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*
⟨*proof*⟩

lemma(**in** *domain*) *nonzero-is-submonoid*: *submonoid R* (*nonzero R*)
⟨*proof*⟩

lemma(**in** *domain*) *nonzero-closed*:
assumes $a \in$ *nonzero R*
shows $a \in$ *carrier R*
⟨*proof*⟩

lemma(**in** *domain*) *nonzero-mult-closed*:
assumes $a \in$ *nonzero R*
assumes $b \in$ *nonzero R*
shows $a \otimes b \in$ *carrier R*
⟨*proof*⟩

lemma(**in** *domain*) *nonzero-one-closed*:
$\mathbf{1} \in$ *nonzero R*
⟨*proof*⟩

lemma(**in** *domain*) *nonzero-memI*:
assumes $a \in$ *carrier R*
assumes $a \neq \mathbf{0}$
shows $a \in$ *nonzero R*
⟨*proof*⟩

lemma(**in** *domain*) *nonzero-memE*:
assumes $a \in$ *nonzero R*
shows $a \in$ *carrier R* $a \neq \mathbf{0}$
⟨*proof*⟩

lemma(**in** *domain*) *not-nonzero-memE*:
assumes $a \notin$ *nonzero R*
assumes $a \in$ *carrier R*

shows $a = \mathbf{0}$
⟨*proof*⟩

lemma(**in** *domain*) *not-nonzero-memI*:
assumes $a = \mathbf{0}$
shows $a \notin$ *nonzero R*
⟨*proof*⟩

lemma(**in** *domain*) *one-nonzero*:
$\mathbf{1} \in$ *nonzero R*
⟨*proof*⟩

lemma(**in** *domain-frac*) *eq-obj-rng-of-frac-nonzero*:
eq-obj-rng-of-frac R (*nonzero R*)
⟨*proof*⟩

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 $a = ((\textit{numer}\ a)\ |_{\textit{rel}}\ (\textit{denom}\ a))$
(*numer a*) $\in$ *carrier R*
(*denom a*) $\in S$
numer a $= \mathbf{0} \Longrightarrow a = \mathbf{0}_{\textit{rec-rng-of-frac}}$
$a \otimes_{\textit{rec-rng-of-frac}}$ (*rng-to-rng-of-frac*(*denom a*)) = *rng-to-rng-of-frac* (*numer a*)
(*rng-to-rng-of-frac*(*denom a*)) $\otimes_{\textit{rec-rng-of-frac}}$ a = *rng-to-rng-of-frac* (*numer a*)
⟨*proof*⟩

lemma(**in** *eq-obj-rng-of-frac*) *frac-zero*:
assumes *domain R*
assumes $\mathbf{0} \notin S$
assumes $a \in$ *carrier R*
assumes $b \in S$
assumes $(a\ |_{\textit{rel}}\ b) = \mathbf{0}_{\textit{rec-rng-of-frac}}$
shows $a = \mathbf{0}$

⟨*proof*⟩

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*⟩

lemma(**in** *eq-obj-rng-of-frac*) *numer-denom-facts*:
  assumes *domain R*
  assumes $\mathbf{0} \notin S$
  assumes $a \in$ *carrier rec-rng-of-frac*
  shows $a = (numer\ a\ |_{rel}\ denom\ a)$
    $(numer\ a) \in carrier\ R$
    $(denom\ a) \in S$
    $a \neq \mathbf{0}_{rec\text{-}rng\text{-}of\text{-}frac} \Longrightarrow (numer\ a) \neq \mathbf{0}$
    $a \otimes_{rec\text{-}rng\text{-}of\text{-}frac} (rng\text{-}to\text{-}rng\text{-}of\text{-}frac(denom\ a)) = rng\text{-}to\text{-}rng\text{-}of\text{-}frac\ (numer\ a)$
    $(rng\text{-}to\text{-}rng\text{-}of\text{-}frac(denom\ a)) \otimes_{rec\text{-}rng\text{-}of\text{-}frac} a = rng\text{-}to\text{-}rng\text{-}of\text{-}frac\ (numer\ a)$
  ⟨*proof*⟩

lemma(**in** *eq-obj-rng-of-frac*) *numer-denom-closed*:
  assumes *domain R*
  assumes $\mathbf{0} \notin S$
  assumes $a \in$ *carrier rec-rng-of-frac*
  shows $(numer\ a,\ denom\ a) \in carrier\ rel$
  ⟨*proof*⟩

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-rng-of-frac*
  shows $a = fraction\ (numer\ a)\ (denom\ a)$
  ⟨*proof*⟩

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 $(\mathit{fraction}\ a\ b) \oplus_{\mathit{rec\text{-}rng\text{-}of\text{-}frac}} (\mathit{fraction}\ c\ d) = (\mathit{fraction}\ ((a \otimes d) \oplus (b \otimes c))\ (b \otimes d))$
$\langle \mathit{proof} \rangle$

lemma(in *eq-obj-rng-of-frac*) *mult-fraction*:
assumes *domain R*
assumes $\mathbf{0} \notin S$
assumes $a \in \mathit{carrier}\ R$
assumes $b \in S$
assumes $c \in \mathit{carrier}\ R$
assumes $d \in S$
shows $(\mathit{fraction}\ a\ b) \otimes_{\mathit{rec\text{-}rng\text{-}of\text{-}frac}} (\mathit{fraction}\ c\ d) = (\mathit{fraction}\ (a \otimes c)\ (b \otimes d))$
$\langle \mathit{proof} \rangle$

lemma(in *eq-obj-rng-of-frac*) *fraction-zero*:
$\mathbf{0}_{\mathit{rec\text{-}rng\text{-}of\text{-}frac}} = \mathit{fraction}\ \mathbf{0}\ \mathbf{1}$
$\langle \mathit{proof} \rangle$

lemma(in *eq-obj-rng-of-frac*) *fraction-zero′*:
assumes $a \in S$
assumes $\mathbf{0} \notin S$
shows $\mathbf{0}_{\mathit{rec\text{-}rng\text{-}of\text{-}frac}} = \mathit{fraction}\ \mathbf{0}\ a$
$\langle \mathit{proof} \rangle$

lemma(in *eq-obj-rng-of-frac*) *fraction-one*:
$\mathbf{1}_{\mathit{rec\text{-}rng\text{-}of\text{-}frac}} = \mathit{fraction}\ \mathbf{1}\ \mathbf{1}$
$\langle \mathit{proof} \rangle$

lemma(in *eq-obj-rng-of-frac*) *fraction-one′*:
assumes *domain R*
assumes $\mathbf{0} \notin S$
assumes $a \in S$
shows $\mathit{fraction}\ a\ a = \mathbf{1}_{\mathit{rec\text{-}rng\text{-}of\text{-}frac}}$
$\langle \mathit{proof} \rangle$

lemma(in *eq-obj-rng-of-frac*) *fraction-closed*:
assumes *domain R*
assumes $\mathbf{0} \notin S$
assumes $a \in \mathit{carrier}\ R$
assumes $b \in S$
shows $\mathit{fraction}\ a\ b \in \mathit{carrier}\ \mathit{rec\text{-}rng\text{-}of\text{-}frac}$
$\langle \mathit{proof} \rangle$

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*)
⟨*proof*⟩

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 $a \in$ *carrier R*
shows ι a = *frac* a **1**
⟨*proof*⟩

lemma(**in** *domain-frac*) *inc-is-hom*:
inc ∈ *ring-hom R* (*Frac R*)
⟨*proof*⟩

lemma(**in** *domain-frac*) *inc-is-hom1*:
ring-hom-ring R (*Frac R*) *inc*
⟨*proof*⟩

Inclusion map is injective:

lemma(**in** *domain-frac*) *inc-inj0*:
a-kernel R (*Frac R*) *inc* = {**0**}
⟨*proof*⟩

lemma(**in** *domain-frac*) *inc-inj1*:

assumes $a \in carrier\ R$
assumes $inc\ a = \mathbf{0}_{Frac\ R}$
shows $a = \mathbf{0}$
⟨*proof*⟩

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*⟩

Image of inclusion map is a subring:

lemma(**in** *domain-frac*) *inc-im-is-subring*:
$subring\ (\iota\ `\ (carrier\ R))\ (Frac\ R)$
⟨*proof*⟩

1.3.3 Basic Properties of numer, denom, and frac

lemma(**in** *domain-frac*) *numer-denom-facts*:
assumes $a \in carrier\ (Frac\ R)$
shows $a = frac\ (numer\ a)\ (denom\ a)$
$(numer\ a) \in carrier\ R$
$(denom\ a) \in nonzero\ R$
$a \neq \mathbf{0}_{Frac\ R} \Longrightarrow numer\ a \neq \mathbf{0}$
$a \otimes_{Frac\ R} (inc\ (denom\ a)) = inc\ (numer\ a)$
⟨*proof*⟩

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 $(frac\ a\ b) \oplus_{Frac\ R} (frac\ c\ d) = (frac\ ((a \otimes d) \oplus (b \otimes c))\ (b \otimes d))$
⟨*proof*⟩

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 $(frac\ a\ b) \otimes_{Frac\ R} (frac\ c\ d) = (frac\ (a \otimes c)\ (b \otimes d))$
⟨*proof*⟩

lemma(**in** *domain-frac*) *frac-one*:
assumes $a \in nonzero\ R$
shows $frac\ a\ a = \mathbf{1}_{Frac\ R}$
⟨*proof*⟩

lemma(**in** *domain-frac*) *frac-closed*:
  assumes $a \in carrier\ R$
  assumes $b \in nonzero\ R$
  shows $frac\ a\ b \in carrier\ (Frac\ R)$
  ⟨*proof*⟩

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*⟩

lemma(**in** *comm-monoid*) *UnitsI*:
  assumes $a \in carrier\ G$
  assumes $b \in carrier\ G$
  assumes $a \otimes b = \mathbf{1}$
  shows $a \in Units\ G\ b \in Units\ G$
  ⟨*proof*⟩

lemma(**in** *comm-monoid*) *is-invI*:
  assumes $a \in carrier\ G$
  assumes $b \in carrier\ G$
  assumes $a \otimes b = \mathbf{1}$
  shows $inv_G\ b = a\ inv_G\ a = b$
  ⟨*proof*⟩

lemma(**in** *ring*) *ring-in-Units-imp-not-zero*:
  assumes $\mathbf{1} \neq \mathbf{0}$
  assumes $a \in Units\ R$
  shows $a \neq \mathbf{0}$
  ⟨*proof*⟩

lemma(**in** *domain-frac*) *in-Units-imp-not-zero*:
  assumes $a \in Units\ R$
  shows $a \neq \mathbf{0}$
  ⟨*proof*⟩

lemma(**in** *domain-frac*) *units-of-fraction-field0*:
  assumes $a \in carrier\ (Frac\ R)$
  assumes $a \neq \mathbf{0}_{Frac\ R}$
  shows $inv_{Frac\ R}\ a = frac\ (denom\ a)\ (numer\ a)$
    $a \otimes_{Frac\ R} (inv_{Frac\ R}\ a) = \mathbf{1}_{Frac\ R}$
    $(inv_{Frac\ R}\ a) \otimes_{Frac\ R} a = \mathbf{1}_{Frac\ R}$
⟨*proof*⟩

lemma(**in** *domain-frac*) *units-of-fraction-field*:
$Units\ (Frac\ R) = carrier\ (Frac\ R) - \{\mathbf{0}_{Frac\ R}\}$
⟨*proof*⟩

1.4 The Fraction Field as a Field

lemma(**in** *domain-frac*) *fraction-field-is-field*:
field (*Frac R*)
⟨*proof*⟩

lemma(**in** *domain-frac*) *frac-inv*:
assumes $a \in$ *nonzero R*
assumes $b \in$ *nonzero R*
shows $inv_{Frac\ R}$ (*frac a b*) = (*frac b a*)
⟨*proof*⟩

lemma(**in** *domain-frac*) *frac-inv-id*:
assumes $a \in$ *nonzero R*
assumes $b \in$ *nonzero R*
assumes $c \in$ *nonzero R*
assumes $d \in$ *nonzero R*
assumes *frac a b* = *frac c d*
shows *frac b a* = *frac d c*
⟨*proof*⟩

lemma(**in** *domain-frac*) *frac-uminus*:
assumes $a \in$ *carrier R*
assumes $b \in$ *nonzero R*
shows $\ominus_{Frac\ R}$ (*frac a b*) = *frac* ($\ominus$ *a*) *b*
⟨*proof*⟩

lemma(**in** *domain-frac*) *frac-eqI*:
assumes $a \in$ *carrier R*
assumes $b \in$ *nonzero R*
assumes $c \in$ *carrier R*
assumes $d \in$ *nonzero R*
assumes $a \otimes d \ominus b \otimes c = \mathbf{0}$
shows *frac a b* = *frac c d*
⟨*proof*⟩

lemma(**in** *domain-frac*) *frac-eqI'*:
assumes $a \in$ *carrier R*
assumes $b \in$ *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*
⟨*proof*⟩

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*

$\langle proof \rangle$

lemma(**in** *domain-frac*) *frac-cancel-r*:
  assumes $a \in nonzero\ R$
  assumes $b \in nonzero\ R$
  assumes $c \in carrier\ R$
  shows $frac\ (c \otimes a)\ (b \otimes a) = frac\ c\ b$
  $\langle proof \rangle$

lemma(**in** *domain-frac*) *frac-cancel-lr*:
  assumes $a \in nonzero\ R$
  assumes $b \in nonzero\ R$
  assumes $c \in carrier\ R$
  shows $frac\ (a \otimes c)\ (b \otimes a) = frac\ c\ b$
  $\langle proof \rangle$

lemma(**in** *domain-frac*) *frac-cancel-rl*:
  assumes $a \in nonzero\ R$
  assumes $b \in nonzero\ R$
  assumes $c \in carrier\ R$
  shows $frac\ (c \otimes a)\ (a \otimes b) = frac\ c\ b$
  $\langle proof \rangle$

lemma(**in** *domain-frac*) *i-mult*:
  assumes $a \in carrier\ R$
  assumes $c \in carrier\ R$
  assumes $d \in nonzero\ R$
  shows $(\iota\ a) \otimes_{Frac\ R} (frac\ c\ d) = frac\ (a \otimes c)\ d$
$\langle proof \rangle$

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 $a \otimes d = b \otimes c$
$\langle proof \rangle$

lemma(**in** *domain-frac*) *frac-add-common-denom*:
  assumes $a \in carrier\ R$
  assumes $b \in carrier\ R$
  assumes $c \in nonzero\ R$
  shows $(frac\ a\ c) \oplus_{Frac\ R} (frac\ b\ c) = frac\ (a \oplus b)\ c$
$\langle proof \rangle$

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*⟩

sublocale *domain-frac* < *F*: *field Frac R*
⟨*proof*⟩

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

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

lemma(**in** *domain-frac*) *nat-suc*:
$[Suc\ n]\cdot\mathbf{1} = \mathbf{1} \oplus [n]\cdot\mathbf{1}$
⟨*proof*⟩

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*⟩

lemma(**in** *domain-frac*) *pow-0*:
assumes $a \in nonzero\ R$
shows $a[\widehat{\ }](0::nat) = \mathbf{1}$
⟨*proof*⟩

lemma(**in** *domain-frac*) *pow-suc*:
assumes $a \in carrier\ R$
shows $a[\widehat{\ }](Suc\ n) = a \otimes (a[\widehat{\ }]n)$
⟨*proof*⟩

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

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

lemma(**in** *domain-frac*) *nat-inc-mult*:
$[((n::nat) * (m::nat))]\cdot\mathbf{1} = [n]\cdot\mathbf{1} \otimes [m]\cdot\mathbf{1}$
⟨*proof*⟩

lemma(**in** *domain-frac*) *int-inc-mult*:
$[((n::int) * (m::int))]\cdot\mathbf{1} = [n]\cdot\mathbf{1} \otimes [m]\cdot\mathbf{1}$

⟨*proof*⟩

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*⟩

lemma(**in** *ring*) *Units-inverse*:
assumes $u \in$ *Units R*
shows *inv* $u \in$ *Units R*
⟨*proof*⟩

lemma(**in** *cring*) *invI*:
assumes $x \in$ *carrier R*
assumes $y \in$ *carrier R*
assumes $x \otimes_R y = \mathbf{1}_R$
shows $y = inv\ _R\ x$
$x = inv\ _R\ y$
⟨*proof*⟩

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 $inv_R\ x \otimes_R y = z$
$y \otimes_R (inv_R\ x) = z$
⟨*proof*⟩

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 $inv_R\ x \otimes_R y = z$
$y \otimes_R (inv_R\ x) = z$
⟨*proof*⟩

lemma(**in** *domain-frac*) *nat-pow-nonzero*:
assumes $x \in$*nonzero R*
shows $x[\widehat{\ }](n{::}nat) \in$ *nonzero R*
⟨*proof*⟩

lemma(**in** *monoid*) *Units-int-pow-closed*:
assumes $x \in$ *Units G*
shows $x[\widehat{\ }](n{::}int) \in$ *Units G*

⟨*proof*⟩

1.6 Facts About Fraction Field Units

lemma(**in** *domain-frac*) *frac-nonzero-Units*:
nonzero (*Frac R*) = *Units* (*Frac R*)
⟨*proof*⟩

lemma(**in** *domain-frac*) *frac-nonzero-inv-Unit*:
assumes $b \in \mathit{nonzero}\ (\mathit{Frac}\ R)$
shows $\mathit{inv}_{\mathit{Frac}\ R}\ b \in \mathit{Units}\ (\mathit{Frac}\ R)$
⟨*proof*⟩

lemma(**in** *domain-frac*) *frac-nonzero-inv-closed*:
assumes $b \in \mathit{nonzero}\ (\mathit{Frac}\ R)$
shows $\mathit{inv}_{\mathit{Frac}\ R}\ b \in \mathit{carrier}\ (\mathit{Frac}\ R)$
⟨*proof*⟩

lemma(**in** *domain-frac*) *frac-nonzero-inv*:
assumes $b \in \mathit{nonzero}\ (\mathit{Frac}\ R)$
shows $b \otimes_{\mathit{Frac}\ R} \mathit{inv}\ _{\mathit{Frac}\ R}\ b = \mathbf{1}_{\mathit{Frac}\ R}$
$\mathit{inv}\ _{\mathit{Frac}\ R}\ b \otimes_{\mathit{Frac}\ R}\ b = \mathbf{1}_{\mathit{Frac}\ R}$
⟨*proof*⟩

lemma(**in** *domain-frac*) *fract-cancel-right*[*simp*]:
assumes $a \in \mathit{carrier}\ (\mathit{Frac}\ R)$
assumes $b \in \mathit{nonzero}\ (\mathit{Frac}\ R)$
shows $b \otimes_{\mathit{Frac}\ R} (a \otimes_{\mathit{Frac}\ R} \mathit{inv}_{\mathit{Frac}\ R}\ b) = a$
⟨*proof*⟩

lemma(**in** *domain-frac*) *fract-cancel-left*[*simp*]:
assumes $a \in \mathit{carrier}\ (\mathit{Frac}\ R)$
assumes $b \in \mathit{nonzero}\ (\mathit{Frac}\ R)$
shows $(a \otimes_{\mathit{Frac}\ R} \mathit{inv}_{\mathit{Frac}\ R}\ b) \otimes_{\mathit{Frac}\ R}\ b = a$
⟨*proof*⟩

lemma(**in** *domain-frac*) *fract-mult-inv*:
assumes $b \in \mathit{nonzero}\ (\mathit{Frac}\ R)$
assumes $d \in \mathit{nonzero}\ (\mathit{Frac}\ R)$
shows $(\mathit{inv}_{\mathit{Frac}\ R}\ b) \otimes_{\mathit{Frac}\ R} (\mathit{inv}_{\mathit{Frac}\ R}\ d) = (\mathit{inv}_{\mathit{Frac}\ R}\ (b \otimes_{\mathit{Frac}\ R} d))$
⟨*proof*⟩

lemma(**in** *domain-frac*) *fract-mult*:
assumes $a \in \mathit{carrier}\ (\mathit{Frac}\ R)$
assumes $b \in \mathit{nonzero}\ (\mathit{Frac}\ R)$
assumes $c \in \mathit{carrier}\ (\mathit{Frac}\ R)$
assumes $d \in \mathit{nonzero}\ (\mathit{Frac}\ R)$
shows $(a \otimes_{\mathit{Frac}\ R} \mathit{inv}_{\mathit{Frac}\ R}\ b) \otimes_{\mathit{Frac}\ R} (c \otimes_{\mathit{Frac}\ R} \mathit{inv}_{\mathit{Frac}\ R}\ d) = ((a \otimes_{\mathit{Frac}\ R}$
$c) \otimes_{\mathit{Frac}\ R} \mathit{inv}_{\mathit{Frac}\ R}\ (b \otimes_{\mathit{Frac}\ R} d))$

$\langle proof \rangle$

lemma(**in** *domain-frac*) *Frac-nat-pow-nonzero*:
assumes $x \in$ *nonzero* (*Frac R*)
shows $x[\lceil]_{Frac\ R}(n::nat) \in$ *nonzero* (*Frac R*)
$\langle proof \rangle$

lemma(**in** *domain-frac*) *Frac-nonzero-nat-pow*:
assumes $x \in$ *carrier* (*Frac R*)
assumes $n > 0$
assumes $x[\lceil]_{Frac\ R}(n::nat) \in$ *nonzero* (*Frac R*)
shows $x \in$ *nonzero* (*Frac R*)
$\langle proof \rangle$

lemma(**in** *domain-frac*) *Frac-int-pow-nonzero*:
assumes $x \in$ *nonzero* (*Frac R*)
shows $x[\lceil]_{Frac\ R}(n::int) \in$ *nonzero* (*Frac R*)
$\langle proof \rangle$

lemma(**in** *domain-frac*) *Frac-nonzero-int-pow*:
assumes $x \in$ *carrier* (*Frac R*)
assumes $n > 0$
assumes $x[\lceil]_{Frac\ R}(n::int) \in$ *nonzero* (*Frac R*)
shows $x \in$ *nonzero* (*Frac R*)
$\langle proof \rangle$

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*
$\langle proof \rangle$

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*))
$\langle proof \rangle$

lemma(**in** *domain-frac*) *numer-nonzero*:
assumes $a \in$ *nonzero* (*Frac R*)
shows *numer* $a \in$ *nonzero R*
$\langle proof \rangle$

lemma(**in** *domain-frac*) *fraction-zero*[*simp*]:
assumes $b \in$ *nonzero R*
shows *frac* $\mathbf{0}$ $b = \mathbf{0}_{Frac\ R}$
$\langle proof \rangle$

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* $\Rightarrow$ $'b$
type-synonym $('a,'b)$ *ring-hom* = $'a \Rightarrow 'b$
type-synonym $'a$ *u-poly* = *nat* $\Rightarrow$ $'a$

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* $m\ i \leq K$
  ⟨*proof*⟩

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

definition *monomials-of* :: $('a,'c)$ *ring-scheme* $\Rightarrow$ $('a, 'b)$ *mvar-poly* $\Rightarrow$ $('b$ *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* $m\ i = 0$
  ⟨*proof*⟩

lemma *index-freeI*:
  assumes $\bigwedge m.\ m \in \textit{monomials-of}\ R\ P \Longrightarrow \textit{count}\ m\ i = 0$
  shows *index-free* $P\ i$
  ⟨*proof*⟩

`monomials_of` R is subadditive

lemma *monomials-of-add*:
$\textit{monomials-of}\ R\ (P \bigoplus\ Q) \subseteq (\textit{monomials-of}\ R\ P) \cup (\textit{monomials-of}\ R\ Q)$
⟨*proof*⟩

lemma *monomials-of-add-finite*:
  assumes *finite* $(\textit{monomials-of}\ R\ P)$
  assumes *finite* $(\textit{monomials-of}\ R\ Q)$
  shows *finite* $(\textit{monomials-of}\ R\ (P \bigoplus\ Q))$
  ⟨*proof*⟩

lemma *monomials-ofE*:
  assumes $m \in \textit{monomials-of}\ R\ p$
  shows $p\ m \neq \mathbf{0}$
  ⟨*proof*⟩

lemma *complement-of-monomials-of*:
  assumes $m \notin \textit{monomials-of}\ R\ P$
  shows $P\ m = \mathbf{0}$
  ⟨*proof*⟩

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

lemma *monomials-of-p-mult*:
$\textit{monomials-of}\ R\ (P \bigotimes\ i) = \{m.\ \exists\ n \in (\textit{monomials-of}\ R\ P).\ m = \textit{add-mset}\ i\ n\}$
⟨*proof*⟩

lemma *monomials-of-p-mult'*:
$\textit{monomials-of}\ R\ (p \bigotimes\ i) = \textit{add-mset}\ i\ \text{‘}\ (\textit{monomials-of}\ R\ p)$
  ⟨*proof*⟩

lemma *monomials-of-p-mult-finite*:
  assumes *finite* $(\textit{monomials-of}\ R\ P)$
  shows *finite* $(\textit{monomials-of}\ R\ (P \bigotimes\ i))$
  ⟨*proof*⟩

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

lemma *monomials-of-const*:
$(\textit{monomials-of}\ R\ (\textit{indexed-const}\ k)) = (\textit{if}\ (k = \mathbf{0})\ \textit{then}\ \{\}\ \textit{else}\ \{\{\#\}\})$
  ⟨*proof*⟩

lemma *monomials-of-const-finite*:
finite $(\textit{monomials-of}\ R\ (\textit{indexed-const}\ k))$
  ⟨*proof*⟩

A polynomial always has finitely many monomials:

lemma *monomials-finite*:
  assumes $P \in$ *indexed-pset K I*
  shows *finite* (*monomials-of R P*)
  ⟨*proof*⟩
end

2.2 Turning monomials into polynomials

Constructor for turning a monomial into a polynomial

definition *mset-to-IP* :: ($'a$, $'b$) *ring-scheme* $\Rightarrow$ $'c$ *monomial* $\Rightarrow$ ($'a$,$'c$) *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* $\Rightarrow$ $'c$ $\Rightarrow$ ($'a$,$'c$) *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**
  ⟨*proof*⟩

lemma *mset-to-IP-simp'*[*simp*]:
  assumes $n \neq m$
  shows *mset-to-IP R m n* = **0**
  ⟨*proof*⟩

lemma(**in** *cring*) *monomials-of-mset-to-IP*:
  assumes **1** $\neq$**0**
  shows *monomials-of R* (*mset-to-IP R m*) = {*m*}
  ⟨*proof*⟩

end

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

definition *monomials-with* :: ($'a$, $'b$) *ring-scheme* $\Rightarrow$ $'c$ $\Rightarrow$ ($'a$,$'c$) *mvar-poly* $\Rightarrow$ ($'c$ *monomial*) *set* **where**
monomials-with R i P = {*m*. $m \in$ *monomials-of R P* $\wedge$ $i \in\#$ *m*}

context *ring*
begin

lemma *monomials-withE*:
  assumes $m \in$ *monomials-with R i P*
  shows $i \in\#$ *m*

$m \in$ *monomials-of R P*
⟨*proof*⟩

lemma *monomials-withI*:
assumes $i \in\# m$
assumes $m \in$ *monomials-of R P*
shows $m \in$ *monomials-with R i P*
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *restrict-poly-to-monom-set-coeff'*:
assumes $P \in$ *indexed-pset K I*
assumes $I \neq \{\}$
assumes $\bigwedge m.\ P\ m \in S$
assumes $\mathbf{0} \in S$
shows $\bigwedge m.$ (*restrict-poly-to-monom-set R P m-set m*) $\in S$
⟨*proof*⟩

lemma *restrict-poly-to-monom-set-monoms*:
assumes $P \in$ *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*
⟨*proof*⟩
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*
⟨*proof*⟩

Total degree function

definition *total-degree* :: $('a, 'b)$ *ring-scheme* $\Rightarrow$ $('a, 'c)$ *mvar-poly* $\Rightarrow$ *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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩
end

2.3.2 Degree in One Variable

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

context *ring*
begin

lemma *degree-in-var-ineq*:
assumes $m \in$ *monomials-of R P*
assumes *finite* (*monomials-of R P*)
shows *degree-in-var R P i* $\geq$ *count m i*
⟨*proof*⟩

lemma *degree-in-var-in-monomials-of*:
assumes *monomials-of R P* $\neq$ {}
assumes *finite* (*monomials-of R P*)
obtains *m* **where** $m \in$ *monomials-of R P* $\wedge$ *count m i* = *degree-in-var R P i*
⟨*proof*⟩

lemma *degree-in-varI*:
  assumes *finite* (*monomials-of R P*)
  assumes $\exists\, m.\ m \in$ *monomials-of R P* $\wedge$ *count m i* $= n$
  assumes $\bigwedge c.\ c \in$ *monomials-of R P* $\Longrightarrow$ *count c i* $\leq n$
  shows $n =$ *degree-in-var R P i*
⟨*proof*⟩

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* $\geq$ *degree-in-var R P i*
⟨*proof*⟩
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*
  ⟨*proof*⟩

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*
  ⟨*proof*⟩
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*
  ⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *fixed-degree-in-varI*:
assumes $m \in$ *monomials-of R P*
assumes *count m i* = *n*
shows $m \in$ *fixed-degree-in-var R P i n*
⟨*proof*⟩

lemma *fixed-degree-in-varE*:
assumes $m \in$ *fixed-degree-in-var R P i n*
shows $m \in$ *monomials-of R P*
count m i = *n*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *index-free-degree-in-var*[*simp*]:
assumes *index-free P i*
shows *degree-in-var R P i* = *0*
⟨*proof*⟩

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*
⟨*proof*⟩

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*:
$n \in (\textit{mset-factors } m) \longleftrightarrow n \subseteq\# \ m$
⟨*proof*⟩

lemma *mset-factors-mono*:
assumes $n \subseteq\# \ m$
shows $\textit{mset-factors } n \subseteq \textit{mset-factors } m$
⟨*proof*⟩

lemma *mset-factors-size-bound*:
assumes $n \in \textit{mset-factors } m$
shows $\textit{size } n \leq \textit{size } m$
⟨*proof*⟩

lemma *sets-to-inds-finite*:
assumes *finite I*
shows $\textit{finite } S \Longrightarrow \textit{finite } (Pi_E \ S \ (\lambda\text{-}. \ I))$
⟨*proof*⟩

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

lemma *mset-factors-to-fun-prop*:
assumes $\textit{size } m = n$
shows $\textit{mset-factors-to-fun } R \ m \in (\textit{mset-factors } m) \rightarrow (Pi_E \ (\textit{set-mset } m) \ (\lambda\text{-}. \ \{0.. \ n\}))$
⟨*proof*⟩

lemma *mset-factors-to-fun-inj*:
shows $\textit{inj-on } (\textit{mset-factors-to-fun } R \ m) \ (\textit{mset-factors } m)$
⟨*proof*⟩

lemma *finite-target*:
finite $(Pi_E$ *(set-mset m)* $(\lambda$-. $\{0..(n::nat)\}))$
⟨*proof*⟩

A multiset has only finitely many factors:

lemma *mset-factors-finite*[*simp*]:
finite (*mset-factors m*)
⟨*proof*⟩

end

2.4.3 Definition of Indexed Polynomial Multiplication.

context *ring*
begin

Monomial division:

lemma *monom-divide*:
  assumes $n \in$ *mset-factors m*
  shows (*THE k.* $n + k = m$) $= m - n$
  ⟨*proof*⟩

A monomial is a factor of itself:

lemma *m-factor*[*simp*]:
$m \in$ *mset-factors m*
  ⟨*proof*⟩

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* (λx. (*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*⟩

lemma *P-ring-ldistr*:
  assumes *carrier-coeff a*
  assumes *carrier-coeff b*
  assumes *carrier-coeff c*
  shows $(b \bigoplus\ c) \bigotimes_p\ a = (b \bigotimes_p\ a) \bigoplus\ (c \bigotimes_p\ a)$
⟨*proof*⟩
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*⟩

end

2.4.6 Associativity of Polynomial Multiplication.

context *ring*
begin

lemma *finsum-eq*:
  assumes $f \in S \to$ *carrier R*
  assumes $g \in S \to$ *carrier R*
  assumes $(\lambda\ x \in S.\ f\ x) = (\lambda\ x \in S.\ g\ x)$
  shows *finsum R f S = finsum R g S*
    ⟨*proof*⟩

lemma *finsum-eq-induct*:
  assumes $f \in S \to$ *carrier R*
  assumes $g \in T \to$ *carrier R*
  assumes *finite S*
  assumes *finite T*
  assumes *bij-betw h S T*
  assumes $\bigwedge s.\ s \in S \Longrightarrow f\ s = g\ (h\ s)$
  shows *finite U* $\Longrightarrow U \subseteq S \Longrightarrow$ *finsum R f U = finsum R g* (*h* ‘ *U*)
  ⟨*proof*⟩

lemma *finsum-bij-eq*:
  assumes $f \in S \to$ *carrier R*
  assumes $g \in T \to$ *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*⟩

lemma *monom-comp*:
assumes $x \subseteq\#\ m$
assumes $y \subseteq\#\ m - x$
shows $x \subseteq\#\ m - y$
⟨*proof*⟩

lemma *monom-comp′*:
assumes $x \subseteq\#\ m$
assumes $y = m - x$
shows $x = m - y$
⟨*proof*⟩

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 $R\ (\lambda\ c.\ g\ (fst\ c)\ (snd\ c))\ (P\ \ T)$
⟨*proof*⟩

lemma *double-finsum*:
assumes *finite* S
assumes $\bigwedge s.\ s \in S \Longrightarrow$ *finite* $(F\ s)$
assumes $P = \{(s,\ t).\ s \in S \wedge\ \ t \in (F\ s)\}$
assumes $\bigwedge s\ y.\ s \in S \Longrightarrow y \in (F\ s) \Longrightarrow g\ s\ y \in$ *carrier* R
shows *finsum* $R\ (\lambda s.$ *finsum* $R\ (g\ s)\ (F\ s))\ S =$
finsum $R\ (\lambda\ p.\ g\ (fst\ p)\ (snd\ p))\ P$
⟨*proof*⟩

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* $\Rightarrow$ $('c$ *monomial* $\times$ $'c$ *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 $m = \{(x,\ y).\ x \in$ *mset-factors* $m \wedge y \in$ *mset-factors* $x\}$
⟨*proof*⟩

lemma *right-cuts-finite*:
finite (*right-cuts* m)
⟨*proof*⟩

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 $\bigwedge s\ y.\ s \in$ *mset-factors* $m \Longrightarrow y \in$ *mset-factors* $(m - x)$
$\Longrightarrow g\ s\ y \in$ *carrier* R
⟨*proof*⟩

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 $\bigwedge s\ y.\ s \in$ *mset-factors* $m \Longrightarrow y \in$ *mset-factors* $x \Longrightarrow g\ s\ y \in$ *carrier* R
⟨*proof*⟩
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 $m = \{(x,\ y).\ x \in$ *mset-factors* $m \wedge y \in$ *mset-factors* $(m - x)\}$
⟨*proof*⟩

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)
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*)
⟨*proof*⟩

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*⟩

2.4.8 Closure properties for multiplication

Building monomials from polynomials:

lemma *indexed-const-P-mult-eq*:
assumes $a \in \textit{carrier}\ R$
assumes $b \in \textit{carrier}\ R$
shows $(\textit{indexed-const}\ a) \bigotimes_p (\textit{indexed-const}\ b) = \textit{indexed-const}\ (a \otimes b)$
⟨*proof*⟩

lemma *indexed-const-P-multE*:
assumes $P \in \textit{indexed-pset}\ I\ (\textit{carrier}\ R)$
assumes $c \in \textit{carrier}\ R$

shows $(P \bigotimes_p (\textit{indexed-const } c))\ m = \ (P\ m) \otimes c$
⟨*proof*⟩

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*⟩

lemma *indexed-const-P-mult-closed*:
assumes $a \in \textit{indexed-pset } I\ (\textit{carrier } R)$
assumes $c \in \textit{carrier } R$
shows $a \bigotimes_p (\textit{indexed-const } c) \in \textit{indexed-pset } I\ (\textit{carrier } R)$
⟨*proof*⟩

lemma *monom-add-mset*:
$\textit{mset-to-IP } R\ (\textit{add-mset } i\ m) = \textit{mset-to-IP } R\ m \bigotimes\ i$
⟨*proof*⟩

Monomials are closed under multiplication:

lemma *monom-mult*:
$\textit{mset-to-IP } R\ m \bigotimes_p \textit{mset-to-IP } R\ n = \textit{mset-to-IP } R\ (m + n)$
⟨*proof*⟩

lemma *poly-index-mult*:
assumes $a \in \textit{indexed-pset } I\ (\textit{carrier } R)$
assumes $i \in I$
shows $a \bigotimes\ i = a \bigotimes_p \textit{mset-to-IP } R\ \{\#i\#\}$
⟨*proof*⟩

lemma *mset-to-IP-mult-closed*:
assumes $a \in \textit{indexed-pset } I\ (\textit{carrier } R)$
shows $\textit{set-mset } m \subseteq I \Longrightarrow a \bigotimes_p \textit{mset-to-IP } R\ m \in \textit{indexed-pset } I\ (\textit{carrier } R)$
⟨*proof*⟩

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**
$\textit{monoms-in } I\ P \equiv (\forall m \in \textit{monomials-of } R\ P.\ \textit{set-mset } m \subseteq I)$

lemma *monoms-of-const*:
$\textit{monomials-of } R\ (\textit{indexed-const } k) = (\textit{if } k = \mathbf{0}\ \textit{then } \{\}\ \textit{else } \{\{\#\}\})$
⟨*proof*⟩

lemma *const-monoms-in*:
$\textit{monoms-in } I\ (\textit{indexed-const } k)$
⟨*proof*⟩

lemma *mset-to-IP-indices*:
shows $P \in$ *indexed-pset* I (*carrier* R) $\Longrightarrow$ *monoms-in* I P
⟨*proof*⟩

lemma *mset-to-IP-indices'*:
assumes $P \in$ *indexed-pset* I (*carrier* R)
assumes $m \in$ *monomials-of* R P
shows *set-mset* $m \subseteq I$
⟨*proof*⟩

lemma *one-mset-to-IP*:
mset-to-IP R {#} = *indexed-const* **1**
⟨*proof*⟩

lemma *mset-to-IP-closed*:
shows *set-mset* $m \subseteq I \Longrightarrow$ *mset-to-IP* R $m \in$ *indexed-pset* I (*carrier* R)
⟨*proof*⟩

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)
⟨*proof*⟩

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)
⟨*proof*⟩

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*⟩

lemma *monomials-of-card-zero*:
assumes $Q \in$ *indexed-pset* I (*carrier* R) $\wedge$ *card* (*monomials-of* R Q) = *0*
shows Q = *indexed-const* **0**
⟨*proof*⟩

Polynomial induction on the number of monomials with nonzero coefficient:

lemma *mpoly-induct'*:
assumes P (*indexed-const* $\mathbf{0}$)
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) $=$ (*Suc* n) $\Longrightarrow P$ Q)
assumes $Q \in$ *indexed-pset* I (*carrier* R)
shows P Q
⟨*proof*⟩

lemma *monomial-poly-split*:
assumes $P \in$ *indexed-pset* I (*carrier* R)
assumes $m \in$ *monomials-of* R P
shows (*restrict-poly-to-monom-set* R P ((*monomials-of* R P) $-$ $\{m\}$)) $\bigoplus$ ((*mset-to-IP* R m) $\bigotimes_p$ (*indexed-const* (P m))) $= P$
⟨*proof*⟩

lemma *restrict-not-in-monoms*:
assumes $a \notin$ *monomials-of* R P
shows *restrict-poly-to-monom-set* R P $A =$ *restrict-poly-to-monom-set* R P (*insert* a A)
⟨*proof*⟩

lemma *restriction-closed'*:
assumes $P \in$ *indexed-pset* I (*carrier* R)
assumes *finite ms*
shows (*restrict-poly-to-monom-set* R P *ms*) $\in$ *indexed-pset* I (*carrier* R)
⟨*proof*⟩

lemma *restriction-restrict*:
restrict-poly-to-monom-set R P *ms* $=$ *restrict-poly-to-monom-set* R P (*ms* $\cap$ *monomials-of* R P)
⟨*proof*⟩

lemma *restriction-closed*:
assumes $P \in$ *indexed-pset* I (*carrier* R)
assumes $Q =$ *restrict-poly-to-monom-set* R P *ms*
shows $Q \in$ *indexed-pset* I (*carrier* R)
⟨*proof*⟩

lemma *monomial-split-card*:
assumes $P \in$ *indexed-pset* I (*carrier* R)
assumes $m \in$ *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*⟩

lemma *P-ring-mult-closed'*:

assumes $a \in$ *indexed-pset* I (*carrier* R)
assumes $b \in$ *indexed-pset* I (*carrier* R)
shows $a \bigotimes_p b \in$ *indexed-pset* I (*carrier* R)
⟨*proof*⟩

end

2.6 Subtraction of Polynomials

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

context *ring*

begin

lemma *P-ring-uminus-eq*:
assumes $a \in$ *indexed-pset* I (*carrier* R)
shows *P-ring-uminus* R $a = a \bigotimes_p$ (*indexed-const* ($\ominus$ **1**))
⟨*proof*⟩

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)
⟨*proof*⟩

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*⟩

multiplication by 1

lemma *one-mult-left*:
assumes $a \in$ *indexed-pset* I (*carrier* R)
shows (*indexed-const* **1**) $\bigotimes_p a = a$
⟨*proof*⟩

end

2.7 The Carrier of the Ring of Indexed Polynomials

abbreviation(*input*) *Pring-set* **where**
Pring-set R $I \equiv$ *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*⟩

lemma(**in** *ring*) *Pring-cfs-closed*:
assumes $P \in$ *Pring-set* R I
shows $P\ m \in$ *carrier* R
⟨*proof*⟩

lemma *indexed-pset-mono-aux*:
assumes $P \in$ *indexed-pset* I S
shows $S \subseteq T \Longrightarrow P \in$ *indexed-pset* I T
⟨*proof*⟩

lemma *indexed-pset-mono*:
assumes $S \subseteq T$
shows *indexed-pset* I $S \subseteq$ *indexed-pset* I T
⟨*proof*⟩

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 $R\ c\ P = (\lambda\ m.\ c \otimes_R P\ m)$

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*⟩

lemma(**in** *cring*) *poly-scalar-mult-const*:
assumes $c \in$ *carrier* R
assumes $k \in$ *carrier* R
shows *poly-scalar-mult* $R\ k$ (*indexed-const* c) = *indexed-const* $(k \otimes c)$
⟨*proof*⟩

lemma(**in** *cring*) *poly-scalar-mult-closed*:
assumes $c \in$ *carrier* R
assumes $P \in$ *Pring-set* R I
shows *poly-scalar-mult* $R\ c\ P \in$ *Pring-set* R I
⟨*proof*⟩

lemma(**in** *cring*) *poly-scalar-mult-zero*:

assumes $P \in$ *Pring-set R I*
shows *poly-scalar-mult R* $\mathbf{0}$ $P =$ *indexed-const* $\mathbf{0}$
⟨*proof*⟩

lemma(**in** *cring*) *poly-scalar-mult-one*:
assumes $P \in$ *Pring-set R I*
shows *poly-scalar-mult R* $\mathbf{1}$ $P = P$
⟨*proof*⟩

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*⟩

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*) $\bigotimes_p P$
⟨*proof*⟩

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*
⟨*proof*⟩

Definitions of the operations and constants:

lemma *Pring-mult*:
$a \otimes_{Pring\ R\ I} b = a \bigotimes_p b$
⟨*proof*⟩

lemma *Pring-add*:

$a \oplus_{Pring\ R\ I}\ b = a \bigoplus\ b$
  ⟨*proof*⟩

lemma *Pring-zero*:
$\mathbf{0}_{Pring\ R\ I} = indexed\text{-}const\ \mathbf{0}$
  ⟨*proof*⟩

lemma *Pring-one*:
$\mathbf{1}_{Pring\ R\ I} = indexed\text{-}const\ \mathbf{1}$
  ⟨*proof*⟩

lemma *Pring-smult*:
$(\odot_{Pring\ R\ I}) = (poly\text{-}scalar\text{-}mult\ R)$
  ⟨*proof*⟩

lemma *Pring-carrier-coeff*:
  assumes $a \in carrier\ (Pring\ R\ I)$
  shows *carrier-coeff a*
  ⟨*proof*⟩

lemma *Pring-carrier-coeff'*[*simp*]:
  assumes $a \in carrier\ (Pring\ R\ I)$
  shows $a\ m \in carrier\ R$
  ⟨*proof*⟩

lemma *Pring-add-closed*:
  assumes $a \in carrier\ (Pring\ R\ I)$
  assumes $b \in carrier\ (Pring\ R\ I)$
  shows $a \oplus_{Pring\ R\ I}\ b \in carrier\ (Pring\ R\ I)$
  ⟨*proof*⟩

lemma *Pring-mult-closed*:
  assumes $a \in carrier\ (Pring\ R\ I)$
  assumes $b \in carrier\ (Pring\ R\ I)$
  shows $a \otimes_{Pring\ R\ I}\ b \in carrier\ (Pring\ R\ I)$
  ⟨*proof*⟩

lemma *Pring-one-closed*:
$\mathbf{1}_{Pring\ R\ I} \in carrier\ (Pring\ R\ I)$
⟨*proof*⟩

lemma *Pring-zero-closed*:
$\mathbf{0}_{Pring\ R\ I} \in carrier\ (Pring\ R\ I)$
⟨*proof*⟩

lemma *Pring-var-closed*:
  assumes $i \in I$
  shows $var\text{-}to\text{-}IP\ R\ i \in carrier\ (Pring\ R\ I)$
⟨*proof*⟩

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*⟩

lemma *Pring-add-comm*:
assumes $a \in carrier\ (Pring\ R\ I)$
assumes $b \in carrier\ (Pring\ R\ I)$
shows $a \oplus_{Pring\ R\ I} b = b \oplus_{Pring\ R\ I} a$
⟨*proof*⟩

lemma *Pring-add-zero*:
assumes $a \in carrier\ (Pring\ R\ I)$
shows $a \oplus_{Pring\ R\ I} \mathbf{0}_{Pring\ R\ I} = a$
$\mathbf{0}_{Pring\ R\ I} \oplus_{Pring\ R\ I} a = a$
⟨*proof*⟩

Properties of multiplication

lemma *Pring-mult-assoc*:
assumes $a \in carrier\ (Pring\ R\ I)$
assumes $b \in carrier\ (Pring\ R\ I)$
assumes $c \in carrier\ (Pring\ R\ I)$
shows $a \otimes_{Pring\ R\ I} (b \otimes_{Pring\ R\ I} c) = (a \otimes_{Pring\ R\ I} b) \otimes_{Pring\ R\ I} c$
⟨*proof*⟩

lemma *Pring-mult-comm*:
assumes *cring R*
assumes $a \in carrier\ (Pring\ R\ I)$
assumes $b \in carrier\ (Pring\ R\ I)$
shows $a \otimes_{Pring\ R\ I} b = b \otimes_{Pring\ R\ I} a$
⟨*proof*⟩

lemma *Pring-mult-one*:
assumes $a \in carrier\ (Pring\ R\ I)$
shows $a \otimes_{Pring\ R\ I} \mathbf{1}_{Pring\ R\ I} = a$
⟨*proof*⟩

lemma *Pring-mult-one'*:
assumes $a \in carrier\ (Pring\ R\ I)$
shows $\mathbf{1}_{Pring\ R\ I} \otimes_{Pring\ R\ I} a = a$
⟨*proof*⟩

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*⟩

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*⟩

Properties of subtraction:

lemma *Pring-uminus*:
assumes $a \in$ *carrier* (*Pring R I*)
shows *P-ring-uminus R a* $\in$ *carrier* (*Pring R I*)
⟨*proof*⟩

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* $= \mathbf{0}_{Pring\ R\ I}$
⟨*proof*⟩

Pring R I is a ring

lemma *Pring-is-abelian-group*:
shows *abelian-group* (*Pring R I*)
⟨*proof*⟩

lemma *Pring-is-monoid*:
Group.monoid (*Pring R I*)
⟨*proof*⟩

lemma *Pring-is-ring*:
shows *ring* (*Pring R I*)
⟨*proof*⟩

lemma *Pring-is-cring*:
assumes *cring R*
shows *cring* (*Pring R I*)
⟨*proof*⟩

lemma *Pring-a-inv*:
assumes $P \in$ *carrier* (*Pring R I*)
shows $\ominus_{Pring\ R\ I}\ P =$ *P-ring-uminus R P*
⟨*proof*⟩

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 $(a \odot_{Pring\ R\ I} P)\ m = a \otimes (P\ m)$
  ⟨*proof*⟩

lemma *Pring-smult-closed*:
  $\bigwedge a\ x.\ [|\ a \in carrier\ R;\ x \in carrier\ (Pring\ R\ I)\ |] ==> a \odot_{(Pring\ R\ I)} x \in carrier\ (Pring\ R\ I)$
  ⟨*proof*⟩

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*⟩

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*⟩

lemma *Pring-smult-assoc1*:
  $!!a\ b\ x.\ [|\ a \in carrier\ R;\ b \in carrier\ R;\ x \in carrier\ (Pring\ R\ I)\ |] ==>$
  $(a \otimes b) \odot_{Pring\ R\ I} x = a \odot_{Pring\ R\ I} (b \odot_{Pring\ R\ I} x)$
⟨*proof*⟩

lemma *Pring-smult-one*:
  $!!x.\ x \in carrier\ (Pring\ R\ I) ==> (one\ R) \odot_{Pring\ R\ I} x = x$
  ⟨*proof*⟩

lemma *Pring-smult-assoc2*:
  $!!a\ x\ y.\ [|\ a \in carrier\ R;\ x \in carrier\ (Pring\ R\ I);\ y \in carrier\ (Pring\ R\ I)\ |] ==>$
  $(a \odot_{Pring\ R\ I} x) \otimes_{Pring\ R\ I} y = a \odot_{Pring\ R\ I} (x \otimes_{Pring\ R\ I} y)$
  ⟨*proof*⟩

lemma *Pring-algebra*:

algebra R (*Pring R I*)
  ⟨*proof*⟩

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* ($\ominus_R$ *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*)
  ⟨*proof*⟩

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

lemma *remove-monom-closed*:
  assumes *P* ∈ *Pring-set R I*
  shows *remove-monom R m P* ∈ *Pring-set R I*

  ⟨*proof*⟩

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*⟩

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*)
  ⟨*proof*⟩

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*⟩

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*⟩

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*⟩

lemma(**in** *cring*) *Pring-set-restrict*:
  assumes $P \in$ *Pring-set R I*
  assumes ($\bigwedge m.$ $m \in$ *monomials-of R P* $\Longrightarrow$ *set-mset m* $\subseteq J$)
  shows $P \in$ *Pring-set R J*
  ⟨*proof*⟩

lemma(**in** *ring*) *Pring-mult-eq*:
  fixes I:: $'c$ *set*
  fixes J:: $'c$ *set*
  shows $(\otimes_{Pring\ R\ I}) = (\otimes_{Pring\ R\ J})$
  ⟨*proof*⟩

lemma(**in** *ring*) *Pring-add-eq*:
  fixes I:: $'c$ *set*
  fixes J:: $'c$ *set*

shows $(\oplus_{Pring\ R\ I}) = (\oplus_{Pring\ R\ J})$
⟨*proof*⟩

lemma(**in** *ring*) *Pring-one-eq*:
fixes I:: $'c$ *set*
fixes J:: $'c$ *set*
shows $(\mathbf{1}_{Pring\ R\ I}) = (\mathbf{1}_{Pring\ R\ J})$
⟨*proof*⟩

lemma(**in** *ring*) *Pring-zero-eq*:
fixes I:: $'c$ *set*
fixes J:: $'c$ *set*
shows $(\mathbf{0}_{Pring\ R\ I}) = (\mathbf{0}_{Pring\ R\ J})$
⟨*proof*⟩

lemma(**in** *ring*) *index-subset-Pring-subring*:
assumes $J \subseteq I$
shows *subring* (*carrier* (*Pring R J*)) (*Pring R I*)
⟨*proof*⟩

2.11.2 Inclusion Maps

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

lemma(**in** *ring*) *Princ-inc-is-ring-hom*:
assumes $J \subseteq I$
shows *ring-hom-ring* (*Pring R J*) (*Pring R I*) *Pring-inc*
⟨*proof*⟩

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* ($\lambda i.\ i \in S$) *m*

lemma *restrict-to-indicesE*:
assumes i ∈# *restrict-to-indices m S*
shows $i \in S$
⟨*proof*⟩

lemma *restrict-to-indicesI*[*simp*]:
assumes i ∈# *m*
assumes $i \in S$
shows i ∈# *restrict-to-indices m S*
⟨*proof*⟩

lemma *restrict-to-indices-not-in*[*simp*]:
assumes i ∈# *m*
assumes $i \notin S$

shows $i \notin\# $ *restrict-to-indices m S*
⟨*proof*⟩

lemma *restrict-to-indices-submultiset*[*simp*]:
restrict-to-indices m S $\subseteq\#$ *m*
⟨*proof*⟩

lemma *restrict-to-indices-add-element*:
restrict-to-indices (*add-mset x m*) *S* = (*if* $x \in S$ *then* (*add-mset x* (*restrict-to-indices m S*))
else (*restrict-to-indices m S*))
⟨*proof*⟩

lemma *restrict-to-indices-count*[*simp*]:
count (*restrict-to-indices m S*) *i* = (*if* ($i \in S$) *then* (*count m i*) *else 0*)
⟨*proof*⟩

lemma *restrict-to-indices-subset*:
restrict-to-indices m S = *restrict-to-indices m* (*set-mset m* $\cap$ *S*)
⟨*proof*⟩

`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*) $\cap$ *S* = (*set-mset m*) $\cap$ *S′*
shows *restrict-to-indices m S* = *restrict-to-indices m S′*
⟨*proof*⟩

lemma *mset-add-plus*:
assumes $m = n + k$
shows *add-mset x m* = (*add-mset x n*) + *k*
⟨*proof*⟩

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*))
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *remove-indices-indices*[*simp*]:
assumes *set-mset m* $\subseteq$ *I*
shows *set-mset* (*remove-indices m S*) $\subseteq$ *I* − *S*
⟨*proof*⟩

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 $\equiv$ *g* $\in$ *UNIV* $\to$ *carrier R*

definition *monom-eval* :: $('a, 'b)$ *ring-scheme* $\Rightarrow$ $'c$ *monomial* $\Rightarrow$ $('c \Rightarrow 'a) \Rightarrow 'a$
where
monom-eval R (*m*:: $'c$ *monomial*) *g* = *fold-mset* (λ *x* . λ *y*. *if y* $\in$ *carrier R then* (*g x*) $\otimes_R$ *y else* $\mathbf{0}_R$) $\mathbf{1}_R$ *m*

context *cring*
begin

lemma *closed-fun-simp*:
  assumes *closed-fun R g*
  shows *g n* $\in$ *carrier R*
  ⟨*proof*⟩

lemma *closed-funI*:
  assumes $\bigwedge x$. *g x* $\in$ *carrier R*
  shows *closed-fun R g*
  ⟨*proof*⟩

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

lemma *monom-eval-comp-fun*:
  fixes *g*:: $'c \Rightarrow 'a$
  assumes *closed-fun R g*
  shows *comp-fun-commute* (λ *x* . λ*y*. *if y* $\in$ *carrier R then* (*g x*) $\otimes$ *y else* $\mathbf{0}$)
  ⟨*proof*⟩

lemma *monom-eval-car*:
  assumes *closed-fun R g*
  shows *monom-eval R* (*m*:: $'c$ *monomial*) *g* $\in$ *carrier R*
⟨*proof*⟩

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*) $\otimes$ (*monom-eval R M g*)
⟨*proof*⟩

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* $\Rightarrow$ *'c monomial* $\Rightarrow$ $('a,'c)$ *mvar-poly* $\Rightarrow$ *'c set* $\Rightarrow$ *('c monomial) set* **where**
monomials-reducing-to R n P S $= \{m \in$ *monomials-of R P*. *remove-indices m S* $= n\}$

lemma *monomials-reducing-to-subset*[*simp*]:
monomials-reducing-to R n P s $\subseteq$ *monomials-of R P*
⟨*proof*⟩

context *cring*
begin

lemma *monomials-reducing-to-finite*:
assumes $P \in$ *Pring-set R I*
shows *finite* (*monomials-reducing-to R n P s*)
⟨*proof*⟩

lemma *monomials-reducing-to-disjoint*:
assumes $n1 \neq n2$
shows *monomials-reducing-to R n1 P S* $\cap$ *monomials-reducing-to R n2 P S* $= \{\}$
⟨*proof*⟩

lemma *monomials-reducing-to-submset*:
assumes $n \subset\# m$
shows $n \notin$ *monomials-reducing-to R m P S*
⟨*proof*⟩

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* $\Rightarrow$ *'c set* $\Rightarrow$ $('c \Rightarrow 'a)$ $\Rightarrow$ $('a, 'c)$ *mvar-poly* $\Rightarrow$ $('a, 'c)$ *mvar-poly***where**
poly-eval R S g P m = (*finsum R* (λn. *monom-eval R* (*restrict-to-indices n S*) *g* $\otimes_R$ (*P n*)) (*monomials-reducing-to R m P S*))

context *cring*
begin

lemma *finsum-singleton*:
assumes $S = \{s\}$

assumes $f\ s \in carrier\ R$
shows $finsum\ R\ f\ S = f\ s$
$\langle proof \rangle$

lemma *poly-eval-constant*:
assumes $k \in carrier\ R$
shows $poly\text{-}eval\ R\ S\ g\ (indexed\text{-}const\ k) = (indexed\text{-}const\ k)$
$\langle proof \rangle$

lemma *finsum-partition*:
assumes *finite S*
assumes $f \in S \rightarrow carrier\ R$
assumes $T \subseteq S$
shows $finsum\ \ R\ f\ S\ = finsum\ R\ f\ T\ \oplus finsum\ \ R\ f\ (S\ -\ T)$
$\langle proof \rangle$

lemma *finsum-eq-parition*:
assumes *finite S*
assumes $f \in S \rightarrow carrier\ R$
assumes $T \subseteq S$
assumes $\bigwedge x.\ x \in S\ -\ T \Longrightarrow f\ x = \mathbf{0}$
shows $finsum\ \ R\ f\ S\ = finsum\ R\ f\ T$
$\langle proof \rangle$

lemma *poly-eval-scalar-mult*:
assumes $k \in carrier\ R$
assumes *closed-fun R g*
assumes $P \in Pring\text{-}set\ R\ I$
shows $poly\text{-}eval\ R\ \ S\ g\ (poly\text{-}scalar\text{-}mult\ R\ k\ P)=$
$(poly\text{-}scalar\text{-}mult\ R\ k\ (poly\text{-}eval\ R\ S\ g\ P))$
$\langle proof \rangle$

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

lemma(in *cring*) *poly-eval-monomial-closed*:
assumes *closed-fun R g*
assumes $\mathbf{1} \neq \mathbf{0}$
assumes $set\text{-}mset\ m \subseteq I$
shows $poly\text{-}eval\ R\ S\ g\ (mset\text{-}to\text{-}IP\ R\ m) \in Pring\text{-}set\ R\ (I\ -\ S)$
$\langle proof \rangle$

lemma *poly-scalar-mult-iter*:

assumes $\mathbf{1} \neq \mathbf{0}$
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*
⟨*proof*⟩

lemma *poly-scalar-mult-comm*:
assumes $\mathbf{1} \neq \mathbf{0}$
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*)
⟨*proof*⟩

lemma *poly-eval-monomial-term*:
assumes *closed-fun R g*
assumes $\mathbf{1} \neq \mathbf{0}$
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*⟩

lemma *poly-eval-monomial-term-closed*:
assumes *closed-fun R g*
assumes $\mathbf{1} \neq \mathbf{0}$
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*)) $\in$ *Pring-set R* ($I - S$)
⟨*proof*⟩

lemma *finsum-split*:
assumes *finite S*
assumes $f \in S \rightarrow$ *carrier R*
assumes $g \in S \rightarrow$ *carrier R*
assumes $k \in$ *carrier R*
assumes $c \in S$
assumes $\bigwedge s.\ s \in S \wedge s \neq c \Longrightarrow f\ s = g\ s$
assumes $g\ c = f\ c \oplus k$
shows *finsum R g S* = $k \oplus$ *finsum R f S*
⟨*proof*⟩

lemma *poly-monom-induction*:
assumes *P* (*indexed-const* $\mathbf{0}$)

assumes $\bigwedge m\ k.\ \textit{set-mset}\ m \subseteq I \wedge k \in \textit{carrier}\ R \Longrightarrow P\ (\textit{poly-scalar-mult}\ R\ k\ (\textit{mset-to-IP}\ R\ m))$
assumes $\bigwedge Q\ m\ k.\ Q \in \textit{Pring-set}\ R\ I \wedge (P\ Q) \wedge \textit{set-mset}\ m \subseteq I \wedge k \in \textit{carrier}\ R \Longrightarrow P\ (Q \bigoplus (\textit{poly-scalar-mult}\ R\ k\ (\textit{mset-to-IP}\ R\ m)))$
shows $\bigwedge Q.\ Q \in \textit{Pring-set}\ R\ I \Longrightarrow P\ Q$
⟨*proof*⟩

lemma *Pring-car-induct*:
assumes $q \in \textit{carrier}\ (\textit{Pring}\ R\ I)$
assumes $P\ \mathbf{0}_{\textit{Pring}\ R\ I}$
assumes $\bigwedge m\ k.\ \textit{set-mset}\ m \subseteq I \wedge k \in \textit{carrier}\ R \Longrightarrow P\ (k \odot_{\textit{Pring}\ R\ I}(\textit{mset-to-IP}\ R\ m))$
assumes $\bigwedge Q\ m\ k.\ Q \in \textit{carrier}\ (\textit{Pring}\ R\ I) \wedge (P\ Q) \wedge \textit{set-mset}\ m \subseteq I \wedge k \in \textit{carrier}\ R \Longrightarrow$
$P\ (Q \bigoplus (k \odot_{\textit{Pring}\ R\ I}\ (\textit{mset-to-IP}\ R\ m)))$
shows $P\ q$
⟨*proof*⟩

lemma *poly-monom-induction2*:
assumes $P\ (\textit{indexed-const}\ \mathbf{0})$
assumes $\bigwedge m\ k.\ \textit{set-mset}\ m \subseteq I \wedge k \in \textit{carrier}\ R \Longrightarrow P\ (\textit{poly-scalar-mult}\ R\ k\ (\textit{mset-to-IP}\ R\ m))$
assumes $\bigwedge Q\ m\ k.\ Q \in \textit{Pring-set}\ R\ I \wedge (P\ Q) \wedge \textit{set-mset}\ m \subseteq I \wedge k \in \textit{carrier}\ R \Longrightarrow P\ (Q \bigoplus (\textit{poly-scalar-mult}\ R\ k\ (\textit{mset-to-IP}\ R\ m)))$
assumes $Q \in \textit{Pring-set}\ R\ I$
shows $P\ Q$
⟨*proof*⟩

lemma *poly-monom-induction3*:
assumes $Q \in \textit{Pring-set}\ R\ I$
assumes $P\ (\textit{indexed-const}\ \mathbf{0})$
assumes $\bigwedge m\ k.\ \textit{set-mset}\ m \subseteq I \wedge k \in \textit{carrier}\ R \Longrightarrow P\ (\textit{poly-scalar-mult}\ R\ k\ (\textit{mset-to-IP}\ R\ m))$
assumes $\bigwedge p\ q.\ p \in \textit{Pring-set}\ R\ I \Longrightarrow (P\ p) \Longrightarrow q \in \textit{Pring-set}\ R\ I \Longrightarrow (P\ q) \Longrightarrow P\ (p \bigoplus q)$
shows $P\ Q$
⟨*proof*⟩

lemma *Pring-car-induct'*:
assumes $Q \in \textit{carrier}\ (\textit{Pring}\ R\ I)$
assumes $P\ \mathbf{0}_{\textit{Pring}\ R\ I}$
assumes $\bigwedge m\ k.\ \textit{set-mset}\ m \subseteq I \wedge k \in \textit{carrier}\ R \Longrightarrow P\ (k \odot_{\textit{Pring}\ R\ I}\ \textit{mset-to-IP}\ R\ m)$
assumes $\bigwedge p\ q.\ p \in \textit{carrier}\ (\textit{Pring}\ R\ I) \Longrightarrow (P\ p) \Longrightarrow q \in \textit{carrier}\ (\textit{Pring}\ R\ I) \Longrightarrow (P\ q) \Longrightarrow P\ (p \oplus_{\textit{Pring}\ R\ I}\ q)$
shows $P\ Q$
⟨*proof*⟩

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*⟩

lemma *finsum-group*:
assumes $\bigwedge n.\ f\ n \in$ *carrier* R
assumes $\bigwedge n.\ g\ n \in$ *carrier* R
shows *finite* $S \Longrightarrow$ *finsum* R f $S \oplus$ *finsum* R g $S =$ *finsum* R $(\lambda n.\ f\ n \oplus g\ n)$ S
⟨*proof*⟩

lemma *poly-eval-add*:
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 $(P \bigoplus Q)$ $=$ *poly-eval* R S g P $\bigoplus$ *poly-eval* R S g Q
⟨*proof*⟩

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
⟨*proof*⟩

Closure of partial evaluation maps:

lemma(**in** *cring*) *poly-eval-closed*:
assumes *closed-fun* R g
assumes $P \in$ *Pring-set* R I
shows *poly-eval* R S g $P \in$ *Pring-set* R $(I - S)$
⟨*proof*⟩

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*⟩

lemma *remove-indices-add-mset*:
assumes $i \notin S$
shows *remove-indices* (*add-mset* i m) $S =$ *add-mset* i (*remove-indices* m S)
⟨*proof*⟩

lemma *poly-eval-monom-insert*:

assumes *closed-fun R g*
assumes $\mathbf{1} \neq \mathbf{0}$
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*⟩

lemma *poly-eval-monom-insert′*:
assumes *closed-fun R g*
assumes $\mathbf{1} \neq \mathbf{0}$
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*⟩

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*) $\bigotimes$ *i*) =
poly-scalar-mult R (*g i*) (*poly-eval R S g* (*poly-scalar-mult R k* (*mset-to-IP R m*)))
⟨*proof*⟩

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*⟩

lemma *indexed-pmult-add*:
assumes $p \in$ *Pring-set R I*
assumes $q \in$ *Pring-set R I*
shows $p \bigoplus q \bigotimes i = (p \bigotimes i) \bigoplus (q \bigotimes i)$
⟨*proof*⟩

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*⟩

lemma *poly-eval-index*:

assumes $\mathbf{1} \neq \mathbf{0}$
assumes *closed-fun R g*
shows *poly-eval R S g* (*mset-to-IP R* {#*i*#})= (*if* $i \in S$ *then* (*indexed-const* (*g i*)) *else mset-to-IP R* {#*i*#})
⟨*proof*⟩

lemma *poly-eval-indexed-pmult'*:
assumes $P \in$ *Pring-set R I*
assumes *closed-fun R g*
assumes $i \in I$
shows *poly-eval R S g* ($P \bigotimes_p$ (*mset-to-IP R* {#*i*#})) = *poly-eval R S g P* $\bigotimes_p$ *poly-eval R S g* (*mset-to-IP R* {#*i*#})
⟨*proof*⟩

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

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 $P \in$ *Pring-set R I*
assumes *closed-fun R g*
assumes $k \in$ *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*⟩

lemma *poly-eval-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* ($P \bigotimes_p Q$) = *poly-eval R S g P* $\bigotimes_p$ *poly-eval R S g Q*
⟨*proof*⟩

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*
⟨*proof*⟩

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
⟨*proof*⟩

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)
⟨*proof*⟩

`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 $P = P$
⟨*proof*⟩

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
⟨*proof*⟩

lemma *poly-eval-eval-set-eq*:
assumes *closed-fun* R g
assumes $S \cap I = S' \cap I$
assumes $P \in$ *Pring-set* R I
assumes $\mathbf{1} \neq \mathbf{0}$
shows *poly-eval* R S g $P =$ *poly-eval* R S' g P
⟨*proof*⟩

lemma *poly-eval-trivial*:
assumes *closed-fun* R g
assumes $P \in$ *Pring-set* R $(I - S)$
shows *poly-eval* R S g $P = P$
⟨*proof*⟩

2.11.7 Total Evaluation of a Polynomial

lemma *zero-fun-closed*:
closed-fun R $(\lambda n.\ \mathbf{0})$
⟨*proof*⟩

lemma *deg-zero-cf-eval*:
  shows $P \in \textit{Pring-set } R\ I \Longrightarrow \textit{poly-eval } R\ I\ (\lambda n.\ \mathbf{0})\ P = \textit{indexed-const}\ (P\ \{\#\})$
  ⟨*proof*⟩

lemma *deg-zero-cf-mult*:
  assumes $P \in \textit{Pring-set } R\ I$
  assumes $Q \in \textit{Pring-set } R\ I$
  shows $(P \bigotimes_p Q)\ \{\#\} = P\ \{\#\} \otimes Q\ \{\#\}$
⟨*proof*⟩

definition *deg-zero-cf* :: $('a, 'c)\ \textit{mvar-poly} \Rightarrow 'a$ **where**
$\textit{deg-zero-cf}\ P = P\ \{\#\}$

lemma *deg-zero-cf-ring-hom*:
  shows $\textit{ring-hom-ring}\ (\textit{Pring } R\ I)\ R\ (\textit{deg-zero-cf})$
  ⟨*proof*⟩

end

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

definition *total-eval* ::
$('a,'b)\ \textit{ring-scheme} \Rightarrow ('c \Rightarrow 'a) \Rightarrow ('a, 'c)\ \textit{mvar-poly} \Rightarrow 'a$ **where**
$\textit{total-eval}\ R\ g\ P = \textit{eval-in-ring}\ R\ \textit{UNIV}\ g\ P$

context *cring*
begin

lemma *eval-in-ring-ring-hom*:
  assumes $\textit{closed-fun}\ R\ g$
  shows $\textit{ring-hom-ring}\ (\textit{Pring } R\ I)\ R\ (\textit{eval-in-ring}\ R\ S\ g)$
  ⟨*proof*⟩

lemma *eval-in-ring-smult*:
  assumes $P \in \textit{carrier}\ (\textit{Pring } R\ I)$
  assumes $a \in \textit{carrier}\ R$
  assumes $\textit{closed-fun}\ R\ g$
  shows $\textit{eval-in-ring}\ R\ S\ g\ (a \odot_{\textit{Pring } R\ I}\ P) = a \otimes \textit{eval-in-ring}\ R\ S\ g\ P$
  ⟨*proof*⟩

lemma *total-eval-ring-hom*:
  assumes $\textit{closed-fun}\ R\ g$
  shows $\textit{ring-hom-ring}\ (\textit{Pring } R\ I)\ R\ (\textit{total-eval}\ R\ g)$
  ⟨*proof*⟩

lemma *total-eval-smult*:
assumes $P \in$ *carrier* (*Pring R I*)
assumes $a \in$ *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*
⟨*proof*⟩

lemma *total-eval-const*:
assumes $k \in$ *carrier R*
shows *total-eval R g* (*indexed-const k*) $= k$
⟨*proof*⟩

lemma *total-eval-var*:
assumes *closed-fun R g*
shows (*total-eval R g* (*mset-to-IP R* $\{\#i\#\}$)) $= g\ i$
⟨*proof*⟩

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*⟩

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)
⟨*proof*⟩

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)
⟨*proof*⟩

lemma *total-eval-one*:
assumes *closed-fun R g*
shows *total-eval R g* $\mathbf{1}_{Pring\ R\ I} = \mathbf{1}$
⟨*proof*⟩

lemma *total-eval-zero*:
assumes *closed-fun R g*
shows *total-eval R g* $\mathbf{0}_{Pring\ R\ I} = \mathbf{0}$
⟨*proof*⟩

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*
  ⟨*proof*⟩

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*)
  ⟨*proof*⟩

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

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* $\Rightarrow$ $('a, 'c)$ *mvar-poly* $\Rightarrow$ $'c$ *monomial* $\Rightarrow$ $'e$ **where**
ring-hom-to-IP-ring-hom φ P m $= \varphi$ $(P\ m)$

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 $\mathbf{1}_{Pring\ R\ I} = \mathbf{1}_{Pring\ S\ I}$
  ⟨*proof*⟩

lemma *ring-hom-to-IP-ring-hom-constant*:
  assumes *cring S*
  assumes *ring-hom-ring R S* φ
  assumes $a \in$ *carrier R*
  shows *ring-hom-to-IP-ring-hom* φ ((*indexed-const* a):: $'c$ *monomial* $\Rightarrow$ $'a$) $=$
ring.indexed-const S (φ a)
  ⟨*proof*⟩

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)
⟨*proof*⟩

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*)
⟨*proof*⟩

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*⟩

lemma *Pring-morphism*:
assumes *cring S*
assumes $\varphi \in$ (*carrier* (*Pring R I*)) $\to$ (*carrier S*)
assumes $\varphi\ \mathbf{1}_{Pring\ R\ I} = \mathbf{1}_S$
assumes $\varphi\ \mathbf{0}_{Pring\ R\ I} = \mathbf{0}_S$
assumes $\bigwedge P\ Q.\ P \in$ *carrier* (*Pring R I*) $\Longrightarrow Q \in$ *carrier* (*Pring R I*) $\Longrightarrow$
$\varphi\ (P \oplus_{Pring\ R\ I} Q) = (\varphi\ P) \oplus_S (\varphi\ Q)$
assumes $\bigwedge\ i\ .\ \bigwedge\ P.\ i \in I \Longrightarrow P \in$ *carrier* (*Pring R I*) $\Longrightarrow\ \varphi\ (P \bigotimes\ i) =\ (\varphi\ P) \otimes_S (\varphi$ (*mset-to-IP R* {#*i*#}))
assumes $\bigwedge k\ Q.\ k \in$ *carrier R* $\Longrightarrow Q \in$ *carrier* (*Pring R I*) $\Longrightarrow\ \varphi$ (*poly-scalar-mult R k Q*) =
(φ (*indexed-const k*)) $\otimes_S$ (φ Q)
shows *ring-hom-ring* (*Pring R I*) *S* φ
⟨*proof*⟩

lemma(**in** *cring*) *indexed-const-Pring-mult*:
assumes $k \in$ *carrier R*
assumes $P \in$ *carrier* (*Pring R I*)
shows (*indexed-const k* $\otimes_{Pring\ R\ I}$ P) $m = k \otimes_R (P\ m)$
($P \otimes_{Pring\ R\ I}$ *indexed-const k*) $m = k \otimes_R (P\ m)$
⟨*proof*⟩

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*⟩

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

assumes $a \in carrier\ R$
shows $ring\text{-}hom\text{-}to\text{-}IP\text{-}ring\text{-}hom\ \varphi\ (a \odot_{Pring\ R\ I} P) =$
$\varphi\ a \odot_{Pring\ S\ I}\ (ring\text{-}hom\text{-}to\text{-}IP\text{-}ring\text{-}hom\ \varphi\ P)$
$\langle proof \rangle$

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 $\psi = (total\text{-}eval\ S\ g) \circ (ring\text{-}hom\text{-}to\text{-}IP\text{-}ring\text{-}hom\ \varphi)$
shows $(ring\text{-}hom\text{-}ring\ (Pring\ R\ I)\ S\ \psi)$
$(\forall i \in I.\ \psi\ (mset\text{-}to\text{-}IP\ R\ \{\#i\#\}) = g\ i)$
$(\forall a \in carrier\ R.\ \psi\ (indexed\text{-}const\ a) = \varphi\ a)$
$\forall\ \varrho.\ (ring\text{-}hom\text{-}ring\ (Pring\ R\ I)\ S\ \varrho) \wedge$
$(\forall i \in I.\ \varrho\ (mset\text{-}to\text{-}IP\ R\ \{\#i\#\}) = g\ i) \wedge$
$(\forall a \in carrier\ R.\ \varrho\ (indexed\text{-}const\ a) = \varphi\ a) \longrightarrow$
$(\forall x \in carrier\ (Pring\ R\ I).\ \varrho\ x = \psi\ x)$
$\langle proof \rangle$

end

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

context *cring*
begin

lemma *close-funE*:
assumes *cring S*
assumes $g \in I \to carrier\ S$
shows *closed-fun S* $(close\text{-}fun\ I\ S\ g)$
$\langle proof \rangle$

end

definition *indexed-poly-induced-morphism* ::
$'c\ set \Rightarrow ('e,\ 'f)\ ring\text{-}scheme \Rightarrow ('a,\ 'e)\ ring\text{-}hom \Rightarrow ('c \Rightarrow 'e) \Rightarrow (('a,'c)\ mvar\text{-}poly,$
$'e)\ ring\text{-}hom$ **where**
$indexed\text{-}poly\text{-}induced\text{-}morphism\ I\ S\ \varphi\ g = (total\text{-}eval\ S\ (close\text{-}fun\ I\ S\ g)) \circ (ring\text{-}hom\text{-}to\text{-}IP\text{-}ring\text{-}hom$
$\varphi)$

context *cring*
begin

lemma *Pring-universal-prop*:
assumes *a-cring*: *cring S*

assumes *index-map*: $g \in I \rightarrow$ *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)
(∀ $i \in I$. ψ (*mset-to-IP* R {#i#}) = g i)
(∀ $a \in$ *carrier* R. ψ (*indexed-const* a) = φ a)
∀ ϱ. (*ring-hom-ring* (*Pring* R I) S ϱ) ∧
(∀ $i \in I$. ϱ (*mset-to-IP* R {#i#}) = g i) ∧
(∀ $a \in$ *carrier* R. ϱ (*indexed-const* a) = φ a) ⟶
(∀ $x \in$ *carrier* (*Pring* R I). ϱ x = ψ x)
⟨*proof*⟩

2.13 Mapping Mulitvariate Polynomials over a Single Variable to Univariate Polynomials

Constructor for multisets which have one distinct element

definition *nat-to-mset* :: *'c* ⇒ *nat* ⇒ *'c monomial* **where**
nat-to-mset i n = *Abs-multiset* (λj. *if* ($j = i$) *then* n *else* *0*)

lemma *nat-to-msetE*: *count* (*nat-to-mset* i n) $i = n$
⟨*proof*⟩

lemma *nat-to-msetE'*:
assumes $j \neq i$
shows *count* (*nat-to-mset* i n) j = *0*
⟨*proof*⟩

lemma *nat-to-mset-add*: *nat-to-mset* i ($n + m$) = (*nat-to-mset* i n) + (*nat-to-mset* i m)
⟨*proof*⟩

lemma *nat-to-mset-inj*:
assumes $n \neq m$
shows (*nat-to-mset* i n) ≠ (*nat-to-mset* i m)
⟨*proof*⟩

lemma *nat-to-mset-zero*: *nat-to-mset* i *0* = {#}
⟨*proof*⟩

lemma *nat-to-mset-Suc*: *nat-to-mset* i (*Suc* n) = *add-mset* i (*nat-to-mset* i n)
⟨*proof*⟩

lemma *nat-to-mset-Pring-singleton*:
assumes *cring* R
assumes $P \in$ *carrier* (*Pring* R {i})
assumes $m \in$ *monomials-of* R P
shows m = *nat-to-mset* i (*count* m i)
⟨*proof*⟩

definition *IP-to-UP* :: $'d \Rightarrow ('e, 'd)$ *mvar-poly* $\Rightarrow$ *'e u-poly* **where**
IP-to-UP i P $= (\lambda\ (n{::}nat).\ P\ (nat\text{-}to\text{-}mset\ i\ n))$

lemma *IP-to-UP-closed*:
  assumes *cring R*
  assumes $P \in carrier\ (Pring\ R\ \{i{::}'c\})$
  shows *IP-to-UP i P* $\in carrier\ (UP\ R)$
⟨*proof*⟩

lemma *IP-to-UP-var*:
  shows *IP-to-UP i* $(mset\text{-}to\text{-}IP\ R\ \{\#i\#\}) = X\text{-}poly\ R$
⟨*proof*⟩

end

context *UP-cring*
begin

lemma *IP-to-UP-monom*:
  shows *IP-to-UP i* $(mset\text{-}to\text{-}IP\ R\ (nat\text{-}to\text{-}mset\ i\ n)) = ((X\text{-}poly\ R)[\hat{\ }]_{UP\ R} n)$
⟨*proof*⟩

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

lemma *IP-to-UP-zero*:
 IP-to-UP i $\mathbf{0}_{Pring\ R\ \{i\}} = \mathbf{0}_{UP\ R}$
⟨*proof*⟩

lemma *IP-to-UP-add*:
  assumes $x \in carrier\ (Pring\ R\ \{i\})$
  assumes $y \in carrier\ (Pring\ R\ \{i\})$
  shows *IP-to-UP i* $(x \oplus_{Pring\ R\ \{i\}} y) =$
      IP-to-UP i x $\oplus_{UP\ R}$ *IP-to-UP i y*
⟨*proof*⟩

lemma *IP-to-UP-indexed-const*:
  assumes $k \in carrier\ R$
  shows *IP-to-UP i* $(ring.indexed\text{-}const\ R\ k) = to\text{-}polynomial\ R\ k$
⟨*proof*⟩

lemma *IP-to-UP-indexed-pmult*:
  assumes $p \in carrier\ (Pring\ R\ \{i\})$
  shows *IP-to-UP i* $(ring.indexed\text{-}pmult\ R\ p\ i) = (IP\text{-}to\text{-}UP\ i\ p) \otimes_{UP\ R} (X\text{-}poly$
$R)$
⟨*proof*⟩

lemma *IP-to-UP-ring-hom*:

shows *ring-hom-ring* (*Pring R* {*i*}) (*UP R*) (*IP-to-UP i*)
⟨*proof*⟩

lemma *IP-to-UP-ring-hom-inj*:
shows *inj-on* (*IP-to-UP i*) (*carrier* (*Pring R* {*i*}))
⟨*proof*⟩

lemma *IP-to-UP-scalar-mult*:
assumes $a \in$ *carrier R*
assumes $p \in$ *carrier* (*Pring R* {*i*})
shows (*IP-to-UP i* ($a \odot_{Pring\ R\ \{i\}}$ *p*)) = $a \odot_{UP\ R}$ (*IP-to-UP i p*)
⟨*proof*⟩

end

Evaluation of indexed polynomials commutes with evaluation of univariate polynomials:

lemma *pvar-closed*:
assumes *cring R*
assumes $i \in I$
shows (*pvar R i*) $\in$ *carrier* (*Pring R I*)
⟨*proof*⟩

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-to-IP R* {#*i*, *j*#}
⟨*proof*⟩

lemma *pvar-pow*:

assumes $i \in I$
shows (*pvar R i*)$[\hat{}]_{Pring\ R\ I}$(*n*::*nat*) = *mset-to-IP R* (*nat-to-mset i n*)
⟨*proof*⟩

lemma *IP-to-UP-poly-eval*:
assumes $p \in$ *Pring-set R* {*i*}
assumes *closed-fun R g*
shows *total-eval R g p* = *to-function R* (*IP-to-UP i p*) (*g i*)
⟨*proof*⟩
end

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

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

context *UP-cring*
begin

lemma *UP-to-IP-inv*:
  assumes $p \in$ *Pring-set R* $\{i\}$
  shows *UP-to-IP R i* (*IP-to-UP i p*) = *p*
⟨*proof*⟩

lemma *UP-to-IP-const*:
  assumes $a \in$ *carrier R*
  shows *UP-to-IP R i* (*to-polynomial R a*) = *ring.indexed-const R a*
⟨*proof*⟩

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*⟩

lemma *UP-to-IP-var*:
  shows *UP-to-IP R i* (*X-poly R*) = *pvar R i*
⟨*proof*⟩

lemma *UP-to-IP-var-pow*:
  shows *UP-to-IP R i* ((*X-poly R*)$[\urcorner]_{UP\ R}$ (*n::nat*)) = (*pvar R i*)$[\urcorner]_{Pring\ R\ \{i\}}\ n$
⟨*proof*⟩

lemma *one-var-indexed-poly-monom-simp*:
  assumes $a \in$ *carrier R*
  shows ($a \odot_{Pring\ R\ \{i\}}$ ((*pvar R i*) $[\urcorner]_{Pring\ R\ \{i\}}$ *n*)) *x* = (*if x* = (*nat-to-mset i n*) *then a else* $\mathbf{0}$)
⟨*proof*⟩

lemma *UP-to-IP-monom*:
  assumes $a \in$ *carrier R*
  shows *UP-to-IP R i* (*up-ring.monom* (*UP R*) *a n*) = $a \odot_{Pring\ R\ \{i\}}$ ((*pvar R i*)$[\urcorner]_{Pring\ R\ \{i\}}\ n$)
⟨*proof*⟩

lemma *UP-to-IP-monom'*:

assumes $a \in carrier\ R$
shows $UP\text{-}to\text{-}IP\ R\ i\ (up\text{-}ring.monom\ (UP\ R)\ a\ n) = a \odot_{Pring\ R\ \{i\}} ((pvar\ R\ i)[\widehat{\ }]_{Pring\ R\ \{i\}}\, n)$
$\langle proof \rangle$

lemma $UP\text{-}to\text{-}IP\text{-}closed$:
assumes $p \in carrier\ P$
shows $(UP\text{-}to\text{-}IP\ R\ i\ p) \in carrier\ (Pring\ R\ \{i\})$
$\langle proof \rangle$

lemma $IP\text{-}to\text{-}UP\text{-}inv$:
assumes $p \in carrier\ P$
shows $IP\text{-}to\text{-}UP\ i\ (UP\text{-}to\text{-}IP\ R\ i\ p) = p$
$\langle proof \rangle$

lemma $UP\text{-}to\text{-}IP\text{-}mult$:
assumes $p \in carrier\ (UP\ R)$
assumes $Q \in carrier\ (UP\ R)$
shows $UP\text{-}to\text{-}IP\ R\ i\ (p \otimes_{UP\ R} Q) =$
$UP\text{-}to\text{-}IP\ R\ i\ p \otimes_{Pring\ R\ \{i\}} UP\text{-}to\text{-}IP\ R\ i\ Q$
$\langle proof \rangle$

lemma $UP\text{-}to\text{-}IP\text{-}ring\text{-}hom$:
shows $ring\text{-}hom\text{-}ring\ (UP\ R)\ (Pring\ R\ \{i\})\ (UP\text{-}to\text{-}IP\ R\ i)$
$\langle proof \rangle$

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 property 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*
$::\ 'd\ set \Rightarrow\ 'd\ set \Rightarrow$
$('a, (('a, 'd)\ mvar\text{-}poly, 'd)\ mvar\text{-}poly)\ ring\text{-}hom$ **where**
$dist\text{-}varset\text{-}morphism\ (I{::}\ 'd\ set)\ (J{::}\ 'd\ set) =$
$(ring.indexed\text{-}const\ (Pring\ R\ J) :: ('d\ multiset \Rightarrow 'a) \Rightarrow 'd\ multiset \Rightarrow ('d\ multiset \Rightarrow 'a)) \circ (ring.indexed\text{-}const\ R :: 'a \Rightarrow 'd\ multiset \Rightarrow 'a)$

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*) = (λi. *if* $i \in J$ *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*⟩

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*)
⟨*proof*⟩

lemma *var-factor-morphism*:
assumes (*I*:: $'d$ *set*) ⊆ *J0* ∪ *J1*
assumes *J1* ⊆ *I*
assumes *J1* ∩ *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* ∩ *I* ⟹ ψ (*pvar R i*) = *ring.indexed-const* (*Pring R J0*) (*pvar R i*)
$\bigwedge i$. $i \in$ *J1* ⟹ ψ (*pvar R i*) = *pvar* (*Pring R J0*) *i*
$\bigwedge a$. $a \in$ *carrier* (*Pring R* (*J0* ∩ *I*)) ⟹ ψ *a* = *ring.indexed-const* (*Pring R J0*) *a*
⟨*proof*⟩

lemma *var-factor-morphism*$'$:
assumes *I* = *J0* ∪ *J1*
assumes *J1* ⊆ *I*
assumes *J1* ∩ *J0* = {}

assumes ψ = (*var-factor I J0 J1*)
shows *ring-hom-ring* (*Pring R I*) (*Pring* (*Pring R J0*) *J1*) ψ
$\bigwedge i.\ i \in J1 \Longrightarrow \psi\ (pvar\ R\ i) = pvar\ (Pring\ R\ J0)\ i$
$\bigwedge a.\ a \in carrier\ (Pring\ R\ (J0 \cap I)) \Longrightarrow \psi\ a = ring.indexed\text{-}const\ (Pring\ R\ J0)\ a$
⟨*proof*⟩

Constructing the inverse morphism for `var_factor_morphism`

lemma *pvar-ass-closed*:
assumes $J1 \subseteq I$
shows $pvar\ R \in J1 \rightarrow carrier\ (Pring\ R\ I)$
⟨*proof*⟩

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

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 ψ = (*var-factor-inv I J0 J1*)
shows *ring-hom-ring* (*Pring* (*Pring R J0*) *J1*) (*Pring R I*) ψ
$\bigwedge i.\ i \in J1 \Longrightarrow \psi\ (pvar\ (Pring\ R\ J0)\ i) = pvar\ R\ i$
$\bigwedge a.\ a \in carrier\ (Pring\ R\ J0) \Longrightarrow \psi\ (ring.indexed\text{-}const\ (Pring\ R\ J0)\ a) = a$
⟨*proof*⟩

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 $\psi 1\ (\psi 0\ P) = P$
⟨*proof*⟩

lemma *var-factor-total-eval*:
assumes $I = J0 \cup J1$
assumes $J1 \subseteq I$
assumes $J1 \cap J0 = \{\}$
assumes ψ = (*var-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* ∘ *g*) (ψ *P*))
⟨*proof*⟩

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* ∈ *carrier* (*Pring* (*Pring R J0*) *J1*)
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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* ⇒ ′*c* ⇒ (′*a*,′*c*) *mvar-poly* ⇒
((′*a*,′*c*) *mvar-poly*) *u-poly* **where**
multivar-poly-to-univ-poly I i P = ((*IP-to-UP i*) ∘ (*var-factor I* ($I - \{i\}$) $\{i\}$)) *P*

definition *univ-poly-to-multivar-poly* ::
′*c set* ⇒ ′*c* ⇒ ((′*a*,′*c*) *mvar-poly*) *u-poly* ⇒
(′*a*,′*c*) *mvar-poly* **where**
univ-poly-to-multivar-poly I i P =((*var-factor-inv I* ($I - \{i\}$) $\{i\}$) ∘ (*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* $-$ $\{i\}$)))
⟨*proof*⟩

lemma *multivar-poly-to-univ-poly-inverse*:
assumes $i \in I$
assumes $\psi 0$ = *multivar-poly-to-univ-poly I i*
assumes $\psi 1$ = *univ-poly-to-multivar-poly I i*
assumes $P \in$ *carrier* (*Pring R I*)
shows $\psi 1$ ($\psi 0$ P) = P
⟨*proof*⟩

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*⟩

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*
⟨*proof*⟩

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 $a \in$ *carrier R*
assumes $b \in$ *carrier R*
shows *indexed-const a* $\otimes_{Pring\ R\ I}$ *indexed-const b* = *indexed-const* ($a \otimes\ b$)
⟨*proof*⟩

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

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*⟩

lemma(**in** *domain*) *Pring-is-domain*:
  domain (*Pring R I*)
⟨*proof*⟩

2.14.4 Relabelling of Variables for Indexed Polynomial Rings

definition *relabel-vars* :: $'d\ set \Rightarrow\ 'e\ set \Rightarrow ('d \Rightarrow\ 'e) \Rightarrow$
  $('a,\ 'd)\ mvar\text{-}poly \Rightarrow ('a,\ 'e)\ mvar\text{-}poly$ **where**
relabel-vars I J g = indexed-poly-induced-morphism I (Pring R J) indexed-const ($\lambda i.$ *pvar R* (*g i*))

lemma *relabel-vars-is-morphism*:
  assumes $g \in I \rightarrow J$
  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*
  ⟨*proof*⟩

lemma *relabel-vars-add*:
  assumes $g \in I \rightarrow J$
  assumes $P \in carrier\ (Pring\ R\ I)$
  assumes $Q \in carrier\ (Pring\ R\ I)$
  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*
  ⟨*proof*⟩

lemma *relabel-vars-mult*:
  assumes $g \in I \rightarrow J$
  assumes $P \in carrier\ (Pring\ R\ I)$
  assumes $Q \in carrier\ (Pring\ R\ I)$
  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*
  ⟨*proof*⟩

lemma *relabel-vars-closed*:
  assumes $g \in I \rightarrow J$
  assumes $P \in carrier\ (Pring\ R\ I)$
  shows *relabel-vars I J g P* $\in$ *carrier* (*Pring R J*)

⟨*proof*⟩

lemma *relabel-vars-smult*:
assumes $g \in I \rightarrow J$
assumes $P \in$ *carrier* (*Pring R I*)
assumes $a \in$ *carrier R*
shows *relabel-vars I J g* $(a \odot_{Pring\ R\ I} P) = a \odot_{Pring\ R\ J}$ *relabel-vars I J g P*
⟨*proof*⟩

lemma *relabel-vars-inverse*:
assumes $g \in I \rightarrow J$
assumes $h \in J \rightarrow I$
assumes $\bigwedge i.\ i \in I \Longrightarrow h\ (g\ i) = i$
assumes $P \in$ *carrier* (*Pring R I*)
shows *relabel-vars J I h* (*relabel-vars I J g P*) = *P*
⟨*proof*⟩

lemma *relabel-vars-total-eval*:
assumes $g \in I \rightarrow J$
assumes $P \in$ *carrier* (*Pring R I*)
assumes *closed-fun R f*
shows *total-eval R* $(f \circ g)$ *P* = *total-eval R f* (*relabel-vars I J g P*)
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *index-list-indices*:
$k < n \Longrightarrow$ (*index-list n*)!*k* = *k*
⟨*proof*⟩

lemma *index-list-set*:
set (*index-list n*) = {..<*n*}
⟨*proof*⟩

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*
  ⟨*proof*⟩

lemma *insert-at-index-eq*[*simp*]:
  assumes *n* ≤ *length as*
  shows (*insert-at-index as a n*)!*n* = *a*
  ⟨*proof*⟩

lemma *insert-at-index-eq'*[*simp*]:
  assumes *n* ≤ *length as*
  assumes *k* < *n*
  shows (*insert-at-index as a n*)!*k* = *as* ! *k*
  ⟨*proof*⟩

lemma *insert-at-index-eq''*[*simp*]:
  assumes *n* < *length as*
  assumes *k* ≤ *n*
  shows (*insert-at-index as a k*)!(*Suc n*) = *as* ! *n*
  ⟨*proof*⟩

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*⟩

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*
  ⟨*proof*⟩

lemma *set-to-list-length*:
  assumes *finite S*
  shows *length* (*set-to-list S*) = *card S*
  ⟨*proof*⟩

lemma *set-to-list-empty*:
  assumes *card S* = *0*
  shows *set-to-list S* = []
  ⟨*proof*⟩

lemma *set-to-list-first*:
  assumes *card S* > *0*
  shows *Min S* = *set-to-list S* ! *0*
⟨*proof*⟩

lemma *set-to-list-last*:
  assumes *card S* > *0*
  shows *Max S* = *last* (*set-to-list S*)
⟨*proof*⟩

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*]
  ⟨*proof*⟩

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*
  ⟨*proof*⟩

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*
  ⟨*proof*⟩

lemma *nth-elem-Min*:
  assumes *card S* > *0*
  shows *nth-elem S 0* = *Min S*
  ⟨*proof*⟩

lemma *nth-elem-Max*:
  assumes *card S* > *0*
  shows *nth-elem S* (*card S* − *1*) = *Max S*

⟨proof⟩

lemma *nth-elem-Suc*:
assumes *card S > Suc n*
shows *nth-elem S (Suc n) > nth-elem S n*
⟨proof⟩

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*
⟨proof⟩

lemma *set-to-list-Suc-map*:
assumes *finite S*
shows *set-to-list (Suc ‘ S) = map Suc (set-to-list S)*
⟨proof⟩

lemma *nth-elem-Suc-im*:
assumes *i < card S*
shows *nth-elem (Suc ‘ S) i = Suc (nth-elem S i)*
⟨proof⟩

lemma *set-to-list-upto*:
set-to-list {..<n} = [0..<n]
⟨proof⟩

lemma *nth-elem-upto*:
assumes *i < n*
shows *nth-elem {..<n} i = i*
⟨proof⟩

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
⟨proof⟩

lemma *project-at-indices-nth*:
assumes *S ⊆ indices-of as*
assumes *card S > i*
shows *project-at-indices S as ! i = as ! (nth-elem S i)*
⟨proof⟩

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 $x \in S$
shows $\exists\, i.\ i < \mathit{card}\ S \wedge x = \mathit{nth\text{-}elem}\ S\ i$
⟨*proof*⟩

lemma *set-rank-unique*:
assumes *finite S*
assumes $x \in S$
assumes $i < \mathit{card}\ S \wedge x = \mathit{nth\text{-}elem}\ S\ i$
assumes $j < \mathit{card}\ S \wedge x = \mathit{nth\text{-}elem}\ S\ j$
shows $i = j$
⟨*proof*⟩

lemma *nth-elem-set-rank-inv*:
assumes *finite S*
assumes $x \in S$
shows $\mathit{nth\text{-}elem}\ S\ (\mathit{set\text{-}rank}\ S\ x) = x$
⟨*proof*⟩

lemma *set-rank-nth-elem-inv*:
assumes *finite S*
assumes $i < \mathit{card}\ S$
shows $\mathit{set\text{-}rank}\ S\ (\mathit{nth\text{-}elem}\ S\ i) = i$
⟨*proof*⟩

lemma *set-rank-range*:
assumes *finite S*
assumes $x \in S$
shows $\mathit{set\text{-}rank}\ S\ x < \mathit{card}\ S$
⟨*proof*⟩

lemma *project-at-indices-nth′*:
assumes $S \subseteq \mathit{indices\text{-}of}\ \mathit{as}$
assumes $i \in S$
shows $\mathit{as}\ !\ i = \mathit{project\text{-}at\text{-}indices}\ S\ \mathit{as}\ !\ (\mathit{set\text{-}rank}\ S\ i)$
⟨*proof*⟩

fun *proj-away-from-index* :: $\mathit{nat} \Rightarrow {'}a\ \mathit{list} \Rightarrow {'}a\ \mathit{list}$ (‹$\pi_{\neq\,\_}$›)**where**
$\mathit{proj\text{-}away\text{-}from\text{-}index}\ n\ \mathit{as} = (\mathit{take}\ n\ \mathit{as})@(\mathit{drop}\ (\mathit{Suc}\ n)\ \mathit{as})$

proj_away_from_index is an inverse to insert_at_index

lemma *insert-at-index-project-away*[*simp*]:
assumes $k < \mathit{length}\ \mathit{as}$
assumes $\mathit{bs} = (\mathit{insert\text{-}at\text{-}index}\ \mathit{as}\ a\ k)$
shows $\pi_{\neq\ k}\ \mathit{bs} = \mathit{as}$
⟨*proof*⟩

definition *fibred-cell* :: ${'}a\ \mathit{list\ set} \Rightarrow ({'}a\ \mathit{list} \Rightarrow {'}a \Rightarrow \mathit{bool}) \Rightarrow {'}a\ \mathit{list\ set}$ **where**
$\mathit{fibred\text{-}cell}\ C\ P = \{\mathit{as}\ .\ \exists\, x\ t.\ \mathit{as} = (t\#x) \wedge x \in C \wedge (P\ x\ t)\}$

definition *fibred-cell-at-ind* :: $nat \Rightarrow 'a\ list\ set \Rightarrow ('a\ list \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a\ list\ set$ **where**
$fibred\text{-}cell\text{-}at\text{-}ind\ n\ C\ P = \{as\ .\ \exists\, x\ t.\ as = (insert\text{-}at\text{-}index\ x\ t\ n) \wedge x \in C \wedge (P\ x\ t)\}$

lemma *fibred-cell-lengths*:
  assumes $\bigwedge k.\ k \in C \Longrightarrow length\ k = n$
  shows $k \in (fibred\text{-}cell\ C\ P) \Longrightarrow length\ k = Suc\ n$
⟨*proof*⟩

lemma *fibred-cell-at-ind-lengths*:
  assumes $\bigwedge k.\ k \in C \Longrightarrow length\ k = n$
  assumes $k \leq n$
  shows $c \in (fibred\text{-}cell\text{-}at\text{-}ind\ k\ C\ P) \Longrightarrow length\ c = Suc\ n$
⟨*proof*⟩

lemma *project-fibred-cell*:
  assumes $\bigwedge k.\ k \in C \Longrightarrow length\ k = n$
  assumes $k < n$
  assumes $\forall\, x \in C.\ \exists\, t.\ P\ x\ t$
  shows $\pi_{\neq\ k}$ ‘ $(fibred\text{-}cell\text{-}at\text{-}ind\ k\ C\ P) = C$
⟨*proof*⟩

definition *list-segment* **where**
$list\text{-}segment\ i\ j\ as = map\ (nth\ as)\ \ [i..<j]$

lemma *list-segment-length*:
  assumes $i \leq j$
  assumes $j \leq\ length\ as$
  shows $length\ (list\text{-}segment\ i\ j\ as) = j - i$
  ⟨*proof*⟩

lemma *list-segment-drop*:
  assumes $i < length\ as$
  shows $(list\text{-}segment\ i\ (length\ as)\ as) = drop\ i\ \ as$
  ⟨*proof*⟩

lemma *list-segment-concat*:
  assumes $j \leq k$
  assumes $i \leq j$
  shows $(list\text{-}segment\ i\ j\ as)\ @\ (list\text{-}segment\ j\ k\ as) = (list\text{-}segment\ i\ k\ as)$
  ⟨*proof*⟩

lemma *list-segment-subset*:
  assumes $j \leq k$
  shows $set\ (list\text{-}segment\ i\ j\ as) \subseteq set\ (list\text{-}segment\ i\ k\ as)$
  ⟨*proof*⟩

lemma *list-segment-subset-list-set*:
  assumes $j \leq$ *length as*
  shows *set* (*list-segment i j as*) $\subseteq$ *set as*
  ⟨*proof*⟩

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*
  ⟨*proof*⟩

lemma *R-list-nth*:
$i < n \Longrightarrow$ *R-list n R* ! *i* = *R*
  ⟨*proof*⟩

lemma *cartesian-power-car-memI*:
  assumes *length as* = *n*
  assumes *set as* $\subseteq$ *carrier R*
  shows *as* $\in$ *carrier* (R^n)
  ⟨*proof*⟩

lemma *cartesian-power-car-memI'*:
assumes *length as* $= n$
assumes $\bigwedge i.\ i < n \Longrightarrow$ *as* ! $i \in$ *carrier R*
shows *as* $\in$ *carrier* (R^n)
⟨*proof*⟩

lemma *cartesian-power-car-memE*:
assumes *as* $\in$ *carrier* (R^n)
shows *length as* $= n$
⟨*proof*⟩

lemma *cartesian-power-car-memE'*:
assumes *as* $\in$ *carrier* (R^n)
assumes $i < n$
shows *as* ! $i \in$ *carrier R*
⟨*proof*⟩

lemma *cartesian-power-car-memE''*:
assumes *as* $\in$ *carrier* (R^n)
shows *set as* $\subseteq$ *carrier R*
⟨*proof*⟩

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})
⟨*proof*⟩

lemma *cartesian-power-cons*:
assumes *as* $\in$ *carrier* (R^n)
assumes $a \in$ *carrier R*
shows *a#as* $\in$ *carrier* (R^{n+1})
⟨*proof*⟩

lemma *cartesian-power-append*:
assumes *as* $\in$ *carrier* (R^n)
assumes $a \in$ *carrier R*
shows *as*@[*a*] $\in$ *carrier* (R^{n+1})
⟨*proof*⟩

lemma *cartesian-power-head*:
assumes *as* $\in$ *carrier* $(R^{Suc\ n})$
shows *hd as* $\in$ *carrier R*
⟨*proof*⟩

lemma *cartesian-power-tail*:
assumes *as* $\in$ *carrier* $(R^{Suc\ n})$

shows *tl as* $\in$ *carrier* (R^n)
⟨*proof*⟩

lemma *insert-at-index-closed*:
assumes *length as* $=$ *n*
assumes *as* $\in$ *carrier* (R^n)
assumes *a* $\in$ *carrier R*
assumes $k \leq n$
shows (*insert-at-index as a k*) $\in$ *carrier* $(R^{Suc\ n})$
⟨*proof*⟩

lemma *insert-at-index-pow-not-car*:
assumes $k \leq n$
assumes *length x* $=$ *n*
assumes (*insert-at-index x a k*) $\in$ *carrier* $(R^{Suc\ n})$
shows *x* $\in$ *carrier* (R^n)
⟨*proof*⟩

lemma *insert-at-index-pow-not-car'*:
assumes $k \leq n$
assumes *length x* $=$ *n*
assumes *x* $\notin$ *carrier* (R^n)
shows (*insert-at-index x a n*) $\notin$ *carrier* $(R^{Suc\ n})$
⟨*proof*⟩

lemma *take-closed*:
assumes $k \leq n$
assumes *x* $\in$ *carrier* (R^n)
shows *take k x* $\in$ *carrier* (R^k)
⟨*proof*⟩

lemma *drop-closed*:
assumes $k < n$
assumes *x* $\in$ *carrier* (R^n)
shows *drop k x* $\in$ *carrier* $(R^{n\ -\ k})$
⟨*proof*⟩

lemma *last-closed*:
assumes $n > 0$
assumes *x* $\in$ *carrier* (R^n)
shows *last x* $\in$ *carrier R*
⟨*proof*⟩

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})
⟨*proof*⟩

lemma *cartesian-power-decomp*:
  assumes $a \in carrier\ (R^{n+k})$
  obtains $a0\ a1$ **where** $a0 \in carrier\ (R^n) \wedge a1 \in carrier\ (R^k) \wedge a0@a1 = a$
  ⟨*proof*⟩

lemma *list-segment-pow*:
  assumes $as \in carrier\ (R^n)$
  assumes $j \leq n$
  assumes $i \leq j$
  shows $list\text{-}segment\ i\ j\ as \in carrier\ (R^{j - i})$
  ⟨*proof*⟩

lemma *nth-list-segment*:
  assumes $as \in carrier\ (R^n)$
  assumes $j \leq n$
  assumes $i \leq j$
  assumes $k < j - i$
  shows $(list\text{-}segment\ i\ j\ as)\ !\ k = as\ !\ (i + k)$
  ⟨*proof*⟩

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 $as!0 \in carrier\ R$
  shows $as \in carrier\ (R^1)$
  ⟨*proof*⟩

abbreviation(*input*) *to-R1* **where**
$to\text{-}R1\ a \equiv [a]$

abbreviation(*input*) *to-R* :: $'a\ list \Rightarrow\ 'a$ **where**
$to\text{-}R\ as \equiv as!0$

lemma *to-R1-to-R*:
  assumes $a \in carrier\ (R^1)$
  shows $to\text{-}R1\ (to\text{-}R\ a) = a$
⟨*proof*⟩

lemma *to-R-to-R1*:
  shows $to\text{-}R\ (to\text{-}R1\ a) = a$
  ⟨*proof*⟩

lemma *to-R1-closed*:

assumes $a \in$ *carrier* R
shows *to-R1* $a \in$ *carrier* (R^1)
⟨*proof*⟩

lemma *to-R-pow-closed*:
assumes $a \in$ *carrier* (R^1)
shows *to-R* $a \in$ *carrier* R
⟨*proof*⟩

lemma *to-R1-intersection*:
assumes $A \subseteq$ *carrier* R
assumes $B \subseteq$ *carrier* R
shows *to-R1* ‘ $(A \cap B)$ = *to-R1*‘ $A \cap$ *to-R1* ‘ B
⟨*proof*⟩

lemma *to-R1-finite*:
assumes *finite* A
shows *finite* (*to-R1*‘ A)
card A = *card* (*to-R1*‘ A)
⟨*proof*⟩

lemma *to-R1-carrier*:
to-R1‘ (*carrier* R)= *carrier* (R^1)
⟨*proof*⟩

lemma *to-R1-diff*:
to-R1‘ $(A - B)$ = *to-R1*‘ A − *to-R1*‘ B
⟨*proof*⟩

lemma *to-R1-complement*:
shows *to-R1*‘ (*carrier* $R - A$) = *carrier* (R^1) − *to-R1*‘ A
⟨*proof*⟩

lemma *to-R1-subset*:
assumes $A \subseteq B$
shows *to-R1*‘ $A \subseteq$ *to-R1*‘ B
⟨*proof*⟩

lemma *to-R1-car-subset*:
assumes $A \subseteq$ *carrier* R
shows *to-R1*‘ $A \subseteq$ *carrier* (R^1)
⟨*proof*⟩
end

4.3 Simple Cartesian Products

definition *cartesian-product* :: (*'a list*) *set* ⇒ (*'a list*) *set* ⇒ (*'a list*) *set* **where**
cartesian-product A B ≡ {*xs*. ∃ *as* ∈ A. ∃ *bs* ∈ 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*⟩

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*⟩

lemma *cartesian-product-carrier*:
cartesian-product (*carrier* (R^n)) (*carrier* (R^m)) = *carrier* $(R^{n\,+\,m})$
⟨*proof*⟩

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*⟩

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
  ⟨*proof*⟩

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 $a \in B$
  ⟨*proof*⟩

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*⟩

lemma *cartesian-product-subsetI*:

assumes $C \subseteq A$
assumes $D \subseteq B$
shows *cartesian-product* C $D \subseteq$ *cartesian-product* A B
⟨*proof*⟩

lemma *cartesian-product-binary-union-right*:
assumes $C \subseteq$ *carrier* (R^n)
assumes $D \subseteq$ *carrier* (R^n)
shows *cartesian-product* A $(C \cup D) = ($*cartesian-product* A $C) \cup ($*cartesian-product* A $D)$
⟨*proof*⟩

lemma *cartesian-product-binary-union-left*:
assumes $C \subseteq$ *carrier* (R^n)
assumes $D \subseteq$ *carrier* (R^n)
shows *cartesian-product* $(C \cup D)$ $A = ($*cartesian-product* C $A) \cup ($*cartesian-product* D $A)$
⟨*proof*⟩

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*⟩

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*⟩

lemma *cartesian-product-car-complement-right*:
assumes $A \subseteq$ *carrier* (R^m)
shows *carrier* $(R^{n + m})$ $-$ *cartesian-product* $($*carrier* $(R^n))$ $A =$
cartesian-product $($*carrier* $(R^n))$ $(($*carrier* $(R^m))$ $-$ $A)$
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

lemma *cartesian-product-empty-right*:
assumes $A \subseteq$ *carrier* (R^n)
assumes $B = \{[]\}$
shows *cartesian-product* A $B = A$
⟨*proof*⟩

lemma *cartesian-product-empty-left*:
assumes $B \subseteq$ *carrier* (R^n)
assumes $A = \{[]\}$
shows *cartesian-product* A $B = B$
⟨*proof*⟩

4.4 Cartesian Products at Arbitrary Indices

definition(**in** *ring*) *ring-pow-proj* :: *nat* $\Rightarrow$ (*nat set*) $\Rightarrow$ (*'a list*) $\Rightarrow$ (*'a list*)
(‹π-, -›) **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 π_S $a \in$ *carrier* $(R^{card\ S})$
⟨*proof*⟩

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*⟩

lemma(**in** *ring*) *project-at-indices-ring-pow-proj*:
assumes $x \in$ *carrier* (R^n)
shows π_S $x = \pi_{n,S}$ x
⟨*proof*⟩

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 $S\ S'\ A\ B = \{a\ .\ length\ a = card\ S + card\ S' \wedge \pi_S\ a \in A \wedge \pi_{S'}\ a \in B\}$

lemma *twisted-cartesian-product-mem-length*:
  assumes $card\ S = n$
  assumes $card\ S' = m$
  assumes $a \in Prod_{S,S'}\ A\ B$
  shows $length\ a = n + m$
  ⟨*proof*⟩

lemma *twisted-cartesian-product-closed*:
  assumes $A \subseteq carrier\ (R^n)$
  assumes $B \subseteq carrier\ (R^m)$
  assumes $card\ S = n$
  assumes $card\ S' = m$
  assumes $S \cup S' = \{..<n + m\}$
  shows *twisted-cartesian-product* $S\ S'\ A\ B \subseteq carrier\ (R^{n + m})$
⟨*proof*⟩

lemma *twisted-cartesian-product-memE*:
  assumes $a \in$ *twisted-cartesian-product* $S\ S'\ A\ B$
  shows $\pi_S\ a \in A\ \pi_{S'}\ a \in B$
  ⟨*proof*⟩

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* $S\ S'\ A\ B$
  ⟨*proof*⟩

lemma *twisted-cartesian-product-empty-left-factor*:
  assumes $A = \{\}$
  shows *twisted-cartesian-product* $S\ S'\ A\ B = \{\}$
  ⟨*proof*⟩

lemma *twisted-cartesian-product-empty-right-factor*:
  assumes $B = \{\}$
  shows *twisted-cartesian-product* $S\ S'\ A\ B = \{\}$
  ⟨*proof*⟩

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*⟩

lemma *twisted-cartesian-product-swap*:
shows $(Prod_{S,S'}\ A\ B) = (Prod_{S',S}\ B\ A)$
⟨*proof*⟩

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 $\pi_{S'}$ ‘ $(Prod_{S,S'}\ A\ B) = B$
⟨*proof*⟩

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* $i \leq j \wedge j <$ *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 $j = y$
⟨*proof*⟩

lemma *splitting-permutation-case-1-exists*:
assumes $i \leq$ *l1*
assumes $y < i$
shows *splitting-permutation l1 l2 i y* = y
⟨*proof*⟩

lemma *splitting-permutation-case-2-unique*:
assumes $i \leq$ *l1*
assumes $i \leq y \wedge y <$ *l2* + i
assumes *splitting-permutation l1 l2 i j* = y
shows $j = y +$ *l1* $- i$
⟨*proof*⟩

lemma *splitting-permutation-case-2-exists*:

assumes $i \leq l1$
assumes $i \leq y \wedge y < l2 + i$
shows *splitting-permutation l1 l2 i* $(y + l1 - i) = y$
⟨*proof*⟩

lemma *splitting-permutation-case-3-unique*:
assumes $i \leq l1$
assumes $l2 + i \leq y \wedge y < l1 + l2$
assumes *splitting-permutation l1 l2 i j* $= y$
shows $j = y - l2$
⟨*proof*⟩

lemma *splitting-permutation-case-3-exists*:
assumes $i \leq l1$
assumes $l2 + i \leq y \wedge y < l1 + l2$
shows *splitting-permutation l1 l2 i* $(y - l2) = y$
⟨*proof*⟩

lemma *splitting-permutation-case-4-unique*:
assumes $i \leq l1$
assumes $l1 + l2 \leq y$
assumes *splitting-permutation l1 l2 i j* $= y$
shows $j = y$
⟨*proof*⟩

lemma *splitting-permutation-case-4-exists*:
assumes $i \leq l1$
assumes $l1 + l2 \leq y$
shows *splitting-permutation l1 l2 i y* $= y$
⟨*proof*⟩

lemma *splitting-permutation-permutes*:
assumes $i \leq l1$
shows (*splitting-permutation l1 l2 i*) *permutes* $\{..< l1 + l2\}$
⟨*proof*⟩

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*⟩

definition *scp-permutation* **where**
scp-permutation l1 l2 i = *fun-inv* (*splitting-permutation l1 l2 i*)

lemma *scp-permutation-action*:

assumes $i \le l1$
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*⟩

lemma *scp-permutes*:
assumes $i \le l1$
shows (*scp-permutation* $l1$ $l2$ i) *permutes* $\{..< l1 + l2\}$
⟨*proof*⟩

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 $a1$@$a2 \in A$
assumes $b \in B$
assumes $A \subseteq$ *carrier* (R^{l1})
assumes $B \subseteq$ *carrier* (R^{l2})
assumes *length* $a1 = i$
shows $a1$@b@$a2 \in$ *split-cartesian-product* $l1$ $l2$ i A B
⟨*proof*⟩

lemma *split-cartesian-product-memI'*:
assumes $a \in A$
assumes $b \in B$
assumes $A \subseteq$ *carrier* (R^{l1})
assumes $B \subseteq$ *carrier* (R^{l2})
assumes $i \le l1$
shows (*take* i a)@b@(*drop* i a) $\in$ *split-cartesian-product* $l1$ $l2$ i A B
⟨*proof*⟩

lemma *split-cartesian-product-memE*:
assumes $a \in$ *split-cartesian-product* $l1$ $l2$ i A B
assumes $A \subseteq$ *carrier* (R^{l1})
assumes $B \subseteq$ *carrier* (R^{l2})
assumes $i \le l1$
shows (*take* i a)@(*drop* $(i + l2)$ a) $\in A$
(*drop* i (*take* $(i + l2)$ a)) $\in B$
⟨*proof*⟩

lemma *split-cartesian-product-mem-length*:
assumes $a \in$ *split-cartesian-product* $l1$ $l2$ i A B
assumes $A \subseteq$ *carrier* (R^{l1})
assumes $B \subseteq$ *carrier* (R^{l2})
assumes $i \le l1$
shows *length* $a = l1 + l2$

⟨*proof*⟩

lemma *split-cartesian-product-memE′*:
assumes $a1@b@a2 \in$ *split-cartesian-product l1 l2 i A B*
assumes $A \subseteq$ *carrier* (R^{l1})
assumes $B \subseteq$ *carrier* (R^{l2})
assumes $i \leq l1$
assumes *length* $a1 = i$
assumes *length* $b = l2$
assumes *length* $as = (l1 - i)$
shows $a1@a2 \in A$
$b \in B$
⟨*proof*⟩

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*⟩

General function for permuting the elements of a simple cartesian product:

definition *intersperse* :: $(nat \Rightarrow nat) \Rightarrow 'a\ tuple \Rightarrow 'a\ tuple \Rightarrow 'a\ tuple$ **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
⟨*proof*⟩

lemma *intersperseE′*:
assumes σ *permutes* $(\{..<n\})$
assumes *length as* + *length bs* = n
assumes *length as* = k
assumes $\sigma\ i < k$
shows (*intersperse* σ *as bs*)! i = *as* ! $\sigma\ i$
⟨*proof*⟩

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* σ *as bs*)! i = *bs* ! $((\sigma\ i) - k)$
⟨*proof*⟩

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-at-indices* $\{i..<j\}$ a) $= j - i$
⟨*proof*⟩

lemma *project-at-indices-consecutive-ind-length′*:
assumes $(i::nat) < j$
assumes $j \leq n$
assumes $a \in$ *carrier* (R^n)
shows *length* (*project-at-indices* $\{i..<j\}$ a) $= j - i$
⟨*proof*⟩

lemma *sorted-list-of-set-from-up-to*:
assumes $(i::nat) < j$
assumes $k < j - i$
shows *sorted-list-of-set* $\{i..<j\}$! $k = i + k$
⟨*proof*⟩

lemma *nth-elem-consecutive-indices*:
assumes $(i::nat) < j$
assumes $k < j - i$
shows *nth-elem* $\{i..<j\}$ $k = i + k$
⟨*proof*⟩

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) ! $k = a!$ $(i + k)$
⟨*proof*⟩

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) ! $k = a!$ $(i + k)$
⟨*proof*⟩

lemma *tl-as-projection*:
assumes $a \in$ *carrier* (R^n)
shows *tl* a = *project-at-indices* $\{1::nat..<n\}$ a
⟨*proof*⟩

4.5 Function Rings on Cartesian Powers

Complement operator

definition *ring-pow-comp* :: ($'a$, $'b$) *ring-scheme* $\Rightarrow$ *arity* $\Rightarrow$ $'a$ *tuple set* $\Rightarrow$ $'a$ *tuple set* **where**
ring-pow-comp R n S $\equiv$ *carrier* $(R^n) - S$

lemma *ring-pow-comp-closed*:
ring-pow-comp R n $S \subseteq$ *carrier* (R^n)
⟨*proof*⟩

lemma *ring-pow-comp-disjoint*:
ring-pow-comp R n $S \cap S = \{\}$
⟨*proof*⟩

lemma *ring-pow-comp-union*:
assumes $S \subseteq$ *carrier* (R^n)
shows (*ring-pow-comp* R n S) $\cup$ S = *carrier* (R^n)
⟨*proof*⟩

lemma *ring-pow-comp-carrier*:
ring-pow-comp R n (*carrier* (R^n)) = $\{\}$
⟨*proof*⟩

lemma *ring-pow-comp-empty*:
ring-pow-comp R n $\{\}$ = (*carrier* (R^n))
⟨*proof*⟩

lemma *ring-pow-comp-demorgans*:
assumes $A \subseteq$ *carrier* (R^n)
assumes $B \subseteq$ *carrier* (R^n)
shows *ring-pow-comp* R n $(A \cup B)$ = (*ring-pow-comp* R n A) $\cap$ (*ring-pow-comp* R n B)
⟨*proof*⟩

lemma *ring-pow-comp-demorgans'*:
assumes $A \subseteq$ *carrier* (R^n)
assumes $B \subseteq$ *carrier* (R^n)
shows *ring-pow-comp* R n $(A \cap B)$ = (*ring-pow-comp* R n A) $\cup$ (*ring-pow-comp* R n B)
⟨*proof*⟩

lemma *ring-pow-comp-inv*:
assumes $A \subseteq$ *carrier* (R^n)
shows *ring-pow-comp* R n (*ring-pow-comp* R n A) = A
⟨*proof*⟩

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*
  ⟨*proof*⟩

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*
  ⟨*proof*⟩

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*
  ⟨*proof*⟩

lemma(**in** *ring*) *partial-eval-domain*:
  assumes $f \in$ *carrier* ($Fun_{Suc\ k}$ *R*)
  assumes $a \in$ *carrier R*
  assumes $n \le k$
  shows (*partial-eval R k n f a*) $\in$ *carrier* (Fun_k *R*)
  ⟨*proof*⟩

Pullbacks preserve ring power functions

lemma *fun-struct-maps*:
struct-maps (R^n) *R* = *carrier* (Fun_n *R*)
⟨*proof*⟩

lemma *pullback-fun-closed*:
  assumes $f \in$ *struct-maps* (R^n) (R^m)
  assumes $g \in$ *carrier* (Fun_m *R*)
  shows *pullback* (R^n) *f g* $\in$ *carrier* (Fun_n *R*)
  ⟨*proof*⟩

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*) $\Rightarrow$ *arity* $\Rightarrow$ (*'a tuple* $\Rightarrow$ *'a*) => (*'a tuple* $\Rightarrow$ *'a*) **where**
ring-pow-inc S n f = *pullback* (R^n) ($\pi_{n,S}$) *f*

lemma *ring-pow-inc-is-fun*:
  assumes $S \subseteq \{..<n\}$
  assumes $f \in$ *carrier* ($Fun_{card\ S}$ *R*)
  shows *ring-pow-inc S n f* $\in$ *carrier* (Fun_n *R*)
  ⟨*proof*⟩

The "standard" inclusion of powers of function rings into one another

abbreviation(*input*) *std-proj*:: *nat* $\Rightarrow$ *nat* $\Rightarrow$ (*'a list*) $\Rightarrow$ (*'a list*) **where**
std-proj n m $\equiv$ *ring-pow-proj n* ($\{..<m\}$)

lemma *std-proj-id*:
  assumes $m \leq n$
  assumes *as* $\in$ *carrier* (R^n)
  assumes $i < m$
  shows *std-proj n m as* ! *i* = *as* ! *i*
⟨*proof*⟩

abbreviation(*input*) *std-inc*:: *nat* $\Rightarrow$ *nat* $\Rightarrow$ (*'a list* $\Rightarrow$ *'a*) => (*'a list* $\Rightarrow$ *'a*) **where**
std-inc n m f $\equiv$ *ring-pow-inc* ($\{..<m\}$) *n f*

lemma *std-proj-is-map*[*simp*]:
  assumes $m \leq n$
  shows *std-proj n m* $\in$ *struct-maps* (R^n) (R^m)
  ⟨*proof*⟩

end

4.6 Coordinate Functions

definition *var* :: (*'a*, *'b*) *ring-scheme* $\Rightarrow$ *nat* $\Rightarrow$ *nat* $\Rightarrow$ (*'a list* $\Rightarrow$ *'a*) **where**
var R n i = *restrict* ($\lambda x.\ x!i$) (*carrier* (R^n))

context *ring*
begin

lemma *var-in-car*:
  assumes $i < n$
  shows *var R n i* $\in$ *carrier* (Fun_n *R*)
  ⟨*proof*⟩

lemma *varE*[*simp*]:
  assumes $i < n$
  assumes $x \in carrier\ (R^n)$
  shows $var\ R\ n\ i\ x = x\ !\ i$
  ⟨*proof*⟩

lemma *std-inc-of-var*:
  assumes $i < n$
  assumes $n \leq m$
  shows $std\text{-}inc\ m\ n\ (var\ R\ n\ i) = \ (var\ R\ m\ i)$
  ⟨*proof*⟩

abbreviation *variable* (‹$\mathfrak{v}_{\text{-}}$›) **where**
$variable\ n\ i \equiv var\ R\ n\ i$

end

definition $var\text{-}set :: ('a, 'b)\ ring\text{-}scheme \Rightarrow\ nat \Rightarrow ('a\ list \Rightarrow 'a)\ set$ **where**
$var\text{-}set\ R\ n = var\ R\ n\ `\ \{..<n\}$

lemma *var-setE*:
  assumes $f \in var\text{-}set\ R\ n$
  obtains i **where** $f = var\ R\ n\ i \wedge i \in \{..<n\}$
  ⟨*proof*⟩

lemma *var-setI*:
  assumes $i \in \{..<n\}$
  assumes $f = var\ R\ n\ i$
  shows $f \in var\text{-}set\ R\ n$
  ⟨*proof*⟩

context *ring*
begin

lemma *var-set-in-fun-ring-car*:
  shows $var\text{-}set\ R\ n \subseteq carrier\ Fun_n\ R$
⟨*proof*⟩

end

4.7 Graphs of functions

definition $fun\text{-}graph:: ('a, 'b)\ ring\text{-}scheme \Rightarrow nat \Rightarrow ('a\ list \Rightarrow 'a) \Rightarrow 'a\ list\ set$
where
$fun\text{-}graph\ R\ n\ f = \{as.\ (\exists\, x \in carrier\ (R^n).\ as = x\ @\ [f\ x])\}$

context *ring*
begin

lemma *function-ring-car-memE*:
  assumes $f \in carrier\ (Fun_n\ R)$
  assumes $a \in carrier\ (R^n)$
  shows $f\ a \in carrier\ R$
  ⟨*proof*⟩

lemma *graph-range*:
  assumes $f \in carrier\ (Fun_n\ R)$
  shows $fun\text{-}graph\ R\ n\ f \subseteq carrier\ (R^{Suc\ n}\ )$
⟨*proof*⟩

lemma *fun-graph-memE*:
  assumes $f \in carrier\ (Fun_n\ R)$
  assumes $p \in fun\text{-}graph\ R\ n\ f$
  shows $(take\ n\ p) \in carrier\ (R^n)$
    ⟨*proof*⟩

lemma *fun-graph-memE'*:
  assumes $f \in carrier\ (Fun_n\ R)$
  assumes $p \in fun\text{-}graph\ R\ n\ f$
  shows $f\ (take\ n\ p) = p!n$
  ⟨*proof*⟩

apply a function f to the tuple consisting of the first n indices, leaving the remaining indices unchanged

definition *partial-image* :: $arity \Rightarrow ('c\ list \Rightarrow 'c) \Rightarrow 'c\ list \Rightarrow 'c\ list$ **where**
$partial\text{-}image\ n\ f\ as = (f\ (take\ n\ as))\ \#\ (drop\ n\ as)$

lemma *partial-image-range*:
  assumes $f \in carrier\ (Fun_n\ R)$
  assumes $m \geq n$
  assumes $as \in carrier\ (R^m)$
  shows $partial\text{-}image\ n\ f\ as \in carrier\ (R^{m\ -\ n\ +\ 1})$
⟨*proof*⟩

end

5 Coordinate Rings on Cartesian Powers

5.1 Basic Facts and Definitions

locale *cring-coord-rings* = *UP-cring* +
  assumes *one-neq-zero*: $\mathbf{1} \neq \mathbf{0}$

coordinate polynomial ring in n variables over a commutative ring

definition *coord-ring* :: $('a,\ 'b)$ *ring-scheme* $\Rightarrow$ *nat* $\Rightarrow$ $('a,\ ('a,\ nat)$ *mvar-poly*$)$ *module*
(‹- $[\mathcal{X}_-]$› *80*) **where** $R[\mathcal{X}_n] \equiv$ *Pring R* $\{..<\ n::nat\}$

sublocale *cring-coord-rings* < *cring-functions R carrier* (R^n) Fun_n *R*
⟨*proof*⟩

sublocale *cring-coord-rings* < *MP?*: *cring* $R[\mathcal{X}_n]$
⟨*proof*⟩

sublocale *cring-coord-rings* < *F?*: *cring* Fun_n *R*
⟨*proof*⟩

context *cring-coord-rings*
begin

lemma *coord-cring-cring*:
cring $(R[\mathcal{X}_n])$ ⟨*proof*⟩

coordinate constant functions

abbreviation(*input*) *coord-const* :: $'a \Rightarrow ('a,\ nat)$ *mvar-poly* **where**
coord-const k $\equiv$ *ring.indexed-const R k*

lemma *coord-const-ring-hom*:
ring-hom-ring R $(R[\mathcal{X}_n])$ *coord-const*
⟨*proof*⟩

coordinate functions

lemma *pvar-closed*:
assumes $i < n$
shows *pvar R i* $\in$ *carrier* $(R[\mathcal{X}_n])$
⟨*proof*⟩

relationship between multiplciation by a variable and index multiplcation

lemma *pvar-indexed-pmult*:
assumes $p \in$ *carrier* $(R[\mathcal{X}_n])$
shows $(p \bigotimes i) = p \otimes_{R[\mathcal{X}_n]}$ *pvar R i*
⟨*proof*⟩

lemma *coord-ring-cfs-closed*:
assumes $p \in$ *carrier* $(R[\mathcal{X}_n])$
shows $p\ m \in$ *carrier R*
⟨*proof*⟩

lemma *coord-ring-plus*:
assumes $p \in$ *carrier* $(R[\mathcal{X}_n])$
assumes $Q \in$ *carrier* $(R[\mathcal{X}_n])$
shows $(p \oplus_{R[\mathcal{X}_n]} Q)\ m = p\ m \oplus Q\ m$

⟨*proof*⟩

lemma *coord-ring-uminus*:
  assumes $p \in carrier\ (R[\mathcal{X}_n])$
  shows $(\ominus_{R[\mathcal{X}_n]}\ p)\ m = \ominus\ (p\ m)$
  ⟨*proof*⟩

lemma *coord-ring-minus*:
  assumes $p \in carrier\ (R[\mathcal{X}_n])$
  assumes $Q \in carrier\ (R[\mathcal{X}_n])$
  shows $(p \ominus_{R[\mathcal{X}_n]} Q)\ m = p\ m \ominus Q\ m$
  ⟨*proof*⟩

lemma *coord-ring-one*:
$\mathbf{1}_{R[\mathcal{X}_n]}\ m = (if\ m = \{\#\}\ then\ \mathbf{1}\ else\ \mathbf{0})$
  ⟨*proof*⟩

lemma *coord-ring-zero*:
$\mathbf{0}_{R[\mathcal{X}_n]}\ m = \mathbf{0}$
  ⟨*proof*⟩

Evaluation of a polynomial at a point

end

abbreviation(*input*) *point-to-eval-map* **where**
point-to-eval-map R as ≡ (λ*i*. (*if i*< *length as then as* ! *i 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* (λ*i*. (*if i*< *length as then as* ! *i 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*
  ⟨*proof*⟩

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* $\in$ *carrier R*
⟨*proof*⟩

lemma *eval-pvar*:
assumes $i < (n::nat)$
assumes $a \in$ *carrier* (R^n)
shows *eval-at-point R a* (*pvar R i*) = *a*!*i*
⟨*proof*⟩

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*
⟨*proof*⟩

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* $(A \oplus_{coord\text{-}ring\ R\ \ n} B)$ =
eval-at-point R a A $\oplus_R$ *eval-at-point R a B*
⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

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*)
⟨*proof*⟩

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*)

⟨*proof*⟩

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 $eval\text{-}at\text{-}point\ R\ a\ (k \odot_{R[\mathcal{X}_n]} A) = k \otimes_R (eval\text{-}at\text{-}point\ R\ a\ A)$
  ⟨*proof*⟩

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 $eval\text{-}at\text{-}point\ R\ a\ (A \ominus_{coord\text{-}ring\ R\ \ n} B) =$
    $eval\text{-}at\text{-}point\ R\ a\ A \ominus_R eval\text{-}at\text{-}point\ R\ a\ B$
  ⟨*proof*⟩

lemma *eval-at-point-a-inv*:
  assumes $a \in carrier\ (R^n)$
  assumes $B \in carrier\ (coord\text{-}ring\ R\ \ n)$
  shows $eval\text{-}at\text{-}point\ R\ a\ (\ominus_{R[\mathcal{X}_n]} B) = \ominus_R\ eval\text{-}at\text{-}point\ R\ a\ B$
  ⟨*proof*⟩

lemma *eval-at-point-nat-pow*:
  assumes $a \in carrier\ (R^n)$
  assumes $A \in carrier\ (R[\mathcal{X}_n])$
  shows $eval\text{-}at\text{-}point\ R\ a\ (A[\urcorner]_{R[\mathcal{X}_n]}(k{::}nat)) = (eval\text{-}at\text{-}point\ R\ a\ A)[\urcorner]k$
  ⟨*proof*⟩

end

5.3 Partial Evaluation of a Polynomial

definition *coord-partial-eval* ::
  $('a,\ 'b)\ ring\text{-}scheme \Rightarrow nat\ set \Rightarrow\ 'a\ list \Rightarrow ('a,\ nat)\ mvar\text{-}poly \Rightarrow ('a,\ nat)$ *mvar-poly* **where**
$coord\text{-}partial\text{-}eval\ R\ S\ as = poly\text{-}eval\ R\ S\ (point\text{-}to\text{-}eval\text{-}map\ R\ as)$

context *cring-coord-rings*
begin

lemma *point-to-eval-map-closed*:
  assumes $as \in carrier\ (R^n)$
  shows $closed\text{-}fun\ R\ (point\text{-}to\text{-}eval\text{-}map\ R\ as)$
  ⟨*proof*⟩

lemma *coord-partial-eval-hom*:
  assumes $as \in carrier\ (R^n)$
  shows $coord\text{-}partial\text{-}eval\ R\ S\ as \in ring\text{-}hom\ (R[\mathcal{X}_n])\ (R[\mathcal{X}_n])$

$\langle proof \rangle$

lemma *coord-partial-eval-hom'*:
assumes $as \in carrier\ (R^n)$
shows $coord\text{-}partial\text{-}eval\ R\ S\ as \in ring\text{-}hom\ (R[\mathcal{X}_n])\ (Pring\ R\ (\{..<n\} - S))$
$\langle proof \rangle$

lemma *coord-partial-eval-closed*:
assumes $S \subseteq \{..<n\}$
assumes $\{..<n\} - S \subseteq I$
assumes $as \in carrier\ (R^n)$
assumes $p \in carrier\ (R[\mathcal{X}_n])$
shows $coord\text{-}partial\text{-}eval\ R\ S\ as\ p \in carrier\ (Pring\ R\ I)$
$\langle proof \rangle$

lemma *coord-partial-eval-add*:
assumes $as \in carrier\ (R^n)$
assumes $p \in carrier\ (R[\mathcal{X}_n])$
assumes $Q \in carrier\ (R[\mathcal{X}_n])$
shows $coord\text{-}partial\text{-}eval\ R\ S\ as\ \ (p \oplus_{(R[\mathcal{X}_n])} Q) =$
$(coord\text{-}partial\text{-}eval\ R\ S\ as\ \ p) \oplus_{(R[\mathcal{X}_n])} (coord\text{-}partial\text{-}eval\ R\ S\ as\ \ Q)$
$\langle proof \rangle$

lemma *coord-partial-eval-mult*:
assumes $as \in carrier\ (R^n)$
assumes $p \in carrier\ (R[\mathcal{X}_n])$
assumes $Q \in carrier\ (R[\mathcal{X}_n])$
shows $coord\text{-}partial\text{-}eval\ R\ S\ as\ \ (p \otimes_{(R[\mathcal{X}_n])} Q) =$
$(coord\text{-}partial\text{-}eval\ R\ S\ as\ \ p) \otimes_{(R[\mathcal{X}_n])} (coord\text{-}partial\text{-}eval\ R\ S\ as\ \ Q)$
$\langle proof \rangle$

lemma *coord-partial-eval-pvar*:
assumes $\mathbf{1} \neq \mathbf{0}$
assumes $as \in carrier\ (R^n)$
assumes $i \in S \cap \{..<n\}$
shows $coord\text{-}partial\text{-}eval\ R\ S\ as\ (pvar\ R\ i) = coord\text{-}const\ (as!i)$
$\langle proof \rangle$

lemma *coord-partial-eval-pvar'*:
assumes $\mathbf{1} \neq \mathbf{0}$
assumes $as \in carrier\ (R^n)$
assumes $i \notin S$
shows $coord\text{-}partial\text{-}eval\ R\ S\ as\ (pvar\ R\ i) = (pvar\ R\ i)$
$\langle proof \rangle$

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 $p\ A$
⟨*proof*⟩

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* $\Rightarrow$ *nat* $\Rightarrow$ $('a,\ nat)$ *mvar-poly* $\Rightarrow$ $'a$ *list set*
(‹V_1›) **where**
$zero\text{-}set\ R\ n\ p =\ \{a \in carrier\ (R^n).\ eval\text{-}at\text{-}point\ R\ a\ p = \mathbf{0}_R\}$

context *cring-coord-rings*
begin

lemma *zero-setI*:
assumes $a \in carrier\ (R^n)$
assumes $eval\text{-}at\text{-}point\ R\ a\ p = \mathbf{0}_R$
shows $a \in zero\text{-}set\ R\ n\ p$
⟨*proof*⟩

lemma *zero-setE*:
assumes $a \in zero\text{-}set\ R\ n\ p$
shows $a \in carrier\ (R^n)$
$eval\text{-}at\text{-}point\ R\ a\ p = \mathbf{0}_R$
⟨*proof*⟩

lemma *zero-set-closed*:
$zero\text{-}set\ R\ n\ p \subseteq carrier\ (R^n)$
⟨*proof*⟩

end

5.5.2 The Zero Set of a Collection of Polynomials

definition *affine-alg-set* :: $('a,\ 'b)$ *ring-scheme* $\Rightarrow$ *nat* $\Rightarrow$ $('a,\ nat)$ *mvar-poly set* $\Rightarrow$ $'a$ *list set*
where $affine\text{-}alg\text{-}set\ R\ n\ as = \{a \in carrier\ (R^n).\ \forall\ b \in as.\ a \in (zero\text{-}set\ R\ n\ b)\}$

context *cring-coord-rings*
begin

lemma *affine-alg-set-empty*:
affine-alg-set R n $\{\}$ = *carrier* (R^n)
⟨*proof*⟩

lemma *affine-alg-set-subset-zero-set*:
assumes $b \in as$
shows *affine-alg-set R n as* $\subseteq$ (*zero-set R n b*)
⟨*proof*⟩

lemma(**in** *cring-coord-rings*) *affine-alg-set-memE*:
assumes $b \in as$
assumes $a \in$ *affine-alg-set R n as*
shows *eval-at-poly R b a* = $\mathbf{0}$
⟨*proof*⟩

lemma *affine-alg-set-subset*:
assumes $as \subseteq bs$
shows *affine-alg-set R n bs* $\subseteq$ *affine-alg-set R n as*
⟨*proof*⟩

lemma *affine-alg-set-empty-set*:
assumes $as = \{\}$
shows *affine-alg-set R n as* = *carrier* (R^n)
⟨*proof*⟩

lemma *affine-alg-set-closed*:
shows *affine-alg-set R n as* $\subseteq$ *carrier* (R^n)
⟨*proof*⟩

lemma *affine-alg-set-singleton*:
affine-alg-set R n $\{a\}$ = *zero-set R n a*
⟨*proof*⟩

lemma *affine-alg-set-insert*:
affine-alg-set R n (*insert a A*) = *zero-set R n a* $\cap$ (*affine-alg-set R n A*)
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *affine-alg-set-memI*:
assumes $a \in$ *carrier* (R^n)
assumes $\bigwedge p.\ p \in B \Longrightarrow$ *eval-at-point R a p* = $\mathbf{0}$
shows $a \in$ (*affine-alg-set R n B*)
⟨*proof*⟩

lemma *affine-alg-set-not-memE*:
assumes $a \in$ *carrier* (R^n)

assumes $a \notin (\textit{affine-alg-set } R\ n\ B)$
shows $\exists\, b \in B.\ \textit{eval-at-poly } R\ b\ a \neq \mathbf{0}$
⟨*proof*⟩

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**
$\textit{prod-set as bs} = (\lambda x.\ \textit{fst } x \otimes \textit{snd } x)\ `\ (\textit{as} \times \textit{bs})$

lemma(**in** *ring*) *prod-setI*:
assumes $c \in \textit{prod-set as bs}$
shows $\exists\, a \in \textit{as}.\ \exists\, b \in \textit{bs}.\ c = a \otimes b$
⟨*proof*⟩

lemma(**in** *ring*) *prod-set-closed*:
assumes $\textit{as} \subseteq \textit{carrier } R$
assumes $\textit{bs} \subseteq \textit{carrier } R$
shows $\textit{prod-set as bs} \subseteq \textit{carrier } R$
⟨*proof*⟩

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 $\textit{finite } (\textit{prod-set as bs})\ \textit{card } (\textit{prod-set as bs}) \leq \textit{card as} * \textit{card bs}$
⟨*proof*⟩

definition *poly-prod-set* **where**
$\textit{poly-prod-set } n\ \textit{as bs} = \textit{ring.prod-set } (R[\mathcal{X}_n])\ \textit{as bs}$

lemma *poly-prod-setE*:
assumes $c \in \textit{poly-prod-set } n\ \textit{as bs}$
shows $\exists\, a \in \textit{as}.\ \exists\, b \in \textit{bs}.\ c = a \otimes_{R[\mathcal{X}_n]} b$
⟨*proof*⟩

lemma *poly-prod-setI*:
assumes $a \in \textit{as}$
assumes $b \in \textit{bs}$
shows $a \otimes_{R[\mathcal{X}_n]} b \in \textit{poly-prod-set } n\ \textit{as bs}$
⟨*proof*⟩

lemma *poly-prod-set-closed*:
assumes $\textit{as} \subseteq \textit{carrier } (R[\mathcal{X}_n])$
assumes $\textit{bs} \subseteq \textit{carrier } (R[\mathcal{X}_n])$
shows $\textit{poly-prod-set } n\ \textit{as bs} \subseteq \textit{carrier } (R[\mathcal{X}_n])$

⟨proof⟩

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*

⟨proof⟩

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⟩

definition *is-algebraic* :: ('*a*, '*b*) *ring-scheme* $\Rightarrow$ *nat* $\Rightarrow$ '*a list set* $\Rightarrow$ *bool* **where**
is-algebraic R n S = ($\exists$ *ps*. *finite ps* $\wedge$ *ps* $\subseteq$ *carrier* ($R[\mathcal{X}_n]$) $\wedge$ *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* $\subseteq$ *carrier* ($R[\mathcal{X}_n]$) *S* = *affine-alg-set R n ps*
⟨proof⟩

lemma *is-algebraicI*:
assumes *finite ps*
assumes *ps* $\subseteq$ *carrier* ($R[\mathcal{X}_n]$)
assumes *S* = *affine-alg-set R n ps*
shows *is-algebraic R n S*
⟨proof⟩

lemma *is-algebraicI'*:
assumes *p* $\in$ *carrier* ($R[\mathcal{X}_n]$)
assumes *S* = *zero-set R n p*
shows *is-algebraic R n S*
⟨proof⟩

end

definition *alg-sets* :: *arity* $\Rightarrow$ ('*a*, '*b*) *ring-scheme* $\Rightarrow$ ('*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 $(A \cap B)$
⟨*proof*⟩

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 $(A \cup B)$
⟨*proof*⟩

lemma *zero-set-zero*:
zero-set R n $\mathbf{0}_{R[\mathcal{X}_n]}$ = *carrier* (R^n)
  ⟨*proof*⟩

lemma *affine-alg-set-set*:
affine-alg-set R n $\{\mathbf{0}_{R[\mathcal{X}_n]}\}$ = *carrier* (R^n)
⟨*proof*⟩

lemma *car-is-alg*:
is-algebraic R n (*carrier* (R^n))
  ⟨*proof*⟩

lemma *zero-set-nonzero-constant*:
  assumes $a \neq \mathbf{0}$
  assumes $a \in$ *carrier* R
  shows *zero-set* R n (*coord-const* a) = {}
⟨*proof*⟩

lemma *zero-set-one*:
  assumes $a \neq \mathbf{0}$
  assumes $a \in$ *carrier* R
  shows *zero-set* R n $\mathbf{1}_{R[\mathcal{X}_n]}$ = {}
  ⟨*proof*⟩

lemma *empty-set-as-affine-alg-set*:
affine-alg-set R n $\{\mathbf{1}_{R[\mathcal{X}_n]}\}$ = {}
  ⟨*proof*⟩

lemma *empty-is-alg*:
is-algebraic R n {}
  ⟨*proof*⟩

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* ($R[\mathcal{X}_n]$)
  ⟨*proof*⟩

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*⟩

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*
  ⟨*proof*⟩

lemma *point-to-polysE*:
  assumes *i* < *length as*
  shows (*point-to-polys as*) ! *i* = (*pvar-trans* (*length as*) *i* (*as* ! *i*))
⟨*proof*⟩

lemma *point-to-polysE'*:
  assumes *as* ∈ *carrier* (R^n)
  assumes *i* < *n*
  shows *eval-at-point R as* ((*point-to-polys as*) ! *i*) = **0**
  ⟨*proof*⟩

lemma *point-to-polysE''*:
  assumes *as* ∈ *carrier* (R^n)
  assumes *b* ∈ *set* (*point-to-polys as*)
  shows *eval-at-point R as b* = **0**
  ⟨*proof*⟩

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*

$\langle proof \rangle$

lemma *point-to-polys-closed*:
assumes *as* $\in$ *carrier* (R^n)
shows *set* (*point-to-polys as*) $\subseteq$ *carrier* $(R[\mathcal{X}_n])$
$\langle proof \rangle$

lemma *point-to-polys-affine-alg-set*:
assumes *as* $\in$ *carrier* (R^n)
shows *affine-alg-set R n* (*set* (*point-to-polys as*)) = $\{as\}$
$\langle proof \rangle$

lemma *singleton-is-algebraic*:
assumes *as* $\in$ *carrier* (R^n)
shows *is-algebraic R n* $\{as\}$
$\langle proof \rangle$

lemma(**in** *domain-coord-rings*) *finite-sets-are-algebraic*:
assumes *finite F*
shows $F \subseteq$ *carrier* $(R^n) \longrightarrow$ *is-algebraic R n F*
$\langle proof \rangle$

5.6 Polynomial Maps

5.7 The Action of Index Permutations on Polynomials

definition *permute-poly-args* ::
nat $\Rightarrow$ (*nat* $\Rightarrow$ *nat*) $\Rightarrow$ ('*a*, *nat*) *mvar-poly* $\Rightarrow$ ('*a*, *nat*) *mvar-poly* **where**
permute-poly-args (*n*::*nat*) σ *p* = *indexed-poly-induced-morphism* $\{..<n\}$ $(R[\mathcal{X}_n])$
coord-const (λi. *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))
($\forall i \in \{..<n\}$. (*permute-poly-args* (*n*::*nat*) σ) (*pvar R i*) = *pvar R* (σ *i*))
($\forall a \in$ *carrier R*. *permute-poly-args* (*n*::*nat*) σ (*coord-const a*) = (*coord-const* *a*))
$\langle proof \rangle$

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* $(R[\mathcal{X}_n])$
$\langle proof \rangle$

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*)

⟨*proof*⟩

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* $n\ \sigma\ (p \oplus_{R[\mathcal{X}_n]} q) = ($*permute-poly-args* $n\ \sigma\ p) \oplus_{R[\mathcal{X}_n]}$ $($*permute-poly-args* $n\ \sigma\ q)$
⟨*proof*⟩

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* $n\ \sigma\ (p \otimes_{R[\mathcal{X}_n]} q) = ($*permute-poly-args* $n\ \sigma\ p) \otimes_{R[\mathcal{X}_n]}$ $($*permute-poly-args* $n\ \sigma\ q)$
⟨*proof*⟩

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* $n\ \sigma\ (p \bigotimes\ i)) = ($*permute-poly-args* $n\ \sigma\ p) \bigotimes\ (\sigma\ i)$
⟨*proof*⟩

lemma *permute-list-closed*:
assumes $a \in$ *carrier* (R^n)
assumes σ *permutes* $\{..<n\}$
shows $($*permute-list* $\sigma\ a) \in$ *carrier* (R^n)
⟨*proof*⟩

lemma *permute-list-set*:
assumes $a \in$ *carrier* (R^n)
assumes σ *permutes* $\{..<n\}$
shows *set* $($*permute-list* $\sigma\ a) =$ *set* a
⟨*proof*⟩

end

definition *perm-map* :: $('a, 'b)$ *ring-scheme* $\Rightarrow$ *nat* $\Rightarrow$ $($*nat* $\Rightarrow$ *nat*$)$ $\Rightarrow$ $'a$ *list* $\Rightarrow$ $'a$ *list* **where**
perm-map $R\ n\ \sigma =$ *restrict* $($*permute-list* $\sigma)$ $($*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* $(R^n)\ (R^n)$

⟨*proof*⟩

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* $n\ \sigma\ p$) = *eval-at-point* R (*permute-list* $\sigma\ a$) p
⟨*proof*⟩

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* ! $j \in$ *carrier* ($R[\mathcal{X}_n]$)
⟨*proof*⟩

lemma *is-poly-tuple-Cons*:
assumes *is-poly-tuple n fs*
assumes $f \in$ *carrier* ($R[\mathcal{X}_n]$)
shows *is-poly-tuple n* (f#*fs*)
⟨*proof*⟩

lemma *is-poly-tuple-append*:
assumes *is-poly-tuple n fs*
assumes $f \in$ *carrier* ($R[\mathcal{X}_n]$)
shows *is-poly-tuple n* (*fs*@[f])
⟨*proof*⟩

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*⟩

lemma *poly-tuple-evalE$'$*:
shows *length* (*poly-tuple-eval fs as*) = *length fs*
⟨*proof*⟩

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*)
⟨*proof*⟩

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*⟩

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*)
⟨*proof*⟩

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)
⟨*proof*⟩

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*
⟨*proof*⟩

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*

$\langle proof \rangle$

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 $\equiv$ *map* (*pvar R*) (*index-list n*)

lemma *pvar-list-elements*:
  assumes $i < n$
  shows *pvar-list R n* ! i = *pvar R i*
  $\langle proof \rangle$

lemma *pvar-list-length*:
length (*pvar-list R n*) = n
  $\langle proof \rangle$

context *cring-coord-rings*
begin

lemma *pvar-list-is-poly-tuple*:
  shows *is-poly-tuple n* (*pvar-list R n*)
  $\langle proof \rangle$

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*)
  $\langle proof \rangle$

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)
  $\langle proof \rangle$

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*)
  $\langle proof \rangle$

lemma *pushforward-by-pvar-list*:
  assumes $as \in \ carrier\ (R^n)$
  shows $poly\text{-}tuple\text{-}eval\ (pvar\text{-}list\ R\ n)\ as = as$
  ⟨*proof*⟩

lemma *pushforward-by-permuted-pvar-list*:
  assumes $\sigma\ permutes\ \{..< n\}$
  assumes $as \in \ carrier\ (R^n)$
  shows $poly\text{-}tuple\text{-}eval\ (permute\text{-}list\ \sigma\ (pvar\text{-}list\ R\ n))\ as = permute\text{-}list\ \sigma\ as$
  ⟨*proof*⟩

lemma *pullback-by-permutation-of-poly-list*:
  assumes $\sigma\ permutes\ \{..< n\}$
  assumes $S \subseteq carrier\ (R^n)$
  shows $poly\text{-}tuple\text{-}pullback\ n\ S\ (permute\text{-}list\ \sigma\ (pvar\text{-}list\ R\ n)) =$
        $permute\text{-}list\ (fun\text{-}inv\ \sigma)$ ‘ S
⟨*proof*⟩

lemma *pullback-by-permutation-of-poly-list'*:
  assumes $\sigma\ permutes\ \{..< n\}$
  assumes $S \subseteq carrier\ (R^n)$
  shows $poly\text{-}tuple\text{-}pullback\ n\ S\ (permute\text{-}list\ (fun\text{-}inv\ \sigma)\ (pvar\text{-}list\ R\ n)) =$
        $permute\text{-}list\ \sigma$ ‘ S
⟨*proof*⟩

5.9 Composing Polynomial Tuples With Polynomials

composition of a multivaribale polynomial by a list of polynomials

definition *poly-compose* ::
  $nat \Rightarrow nat \Rightarrow ('a, nat)\ mvar\text{-}poly\ list \Rightarrow ('a, nat)\ mvar\text{-}poly \Rightarrow ('a, nat)\ mvar\text{-}poly$
where
$poly\text{-}compose\ n\ m\ fs = indexed\text{-}poly\text{-}induced\text{-}morphism\ \{..<n\}\ (coord\text{-}ring\ R\ m)\ (\lambda\ s.\ R.indexed\text{-}const\ s)\ (\lambda i.\ fs!i)$

lemma *poly-compose-var*:
  assumes $is\text{-}poly\text{-}tuple\ m\ fs$
  assumes $length\ fs = n$
  assumes $j < n$
  shows $poly\text{-}compose\ n\ m\ fs\ (pvar\ R\ j) = (fs!j)$
⟨*proof*⟩

lemma *Pring-universal-prop-assms*:
  assumes $is\text{-}poly\text{-}tuple\ m\ fs$
  assumes $length\ fs = n$
  shows $(\lambda i.\ fs!i) \in \{..<n\} \rightarrow carrier\ (coord\text{-}ring\ R\ m)$
        $ring\text{-}hom\text{-}ring\ R\ (coord\text{-}ring\ R\ m)\ coord\text{-}const$
⟨*proof*⟩

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*))
⟨*proof*⟩

lemma *poly-compose-closed*:
assumes *is-poly-tuple m fs*
assumes *length fs* = *n*
assumes $f \in$ *carrier* ($R[\mathcal{X}_n]$)
shows (*poly-compose n m fs f*) $\in$ *carrier* (*coord-ring R m*)
⟨*proof*⟩

lemma *poly-compose-add*:
assumes *is-poly-tuple m fs*
assumes *length fs* = *n*
assumes $f \in$ *carrier* ($R[\mathcal{X}_n]$)
assumes $g \in$ *carrier* ($R[\mathcal{X}_n]$)
shows *poly-compose n m fs* ($f \oplus_{R[\mathcal{X}_n]} g$) = (*poly-compose n m fs f*) $\oplus_{coord\text{-}ring\ R\ m}$ (*poly-compose n m fs g*)
⟨*proof*⟩

lemma *poly-compose-mult*:
assumes *is-poly-tuple m fs*
assumes *length fs* = *n*
assumes $f \in$ *carrier* ($R[\mathcal{X}_n]$)
assumes $g \in$ *carrier* ($R[\mathcal{X}_n]$)
shows *poly-compose n m fs* ($f \otimes_{R[\mathcal{X}_n]} g$) = (*poly-compose n m fs f*) $\otimes_{coord\text{-}ring\ R\ m}$ (*poly-compose n m fs g*)
⟨*proof*⟩

lemma *poly-compose-indexed-pmult*:
assumes *is-poly-tuple m fs*
assumes *length fs* = *n*
assumes $f \in$ *carrier* ($R[\mathcal{X}_n]$)
assumes $i < n$
shows *poly-compose n m fs* ($f \bigotimes i$) = (*poly-compose n m fs f*) $\otimes_{coord\text{-}ring\ R\ m}$ (*fs*!*i*)
⟨*proof*⟩

lemma *poly-compose-const*:
assumes *is-poly-tuple m fs*
assumes *length fs* = *n*
assumes $a \in$ *carrier R*
shows *poly-compose n m fs* (*coord-const a*) = *coord-const a*
⟨*proof*⟩

evaluating polynomial compositions

lemma *poly-compose-eval*:
assumes *is-poly-tuple m fs*

assumes *length fs* = *n*
assumes $f \in carrier\ (R[\mathcal{X}_n])$
assumes $as \in carrier\ (R^m)$
shows *eval-at-point R* (*poly-tuple-eval fs as*) *f* = *eval-at-point R as* (*poly-compose n m fs f*)
⟨*proof*⟩

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 = ($\lambda a \in carrier\ (R^n)$. *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)
⟨*proof*⟩

lemma *poly-map-closed*:
assumes *is-poly-tuple n fs*
assumes *length fs* = *m*
assumes $as \in carrier\ (R^n)$
shows *poly-map n fs as* ∈ *carrier* (R^m)
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*)
⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

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*)
⟨*proof*⟩

lemma *poly-ring-car-mono'*[*simp*]:
shows *carrier* ($R[\mathcal{X}_n]$) ⊆ *carrier* ($R[\mathcal{X}_{Suc\ n}]$)
carrier ($R[\mathcal{X}_n]$) ⊆ *carrier* ($R[\mathcal{X}_{n+m}]$)
⟨*proof*⟩

lemma *poly-ring-add-mono*:
assumes $n \leq m$
assumes A ∈ *carrier* ($R[\mathcal{X}_n]$)

assumes $B \in$ *carrier* $(R[\mathcal{X}_n])$
shows $A \oplus_{R[\mathcal{X}_n]} B = A \oplus_{\textit{coord-ring } R\ m} B$
⟨*proof*⟩

lemma *poly-ring-add-mono'*:
assumes $A \in$ *carrier* $(R[\mathcal{X}_n])$
assumes $B \in$ *carrier* $(R[\mathcal{X}_n])$
shows $A \oplus_{R[\mathcal{X}_n]} B = A \oplus_{R[\mathcal{X}_{Suc\ n}]} B$
$A \oplus_{R[\mathcal{X}_n]} B = A \oplus_{R[\mathcal{X}_{n+m}]} B$
⟨*proof*⟩

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 $A \otimes_{R[\mathcal{X}_n]} B = A \otimes_{\textit{coord-ring } R\ m} B$
⟨*proof*⟩

lemma *poly-ring-times-mono'*:
assumes $A \in$ *carrier* $(R[\mathcal{X}_n])$
assumes $B \in$ *carrier* $(R[\mathcal{X}_n])$
shows $A \otimes_{R[\mathcal{X}_n]} B = A \otimes_{R[\mathcal{X}_{Suc\ n}]} B$
$A \otimes_{R[\mathcal{X}_n]} B = A \otimes_{R[\mathcal{X}_{n+m}]} B$
⟨*proof*⟩

lemma *poly-ring-one-mono*:
assumes $n \leq m$
shows $\mathbf{1}_{R[\mathcal{X}_n]} = \mathbf{1}_{\textit{coord-ring } R\ m}$
⟨*proof*⟩

lemma *poly-ring-zero-mono*:
assumes $n \leq m$
shows $\mathbf{0}_{R[\mathcal{X}_n]} = \mathbf{0}_{\textit{coord-ring } R\ m}$
⟨*proof*⟩

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* $(\lambda i.\ \textit{pvar}\ R\ (i + m))\ p$

lemma *shift-vars-char*:
shows (*ring-hom-ring* $(R[\mathcal{X}_n])$ $(R[\mathcal{X}_{n+m}])$ (*shift-vars n m*))
$(\forall i \in \{..<n\}.\ (\textit{shift-vars } n\ m)\ (\textit{pvar } R\ i) = \textit{pvar } R\ (i + m))$
$(\forall a \in \textit{carrier } R.\ (\textit{shift-vars } n\ m)\ (R.\textit{indexed-const } a) = (\textit{coord-const } a))$
⟨*proof*⟩

lemma *shift-vars-constant*:

assumes $a \in$ *carrier* R
shows *shift-vars* n m (*coord-const* a) = *coord-const* a
⟨*proof*⟩

lemma *shift-vars-pvar*:
assumes $i \in \{..<n\}$
shows *shift-vars* n m (*pvar* R i) = *pvar* R ($i + m$)
⟨*proof*⟩

lemma *shift-vars-add*:
assumes $p \in$ *carrier* ($R[\mathcal{X}_n]$)
assumes $Q \in$ *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
⟨*proof*⟩

lemma *shift-vars-mult*:
assumes $p \in$ *carrier* ($R[\mathcal{X}_n]$)
assumes $Q \in$ *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
⟨*proof*⟩

lemma *shift-vars-indexed-pmult*:
assumes $p \in$ *carrier* ($R[\mathcal{X}_n]$)
assumes $i \in \{..<n\}$
shows *shift-vars* n m ($p \bigotimes i$) = (*shift-vars* n m p) $\otimes_{R[\mathcal{X}_{n+m}]}$ (*pvar* R ($i + m$))
⟨*proof*⟩

lemma *shift-vars-closed*:
assumes $p \in$ *carrier* ($R[\mathcal{X}_n]$)
shows *shift-vars* n m $p \in$ *carrier* ($R[\mathcal{X}_{n+m}]$)
⟨*proof*⟩

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
⟨*proof*⟩

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
⟨*proof*⟩

Evaluating polynomials at points in higher powers:

lemma *eval-at-points-higher-pow*:
  assumes $p \in carrier\ (R[\mathcal{X}_n])$
  assumes $k \geq n$
  assumes $a \in carrier\ (R^k)$
  shows $eval\text{-}at\text{-}point\ R\ a\ p = eval\text{-}at\text{-}point\ R\ (take\ n\ a)\ p$
  ⟨*proof*⟩

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* :: $nat \Rightarrow 'a\ list\ set$ **where**
$diagonal\ m = \{x \in carrier\ (R^{m+m}).\ take\ m\ x = drop\ m\ x\}$

lemma *diagonalE*:
  assumes $x \in diagonal\ \ m$
  shows $x = (take\ m\ x)@(take\ m\ x)$
    $x \in carrier\ (R^{m+m})$
    $take\ m\ x \in carrier\ (R^m)$
    $\bigwedge i.\ i < m \Longrightarrow x!i = x!(i + m)$
  ⟨*proof*⟩

lemma *diagonalI*:
  assumes $x = (take\ m\ x)@(take\ m\ x)$
  assumes $take\ m\ x \in carrier\ (R^m)$
  shows $x \in diagonal\ m$
  ⟨*proof*⟩

definition *diag-def-poly* :: $nat \Rightarrow nat \Rightarrow ('a,\ nat)\ mvar\text{-}poly$ **where**
$diag\text{-}def\text{-}poly\ n\ i = pvar\ R\ i \ominus_{coord\text{-}ring\ R\ (n\ +\ n)} pvar\ R\ (i + n)$

lemma *diag-def-poly-closed*:
  assumes $i < n$
  shows $diag\text{-}def\text{-}poly\ n\ i \in carrier\ (coord\text{-}ring\ R\ (n + n))$
  ⟨*proof*⟩

lemma *diag-def-poly-eval*:
  assumes $i < n$
  assumes $x \in carrier\ (R^{n+n})$
  shows $eval\text{-}at\text{-}point\ R\ x\ (diag\text{-}def\text{-}poly\ n\ i)\ \ = (x!i) \ominus (x!(i + n))$

  ⟨*proof*⟩

definition *diag-def-poly-set* :: $nat \Rightarrow ('a,\ nat)\ mvar\text{-}poly\ set$ **where**
$diag\text{-}def\text{-}poly\text{-}set\ n = diag\text{-}def\text{-}poly\ n\ `\ \{..<n\}$

lemma *diag-def-poly-set-in-coord-ring*:
  shows *diag-def-poly-set* $n \subseteq$ *carrier* (*coord-ring* R ($n + n$))
⟨*proof*⟩

lemma *diag-def-poly-set-finite*:
finite (*diag-def-poly-set* n)
  ⟨*proof*⟩

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) = **0**
⟨*proof*⟩

lemma *diagonal-as-affine-alg-set*:
  shows *diagonal* n = *affine-alg-set* R ($n + n$) (*diag-def-poly-set* n)
⟨*proof*⟩

lemma *diagonal-is-algebraic*:
  shows *is-algebraic* R ($n + n$) (*diagonal* n)
  ⟨*proof*⟩

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*
  ⟨*proof*⟩

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*)
  ⟨*proof*⟩

lemma *is-function-tuple-Cons*:
  assumes *is-function-tuple* R n *fs*
  assumes $f \in$ *carrier* (R^n) → *carrier* R
  shows *is-function-tuple* R n (f#*fs*)
  ⟨*proof*⟩

lemma *is-function-tuple-snoc*:

assumes *is-function-tuple R n fs*
assumes $f \in$ *carrier* $(R^n) \to$ *carrier R*
shows *is-function-tuple R n* $(fs@[f])$
⟨*proof*⟩

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* $(fs[i := f])$
⟨*proof*⟩

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* $(R^{length\ fs})$
⟨*proof*⟩

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*⟩

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*
⟨*proof*⟩

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* @ $[(\lambda(x::'a\ list).\ x!\ n)]$

lemma *id-function-tuple-is-function-tuple*:
$\bigwedge k.\ k \geq n \Longrightarrow$ *is-function-tuple R k* (*id-function-tuple R n*)
⟨*proof*⟩

lemma *id-function-tuple-is-function-tuple'*:
is-function-tuple R n (*id-function-tuple R n*)
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *id-function-tuple-eval-is-id*:
assumes $a \in$ *carrier* (R^n)
shows *function-tuple-eval R n* (*id-function-tuple R n*) $a = a$
⟨*proof*⟩

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 $f \in$ *carrier* (*coord-ring R* (*length fs*))
shows *poly-function-tuple-comp R n fs f* ∈ *carrier* (R^n) → *carrier R*
⟨*proof*⟩

lemma *poly-function-tuple-comp-eq*:
assumes *is-function-tuple R n fs*
assumes $f \in$ *carrier* (*coord-ring R* (*length fs*))
assumes $a \in$ *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*)
⟨*proof*⟩

lemma *poly-function-tuple-comp-constant*:
assumes *is-function-tuple R n fs*
assumes $a \in$ *carrier R*
assumes $x \in$ *carrier* (R^n)
shows *poly-function-tuple-comp R n fs* (*coord-const a*) $x = a$
⟨*proof*⟩

lemma *poly-function-tuple-comp-add*:
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* $(p \oplus_{R[\mathcal{X}_n]} Q)$ $x =$
(*poly-function-tuple-comp R n fs p x*) $\oplus$ (*poly-function-tuple-comp R n fs Q x*)
⟨*proof*⟩

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*⟩

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*⟩

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 $a \otimes_{(R[\mathcal{X}_n])} b[\urcorner]_{(R[\mathcal{X}_n])}(n{::}nat) \in$ *carrier* $(R[\mathcal{X}_n])$
⟨*proof*⟩

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 $c \oplus_{(R[\mathcal{X}_n])} a \otimes_{(R[\mathcal{X}_n])} b[\urcorner]_{(R[\mathcal{X}_n])}(n{::}nat) \in$ *carrier* $(R[\mathcal{X}_n])$
⟨*proof*⟩

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* ⇒ *nat* ⇒ ('a, *nat*) *mvar-poly*
where
generic-poly-lt R n = (*pvar R* (*Suc n*)) $\otimes_{\textit{coord-ring } R\ (\textit{Suc}\ (\textit{Suc}\ n))}$ (*pvar R 0*)$[\char`^]_{\textit{coord-ring } R\ (\textit{Suc}\ (\textit{Suc}\ n))}$ *n*

fun *generic-poly* **where**
generic-poly R (*0*::*nat*) = *pvar R 1*|
generic-poly R (*Suc n*) = (*generic-poly R n*) $\oplus_{(\textit{coord-ring } R\ (n+3))}$ *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*⟩

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*⟩

lemma *generic-poly-closed*:
generic-poly R n ∈ *carrier* (*coord-ring R* (*Suc* (*Suc n*)))
⟨*proof*⟩

lemma *generic-poly-closed'*:
assumes *k* ≤*n*
shows *generic-poly R k* ∈ *carrier* (*coord-ring R* (*Suc* (*Suc n*)))
⟨*proof*⟩

lemma *generic-poly-eval-at-point*:
assumes *a* ∈ *carrier* (R^{n+3})
shows *eval-at-point R a* (*generic-poly R* (*Suc n*)) = (*eval-at-point R a* (*generic-poly R n*)) ⊕

$(a!(n + 2)) \otimes (a!0)[^\frown](Suc\ n)$
⟨*proof*⟩

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*} ($\mathbf{0}_R$#*as*) (*generic-poly R n*)

context *cring-coord-rings*
begin

lemma *ring-cfs-to-poly-closed*:
 assumes *as* ∈ *carrier* ($R^{Suc\ n}$)
 shows *ring-cfs-to-poly R n as* ∈ *carrier* (*coord-ring R 1*)
⟨*proof*⟩

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*)
 ⟨*proof*⟩

lemma *ring-cfs-to-poly-eq*:
 assumes *as* ∈ *carrier* ($R^{Suc\ n}$)
 assumes $k \leq n$
 shows *ring-cfs-to-poly R k as* = *ring-cfs-to-poly R k* (*take* (*Suc k*) *as*)
 ⟨*proof*⟩

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*)$[^\frown]_{coord\text{-}ring\ R\ (n+2)}$ *n*)
⟨*proof*⟩

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*)$[^\frown]_{coord\text{-}ring\ R\ 1}$ *n*)
⟨*proof*⟩

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* $\oplus_{coord\text{-}ring\ R\ 1}$
poly-scalar-mult R (*as*!(*Suc n*)) ((*pvar R 0*)$[\widehat{\ }]_{coord\text{-}ring\ R\ 1}$ (*Suc n*))
⟨*proof*⟩

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*)$[\widehat{\ }]_{coord\text{-}ring\ R\ 1}$ (*Suc n*))
⟨*proof*⟩

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* $[\widehat{\ }]_{UP\ R}$ (*Suc n*))
⟨*proof*⟩

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*)$[\widehat{\ }]_{UP\ R}$ (*Suc k*)
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

lemma(**in** *UP-cring*) *monom-plus-lower-degree-top-coeff*:
assumes *degree p* < *n*
assumes *p* ∈ *carrier* (*UP R*)
assumes *a* ∈ *carrier R*
shows (*p* $\oplus_{UP\ R}$ (*a* $\odot_{UP\ R}$ (*X-poly R*)$[\widehat{\ }]_{UP\ R}$ *n*)) *n* = *a*
⟨*proof*⟩

lemma(**in** *UP-cring*) *monom-closed*:
  assumes $a \in carrier\ R$
  shows $a \odot_{UP\ R} ((X\text{-}poly\ R)[\urcorner]_{UP\ R}\ (n::nat)) \in carrier\ (UP\ R)$
  ⟨*proof*⟩

lemma(**in** *UP-cring*) *monom-bottom-coeff*:
  assumes $a \in carrier\ R$
  assumes $n > 0$
  shows $(a \odot_{UP\ R} ((X\text{-}poly\ R)[\urcorner]_{UP\ R}\ (n::nat)))\ 0 = \mathbf{0}$
  ⟨*proof*⟩

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 $(p \oplus_{UP\ R} (a \odot_{UP\ R} (X\text{-}poly\ R)[\urcorner]_{UP\ R}\ (n::nat)))\ 0 = p\ 0$
⟨*proof*⟩

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*⟩

lemma *ring-cfs-to-univ-poly-chain*:
  assumes $as \in carrier\ (R^{Suc\ n})$
  assumes $l \leq n$
  shows $l \leq k \wedge k \leq n \implies (ring\text{-}cfs\text{-}to\text{-}univ\text{-}poly\ k\ (take\ (Suc\ k)\ as))\ l = (ring\text{-}cfs\text{-}to\text{-}univ\text{-}poly\ l\ (take\ (Suc\ l)\ as))\ l$
  ⟨*proof*⟩

lemma *ring-cfs-to-univ-poly-coeffs*:
  assumes $as \in carrier\ (R^{Suc\ n})$
  assumes $l \leq n$
  shows $(ring\text{-}cfs\text{-}to\text{-}univ\text{-}poly\ n\ as)\ l = (ring\text{-}cfs\text{-}to\text{-}univ\text{-}poly\ l\ (take\ (Suc\ l)\ as))\ l$
⟨*proof*⟩

lemma *ring-cfs-to-univ-poly-coeffs'*:
  assumes $as \in carrier\ (R^{Suc\ n})$
  assumes $l \leq n$
  shows $(ring\text{-}cfs\text{-}to\text{-}univ\text{-}poly\ n\ as)\ l = as!\ l$
⟨*proof*⟩

lemma *ring-cfs-to-univ-poly-coeffs''*:
  assumes $as \in carrier\ (R^{Suc\ n})$
  shows $(ring\text{-}cfs\text{-}to\text{-}univ\text{-}poly\ n\ as)\ l = (if\ l \leq n\ then\ as!\ l\ else\ \mathbf{0})$
  ⟨*proof*⟩
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*)
 ⟨*proof*⟩

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*) $\leq m$
 ⟨*proof*⟩

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*
 ⟨*proof*⟩

6.4 Factoring a Polynomial as a Univariate Polynomial over a Multivariable Polynomial Ring

definition *pre-to-univ-poly-hom* :: *nat* ⇒ *nat* ⇒ ('*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*)
 ⟨*proof*⟩

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*

$\mathbf{0}_{Pring\ (coord\text{-}ring\ R\ (n-1))\ \{i\}})))$

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*⟩

lemma *pre-to-univ-poly-var-ass-closed'*:
assumes $i < n$
shows (*pre-to-univ-poly-var-ass* n i) $\in \{..<n\} \to$ *carrier* (*Pring* (*coord-ring* R ($n-1$)) $\{i\}$)
⟨*proof*⟩

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 $p \in$ *carrier* (*Pring* (*coord-ring* R ($n-1$)) $\{i\}$)
⟨*proof*⟩

lemma *insert-at-index-closed*:
assumes $a \in$ *carrier* (R^n)
assumes $x \in$ *carrier* R
assumes $i \leq n$
shows *insert-at-index* a x $i \in$ *carrier* ($R^{Suc\ n}$)
⟨*proof*⟩

lemma *pre-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-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*))
⟨*proof*⟩

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*⟩

lemma *pre-to-univ-poly-inv-hom-const*:
assumes *i* < *Suc n*
assumes *k* ∈ *carrier R*
shows (*pre-to-univ-poly-inv-hom* (*Suc n*) *i*) (*R.indexed-const k*) = *R.indexed-const k*
⟨*proof*⟩

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
⟨*proof*⟩

lemma *pre-to-univ-poly-inv-hom-pvar-1*:
assumes *i* < *Suc n*
assumes *i* ≤ *j*
assumes *j* < *n*
shows *pre-to-univ-poly-inv-hom* (*Suc n*) *i* (*pvar R j*) =
pvar R (*j* + *1*)
⟨*proof*⟩

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* ($R[\mathcal{X}_{Suc\ n}]$)
⟨*proof*⟩

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*} ($R[\mathcal{X}_n]$)

$(pre\text{-}to\text{-}univ\text{-}poly\text{-}inv\text{-}hom\ n\ i)$ $(pre\text{-}to\text{-}univ\text{-}poly\text{-}inv\text{-}var\text{-}ass$
$n\ i)$

lemma *pre-to-univ-poly-inv-is-hom*:
assumes $i < Suc\ n$
shows $ring\text{-}hom\text{-}ring\ (Pring\ (R[\mathcal{X}_n])\ \{i\})\ (R[\mathcal{X}_{Suc\ n}])\ (pre\text{-}to\text{-}univ\text{-}poly\text{-}inv\ (Suc$
$n)\ i)$
⟨*proof*⟩

lemma *pre-to-univ-poly-inv-pvar*:
assumes $i < Suc\ n$
shows $(pre\text{-}to\text{-}univ\text{-}poly\text{-}inv\ (Suc\ n)\ i)\ (pvar\ (R[\mathcal{X}_n])\ i) = pvar\ R\ i$
⟨*proof*⟩

lemma *pre-to-univ-poly-inv-const*:
assumes $i < Suc\ n$
assumes $p \in carrier\ (R[\mathcal{X}_n])$
shows $(pre\text{-}to\text{-}univ\text{-}poly\text{-}inv\ (Suc\ n)\ i)\ (ring.indexed\text{-}const\ (R[\mathcal{X}_n])\ p) = pre\text{-}to\text{-}univ\text{-}poly\text{-}inv\text{-}hom$
$(Suc\ n)\ i\ p$
⟨*proof*⟩

lemma *pre-to-univ-poly-inverse*:
assumes $i < Suc\ n$
assumes $p \in carrier\ (R[\mathcal{X}_{Suc\ n}])$
shows $pre\text{-}to\text{-}univ\text{-}poly\text{-}inv\ (Suc\ n)\ i\ (pre\text{-}to\text{-}univ\text{-}poly\ (Suc\ n)\ i\ p) = p$
⟨*proof*⟩

lemma *coord-ring-car-induct*:
assumes $Q \in carrier\ (R[\mathcal{X}_n])$
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 $A\ Q$
⟨*proof*⟩

lemma *pre-to-univ-poly-inverse'*:
assumes $i < Suc\ n$
assumes $p \in carrier\ (R[\mathcal{X}_n])$
shows $pre\text{-}to\text{-}univ\text{-}poly\ (Suc\ n)\ i\ (pre\text{-}to\text{-}univ\text{-}poly\text{-}inv\ (Suc\ n)\ i\ (MP.indexed\text{-}const$
$n\ p)) = MP.indexed\text{-}const\ n\ p$
⟨*proof*⟩

definition $to\text{-}univ\text{-}poly :: nat \Rightarrow nat \Rightarrow$
$(('a,\ nat)\ mvar\text{-}poly\ ,\ ('a,\ nat)\ mvar\text{-}poly\ u\text{-}poly)\ ring\text{-}hom$ **where**
$to\text{-}univ\text{-}poly\ n\ i\ = IP\text{-}to\text{-}UP\ i \circ (pre\text{-}to\text{-}univ\text{-}poly\ n\ i)$

definition $from\text{-}univ\text{-}poly :: nat \Rightarrow nat \Rightarrow$

(('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 $i \leq n$
shows (*to-univ-poly* (*Suc n*) *i*) ∈ *ring-hom* ($R[\mathcal{X}_{Suc\ n}]$) (*UP* ($R[\mathcal{X}_n]$))
⟨*proof*⟩

lemma *from-univ-poly-is-hom*:
assumes $i \leq n$
shows (*from-univ-poly* (*Suc n*) *i*) ∈ *ring-hom* (*UP* ($R[\mathcal{X}_n]$)) ($R[\mathcal{X}_{Suc\ n}]$)
⟨*proof*⟩

lemma *to-univ-poly-inverse*:
assumes $i \leq n$
assumes $p \in$ *carrier* ($R[\mathcal{X}_{Suc\ n}]$)
shows *from-univ-poly* (*Suc n*) *i* (*to-univ-poly* (*Suc n*) *i p*) = *p*
⟨*proof*⟩

lemma *to-univ-poly-closed*:
assumes $i \leq n$
assumes $p \in$ *carrier* ($R[\mathcal{X}_{Suc\ n}]$)
shows *to-univ-poly* (*Suc n*) *i p* ∈ *carrier* (*UP* ($R[\mathcal{X}_n]$))
⟨*proof*⟩

lemma *to-univ-poly-add*:
assumes $i \leq n$
assumes $p \in$ *carrier* ($R[\mathcal{X}_{Suc\ n}]$)
assumes $Q \in$ *carrier* ($R[\mathcal{X}_{Suc\ n}]$)
shows *to-univ-poly* (*Suc n*) *i* ($p \oplus_{R[\mathcal{X}_{Suc\ n}]} Q$) =
to-univ-poly (*Suc n*) *i p* $\oplus_{UP\ (R[\mathcal{X}_n])}$ *to-univ-poly* (*Suc n*) *i Q*
⟨*proof*⟩

lemma *to-univ-poly-mult*:
assumes $i \leq n$
assumes $p \in$ *carrier* ($R[\mathcal{X}_{Suc\ n}]$)
assumes $Q \in$ *carrier* ($R[\mathcal{X}_{Suc\ n}]$)
shows *to-univ-poly* (*Suc n*) *i* ($p \otimes_{R[\mathcal{X}_{Suc\ n}]} Q$) =
to-univ-poly (*Suc n*) *i p* $\otimes_{UP\ (R[\mathcal{X}_n])}$ *to-univ-poly* (*Suc n*) *i Q*
⟨*proof*⟩

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* ∈ *carrier* ($R[\mathcal{X}_{Suc\ n}]$)
⟨*proof*⟩

lemma *from-univ-poly-add*:
  assumes $i \le n$
  assumes $p \in carrier\ (UP\ (R[\mathcal{X}_n]))$
  assumes $Q \in carrier\ (UP\ (R[\mathcal{X}_n]))$
  shows $from\text{-}univ\text{-}poly\ (Suc\ n)\ i\ (p \oplus_{UP\ (R[\mathcal{X}_n])} Q) =$
    $from\text{-}univ\text{-}poly\ (Suc\ n)\ i\ p \oplus_{R[\mathcal{X}_{Suc\ n}]} from\text{-}univ\text{-}poly\ (Suc\ n)\ i\ Q$
  ⟨*proof*⟩

lemma *from-univ-poly-mult*:
  assumes $i \le n$
  assumes $p \in carrier\ (UP\ (R[\mathcal{X}_n]))$
  assumes $Q \in carrier\ (UP\ (R[\mathcal{X}_n]))$
  shows $from\text{-}univ\text{-}poly\ (Suc\ n)\ i\ (p \otimes_{UP\ (R[\mathcal{X}_n])} Q) =$
    $from\text{-}univ\text{-}poly\ (Suc\ n)\ i\ p \otimes_{R[\mathcal{X}_{Suc\ n}]} from\text{-}univ\text{-}poly\ (Suc\ n)\ i\ Q$
  ⟨*proof*⟩

lemma(**in** *UP-cring*) *monom-as-mult*:
  assumes $a \in carrier\ R$
  shows $up\text{-}ring.monom\ (UP\ R)\ a\ n = to\text{-}poly\ a\ \otimes_{UP\ R} up\text{-}ring.monom\ (UP\ R)$
1 n
  ⟨*proof*⟩

lemma *cring-coord-rings-coord-ring*:
$cring\text{-}coord\text{-}rings\ (R[\mathcal{X}_n])$
  ⟨*proof*⟩

lemma *from-univ-poly-monom-inverse*:
  assumes $i < Suc\ n$
  assumes $a \in carrier\ (R[\mathcal{X}_n])$
  shows $to\text{-}univ\text{-}poly\ (Suc\ n)\ i\ (from\text{-}univ\text{-}poly\ (Suc\ n)\ i\ (up\text{-}ring.monom\ (UP\ (R$
$[\mathcal{X}_n]))\ a\ m)) = up\text{-}ring.monom\ (UP\ (R\ [\mathcal{X}_n]))\ a\ m$
⟨*proof*⟩

lemma *from-univ-poly-inverse*:
  assumes $i \le n$
  assumes $p \in carrier\ (UP\ (R[\mathcal{X}_n]))$
  shows $to\text{-}univ\text{-}poly\ (Suc\ n)\ i\ (from\text{-}univ\text{-}poly\ (Suc\ n)\ i\ p) = p$
⟨*proof*⟩

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 $eval\text{-}at\text{-}point\ R\ as\ p = eval\text{-}at\text{-}point\ R\ a\ (to\text{-}function\ (R[\mathcal{X}_n])\ (to\text{-}univ\text{-}poly$
$(Suc\ n)\ i\ p)\ (coord\text{-}const\ x))$
⟨*proof*⟩

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 $Q \in carrier\ (R[\mathcal{X}_{Suc\ n}])$
$eval\text{-}at\text{-}point\ R\ b\ Q = (x[\urcorner]N) \otimes (eval\text{-}at\text{-}point\ R\ c\ p)$
⟨*proof*⟩

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 $N = UP\text{-}ring.degree\ (R[\mathcal{X}_n])\ (to\text{-}univ\text{-}poly\ (Suc\ n)\ i\ p)$
assumes $q = (pvar\ R\ i)[\urcorner]_{R[\mathcal{X}_{Suc\ n}]}(k{::}nat) \otimes_{R[\mathcal{X}_{Suc\ n}]} Q$
shows $q \in carrier\ (R[\mathcal{X}_{Suc\ n}])$
$eval\text{-}at\text{-}point\ R\ b\ q = (x[\urcorner](N + k)) \otimes (eval\text{-}at\text{-}point\ R\ c\ p)$
⟨*proof*⟩

lemma *to-univ-poly-one-over-poly''*:
assumes *field R*
assumes $i < (Suc\ n)$
assumes $p \in carrier\ (R[\mathcal{X}_{Suc\ n}])$
assumes $N \geq UP\text{-}ring.degree\ (R[\mathcal{X}_n])\ (to\text{-}univ\text{-}poly\ (Suc\ n)\ i\ p)$
shows $\exists\ q \in carrier\ (R[\mathcal{X}_{Suc\ n}]).\ (\forall\ x \in carrier\ R - \{\mathbf{0}\}.\ (\forall\ a \in carrier\ (R^n).$
$eval\text{-}at\text{-}point\ R\ (insert\text{-}at\text{-}index\ a\ x\ i)\ q = (x[\urcorner]N) \otimes (eval\text{-}at\text{-}point\ R\ (insert\text{-}at\text{-}index\ a\ (inv\ x)\ i)\ p)))$
⟨*proof*⟩

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** ‹$^{-1}$ı› *90*) **where**
evimage $n\ f\ S = ((f -` S) \cap carrier\ (R^n))$

definition *euminus-set* :: *nat* ⇒ *'a list set* ⇒ *'a list set* (‹- ^{c}ı› *70*) **where**
$S^c{}_n = carrier\ (R^n) - S$

lemma *extensional-vimage-closed*:
$f\ ^{-1}{}_n\ S \subseteq carrier\ (R^n)$
⟨*proof*⟩

7.1 Inverse image of a function

lemma *evimage-eq* [*simp*]: $a \in f\ ^{-1}{}_n\ B \longleftrightarrow a \in carrier\ (R^n) \wedge f\ a \in B$
⟨*proof*⟩

lemma *evimage-singleton-eq*: $a \in f\ ^{-1}{}_n\ \{b\} \longleftrightarrow a \in carrier\ (R^n) \wedge\ f\ a = b$
⟨*proof*⟩

lemma *evimageI* [*intro*]: $a \in carrier\ (R^n) \Longrightarrow f\ a = b \Longrightarrow b \in B \Longrightarrow a \in f\ ^{-1}{}_n\ B$
⟨*proof*⟩

lemma *evimageI2*: $a \in carrier\ (R^n) \Longrightarrow f\ a \in A \Longrightarrow a \in f\ ^{-1}{}_n\ A$
⟨*proof*⟩

lemma *evimageE* [*elim!*]: $a \in f\ ^{-1}{}_n\ B \Longrightarrow (\bigwedge x.\ f\ a = x \Longrightarrow x \in B \Longrightarrow p) \Longrightarrow p$
⟨*proof*⟩

lemma *evimageD*: $a \in f^{-1}{}_n\ A \Longrightarrow f\ a \in A$
⟨*proof*⟩

lemma *evimage-empty* [*simp*]: $f\ ^{-1}{}_n\ \{\} = \{\}$
⟨*proof*⟩

lemma *evimage-Compl*:
assumes $f \in carrier\ (R^n) \rightarrow carrier\ (R^m)$
shows $(f\ ^{-1}{}_n(A^c{}_m)) = ((f -` A)^c{}_n)$
⟨*proof*⟩

lemma *evimage-Un* [*simp*]: $f\ ^{-1}{}_n\ (A \cup B) = (f\ ^{-1}{}_n\ A) \cup (f\ ^{-1}{}_n\ B)$

$\langle proof \rangle$

lemma *evimage-Int* [*simp*]: $f\ ^{-1}{}_n\ (A \cap B) = (f\ ^{-1}{}_n\ A) \cap (f\ ^{-1}{}_n\ B)$
$\langle proof \rangle$

lemma *evimage-Collect-eq* [*simp*]: $f\ ^{-1}{}_n\ Collect\ p = \{y \in carrier\ (R^n).\ p\ (f\ y)\}$
$\langle proof \rangle$

lemma *evimage-Collect*: $(\bigwedge x.\ x \in carrier\ (R^n) \Longrightarrow p\ (f\ x) = Q\ x) \Longrightarrow f\ ^{-1}{}_n\ (Collect\ p) = Collect\ Q \cap carrier\ (R^n)$
$\langle proof \rangle$

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 $\{a\}$.
$\langle proof \rangle$

lemma *evimage-Diff*: $f\ ^{-1}{}_n\ (A - B) = (f\ ^{-1}{}_n\ A) - (f\ ^{-1}{}_n\ B)$
$\langle proof \rangle$

lemma *evimage-UNIV* [*simp*]: $f\ ^{-1}{}_n\ UNIV = carrier\ (R^n)$
$\langle proof \rangle$

lemma *evimage-mono*: $A \subseteq B \Longrightarrow f\ ^{-1}{}_n\ A \subseteq f\ ^{-1}{}_n\ B$
— monotonicity
$\langle proof \rangle$

lemma *evimage-image-eq*: $(f\ ^{-1}{}_n\ (f\ `\ A)) = \{y \in carrier\ (R^n).\ \exists x{\in}A.\ f\ x = f\ y\}$
$\langle proof \rangle$

lemma *image-evimage-subset*: $f\ `\ (f\ ^{-1}{}_n\ A) \subseteq A$
$\langle proof \rangle$

lemma *image-evimage-eq* [*simp*]: $f\ `\ (f\ ^{-1}{}_n\ A) = A \cap (f\ `\ carrier\ (R^n))$
$\langle proof \rangle$

lemma *image-subset-iff-subset-evimage*: $A \subseteq carrier\ (R^n) \Longrightarrow f\ `\ A \subseteq B \longleftrightarrow A \subseteq f\ ^{-1}{}_n\ B$
$\langle proof \rangle$

lemma *evimage-const* [*simp*]: $((\lambda x.\ c)\ ^{-1}{}_n\ A) = (if\ c \in A\ then\ carrier\ (R^n)\ else\ \{\})$
$\langle proof \rangle$

lemma *evimage-if* [*simp*]: $((\lambda x.\ if\ x \in B\ then\ c\ else\ d)\ ^{-1}{}_n\ A) =$
$(if\ c \in A\ then\ (if\ d \in A\ then\ carrier\ (R^n)\ else\ B \cap carrier\ (R^n)\ )$
$else\ if\ d \in A\ then\ B^c{}_n\ \ else\ \{\})$
$\langle proof \rangle$

lemma *evimage-inter-cong*: $(\bigwedge\ w.\ w \in S \Longrightarrow f\ w = g\ w) \Longrightarrow f\ ^{-1}{}_n\ y \cap S = g$

$^{-1}{}_n\ y \cap S$
⟨*proof*⟩

lemma *evimage-ident* [*simp*]: $(\lambda x.\ x)\ ^{-1}{}_n\ Y = Y \cap carrier\ (R^n)$
⟨*proof*⟩

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* ⇒ *int*) × (*nat* ⇒ *int*)) *set*
locale *padic-fields*=
fixes Q_p:: - *ring* (**structure**)
fixes Z_p:: - *ring* (**structure**)
fixes *p*
fixes ι
defines Z_p ≡ *padic-int p*
defines Q_p ≡ *Frac* Z_p
defines ι ≡ *domain-frac.inc* Z_p
assumes *prime*: *prime p*

sublocale *padic-fields* < *Zp?*: *domain-frac* Z_p
⟨*proof*⟩

sublocale *padic-fields* < *Qp?*: *ring* Q_p
⟨*proof*⟩

sublocale *padic-fields* < *Qp?*: *cring* Q_p

⟨*proof*⟩

sublocale *padic-fields* < *Qp?*: *field* Q_p
⟨*proof*⟩

sublocale *padic-fields* < *Qp?*: *domain* Q_p
⟨*proof*⟩

sublocale *padic-fields* < *padic-integers* Z_p
⟨*proof*⟩

sublocale *padic-fields* < *UPQ?*: *UP-cring* Q_p *UP* Q_p
⟨*proof*⟩

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
⟨*proof*⟩

lemma *inc-is-hom*:
$\iota \in$ *ring-hom* $Z_p\ Q_p$
⟨*proof*⟩

An alternate formula of the map ι

lemma *inc-def*:
assumes $a \in$ *carrier* Z_p
shows $\iota\ a =$ *frac* $a\ \mathbf{1}_{Z_p}$
⟨*proof*⟩

lemma *inc-of-nonzero*:
assumes $a \in$ *nonzero* Z_p
shows $\iota\ a \in$ *nonzero* Q_p
⟨*proof*⟩

lemma *inc-of-one*:
$\iota\ \mathbf{1}_{Z_p} = \mathbf{1}$
⟨*proof*⟩

lemma *inc-of-zero*:
$\iota\ \mathbf{0}_{Z_p} = \mathbf{0}$
⟨*proof*⟩

lemma *inc-of-sum*:
assumes $a \in carrier\ Z_p$
assumes $b \in carrier\ Z_p$
shows $\iota\ (a \oplus_{Z_p} b) = (\iota\ a) \oplus (\iota\ b)$
⟨*proof*⟩

lemma *inc-of-prod*:
assumes $a \in carrier\ Z_p$
assumes $b \in carrier\ Z_p$
shows $\iota\ (a \otimes_{Z_p} b) = (\iota\ a) \otimes (\iota\ b)$
⟨*proof*⟩

lemma *inc-pow*:
assumes $a \in nonzero\ Z_p$
shows $\iota\ (a[\widehat{\ }]_{Z_p}(n::nat)) = (\iota\ a)[\widehat{\ }]\ n$
⟨*proof*⟩

lemma *inc-of-diff*:
assumes $a \in carrier\ Z_p$
assumes $b \in carrier\ Z_p$
shows $\iota\ (a \ominus_{Z_p} b) = (\iota\ a) \ominus (\iota\ b)$
⟨*proof*⟩

lemma *Units-nonzero-Qp*:
assumes $u \in Units\ Q_p$
shows $u \in nonzero\ Q_p$
⟨*proof*⟩

lemma *Units-eq-nonzero*:
$Units\ Q_p = nonzero\ Q_p$
⟨*proof*⟩

lemma *Units-inverse-Qp*:
assumes $u \in Units\ Q_p$
shows $inv_{Q_p}\ u \in Units\ Q_p$
⟨*proof*⟩

lemma *nonzero-inverse-Qp*:
assumes $u \in nonzero\ Q_p$
shows $inv_{Q_p}\ u \in nonzero\ Q_p$
⟨*proof*⟩

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 $(frac\ a\ b) \oplus (frac\ c\ d) = (frac\ ((a \otimes_{Z_p} d) \oplus_{Z_p} (b \otimes_{Z_p} c))\ (b \otimes_{Z_p} d))$
⟨*proof*⟩

lemma *frac-add-common-denom*:
assumes $a \in$ *carrier* Z_p
assumes $b \in$ *carrier* Z_p
assumes $c \in$ *nonzero* Z_p
shows $(frac\ a\ c) \oplus (frac\ b\ c) = frac\ (a \oplus_{Z_p} b)\ c$
⟨*proof*⟩

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 $(frac\ a\ b) \otimes (frac\ c\ d) = (frac\ (a \otimes_{Z_p} c)\ (b \otimes_{Z_p} d))$
⟨*proof*⟩

lemma *frac-one*:
assumes $a \in$ *nonzero* Z_p
shows *frac* $a\ a = \mathbf{1}$
⟨*proof*⟩

lemma *frac-closed*:
assumes $a \in$ *carrier* Z_p
assumes $b \in$ *nonzero* Z_p
shows *frac* $a\ b \in$ *carrier* Q_p
⟨*proof*⟩

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
⟨*proof*⟩

lemma *nonzero-numer-imp-nonzero-fraction*:
assumes $a \in$ *nonzero* Z_p
assumes $b \in$ *nonzero* Z_p
shows *frac* $a\ b \neq \mathbf{0}$
⟨*proof*⟩

lemma *nonzero-fraction-imp-numer-not-zero*:
assumes $a \in$ *carrier* Z_p
assumes $b \in$ *nonzero* Z_p

assumes *frac a b* $\neq$ **0**
shows $a \neq \mathbf{0}_{Z_p}$
$\langle proof \rangle$

lemma *nonzero-fraction-imp-nonzero-numer*:
assumes $a \in$ *carrier* Z_p
assumes $b \in$ *nonzero* Z_p
assumes *frac a b* $\neq$ **0**
shows $a \in$ *nonzero* Z_p
$\langle proof \rangle$

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*
$\langle proof \rangle$

lemma(**in** *padic-fields*) *frac-uminus*:
assumes $a \in$ *carrier* Z_p
assumes $b \in$ *nonzero* Z_p
shows $\ominus$ (*frac a b*) = *frac* $(\ominus_{Z_p}\ a)\ b$
$\langle proof \rangle$

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* $(a \otimes_{Z_p} c)\ d$
$\langle proof \rangle$

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*)) = ι (*numer a*)
a = *frac* (*numer a*) (*denom a*)
$\langle proof \rangle$

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

$x = frac\ a\ c$
$y = frac\ b\ c$
⟨*proof*⟩

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 $(frac\ a\ b) = (\iota\ a \div \iota\ b)$
⟨*proof*⟩

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*⟩

lemma *fract-cancel-right*:
assumes $a \in carrier\ Q_p$
assumes $b \in nonzero\ Q_p$
shows $b \otimes (a \div b) = a$
⟨*proof*⟩

lemma *fract-cancel-left*:
assumes $a \in carrier\ Q_p$
assumes $b \in nonzero\ Q_p$
shows $(a \div b) \otimes b = a$
⟨*proof*⟩

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 $(a \div b) \otimes (c \div d) = ((a \otimes c) \div (b \otimes d))$
⟨*proof*⟩

lemma *Qp-nat-pow-nonzero*:
assumes $x \in nonzero\ Q_p$
shows $x[\widehat{\ }](n::nat) \in nonzero\ Q_p$
⟨*proof*⟩

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
⟨*proof*⟩

lemma *Qp-int-pow-nonzero*:
assumes $x \in$ *nonzero* Q_p
shows $x[\hat{}](n::int) \in$ *nonzero* Q_p
⟨*proof*⟩

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
⟨*proof*⟩

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*⟩

lemma *pow-p-frac*:
assumes $(m::int) \leq n$
assumes $m \geq 0$
shows $(frac\ (\mathrm{p}[\hat{}]_{Z_p} m)\ (\mathrm{p}[\hat{}]_{Z_p} n)) = frac\ \mathbf{1}_{Z_p}\ (\mathrm{p}[\hat{}]_{Z_p}(n-m))$
⟨*proof*⟩

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*:
$Frac\text{-}inc\ R\ ([(n::nat)] \cdot \mathbf{1}) = [n] \cdot_{Frac\ R} \mathbf{1}_{Frac\ R}$
⟨*proof*⟩

lemma *inc-of-nat*:
$(\iota\ ([(n::nat)] \cdot_{Z_p} \mathbf{1}_{Z_p})) = [n] \cdot_{Q_p} \mathbf{1}$
⟨*proof*⟩

lemma(**in** *domain-frac*) *frac-inc-of-int*:
$Frac\text{-}inc\ R\ ([(n::int)] \cdot \mathbf{1}) = [n] \cdot_{Frac\ R} \mathbf{1}_{Frac\ R}$
⟨*proof*⟩

lemma *inc-of-int*:
$(\iota\ ([(n::int)] \cdot_{Z_p} \mathbf{1}_{Z_p})) = [n] \cdot_{Q_p} \mathbf{1}$
⟨*proof*⟩

lemma *p-inc*:

$\mathfrak{p} = \iota$ p
  ⟨*proof*⟩

lemma *p-nonzero*:
$\mathfrak{p} \in$ *nonzero* Q_p
  ⟨*proof*⟩

lemma *p-natpow-inc*:
  fixes *n*::*nat*
  shows $\mathfrak{p}[\urcorner]n = \iota$ (p $[\urcorner]_{Z_p}$ *n*)
  ⟨*proof*⟩

lemma *p-intpow-inc*:
  fixes *n*::*int*
  assumes $n \geq 0$
  shows $\mathfrak{p}[\urcorner]n = \iota$ (p $[\urcorner]_{Z_p}$ *n*)
  ⟨*proof*⟩

lemma *p-intpow*:
  fixes *n*::*int*
  assumes $n < 0$
  shows $\mathfrak{p}[\urcorner]n = (frac\ \mathbf{1}_{Z_p}$ (p $[\urcorner]_{Z_p}$ $(-n)))$
⟨*proof*⟩

lemma *p-natpow-closed*[*simp*]:
  fixes *n*::*nat*
  shows $(\mathfrak{p}[\urcorner]n) \in (carrier\ Q_p)$
        $(\mathfrak{p}[\urcorner]n) \in (nonzero\ Q_p)$
  ⟨*proof*⟩

lemma *nonzero-int-pow-distrib*:
  assumes $a \in nonzero\ Q_p$
  assumes $b \in nonzero\ Q_p$
  shows $(a \otimes b)\ [\urcorner](k::int) = a[\urcorner]k \otimes b[\urcorner]k$
⟨*proof*⟩

lemma *val-ring-subring*:
subring $\mathcal{O}_p\ Q_p$
  ⟨*proof*⟩

lemma *val-ring-closed*:
$\mathcal{O}_p \subseteq carrier\ Q_p$
  ⟨*proof*⟩

lemma *p-pow-diff*:
  fixes *n*::*int*
  fixes *m*::*int*
  assumes $n \geq 0$
  assumes $m \geq 0$

shows $\mathfrak{p}$ $[\urcorner]$ $(n - m)$ = *frac* $(\mathrm{p}[\urcorner]_{Z_p} n)$ $(\mathrm{p}[\urcorner]_{Z_p} m)$
⟨*proof*⟩

lemma *Qp-int-pow-add*:
fixes n::*int*
fixes m::*int*
assumes $a \in$ *nonzero* Q_p
shows a $[\urcorner]$ $(n + m) = (a$ $[\urcorner]$ $n) \otimes (a$ $[\urcorner]$ $m)$
⟨*proof*⟩

lemma *Qp-nat-pow-pow*:
fixes n::*nat*
fixes m::*nat*
assumes $a \in$ *carrier* Q_p
shows $(a[\urcorner](n*m)) = ((a[\urcorner]n)[\urcorner]m)$
⟨*proof*⟩

lemma *Qp-p-nat-pow-pow*:
fixes n::*nat*
fixes m::*nat*
shows $(\mathfrak{p}$ $[\urcorner]$ $(n*m)) = ((\mathfrak{p}[\urcorner]n)[\urcorner]m)$
⟨*proof*⟩

lemma *Qp-units-int-pow*:
fixes n::*int*
assumes $a \in$ *nonzero* Q_p
shows $a[\urcorner]n = a[\urcorner]_{units\text{-}of\ Q_p} n$
⟨*proof*⟩

lemma *Qp-int-pow-pow*:
fixes n::*int*
fixes m::*int*
assumes $a \in$ *nonzero* Q_p
shows $(a[\urcorner](n*m)) = ((a[\urcorner]n)[\urcorner]m)$
⟨*proof*⟩

lemma *Qp-p-int-pow-pow*:
fixes n::*int*
fixes m::*int*
shows $(\mathfrak{p}$ $[\urcorner]$ $(n*m)) = ((\mathfrak{p}[\urcorner]n)[\urcorner]m)$
⟨*proof*⟩

lemma *Qp-int-nat-pow-pow*:
fixes n::*int*
fixes m::*nat*
assumes $a \in$ *nonzero* Q_p
shows $(a[\urcorner](n*m)) = ((a[\urcorner]n)[\urcorner]m)$
⟨*proof*⟩

lemma *Qp-p-int-nat-pow-pow*:
  fixes $n::int$
  fixes $m::nat$
  shows $(\mathfrak{p}\ [\urcorner]\ (n*m)) = ((\mathfrak{p}[\urcorner]n)[\urcorner]m)$
  ⟨*proof*⟩

lemma *Qp-nat-int-pow-pow*:
  fixes $n::nat$
  fixes $m::int$
  assumes $a \in nonzero\ Q_p$
  shows $(a[\urcorner](n*m)) = ((a[\urcorner]n)[\urcorner]m)$
  ⟨*proof*⟩

lemma *Qp-p-nat-int-pow-pow*:
  fixes $n::nat$
  fixes $m::int$
  shows $(\mathfrak{p}\ [\urcorner]\ (n*m)) = ((\mathfrak{p}[\urcorner]n)[\urcorner]m)$
  ⟨*proof*⟩

lemma *p-intpow-closed*:
  fixes $n::int$
  shows $(\mathfrak{p}[\urcorner]n) \in (carrier\ Q_p)$
    $(\mathfrak{p}[\urcorner]n) \in (nonzero\ Q_p)$
  ⟨*proof*⟩

lemma *p-intpow-add*:
  fixes $n::int$
  fixes $m::int$
  shows $\mathfrak{p}\ [\urcorner]\ (n + m) = (\mathfrak{p}\ [\urcorner]\ n) \otimes (\mathfrak{p}\ [\urcorner]\ m)$
  ⟨*proof*⟩

lemma *p-intpow-inv*:
  fixes $n::int$
  shows $(\mathfrak{p}\ [\urcorner]\ n) \otimes (\mathfrak{p}\ [\urcorner]\ -n) = \mathbf{1}$
  ⟨*proof*⟩

lemma *p-intpow-inv'*:
  fixes $n::int$
  shows $(\mathfrak{p}\ [\urcorner]\ -n) \otimes (\mathfrak{p}\ [\urcorner]\ n) = \mathbf{1}$
  ⟨*proof*⟩

lemma *p-intpow-inv''*:
  fixes $n::int$
  shows $(\mathfrak{p}\ [\urcorner]\ -n) = inv_{Q_p}\ (\mathfrak{p}\ [\urcorner]\ n)$
  ⟨*proof*⟩

lemma *p-int-pow-factor-int-pow*:
  assumes $a \in nonzero\ Q_p$
  shows $(\mathfrak{p}[\urcorner](n::int) \otimes a)[\urcorner](k::int) = \mathfrak{p}[\urcorner](n*k) \otimes a[\urcorner]k$

$\langle proof \rangle$

lemma *p-nat-pow-factor-int-pow*:
assumes $a \in nonzero\ Q_p$
shows $(\mathfrak{p}[\widehat{\ }](n::nat) \otimes a)[\widehat{\ }](k::int) = \mathfrak{p}[\widehat{\ }](n*k) \otimes a[\widehat{\ }]k$
$\langle proof \rangle$

lemma *p-pow-factor*:
$\mathfrak{p}[\widehat{\ }]((int\ N)*l\ + k) = (\mathfrak{p}[\widehat{\ }]l)[\widehat{\ }](N::nat) \otimes \mathfrak{p}[\widehat{\ }]\ k$
$\langle proof \rangle$

lemma *p-nat-pow-factor-nat-pow*:
assumes $a \in carrier\ Q_p$
shows $(\mathfrak{p}[\widehat{\ }](n::nat) \otimes a)[\widehat{\ }](k::nat) = \mathfrak{p}[\widehat{\ }](n*k) \otimes a[\widehat{\ }]k$
$\langle proof \rangle$

lemma *p-int-pow-factor-nat-pow*:
assumes $a \in carrier\ Q_p$
shows $(\mathfrak{p}[\widehat{\ }](n::int) \otimes a)[\widehat{\ }](k::nat) = \mathfrak{p}[\widehat{\ }](n*k) \otimes a[\widehat{\ }]k$
$\langle proof \rangle$

lemma(**in** *ring*) *r-minus-distr*:
assumes $a \in carrier\ R$
assumes $b \in carrier\ R$
assumes $c \in carrier\ R$
shows $a \otimes b \ominus a \otimes c = a \otimes (b \ominus c)$
$\langle proof \rangle$

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**
$ord\ x = (ord\text{-}Zp\ (numer\ x)) - (ord\text{-}Zp\ (denom\ x))$

definition *val* **where**
$val\ x = (if\ x = \mathbf{0}\ then\ (\infty::eint)\ else\ eint\ (ord\ x))$

lemma *val-ord*[*simp*]:
assumes $a \in nonzero\ Q_p$
shows $val\ a = ord\ a$
$\langle proof \rangle$

8.3.2 Properties of the Valuation

lemma *ord-of-frac*:
assumes $a \in nonzero\ Z_p$

assumes $b \in$ *nonzero* Z_p
shows *ord* (*frac a b*) = (*ord-Zp a*) − (*ord-Zp b*)
⟨*proof*⟩

lemma *val-zero*:
val $\mathbf{0} = \infty$ ⟨*proof*⟩

lemma *ord-one*[*simp*]:
ord $\mathbf{1}$ = *0*
⟨*proof*⟩

lemma *val-one*[*simp*]:
val ($\mathbf{1}$) = *0*
⟨*proof*⟩

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*⟩

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*⟩

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
ι *w* = *frac u v*
⟨*proof*⟩

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*⟩

lemma *val-ring-val-criterion*:
assumes $a \in$ *carrier* Q_p
assumes *val a* $\geq$ *0*
shows $a \in \mathcal{O}_p$
⟨*proof*⟩

lemma *ord-of-inv*:
assumes $a \in$ *carrier* Q_p

assumes $a \neq \mathbf{0}$
shows $ord\ (inv_{Q_p}\ a) = -\ (ord\ a)$
⟨*proof*⟩

lemma *val-of-inv*:
assumes $a \in carrier\ Q_p$
assumes $a \neq \mathbf{0}$
shows $val\ (inv_{Q_p}\ a) = -\ (val\ a)$
⟨*proof*⟩

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*⟩

lemma *Qp-val-ringI*:
assumes $a \in carrier\ Q_p$
assumes $val\ a \geq 0$
shows $a \in \mathcal{O}_p$
⟨*proof*⟩

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*⟩

lemma *val-ineq*:
assumes $a \in carrier\ Q_p$
assumes $val\ \mathbf{0} \leq val\ a$
shows $a = \mathbf{0}$
⟨*proof*⟩

lemma *ord-minus*:
assumes $a \in\ nonzero\ Q_p$
shows $ord\ a = ord\ (\ominus a)$
⟨*proof*⟩

lemma *val-minus*:
assumes $a \in\ carrier\ Q_p$
shows $val\ a = val\ (\ominus a)$
⟨*proof*⟩

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*⟩

lemma *val-mult0*:
assumes $x \in$ *nonzero* Q_p
assumes $y \in$ *nonzero* Q_p
shows $(val\ (x \otimes y)) = (val\ x) + (val\ y)$
⟨*proof*⟩

val is multiplicative everywhere

lemma *val-mult*:
assumes $x \in$ *carrier* Q_p
assumes $y \in$ *carrier* Q_p
shows $(val\ (x \otimes y)) = (val\ x) + (val\ y)$
⟨*proof*⟩

val and ord are compatible with inclusion

lemma *ord-of-inc*:
assumes $x \in$ *nonzero* Z_p
shows *ord-Zp* $x = ord(\iota\ x)$
⟨*proof*⟩

lemma *val-of-inc*:
assumes $x \in$ *carrier* Z_p
shows *val-Zp* $x = val\ (\iota\ x)$
⟨*proof*⟩

lemma *Qp-inc-id*:
assumes $a \in$ *nonzero* Q_p
assumes $ord\ a \geq 0$
obtains b **where** $b \in$ *nonzero* Z_p **and** $a = \iota\ b$
⟨*proof*⟩

lemma *val-ring-memI*:
assumes $a \in$ *carrier* Q_p
assumes $val\ a \geq 0$
shows $a \in \mathcal{O}_p$
⟨*proof*⟩

lemma *val-ring-memE*:
assumes $a \in \mathcal{O}_p$
shows $val\ a \geq 0$ $a \in$ *carrier* Q_p
⟨*proof*⟩

lemma *val-ring-add-closed*:
assumes $a \in \mathcal{O}_p$
assumes $b \in \mathcal{O}_p$
shows $a \oplus b \in \mathcal{O}_p$

$\langle proof \rangle$

lemma *val-ring-times-closed*:
assumes $a \in \mathcal{O}_p$
assumes $b \in \mathcal{O}_p$
shows $a \otimes b \in \mathcal{O}_p$
$\langle proof \rangle$

lemma *val-ring-ainv-closed*:
assumes $a \in \mathcal{O}_p$
shows $\ominus\ a \in \mathcal{O}_p$
$\langle proof \rangle$

lemma *val-ring-minus-closed*:
assumes $a \in \mathcal{O}_p$
assumes $b \in \mathcal{O}_p$
shows $a \ominus b \in \mathcal{O}_p$
$\langle proof \rangle$

lemma *one-in-val-ring*:
$\mathbf{1} \in \mathcal{O}_p$
$\langle proof \rangle$

lemma *zero-in-val-ring*:
$\mathbf{0} \in \mathcal{O}_p$
$\langle proof \rangle$

lemma *ord-p*:
$ord\ \mathfrak{p} = \mathit{1}$
$\langle proof \rangle$

lemma *ord-p-pow-nat*:
$ord\ (\mathfrak{p}\ [\urcorner]\ (n::nat)) = n$
$\langle proof \rangle$

lemma *ord-p-pow-int*:
$ord\ (\mathfrak{p}\ [\urcorner]\ (n::int)) = n$
$\langle proof \rangle$

lemma *ord-nonneg*:
assumes $x \in\ \mathcal{O}_p$
assumes $x \neq \mathbf{0}$
shows $ord\ x \geq \mathit{0}$
$\langle proof \rangle$

lemma *val-p*:
$val\ \mathfrak{p} = \mathit{1}$
$\langle proof \rangle$

lemma *val-p-int-pow*:
val $(\mathfrak{p}[\urcorner](k::int)) = k$
⟨*proof*⟩

lemma *val-p-int-pow-neg*:
val $(\mathfrak{p}[\urcorner](-k::int)) = -$ *eint* k
⟨*proof*⟩

lemma *nonzero-nat-pow-ord*:
assumes $a \in$ *nonzero* Q_p
shows *ord* $(a\ [\urcorner]\ (n::nat)) = n *$ *ord* a
⟨*proof*⟩

lemma *add-cancel-eint-geq*:
assumes $(eint\ a) + x \geq (eint\ a\ ) + y$
shows $x \geq y$
⟨*proof*⟩

lemma(**in** *padic-fields*) *prod-equal-val-imp-equal-val*:
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
⟨*proof*⟩

lemma *two-times-eint*:
shows $2*(x::eint) = x + x$
⟨*proof*⟩

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*⟩

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*⟩

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* $(u \otimes b) + \alpha$
⟨*proof*⟩

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
⟨*proof*⟩

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* $(a \otimes c)$
⟨*proof*⟩

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
⟨*proof*⟩

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* $(a \otimes c)$
⟨*proof*⟩

lemma *finite-val-imp-nonzero*:
assumes $a \in$ *carrier* Q_p
assumes *val* $a \neq \infty$
shows $a \in$ *nonzero* Q_p
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*⟩

lemma *val-ultrametric*:
assumes $x \in carrier\ Q_p$
assumes $y \in carrier\ Q_p$
shows $min\ (val\ x)\ (val\ y) \leq val\ (x \oplus y)$
⟨*proof*⟩

lemma *val-ultrametric'*:
assumes $x \in carrier\ Q_p$
assumes $y \in carrier\ Q_p$
shows $min\ (val\ x)\ (val\ y) \leq val\ (x \ominus y)$
⟨*proof*⟩

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*⟩

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*⟩

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* $y)$
⟨*proof*⟩

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* $x)$
⟨*proof*⟩

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
⟨*proof*⟩

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
⟨*proof*⟩

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
⟨*proof*⟩

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
⟨*proof*⟩

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
⟨*proof*⟩

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
⟨*proof*⟩

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*⟩

lemma *eint-ord-trans*:
assumes $(a::eint) \leq b$
assumes $b \leq c$
shows $a \leq c$
⟨*proof*⟩

lemma *eint-Min-geq*:
assumes *finite* $(A::eint\ set)$
assumes $\bigwedge x.$ $x \in A \Longrightarrow x \geq c$
assumes $A \neq \{\}$
shows *Min* $A \geq c$
⟨*proof*⟩

lemma *eint-Min-gr*:
assumes *finite* $(A::eint\ set)$
assumes $\bigwedge x.$ $x \in A \Longrightarrow x > c$
assumes $A \neq \{\}$
shows *Min* $A > c$
⟨*proof*⟩

lemma *finsum-val-ultrametric*:
assumes $g \in A \rightarrow$ *carrier* Q_p
assumes *finite* A
assumes $A \neq \{\}$
shows *val* (*finsum* Q_p $g\ A) \geq$ *Min* (*val* ` $(g`A))$
⟨*proof*⟩

lemma (**in** *padic-fields*) *finsum-val-ultrametric'*:
assumes $g \in A \rightarrow$ *carrier* Q_p
assumes *finite* A
assumes $\bigwedge i.$ $i \in A \Longrightarrow$ *val* $(g\ i) \geq c$

shows *val* (*finsum* Q_p *g* *A*) $\geq$ *c*
⟨*proof*⟩

lemma (**in** *padic-fields*) *finsum-val-ultrametric''*:
assumes $g \in A \to$ *carrier* Q_p
assumes *finite* *A*
assumes $\bigwedge i.\ i \in A \Longrightarrow$ *val* (*g* *i*) $>$ *c*
assumes $c < \infty$
shows *val* (*finsum* Q_p *g* *A*) $>$ *c*
⟨*proof*⟩

lemma *Qp-diff-diff*:
assumes $x \in$ *carrier* Q_p
assumes $c \in$ *carrier* Q_p
assumes $d \in$ *carrier* Q_p
shows $(x \ominus c) \ominus (d \ominus c) = x \ominus d$
⟨*proof*⟩

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*⟩

More variants on the ultrametric inequality

lemma *MinE*:
assumes *finite* (*A*::*eint set*)
assumes $a = $ *Min* *A*
assumes $b \in A$
shows $a \leq b$
⟨*proof*⟩

lemma *MinE'*:
assumes *finite* (*A*::*eint set*)
assumes $a = $ *Min* *A*
assumes $b \in A - \{a\}$
shows $a < b$
⟨*proof*⟩

lemma *MinE''*:
assumes *finite* *A*
assumes $f \in A \to$ (*UNIV* :: *eint set*)
assumes $a = $ *Min* (*f* ‘ *A*)
assumes $b \in A$

shows $a \leq f\ b$
⟨*proof*⟩

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*⟩

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*⟩

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**
$angular\text{-}component\ a = (ac\text{-}Zp\ (numer\ a)) \otimes_{Z_p} (inv_{Z_p}\ ac\text{-}Zp\ (denom\ a))$

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 $angular\text{-}component\ c = (ac\text{-}Zp\ a) \otimes_{Z_p} inv\ _{Z_p}(ac\text{-}Zp\ b)$
⟨*proof*⟩

lemma *angular-component-closed*:
assumes $a \in$ *nonzero* Q_p
shows *angular-component* $a \in$ *carrier* Z_p
⟨*proof*⟩

lemma *angular-component-unit*:
assumes $a \in$ *nonzero* Q_p
shows *angular-component* $a \in$ *Units* Z_p
⟨*proof*⟩

lemma *angular-component-factors-x*:
assumes $x \in$ *nonzero* Q_p
shows $x = (\mathfrak{p}[\urcorner](ord\ x)) \otimes \iota\ (angular\text{-}component\ x)$
⟨*proof*⟩

lemma *angular-component-mult*:
assumes $x \in$ *nonzero* Q_p
assumes $y \in$ *nonzero* Q_p
shows *angular-component* $(x \otimes y) = (angular\text{-}component\ \ x) \otimes_{Z_p} (angular\text{-}component\ y)$
⟨*proof*⟩

lemma *angular-component-inv*:
assumes $x \in$ *nonzero* Q_p
shows *angular-component* $(inv_{Q_p}\ x) = inv_{Z_p}\ (angular\text{-}component\ x)$
⟨*proof*⟩

lemma *angular-component-one*:
angular-component $\mathbf{1} = \mathbf{1}_{Z_p}$
⟨*proof*⟩

lemma *angular-component-ord-zero*:
assumes *ord* $x = 0$
assumes $x \in$ *nonzero* Q_p
shows ι (*angular-component* x) $= x$
⟨*proof*⟩

lemma *angular-component-of-inclusion*:
assumes $x \in$ *nonzero* Z_p
assumes $y = \iota\ x$
shows *angular-component* $y =$ *ac-Zp* x
⟨*proof*⟩

lemma *res-uminus*:
assumes $k > 0$
assumes $f \in$ *carrier* Z_p
assumes $c \in$ *carrier* (*Zp-res-ring* k)
assumes $c = \ominus_{Zp\text{-}res\text{-}ring\ k}\ (f\ k)$

shows $c = ((\ominus_{Z_p}\ f)\ k)$
$\langle proof \rangle$

lemma *ord-fract*:
assumes $a \in nonzero\ Q_p$
assumes $b \in nonzero\ Q_p$
shows $ord\ (a \div b) = ord\ a - ord\ b$
$\langle proof \rangle$

lemma *val-fract*:
assumes $a \in carrier\ Q_p$
assumes $b \in nonzero\ Q_p$
shows $val\ (a \div b) = val\ a - val\ b$
$\langle proof \rangle$

lemma *zero-fract*:
assumes $a \in nonzero\ Q_p$
shows $\mathbf{0} \div a = \mathbf{0}$
$\langle proof \rangle$

lemma *fract-closed*:
assumes $a \in carrier\ Q_p$
assumes $b \in nonzero\ Q_p$
shows $(a \div b) \in carrier\ Q_p$
$\langle proof \rangle$

lemma *val-of-power*:
assumes $a \in nonzero\ Q_p$
shows $val\ (a[\widehat{\ }](n::nat)) = n*(val\ a)$
$\langle proof \rangle$

lemma *val-zero-imp-val-pow-zero*:
assumes $a \in carrier\ Q_p$
assumes $val\ a = 0$
shows $val\ (a[\widehat{\ }](n::nat)) = 0$
$\langle proof \rangle$

val and ord of powers of p

lemma *val-p-nat-pow*:
$val\ (\mathfrak{p}[\widehat{\ }](k::nat)) = eint\ k$
$\langle proof \rangle$

lemma *ord-p-int-pow*:
$ord\ (\mathfrak{p}[\widehat{\ }](k::int)) = k$
$\langle proof \rangle$

lemma *ord-p-nat-pow*:
$ord\ (\mathfrak{p}[\widehat{\ }](k::nat)) = k$
$\langle proof \rangle$

lemma *val-nonzero-frac*:
  assumes $a \in$ *nonzero* Q_p
  assumes $b \in$ *nonzero* Q_p
  assumes *val* $(a \div b) = c$
  shows *val* $a =$ *val* $b + c$
⟨*proof*⟩

lemma *val-nonzero-frac′*:
  assumes $a \in$ *nonzero* Q_p
  assumes $b \in$ *nonzero* Q_p
  assumes *val* $(a \div b) = 0$
  shows *val* $a =$ *val* b
  ⟨*proof*⟩

lemma *equal-val-imp-equal-ord*:
  assumes $a \in$ *nonzero* Q_p
  assumes $b \in$ *carrier* Q_p
  assumes *val* $a =$ *val* b
  shows *ord* $a =$ *ord* b $b \in$ *nonzero* Q_p
  ⟨*proof*⟩

lemma *int-pow-ord*:
  assumes $a \in$ *nonzero* Q_p
  shows *ord* $(a[\widehat{\ }](i::int)) = i*$ (*ord* a)
⟨*proof*⟩

lemma *int-pow-val*:
  assumes $a \in$ *nonzero* Q_p
  shows *val* $(a[\widehat{\ }](i::int)) = i*$ (*val* a)
  ⟨*proof*⟩

lemma *neg-int-pow-val*:
  assumes $a \in$ *nonzero* Q_p
  shows *val* $(a[\widehat{\ }]-(i::int)) = -$ (*val* $(a[\widehat{\ }]i)$)
  ⟨*proof*⟩

lemma *int-pow-sum-val*:
  assumes $a \in$ *nonzero* Q_p
  shows *val* $(a[\widehat{\ }]((i::int) + j)) =$ (*val* $(a[\widehat{\ }]i)$) + *val* $(a[\widehat{\ }]j)$
  ⟨*proof*⟩

lemma *int-pow-diff-val*:
  assumes $a \in$ *nonzero* Q_p
  shows *val* $(a[\widehat{\ }]((i::int) - j)) =$ (*val* $(a[\widehat{\ }]i)$) − *val* $(a[\widehat{\ }]j)$
⟨*proof*⟩

lemma *nat-add-pow-mult-assoc*:
  assumes $a \in$ *carrier* Q_p

shows $[(n::nat)]\cdot a = [n]\cdot\mathbf{1} \otimes a$
⟨*proof*⟩

lemma(**in** *padic-integers*) *equal-res-imp-equal-ord-Zp*:
assumes $N > 0$
assumes $a \in carrier\ Zp$
assumes $b \in carrier\ Zp$
assumes $a\ N = b\ N$
assumes $a\ N \neq 0$
shows $ord\text{-}Zp\ a = ord\text{-}Zp\ b$
⟨*proof*⟩

lemma(**in** *padic-integers*) *equal-res-mod*:
assumes $N > k$
assumes $a \in carrier\ Zp$
assumes $b \in carrier\ Zp$
assumes $a\ N = b\ N$
shows $a\ k = b\ k$
⟨*proof*⟩

lemma *Qp-char-0*:
assumes $(n::nat) \neq 0$
shows $[n]\cdot\mathbf{1} \neq \mathbf{0}$
⟨*proof*⟩

lemma *Qp-char-0-int*:
assumes $(n::int) \neq 0$
shows $[n]\cdot\mathbf{1} \neq \mathbf{0}$
⟨*proof*⟩

lemma *add-int-pow-inject*:
assumes $[(k::int)]\cdot\mathbf{1} = [(j::int)]\cdot\mathbf{1}$
shows $k = j$
⟨*proof*⟩

lemma *val-ord-nat-inc*:
assumes $(n::nat) > 0$
shows $ord\ ([n]\cdot\mathbf{1}) = val([n]\cdot\mathbf{1})$
⟨*proof*⟩

lemma *val-ord-int-inc*:
assumes $(n::int) \neq 0$
shows $ord\ ([n]\cdot\mathbf{1}) = val([n]\cdot\mathbf{1})$
⟨*proof*⟩

8.4.2 Reduced Angular Component Maps

definition $ac :: nat \Rightarrow padic\text{-}number \Rightarrow int$ **where**
$ac\ n\ x = (if\ x = \mathbf{0}\ then\ 0\ else\ (angular\text{-}component\ x)\ n)$

lemma *ac-in-res-ring*:
  assumes $x \in nonzero\ Q_p$
  shows $ac\ n\ x \in carrier\ (Zp\text{-}res\text{-}ring\ n)$
  ⟨*proof*⟩

lemma *ac-in-res-ring'*[*simp*]:
  assumes $x \in carrier\ Q_p$
  shows $ac\ n\ x \in carrier\ (Zp\text{-}res\text{-}ring\ n)$
  ⟨*proof*⟩

lemma *ac-mult'*:
  assumes $x \in nonzero\ Q_p$
  assumes $y \in nonzero\ Q_p$
  shows $ac\ n\ (x \otimes y) = (ac\ n\ x) \otimes_{Zp\text{-}res\text{-}ring\ n} (ac\ n\ y)$
  ⟨*proof*⟩

lemma *ac-mult*:
  assumes $x \in carrier\ Q_p$
  assumes $y \in carrier\ Q_p$
  shows $ac\ n\ (x \otimes y) = (ac\ n\ x) \otimes_{Zp\text{-}res\text{-}ring\ n} (ac\ n\ y)$
  ⟨*proof*⟩

lemma *ac-one*[*simp*]:
  assumes $n \geq 1$
  shows $ac\ n\ \mathbf{1} = 1$
⟨*proof*⟩

lemma *ac-one'*:
  assumes $n > 0$
  shows $ac\ n\ \mathbf{1} = \mathbf{1}_{Zp\text{-}res\text{-}ring\ n}$
  ⟨*proof*⟩

lemma *ac-units*:
  assumes $x \in nonzero\ Q_p$
  assumes $n > 0$
  shows $ac\ n\ x \in Units\ (Zp\text{-}res\text{-}ring\ n)$
⟨*proof*⟩

lemma *ac-inv*:
  assumes $x \in nonzero\ Q_p$
  assumes $n > 0$
  shows $ac\ n\ (inv\ x) = inv_{Zp\text{-}res\text{-}ring\ n}\ (ac\ n\ x)$
⟨*proof*⟩

lemma *ac-inv'*:
  assumes $x \in nonzero\ Q_p$
  assumes $n > 0$
  shows $ac\ n\ (inv\ x) \otimes_{Zp\text{-}res\text{-}ring\ n} (ac\ n\ x) = \mathbf{1}_{Zp\text{-}res\text{-}ring\ n}$

$\langle proof \rangle$

lemma *ac-inv''*:
assumes $x \in nonzero\ Q_p$
assumes $n > 0$
shows $(ac\ n\ x) \otimes_{Zp\text{-}res\text{-}ring\ n} ac\ n\ (inv\ x) = \mathbf{1}_{Zp\text{-}res\text{-}ring\ n}$
$\langle proof \rangle$

lemma *ac-inv'''*:
assumes $x \in nonzero\ Q_p$
assumes $n > 0$
shows $(ac\ n\ x) \otimes_{Zp\text{-}res\text{-}ring\ n} ac\ n\ (inv\ x) = 1$
$ac\ n\ (inv\ x) \otimes_{Zp\text{-}res\text{-}ring\ n} (ac\ n\ x) = 1$
$\langle proof \rangle$

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 $ac\ n\ a = ac\ n\ b$
$\langle proof \rangle$

lemma *angular-component-nat-pow*:
assumes $a \in nonzero\ Q_p$
shows $angular\text{-}component\ (a\ [\char`\^]\ (k::nat)) = (angular\text{-}component\ a)\ [\char`\^]_{Z_p}\ k$
$\langle proof \rangle$

lemma *angular-component-int-pow*:
assumes $a \in nonzero\ Q_p$
shows $angular\text{-}component\ (a\ [\char`\^]\ (k::int)) = (angular\text{-}component\ a)\ [\char`\^]_{Z_p}\ k$
$\langle proof \rangle$

lemma *ac-nat-pow*:
assumes $a \in nonzero\ Q_p$
shows $ac\ n\ (a\ [\char`\^]\ (k::nat)) = (ac\ n\ a)\char`\^\ k\ mod\ (p\char`\^n)$
$\langle proof \rangle$

lemma *ac-nat-pow'*:
assumes $a \in nonzero\ Q_p$
assumes $n \neq 0$
shows $ac\ n\ (a\ [\char`\^]\ (k::nat)) = (ac\ n\ a)[\char`\^]_{Zp\text{-}res\text{-}ring\ n}\ k$
$\langle proof \rangle$

lemma *ac-int-pow*:
assumes $a \in nonzero\ Q_p$
assumes $n > 0$
shows $ac\ n\ (a\ [\char`\^]\ (k::int)) = (ac\ n\ a)[\char`\^]_{Zp\text{-}res\text{-}ring\ n}\ k$
$\langle proof \rangle$

lemma *angular-component-p*:
angular-component $\mathfrak{p} = \mathbf{1}_{Z_p}$
⟨*proof*⟩

lemma *angular-component-p-nat-pow*:
angular-component $(\mathfrak{p}\ [\widehat{\ }]\ (n::nat)) = \mathbf{1}_{Z_p}$
⟨*proof*⟩

lemma *angular-component-p-int-pow*:
angular-component $(\mathfrak{p}\ [\widehat{\ }]\ (n::int)) = \mathbf{1}_{Z_p}$
⟨*proof*⟩

lemma *ac-p-nat-pow*:
assumes $k > 0$
shows $ac\ k\ (\mathfrak{p}\ [\widehat{\ }]\ (n::nat)) = 1$
⟨*proof*⟩

lemma *ac-p*:
assumes $k > 0$
shows $ac\ k\ \mathfrak{p} = 1$
⟨*proof*⟩

lemma *ac-p-int-pow*:
assumes $k > 0$
shows $ac\ k\ (\mathfrak{p}\ [\widehat{\ }]\ (n::int)) = 1$
⟨*proof*⟩

lemma *angular-component-p-nat-pow-factor*:
assumes $a \in nonzero\ Q_p$
shows *angular-component* $((\mathfrak{p}\ [\widehat{\ }]\ (n::nat)) \otimes a) =$ *angular-component* a
⟨*proof*⟩

lemma *ac-p-nat-pow-factor*:
assumes $m > 0$
assumes $a \in nonzero\ Q_p$
shows $ac\ m\ ((\mathfrak{p}\ [\widehat{\ }]\ (n::nat)) \otimes a) = ac\ m\ a$
⟨*proof*⟩

lemma *angular-component-p-nat-pow-factor-right*:
assumes $a \in nonzero\ Q_p$
shows *angular-component* $(a \otimes (\mathfrak{p}\ [\widehat{\ }]\ (n::nat))) =$ *angular-component* a
⟨*proof*⟩

lemma *ac-p-nat-pow-factor-right*:
assumes $m > 0$
assumes $a \in carrier\ Q_p$
shows $ac\ m\ (a \otimes (\mathfrak{p}\ [\widehat{\ }]\ (n::nat))) = ac\ m\ a$
⟨*proof*⟩

lemma *angular-component-p-int-pow-factor*:
  assumes $a \in carrier\ Q_p$
  shows *angular-component* $((\mathfrak{p}\ [\urcorner]\ (n::int)) \otimes a) = angular\text{-}component\ a$
  ⟨*proof*⟩

lemma *ac-p-int-pow-factor*:
  assumes $a \in nonzero\ Q_p$
  shows $ac\ m\ ((\mathfrak{p}\ [\urcorner]\ (n::int)) \otimes a) = ac\ m\ a$
  ⟨*proof*⟩

lemma *angular-component-p-int-pow-factor-right*:
  assumes $a \in carrier\ Q_p$
  shows *angular-component* $(a \otimes (\mathfrak{p}\ [\urcorner]\ (n::int))) = angular\text{-}component\ a$
  ⟨*proof*⟩

lemma *ac-p-int-pow-factor-right*:
  assumes $a \in carrier\ Q_p$
  shows $ac\ m\ (a \otimes (\mathfrak{p}\ [\urcorner]\ (n::int))) = ac\ m\ a$
  ⟨*proof*⟩

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* $a \in carrier\ Z_p$
  ⟨*proof*⟩

lemma *to-Zp-inc*:
  assumes $a \in \mathcal{O}_p$
  shows $\iota\ (to\text{-}Zp\ a) = a$
⟨*proof*⟩

lemma *inc-to-Zp*:
 assumes $b \in carrier\ Z_p$
 shows *to-Zp* $(\iota\ b) = b$
⟨*proof*⟩

lemma *to-Zp-add*:
  assumes $a \in \mathcal{O}_p$
  assumes $b \in \mathcal{O}_p$
  shows *to-Zp* $(a \oplus b) = to\text{-}Zp\ a \oplus_{Z_p} (to\text{-}Zp\ b)$
  ⟨*proof*⟩

lemma *to-Zp-mult*:
  assumes $a \in \mathcal{O}_p$

assumes $b \in \mathcal{O}_p$
shows $\textit{to-Zp}\ (a \otimes b) = \textit{to-Zp}\ a \otimes_{Z_p} (\textit{to-Zp}\ b)$
$\langle \textit{proof} \rangle$

lemma *to-Zp-minus*:
assumes $a \in \mathcal{O}_p$
assumes $b \in \mathcal{O}_p$
shows $\textit{to-Zp}\ (a \ominus b) = \textit{to-Zp}\ a \ominus_{Z_p} (\textit{to-Zp}\ b)$
$\langle \textit{proof} \rangle$

lemma *to-Zp-one*:
shows $\textit{to-Zp}\ \mathbf{1} = \mathbf{1}_{Z_p}$
$\langle \textit{proof} \rangle$

lemma *to-Zp-zero*:
shows $\textit{to-Zp}\ \mathbf{0} = \mathbf{0}_{Z_p}$
$\langle \textit{proof} \rangle$

lemma *to-Zp-ominus*:
assumes $a \in \mathcal{O}_p$
shows $\textit{to-Zp}\ (\ominus\ a) = \ominus_{Z_p} (\textit{to-Zp}\ a)$
$\langle \textit{proof} \rangle$

lemma *to-Zp-val*:
assumes $a \in \mathcal{O}_p$
shows $\textit{val-Zp}\ (\textit{to-Zp}\ a) = \textit{val}\ a$
$\langle \textit{proof} \rangle$

lemma *val-of-nat-inc*:
$\textit{val}\ ([(k{::}\textit{nat})]{\cdot}\mathbf{1}) \geq \textit{0}$
$\langle \textit{proof} \rangle$

lemma *val-of-int-inc*:
$\textit{val}\ ([(k{::}\textit{int})]{\cdot}\mathbf{1}) \geq \textit{0}$
$\langle \textit{proof} \rangle$

lemma *to-Zp-nat-inc*:
$\textit{to-Zp}\ ([(a{::}\textit{nat})]{\cdot}\mathbf{1}) = [a]{\cdot}_{Z_p}\mathbf{1}\ _{Z_p}$
$\langle \textit{proof} \rangle$

lemma *to-Zp-int-neg*:
$\textit{to-Zp}\ ([(-\textit{int}\ (a{::}\textit{nat}))]{\cdot}\mathbf{1}) = \ominus_{Z_p}([\textit{int}\ a]{\cdot}_{Z_p}\mathbf{1}\ _{Z_p})$
$\langle \textit{proof} \rangle$

lemma(**in** *ring*) *int-add-pow*:
$[\textit{int}\ n] \cdot \mathbf{1} = [n]{\cdot}\mathbf{1}$
$\langle \textit{proof} \rangle$

lemma *int-add-pow*:
$[int\ n] \cdot \mathbf{1} = [n]\cdot\mathbf{1}$
⟨*proof*⟩

lemma *Zp-int-add-pow*:
$[int\ n] \cdot_{Z_p} \mathbf{1}_{Z_p} = [n]\cdot_{Z_p}\mathbf{1}_{Z_p}$
⟨*proof*⟩

lemma *to-Zp-int-inc*:
$to\text{-}Zp\ ([(a::int)]\cdot\mathbf{1}) = ([a]\cdot_{Z_p}\mathbf{1}\ _{Z_p})$
⟨*proof*⟩

lemma *to-Zp-nat-add-pow*:
assumes $a \in \mathcal{O}_p$
shows $to\text{-}Zp\ ([(n::nat)]\cdot a) = [n]\cdot_{Z_p}\ to\text{-}Zp\ a$
⟨*proof*⟩

lemma *val-ring-res*:
assumes $a \in \mathcal{O}_p$
assumes $b \in \mathcal{O}_p$
shows $to\text{-}Zp\ (a \ominus b)\ N = to\text{-}Zp\ a\ N \ominus_{Zp\text{-}res\text{-}ring\ N}\ to\text{-}Zp\ b\ N$
⟨*proof*⟩

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*⟩

lemma(**in** *padic-fields*) *equal-res-imp-res-diff-zero*:
assumes $a \in \mathcal{O}_p$
assumes $b \in \mathcal{O}_p$
assumes $to\text{-}Zp\ a\ N = to\text{-}Zp\ b\ N$
shows $to\text{-}Zp\ (a \ominus b)\ N = 0$
⟨*proof*⟩

lemma(**in** *padic-fields*) *equal-res-imp-val-diff-bound*:
assumes $a \in \mathcal{O}_p$
assumes $b \in \mathcal{O}_p$
assumes $to\text{-}Zp\ a\ N = to\text{-}Zp\ b\ N$
shows $val\ (a \ominus b) \geq N$
⟨*proof*⟩

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 $to\text{-}Zp\ a\ N = to\text{-}Zp\ b\ N$

shows *val a = val b*
⟨*proof*⟩

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*⟩

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 ⇒ padic-number*

type-synonym *padic-field-fun = padic-number ⇒ 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 $g \in$ *carrier* Q_p-x
shows *gauss-norm* $g \leq$ *val* (g k)
⟨*proof*⟩

lemma *gauss-norm-geqI*:
assumes $g \in$ *carrier* (*UP* Q_p)
assumes $\bigwedge n$. *val* (g n) $\geq \alpha$
shows *gauss-norm* $g \geq \alpha$
⟨*proof*⟩

lemma *gauss-norm-eqI*:
assumes $g \in$ *carrier* (*UP* Q_p)
assumes $\bigwedge n$. *val* (g n) $\geq \alpha$
assumes *val* (g i) = α
shows *gauss-norm* $g = \alpha$
⟨*proof*⟩

lemma *nonzero-poly-nonzero-coeff*:
assumes $g \in$ *carrier* Q_p-x
assumes $g \neq \mathbf{0}_{Q_p\text{-}x}$
shows $\exists k$. $k \leq$*degree* $g \wedge g$ $k \neq \mathbf{0}_{Q_p}$
⟨*proof*⟩

lemma *gauss-norm-prop*:
assumes $g \in$ *carrier* Q_p-x
assumes $g \neq \mathbf{0}_{Q_p\text{-}x}$
shows *gauss-norm* $g \neq \infty$
⟨*proof*⟩

lemma *gauss-norm-coeff-norm*:
$\exists n \leq$ *degree* g. (*gauss-norm* g) = *val* (g n)
⟨*proof*⟩

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*⟩

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
⟨*proof*⟩

lemma *gauss-norm-ultrametric*:
  assumes $g \in carrier\ Q_p\text{-}x$
  assumes $h \in carrier\ Q_p\text{-}x$
  shows $gauss\text{-}norm\ (g \oplus_{Q_p\text{-}x} h) \geq min\ (gauss\text{-}norm\ g)\ (gauss\text{-}norm\ h)$
⟨*proof*⟩

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*⟩

lemma *gauss-norm-ultrametric'*:
  assumes $f \in carrier\ (UP\ Q_p)$
  assumes $g \in carrier\ (UP\ Q_p)$
  shows $gauss\text{-}norm\ (f \ominus_{UP\ Q_p}\ g) \geq min\ (gauss\text{-}norm\ f)\ (gauss\text{-}norm\ g)$
  ⟨*proof*⟩

lemma *gauss-norm-finsum*:
  assumes $f \in A \rightarrow carrier\ Q_p\text{-}x$
  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*⟩

lemma *gauss-norm-monom*:
  assumes $a \in carrier\ Q_p$
  shows $gauss\text{-}norm\ (monom\ Q_p\text{-}x\ a\ n) = val\ a$
⟨*proof*⟩

lemma *val-val-ring-prod*:
  assumes $a \in \mathcal{O}_p$
  assumes $b \in carrier\ Q_p$
  shows $val\ (a \otimes_{Q_p}\ b) \geq val\ b$
⟨*proof*⟩

lemma *val-val-ring-prod'*:
  assumes $a \in \mathcal{O}_p$
  assumes $b \in carrier\ Q_p$
  shows $val\ (b \otimes_{Q_p}\ a) \geq val\ b$
  ⟨*proof*⟩

lemma *val-ring-nat-pow-closed*:
  assumes $a \in \mathcal{O}_p$
  shows $(a[\hat{}](n::nat)) \in \mathcal{O}_p$
  ⟨*proof*⟩

lemma *val-ringI*:
  assumes $a \in carrier\ Q_p$
  assumes $val\ a \geq 0$

shows $a \in \mathcal{O}_p$
⟨*proof*⟩

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* $(g \cdot a) \geq$ *gauss-norm g*
⟨*proof*⟩

lemma *positive-gauss-norm-eval*:
assumes $g \in$ *carrier* Q_p*-x*
assumes *gauss-norm* $g \geq 0$
assumes $a \in \mathcal{O}_p$
shows $(g \cdot a) \in \mathcal{O}_p$
⟨*proof*⟩

lemma *positive-gauss-norm-valuation-ring-coeffs*:
assumes $g \in$ *carrier* Q_p*-x*
assumes *gauss-norm* $g \geq 0$
shows $g\ n \in \mathcal{O}_p$
⟨*proof*⟩

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* $g \geq 0$
⟨*proof*⟩

lemma *val-of-add-pow*:
assumes $a \in$ *carrier* Q_p
shows *val* $([(n::nat)] \cdot a) \geq$ *val a*
⟨*proof*⟩

lemma *gauss-norm-pderiv*:
assumes $g \in$ *carrier* $(UP\ Q_p)$
shows *gauss-norm* $g \leq$ *gauss-norm* (*pderiv g*)
⟨*proof*⟩

9.2 Mapping Polynomials with Value Ring Coefficients to Polynomials over $\mathbb{Z}_p$

definition *to-Zp-poly* **where**
to-Zp-poly $g = (\lambda n.$ *to-Zp* $(g\ n))$

lemma *to-Zp-poly-closed*:
assumes $g \in$ *carrier* Q_p*-x*
assumes *gauss-norm* $g \geq 0$

shows *to-Zp-poly* $g \in$ *carrier* $(UP\ Z_p)$
⟨*proof*⟩

definition *poly-inc* **where**
poly-inc $g = (\lambda n{::}nat.\ \iota\ (g\ n))$

lemma *poly-inc-closed*:
assumes $g \in$ *carrier* $(UP\ Z_p)$
shows *poly-inc* $g \in$ *carrier* Q_p*-x*
⟨*proof*⟩

lemma *poly-inc-inverse-right*:
assumes $g \in$ *carrier* $(UP\ Z_p)$
shows *to-Zp-poly* (*poly-inc* g) $= g$
⟨*proof*⟩

lemma *poly-inc-inverse-left*:
assumes $g \in$ *carrier* Q_p*-x*
assumes *gauss-norm* $g \geq 0$
shows *poly-inc* (*to-Zp-poly* g) $= g$
⟨*proof*⟩

lemma *poly-inc-plus*:
assumes $f \in$ *carrier* $(UP\ Z_p)$
assumes $g \in$ *carrier* $(UP\ Z_p)$
shows *poly-inc* $(f \oplus_{UP\ Z_p} g) =$ *poly-inc* $f \oplus_{UP\ Q_p}$ *poly-inc* g
⟨*proof*⟩

lemma *poly-inc-monom*:
assumes $a \in$ *carrier* Z_p
shows *poly-inc* (*monom* $(UP\ Z_p)$ a m) $=$ *monom* $(UP\ Q_p)$ $(\iota\ a)$ m
⟨*proof*⟩

lemma *poly-inc-times*:
assumes $f \in$ *carrier* $(UP\ Z_p)$
assumes $g \in$ *carrier* $(UP\ Z_p)$
shows *poly-inc* $(f \otimes_{UP\ Z_p} g) =$ *poly-inc* $f \otimes_{UP\ Q_p}$ *poly-inc* g
⟨*proof*⟩

lemma *poly-inc-one*:
poly-inc $(\mathbf{1}_{UP\ Z_p}) = \mathbf{1}_{UP\ Q_p}$
⟨*proof*⟩

lemma *poly-inc-zero*:
poly-inc $(\mathbf{0}_{UP\ Z_p}) = \mathbf{0}_{UP\ Q_p}$
⟨*proof*⟩

lemma *poly-inc-hom*:
poly-inc $\in$ *ring-hom* $(UP\ Z_p)$ $(UP\ Q_p)$

$\langle proof \rangle$

lemma *poly-inc-as-poly-lift-hom*:
assumes $f \in carrier\ (UP\ Z_p)$
shows $poly\text{-}inc\ f = poly\text{-}lift\text{-}hom\ Z_p\ Q_p\ \iota\ f$
$\langle proof \rangle$

lemma *poly-inc-eval*:
assumes $g \in carrier\ (UP\ Z_p)$
assumes $a \in carrier\ Z_p$
shows $to\text{-}function\ Q_p\ (poly\text{-}inc\ g)\ (\iota\ a) = \iota\ (to\text{-}function\ Z_p\ g\ a)$
$\langle proof \rangle$

lemma *val-ring-poly-eval*:
assumes $f \in carrier\ (UP\ Q_p)$
assumes $\bigwedge\ i.\ f\ i \in \mathcal{O}_p$
shows $\bigwedge x.\ x \in \mathcal{O}_p \Longrightarrow f \cdot x \in \mathcal{O}_p$
$\langle proof \rangle$

lemma *Zp-res-of-pow*:
assumes $a \in carrier\ Z_p$
assumes $b \in carrier\ Z_p$
assumes $a\ n = b\ n$
shows $(a[\widehat{\ }]_{Z_p}(k::nat))\ n = (b[\widehat{\ }]_{Z_p}(k::nat))\ n$
$\langle proof \rangle$

lemma *to-Zp-nat-pow*:
assumes $a \in \mathcal{O}_p$
shows $to\text{-}Zp\ (a[\widehat{\ }](n::nat)) = (to\text{-}Zp\ a)[\widehat{\ }]_{Z_p}(n::nat)$
$\langle proof \rangle$

lemma *to-Zp-res-of-pow*:
assumes $a \in \mathcal{O}_p$
assumes $b \in \mathcal{O}_p$
assumes $to\text{-}Zp\ a\ n = to\text{-}Zp\ b\ n$
shows $to\text{-}Zp\ (a[\widehat{\ }](k::nat))\ n = to\text{-}Zp\ (b[\widehat{\ }](k::nat))\ n$
$\langle proof \rangle$

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 $to\text{-}Zp\ a\ k = to\text{-}Zp\ b\ k$
shows $to\text{-}Zp\ (g \cdot a)\ k = to\text{-}Zp\ (g \cdot b)\ k$
$\langle proof \rangle$

lemma *to-Zp-poly-eval*:
assumes $g \in carrier\ Q_p\text{-}x$

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*⟩

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* $(g \cdot a) <$ *eint* n
assumes *to-Zp* a n = *to-Zp* b n
shows *val* $(g \cdot b)$ = *val* $(g \cdot a)$
⟨*proof*⟩

lemma *to-Zp-poly-monom*:
assumes $a \in \mathcal{O}_p$
shows *to-Zp-poly* (*monom* (*UP* Q_p) a n) = *monom* (*UP* Z_p) (*to-Zp* a) n
⟨*proof*⟩

lemma *to-Zp-poly-add*:
assumes $f \in$ *carrier* (*UP* Q_p)
assumes *gauss-norm* $f \geq 0$
assumes $g \in$ *carrier* (*UP* Q_p)
assumes *gauss-norm* $g \geq 0$
shows *to-Zp-poly* ($f \oplus_{UP\ Q_p} g$) = *to-Zp-poly* $f \oplus_{UP\ Z_p}$ *to-Zp-poly* g
⟨*proof*⟩

lemma *to-Zp-poly-zero*:
to-Zp-poly ($\mathbf{0}_{UP\ Q_p}$) = $\mathbf{0}_{UP\ Z_p}$
⟨*proof*⟩

lemma *to-Zp-poly-one*:
to-Zp-poly ($\mathbf{1}_{UP\ Q_p}$) = $\mathbf{1}_{UP\ Z_p}$
⟨*proof*⟩

lemma *val-ring-add-pow*:
assumes $a \in$ *carrier* Q_p
assumes *val* $a \geq 0$
shows *val* ([(*n*::*nat*)]$\cdot a$) ≥ 0
⟨*proof*⟩

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*⟩

lemma *val-p-int-pow*:

$val\ (\mathfrak{p}[\widehat{\ }]k) = eint\ (k)$
⟨proof⟩

definition *int-gauss-norm* **where**
$int\text{-}gauss\text{-}norm\ g = (SOME\ n{::}int.\ eint\ n = gauss\text{-}norm\ g)$

lemma *int-gauss-norm-eq*:
assumes $g \in carrier\ (UP\ Q_p)$
assumes $g \neq \mathbf{0}_{UP\ Q_p}$
shows $eint\ (int\text{-}gauss\text{-}norm\ g) = gauss\text{-}norm\ g$
⟨proof⟩

lemma *int-gauss-norm-smult*:
assumes $g \in carrier\ (UP\ Q_p)$
assumes $g \neq \mathbf{0}_{UP\ Q_p}$
assumes $a \in nonzero\ Q_p$
shows $int\text{-}gauss\text{-}norm\ (a \odot_{UP\ Q_p}\ g) = ord\ a + int\text{-}gauss\text{-}norm\ g$
⟨proof⟩

definition *normalize-poly* **where**
$normalize\text{-}poly\ g = (if\ g = \mathbf{0}_{UP\ Q_p}\ then\ g\ else\ (\mathfrak{p}[\widehat{\ }](-\ int\text{-}gauss\text{-}norm\ g)) \odot_{Q_p\text{-}x}\ g)$

lemma *normalize-poly-zero*:
$normalize\text{-}poly\ \mathbf{0}_{UP\ Q_p} = \mathbf{0}_{UP\ Q_p}$
⟨proof⟩

lemma *normalize-poly-nonzero-eq*:
assumes $g \neq \mathbf{0}_{UP\ Q_p}$
assumes $g \in carrier\ (UP\ Q_p)$
shows $normalize\text{-}poly\ g = (\mathfrak{p}[\widehat{\ }](-\ int\text{-}gauss\text{-}norm\ g)) \odot_{UP\ Q_p}\ g$
⟨proof⟩

lemma *int-gauss-norm-normalize-poly*:
assumes $g \neq \mathbf{0}_{UP\ Q_p}$
assumes $g \in carrier\ (UP\ Q_p)$
shows $int\text{-}gauss\text{-}norm\ (normalize\text{-}poly\ g) = 0$
⟨proof⟩

lemma *normalize-poly-closed*:
assumes $g \in carrier\ (UP\ Q_p)$
shows $normalize\text{-}poly\ g \in carrier\ (UP\ Q_p)$
⟨proof⟩

lemma *normalize-poly-nonzero*:
assumes $g \neq \mathbf{0}_{UP\ Q_p}$
assumes $g \in carrier\ (UP\ Q_p)$
shows $normalize\text{-}poly\ g \neq \mathbf{0}_{UP\ Q_p}$

$\langle proof \rangle$

lemma *gauss-norm-normalize-poly*:
assumes $g \neq \mathbf{0}_{UP\ Q_p}$
assumes $g \in carrier\ (UP\ Q_p)$
shows $gauss\text{-}norm\ (normalize\text{-}poly\ g) = 0$
$\langle proof \rangle$

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 $val\ (f \cdot t) = val\ (UPQ.taylor\text{-}term\ x\ f\ i \cdot t)$
$\langle proof \rangle$

9.3 Hensel's Lemma for p-adic fields

theorem *hensels-lemma*:
assumes $f \in carrier\ (UP\ Q_p)$
assumes $a \in \mathcal{O}_p$
assumes $gauss\text{-}norm\ f \geq 0$
assumes $val\ (f \cdot a) > 2 * val\ ((pderiv\ f) \cdot a)$
shows $\exists! \alpha \in \mathcal{O}_p.\ f \cdot \alpha = \mathbf{0} \wedge val\ (a \ominus \alpha) > val\ ((pderiv\ f) \cdot a)$
$\langle proof \rangle$

lemma *nth-root-poly-root-fixed*:
assumes $(n::nat) > 1$
assumes $a \in \mathcal{O}_p$
assumes $val\ (\mathbf{1} \ominus_{Q_p} a) > 2 *\ val\ ([n] \cdot \mathbf{1})$
shows $(\exists!\ b \in \mathcal{O}_p.\ (b[\widehat{\ }]n) = a \wedge\ val\ (b \ominus \mathbf{1}) > val\ ([n] \cdot \mathbf{1}))$
$\langle proof \rangle$

lemma *mod-zeroE*:
assumes $(a::int)\ mod\ k\ = 0$
shows $\exists\, l.\ a = l * k$
$\langle proof \rangle$

lemma *to-Zp-poly-closed'*:
assumes $g \in carrier\ (UP\ Q_p)$
assumes $\bigwedge i.\ g\ i \in \mathcal{O}_p$
shows $to\text{-}Zp\text{-}poly\ g \in carrier\ (UP\ Z_p)$
$\langle proof \rangle$

lemma *to-Zp-poly-eval-to-Zp*:
assumes $g \in carrier\ (UP\ Q_p)$
assumes $\bigwedge i.\ g\ i \in \mathcal{O}_p$
assumes $a \in \mathcal{O}_p$

shows *to-function* Z_p (*to-Zp-poly g*) (*to-Zp a*) = *to-Zp* ($g \cdot a$)
⟨*proof*⟩

lemma *inc-nat-pow*:
assumes $a \in$ *carrier* Z_p
shows ι ([(*n*::*nat*)] $\cdot_{Z_p} a$) = [*n*]·(ι *a*)
⟨*proof*⟩

lemma *poly-inc-pderiv*:
assumes $g \in$ *carrier* (*UP* Z_p)
shows *poly-inc* (*Zp.pderiv g*) = *UPQ.pderiv* (*poly-inc g*)
⟨*proof*⟩

lemma *Zp-hensels-lemma*:
assumes $f \in$ *carrier Zp-x*
assumes $a \in$ *carrier* Z_p
assumes *Zp.to-fun* (*Zp.pderiv f*) $a \neq \mathbf{0}_{Z_p}$
assumes *Zp.to-fun f* $a \neq \mathbf{0}_{Z_p}$
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 $\alpha = \mathbf{0}_{Z_p}$ **and** $\alpha \in$ *carrier* Z_p
val-Zp ($a \ominus_{Z_p} \alpha$) > *val-Zp* (*Zp.to-fun* (*Zp.pderiv f*) *a*)
val-Zp ($a \ominus_{Z_p} \alpha$) = *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*⟩

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* $(x \ominus c) \geq n\}$

lemma *c-ballI*:
  assumes $x \in carrier\ Q_p$
  assumes $val\ (x \ominus c) \geq n$
  shows $x \in c\text{-}ball\ n\ c$
  ⟨*proof*⟩

lemma *c-ballE*:
  assumes $x \in c\text{-}ball\ n\ c$
  shows $x \in carrier\ Q_p$
    $val\ (x \ominus c) \geq n$
  ⟨*proof*⟩

lemma *c-ball-in-Qp*:
  $B_n[c] \subseteq carrier\ Q_p$
  ⟨*proof*⟩

definition
$q\text{-}ball :: nat \Rightarrow int \Rightarrow int \Rightarrow padic\text{-}number \Rightarrow padic\text{-}number\ set$ **where**
$q\text{-}ball\ n\ k\ m\ c = \{x \in carrier\ Q_p.\ (ac\ n\ (x \ominus c) = k \wedge (ord\ (x \ominus c)) = m)\ \}$

lemma *q-ballI*:
  assumes $x \in carrier\ Q_p$
  assumes $ac\ n\ (x \ominus c) = k$
  assumes $(ord\ (x \ominus c)) = m$
  shows $x \in q\text{-}ball\ n\ k\ m\ c$
  ⟨*proof*⟩

lemma *q-ballE*:
  assumes $x \in q\text{-}ball\ n\ k\ m\ c$
  shows $x \in carrier\ Q_p$

  ⟨*proof*⟩

lemma *q-ballE'*:
  assumes $x \in q\text{-}ball\ n\ k\ m\ c$
  shows $ac\ n\ (x \ominus c) = k$
    $(ord\ (x \ominus c)) = m$
  ⟨*proof*⟩

lemma *q-ball-in-Qp*:
  $q\text{-}ball\ n\ k\ m\ c \subseteq carrier\ Q_p$
  ⟨*proof*⟩

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*⟩

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*⟩

definition *is-ball* :: *padic-number set* ⇒ *bool* **where**
$is\text{-}ball\ B = (\exists (m::int).\ \exists\ c \in carrier\ Q_p.\ (B = B_m[c]))$

lemma *is-ball-imp-in-Qp*:
assumes *is-ball B*
shows $B \subseteq carrier\ Q_p$
⟨*proof*⟩

lemma *c-ball-centers*:
assumes *is-ball B*
assumes $B = B_n[c]$
assumes $d \in B$
assumes $c \in carrier\ Q_p$
shows $B = B_n[d]$
⟨*proof*⟩

lemma *c-ball-center-in*:
assumes *is-ball B*
assumes $B = B_n[c]$
assumes $c \in carrier\ Q_p$
shows $c \in B$
⟨*proof*⟩

Every point a has a point b of distance exactly n away from it.

lemma *dist-nonempty*:
assumes $a \in carrier\ Q_p$
shows $\exists\, b \in carrier\ Q_p.\ val\ (b \ominus a) = eint\ n$
⟨*proof*⟩

lemma *dist-nonempty'*:
assumes $a \in carrier\ Q_p$
shows $\exists\, b \in carrier\ Q_p.\ val\ (b \ominus a) = \alpha$
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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 $n = m$
⟨*proof*⟩

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*⟩

lemma *nested-balls'*:
assumes *is-ball B*
assumes *is-ball B'*
assumes $B \cap B' \neq \{\}$
shows $B \subseteq B' \lor B' \subseteq B$
⟨*proof*⟩

definition *is-bounded*:: *padic-number set* ⇒ *bool* **where**

is-bounded $S = (\exists\, n.\ \exists\, c \in$ *carrier* $Q_p.\ S \subseteq B_n[c]\ )$

lemma *empty-is-bounded*:
is-bounded $\{\}$
⟨*proof*⟩

10.2 p-adic Open Sets

definition *is-open*:: *padic-number set* $\Rightarrow$ *bool* **where**
is-open $U \equiv (U \subseteq$ *carrier* $Q_p) \wedge (\forall\, c \in U.\ \exists\, n.\ B_n[c] \subseteq U\ )$

lemma *is-openI*:
assumes $U \subseteq$ *carrier* Q_p
assumes $\bigwedge c.\ c \in U \Longrightarrow \exists\, n.\ B_n[c] \subseteq U$
shows *is-open* U
⟨*proof*⟩

lemma *ball-is-open*:
assumes *is-ball* B
shows *is-open* B
⟨*proof*⟩

lemma *is-open-imp-in-Qp*:
assumes *is-open* U
shows $U \subseteq$ *carrier* Q_p
⟨*proof*⟩

lemma *is-open-imp-in-Qp′*:
assumes *is-open* U
assumes $x \in U$
shows $x \in$ *carrier* Q_p
⟨*proof*⟩

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* $\Rightarrow$ *padic-number set* $\Rightarrow$ *bool* **where**
is-max-ball-of $U\ B \equiv$ (*is-ball* B) $\wedge$ $(B \subseteq U) \wedge (\forall\, B'.\ (($*is-ball* $B') \wedge (B' \subseteq U) \wedge B \subseteq B') \longrightarrow B' \subseteq B)$

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*⟩

lemma *int-prop*:

fixes P:: *int* $\Rightarrow$ *bool*
assumes $P\ n$
assumes $\forall\, m.\ m \leq N \longrightarrow \neg\ P\ m$
shows $\exists\, n.\ P\ n \wedge (\forall\, n'.\ P\ n' \longrightarrow n' \geq n)$
$\langle proof \rangle$

lemma *open-max-ball*:
assumes *is-open* U
assumes $U \neq$ *carrier* Q_p
assumes $c \in U$
shows $\exists\, B.$ *is-max-ball-of* $U\ B \wedge c \in B$
$\langle proof \rangle$

definition *interior* **where**
interior $U = \{a.\ \exists\, B.$ *is-open* $B \wedge B \subseteq U \wedge a \in B\}$

lemma *interior-subset*:
assumes $U \subseteq$ *carrier* Q_p
shows *interior* $U \subseteq U$
$\langle proof \rangle$

lemma *interior-open*:
assumes $U \subseteq$ *carrier* Q_p
shows *is-open* (*interior* U)
$\langle proof \rangle$

lemma *interiorI*:
assumes $W \subseteq U$
assumes *is-open* W
shows $W \subseteq$ *interior* U
$\langle proof \rangle$

lemma *max-ball-interior*:
assumes $U \subseteq$ *carrier* Q_p
assumes *is-max-ball-of* (*interior* U) B
shows *is-max-ball-of* $U\ B$
$\langle proof \rangle$

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* $U\ B' \wedge c \in B'$
$\langle proof \rangle$

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*⟩

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 $B \cap B' = \{\}$
⟨*proof*⟩

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-balls* $U)$. $x \in B)\}$
⟨*proof*⟩

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
⟨*proof*⟩

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. $a \in B$
⟨*proof*⟩

lemma *open-interior*:
assumes *is-open* U
shows *interior* $U = U$
⟨*proof*⟩

lemma *interior-idempotent*:
assumes $U \subseteq$ *carrier* Q_p
shows *interior* (*interior* U) = *interior* U
⟨*proof*⟩

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
  ⟨*proof*⟩

lemma *is-convexE*:
  assumes *is-convex* A
  assumes $x \in A$
  assumes $y \in A$
  assumes $x \leq a$
  assumes $a \leq y$
  shows $a \in A$
  ⟨*proof*⟩

lemma *empty-convex*:
is-convex {}
  ⟨*proof*⟩

lemma *UNIV-convex*:
is-convex UNIV
  ⟨*proof*⟩

definition *closed-interval* (‹*I*[- -]›) **where**
  closed-interval $\alpha\ \beta = \{a\ .\ \alpha \leq a \wedge\ a \leq \beta\}$

lemma *closed-interval-is-convex*:
  assumes $A =$ *closed-interval* $\alpha\ \beta$
  shows *is-convex* A
  ⟨*proof*⟩

lemma *empty-closed-interval*:
$\{\} =$ *closed-interval* ∞ (*eint 1*)
  ⟨*proof*⟩

definition *left-closed-interval* **where**
left-closed-interval $\alpha\ \beta = \{a\ .\ \alpha \leq\ a \wedge\ a < \beta\}$

lemma *left-closed-interval-is-convex*:
  assumes $A =$ *left-closed-interval* $\alpha\ \beta$
  shows *is-convex* A
  ⟨*proof*⟩

definition *closed-ray* **where**
closed-ray $\alpha\ \beta\ = \{a\ .\ \ a \leq \beta\ \}$

lemma *closed-ray-is-convex*:
  assumes $A = \textit{closed-ray}\ \alpha\ \beta$
  shows *is-convex A*
  $\langle proof \rangle$

lemma *UNIV-closed-ray*:
$(\textit{UNIV}::\textit{eint set}) = \textit{closed-ray}\ \alpha\ \infty$
  $\langle proof \rangle$

definition *open-ray* :: *eint* ⇒ *eint* ⇒ *eint set* **where**
$\textit{open-ray}\ \alpha\ \beta\ = \{a\ .\quad a < \beta\ \}$

lemma *open-ray-is-convex*:
  assumes $A = \textit{open-ray}\ \alpha\ \beta$
  shows *is-convex A*
  $\langle proof \rangle$

lemma *open-rayE*:
  assumes $a < \beta$
  shows $a \in \textit{open-ray}\ \alpha\ \beta$
  $\langle proof \rangle$

lemma *value-group-is-open-ray*:
$\textit{UNIV} - \{\infty\} = \textit{open-ray}\ \alpha\ \infty$
$\langle proof \rangle$

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* ⇒ *eint* ⇒ *eint set*) ⇒ *bool*
  where *is-convex-condition I* ≡
    $I = \textit{closed-interval} \lor I = \textit{left-closed-interval} \lor I = \textit{closed-ray} \lor I = \textit{open-ray}$

lemma *convex-condition-imp-convex*:
  assumes *is-convex-condition I*
  shows *is-convex* $(I\ \alpha\ \beta)$
  $\langle proof \rangle$

lemma *bounded-order*:
  assumes $(a::\textit{eint}) < \infty$
  assumes $b \leq\ a$
  obtains $k::\textit{nat}$ **where** $a = b + k$
$\langle proof \rangle$

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* $I \wedge A = (I\ \alpha\ \beta)$
$\langle proof \rangle$

lemma *ex-val-less*:
shows $\exists\ (\alpha{::}eint).\ \alpha < \beta$
$\langle proof \rangle$

lemma *ex-dist-less*:
assumes $c \in carrier\ Q_p$
shows $\exists\ a \in carrier\ Q_p.\ val\ (a \ominus c) < \beta$
$\langle proof \rangle$
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 $A = B$
$\langle proof \rangle$

lemma *equalityI''*:
assumes $\bigwedge x.\ A\ x \Longrightarrow B\ x$
assumes $\bigwedge x.\ B\ x \Longrightarrow A\ x$
shows $\{x.\ A\ x\} = \{x.\ B\ x\}$
$\langle proof \rangle$

lemma *SomeE*:
assumes $a = (SOME\ x.\ P\ x)$
assumes $P\ c$
shows $P\ a$
$\langle proof \rangle$

lemma *SomeE'*:
assumes $a = (SOME\ x.\ P\ x)$
assumes $\exists\ x.\ P\ x$
shows $P\ a$
$\langle proof \rangle$

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-boolean-algebra* S B
| *generator*: $A \in B \Longrightarrow A \cap S \in$ *gen-boolean-algebra* S B
| *union*: $[\![A \in$ *gen-boolean-algebra* S B; $C \in$ *gen-boolean-algebra* S $B]\!] \Longrightarrow A \cup C \in$ *gen-boolean-algebra* S B
| *complement*: $A \in$ *gen-boolean-algebra* S $B \Longrightarrow S - A \in$ *gen-boolean-algebra* S B

lemma *gen-boolean-algebra-subset*:
shows $A \in$ *gen-boolean-algebra* S $B \Longrightarrow A \subseteq S$
⟨*proof*⟩

lemma *gen-boolean-algebra-intersect*:
assumes $A \in$ *gen-boolean-algebra* S B
assumes $C \in$ *gen-boolean-algebra* S B
shows $A \cap C \in$ *gen-boolean-algebra* S B
⟨*proof*⟩

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*⟩

lemma *gen-boolean-algebra-diff-eq*:
assumes $A \in$ *gen-boolean-algebra* S B
assumes $C \in$ *gen-boolean-algebra* S B
shows $A - C = A \cap (S - C)$
⟨*proof*⟩

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*⟩

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*⟩

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* $S\ B$
  ⟨*proof*⟩

lemma *gen-boolean-algebra-generator-subset*:
  assumes $A \in$ *gen-boolean-algebra* $S\ As$
  assumes $As \subseteq Bs$
  shows $A \in$ *gen-boolean-algebra* $S\ Bs$
  ⟨*proof*⟩

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* $S\ (As \cup Cs)$
  ⟨*proof*⟩

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*⟩

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*⟩

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*⟩

lemma *gen-boolean-algebra-idempotent*:
  assumes $S = \bigcup\ Xs$
  shows *gen-boolean-algebra* S (*gen-boolean-algebra* $S\ Xs$) $=$ (*gen-boolean-algebra* $S\ Xs$)
  ⟨*proof*⟩

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* $S\ ((\cap)\ S\ `\ Xs)$
⟨*proof*⟩

Adding a generator to a generated boolean algebra is redundant if the generator already lies in the algebra.

lemma *add-generators*:
assumes $A \in$ *gen-boolean-algebra S Xs*
shows *gen-boolean-algebra S Xs* = *gen-boolean-algebra S* (*insert A Xs*)
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *rec-disjointify-finite*:
finite (*rec-disjointify m f*)
⟨*proof*⟩

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*⟩

lemma *rec-disjointify-union*:
⋃ (*rec-disjointify m f*) = (⋃ *i* ∈ {..<*m*}. *f i*)
⟨*proof*⟩

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*}
⟨*proof*⟩

lemma *enum-rec-disjointify-subset*:
enum-rec-disjointify f m ⊆ *f m*
⟨*proof*⟩

lemma *enum-rec-disjointify-disjoint*:
assumes *k* < *m*
shows *enum-rec-disjointify f m* ∩ *enum-rec-disjointify f k* = {}
⟨*proof*⟩

lemma *enum-rec-disjointify-disjoint'*:
  assumes $k \neq m$
  shows *enum-rec-disjointify f m* $\cap$ *enum-rec-disjointify f k* $= \{\}$
  ⟨*proof*⟩

lemma *rec-disjointify-is-disjoint*:
  assumes $A \in$ *rec-disjointify m f*
  assumes $B \in$ *rec-disjointify m f*
  assumes $A \neq B$
  shows $A \cap B = \{\}$
  ⟨*proof*⟩

definition *enumerates* **where**
enumerates A f $\equiv$ *finite A* $\wedge$ $A = f$ ‘ $\{..<$ (*card A*)$\}$ $\wedge$ *inj-on f* $\{..<$ (*card A*)$\}$

lemma *finite-imp-exists-enumeration*:
  assumes *finite A*
  shows $\exists f$. *enumerates A f*
  ⟨*proof*⟩

lemma *enumeratesE*:
  assumes *enumerates A f*
  shows *finite A* $A = f$ ‘ $\{..<$ *card A*$\}$ *inj-on f* $\{..<$ *card A*$\}$
  ⟨*proof*⟩

lemma *rec-disjointify-finite-set*:
  assumes *enumerates A f*
  shows $\bigcup$ (*rec-disjointify* (*card A*) *f*) $= \bigcup$ *A*
  ⟨*proof*⟩

definition *enumerate* **where**
enumerate A = (*SOME f*. *enumerates A f*)

lemma *enumerate-enumerates*:
  assumes *finite A*
  shows *enumerates A* (*enumerate A*)
  ⟨*proof*⟩

lemma *enumerateE*:
  assumes *finite A*
  assumes $a \in A$
  shows $\exists\ i <$ *card A*. $a =$ (*enumerate A*) *i*
  ⟨*proof*⟩

definition *disjointify* **where**
disjointify As = *rec-disjointify* (*card As*) (*enumerate As*)

lemma *disjointify-is-disjoint*:

assumes *finite As*
assumes $A \in$ *disjointify As*
assumes $B \in$ *disjointify As*
assumes $A \neq B$
shows $A \cap B = \{\}$
⟨*proof*⟩

lemma *disjointify-union*:
assumes *finite As*
shows $\bigcup$ (*disjointify As*) $= \bigcup$ *As*
⟨*proof*⟩

lemma *disjointify-gen-boolean-algebra*:
assumes *finite As*
assumes *As* $\subseteq$ *gen-boolean-algebra S B*
shows *disjointify As* $\subseteq$ *gen-boolean-algebra S B*
⟨*proof*⟩

lemma *disjointify-finite*:
assumes *finite As*
shows *finite* (*disjointify As*)
⟨*proof*⟩

lemma *disjointify-card*:
assumes *finite As*
shows*card* (*disjointify As*) $\leq$ *card As*
⟨*proof*⟩

lemma *disjointify-subset*:
assumes *finite As*
assumes $A \in$ *disjointify As*
shows $\exists B \in As.\ A \subseteq B$
⟨*proof*⟩

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 $As \subseteq Xs$ 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**

$subset\text{-}to\text{-}atom\ Xs\ As = \bigcap\ As - \bigcup\ (Xs - As)$

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 $x \in subset\text{-}to\text{-}atom\ Xs\ As$
  ⟨*proof*⟩

lemma *subset-to-atom-memE*:
  assumes $x \in subset\text{-}to\text{-}atom\ Xs\ As$
  shows $\bigwedge A.\ A \in As \Longrightarrow x \in A$
    $\bigwedge A.\ A \in Xs \Longrightarrow A \notin As \Longrightarrow x \notin A$
  ⟨*proof*⟩

lemma *subset-to-atom-closed*:
  assumes $As \neq \{\}$
  assumes $As \subseteq Xs$
  shows $subset\text{-}to\text{-}atom\ Xs\ As \subseteq \bigcup\ Xs$
⟨*proof*⟩

lemma *subset-to-atom-as-intersection*:
  assumes $As \neq \{\}$
  assumes $As \subseteq Xs$
  assumes $S = \bigcup\ Xs$
  shows $subset\text{-}to\text{-}atom\ Xs\ As = \bigcap\ As \cap (\bigcap\ X \in Xs - As.\ S - X)$
  ⟨*proof*⟩

definition *atoms-of* **where**
$atoms\text{-}of\ Xs = (subset\text{-}to\text{-}atom\ Xs\ \text{`}\ ((Pow\ Xs) - \{\{\}\})) - \{\{\}\}$

lemma *atoms-nonempty*:
  assumes $A \in atoms\text{-}of\ Xs$
  shows $A \neq \{\}$
  ⟨*proof*⟩

lemma *atoms-of-disjoint*:
  assumes $A \in atoms\text{-}of\ Xs$
  assumes $B \in atoms\text{-}of\ Xs$
  assumes $A \neq B$
  shows $A \cap B = \{\}$
⟨*proof*⟩

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 $A \in atoms\text{-}of\ Xs$
  assumes $X \in Xs$
  shows $X \cap A = \{\} \vee A \subseteq X$
⟨*proof*⟩

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 $x \in subset\text{-}to\text{-}atom\ Xs\ (point\text{-}to\text{-}type\ Xs\ x)$
⟨*proof*⟩

lemma *point-to-type-nonempty*:
assumes $x \in \bigcup\ Xs$
shows $point\text{-}to\text{-}type\ Xs\ x \neq \{\}$
⟨*proof*⟩

lemma *point-to-type-closed*:
$point\text{-}to\text{-}type\ Xs\ x \subseteq Pow\ (\bigcup\ Xs)$
⟨*proof*⟩

lemma *atoms-of-covers*:
assumes $X = \bigcup\ Xs$
shows $\bigcup\ (atoms\text{-}of\ Xs) = X$
⟨*proof*⟩

lemma *atoms-of-covers'*:
shows $\bigcup\ (atoms\text{-}of\ Xs) = \bigcup\ Xs$
⟨*proof*⟩

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 $A \in atoms\text{-}of\ Xs$
assumes $a \in A$
shows $A = subset\text{-}to\text{-}atom\ Xs\ (point\text{-}to\text{-}type\ Xs\ a)$
⟨*proof*⟩

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**
$point\text{-}to\text{-}atom\ Xs\ a = subset\text{-}to\text{-}atom\ Xs\ (point\text{-}to\text{-}type\ Xs\ a)$

lemma *point-to-atom-closed*:
assumes $x \in \bigcup\ Xs$
shows $point\text{-}to\text{-}atom\ Xs\ x \in atoms\text{-}of\ Xs$

⟨*proof*⟩

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* ‘ ($\bigcup$ *Xs*)
⟨*proof*⟩

12.2.3 Atoms of Generated Boolean Algebras

lemma *atoms-of-gen-boolean-algebra*:
assumes *Xs* $\subseteq$ *gen-boolean-algebra S B*
assumes *finite Xs*
shows *atoms-of Xs* $\subseteq$ *gen-boolean-algebra S B*
⟨*proof*⟩

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 $\bigcup$ *Xs* $\subseteq$ *S*
assumes *A* $\in$ *atoms-of Xs*
assumes *X* $\in$ *gen-boolean-algebra S Xs*
shows $X \cap A = \{\} \vee A \subseteq X$
⟨*proof*⟩

lemma *finite-set-imp-finite-atoms*:
assumes *finite Xs*
shows *finite* (*atoms-of Xs*)
⟨*proof*⟩

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*⟩

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*⟩

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*)
⟨*proof*⟩

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*⟩

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*⟩

lemma *atoms-closed*:
assumes *finite Xs*
assumes $A \in$ *atoms-of* (*gen-boolean-algebra S Xs*)
assumes $S = \bigcup\ Xs$
shows $A \in$ (*gen-boolean-algebra S Xs*)
⟨*proof*⟩

lemma *atoms-finite*:
assumes *finite Xs*
shows *finite* ((*atoms-of* (*gen-boolean-algebra S Xs*)))
⟨*proof*⟩

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*⟩

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 $A \cap B = \{\}$
⟨*proof*⟩

lemma *disjointI*:
assumes $\bigwedge A\ B.\ A \in Ss \Longrightarrow B \in Ss \Longrightarrow A \neq B \Longrightarrow A \cap B = \{\}$
shows *disjoint Ss*
⟨*proof*⟩

definition *is-partition* :: *'a set set* ⇒ *'a set* ⇒ *bool* (**infixl** ‹*partitions*› *75*) **where**
S partitions A = (*disjoint S* ∧ $\bigcup$ *S* = *A*)

lemma *is-partitionE*:
assumes *S partitions A*
shows *disjoint S*
$\bigcup$ *S* = *A*
⟨*proof*⟩

lemma *is-partitionI*:
assumes *disjoint S*
assumes $\bigcup$ *S* = *A*
shows *S partitions A*
⟨*proof*⟩

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* ∧ *Bs partitions a* ∧ $(\forall b \in Bs.\ P\ b)$
shows $\exists Bs.$ *finite Bs* ∧ *Bs partitions A* ∧ $(\forall b \in Bs.\ P\ b)$
⟨*proof*⟩

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*⟩

lemma *pairwise-intersect-finite*:
  assumes *finite As*
  assumes *finite Bs*
  shows *finite* (*pairwise-intersect As Bs*)
⟨*proof*⟩

definition *family-intersect* **where**
family-intersect parts = *atoms-of* ($\bigcup$ *parts*)

lemma *family-intersect-partitions*:
  assumes $\bigwedge$*Ps*. *Ps* ∈ *parts* ⟹ *Ps partitions A*
  assumes $\bigwedge$*Ps*. *Ps* ∈ *parts* ⟹ *finite Ps*
  assumes *finite parts*
  assumes *parts* ≠ {}
  shows *family-intersect parts partitions A*
⟨*proof*⟩

lemma *family-intersect-memE*:
  assumes $\bigwedge$*Ps*. *Ps* ∈ *parts* ⟹ *Ps partitions A*
  assumes $\bigwedge$*Ps*. *Ps* ∈ *parts* ⟹ *finite Ps*
  assumes *finite parts*
  assumes *parts* ≠ {}
  shows $\bigwedge$*Ps a*. *a* ∈ *family-intersect parts* ⟹ *Ps* ∈ *parts* ⟹ ∃ *P* ∈ *Ps*. *a* ⊆ *P*
⟨*proof*⟩

lemma *family-intersect-mem-inter*:
  assumes $\bigwedge$*Ps*. *Ps* ∈ (*parts*:: '*a set set set*) ⟹ *Ps partitions A*
  assumes $\bigwedge$*Ps*. *Ps* ∈ *parts* ⟹ *finite Ps*
  assumes *finite parts*
  assumes *parts* ≠ {}
  assumes *a* ∈ *family-intersect parts*
  shows ∃*f*. ∀ *Ps* ∈ *parts*. *f Ps* ∈ *Ps* ∧ *a* = ($\bigcap$ *Ps* ∈ *parts*. *f Ps*)
⟨*proof*⟩

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 $\bigwedge$*Ps*. *Ps* ∈ *parts* ⟹ *Ps partitions A*
  assumes $\bigwedge$*Ps*. *Ps* ∈ *parts* ⟹ *finite Ps*
  assumes $\bigwedge$*Ps P*. *Ps* ∈ *parts* ⟹ *P* ∈ *Ps* ⟹ *P* ∈ *gen-boolean-algebra S B*
  assumes *finite parts*
  assumes *parts* ≠ {}
  shows $\bigwedge$*P*. *P* ∈ *family-intersect parts* ⟹ *P* ∈ *gen-boolean-algebra S B*
⟨*proof*⟩

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 $(t\#x) = x$
⟨*proof*⟩

lemma *list-hd*:
hd $(t\#x) = t$
⟨*proof*⟩

sublocale *padic-fields* < *cring-coord-rings* Q_p *UP* Q_p
⟨*proof*⟩

sublocale *padic-fields* < *Qp*: *domain-coord-rings* Q_p *UP* Q_p
⟨*proof*⟩

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 $t \in carrier\ \ Q_p$
  ⟨*proof*⟩

lemma *segment-in-car'*:
  assumes $x@[t] \in carrier\ (Q_p{}^{Suc\ n})$
  shows $x \in carrier\ (Q_p{}^n)$
⟨*proof*⟩

lemma *Qp-zero*:
$Q_p{}^0 = nil\text{-}ring$
  ⟨*proof*⟩

lemma *Qp-zero-carrier*:
$carrier\ (Q_p{}^0) = \{[]\}$
  ⟨*proof*⟩

Abbreviation for constant polynomials

abbreviation(*input*) *Qp-to-IP* **where**
$Qp\text{-}to\text{-}IP\ k \equiv Qp.indexed\text{-}const\ k$

lemma *Qp-to-IP-car*:
  assumes $k \in carrier\ Q_p$
  shows $Qp\text{-}to\text{-}IP\ k \in carrier\ (Q_p[\mathcal{X}_n])$
  ⟨*proof*⟩

lemma(**in** *cring-coord-rings*) *smult-closed*:
  assumes $a \in carrier\ R$
  assumes $q \in carrier\ (R[\mathcal{X}_n])$
  shows $a \odot_{R[\mathcal{X}_n]} q \in carrier\ (R[\mathcal{X}_n])$
  ⟨*proof*⟩

lemma *Qp-poly-smult-cfs*:
  assumes $a \in carrier\ Q_p$
  assumes $P \in carrier\ (Q_p[\mathcal{X}_n])$
  shows $(a \odot_{Q_p[\mathcal{X}_n]} P)\ m = a \otimes (P\ m)$
  ⟨*proof*⟩

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 $a \odot_{Q_p[\mathcal{X}_n]} (P \oplus_{Q_p[\mathcal{X}_n]} q) = (a \odot_{Q_p[\mathcal{X}_n]} P)\ \oplus_{Q_p[\mathcal{X}_n]} (a \odot_{Q_p[\mathcal{X}_n]} q)$

⟨*proof*⟩

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 $(a \oplus b) \odot_{Q_p[\mathcal{X}_n]} P = (a \odot_{Q_p[\mathcal{X}_n]} P) \oplus_{Q_p[\mathcal{X}_n]} (b \odot_{Q_p[\mathcal{X}_n]} P)$
⟨*proof*⟩

abbreviation(*input*) *Qp-funs* **where**
$Qp\text{-}funs\ n \equiv Fun_n\ Q_p$

13.2 Evaluation of Polynomials in $\mathbb{Q}_p$

abbreviation(*input*) *Qp-ev* **where**
$Qp\text{-}ev\ P\ q \equiv (eval\text{-}at\text{-}point\ Q_p\ q\ P)$

lemma *Qp-ev-one*:
assumes $a \in carrier\ (Q_p{}^n)$
shows $Qp\text{-}ev\ \mathbf{1}_{Q_p[\mathcal{X}_n]}\ a = \mathbf{1}$ ⟨*proof*⟩

lemma *Qp-ev-zero*:
assumes $a \in carrier\ (Q_p{}^n)$
shows $Qp\text{-}ev\ \mathbf{0}_{Q_p[\mathcal{X}_n]}\ a = \mathbf{0}$⟨*proof*⟩

lemma *Qp-eval-pvar-pow*:
assumes $a \in carrier\ (Q_p{}^n)$
assumes $k < n$
assumes $(m::nat) \neq 0$
shows $Qp\text{-}ev\ ((pvar\ Q_p\ k)[\urcorner]_{Q_p[\mathcal{X}_n]}\ m)\ a = ((a!k)[\urcorner]m)$
⟨*proof*⟩

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 $\bigwedge i.\ i < length\ fs \Longrightarrow fs!i \in carrier\ (Q_p[\mathcal{X}_m])$
shows *is-poly-tuple m fs*
⟨*proof*⟩

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*⟩

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
⟨*proof*⟩

lemma *Qp-poly-mapE′*:
assumes *as* ∈ *carrier* $(Q_p{}^n)$
shows *length* (*poly-map n fs as*) = *length fs*
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *poly-map-apply*:
assumes *as* ∈ *carrier* $(Q_p{}^n)$
shows *poly-map n fs as* = *poly-tuple-eval fs as*
⟨*proof*⟩

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*
⟨*proof*⟩

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*)
⟨*proof*⟩

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*))
⟨*proof*⟩

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-poly-comp m fs f*) $\in$ *carrier* $(Q_p[\mathcal{X}_m])$
⟨*proof*⟩

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-poly-comp m fs* $(f \oplus_{Q_p[\mathcal{X}_n]} g)$ $=$ (*Qp-poly-comp m fs f*) $\oplus_{Q_p[\mathcal{X}_m]}$ (*Qp-poly-comp m fs g*)
⟨*proof*⟩

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-poly-comp m fs* $(f \otimes_{Q_p[\mathcal{X}_n]} g)$ $=$ (*Qp-poly-comp m fs f*) $\otimes_{Q_p[\mathcal{X}_m]}$ (*Qp-poly-comp m fs g*)
⟨*proof*⟩

lemma *poly-compose-const*:
assumes *is-poly-tuple m fs*
assumes *length fs* $=$ *n*
assumes $a \in$ *carrier* Q_p
shows *Qp-poly-comp m fs* (*Qp-to-IP a*) $=$ *Qp-to-IP a*
⟨*proof*⟩

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-ev* (*Qp-poly-comp m fs f*) *as* $=$ *Qp-ev f* (*poly-map m fs as*)
⟨*proof*⟩

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 $q \in$ *carrier* Q_p-*x*
shows *from-Qp-x* $q \in$ *carrier* $(Q_p[\mathcal{X}_1])$
⟨*proof*⟩

lemma *to-Qp-x-closed*:
assumes $q \in$ *carrier* $(Q_p[\mathcal{X}_1])$
shows *to-Qp-x* $q \in$ *carrier* Q_p-*x*
⟨*proof*⟩

lemma *to-Qp-x-from-Qp-x*:
assumes $q \in$ *carrier* $(Q_p[\mathcal{X}_1])$
shows *from-Qp-x* (*to-Qp-x* q) $= q$
⟨*proof*⟩

lemma *from-Qp-x-to-Qp-x*:
assumes $q \in$ *carrier* Q_p-*x*
shows *to-Qp-x* (*from-Qp-x* q) $= q$
⟨*proof*⟩

ring hom properties of these maps

lemma *to-Qp-x-ring-hom*:
to-Qp-x $\in$ *ring-hom* $(Q_p[\mathcal{X}_1])$ Q_p-*x*
⟨*proof*⟩

lemma *from-Qp-x-ring-hom*:
from-Qp-x $\in$ *ring-hom* Q_p-*x* $(Q_p[\mathcal{X}_1])$
⟨*proof*⟩

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
⟨*proof*⟩

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
⟨*proof*⟩

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*⟩

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*⟩

13.4 n^{th}-Power Sets over $\mathbb{Q}_p$

definition *P-set* **where**
P-set $(n::nat) = \{a \in$ *nonzero* Q_p. $(\exists\, y \in$ *carrier* Q_p . $(y[\hat{\ }]\ n) = a)\}$

lemma *P-set-carrier*:
P-set $n \subseteq$ *carrier* Q_p
⟨*proof*⟩

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
⟨*proof*⟩

lemma *P-set-nonzero*:
P-set $n \subseteq$ *nonzero* Q_p
⟨*proof*⟩

lemma *P-set-nonzero'*:
assumes $a \in$ *P-set* n
shows $a \in$ *nonzero* Q_p
$a \in$ *carrier* Q_p
⟨*proof*⟩

lemma *P-set-one*:
assumes $n \neq 0$
shows $\mathbf{1} \in$ *P-set* $(n::nat)$
⟨*proof*⟩

lemma *zeroth-P-set*:
P-set $0 = \{\mathbf{1}\}$
⟨*proof*⟩

lemma *P-set-mult-closed*:
assumes $n \neq 0$
assumes $a \in$ *P-set* n
assumes $b \in$ *P-set* n
shows $a \otimes b \in$ *P-set* n
⟨*proof*⟩

lemma *P-set-inv-closed*:
  assumes $a \in$ *P-set n*
  shows *inv a* $\in$ *P-set n*
$\langle proof \rangle$

lemma *P-set-val*:
  assumes $a \in$ *P-set* (*n*::*nat*)
  shows (*ord a*) *mod n* = *0*
$\langle proof \rangle$

lemma *P-set-pow*:
  assumes $n > 0$
  assumes $s \in$ *P-set n*
  shows $s[\urcorner]k \in$ *P-set* $(n*k)$
$\langle proof \rangle$

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 $\mathrm{val}(x) \leq \mathrm{val}(y)$, 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*) $P = \{q \in$ *carrier* $(Q_p{}^m)$. $\exists\, y \in$ *carrier* Q_p. *Qp-ev P q* $= (y[\urcorner]n)\}$

lemma *basic-semialg-set-zero-set*:
  assumes $P \in$ *carrier* $(Q_p[\mathcal{X}_m])$
  assumes $q \in$ *carrier* $(Q_p{}^m)$
  assumes *Qp-ev P q* $= \mathbf{0}$
  assumes $n \neq 0$
  shows $q \in$ *basic-semialg-set* (*m*::*nat*) (*n*::*nat*) *P*
$\langle proof \rangle$

lemma *basic-semialg-set-def'*:
  assumes $n \neq 0$
  assumes $P \in$ *carrier* $(Q_p[\mathcal{X}_m])$

shows *basic-semialg-set* $(m{::}nat)$ $(n{::}nat)$ $P = \{q \in carrier\ ({Q_p}^m).\ Qp\text{-}ev\ P\ q = \mathbf{0} \lor Qp\text{-}ev\ P\ q \in (P\text{-}set\ n)\}$
⟨*proof*⟩

lemma *basic-semialg-set-memI*:
assumes $q \in carrier\ ({Q_p}^m)$
assumes $y \in carrier\ Q_p$
assumes $Qp\text{-}ev\ P\ q = (y[\widehat{\ }]n)$
shows $q \in basic\text{-}semialg\text{-}set\ m\ n\ P$
⟨*proof*⟩

lemma *basic-semialg-set-memE*:
assumes $q \in basic\text{-}semialg\text{-}set\ m\ n\ P$
shows $q \in carrier\ ({Q_p}^m)$
$\exists\, y \in carrier\ Q_p.\ Qp\text{-}ev\ P\ q = (y[\widehat{\ }]n)$
⟨*proof*⟩

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*:
$(carrier\ ({Q_p}^n)) \in semialg\text{-}sets\ n$
⟨*proof*⟩

lemma *empty-set-is-semialg*:
$\{\} \in semialg\text{-}sets\ n$
⟨*proof*⟩

lemma *semialg-intersect*:
assumes $A \in semialg\text{-}sets\ n$
assumes $B \in semialg\text{-}sets\ n$
shows $(A \cap B) \in semialg\text{-}sets\ n$
⟨*proof*⟩

lemma *semialg-union*:
assumes $A \in semialg\text{-}sets\ n$
assumes $B \in semialg\text{-}sets\ n$
shows $(A \cup B) \in semialg\text{-}sets\ n$
⟨*proof*⟩

lemma *semialg-complement*:

assumes $A \in$ *semialg-sets n*
shows (*carrier* $(Q_p{}^n) - A) \in$ *semialg-sets n*
⟨*proof*⟩

lemma *semialg-zero*:
assumes $A \in$ *semialg-sets 0*
shows $A = \{[]\} \vee A = \{\}$
⟨*proof*⟩

lemma *basic-semialg-is-semialg*:
assumes *is-basic-semialg n A*
shows $A \in$ *semialg-sets n*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *is-semialgebraic-closed*:
assumes *is-semialgebraic n S*
shows $S \subseteq$ *carrier* $(Q_p{}^n)$
⟨*proof*⟩

lemma *is-semialgebraicI*:
assumes $S \in$ *semialg-sets n*
shows *is-semialgebraic n S*
⟨*proof*⟩

lemma *basic-semialg-is-semialgebraic*:
assumes *is-basic-semialg n A*
shows *is-semialgebraic n A*
⟨*proof*⟩

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*

⟨*proof*⟩

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*⟩

lemma *p-times-square-not-square'*:
assumes $a \in$ *carrier* Q_p
shows $\mathfrak{p} \otimes (a\ [\widehat{\ }]\ (2::nat)) = \mathbf{0} \Longrightarrow a = \mathbf{0}$
⟨*proof*⟩

lemma *zero-set-semialg-set*:
assumes $q \in$ *carrier* $(Q_p[\mathcal{X}_n])$
assumes $a \in$ *carrier* $(Q_p{}^n)$
shows *Qp-ev* $q\ a = \mathbf{0} \longleftrightarrow (\ \exists\, y \in$ *carrier* $Q_p.\ \mathfrak{p} \otimes ((Qp\text{-}ev\ q\ a)\ [\widehat{\ }]\ (2::nat)) = y[\widehat{\ }](2::nat))$
⟨*proof*⟩

lemma *alg-as-semialg*:
assumes $P \in$ *carrier* $(Q_p[\mathcal{X}_n])$
assumes $q = \mathfrak{p} \odot_{Q_p[\mathcal{X}_n]} (P[\widehat{\ }]_{Q_p[\mathcal{X}_n]}\ (2::nat))$
shows *zero-set* $Q_p\ n\ P =$ *basic-semialg-set* $n\ (2::nat)\ q$
⟨*proof*⟩

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* $n\ S$
⟨*proof*⟩

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* $n\ S$
⟨*proof*⟩

Algebraic sets are semialgebraic

lemma *is-algebraic-imp-is-semialg*:
assumes *is-algebraic* $Q_p\ n\ S$
shows *is-semialgebraic* $n\ S$
⟨*proof*⟩

13.5.3 Basic Lemmas about the Semialgebraic Predicate

Finite and cofinite sets are semialgebraic

lemma *finite-is-semialg*:

assumes $F \subseteq$ *carrier* $(Q_p{}^n)$
assumes *finite* F
shows *is-semialgebraic* n F
⟨*proof*⟩

definition *is-cofinite* **where**
is-cofinite n F = *finite* (*ring-pow-comp* Q_p n F)

lemma *is-cofiniteE*:
assumes $F \subseteq$ *carrier* $(Q_p{}^n)$
assumes *is-cofinite* n F
shows *finite* (*carrier* $(Q_p{}^n) - F$)
⟨*proof*⟩

lemma *complement-is-semialg*:
assumes *is-semialgebraic* n F
shows *is-semialgebraic* n ((*carrier* $(Q_p{}^n)) - F$)
⟨*proof*⟩

lemma *cofinite-is-semialgebraic*:
assumes $F \subseteq$ *carrier* $(Q_p{}^n)$
assumes *is-cofinite* n F
shows *is-semialgebraic* n F
⟨*proof*⟩

lemma *diff-is-semialgebraic*:
assumes *is-semialgebraic* n A
assumes *is-semialgebraic* n B
shows *is-semialgebraic* n $(A - B)$
⟨*proof*⟩

lemma *intersection-is-semialg*:
assumes *is-semialgebraic* n A
assumes *is-semialgebraic* n B
shows *is-semialgebraic* n $(A \cap B)$
⟨*proof*⟩

lemma *union-is-semialgebraic*:
assumes *is-semialgebraic* n A
assumes *is-semialgebraic* n B
shows *is-semialgebraic* n $(A \cup B)$
⟨*proof*⟩

lemma *carrier-is-semialgebraic*:
is-semialgebraic n (*carrier* $(Q_p{}^n)$)
⟨*proof*⟩

lemma *empty-is-semialgebraic*:
is-semialgebraic n $\{\}$

$\langle proof \rangle$

13.5.4 One-Dimensional Semialgebraic Sets

definition *one-var-semialg* **where**
one-var-semialg $S = ((\mathit{to\text{-}R1}\ `\ S) \in (\mathit{semialg\text{-}sets}\ 1))$

definition *univ-basic-semialg-set* **where**
univ-basic-semialg-set $(m{::}nat)\ P = \{a \in \mathit{carrier}\ Q_p.\ (\exists\, y \in \mathit{carrier}\ Q_p.\ (P \cdot a = (y[\urcorner]m)))\}$

Equivalence of univ_basic_semialg_sets and semialgebraic subsets of $\mathbb{Q}^1$

lemma *univ-basic-semialg-set-to-semialg-set*:
assumes $P \in \mathit{carrier}\ Q_p\text{-}x$
assumes $m \neq 0$
shows $\mathit{to\text{-}R1}\ `\ (\mathit{univ\text{-}basic\text{-}semialg\text{-}set}\ m\ P) = \mathit{basic\text{-}semialg\text{-}set}\ 1\ m\ (\mathit{from\text{-}Qp\text{-}x}\ P)$
$\langle proof \rangle$

definition *is-univ-semialgebraic* **where**
is-univ-semialgebraic $S = (S \subseteq \mathit{carrier}\ Q_p \wedge \mathit{is\text{-}semialgebraic}\ 1\ (\mathit{to\text{-}R1}\ `\ S))$

lemma *is-univ-semialgebraicE*:
assumes *is-univ-semialgebraic S*
shows $\mathit{is\text{-}semialgebraic}\ 1\ (\mathit{to\text{-}R1}\ `\ S)$
$\langle proof \rangle$

lemma *is-univ-semialgebraicI*:
assumes $\mathit{is\text{-}semialgebraic}\ 1\ (\mathit{to\text{-}R1}\ `\ S)$
shows *is-univ-semialgebraic S*
$\langle proof \rangle$

lemma *univ-basic-semialg-set-is-univ-semialgebraic*:
assumes $P \in \mathit{carrier}\ Q_p\text{-}x$
assumes $m \neq 0$
shows *is-univ-semialgebraic* (*univ-basic-semialg-set m P*)
$\langle proof \rangle$

lemma *intersection-is-univ-semialgebraic*:
assumes *is-univ-semialgebraic A*
assumes *is-univ-semialgebraic B*
shows *is-univ-semialgebraic* $(A \cap B)$
$\langle proof \rangle$

lemma *union-is-univ-semialgebraic*:
assumes *is-univ-semialgebraic A*
assumes *is-univ-semialgebraic B*
shows *is-univ-semialgebraic* $(A \cup B)$
$\langle proof \rangle$

lemma *diff-is-univ-semialgebraic*:
  assumes *is-univ-semialgebraic* A
  assumes *is-univ-semialgebraic* B
  shows *is-univ-semialgebraic* $(A - B)$
  ⟨*proof*⟩

lemma *finite-is-univ-semialgebraic*:
  assumes $A \subseteq$ *carrier* Q_p
  assumes *finite* A
  shows *is-univ-semialgebraic* A
  ⟨*proof*⟩

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. $a[\urcorner](2::nat) \oplus_{Q_p} \mathfrak{p}\otimes b[\urcorner](2::nat) = (y\ [\urcorner]\ (2::nat))$
⟨*proof*⟩

lemma *eint-minus-ineq′*:
  assumes $(a::eint) \geq b$
  shows $a - b \geq 0$
  ⟨*proof*⟩

lemma *Qp-square-root-criterion*:
  assumes $p \neq 2$
  assumes $a \in$ *carrier* Q_p
  assumes $b \in$ *carrier* Q_p
  assumes *ord* $b \geq$ *ord* a
  assumes $a \neq \mathbf{0}$
  assumes $b \neq \mathbf{0}$
  shows $\exists y \in$ *carrier* Q_p. $a[\urcorner](2::nat) \oplus_{Q_p} \mathfrak{p}\otimes b[\urcorner](2::nat) = (y\ [\urcorner]\ (2::nat))$
⟨*proof*⟩

lemma *Qp-val-ring-alt-def0*:
  assumes $a \in$ *nonzero* Q_p
  assumes *ord* $a \geq 0$
  shows $\exists y \in$ *carrier* Q_p. $\mathbf{1} \oplus_{Q_p} (\mathfrak{p}[\urcorner](3::nat)) \otimes (a[\urcorner](4::nat)) = (y[\urcorner](2::nat))$
⟨*proof*⟩

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 $val\ a \leq val\ b \longleftrightarrow (\exists\, y \in carrier\ Q_p.\ (a[\widehat{\ }](2::nat)) \oplus_{Q_p} (\mathfrak{p} \otimes (b[\widehat{\ }](2::nat))) = (y[\widehat{\ }](2::nat)))$
$\langle proof \rangle$

Defining the valuation ring semialgebraically for all primes

lemma *Qp-val-ring-alt-def*:
assumes $a \in carrier\ Q_p$
shows $a \in \mathcal{O}_p \longleftrightarrow (\exists\, y \in carrier\ Q_p.\ \mathbf{1} \oplus_{Q_p} (\mathfrak{p}[\widehat{\ }](3::nat)) \otimes (a[\widehat{\ }](4::nat)) = (y[\widehat{\ }](2::nat)))$
$\langle proof \rangle$

lemma *Qp-val-alt-def*:
assumes $a \in carrier\ Q_p$
assumes $b \in carrier\ Q_p$
shows $val\ b \leq val\ a \longleftrightarrow (\exists\, y \in carrier\ Q_p.\ (b[\widehat{\ }](4::nat)) \oplus_{Q_p} (\mathfrak{p}[\widehat{\ }](3::nat)) \otimes (a[\widehat{\ }](4::nat)) = (y[\widehat{\ }](2::nat)))$
$\langle proof \rangle$

The polynomial in two variables which semialgebraically defines the valuation relation

definition *Qp-val-poly* **where**
$Qp\text{-}val\text{-}poly = (pvar\ Q_p\ 1)[\widehat{\ }]_{Q_p[\mathcal{X}_2]}(4::nat) \oplus_{Q_p[\mathcal{X}_2]} (\mathfrak{p}[\widehat{\ }](3::nat) \odot_{Q_p[\mathcal{X}_2]} ((pvar\ Q_p\ 0)[\widehat{\ }]_{Q_p[\mathcal{X}_2]}(4::nat)))$

lemma *Qp-val-poly-closed*:
$Qp\text{-}val\text{-}poly \in carrier\ (Q_p[\mathcal{X}_2])$
$\langle proof \rangle$

lemma *Qp-val-poly-eval*:
assumes $a \in carrier\ Q_p$
assumes $b \in carrier\ Q_p$
shows $Qp\text{-}ev\ Qp\text{-}val\text{-}poly\ [a,\ b] = (b[\widehat{\ }](4::nat)) \oplus_{Q_p} (\mathfrak{p}[\widehat{\ }](3::nat)) \otimes (a[\widehat{\ }](4::nat))$
$\langle proof \rangle$

lemma *Qp-2I*:
assumes $a \in carrier\ Q_p$
assumes $b \in carrier\ Q_p$
shows $[a,b] \in carrier\ (Q_p{}^2)$
$\langle proof \rangle$

lemma *pair-id*:
assumes $length\ as = 2$
shows $as = [as!0,\ as!1]$
$\langle proof \rangle$

lemma *Qp-val-semialg*:
assumes $a \in carrier\ Q_p$
assumes $b \in carrier\ Q_p$
shows $val\ b \leq val\ a \longleftrightarrow [a,b] \in basic\text{-}semialg\text{-}set\ 2\ (2::nat)\ Qp\text{-}val\text{-}poly$
$\langle proof \rangle$

definition *val-relation-set* **where**
$val\text{-}relation\text{-}set = \{as \in carrier\ (Q_p{}^2).\ val\ (as!1) \leq val\ (as!0)\}$

lemma *val-relation-setE*:
assumes $as \in val\text{-}relation\text{-}set$
shows $as!0 \in carrier\ Q_p \wedge as!1 \in carrier\ Q_p \wedge as = [as!0,as!1] \wedge val\ (as!1) \leq val\ (as!0)$
$\langle proof \rangle$

lemma *val-relation-setI*:
assumes $as!0 \in carrier\ Q_p$
assumes $as!1 \in carrier\ Q_p$
assumes $length\ as = 2$
assumes $val\ (as!1) \leq val(as!0)$
shows $as \in val\text{-}relation\text{-}set$
$\langle proof \rangle$

lemma *val-relation-semialg*:
$val\text{-}relation\text{-}set\ = basic\text{-}semialg\text{-}set\ 2\ (2::nat)\ Qp\text{-}val\text{-}poly$
$\langle proof \rangle$

lemma *val-relation-is-semialgebraic*:
$is\text{-}semialgebraic\ 2\ val\text{-}relation\text{-}set$
$\langle proof \rangle$

lemma *Qp-val-ring-is-semialg*:
obtains P **where** $P \in carrier\ Q_p\text{-}x \wedge \mathcal{O}_p = univ\text{-}basic\text{-}semialg\text{-}set\ 2\ P$
$\langle proof \rangle$

lemma *Qp-val-ring-is-univ-semialgebraic*:
$is\text{-}univ\text{-}semialgebraic\ \mathcal{O}_p$
$\langle proof \rangle$

lemma *Qp-val-ring-is-semialgebraic*:
$is\text{-}semialgebraic\ 1\ (to\text{-}R1`\ \mathcal{O}_p)$
$\langle proof \rangle$

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 $is\text{-}poly\text{-}tuple\ n\ fs$
assumes $length\ fs = k$

assumes $S = basic\text{-}semialg\text{-}set\ k\ m\ f$
assumes $m\ \neq 0$
shows $poly\text{-}map\ n\ fs\ \ ^{-1}{}_n\ S = basic\text{-}semialg\text{-}set\ n\ m\ (Qp\text{-}poly\text{-}comp\ n\ fs\ f)$
⟨*proof*⟩

lemma *basic-semialg-pullback′*:
assumes *is-poly-tuple n fs*
assumes $length\ fs = k$
assumes $A \in basic\text{-}semialgs\ k$
shows $poly\text{-}map\ n\ fs\ \ ^{-1}{}_n\ A \in (basic\text{-}semialgs\ n)$
⟨*proof*⟩

lemma *semialg-pullback*:
assumes *is-poly-tuple n fs*
assumes $length\ fs = k$
assumes $S \in semialg\text{-}sets\ k$
shows $poly\text{-}map\ n\ fs\ \ ^{-1}{}_n\ S \in semialg\text{-}sets\ n$
⟨*proof*⟩

lemma *pullback-is-semialg*:
assumes *is-poly-tuple n fs*
assumes $length\ fs = k$
assumes $S \in semialg\text{-}sets\ k$
shows $is\text{-}semialgebraic\ n\ (poly\text{-}map\ n\ fs\ \ ^{-1}{}_n\ S)$
⟨*proof*⟩

Equality and inequality sets for a pair of polynomials

definition *val-ineq-set* **where**
$val\text{-}ineq\text{-}set\ n\ f\ g = \{x \in carrier\ (Q_p{}^n).\ val\ (Qp\text{-}ev\ f\ x) \leq val\ (Qp\text{-}ev\ g\ x)\}$

lemma *poly-map-length* :
assumes $length\ fs = m$
assumes $as \in carrier\ (Q_p{}^n)$
shows $length\ (poly\text{-}map\ n\ fs\ as) = m$
⟨*proof*⟩

lemma *val-ineq-set-pullback*:
assumes $f \in carrier\ (Q_p[\mathcal{X}_n])$
assumes $g \in carrier\ (Q_p[\mathcal{X}_n])$
shows $val\text{-}ineq\text{-}set\ n\ f\ g = poly\text{-}map\ n\ [g,f]\ ^{-1}{}_n\ val\text{-}relation\text{-}set$
⟨*proof*⟩

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 $val\text{-}ineq\text{-}set\ n\ f\ g \in semialg\text{-}sets\ n$
⟨*proof*⟩

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)*
⟨*proof*⟩

lemma *val-ineq-setI*:
assumes $f \in carrier\ (Q_p[\mathcal{X}_n])$
assumes $g \in carrier\ (Q_p[\mathcal{X}_n])$
assumes $x \in (val\text{-}ineq\text{-}set\ n\ f\ g)$
shows $x \in carrier\ (Q_p{}^n)$
$val\ (Qp\text{-}ev\ f\ x) \leq val\ (Qp\text{-}ev\ g\ x)$
⟨*proof*⟩

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 $val\ (Qp\text{-}ev\ f\ x) \leq val\ (Qp\text{-}ev\ g\ x)$
shows $x \in (val\text{-}ineq\text{-}set\ n\ f\ g)$
⟨*proof*⟩

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* $\{x \in carrier\ (Q_p{}^n).\ val\ (Qp\text{-}ev\ f\ x) \leq val\ (Qp\text{-}ev\ g\ x)\}$
⟨*proof*⟩

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*⟩

lemma *equalityI′′*:
assumes $\bigwedge x.\ A\ x \Longrightarrow B\ x$
assumes $\bigwedge x.\ B\ x \Longrightarrow A\ x$
shows $\{x.\ A\ x\} = \{x.\ B\ x\}$
⟨*proof*⟩

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* $\{x \in carrier\ (Q_p{}^n).\ val\ (Qp\text{-}ev\ f\ x) < val\ (Qp\text{-}ev\ g\ x)\}$
⟨*proof*⟩

lemma *constant-poly-val-exists*:
shows $\exists\, g \in carrier\ (Q_p[\mathcal{X}_n]).\ (\forall\ x \in carrier\ (Q_p{}^n).\ val\ (Qp\text{-}ev\ g\ x) = c)$
⟨*proof*⟩

lemma *val-ineq-set-is-semialgebraic''*:
  assumes $f \in carrier\ (Q_p[\mathcal{X}_n])$
  shows *is-semialgebraic* $n\ \{x \in carrier\ (Q_p{}^n).\ val\ (Qp\text{-}ev\ f\ x) \leq c\}$
⟨*proof*⟩

lemma *val-ineq-set-is-semialgebraic'''*:
  assumes $f \in carrier\ (Q_p[\mathcal{X}_n])$
  shows *is-semialgebraic* $n\ \{x \in carrier\ (Q_p{}^n).\ c \leq val\ (Qp\text{-}ev\ f\ x)\}$
⟨*proof*⟩

lemma *val-eq-set-is-semialgebraic'*:
  assumes $f \in carrier\ (Q_p[\mathcal{X}_n])$
  shows *is-semialgebraic* $n\ \{x \in carrier\ (Q_p{}^n).\ val\ (Qp\text{-}ev\ f\ x) = c\}$
⟨*proof*⟩

lemma *val-strict-ineq-set-is-semialgebraic'*:
  assumes $f \in carrier\ (Q_p[\mathcal{X}_n])$
  shows *is-semialgebraic* $n\ \{x \in carrier\ (Q_p{}^n).\ val\ (Qp\text{-}ev\ f\ x) < c\}$
⟨*proof*⟩

lemma *val-strict-ineq-set-is-semialgebraic''*:
  assumes $f \in carrier\ (Q_p[\mathcal{X}_n])$
  shows *is-semialgebraic* $n\ \{x \in carrier\ (Q_p{}^n).\ c < val\ (Qp\text{-}ev\ f\ x)\}$
⟨*proof*⟩

lemma(**in** *cring*) *R1-memE*:
  assumes $x \in carrier\ (R^1)$
  shows $x = [(hd\ x)]$
  ⟨*proof*⟩

lemma(**in** *cring*) *R1-memE'*:
 assumes $x \in carrier\ (R^1)$
 shows $hd\ x \in carrier\ R$
  ⟨*proof*⟩

lemma *univ-val-ineq-set-is-univ-semialgebraic*:
is-univ-semialgebraic $\{x \in carrier\ Q_p.\ val\ x \leq c\}$
⟨*proof*⟩

lemma *univ-val-strict-ineq-set-is-univ-semialgebraic*:
is-univ-semialgebraic $\{x \in carrier\ Q_p.\ val\ x < c\}$
⟨*proof*⟩

lemma *univ-val-eq-set-is-univ-semialgebraic*:
is-univ-semialgebraic $\{x \in carrier\ Q_p.\ val\ x = c\}$
⟨*proof*⟩

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*⟩

lemma *times-p-pow-val*:
assumes $a \in carrier\ Q_p$
assumes $b = \mathfrak{p}[\urcorner]n \otimes a$
shows $val\ b = val\ a + n$
⟨*proof*⟩

lemma *times-p-pow-neg-val*:
assumes $a \in carrier\ Q_p$
assumes $b = \mathfrak{p}[\urcorner]{-n} \otimes a$
shows $val\ b = val\ a - n$
⟨*proof*⟩

lemma *eint-minus-int-pos*:
assumes $a - eint\ n \geq 0$
shows $a \geq n$
⟨*proof*⟩

p-adic balls as pullbacks of polynomial maps

lemma *balls-as-pullbacks*:
assumes $c \in carrier\ Q_p$
shows $\exists P \in carrier\ (Q_p[\mathcal{X}_1]).\ to\text{-}R1\text{`}\ B_n[c] = poly\text{-}map\ 1\ [P]\ \ ^{-1}{}_1\ (to\text{-}R1\ \text{`}\ \mathcal{O}_p)$
⟨*proof*⟩

lemma *ball-is-semialgebraic*:
assumes $c \in carrier\ Q_p$
shows $is\text{-}semialgebraic\ 1\ (to\text{-}R1\text{`}\ B_n[c])$
⟨*proof*⟩

lemma *ball-is-univ-semialgebraic*:
assumes $c \in carrier\ Q_p$
shows $is\text{-}univ\text{-}semialgebraic\ (B_n[c])$
⟨*proof*⟩

abbreviation *Qp-to-R1-set* **where**
$Qp\text{-}to\text{-}R1\text{-}set\ S \equiv to\text{-}R1\ \text{`}\ S$

13.5.8 Finite Unions and Intersections of Semialgebraic Sets

definition *are-semialgebraic* **where**
$are\text{-}semialgebraic\ n\ Xs = (\forall\ x.\ x \in Xs \longrightarrow is\text{-}semialgebraic\ n\ x)$

lemma *are-semialgebraicI*:
assumes $\bigwedge x.\ x \in Xs \Longrightarrow is\text{-}semialgebraic\ n\ x$
shows $are\text{-}semialgebraic\ n\ Xs$

$\langle proof \rangle$

lemma *are-semialgebraicE*:
  assumes *are-semialgebraic n Xs*
  assumes $x \in Xs$
  shows *is-semialgebraic n x*
  $\langle proof \rangle$

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*
  $\langle proof \rangle$

lemma *are-univ-semialgebraicE*:
  assumes *are-univ-semialgebraic Xs*
  assumes $x \in Xs$
  shows *is-univ-semialgebraic x*
  $\langle proof \rangle$

lemma *are-univ-semialgebraic-semialgebraic*:
  assumes *are-univ-semialgebraic Xs*
  shows *are-semialgebraic 1* (*Qp-to-R1-set* ‘ *Xs*)
  $\langle proof \rangle$

lemma *to-R1-set-union*:
to-R1 ‘ ($\bigcup$ *Xs*) = $\bigcup$ (*Qp-to-R1-set* ‘ *Xs*)
  $\langle proof \rangle$

lemma *to-R1-inter*:
  assumes $Xs \neq \{\}$
  shows *to-R1* ‘ ($\bigcap$ *Xs*) = $\bigcap$ (*Qp-to-R1-set* ‘ *Xs*)
$\langle proof \rangle$

lemma *finite-union-is-semialgebraic*:
  assumes *finite Xs*
  shows $Xs \subseteq$ *semialg-sets n* $\longrightarrow$ *is-semialgebraic n* ($\bigcup$ *Xs*)
  $\langle proof \rangle$

lemma *finite-union-is-semialgebraic'*:
  assumes *finite Xs*
  assumes $Xs \subseteq$ *semialg-sets n*
  shows *is-semialgebraic n* ($\bigcup$ *Xs*)
  $\langle proof \rangle$

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 $(\bigcup\ x \in S.\ F\ x)$
⟨*proof*⟩

lemma *finite-union-is-univ-semialgebraic′*:
assumes *finite Xs*
assumes *are-univ-semialgebraic Xs*
shows *is-univ-semialgebraic* $(\bigcup\ Xs)$
⟨*proof*⟩

lemma *finite-intersection-is-semialgebraic*:
assumes *finite Xs*
shows $Xs \subseteq$ *semialg-sets* $n \wedge Xs \neq \{\} \longrightarrow$ *is-semialgebraic* n $(\bigcap\ Xs)$
⟨*proof*⟩

lemma *finite-intersection-is-semialgebraic′*:
assumes *finite Xs*
assumes $Xs \subseteq$ *semialg-sets* $n \wedge Xs \neq \{\}$
shows *is-semialgebraic* n $(\bigcap\ Xs)$
⟨*proof*⟩

lemma *finite-intersection-is-semialgebraic′′*:
assumes *finite Xs*
assumes *are-semialgebraic* n $Xs \wedge Xs \neq \{\}$
shows *is-semialgebraic* n $(\bigcap\ Xs)$
⟨*proof*⟩

lemma *finite-intersection-is-univ-semialgebraic*:
assumes *finite Xs*
assumes *are-univ-semialgebraic Xs*
assumes $Xs \neq \{\}$
shows *is-univ-semialgebraic* $(\bigcap\ Xs)$
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

lemma *cartesian-product-memE'*:
assumes $x \in$ *cartesian-product A B*
obtains *a b* **where** $a \in A \wedge b \in B \wedge x = a@b$
⟨*proof*⟩

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*⟩

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*))
⟨*proof*⟩

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*⟩

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*⟩

lemma *product-of-basic-semialgs-is-semialg'*:
assumes $A \in$ (*basic-semialgs n*)
assumes $B \in$ (*basic-semialgs m*)
shows *is-semialgebraic* $(n + m)$ (*cartesian-product A B*)
⟨*proof*⟩

lemma *car-times-semialg-is-semialg*:
assumes *is-semialgebraic m B*
shows *is-semialgebraic* $(n + m)$ (*cartesian-product* (*carrier* $(Q_p{}^n)$) *B*)
⟨*proof*⟩

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*)
⟨*proof*⟩

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*)
⟨*proof*⟩

13.7 N^{th} Power Residues

definition *nth-root-poly* **where**
nth-root-poly $(n::nat)$ a = ((*X-poly* Q_p) $[^\frown]_{Q_p\text{-}x}$ n) $\ominus_{Q_p\text{-}x}$ (*to-poly a*)

lemma *nth-root-poly-closed*:
assumes $a \in$ *carrier* Q_p
shows *nth-root-poly n a* $\in$ *carrier* Q_p-x
⟨*proof*⟩

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 = $(b[^\frown]n) \ominus_{Q_p} a$
⟨*proof*⟩

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 $(\exists\ b \in \mathcal{O}_p.\ ((b[^\frown]n) = a))$
⟨*proof*⟩

All points sufficiently close to 1 have nth roots

lemma *eint-nat-times-2*:
$2*(n::nat) = 2*$*eint n*
⟨*proof*⟩

lemma *P-set-of-one*:
P-set 1 = *nonzero* Q_p
⟨*proof*⟩

lemma *nth-power-fact*:
assumes $(n::nat) \geq 1$

shows $\exists\ (m{::}nat) > 0.\ \forall\ u \in carrier\ Q_p.\ ac\ m\ u = 1 \wedge val\ u = 0 \longrightarrow u \in P\text{-}set\ n$
$\langle proof\rangle$

definition *pow-res* **where**
$pow\text{-}res\ (n{::}nat)\ x = \{a.\ a \in carrier\ Q_p \wedge\ (\exists\, y \in nonzero\ Q_p.\ (a = x \otimes (y[\urcorner]n)))\}$

lemma *nonzero-pow-res*:
assumes $x \in nonzero\ Q_p$
shows $pow\text{-}res\ (n{::}nat)\ x \subseteq nonzero\ Q_p$
$\langle proof\rangle$

lemma *pow-res-of-zero*:
shows $pow\text{-}res\ n\ \mathbf{0} = \{\mathbf{0}\}$
$\langle proof\rangle$

lemma *equal-pow-resI*:
assumes $x \in carrier\ Q_p$
assumes $y \in pow\text{-}res\ n\ x$
shows $pow\text{-}res\ n\ x = pow\text{-}res\ n\ y$
$\langle proof\rangle$

lemma *zeroth-pow-res*:
assumes $x \in carrier\ Q_p$
shows $pow\text{-}res\ 0\ x = \{x\}$
$\langle proof\rangle$

lemma *Zp-car-zero-res*: **assumes** $x \in carrier\ Z_p$
shows $x\ 0 = 0$
$\langle proof\rangle$

lemma *zeroth-ac*:
assumes $x \in carrier\ Q_p$
shows $ac\ 0\ x = 0$
$\langle proof\rangle$

lemma *nonzero-ac-imp-nonzero*:
assumes $x \in carrier\ Q_p$
assumes $ac\ m\ x \neq 0$
shows $x \in nonzero\ Q_p$
$\langle proof\rangle$

lemma *nonzero-ac-val-ord*:
assumes $x \in carrier\ Q_p$
assumes $ac\ m\ x \neq 0$
shows $val\ x = ord\ x$
$\langle proof\rangle$

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*⟩

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*⟩

definition *pow-res-classes* **where**
$pow\text{-}res\text{-}classes\ n = \{S. \exists x \in nonzero\ Q_p.\ S = pow\text{-}res\ n\ x\ \}$

lemma *pow-res-semialg-def*:
assumes $x \in nonzero\ Q_p$
assumes $n \geq 1$
shows $\exists P \in carrier\ Q_p\text{-}x.\ pow\text{-}res\ n\ x = (univ\text{-}basic\text{-}semialg\text{-}set\ n\ P) - \{\mathbf{0}\}$
⟨*proof*⟩

lemma *pow-res-is-univ-semialgebraic*:
assumes $x \in carrier\ Q_p$
shows $is\text{-}univ\text{-}semialgebraic\ (pow\text{-}res\ n\ x)$
⟨*proof*⟩

lemma *pow-res-is-semialg*:
assumes $x \in carrier\ Q_p$
shows $is\text{-}semialgebraic\ 1\ (to\text{-}R1\ `\ (pow\text{-}res\ n\ x))$
⟨*proof*⟩

lemma *pow-res-refl*:
assumes $x \in carrier\ Q_p$
shows $x \in pow\text{-}res\ n\ x$
⟨*proof*⟩

lemma *equal-pow-resE*:
assumes $a \in carrier\ Q_p$
assumes $b \in carrier\ Q_p$
assumes $n > 0$
assumes $pow\text{-}res\ n\ a = pow\text{-}res\ n\ b$
shows $\exists\ s \in P\text{-}set\ n.\ a = b \otimes s$
⟨*proof*⟩

lemma *pow-res-one*:
assumes $x \in nonzero\ Q_p$
shows $pow\text{-}res\ 1\ x = nonzero\ Q_p$
⟨*proof*⟩

lemma *pow-res-zero*:
  assumes $n > 0$
  shows *pow-res* n $\mathbf{0} = \{\mathbf{0}\}$
⟨*proof*⟩

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*⟩

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-set* n
  shows *pow-res* n a = *pow-res* n b
  ⟨*proof*⟩

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*⟩

lemma *equal-pow-resI''''*:
  assumes $n > 0$
  assumes $a \in$ *carrier* Q_p
  assumes $b \in$ *carrier* Q_p
  assumes $a = b \otimes u$
  assumes $u \in$ *P-set* n
  shows *pow-res* n a = *pow-res* n b
⟨*proof*⟩

lemma *Zp-Units-ord-zero*:
  assumes $a \in$ *Units* Z_p
  shows *ord-Zp* $a = 0$
⟨*proof*⟩

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*⟩

lemma *pow-res-of-p-pow*:
assumes $n > 0$
shows *pow-res n* $(\mathfrak{p}[\widehat{\ }]((l::int)*n))$ = *pow-res n* **1**
⟨*proof*⟩

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
⟨*proof*⟩

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*⟩

lemma *pow-res-p-pow-factor*:
assumes $n > 0$
assumes $a \in$ *carrier* Q_p
shows *pow-res n a* = *pow-res n* $(\mathfrak{p}[\widehat{\ }]((l::int)*n) \otimes a)$
⟨*proof*⟩

lemma *pow-res-classes-finite*:
assumes $n \geq 1$
shows *finite* (*pow-res-classes n*)
⟨*proof*⟩

lemma *pow-res-classes-univ-semialg*:
assumes $S \in$ *pow-res-classes n*
shows *is-univ-semialgebraic S*
⟨*proof*⟩

lemma *pow-res-classes-semialg*:
assumes $S \in$ *pow-res-classes n*
shows *is-semialgebraic 1* (*to-R1‘ S*)
⟨*proof*⟩

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*⟩

lemma *nth-pow-wits-covers*:
assumes $n > 0$
assumes $x \in nonzero\ Q_p$
shows $\exists\, y \in (nth\text{-}pow\text{-}wits\ n).\ y \in nonzero\ Q_p \wedge y \in \mathcal{O}_p \wedge x \in pow\text{-}res\ n\ y$
⟨*proof*⟩

lemma *nth-pow-wits-closed*:
assumes $n > 0$
assumes $x \in nth\text{-}pow\text{-}wits\ n$
shows $x \in carrier\ Q_p\ \ x \in \mathcal{O}_p\ \ x \in nonzero\ Q_p\ \ \exists\ y \in pow\text{-}res\text{-}classes\ n.\ y = pow\text{-}res\ n\ x$
⟨*proof*⟩

lemma *finite-extensional-funcset*:
assumes $finite\ A$
assumes $finite\ (B::'b\ set)$
shows $finite\ (A \rightarrow_E B)$
⟨*proof*⟩

lemma *nth-pow-wits-exists*:
assumes $m > 0$
assumes $c \in pow\text{-}res\text{-}classes\ m$
shows $\exists\, x.\ x \in c \cap \mathcal{O}_p$
⟨*proof*⟩

lemma *pow-res-classes-mem-eq*:
assumes $m > 0$
assumes $a \in pow\text{-}res\text{-}classes\ m$
assumes $x \in a$
shows $a = pow\text{-}res\ m\ x$
⟨*proof*⟩

lemma *nth-pow-wits-neq-pow-res*:
assumes $m > 0$
assumes $x \in nth\text{-}pow\text{-}wits\ m$
assumes $y \in nth\text{-}pow\text{-}wits\ m$
assumes $x \neq y$
shows $pow\text{-}res\ m\ x \neq pow\text{-}res\ m\ y$
⟨*proof*⟩

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* $m\ y = \{\}$
⟨*proof*⟩

lemma *nth-power-fact'*:
assumes $0 < (n::nat)$
shows $\exists\, m>0.\ \forall\, u\in carrier\ Q_p.\ ac\ m\ u = 1 \wedge val\ u = 0 \longrightarrow u \in P\text{-}set\ n$
⟨*proof*⟩

lemma *equal-pow-res-criterion*:
assumes $N > 0$
assumes $n > 0$
assumes $\forall\ u \in carrier\ Q_p.\ ac\ N\ u = 1 \wedge val\ u = 0 \longrightarrow u \in P\text{-}set\ n$
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*⟩

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* $n\ (a[\urcorner](k::nat)) =$ *pow-res* $n\ (b[\urcorner]k)$
⟨*proof*⟩

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*⟩

lemma *pow-res-disjoint*:

assumes $n > 0$
assumes $a \in \mathit{nonzero}\ Q_p$
assumes $a \notin \mathit{pow\text{-}res}\ n\ \mathbf{1}$
shows $\neg\ (\exists\, y \in \mathit{nonzero}\ Q_p.\ a = y[\urcorner]n)$
$\langle \mathit{proof} \rangle$

lemma *pow-res-disjoint'*:
assumes $n > 0$
assumes $a \in \mathit{nonzero}\ Q_p$
assumes $\mathit{pow\text{-}res}\ n\ a \neq \mathit{pow\text{-}res}\ n\ \mathbf{1}$
shows $\neg\ (\exists\, y \in \mathit{nonzero}\ Q_p.\ a = y[\urcorner]n)$
$\langle \mathit{proof} \rangle$

lemma *pow-res-one-imp-nth-pow*:
assumes $n > 0$
assumes $a \in \mathit{pow\text{-}res}\ n\ \mathbf{1}$
shows $\exists\, y \in \mathit{nonzero}\ Q_p.\ a = y[\urcorner]n$
$\langle \mathit{proof} \rangle$

lemma *pow-res-eq*:
assumes $n > 0$
assumes $a \in \mathit{carrier}\ Q_p$
assumes $b \in \mathit{pow\text{-}res}\ n\ a$
shows $\mathit{pow\text{-}res}\ n\ b = \mathit{pow\text{-}res}\ n\ a$
$\langle \mathit{proof} \rangle$

lemma *pow-res-classes-n-eq-one*:
$\mathit{pow\text{-}res\text{-}classes}\ 1 = \{\mathit{nonzero}\ Q_p\}$
$\langle \mathit{proof} \rangle$

lemma *nth-pow-wits-closed'*:
assumes $n > 0$
assumes $x \in \mathit{nth\text{-}pow\text{-}wits}\ n$
shows $x \in \mathcal{O}_p \wedge x \in \mathit{nonzero}\ Q_p$ $\langle \mathit{proof} \rangle$

13.8 Semialgebraic Sets Defined by Congruences

13.8.1 p-adic ord Congruence Sets

lemma *carrier-is-univ-semialgebraic*:
$\mathit{is\text{-}univ\text{-}semialgebraic}\ (\mathit{carrier}\ Q_p)$
$\langle \mathit{proof} \rangle$

lemma *nonzero-is-univ-semialgebraic*:
$\mathit{is\text{-}univ\text{-}semialgebraic}\ (\mathit{nonzero}\ Q_p)$
$\langle \mathit{proof} \rangle$

definition *ord-congruence-set* **where**
$\mathit{ord\text{-}congruence\text{-}set}\ n\ a = \{x \in \mathit{nonzero}\ Q_p.\ \mathit{ord}\ x\ \mathit{mod}\ n = a\}$

lemma *ord-congruence-set-nonzero*:
ord-congruence-set n $a \subseteq$ *nonzero* Q_p
⟨*proof*⟩

lemma *ord-congruence-set-closed*:
ord-congruence-set n $a \subseteq$ *carrier* Q_p
⟨*proof*⟩

lemma *ord-congruence-set-memE*:
assumes $x \in$ *ord-congruence-set* n a
shows $x \in$ *nonzero* Q_p
ord x *mod* $n = a$
⟨*proof*⟩

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
⟨*proof*⟩

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*⟩

lemma *pow-res-classes-are-univ-semialgebraic*:
shows *are-univ-semialgebraic* (*pow-res-classes* n)
⟨*proof*⟩

lemma *ord-congruence-set-univ-semialg*:
assumes $n \geq 0$
shows *is-univ-semialgebraic* (*ord-congruence-set* n a)
⟨*proof*⟩

lemma *ord-congruence-set-is-semialg*:
assumes $n \geq 0$
shows *is-semialgebraic* 1 (*Qp-to-R1-set* (*ord-congruence-set* n a))
⟨*proof*⟩

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 $[f]$ $x = [(Qp\text{-}ev\ f\ x)]$

⟨*proof*⟩

definition *poly-cong-set* **where**
poly-cong-set $n\ f\ m\ a = \{x \in carrier\ (Q_p{}^n).\ (Qp\text{-}ev\ f\ x) \neq \mathbf{0}\ \wedge (ord\ (Qp\text{-}ev\ f\ x)\ mod\ m = a)\}$

lemma *poly-cong-set-as-pullback*:
assumes $f \in carrier\ (Q_p[\mathcal{X}_n])$
shows *poly-cong-set* $n\ f\ m\ a = poly\text{-}map\ n\ [f]\ ^{-1}{}_n(Qp\text{-}to\text{-}R1\text{-}set\ (ord\text{-}congruence\text{-}set\ m\ a))$
⟨*proof*⟩

lemma *singleton-poly-tuple*:
is-poly-tuple $n\ [f] \longleftrightarrow f \in carrier\ (Q_p[\mathcal{X}_n])$
⟨*proof*⟩

lemma *poly-cong-set-is-semialgebraic*:
assumes $m \geq 0$
assumes $f \in carrier\ (Q_p[\mathcal{X}_n])$
shows *is-semialgebraic* n (*poly-cong-set* $n\ f\ m\ a$)
⟨*proof*⟩

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 $n \geq 1$
assumes $S \subseteq nonzero\ Q_p$
assumes $\bigwedge x.\ x \in S \Longrightarrow pow\text{-}res\ n\ x \subseteq S$
shows *is-univ-semialgebraic* S
⟨*proof*⟩

definition *ac-cong-set1* **where**
ac-cong-set1 $n\ y = \{x \in carrier\ Q_p.\ x \neq \mathbf{0} \wedge ac\ n\ x = ac\ n\ y\}$

lemma *ac-cong-set1-is-univ-semialg*:
assumes $n > 0$
assumes $b \in nonzero\ Q_p$
assumes $b \in \mathcal{O}_p$
shows *is-univ-semialgebraic* (*ac-cong-set1* $n\ b$)
⟨*proof*⟩

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 $k \in Units\ (Zp\text{-}res\text{-}ring\ n)$
shows *is-univ-semialgebraic* (*ac-cong-set* $n\ k$)

⟨proof⟩

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* ∈ *carrier* Q_p
  assumes *val a* = *eint k*
  shows *a* ∈ *nonzero* Q_p
  ⟨proof⟩

lemma *val-ord′*:
  assumes *a* ∈ *carrier* Q_p
  assumes *a* ≠**0**
  shows *val a* = *ord a*
  ⟨proof⟩

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⟩

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⟩

definition *ac-cong-set3* **where**
ac-cong-set3 n = {*as*. ∃ *a b*. *a* ∈ *nonzero* Q_p ∧ *b* ∈ $\mathcal{O}_p$ ∧ *val b* = *0* ∧ (*ac n a* = *ac n b*) ∧ *as* = [*a*, *b*] }

definition *ac-cong-set2* **where**
ac-cong-set2 n k = {*as*. ∃ *a b*. *a* ∈ *nonzero* Q_p ∧ *b* ∈ $\mathcal{O}_p$ ∧ *val b* = *0* ∧ (*ac n a* = *k*) ∧ (*ac n b*) = *k* ∧ *as* = [*a*, *b*] }

lemma *ac-cong-set2-cartesian-product*:
  assumes *k* ∈ *Units* (*Zp-res-ring n*)
  assumes *n* > *0*
  shows *ac-cong-set2 n k* = *cartesian-product* (*to-R1*‘ (*ac-cong-set n k*)) (*to-R1*‘ (*val-ring-constant-ac-set n k*))
⟨proof⟩

lemma *ac-cong-set2-is-semialg*:
  assumes *k* ∈ *Units* (*Zp-res-ring n*)
  assumes *n* > *0*

shows *is-semialgebraic 2* (*ac-cong-set2 n k*)
⟨*proof*⟩

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*⟩

lemma *ac-cong-set3-is-semialgebraic*:
assumes $n > 0$
shows *is-semialgebraic 2* (*ac-cong-set3 n*)
⟨*proof*⟩

13.9 Permutations of indices of semialgebraic sets

lemma *fun-inv-permute*:
assumes σ *permutes* $\{..<n\}$
shows *fun-inv* σ *permutes* $\{..<n\}$
$\sigma \circ (\textit{fun-inv}\ \sigma) = \textit{id}$
$(\textit{fun-inv}\ \sigma) \circ \sigma = \textit{id}$
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *permutation-is-semialgebraic*:
assumes *is-semialgebraic n S*
assumes σ *permutes* $\{..<n\}$
shows *is-semialgebraic n* (*permute-list* σ ‘ S)
⟨*proof*⟩

lemma *permute-list-closed*:
assumes $a \in$ *carrier* $({Q_p}^n)$
assumes σ *permutes* $\{..<n\}$
shows *permute-list* σ $a \in$ *carrier* $({Q_p}^n)$
⟨*proof*⟩

lemma *permute-list-closed′*:
assumes σ *permutes* $\{..<n\}$
assumes *permute-list* σ $a \in$ *carrier* $({Q_p}^n)$
shows $a \in$ *carrier* $({Q_p}^n)$
⟨*proof*⟩

lemma *permute-list-compose-inv*:
assumes σ *permutes* $\{..<n\}$

assumes $a \in carrier\ (Q_p{}^n)$
shows $permute\text{-}list\ \sigma\ (permute\text{-}list\ (fun\text{-}inv\ \sigma)\ a) = a$
$permute\text{-}list\ (fun\text{-}inv\ \sigma)\ (permute\text{-}list\ \sigma\ a) = a$
⟨*proof*⟩

lemma *permutation-is-semialgebraic-imp-is-semialgebraic*:
assumes *is-semialgebraic* n $(permute\text{-}list\ \sigma$ ‘ $S)$
assumes σ *permutes* $\{..<n\}$
shows *is-semialgebraic* n S
⟨*proof*⟩

lemma *split-cartesian-product-is-semialgebraic*:
assumes $i \leq n$
assumes *is-semialgebraic* n A
assumes *is-semialgebraic* m B
shows *is-semialgebraic* $(n + m)$ $(split\text{-}cartesian\text{-}product\ n\ m\ i\ A\ B)$
⟨*proof*⟩

definition *reverse-val-relation-set* **where**
$reverse\text{-}val\text{-}relation\text{-}set = \{as \in carrier\ (Q_p{}^2).\ val\ (as\ !\ 0) \leq val\ (as\ !\ 1)\}$

lemma *Qp-2-car-memE*:
assumes $x \in carrier\ (Q_p{}^2)$
shows $x = [x!0,\ x!1]$
⟨*proof*⟩

definition *flip* **where**
$flip = (\lambda i{::}nat.\ (if\ i = 0\ then\ 1\ else\ (if\ i = 1\ then\ 0\ else\ i)))$

lemma *flip-permutes*:
flip permutes $\{0,1\}$
⟨*proof*⟩

lemma *flip-eval*:
$flip\ 0 = 1$
$flip\ 1 = 0$
⟨*proof*⟩

lemma *flip-x*:
assumes $x \in carrier\ (Q_p{}^2)$
shows $permute\text{-}list\ flip\ x = [x!1,\ x!0]$
⟨*proof*⟩

lemma *permute-with-flip-closed*:
assumes $x \in carrier\ (Q_p{}^{2::nat})$
shows $permute\text{-}list\ flip\ x \in carrier\ (Q_p{}^{2::nat})$
⟨*proof*⟩

lemma *reverse-val-relation-set-semialg*:

is-semialgebraic 2 reverse-val-relation-set
⟨*proof*⟩

definition *strict-val-relation-set* **where**
strict-val-relation-set = $\{as \in carrier\ (Q_p{}^2).\ val\ (as\ !\ 0) < val\ (as\ !\ 1)\}$

definition *val-diag* **where**
val-diag = $\{as \in carrier\ (Q_p{}^2).\ val\ (as\ !\ 0) = val\ (as\ !\ 1)\}$

lemma *val-diag-semialg*:
is-semialgebraic 2 val-diag
⟨*proof*⟩

lemma *strict-val-relation-set-is-semialg*:
is-semialgebraic 2 strict-val-relation-set
⟨*proof*⟩

lemma *singleton-length*:
length $[a] = 1$
⟨*proof*⟩

lemma *take-closed'*:
assumes $m > 0$
assumes $x \in carrier\ (Q_p{}^{m+l})$
shows $take\ m\ x \in carrier\ (Q_p{}^m)$
⟨*proof*⟩

lemma *triple-val-ineq-set-semialg*:
shows *is-semialgebraic 3* $\{as \in carrier\ (Q_p{}^3).\ val\ (as!0) \leq val\ (as!1) \wedge val\ (as!1) \leq val\ (as!2)\}$
⟨*proof*⟩

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*⟩

lemma *triple-val-ineq-set-semialg''*:
shows *is-semialgebraic 3* $\{as \in carrier\ (Q_p{}^3).\ val\ (as!1) < val\ (as!2)\}$
⟨*proof*⟩

lemma *triple-val-ineq-set-semialg'''*:
shows *is-semialgebraic 3* $\{as \in carrier\ (Q_p{}^3).\ val\ (as!1) \leq val\ (as!2)\}$
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *partial-pullback-closed*:
partial-pullback m f l S ⊆ *carrier* ($Q_p{}^{m\ +\ l}$)
⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

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*
⟨*proof*⟩

Partial pullbacks have the same algebraic properties as pullbacks

lemma *partial-pullback-intersect*:

partial-pullback m f l $(S1 \cap S2) = ($*partial-pullback m f l S1*$) \cap ($*partial-pullback m f l S2*$)$
⟨*proof*⟩

lemma *partial-pullback-union*:
partial-pullback m f l $(S1 \cup S2) = ($*partial-pullback m f l S1*$) \cup ($*partial-pullback m f l S2*$)$
⟨*proof*⟩

lemma *cartesian-power-drop*:
assumes $x \in$ *carrier* $(Q_p{}^{n+l})$
shows *drop n x* $\in$ *carrier* $(Q_p{}^{l})$
⟨*proof*⟩

lemma *partial-pullback-complement*:
assumes $f \in$ *carrier* $(Q_p{}^{m}) \to$ *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*)
⟨*proof*⟩

lemma *partial-pullback-carrier*:
assumes $f \in$ *carrier* $(Q_p{}^{m}) \to$ *carrier* Q_p
shows *partial-pullback m f l* (*carrier* $(Q_p{}^{Suc\ l})) =$ *carrier* $(Q_p{}^{m\ +\ l})$
⟨*proof*⟩

Definition 1.4 from Denef

definition *is-semialg-function* **where**
is-semialg-function m f = $((f \in$ *carrier* $(Q_p{}^{m}) \to$ *carrier* $Q_p) \wedge$
$(\forall\, l \geq 0.\ \forall\, S \in$ *semialg-sets* $(1 + l)$. *is-semialgebraic* $(m + l)$ (*partial-pullback m f l S*)))

lemma *is-semialg-function-closed*:
assumes *is-semialg-function m f*
shows $f \in$ *carrier* $(Q_p{}^{m}) \to$ *carrier* Q_p
⟨*proof*⟩

lemma *is-semialg-functionE*:
assumes *is-semialg-function m f*
assumes *is-semialgebraic* $(1 + k)$ S
shows *is-semialgebraic* $(m + k)$ (*partial-pullback m f k S*)
⟨*proof*⟩

lemma *is-semialg-functionI*:
assumes $f \in$ *carrier* $(Q_p{}^{m}) \to$ *carrier* Q_p
assumes $\bigwedge k\ S.\ S \in$ *semialg-sets* $(1 + k) \Longrightarrow$ *is-semialgebraic* $(m + k)$ (*partial-pullback m f k S*)
shows *is-semialg-function m f*
⟨*proof*⟩

Semialgebraicity for functions can be verified on basic semialgebraic sets

lemma *is-semialg-functionI'*:
  assumes $f \in carrier\ (Q_p{}^m) \to carrier\ Q_p$
  assumes $\bigwedge k\ \ S.\ S \in basic\text{-}semialgs\ (1\ +\ k) \Longrightarrow is\text{-}semialgebraic\ (m\ +\ k)$
$(partial\text{-}pullback\ m\ f\ k\ S)$
  shows *is-semialg-function m f*
  ⟨*proof*⟩

Graphs of semialgebraic functions are semialgebraic

abbreviation *graph* **where**
$graph \equiv fun\text{-}graph\ Q_p$

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*⟩

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*⟩

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*⟩

lemma *graph-closed*:
  assumes $f \in carrier\ (Q_p{}^m) \to carrier\ Q_p$
  shows $graph\ m\ f \subseteq carrier\ (Q_p{}^{m+1})$
  ⟨*proof*⟩

The m-dimensional diagonal set is semialgebraic

notation *diagonal* (‹Δ ›)

lemma *diag-is-algebraic*:
  shows $is\text{-}algebraic\ Q_p\ (n\ +\ n)\ (\Delta\ n)$
  ⟨*proof*⟩

lemma *diag-is-semialgebraic*:
  shows $is\text{-}semialgebraic\ (n\ +\ n)\ (\Delta\ n)$
  ⟨*proof*⟩

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*}
⟨*proof*⟩

lemma *transpose-alt-def*:
transpose a b x = (*if x* = *a then b else if x* = *b then a else x*)
⟨*proof*⟩

definition *last-to-first* **where**
last-to-first n = (λ*i*. *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*}
⟨*proof*⟩

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*⟩

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*⟩

lemma *graph-swapI*:
assumes *as* ∈ *carrier* ($Q_p{}^n$)
assumes *f* ∈ *carrier* ($Q_p{}^n$) → *carrier* Q_p
shows (*f as*)#*as* ∈ *graph-swap n f*
⟨*proof*⟩

lemma *graph-swapE*:
assumes *x* ∈ *graph-swap n f*
assumes *f* ∈ *carrier* ($Q_p{}^n$) → *carrier* Q_p
shows *hd x* = *f* (*tl x*)

$\langle proof \rangle$

Semialgebraic functions have semialgebraic graphs

lemma *graph-as-partial-pullback*:
  assumes $f \in carrier\ (Q_p{}^n) \to carrier\ Q_p$
  shows $partial\text{-}pullback\ n\ f\ 1\ (\Delta\ 1) = graph\ n\ f$
$\langle proof \rangle$

lemma *semialg-graph*:
  assumes $is\text{-}semialg\text{-}function\ n\ f$
  shows $is\text{-}semialgebraic\ (n + 1)\ (graph\ n\ f)$
  $\langle proof \rangle$

Functions induced by polynomials are semialgebraic

definition *var-list-segment* **where**
$var\text{-}list\text{-}segment\ i\ j = map\ (\lambda i.\ pvar\ Q_p\ i)\ [i..<\ j]$

lemma *var-list-segment-length*:
  assumes $i \leq j$
  shows $length\ (var\text{-}list\text{-}segment\ i\ j) = j - i$
  $\langle proof \rangle$

lemma *var-list-segment-entry*:
  assumes $k < j - i$
  assumes $i \leq j$
  shows $var\text{-}list\text{-}segment\ i\ j\ !\ k = pvar\ Q_p\ (i + k)$
  $\langle proof \rangle$

lemma *var-list-segment-is-poly-tuple*:
  assumes $i \leq j$
  assumes $j \leq n$
  shows $is\text{-}poly\text{-}tuple\ n\ (var\text{-}list\text{-}segment\ i\ j)$
  $\langle proof \rangle$

lemma *map-by-var-list-segment*:
  assumes $as \in carrier\ (Q_p{}^n)$
  assumes $j \leq n$
  assumes $i \leq j$
  shows $poly\text{-}map\ n\ (var\text{-}list\text{-}segment\ i\ j)\ as = list\text{-}segment\ i\ j\ as$
  $\langle proof \rangle$

lemma *map-by-var-list-segment-to-length*:
  assumes $as \in carrier\ (Q_p{}^n)$
  assumes $i \leq n$
  shows $poly\text{-}map\ n\ (var\text{-}list\text{-}segment\ i\ n)\ as = drop\ i\ as$
  $\langle proof \rangle$

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-list-segment* 1 $(n+1)$) $(a\#as) = as$
⟨*proof*⟩

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 $(f\#fs)$
⟨*proof*⟩

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-ev f a*)#*poly-map n fs a*
⟨*proof*⟩

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-map n fs a* @ *poly-map n gs a*
⟨*proof*⟩

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-ev* f) k S = *poly-tuple-pullback* $(n+k)$ S $(f\#$ (*var-list-segment* n $(n+k)$))
⟨*proof*⟩

lemma *poly-is-semialg*:
assumes $f \in carrier\ (Q_p[\mathcal{X}_n])$
shows *is-semialg-function n* (*Qp-ev f*)
⟨*proof*⟩

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*⟩

lemma *semialg-function-on-carrier'*:
assumes *is-semialg-function n f*
assumes $\bigwedge a.\ a \in carrier\ (Q_p{}^n) \Longrightarrow f\ a = g\ a$
shows *is-semialg-function n g*

⟨*proof*⟩

lemma *constant-function-is-semialg*:
assumes $n > 0$
assumes $x \in carrier\ Q_p$
assumes $\bigwedge a.\ a \in carrier\ (Q_p{}^n) \Longrightarrow f\ a = x$
shows *is-semialg-function n f*
⟨*proof*⟩

lemma *cartesian-product-singleton-factor-projection-is-semialg*:
assumes $A \subseteq carrier\ (Q_p{}^m)$
assumes $b \in carrier\ (Q_p{}^n)$
assumes *is-semialgebraic* $(m+n)$ (*cartesian-product* A $\{b\}$)
shows *is-semialgebraic m A*
⟨*proof*⟩

lemma *cartesian-product-factor-projection-is-semialg*:
assumes $A \subseteq carrier\ (Q_p{}^m)$
assumes $B \subseteq carrier\ (Q_p{}^n)$
assumes $B \neq \{\}$
assumes *is-semialgebraic* $(m+n)$ (*cartesian-product* A B)
shows *is-semialgebraic m A*
⟨*proof*⟩

lemma *partial-pullback-cartesian-product*:
assumes $\xi \in carrier\ (Q_p{}^m) \rightarrow carrier\ Q_p$
assumes $S \subseteq carrier\ (Q_p{}^1)$
shows *cartesian-product* (*partial-pullback* $m\ \xi\ 0\ S$) (*carrier* $(Q_p{}^1)$) = *partial-pullback* $m\ \xi\ 1$ (*cartesian-product* S (*carrier* $(Q_p{}^1)$))
⟨*proof*⟩

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*⟩

lemma *Qp-zero-subset-is-semialg*:
assumes $S \subseteq carrier\ (Q_p{}^0)$
shows *is-semialgebraic 0 S*
⟨*proof*⟩

lemma *cartesian-product-empty-list*:
cartesian-product A $\{[]\} = A$
cartesian-product $\{[]\}$ $A = A$
⟨*proof*⟩

lemma *cartesian-product-singleton-factor-projection-is-semialg′*:

assumes $A \subseteq$ *carrier* $(Q_p{}^m)$
assumes $b \in$ *carrier* $(Q_p{}^n)$
assumes *is-semialgebraic* $(m+n)$ (*cartesian-product* A $\{b\}$)
shows *is-semialgebraic* m A
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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 $(take\ n\ as) = as!n\}$
⟨*proof*⟩

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* $(Q_p{}^{Suc\ n})$
⟨*proof*⟩

lemma *restricted-graph-memE*:

assumes $a \in$ *restricted-graph n g S*
shows $a \in$ *carrier* $(Q_p{}^{Suc\ n})$ *take n a* $\in S$ *g* (*take n a*) $= a!n$
⟨*proof*⟩

lemma *restricted-graph-mem-formula*:
assumes $a \in$ *restricted-graph n g S*
shows $a =$ (*take n a*)@[*g* (*take n a*)]
⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

lemma *take-closed*:
assumes $n \leq k$

assumes $x \in carrier\ (Q_p{}^k)$
shows $take\ n\ x \in carrier\ (Q_p{}^n)$
⟨*proof*⟩

lemma *take-compose-closed*:
assumes $g \in carrier\ (Q_p{}^n) \to carrier\ Q_p$
assumes $n < k$
shows $g \circ take\ n \in carrier\ (Q_p{}^k) \to carrier\ Q_p$
⟨*proof*⟩

lemma *take-graph-formula*:
assumes $g \in carrier\ (Q_p{}^n) \to carrier\ Q_p$
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*⟩

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 $a \in graph\ n\ g$
⟨*proof*⟩

lemma *graph-memI″*:
assumes $a \in carrier\ (Q_p{}^n)$
assumes $g \in carrier\ (Q_p{}^n) \to carrier\ Q_p$
shows $(a@[g\ a]) \in graph\ n\ g$
⟨*proof*⟩

lemma *graph-as-restricted-graph*:
assumes $f \in carrier\ (Q_p{}^n) \to carrier\ Q_p$
shows $graph\ n\ f = restricted\text{-}graph\ n\ f\ (carrier\ (Q_p{}^n))$
⟨*proof*⟩

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*⟩

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*)
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *add-vars-is-semialg*:
assumes $i < n$
assumes $j < n$
assumes $a \in$ *carrier* $(Q_p{}^n)$
shows *is-semialg-function n* (*add-vars i j*)
⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

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*
⟨*proof*⟩

lemma *minus-vars-is-semialg*:
assumes $i < n$
assumes $a \in$ *carrier* $(Q_p{}^n)$

shows *is-semialg-function* n (*minus-vars* i)
⟨*proof*⟩

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*⟩

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* ($Q_p{}^{length\ fs}$)
⟨*proof*⟩

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*⟩

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
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *is-semialg-function-tupleE*:
  assumes *is-semialg-function-tuple n fs*
  assumes $i < length\ fs$
  shows *is-semialg-function n* (*fs* ! *i*)
  ⟨*proof*⟩

lemma *is-semialg-function-tupleE'*:
  assumes *is-semialg-function-tuple n fs*
  assumes $f \in set\ fs$
  shows *is-semialg-function n f*
  ⟨*proof*⟩

lemma *semialg-function-tuple-is-function-tuple*:
  assumes *is-semialg-function-tuple n fs*
  shows *is-function-tuple* Q_p *n fs*
  ⟨*proof*⟩

lemma *const-poly-function-tuple-comp-is-semialg*:
  assumes $n > 0$
  assumes *is-semialg-function-tuple n fs*
  assumes $a \in carrier\ Q_p$
  shows *is-semialg-function n* (*poly-function-tuple-comp* Q_p *n fs* (*Qp-to-IP a*))
  ⟨*proof*⟩

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*))
  ⟨*proof*⟩

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 $x \in carrier\ ({Q_p}^{n+l})$
  shows *tuple-partial-image n fs x* $\in carrier\ ({Q_p}^{length\ fs\ +\ l})$
  ⟨*proof*⟩

lemma *tuple-partial-image-indices*:
  assumes $length\ fs > 0$
  assumes *is-function-tuple* Q_p *n fs*

assumes $x \in$ *carrier* $({Q_p}^{n+l})$
assumes $i <$ *length fs*
shows (*tuple-partial-image n fs x*) ! i = (*fs*!i) (*take n x*)
⟨*proof*⟩

lemma *tuple-partial-image-indices'*:
assumes *length fs* > *0*
assumes *is-function-tuple* Q_p *n fs*
assumes $x \in$ *carrier* $({Q_p}^{n+l})$
assumes $i < l$
shows (*tuple-partial-image n fs x*) ! (*length fs* + *i*) = *x*!(*n* + *i*)
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *tuple-partial-pullback-closed*:
tuple-partial-pullback m fs l S ⊆ *carrier* $({Q_p}^{m\ +\ l})$
⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

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*
⟨*proof*⟩

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)
⟨*proof*⟩

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)
⟨*proof*⟩

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)*
⟨*proof*⟩

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* $(Q_p{}^{m\ +\ l})$
⟨*proof*⟩

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*
⟨*proof*⟩

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)*
⟨*proof*⟩

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*
⟨*proof*⟩

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*

⟨*proof*⟩

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*
⟨*proof*⟩

lemma *twisted-partial-pullback-closed*:
twisted-partial-pullback n m l f S ⊆ *carrier* $(Q_p{}^{n+m+l})$
⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

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*
⟨*proof*⟩

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 $(i::nat) = (\lambda j.\ if\ j < i\ then\ j + 1\ else$
$(if\ j = i\ then\ 0\ else\ j))$

lemma *twisting-permutation-permutes*:
assumes $i < n$
shows *twisting-permutation i permutes* $\{..<n\}$
⟨*proof*⟩

lemma *twisting-permutation-action*:
assumes *length as* $= i$
shows *permute-list* (*twisting-permutation i*) $(b\#(as@bs)) = as@(b\#bs)$
⟨*proof*⟩

lemma *twisting-permutation-action'*:
assumes *length as* $= i$
shows *permute-list* (*fun-inv* (*twisting-permutation i*)) $(as@(b\#bs)) = (b\#(as@bs))$

⟨*proof*⟩

lemma *twisting-semialg*:
assumes *is-semialgebraic n S*
assumes $n > i$
shows *is-semialgebraic n* ((*permute-list* ((*twisting-permutation i*)) ‘ S))
⟨*proof*⟩

lemma *twisting-semialg'*:
assumes *is-semialgebraic n S*
assumes $n > i$
shows *is-semialgebraic n* ((*permute-list* (*fun-inv* (*twisting-permutation i*)) ‘ S))
⟨*proof*⟩

Defining a permutation that does: $(x0, x1, y) \mapsto (x_1, x0, y)$

definition *tp-1* **where**
tp-1 i j $= (\lambda\ n.\ (if\ n<i\ then\ j + n\ else$
$(if\ i \leq n \wedge n < i + j\ then\ n - i\ else$
$n)))$
lemma *permutes-I*:
assumes $\bigwedge x.\ x \notin S \Longrightarrow f\ x = x$
assumes $\bigwedge y.\ y \in S \Longrightarrow \exists!x \in S.\ f\ x = y$
assumes $\bigwedge x.\ x \in S \Longrightarrow f\ x \in S$
shows *f permutes S*
⟨*proof*⟩

lemma *tp-1-permutes*:
(*tp-1* $(i::nat)$ j) *permutes* $\{..< i + j\}$

⟨proof⟩

lemma *tp-1-permutes'*:
(*tp-1* (*i*::*nat*) *j*) *permutes* {..< *i* + *j* + *k*}
⟨proof⟩

lemma *tp-1-permutation-action*:
assumes $a \in carrier\ (Q_p{}^i)$
assumes $b \in carrier\ (Q_p{}^j)$
assumes $c \in carrier\ (Q_p{}^n)$
shows *permute-list* (*tp-1 i j*) (*b*@*a*@*c*)= *a*@*b*@*c*
⟨proof⟩

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*)
⟨proof⟩

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-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⟩

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⟩

definition *augment* **where**
augment n x = *take n x* @ *take n x* @ *drop n x*

lemma *augment-closed*:
assumes $x \in carrier\ (Q_p{}^{n+l})$
shows *augment n x* $\in carrier\ (Q_p{}^{n+n+\ l})$
⟨proof⟩

lemma *tuple-partial-image-factor*:
assumes *is-function-tuple* Q_p *m fs*
assumes $f \in carrier\ (Q_p{}^m) \to carrier\ Q_p$
assumes *length fs* = *n*

assumes $x \in$ *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*⟩

definition *diagonalize* **where**
diagonalize n m S $= S \cap$ *cartesian-product* $(\Delta\ n)$ (*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*⟩

lemma *list-segment-take*:
assumes *length a* $\geq n$
shows *list-segment 0 n a* $=$ *take n a*
⟨*proof*⟩

lemma *augment-inverse-is-semialgebraic*:
assumes *is-semialgebraic* $(n{+}n{+}l)$ S
shows *is-semialgebraic* $(n{+}l)$ ((*augment n* $-$‘ S) $\cap$ *carrier* $(Q_p{}^{n+l})$)
⟨*proof*⟩

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*⟩

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*⟩

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*
⟨*proof*⟩

lemma *tuple-partial-pullback-is-semialg-map-tuple*:
assumes *is-semialg-function-tuple m fs*
shows *is-semialg-map-tuple m fs*
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

Multiplication is semialgebraic:

lemma *multiplication-is-semialg*:
is-semialg-function 2 Qp-mult-fun
⟨*proof*⟩

Inversion is semialgebraic:

lemma(**in** *field*) *field-nat-pow-inv*:
  assumes $a \in$ *carrier R*
  assumes $a \neq$ **0**
  shows *inv* (*a* [^] (*n*::*nat*)) = (*inv a*) [^] (*n* :: *nat*)

⟨proof⟩

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⟩

lemma *Qp-invert-is-semialg*:
is-semialg-function 1 Qp-invert
⟨proof⟩

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⟩

end

13.12 Sets Defined by Residues of Valuation Ring Elements

sublocale *padic-fields* < *Res*: *cring Zp-res-ring* (*Suc n*)
⟨proof⟩

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*)
⟨proof⟩

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* $\oplus_{Zp\text{-}res\text{-}ring\ n}$ *Qp-res y n*
⟨proof⟩

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* $\otimes_{Zp\text{-}res\text{-}ring\ n}$ *Qp-res y n*

$\langle proof \rangle$

lemma *Qp-res-diff*:
assumes $x \in \mathcal{O}_p$
assumes $y \in \mathcal{O}_p$
shows $Qp\text{-}res\ (x \ominus y)\ n = Qp\text{-}res\ x\ n \ominus_{Zp\text{-}res\text{-}ring\ n}\ Qp\text{-}res\ y\ n$
$\langle proof \rangle$

lemma *Qp-res-zero*:
shows $Qp\text{-}res\ \mathbf{0}\ n = 0$
$\langle proof \rangle$

lemma *Qp-res-one*:
assumes $n > 0$
shows $Qp\text{-}res\ \mathbf{1}\ n = (1::int)$
$\langle proof \rangle$

lemma *Qp-res-nat-inc*:
shows $Qp\text{-}res\ ([(n::nat)]\cdot\mathbf{1})\ n = n\ mod\ p\hat{\ }n$
$\langle proof \rangle$

lemma *Qp-res-int-inc*:
shows $Qp\text{-}res\ ([(k::int)]\cdot\mathbf{1})\ n = k\ mod\ p\hat{\ }n$
$\langle proof \rangle$

lemma *Qp-poly-res-monom*:
assumes $a \in \mathcal{O}_p$
assumes $x \in \mathcal{O}_p$
assumes $Qp\text{-}res\ a\ n = 0$
assumes $k > 0$
shows $Qp\text{-}res\ (up\text{-}ring.monom\ (UP\ Q_p)\ a\ k \cdot x)\ n = 0$
$\langle proof \rangle$

lemma *Qp-poly-res-zero*:
assumes $q \in carrier\ (UP\ Q_p)$
assumes $\bigwedge i.\ q\ i \in \mathcal{O}_p$
assumes $\bigwedge i.\ Qp\text{-}res\ (q\ i)\ n = 0$
assumes $x \in \mathcal{O}_p$
shows $Qp\text{-}res\ (q \cdot x)\ n = 0$
$\langle proof \rangle$

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 $\bigwedge i.\ Qp\text{-}res\ (f\ i)\ n = Qp\text{-}res\ (g\ i)\ n$
shows $Qp\text{-}res\ (f \cdot x)\ n = Qp\text{-}res\ (g \cdot x)\ n$

$\langle proof \rangle$

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 $Qp\text{-}res\ x\ n = Qp\text{-}res\ y\ n$
  shows $Qp\text{-}res\ (f \cdot x)\ n = Qp\text{-}res\ (f \cdot y)\ n$
$\langle proof \rangle$

lemma *Qp-poly-res-eval-2*:
  assumes $f \in carrier\ (UP\ Q_p)$
  assumes $g \in carrier\ (UP\ Q_p)$
  assumes $x \in \mathcal{O}_p$
  assumes $y \in \mathcal{O}_p$
  assumes $\bigwedge i.\ f\ i \in \mathcal{O}_p$
  assumes $\bigwedge i.\ g\ i \in \mathcal{O}_p$
  assumes $\bigwedge i.\ Qp\text{-}res\ (f\ i)\ n = Qp\text{-}res\ (g\ i)\ n$
  assumes $Qp\text{-}res\ x\ n = Qp\text{-}res\ y\ n$
  shows $Qp\text{-}res\ (f \cdot x)\ n = Qp\text{-}res\ (g \cdot y)\ n$
$\langle proof \rangle$

definition *poly-res-class* **where**
$poly\text{-}res\text{-}class\ n\ d\ f = \{q \in carrier\ (UP\ Q_p).\ deg\ Q_p\ q \leq d \wedge (\forall i.\ q\ i \in \mathcal{O}_p \wedge Qp\text{-}res\ (f\ i)\ n = Qp\text{-}res\ (q\ i)\ n)\ \}$

lemma *poly-res-class-closed*:
  assumes $f \in carrier\ (UP\ Q_p)$
  assumes $g \in carrier\ (UP\ Q_p)$
  assumes $deg\ Q_p\ f \leq d$
  assumes $deg\ Q_p\ g \leq d$
  assumes $g \in poly\text{-}res\text{-}class\ n\ d\ f$
  shows $poly\text{-}res\text{-}class\ n\ d\ f = poly\text{-}res\text{-}class\ n\ d\ g$
  $\langle proof \rangle$

lemma *poly-res-class-memE*:
  assumes $f \in poly\text{-}res\text{-}class\ n\ d\ g$
  shows $f \in carrier\ (UP\ Q_p)$
    $deg\ Q_p\ f \leq d$
    $f\ i \in \mathcal{O}_p$
    $Qp\text{-}res\ (g\ i)\ n = Qp\text{-}res\ (f\ i)\ n$
  $\langle proof \rangle$

definition *val-ring-polys* **where**
$val\text{-}ring\text{-}polys = \{f \in carrier\ (UP\ Q_p).\ (\forall i.\ f\ i \in \mathcal{O}_p)\}$

lemma *val-ring-polys-closed*:
$val\text{-}ring\text{-}polys \subseteq carrier\ (UP\ Q_p)$

$\langle proof \rangle$

lemma *val-ring-polys-memI*:
assumes $f \in carrier\ (UP\ Q_p)$
assumes $\bigwedge i.\ f\ i \in \mathcal{O}_p$
shows $f \in val\text{-}ring\text{-}polys$
$\langle proof \rangle$

lemma *val-ring-polys-memE*:
assumes $f \in val\text{-}ring\text{-}polys$
shows $f \in carrier\ (UP\ Q_p)$
$f\ i \in \mathcal{O}_p$
$\langle proof \rangle$

definition *val-ring-polys-grad* **where**
$val\text{-}ring\text{-}polys\text{-}grad\ d = \{f \in val\text{-}ring\text{-}polys.\ deg\ Q_p\ f \le d\}$

lemma *val-ring-polys-grad-closed*:
$val\text{-}ring\text{-}polys\text{-}grad\ d \subseteq val\text{-}ring\text{-}polys$
$\langle proof \rangle$

lemma *val-ring-polys-grad-closed'*:
$val\text{-}ring\text{-}polys\text{-}grad\ d \subseteq carrier\ (UP\ Q_p)$
$\langle proof \rangle$

lemma *val-ring-polys-grad-memI*:
assumes $f \in carrier\ (UP\ Q_p)$
assumes $\bigwedge i.\ f\ i \in \mathcal{O}_p$
assumes $deg\ Q_p\ f \le d$
shows $f \in val\text{-}ring\text{-}polys\text{-}grad\ d$
$\langle proof \rangle$

lemma *val-ring-polys-grad-memE*:
assumes $f \in val\text{-}ring\text{-}polys\text{-}grad\ d$
shows $f \in carrier\ (UP\ Q_p)$
$deg\ Q_p\ f \le d$
$f\ i \in \mathcal{O}_p$
$\langle proof \rangle$

lemma *poly-res-classes-in-val-ring-polys-grad*:
assumes $f \in val\text{-}ring\text{-}polys\text{-}grad\ d$
shows $poly\text{-}res\text{-}class\ n\ d\ f \subseteq val\text{-}ring\text{-}polys\text{-}grad\ d$
$\langle proof \rangle$

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 $poly\text{-}res\text{-}class\ n\ d\ f \cap poly\text{-}res\text{-}class\ n\ d\ g = \{\}$
$\langle proof \rangle$

lemma *poly-res-class-refl*:
  assumes $f \in$ *val-ring-polys-grad d*
  shows $f \in$ *poly-res-class n d f*
  ⟨*proof*⟩

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*
  ⟨*proof*⟩

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*⟩

definition *int-fun-to-poly* **where**
int-fun-to-poly $(f::nat \Rightarrow int)\ i = [(f\ i)]\cdot\mathbf{1}$

lemma *int-fun-to-poly-closed*:
  assumes $\bigwedge i.\ i > d \Longrightarrow f\ i = 0$
  shows *int-fun-to-poly f* $\in$ *carrier* $(UP\ Q_p)$
  ⟨*proof*⟩

lemma *int-fun-to-poly-deg*:
  assumes $\bigwedge i.\ i > d \Longrightarrow f\ i = 0$
  shows *deg* Q_p (*int-fun-to-poly f*) $\leq d$
  ⟨*proof*⟩

lemma *Qp-res-mod-triv*:
  assumes $a \in \mathcal{O}_p$
  shows *Qp-res a n mod p* ⌢ *n* = *Qp-res a n*
  ⟨*proof*⟩

lemma *int-fun-to-poly-is-class-wit*:
  assumes $f \in$ *poly-res-class n d g*
  shows (*int-fun-to-poly* ($\lambda i::nat.$ *Qp-res* $(f\ i)\ n$)) $\in$ *poly-res-class n d g*
⟨*proof*⟩

lemma *finite-support-funs-finite*:
finite $((\{..d\} \rightarrow$ *carrier* (*Zp-res-ring n*)$) \cap \{(f::nat \Rightarrow int).\ \forall\, i > d.\ f\ i = 0\})$

$\langle proof \rangle$

lemma *poly-res-classes-finite*:
finite (*poly-res-classes* n d)
$\langle proof \rangle$

lemma *Qp-res-eq-zeroI*:
assumes $a \in \mathcal{O}_p$
assumes *val* $a \geq n$
shows *Qp-res* a n = *0*
$\langle proof \rangle$

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
$\langle proof \rangle$

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
$\langle proof \rangle$

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* $(a \ominus b) \geq n$
$\langle proof \rangle$

lemma *notin-closed*:
$(\neg ((c{::}eint) \leq x \wedge x \leq d)) = (x < c \vee d < x)$
$\langle proof \rangle$

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
$\langle proof \rangle$

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
$\langle proof \rangle$

definition *Qp-res-class* **where**
$Qp\text{-}res\text{-}class\ n\ b = \{a \in \mathcal{O}_p.\ Qp\text{-}res\ a\ n = Qp\text{-}res\ b\ n\}$

definition *Qp-res-classes* **where**
$Qp\text{-}res\text{-}classes\ n = Qp\text{-}res\text{-}class\ n\ `\ \mathcal{O}_p$

lemma *val-ring-int-inc-closed*:
$[(k::int)]\cdot\mathbf{1} \in \mathcal{O}_p$
⟨*proof*⟩

lemma *val-ring-nat-inc-closed*:
$[(k::nat)]\cdot\mathbf{1} \in \mathcal{O}_p$
⟨*proof*⟩

lemma *Qp-res-classes-wits*:
$Qp\text{-}res\text{-}classes\ n\ =\ (\lambda l::int.\ Qp\text{-}res\text{-}class\ n\ ([l]\cdot\mathbf{1}))\ `\ (carrier\ (Zp\text{-}res\text{-}ring\ n))$
⟨*proof*⟩

lemma *Qp-res-classes-finite*:
finite (*Qp-res-classes n*)
⟨*proof*⟩

definition *Qp-cong-set* **where**
$Qp\text{-}cong\text{-}set\ \alpha\ a = \ \{x \in \mathcal{O}_p.\ to\text{-}Zp\ x\ \alpha = a\ \alpha\}$

lemma *Qp-cong-set-as-ball*:
  assumes $a \in carrier\ Z_p$
  assumes $a\ \alpha = 0$
  shows $Qp\text{-}cong\text{-}set\ \alpha\ a = B_\alpha[\mathbf{0}]$
⟨*proof*⟩

lemma *Qp-cong-set-as-ball'*:
  assumes $a \in carrier\ Z_p$
  assumes $val\text{-}Zp\ a < eint\ (int\ \alpha)$
  shows $Qp\text{-}cong\text{-}set\ \alpha\ a = B_\alpha[(\iota\ a)]$
⟨*proof*⟩

lemma *Qp-cong-set-is-univ-semialgebraic*:
  assumes $a \in carrier\ Z_p$
  shows *is-univ-semialgebraic* (*Qp-cong-set* α *a*)
⟨*proof*⟩

lemma *constant-res-set-semialg*:
  assumes $l \in carrier\ (Zp\text{-}res\text{-}ring\ n)$
  shows *is-univ-semialgebraic* $\{x \in \mathcal{O}_p.\ Qp\text{-}res\ x\ n = l\}$
⟨*proof*⟩

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 $b - eint\ N \leq c$
  ⟨*proof*⟩

lemma *eint-minus-plus*:
$a - (eint\ b + eint\ c) = a - eint\ b - eint\ c$
  ⟨*proof*⟩

lemma *eint-minus-plus'*:
$a - (eint\ b + eint\ c) = a - eint\ c - eint\ b$
  ⟨*proof*⟩

lemma *eint-minus-plus''*:
  assumes $a - eint\ c - eint\ b = eint\ f$
  shows $a - eint\ c - eint\ f = eint\ b$
  ⟨*proof*⟩

lemma *uminus-involutive*[*simp*]:
$-(-x::eint) = x$

⟨*proof*⟩

lemma *eint-minus*:
$(a::eint) - (b::eint) = a + (-b)$
⟨*proof*⟩

lemma *eint-mult-Suc*:
$eint\ (Suc\ k) * a = eint\ k * a + a$
⟨*proof*⟩

lemma *eint-mult-Suc-mono*:
assumes $a \leq eint\ b \longrightarrow eint\ (int\ k) * a \leq eint\ (int\ k) * eint\ b$
shows $a \leq eint\ b \longrightarrow eint\ (int\ (Suc\ k)) * a \leq eint\ (int\ (Suc\ k)) * eint\ b$
⟨*proof*⟩

lemma *eint-nat-mult-mono*:
assumes $(a::eint) \leq b$
shows $eint\ (k::nat)*a \leq eint\ k*b$
⟨*proof*⟩

lemma *eint-Suc-zero*:
$eint\ (int\ (Suc\ 0)) * a = a$
⟨*proof*⟩

lemma *eint-add-mono*:
assumes $(a::eint) \leq b$
assumes $(c::eint) \leq d$
shows $a + c \leq b + d$
⟨*proof*⟩

lemma *eint-nat-mult-mono-rev*:
assumes $k > 0$
assumes $eint\ (k::nat)*a \leq eint\ k*b$
shows $(a::eint) \leq b$
⟨*proof*⟩

14.2 Lemmas on Function Ring Operations

lemma *Qp-funs-is-cring*:
$cring\ (Fun_n\ Q_p)$
⟨*proof*⟩

lemma *Qp-funs-is-monoid*:
$monoid\ (Fun_n\ Q_p)$
⟨*proof*⟩

lemma *Qp-funs-car-memE*:
assumes $f \in carrier\ (Fun_n\ Q_p)$
shows $f \in (carrier\ ({Q_p}^n)) \to (carrier\ Q_p)$

$\langle proof \rangle$

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 $g \in carrier\ (Fun_n\ Q_p)$
$\langle proof \rangle$

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 $g \in carrier\ (Fun_n\ Q_p)$
$\langle proof \rangle$

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 $g \in carrier\ (Fun_n\ Q_p)$
$\langle proof \rangle$

lemma *Qp-funs-one*:
$\mathbf{1}_{Fun_n\ Q_p} = (\lambda\ x \in carrier\ (Q_p{}^n).\ \mathbf{1})$
$\langle proof \rangle$

lemma *Qp-funs-zero*:
$\mathbf{0}_{Fun_n\ Q_p} = (\lambda\ x \in carrier\ (Q_p{}^n).\ \mathbf{0}_{Q_p})$
$\langle proof \rangle$

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 $(f \oplus_{Fun_n\ Q_p} g)\ x = f\ x \oplus_{Q_p} g\ x$
$\langle proof \rangle$

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 $(f \oplus_{Fun_n\ Q_p} g)\ x = f\ x \oplus_{Q_p} g\ x$
$\langle proof \rangle$

lemma *Qp-funs-add''*:
assumes $f \in (carrier\ (Fun_n\ Q_p))$
assumes $g \in (carrier\ (Fun_n\ Q_p))$
shows $(f \oplus_{Fun_n\ Q_p} g) = (\lambda\ x \in carrier\ (Q_p{}^n).\ f\ x \oplus_{Q_p} g\ x)$
$\langle proof \rangle$

lemma *Qp-funs-add'''*:

assumes $x \in carrier\ (Q_p{}^n)$
shows $(f \oplus_{Fun_n\ Q_p}\ g)\ x = f\ x \oplus_{Q_p}\ g\ x$
$\langle proof \rangle$

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 $(f \otimes_{Fun_n\ Q_p}\ g)\ x = f\ x \otimes\ g\ x$
$\langle proof \rangle$

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 $(f \otimes_{Fun_n\ Q_p}\ g)\ x = f\ x \otimes\ g\ x$
$\langle proof \rangle$

lemma *Qp-funs-mult″*:
assumes $f \in (carrier\ (Fun_n\ Q_p))$
assumes $g \in (carrier\ (Fun_n\ Q_p))$
shows $(f \otimes_{Fun_n\ Q_p}\ g) = (\lambda\ x \in carrier\ (Q_p{}^n).\ f\ x \otimes\ g\ x)$
$\langle proof \rangle$

lemma *Qp-funs-mult‴*:
assumes $x \in carrier\ (Q_p{}^n)$
shows $(f \otimes_{Fun_n\ Q_p}\ g)\ x = f\ x \otimes\ g\ x$
$\langle proof \rangle$

lemma *Qp-funs-a-inv*:
assumes $x \in carrier\ (Q_p{}^n)$
assumes $f \in (carrier\ (Fun_n\ Q_p))$
shows $(\ominus_{Fun_n\ Q_p}\ f)\ x = \ominus\ (f\ x)$
$\langle proof \rangle$

lemma *Qp-funs-a-inv′*:
assumes $f \in (carrier\ (Fun_n\ Q_p))$
shows $(\ominus_{Fun_n\ Q_p}\ f) = (\lambda\ x \in carrier\ (Q_p{}^n).\ \ominus\ (f\ x))$
$\langle proof \rangle$

abbreviation(*input*) *Qp-const* (‹𝔠-›) **where**
$Qp\text{-}const\ n\ c \equiv constant\text{-}function\ (carrier\ (Q_p{}^n))\ c$

lemma *Qp-constE*:
assumes $c \in carrier\ Q_p$
assumes $x \in carrier\ (Q_p{}^n)$
shows $Qp\text{-}const\ n\ c\ x = c$
$\langle proof \rangle$

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*⟩

lemma *Qp-funs-Units-memE*:
assumes $f \in (Units\ (Fun_n\ Q_p))$
shows $f \otimes_{Fun_n\ Q_p} inv_{Fun_n\ Q_p}\ f = \mathbf{1}_{Fun_n\ Q_p}$
$inv_{Fun_n\ Q_p}\ f \otimes_{Fun_n\ Q_p} f = \mathbf{1}_{Fun_n\ Q_p}$
$\bigwedge x.\ x \in carrier\ (Q_p{}^n) \Longrightarrow f\ x \neq \mathbf{0}_{Q_p}$
⟨*proof*⟩

lemma *Qp-funs-m-inv*:
assumes $x \in carrier\ (Q_p{}^n)$
assumes $f \in (Units\ (Fun_n\ Q_p))$
shows $(inv_{Fun_n\ Q_p}\ f)\ x = inv_{Q_p}\ (f\ x)$
⟨*proof*⟩

14.3 Defining the Rings of Semialgebraic Functions

definition *semialg-functions* **where**
$semialg\text{-}functions\ n = \{f \in (carrier\ (Q_p{}^n)) \to carrier\ Q_p.\ is\text{-}semialg\text{-}function\ n\ f \wedge f = restrict\ f\ (carrier\ (Q_p{}^n))\}$

lemma *semialg-functions-memE*:
assumes $f \in semialg\text{-}functions\ n$
shows $is\text{-}semialg\text{-}function\ n\ f$
$f \in carrier\ (Fun_n\ Q_p)$
$f \in carrier\ (Q_p{}^n) \to carrier\ Q_p$
⟨*proof*⟩

lemma *semialg-functions-in-Qp-funs*:
$semialg\text{-}functions\ n \subseteq carrier\ (Fun_n\ Q_p)$
⟨*proof*⟩

lemma *semialg-functions-memI*:
assumes $f \in carrier\ (Fun_n\ Q_p)$
assumes $is\text{-}semialg\text{-}function\ n\ f$
shows $f \in semialg\text{-}functions\ n$
⟨*proof*⟩

lemma *restrict-is-semialg*:
assumes $is\text{-}semialg\text{-}function\ n\ f$
shows $is\text{-}semialg\text{-}function\ n\ (restrict\ f\ (carrier\ (Q_p{}^n)))$
⟨*proof*⟩

lemma *restrict-in-semialg-functions*:
assumes *is-semialg-function n f*
shows (*restrict f* (*carrier* $(Q_p{}^n)$)) $\in$ *semialg-functions n*
⟨*proof*⟩

lemma *constant-function-is-semialg*:
assumes $a \in$ *carrier* Q_p
shows *is-semialg-function n* (*constant-function* (*carrier* $(Q_p{}^n)$) *a*)
⟨*proof*⟩

lemma *constant-function-in-semialg-functions*:
assumes $a \in$ *carrier* Q_p
shows *Qp-const n a* $\in$ *semialg-functions n*
⟨*proof*⟩

lemma *function-one-as-constant*:
$\mathbf{1}_{Fun_n\ Q_p}$ = *Qp-const n* $\mathbf{1}$
⟨*proof*⟩

lemma *function-zero-as-constant*:
$\mathbf{0}_{Fun_n\ Q_p}$ = *Qp-const n* $\mathbf{0}_{Q_p}$
⟨*proof*⟩

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*⟩

lemma *prod-in-semialg-functions*:
assumes $f \in$ *semialg-functions n*
assumes $g \in$ *semialg-functions n*
shows $f \otimes_{Fun_n\ Q_p} g \in$ *semialg-functions n*
⟨*proof*⟩

lemma *inv-in-semialg-functions*:
assumes $f \in$ *semialg-functions n*
assumes $\bigwedge x.\ x \in$ *carrier* $(Q_p{}^n) \Longrightarrow f\,x \neq \mathbf{0}_{Q_p}$
shows $inv_{Fun_n\ Q_p}\ f \in$ *semialg-functions n*
⟨*proof*⟩

lemma *a-inv-in-semialg-functions*:
assumes $f \in$ *semialg-functions n*
shows $\ominus_{Fun_n\ Q_p}\ f \in$ *semialg-functions n*
⟨*proof*⟩

lemma *semialg-functions-subring*:
shows *subring* (*semialg-functions n*) $(Fun_n\ Q_p)$
⟨*proof*⟩

lemma *semialg-functions-subcring*:
  shows *subcring* (*semialg-functions* n) ($Fun_n\ Q_p$)
  ⟨*proof*⟩

definition *SA* **where**
$SA\ n = (Fun_n\ Q_p)(\!|\ carrier := semialg\text{-}functions\ n|\!)$

lemma *SA-is-ring*:
  shows *ring* (*SA n*)
⟨*proof*⟩

lemma *SA-is-cring*:
  shows *cring* (*SA n*)
  ⟨*proof*⟩

lemma *SA-is-monoid*:
  shows *monoid* (*SA n*)
  ⟨*proof*⟩

lemma *SA-is-abelian-monoid*:
  shows *abelian-monoid* (*SA n*)
  ⟨*proof*⟩

lemma *SA-car*:
carrier (*SA n*) = *semialg-functions n*
    ⟨*proof*⟩

lemma *SA-car-in-Qp-funs-car*:
carrier (*SA n*) ⊆ *carrier* ($Fun_n\ Q_p$)
  ⟨*proof*⟩

lemma *SA-car-memI*:
  assumes $f \in$ *carrier* ($Fun_n\ Q_p$)
  assumes *is-semialg-function n f*
  shows $f \in$ *carrier* (*SA n*)
  ⟨*proof*⟩

lemma *SA-car-memE*:
  assumes $f \in$ *carrier* (*SA n*)
  shows *is-semialg-function n f*
        $f \in$ *carrier* ($Fun_n\ Q_p$)
        $f \in$ *carrier* (${Q_p}^n$) → *carrier* Q_p
  ⟨*proof*⟩

lemma *SA-plus*:
$(\oplus_{SA\ n}) = (\oplus_{Fun_n\ Q_p})$
  ⟨*proof*⟩

lemma *SA-times*:
$(\otimes_{SA\ n}) = (\otimes_{Fun_n\ Q_p})$
  $\langle proof \rangle$

lemma *SA-one*:
$(\mathbf{1}_{SA\ n}) = (\mathbf{1}_{Fun_n\ Q_p})$
  $\langle proof \rangle$

lemma *SA-zero*:
$(\mathbf{0}_{SA\ n}) = (\mathbf{0}_{Fun_n\ Q_p})$
  $\langle proof \rangle$

lemma *SA-zero-is-function-ring*:
$(Fun_0\ Q_p) = SA\ 0$
$\langle proof \rangle$

lemma *constant-fun-closed*:
  assumes $c \in carrier\ Q_p$
  shows $constant\text{-}function\ (carrier\ (Q_p{}^m))\ c \in carrier\ (SA\ m)$
  $\langle proof \rangle$

lemma *SA-0-car-memI*:
  assumes $\xi \in carrier\ (Q_p{}^0) \to carrier\ Q_p$
  assumes $\bigwedge x.\ x \notin carrier\ (Q_p{}^0) \Longrightarrow \xi\ x = undefined$
  shows $\xi \in carrier\ (SA\ 0)$
$\langle proof \rangle$

lemma *car-SA-0-mem-imp-const*:
  assumes $a \in carrier\ (SA\ 0)$
  shows $\exists\ c \in carrier\ Q_p.\ a = Qp\text{-}const\ 0\ c$
$\langle proof \rangle$

lemma *SA-zeroE*:
  assumes $a \in carrier\ (Q_p{}^n)$
  shows $\mathbf{0}\ _{SA\ n}\ a = \mathbf{0}$
  $\langle proof \rangle$

lemma *SA-oneE*:
  assumes $a \in carrier\ (Q_p{}^n)$
  shows $\mathbf{1}\ _{SA\ n}\ a = \mathbf{1}$
  $\langle proof \rangle$
end

sublocale *padic-fields* $<$ *UPSA?*: *UP-cring SA m UP* $(SA\ m)$
  $\langle proof \rangle$

context *padic-fields*
begin

lemma *SA-add*:
  assumes $x \in carrier\ (Q_p{}^n)$
  shows $(f \oplus_{SA\ n}\ g)\ x = f\ x \oplus_{Q_p}\ g\ x$
  ⟨*proof*⟩

lemma *SA-add'*:
  assumes $x \notin carrier\ (Q_p{}^n)$
  shows $(f \oplus_{SA\ n}\ g)\ x = undefined$
⟨*proof*⟩

lemma *SA-mult*:
  assumes $x \in carrier\ (Q_p{}^n)$
  shows $(f \otimes_{SA\ n}\ g)\ x = f\ x \otimes\ g\ x$
  ⟨*proof*⟩

lemma *SA-mult'*:
  assumes $x \notin carrier\ (Q_p{}^n)$
  shows $(f \otimes_{SA\ n}\ g)\ x = undefined$
⟨*proof*⟩

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*⟩

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*⟩

lemma *SA-nat-pow*:
  assumes $x \in carrier\ (Q_p{}^n)$
  shows $(f\ [\urcorner]_{SA\ n}\ (k{::}nat))\ x = (f\ x)\ [\urcorner]_{Q_p}\ k$
  ⟨*proof*⟩

lemma *SA-nat-pow'*:
  assumes $x \notin carrier\ (Q_p{}^n)$
  shows $(f\ [\urcorner]_{SA\ n}\ (k{::}nat))\ x = undefined$
  ⟨*proof*⟩

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*⟩

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*⟩

lemma *SA-add-closed*:
assumes *is-semialg-function* n f
assumes *is-semialg-function* n g
shows $f \oplus_{SA\ n} g \in$ *carrier* (*SA* n)
⟨*proof*⟩

lemma *SA-mult-closed*:
assumes *is-semialg-function* n f
assumes *is-semialg-function* n g
shows $f \otimes_{SA\ n} g \in$ *carrier* (*SA* n)
⟨*proof*⟩

lemma *SA-add-closed-right*:
assumes *is-semialg-function* n f
assumes $g \in$ *carrier* (*SA* n)
shows $f \oplus_{SA\ n} g \in$ *carrier* (*SA* n)
⟨*proof*⟩

lemma *SA-mult-closed-right*:
assumes *is-semialg-function* n f
assumes $g \in$ *carrier* (*SA* n)
shows $f \otimes_{SA\ n} g \in$ *carrier* (*SA* n)
⟨*proof*⟩

lemma *SA-add-closed-left*:
assumes $f \in$ *carrier* (*SA* n)
assumes *is-semialg-function* n g
shows $f \oplus_{SA\ n} g \in$ *carrier* (*SA* n)
⟨*proof*⟩

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)
⟨*proof*⟩

lemma *SA-nat-pow-closed*:
assumes *is-semialg-function* n f
shows $f\ [\urcorner]_{SA\ n}$ (k::*nat*) $\in$ *carrier* (*SA* n)
⟨*proof*⟩

lemma *SA-imp-semialg*:
assumes $f \in$ *carrier* (*SA* n)
shows *is-semialg-function* n f

$\langle proof \rangle$

lemma *SA-minus-closed*:
assumes $f \in carrier\ (SA\ n)$
assumes $g \in carrier\ (SA\ n)$
shows $(f\ \ominus_{SA\ n}\ g) \in carrier\ (SA\ n)$
$\langle proof \rangle$

lemma(**in** *ring*) *add-pow-closed* :
assumes $b \in carrier\ R$
shows $([(m::nat)] \cdot_R b) \in carrier\ R$
$\langle proof \rangle$

lemma(**in** *ring*) *add-pow-Suc*:
assumes $b \in carrier\ R$
shows $[(Suc\ m)] \cdot b = [m] \cdot b \oplus b$
$\langle proof \rangle$

lemma(**in** *ring*) *add-pow-zero*:
assumes $b \in carrier\ R$
shows $[(0::nat)] \cdot b = \mathbf{0}$
$\langle proof \rangle$

lemma *Fun-add-pow-apply*:
assumes $b \in carrier\ (Fun_n\ Q_p)$
assumes $a \in carrier\ ({Q_p}^n)$
shows $([(m::nat)] \cdot_{Fun_n\ Q_p}\ b)\ a = [m] \cdot (\ b\ a)$
$\langle proof \rangle$

lemma *SA-add-pow-apply*:
assumes $b \in carrier\ (SA\ n)$
assumes $a \in carrier\ ({Q_p}^n)$
shows $([(m::nat)] \cdot_{SA\ n}\ b)\ a = [m] \cdot (\ b\ a)$
$\langle proof \rangle$

lemma *Qp-funs-Units-SA-Units*:
assumes $f \in Units\ (Fun_n\ Q_p)$
assumes *is-semialg-function n f*
shows $f \in Units\ (SA\ n)$
$\langle proof \rangle$

lemma *SA-Units-memE*:
assumes $f \in (Units\ (SA\ n))$
shows $f \otimes_{SA\ n} inv_{SA\ n}\ f = \mathbf{1}_{SA\ n}$
$inv_{SA\ n}\ f \otimes_{SA\ n} f = \mathbf{1}_{SA\ n}$
$\langle proof \rangle$

lemma *SA-Units-closed*:
assumes $f \in (Units\ (SA\ n))$

shows $f \in carrier\ (SA\ n)$
⟨*proof*⟩

lemma *SA-Units-inv-closed*:
assumes $f \in (Units\ (SA\ n))$
shows $inv_{SA\ n}\ f \in carrier\ (SA\ n)$
⟨*proof*⟩

lemma *SA-Units-Qp-funs-Units*:
assumes $f \in (Units\ (SA\ n))$
shows $f \in (Units\ (Fun_n\ Q_p))$
⟨*proof*⟩

lemma *SA-Units-Qp-funs-inv*:
assumes $f \in (Units\ (SA\ n))$
shows $inv_{SA\ n}\ f = inv_{Fun n\ Q_p}\ f$
⟨*proof*⟩

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 $f \in (Units\ (SA\ n))$
⟨*proof*⟩

lemma *SA-Units-memE'*:
assumes $f \in (Units\ (SA\ n))$
shows $\bigwedge x.\ x \in carrier\ (Q_p{}^n) \Longrightarrow f\ x \neq \mathbf{0}_{Q_p}$
⟨*proof*⟩

lemma *Qp-n-nonempty*:
shows $carrier\ (Q_p{}^n) \neq \{\}$
⟨*proof*⟩

lemma *SA-one-not-zero*:
shows $\mathbf{1}_{SA\ n} \neq \mathbf{0}_{SA\ n}$
⟨*proof*⟩

lemma *SA-units-not-zero*:
assumes $f \in Units\ (SA\ n)$
shows $f \neq \mathbf{0}_{SA\ n}$
⟨*proof*⟩

lemma *SA-Units-nonzero*:
assumes $f \in Units\ (SA\ m)$
assumes $x \in carrier\ (Q_p{}^m)$
shows $f\ x \in nonzero\ Q_p$
⟨*proof*⟩

lemma *SA-car-closed*:

assumes $f \in carrier\ (SA\ m)$
assumes $x \in carrier\ (Q_p{}^m)$
shows $f\,x \in carrier\ Q_p$
⟨*proof*⟩

lemma *SA-Units-closed-fun*:
assumes $f \in Units\ (SA\ m)$
assumes $x \in carrier\ (Q_p{}^m)$
shows $f\,x \in carrier\ Q_p$
⟨*proof*⟩

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*⟩

lemma *SA-div-eval*:
assumes $f \in Units\ (SA\ n)$
assumes $h \in carrier\ (SA\ n)$
assumes $x \in carrier\ (Q_p{}^n)$
shows $(h \otimes_{SA\ n} (inv_{SA\ n}\ f))\ x = h\ x \otimes inv\ (f\ x)$
⟨*proof*⟩

lemma *SA-unit-int-pow*:
assumes $f \in Units\ (SA\ m)$
assumes $x \in carrier\ (Q_p{}^m)$
shows $(f[\urcorner]_{SA\ m}(i::int))\ x = (f\ x)[\urcorner]i$
⟨*proof*⟩

lemma *restrict-in-SA-car*:
assumes *is-semialg-function* $n\ f$
shows $restrict\ f\ (carrier\ (Q_p{}^n)) \in carrier\ (SA\ n)$
⟨*proof*⟩

lemma *SA-smult*:
$(\odot_{SA\ n}) = (\odot_{Fun_n\ Q_p})$
⟨*proof*⟩

lemma *SA-smult-formula*:
assumes $h \in carrier\ (SA\ n)$
assumes $q \in carrier\ Q_p$
assumes $a \in carrier\ (Q_p{}^n)$
shows $(q \odot_{SA\ n}\ h)\ a = q \otimes (h\ a)$
⟨*proof*⟩

lemma *SA-smult-closed*:
assumes $h \in carrier\ (SA\ n)$
assumes $q \in carrier\ Q_p$

shows $q \odot_{SA\ n} h \in carrier\ (SA\ n)$
⟨*proof*⟩

lemma *p-mult-function-val*:
assumes $f \in carrier\ (SA\ m)$
assumes $x \in carrier\ (Q_p{}^m)$
shows $val\ ((\mathfrak{p} \odot_{SA\ m} f)\ x) = val\ (f\ x) + 1$
⟨*proof*⟩

lemma *Qp-char-0''*:
assumes $a \in carrier\ Q_p$
assumes $a \neq \mathbf{0}$
assumes $(k::nat) > 0$
shows $[k]\cdot a \neq \mathbf{0}$
⟨*proof*⟩

lemma *SA-char-zero*:
assumes $f \in carrier\ (SA\ m)$
assumes $f \neq \mathbf{0}_{SA\ m}$
assumes $n > 0$
shows $[(n::nat)]\cdot_{SA\ m} f \neq \mathbf{0}_{SA\ m}$
⟨*proof*⟩

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**
$map\text{-}partial\text{-}image\ m\ f\ xs = (f\ (take\ m\ xs))@(drop\ m\ xs)$

definition *map-partial-pullback* **where**
$map\text{-}partial\text{-}pullback\ m\ f\ l\ S = (map\text{-}partial\text{-}image\ m\ f)\ ^{-1}{}_{m+l}\ S$

lemma *map-partial-pullback-memE*:

assumes $as \in$ *map-partial-pullback m f l S*
shows $as \in$ *carrier* $(Q_p{}^{m+l})$ *map-partial-image m f as* $\in S$
⟨*proof*⟩

lemma *map-partial-pullback-closed*:
map-partial-pullback m f l S $\subseteq$ *carrier* $(Q_p{}^{m+l})$
⟨*proof*⟩

lemma *map-partial-pullback-memI*:
assumes $as \in$ *carrier* $(Q_p{}^{m+k})$
assumes $(f\ (take\ m\ as))@(drop\ m\ as) \in S$
shows $as \in$ *map-partial-pullback m f k S*
⟨*proof*⟩

lemma *map-partial-image-eq*:
assumes $as \in$ *carrier* $(Q_p{}^{n})$
assumes $bs \in$ *carrier* $(Q_p{}^{k})$
assumes $x = as\ @\ bs$
shows *map-partial-image n f x* $= (f\ as)@bs$
⟨*proof*⟩

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-partial-pullback n f k S*
shows $(f\ as)@bs \in S$
⟨*proof*⟩

Partial pullbacks have the same algebraic properties as pullbacks.

lemma *map-partial-pullback-intersect*:
map-partial-pullback m f l $(S1 \cap S2) =$ (*map-partial-pullback m f l S1*) $\cap$ (*map-partial-pullback m f l S2*)
⟨*proof*⟩

lemma *map-partial-pullback-union*:
map-partial-pullback m f l $(S1 \cup S2) =$ (*map-partial-pullback m f l S1*) $\cup$ (*map-partial-pullback m f l S2*)
⟨*proof*⟩

lemma *map-partial-pullback-complement*:
assumes $f \in$ *carrier* $(Q_p{}^{m}) \to$ *carrier* $(Q_p{}^{n})$
shows *map-partial-pullback m f l* (*carrier* $(Q_p{}^{n+l}) - S$) $=$ *carrier* $(Q_p{}^{m+l}) -$ (*map-partial-pullback m f l S*)
⟨*proof*⟩

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* $(Q_p{}^{m+l})$

$\langle proof \rangle$

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* $(Q_p{}^n)$
$\langle proof \rangle$

lemma *is-semialg-map-closed'*:
assumes *is-semialg-map* m n f $x \in$ *carrier* $(Q_p{}^m)$
shows $f\ x \in$ *carrier* $(Q_p{}^n)$
$\langle proof \rangle$

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)
$\langle proof \rangle$

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)
$\langle proof \rangle$

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
$\langle proof \rangle$

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
$\langle proof \rangle$

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

$\langle proof \rangle$

lemma *is-semialg-mapI'''*:
assumes $f \in carrier\ (Q_p{}^m) \rightarrow carrier\ (Q_p{}^n)$
assumes $\bigwedge k\ \ S.\ S \in basic\text{-}semialgs\ (n + k) \Longrightarrow is\text{-}semialgebraic\ (m + k)$ $(map\text{-}partial\text{-}image\ m\ f\ {}^{-1}{}_{m+k}\ S)$
shows *is-semialg-map m n f*
$\langle proof \rangle$

lemma *id-is-semialg-map*:
is-semialg-map n n $(\lambda\ x.\ x)$
$\langle proof \rangle$

lemma *map-partial-pullback-comp*:
assumes *is-semialg-map m n f*
assumes *is-semialg-map k m g*
shows $(map\text{-}partial\text{-}pullback\ k\ (f \circ g)\ l\ S) = (map\text{-}partial\text{-}pullback\ k\ g\ l\ (map\text{-}partial\text{-}pullback\ m\ f\ l\ S))$
$\langle proof \rangle$

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* $(f \circ g)$
$\langle proof \rangle$

lemma *partial-pullback-comp*:
assumes *is-semialg-function m f*
assumes *is-semialg-map k m g*
shows $(partial\text{-}pullback\ k\ (f \circ g)\ l\ S) = (map\text{-}partial\text{-}pullback\ k\ g\ l\ (partial\text{-}pullback\ m\ f\ l\ S))$
$\langle proof \rangle$

lemma *semialg-function-comp-closed*:
assumes *is-semialg-function m f*
assumes *is-semialg-map k m g*
shows *is-semialg-function k* $(f \circ g)$
$\langle proof \rangle$

lemma *semialg-map-evimage-is-semialg*:
assumes *is-semialg-map k m g*
assumes *is-semialgebraic m S*
shows *is-semialgebraic k* $(g\ \ {}^{-1}{}_{k}\ S)$
$\langle proof \rangle$

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*⟩

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*)
⟨*proof*⟩

lemma *index-is-semialg-function*:
assumes $n > k$
shows *is-semialg-function n* ($\lambda as.\ as!k$)
⟨*proof*⟩

definition *Qp-ith* **where**
Qp-ith m i = (λ $x \in$ *carrier* ($Q_p{}^m$). $x!i$)

lemma *Qp-ith-closed*:
assumes $i < m$
shows *Qp-ith m i* $\in$ *carrier* (*SA m*)
⟨*proof*⟩

lemma *take-is-semialg-map*:
assumes $n \geq k$
shows *is-semialg-map n k* (*take k*)
⟨*proof*⟩

lemma *drop-is-semialg-map*:
shows *is-semialg-map* ($k + n$) *n* (*drop k*)
⟨*proof*⟩

lemma *project-at-indices-is-semialg-map*:
assumes $S \subseteq \{..<n\}$
shows *is-semialg-map n* (*card S*) π_S
⟨*proof*⟩

lemma *tl-is-semialg-map*:
shows *is-semialg-map* (*Suc n*) *n tl*
⟨*proof*⟩

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* $\in$ *carrier* (*SA n*)
⟨*proof*⟩

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*⟩

lemma *semialg-map-cons*:
assumes *is-semialg-map n m g*
assumes $f \in$ *carrier* (*SA n*)
shows *is-semialg-map n* (*Suc m*) (λ $x \in$ *carrier* (${Q_p}^n$). *f x* # *g x*)
⟨*proof*⟩

Extensional Semialgebraic Maps:

definition *semialg-maps* **where**
semialg-maps n m $\equiv$ {*f*. *is-semialg-map n m f* $\wedge$ $f \in$ *struct-maps* (${Q_p}^n$) (${Q_p}^m$)}

lemma *semialg-mapsE*:
assumes $f \in$ (*semialg-maps n m*)
shows *is-semialg-map n m f*
$f \in$ *struct-maps* (${Q_p}^n$) (${Q_p}^m$)
$f \in$ *carrier* (${Q_p}^n$) $\to$ *carrier* (${Q_p}^m$)
⟨*proof*⟩

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* $\in$ *semialg-maps n m*
⟨*proof*⟩

14.6 Application of Functions to Segments of Tuples

definition *take-apply* **where**
take-apply m n f = *restrict* ($f \circ$ *take n*) (*carrier* (${Q_p}^m$))

definition *drop-apply* **where**
drop-apply m n f = *restrict* ($f \circ$ *drop n*) (*carrier* (${Q_p}^m$))

lemma *take-apply-closed*:
assumes $f \in$ *carrier* (Fun_n Q_p)
assumes $k \geq n$
shows *take-apply k n f* $\in$ *carrier* (Fun_k Q_p)
⟨*proof*⟩

lemma *take-apply-SA-closed*:
assumes $f \in$ *carrier* (*SA n*)
assumes $k \geq n$
shows *take-apply k n f* $\in$ *carrier* (*SA k*)
⟨*proof*⟩

lemma *drop-apply-closed*:
assumes $f \in carrier\ (Fun_{k\ -\ n}\ Q_p)$
assumes $k \geq n$
shows $drop\text{-}apply\ k\ n\ f \in carrier\ (Fun_k\ Q_p)$
⟨*proof*⟩

lemma *drop-apply-SA-closed*:
assumes $f \in carrier\ (SA\ (k-n))$
assumes $k \geq n$
shows $drop\text{-}apply\ k\ n\ f \in carrier\ (SA\ k)$
⟨*proof*⟩

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 $take\text{-}apply\ (n+k)\ n\ f\ (a@b) = f\ a$
⟨*proof*⟩

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 $drop\text{-}apply\ (n+k)\ n\ f\ (a@b) = f\ b$
⟨*proof*⟩

lemma *drop-apply-add*:
assumes $f \in carrier\ (SA\ n)$
assumes $g \in carrier\ (SA\ n)$
shows $drop\text{-}apply\ (n+k)\ k\ (f \oplus_{SA\ n}\ g) = drop\text{-}apply\ (n+k)\ k\ f \oplus_{SA\ (n\ +\ k)}$ $drop\text{-}apply\ (n+k)\ k\ g$
⟨*proof*⟩

lemma *drop-apply-mult*:
assumes $f \in carrier\ (SA\ n)$
assumes $g \in carrier\ (SA\ n)$
shows $drop\text{-}apply\ (n+k)\ k\ (f \otimes_{SA\ n}\ g) = drop\text{-}apply\ (n+k)\ k\ f \otimes_{SA\ (n\ +\ k)}$ $drop\text{-}apply\ (n+k)\ k\ g$
⟨*proof*⟩

lemma *drop-apply-one*:
shows $drop\text{-}apply\ (n+k)\ k\ \mathbf{1}_{SA\ n} = \mathbf{1}_{SA\ (n+k)}$
⟨*proof*⟩

lemma *drop-apply-is-hom*:
shows $drop\text{-}apply\ (n\ +\ k)\ k \in ring\text{-}hom\ (SA\ n)\ (SA\ (n\ +\ k))$
⟨*proof*⟩

lemma *take-apply-add*:
  assumes $f \in$ *carrier* (*SA* k)
  assumes $g \in$ *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
  ⟨*proof*⟩

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
  ⟨*proof*⟩

lemma *take-apply-one*:
  shows *take-apply* $(n+k)$ k $\mathbf{1}_{SA\ k} = \mathbf{1}_{SA\ (n+k)}$
  ⟨*proof*⟩

lemma *take-apply-is-hom*:
  shows *take-apply* $(n + k)$ $k \in$ *ring-hom* (*SA* k) (*SA* $(n + k)$)
  ⟨*proof*⟩

lemma *drop-apply-units*:
  assumes $f \in$ *Units* (*SA* n)
  shows *drop-apply* $(n+k)$ k $f \in$ *Units* (*SA* $(n+k)$)
  ⟨*proof*⟩

lemma *take-apply-units*:
  assumes $f \in$ *Units* (*SA* k)
  shows *take-apply* $(n+k)$ k $f \in$ *Units* (*SA* $(n+k)$)
  ⟨*proof*⟩

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*⟩

lemma *semialg-val-ineq-set-is-semialg*:
  assumes $g \in$ *carrier* (*SA* k)
  assumes $f \in$ *carrier* (*SA* k)
  shows *is-semialgebraic* k $\{x \in$ *carrier* $(Q_p{}^k)$. *val* $(g\ x) \leq$ *val* $(f\ x)\}$
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*⟩

lemma *semialg-val-eq-set-is-semialg*:
assumes $g \in carrier\ (SA\ k)$
assumes $f \in carrier\ (SA\ k)$
shows *is-semialgebraic* $k\ \{x \in carrier\ (Q_p{}^k).\ val\ (g\ x) = val\ (f\ x)\}$
⟨*proof*⟩

lemma *semialg-val-strict-ineq-set-formula*:
$\{x \in carrier\ (Q_p{}^k).\ val\ (g\ x) < val\ (f\ x)\} = \{x \in carrier\ (Q_p{}^k).\ val\ (g\ x) \leq val\ (f\ x)\} - \{x \in carrier\ (Q_p{}^k).\ val\ (g\ x) = val\ (f\ x)\}$
⟨*proof*⟩

lemma *semialg-val-ineq-set-complement*:
$carrier\ (Q_p{}^k) - \{x \in carrier\ (Q_p{}^k).\ val\ (g\ x) \leq val\ (f\ x)\} = \{x \in carrier\ (Q_p{}^k).\ val\ (f\ x) < val\ (g\ x)\}$
⟨*proof*⟩

lemma *semialg-val-strict-ineq-set-complement*:
$carrier\ (Q_p{}^k) - \{x \in carrier\ (Q_p{}^k).\ val\ (g\ x) < val\ (f\ x)\} = \{x \in carrier\ (Q_p{}^k).\ val\ (f\ x) \leq val\ (g\ x)\}$
⟨*proof*⟩

lemma *semialg-val-strict-ineq-set-is-semialg*:
assumes $g \in carrier\ (SA\ k)$
assumes $f \in carrier\ (SA\ k)$
shows *is-semialgebraic* $k\ \{x \in carrier\ (Q_p{}^k).\ val\ (g\ x) < val\ (f\ x)\}$
⟨*proof*⟩

lemma *semialg-val-strict-ineq-set-is-semialg'*:
assumes $f \in carrier\ (SA\ k)$
shows *is-semialgebraic* $k\ \{x \in carrier\ (Q_p{}^k).\ val\ (f\ x) < C\}$

⟨*proof*⟩

lemma *semialg-val-strict-ineq-set-is-semialg''*:
  assumes $f \in carrier\ (SA\ k)$
  shows *is-semialgebraic* $k\ \{x \in carrier\ (Q_p{}^k).\ val\ (f\ x) > C\}$
⟨*proof*⟩

lemma *semialg-val-ineq-set-plus*:
  assumes $N > 0$
  assumes $c \in carrier\ (SA\ N)$
  assumes $a \in carrier\ (SA\ N)$
  shows *is-semialgebraic* $N\ \{x \in carrier\ (Q_p{}^N).\ val\ (c\ x) \leq val\ (a\ x) + eint\ n\}$
⟨*proof*⟩

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*⟩

definition *SA-zero-set* **where**
SA-zero-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* $n\ f$)
  ⟨*proof*⟩

lemma *SA-zero-set-memE*:
  assumes $f \in carrier\ (SA\ n)$
  assumes $x \in$ *SA-zero-set* $n\ f$
  shows $f\ x = \mathbf{0}$
  ⟨*proof*⟩

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-zero-set* $n\ f$
  ⟨*proof*⟩

lemma *SA-zero-set-of-zero*:
SA-zero-set $m\ (\mathbf{0}_{SA\ m}) = carrier\ (Q_p{}^m)$
  ⟨*proof*⟩

definition *SA-nonzero-set* **where**
SA-nonzero-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-semialgebraic n* $\{x \in$ *carrier* $(Q_p{}^n).\ f\ x \neq \mathbf{0}\}$
⟨*proof*⟩

lemma *SA-nonzero-set-is-semialgebraic*:
assumes $f \in$ *carrier* (*SA n*)
shows *is-semialgebraic n* (*SA-nonzero-set n f*)
⟨*proof*⟩

lemma *SA-nonzero-set-memE*:
assumes $f \in$ *carrier* (*SA n*)
assumes $x \in$ *SA-nonzero-set n f*
shows $f\ x \neq \mathbf{0}$
⟨*proof*⟩

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-nonzero-set n f*
⟨*proof*⟩

lemma *SA-nonzero-set-of-zero*:
SA-nonzero-set m $(\mathbf{0}_{SA\ m}) = \{\}$
⟨*proof*⟩

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 $f \in$ *carrier* (*SA m*)
⟨*proof*⟩

lemma(**in** *padic-fields*) *SA-zero-set-is-semialg*:
assumes $a \in$ *carrier* (*SA m*)
shows *is-semialgebraic m* $\{x \in$ *carrier* $(Q_p{}^m).\ a\ x = \mathbf{0}\}$
⟨*proof*⟩

lemma(**in** *padic-fields*) *SA-nonzero-set-is-semialg*:
assumes $a \in$ *carrier* (*SA m*)
shows *is-semialgebraic m* $\{x \in$ *carrier* $(Q_p{}^m).\ a\ x \neq \mathbf{0}\}$
⟨*proof*⟩

lemma *zero-set-nonzero-set-covers*:
carrier $(Q_p{}^n) =$ *SA-zero-set n f* $\cup$ *SA-nonzero-set n f*
⟨*proof*⟩

lemma *zero-set-nonzero-set-covers'*:
assumes $S \subseteq$ *carrier* $(Q_p{}^n)$
shows $S = (S \cap SA\text{-}zero\text{-}set\ n\ f) \cup (S \cap SA\text{-}nonzero\text{-}set\ n\ f)$
⟨*proof*⟩

lemma *zero-set-nonzero-set-covers-semialg-set*:
assumes *is-semialgebraic n S*
shows $S = (S \cap SA\text{-}zero\text{-}set\ n\ f) \cup (S \cap SA\text{-}nonzero\text{-}set\ n\ f)$
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *semialgebraic-subalgebra*:
assumes *finite Xs*
assumes $Xs \subseteq$ *semialg-sets n*
shows *atoms-of Xs* $\subseteq$ *semialg-sets n*
⟨*proof*⟩

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*⟩

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 $val\ (a\ x) \leq val\ (f\ x)$

$\langle proof \rangle$

lemma *Min-set-closed*:
Min-set m As a $\subseteq$ *carrier* $(Q_p{}^m)$
$\langle proof \rangle$

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*)
$\langle proof \rangle$

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*)
$\langle proof \rangle$

lemma *Min-sets-cover*:
assumes $As \neq \{\}$
assumes *finite As*
shows *carrier* $(Q_p{}^m) = (\bigcup\ a \in As.$ *Min-set m As a*)
$\langle proof \rangle$

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-set-partition m As B = *disjointify* $((\cap)B$ ` (*Min-set m As* ` *As*))

lemma *Min-set-partition-finite*:
assumes *finite As*
shows *finite* (*Min-set-partition m As B*)
$\langle proof \rangle$

lemma *Min-set-partition-semialg0*:
assumes *finite As*
assumes *As* $\subseteq$ *carrier* (*SA m*)
assumes *is-semialgebraic m B*
assumes $S \in ((\cap)B$ ` (*Min-set m As* ` *As*))
shows *is-semialgebraic m S*
$\langle proof \rangle$

lemma *Min-set-partition-semialg*:
assumes *finite As*
assumes *As* $\subseteq$ *carrier* (*SA m*)

assumes *is-semialgebraic m B*
assumes $S \in$ (*Min-set-partition m As B*)
shows *is-semialgebraic m S*
⟨*proof*⟩

lemma *Min-set-partition-covers0*:
assumes *finite As*
assumes $As \neq \{\}$
assumes $As \subseteq$ *carrier* (*SA m*)
assumes *is-semialgebraic m B*
shows $\bigcup$ $((\cap)B$ ‘ (*Min-set m As* ‘ *As*)) = *B*
⟨*proof*⟩

lemma *Min-set-partition-covers*:
assumes *finite As*
assumes $As \subseteq$ *carrier* (*SA m*)
assumes $As \neq \{\}$
assumes *is-semialgebraic m B*
shows $\bigcup$ (*Min-set-partition m As B*) = *B*
⟨*proof*⟩

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 $s \cap s' = \{\}$
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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 = ($\lambda x \in$ *carrier* ($Q_p{}^n$). *if* $x \in S$ *then* f x *else* g x)

lemma *fun-glueE*:
  assumes $f \in$ *carrier* (*SA* n)
  assumes $g \in$ *carrier* (*SA* n)
  assumes $S \subseteq$ *carrier* ($Q_p{}^n$)
  assumes $x \in S$
  shows *fun-glue* n S f g x = f x
⟨*proof*⟩

lemma *fun-glueE'*:
  assumes $f \in$ *carrier* (*SA* n)
  assumes $g \in$ *carrier* (*SA* n)
  assumes $S \subseteq$ *carrier* ($Q_p{}^n$)
  assumes $x \in$ *carrier* ($Q_p{}^n$) − S
  shows *fun-glue* n S f g x = g x
⟨*proof*⟩

lemma *fun-glue-evimage*:
  assumes $f \in$ *carrier* (*SA* n)
  assumes $g \in$ *carrier* (*SA* n)
  assumes $S \subseteq$ *carrier* ($Q_p{}^n$)
  shows *fun-glue* n S f g $^{-1}{}_n$ T = ((f $^{-1}{}_n$ T) $\cap$ S) $\cup$ ((g $^{-1}{}_n$ T) − S)
⟨*proof*⟩

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*⟩

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
  ⟨*proof*⟩

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)
  ⟨*proof*⟩

lemma *fun-glue-unit*:
  assumes $f \in carrier\ (SA\ n)$
  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*⟩

definition *parametric-fun-glue* **where**
parametric-fun-glue $n\ Xs\ fs = (\lambda x \in carrier\ (Q_p{}^n).\ let\ S = (THE\ S.\ S \in Xs \wedge x \in S)\ in\ (fs\ S\ x))$

lemma *parametric-fun-glue-formula*:
  assumes *Xs partitions* $(carrier\ (Q_p{}^n))$
  assumes $x \in S$
  assumes $S \in Xs$
  shows *parametric-fun-glue* $n\ Xs\ fs\ x = fs\ S\ x$
⟨*proof*⟩

definition *char-fun* **where**
char-fun $n\ S = (\lambda\ x \in carrier\ (Q_p{}^n).\ if\ x \in S\ then\ \mathbf{1}\ else\ \mathbf{0})$

lemma *char-fun-is-semialg*:
  assumes *is-semialgebraic* $n\ S$
  shows *char-fun* $n\ S \in carrier\ (SA\ n)$
⟨*proof*⟩

lemma *SA-finsum-apply*:
  assumes *finite* S
  assumes $x \in carrier\ (Q_p{}^n)$
  shows $F \in S \rightarrow carrier\ (SA\ n) \longrightarrow finsum\ (SA\ n)\ F\ S\ x = (\bigoplus s{\in}S.\ F\ s\ x)$
⟨*proof*⟩

lemma *SA-finsum-apply-zero*:
  assumes *finite* S
  assumes $F \in S \rightarrow carrier\ (SA\ n)$
  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*⟩

lemma *parametric-fun-glue-is-SA*:
  assumes *finite Xs*
  assumes *Xs partitions* $(carrier\ (Q_p{}^n))$
  assumes $fs \in Xs \rightarrow carrier\ (SA\ n)$
  assumes $\forall\ S \in Xs.$ *is-semialgebraic* $n\ S$
  shows *parametric-fun-glue* $n\ Xs\ fs \in carrier\ (SA\ n)$
⟨*proof*⟩

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* $(Q_p{}^n).\ f\ x \neq \mathbf{0}\}\ f\ \mathbf{1}_{SA\ n}$

lemma *to-fun-unit-is-unit*:
  assumes $f \in$ *carrier* $(SA\ n)$
  shows *to-fun-unit* $n\ f \in$ *Units* $(SA\ n)$
  ⟨*proof*⟩

lemma *to-fun-unit-closed*:
  assumes $f \in$ *carrier* $(SA\ n)$
  shows *to-fun-unit* $n\ f \in$ *carrier* $(SA\ n)$
  ⟨*proof*⟩

lemma *to-fun-unit-eq*:
  assumes $f \in$ *carrier* $(SA\ n)$
  assumes $x \in$ *carrier* $(Q_p{}^n)$
  assumes $f\ x \neq \mathbf{0}$
  shows *to-fun-unit* $n\ f\ x = f\ x$
  ⟨*proof*⟩

lemma *to-fun-unit-eq'*:
  assumes $f \in$ *carrier* $(SA\ n)$
  assumes $x \in$ *carrier* $(Q_p{}^n)$
  assumes $f\ x = \mathbf{0}$
  shows *to-fun-unit* $n\ f\ x = \mathbf{1}$
  ⟨*proof*⟩

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* $(SA\ n)$
  ⟨*proof*⟩

lemma *one-over-fun-eq*:
  assumes $f \in$ *carrier* $(SA\ n)$
  assumes $x \in$ *carrier* $(Q_p{}^n)$
  assumes $f\ x \neq \mathbf{0}$
  shows *one-over-fun* $n\ f\ x = inv\ (f\ x)$

$\langle proof \rangle$

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 $one\text{-}over\text{-}fun\ n\ (a \odot_{SA\ n} f)\ x = inv\ (a \otimes (f\ x))$
$\langle proof \rangle$

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 $one\text{-}over\text{-}fun\ n\ (a \odot_{SA\ n} f)\ x = inv\ a \otimes inv\ (f\ x)$
$\langle proof \rangle$

lemma *SA-add-pow-closed*:
assumes $f \in carrier\ (SA\ n)$
shows $([(k::nat)] \cdot_{SA\ n} f) \in carrier\ (SA\ n)$
$\langle proof \rangle$

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 $one\text{-}over\text{-}fun\ n\ ([k] \cdot_{SA\ n} f)\ x = inv\ ([k]\ \cdot f\ x)$
$\langle proof \rangle$

lemma *one-over-fun-pow-closed*:
assumes $f \in carrier\ (SA\ n)$
shows $one\text{-}over\text{-}fun\ n\ (f[\urcorner]_{SA\ n}(k::nat)) \in carrier\ (SA\ n)$
$\langle proof \rangle$

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 $one\text{-}over\text{-}fun\ n\ (f[\urcorner]_{SA\ n}(k::nat))\ x = inv\ ((f\ x)\ [\urcorner]\ k)$
$\langle proof \rangle$

14.11 Inclusions of Lower Dimensional Function Rings

definition *fun-inc* **where**
fun-inc m n f = (λ *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*⟩

lemma *fun-inc-eval*:
assumes *x* $\in$ *carrier* ($Q_p{}^m$)
shows *fun-inc m n f x* = *f* (*take n x*)
⟨*proof*⟩

lemma *ord-congruence-set-univ-semialg-fixed*:
assumes $n \geq 0$
shows *is-univ-semialgebraic* (*ord-congruence-set n a*)
⟨*proof*⟩

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* $\in$ *nonzero* Q_p $\wedge$ *ord* (*c x*) *mod n* = *a*}
⟨*proof*⟩

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* $\in$ *nonzero* Q_p $\wedge$ *ac n* (*c x*) = *k*}
⟨*proof*⟩

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* $\in$ *nonzero* Q_p $\wedge$ *ac n* (*c x*) = *k*}
⟨*proof*⟩

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*⟩

14.12 Miscellaneous

lemma *nth-pow-wits-SA-fun-prep*:
assumes $n > 0$
assumes $h \in carrier\ (SA\ m)$
assumes $\varrho \in nth\text{-}pow\text{-}wits\ n$
shows *is-semialgebraic* $m\ (h\ ^{-1}{}_m pow\text{-}res\ n\ \varrho)$
⟨*proof*⟩

definition *kth-rt* **where**
$kth\text{-}rt\ m\ (k{::}nat)\ f\ x = (if\ x \in carrier\ (Q_p{}^m)\ then\ (THE\ b.\ b \in carrier\ Q_p \wedge b[\widehat{\ }]k = (f\ x) \wedge ac\ (nat\ (ord\ ([k]\cdot\mathbf{1})) + 1)\ b = 1)$
$else\ undefined\ )$

Normalizing a semialgebraic function to have a constant angular component

lemma *ac-res-Unit-inc*:
assumes $n > 0$
assumes $t \in Units\ (Zp\text{-}res\text{-}ring\ n)$
shows $ac\ n\ ([t]\cdot\mathbf{1}) = t$
⟨*proof*⟩

lemma *val-of-res-Unit*:
assumes $n > 0$
assumes $t \in Units\ (Zp\text{-}res\text{-}ring\ n)$
shows $val\ ([t]\cdot\mathbf{1}) = 0$
⟨*proof*⟩

lemma(**in** *padic-integers*) *res-map-is-hom*:
assumes $N > 0$
shows $ring\text{-}hom\text{-}ring\ Zp\ (Zp\text{-}res\text{-}ring\ N)\ (\lambda\ x.\ x\ N)$
⟨*proof*⟩

lemma *ac-of-fraction*:
assumes $N > 0$
assumes $a \in nonzero\ Q_p$
assumes $b \in nonzero\ Q_p$
shows $ac\ N\ (a \div b) = ac\ N\ a \otimes_{Zp\text{-}res\text{-}ring\ N} inv\ _{Zp\text{-}res\text{-}ring\ N}\ ac\ N\ b$
⟨*proof*⟩

lemma *pow-res-eq-rel*:
assumes $n > 0$
assumes $b \in carrier\ Q_p$
shows $\{x \in carrier\ Q_p.\ pow\text{-}res\ n\ x = pow\text{-}res\ n\ b\} = pow\text{-}res\ n\ b$
⟨*proof*⟩

lemma *pow-res-is-univ-semialgebraic'*:
  assumes $n > 0$
  assumes $b \in carrier\ Q_p$
  shows *is-univ-semialgebraic* $\{x \in carrier\ Q_p.\ pow\text{-}res\ n\ x = pow\text{-}res\ n\ b\}$
  ⟨*proof*⟩

lemma *evimage-eqI*:
  assumes $c \in carrier\ (SA\ n)$
  shows $c\ \ ^{-1}{}_n\ \{x \in carrier\ Q_p.\ P\ x\} = \{x \in carrier\ (Q_p{}^n).\ P\ (c\ x)\}$
  ⟨*proof*⟩

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*⟩

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 $b \in nonzero\ Q_p$
  ⟨*proof*⟩

lemma *SA-minus-eval*:
  assumes $f \in carrier\ (SA\ n)$
  assumes $g \in carrier\ (SA\ n)$
  assumes $x \in carrier\ (Q_p{}^n)$
  shows $(f\ \ominus_{SA\ n}\ g)\ x = f\ x \ominus\ g\ x$
  ⟨*proof*⟩

lemma *Qp-cong-set-evimage*:
  assumes $f \in carrier\ (SA\ n)$
  assumes $a \in carrier\ Z_p$
  shows *is-semialgebraic* $n\ (f\ ^{-1}{}_n\ (Qp\text{-}cong\text{-}set\ \alpha\ a))$
  ⟨*proof*⟩

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*⟩

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*⟩

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*⟩

lemma(**in** *padic-fields*) *res-eq-set-is-semialg*:
assumes $k > 0$
assumes $c \in$ *carrier* $(SA\ k)$
assumes $s \in$ *carrier* (*Zp-res-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*⟩

lemma *SA-constant-res-set-semialg'*:
assumes $f \in$ *carrier* $(SA\ m)$
assumes $C \in$ *Qp-res-classes n*
shows *is-semialgebraic m* $(f\ ^{-1}{}_m\ C)$
⟨*proof*⟩

14.13 Semialgebraic Polynomials

lemma *UP-SA-n-is-ring*:
shows *ring* (*UP* (*SA n*))
⟨*proof*⟩

lemma *UP-SA-n-is-cring*:
shows *cring* (*UP* (*SA n*))
⟨*proof*⟩

The evaluation homomorphism from `Qp_funs` to `Qp`

definition *eval-hom* **where**
eval-hom a = $(\lambda f.\ f\ a)$

lemma *eval-hom-is-hom*:
assumes $a \in$ *carrier* $(Q_p{}^n)$
shows *ring-hom-ring* $(Fun_n\ Q_p)\ Q_p$ (*eval-hom a*)
⟨*proof*⟩

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 $a \in$ *carrier* $(Q_p{}^n)$
shows (*Qp-fpoly-to-Qp-poly n a*) $\in$ *ring-hom* (*UP* $(Fun_n\ Q_p)$) (Q_p-*x*)

⟨proof⟩

lemma *Qp-fpoly-to-Qp-poly-extends-apply*:
assumes $a \in carrier\ (Q_p{}^n)$
assumes $f \in carrier\ (Fun_n\ Q_p)$
shows *Qp-fpoly-to-Qp-poly n a* $(to\text{-}polynomial\ (Fun_n\ Q_p)\ f) = to\text{-}polynomial\ Q_p\ (f\ a)$
⟨proof⟩

lemma *Qp-fpoly-to-Qp-poly-X-var*:
assumes $a \in carrier\ (Q_p{}^n)$
shows *Qp-fpoly-to-Qp-poly n a* $(X\text{-}poly\ (Fun_n\ Q_p)) = X\text{-}poly\ Q_p$
⟨proof⟩

lemma *Qp-fpoly-to-Qp-poly-monom*:
assumes $a \in carrier\ (Q_p{}^n)$
assumes $f \in carrier\ (Fun_n\ Q_p)$
shows *Qp-fpoly-to-Qp-poly n a* $(up\text{-}ring.monom\ (UP\ (Fun_n\ Q_p))\ f\ m) = up\text{-}ring.monom\ Q_p\text{-}x\ (f\ a)\ m$
⟨proof⟩

lemma *Qp-fpoly-to-Qp-poly-coeff*:
assumes $a \in carrier\ (Q_p{}^n)$
assumes $f \in carrier\ (UP\ (Fun_n\ Q_p))$
shows *Qp-fpoly-to-Qp-poly n a f k* $= (f\ k)\ a$
⟨proof⟩

lemma *Qp-fpoly-to-Qp-poly-eval*:
assumes $a \in carrier\ (Q_p{}^n)$
assumes $P \in carrier\ (UP\ (Fun_n\ Q_p))$
assumes $f \in carrier\ (Fun_n\ Q_p)$
shows $(UP\text{-}cring.to\text{-}fun\ (Fun_n\ Q_p)\ P\ f)\ a = UP\text{-}cring.to\text{-}fun\ Q_p\ (Qp\text{-}fpoly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ P)\ (f\ a)$
⟨proof⟩

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* $(compose\ (Fun_n\ Q_p)\ f\ g) = compose\ Q_p\ (Qp\text{-}fpoly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ f)\ (Qp\text{-}fpoly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ g)$
⟨proof⟩

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* $(taylor\text{-}expansion\ (Fun_n\ Q_p)\ c\ F) =$
$taylor\text{-}expansion\ Q_p\ (c\ a)\ (Qp\text{-}fpoly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ F)$

⟨*proof*⟩

lemma *SA-is-UP-cring*:
shows *UP-cring* (*SA n*)
⟨*proof*⟩

lemma *eval-hom-is-SA-hom*:
assumes $a \in carrier\ (Q_p{}^n)$
shows *ring-hom-ring* (*SA n*) Q_p (*eval-hom a*)
⟨*proof*⟩

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 $a \in carrier\ (Q_p{}^n)$
shows (*SA-poly-to-Qp-poly n a*) ∈ *ring-hom* (*UP* (*SA n*)) (Q_p-*x*)
⟨*proof*⟩

lemma *SA-poly-to-Qp-poly-closed*:
assumes $a \in carrier\ (Q_p{}^n)$
assumes *P* ∈ *carrier* (*UP* (*SA n*))
shows *SA-poly-to-Qp-poly n a P* ∈ *carrier* Q_p-*x*
⟨*proof*⟩

lemma *SA-poly-to-Qp-poly-add*:
assumes $a \in carrier\ (Q_p{}^n)$
assumes *f* ∈ *carrier* (*UP* (*SA n*))
assumes *g* ∈ *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*
⟨*proof*⟩

lemma *SA-poly-to-Qp-poly-minus*:
assumes $a \in carrier\ (Q_p{}^n)$
assumes *f* ∈ *carrier* (*UP* (*SA n*))
assumes *g* ∈ *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*
⟨*proof*⟩

lemma *SA-poly-to-Qp-poly-mult*:
assumes $a \in carrier\ (Q_p{}^n)$
assumes *f* ∈ *carrier* (*UP* (*SA n*))
assumes *g* ∈ *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*

⟨*proof*⟩

lemma *SA-poly-to-Qp-poly-extends-apply*:
assumes $a \in carrier\ (Q_p{}^n)$
assumes $f \in carrier\ (SA\ n)$
shows $SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ (to\text{-}polynomial\ (SA\ n)\ f) = to\text{-}polynomial\ Q_p\ (f\ a)$
⟨*proof*⟩

lemma *SA-poly-to-Qp-poly-X-var*:
assumes $a \in carrier\ (Q_p{}^n)$
shows $SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ (X\text{-}poly\ (SA\ n)) = X\text{-}poly\ Q_p$
⟨*proof*⟩

lemma *SA-poly-to-Qp-poly-X-plus*:
assumes $a \in carrier\ (Q_p{}^n)$
assumes $c \in carrier\ (SA\ n)$
shows $SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ (X\text{-}poly\text{-}plus\ (SA\ n)\ c) = UPQ.X\text{-}plus\ (c\ a)$
⟨*proof*⟩

lemma *SA-poly-to-Qp-poly-X-minus*:
assumes $a \in carrier\ (Q_p{}^n)$
assumes $c \in carrier\ (SA\ n)$
shows $SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ (X\text{-}poly\text{-}minus\ (SA\ n)\ c) = UPQ.X\text{-}minus\ (c\ a)$
⟨*proof*⟩

lemma *SA-poly-to-Qp-poly-monom*:
assumes $a \in carrier\ (Q_p{}^n)$
assumes $f \in carrier\ (SA\ n)$
shows $SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ (up\text{-}ring.monom\ (UP\ (SA\ n))\ f\ m) = up\text{-}ring.monom\ Q_p\text{-}x\ (f\ a)\ m$
⟨*proof*⟩

lemma *SA-poly-to-Qp-poly-coeff*:
assumes $a \in carrier\ (Q_p{}^n)$
assumes $f \in carrier\ (UP\ (SA\ n))$
shows $SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ f\ k = (f\ k)\ a$
⟨*proof*⟩

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 $(UP\text{-}cring.to\text{-}fun\ (SA\ n)\ P\ f)\ a = UP\text{-}cring.to\text{-}fun\ Q_p\ (SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ P)\ (f\ a)$
⟨*proof*⟩

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*)
⟨*proof*⟩

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*
⟨*proof*⟩

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*)
⟨*proof*⟩

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*
⟨*proof*⟩

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*)
⟨*proof*⟩

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-poly-to-Qp-poly m x g*) = (*SA-poly-to-Qp-poly m x* (*pderiv m g*))
⟨*proof*⟩

lemma(**in** *UP-cring*) *pderiv-deg-lt*:
assumes $f \in$ *carrier* (*UP R*)
assumes *deg R f* > *0*
shows *deg R* (*pderiv f*) < *deg R f*
⟨*proof*⟩

lemma *deg-pderiv*:

assumes $f \in carrier\ (UP\ (SA\ m))$
assumes $deg\ (SA\ m)\ f > 0$
shows $deg\ (SA\ m)\ (pderiv\ m\ f) = deg\ (SA\ m)\ f - 1$
$\langle proof \rangle$

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 $SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ m\ x\ (a \odot_{UP\ (SA\ m)}\ f) = a\ x \odot_{UP\ Q_p}\ SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ m\ x\ f$
$\langle proof \rangle$

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 $is\text{-}semialgebraic\ m\ \{x \in carrier\ (Q_p{}^m).\ SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ m\ x\ f \in C\}$
$\langle proof \rangle$

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**
$SA\text{-}poly\text{-}to\text{-}SA\text{-}fun\ n\ P = (\lambda\ a \in carrier\ (Q_p{}^{Suc\ n}).\ UP\text{-}cring.to\text{-}fun\ Q_p\ (SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ (tl\ a)\ P)\ (hd\ a))$

lemma *SA-poly-to-SA-fun-is-fun*:
assumes $P \in carrier\ (UP\ (SA\ n))$
shows $SA\text{-}poly\text{-}to\text{-}SA\text{-}fun\ n\ P \in (carrier\ (Q_p{}^{Suc\ n}) \to carrier\ Q_p)$
$\langle proof \rangle$

lemma *SA-poly-to-SA-fun-formula*:
assumes $P \in carrier\ (UP\ (SA\ n))$
assumes $x \in carrier\ (Q_p{}^n)$
assumes $t \in carrier\ Q_p$
shows $SA\text{-}poly\text{-}to\text{-}SA\text{-}fun\ n\ P\ (t\#x) = (SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ x\ P){\cdot}t$
$\langle proof \rangle$

lemma *semialg-map-comp-in-SA*:
assumes $f \in carrier\ (SA\ n)$
assumes $is\text{-}semialg\text{-}map\ m\ n\ g$
shows $(\lambda\ a \in carrier\ (Q_p{}^m).\ f\ (g\ a)) \in carrier\ (SA\ m)$
$\langle proof \rangle$

lemma *tl-comp-in-SA*:
assumes $f \in carrier\ (SA\ n)$
shows $(\lambda\ a \in carrier\ (Q_p{}^{Suc\ n}).\ f\ (tl\ a)) \in carrier\ (SA\ (Suc\ n))$
$\langle proof \rangle$

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 $SA\text{-}poly\text{-}to\text{-}SA\text{-}fun\ n\ (f \oplus_{UP\ (SA\ n)}\ g)\ a = SA\text{-}poly\text{-}to\text{-}SA\text{-}fun\ n\ f\ a \oplus_{Q_p} SA\text{-}poly\text{-}to\text{-}SA\text{-}fun\ n\ g\ a$
⟨*proof*⟩

lemma *SA-poly-to-SA-fun-add*:
assumes $f \in carrier\ (UP\ (SA\ n))$
assumes $g \in carrier\ (UP\ (SA\ n))$
shows $SA\text{-}poly\text{-}to\text{-}SA\text{-}fun\ n\ (f \oplus_{UP\ (SA\ n)}\ g) = SA\text{-}poly\text{-}to\text{-}SA\text{-}fun\ n\ f \oplus_{SA\ (Suc\ n)} SA\text{-}poly\text{-}to\text{-}SA\text{-}fun\ n\ g$
⟨*proof*⟩

lemma *SA-poly-to-SA-fun-monom*:
assumes $f \in carrier\ (SA\ n)$
assumes $a \in carrier\ (Q_p{}^{Suc\ n})$
shows $SA\text{-}poly\text{-}to\text{-}SA\text{-}fun\ n\ (up\text{-}ring.monom\ (UP\ (SA\ n))\ f\ k)\ a = (f\ (tl\ a)) \otimes (hd\ a)[\urcorner]_{Q_p} k$
⟨*proof*⟩

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 $SA\text{-}poly\text{-}to\text{-}SA\text{-}fun\ n\ (up\text{-}ring.monom\ (UP\ (SA\ n))\ f\ k)\ (t\#x) = (f\ x) \otimes t[\urcorner]_{Q_p} k$
⟨*proof*⟩

lemma *hd-is-semialg-function*:
assumes $n > 0$
shows *is-semialg-function n hd*
⟨*proof*⟩

lemma *SA-poly-to-SA-fun-monom-closed*:
assumes $f \in carrier\ (SA\ n)$
shows $SA\text{-}poly\text{-}to\text{-}SA\text{-}fun\ n\ (up\text{-}ring.monom\ (UP\ (SA\ n))\ f\ k) \in carrier\ (SA\ (Suc\ n))$
⟨*proof*⟩

lemma *SA-poly-to-SA-fun-is-SA*:
assumes $P \in carrier\ (UP\ (SA\ n))$
shows $SA\text{-}poly\text{-}to\text{-}SA\text{-}fun\ n\ P \in carrier\ (SA\ (Suc\ n))$
⟨*proof*⟩

lemma *SA-poly-to-SA-fun-mult*:
assumes $f \in carrier\ (UP\ (SA\ n))$

assumes $g \in carrier\ (UP\ (SA\ n))$
shows $SA\text{-}poly\text{-}to\text{-}SA\text{-}fun\ n\ (f \otimes_{UP\ (SA\ n)}\ g) = SA\text{-}poly\text{-}to\text{-}SA\text{-}fun\ n\ f \otimes_{SA\ (Suc\ n)}$
$SA\text{-}poly\text{-}to\text{-}SA\text{-}fun\ n\ g$
⟨*proof*⟩

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*⟩

lemma *SA-poly-to-SA-fun-ring-hom*:
shows $SA\text{-}poly\text{-}to\text{-}SA\text{-}fun\ n \in ring\text{-}hom\ (UP\ (SA\ n))\ (SA\ (Suc\ n))$
⟨*proof*⟩

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 $f = SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ x\ F$
shows $SA\text{-}poly\text{-}to\text{-}SA\text{-}fun\ n\ (UP\text{-}cring.taylor\text{-}term\ (SA\ n)\ c\ F\ k)\ (t\#x) = (taylor\text{-}expansion$
$Q_p\ (c\ x)\ f\ k) \otimes (t \ominus c\ x)[\hat{}]_{Q_p}\ k$
⟨*proof*⟩

lemma *SA-finsum-eval*:
assumes *finite I*
assumes $F \in I \rightarrow carrier\ (SA\ m)$
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*⟩

lemma(**in** *ring*) *finsum-ring-hom*:
assumes *ring S*
assumes $h \in ring\text{-}hom\ R\ S$
assumes $F \in I \rightarrow carrier\ R$
assumes *finite I*
shows $h\ (\bigoplus i{\in}I.\ F\ i) = (\bigoplus_S i{\in}I.\ h\ (F\ i))$
⟨*proof*⟩

lemma *SA-poly-to-SA-fun-finsum*:
assumes *finite I*
assumes $F \in I \rightarrow carrier\ (UP\ (SA\ m))$
assumes $f = (\bigoplus_{UP\ (SA\ m)} i{\in}I.\ F\ i)$
assumes $x \in carrier\ (Q_p{}^{Suc\ m})$
shows $SA\text{-}poly\text{-}to\text{-}SA\text{-}fun\ m\ f\ x = (\bigoplus i{\in}I.\ SA\text{-}poly\text{-}to\text{-}SA\text{-}fun\ m\ (F\ i)\ x)$
⟨*proof*⟩

lemma *SA-poly-to-SA-fun-taylor-expansion*:
assumes $f \in carrier\ (UP\ (SA\ m))$

assumes $c \in$ *carrier* (*SA m*)
assumes $x \in$ *carrier* ($Q_p{}^{Suc\ m}$)
shows *SA-poly-to-SA-fun m f x* = ($\bigoplus i \in \{..deg\ (SA\ m)\ f\}.\ taylor\text{-}expansion\ (SA\ m)\ c\ f\ i\ (tl\ x) \otimes (hd\ x \ominus c\ (tl\ x))\ [\urcorner]\ i$)
⟨*proof*⟩

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-poly-to-SA-fun m g*) (ξ $x\#x$) = $\mathbf{0}$
shows $\xi = \ominus_{SA\ m}(g\ 0) \otimes_{SA\ m} (inv_{SA\ m}\ (g\ 1))$
⟨*proof*⟩

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-poly-to-SA-fun m g*) (ξ $x\#x$) = $\mathbf{0}$
shows $\xi \in$ *carrier* (*SA m*)
⟨*proof*⟩

lemma *Qp-pow-ConsI*:
assumes $t \in$ *carrier* Q_p
assumes $x \in$ *carrier* ($Q_p{}^m$)
shows $t\#x \in$ *carrier* ($Q_p{}^{Suc\ m}$)
⟨*proof*⟩

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
⟨*proof*⟩

lemma(**in** *ring*) *add-monoid-one*:
$\mathbf{1}_{add\text{-}monoid\ R} = \mathbf{0}$
⟨*proof*⟩

lemma(**in** *ring*) *add-monoid-carrier*:
carrier (*add-monoid R*) = *carrier R*
⟨*proof*⟩

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*
⟨*proof*⟩

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 $\bigwedge i.\ A \subseteq SA\text{-}zero\text{-}set\ m\ (taylor\text{-}expansion\ (SA\ m)\ c\ f\ i) \vee A \subseteq SA\text{-}nonzero\text{-}set\ m\ (taylor\text{-}expansion\ (SA\ m)\ c\ f\ i)$
assumes $a = to\text{-}fun\text{-}unit\ m \circ taylor\text{-}expansion\ (SA\ m)\ c\ f$
assumes $inds = \{i.\ i \leq deg\ (SA\ m)\ f \wedge A \subseteq SA\text{-}nonzero\text{-}set\ m\ (taylor\text{-}expansion\ (SA\ m)\ c\ f\ i)\}$
assumes $x \in A$
assumes $t \in carrier\ Q_p$
shows $SA\text{-}poly\text{-}to\text{-}SA\text{-}fun\ m\ f\ (t\#x) = (\bigoplus i{\in}inds.\ (a\ i\ x)\otimes(t \ominus c\ x)[\urcorner]i)$
⟨*proof*⟩

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 $taylor\text{-}expansion\ (SA\ m)\ c\ f\ i\ x =$
$taylor\text{-}expansion\ Q_p\ (c\ x)\ (SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ m\ x\ f)\ i$
⟨*proof*⟩

14.13.1 Common Morphisms on Polynomial Rings

Evaluation homomorphism from multivariable polynomials to semialgebraic functions

definition *Qp-ev-hom* **where**
$Qp\text{-}ev\text{-}hom\ n\ P = restrict\ (Qp\text{-}ev\ P)\ (carrier\ (Q_p{}^n))$

lemma *Qp-ev-hom-ev*:
assumes $a \in carrier\ (Q_p{}^n)$
shows $Qp\text{-}ev\text{-}hom\ n\ P\ a = Qp\text{-}ev\ P\ a$
⟨*proof*⟩

lemma *Qp-ev-hom-closed*:
assumes $f \in carrier\ (Q_p[\mathcal{X}_n])$
shows $Qp\text{-}ev\text{-}hom\ n\ f \in carrier\ (Q_p{}^n) \to carrier\ Q_p$
⟨*proof*⟩

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)*
⟨*proof*⟩

lemma *Qp-ev-hom-closed'*:
  assumes $f \in$ *carrier* $(Q_p[\mathcal{X}_n])$
  shows *Qp-ev-hom* $n\ f \in$ *carrier* $(Fun_n\ Q_p)$
  ⟨*proof*⟩

lemma *Qp-ev-hom-in-SA*:
  assumes $f \in$ *carrier* $(Q_p[\mathcal{X}_n])$
  shows *Qp-ev-hom* $n\ f \in$ *carrier* $(SA\ n)$
  ⟨*proof*⟩

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* $n\ (f \oplus_{Q_p[\mathcal{X}_n]} g) = (Qp\text{-}ev\text{-}hom\ n\ f) \oplus_{SA\ n} (Qp\text{-}ev\text{-}hom\ n\ g)$
  ⟨*proof*⟩

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* $n\ (f \otimes_{Q_p[\mathcal{X}_n]} g) = (Qp\text{-}ev\text{-}hom\ n\ f) \otimes_{SA\ n} (Qp\text{-}ev\text{-}hom\ n\ g)$
  ⟨*proof*⟩

lemma *Qp-ev-hom-one*:
  shows *Qp-ev-hom* $n\ \mathbf{1}_{Q_p[\mathcal{X}_n]} = \mathbf{1}_{SA\ n}$
  ⟨*proof*⟩

lemma *Qp-ev-hom-is-hom*:
  shows *Qp-ev-hom* $n \in$ *ring-hom* $(Q_p[\mathcal{X}_n])\ (SA\ n)$
  ⟨*proof*⟩

lemma *Qp-ev-hom-constant*:
  assumes $c \in$ *carrier* Q_p
  shows *Qp-ev-hom* n (*Qp.indexed-const* c) $= \mathfrak{c}_n\ c$
  ⟨*proof*⟩

notation *Qp.variable* $(\langle \mathfrak{v}_{\_},\ \_ \rangle)$

lemma *Qp-ev-hom-pvar*:
  assumes $i < n$
  shows *Qp-ev-hom* n (*pvar* $Q_p\ i$) $= \mathfrak{v}_{n,\ i}$
  ⟨*proof*⟩

definition *ext-hd* **where**
ext-hd $m = (\lambda\ x \in$ *carrier* $({Q_p}^m)$. *hd* $x)$

lemma *hd-zeroth*:
length $x > 0 \Longrightarrow x!0 =$ *hd* x
  ⟨*proof*⟩

lemma *ext-hd-pvar*:
  assumes $m > 0$
  shows *ext-hd m* = ($\lambda x \in$ *carrier* ($Q_p{}^m$). *eval-at-point* Q_p *x* (*pvar* Q_p *0*))
  ⟨*proof*⟩

lemma *ext-hd-closed*:
  assumes $m > 0$
  shows *ext-hd m* ∈ *carrier* (*SA m*)
  ⟨*proof*⟩

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*))
  ⟨*proof*⟩

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*))
  ⟨*proof*⟩

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*
  ⟨*proof*⟩

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*
  ⟨*proof*⟩

lemma *coord-ring-to-UP-SA-one*:
  shows *coord-ring-to-UP-SA n* $\mathbf{1}_{Q_p[\mathcal{X}_{Suc\ n}]} = \mathbf{1}_{UP\ (SA\ n)}$
  ⟨*proof*⟩

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*))
  ⟨*proof*⟩

lemma *coord-ring-to-UP-SA-constant*:

assumes $c \in carrier\ Q_p$
shows $coord\text{-}ring\text{-}to\text{-}UP\text{-}SA\ n\ (Qp.indexed\text{-}const\ c) = to\text{-}polynomial\ (SA\ n)\ (\mathfrak{c}_n\ c)$
$\langle proof \rangle$

lemma *coord-ring-to-UP-SA-pvar-0*:
shows $coord\text{-}ring\text{-}to\text{-}UP\text{-}SA\ n\ (pvar\ Q_p\ 0) = up\text{-}ring.monom\ (UP\ (SA\ n))\ \mathbf{1}_{SA\ n}\ 1$
$\langle proof \rangle$

lemma *coord-ring-to-UP-SA-pvar-Suc*:
assumes $i > 0$
assumes $i < Suc\ n$
shows $coord\text{-}ring\text{-}to\text{-}UP\text{-}SA\ n\ (pvar\ Q_p\ i) = to\text{-}polynomial\ (SA\ n)\ (\mathfrak{v}_{n,\ i-1})$
$\langle proof \rangle$

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 $Qp\text{-}ev\ f\ (t\#a) = ((SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ n\ a\ (coord\text{-}ring\text{-}to\text{-}UP\text{-}SA\ n\ f)))\cdot t$
$\langle proof \rangle$

14.13.2 Gluing Semialgebraic Polynomials

definition *SA-poly-glue* **where**
$SA\text{-}poly\text{-}glue\ m\ S\ f\ g = (\lambda\ n.\ fun\text{-}glue\ m\ S\ (f\ n)\ (g\ n))$

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 $SA\text{-}poly\text{-}glue\ m\ S\ f\ g \in carrier\ (UP\ (SA\ m))$
$\langle proof \rangle$

lemma *SA-poly-glue-deg*:
assumes $f \in carrier\ (UP\ (SA\ m))$
assumes $g \in carrier\ (UP\ (SA\ m))$
assumes *is-semialgebraic m S*
assumes $deg\ (SA\ m)\ f \le d$
assumes $deg\ (SA\ m)\ g \le d$
shows $deg\ (SA\ m)\ (SA\text{-}poly\text{-}glue\ m\ S\ f\ g) \le d$
$\langle proof \rangle$

lemma *UP-SA-cfs-closed*:
assumes $g \in carrier\ (UP\ (SA\ m))$
shows $g\ k \in carrier\ (SA\ m)$
$\langle proof \rangle$

lemma *SA-poly-glue-cfs1*:
assumes $f \in$ *carrier* (*UP* (*SA m*))
assumes $g \in$ *carrier* (*UP* (*SA m*))
assumes *is-semialgebraic m S*
assumes $x \in S$
shows (*SA-poly-glue m S f g*) $n\ x = f\ n\ x$
⟨*proof*⟩

lemma *SA-poly-glue-cfs2*:
assumes $f \in$ *carrier* (*UP* (*SA m*))
assumes $g \in$ *carrier* (*UP* (*SA m*))
assumes *is-semialgebraic m S*
assumes $x \notin S$
assumes $x \in$ *carrier* ($Q_p{}^m$)
shows (*SA-poly-glue m S f g*) $n\ x = g\ n\ x$
⟨*proof*⟩

lemma *SA-poly-glue-to-Qp-poly1*:
assumes $f \in$ *carrier* (*UP* (*SA m*))
assumes $g \in$ *carrier* (*UP* (*SA m*))
assumes *is-semialgebraic m S*
assumes $x \in S$
shows *SA-poly-to-Qp-poly m x* (*SA-poly-glue m S f g*) = *SA-poly-to-Qp-poly m x f*
⟨*proof*⟩

lemma *SA-poly-glue-to-Qp-poly2*:
assumes $f \in$ *carrier* (*UP* (*SA m*))
assumes $g \in$ *carrier* (*UP* (*SA m*))
assumes *is-semialgebraic m S*
assumes $x \notin S$
assumes $x \in$ *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*⟩

14.13.3 Polynomials over the Valuation Ring

definition *integral-on* **where**
integral-on m B = $\{f \in$ *carrier* (*UP* (*SA m*)). $(\forall x \in B.\ \forall i.$ *SA-poly-to-Qp-poly m x f i* $\in \mathcal{O}_p)\}$

lemma *integral-on-memI*:
assumes $f \in$ *carrier* (*UP* (*SA m*))
assumes $\bigwedge x\ i.\ x \in B \Longrightarrow$ *SA-poly-to-Qp-poly m x f i* $\in \mathcal{O}_p$
shows $f \in$ *integral-on m B*
⟨*proof*⟩

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* $\in \mathcal{O}_p$
⟨*proof*⟩

lemma *one-integral-on*:
assumes $B \subseteq$ *carrier* $(Q_p{}^m)$
shows $\mathbf{1}_{UP\ (SA\ m)} \in$ *integral-on m B*
⟨*proof*⟩

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*⟩

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*⟩

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*
⟨*proof*⟩

lemma *integral-on-subring*:
assumes $B \subseteq$ *carrier* $(Q_p{}^m)$
shows *subring* (*integral-on m B*) (*UP* (*SA m*))
⟨*proof*⟩

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*⟩

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*⟩

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 $deg\ (SA\ m)\ f \leq d$
assumes $C \in poly\text{-}res\text{-}classes\ n\ d$
assumes *is-semialgebraic* $m\ B$
shows *is-semialgebraic* $m\ \{x \in B.\ SA\text{-}poly\text{-}to\text{-}Qp\text{-}poly\ m\ x\ f \in C\}$
⟨*proof*⟩

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*⟩

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*⟩

lemma(**in** *cring*) *minus-zero*:
$a \in carrier\ R \Longrightarrow a \ominus \mathbf{0} = a$
⟨*proof*⟩

lemma (**in** *UP-cring*) *taylor-expansion-at-zero*:
assumes $g \in carrier\ (UP\ R)$
shows *taylor-expansion* $R\ \mathbf{0}\ g = g$
⟨*proof*⟩
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 $n\ Fs\ S = atoms\text{-}of\ ((\cap)\ S\ `\ ((SA\text{-}zero\text{-}set\ n\ `\ Fs) \cup (SA\text{-}nonzero\text{-}set\ n\ `\ Fs)))$

lemma *SA-funs-to-SA-decomp-closed-0*:
assumes $Fs \subseteq carrier\ (SA\ n)$
assumes *is-semialgebraic n S*
shows $(\cap)\ S\ `\ ((SA\text{-}zero\text{-}set\ n\ `\ Fs) \cup (SA\text{-}nonzero\text{-}set\ n\ `\ Fs)) \subseteq semialg\text{-}sets\ n$
⟨*proof*⟩

lemma *SA-funs-to-SA-decomp-closed*:
assumes *finite Fs*
assumes $Fs \subseteq carrier\ (SA\ n)$
assumes *is-semialgebraic n S*
shows $SA\text{-}funs\text{-}to\text{-}SA\text{-}decomp\ n\ Fs\ S \subseteq semialg\text{-}sets\ n$
⟨*proof*⟩

lemma *SA-funs-to-SA-decomp-finite*:
assumes *finite Fs*
assumes $Fs \subseteq carrier\ (SA\ n)$
assumes *is-semialgebraic n S*
shows *finite* (*SA-funs-to-SA-decomp n Fs S*)
⟨*proof*⟩

lemma *SA-funs-to-SA-decomp-disjoint*:
assumes *finite Fs*
assumes $Fs \subseteq carrier\ (SA\ n)$
assumes *is-semialgebraic n S*
shows *disjoint* (*SA-funs-to-SA-decomp n Fs S*)
⟨*proof*⟩

lemma *pre-SA-funs-to-SA-decomp-in-algebra*:
shows $((\cap)\ S\ `\ (SA\text{-}zero\text{-}set\ n\ `\ Fs \cup SA\text{-}nonzero\text{-}set\ n\ `\ Fs)) \subseteq gen\text{-}boolean\text{-}algebra\ S\ (SA\text{-}zero\text{-}set\ n\ `\ Fs \cup SA\text{-}nonzero\text{-}set\ n\ `\ Fs)$
⟨*proof*⟩

lemma *SA-funs-to-SA-decomp-in-algebra*:
assumes *finite Fs*
shows $SA\text{-}funs\text{-}to\text{-}SA\text{-}decomp\ n\ Fs\ S \subseteq gen\text{-}boolean\text{-}algebra\ S\ (SA\text{-}zero\text{-}set\ n\ `\ Fs \cup SA\text{-}nonzero\text{-}set\ n\ `\ Fs)$
⟨*proof*⟩

lemma *SA-funs-to-SA-decomp-subset*:
assumes *finite Fs*
assumes $Fs \subseteq carrier\ (SA\ n)$
assumes *is-semialgebraic n S*
assumes $A \in SA\text{-}funs\text{-}to\text{-}SA\text{-}decomp\ n\ Fs\ S$
shows $A \subseteq S$
⟨*proof*⟩

lemma *SA-funs-to-SA-decomp-memE*:
assumes *finite Fs*
assumes $Fs \subseteq carrier\ (SA\ n)$

assumes *is-semialgebraic n S*
assumes $A \in$ (*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*⟩

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*⟩

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.