Units\ (SA\ n)$
shows *is-semialgebraic n* $\{x \in carrier\ ({Q_p}^n).\ ord\ (f\ x)\ mod\ k\ = a\ \}$
proof−
have *0*: $\{x \in carrier\ ({Q_p}^n).\ ord\ (f\ x)\ mod\ k\ = a\ \} = f\ ^{-1}_n$ *ord-congruence-set*

k a
unfolding *ord-congruence-set-def*
apply(*rule equalityI'*)
using *assms* **unfolding** *evimage-def vimage-def mem-Collect-eq*
apply (*metis* (*mono-tags*, *lifting*) *IntI Qp.function-ring-car-memE SA-Units-closed SA-Units-memE' SA-car-memE*(*2*) *mem-Collect-eq not-nonzero-Qp*)
using *assms* **by** *blast*
show *?thesis* **unfolding** *0*
apply(*rule evimage-is-semialg*)
using *assms* **apply** *blast*
using *assms ord-congruence-set-univ-semialg*[*of k a*]
by *blast*
qed

lemma *SA-unit-cong-set-is-semialg*:
assumes $f \in$ *Units* (*SA n*)
shows *is-semialgebraic n* $\{x \in$ *carrier* $(Q_p{}^n)$. *ord* $(f\ x)$ *mod* $k = a\}$
proof(*cases* $k \geq 0$)
case *True*
then show *?thesis*
using *assms pre-SA-unit-cong-set-is-semialg* **by** *presburger*
next
case *False*
show *?thesis*
proof(*cases* $a = 0$)
case *True*
have *T0*: $\{x \in$ *carrier* $(Q_p{}^n)$. *ord* $(f\ x)$ *mod* $k\ = a\ \} = \{x \in$ *carrier* $(Q_p{}^n)$. *ord* $(f\ x)$ *mod* $(-k)\ = a\ \}$
apply(*rule equalityI'*)
unfolding *mem-Collect-eq* **using** *True zmod-zminus2-not-zero* **apply** *meson*
using *True zmod-zminus2-not-zero*
by (*metis equation-minus-iff*)
show *?thesis* **unfolding** *T0* **apply**(*rule pre-SA-unit-cong-set-is-semialg*[*of* $-k$ *f n a*])
using *False* **apply** *presburger* **using** *assms* **by** *blast*
next
case *F*: *False*
show *?thesis*
proof(*cases* $a = k$)
case *True*
have *0*: $\{x \in$ *carrier* $(Q_p{}^n)$. *ord* $(f\ x)$ *mod* $k = a\} = \{x \in$ *carrier* $(Q_p{}^n)$. *ord* $(f\ x)$ *mod* $k = k\}$
using *True* **by** *blast*
have *1*: $\{x \in$ *carrier* $(Q_p{}^n)$. *ord* $(f\ x)$ *mod* $k = a\} = \{\} \vee k = 0$
proof(*cases* $\{x \in$ *carrier* $(Q_p{}^n)$. *ord* $(f\ x)$ *mod* $k = a\} \neq \{\}$)
case *T*: *True*
then obtain *x* **where** *ord* $(f\ x)$ *mod* $k = k$
unfolding *True* **by** *blast*
then have $k = 0$

```
      by (metis mod-mod-trivial mod-self)
    thus ?thesis by blast
  next
    case False
    then show ?thesis by blast
  qed
  show ?thesis apply(cases {x ∈ carrier (Q_p^n). ord (f x) mod k = a} = {})
    using empty-is-semialgebraic apply presburger
    using 1 pre-SA-unit-cong-set-is-semialg assms by blast
 next
  case F': False
  have 0: {x ∈ carrier (Q_p^n). ord (f x) mod k = a} = {x ∈ carrier (Q_p^n). ord
(f x) mod (−k) = a − k}
    apply(rule equalityI')
     unfolding mem-Collect-eq using zmod-zminus2-eq-if assms apply (metis
F)
    unfolding mem-Collect-eq zmod-zminus2-eq-if using False F F' assms
  by (metis (no-types, opaque-lifting) cancel-ab-semigroup-add-class.diff-right-commute
group-add-class.right-minus-eq)
  show ?thesis unfolding 0 apply(rule pre-SA-unit-cong-set-is-semialg)
    using False apply presburger using assms by blast
 qed
qed
qed
```

14.10 Gluing Functions Along Semialgebraic Sets

Semialgebraic functions have the useful property that they are closed under piecewise definitions. That is, if f, g are semialgebraic and $C \subseteq \mathbb{Q}_p^m$ is a semialgebraic set, then the function:

$$h(x) = \begin{cases} f(x) & \text{if } x \in C \\ g(x) & \text{if } x \in \mathbb{Q}_p^m - C \\ undefined & \text{otherwise} \end{cases}$$

is again semialgebraic. The function h can be obtained by the definition

```
h = fun_glue m C f g
```

which is defined below. This closure property means that we can avoid having to define partial semialgebraic functions which are undefined outside of some proper subset of $\mathbb{Q}_p^m$, since it usually suffices to just define the function as some arbitrary constant outside of the desired domain. This is useful for defining partial multiplicative inverses of arbitrary functions. If f is semialgebraic, then its nonzero set $\{x \in \mathbb{Q}_p^m \mid fx \neq 0\}$ is semialgebraic. By gluing f to the constant function 1 outside of its nonzero set, we obtain an invertible element in the ring `SA(m)` which evaluates to a multiplicative inverse of $f(x)$ on the largest domain possible.

14.10.1 Defining Piecewise Semialgebraic Functions

An important property that will be repeatedly used is that we can define piecewise semialgebraic functions, which will themselves be semialgebraic as long as the pieces are semialgebraic sets. An important application of this principle will be that a function f which is always nonzero on some semialgebraic set A can be replaced with a global unit in the ring of semialgebraic functions. This global unit admits a global multiplicative inverse that inverts f pointwise on A, and allows us to avoid having to consider localizations of function rings to locally invert such functions.

definition *fun-glue* **where**
fun-glue n S f g = (λ*x* ∈ *carrier* (*Q*$_p$^*n*). *if x* ∈ *S then f x else g x*)

lemma *fun-glueE*:
assumes *f* ∈ *carrier* (*SA n*)
assumes *g* ∈ *carrier* (*SA n*)
assumes *S* ⊆ *carrier* (*Q*$_p$^*n*)
assumes *x* ∈ *S*
shows *fun-glue n S f g x* = *f x*
proof−
have *x* ∈ *carrier* (*Q*$_p$^*n*)
using *assms* **by** *blast*
thus *?thesis*
unfolding *fun-glue-def* **using** *assms*
by (*metis* (*mono-tags*, *lifting*) *restrict-apply'*)
qed

lemma *fun-glueE'*:
assumes *f* ∈ *carrier* (*SA n*)
assumes *g* ∈ *carrier* (*SA n*)
assumes *S* ⊆ *carrier* (*Q*$_p$^*n*)
assumes *x* ∈ *carrier* (*Q*$_p$^*n*) − *S*
shows *fun-glue n S f g x* = *g x*
proof−
have *0*: *x* ∈ *carrier* (*Q*$_p$^*n*)
using *assms* **by** *blast*
have *1*: *x* ∉ *S*
using *assms* **by** *blast*
show *?thesis*
unfolding *fun-glue-def* **using** *assms 0 1*
by (*metis* (*mono-tags*, *lifting*) *restrict-apply'*)
qed

lemma *fun-glue-evimage*:
assumes *f* ∈ *carrier* (*SA n*)
assumes *g* ∈ *carrier* (*SA n*)
assumes *S* ⊆ *carrier* (*Q*$_p$^*n*)
shows *fun-glue n S f g* ^{-1}n *T* = ((*f* ^{-1}n *T*) ∩ *S*) ∪ ((*g* ^{-1}n *T*) − *S*)

proof
show *fun-glue n S f g* $^{-1}_n$ *T* $\subseteq$ ((*f* $^{-1}_n$ *T*) $\cap$ *S*) $\cup$ ((*g* $^{-1}_n$ *T*) $-$ *S*)
proof fix *x* **assume** *A*: *x* $\in$ *fun-glue n S f g* $^{-1}_n$ *T*
then have *0*: *fun-glue n S f g x* $\in$ *T*
by *blast*
have *1*: *x* $\in$ *carrier* ($Q_p{}^n$)
using *A* **by** (*meson evimage-eq*)
show *x* $\in$ ((*f* $^{-1}_n$ *T*) $\cap$ *S*) $\cup$ ((*g* $^{-1}_n$ *T*) $-$ *S*)
apply(*cases x* $\in$ *S*)
apply *auto*[*1*]
using *1* **apply** *force*
using *0 assms*(*1*) *assms*(*2*) *assms*(*3*) *fun-glueE* **apply** *force*
apply *auto*[*1*] **using** *1* **apply** *blast*
using *A 1* **unfolding** *fun-glue-def evimage-def Int-iff* **by** *auto*
qed
show *f* $^{-1}_n$ *T* $\cap$ *S* $\cup$ (*g* $^{-1}_n$ *T* $-$ *S*) $\subseteq$ *fun-glue n S f g* $^{-1}_n$ *T*
proof fix *x* **assume** *A*: *x* $\in$ *f* $^{-1}_n$ *T* $\cap$ *S* $\cup$ (*g* $^{-1}_n$ *T* $-$ *S*)
then have *x-closed*: *x* $\in$ *carrier* ($Q_p{}^n$)
by (*metis* (*no-types*, *opaque-lifting*) *Diff-iff Int-iff UnE extensional-vimage-closed subsetD*)
show *x* $\in$ *fun-glue n S f g* $^{-1}_n$ *T*
apply(*cases x* $\in$ *S*)
using *x-closed fun-glueE assms*
apply (*metis A DiffD2 IntD1 UnE evimage-eq*)
using *x-closed fun-glueE*′ *assms*
by (*metis A Diff-iff Int-iff Un-iff evimageD evimageI2*)
qed
qed

lemma *fun-glue-partial-pullback*:
assumes *f* $\in$ *carrier* (*SA k*)
assumes *g* $\in$ *carrier* (*SA k*)
assumes *S* $\subseteq$ *carrier* ($Q_p{}^k$)
shows *partial-pullback k* (*fun-glue k S f g*) *n T* =
((*cartesian-product S* (*carrier* ($Q_p{}^n$))) $\cap$ *partial-pullback k f n T*) $\cup$ ((*partial-pullback k g n T*)$-$ (*cartesian-product S* (*carrier* ($Q_p{}^n$))))
proof
show *partial-pullback k* (*fun-glue k S f g*) *n T* $\subseteq$ (*cartesian-product S* (*carrier* ($Q_p{}^n$))) $\cap$ *partial-pullback k f n T* $\cup$ (*partial-pullback k g n T* $-$ (*cartesian-product S* (*carrier* ($Q_p{}^n$))))
proof fix *x* **assume** *A*: *x* $\in$ *partial-pullback k* (*fun-glue k S f g*) *n T*
then have *x-closed*: *x* $\in$ *carrier* ($Q_p{}^{k+n}$) **unfolding** *partial-pullback-def partial-image-def*
by (*meson evimage-eq*)
show *x* $\in$ *cartesian-product S* (*carrier* ($Q_p{}^n$)) $\cap$ *partial-pullback k f n T* $\cup$ (*partial-pullback k g n T* $-$ (*cartesian-product S* (*carrier* ($Q_p{}^n$))))
proof(*cases x* $\in$ *cartesian-product S* (*carrier* ($Q_p{}^n$)))
case *True*
then have *T0*: *take k x* $\in$ *S*

using *assms cartesian-product-memE*(*1*) **by** *blast*
then have (*fun-glue k S f g*) (*take k x*) = *f* (*take k x*)
using *assms fun-glueE*[*of f k g S take k x*]
by *blast*
then have *partial-image k* (*fun-glue k S f g*) *x* = *partial-image k f x*
using *A x-closed* **unfolding** *partial-pullback-def partial-image-def*
by *blast*
then show *?thesis* **using** *T0 A* **unfolding** *partial-pullback-def evimage-def*
by (*metis IntI Int-iff True Un-iff vimageI vimage-eq x-closed*)
next
case *False*
then have *F0*: *take k x* ∈ *carrier* ($Q_p{}^k$) − *S*
using *A x-closed assms cartesian-product-memI*
by (*metis* (*no-types*, *lifting*) *DiffI carrier-is-semialgebraic cartesian-power-drop is-semialgebraic-closed le-add1 local.take-closed*)
then have (*fun-glue k S f g*) (*take k x*) = *g* (*take k x*)
using *assms fun-glueE'*[*of f k g S take k x*]
by *blast*
then have *partial-image k* (*fun-glue k S f g*) *x* = *partial-image k g x*
using *A x-closed* **unfolding** *partial-pullback-def partial-image-def*
by *blast*
then have *x* ∈ *partial-pullback k g n T*
using *F0 x-closed A* **unfolding** *partial-pullback-def partial-image-def evimage-def*
by (*metis* (*no-types*, *lifting*) *A IntI local.partial-image-def partial-pullback-memE*(*2*) *vimageI*)
then have *x* ∈ (*partial-pullback k g n T* − *cartesian-product S* (*carrier* ($Q_p{}^n$)))
using *False* **by** *blast*
then show *?thesis* **by** *blast*
qed
qed
show *cartesian-product S* (*carrier* ($Q_p{}^n$)) ∩ *partial-pullback k f n T* ∪ (*partial-pullback k g n T* − *cartesian-product S* (*carrier* ($Q_p{}^n$)))
⊆ *partial-pullback k* (*fun-glue k S f g*) *n T*
proof fix *x* **assume** *A*: *x* ∈ *cartesian-product S* (*carrier* ($Q_p{}^n$)) ∩ *partial-pullback k f n T* ∪ (*partial-pullback k g n T* − *cartesian-product S* (*carrier* ($Q_p{}^n$)))
then have *x-closed*: *x* ∈ *carrier* ($Q_p{}^{n+k}$)
by (*metis DiffD1 Int-iff Un-iff add.commute partial-pullback-memE*(*1*))
show *x* ∈ *partial-pullback k* (*fun-glue k S f g*) *n T*
proof(*cases x* ∈ *cartesian-product S* (*carrier* ($Q_p{}^n$)) ∩ *partial-pullback k f n T*)
case *True*
show *?thesis* **apply**(*rule partial-pullback-memI*)
using *x-closed* **apply** (*metis add.commute*)
using *x-closed True assms fun-glueE*[*of f k g S take k x*] *partial-pullback-memE*[*of x k f n T*]
unfolding *partial-image-def* **by** (*metis Int-iff cartesian-product-memE*(*1*))
next
case *False*

show *?thesis* **apply**(*rule partial-pullback-memI*)
using *x-closed* **apply** (*metis add.commute*)
using *A x-closed False assms fun-glueE'*[*of f k g S take k x*] *partial-pullback-memE*[*of x k g n T*]
unfolding *partial-image-def*
by (*metis* (*no-types*, *lifting*) *Diff-iff Un-iff carrier-is-semialgebraic cartesian-power-drop cartesian-product-memI is-semialgebraic-closed le-add2 local.take-closed*)
qed
qed
qed

lemma *fun-glue-eval-closed*:
assumes $f \in$ *carrier* (*SA n*)
assumes $g \in$ *carrier* (*SA n*)
assumes *is-semialgebraic n S*
assumes $x \in$ *carrier* ($Q_p{}^n$)
shows *fun-glue n S f g x* $\in$ *carrier* Q_p
apply(*cases* $x \in S$)
using *assms fun-glueE SA-car-memE*
apply (*metis Qp.function-ring-car-mem-closed is-semialgebraic-closed*)
proof− **assume** *A*: $x \notin S$
then have *0*: $x \in$ *carrier* ($Q_p{}^n$) − *S*
using *assms* **by** *auto*
hence *1*: *fun-glue n S f g x* = *g x*
using *assms fun-glueE' is-semialgebraic-closed* **by** *auto*
show *fun-glue n S f g x* $\in$ *carrier* Q_p
unfolding *1* **using** *assms SA-car-memE* **by** *blast*
qed

lemma *fun-glue-closed*:
assumes $f \in$ *carrier* (*SA n*)
assumes $g \in$ *carrier* (*SA n*)
assumes *is-semialgebraic n S*
shows *fun-glue n S f g* $\in$ *carrier* (*SA n*)
apply(*rule SA-car-memI*)
apply(*rule Qp.function-ring-car-memI*)
using *fun-glue-eval-closed assms* **apply** *blast*
using *fun-glue-def* **unfolding** *restrict-apply* **apply** *metis*
apply(*rule is-semialg-functionI*, *intro Pi-I fun-glue-eval-closed assms*, *blast*)
proof−
fix *k T* **assume** *A*: *T* $\in$ *semialg-sets* (*1* + *k*)
have *0*: *is-semialgebraic* (*n*+*k*) (*partial-pullback n f k T*)
using *assms A SA-car-memE*[*of f n*] *is-semialg-functionE*[*of n f k T*] *padic-fields.is-semialgebraicI padic-fields-axioms* **by** *blast*
have *1*: *is-semialgebraic* (*n*+*k*) (*partial-pullback n g k T*)
using *assms A SA-car-memE*[*of g n*] *is-semialg-functionE*[*of n g k T*] *padic-fields.is-semialgebraicI padic-fields-axioms* **by** *blast*
have *2*: *partial-pullback n* (*fun-glue n S f g*) *k T* =
cartesian-product S (*carrier* ($Q_p{}^k$)) ∩ *partial-pullback n f k T* ∪ (*partial-pullback*

n g k T − cartesian-product S (carrier ($Q_p{}^k$*)))*
using *assms fun-glue-partial-pullback[of f n g S k T] ‹f ∈ carrier (SA n)› ‹g ∈ carrier (SA n)› is-semialgebraic-closed*
by *blast*
show *is-semialgebraic (n + k) (partial-pullback n (fun-glue n S f g) k T)*
using *assms 0 1 2 cartesian-product-is-semialgebraic carrier-is-semialgebraic diff-is-semialgebraic intersection-is-semialg union-is-semialgebraic* **by** *presburger*
qed

lemma *fun-glue-unit*:
assumes *f ∈ carrier (SA n)*
assumes *is-semialgebraic n S*
assumes $\bigwedge x.\ x \in S \Longrightarrow f\ x \neq \mathbf{0}$
shows *fun-glue n S f* $\mathbf{1}_{SA\ n} \in$ *Units (SA n)*
proof(*rule SA-Units-memI*)
show *fun-glue n S f* $\mathbf{1}_{SA\ n} \in$ *carrier (SA n)*
using *fun-glue-closed assms SA-is-cring cring.cring-simprules(6)* **by** *blast*
show $\bigwedge x.\ x \in$ *carrier* ($Q_p{}^n$) $\Longrightarrow$ *fun-glue n S f* $\mathbf{1}_{SA\ n}\ x \neq \mathbf{0}$
proof− **fix** *x* **assume** *A*: *x ∈ carrier* ($Q_p{}^n$)
show *fun-glue n S f* $\mathbf{1}_{SA\ n}\ x \neq \mathbf{0}$
apply(*cases x ∈ S*)
using *assms SA-is-cring cring.cring-simprules(6) assms(3)[of x] fun-glueE[of f n* $\mathbf{1}_{SA\ n}$ *S x]*
apply (*metis is-semialgebraic-closed*)
using *assms SA-is-cring[of n] cring.cring-simprules(6)[of SA n] A fun-glueE'[of f n* $\mathbf{1}_{SA\ n}$ *S x] is-semialgebraic-closed[of n S]*
unfolding *SA-one[of n] function-ring-defs(4)[of n] function-one-def*
by (*metis Diff-iff function-one-eval Qp-funs-one local.one-neq-zero*)
qed
qed

definition *parametric-fun-glue* **where**
parametric-fun-glue n Xs fs = (*λx ∈ carrier* ($Q_p{}^n$). *let S* = (*THE S. S ∈ Xs ∧ x ∈ S*) *in* (*fs S x*))

lemma *parametric-fun-glue-formula*:
assumes *Xs partitions* (*carrier* ($Q_p{}^n$))
assumes *x ∈ S*
assumes *S ∈ Xs*
shows *parametric-fun-glue n Xs fs x = fs S x*
proof−
have *0*: (*THE S. S ∈ Xs ∧ x ∈ S*) = *S*
apply(*rule the-equality*)
using *assms* **apply** *blast*
using *assms* **unfolding** *is-partition-def* **by** (*metis Int-iff empty-iff disjointE*)
have *1*: *x ∈ carrier* ($Q_p{}^n$)
using *assms* **unfolding** *is-partition-def* **by** *blast*
then show *?thesis* **using** *0* **unfolding** *parametric-fun-glue-def restrict-def* **by**

metis
qed

definition *char-fun* **where**
char-fun n S = (λ *x* ∈ *carrier* (${Q_p}^n$). *if x* ∈ *S then* **1** *else* **0**)

lemma *char-fun-is-semialg*:
assumes *is-semialgebraic n S*
shows *char-fun n S* ∈ *carrier* (*SA n*)
proof−
have *char-fun n S* = *fun-glue n S* $\mathbf{1}_{SA\ n}$ $\mathbf{0}_{SA\ n}$
unfolding *char-fun-def fun-glue-def*
by (*metis* (*no-types*, *lifting*) *function-one-eval function-zero-eval SA-one SA-zero restrict-ext*)
then show *?thesis*
using *assms fun-glue-closed*
by (*metis SA-is-cring cring.cring-simprules*(*2*) *cring.cring-simprules*(*6*))
qed

lemma *SA-finsum-apply*:
assumes *finite S*
assumes *x* ∈ *carrier* (${Q_p}^n$)
shows *F* ∈ *S* → *carrier* (*SA n*) ⟶ *finsum* (*SA n*) *F S x* = (⨁ *s*∈*S*. *F s x*)
proof(*rule finite.induct*[*of S*])
show *finite S*
using *assms* **by** *blast*
show *F* ∈ {} → *carrier* (*SA n*) ⟶ *finsum* (*SA n*) *F* {} *x* = (⨁ *s*∈{}. *F s x*)
using *assms abelian-monoid.finsum-empty*[*of SA n*] *Qp.abelian-monoid-axioms SA-is-abelian-monoid*
by (*simp add*: *SA-zeroE*)
show ⋀*A a*. *finite A* ⟹
F ∈ *A* → *carrier* (*SA n*) ⟶ *finsum* (*SA n*) *F A x* = (⨁ *s*∈*A*. *F s x*) ⟹
F ∈ *insert a A* → *carrier* (*SA n*) ⟶ *finsum* (*SA n*) *F* (*insert a A*) *x* = (⨁ *s*∈*insert a A*. *F s x*)
proof− **fix** *A a* **assume** *A*: *finite A F* ∈ *A* → *carrier* (*SA n*) ⟶ *finsum* (*SA n*) *F A x* = (⨁ *s*∈*A*. *F s x*)
show *F* ∈ *insert a A* → *carrier* (*SA n*) ⟶ *finsum* (*SA n*) *F* (*insert a A*) *x* = (⨁ *s*∈*insert a A*. *F s x*)
proof assume *A'*: *F* ∈ *insert a A* → *carrier* (*SA n*)
then have *0*: *F* ∈ *A* → *carrier* (*SA n*)
by *blast*
hence *1*: *finsum* (*SA n*) *F A x* = (⨁ *s*∈*A*. *F s x*)
using *A* **by** *blast*
show *finsum* (*SA n*) *F* (*insert a A*) *x* = (⨁ *s*∈*insert a A*. *F s x*)
proof(*cases a* ∈ *A*)
case *True*
then show *?thesis*
using *1* **by** (*metis insert-absorb*)
next

case *False*
have *F00*: $(\lambda s.\ F\ s\ x) \in A \to carrier\ Q_p$
apply(*rule Pi-I*, *rule SA-car-closed*[*of - n*])
using *0 assms* **by** *auto*
have *F01*: $F\ a\ x \in carrier\ Q_p$
using A' *assms*
by (*metis* (*no-types*, *lifting*) *Qp.function-ring-car-mem-closed Pi-split-insert-domain SA-car-in-Qp-funs-car subsetD*)
have *F0*: $(\bigoplus s{\in}insert\ a\ A.\ F\ s\ x) = F\ a\ x \oplus (\bigoplus s{\in}A.\ F\ s\ x)$
using *F00 F01* A' *False A*(*1*) *Qp.finsum-insert*[*of A a* $\lambda s.\ F\ s\ x$] **by** *blast*
have *F1*: *finsum* (*SA n*) *F* (*insert a A*) = $F\ a \oplus_{SA\ n}$ *finsum* (*SA n*) *F A*
using *abelian-monoid.finsum-insert*[*of SA n A a F*]
by (*metis* (*no-types*, *lifting*) A' *A*(*1*) *False Pi-split-insert-domain SA-is-abelian-monoid assms*(*1*))
show *?thesis*
using *Qp.finsum-closed*[*of* $\lambda s.\ F\ s\ x\ A$] *abelian-monoid.finsum-closed*[*of SA n F A*]
SA-is-abelian-monoid[*of n*] *assms F0 F1 0 A*(*2*) *SA-add* **by** *presburger*
qed
qed
qed
qed

lemma *SA-finsum-apply-zero*:
assumes *finite S*
assumes $F \in S \to carrier\ (SA\ n)$
assumes $x \in carrier\ ({Q_p}^n)$
assumes $\bigwedge s.\ s \in S \Longrightarrow F\ s\ x = \mathbf{0}$
shows *finsum* (*SA n*) *F S* $x = \mathbf{0}$
proof−
have *finsum* (*SA n*) *F S* $x = (\bigoplus s{\in}S.\ F\ s\ x)$
using *SA-finsum-apply assms* **by** *blast*
then show *?thesis* **using** *assms*
by (*metis Qp.add.finprod-one-eqI*)
qed

lemma *parametric-fun-glue-is-SA*:
assumes *finite Xs*
assumes *Xs partitions* (*carrier* $({Q_p}^n)$)
assumes *fs* $\in Xs \to carrier\ (SA\ n)$
assumes $\forall\ S \in Xs.$ *is-semialgebraic n S*
shows *parametric-fun-glue n Xs fs* $\in carrier\ (SA\ n)$
proof−
obtain *F* **where** *F-def*: $F = (\lambda S.\ fs\ S \otimes_{SA\ n}$ *char-fun n S*)
by *blast*
have *0*: $F \in Xs \to carrier\ (SA\ n)$ **proof fix** *S* **assume** $S \in Xs$ **then show** $F\ S \in carrier\ (SA\ n)$
using *SA-mult-closed*[*of n fs S char-fun n S*] *char-fun-is-semialg*[*of n S*] *assms SA-car-memE*

unfolding *F-def* **by** *blast* **qed**
have *1*: $\bigwedge S\ x.\ S \in Xs \Longrightarrow x \in S \Longrightarrow F\ S\ x = fs\ S\ x$
proof− **fix** *S x* **assume** *A*: $S \in Xs\ x \in S$
then have *x-closed*: $x \in carrier\ ({Q_p}^n)$
using *assms* **unfolding** *is-partition-def* **by** *blast*
then have *0*: $F\ S\ x = fs\ S\ x \otimes char\text{-}fun\ n\ S\ x$
unfolding *F-def* **using** *SA-mult* **by** *blast*
have *1*: $char\text{-}fun\ n\ S\ x = \mathbf{1}$
using *char-fun-def A x-closed* **by** *auto*
have *2*: $fs\ S\ x \in carrier\ Q_p$
apply(*intro SA-car-closed*[*of - n*] *x-closed*)
using *assms A* **by** *auto*
show $F\ S\ x = fs\ S\ x$
unfolding *0 1* **using** *2 Qp.cring-simprules*(*12*) **by** *auto*
qed
have *2*: $\bigwedge S\ x.\ S \in Xs \Longrightarrow x \in carrier\ ({Q_p}^n) \Longrightarrow x \notin S \Longrightarrow F\ S\ x = \mathbf{0}$
proof− **fix** *S x* **assume** *A*: $S \in Xs\ x \in carrier\ ({Q_p}^n)\ x \notin S$
then have *x-closed*: $x \in carrier\ ({Q_p}^n)$
using *assms* **unfolding** *is-partition-def* **by** *blast*
hence *20*: $F\ S\ x = fs\ S\ x \otimes char\text{-}fun\ n\ S\ x$
unfolding *F-def* **using** *SA-mult* **by** *blast*
have *21*: $char\text{-}fun\ n\ S\ x = \mathbf{0}$
unfolding *char-fun-def* **using** *A x-closed* **by** *auto*
have *22*: $fs\ S\ x \in carrier\ Q_p$
apply(*intro SA-car-closed*[*of - n*] *x-closed*)
using *assms A* **by** *auto*
show $F\ S\ x = \mathbf{0}$
using *22* **unfolding** *20 21* **by** *auto*
qed
obtain *g* **where** *g-def*: $g = finsum\ (SA\ n)\ F\ Xs$
by *blast*
have *g-closed*: $g \in carrier\ (SA\ n)$
using *abelian-monoid.finsum-closed*[*of SA n F Xs*] *assms SA-is-ring 0*
unfolding *g-def ring-def abelian-group-def* **by** *blast*
have $g = parametric\text{-}fun\text{-}glue\ n\ Xs\ fs$
proof fix *x* **show** $g\ x = parametric\text{-}fun\text{-}glue\ n\ Xs\ fs\ x$
proof(*cases* $x \in carrier\ ({Q_p}^n)$)
case *True*
then obtain *S* **where** *S-def*: $S \in Xs \wedge x \in S$
using *assms is-partitionE* **by** *blast*
then have *T0*: $parametric\text{-}fun\text{-}glue\ n\ Xs\ fs\ x = F\ S\ x$
using *1 assms parametric-fun-glue-formula* **by** *blast*
have *T1*: $g\ x = F\ S\ x$
proof−
have *00*: $F\ S \oplus_{SA\ n} finsum\ (SA\ n)\ F\ (Xs - \{S\}) = finsum\ (SA\ n)\ F\ (insert\ S\ (Xs - \{S\}))$
using *abelian-monoid.finsum-insert*[*of SA n Xs* − {*S*} *S F*]
by (*metis* (*no-types*, *lifting*) *0 Diff-iff Pi-anti-mono Pi-split-insert-domain SA-is-abelian-monoid S-def Set.basic-monos*(*7*) *assms*(*1*) *finite-Diff insert-iff sub-*

```
setI)
      have T10: g  = F S ⊕_SA n finsum (SA n) F (Xs − {S})
        using S-def unfolding 00 g-def
        by (simp add: insert-absorb)
      have T11: finsum (SA n) F (Xs − {S}) ∈ carrier (SA n)
       using abelian-monoid.finsum-closed[of SA n F Xs − {S}] assms SA-is-ring
0
        unfolding g-def ring-def abelian-group-def by blast
      hence T12: g x = F S x ⊕ finsum (SA n) F (Xs − {S}) x
        using SA-add S-def T10 assms is-semialgebraic-closed by blast
      have T13: finsum (SA n) F (Xs − {S}) x = 0
        apply(rule SA-finsum-apply-zero[of Xs − {S} F n x])
            using assms apply blast
              using 0 apply blast
              using True apply blast
      proof−
        fix s assume AA: s ∈ Xs − {S}
        then have x ∉ s
          using True assms S-def is-partitionE[of Xs carrier (Q_p^n)]  disjointE[of
Xs S s]
          by blast
        then show F s x = 0
          using AA 2[of s x] True by blast
      qed
      have T14: F S x ∈ carrier Q_p
     using assms True S-def by (metis (no-types, lifting) 0 Qp.function-ring-car-mem-closed
PiE SA-car-memE(2))
    then show ?thesis using T12 T13 assms True Qp.add.l-cancel-one Qp.zero-closed
by presburger
    qed
    show ?thesis using T0 T1 by blast
  next
    case False
    then show ?thesis
      using g-closed unfolding parametric-fun-glue-def
       by (metis (mono-tags, lifting) function-ring-not-car SA-car-memE(2) re-
strict-def)
  qed
 qed
 then show ?thesis using g-closed by blast
qed
```

14.10.2 Turning Functions into Units Via Gluing

By gluing a function to the multiplicative unit on its zero set, we can get a canonical choice of local multiplicative inverse of a function f. Denef's proof frequently reasons about functions of the form $\frac{f(x)}{g(x)}$ with the tacit understanding that they are meant to be defined on the largest domain of definition possible. This technical tool allows us to replicate this kind of

reasoning in our formal proofs.

definition *to-fun-unit* **where**
to-fun-unit n f = *fun-glue n* {$x \in$ *carrier* ($Q_p{}^n$). *f x* $\neq$ **0**} *f* $\mathbf{1}_{SA\ n}$

lemma *to-fun-unit-is-unit*:
assumes *f* $\in$ *carrier* (*SA n*)
shows *to-fun-unit n f* $\in$ *Units* (*SA n*)
unfolding *to-fun-unit-def*
apply(*rule fun-glue-unit*)
apply (*simp add*: *assms*)
using *assms nonzero-evimage-closed*[*of f*] **apply** *blast*
by *blast*

lemma *to-fun-unit-closed*:
assumes *f* $\in$ *carrier* (*SA n*)
shows *to-fun-unit n f* $\in$ *carrier* (*SA n*)
using *assms to-fun-unit-is-unit SA-is-ring SA-Units-closed* **by** *blast*

lemma *to-fun-unit-eq*:
assumes *f* $\in$ *carrier* (*SA n*)
assumes *x* $\in$ *carrier* ($Q_p{}^n$)
assumes *f x* $\neq$ **0**
shows *to-fun-unit n f x* = *f x*
unfolding *to-fun-unit-def fun-glue-def* **using** *assms*
by *simp*

lemma *to-fun-unit-eq′*:
assumes *f* $\in$ *carrier* (*SA n*)
assumes *x* $\in$ *carrier* ($Q_p{}^n$)
assumes *f x* = **0**
shows *to-fun-unit n f x* = **1**
unfolding *to-fun-unit-def fun-glue-def* **using** *assms*
by (*simp add*: *SA-oneE*)

definition *one-over-fun* **where**
one-over-fun n f = $inv_{SA\ n}$ (*to-fun-unit n f*)

lemma *one-over-fun-closed*:
assumes *f* $\in$ *carrier* (*SA n*)
shows *one-over-fun n f* $\in$ *carrier* (*SA n*)
using *assms SA-is-ring*[*of n*] *to-fun-unit-is-unit*[*of f n*]
by (*metis SA-Units-closed one-over-fun-def ring.Units-inverse*)

lemma *one-over-fun-eq*:
assumes *f* $\in$ *carrier* (*SA n*)
assumes *x* $\in$ *carrier* ($Q_p{}^n$)
assumes *f x* $\neq$ **0**
shows *one-over-fun n f x* = *inv* (*f x*)
using *assms to-fun-unit-eq* **unfolding** *one-over-fun-def*

using *Qp-funs-m-inv SA-Units-Qp-funs-Units SA-Units-Qp-funs-inv to-fun-unit-is-unit*
by *presburger*

lemma *one-over-fun-smult-eval*:
assumes $f \in carrier\ (SA\ n)$
assumes $a \neq \mathbf{0}$
assumes $a \in carrier\ Q_p$
assumes $x \in carrier\ ({Q_p}^n)$
assumes $f\ x \neq \mathbf{0}$
shows $one\text{-}over\text{-}fun\ n\ (a \odot_{SA\ n} f)\ x = inv\ (a \otimes (f\ x))$
using *one-over-fun-eq*[*of a* $\odot_{SA\ n}$ *f n x*] *assms*
by (*metis Qp.function-ring-car-memE Qp.integral SA-car-memE*(*2*) *SA-smult-closed SA-smult-formula*)

lemma *one-over-fun-smult-eval'*:
assumes $f \in carrier\ (SA\ n)$
assumes $a \neq \mathbf{0}$
assumes $a \in carrier\ Q_p$
assumes $x \in carrier\ ({Q_p}^n)$
assumes $f\ x \neq \mathbf{0}$
shows $one\text{-}over\text{-}fun\ n\ (a \odot_{SA\ n} f)\ x = inv\ a \otimes inv\ (f\ x)$
proof−
have *0*: $one\text{-}over\text{-}fun\ n\ (a \odot_{SA\ n} f)\ x = inv\ (a \otimes f\ x)$
using *assms one-over-fun-smult-eval*[*of f n a x*]
by *fastforce*
have *1*: $f\ x \in nonzero\ Q_p$
by(*intro nonzero-memI SA-car-closed*[*of - n*] *assms*)
show *?thesis*
unfolding *0* **using** *1 assms*
using *Qp.comm-inv-char Qp.cring-simprules*(*11*) *Qp.cring-simprules*(*5*) *SA-car-closed field-inv*(*2*) *field-inv*(*3*) *local.fract-cancel-right* **by** *presburger*
qed

lemma *SA-add-pow-closed*:
assumes $f \in carrier\ (SA\ n)$
shows $([(k::nat)] \cdot_{SA\ n} f) \in carrier\ (SA\ n)$
using *assms SA-is-ring*[*of n*]
by (*meson ring.nat-mult-closed*)

lemma *one-over-fun-add-pow-eval*:
assumes $f \in carrier\ (SA\ n)$
assumes $x \in carrier\ ({Q_p}^n)$
assumes $f\ x \neq \mathbf{0}$
assumes $(k::nat) > 0$
shows $one\text{-}over\text{-}fun\ n\ ([k] \cdot_{SA\ n} f)\ x = inv\ ([k] \cdot f\ x)$
proof−
have *0*: $([k] \cdot_{SA\ n} f)\ x = [k] \cdot f\ x$

using *assms SA-add-pow-apply*[*of f n x k*] **by** *linarith*
hence *1*: $([k] \cdot_{SA\ n} f)\ x \neq \mathbf{0}$
using *assms Qp-char-0'' Qp.function-ring-car-mem-closed SA-car-memE*(*2*)
by *metis*
have *2*: *one-over-fun n* $([k] \cdot_{SA\ n} f)\ x = inv\ ([k] \cdot_{SA\ n} f)\ x$
using *assms one-over-fun-eq*[*of* $[k] \cdot_{SA\ n} f$ *n x*] *1 SA-add-pow-closed* **by** *blast*
thus *?thesis* **using** *1 0* **by** *presburger*
qed

lemma *one-over-fun-pow-closed*:
assumes $f \in carrier\ (SA\ n)$
shows *one-over-fun n* $(f[\hat{}]_{SA\ n}(k::nat)) \in carrier\ (SA\ n)$
using *assms*
by (*meson SA-nat-pow-closed one-over-fun-closed padic-fields.SA-imp-semialg padic-fields-axioms*)

lemma *one-over-fun-pow-eval*:
assumes $f \in carrier\ (SA\ n)$
assumes $x \in carrier\ ({Q_p}^n)$
assumes $f\ x \neq \mathbf{0}$
shows *one-over-fun n* $(f[\hat{}]_{SA\ n}(k::nat))\ x = inv\ ((f\ x)\ [\hat{}]\ k)$
using *one-over-fun-eq*[*of* $f[\hat{}]_{SA\ n} k$ *n x*] *assms*
by (*metis Qp.function-ring-car-memE Qp.nonzero-pow-nonzero SA-car-memE*(*2*) *SA-nat-pow SA-nat-pow-closed padic-fields.SA-car-memE*(*1*) *padic-fields-axioms*)

14.11 Inclusions of Lower Dimensional Function Rings

definition *fun-inc* **where**
fun-inc m n f $= (\lambda\ x \in carrier\ ({Q_p}^m).\ f\ (take\ n\ x))$

lemma *fun-inc-closed*:
assumes $f \in carrier\ (SA\ n)$
assumes $m \geq n$
shows *fun-inc m n f* $\in carrier\ (SA\ m)$
proof−
have *0*: $\bigwedge x.\ x \in carrier\ ({Q_p}^m) \Longrightarrow$ *fun-inc m n f x* $= (f \circ take\ n)\ x$
unfolding *fun-inc-def* **by** (*metis comp-apply restrict-apply'*)
have *1*: *is-semialg-function m* $(f \circ take\ n)$
using *assms comp-take-is-semialg*
by (*metis SA-imp-semialg le-neq-implies-less padic-fields.semialg-function-comp-closed padic-fields-axioms take-is-semialg-map*)
have *2*: *is-semialg-function m* (*fun-inc m n f*)
using *0 1 semialg-function-on-carrier'* **by** *blast*
show *?thesis* **apply**(*rule SA-car-memI*) **apply**(*rule Qp.function-ring-car-memI*)
using *2 is-semialg-function-closed* **apply** *blast*
using *fun-inc-def*[*of m n f*] **unfolding** *restrict-def* **apply** *presburger*
using *2* **by** *blast*
qed

lemma *fun-inc-eval*:
assumes $x \in$ *carrier* $({Q_p}^m)$
shows *fun-inc m n f x* = *f* (*take n x*)
unfolding *fun-inc-def* **using** *assms*
by (*meson restrict-apply'*)

lemma *ord-congruence-set-univ-semialg-fixed*:
assumes $n \geq 0$
shows *is-univ-semialgebraic* (*ord-congruence-set n a*)
using *ord-congruence-set-univ-semialg assms*
by *auto*

lemma *ord-congruence-set-SA-function*:
assumes $n \geq 0$
assumes $c \in$ *carrier* (*SA* (*m*+*l*))
shows *is-semialgebraic* (*m*+*l*) {$x \in$ *carrier* $({Q_p}^{m+l})$. *c x* ∈ *nonzero* $Q_p \wedge$ *ord* (*c x*) *mod n* = *a*}
proof−
have *0*: {$x \in$ *carrier* $({Q_p}^{m+l})$. *c x* ∈ *nonzero* $Q_p \wedge$ *ord* (*c x*) *mod n* = *a*} = *c* $^{-1}{}_{m+l}$ (*ord-congruence-set n a*)
unfolding *ord-congruence-set-def evimage-def* **using** *assms* **by** *blast*
show *?thesis* **unfolding** *0* **apply**(*rule evimage-is-semialg*)
using *assms* **apply** *blast* **using** *assms ord-congruence-set-univ-semialg-fixed*[*of n a*]
by *blast*
qed

lemma *ac-cong-set-SA*:
assumes $n > 0$
assumes $k \in$ *Units* (*Zp-res-ring n*)
assumes $c \in$ *carrier* (*SA* (*m*+*l*))
shows *is-semialgebraic* (*m*+*l*) {$x \in$ *carrier* $({Q_p}^{m+l})$. *c x* ∈ *nonzero* $Q_p \wedge$ *ac n* (*c x*) = *k*}
proof−
have {$x \in$ *carrier* $({Q_p}^{m+l})$. *c x* ∈ *nonzero* $Q_p \wedge$ *ac n* (*c x*) = *k*}= (*c* $^{-1}{}_{m + l}$ *ac-cong-set n k*)
unfolding *ac-cong-set-def evimage-def nonzero-def mem-Collect-eq* **using** *assms* **by** *blast*
thus *?thesis*
using *assms ac-cong-set-is-univ-semialg*[*of n k*] *evimage-is-semialg*[*of c m*+*l ac-cong-set n k*]
by *presburger*
qed

lemma *ac-cong-set-SA'*:
assumes $n > 0$
assumes $k \in$ *Units* (*Zp-res-ring n*)
assumes $c \in$ *carrier* (*SA m*)
shows *is-semialgebraic m* {$x \in$ *carrier* $({Q_p}^m)$. *c x* ∈ *nonzero* $Q_p \wedge$ *ac n* (*c x*)

$= k\}$
using *assms ac-cong-set-SA*[*of n k c m 0*] **unfolding** *Nat.add-0-right* **by** *blast*

lemma *ac-cong-set-SA''*:
assumes $n > 0$
assumes $m > 0$
assumes $k \in$ *Units* (*Zp-res-ring n*)
assumes $c \in$ *carrier* (*SA m*)
assumes $\bigwedge x.\ x \in$ *carrier* $({Q_p}^m) \Longrightarrow c\ x \neq \mathbf{0}$
shows *is-semialgebraic m* $\{x \in$ *carrier* $({Q_p}^m).$ *ac n* $(c\ x) = k\}$
proof−
have $\{x \in$ *carrier* $({Q_p}^m).$ $c\ x \in$ *nonzero* $Q_p \wedge$ *ac n* $(c\ x) = k\} = \{x \in$ *carrier* $({Q_p}^m).$ *ac n* $(c\ x) = k\}$
apply(*rule subset-antisym*) **apply** *blast*
apply(*rule subsetI*) **using** *assms* **unfolding** *nonzero-def mem-Collect-eq*
using *Qp.function-ring-car-memE SA-car-memE*(*2*) **by** *blast*
thus *?thesis* **using** *assms ac-cong-set-SA'*[*of n k c m*] **by** *metis*
qed

14.12 Miscellaneous

lemma *nth-pow-wits-SA-fun-prep*:
assumes $n > 0$
assumes $h \in$ *carrier* (*SA m*)
assumes $\varrho \in$ *nth-pow-wits n*
shows *is-semialgebraic m* $(h\ {}^{-1}{}_m$*pow-res n* $\varrho)$
by(*intro evimage-is-semialg assms pow-res-is-univ-semialgebraic nth-pow-wits-closed*(*1*)[*of n*])

definition *kth-rt* **where**
kth-rt m (*k*::*nat*) *f x* = (*if x* $\in$ *carrier* $({Q_p}^m)$ *then* (*THE b.* $b \in$ *carrier* $Q_p \wedge b[\hat{}]k = (f\ x) \wedge$ *ac* (*nat* (*ord* $([k]\cdot\mathbf{1})) + 1)\ b = 1$)
else undefined)

Normalizing a semialgebraic function to have a constant angular component

lemma *ac-res-Unit-inc*:
assumes $n > 0$
assumes $t \in$ *Units* (*Zp-res-ring n*)
shows *ac n* $([t]\cdot\mathbf{1}) = t$
proof−
have *0*: $[t]\cdot\mathbf{1} \neq \mathbf{0}$
using *assms* **by** (*metis Qp-char-0-int less-one less-or-eq-imp-le nat-neq-iff zero-not-in-residue-units*)
have *1*: $[t]\cdot\mathbf{1} \in \mathcal{O}_p$
by (*metis Zp.one-closed Zp-int-mult-closed image-eqI inc-of-int*)
hence *2*: *angular-component* $([t]\cdot\mathbf{1}) =$ *ac-Zp* $([t]\cdot_{Z_p}\mathbf{1}_{Z_p})$
using *angular-component-of-inclusion*[*of* $[t]\cdot_{Z_p}\mathbf{1}_{Z_p}$]
by (*metis 0 Qp.int-inc-zero Zp.int-inc-zero Zp-int-inc-closed inc-of-int not-nonzero-Qp*)
hence *3*: *ac n* $([t]\cdot\mathbf{1}) =$ *ac-Zp* $([t]\cdot_{Z_p}\mathbf{1}_{Z_p})\ n$

unfolding *ac-def* **using** *0* **by** *presburger*
hence *val-Zp* ($[t]\cdot_{Z_p}\mathbf{1}_{Z_p}$) = *0*
proof−
have *coprime p t*
using *assms*
by (*metis coprime-commute less-one less-or-eq-imp-le nat-neq-iff padic-integers.residue-UnitsE padic-integers-axioms*)
then show *?thesis*
by (*metis Zp-int-inc-closed Zp-int-inc-res coprime-mod-right-iff coprime-power-right-iff mod-by-0 order-refl p-residues residues.m-gt-one residues.mod-in-res-units val-Zp-0-criterion val-Zp-p val-Zp-p-int-unit zero-less-one zero-neq-one-class.one-neq-zero zero-not-in-residue-units*)
qed
hence *4*: $[t]\cdot_{Z_p}\mathbf{1}_{Z_p} \in$ *Units* Z_p
using *val-Zp-0-imp-unit* **by** *blast*
hence *5*: *ac-Zp* ($[t]\cdot_{Z_p}\mathbf{1}_{Z_p}$) = $[t]\cdot_{Z_p}\mathbf{1}_{Z_p}$ **using**
ac-Zp-of-Unit ‹*val-Zp* ($[t]\cdot_{Z_p}\mathbf{1}_{Z_p}$) = *0*› **by** *blast*
have *6*: *ac-Zp* ($[t]\cdot_{Z_p}\mathbf{1}_{Z_p}$) *n* = *t*
proof−
have *t* ∈ *carrier* (*Zp-res-ring n*)
using *assms monoid.Units-closed*[*of Zp-res-ring n t*] *cring-def padic-integers.R-cring padic-integers-axioms ring-def* **by** *blast*
hence $t < p\hat{}n \wedge t \geq 0$
using *p-residue-ring-car-memE* **by** *auto*
thus *?thesis*
unfolding *5* **unfolding** *Zp-int-inc-rep p-residue-def residue-def* **by** *auto*
qed
show *?thesis*
unfolding *3* **using** *6* **by** *blast*
qed

lemma *val-of-res-Unit*:
assumes *n* > *0*
assumes *t* ∈ *Units* (*Zp-res-ring n*)
shows *val* ($[t]\cdot\mathbf{1}$) = *0*
proof−
have *0*: $[t]\cdot\mathbf{1} \neq \mathbf{0}$
using *assms* **by** (*metis Qp-char-0-int less-one less-or-eq-imp-le nat-neq-iff zero-not-in-residue-units*)
have *1*: $[t]\cdot\mathbf{1} \in \mathcal{O}_p$
by (*metis Zp.one-closed Zp-int-mult-closed image-eqI inc-of-int*)
hence *2*: *angular-component* ($[t]\cdot\mathbf{1}$) = *ac-Zp* ($[t]\cdot_{Z_p}\mathbf{1}_{Z_p}$)
using *angular-component-of-inclusion*[*of* $[t]\cdot_{Z_p}\mathbf{1}_{Z_p}$]
by (*metis 0 Qp.int-inc-zero Zp.int-inc-zero Zp-int-inc-closed inc-of-int not-nonzero-Qp*)
hence *3*: *ac n* ($[t]\cdot\mathbf{1}$) = *ac-Zp* ($[t]\cdot_{Z_p}\mathbf{1}_{Z_p}$) *n*
unfolding *ac-def* **using** *0* **by** *presburger*
hence *val-Zp* ($[t]\cdot_{Z_p}\mathbf{1}_{Z_p}$) = *0*
proof−
have *coprime p t*
using *assms*

by (*metis coprime-commute less-one less-or-eq-imp-le nat-neq-iff padic-integers.residue-UnitsE padic-integers-axioms*)
then show *?thesis*
by (*metis Zp-int-inc-closed Zp-int-inc-res coprime-mod-right-iff coprime-power-right-iff mod-by-0 order-refl p-residues residues.m-gt-one residues.mod-in-res-units val-Zp-0-criterion val-Zp-p val-Zp-p-int-unit zero-less-one zero-neq-one-class.one-neq-zero zero-not-in-residue-units*)
qed
then show *?thesis* **using** *assms*
by (*metis Zp-int-inc-closed inc-of-int val-of-inc*)
qed

lemma(**in** *padic-integers*) *res-map-is-hom*:
assumes $N > 0$
shows *ring-hom-ring Zp* (*Zp-res-ring N*) (λ *x*. *x N*)
apply(*rule ring-hom-ringI*)
apply (*simp add*: *R.ring-axioms*)
using *assms cring.axioms*(*1*) *local.R-cring* **apply** *blast*
using *residue-closed* **apply** *blast*
using *residue-of-prod* **apply** *blast*
using *residue-of-sum* **apply** *blast*
using *assms residue-of-one*(*1*) **by** *blast*

lemma *ac-of-fraction*:
assumes $N > 0$
assumes $a \in$ *nonzero* Q_p
assumes $b \in$ *nonzero* Q_p
shows *ac N* ($a \div b$) = *ac N a* $\otimes_{Zp\text{-}res\text{-}ring\ N}$ *inv* $_{Zp\text{-}res\text{-}ring\ N}$ *ac N b*
using *ac-mult*[*of a inv b N*] *ac-inv assms Qp.nonzero-closed nonzero-inverse-Qp*
by *presburger*

lemma *pow-res-eq-rel*:
assumes $n > 0$
assumes $b \in$ *carrier* Q_p
shows {$x \in$ *carrier* Q_p. *pow-res n x* = *pow-res n b*} = *pow-res n b*
apply(*rule equalityI'*, *unfold mem-Collect-eq*, *metis pow-res-refl*,
intro conjI)
using *pow-res-def* **apply** *auto*[*1*]
apply(*rule equal-pow-resI*)
using *pow-res-def* **apply** *auto*[*1*]
using *pow-res-refl assms* **by** (*metis equal-pow-resI*)

lemma *pow-res-is-univ-semialgebraic'*:
assumes $n > 0$
assumes $b \in$ *carrier* Q_p
shows *is-univ-semialgebraic* {$x \in$ *carrier* Q_p. *pow-res n x* = *pow-res n b*}
using *assms pow-res-eq-rel pow-res-is-univ-semialgebraic* **by** *presburger*

lemma *evimage-eqI*:

assumes $c \in carrier\ (SA\ n)$
shows $c\ ^{-1}n\ \{x \in carrier\ Q_p.\ P\ x\} = \{x \in carrier\ ({Q_p}^n).\ P\ (c\ x)\}$
by(*rule equalityI'*, *unfold evimage-def mem-Collect-eq Int-iff*, *intro conjI*, *auto* , *rule SA-car-closed*[*of* - *n*], *auto simp*: *assms*)

lemma *SA-pow-res-is-semialgebraic*:
assumes $n > 0$
assumes $b \in carrier\ Q_p$
assumes $c \in carrier\ (SA\ N)$
shows *is-semialgebraic N* $\{x \in carrier\ ({Q_p}^N).\ pow\text{-}res\ n\ (c\ x) = pow\text{-}res\ n\ b\}$
proof−
have $c\ ^{-1}N\ \{x \in carrier\ Q_p.\ pow\text{-}res\ n\ x = pow\text{-}res\ n\ b\} = \{x \in carrier\ ({Q_p}^N).\ pow\text{-}res\ n\ (c\ x) = pow\text{-}res\ n\ b\}$
apply(*rule evimage-eqI*) **using** *assms* **by** *blast*
thus *?thesis*
using *pow-res-is-univ-semialgebraic' evimage-is-semialg assms*
by (*metis* (*no-types*, *lifting*))
qed

lemma *eint-diff-imp-eint*:
assumes $a \in nonzero\ Q_p$
assumes $b \in carrier\ Q_p$
assumes $val\ a = val\ b + eint\ i$
shows $b \in nonzero\ Q_p$
using *assms val-zero*
by (*metis Qp.nonzero-closed Qp.not-nonzero-memE not-eint-eq plus-eint-simps*(*2*) *val-ord'*)

lemma *SA-minus-eval*:
assumes $f \in carrier\ (SA\ n)$
assumes $g \in carrier\ (SA\ n)$
assumes $x \in carrier\ ({Q_p}^n)$
shows $(f\ \ominus_{SA\ n}\ g)\ x = f\ x \ominus g\ x$
using *assms* **unfolding** *a-minus-def*
using *SA-a-inv-eval SA-add* **by** *metis*

lemma *Qp-cong-set-evimage*:
assumes $f \in carrier\ (SA\ n)$
assumes $a \in carrier\ Z_p$
shows *is-semialgebraic n* $(f\ ^{-1}n\ (Qp\text{-}cong\text{-}set\ \alpha\ a))$
using *assms Qp-cong-set-is-univ-semialgebraic evimage-is-semialg* **by** *blast*

lemma *SA-constant-res-set-semialg*:
assumes $l \in carrier\ (Zp\text{-}res\text{-}ring\ n)$
assumes $f \in carrier\ (SA\ m)$
shows *is-semialgebraic m* $\{x \in carrier\ ({Q_p}^m).\ f\ x \in \mathcal{O}_p \wedge Qp\text{-}res\ (f\ x)\ n = l\}$
proof−
have *0*: $\{x \in carrier\ ({Q_p}^m).\ f\ x \in \mathcal{O}_p \wedge Qp\text{-}res\ (f\ x)\ n = l\} = f\ ^{-1}m\ \{x \in \mathcal{O}_p.\ Qp\text{-}res\ x\ n = l\}$

unfolding *evimage-def* **by** *blast*
show *?thesis* **unfolding** *0*
by(*rule evimage-is-semialg*, *rule assms*, *rule constant-res-set-semialg*, *rule assms*)
qed

lemma *val-ring-cong-set*:
assumes $f \in$ *carrier* $(SA\ k)$
assumes $\bigwedge x.\ x \in$ *carrier* $({Q_p}^k) \Longrightarrow f\ x \in \mathcal{O}_p$
assumes $t \in$ *carrier* $(Zp\text{-}res\text{-}ring\ n)$
shows *is-semialgebraic* k $\{x \in$ *carrier* $({Q_p}^k).$ *to-Zp* $(f\ x)\ n = t\}$
proof−
have *0*: $[t] \cdot_{Z_p} \mathbf{1}_{Z_p} \in$ *carrier* Z_p
by *blast*
have *1*: $([t] \cdot_{Z_p} \mathbf{1}_{Z_p})\ n = t$
using *assms*
unfolding *Zp-int-inc-rep p-residue-def residue-def residue-ring-def* **by** *simp*
have $\{x \in$ *carrier* $({Q_p}^k).$ *to-Zp* $(f\ x)\ n = t\} = f\ {}^{-1}{}_k\ \{x \in \mathcal{O}_p.\ (to\text{-}Zp\ x)\ n = t\}$
unfolding *evimage-def* **using** *assms* **by** *auto*
then show *?thesis* **using** *0 1 assms Qp-cong-set-evimage*[*of f k* $[t] \cdot_{Z_p} \mathbf{1}_{Z_p}$ *n*]
unfolding *Qp-cong-set-def*
by *presburger*
qed

lemma *val-ring-pullback-SA*:
assumes $N > 0$
assumes $c \in$ *carrier* $(SA\ N)$
shows *is-semialgebraic* N $\{x \in$ *carrier* $({Q_p}^N).\ c\ x \in \mathcal{O}_p\}$
proof−
have *0*: $\{x \in$ *carrier* $({Q_p}^N).\ c\ x \in \mathcal{O}_p\} = c\ {}^{-1}{}_N\ \mathcal{O}_p$
unfolding *evimage-def* **by** *blast*
have *1*: *is-univ-semialgebraic* $\mathcal{O}_p$
using *Qp-val-ring-is-univ-semialgebraic* **by** *blast*
show *?thesis* **using** *assms 0 1 evimage-is-semialg* **by** *presburger*
qed

lemma(**in** *padic-fields*) *res-eq-set-is-semialg*:
assumes $k > 0$
assumes $c \in$ *carrier* $(SA\ k)$
assumes $s \in$ *carrier* $(Zp\text{-}res\text{-}ring\ n)$
shows *is-semialgebraic* k $\{x \in$ *carrier* $({Q_p}^k).\ c\ x \in \mathcal{O}_p \wedge$ *to-Zp* $(c\ x)\ n = s\}$
proof−
obtain a **where** *a-def*: $a = [s] \cdot \mathbf{1}$
by *blast*
have *0*: $a \in \mathcal{O}_p$
using *a-def*
by (*metis Zp.one-closed Zp-int-mult-closed image-iff inc-of-int*)
have *1*: *to-Zp* $a = [s] \cdot_{Z_p} \mathbf{1}_{Z_p}$
using *0* **unfolding** *a-def*
by (*metis* Q_p*-def Zp-def Zp-int-inc-closed* ι*-def inc-to-Zp padic-fields.inc-of-int*

padic-fields-axioms)
have *2*: $([s]\cdot_{Z_p} \mathbf{1}_{Z_p})\ n = s$
using *assms*
by (*metis Zp-int-inc-res mod-pos-pos-trivial p-residue-ring-car-memE*(*1*) *p-residue-ring-car-memE*(*2*))
have *3*: $\{x \in carrier\ ({Q_p}^k).\ c\ x \in \mathcal{O}_p \wedge to\text{-}Zp\ (c\ x)\ n = s\} = c\ {}^{-1}{}_k\ B_n[a]$
proof
show $\{x \in carrier\ ({Q_p}^k).\ c\ x \in \mathcal{O}_p \wedge to\text{-}Zp\ (c\ x)\ n = s\} \subseteq c\ {}^{-1}{}_k\ B_{int\ n}[a]$
proof fix *x* **assume** *A*: $x \in \{x \in carrier\ ({Q_p}^k).\ c\ x \in \mathcal{O}_p \wedge to\text{-}Zp\ (c\ x)\ n = s\}$
then have *30*: $x \in carrier\ ({Q_p}^k)$ **by** *blast*
have *31*: $c\ x \in \mathcal{O}_p$ **using** *A* **by** *blast*
have *32*: $to\text{-}Zp\ (c\ x)\ n = s$ **using** *A* **by** *blast*
have *33*: $to\text{-}Zp\ (c\ x) \in carrier\ Z_p$
using *31 val-ring-memE to-Zp-closed* **by** *blast*
have *34*: $to\text{-}Zp\ (c\ x)\ n = (to\text{-}Zp\ a)\ n$
using *1 2 32* **by** *presburger*
hence $((to\text{-}Zp\ (c\ x)) \ominus_{Z_p} (to\text{-}Zp\ a))\ n = 0$
using *1 33 Zp-int-inc-closed res-diff-zero-fact''* **by** *presburger*
hence *35*: $val\text{-}Zp\ ((to\text{-}Zp\ (c\ x)) \ominus_{Z_p} (to\text{-}Zp\ a)) \geq n$
using *1 33 34 Zp.one-closed Zp-int-mult-closed val-Zp-dist-def val-Zp-dist-res-eq2*
by *presburger*
have *36*: $val\ (c\ x \ominus a) = val\text{-}Zp\ ((to\text{-}Zp\ (c\ x)) \ominus_{Z_p} (to\text{-}Zp\ a))$
using *31 0*
by (*metis to-Zp-minus to-Zp-val val-ring-minus-closed*)
hence $val\ (c\ x \ominus a) \geq n$
using *35* **by** *presburger*
hence $c\ x \in B_{int\ n}[a]$
using *31 c-ballI val-ring-memE* **by** *blast*
thus $x \in c\ {}^{-1}{}_k\ B_{int\ n}[a]$
using *30* **by** *blast*
qed
show $c\ {}^{-1}{}_k\ B_{int\ n}[a] \subseteq \{x \in carrier\ ({Q_p}^k).\ c\ x \in \mathcal{O}_p \wedge to\text{-}Zp\ (c\ x)\ n = s\}$
proof fix *x* **assume** *A*: $x \in c\ {}^{-1}{}_k\ B_{int\ n}[a]$
have *x-closed*: $x \in carrier\ ({Q_p}^k)$
using *A* **by** (*meson evimage-eq*)
have *00*: $val\ (c\ x \ominus a) \geq n$
using *A c-ballE*(*2*) **by** *blast*
have *cx-closed*: $c\ x \in carrier\ Q_p$
using *x-closed assms function-ring-car-closed SA-car-memE*(*2*) **by** *blast*
hence *11*: $c\ x \ominus a \in \mathcal{O}_p$
proof−
have $(0::eint) \leq n$
by (*metis eint-ord-simps*(*1*) *of-nat-0-le-iff zero-eint-def*)
thus *?thesis* **using** *00 order-trans*[*of 0::eint n*] *Qp-val-ringI*
by (*meson 0 Qp.minus-closed val-ring-memE cx-closed*)
qed
hence *22*: $c\ x \in \mathcal{O}_p$
proof−
have *00*: $c\ x = (c\ x \ominus a) \oplus a$

```
      using cx-closed 0
      by (metis 11 Qp.add.inv-solve-right' Qp.minus-eq val-ring-memE(2))
    have 01: (c x ⊖ a) ⊕ a ∈ 𝒪_p
      by(intro val-ring-add-closed 0 11)
    then show ?thesis
      using 0 11 image-iff 00 by auto
  qed
  have 33: val (c x ⊖ a) = val-Zp (to-Zp (c x) ⊖_{Z_p} to-Zp a)
    using 11 22 0
    by (metis to-Zp-minus to-Zp-val)
  have 44: val-Zp (to-Zp (c x) ⊖_{Z_p} to-Zp a) ≥ n
    using 33 00 by presburger
  have tzpcx: to-Zp (c x) ∈ carrier Z_p
    using 22 by (metis image-iff inc-to-Zp)
  have tzpa: to-Zp a ∈ carrier Z_p
    using 0 val-ring-memE to-Zp-closed by blast
  have 55: (to-Zp (c x) ⊖_{Z_p} to-Zp a) n = 0
    using 44 tzpcx tzpa Zp.minus-closed zero-below-val-Zp by blast
  hence 66: to-Zp (c x) n = s
    using 0 1 2 tzpa tzpcx
    by (metis res-diff-zero-fact(1))
  then show x ∈ {x ∈ carrier ((Q_p)^k). c x ∈ 𝒪_p ∧ to-Zp (c x) n = s}
    using 22 x-closed by blast
 qed
qed
thus ?thesis
 using evimage-is-semialg[of c k] 0 val-ring-memE assms(2) ball-is-univ-semialgebraic
by presburger
qed

lemma SA-constant-res-set-semialg':
 assumes f ∈ carrier (SA m)
 assumes C ∈ Qp-res-classes n
 shows is-semialgebraic m (f ^{-1}_m C)
proof−
 obtain l where l-def: l ∈ carrier (Zp-res-ring n) ∧ C = Qp-res-class n ([l]·1)
  using Qp-res-classes-wits assms by blast
 have C-eq: C = Qp-res-class n ([l]·1)
  using l-def by blast
 have 0: Qp-res ([l] · 1) n = l
  using l-def
 by (metis Qp-res-int-inc mod-pos-pos-trivial p-residue-ring-car-memE(1) p-residue-ring-car-memE(2))
 have 1: f ^{-1}_m C = {x ∈ carrier ((Q_p)^m). f x ∈ 𝒪_p ∧ Qp-res (f x) n = l}
  apply(rule equalityI')
  unfolding evimage-def C-eq Qp-res-class-def mem-Collect-eq unfolding 0 ap-
ply blast
  by blast
 show ?thesis
  unfolding 1 apply(rule SA-constant-res-set-semialg )
```

using *l-def* **apply** *blast* **by**(*rule assms*)
qed

14.13 Semialgebraic Polynomials

lemma *UP-SA-n-is-ring*:
shows *ring* (*UP* (*SA n*))
using *SA-is-ring*
by (*simp add*: *UP-ring.UP-ring UP-ring.intro*)

lemma *UP-SA-n-is-cring*:
shows *cring* (*UP* (*SA n*))
using *SA-is-cring*
by (*simp add*: *UP-cring.UP-cring UP-cring.intro*)

The evaluation homomorphism from `Qp_funs` to `Qp`

definition *eval-hom* **where**
eval-hom a = (λ*f*. *f a*)

lemma *eval-hom-is-hom*:
assumes *a* ∈ *carrier* (*Q*$_p$^*n*)
shows *ring-hom-ring* (*Fun*$_n$ *Q*$_p$) *Q*$_p$ (*eval-hom a*)
apply(*rule ring-hom-ringI*)
using *Qp-funs-is-cring cring.axioms*(*1*) **apply** *blast*
apply (*simp add*: *Qp.ring-axioms*)
apply (*simp add*: *Qp.function-ring-car-mem-closed assms eval-hom-def*)
apply (*metis Qp-funs-mult' assms eval-hom-def*)
apply (*metis Qp-funs-add' assms eval-hom-def*)
by (*metis function-one-eval assms eval-hom-def*)

the homomorphism from `Fun n Qp [x]` to `Qp [x]` induced by evaluation of coefficients

definition *Qp-fpoly-to-Qp-poly* **where**
Qp-fpoly-to-Qp-poly n a = *poly-lift-hom* (*Fun*$_n$ *Q*$_p$) *Q*$_p$ (*eval-hom a*)

lemma *Qp-fpoly-to-Qp-poly-is-hom*:
assumes *a* ∈ *carrier* (*Q*$_p$^*n*)
shows (*Qp-fpoly-to-Qp-poly n a*) ∈ *ring-hom* (*UP* (*Fun*$_n$ *Q*$_p$)) (*Q*$_p$-*x*)
unfolding *Qp-fpoly-to-Qp-poly-def*
apply(*rule UP-cring.poly-lift-hom-is-hom*)
unfolding *UP-cring-def*
apply (*simp add*: *Qp-funs-is-cring*)
apply (*simp add*: *UPQ.R-cring*)
using *assms eval-hom-is-hom*[*of a*] *ring-hom-ring.homh* **by** *blast*

lemma *Qp-fpoly-to-Qp-poly-extends-apply*:
assumes *a* ∈ *carrier* (*Q*$_p$^*n*)
assumes *f* ∈ *carrier* (*Fun*$_n$ *Q*$_p$)

shows *Qp-fpoly-to-Qp-poly n a* (*to-polynomial* (*Fun*$_n$ Q_p) *f*) = *to-polynomial* Q_p (*f a*)
unfolding *Qp-fpoly-to-Qp-poly-def*
using *assms eval-hom-is-hom*[*of a*] *UP-cring.poly-lift-hom-extends-hom*[*of Fun*$_n$ Q_p Q_p *eval-hom a*]
Qp.function-ring-car-memE[*of f n*] *ring-hom-ring.homh*
unfolding *eval-hom-def UP-cring-def*
using *Qp-funs-is-cring UPQ.R-cring* **by** *blast*

lemma *Qp-fpoly-to-Qp-poly-X-var*:
assumes *a* ∈ *carrier* ($Q_p{}^n$)
shows *Qp-fpoly-to-Qp-poly n a* (*X-poly* (*Fun*$_n$ Q_p)) = *X-poly* Q_p
unfolding *X-poly-def Qp-fpoly-to-Qp-poly-def*
apply(*rule UP-cring.poly-lift-hom-X-var*) **unfolding** *UP-cring-def*
apply (*simp add*: *Qp-funs-is-cring*)
apply (*simp add*: *UPQ.R-cring*)
using *assms*(*1*) *eval-hom-is-hom ring-hom-ring.homh*
by *blast*

lemma *Qp-fpoly-to-Qp-poly-monom*:
assumes *a* ∈ *carrier* ($Q_p{}^n$)
assumes *f* ∈ *carrier* (*Fun*$_n$ Q_p)
shows *Qp-fpoly-to-Qp-poly n a* (*up-ring.monom* (*UP* (*Fun*$_n$ Q_p)) *f m*) = *up-ring.monom* Q_p*-x* (*f a*) *m*
unfolding *Qp-fpoly-to-Qp-poly-def*
using *UP-cring.poly-lift-hom-monom*[*of Fun*$_n$ Q_p Q_p *eval-hom a f m*] *assms ring-hom-ring.homh*
eval-hom-is-hom[*of a*] **unfolding** *eval-hom-def UP-cring-def*
using *Qp-funs-is-cring UPQ.R-cring* **by** *blast*

lemma *Qp-fpoly-to-Qp-poly-coeff*:
assumes *a* ∈ *carrier* ($Q_p{}^n$)
assumes *f* ∈ *carrier* (*UP* (*Fun*$_n$ Q_p))
shows *Qp-fpoly-to-Qp-poly n a f k* = (*f k*) *a*
using *assms UP-cring.poly-lift-hom-cf*[*of Fun*$_n$ Q_p Q_p *eval-hom a f k*] *eval-hom-is-hom*[*of a*]
unfolding *Qp-fpoly-to-Qp-poly-def eval-hom-def*
using *Qp-funs-is-cring ring-hom-ring.homh ring-hom-ring.homh*
unfolding *eval-hom-def UP-cring-def*
using *UPQ.R-cring* **by** *blast*

lemma *Qp-fpoly-to-Qp-poly-eval*:
assumes *a* ∈ *carrier* ($Q_p{}^n$)
assumes *P* ∈ *carrier* (*UP* (*Fun*$_n$ Q_p))
assumes *f* ∈ *carrier* (*Fun*$_n$ Q_p)
shows (*UP-cring.to-fun* (*Fun*$_n$ Q_p) *P f*) *a* = *UP-cring.to-fun* Q_p (*Qp-fpoly-to-Qp-poly n a P*) (*f a*)
unfolding *Qp-fpoly-to-Qp-poly-def*
using *UP-cring.poly-lift-hom-eval*[*of Fun*$_n$ Q_p Q_p *eval-hom a P f*]

eval-hom-is-hom[*of a*] *eval-hom-def assms ring-hom-ring.homh Qp-funs-is-cring*
unfolding *eval-hom-def UP-cring-def*
using *UPQ.R-cring* **by** *blast*

lemma *Qp-fpoly-to-Qp-poly-sub*:
assumes $f \in carrier\ (UP\ (Fun_n\ Q_p))$
assumes $g \in carrier\ (UP\ (Fun_n\ Q_p))$
assumes $a \in carrier\ ({Q_p}^n)$
shows *Qp-fpoly-to-Qp-poly n a* (*compose* (*Fun*$_n$ Q_p) *f g*) = *compose* Q_p (*Qp-fpoly-to-Qp-poly n a f*) (*Qp-fpoly-to-Qp-poly n a g*)
unfolding *Qp-fpoly-to-Qp-poly-def*
using *assms UP-cring.poly-lift-hom-sub*[*of Fun*$_n$ Q_p Q_p *eval-hom a f g*]
eval-hom-is-hom[*of a*] *ring-hom-ring.homh*[*of Fun*$_n$ Q_p Q_p *eval-hom a*]
Qp-funs-is-cring
unfolding *eval-hom-def UP-cring-def*
using *UPQ.R-cring* **by** *blast*

lemma *Qp-fpoly-to-Qp-poly-taylor-poly*:
assumes $F \in carrier\ (UP\ (Fun_n\ Q_p))$
assumes $c \in carrier\ (Fun_n\ Q_p)$
assumes $a \in carrier\ ({Q_p}^n)$
shows *Qp-fpoly-to-Qp-poly n a* (*taylor-expansion* (*Fun*$_n$ Q_p) *c F*) =
taylor-expansion Q_p (*c a*) (*Qp-fpoly-to-Qp-poly n a F*)
proof−
have *0*: *X-poly* (*Fun*$_n$ Q_p) $\oplus_{UP\ (Fun_n\ Q_p)}$ *to-polynomial* (*Fun*$_n$ Q_p) $c \in carrier\ (UP\ (Fun_n\ Q_p))$
by (*metis Qp-funs-is-cring UP-cring.X-plus-closed UP-cring-def X-poly-plus-def assms*(*2*))
have *1*: *poly-lift-hom* (*Fun*$_n$ Q_p) Q_p (*eval-hom a*) (*X-poly* (*Fun*$_n$ Q_p) $\oplus_{UP\ (Fun_n\ Q_p)}$ *to-polynomial* (*Fun*$_n$ Q_p) *c*) = *X-poly* Q_p $\oplus_{Q_p\text{-}x}$ *UPQ.to-poly* (*c a*)
proof−
have *10*: *poly-lift-hom* (*Fun*$_n$ Q_p) Q_p (*eval-hom a*) ∈ *ring-hom* (*UP* (*Fun*$_n$ Q_p)) Q_p*-x*
using *Qp-fpoly-to-Qp-poly-def Qp-fpoly-to-Qp-poly-is-hom assms*
by *presburger*
have *11*: *to-polynomial* (*Fun*$_n$ Q_p) $c \in carrier\ (UP\ (Fun_n\ Q_p))$
by (*meson Qp-funs-is-cring UP-cring.intro UP-cring.to-poly-closed assms*)
have *12*: *X-poly* (*Fun*$_n$ Q_p) ∈ *carrier* (*UP* (*Fun*$_n$ Q_p))
using *UP-cring.X-closed*[*of Fun*$_n$ Q_p] **unfolding** *UP-cring-def*
using *Qp-funs-is-cring*
by *blast*
have *Qp-fpoly-to-Qp-poly n a* (*X-poly* (*Fun*$_n$ Q_p) $\oplus_{UP\ (Fun_n\ Q_p)}$ *to-polynomial* (*Fun*$_n$ Q_p) *c*) =
Qp-fpoly-to-Qp-poly n a (*X-poly* (*Fun*$_n$ Q_p)) $\oplus_{Q_p\text{-}x}$ *Qp-fpoly-to-Qp-poly n a* (*to-polynomial* (*Fun*$_n$ Q_p) *c*)
using *assms 0 10 11 12 Qp-fpoly-to-Qp-poly-extends-apply*[*of a n c*] *Qp-fpoly-to-Qp-poly-is-hom*[*of a*] *Qp-fpoly-to-Qp-poly-X-var*[*of a*]
using *ring-hom-add*[*of Qp-fpoly-to-Qp-poly n a UP* (*Fun*$_n$ Q_p) Q_p*-x X-poly* (*Fun*$_n$ Q_p) *to-polynomial* (*Fun*$_n$ Q_p) *c*]

unfolding *Qp-fpoly-to-Qp-poly-def*
by *blast*
then show *?thesis*
using *Qp-fpoly-to-Qp-poly-X-var Qp-fpoly-to-Qp-poly-def Qp-fpoly-to-Qp-poly-extends-apply assms*
by *metis*
qed
have *2*: *poly-lift-hom* (*Fun*$_n$ Q_p) Q_p (*eval-hom a*) (*compose* (*Fun*$_n$ Q_p) *F* (*X-poly* (*Fun*$_n$ Q_p) $\oplus_{UP}$ (*Fun*$_n$ Q_p) *to-polynomial* (*Fun*$_n$ Q_p) *c*)) =
UPQ.sub (*poly-lift-hom* (*Fun*$_n$ Q_p) Q_p (*eval-hom a*) *F*)
(*poly-lift-hom* (*Fun*$_n$ Q_p) Q_p (*eval-hom a*) (*X-poly* (*Fun*$_n$ Q_p) $\oplus_{UP}$ (*Fun*$_n$ Q_p) *to-polynomial* (*Fun*$_n$ Q_p) *c*))
using *0 1 Qp-fpoly-to-Qp-poly-sub*[*of F n X-poly-plus* (*Fun*$_n$ Q_p) *c a*] *assms*
unfolding *Qp-fpoly-to-Qp-poly-def X-poly-plus-def*
by *blast*
show *?thesis*
using *assms 0 1*
unfolding *Qp-fpoly-to-Qp-poly-def taylor-expansion-def X-poly-plus-def*
using *2* **by** *presburger*
qed

lemma *SA-is-UP-cring*:
shows *UP-cring* (*SA n*)
unfolding *UP-cring-def*
by (*simp add*: *SA-is-cring*)

lemma *eval-hom-is-SA-hom*:
assumes *a* ∈ *carrier* ($Q_p{}^n$)
shows *ring-hom-ring* (*SA n*) Q_p (*eval-hom a*)
apply(*rule ring-hom-ringI*)
using *SA-is-cring cring.axioms*(*1*) *assms*(*1*) **apply** *blast*
using *Qp.ring-axioms* **apply** *blast*
apply (*metis* (*no-types*, *lifting*) *SA-car assms eval-hom-def Qp.function-ring-car-mem-closed semialg-functions-memE*(*2*))
apply (*metis* (*mono-tags*, *lifting*) *Qp-funs-mult' SA-car SA-times assms eval-hom-def semialg-functions-memE*(*2*))
apply (*metis* (*mono-tags*, *lifting*) *Qp-funs-add' SA-car SA-plus assms eval-hom-def semialg-functions-memE*(*2*))
using *Qp-constE Qp.one-closed SA-one assms eval-hom-def function-one-as-constant*
by (*metis function-one-eval*)

the homomorphism from `(SA n)[x]` to `Qp [x]` induced by evaluation of coefficients

definition *SA-poly-to-Qp-poly* **where**
SA-poly-to-Qp-poly n a = *poly-lift-hom* (*SA n*) Q_p (*eval-hom a*)

lemma *SA-poly-to-Qp-poly-is-hom*:
assumes *a* ∈ *carrier* ($Q_p{}^n$)
shows (*SA-poly-to-Qp-poly n a*) ∈ *ring-hom* (*UP* (*SA n*)) (Q_p-*x*)

unfolding *SA-poly-to-Qp-poly-def*
apply(*rule UP-cring.poly-lift-hom-is-hom*)
using *SA-is-cring assms*(*1*) *UP-cring.intro* **apply** *blast*
apply (*simp add*: *UPQ.R-cring*)
using *assms eval-hom-is-SA-hom ring-hom-ring.homh* **by** *blast*

lemma *SA-poly-to-Qp-poly-closed*:
assumes $a \in carrier\ ({Q_p}^n)$
assumes $P \in carrier\ (UP\ (SA\ n))$
shows *SA-poly-to-Qp-poly n a P* $\in carrier\ Q_p$*-x*
using *assms SA-poly-to-Qp-poly-is-hom*[*of a*] *ring-hom-closed*[*of SA-poly-to-Qp-poly n a UP* (*SA n*) Q_p*-x P*]
by *blast*

lemma *SA-poly-to-Qp-poly-add*:
assumes $a \in carrier\ ({Q_p}^n)$
assumes $f \in carrier\ (UP\ (SA\ n))$
assumes $g \in carrier\ (UP\ (SA\ n))$
shows *SA-poly-to-Qp-poly n a* $(f \oplus_{UP\ (SA\ n)} g) =$ *SA-poly-to-Qp-poly n a f* $\oplus_{Q_p\text{-}x}$ *SA-poly-to-Qp-poly n a g*
using *SA-poly-to-Qp-poly-is-hom ring-hom-add assms*
by (*metis* (*no-types*, *opaque-lifting*))

lemma *SA-poly-to-Qp-poly-minus*:
assumes $a \in carrier\ ({Q_p}^n)$
assumes $f \in carrier\ (UP\ (SA\ n))$
assumes $g \in carrier\ (UP\ (SA\ n))$
shows *SA-poly-to-Qp-poly n a* $(f \ominus_{UP\ (SA\ n)} g) =$ *SA-poly-to-Qp-poly n a f* $\ominus_{Q_p\text{-}x}$ *SA-poly-to-Qp-poly n a g*
using *SA-poly-to-Qp-poly-is-hom*[*of a*] *assms SA-is-ring*[*of n*]
ring.ring-hom-minus[*of UP* (*SA n*) Q_p*-x SA-poly-to-Qp-poly n a f g*]
UP-SA-n-is-ring
UPQ.UP-ring
by *blast*

lemma *SA-poly-to-Qp-poly-mult*:
assumes $a \in carrier\ ({Q_p}^n)$
assumes $f \in carrier\ (UP\ (SA\ n))$
assumes $g \in carrier\ (UP\ (SA\ n))$
shows *SA-poly-to-Qp-poly n a* $(f \otimes_{UP\ (SA\ n)} g) =$ *SA-poly-to-Qp-poly n a f* $\otimes_{Q_p\text{-}x}$ *SA-poly-to-Qp-poly n a g*
using *SA-poly-to-Qp-poly-is-hom ring-hom-mult assms*
by (*metis* (*no-types*, *opaque-lifting*))

lemma *SA-poly-to-Qp-poly-extends-apply*:
assumes $a \in carrier\ ({Q_p}^n)$
assumes $f \in carrier\ (SA\ n)$
shows *SA-poly-to-Qp-poly n a* (*to-polynomial* (*SA n*) *f*) = *to-polynomial* Q_p (*f a*)

unfolding *SA-poly-to-Qp-poly-def*
using *assms eval-hom-is-SA-hom*[*of a*] *UP-cring.poly-lift-hom-extends-hom*[*of SA n* Q_p *eval-hom a f*]
eval-hom-def SA-is-cring Qp.cring-axioms ring-hom-ring.homh
unfolding *eval-hom-def UP-cring-def*
by *blast*

lemma *SA-poly-to-Qp-poly-X-var*:
assumes *a* ∈ *carrier* ($Q_p{}^n$)
shows *SA-poly-to-Qp-poly n a* (*X-poly* (*SA n*)) = *X-poly* Q_p
unfolding *X-poly-def SA-poly-to-Qp-poly-def*
apply(*rule UP-cring.poly-lift-hom-X-var*)
using *SA-is-cring assms*(*1*)
using *UP-cring.intro* **apply** *blast*
apply (*simp add*: *Qp.cring-axioms*)
using *assms eval-hom-is-SA-hom ring-hom-ring.homh* **by** *blast*

lemma *SA-poly-to-Qp-poly-X-plus*:
assumes *a* ∈ *carrier* ($Q_p{}^n$)
assumes *c* ∈ *carrier* (*SA n*)
shows *SA-poly-to-Qp-poly n a* (*X-poly-plus* (*SA n*) *c*) = *UPQ.X-plus* (*c a*)
unfolding *X-poly-plus-def*
using *assms SA-poly-to-Qp-poly-add*[*of a n X-poly* (*SA n*) *to-polynomial* (*SA n*) *c*]
SA-poly-to-Qp-poly-extends-apply[*of a n c*] *UP-cring.X-closed*[*of SA n*] *SA-is-cring*[*of n*]
SA-poly-to-Qp-poly-X-var[*of a*] *UP-cring.to-poly-closed*[*of SA n c*]
unfolding *UP-cring-def*
by *metis*

lemma *SA-poly-to-Qp-poly-X-minus*:
assumes *a* ∈ *carrier* ($Q_p{}^n$)
assumes *c* ∈ *carrier* (*SA n*)
shows *SA-poly-to-Qp-poly n a* (*X-poly-minus* (*SA n*) *c*) = *UPQ.X-minus* (*c a*)
unfolding *X-poly-minus-def*
using *assms SA-poly-to-Qp-poly-minus*[*of a n X-poly* (*SA n*) *to-polynomial* (*SA n*) *c*]
SA-poly-to-Qp-poly-extends-apply[*of a n c*] *UP-cring.X-closed*[*of SA n*] *SA-is-cring*[*of n*]
SA-poly-to-Qp-poly-X-var[*of a n*] *UP-cring.to-poly-closed*[*of SA n c*]
unfolding *UP-cring-def*
by *metis*

lemma *SA-poly-to-Qp-poly-monom*:
assumes *a* ∈ *carrier* ($Q_p{}^n$)
assumes *f* ∈ *carrier* (*SA n*)
shows *SA-poly-to-Qp-poly n a* (*up-ring.monom* (*UP* (*SA n*)) *f m*) = *up-ring.monom* Q_p-*x* (*f a*) *m*
unfolding *SA-poly-to-Qp-poly-def*

using *UP-cring.poly-lift-hom-monom*[*of SA n* Q_p *eval-hom a f n*] *assms eval-hom-is-SA-hom eval-hom-def*
SA-is-cring Qp.cring-axioms UP-cring.poly-lift-hom-monom ring-hom-ring.homh
by (*metis UP-cring.intro*)

lemma *SA-poly-to-Qp-poly-coeff*:
assumes $a \in$ *carrier* ($Q_p{}^n$)
assumes $f \in$ *carrier* (*UP* (*SA n*))
shows *SA-poly-to-Qp-poly n a f k* = (*f k*) *a*
using *assms UP-cring.poly-lift-hom-cf*[*of SA n* Q_p *eval-hom a f k*] *eval-hom-is-SA-hom*[*of a*]
using *SA-is-cring Qp.cring-axioms ring-hom-ring.homh*
unfolding *SA-poly-to-Qp-poly-def eval-hom-def UP-cring-def*
by *blast*

lemma *SA-poly-to-Qp-poly-eval*:
assumes $a \in$ *carrier* ($Q_p{}^n$)
assumes $P \in$ *carrier* (*UP* (*SA n*))
assumes $f \in$ *carrier* (*SA n*)
shows (*UP-cring.to-fun* (*SA n*) *P f*) *a* = *UP-cring.to-fun* Q_p (*SA-poly-to-Qp-poly n a P*) (*f a*)
unfolding *SA-poly-to-Qp-poly-def*
using *UP-cring.poly-lift-hom-eval*[*of SA n* Q_p *eval-hom a P f*]
eval-hom-is-SA-hom[*of a*] *eval-hom-def assms SA-is-cring Qp.cring-axioms ring-hom-ring.homh*
unfolding *SA-poly-to-Qp-poly-def eval-hom-def UP-cring-def*
by *blast*

lemma *SA-poly-to-Qp-poly-sub*:
assumes $f \in$ *carrier* (*UP* (*SA n*))
assumes $g \in$ *carrier* (*UP* (*SA n*))
assumes $a \in$ *carrier* ($Q_p{}^n$)
shows *SA-poly-to-Qp-poly n a* (*compose* (*SA n*) *f g*) = *compose* Q_p (*SA-poly-to-Qp-poly n a f*) (*SA-poly-to-Qp-poly n a g*)
unfolding *SA-poly-to-Qp-poly-def*
using *assms UP-cring.poly-lift-hom-sub*[*of SA n* Q_p *eval-hom a f g*]
eval-hom-is-SA-hom[*of a*] *ring-hom-ring.homh*[*of SA n* Q_p *eval-hom a*]
SA-is-cring Qp.cring-axioms
unfolding *SA-poly-to-Qp-poly-def eval-hom-def UP-cring-def*
by *blast*

lemma *SA-poly-to-Qp-poly-deg-bound*:
assumes $g \in$ *carrier* (*UP* (*SA m*))
assumes $x \in$ *carrier* ($Q_p{}^m$)
shows *deg* Q_p (*SA-poly-to-Qp-poly m x g*) $\leq$ *deg* (*SA m*) *g*
apply(*rule UPQ.deg-leqI*)
using *assms SA-poly-to-Qp-poly-closed*[*of x m g*] **apply** *blast*
proof− **fix** *n* **assume** *A*: *deg* (*SA m*) *g* < *n*
then have *g n* = $\mathbf{0}_{SA\ m}$

using *assms SA-is-UP-cring*[*of m*] *UP-cring.UP-car-memE*(*2*) **by** *blast*
thus *SA-poly-to-Qp-poly m x g n* = **0**
using *assms SA-poly-to-Qp-poly-coeff*[*of x m g n*] *function-zero-eval SA-zero*
by *presburger*
qed

lemma *SA-poly-to-Qp-poly-taylor-poly*:
assumes $F \in$ *carrier* (*UP* (*SA n*))
assumes $c \in$ *carrier* (*SA n*)
assumes $a \in$ *carrier* ($Q_p{}^n$)
shows *SA-poly-to-Qp-poly n a* (*taylor-expansion* (*SA n*) *c F*) =
taylor-expansion Q_p (*c a*) (*SA-poly-to-Qp-poly n a F*)
unfolding *SA-poly-to-Qp-poly-def* **using** *assms Qp.cring-axioms SA-is-cring eval-hom-def*
eval-hom-is-SA-hom UP-cring.poly-lift-hom-comm-taylor-expansion[*of SA n*
Q_p *eval-hom a F c*] *ring-hom-ring.homh*
unfolding *SA-poly-to-Qp-poly-def eval-hom-def UP-cring-def*
by *metis*

lemma *SA-poly-to-Qp-poly-comm-taylor-term*:
assumes $F \in$ *carrier* (*UP* (*SA n*))
assumes $c \in$ *carrier* (*SA n*)
assumes $a \in$ *carrier* ($Q_p{}^n$)
shows *SA-poly-to-Qp-poly n a* (*UP-cring.taylor-term* (*SA n*) *c F i*) =
UP-cring.taylor-term Q_p (*c a*) (*SA-poly-to-Qp-poly n a F*) *i*
unfolding *SA-poly-to-Qp-poly-def* **using** *assms Qp.cring-axioms SA-is-cring*
eval-hom-def
eval-hom-is-SA-hom UP-cring.poly-lift-hom-comm-taylor-term[*of SA n* Q_p
eval-hom a F c i] *ring-hom-ring.homh*
unfolding *SA-poly-to-Qp-poly-def eval-hom-def UP-cring-def*
by *metis*

lemma *SA-poly-to-Qp-poly-comm-pderiv*:
assumes $F \in$ *carrier* (*UP* (*SA n*))
assumes $a \in$ *carrier* ($Q_p{}^n$)
shows *SA-poly-to-Qp-poly n a* (*UP-cring.pderiv* (*SA n*) *F*) =
UP-cring.pderiv Q_p (*SA-poly-to-Qp-poly n a F*)
apply(*rule UP-ring.poly-induct3*[*of SA n F*]) **unfolding** *UP-ring-def*
apply (*simp add*: *SA-is-ring assms*(*1*))
using *assms* **apply** *blast*
proof−
show $\bigwedge p\ q.\ q \in$ *carrier* (*UP* (*SA n*)) $\Longrightarrow$
$p \in$ *carrier* (*UP* (*SA n*)) $\Longrightarrow$
SA-poly-to-Qp-poly n a (*UP-cring.pderiv* (*SA n*) *p*) = *UPQ.pderiv*
(*SA-poly-to-Qp-poly n a p*) $\Longrightarrow$
SA-poly-to-Qp-poly n a (*UP-cring.pderiv* (*SA n*) *q*) = *UPQ.pderiv*
(*SA-poly-to-Qp-poly n a q*) $\Longrightarrow$
SA-poly-to-Qp-poly n a (*UP-cring.pderiv* (*SA n*) ($p \oplus_{UP\ (SA\ n)} q$)) =
UPQ.pderiv (*SA-poly-to-Qp-poly n a* ($p \oplus_{UP\ (SA\ n)} q$))
proof− **fix** *p q* **assume** *A*: $q \in$ *carrier* (*UP* (*SA n*))

$p \in carrier\ (UP\ (SA\ n))$
$SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ (UP\text{-}cring.pderiv\ (SA\ n)\ p) = UPQ.pderiv\ (SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ p)$
$SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ (UP\text{-}cring.pderiv\ (SA\ n)\ q) = UPQ.pderiv\ (SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ q)$
show $SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ (UP\text{-}cring.pderiv\ (SA\ n)\ (p \oplus_{UP\ (SA\ n)} q)) = UPQ.pderiv\ (SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ (p \oplus_{UP\ (SA\ n)} q))$
proof −
have *0*: $SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ p \in carrier\ (UP\ Q_p)$
using *A assms SA-poly-to-Qp-poly-closed*[*of a n p*]
by *blast*
have *1*: $SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ q \in carrier\ (UP\ Q_p)$
using *A SA-poly-to-Qp-poly-closed*[*of a n q*] *assms* **by** *blast*
have *2*: $UPQ.pderiv\ (SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ p) \in carrier\ (UP\ Q_p)$
using *UPQ.pderiv-closed*[*of SA-poly-to-Qp-poly n a p*] *0* **by** *blast*
have *3*: $UPQ.pderiv\ (SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ q) \in carrier\ (UP\ Q_p)$
using *A assms UPQ.pderiv-closed*[*of SA-poly-to-Qp-poly n a q*] *1* **by** *blast*
have *4*: $UP\text{-}cring.pderiv\ (SA\ n)\ p \in carrier\ (UP\ (SA\ n))$
using *A UP-cring.pderiv-closed*[*of SA n p*] **unfolding** *UP-cring-def*
using *SA-is-cring assms*(*1*) **by** *blast*
have *5*: $UP\text{-}cring.pderiv\ (SA\ n)\ q \in carrier\ (UP\ (SA\ n))$
using *A UP-cring.pderiv-closed*[*of SA n q*] **unfolding** *UP-cring-def*
using *SA-is-cring assms*(*1*) **by** *blast*
have *6*: $SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ (UP\text{-}cring.pderiv\ (SA\ n)\ p \oplus_{UP\ (SA\ n)} UP\text{-}cring.pderiv\ (SA\ n)\ q) = SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ (UP\text{-}cring.pderiv\ (SA\ n)\ p) \oplus_{UP\ Q_p} SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ (UP\text{-}cring.pderiv\ (SA\ n)\ q)$
using *A 4 5 SA-poly-to-Qp-poly-add assms* **by** *blast*
have *7*: $UPQ.pderiv\ (SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ p \oplus_{UP\ Q_p} SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ q) = UPQ.pderiv\ (SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ p) \oplus_{UP\ Q_p} UPQ.pderiv\ (SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ q)$
using *0 1 UPQ.pderiv-add* **by** *blast*
have *8*: $UP\text{-}cring.pderiv\ (SA\ n)\ (p \oplus_{UP\ (SA\ n)} q) = UP\text{-}cring.pderiv\ (SA\ n)\ p \oplus_{UP\ (SA\ n)} UP\text{-}cring.pderiv\ (SA\ n)\ q$
using *A assms UP-cring.pderiv-add*[*of SA n p q*]
unfolding *UP-cring-def* **using** *SA-is-cring* **by** *blast*
have *9*: $SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ (UP\text{-}cring.pderiv\ (SA\ n)\ (p \oplus_{UP\ (SA\ n)} q)) = UPQ.pderiv\ (SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ p) \oplus_{UP\ Q_p} UPQ.pderiv\ (SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ q)$
using *A 6 8* **by** *presburger*
have *10*: $UPQ.pderiv\ (SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ (p \oplus_{UP\ (SA\ n)} q)) = UPQ.pderiv\ (SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ p) \oplus_{UP\ Q_p} UPQ.pderiv\ (SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ q)$
using *7 A*(*1*) *A*(*2*) *SA-poly-to-Qp-poly-add assms* **by** *presburger*
show *?thesis* **using** *9 10*
by *presburger*

qed
qed
show $\bigwedge aa\ na.$
$aa \in carrier\ (SA\ n) \Longrightarrow$
$SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ (UP\text{-}cring.pderiv\ (SA\ n)\ (up\text{-}ring.monom\ (UP\ (SA\ n))\ aa\ na)) =$
$UPQ.pderiv\ (SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ (up\text{-}ring.monom\ (UP\ (SA\ n))\ aa\ na))$
proof− **fix** $b\ m$ **assume** A: $b \in carrier\ (SA\ n)$
show $SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ (UP\text{-}cring.pderiv\ (SA\ n)\ (up\text{-}ring.monom\ (UP\ (SA\ n))\ b\ m)) =$
$UPQ.pderiv\ (SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ (up\text{-}ring.monom\ (UP\ (SA\ n))\ b\ m))$
proof−
have *0*: $(UP\text{-}cring.pderiv\ (SA\ n)\ (up\text{-}ring.monom\ (UP\ (SA\ n))\ b\ m)) =$
$(up\text{-}ring.monom\ (UP\ (SA\ n))\ ([m]\cdot_{SA\ n} b)\ (m-1))$
using $UP\text{-}cring.pderiv\text{-}monom[of\ SA\ n\ b\ m]$ **unfolding** $UP\text{-}cring\text{-}def$
using $SA\text{-}is\text{-}cring\ \langle b \in carrier\ (SA\ n)\rangle\ assms(1)$ **by** *blast*
have *1*: $(SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ (up\text{-}ring.monom\ (UP\ (SA\ n))\ b\ m)) = up\text{-}ring.monom\ (UP\ Q_p)\ (b\ a)\ m$
using $SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\text{-}monom\ \langle b \in carrier\ (SA\ n)\rangle\ assms$ **by** *blast*
have *2*: $b\ a \in carrier\ Q_p$
using $A\ assms\ Qp.function\text{-}ring\text{-}car\text{-}mem\text{-}closed\ SA\text{-}car\text{-}memE(2)$ **by** *metis*
hence *3*: $UPQ.pderiv\ (SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ (up\text{-}ring.monom\ (UP\ (SA\ n))\ b\ m)) = up\text{-}ring.monom\ (UP\ Q_p)\ ([m]\cdot(b\ a))\ (m-1)$
using *1 2 A* $UPQ.pderiv\text{-}monom[of\ b\ a\ m]$
by *presburger*
have *4*: $[m] \cdot_{SA\ n} b \in carrier\ (SA\ n)$
using $A\ assms\ SA\text{-}is\text{-}cring[of\ n]\ ring.add\text{-}pow\text{-}closed[of\ SA\ n\ b\ m]\ SA\text{-}is\text{-}ring$
by *blast*
have *5*: $SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ (up\text{-}ring.monom\ (UP\ (SA\ n))\ ([m] \cdot_{SA\ n} b)\ (m-1)) = up\text{-}ring.monom\ (UP\ Q_p)\ (([m] \cdot_{SA\ n} b)\ a)\ (m-1)$
using $SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\text{-}monom[of\ a\ n\ [m]\cdot_{SA\ n} b\ m-1]\ assms\ 4$ **by** *blast*
have *6*: $SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ (UP\text{-}cring.pderiv\ (SA\ n)\ (up\text{-}ring.monom\ (UP\ (SA\ n))\ b\ m)) = up\text{-}ring.monom\ (UP\ Q_p)\ (([m] \cdot_{SA\ n} b)\ a)\ (m-1)$
using *5 0* **by** *presburger*
thus *?thesis* **using** $assms\ A\ 3\ 6\ SA\text{-}add\text{-}pow\text{-}apply[of\ b\ n\ a]$
by *auto*
qed
qed
qed

lemma *SA-poly-to-Qp-poly-pderiv*:
assumes $g \in carrier\ (UP\ (SA\ m))$
assumes $x \in carrier\ ({Q_p}^m)$
shows $UPQ.pderiv\ (SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ m\ x\ g) = (SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ m\ x\ (pderiv\ m\ g))$
proof
fix n
have *0*: $UPQ.pderiv\ (SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ m\ x\ g)\ n = [Suc\ n] \cdot SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ m\ x\ g\ (Suc\ n)$

by(*rule UPQ.pderiv-cfs*[*of SA-poly-to-Qp-poly m x gn*], *rule SA-poly-to-Qp-poly-closed*, *rule assms* , *rule assms*)
have *1*: *SA-poly-to-Qp-poly m x* (*UPSA.pderiv m g*) *n* = *UPSA.pderiv m g n x*
by(*rule SA-poly-to-Qp-poly-coeff*[*of x m UPSA.pderiv m g n*], *rule assms*, *rule UPSA.pderiv-closed*, *rule assms*)
have *2*: *SA-poly-to-Qp-poly m x* (*UPSA.pderiv m g*) *n* = ([*Suc n*] ·$_{SA\ m}$ *g* (*Suc n*)) *x*
using *UPSA.pderiv-cfs*[*of g m n*] *assms* **unfolding** *1* **by** *auto*
show *UPQ.pderiv* (*SA-poly-to-Qp-poly m x g*) *n* = *SA-poly-to-Qp-poly m x* (*UPSA.pderiv m g*) *n*
unfolding *0 2* **using** *SA-poly-to-Qp-poly-coeff assms*
by (*metis 0 2 SA-poly-to-Qp-poly-comm-pderiv*)
qed

lemma(**in** *UP-cring*) *pderiv-deg-lt*:
assumes *f* ∈ *carrier* (*UP R*)
assumes *deg R f* > *0*
shows *deg R* (*pderiv f*) < *deg R f*
proof−
obtain *n* **where** *n-def*: *n* = *deg R f*
by *blast*
have *0*: ⋀*k*. *k* ≥ *n* ⟹ *pderiv f k* = **0**
using *pderiv-cfs assms* **unfolding** *n-def*
by (*simp add*: *UP-car-memE*(*2*))
obtain *k* **where** *k-def*: *n* = *Suc k*
using *n-def assms gr0-implies-Suc* **by** *presburger*
have *deg R* (*pderiv f*) ≤ *k*
apply(*rule deg-leqI*)
using *P-def assms*(*1*) *pderiv-closed* **apply** *presburger*
apply(*rule 0*)
unfolding *k-def* **by** *presburger*
thus *?thesis* **using** *k-def* **unfolding** *n-def* **by** *linarith*
qed

lemma *deg-pderiv*:
assumes *f* ∈ *carrier* (*UP* (*SA m*))
assumes *deg* (*SA m*) *f* > *0*
shows *deg* (*SA m*) (*pderiv m f*) = *deg* (*SA m*) *f* − *1*
proof−
obtain *n* **where** *n-def*: *n* = *deg* (*SA m*) *f*
by *blast*
have *0*: *f n* ≠ **0**$_{SA\ m}$
unfolding *n-def* **using** *assms UPSA.deg-ltrm* **by** *fastforce*
have *1*: (*pderiv m f*) (*n*−*1*) = [*n*]·$_{SA\ m}$ (*f n*)
using *assms* **unfolding** *n-def* **using** *Suc-diff-1 UPSA.pderiv-cfs* **by** *presburger*
have *2*: *deg* (*SA m*) (*pderiv m f*) ≥ (*n*−*1*)
using *0 assms SA-char-zero*
by (*metis 1 UPSA.deg-eqI UPSA.lcf-closed UPSA.pderiv-closed n-def nat-le-linear*)
have *3*: *deg* (*SA m*) (*pderiv m f*) < *deg* (*SA m*) *f*

using *assms pderiv-deg-lt* **by** *auto*
thus *?thesis* **using** *2* **unfolding** *n-def* **by** *presburger*
qed

lemma *SA-poly-to-Qp-poly-smult*:
assumes $a \in carrier\ (SA\ m)$
assumes $f \in carrier\ (UP\ (SA\ m))$
assumes $x \in carrier\ ({Q_p}^m)$
shows $SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ m\ x\ (a \odot_{UP\ (SA\ m)} f) = a\ x \odot_{UP\ Q_p} SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ m\ x\ f$
proof−
have *0*: $a \odot_{UP\ (SA\ m)} f = to\text{-}polynomial\ (SA\ m)\ a \otimes_{UP\ (SA\ m)} f$
using *assms UPSA.to-poly-mult-simp*(*1*) **by** *presburger*
have *1*: $SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ m\ x\ (a \odot_{UP\ (SA\ m)} f) = SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ m\ x\ (to\text{-}polynomial\ (SA\ m)\ a) \otimes_{UP\ Q_p} SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ m\ x\ f$
unfolding *0* **apply**(*rule SA-poly-to-Qp-poly-mult*)
using *assms* **apply** *blast*
using *assms to-poly-closed* **apply** *blast*
using *assms* **by** *blast*
have *2*: $SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ m\ x\ (a \odot_{UP\ (SA\ m)} f) = (to\text{-}polynomial\ Q_p\ (a\ x)) \otimes_{UP\ Q_p} SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ m\ x\ f$
unfolding *1* **using** *assms SA-poly-to-Qp-poly-monom* **unfolding** *to-polynomial-def*
by *presburger*
show *?thesis*
unfolding *2* **apply**(*rule UP-cring.to-poly-mult-simp*(*1*)[*of* Q_p *a x SA-poly-to-Qp-poly m x f*])
unfolding *UP-cring-def*
using *F.R-cring* **apply** *blast*
using *assms SA-car-memE* **apply** *blast*
using *assms SA-poly-to-Qp-poly-closed*[*of x m f*] **by** *blast*
qed

lemma *SA-poly-constant-res-class-semialg*:
assumes $f \in carrier\ (UP\ (SA\ m))$
assumes $\bigwedge i\ x.\ x \in carrier\ ({Q_p}^m) \Longrightarrow f\ i\ x \in \mathcal{O}_p$
assumes $deg\ (SA\ m)\ f \leq d$
assumes $C \in poly\text{-}res\text{-}classes\ n\ d$
shows $is\text{-}semialgebraic\ m\ \{x \in carrier\ ({Q_p}^m).\ SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ m\ x\ f \in C\}$
proof−
obtain *Fs* **where** *Fs-def*: $Fs = f\ `\ \{..d\}$
by *blast*
obtain *g* **where** *g-def*: $g \in val\text{-}ring\text{-}polys\text{-}grad\ d \wedge C = poly\text{-}res\text{-}class\ n\ d\ g$
using *assms* **unfolding** *poly-res-classes-def* **by** *blast*
have *C-eq*: $C = poly\text{-}res\text{-}class\ n\ d\ g$
using *g-def* **by** *blast*
have *0*: $\{x \in carrier\ ({Q_p}^m).\ SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ m\ x\ f \in C\} = (\bigcap\ i \in \{..d\}.\ \{x \in carrier\ ({Q_p}^m).\ f\ i\ x \in \mathcal{O}_p \wedge Qp\text{-}res\ (f\ i\ x)\ n = Qp\text{-}res\ (g\ i)\ n\})$
apply(*rule equalityI'*)

unfolding *mem-Collect-eq*
proof(*rule InterI*)
fix *x S*
assume *A*: $x \in$ *carrier* $(Q_p{}^m) \wedge$ *SA-poly-to-Qp-poly m x f* $\in C$
$S \in (\lambda i.\ \{x \in$ *carrier* $(Q_p{}^m).\ f\ i\ x \in \mathcal{O}_p \wedge$ *Qp-res* $(f\ i\ x)\ n =$ *Qp-res* $(g\ i)\ n\})$ ' $\{..d\}$
have *0*: $C =$ *poly-res-class n d* (*SA-poly-to-Qp-poly m x f*)
unfolding *C-eq*
apply(*rule equalityI'*, *rule poly-res-class-memI*, *rule poly-res-class-memE*[*of - n d g*], *blast*
, *rule poly-res-class-memE*[*of - n d g*], *blast*, *rule poly-res-class-memE*[*of - n d g*], *blast*)
using *poly-res-class-memE*[*of - n d*]*A*
apply (*metis* (*no-types*, *lifting*) *C-eq*)
apply(*rule poly-res-class-memI*, *rule poly-res-class-memE*[*of - n d SA-poly-to-Qp-poly m x f*], *blast*,
rule poly-res-class-memE[*of - n d SA-poly-to-Qp-poly m x f*], *blast*,
rule poly-res-class-memE[*of - n d SA-poly-to-Qp-poly m x f*], *blast*)
using *poly-res-class-memE*[*of - n d*]*A*
by (*metis* (*no-types*, *lifting*) *C-eq*)
obtain *i* **where** *i-def*: $i \in \{..d\} \wedge$
$S = \{x \in$ *carrier* $(Q_p{}^m).\ f\ i\ x \in \mathcal{O}_p \wedge$ *Qp-res* $(f\ i\ x)\ n =$ *Qp-res* $(g\ i)\ n\}$
using *A* **by** *blast*
have *S-eq*: $S = \{x \in$ *carrier* $(Q_p{}^m).\ f\ i\ x \in \mathcal{O}_p \wedge$ *Qp-res* $(f\ i\ x)\ n =$ *Qp-res* $(g\ i)\ n\}$
using *i-def* **by** *blast*
have *1*: $\bigwedge i.$ *SA-poly-to-Qp-poly m x f i* $=$ *f i x*
apply(*rule SA-poly-to-Qp-poly-coeff*)
using *A* **apply** *blast* **by**(*rule assms*)
have *2*: *deg* Q_p (*SA-poly-to-Qp-poly m x f*) $\leq$ *d*
using *assms SA-poly-to-Qp-poly-deg-bound*[*of f m x*]
using *A*(*1*) **by** *linarith*
have *3*: *Qp-res* (*SA-poly-to-Qp-poly m x f i*) *n* = *Qp-res* (*g i*) *n*
apply(*rule poly-res-class-memE*[*of - - d*], *rule poly-res-class-memI*)
using *g-def val-ring-polys-grad-memE* **apply** *blast*
using *g-def val-ring-polys-grad-memE* **apply** *blast*
using *g-def val-ring-polys-grad-memE* **apply** *blast*
apply(*rule poly-res-class-memE*[*of - - d*],*rule poly-res-class-memI*)
apply(*rule SA-poly-to-Qp-poly-closed*)
using *A* **apply** *blast*
apply(*rule assms*)
apply(*rule 2*)
unfolding *1* **using** *assms A* **apply** *blast*
using *A* **unfolding** *C-eq*
using *poly-res-class-memE*(*4*)[*of SA-poly-to-Qp-poly m x f n d g*]
unfolding *1* **by** *metis*
show *x* $\in$ *S*
using *A 3 assms*

unfolding *S-eq mem-Collect-eq* **unfolding** *1*
by *blast*
next

show $\bigwedge x.\ x \in (\bigcap i{\leq}d.\ \{x \in carrier\ ({Q_p}^m).\ f\ i\ x \in \mathcal{O}_p \wedge Qp\text{-}res\ (f\ i\ x)\ n = Qp\text{-}res\ (g\ i)\ n\}) \Longrightarrow x \in carrier\ ({Q_p}^m) \wedge SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ m\ x\ f \in C$
proof−
fix *x* **assume** *A*: $x \in (\bigcap i{\leq}d.\ \{x \in carrier\ ({Q_p}^m).\ f\ i\ x \in \mathcal{O}_p \wedge Qp\text{-}res\ (f\ i\ x)\ n = Qp\text{-}res\ (g\ i)\ n\})$
have *x-closed*: $x \in carrier\ ({Q_p}^m)$
using *A* **by** *blast*
have *0*: $\bigwedge i.\ SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ m\ x\ f\ i = f\ i\ x$
apply(*rule SA-poly-to-Qp-poly-coeff*)
using *A* **apply** *blast* **by**(*rule assms*)
have *1*: $deg\ Q_p\ (SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ m\ x\ f) \leq d$
using *assms SA-poly-to-Qp-poly-deg-bound*[*of f m x*]
using *x-closed* **by** *linarith*
have *2*: $\bigwedge i.\ Qp\text{-}res\ (f\ i\ x)\ n = Qp\text{-}res\ (g\ i)\ n$
proof− **fix** *i*
have *20*: $i > d \Longrightarrow i > deg\ Q_p\ g$
using *g-def val-ring-polys-grad-memE*(*2*) **by** *fastforce*
have *21*: $i > d \Longrightarrow g\ i{=}\ \mathbf{0}$
using *20 g-def val-ring-polys-grad-memE UPQ.deg-leE* **by** *blast*
have *22*: $i > d \Longrightarrow Qp\text{-}res\ (g\ i)\ n = 0$
unfolding *21 Qp-res-zero* **by** *blast*
have *23*: $i > d \Longrightarrow SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ m\ x\ f\ i = \mathbf{0}$
apply(*rule UPQ.deg-leE*, *rule SA-poly-to-Qp-poly-closed*, *rule x-closed*, *rule assms*)
by(*rule le-less-trans*[*of - d*], *rule 1*, *blast*)
show $Qp\text{-}res\ (f\ i\ x)\ n = Qp\text{-}res\ (g\ i)\ n$
apply(*cases* $i \leq d$)
using *A* **apply** *blast*
using *22 21 1 23* **unfolding** *0*
by (*metis less-or-eq-imp-le linorder-neqE-nat*)
qed
have *3*: $SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ m\ x\ f \in C$
unfolding *C-eq*
apply(*rule poly-res-class-memI*, *rule SA-poly-to-Qp-poly-closed*, *rule x-closed*, *rule assms*, *rule 1*)
unfolding *0*
by(*rule assms*, *rule x-closed*, *rule 2*)
show $x \in carrier\ ({Q_p}^m) \wedge SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ m\ x\ f \in C$
using *x-closed 3* **by** *blast*
qed
qed
have *1*: $\bigwedge i.\ is\text{-}semialgebraic\ m\ \{x \in carrier\ ({Q_p}^m).\ f\ i\ x \in \mathcal{O}_p \wedge Qp\text{-}res\ (f\ i\ x)\ n = Qp\text{-}res\ (g\ i)\ n\}$
apply(*rule SA-constant-res-set-semialg*, *rule Qp-res-closed*, *rule val-ring-polys-grad-memE*[*of - d*])

using *g-def* **apply** *blast*
using *assms UPSA.UP-car-memE(1)* **by** *blast*
show *is-semialgebraic m* {*x* ∈ *carrier* (*Q*$_p$^*m*). *SA-poly-to-Qp-poly m x f* ∈ *C*}
unfolding *0*
apply(*rule finite-intersection-is-semialg*, *blast*, *blast*, *rule subsetI*)
using *1* **unfolding** *is-semialgebraic-def* **by** *blast*
qed

Maps a polynomial $F(t) \in UP(SAn)$ to a function sending $(t, a) \in (Q_p(n+1) \mapsto F(a)(t) \in Q_p$

definition *SA-poly-to-SA-fun* **where**
SA-poly-to-SA-fun n P = (λ *a* ∈ *carrier* (*Q*$_p$^*Suc n*). *UP-cring.to-fun Q*$_p$ (*SA-poly-to-Qp-poly n* (*tl a*) *P*) (*hd a*))

lemma *SA-poly-to-SA-fun-is-fun*:
assumes *P* ∈ *carrier* (*UP* (*SA n*))
shows *SA-poly-to-SA-fun n P* ∈ (*carrier* (*Q*$_p$^*Suc n*) → *carrier Q*$_p$)
proof fix *x* **assume** *A*: *x* ∈ *carrier* (*Q*$_p$^*Suc n*)
obtain *t* **where** *t-def*: *t* = *hd x* **by** *blast*
obtain *a* **where** *a-def*: *a* = *tl x* **by** *blast*
have *t-closed*: *t* ∈ *carrier Q*$_p$
using *A t-def cartesian-power-head*
by *blast*
have *a-closed*: *a* ∈ *carrier* (*Q*$_p$^*n*)
using *A a-def cartesian-power-tail*
by *blast*
have *0*: *SA-poly-to-SA-fun n P x* = *UP-cring.to-fun Q*$_p$ (*SA-poly-to-Qp-poly n a P*) *t*
unfolding *SA-poly-to-SA-fun-def* **using** *t-def a-def*
by (*meson A restrict-apply'*)
show *SA-poly-to-SA-fun n P x* ∈ *carrier Q*$_p$
using *assms t-closed a-def 0 UP-cring.to-fun-closed*[*of Q*$_p$ *SA-poly-to-Qp-poly n a P*]
unfolding *SA-poly-to-SA-fun-def*
using *SA-poly-to-Qp-poly-closed a-closed UPQ.to-fun-closed* **by** *presburger*
qed

lemma *SA-poly-to-SA-fun-formula*:
assumes *P* ∈ *carrier* (*UP* (*SA n*))
assumes *x* ∈ *carrier* (*Q*$_p$^*n*)
assumes *t* ∈ *carrier Q*$_p$
shows *SA-poly-to-SA-fun n P* (*t*#*x*) = (*SA-poly-to-Qp-poly n x P*)•*t*
proof−
have *0*: *hd* (*t*#*x*) = *t*
by *simp*
have *1*: *tl* (*t*#*x*) = *x*
by *auto*
have *2*: (*t*#*x*) ∈ *carrier* (*Q*$_p$^*Suc n*)
by (*metis add.commute assms cartesian-power-cons plus-1-eq-Suc*)

show *?thesis*
unfolding *SA-poly-to-SA-fun-def*
using *0 1 2 assms*
by (*metis* (*no-types*, *lifting*) *restrict-apply'*)
qed

lemma *semialg-map-comp-in-SA*:
assumes $f \in carrier\ (SA\ n)$
assumes *is-semialg-map m n g*
shows $(\lambda\ a \in carrier\ ({Q_p}^m).\ f\ (g\ a)) \in carrier\ (SA\ m)$
proof(*rule SA-car-memI*)
show $(\lambda a{\in}carrier\ ({Q_p}^m).\ f\ (g\ a)) \in carrier\ (Qp\text{-}funs\ m)$
proof(*rule Qp-funs-car-memI*)
show $(\lambda a{\in}carrier\ ({Q_p}^m).\ f\ (g\ a)) \in carrier\ ({Q_p}^m) \to carrier\ Q_p$
proof fix *a* **assume** *A*: $a \in carrier\ ({Q_p}^m)$
then have $g\ a \in carrier\ ({Q_p}^n)$
using *is-semialg-map-def*[*of m n g*] *assms*
by *blast*
then show $f\ (g\ a) \in carrier\ Q_p$
using *A assms SA-car-memE*(*3*)[*of f n*]
by *blast*
qed
show $\bigwedge x.\ x \notin carrier\ ({Q_p}^m) \Longrightarrow (\lambda a{\in}carrier\ ({Q_p}^m).\ f\ (g\ a))\ x = undefined$
unfolding *restrict-def* **by** *metis*
qed
have *0*: *is-semialg-function m* $(f \circ g)$
using *assms semialg-function-comp-closed*[*of n f m g*] *SA-car-memE*(*1*)[*of f n*]
by *blast*
have *1*: $(\bigwedge a.\ a \in carrier\ ({Q_p}^m) \Longrightarrow (f \circ g)\ a = (\lambda a{\in}carrier\ ({Q_p}^m).\ f\ (g\ a))\ a)$
using *assms comp-apply*[*of f g*] **unfolding** *restrict-def*
by *metis*
then show *is-semialg-function m* $(\lambda a{\in}carrier\ ({Q_p}^m).\ f\ (g\ a))$
using *0 1 semialg-function-on-carrier'*[*of m f* $\circ$ *g* $\lambda\ a \in carrier\ ({Q_p}^m).\ f\ (g\ a)$]
by *blast*
qed

lemma *tl-comp-in-SA*:
assumes $f \in carrier\ (SA\ n)$
shows $(\lambda\ a \in carrier\ ({Q_p}^{Suc\ n}).\ f\ (tl\ a)) \in carrier\ (SA\ (Suc\ n))$
using *assms semialg-map-comp-in-SA*[*of f - - tl*] *tl-is-semialg-map*[*of n*]
by *blast*

lemma *SA-poly-to-SA-fun-add-eval*:
assumes $f \in carrier\ (UP\ (SA\ n))$
assumes $g \in carrier\ (UP\ (SA\ n))$
assumes $a \in carrier\ ({Q_p}^{Suc\ n})$
shows *SA-poly-to-SA-fun n* $(f \oplus_{UP\ (SA\ n)} g)\ a = SA\text{-}poly\text{-}to\text{-}SA\text{-}fun\ n\ f\ a \oplus_{Q_p}$

SA-poly-to-SA-fun n g a
unfolding *SA-poly-to-SA-fun-def*
using *assms SA-poly-to-Qp-poly-add*[*of tl a n f g*]
by (*metis* (*no-types*, *lifting*) *SA-poly-to-Qp-poly-closed UPQ.to-fun-plus cartesian-power-head cartesian-power-tail restrict-apply'*)

lemma *SA-poly-to-SA-fun-add*:
assumes $f \in$ *carrier* (*UP* (*SA n*))
assumes $g \in$ *carrier* (*UP* (*SA n*))
shows *SA-poly-to-SA-fun n* (*f* $\oplus_{UP\ (SA\ n)}$ *g*) = *SA-poly-to-SA-fun n f* $\oplus_{SA\ (Suc\ n)}$ *SA-poly-to-SA-fun n g*
proof fix *x*
show *SA-poly-to-SA-fun n* (*f* $\oplus_{UP\ (SA\ n)}$ *g*) *x* = (*SA-poly-to-SA-fun n f* $\oplus_{SA\ (Suc\ n)}$ *SA-poly-to-SA-fun n g*) *x*
proof(*cases x* $\in$ *carrier* ($Q_p{}^{Suc\ n}$))
case *True*
then show *?thesis* **using** *SA-poly-to-SA-fun-add-eval*[*of f n g x*] *SA-add*[*of x n*]
using *SA-mult assms*(*1*) *assms*(*2*) *SA-add*
by *blast*
next
case *False*
have *F0*: *SA-poly-to-SA-fun n* (*f* $\oplus_{UP\ (SA\ n)}$ *g*) *x* = *undefined*
unfolding *SA-poly-to-SA-fun-def*
by (*meson False restrict-apply*)
have *F1*: (*SA-poly-to-SA-fun n f* $\oplus_{SA\ (Suc\ n)}$ *SA-poly-to-SA-fun n g*) *x* = *undefined*
using *False SA-add'* **by** *blast*
then show *?thesis*
using *F0* **by** *blast*
qed
qed

lemma *SA-poly-to-SA-fun-monom*:
assumes *f* $\in$ *carrier* (*SA n*)
assumes *a* $\in$ *carrier* ($Q_p{}^{Suc\ n}$)
shows *SA-poly-to-SA-fun n* (*up-ring.monom* (*UP* (*SA n*)) *f k*) *a* = (*f* (*tl a*))$\otimes$(*hd a*)[$\hat{}$]$_{Q_p}$ *k*
proof−
have *SA-poly-to-SA-fun n* (*up-ring.monom* (*UP* (*SA n*)) *f k*) *a* = *SA-poly-to-Qp-poly n* (*tl a*) (*up-ring.monom* (*UP* (*SA n*)) *f k*) ⋅ *lead-coeff a*
unfolding *SA-poly-to-SA-fun-def* **using** *assms*
by (*meson restrict-apply*)
then have *SA-poly-to-SA-fun n* (*up-ring.monom* (*UP* (*SA n*)) *f k*) *a* = *up-ring.monom* Q_p*-x* (*f* (*tl a*)) *k*⋅ *lead-coeff a*
using *SA-poly-to-Qp-poly-monom*[*of tl a n f k*] *assms*
by (*metis cartesian-power-tail*)
then show *?thesis* **using** *assms*

by (*metis* (*no-types*, *lifting*) *SA-car cartesian-power-head cartesian-power-tail UPQ.to-fun-monom Qp.function-ring-car-mem-closed semialg-functions-memE*(*2*))
qed

lemma *SA-poly-to-SA-fun-monom'*:
assumes $f \in$ *carrier* (*SA n*)
assumes $x \in$ *carrier* ($Q_p{}^n$)
assumes $t \in$ *carrier* Q_p
shows *SA-poly-to-SA-fun n* (*up-ring.monom* (*UP* (*SA n*)) *f k*) (*t*#*x*) = (*f x*)⊗*t*[^]$_{Q_p}$*k*
proof−
have *0*: *hd* (*t*#*x*) = *t*
by *simp*
have *1*: *tl* (*t*#*x*) = *x*
by *auto*
have *2*: (*t*#*x*) ∈ *carrier* ($Q_p{}^{Suc\ n}$)
by (*metis add.commute assms cartesian-power-cons plus-1-eq-Suc*)
show *?thesis*
using *0 1 2 SA-poly-to-SA-fun-monom*[*of f n t*#*x k*] *assms SA-poly-to-SA-fun-monom*
by *presburger*
qed

lemma *hd-is-semialg-function*:
assumes *n* > *0*
shows *is-semialg-function n hd*
proof−
have *0*: *is-semialg-function n* (λ *a* ∈ *carrier* ($Q_p{}^n$). *a*!*0*)
using *assms index-is-semialg-function restrict-is-semialg* **by** *blast*
have *1*: *restrict* (λ*a*∈*carrier* ($Q_p{}^n$). *a* ! *0*) (*carrier* ($Q_p{}^n$)) = *restrict lead-coeff* (*carrier* ($Q_p{}^n$))
proof fix *x*
show *restrict* (λ*a*∈*carrier* ($Q_p{}^n$). *a* ! *0*) (*carrier* ($Q_p{}^n$)) *x* = *restrict lead-coeff* (*carrier* ($Q_p{}^n$)) *x*
unfolding *restrict-def*
apply(*cases x* ∈ *carrier* ($Q_p{}^n$))
apply (*metis assms cartesian-power-car-memE drop-0 hd-drop-conv-nth*)
by *presburger*
qed
show *?thesis*
using *0 1 assms semialg-function-on-carrier*[*of n* (λ *a* ∈ *carrier* ($Q_p{}^n$). *a*!*0*) *hd*]
by *blast*
qed

lemma *SA-poly-to-SA-fun-monom-closed*:
assumes *f* ∈ *carrier* (*SA n*)
shows *SA-poly-to-SA-fun n* (*up-ring.monom* (*UP* (*SA n*)) *f k*) ∈ *carrier* (*SA* (*Suc n*))
proof−

have *0*: *is-semialg-function* (*Suc n*) (*f* ∘ *tl*)
using *SA-imp-semialg assms(1) semialg-function-comp-closed tl-is-semialg-map*
by *blast*
obtain *h* **where** *h-def*: *h* = *restrict hd* (*carrier* ($Q_p{}^{Suc\ n}$))
by *blast*
have *h-closed*: *h* ∈ *carrier* (*SA* (*Suc n*))
using *hd-is-semialg-function SA-car h-def restrict-in-semialg-functions*
by *blast*
have *h-pow-closed*: $h[\widehat{}]_{SA\ (Suc\ n)}\ k \in carrier\ (SA\ (Suc\ n))$
using *assms(1) h-closed monoid.nat-pow-closed*[*of SA* (*Suc n*) *h k*] *SA-is-monoid*[*of Suc n*]
by *blast*
have *monom-term-closed*: $(f \circ tl) \otimes_{SA\ (Suc\ n)} h[\widehat{}]_{SA\ (Suc\ n)}\ k \in carrier\ (SA\ (Suc\ n))$
apply(*rule SA-mult-closed-right*)
using *0* **apply** *linarith*
using *h-pow-closed* **by** *blast*

have *0*: $\bigwedge a.\ a \in carrier\ (Q_p{}^{Suc\ n}) \Longrightarrow SA\text{-}poly\text{-}to\text{-}SA\text{-}fun\ n\ (up\text{-}ring.monom\ (UP\ (SA\ n))\ f\ k)\ a = ((f \circ tl) \otimes_{SA\ (Suc\ n)} h[\widehat{}]_{SA\ (Suc\ n)}\ k)\ a$
proof− **fix** *a* **assume** *a* ∈ *carrier* ($Q_p{}^{Suc\ n}$)
then show *SA-poly-to-SA-fun n* (*up-ring.monom* (*UP* (*SA n*)) *f k*) *a* = ((*f* ∘ *tl*) $\otimes_{SA\ (Suc\ n)} h[\widehat{}]_{SA\ (Suc\ n)}\ k$) *a*
using *assms SA-poly-to-SA-fun-monom*[*of f n a k*] *comp-apply*[*of f tl a*] *h-def restrict-apply*
SA-mult[*of a Suc n f* ∘ *tl h* $[\widehat{}]_{SA\ (Suc\ n)}\ k$] *SA-nat-pow*[*of a Suc n h k*]
by *metis*
qed
have *1*: *SA-poly-to-SA-fun n* (*up-ring.monom* (*UP* (*SA n*)) *f k*) = ((*f* ∘ *tl*) $\otimes_{SA\ (Suc\ n)} h[\widehat{}]_{SA\ (Suc\ n)}\ k$)
proof fix *x*
show *SA-poly-to-SA-fun n* (*up-ring.monom* (*UP* (*SA n*)) *f k*) *x* = ((*f* ∘ *tl*) $\otimes_{SA\ (Suc\ n)} h\ [\widehat{}]_{SA\ (Suc\ n)}\ k$) *x*
apply(*cases x* ∈ *carrier* ($Q_p{}^{Suc\ n}$))
using *0* **apply** *blast*
using *monom-term-closed* **unfolding** *SA-poly-to-SA-fun-def*
using *restrict-apply*
by (*metis* (*no-types*, *lifting*) *SA-mult'*)
qed
show *?thesis*
using *1 monom-term-closed*
by *presburger*
qed

lemma *SA-poly-to-SA-fun-is-SA*:
assumes *P* ∈ *carrier* (*UP* (*SA n*))
shows *SA-poly-to-SA-fun n P* ∈ *carrier* (*SA* (*Suc n*))
apply(*rule UP-ring.poly-induct3*[*of SA n P*])

unfolding *UP-ring-def* **using** *assms SA-is-ring* **apply** *blast*
using *assms* **apply** *blast*
using *assms SA-poly-to-SA-fun-add*[*of*]
using *SA-add-closed-right SA-imp-semialg zero-less-Suc* **apply** *presburger*
using *SA-poly-to-SA-fun-monom-closed assms*(*1*)
by *blast*

lemma *SA-poly-to-SA-fun-mult*:
assumes $f \in$ *carrier* (*UP* (*SA n*))
assumes $g \in$ *carrier* (*UP* (*SA n*))
shows *SA-poly-to-SA-fun n* (*f* $\otimes_{UP\ (SA\ n)}$ *g*) = *SA-poly-to-SA-fun n f* $\otimes_{SA\ (Suc\ n)}$ *SA-poly-to-SA-fun n g*
proof(*rule function-ring-car-eqI*[*of* - *Suc n*])
show *SA-poly-to-SA-fun n* (*f* $\otimes_{UP\ (SA\ n)}$ *g*) $\in$ *carrier* (*function-ring* (*carrier* ($Q_p{}^{Suc\ n}$)) Q_p)
proof−
have *f* $\otimes_{UP\ (SA\ n)}$ *g* $\in$ *carrier* (*UP* (*SA n*))
using *assms SA-is-UP-cring*
by (*meson cring.cring-simprules*(*5*) *padic-fields.UP-SA-n-is-cring padic-fields-axioms*)
thus *?thesis*
using *SA-is-UP-cring assms SA-poly-to-SA-fun-is-SA*[*of f* $\otimes_{UP\ (SA\ n)}$ *g*] *SA-car-in-Qp-funs-car*[*of Suc n*]
by *blast*
qed
show *SA-poly-to-SA-fun n f* $\otimes_{SA\ (Suc\ n)}$ *SA-poly-to-SA-fun n g* $\in$ *carrier* (*function-ring* (*carrier* ($Q_p{}^{Suc\ n}$)) Q_p)
using *SA-is-UP-cring SA-poly-to-SA-fun-is-SA assms*
by (*meson SA-car-memE*(*2*) *SA-mult-closed-left less-Suc-eq-0-disj padic-fields.SA-car-memE*(*1*) *padic-fields-axioms*)
show $\bigwedge a.$ *a* $\in$ *carrier* ($Q_p{}^{Suc\ n}$) $\Longrightarrow$ *SA-poly-to-SA-fun n* (*f* $\otimes_{UP\ (SA\ n)}$ *g*) *a* = (*SA-poly-to-SA-fun n f* $\otimes_{SA\ (Suc\ n)}$ *SA-poly-to-SA-fun n g*) *a*
proof− **fix** *a* **assume** *A*: *a* $\in$ *carrier* ($Q_p{}^{Suc\ n}$)
then obtain *t b* **where** *tb-def*: *a* = *t*#*b*
using *cartesian-power-car-memE*[*of a* Q_p *Suc n*] **by** (*meson length-Suc-conv*)
have *t-closed*: *t* $\in$ *carrier* Q_p
using *tb-def A cartesian-power-head*[*of a* Q_p *n*] **by** (*metis list.sel*(*1*))
have *b-closed*: *b* $\in$ *carrier* ($Q_p{}^{n}$)
using *tb-def A cartesian-power-tail*[*of a* Q_p *n*] **by** (*metis list.sel*(*3*))
have *0*: *f* $\otimes_{UP\ (SA\ n)}$ *g* $\in$ *carrier* (*UP* (*SA n*))
using *assms* **by** (*meson UP-SA-n-is-cring cring.cring-simprules*(*5*))
have *1*: *SA-poly-to-SA-fun n* (*f* $\otimes_{UP\ (SA\ n)}$ *g*) *a* = (*SA-poly-to-Qp-poly n b* (*f* $\otimes_{UP\ (SA\ n)}$ *g*))$\bullet$*t*
using *SA-poly-to-SA-fun-formula*[*of f* $\otimes_{UP\ (SA\ n)}$ *g n b t*] *t-closed b-closed tb-def 0 assms*(*1*)
by *blast*
have *2*: (*SA-poly-to-SA-fun n f* $\otimes_{SA\ (Suc\ n)}$ *SA-poly-to-SA-fun n g*) *a* =
SA-poly-to-SA-fun n f a $\otimes$ *SA-poly-to-SA-fun n g a*

using *SA-poly-to-SA-fun-is-fun assms A SA-mult* **by** *blast*
hence *3*: $(SA\text{-}poly\text{-}to\text{-}SA\text{-}fun\ n\ f \otimes_{SA\ (Suc\ n)} SA\text{-}poly\text{-}to\text{-}SA\text{-}fun\ n\ g)\ a =$
$((SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ b\ f) \cdot t) \otimes ((SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ b\ g) \cdot t)$
using *SA-poly-to-SA-fun-formula assms*
by (*metis b-closed t-closed tb-def*)
have *4*: $SA\text{-}poly\text{-}to\text{-}SA\text{-}fun\ n\ (f \otimes_{UP\ (SA\ n)} g)\ a = ((SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ b\ f) \cdot t) \otimes ((SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ b\ g) \cdot t)$
using *1 assms SA-poly-to-Qp-poly-closed*[*of b*] *SA-poly-to-Qp-poly-mult UPQ.to-fun-mult b-closed t-closed*
by *presburger*
show $SA\text{-}poly\text{-}to\text{-}SA\text{-}fun\ n\ (f \otimes_{UP\ (SA\ n)} g)\ a = (SA\text{-}poly\text{-}to\text{-}SA\text{-}fun\ n\ f \otimes_{SA\ (Suc\ n)} SA\text{-}poly\text{-}to\text{-}SA\text{-}fun\ n\ g)\ a$
using *3 4* **by** *blast*
qed
qed

lemma *SA-poly-to-SA-fun-one*:
shows $SA\text{-}poly\text{-}to\text{-}SA\text{-}fun\ n\ (\mathbf{1}_{UP\ (SA\ n)}) = \mathbf{1}_{SA\ (Suc\ n)}$
proof(*rule function-ring-car-eqI*[*of - Suc n*])
have $\mathbf{1}_{UP\ (SA\ n)} \in carrier\ (UP\ (SA\ n))$
using *UP-SA-n-is-cring cring.cring-simprules*(*6*) **by** *blast*
thus $SA\text{-}poly\text{-}to\text{-}SA\text{-}fun\ n\ \mathbf{1}_{UP\ (SA\ n)} \in carrier\ (function\text{-}ring\ (carrier\ ({Q_p}^{Suc\ n}))\ Q_p)$
using *SA-poly-to-SA-fun-is-SA*[*of* $\mathbf{1}_{UP\ (SA\ n)}$] *SA-car-in-Qp-funs-car*[*of Suc n*] *SA-is-UP-cring*[*of n*]
by *blast*
show $\mathbf{1}_{SA\ (Suc\ n)} \in carrier\ (function\text{-}ring\ (carrier\ ({Q_p}^{Suc\ n}))\ Q_p)$
unfolding *SA-one*
using *function-one-closed* **by** *blast*
show $\bigwedge a.\ a \in carrier\ ({Q_p}^{Suc\ n}) \Longrightarrow SA\text{-}poly\text{-}to\text{-}SA\text{-}fun\ n\ \mathbf{1}_{UP\ (SA\ n)}\ a = \mathbf{1}_{SA\ (Suc\ n)}\ a$
proof− **fix** *a* **assume** *A*: $a \in carrier\ ({Q_p}^{Suc\ n})$
then obtain *t b* **where** *tb-def*: $a = t\#b$
using *cartesian-power-car-memE*[*of a* Q_p *Suc n*] **by** (*meson length-Suc-conv*)
have *t-closed*: $t \in carrier\ Q_p$
using *tb-def A cartesian-power-head*[*of a* Q_p *n*] **by** (*metis list.sel*(*1*))
have *b-closed*: $b \in carrier\ ({Q_p}^n)$
using *tb-def A cartesian-power-tail*[*of a* Q_p *n*] **by** (*metis list.sel*(*3*))
have *0*: $SA\text{-}poly\text{-}to\text{-}SA\text{-}fun\ n\ \mathbf{1}_{UP\ (SA\ n)}\ a = (SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ b\ \mathbf{1}_{UP\ (SA\ n)}) \cdot t$
using *SA-poly-to-SA-fun-formula* ‹$\mathbf{1}_{UP\ (SA\ n)} \in carrier\ (UP\ (SA\ n))$› *b-closed t-closed tb-def*
by *blast*
have *1*: $SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ b\ \mathbf{1}_{UP\ (SA\ n)} = \mathbf{1}_{UP\ Q_p}$
using *SA-poly-to-Qp-poly-is-hom*[*of b*] *ring-hom-one*[*of - UP* (*SA n*) *UP* Q_p] *b-closed*
by *blast*
thus $SA\text{-}poly\text{-}to\text{-}SA\text{-}fun\ n\ \mathbf{1}_{UP\ (SA\ n)}\ a = \mathbf{1}_{SA\ (Suc\ n)}\ a$

using *0 A function-one-eval SA-one UPQ.to-fun-one t-closed* **by** *presburger*
qed
qed

lemma *SA-poly-to-SA-fun-ring-hom*:
shows *SA-poly-to-SA-fun n ∈ ring-hom (UP (SA n)) (SA (Suc n))*
apply(*rule ring-hom-memI*)
using *SA-poly-to-SA-fun-is-SA* **apply** *blast*
apply (*meson SA-poly-to-SA-fun-mult*)
apply (*meson SA-poly-to-SA-fun-add*)
by (*meson SA-poly-to-SA-fun-one*)

lemma *SA-poly-to-SA-fun-taylor-term*:
assumes $F \in$ *carrier* (*UP* (*SA n*))
assumes $c \in$ *carrier* (*SA n*)
assumes $x \in$ *carrier* ($Q_p{}^n$)
assumes $t \in$ *carrier* Q_p
assumes $f =$ *SA-poly-to-Qp-poly n x F*
shows *SA-poly-to-SA-fun n* (*UP-cring.taylor-term* (*SA n*) *c F k*) (*t*#*x*) = (*taylor-expansion* Q_p (*c x*) *f k*) ⊗($t \ominus$ *c x*)[⌝]$_{Q_p}$ *k*
proof−
have *0*: *hd* (*t*#*x*) = *t*
by *simp*
have *1*: *tl* (*t*#*x*) = *x*
by *auto*
have *2*: (*t*#*x*) ∈ *carrier* ($Q_p{}^{Suc\ n}$)
by (*metis Suc-eq-plus1 assms cartesian-power-cons*)
show *?thesis*
using *assms 0 1 2 Pi-iff SA-car-memE*(*3*)
SA-poly-to-Qp-poly-closed SA-poly-to-Qp-poly-comm-taylor-term[*of F n c x k*] *restrict-apply'*
unfolding *SA-poly-to-SA-fun-def*
by (*metis* (*no-types*, *lifting*) *UPQ.taylor-def UPQ.to-fun-taylor-term*)
qed

lemma *SA-finsum-eval*:
assumes *finite I*
assumes $F \in I \to$ *carrier* (*SA m*)
assumes $x \in$ *carrier* ($Q_p{}^m$)
shows ($\bigoplus_{SA\ m} i \in I.\ F\ i$) x = ($\bigoplus i \in I.\ F\ i\ x$)
proof−
have $F \in I \to$ *carrier* (*SA m*) ⟶ ($\bigoplus_{SA\ m} i \in I.\ F\ i$) x = ($\bigoplus i \in I.\ F\ i\ x$)
apply(*rule finite.induct*[*of I*])
apply (*simp add*: *assms*(*1*))
using *abelian-monoid.finsum-empty*[*of SA m F*] *assms* **unfolding** *Qp.finsum-empty*
using *SA-is-abelian-monoid SA-zeroE* **apply** *presburger*
proof fix *A a* **assume** *IH*: *finite A* $F \in A \to$ *carrier* (*SA m*) ⟶ *finsum* (*SA m*) *F A x* = ($\bigoplus i \in A.\ F\ i\ x$)
$F \in$ *insert a A* $\to$ *carrier* (*SA m*)

```
    then have 0: F ∈ A → carrier (SA m)
      by blast
    then have 1: finsum (SA m) F A x = (⨁ i∈A. F i x)
      using IH by blast
    show finsum (SA m) F (insert a A) x = (⨁ i∈insert a A. F i x)
    proof(cases a ∈ A)
      case True
      then show ?thesis using insert-absorb[of a A] IH
        by presburger
    next
      case False
      have F0: (λi. F i x) ∈ A → carrier Q_p proof fix i assume i ∈ A thus F i
x ∈ carrier Q_p
        using 0 assms(3) SA-car-memE(3)[of F i m] by blast qed
      have F1: F a x ∈ carrier Q_p
        using IH assms(3) SA-car-memE(3)[of F a m] by blast
      have F2: finsum (SA m) F (insert a A) x = F a x ⊕ finsum (SA m) F A x
      proof−
        have  finsum (SA m) F (insert a A) = F a ⊕_SA m finsum (SA m) F A
          using False IH abelian-monoid.finsum-insert[of SA m A a F ] 0
          by (meson Pi-split-insert-domain SA-is-abelian-monoid)
        thus ?thesis using    abelian-monoid.finsum-closed[of SA m F A] 1 F0  F1
            SA-add[of x m F a finsum (SA m) F A]
          using assms(3) by presburger
      qed
     show ?thesis using False 0 IH  Qp.finsum-insert[of A a λi. F i x] unfolding
F2 1
        using F0 F1 by blast
    qed
  qed
  thus ?thesis using assms by blast
qed

lemma(in ring) finsum-ring-hom:
  assumes ring S
  assumes h ∈ ring-hom R S
  assumes F ∈ I → carrier R
  assumes finite I
  shows h (⨁ i∈I. F i) = (⨁ _S i∈I. h (F i))
proof−
  have F ∈ I → carrier R ⟶ h (⨁ i∈I. F i) = (⨁ _S i∈I. h (F i))
    apply(rule finite.induct[of I])
      apply (simp add: assms(4))
    unfolding finsum-empty using assms abelian-monoid.finsum-empty[of S]
    unfolding ring-def abelian-group-def
     apply (simp add: ‹⋀f. abelian-monoid S ⟹ finsum S f {} = 𝟬_S› assms(1)
local.ring-axioms ring-hom-zero)
  proof fix A a assume A: finite A   F ∈ A → carrier R ⟶ h (finsum R F A)
= (⨁ _S i∈A. h (F i))
```

```
        F ∈ insert a A → carrier R
    then have 0: F ∈ A → carrier R
      by blast
    have 1: h (finsum R F A) = (⨁_S i∈A. h (F i))
      using 0 A(2) by linarith
    have 2: (finsum R F A) ∈ carrier R
      using finsum-closed[of F A] 0 by blast
    have 3: (⨁_S i∈A. h (F i)) ∈ carrier S
    using assms 1 2 ring-hom-closed
    by metis
    have 4: F a ∈ carrier R using A by blast
    have 5: h (F a) ∈ carrier S
      using assms 4
      by (meson ring-hom-closed)
    show h (finsum R F (insert a A)) = (⨁_S i∈insert a A. h (F i))
      apply(cases a ∈ A)
      using insert-absorb 1
       apply metis
    proof− assume C: a ∉A
      have 6: finsum R F (insert a A) = F a ⊕ finsum R F A
      using A finsum-insert[of A a F]  C by blast
    have 7: (⨁_S i∈insert a A. h (F i)) = h (F a) ⊕_S (⨁_S i∈A. h (F i))
    apply(rule abelian-monoid.finsum-insert[of S A a λi. h (F i)])
         apply (simp add: abelian-group.axioms(1) assms(1) ring.is-abelian-group)
        apply (simp add: A(1))
       apply (simp add: C)
      apply(intro Pi-I ring-hom-closed[of h R S] assms)
      using 0 A 5 assms by auto
    thus ?thesis
    using    0 1 2 3 4 5 6  7 assms ring-hom-add[of h R S F a finsum R F A ]

      unfolding 1 ring-def abelian-group-def
      by presburger
  qed
  qed
  thus ?thesis using assms by auto
qed

lemma SA-poly-to-SA-fun-finsum:
  assumes finite I
  assumes F ∈ I → carrier (UP (SA m))
  assumes f = (⨁_{UP (SA m)} i∈I. F i)
  assumes x ∈ carrier (Q_p^{Suc m})
  shows SA-poly-to-SA-fun m f x = (⨁ i∈I. SA-poly-to-SA-fun m (F i) x)
proof−
  have SA-poly-to-SA-fun m ∈ ring-hom (UP (SA m)) (SA (Suc m))
    using SA-poly-to-SA-fun-ring-hom by blast
  have f-closed: f ∈ carrier (UP (SA m))
    unfolding assms apply(rule finsum-closed)  using assms by blast
```

have *0*: *SA-poly-to-SA-fun m f* = $(\bigoplus_{SA\ (Suc\ m)} i \in I.$ *SA-poly-to-SA-fun m* (*F i*))
unfolding *assms*
apply(*rule finsum-ring-hom*)
apply (*simp add*: *SA-is-ring*)
using ‹*SA-poly-to-SA-fun m* ∈ *ring-hom* (*UP* (*SA m*)) (*SA* (*Suc m*))› **apply** *blast*
using *assms*(*2*) **apply** *blast*
by (*simp add*: *assms*(*1*))
show *?thesis* **unfolding** *0* **apply**(*rule SA-finsum-eval*)
using *assms* **apply** *blast* **using** *assms*
apply (*meson Pi-I Pi-mem padic-fields.SA-poly-to-SA-fun-is-SA padic-fields-axioms*)
using *assms* **by** *blast*
qed

lemma *SA-poly-to-SA-fun-taylor-expansion*:
assumes *f* ∈ *carrier* (*UP* (*SA m*))
assumes *c* ∈ *carrier* (*SA m*)
assumes *x* ∈ *carrier* ($Q_p{}^{Suc\ m}$)
shows *SA-poly-to-SA-fun m f x* = $(\bigoplus i \in \{..deg\ (SA\ m)\ f\}.$ *taylor-expansion* (*SA m*) *c f i* (*tl x*) ⊗ (*hd x* ⊖ *c* (*tl x*)) [^] *i*)
proof−
have *0*: *f* = $(\bigoplus_{UP\ (SA\ m)} i \in \{..deg\ (SA\ m)\ f\}.$ *UP-cring.taylor-term* (*SA m*) *c f i*)
using *taylor-sum*[*of f m deg* (*SA m*) *f c*] *assms* **unfolding** *UPSA.taylor-term-def UPSA.taylor-def*
by *blast*
have *1*: *SA-poly-to-SA-fun m f x* = $(\bigoplus i \in \{..deg\ (SA\ m)\ f\}.$ *SA-poly-to-SA-fun m* (*UP-cring.taylor-term* (*SA m*) *c f i*) *x*)
apply(*rule SA-poly-to-SA-fun-finsum*)
apply *simp*
apply (*meson Pi-I UPSA.taylor-term-closed assms*(*1*) *assms*(*2*))
using *0* **apply** *blast*
using *assms* **by** *blast*
have *hd-closed*: *hd x* ∈ *carrier* Q_p
using *assms cartesian-power-head* **by** *blast*
have *tl-closed*: *tl x* ∈ *carrier* ($Q_p{}^m$)
using *assms cartesian-power-tail* **by** *blast*
obtain *t a* **where** *ta-def*: *x* = *t*#*a*
using *assms cartesian-power-car-memE*[*of x* Q_p *Suc m*]
by (*metis Suc-length-conv*)
have *2*: *t* = *hd x*
by (*simp add*: *ta-def*)
have *3*: *a* = *tl x*
by (*simp add*: *ta-def*)
have *4*: ⋀*i*. *SA-poly-to-SA-fun m* (*UPSA.taylor-term m c f i*) *x* =
taylor-expansion Q_p (*c* (*tl x*)) (*SA-poly-to-Qp-poly m* (*tl x*) *f*) *i* ⊗ (*lead-coeff x* ⊖ *c* (*tl x*)) [^] *i*
using *tl-closed hd-closed assms SA-poly-to-SA-fun-taylor-term*[*of f m c a t*

SA-poly-to-Qp-poly m a f]
unfolding *2 3* **by** (*metis 2 3 ta-def*)
have *5*: *SA-poly-to-SA-fun m f x* = ($\bigoplus$ *i*∈{..*deg* (*SA m*) *f*}. (*taylor-expansion* Q_p (*c* (*tl x*)) (*SA-poly-to-Qp-poly m* (*tl x*) *f*) *i*) ⊗((*hd x*) ⊖ *c* (*tl x*))[^]$_{Q_p}$ *i*)
using *1 2* **unfolding** *4* **by** *blast*
have *6*: *taylor-expansion* Q_p (*c* (*tl x*)) (*SA-poly-to-Qp-poly m* (*tl x*) *f*) = *SA-poly-to-Qp-poly m* (*tl x*) (*taylor-expansion* (*SA m*) *c f*)

using *SA-poly-to-Qp-poly-taylor-poly*[*of f m c tl x*] *assms*(*1*) *assms*(*2*) *tl-closed* **by** *presburger*
have *7*: ⋀*i*. *taylor-expansion* Q_p (*c* (*tl x*)) (*SA-poly-to-Qp-poly m* (*tl x*) *f*) *i* = *taylor-expansion* (*SA m*) *c f i* (*tl x*)
unfolding *6* **using** *SA-poly-to-Qp-poly-coeff*[*of tl x m taylor-expansion* (*SA m*) *c f*]
by (*metis UPSA.taylor-closed UPSA.taylor-def assms*(*1*) *assms*(*2*) *tl-closed*)
show *?thesis* **using** *5* **unfolding** *7* **by** *blast*
qed

lemma *SA-deg-one-eval*:
assumes *g* ∈ *carrier* (*UP* (*SA m*))
assumes *deg* (*SA m*) *g* = *1*
assumes *ξ* ∈ *carrier* (*Fun*$_m$ Q_p)
assumes *UP-ring.lcf* (*SA m*) *g* ∈ *Units* (*SA m*)
assumes ∀ *x* ∈ *carrier* ($Q_p{}^m$). (*SA-poly-to-SA-fun m g*) (*ξ x*#*x*) = **0**
shows *ξ* = ⊖$_{SA\ m}$(*g 0*)⊗$_{SA\ m}$ (*inv*$_{SA\ m}$ (*g 1*))
proof(*rule ext*)
fix *x* **show** *ξ x* = (⊖$_{SA\ m}$ *g 0* ⊗$_{SA\ m}$ *inv*$_{SA\ m}$ *g 1*) *x*
proof(*cases x* ∈ *carrier* ($Q_p{}^m$))
case *True*
then have (*SA-poly-to-SA-fun m g*) (*ξ x*#*x*) = **0**
using *assms* **by** *blast*
then have *T0*: *SA-poly-to-Qp-poly m x g* ⋅ *ξ x* = **0**
using *SA-poly-to-SA-fun-formula*[*of g m x ξ x*] *assms Qp.function-ring-car-mem-closed True* **by** *metis*
have *deg* Q_p (*SA-poly-to-Qp-poly m x g*) = *1*
proof(*rule UPQ.deg-eqI*)
show *SA-poly-to-Qp-poly m x g* ∈ *carrier* (*UP* Q_p)
using *assms True SA-poly-to-Qp-poly-closed* **by** *blast*
show *deg* Q_p (*SA-poly-to-Qp-poly m x g*) ≤ *1*
using *SA-poly-to-Qp-poly-deg-bound* **by** (*metis True assms*(*1*) *assms*(*2*))
show *SA-poly-to-Qp-poly m x g 1* ≠ **0**
using *SA-poly-to-Qp-poly-coeff*[*of x m g 1*] *assms SA-Units-memE′ True* **by** *presburger*
qed
hence *T1*: *SA-poly-to-Qp-poly m x g* ⋅ *ξ x* = *SA-poly-to-Qp-poly m x g 0* ⊕ *SA-poly-to-Qp-poly m x g 1* ⊗ (*ξ x*)
using *UP-cring.deg-one-eval*[*of* Q_p *- ξ x*]
by (*meson Qp.function-ring-car-mem-closed SA-poly-to-Qp-poly-closed True UPQ.deg-one-eval assms*)

hence *T2*: $g\ 0\ x \oplus g\ 1\ x \otimes (\xi\ x) = \mathbf{0}$
using *True T0 assms SA-poly-to-Qp-poly-coeff*[*of x m g*]
by *metis*
have *T3*: $g\ 0\ x \in carrier\ Q_p$
using *True assms UP-ring.cfs-closed*
by (*metis SA-poly-to-Qp-poly-closed SA-poly-to-Qp-poly-coeff UPQ.is-UP-ring*)
have *T4*: $\xi\ x \in carrier\ Q_p$
using *True assms Qp.function-ring-car-memE* **by** *blast*
have *T5*: $g\ 1\ x \in nonzero\ Q_p$
using *True assms*
by (*metis Qp.function-ring-car-memE SA-Units-closed SA-Units-memE' SA-car-memE*(*2*) *not-nonzero-Qp*)
have *T6*: $\ominus\ (g\ 0\ x) = g\ 1\ x \otimes (\xi\ x)$
using *T2 T3 T4 T5* **by** (*metis Qp.m-closed Qp.minus-equality Qp.minus-minus Qp.nonzero-closed*)
hence *T7*: $\ominus\ (g\ 0\ x) \otimes inv\ (g\ 1\ x) = \xi\ x$
using *T5* **by** (*metis Qp.inv-cancelR*(*2*) *Qp.m-closed Qp.nonzero-closed T4 Units-eq-nonzero*)
have *T8*: $(inv_{SA\ m}\ g\ 1)\ x = inv\ (g\ 1\ x)$
using *assms True Qp-funs-m-inv SA-Units-Qp-funs-Units SA-Units-Qp-funs-inv* **by** *presburger*
have *T9*: $(\ominus_{SA\ m}\ g\ 0)\ x = \ominus\ (g\ 0\ x)$
using *SA-a-inv-eval*[*of g 0 m x*] *UP-ring.cfs-closed*[*of SA m g 0*] *assms True SA-is-ring*
unfolding *UP-ring-def* **by** *blast*
have *T11*: $((\ominus_{SA\ m}\ g\ 0) \otimes_{SA\ m} inv_{SA\ m}\ g\ 1)\ x = \ominus\ (g\ 0\ x) \otimes inv\ (g\ 1\ x)$
using *assms UP-ring.cfs-closed T8 T9 T7 True SA-mult* **by** *presburger*
thus $\xi\ x = (\ominus_{SA\ m}\ g\ 0 \otimes_{SA\ m} inv_{SA\ m}\ g\ 1)\ x$
using *T7* **by** *blast*
next
case *False*
then show *?thesis*
using *SA-mult' SA-times assms function-ring-not-car* **by** *auto*
qed
qed

lemma *SA-deg-one-eval'*:
assumes $g \in carrier\ (UP\ (SA\ m))$
assumes $deg\ (SA\ m)\ g = 1$
assumes $\xi \in carrier\ (Fun_m\ Q_p)$
assumes *UP-ring.lcf* $(SA\ m)\ g \in Units\ (SA\ m)$
assumes $\forall x \in carrier\ ({Q_p}^m).\ (SA\text{-}poly\text{-}to\text{-}SA\text{-}fun\ m\ g)\ (\xi\ x\#x) = \mathbf{0}$
shows $\xi \in carrier\ (SA\ m)$
proof−
have *0*: $\xi = \ominus_{SA\ m}(g\ 0) \otimes_{SA\ m} (inv_{SA\ m}\ (g\ 1))$
using *assms SA-deg-one-eval* **by** *blast*
have *1*: $(inv_{SA\ m}\ (g\ 1)) \in carrier\ (SA\ m)$
using *assms SA-Units-inv-closed* **by** *presburger*
have $(g\ 0) \in carrier\ (SA\ m)$

using *assms(1) assms(2) UP-ring.cfs-closed*[*of SA m g 0*] *SA-is-ring*[*of m*]
unfolding *UP-ring-def*
by *blast*
hence *2*: $\ominus_{SA\ m}(g\ 0) \in carrier\ (SA\ m)$
by (*meson SA-is-cring assms(1) cring.cring-simprules(3)*)
show *?thesis*
using *0 1 2 SA-imp-semialg SA-mult-closed-left assms(1)* **by** *blast*
qed

lemma *Qp-pow-ConsI*:
assumes $t \in carrier\ Q_p$
assumes $x \in carrier\ ({Q_p}^m)$
shows $t\#x \in carrier\ ({Q_p}^{Suc\ m})$
using *assms cartesian-power-cons*[*of x* Q_p *m t*] *Suc-eq-plus1* **by** *presburger*

lemma *Qp-pow-ConsE*:
assumes $x \in carrier\ ({Q_p}^{Suc\ m})$
shows $tl\ x \in carrier\ ({Q_p}^m)$
$hd\ x \in carrier\ Q_p$
using *assms cartesian-power-tail* **apply** *blast*
using *assms cartesian-power-head* **by** *blast*

lemma(**in** *ring*) *add-monoid-one*:
$\mathbf{1}_{add\text{-}monoid\ R} = \mathbf{0}$
by *simp*

lemma(**in** *ring*) *add-monoid-carrier*:
$carrier\ (add\text{-}monoid\ R) = carrier\ R$
unfolding *Congruence.partial-object.simps(1)*
by *simp*

lemma(**in** *ring*) *finsum-mono-neutral-cong*:
assumes $F \in I \to carrier\ R$
assumes *finite I*
assumes $\bigwedge i.\ i \notin J \Longrightarrow F\ i = \mathbf{0}$
assumes $J \subseteq I$
shows *finsum R F I* = *finsum R F J*
unfolding *finsum-def* **apply**(*rule comm-monoid.finprod-mono-neutral-cong*)
using *local.add.comm-monoid-axioms* **apply** *blast*
using *assms(2) assms(4) rev-finite-subset* **apply** *blast*
apply (*simp add*: *assms(2)*)
using *assms(4)* **apply** *blast*
unfolding *add-monoid-one* **using** *assms* **apply** *blast*
apply *blast*
using *add-monoid-carrier assms(1)* **apply** *blast*
using *add-monoid-carrier assms* **by** *blast*

This lemma helps to formalize statements like "by passing to a partition, we can assume the Taylor coefficients are either always zero or never zero"

lemma *SA-poly-to-SA-fun-taylor-on-refined-set*:
assumes $f \in$ *carrier* (*UP* (*SA m*))
assumes $c \in$ *carrier* (*SA m*)
assumes *is-semialgebraic m A*
assumes $\bigwedge i$. $A \subseteq$ *SA-zero-set m* (*taylor-expansion* (*SA m*) *c f i*) $\vee$ $A \subseteq$ *SA-nonzero-set m* (*taylor-expansion* (*SA m*) *c f i*)
assumes $a =$ *to-fun-unit m* $\circ$ *taylor-expansion* (*SA m*) *c f*
assumes *inds* $= \{i.\ i \leq$ *deg* (*SA m*) *f* $\wedge$ $A \subseteq$ *SA-nonzero-set m* (*taylor-expansion* (*SA m*) *c f i*)$\}$
assumes $x \in A$
assumes $t \in$ *carrier* Q_p
shows *SA-poly-to-SA-fun m f* (*t#x*) $= (\bigoplus i \in$ *inds*. (*a i x*)$\otimes$(*t* $\ominus$ *c x*)[$\hat{}$]*i*)
proof−
have *x-closed*: $x \in$ *carrier* (${Q_p}^m$)
using *assms*(*3*) *assms*(*7*)
by (*metis* (*no-types*, *lifting*) *Diff-eq-empty-iff Diff-iff empty-iff is-semialgebraic-closed*)
have *tx-closed*: *t#x* $\in$ *carrier* (${Q_p}^{Suc\ m}$)
using *x-closed assms*(*8*) *cartesian-power-cons*[*of x* Q_p *m t*] *Qp-pow-ConsI* **by** *blast*
have *SA-poly-to-SA-fun m f* (*t # x*) =
($\bigoplus i \in \{..deg\ (SA\ m)\ f\}$. *taylor-expansion* (*SA m*) *c f i* (*tl* (*t # x*)) $\otimes$ (*hd* (*t # x*) $\ominus$ *c* (*tl* (*t # x*))) [$\hat{}$] *i*)
using *tx-closed assms SA-poly-to-SA-fun-taylor-expansion*[*of f m c t#x*]
by *linarith*
then have *0*: *SA-poly-to-SA-fun m f* (*t#x*) $= (\bigoplus i \in \{..deg\ (SA\ m)\ f\}$. *taylor-expansion* (*SA m*) *c f i x* $\otimes$ (*t* $\ominus$ *c x*) [$\hat{}$] *i*)
unfolding *list-tl list-hd* **by** *blast*
have *1*: $\bigwedge i$. $i \notin$ *inds* $\Longrightarrow$ *taylor-expansion* (*SA m*) *c f i x* $= \mathbf{0}$
proof− **fix** *i* **assume** *A*: $i \notin$ *inds*
show *taylor-expansion* (*SA m*) *c f i x* $= \mathbf{0}$
proof(*cases* $i \leq$ *deg* (*SA m*) *f*)
case *True*
then have $A \subseteq \{x \in$ *carrier* (${Q_p}^m$). *taylor-expansion* (*SA m*) *c f i x* $= \mathbf{0}\}$
using *A assms*(*8*) *assms*(*4*)[*of i*] **unfolding** *assms mem-Collect-eq SA-zero-set-def*
by *linarith*
thus *?thesis* **using** *x-closed assms* **by** *blast*
next
case *False*
then have *taylor-expansion* (*SA m*) *c f i* $= \mathbf{0}_{SA\ m}$
using *assms taylor-deg*[*of c m f*] **unfolding** *UPSA.taylor-def*
by (*metis* (*no-types*, *lifting*) *UPSA.taylor-closed UPSA.taylor-def UPSA.deg-eqI nat-le-linear*)
then show *?thesis*
using *x-closed SA-car-memE SA-zeroE* **by** *presburger*
qed
qed
hence *2*: $\bigwedge i$. $i \notin$ *inds* $\Longrightarrow$ *taylor-expansion* (*SA m*) *c f i x* $\otimes$ (*t* $\ominus$ *c x*) [$\hat{}$] *i* $= \mathbf{0}$
using *assms x-closed SA-car-memE*
by (*metis* (*no-types*, *lifting*) *Qp.cring-simprules*(*26*) *Qp.function-ring-car-memE*

Qp.minus-closed Qp.nat-pow-closed)
have *3*: $(\lambda i.\ \textit{taylor-expansion}\ (SA\ m)\ c\ f\ i\ x \otimes (t \ominus c\ x)\ [\widehat{}]\ i) \in \{..\textit{deg}\ (SA\ m)\ f\} \to \textit{carrier}\ Q_p$
proof fix *i* **assume** *A*: $i \in \{..\textit{deg}\ (SA\ m)\ f\}$
have *30*: *taylor-expansion* $(SA\ m)\ c\ f\ i \in \textit{carrier}\ (SA\ m)$
using *assms UPSA.taylor-closed*[*of f m c*] **unfolding** *UPSA.taylor-def*
using *UPSA.UP-car-memE*(*1*) **by** *blast*
hence *31*: *taylor-expansion* $(SA\ m)\ c\ f\ i\ x \in \textit{carrier}\ Q_p$
using *x-closed function-ring-car-closed SA-car-memE*(*2*) **by** *blast*
have *32*: $c\ x \in \textit{carrier}\ Q_p$
using *assms x-closed SA-car-memE*(*3*) **by** *blast*
hence *33*: $(t \ominus c\ x)[\widehat{}]\ i \in \textit{carrier}\ Q_p$
using *assms* **by** *blast*
show *taylor-expansion* $(SA\ m)\ c\ f\ i\ x \otimes (t \ominus c\ x)\ [\widehat{}]\ i \in \textit{carrier}\ Q_p$
using *30 31 32 33* **by** *blast*
qed
have *4*: *SA-poly-to-SA-fun m f* $(t\#x) = (\bigoplus i \in \textit{inds}.\ \textit{taylor-expansion}\ (SA\ m)\ c\ f\ i\ x \otimes (t \ominus c\ x)\ [\widehat{}]\ i)$
unfolding *0* **apply**(*rule Qp.finsum-mono-neutral-cong*)
using *assms UPSA.taylor-closed*[*of f m c*] **unfolding** *UPSA.taylor-def*
using *3* **apply** *blast*
apply *blast*
using *2* **apply** *blast*
unfolding *assms* **by** *blast*
have *A*: $\bigwedge i.\ i \in \textit{inds} \Longrightarrow a\ i\ x = \textit{taylor-expansion}\ (SA\ m)\ c\ f\ i\ x$
unfolding *assms mem-Collect-eq SA-nonzero-set-def comp-apply*
apply(*rule to-fun-unit-eq*[*of - m x*])
using *UPSA.taylor-closed*[*of f m c*] *assms* **unfolding** *UPSA.taylor-def*
using *UPSA.UP-car-memE*(*1*) **apply** *blast*
using *x-closed* **apply** *blast*
using *assms*(*7*) **by** *blast*
have *a-closed*: $a \in \textit{UNIV} \to \textit{carrier}\ (SA\ m)$
apply(*rule Pi-I*) **unfolding** *assms comp-apply* **apply**(*rule to-fun-unit-closed*[*of - m*])
apply(*rule cfs-closed*) **using** *assms UPSA.taylor-closed*[*of f m c*] **unfolding** *UPSA.taylor-def* **by** *blast*
have *5*: $(\lambda i.\ a\ i\ x \otimes (t \ominus c\ x)\ [\widehat{}]\ i) \in \textit{inds} \to \textit{carrier}\ Q_p$
proof fix *i* **assume** *A*: $i \in \textit{inds}$
show $a\ i\ x \otimes (t \ominus c\ x)\ [\widehat{}]\ i \in \textit{carrier}\ Q_p$
using *assms*(*8*) *x-closed a-closed SA-car-memE*(*3*)[*of c m*] *SA-car-memE*(*3*)[*of a i m*]
assms(*2*) **by** *blast*
qed
have *6*: $\bigwedge i.\ i \in \textit{inds} \Longrightarrow \textit{taylor-expansion}\ (SA\ m)\ c\ f\ i\ x \otimes (t \ominus c\ x)\ [\widehat{}]\ i = a\ i\ x \otimes (t \ominus c\ x)\ [\widehat{}]\ i$
unfolding *A* **by** *blast*
show *?thesis*
unfolding *4* **apply**(*rule Qp.finsum-cong'*) **apply** *blast*
using *5* **apply** *blast*

using *6* **by** *blast*
qed

lemma *SA-poly-to-Qp-poly-taylor-cfs*:
assumes $f \in$ *carrier* (*UP* (*SA m*))
assumes $x \in$ *carrier* ($Q_p{}^m$)
assumes $c \in$ *carrier* (*SA m*)
shows *taylor-expansion* (*SA m*) *c f i x* =
taylor-expansion Q_p (*c x*) (*SA-poly-to-Qp-poly m x f*) *i*
proof−
have *0*: *SA-poly-to-Qp-poly m x* (*taylor-expansion* (*SA m*) *c f*) =
taylor-expansion Q_p (*c x*) (*SA-poly-to-Qp-poly m x f*)
using *SA-poly-to-Qp-poly-taylor-poly*[*of f m c x*] *assms* **by** *blast*
hence *1*: *SA-poly-to-Qp-poly m x* (*taylor-expansion* (*SA m*) *c f*) *i* =
taylor-expansion Q_p (*c x*) (*SA-poly-to-Qp-poly m x f*) *i*
by *presburger*
thus *?thesis*
using *assms SA-poly-to-Qp-poly-coeff*[*of x m taylor-expansion* (*SA m*) *c f i*]
UPSA.taylor-closed[*of f m c*] **unfolding** *UPSA.taylor-def assms* **by** *blast*
qed

14.13.1 Common Morphisms on Polynomial Rings

Evaluation homomorphism from multivariable polynomials to semialgebraic functions

definition *Qp-ev-hom* **where**
Qp-ev-hom n P = *restrict* (*Qp-ev P*) (*carrier* ($Q_p{}^n$))

lemma *Qp-ev-hom-ev*:
assumes $a \in$ *carrier* ($Q_p{}^n$)
shows *Qp-ev-hom n P a* = *Qp-ev P a*
using *assms* **unfolding** *Qp-ev-hom-def*
by (*meson restrict-apply'*)

lemma *Qp-ev-hom-closed*:
assumes $f \in$ *carrier* ($Q_p[\mathcal{X}_n]$)
shows *Qp-ev-hom n f* $\in$ *carrier* ($Q_p{}^n$) $\to$ *carrier* Q_p
using *Qp-ev-hom-ev assms* **by** (*metis Pi-I eval-at-point-closed*)

lemma *Qp-ev-hom-is-semialg-function*:
assumes $f \in$ *carrier* ($Q_p[\mathcal{X}_n]$)
shows *is-semialg-function n* (*Qp-ev-hom n f*)
unfolding *Qp-ev-hom-def*
using *assms poly-is-semialg*[*of f*] *restrict-is-semialg* **by** *blast*

lemma *Qp-ev-hom-closed'*:
assumes $f \in$ *carrier* ($Q_p[\mathcal{X}_n]$)
shows *Qp-ev-hom n f* $\in$ *carrier* (*Fun*$_n$ Q_p)
apply(*rule Qp.function-ring-car-memI*)

using *Qp-ev-hom-closed*[*of f n*] *assms* **apply** *blast*
unfolding *Qp-ev-hom-def* **using** *assms* **by** (*meson restrict-apply*)

lemma *Qp-ev-hom-in-SA*:
assumes $f \in carrier\ (Q_p[\mathcal{X}_n])$
shows *Qp-ev-hom n f* $\in$ *carrier* (*SA n*)
apply(*rule SA-car-memI*)
using *Qp-ev-hom-closed'* *assms*(*1*) **apply** *blast*
using *Qp-ev-hom-is-semialg-function assms*(*1*) **by** *blast*

lemma *Qp-ev-hom-add*:
assumes $f \in carrier\ (Q_p[\mathcal{X}_n])$
assumes $g \in carrier\ (Q_p[\mathcal{X}_n])$
shows *Qp-ev-hom n* ($f \oplus_{Q_p[\mathcal{X}_n]} g$) = (*Qp-ev-hom n f*) $\oplus_{SA\ n}$ (*Qp-ev-hom n g*)
apply(*rule function-ring-car-eqI*[*of - n*])
using *assms MP.add.m-closed Qp-ev-hom-closed'* **apply** *blast*
using *assms Qp-ev-hom-closed' fun-add-closed SA-plus* **apply** *presburger*
proof− **fix** *a* **assume** *A*: $a \in carrier\ ({Q_p}^n)$
have *Qp-ev-hom n* ($f \oplus_{Q_p\ [\mathcal{X}_n]} g$) *a* = *Qp-ev-hom n f a* $\oplus$ *Qp-ev-hom n g a*
using *A Qp-ev-hom-ev assms eval-at-point-add* **by** *presburger*
then show *Qp-ev-hom n* ($f \oplus_{Q_p\ [\mathcal{X}_n]} g$) *a* = (*Qp-ev-hom n f* $\oplus_{SA\ n}$ *Qp-ev-hom n g*) *a*
using *A SA-add* **by** *blast*
qed

lemma *Qp-ev-hom-mult*:
assumes $f \in carrier\ (Q_p[\mathcal{X}_n])$
assumes $g \in carrier\ (Q_p[\mathcal{X}_n])$
shows *Qp-ev-hom n* ($f \otimes_{Q_p[\mathcal{X}_n]} g$) = (*Qp-ev-hom n f*) $\otimes_{SA\ n}$ (*Qp-ev-hom n g*)
apply(*rule function-ring-car-eqI*[*of - n*])
using *assms MP.m-closed Qp-ev-hom-closed'* **apply** *blast*
using *assms Qp-ev-hom-closed' fun-mult-closed SA-mult*
apply (*meson Qp-ev-hom-in-SA SA-car-memE*(*2*) *SA-imp-semialg SA-mult-closed*)
proof− **fix** *a* **assume** *A*: $a \in carrier\ ({Q_p}^n)$
have *Qp-ev-hom n* ($f \otimes_{Q_p\ [\mathcal{X}_n]} g$) *a* = *Qp-ev-hom n f a* $\otimes$ *Qp-ev-hom n g a*
using *A Qp-ev-hom-ev assms eval-at-point-mult* **by** *presburger*
then show *Qp-ev-hom n* ($f \otimes_{Q_p\ [\mathcal{X}_n]} g$) *a* = (*Qp-ev-hom n f* $\otimes_{SA\ n}$ *Qp-ev-hom n g*) *a*
using *A SA-mult* **by** *blast*
qed

lemma *Qp-ev-hom-one*:
shows *Qp-ev-hom n* $\mathbf{1}_{Q_p[\mathcal{X}_n]} = \mathbf{1}_{SA\ n}$
apply(*rule function-ring-car-eqI*[*of - n*])
using *Qp-ev-hom-closed'* **apply** *blast*
using *function-one-closed SA-one* **apply** *presburger*
unfolding *Qp-ev-hom-def*
using *function-one-eval Qp-ev-hom-def Qp-ev-hom-ev Qp-ev-one SA-one* **by** *pres-*

burger

lemma *Qp-ev-hom-is-hom*:
shows *Qp-ev-hom n* ∈ *ring-hom* (*Q*$_p$[$\mathcal{X}_n$]) (*SA n*)
apply(*rule ring-hom-memI*)
using *Qp-ev-hom-in-SA* **apply** *blast*
using *Qp-ev-hom-mult* **apply** *blast*
using *Qp-ev-hom-add* **apply** *blast*
using *Qp-ev-hom-one* **by** *blast*

lemma *Qp-ev-hom-constant*:
assumes *c* ∈ *carrier Q*$_p$
shows *Qp-ev-hom n* (*Qp.indexed-const c*) = $\mathfrak{c}_n$ *c*
apply(*rule function-ring-car-eqI*[*of - n*])
using *Qp-ev-hom-closed*′ *Qp-to-IP-car assms*(*1*) **apply** *blast*
using *constant-function-closed assms* **apply** *blast*
by (*metis Qp-constE Qp-ev-hom-ev assms eval-at-point-const*)

notation *Qp.variable* (‹$\mathfrak{v}$_, _›)

lemma *Qp-ev-hom-pvar*:
assumes *i* < *n*
shows *Qp-ev-hom n* (*pvar Q*$_p$ *i*) = $\mathfrak{v}_{n,\ i}$
apply(*rule function-ring-car-eqI*[*of - n*])
using *assms Qp-ev-hom-closed*′ *local.pvar-closed* **apply** *blast*
using *Qp.var-in-car assms* **apply** *blast*
unfolding *Qp-ev-hom-def var-def* **using** *assms eval-pvar*
by (*metis* (*no-types*, *lifting*) *restrict-apply*)

definition *ext-hd* **where**
ext-hd m = (λ *x* ∈ *carrier* (*Q*$_p{}^m$). *hd x*)

lemma *hd-zeroth*:
length x > *0* ⟹ *x*!*0* = *hd x*
apply(*induction x*)
apply *simp*
by *simp*

lemma *ext-hd-pvar*:
assumes *m* > *0*
shows *ext-hd m* = (λ*x* ∈ *carrier* (*Q*$_p{}^m$). *eval-at-point Q*$_p$ *x* (*pvar Q*$_p$ *0*))
unfolding *ext-hd-def restrict-def* **using** *assms eval-pvar*[*of 0 m*]
using *hd-zeroth*
by (*metis* (*no-types*, *opaque-lifting*) *cartesian-power-car-memE*)

lemma *ext-hd-closed*:
assumes *m* > *0*
shows *ext-hd m* ∈ *carrier* (*SA m*)
using *ext-hd-pvar*[*of m*] *assms pvar-closed*[*of 0 m*] *Qp-ev-hom-def Qp-ev-hom-in-SA*

by *presburger*

lemma *UP-Qp-poly-to-UP-SA-is-hom*:
shows *poly-lift-hom* ($Q_p[\mathcal{X}_n]$) (*SA n*) (*Qp-ev-hom n*) ∈ *ring-hom* (*UP* ($Q_p[\mathcal{X}_n]$)) (*UP* (*SA n*))
using *UP-cring.poly-lift-hom-is-hom*[*of* $Q_p[\mathcal{X}_n]$ *SA n Qp-ev-hom n*]
unfolding *UP-cring-def*
using *Qp-ev-hom-is-hom SA-is-cring coord-cring-cring* **by** *blast*

definition *coord-ring-to-UP-SA* **where**
coord-ring-to-UP-SA n = *poly-lift-hom* ($Q_p[\mathcal{X}_n]$) (*SA n*) (*Qp-ev-hom n*) ∘ *to-univ-poly* (*Suc n*) *0*

lemma *coord-ring-to-UP-SA-is-hom*:
shows *coord-ring-to-UP-SA n* ∈ *ring-hom* ($Q_p[\mathcal{X}_{Suc\ n}]$) (*UP* (*SA n*))
unfolding *coord-ring-to-UP-SA-def*
using *UP-Qp-poly-to-UP-SA-is-hom*[*of n*] *to-univ-poly-is-hom*[*of 0 n*] *ring-hom-trans*
by *blast*

lemma *coord-ring-to-UP-SA-add*:
assumes *f* ∈ *carrier* ($Q_p[\mathcal{X}_{Suc\ n}]$)
assumes *g* ∈ *carrier* ($Q_p[\mathcal{X}_{Suc\ n}]$)
shows *coord-ring-to-UP-SA n* (*f* $\oplus_{Q_p[\mathcal{X}_{Suc\ n}]}$ *g*) = *coord-ring-to-UP-SA n f* $\oplus_{UP\ (SA\ n)}$ *coord-ring-to-UP-SA n g*
using *assms coord-ring-to-UP-SA-is-hom ring-hom-add*
by (*metis* (*mono-tags*, *opaque-lifting*))

lemma *coord-ring-to-UP-SA-mult*:
assumes *f* ∈ *carrier* ($Q_p[\mathcal{X}_{Suc\ n}]$)
assumes *g* ∈ *carrier* ($Q_p[\mathcal{X}_{Suc\ n}]$)
shows *coord-ring-to-UP-SA n* (*f* $\otimes_{Q_p[\mathcal{X}_{Suc\ n}]}$ *g*) = *coord-ring-to-UP-SA n f* $\otimes_{UP\ (SA\ n)}$ *coord-ring-to-UP-SA n g*
using *assms coord-ring-to-UP-SA-is-hom ring-hom-mult*
by (*metis* (*no-types*, *opaque-lifting*))

lemma *coord-ring-to-UP-SA-one*:
shows *coord-ring-to-UP-SA n* $\mathbf{1}_{Q_p[\mathcal{X}_{Suc\ n}]}$ = $\mathbf{1}_{UP\ (SA\ n)}$
using *coord-ring-to-UP-SA-is-hom ring-hom-one*
by *blast*

lemma *coord-ring-to-UP-SA-closed*:
assumes *f* ∈ *carrier* ($Q_p[\mathcal{X}_{Suc\ n}]$)
shows *coord-ring-to-UP-SA n f* ∈ *carrier* (*UP* (*SA n*))
using *assms coord-ring-to-UP-SA-is-hom ring-hom-closed*
by (*metis* (*no-types*, *opaque-lifting*))

lemma *coord-ring-to-UP-SA-constant*:
assumes *c* ∈ *carrier* Q_p

shows *coord-ring-to-UP-SA n (Qp.indexed-const c) = to-polynomial (SA n) (*$\mathfrak{c}_n$ *c)*
proof−
have *0*: *pre-to-univ-poly (Suc n) 0 (Qp.indexed-const c) = ring.indexed-const (*$Q_p[\mathcal{X}_n]$*) (Qp.indexed-const c)*
unfolding *to-univ-poly-def*
using *pre-to-univ-poly-is-hom(5)[of 0 Suc n pre-to-univ-poly (Suc n) 0 c] assms*
unfolding *coord-ring-def*
using *diff-Suc-1 zero-less-Suc* **by** *presburger*
hence *1*: *to-univ-poly (Suc n) 0 (Qp.indexed-const c) = to-polynomial (*$Q_p[\mathcal{X}_n]$*) (Qp.indexed-const c)*
unfolding *to-univ-poly-def*
using *UP-cring.IP-to-UP-indexed-const[of* $Q_p[\mathcal{X}_n]$ *Qp.indexed-const c 0::nat] assms*
comp-apply[of IP-to-UP (0::nat) pre-to-univ-poly (Suc n) (0::nat) Qp.indexed-const c]
unfolding *UP-cring-def* **using** *Qp-to-IP-car coord-cring-cring* **by** *presburger*
have *2*: *Qp-ev-hom n (Qp.indexed-const c) =* $\mathfrak{c}_n$ *c*
using *Qp-ev-hom-constant[of c n] assms* **by** *blast*
have *3*: *poly-lift-hom (*Q_p $[\mathcal{X}_n]$*) (SA n) (Qp-ev-hom n) (to-polynomial (*Q_p $[\mathcal{X}_n]$*) (Qp.indexed-const c)) = to-polynomial (SA n) (Qp-ev-hom n (Qp.indexed-const c))*
using *UP-cring.poly-lift-hom-extends-hom[of* $Q_p[\mathcal{X}_n]$ *SA n Qp-ev-hom n Qp.indexed-const c]*
unfolding *UP-cring-def coord-ring-def coord-ring-to-UP-SA-def*
by *(metis Qp-ev-hom-is-hom Qp-to-IP-car SA-is-cring assms(1) coord-cring-cring coord-ring-def)*
hence *4*: *poly-lift-hom (*Q_p $[\mathcal{X}_n]$*) (SA n) (Qp-ev-hom n) (to-univ-poly (Suc n) 0 (Qp.indexed-const c)) = to-polynomial (SA n) (*$\mathfrak{c}_n$ *c)*
using *1 2* **by** *presburger*
show *?thesis*
using *assms 4 Qp.indexed-const-closed[of c {..<n}]*
comp-apply[of poly-lift-hom (Pring Q_p *{..<n}) (SA n) (Qp-ev-hom n) to-univ-poly (Suc n) 0 Qp.indexed-const c]*
unfolding *coord-ring-to-UP-SA-def*
by *(metis coord-ring-def)*
qed

lemma *coord-ring-to-UP-SA-pvar-0*:
shows *coord-ring-to-UP-SA n (pvar* Q_p *0) = up-ring.monom (UP (SA n))* $\mathbf{1}_{SA\ n}$ *1*
proof−
have *0*: *pre-to-univ-poly (Suc n) 0 (pvar* Q_p *0) = pvar (*$Q_p[\mathcal{X}_n]$*) 0*
using *pre-to-univ-poly-is-hom(3)[of 0 Suc n pre-to-univ-poly (Suc n) 0] diff-Suc-1 zero-less-Suc*
by *presburger*
have *1*: *IP-to-UP (0::nat) (pvar (*$Q_p[\mathcal{X}_n]$*) 0) = up-ring.monom (UP (*$Q_p[\mathcal{X}_n]$*))* $\mathbf{1}_{Q_p[\mathcal{X}_n]}$ *1*
using *cring.IP-to-UP-var[of* $Q_p[\mathcal{X}_n]$ *0::nat]* **unfolding** *X-poly-def var-to-IP-def*
using *coord-cring-cring* **by** *blast*

have *2*: *to-univ-poly* (*Suc n*) *0* (*pvar* Q_p *0*) = *up-ring.monom* (*UP* ($Q_p[\mathcal{X}_n]$)) $\mathbf{1}_{Q_p[\mathcal{X}_n]}$ *1*
unfolding *to-univ-poly-def* **using** *0 1 comp-apply*
by *metis*
have *3*: *poly-lift-hom* ($Q_p[\mathcal{X}_n]$) (*SA n*) (*Qp-ev-hom n*) (*up-ring.monom* (*UP* ($Q_p[\mathcal{X}_n]$)) $\mathbf{1}_{Q_p[\mathcal{X}_n]}$ *1*)
= *up-ring.monom* (*UP* (*SA n*)) $\mathbf{1}_{SA\ n}$ *1*
using *UP-cring.poly-lift-hom-monom*[*of* $Q_p[\mathcal{X}_n]$ *SA n Qp-ev-hom n*]
unfolding *UP-cring-def*
using *MP.one-closed Qp-ev-hom-is-hom Qp-ev-hom-one SA-is-cring coord-cring-cring*
by *presburger*
thus *?thesis* **unfolding** *coord-ring-to-UP-SA-def*
using *2 3 comp-apply*
by *metis*
qed

lemma *coord-ring-to-UP-SA-pvar-Suc*:
assumes *i* > *0*
assumes *i* < *Suc n*
shows *coord-ring-to-UP-SA n* (*pvar* Q_p *i*) = *to-polynomial* (*SA n*) ($\mathfrak{v}_{n,\ i-1}$)
proof−
have *0*: *pre-to-univ-poly* (*Suc n*) *0* (*pvar* Q_p *i*) = *ring.indexed-const* ($Q_p[\mathcal{X}_n]$) (*pvar* Q_p (*i*−*1*))
using *pre-to-univ-poly-is-hom*(*4*)[*of 0 Suc n pre-to-univ-poly* (*Suc n*) *0 i*] *diff-Suc-1 zero-less-Suc*
assms
unfolding *coord-ring-def* **by** *presburger*
have *1*: *IP-to-UP* (*0*::*nat*) (*ring.indexed-const* ($Q_p[\mathcal{X}_n]$) (*pvar* Q_p (*i*−*1*))) = *to-polynomial* ($Q_p[\mathcal{X}_n]$) (*pvar* Q_p (*i*−*1*))
using *UP-cring.IP-to-UP-indexed-const*[*of* $Q_p[\mathcal{X}_n]$ *pvar* Q_p (*i*−*1*) *0*::*nat*] *coord-cring-cring*
unfolding *UP-cring-def*
by (*metis assms diff-less less-Suc-eq less-imp-diff-less less-numeral-extra*(*1*) *local.pvar-closed*)
have *2*: *to-univ-poly* (*Suc n*) *0* (*pvar* Q_p *i*) = *to-polynomial* ($Q_p[\mathcal{X}_n]$) (*pvar* Q_p (*i*−*1*))
unfolding *to-univ-poly-def* **using** *0 1 comp-apply*
by *metis*
have *3*: *poly-lift-hom* ($Q_p[\mathcal{X}_n]$) (*SA n*) (*Qp-ev-hom n*) (*to-polynomial* ($Q_p[\mathcal{X}_n]$) (*pvar* Q_p (*i*−*1*)))
= *to-polynomial* (*SA n*) ($\mathfrak{v}_{n,\ (i-1)}$)
using *UP-cring.poly-lift-hom-extends-hom*[*of* $Q_p[\mathcal{X}_n]$ *SA n Qp-ev-hom n pvar* Q_p (*i*−*1*)]
unfolding *UP-cring-def*
by (*metis* (*no-types*, *lifting*) *Qp-ev-hom-is-hom Qp-ev-hom-pvar SA-is-cring Suc-diff-1*
Suc-less-eq assms coord-cring-cring local.pvar-closed)
thus *?thesis* **unfolding** *coord-ring-to-UP-SA-def*
using *2 3 assms comp-apply*

by *metis*
qed

lemma *coord-ring-to-UP-SA-eval*:
assumes $f \in carrier\ (Q_p[\mathcal{X}_{Suc\ n}])$
assumes $a \in carrier\ ({Q_p}^n)$
assumes $t \in carrier\ Q_p$
shows $Qp\text{-}ev\ f\ (t\#a) = ((SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ (coord\text{-}ring\text{-}to\text{-}UP\text{-}SA\ n\ f))) \cdot t$
proof(*rule coord-ring-car-induct*[*of f Suc n*])
have *ta-closed*: $t\ \#\ a \in carrier\ ({Q_p}^{Suc\ n})$
using *assms cartesian-power-cons* **by** (*metis Suc-eq-plus1*)
show $f \in carrier\ (Q_p\ [\mathcal{X}_{Suc\ n}])$
by (*simp add*: *assms*)
show $\bigwedge c.\ c \in carrier\ Q_p \Longrightarrow eval\text{-}at\text{-}point\ Q_p\ (t\ \#\ a)\ (Qp.indexed\text{-}const\ c) = SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ (coord\text{-}ring\text{-}to\text{-}UP\text{-}SA\ n\ (Qp.indexed\text{-}const\ c)) \cdot t$
proof− **fix** *c* **assume** *A*: $c \in carrier\ Q_p$
have *0*: $eval\text{-}at\text{-}point\ Q_p\ (t\ \#\ a)\ (Qp.indexed\text{-}const\ c) = c$
using *A eval-at-point-const ta-closed* **by** *blast*
have *1*: $SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ (coord\text{-}ring\text{-}to\text{-}UP\text{-}SA\ n\ (Qp.indexed\text{-}const\ c)) = to\text{-}polynomial\ Q_p\ c$
using *coord-ring-to-UP-SA-constant*[*of c n*] *A Qp-constE SA-car SA-poly-to-Qp-poly-extends-apply assms constant-function-in-semialg-functions*
by *presburger*
show $eval\text{-}at\text{-}point\ Q_p\ (t\ \#\ a)\ (Qp.indexed\text{-}const\ c) = SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ (coord\text{-}ring\text{-}to\text{-}UP\text{-}SA\ n\ (Qp.indexed\text{-}const\ c)) \cdot t$
using *0 1 A UPQ.to-fun-to-poly assms* **by** *presburger*
qed
show $\bigwedge p\ q.\ p \in carrier\ (Q_p\ [\mathcal{X}_{Suc\ n}]) \Longrightarrow q \in carrier\ (Q_p\ [\mathcal{X}_{Suc\ n}]) \Longrightarrow eval\text{-}at\text{-}point\ Q_p\ (t\ \#\ a)\ p = SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ (coord\text{-}ring\text{-}to\text{-}UP\text{-}SA\ n\ p) \cdot t \Longrightarrow eval\text{-}at\text{-}point\ Q_p\ (t\ \#\ a)\ q = SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ (coord\text{-}ring\text{-}to\text{-}UP\text{-}SA\ n\ q) \cdot t \Longrightarrow eval\text{-}at\text{-}point\ Q_p\ (t\ \#\ a)\ (p \oplus_{Q_p} [\mathcal{X}_{Suc\ n}]\ q) = SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ (coord\text{-}ring\text{-}to\text{-}UP\text{-}SA\ n\ (p \oplus_{Q_p} [\mathcal{X}_{Suc\ n}]\ q)) \cdot t$
proof− **fix** *p q* **assume** *A*: $p \in carrier\ (Q_p\ [\mathcal{X}_{Suc\ n}])\ q \in carrier\ (Q_p\ [\mathcal{X}_{Suc\ n}])$
$eval\text{-}at\text{-}point\ Q_p\ (t\ \#\ a)\ p = SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ (coord\text{-}ring\text{-}to\text{-}UP\text{-}SA\ n\ p) \cdot t$
$eval\text{-}at\text{-}point\ Q_p\ (t\ \#\ a)\ q = SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ (coord\text{-}ring\text{-}to\text{-}UP\text{-}SA\ n\ q) \cdot t$
have *0*: $eval\text{-}at\text{-}point\ Q_p\ (t\ \#\ a)\ (p \oplus_{Q_p} [\mathcal{X}_{Suc\ n}]\ q) = eval\text{-}at\text{-}point\ Q_p\ (t\ \#\ a)\ p \oplus eval\text{-}at\text{-}point\ Q_p\ (t\ \#\ a)\ q$
using *A*(*1*) *A*(*2*) *eval-at-point-add ta-closed* **by** *blast*
have *1*: $SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ (coord\text{-}ring\text{-}to\text{-}UP\text{-}SA\ n\ (p \oplus_{Q_p} [\mathcal{X}_{Suc\ n}]\ q)) = SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ (coord\text{-}ring\text{-}to\text{-}UP\text{-}SA\ n\ p)\ \oplus_{UP\ Q_p} SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ (coord\text{-}ring\text{-}to\text{-}UP\text{-}SA\ n\ q)$
using *coord-ring-to-UP-SA-add*[*of*] *assms coord-ring-to-UP-SA-closed A*

SA-poly-to-Qp-poly-add[*of a n coord-ring-to-UP-SA n p coord-ring-to-UP-SA n q*]
by *presburger*
show *eval-at-point* Q_p $(t \# a)$ $(p \oplus_{Q_p} [\mathcal{X}_{Suc\ n}]\ q) = SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a$ (*coord-ring-to-UP-SA n* $(p \oplus_{Q_p} [\mathcal{X}_{Suc\ n}]\ q)) \cdot t$
using *0 1 assms SA-poly-to-Qp-poly-closed*[*of a n*] *SA-poly-to-Qp-poly-closed A*(*1*) *A*(*2*) *A*(*3*) *A*(*4*) *UPQ.to-fun-plus coord-ring-to-UP-SA-closed*
by *presburger*
qed
show $\bigwedge p\ i.\ p \in carrier\ (Q_p\ [\mathcal{X}_{Suc\ n}]) \Longrightarrow$
$i < Suc\ n \Longrightarrow$
eval-at-point Q_p $(t \# a)$ $p = SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a$ (*coord-ring-to-UP-SA n p*) $\cdot\ t \Longrightarrow$
eval-at-point Q_p $(t \# a)$ $(p \otimes_{Q_p} [\mathcal{X}_{Suc\ n}]\ pvar\ Q_p\ i) = SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a$ (*coord-ring-to-UP-SA n* $(p \otimes_{Q_p} [\mathcal{X}_{Suc\ n}]\ pvar\ Q_p\ i)) \cdot t$
proof− **fix** *p i* **assume** *A*: $p \in carrier\ (Q_p\ [\mathcal{X}_{Suc\ n}])\ i < Suc\ n$
eval-at-point Q_p $(t \# a)$ $p = SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a$ (*coord-ring-to-UP-SA n p*) $\cdot\ t$
show *eval-at-point* Q_p $(t \# a)$ $(p \otimes_{Q_p} [\mathcal{X}_{Suc\ n}]\ pvar\ Q_p\ i) = SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a$ (*coord-ring-to-UP-SA n* $(p \otimes_{Q_p} [\mathcal{X}_{Suc\ n}]\ pvar\ Q_p\ i)) \cdot t$
proof−
have *0*: *eval-at-point* Q_p $(t \# a)$ $(p \otimes_{Q_p} [\mathcal{X}_{Suc\ n}]\ pvar\ Q_p\ i) =$ *eval-at-point* Q_p $(t \# a)$ $p \otimes$ *eval-at-point* Q_p $(t \# a)$ $(pvar\ Q_p\ i)$
using *A*(*1*) *A*(*2*) *eval-at-point-mult local.pvar-closed ta-closed* **by** *blast*
have *1*: *coord-ring-to-UP-SA n* $(p \otimes_{Q_p} [\mathcal{X}_{Suc\ n}]\ pvar\ Q_p\ i) =$ *coord-ring-to-UP-SA n p* $\otimes_{UP\ (SA\ n)}$ *coord-ring-to-UP-SA n* $(pvar\ Q_p\ i)$
using *A*(*1*) *A*(*2*) *assms*(*1*) *coord-ring-to-UP-SA-mult local.pvar-closed* **by** *blast*
hence *2*: *SA-poly-to-Qp-poly n a* (*coord-ring-to-UP-SA n* $(p \otimes_{Q_p} [\mathcal{X}_{Suc\ n}]\ pvar\ Q_p\ i)) =$
SA-poly-to-Qp-poly n a (*coord-ring-to-UP-SA n p*) $\otimes_{UP\ Q_p}$ *SA-poly-to-Qp-poly n a* (*coord-ring-to-UP-SA n* $(pvar\ Q_p\ i)$)
using *SA-poly-to-Qp-poly-mult coord-ring-to-UP-SA-closed A*(*1*) *A*(*2*) *assms local.pvar-closed*
by *presburger*
show *eval-at-point* Q_p $(t \# a)$ $(p \otimes_{Q_p} [\mathcal{X}_{Suc\ n}]\ pvar\ Q_p\ i) = SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a$ (*coord-ring-to-UP-SA n* $(p \otimes_{Q_p} [\mathcal{X}_{Suc\ n}]\ pvar\ Q_p\ i)) \cdot t$
proof(*cases* $i = 0$)
case *True*
have *T0*: *SA-poly-to-Qp-poly n a* (*coord-ring-to-UP-SA n* $(p \otimes_{Q_p} [\mathcal{X}_{Suc\ n}]\ pvar\ Q_p\ i)) =$
SA-poly-to-Qp-poly n a (*coord-ring-to-UP-SA n p*) $\otimes_{UP\ Q_p}$ *up-ring.monom* $(UP\ Q_p)$ **1** *1*
using *True coord-ring-to-UP-SA-pvar-0 SA-poly-to-Qp-poly-monom*
by (*metis 2 function-one-eval Qp.one-closed SA-car SA-one assms constant-function-in-semialg-functions function-one-as-constant*)

have *T1*: *eval-at-point* Q_p $(t \# a)$ $(p \otimes_{Q_p} [\mathcal{X}_{Suc\ n}]$ *pvar* Q_p $i) =$ *eval-at-point* Q_p $(t \# a)$ $p \otimes$ t
using *0 True ta-closed eval-pvar*[*of 0 Suc n t*#*a*]
by (*metis A*(*2*) *nth-Cons-0*)
then show *?thesis*
using *T0 A SA-poly-to-Qp-poly-closed*[*of a n coord-ring-to-UP-SA n p*] *UPQ.to-fun-X*[*of t*] *to-fun-mult*
coord-ring-to-UP-SA-closed[*of*] *UPQ.X-closed*[*of*] **unfolding** *X-poly-def*
using *assms UPQ.to-fun-mult* **by** *presburger*
next
case *False*
have $\mathfrak{v}_{n,\ i\ -\ 1}\ a = a!(i-1)$
by (*metis A*(*2*) *False Qp.varE Suc-diff-1 Suc-less-eq assms less-Suc-eq-0-disj*)
hence *F0*: *SA-poly-to-Qp-poly n a* (*coord-ring-to-UP-SA n* $(p \otimes_{Q_p} [\mathcal{X}_{Suc\ n}]$ *pvar* Q_p $i)) =$
SA-poly-to-Qp-poly n a (*coord-ring-to-UP-SA n p*) $\otimes_{UP\ Q_p}$ *to-polynomial* Q_p $(a!(i-1))$
using *False coord-ring-to-UP-SA-pvar-Suc*[*of i n*] *SA-poly-to-Qp-poly-extends-apply*[*of a n* $\mathfrak{v}_{n,\ i\ -\ 1}$]
by (*metis* (*no-types*, *lifting*) *2 A*(*2*) *Qp-ev-hom-in-SA Qp-ev-hom-pvar Suc-diff-1 Suc-less-eq assms local.pvar-closed neq0-conv*)
have *F1*: *eval-at-point* Q_p $(t \# a)$ $(p \otimes_{Q_p} [\mathcal{X}_{Suc\ n}]$ *pvar* Q_p $i) =$ *eval-at-point* Q_p $(t \# a)$ $p \otimes (a!(i-1))$
using *0 False ta-closed eval-pvar*[*of i Suc n t*#*a*]
by (*metis A*(*2*) *nth-Cons'*)
then show *?thesis*
using *F0 A SA-poly-to-Qp-poly-closed*[*of a n coord-ring-to-UP-SA n p*] *to-fun-mult*
coord-ring-to-UP-SA-closed[*of p n*] *False UPQ.to-fun-to-poly UPQ.to-poly-closed assms*
eval-at-point-closed eval-pvar local.pvar-closed neq0-conv nth-Cons-pos ta-closed
by (*metis* (*no-types*, *lifting*) *UPQ.to-fun-mult*)
qed
qed
qed
qed

14.13.2 Gluing Semialgebraic Polynomials

definition *SA-poly-glue* **where**
SA-poly-glue m S f g $= (\lambda\ n.$ *fun-glue m S* $(f\ n)$ $(g\ n))$

lemma *SA-poly-glue-closed*:
assumes $f \in$ *carrier* $(UP\ (SA\ m))$
assumes $g \in$ *carrier* $(UP\ (SA\ m))$
assumes *is-semialgebraic m S*
shows *SA-poly-glue m S f g* $\in$ *carrier* $(UP\ (SA\ m))$

proof(*rule UP-car-memI*[*of max* (*deg* (*SA m*) *f*) (*deg* (*SA m*) *g*)])
fix *n* **assume** *A*: *max* (*deg* (*SA m*) *f*) (*deg* (*SA m*) *g*) < *n* **show** *SA-poly-glue m S f g n* = $\mathbf{0}_{SA\ m}$
unfolding *SA-poly-glue-def*
proof−
have *0*: *n* > (*deg* (*SA m*) *f*)
using *A* **by** *simp*
have *1*: *n* > (*deg* (*SA m*) *g*)
using *A* **by** *simp*
have *2*: *f n* = $\mathbf{0}_{SA\ m}$
using *0 assms UPSA.deg-leE* **by** *blast*
have *3*: *g n* = $\mathbf{0}_{SA\ m}$
using *1 assms UPSA.deg-leE* **by** *blast*
show *fun-glue m S* (*f n*) (*g n*) = $\mathbf{0}_{SA\ m}$
unfolding *SA-zero function-ring-def ring-record-simps function-zero-def 2 3*
proof fix *x*
show *fun-glue m S* (λ*x*∈*carrier* ($Q_p{}^m$). **0**) (λ*x*∈*carrier* ($Q_p{}^m$). **0**) *x* = (λ*x*∈*carrier* ($Q_p{}^m$). **0**) *x*
apply(*cases x* ∈ *carrier* ($Q_p{}^m$))
unfolding *fun-glue-def restrict-def* **apply** *presburger*
by *auto*
qed
qed
next
show ⋀*n*. *SA-poly-glue m S f g n* ∈ *carrier* (*SA m*)
unfolding *SA-poly-glue-def* **apply**(*rule fun-glue-closed*)
by(*rule cfs-closed*, *rule assms*, *rule cfs-closed*, *rule assms*, *rule assms*)
qed

lemma *SA-poly-glue-deg*:
assumes *f* ∈ *carrier* (*UP* (*SA m*))
assumes *g* ∈ *carrier* (*UP* (*SA m*))
assumes *is-semialgebraic m S*
assumes *deg* (*SA m*) *f* ≤ *d*
assumes *deg* (*SA m*) *g* ≤ *d*
shows *deg* (*SA m*) (*SA-poly-glue m S f g*) ≤ *d*
apply(*rule deg-leqI*, *rule SA-poly-glue-closed*, *rule assms*, *rule assms*, *rule assms*)
proof− **fix** *n* **assume** *A*: *d* < *n*
show *SA-poly-glue m S f g n* = $\mathbf{0}_{SA\ m}$
unfolding *SA-poly-glue-def*
proof−
have *0*: *n* > (*deg* (*SA m*) *f*)
using *A assms* **by** *linarith*
have *1*: *n* > (*deg* (*SA m*) *g*)
using *A assms* **by** *linarith*
have *2*: *f n* = $\mathbf{0}_{SA\ m}$
using *0 assms UPSA.deg-leE* **by** *blast*
have *3*: *g n* = $\mathbf{0}_{SA\ m}$
using *1 assms UPSA.deg-leE* **by** *blast*

show *fun-glue m S (f n) (g n)* = $\mathbf{0}_{SA\ m}$
unfolding *SA-zero function-ring-def ring-record-simps function-zero-def 2 3*
proof fix *x*
show *fun-glue m S* (λ*x*∈*carrier* ($Q_p{}^m$). **0**) (λ*x*∈*carrier* ($Q_p{}^m$). **0**) *x* = (λ*x*∈*carrier* ($Q_p{}^m$). **0**) *x*
apply(*cases x* ∈ *carrier* ($Q_p{}^m$))
unfolding *fun-glue-def restrict-def* **apply** *presburger*
by *auto*
qed
qed
qed

lemma *UP-SA-cfs-closed*:
assumes *g* ∈ *carrier* (*UP* (*SA m*))
shows *g k* ∈ *carrier* (*SA m*)
using *assms UP-ring.cfs-closed*[*of SA m g k*] *SA-is-ring*[*of m*] **unfolding** *UP-ring-def*
by *blast*

lemma *SA-poly-glue-cfs1*:
assumes *f* ∈ *carrier* (*UP* (*SA m*))
assumes *g* ∈ *carrier* (*UP* (*SA m*))
assumes *is-semialgebraic m S*
assumes *x* ∈ *S*
shows (*SA-poly-glue m S f g*) *n x* = *f n x*
unfolding *SA-poly-glue-def fun-glue-def restrict-def*
using *assms*
by (*metis SA-car local.function-ring-not-car padic-fields.UP-SA-cfs-closed padic-fields-axioms semialg-functions-memE*(*2*))

lemma *SA-poly-glue-cfs2*:
assumes *f* ∈ *carrier* (*UP* (*SA m*))
assumes *g* ∈ *carrier* (*UP* (*SA m*))
assumes *is-semialgebraic m S*
assumes *x* ∉ *S*
assumes *x* ∈ *carrier* ($Q_p{}^m$)
shows (*SA-poly-glue m S f g*) *n x* = *g n x*
unfolding *SA-poly-glue-def fun-glue-def restrict-def*
using *assms* **by** *meson*

lemma *SA-poly-glue-to-Qp-poly1*:
assumes *f* ∈ *carrier* (*UP* (*SA m*))
assumes *g* ∈ *carrier* (*UP* (*SA m*))
assumes *is-semialgebraic m S*
assumes *x* ∈ *S*
shows *SA-poly-to-Qp-poly m x* (*SA-poly-glue m S f g*) = *SA-poly-to-Qp-poly m x f*
proof fix *n*
have *x-closed*: *x* ∈ *carrier* ($Q_p{}^m$)

using *assms is-semialgebraic-closed* **by** *blast*
have *0*: *SA-poly-to-Qp-poly m x* (*SA-poly-glue m S f g*) *n* = (*SA-poly-glue m S f g*) *n x*
by(*rule SA-poly-to-Qp-poly-coeff*[*of x m SA-poly-glue m S f g*], *rule x-closed*, *rule SA-poly-glue-closed*
, *rule assms*, *rule assms*, *rule assms*)
have *1*: (*SA-poly-glue m S f g*) *n x* = *f n x*
by(*rule SA-poly-glue-cfs1* , *rule assms*, *rule assms*, *rule assms*, *rule assms*)
show *SA-poly-to-Qp-poly m x* (*SA-poly-glue m S f g*) *n* = *SA-poly-to-Qp-poly m x f n*
unfolding *0 1* **using** *SA-poly-to-Qp-poly-coeff*[*of x m f n*] *assms* (*1*) *x-closed* **by** *blast*
qed

lemma *SA-poly-glue-to-Qp-poly2*:
assumes *f* ∈ *carrier* (*UP* (*SA m*))
assumes *g* ∈ *carrier* (*UP* (*SA m*))
assumes *is-semialgebraic m S*
assumes *x* ∉ *S*
assumes *x* ∈ *carrier* (${Q_p}^m$)
shows *SA-poly-to-Qp-poly m x* (*SA-poly-glue m S f g*) = *SA-poly-to-Qp-poly m x g*
proof fix *n*
have *x-closed*: *x* ∈ *carrier* (${Q_p}^m$)
using *assms is-semialgebraic-closed* **by** *blast*
have *0*: *SA-poly-to-Qp-poly m x* (*SA-poly-glue m S f g*) *n* = (*SA-poly-glue m S f g*) *n x*
by(*rule SA-poly-to-Qp-poly-coeff*[*of x m SA-poly-glue m S f g*], *rule x-closed*, *rule SA-poly-glue-closed*
, *rule assms*, *rule assms*, *rule assms*)
have *1*: (*SA-poly-glue m S f g*) *n x* = *g n x*
by(*rule SA-poly-glue-cfs2* , *rule assms*, *rule assms*, *rule assms*, *rule assms*, *rule x-closed*)
show *SA-poly-to-Qp-poly m x* (*SA-poly-glue m S f g*) *n* = *SA-poly-to-Qp-poly m x g n*
unfolding *0 1* **using** *SA-poly-to-Qp-poly-coeff*[*of x m g n*] *assms x-closed* **by** *blast*
qed

14.13.3 Polynomials over the Valuation Ring

definition *integral-on* **where**
integral-on m B = {*f* ∈ *carrier* (*UP* (*SA m*)). (∀ *x* ∈ *B*. ∀ *i*. *SA-poly-to-Qp-poly m x f i* ∈ $\mathcal{O}_p$)}

lemma *integral-on-memI*:
assumes *f* ∈ *carrier* (*UP* (*SA m*))
assumes ⋀*x i*. *x* ∈ *B* ⟹ *SA-poly-to-Qp-poly m x f i* ∈ $\mathcal{O}_p$
shows *f* ∈ *integral-on m B*

unfolding *integral-on-def mem-Collect-eq* **using** *assms* **by** *blast*

lemma *integral-on-memE*:
assumes $f \in$ *integral-on* m B
shows $f \in$ *carrier* (*UP* (*SA* m))
$\bigwedge x.\ x \in B \Longrightarrow$ *SA-poly-to-Qp-poly* m x f $i \in \mathcal{O}_p$
using *assms* **unfolding** *integral-on-def mem-Collect-eq* **apply** *blast*
using *assms* **unfolding** *integral-on-def mem-Collect-eq* **by** *blast*

lemma *one-integral-on*:
assumes $B \subseteq$ *carrier* (${Q_p}^m$)
shows $\mathbf{1}_{UP\ (SA\ m)} \in$ *integral-on* m B
apply(*rule integral-on-memI*)
apply *blast*
proof− **fix** x i **assume** A: $x \in B$
have *0*: *SA-poly-to-Qp-poly* m x $\mathbf{1}_{UP\ (SA\ m)} = \mathbf{1}_{UP\ Q_p}$
apply(*rule ring-hom-one*[*of - UP* (*SA* m)])
using *SA-poly-to-Qp-poly-is-hom*[*of* x m] A *assms* **by** *blast*
show *SA-poly-to-Qp-poly* m x $\mathbf{1}_{UP\ (SA\ m)}$ $i \in \mathcal{O}_p$
unfolding *0*
apply(*rule val-ring-memI*)
apply(*rule UPQ.cfs-closed*)
apply *blast*
apply(*cases* $i = 0$)
apply (*metis Qp.add.nat-pow-eone Qp.one-closed UPQ.cfs-one val-of-nat-inc val-one*)
by (*metis Qp.int-inc-zero UPQ.cfs-one val-of-int-inc*)
qed

lemma *integral-on-plus*:
assumes $B \subseteq$ *carrier* (${Q_p}^m$)
assumes $f \in$ *integral-on* m B
assumes $g \in$ *integral-on* m B
shows $f \oplus_{UP\ (SA\ m)} g \in$ *integral-on* m B
proof(*rule integral-on-memI*)
show $f \oplus_{UP\ (SA\ m)} g \in$ *carrier* (*UP* (*SA* m))
using *assms integral-on-memE* **by** *blast*
show $\bigwedge x\ i.\ x \in B \Longrightarrow$ *SA-poly-to-Qp-poly* m x $(f \oplus_{UP\ (SA\ m)} g)$ $i \in \mathcal{O}_p$
proof− **fix** x i **assume** A: $x \in B$
have *0*: *SA-poly-to-Qp-poly* m x $(f \oplus_{UP\ (SA\ m)} g) =$ *SA-poly-to-Qp-poly* m x f
$\oplus_{UP\ Q_p}$ *SA-poly-to-Qp-poly* m x g
apply(*rule SA-poly-to-Qp-poly-add*)
using A *assms* **apply** *blast* **using** *assms integral-on-memE* **apply** *blast*
using *assms integral-on-memE* **by** *blast*
have *1*: *SA-poly-to-Qp-poly* m x $(f \oplus_{UP\ (SA\ m)} g)$ $i =$ *SA-poly-to-Qp-poly* m x f i
$\oplus$ *SA-poly-to-Qp-poly* m x g i

unfolding *0* **apply**(*rule UPQ.cfs-add*)
apply(*rule SA-poly-to-Qp-poly-closed*)
using *A assms* **apply** *blast*
using *assms integral-on-memE* **apply** *blast*
apply(*rule SA-poly-to-Qp-poly-closed*)
using *A assms* **apply** *blast*
using *assms integral-on-memE* **by** *blast*
show *SA-poly-to-Qp-poly m x* ($f \oplus_{UP\ (SA\ m)} g$) $i \in \mathcal{O}_p$
unfolding *1* **using** *assms integral-on-memE*
using *A val-ring-add-closed* **by** *presburger*
qed
qed

lemma *integral-on-times*:
assumes $B \subseteq$ *carrier* ($Q_p{}^m$)
assumes $f \in$ *integral-on m B*
assumes $g \in$ *integral-on m B*
shows $f \otimes_{UP\ (SA\ m)} g \in$ *integral-on m B*
proof(*rule integral-on-memI*)
show $f \otimes_{UP\ (SA\ m)} g \in$ *carrier* (*UP* (*SA m*))
using *assms integral-on-memE* **by** *blast*
show $\bigwedge x.\ x \in B \Longrightarrow$ *SA-poly-to-Qp-poly m x* ($f \otimes_{UP\ (SA\ m)} g$) $i \in \mathcal{O}_p$
proof− **fix** *x i* **assume** *A*: $x \in B$
have *0*: *SA-poly-to-Qp-poly m x* ($f \otimes_{UP\ (SA\ m)} g$) = *SA-poly-to-Qp-poly m x f* $\otimes_{UP\ Q_p}$ *SA-poly-to-Qp-poly m x g*
apply(*rule SA-poly-to-Qp-poly-mult*)
using *A assms* **apply** *blast* **using** *assms integral-on-memE* **apply** *blast*
using *assms integral-on-memE* **by** *blast*
obtain *S* **where** *S-def*: $S = UP\ (Q_p\ (\!|\ carrier := \mathcal{O}_p\ |\!))$
by *blast*
have *1*: *cring S*
unfolding *S-def* **apply**(*rule UPQ.UP-ring-subring-is-ring*)
using *val-ring-subring* **by** *blast*
have *2*: *carrier S* = $\{h \in$ *carrier* (*UP* Q_p). $\forall n.\ h\ n \in \mathcal{O}_p\}$
unfolding *S-def* **using** *UPQ.UP-ring-subring-car*[*of* $\mathcal{O}_p$] *val-ring-subring* **by** *blast*
have *3*: *SA-poly-to-Qp-poly m x f* $\in$ *carrier S*
unfolding *2 mem-Collect-eq* **using** *assms integral-on-memE SA-poly-to-Qp-poly-closed A*
by *blast*
have *4*: *SA-poly-to-Qp-poly m x g* $\in$ *carrier S*
unfolding *2 mem-Collect-eq* **using** *assms integral-on-memE SA-poly-to-Qp-poly-closed A*
by *blast*
have *5*: *SA-poly-to-Qp-poly m x* ($f \otimes_{UP\ (SA\ m)} g$) $\in$ *carrier S*
unfolding *0*
using *cring.cring-simprules*(*5*)[*of S*]*3 4 1 UPQ.UP-ring-subring-mult*[*of* $\mathcal{O}_p$

SA-poly-to-Qp-poly m x f SA-poly-to-Qp-poly m x g]
using *S-def val-ring-subring* **by** *metis*
show *SA-poly-to-Qp-poly m x* ($f \otimes_{UP\ (SA\ m)} g$) $i \in \mathcal{O}_p$
using *5* **unfolding** *2 mem-Collect-eq* **by** *blast*
qed
qed

lemma *integral-on-a-minus*:
assumes $B \subseteq$ *carrier* (${Q_p}^m$)
assumes $f \in$ *integral-on m B*
shows $\ominus_{UP\ (SA\ m)}\ f \in$ *integral-on m B*
apply(*rule integral-on-memI*)
using *assms integral-on-memE*(*1*)[*of f m B*]
apply *blast*
proof− **fix** *x i* **assume** *A*: $x \in B$
have *0*: *SA-poly-to-Qp-poly m x* ($\ominus_{UP\ (SA\ m)}\ f$) = $\ominus_{UP\ Q_p}$ *SA-poly-to-Qp-poly m x f*
apply(*rule UP-cring.ring-hom-uminus*[*of UP* Q_p *UP* (*SA m*) *SA-poly-to-Qp-poly m x f*])
unfolding *UP-cring-def*
apply (*simp add*: *UPQ.M-cring*)
apply (*simp add*: *UPSA.P.ring-axioms*)
apply(*rule SA-poly-to-Qp-poly-is-hom*)
using *A assms* **apply** *blast*
using *assms integral-on-memE* **by** *blast*
have *2*: $\bigwedge f.\ f \in$ *carrier* (*UP* Q_p) $\Longrightarrow$ ($\ominus_{UP\ Q_p}\ f$) $i = \ominus$ ($f\ i$)
using *UPQ.cfs-a-inv* **by** *blast*
have *1*: *SA-poly-to-Qp-poly m x* ($\ominus_{UP\ (SA\ m)}\ f$) $i = \ominus$ (*SA-poly-to-Qp-poly m x f*) *i*
unfolding *0* **apply**(*rule 2*)
using *integral-on-memE assms A SA-poly-to-Qp-poly-closed*[*of x m f*] **by** *blast*
show *SA-poly-to-Qp-poly m x* ($\ominus_{UP\ (SA\ m)}\ f$) $i \in \mathcal{O}_p$
unfolding *1* **using** *A assms integral-on-memE*(*2*)[*of f m B x i*]
using *val-ring-ainv-closed* **by** *blast*
qed

lemma *integral-on-subring*:
assumes $B \subseteq$ *carrier* (${Q_p}^m$)
shows *subring* (*integral-on m B*) (*UP* (*SA m*))
proof(*rule subringI*)
show *integral-on m B* $\subseteq$ *carrier* (*UP* (*SA m*))
unfolding *integral-on-def* **by** *blast*
show $\mathbf{1}_{UP\ (SA\ m)} \in$ *integral-on m B*
using *one-integral-on assms* **by** *blast*
show $\bigwedge h.\ h \in$ *integral-on m B* $\Longrightarrow$ $\ominus_{UP\ (SA\ m)}\ h \in$ *integral-on m B*
using *integral-on-a-minus assms* **by** *blast*
show $\bigwedge h1\ h2.\ h1 \in$ *integral-on m B* $\Longrightarrow$ *h2* $\in$ *integral-on m B* $\Longrightarrow$ *h1* $\otimes_{UP\ (SA\ m)}$ *h2* $\in$ *integral-on m B*

using *integral-on-times assms* **by** *blast*
show $\bigwedge h1\ h2.\ h1 \in$ *integral-on* $m\ B \Longrightarrow h2 \in$ *integral-on* $m\ B \Longrightarrow h1 \oplus_{UP\ (SA\ m)}$ $h2 \in$ *integral-on* $m\ B$
using *integral-on-plus assms* **by** *blast*
qed

lemma *val-ring-add-pow*:
assumes $a \in$ *carrier* Q_p
assumes *val* $a \geq 0$
shows *val* $([(n::nat)]\cdot a) \geq 0$
proof−
have *0*: $[(n::nat)]\cdot a = ([n]\cdot \mathbf{1})\otimes a$
using *assms Qp.add-pow-ldistr Qp.cring-simprules*(*12*) *Qp.one-closed* **by** *presburger*
show *?thesis* **unfolding** *0* **using** *assms*
by (*meson Qp.nat-inc-closed val-ring-memE val-of-nat-inc val-ringI val-ring-times-closed*)
qed

lemma *val-ring-poly-eval*:
assumes $f \in$ *carrier* (*UP* Q_p)
assumes $\bigwedge\ i.\ f\ i \in \mathcal{O}_p$
shows $\bigwedge x.\ x \in \mathcal{O}_p \Longrightarrow f \cdot x \in \mathcal{O}_p$
proof− **fix** x **assume** A: $x \in \mathcal{O}_p$
obtain S **where** *S-def*: $S = (Q_p\ (\!|\ carrier := \mathcal{O}_p\ |\!))$
by *blast*
have *0*: *UP-cring* S
unfolding *S-def* **apply**(*rule UPQ.UP-ring-subring*(*1*))
using *val-ring-subring* **by** *blast*
have *1*: *to-function* $Q_p\ f\ x =$ *to-function* $S\ f\ x$
unfolding *S-def* **apply**(*rule UPQ.UP-subring-eval*)
using *val-ring-subring* **apply** *blast*
apply(*rule UPQ.poly-cfs-subring*) **using** *val-ring-subring* **apply** *blast*
using *assms* **apply** *blast*
using *assms* **apply** *blast* **using** *A* **by** *blast*
have *2*: $f \in$ *carrier* (*UP* S)
unfolding *S-def*
using *UPQ.UP-ring-subring-car*[*of* $\mathcal{O}_p$] *assms val-ring-subring* **by** *blast*
have *3*: *to-function* $S\ f\ x \in \mathcal{O}_p$
using *UPQ.UP-subring-eval-closed*[*of* $\mathcal{O}_p\ f\ x$]
using *1 0 UP-cring.to-fun-closed*[*of* $S\ f\ x$]
unfolding *S-def*
by (*metis 2 A S-def UPQ.to-fun-def val-ring-subring*)
thus $f \cdot x \in \mathcal{O}_p$
using *1 UPQ.to-fun-def* **by** *presburger*
qed

lemma *SA-poly-constant-res-class-semialg'*:
assumes $f \in$ *carrier* (*UP* (*SA* m))
assumes $\bigwedge i\ x.\ x \in B \Longrightarrow f\ i\ x \in \mathcal{O}_p$

assumes *deg* (*SA m*) *f* ≤ *d*
assumes *C* ∈ *poly-res-classes n d*
assumes *is-semialgebraic m B*
shows *is-semialgebraic m* {*x* ∈ *B*. *SA-poly-to-Qp-poly m x f* ∈ *C*}
proof−
obtain *g* **where** *g-def*: *g* = *SA-poly-glue m B f* (**1**$_{UP\ (SA\ m)}$)
by *blast*
have *g-closed*: *g* ∈ *carrier* (*UP* (*SA m*))
unfolding *g-def* **by**(*rule SA-poly-glue-closed*, *rule assms*, *blast*, *rule assms*)
have *g-deg*: *deg* (*SA m*) *g* ≤ *d*
unfolding *g-def* **apply**(*rule SA-poly-glue-deg*, *rule assms*, *blast*, *rule assms*, *rule assms*)
unfolding *deg-one* **by** *blast*
have *0*: {*x* ∈ *B*. *SA-poly-to-Qp-poly m x f* ∈ *C*} = *B* ∩ {*x* ∈ *carrier* (*Q*$_p{}^m$). *SA-poly-to-Qp-poly m x g* ∈ *C*}
apply(*rule equalityI'*, *rule IntI*, *blast*)
unfolding *mem-Collect-eq* **apply**(*rule conjI*)
using *assms is-semialgebraic-closed* **apply** *blast*
unfolding *g-def* **using** *SA-poly-glue-cfs1*[*of f m* **1**$_{UP\ (SA\ m)}$ *B*] *assms*
using *SA-poly-glue-to-Qp-poly1 UPSA.P.cring-simprules*(*6*) **apply** *presburger*
unfolding *g-def* **using** *SA-poly-glue-cfs1*[*of f m* **1**$_{UP\ (SA\ m)}$ *B*] *assms*
using *SA-poly-glue-to-Qp-poly1 UPSA.P.cring-simprules*(*6*)
by (*metis* (*no-types*, *lifting*) *Int-iff mem-Collect-eq*)
have *1*: ⋀*i x*. *x* ∈ *B* ⟹ *g i x* = *f i x*
unfolding *g-def* **by**(*rule SA-poly-glue-cfs1*, *rule assms*, *blast*, *rule assms*, *blast*)
have *2*: ⋀*i x*. *x* ∈ *carrier* (*Q*$_p{}^m$) ⟹ *g i x* ∈ $\mathcal{O}_p$
proof− **fix** *i x* **assume** *A*: *x* ∈ *carrier* (*Q*$_p{}^m$)
show *g i x* ∈ $\mathcal{O}_p$
unfolding *g-def* **apply**(*cases x* ∈ *B*)
using *SA-poly-glue-cfs1*[*of f m* **1**$_{UP\ (SA\ m)}$ *B x i*]
assms(*1*) *assms*(*2*)[*of x i*] *1*
using *UPSA.P.cring-simprules*(*6*) *assms*(*5*) **apply** *presburger*
using *SA-poly-glue-cfs2*[*of g m* **1**$_{UP\ (SA\ m)}$ *B x i*] *g-closed assms A*
by (*metis* (*mono-tags*, *opaque-lifting*) *SA-poly-glue-cfs2 SA-poly-to-Qp-poly-coeff UPSA.P.cring-simprules*(*6*) *carrier-is-semialgebraic g-def integral-on-memE*(*2*) *is-semialgebraic-closed one-integral-on*)
qed
have *3*: ⋀*i x*. *x* ∈ *carrier* (*Q*$_p{}^m$) ⟹ *x* ∉ *B* ⟹ *g i x* = **1**$_{UP\ (SA\ m)}$ *i x*
unfolding *g-def* **apply**(*rule SA-poly-glue-cfs2*[*of f m* **1**$_{UP\ (SA\ m)}$ *B*])
by(*rule assms*, *blast*, *rule assms*, *blast*, *blast*)
have *4*: ⋀*i x*. *x* ∈ *carrier* (*Q*$_p{}^m$) ⟹ *x* ∉ *B* ⟹ *g i x* ∈ $\mathcal{O}_p$
using *3 cfs-one*[*of m*] *2* **by** *blast*
have *5*: ⋀*i x*. *x* ∈ *carrier* (*Q*$_p{}^m$) ⟹ *g i x* ∈ $\mathcal{O}_p$
using *1 4 assms 2* **by** *blast*
have *6*: *is-semialgebraic m* {*x* ∈ *carrier* (*Q*$_p{}^m$). *SA-poly-to-Qp-poly m x g* ∈ *C*}
by(*rule SA-poly-constant-res-class-semialg*[*of - - d - n*], *rule g-closed*, *rule 5*, *blast*, *rule g-deg*, *rule assms*)
show *?thesis* **unfolding** *0*

by(*rule intersection-is-semialg*, *rule assms*, *rule 6*)
qed

lemma *SA-poly-constant-res-class-decomp*:
assumes $f \in carrier\ (UP\ (SA\ m))$
assumes $\bigwedge i\ x.\ x \in B \Longrightarrow f\ i\ x \in \mathcal{O}_p$
assumes $deg\ (SA\ m)\ f \leq d$
assumes *is-semialgebraic m B*
shows $B = (\bigcup\ C \in poly\text{-}res\text{-}classes\ n\ d.\ \{x \in B.\ SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ m\ x\ f \in C\})$
proof(*rule equalityI'*)**fix** *x* **assume** *A*: $x \in B$
obtain *g* **where** *g-def*: $g = SA\text{-}poly\text{-}glue\ m\ B\ f\ (\mathbf{1}_{UP\ (SA\ m)})$
by *blast*
have *g-closed*: $g \in carrier\ (UP\ (SA\ m))$
unfolding *g-def* **by**(*rule SA-poly-glue-closed*, *rule assms*, *blast*, *rule assms*)
have *g-deg*: $deg\ (SA\ m)\ g \leq d$
unfolding *g-def* **apply**(*rule SA-poly-glue-deg*, *rule assms*, *blast*, *rule assms*, *rule assms*)
unfolding *deg-one* **by** *blast*
have *1*: $\bigwedge i\ x.\ x \in B \Longrightarrow g\ i\ x = f\ i\ x$
unfolding *g-def* **by**(*rule SA-poly-glue-cfs1*, *rule assms*, *blast*, *rule assms*, *blast*)
have *2*: $\bigwedge i\ x.\ x \in carrier\ ({Q_p}^m) \Longrightarrow g\ i\ x \in \mathcal{O}_p$
proof− **fix** *i x* **assume** *A*: $x \in carrier\ ({Q_p}^m)$
show $g\ i\ x \in \mathcal{O}_p$
unfolding *g-def* **apply**(*cases* $x \in B$)
using *SA-poly-glue-cfs1*[*of f m* $\mathbf{1}_{UP\ (SA\ m)}$ *B x i*]
assms(*1*) *assms*(*2*)[*of x i*] *1*
using *UPSA.P.cring-simprules*(*6*) *assms*(*4*) **apply** *presburger*
using *SA-poly-glue-cfs2*[*of g m* $\mathbf{1}_{UP\ (SA\ m)}$ *B x i*] *g-closed assms A*
by (*metis* (*mono-tags*, *opaque-lifting*) *SA-poly-glue-cfs2 SA-poly-to-Qp-poly-coeff UPSA.P.cring-simprules*(*6*) *carrier-is-semialgebraic g-def integral-on-memE*(*2*) *is-semialgebraic-closed one-integral-on*)
qed
have *3*: $\bigwedge i\ x.\ x \in carrier\ ({Q_p}^m) \Longrightarrow x \notin B \Longrightarrow g\ i\ x = \mathbf{1}_{UP\ (SA\ m)}\ i\ x$
unfolding *g-def* **apply**(*rule SA-poly-glue-cfs2*[*of f m* $\mathbf{1}_{UP\ (SA\ m)}$ *B*])
by(*rule assms*, *blast*, *rule assms*, *blast*, *blast*)
have *4*: $\bigwedge i\ x.\ x \in carrier\ ({Q_p}^m) \Longrightarrow x \notin B \Longrightarrow g\ i\ x \in \mathcal{O}_p$
using *3 cfs-one*[*of m*] *2* **by** *blast*
have *5*: $\bigwedge i\ x.\ x \in carrier\ ({Q_p}^m) \Longrightarrow g\ i\ x \in \mathcal{O}_p$
using *1 4 assms 2* **by** *blast*
have *6*: $\bigwedge\ i\ x.\ x \in carrier\ ({Q_p}^m) \Longrightarrow SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ m\ x\ g\ i = g\ i\ x$
by(*rule SA-poly-to-Qp-poly-coeff*, *blast*, *rule g-closed*)
have *7*: $\bigwedge\ x.\ x \in carrier\ ({Q_p}^m) \Longrightarrow SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ m\ x\ g \in val\text{-}ring\text{-}polys\text{-}grad\ d$
apply(*rule val-ring-polys-grad-memI*, *rule SA-poly-to-Qp-poly-closed*, *blast*, *rule g-closed*)
unfolding *6* **apply**(*rule 5*, *blast*)
using *g-closed SA-poly-to-Qp-poly-deg-bound*[*of g m*] *g-deg le-trans* **by** *blast*

have *8*: $\bigwedge x.\ x \in carrier\ ({Q_p}^m) \Longrightarrow SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ m\ x\ g \in poly\text{-}res\text{-}class\ n\ d\ (SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ m\ x\ g)$
by(*rule poly-res-class-refl*, *rule 7*)
have *9*: $\bigwedge x.\ x \in carrier\ ({Q_p}^m) \Longrightarrow poly\text{-}res\text{-}class\ n\ d\ (SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ m\ x\ g) \in poly\text{-}res\text{-}classes\ n\ d$
unfolding *poly-res-classes-def* **using** *7* **by** *blast*
have *10*: $\bigwedge x.\ x \in B \Longrightarrow SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ m\ x\ g = SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ m\ x\ f$
unfolding *g-def* **by**(*rule SA-poly-glue-to-Qp-poly1*, *rule assms*, *blast*, *rule assms*, *blast*)
have *11*: $x \in carrier\ ({Q_p}^m)$
using *A is-semialgebraic-closed assms* **by** *blast*
have *12*: $SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ m\ x\ g = SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ m\ x\ f$
by(*rule 10*, *rule A*)
have *13*: $x \in \{x \in B.\ SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ m\ x\ f \in poly\text{-}res\text{-}class\ n\ d\ (SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ m\ x\ g)\}$
using *11 10*[*of x*] *A 9*[*of x*] *8*[*of x*] **unfolding** *12 mem-Collect-eq* **by** *blast*
show $x \in (\bigcup C \in poly\text{-}res\text{-}classes\ n\ d.\ \{x \in B.\ SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ m\ x\ f \in C\})$
using *13 9*[*of x*] *11* **unfolding** *12 mem-simps*(*8*) *mem-Collect-eq* **by** *auto*
next
show $\bigwedge x.\ x \in (\bigcup C \in poly\text{-}res\text{-}classes\ n\ d.\ \{x \in B.\ SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ m\ x\ f \in C\}) \Longrightarrow x \in B$
by *blast*
qed

end

context *UP-cring*
begin

lemma *pderiv-deg-bound*:
assumes $p \in carrier\ P$
assumes $deg\ R\ p \leq (Suc\ d)$
shows $deg\ R\ (pderiv\ p) \leq d$
proof−
have $deg\ R\ p \leq (Suc\ d) \longrightarrow deg\ R\ (pderiv\ p) \leq d$
apply(*rule poly-induct*[*of p*])
apply (*simp add*: *assms*(*1*))
using *deg-zero pderiv-deg-0* **apply** *presburger*
proof fix *p* **assume** *A*: $(\bigwedge q.\ q \in carrier\ P \Longrightarrow deg\ R\ q < deg\ R\ p \Longrightarrow deg\ R\ q \leq Suc\ d \longrightarrow deg\ R\ (pderiv\ q) \leq d)$
$p \in carrier\ P\ 0 < deg\ R\ p\ deg\ R\ p \leq Suc\ d$
obtain *q* **where** *q-def*: $q \in carrier\ P \wedge deg\ R\ q < deg\ R\ p \wedge p = q \oplus_P ltrm\ p$
using *A ltrm-decomp* **by** *metis*
have *0*: $deg\ R\ (pderiv\ (ltrm\ p)) \leq d$
proof−
have *1*: $pderiv\ (ltrm\ p) = up\text{-}ring.monom\ P\ ([deg\ R\ p] \cdot p\ (deg\ R\ p))\ (deg\ R\ p - 1)$
using *pderiv-monom*[*of lcf p deg R p*] *A P-def UP-car-memE*(*1*) **by** *auto*
show *?thesis* **unfolding** *1*

by (*metis* (*no-types*, *lifting*) *A*(*2*) *A*(*3*) *A*(*4*) *R.add-pow-closed Suc-diff-1 cfs-closed deg-monom-le le-trans not-less-eq-eq*)
qed
have *deg R q* ≤ *Suc d* **using** *A q-def* **by** *linarith*
then have *1*: *deg R* (*pderiv q*) ≤ *d*
using *A q-def* **by** *blast*
hence *max* (*deg R* (*pderiv q*)) (*deg R* (*pderiv* (*up-ring.monom P* (*p* (*deg R p*)) (*deg R p*)))) ≤ *d*
using *0 max.bounded-iff* **by** *blast*
thus *deg R* (*pderiv p*) ≤ *d*
using *q-def pderiv-add pderiv-monom*[*of lcf p deg R p*] *A deg-add*[*of pderiv q pderiv* (*ltrm p*)]
by (*metis 0 1 ltrm-closed bound-deg-sum pderiv-closed*)
qed
thus *?thesis*
using *assms*(*2*) **by** *blast*
qed

lemma(**in** *cring*) *minus-zero*:
a ∈ *carrier R* ⟹ *a* ⊖ **0** = *a*
unfolding *a-minus-def*
by (*metis add.l-cancel-one′ cring-simprules*(*2*) *cring-simprules*(*22*))

lemma (**in** *UP-cring*) *taylor-expansion-at-zero*:
assumes *g* ∈ *carrier* (*UP R*)
shows *taylor-expansion R* **0** *g* = *g*
proof−
have *0*: *X-plus* **0** = *X-poly R*
unfolding *X-poly-plus-def*
by (*metis ctrm-degree lcf-eq P.r-zero P-def R.zero-closed UP-cring.ctrm-is-poly UP-cring.to-poly-inverse UP-zero-closed X-closed deg-nzero-nzero is-UP-cring to-fun-ctrm to-fun-zero*)
show *?thesis*
unfolding *taylor-expansion-def 0*
using *assms UP-cring.X-sub is-UP-cring* **by** *blast*
qed
end

14.14 Partitioning Semialgebraic Sets By Zero Sets of Function

context *padic-fields*
begin

definition *SA-funs-to-SA-decomp* **where**
SA-funs-to-SA-decomp n Fs S = *atoms-of* ((∩) *S* ‘ ((*SA-zero-set n* ‘ *Fs*) ∪ (*SA-nonzero-set n* ‘ *Fs*)))

lemma *SA-funs-to-SA-decomp-closed-0*:

assumes *Fs* ⊆ *carrier* (*SA n*)
assumes *is-semialgebraic n S*
shows (∩) *S* ‘ ((*SA-zero-set n* ‘ *Fs*) ∪ (*SA-nonzero-set n* ‘ *Fs*)) ⊆ *semialg-sets n*
proof(*rule subsetI*)
fix *x* **assume** *A*: *x* ∈ (∩) *S* ‘ (*SA-zero-set n* ‘ *Fs* ∪ *SA-nonzero-set n* ‘ *Fs*)
show *x* ∈ *semialg-sets n*
proof(*cases x* ∈ (∩) *S* ‘ *SA-zero-set n* ‘ *Fs*)
case *True*
then obtain *f* **where** *f-def*: *f* ∈ *Fs* ∧ *x* = *S* ∩ *SA-zero-set n f*
by *blast*
then show *x* ∈ *semialg-sets n* **using** *assms SA-zero-set-is-semialgebraic*
by (*meson is-semialgebraicE semialg-intersect subset-iff*)
next
case *False*
then have *x* ∈ (∩) *S* ‘ *SA-nonzero-set n* ‘ *Fs*
using *A* **by** *blast*
then obtain *f* **where** *f-def*: *f* ∈ *Fs* ∧ *x* = *S* ∩ *SA-nonzero-set n f*
using *A* **by** *blast*
then show *x* ∈ *semialg-sets n* **using** *assms SA-nonzero-set-is-semialgebraic*
by (*meson padic-fields.is-semialgebraicE padic-fields.semialg-intersect padic-fields-axioms subset-iff*)
qed
qed

lemma *SA-funs-to-SA-decomp-closed*:
assumes *finite Fs*
assumes *Fs* ⊆ *carrier* (*SA n*)
assumes *is-semialgebraic n S*
shows *SA-funs-to-SA-decomp n Fs S* ⊆ *semialg-sets n*
unfolding *SA-funs-to-SA-decomp-def semialg-sets-def*
apply(*rule atoms-of-gen-boolean-algebra*)
using *SA-funs-to-SA-decomp-closed-0*[*of Fs n S*] *assms* **unfolding** *semialg-sets-def*
apply *blast*
using *assms*
by *blast*

lemma *SA-funs-to-SA-decomp-finite*:
assumes *finite Fs*
assumes *Fs* ⊆ *carrier* (*SA n*)
assumes *is-semialgebraic n S*
shows *finite* (*SA-funs-to-SA-decomp n Fs S*)
unfolding *SA-funs-to-SA-decomp-def*
apply(*rule finite-set-imp-finite-atoms*)
using *assms* **by** *blast*

lemma *SA-funs-to-SA-decomp-disjoint*:
assumes *finite Fs*
assumes *Fs* ⊆ *carrier* (*SA n*)
assumes *is-semialgebraic n S*

shows *disjoint* (*SA-funs-to-SA-decomp n Fs S*)
apply(*rule disjointI*) **unfolding** *SA-funs-to-SA-decomp-def*
apply(*rule atoms-of-disjoint*[*of* - ((∩) *S* ‘ (*SA-zero-set n* ‘ *Fs* ∪ *SA-nonzero-set n* ‘ *Fs*))])
apply *blast* **apply** *blast* **by** *blast*

lemma *pre-SA-funs-to-SA-decomp-in-algebra*:
shows ((∩) *S* ‘ (*SA-zero-set n* ‘ *Fs* ∪ *SA-nonzero-set n* ‘ *Fs*)) ⊆ *gen-boolean-algebra S* (*SA-zero-set n* ‘ *Fs* ∪ *SA-nonzero-set n* ‘ *Fs*)
proof(*rule subsetI*) **fix** *x* **assume** *A*: *x* ∈ (∩) *S* ‘ (*SA-zero-set n* ‘ *Fs* ∪ *SA-nonzero-set n* ‘ *Fs*)
then obtain *A* **where** *A-def*: *A* ∈ (*SA-zero-set n* ‘ *Fs* ∪ *SA-nonzero-set n* ‘ *Fs*) ∧ *x* = *S* ∩ *A*
by *blast*
then show *x* ∈ *gen-boolean-algebra S* (*SA-zero-set n* ‘ *Fs* ∪ *SA-nonzero-set n* ‘ *Fs*)
using *gen-boolean-algebra.generator*[*of A SA-zero-set n* ‘ *Fs* ∪ *SA-nonzero-set n* ‘ *Fs S*]
by (*metis inf.commute*)
qed

lemma *SA-funs-to-SA-decomp-in-algebra*:
assumes *finite Fs*
shows *SA-funs-to-SA-decomp n Fs S* ⊆ *gen-boolean-algebra S* (*SA-zero-set n* ‘ *Fs* ∪ *SA-nonzero-set n* ‘ *Fs*)
unfolding *SA-funs-to-SA-decomp-def* **apply**(*rule atoms-of-gen-boolean-algebra*)
using *pre-SA-funs-to-SA-decomp-in-algebra*[*of S n Fs*] **apply** *blast*
using *assms* **by** *blast*

lemma *SA-funs-to-SA-decomp-subset*:
assumes *finite Fs*
assumes *Fs* ⊆ *carrier* (*SA n*)
assumes *is-semialgebraic n S*
assumes *A* ∈ *SA-funs-to-SA-decomp n Fs S*
shows *A* ⊆ *S*
proof−
have *A* ∈ *gen-boolean-algebra S* (*SA-zero-set n* ‘ *Fs* ∪ *SA-nonzero-set n* ‘ *Fs*)
using *assms SA-funs-to-SA-decomp-in-algebra*[*of Fs n S*]
atoms-of-gen-boolean-algebra[*of* - *S* (*SA-zero-set n* ‘ *Fs* ∪ *SA-nonzero-set n* ‘ *Fs*)]
unfolding *SA-funs-to-SA-decomp-def* **by** *blast*
then show *?thesis* **using** *gen-boolean-algebra-subset* **by** *blast*
qed

lemma *SA-funs-to-SA-decomp-memE*:
assumes *finite Fs*
assumes *Fs* ⊆ *carrier* (*SA n*)
assumes *is-semialgebraic n S*
assumes *A* ∈ (*SA-funs-to-SA-decomp n Fs S*)

assumes $f \in Fs$
shows $A \subseteq$ *SA-zero-set n f* $\vee$ $A \subseteq$ *SA-nonzero-set n f*
proof(*cases* $A \subseteq S \cap$ *SA-zero-set n f*)
case *True*
then show *?thesis*
by *blast*
next
case *False*
have *0*: $S \cap$ *SA-zero-set n f* $\in (\cap)$ *S* ‘ (*SA-zero-set n* ‘ *Fs* $\cup$ *SA-nonzero-set n* ‘ *Fs*)
using *assms*
by *blast*
then have *1*: $A \cap (S \cap$ *SA-zero-set n f*$) = \{\}$
using *False assms atoms-are-minimal*[*of A* ((∩) *S* ‘ (*SA-zero-set n* ‘ *Fs* ∪ *SA-nonzero-set n* ‘ *Fs*)) *S* ∩ *SA-zero-set n f*]
unfolding *SA-funs-to-SA-decomp-def*
by *blast*
have *2*: $A \subseteq S$
using *assms SA-funs-to-SA-decomp-subset* **by** *blast*
then show *?thesis*
using *0 1 False zero-set-nonzero-set-covers-semialg-set*[*of n S*] *assms*(*3*) **by** *auto*
qed

lemma *SA-funs-to-SA-decomp-covers*:
assumes *finite Fs*
assumes $Fs \neq \{\}$
assumes $Fs \subseteq$ *carrier* (*SA n*)
assumes *is-semialgebraic n S*
shows $S = \bigcup$ (*SA-funs-to-SA-decomp n Fs S*)
proof−
have *0*: $S = \bigcup$ ((∩) *S* ‘ ((*SA-zero-set n* ‘ *Fs*) ∪ (*SA-nonzero-set n* ‘ *Fs*)))
proof
obtain *f* **where** *f-def*: $f \in Fs$
using *assms* **by** *blast*
have *0*: $S \cap$ *SA-nonzero-set n f* $\in$ ((∩) *S* ‘ ((*SA-zero-set n* ‘ *Fs*) ∪ (*SA-nonzero-set n* ‘ *Fs*)))
using *f-def* **by** *blast*
have *1*: $S \cap$ *SA-zero-set n f* $\in$ ((∩) *S* ‘ ((*SA-zero-set n* ‘ *Fs*) ∪ (*SA-nonzero-set n* ‘ *Fs*)))
using *f-def* **by** *blast*
have *2*: $S = S \cap$ *SA-zero-set n f* $\cup$ $S \cap$ *SA-nonzero-set n f*
by (*simp add*: *assms*(*4*) *zero-set-nonzero-set-covers-semialg-set*)
then show $S \subseteq \bigcup$ ((∩) *S* ‘ (*SA-zero-set n* ‘ *Fs* ∪ *SA-nonzero-set n* ‘ *Fs*))
using *0 1 Sup-upper2 Un-subset-iff subset-refl* **by** *blast*
show $\bigcup$ ((∩) *S* ‘ (*SA-zero-set n* ‘ *Fs* ∪ *SA-nonzero-set n* ‘ *Fs*)) $\subseteq S$
by *blast*
qed
show *?thesis*

unfolding *SA-funs-to-SA-decomp-def atoms-of-covers'* **using** *0* **by** *blast*
qed

end
end

References

[1] J. Denef. p-adic semi-algebraic sets and cell decomposition. *Journal für die reine und angewandte Mathematik*, 369:154–166, 1986.

[2] A. Engler. *Valued fields*. Springer, Berlin New York, 2005.