p-adic Hensel's Lemma

Aaron Crighton

December 14, 2021

Contents

Abstract

We formalize the ring of p-adic integers within the framework of the HOL-Algebra library. The carrier of the ring $\mathbb{Z}_p$ is formalized as the inverse limit of the residue rings $\mathbb{Z}/p^n\mathbb{Z}$ for a fixed prime p. We define a locale for reasoning about $\mathbb{Z}_p$ for a fixed prime p, and define an integer-valued valuation, as well as an extended-integer valued valuation on $\mathbb{Z}_p$ (where $0 \in \mathbb{Z}_p$ is the unique ring element mapped to ∞). Basic topological facts about the p-adic integers are formalized, including the completeness and sequential compactness of $\mathbb{Z}_p$. Taylor expansions of polynomials over a commutative ring are defined, culminating in the formalization of Hensel's Lemma based on a proof due to Keith Conrad [1].

theory *Function-Ring*
imports *HOL−Algebra.Ring HOL−Library.FuncSet HOL−Algebra.Module*
begin

This theory formalizes basic facts about the ring of extensional functions from a fixed set to a fixed ring. This will be useful for providing a generic framework for various constructions related to the p-adics such as polynomial evaluation and sequences. The rings of semialgebraic functions will be defined as subrings of these function rings, which will be necessary for the proof of p-adic quantifier elimination.

1 The Ring of Extensional Functions from a Fixed Base Set to a Fixed Base Ring

1.1 Basic Operations on Extensional Functions

definition *function-mult*:: *'c set* ⇒ (*'a*, *'b*) *ring-scheme* ⇒ (*'c* ⇒ *'a*) ⇒ (*'c* ⇒ *'a*) ⇒ (*'c* ⇒ *'a*) **where**
function-mult S R f g = (λ*x* ∈ *S*. (*f x*) ⊗$_R$ (*g x*))

abbreviation(*input*) *ring-function-mult*:: (*'a*, *'b*) *ring-scheme* ⇒ (*'a* ⇒ *'a*) ⇒ (*'a* ⇒ *'a*) ⇒ (*'a* ⇒ *'a*) **where**
ring-function-mult R f g ≡ *function-mult* (*carrier R*) *R f g*

definition *function-add*:: *'c set* ⇒ (*'a*, *'b*) *ring-scheme* ⇒ (*'c* ⇒ *'a*) ⇒ (*'c* ⇒ *'a*) ⇒ (*'c* ⇒ *'a*) **where**
function-add S R f g = (λ*x* ∈ *S*. (*f x*) ⊕$_R$ (*g x*))

abbreviation(*input*) *ring-function-add*:: (*'a*, *'b*) *ring-scheme* ⇒ (*'a* ⇒ *'a*) ⇒ (*'a* ⇒ *'a*) ⇒ (*'a* ⇒ *'a*) **where**
ring-function-add R f g ≡ *function-add* (*carrier R*) *R f g*

definition *function-one*:: *'c set* ⇒ (*'a*, *'b*) *ring-scheme* ⇒ (*'c* ⇒ *'a*) **where**
function-one S R = (λ*x* ∈ *S*. **1**$_R$)

abbreviation(*input*) *ring-function-one* :: (*'a*, *'b*) *ring-scheme* ⇒ (*'a* ⇒ *'a*) **where**
ring-function-one R ≡ *function-one* (*carrier R*) *R*

definition *function-zero*:: *'c set* ⇒ (*'a*, *'b*) *ring-scheme* ⇒ (*'c* ⇒ *'a*) **where**
function-zero S R = (λ*x* ∈ *S*. **0**$_R$)

abbreviation(*input*) *ring-function-zero* :: (*'a*, *'b*) *ring-scheme* ⇒ (*'a* ⇒ *'a*) **where**
ring-function-zero R ≡ *function-zero* (*carrier R*) *R*

definition *function-uminus*:: *'c set* ⇒ (*'a*, *'b*) *ring-scheme* ⇒ (*'c* ⇒ *'a*) ⇒ (*'c* ⇒ *'a*) **where**
function-uminus S R a = (λ *x* ∈ *S*. ⊖$_R$ (*a x*))

definition *ring-function-uminus*:: (*'a*, *'b*) *ring-scheme* ⇒ (*'a* ⇒ *'a*) ⇒ (*'a* ⇒ *'a*) **where**
ring-function-uminus R a = *function-uminus* (*carrier R*) *R a*

definition *function-scalar-mult*:: *'c set* ⇒ (*'a*, *'b*) *ring-scheme* ⇒ *'a* ⇒ (*'c* ⇒ *'a*) ⇒ (*'c* ⇒ *'a*) **where**
function-scalar-mult S R a f = (λ *x* ∈ *S*. *a* ⊗$_R$ (*f x*))

1.2 Defining the Ring of Extensional Functions

definition *function-ring*:: *'c set* ⇒ (*'a*, *'b*) *ring-scheme* ⇒ (*'a*, *'c* ⇒ *'a*) *module* **where**

function-ring S R = ⦇
  carrier = *extensional-funcset S* (*carrier R*),
  Group.monoid.mult = (*function-mult S R*),
  one = (*function-one S R*),
  zero = (*function-zero S R*),
  add = (*function-add S R*),
  smult = *function-scalar-mult S R* ⦈

The following locale consists of a struct R, and a distinguished set S which is meant to serve as the domain for a ring of functions $S \to carrier R$.

locale *struct-functions* =
  fixes *R* ::('*a*, '*b*) *partial-object-scheme* (**structure**)
    and *S* :: '*c set*

The following are locales which fix a ring R (which may be commutative, a domain, or a field) and a function ring F of extensional functions from a fixed set S to $carrier R$

locale *ring-functions* = *struct-functions* + *R?*: *ring R* +
  fixes *F* (**structure**)
  defines *F-def*: *F* ≡ *function-ring S R*

locale *cring-functions* = *ring-functions* + *R?*: *cring R*

locale *domain-functions* = *ring-functions* + *R?*: *domain R*

locale *field-functions* = *ring-functions* + *R?*: *field R*

sublocale *cring-functions* < *ring-functions*
  ⟨*proof*⟩

sublocale *domain-functions* < *ring-functions*
  ⟨*proof*⟩

sublocale *domain-functions* < *cring-functions*
  ⟨*proof*⟩

sublocale *field-functions* < *domain-functions*
  ⟨*proof*⟩

sublocale *field-functions* < *ring-functions*
  ⟨*proof*⟩

sublocale *field-functions* < *cring-functions*
  ⟨*proof*⟩

abbreviation(*input*) *ring-function-ring*:: ('*a*, '*b*) *ring-scheme* ⇒ ('*a*, '*a* ⇒ '*a*) *module* (*Fun*) **where**
ring-function-ring R ≡ *function-ring* (*carrier R*) *R*

1.3 Algebraic Properties of the Basic Operations

1.3.1 Basic Carrier Facts

lemma(**in** *ring-functions*) *function-ring-defs*:
carrier F = *extensional-funcset* S (*carrier* R)
$(\otimes_F)$ = (*function-mult* S R)
$(\oplus_F)$ = (*function-add* S R)
$\mathbf{1}_F$ = *function-one* S R
$\mathbf{0}_F$ = *function-zero* S R
$(\odot_F)$ = *function-scalar-mult* S R
⟨*proof*⟩

lemma(**in** *ring-functions*) *function-ring-car-memE*:
assumes $a \in$ *carrier* F
shows $a \in$ *extensional* S
$a \in S \to$ *carrier* R
⟨*proof*⟩

lemma(**in** *ring-functions*) *function-ring-car-closed*:
assumes $a \in S$
assumes $f \in$ *carrier* F
shows $f\ a \in$ *carrier* R
⟨*proof*⟩

lemma(**in** *ring-functions*) *function-ring-not-car*:
assumes $a \notin S$
assumes $f \in$ *carrier* F
shows $f\ a =$ *undefined*
⟨*proof*⟩

lemma(**in** *ring-functions*) *function-ring-car-eqI*:
assumes $f \in$ *carrier* F
assumes $g \in$ *carrier* F
assumes $\bigwedge a.\ a \in S \Longrightarrow f\ a = g\ a$
shows $f = g$
⟨*proof*⟩

lemma(**in** *ring-functions*) *function-ring-car-memI*:
assumes $\bigwedge a.\ a \in S \Longrightarrow f\ a \in$ *carrier* R
assumes $\bigwedge\ a.\ a \notin S \Longrightarrow f\ a =$ *undefined*
shows $f \in$ *carrier* F
⟨*proof*⟩

lemma(**in** *ring*) *function-ring-car-memI*:
assumes $\bigwedge a.\ a \in S \Longrightarrow f\ a \in$ *carrier* R
assumes $\bigwedge\ a.\ a \notin S \Longrightarrow f\ a =$ *undefined*
shows $f \in$ *carrier* (*function-ring* S R)
⟨*proof*⟩

1.3.2 Basic Multiplication Facts

lemma(**in** *ring-functions*) *function-mult-eval-car*:
assumes $a \in S$
assumes $f \in carrier\ F$
assumes $g \in carrier\ F$
shows $(f \otimes_F g)\ a = (f\ a) \otimes (g\ a)$
⟨*proof*⟩

lemma(**in** *ring-functions*) *function-mult-eval-closed*:
assumes $a \in S$
assumes $f \in carrier\ F$
assumes $g \in carrier\ F$
shows $(f \otimes_F g)\ a \in carrier\ R$
⟨*proof*⟩

lemma(**in** *ring-functions*) *fun-mult-closed*:
assumes $f \in carrier\ F$
assumes $g \in carrier\ F$
shows $f \otimes_F g \in carrier\ F$
⟨*proof*⟩

lemma(**in** *ring-functions*) *fun-mult-eval-assoc*:
assumes $x \in carrier\ F$
assumes $y \in carrier\ F$
assumes $z \in carrier\ F$
assumes $a \in S$
shows $(x \otimes_F y \otimes_F z)\ a = (x \otimes_F (y \otimes_F z))\ a$
⟨*proof*⟩

lemma(**in** *ring-functions*) *fun-mult-assoc*:
assumes $x \in carrier\ F$
assumes $y \in carrier\ F$
assumes $z \in carrier\ F$
shows $(x \otimes_F y \otimes_F z) = (x \otimes_F (y \otimes_F z))$
⟨*proof*⟩

1.3.3 Basic Addition Facts

lemma(**in** *ring-functions*) *fun-add-eval-car*:
assumes $a \in S$
assumes $f \in carrier\ F$
assumes $g \in carrier\ F$
shows $(f \oplus_F g)\ a = (f\ a) \oplus (g\ a)$
⟨*proof*⟩

lemma(**in** *ring-functions*) *fun-add-eval-closed*:
assumes $a \in S$
assumes $f \in carrier\ F$
assumes $g \in carrier\ F$

shows $(f \oplus_F g)\ a \in \mathit{carrier}\ R$
⟨*proof*⟩

lemma(**in** *ring-functions*) *fun-add-closed*:
assumes $f \in \mathit{carrier}\ F$
assumes $g \in \mathit{carrier}\ F$
shows $f \oplus_F g \in \mathit{carrier}\ F$
⟨*proof*⟩

lemma(**in** *ring-functions*) *fun-add-eval-assoc*:
assumes $x \in \mathit{carrier}\ F$
assumes $y \in \mathit{carrier}\ F$
assumes $z \in \mathit{carrier}\ F$
assumes $a \in S$
shows $(x \oplus_F y \oplus_F z)\ a = (x \oplus_F (y \oplus_F z))\ a$
⟨*proof*⟩

lemma(**in** *ring-functions*) *fun-add-assoc*:
assumes $x \in \mathit{carrier}\ F$
assumes $y \in \mathit{carrier}\ F$
assumes $z \in \mathit{carrier}\ F$
shows $x \oplus_F y \oplus_F z = x \oplus_F (y \oplus_F z)$
⟨*proof*⟩

lemma(**in** *ring-functions*) *fun-add-eval-comm*:
assumes $a \in S$
assumes $x \in \mathit{carrier}\ F$
assumes $y \in \mathit{carrier}\ F$
shows $(x \oplus_F y)\ a = (y \oplus_F x)\ a$
⟨*proof*⟩

lemma(**in** *ring-functions*) *fun-add-comm*:
assumes $x \in \mathit{carrier}\ F$
assumes $y \in \mathit{carrier}\ F$
shows $x \oplus_F y = y \oplus_F x$
⟨*proof*⟩

1.3.4 Basic Facts About the Multiplicative Unit

lemma(**in** *ring-functions*) *function-one-eval*:
assumes $a \in S$
shows $\mathbf{1}_F\ a = \mathbf{1}$
⟨*proof*⟩

lemma(**in** *ring-functions*) *function-one-closed*:
$\mathbf{1}_F \in \mathit{carrier}\ F$
⟨*proof*⟩

lemma(**in** *ring-functions*) *function-times-one-l*:

assumes $a \in$ *carrier* F
shows $\mathbf{1}_F \otimes_F a = a$
⟨*proof*⟩

lemma(**in** *ring-functions*) *function-times-one-r*:
assumes $a \in$ *carrier* F
shows $a \otimes_F \mathbf{1}_F = a$
⟨*proof*⟩

1.3.5 Basic Facts About the Additive Unit

lemma(**in** *ring-functions*) *function-zero-eval*:
assumes $a \in S$
shows $\mathbf{0}_F\ a = \mathbf{0}$
⟨*proof*⟩

lemma(**in** *ring-functions*) *function-zero-closed*:
$\mathbf{0}_F \in$*carrier* F
⟨*proof*⟩

lemma(**in** *ring-functions*) *fun-add-zeroL*:
assumes $a \in$ *carrier* F
shows $\mathbf{0}_F \oplus_F a = a$
⟨*proof*⟩

lemma(**in** *ring-functions*) *fun-add-zeroR*:
assumes $a \in$ *carrier* F
shows $a \oplus_F \mathbf{0}_F = a$
⟨*proof*⟩

1.3.6 Distributive Laws

lemma(**in** *ring-functions*) *function-mult-r-distr*:
assumes $x \in$ *carrier* F
assumes $y \in$ *carrier* F
assumes $z \in$ *carrier* F
shows $(x \oplus_F y) \otimes_F z = x \otimes_F z \oplus_F y \otimes_F z$
⟨*proof*⟩

lemma(**in** *ring-functions*) *function-mult-l-distr*:
assumes $x \in$ *carrier* F
assumes $y \in$ *carrier* F
assumes $z \in$ *carrier* F
shows $z \otimes_F (x \oplus_F y) = z \otimes_F x \oplus_F z \otimes_F y$
⟨*proof*⟩

1.3.7 Additive Inverses

lemma(**in** *ring-functions*) *function-uminus-closed*:
assumes $f \in$ *carrier* F

shows *function-uminus S R f* ∈ *carrier F*
⟨*proof*⟩

lemma(**in** *ring-functions*) *function-uminus-eval*:
assumes $a \in S$
assumes $f \in$ *carrier F*
shows (*function-uminus S R f*) $a = \ominus (f\ a)$
⟨*proof*⟩

lemma(**in** *ring-functions*) *function-uminus-add-r*:
assumes $a \in S$
assumes $f \in$ *carrier F*
shows $f \oplus_F$ *function-uminus S R f* $= \mathbf{0}_F$
⟨*proof*⟩

lemma(**in** *ring-functions*) *function-uminus-add-l*:
assumes $a \in S$
assumes $f \in$ *carrier F*
shows *function-uminus S R f* $\oplus_F f = \mathbf{0}_F$
⟨*proof*⟩

1.3.8 Scalar Multiplication

lemma(**in** *ring-functions*) *function-smult-eval*:
assumes $a \in$ *carrier R*
assumes $f \in$ *carrier F*
assumes $b \in S$
shows $(a \odot_F f)\ b = a \otimes (f\ b)$
⟨*proof*⟩

lemma(**in** *ring-functions*) *function-smult-closed*:
assumes $a \in$ *carrier R*
assumes $f \in$ *carrier F*
shows $a \odot_F f \in$ *carrier F*
⟨*proof*⟩

lemma(**in** *ring-functions*) *function-smult-assoc1*:
assumes $a \in$ *carrier R*
assumes $b \in$ *carrier R*
assumes $f \in$ *carrier F*
shows $b \odot_F (a \odot_F f) = (b \otimes a) \odot_F f$
⟨*proof*⟩

lemma(**in** *ring-functions*) *function-smult-assoc2*:
assumes $a \in$ *carrier R*
assumes $f \in$ *carrier F*
assumes $g \in$ *carrier F*
shows $(a \odot_F f) \otimes_F g = a \odot_F (f \otimes_F g)$
⟨*proof*⟩

lemma(**in** *ring-functions*) *function-smult-one*:
assumes $f \in carrier\ F$
shows $\mathbf{1}_{\odot_F} f = f$
⟨*proof*⟩

lemma(**in** *ring-functions*) *function-smult-l-distr*:
$[\!|\ a \in carrier\ R;\ b \in carrier\ R;\ x \in carrier\ F\ |\!] \Longrightarrow$
$(a \oplus b) \odot_F x = a \odot_F x \oplus_F b \odot_F x$
⟨*proof*⟩

lemma(**in** *ring-functions*) *function-smult-r-distr*:
$[\!|\ a \in carrier\ R;\ x \in carrier\ F;\ y \in carrier\ F\ |\!] \Longrightarrow$
$a \odot_F (x \oplus_F y) = a \odot_F x \oplus_F a \odot_F y$
⟨*proof*⟩

1.3.9 The Ring of Functions Forms an Algebra

lemma(**in** *ring-functions*) *function-ring-is-abelian-group*:
abelian-group F
⟨*proof*⟩

lemma(**in** *ring-functions*) *function-ring-is-monoid*:
monoid F
⟨*proof*⟩

lemma(**in** *ring-functions*) *function-ring-is-ring*:
ring F
⟨*proof*⟩

sublocale *ring-functions* < *F?*: *ring F*
⟨*proof*⟩

lemma(**in** *cring-functions*) *function-mult-comm*:
assumes $x \in carrier\ F$
assumes $y \in carrier\ F$
shows $x \otimes_F y = y \otimes_F x$
⟨*proof*⟩

lemma(**in** *cring-functions*) *function-ring-is-comm-monoid*:
comm-monoid F
⟨*proof*⟩

lemma(**in** *cring-functions*) *function-ring-is-cring*:
cring F
⟨*proof*⟩

lemma(**in** *cring-functions*) *function-ring-is-algebra*:
algebra R F

⟨*proof*⟩

lemma(**in** *ring-functions*) *function-uminus*:
assumes $f \in$ *carrier* F
shows $\ominus_F f = (\textit{function-uminus}\ S\ R)\ f$
⟨*proof*⟩

lemma(**in** *ring-functions*) *function-uminus-eval′*:
assumes $f \in$ *carrier* F
assumes $a \in S$
shows $(\ominus_F f)\ a = (\textit{function-uminus}\ S\ R)\ f\ a$
⟨*proof*⟩

lemma(**in** *ring-functions*) *function-uminus-eval″*:
assumes $f \in$ *carrier* F
assumes $a \in S$
shows $(\ominus_F f)\ a = \ominus\ (f\ a)$
⟨*proof*⟩

sublocale *cring-functions* < *F?*: *algebra R F*
⟨*proof*⟩

1.4 Constant Functions

definition *constant-function* **where**
constant-function S a =$(\lambda x \in S.\ a)$

abbreviation(**in** *ring-functions*)(*input*) *const* **where**
const ≡ *constant-function S*

lemma(**in** *ring-functions*) *constant-function-closed*:
assumes $a \in$ *carrier* R
shows *const a* ∈ *carrier F*
⟨*proof*⟩

lemma(**in** *ring-functions*) *constant-functionE*:
assumes $a \in$ *carrier* R
assumes $b \in S$
shows *const a b* = *a*
⟨*proof*⟩

lemma(**in** *ring-functions*) *constant-function-add*:
assumes $a \in$ *carrier* R
assumes $b \in$ *carrier* R
shows *const* $(a \oplus_R b) = (\textit{const}\ a) \oplus_F (\textit{const}\ b)$
⟨*proof*⟩

lemma(**in** *ring-functions*) *constant-function-mult*:
assumes $a \in$ *carrier* R

assumes $b \in$ *carrier* R
shows *const* $(a \otimes_R b) = (const\ a) \otimes_F (const\ b)$
⟨*proof*⟩

lemma(**in** *ring-functions*) *constant-function-minus*:
assumes $a \in$ *carrier* R
shows $\ominus_F(const\ a) = (const\ (\ominus_R\ a))$
⟨*proof*⟩

lemma(**in** *ring-functions*) *function-one-is-constant*:
const $\mathbf{1} = \mathbf{1}_F$
⟨*proof*⟩

lemma(**in** *ring-functions*) *function-zero-is-constant*:
const $\mathbf{0} = \mathbf{0}_F$
⟨*proof*⟩

1.5 Special Examples of Functions Rings

1.5.1 Functions from the Carrier of a Ring to Itself

locale *U-function-ring* = *ring*

locale *U-function-cring* = *U-function-ring* + *cring*

sublocale *U-function-ring* < *S?*: *struct-functions R carrier R*
⟨*proof*⟩

sublocale *U-function-ring* < *FunR?*: *ring-functions R carrier R Fun R*
⟨*proof*⟩

sublocale *U-function-cring* < *FunR?*: *cring-functions R carrier R Fun R*
⟨*proof*⟩

abbreviation(**in** *U-function-ring*)(*input*) *ring-compose* :: $('a \Rightarrow 'a) \Rightarrow ('a \Rightarrow 'a)$
$\Rightarrow ('a \Rightarrow 'a)$ **where**
ring-compose ≡ *compose* (*carrier R*)

lemma(**in** *U-function-ring*) *ring-function-ring-comp*:
assumes $f \in$ *carrier* (*Fun R*)
assumes $g \in$ *carrier* (*Fun R*)
shows *ring-compose f g* ∈ *carrier* (*Fun R*)
⟨*proof*⟩

abbreviation(**in** *U-function-ring*)(*input*) *ring-const* (𝔠1) **where**
ring-const ≡ *constant-function* (*carrier R*)

lemma(**in** *ring-functions*) *function-nat-pow-eval*:
assumes $f \in$ *carrier F*
assumes $s \in S$

shows $(f[\widehat{\ }]_F(n::nat))\ s = (f\ s)[\widehat{\ }]n$
⟨*proof*⟩

context *U-function-ring*
begin

definition *a-translate* :: $'a \Rightarrow 'a \Rightarrow 'a$ **where**
a-translate = $(\lambda\ r \in carrier\ R.\ restrict\ ((add\ R)\ r)\ (carrier\ R))$

definition *m-translate* :: $'a \Rightarrow 'a \Rightarrow 'a$ **where**
m-translate = $(\lambda\ r \in carrier\ R.\ restrict\ ((mult\ R)\ r)\ (carrier\ R))$

definition *nat-power* :: *nat* $\Rightarrow 'a \Rightarrow 'a$ **where**
nat-power = $(\lambda(n::nat).\ restrict\ (\lambda a.\ \ a[\widehat{\ }]_R n)\ (carrier\ R))$

Restricted operations are in Fs

lemma *a-translate-functions*:
assumes $c \in carrier\ R$
shows *a-translate* $c \in carrier\ (Fun\ R)$
⟨*proof*⟩

lemma *m-translate-functions*:
assumes $c \in carrier\ R$
shows *m-translate* $c \in carrier\ (Fun\ R)$
⟨*proof*⟩

lemma *nat-power-functions*:
shows *nat-power* $n \in carrier\ (Fun\ R)$
⟨*proof*⟩

Restricted operations simps

lemma *a-translate-eq*:
assumes $c \in carrier\ R$
assumes $a \in carrier\ R$
shows *a-translate* $c\ a = c \oplus a$
⟨*proof*⟩

lemma *a-translate-eq'*:
assumes $c \in carrier\ R$
assumes $a \notin carrier\ R$
shows *a-translate* $c\ a = undefined$
⟨*proof*⟩

lemma *a-translate-eq''*:
assumes $c \notin carrier\ R$
shows *a-translate* $c = undefined$
⟨*proof*⟩

lemma *m-translate-eq*:
assumes $c \in$ *carrier* R
assumes $a \in$ *carrier* R
shows *m-translate* $c\ a = c \otimes a$
⟨*proof*⟩

lemma *m-translate-eq′*:
assumes $c \in$ *carrier* R
assumes $a \notin$ *carrier* R
shows *m-translate* $c\ a =$ *undefined*
⟨*proof*⟩

lemma *m-translate-eq″*:
assumes $c \notin$ *carrier* R
shows *m-translate* $c =$ *undefined*
⟨*proof*⟩

lemma *nat-power-eq*:
assumes $a \in$ *carrier* R
shows *nat-power* $n\ a = a[\widehat{}]_R\ n$
⟨*proof*⟩

lemma *nat-power-eq′*:
assumes $a \notin$ *carrier* R
shows *nat-power* $n\ a =$ *undefined*
⟨*proof*⟩

Constant ring_function properties

lemma *constant-function-eq*:
assumes $a \in$ *carrier* R
assumes $b \in$ *carrier* R
shows $\mathfrak{c}_a\ b = a$
⟨*proof*⟩

lemma *constant-function-eq′*:
assumes $a \in$ *carrier* R
assumes $b \notin$ *carrier* R
shows $\mathfrak{c}_a\ b =$ *undefined*
⟨*proof*⟩

Compound expressions from algebraic operations

end

definition *monomial-function* **where**
monomial-function $R\ c\ (n{::}nat) = (\lambda\ x \in$ *carrier* $R.\ c \otimes_R (x[\widehat{}]_R n))$

context *U-function-ring*
begin

abbreviation *monomial* **where**
monomial ≡ *monomial-function R*

lemma *monomial-functions*:
assumes *c* ∈ *carrier R*
shows *monomial c n* ∈ *carrier* (*Fun R*)
⟨*proof*⟩

definition *ring-id* **where**
ring-id ≡ *restrict* (λ*x*. *x*) (*carrier R*)

lemma *ring-id-closed*[*simp*]:
ring-id ∈ *carrier* (*Fun R*)
⟨*proof*⟩

lemma *ring-id-eval*:
assumes *a* ∈ *carrier R*
shows *ring-id a* = *a*
⟨*proof*⟩

lemma *constant-a-trans*:
assumes *a* ∈*carrier R*
shows *m-translate a* = $\mathfrak{c}_a$ ⊗$_{Fun\ R}$ *ring-id*
⟨*proof*⟩

polynomials in one variable

fun *polynomial* :: '*a list* ⇒ ('*a* ⇒ '*a*) **where**
polynomial [] = **0**$_{Fun\ R}$ |
polynomial (*a*#*as*) = (λ*x* ∈ *carrier R*. *a* ⊕ *x* ⊗ (*polynomial as x*))

lemma *polynomial-induct-lemma*:
assumes *f* ∈ *carrier* (*Fun R*)
assumes *a* ∈ *carrier R*
shows (λ*x* ∈ *carrier R*. *a* ⊕ *x* ⊗ (*f x*)) ∈ *carrier* (*Fun R*)
⟨*proof*⟩

lemma *polynomial-function*:
shows *set as* ⊆ *carrier R* ⟹ *polynomial as* ∈ *carrier* (*Fun R*)
⟨*proof*⟩

lemma *polynomial-constant*:
assumes *a* ∈ *carrier R*
shows *polynomial* [*a*] = $\mathfrak{c}_a$
⟨*proof*⟩

end

1.5.2 Sequences Indexed by the Natural Numbers

definition *nat-seqs* (-^{ω})**where**
nat-seqs $R \equiv$ *function-ring* (*UNIV*::*nat set*) R

abbreviation(*input*) *closed-seqs* **where**
closed-seqs $R \equiv$ *carrier* (R^{ω})

lemma *closed-seqs-memI*:
assumes $\bigwedge k.\ s\ k \in$ *carrier* R
shows $s \in$ *closed-seqs* R
⟨*proof*⟩

lemma *closed-seqs-memE*:
assumes $s \in$ *closed-seqs* R
shows $s\ k \in$ *carrier* R
⟨*proof*⟩

definition *is-constant-fun* **where**
is-constant-fun $R\ f = (\exists x \in$ *carrier* $R.\ f =$ *constant-function* (*carrier* R) $R\ x)$

definition *is-constant-seq* **where**
is-constant-seq $R\ s = (\exists x \in$ *carrier* $R.\ s =$ *constant-function* (*UNIV*::*nat set*) $x)$

lemma *is-constant-seqI*:
fixes a
assumes $s \in$ *closed-seqs* R
assumes $\bigwedge k.\ s\ k = a$
shows *is-constant-seq* $R\ s$
⟨*proof*⟩

lemma *is-constant-seqE*:
assumes *is-constant-seq* $R\ s$
assumes $s\ k = a$
shows $s\ n = a$
⟨*proof*⟩

lemma *is-constant-seq-imp-closed*:
assumes *is-constant-seq* $R\ s$
shows $s \in$ *closed-seqs* R
⟨*proof*⟩

context *U-function-ring*
begin

Sequence sums and products are closed

lemma *seq-plus-closed*:
assumes $s \in$ *closed-seqs* R
assumes $s' \in$ *closed-seqs* R
shows $s \oplus_{R^{\omega}} s' \in$ *closed-seqs* R

⟨*proof*⟩

lemma *seq-mult-closed*:
assumes $s \in$ *closed-seqs* R
assumes $s' \in$ *closed-seqs* R
shows $s \otimes_{R^\omega} s' \in$ *closed-seqs* R
⟨*proof*⟩

lemma *constant-function-comp-is-closed-seq*:
assumes $a \in$ *carrier* R
assumes $s \in$ *closed-seqs* R
shows $(const\ a \circ s) \in$ *closed-seqs* R
⟨*proof*⟩

lemma *constant-function-comp-is-constant-seq*:
assumes $a \in$ *carrier* R
assumes $s \in$ *closed-seqs* R
shows *is-constant-seq* R $((const\ a) \circ s)$
⟨*proof*⟩

lemma *function-comp-is-closed-seq*:
assumes $s \in$ *closed-seqs* R
assumes $f \in$ *carrier* $(Fun\ R)$
shows $f \circ s \in$ *closed-seqs* R
⟨*proof*⟩

lemma *function-sum-comp-is-seq-sum*:
assumes $s \in$ *closed-seqs* R
assumes $f \in$ *carrier* $(Fun\ R)$
assumes $g \in$ *carrier* $(Fun\ R)$
shows $(f \oplus_{Fun\ R} g) \circ s = (f \circ s) \oplus_{R^\omega} (g \circ s)$
⟨*proof*⟩

lemma *function-mult-comp-is-seq-mult*:
assumes $s \in$ *closed-seqs* R
assumes $f \in$ *carrier* $(Fun\ R)$
assumes $g \in$ *carrier* $(Fun\ R)$
shows $(f \otimes_{Fun\ R} g) \circ s = (f \circ s) \otimes_{R^\omega} (g \circ s)$
⟨*proof*⟩

lemma *seq-plus-simp*:
assumes $s \in$ *closed-seqs* R
assumes $t \in$ *closed-seqs* R
shows $(s \oplus_{R^\omega} t)\ k = s\ k \oplus t\ k$
⟨*proof*⟩

lemma *seq-mult-simp*:
assumes $s \in$ *closed-seqs* R
assumes $t \in$ *closed-seqs* R

shows $(s \otimes_{R^\omega} t)\ k = s\ k \otimes t\ k$
⟨*proof*⟩

lemma *seq-one-simp*:
$\mathbf{1}_{R^\omega}\ k = \mathbf{1}$
⟨*proof*⟩

lemma *seq-zero-simp*:
$\mathbf{0}_{R^\omega}\ k = \mathbf{0}$
⟨*proof*⟩

lemma(**in** *U-function-ring*) *ring-id-seq-comp*:
assumes $s \in$ *closed-seqs R*
shows *ring-id* $\circ\ s = s$
⟨*proof*⟩

lemma(**in** *U-function-ring*) *ring-seq-smult-closed*:
assumes $s \in$ *closed-seqs R*
assumes $a \in$ *carrier R*
shows $a \odot_{R^\omega} s \in$ *closed-seqs R*
⟨*proof*⟩

lemma(**in** *U-function-ring*) *ring-seq-smult-eval*:
assumes $s \in$ *closed-seqs R*
assumes $a \in$ *carrier R*
shows $(a \odot_{R^\omega} s)\ k = a \otimes (s\ k)$
⟨*proof*⟩

lemma(**in** *U-function-ring*) *ring-seq-smult-comp-assoc*:
assumes $s \in$ *closed-seqs R*
assumes $f \in$ *carrier* (*Fun R*)
assumes $a \in$ *carrier R*
shows $((a \odot_{Fun\ R} f) \circ s) = a \odot_{R^\omega} (f \circ s)$
⟨*proof*⟩

end

2 Extensional Maps Between the Carriers of two Structures

definition *struct-maps* :: ('*a*, '*c*) *partial-object-scheme* ⇒ ('*b*, '*d*) *partial-object-scheme*

⇒ ('*a* ⇒ '*b*) *set* **where**
struct-maps T S = {*f*. (*f* ∈ (*carrier T*) → (*carrier S*)) ∧ *f* = *restrict f* (*carrier T*) }

definition *to-struct-map* **where**
to-struct-map T f = *restrict f* (*carrier T*)

lemma *to-struct-map-closed*:
assumes $f \in (carrier\ T) \to (carrier\ S)$
shows $to\text{-}struct\text{-}map\ T\ f \in (struct\text{-}maps\ T\ S)$
⟨*proof*⟩

lemma *struct-maps-memI*:
assumes $\bigwedge x.\ x \in carrier\ T \Longrightarrow f\ x \in carrier\ S$
assumes $\bigwedge x.\ x \notin carrier\ T \Longrightarrow f\ x = undefined$
shows $f \in struct\text{-}maps\ T\ S$
⟨*proof*⟩

lemma *struct-maps-memE*:
assumes $f \in struct\text{-}maps\ T\ S$
shows $\bigwedge x.\ x \in carrier\ T \Longrightarrow f\ x \in carrier\ S$
$\bigwedge x.\ x \notin carrier\ T \Longrightarrow f\ x = undefined$
⟨*proof*⟩

An abbreviation for restricted composition of function of functions. This is necessary for the composition of two struct maps to again be a struct map.

abbreviation(*input*) *rcomp*
where $rcomp \equiv FuncSet.compose$

lemma *struct-map-comp*:
assumes $g \in (struct\text{-}maps\ T\ S)$
assumes $f \in (struct\text{-}maps\ S\ U)$
shows $rcomp\ (carrier\ T)\ f\ g \in (struct\text{-}maps\ T\ U)$
⟨*proof*⟩

lemma *r-comp-is-compose*:
assumes $g \in (struct\text{-}maps\ T\ S)$
assumes $f \in (struct\text{-}maps\ S\ U)$
assumes $a \in (carrier\ T)$
shows $(rcomp\ (carrier\ T)\ f\ g)\ a = (f \circ g)\ a$
⟨*proof*⟩

lemma *r-comp-not-in-car*:
assumes $g \in (struct\text{-}maps\ T\ S)$
assumes $f \in (struct\text{-}maps\ S\ U)$
assumes $a \notin (carrier\ T)$
shows $(rcomp\ (carrier\ T)\ f\ g)\ a = undefined$
⟨*proof*⟩

The reverse composition of two struct maps:

definition *pullback* ::
$('a, 'd)\ partial\text{-}object\text{-}scheme \Rightarrow ('a \Rightarrow 'b) \Rightarrow ('b \Rightarrow 'c) \Rightarrow ('a \Rightarrow 'c)$ **where**
$pullback\ T\ f\ g = rcomp\ (carrier\ T)\ g\ f$

lemma *pullback-closed*:

assumes $f \in (\mathit{struct\text{-}maps}\ T\ S)$
assumes $g \in (\mathit{struct\text{-}maps}\ S\ U)$
shows $\mathit{pullback}\ T\ f\ g \in (\mathit{struct\text{-}maps}\ T\ U)$
⟨*proof*⟩

Composition of struct maps which takes the structure itself rather than the carrier as a parameter:

definition *pushforward* ::
$('a, 'd)\ \mathit{partial\text{-}object\text{-}scheme} \Rightarrow ('b \Rightarrow 'c) \Rightarrow ('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'c)$ **where**
$\mathit{pushforward}\ T\ f\ g \equiv \mathit{rcomp}\ (\mathit{carrier}\ T)\ f\ g$

lemma *pushforward-closed*:
assumes $g \in (\mathit{struct\text{-}maps}\ T\ S)$
assumes $f \in (\mathit{struct\text{-}maps}\ S\ U)$
shows $\mathit{pushforward}\ T\ f\ g \in (\mathit{struct\text{-}maps}\ T\ U)$
⟨*proof*⟩

end
theory *Cring-Poly*
imports *HOL−Algebra.UnivPoly HOL−Algebra.Subrings Function-Ring*
begin

This theory extends the material in *HOL−Algebra.UnivPoly*. The main additions are material on Taylor expansions of polynomials and polynomial derivatives, and various applications of the universal property of polynomial evaluation. These include construing polynomials as functions from the base ring to itself, composing one polynomial with another, and extending homomorphisms between rings to homomoprhisms of their polynomial rings. These formalizations are necessary components of the proof of Hensel's lemma for p-adic integers, and for the proof of p-adic quantifier elimination.

lemma(**in** *ring*) *ring-hom-finsum*:
assumes $h \in \mathit{ring\text{-}hom}\ R\ S$
assumes $\mathit{ring}\ S$
assumes $\mathit{finite}\ I$
assumes $F \in I \to \mathit{carrier}\ R$
shows $h\ (\mathit{finsum}\ R\ F\ I) = \mathit{finsum}\ S\ (h \circ F)\ I$
⟨*proof*⟩

lemma(**in** *ring*) *ring-hom-a-inv*:
assumes $\mathit{ring}\ S$
assumes $h \in \mathit{ring\text{-}hom}\ R\ S$
assumes $b \in \mathit{carrier}\ R$
shows $h\ (\ominus\ b) = \ \ominus_S\ h\ b$
⟨*proof*⟩

lemma(**in** *ring*) *ring-hom-minus*:

assumes *ring S*
assumes $h \in$ *ring-hom R S*
assumes $a \in$ *carrier R*
assumes $b \in$ *carrier R*
shows $h\ (a \ominus b) = h\ a \ominus_S h\ b$
⟨*proof*⟩

lemma *ring-hom-nat-pow*:
assumes *ring R*
assumes *ring S*
assumes $h \in$ *ring-hom R S*
assumes $a \in$ *carrier R*
shows $h\ (a[\widehat{}]_R(n::nat)) = (h\ a)[\widehat{}]_S(n::nat)$
⟨*proof*⟩

lemma (**in** *ring*) *Units-not-right-zero-divisor*:
assumes $a \in$ *Units R*
assumes $b \in$ *carrier R*
assumes $a \otimes b = \mathbf{0}$
shows $b = \mathbf{0}$
⟨*proof*⟩

lemma (**in** *ring*) *Units-not-left-zero-divisor*:
assumes $a \in$ *Units R*
assumes $b \in$ *carrier R*
assumes $b \otimes a = \mathbf{0}$
shows $b = \mathbf{0}$
⟨*proof*⟩

lemma (**in** *cring*) *finsum-remove*:
assumes $\bigwedge i.\ i \in Y \Longrightarrow f\ i \in$ *carrier R*
assumes *finite Y*
assumes $i \in Y$
shows *finsum R f Y* $= f\ i \oplus$ *finsum R f* $(Y - \{i\})$
⟨*proof*⟩

type-synonym *degree* = *nat*

The composition of two ring homomorphisms is a ring homomorphism

lemma *ring-hom-compose*:
assumes *ring R*
assumes *ring S*
assumes *ring T*
assumes $h \in$ *ring-hom R S*
assumes $g \in$ *ring-hom S T*
assumes $\bigwedge c.\ c \in$ *carrier R* $\Longrightarrow f\ c = g\ (h\ c)$
shows $f \in$ *ring-hom R T*
⟨*proof*⟩

3 Basic Notions about Polynomials

context *UP-ring*
begin

rings are closed under monomial terms

lemma *monom-term-car*:
  assumes $c \in$ *carrier* R
  assumes $x \in$ *carrier* R
  shows $c \otimes x[\hat{}](n::nat) \in$ *carrier* R
  ⟨*proof*⟩

Univariate polynomial ring over R

lemma *P-is-UP-ring*:
UP-ring R
  ⟨*proof*⟩

Degree function

abbreviation(*input*) *degree* **where**
degree $f \equiv$ *deg* R f

lemma *UP-car-memI*:
  assumes $\bigwedge n.\ n > k \Longrightarrow p\ n = \mathbf{0}$
  assumes $\bigwedge n.\ p\ n \in$ *carrier* R
  shows $p \in$ *carrier* P
⟨*proof*⟩

lemma(**in** *UP-cring*) *UP-car-memI'*:
  assumes $\bigwedge x.\ g\ x \in$ *carrier* R
  assumes $\bigwedge x.\ x > k \Longrightarrow g\ x = \mathbf{0}$
  shows $g \in$ *carrier* $(UP\ R)$
⟨*proof*⟩

lemma(**in** *UP-cring*) *UP-car-memE*:
  assumes $g \in$ *carrier* $(UP\ R)$
  shows $\bigwedge x.\ g\ x \in$ *carrier* R
        $\bigwedge x.\ x > (deg\ R\ g) \Longrightarrow g\ x = \mathbf{0}$
  ⟨*proof*⟩

end

3.1 Lemmas About Coefficients

context *UP-ring*
begin

The goal here is to reduce dependence on the function coeff from Univ_Poly, in favour of using a polynomial itself as its coefficient function.

lemma *coeff-simp*:

assumes $f \in$ *carrier* P
shows *coeff* $(UP\ R)\ \ f = f$
⟨*proof*⟩

Coefficients are in R

lemma *cfs-closed*:
assumes $f \in$ *carrier* P
shows $f\ n \in$ *carrier* R
⟨*proof*⟩

lemma *cfs-monom*:
$a \in$ *carrier* R ==> $(monom\ P\ a\ m)\ n = (if\ m{=}n\ then\ a\ else\ \mathbf{0})$
⟨*proof*⟩

lemma *cfs-zero* [*simp*]: $\mathbf{0}_P\ n = \mathbf{0}$
⟨*proof*⟩

lemma *cfs-one* [*simp*]: $\mathbf{1}_P\ n = (if\ n{=}0\ then\ \mathbf{1}\ else\ \mathbf{0})$
⟨*proof*⟩

lemma *cfs-smult* [*simp*]:
[| $a \in$ *carrier* R; $p \in$ *carrier* P |] ==> $(a \odot_P p)\ n = a \otimes p\ n$
⟨*proof*⟩

lemma *cfs-add* [*simp*]:
[| $p \in$ *carrier* P; $q \in$ *carrier* P |] ==> $(p \oplus_P q)\ n = p\ n \oplus q\ n$
⟨*proof*⟩

lemma *cfs-a-inv* [*simp*]:
assumes R: $p \in$ *carrier* P
shows $(\ominus_P p)\ n = \ominus\ (p\ n)$
⟨*proof*⟩

lemma *cfs-minus* [*simp*]:
[| $p \in$ *carrier* P; $q \in$ *carrier* P |] ==> $(p \ominus_P q)\ n = p\ n \ominus q\ n$
⟨*proof*⟩

lemma *cfs-monom-mult-r*:
assumes $p \in$ *carrier* P
assumes $a \in$ *carrier* R
shows $(monom\ P\ a\ n \otimes_P p)\ (k + n) = a \otimes p\ k$
⟨*proof*⟩

lemma(**in** *UP-cring*) *cfs-monom-mult-l*:
assumes $p \in$ *carrier* P
assumes $a \in$ *carrier* R
shows $(p \otimes_P monom\ P\ a\ n)\ (k + n) = a \otimes p\ k$
⟨*proof*⟩

lemma(**in** *UP-cring*) *cfs-monom-mult-l'*:
assumes $f \in carrier\ P$
assumes $a \in carrier\ R$
assumes $m \geq n$
shows $(f \otimes_P (monom\ P\ a\ n))\ m = a \otimes (f\ (m - n))$
⟨*proof*⟩

lemma(**in** *UP-cring*) *cfs-monom-mult-r'*:
assumes $f \in carrier\ P$
assumes $a \in carrier\ R$
assumes $m \geq n$
shows $((monom\ P\ a\ n) \otimes_P f)\ m = a \otimes (f\ (m - n))$
⟨*proof*⟩
end

3.2 Degree Bound Lemmas

context *UP-ring*
begin

lemma *bound-deg-sum*:
assumes $f \in carrier\ P$
assumes $g \in carrier\ P$
assumes $degree\ f \leq n$
assumes $degree\ g \leq n$
shows $degree\ (f \oplus_P g) \leq n$
⟨*proof*⟩

lemma *bound-deg-sum'*:
assumes $f \in carrier\ P$
assumes $g \in carrier\ P$
assumes $degree\ f < n$
assumes $degree\ g < n$
shows $degree\ (f \oplus_P g) < n$
⟨*proof*⟩

lemma *equal-deg-sum*:
assumes $f \in carrier\ P$
assumes $g \in carrier\ P$
assumes $degree\ f < n$
assumes $degree\ g = n$
shows $degree\ (f \oplus_P g) = n$
⟨*proof*⟩

lemma *equal-deg-sum'*:
assumes $f \in carrier\ P$
assumes $g \in carrier\ P$
assumes $degree\ g < n$
assumes $degree\ f = n$

shows *degree* $(f \oplus_P g) = n$
⟨*proof*⟩

lemma *degree-of-sum-diff-degree*:
assumes $p \in$ *carrier* P
assumes $q \in$ *carrier* P
assumes *degree* $q <$ *degree* p
shows *degree* $(p \oplus_P q) =$ *degree* p
⟨*proof*⟩

lemma *degree-of-difference-diff-degree*:
assumes $p \in$ *carrier* P
assumes $q \in$ *carrier* P
assumes *degree* $q <$ *degree* p
shows *degree* $(p \ominus_P q) =$ *degree* p
⟨*proof*⟩

lemma (**in** *UP-ring*) *deg-diff-by-const*:
assumes $g \in$ *carrier* $(UP\ R)$
assumes $a \in$ *carrier* R
assumes $h = g \ominus_{UP\ R}$ *up-ring.monom* $(UP\ R)\ a\ 0$
shows *deg* $R\ g =$ *deg* $R\ h$
⟨*proof*⟩

lemma (**in** *UP-ring*) *deg-diff-by-const′*:
assumes $g \in$ *carrier* $(UP\ R)$
assumes $a \in$ *carrier* R
assumes $h = g \ominus_{UP\ R}$ *up-ring.monom* $(UP\ R)\ a\ 0$
shows *deg* $R\ g =$ *deg* $R\ h$
⟨*proof*⟩

lemma(**in** *UP-ring*) *deg-gtE*:
assumes $p \in$ *carrier* P
assumes $i >$ *deg* $R\ p$
shows $p\ i = \mathbf{0}$
⟨*proof*⟩
end

3.3 Leading Term Function

definition *leading-term* **where**
leading-term $R\ f =$ *monom* $(UP\ R)$ $(f\ (deg\ R\ f))$ $(deg\ R\ f)$

context *UP-ring*
begin

abbreviation(*input*) *ltrm* **where**
ltrm $f \equiv$ *monom* P $(f\ (deg\ R\ f))$ $(deg\ R\ f)$

leading term is a polynomial

lemma *ltrm-closed*:
assumes $f \in carrier\ P$
shows $ltrm\ f \in carrier\ P$
⟨*proof*⟩

Simplified coefficient function description for leading term

lemma *ltrm-coeff*:
assumes $f \in carrier\ P$
shows $coeff\ P\ (ltrm\ f)\ n\ = (if\ (n = degree\ f)\ then\ (f\ (degree\ f))\ else\ \mathbf{0})$
⟨*proof*⟩

lemma *ltrm-cfs*:
assumes $f \in carrier\ P$
shows $(ltrm\ f)\ n\ = (if\ (n = degree\ f)\ then\ (f\ (degree\ f))\ else\ \mathbf{0})$
⟨*proof*⟩

lemma *ltrm-cfs-above-deg*:
assumes $f \in carrier\ P$
assumes $n > degree\ f$
shows $ltrm\ f\ n = \mathbf{0}$
⟨*proof*⟩

The leading term of f has the same degree as f

lemma *deg-ltrm*:
assumes $f \in carrier\ P$
shows $degree\ (ltrm\ f) = degree\ f$
⟨*proof*⟩

Subtracting the leading term yields a drop in degree

lemma *minus-ltrm-degree-drop*:
assumes $f \in carrier\ P$
assumes $degree\ f = Suc\ n$
shows $degree\ (f \ominus_P (ltrm\ f)) \leq n$
⟨*proof*⟩

lemma *ltrm-decomp*:
assumes $f \in carrier\ P$
assumes $degree\ f > (0::nat)$
obtains g **where** $g \in carrier\ P \wedge f = g \oplus_P (ltrm\ f) \wedge degree\ g < degree\ f$
⟨*proof*⟩

leading term of a sum

lemma *coeff-of-sum-diff-degree0*:
assumes $p \in carrier\ P$
assumes $q \in carrier\ P$
assumes $degree\ q < n$
shows $(p \oplus_P q)\ n = p\ n$
⟨*proof*⟩

lemma *coeff-of-sum-diff-degree1*:
assumes $p \in$ *carrier* P
assumes $q \in$ *carrier* P
assumes *degree* $q <$ *degree* p
shows $(p \oplus_P q)$ (*degree* p) = p (*degree* p)
⟨*proof*⟩

lemma *ltrm-of-sum-diff-degree*:
assumes $p \in$ *carrier* P
assumes $q \in$ *carrier* P
assumes *degree* $p >$ *degree* q
shows *ltrm* $(p \oplus_P q)$ = *ltrm* p
⟨*proof*⟩

leading term of a monomial

lemma *ltrm-monom*:
assumes $a \in$ *carrier* R
assumes f = *monom* P a n
shows *ltrm* $f = f$
⟨*proof*⟩

lemma *ltrm-monom-simp*:
assumes $a \in$ *carrier* R
shows *ltrm* (*monom* P a n) = *monom* P a n
⟨*proof*⟩

lemma *ltrm-inv-simp*[*simp*]:
assumes $f \in$ *carrier* P
shows *ltrm* (*ltrm* f) = *ltrm* f
⟨*proof*⟩

lemma *ltrm-deg-0*:
assumes $p \in$ *carrier* P
assumes *degree* $p = 0$
shows *ltrm* $p = p$
⟨*proof*⟩

lemma *ltrm-prod-ltrm*:
assumes $p \in$ *carrier* P
assumes $q \in$ *carrier* P
shows *ltrm* ((*ltrm* p) $\otimes_P$ (*ltrm* q)) = (*ltrm* p) $\otimes_P$ (*ltrm* q)
⟨*proof*⟩

lead coefficient function

abbreviation(*input*) *lcf* **where**
lcf $p \equiv p$ (*deg* R p)

lemma(**in** *UP-ring*) *lcf-ltrm*:
ltrm p = *monom P* (*lcf p*) (*degree p*)
⟨*proof*⟩

lemma *lcf-closed*:
assumes *f* ∈ *carrier P*
shows *lcf f* ∈ *carrier R*
⟨*proof*⟩

lemma(**in** *UP-cring*) *lcf-monom*:
assumes *a* ∈ *carrier R*
shows *lcf* (*monom P a n*) = *a lcf* (*monom* (*UP R*) *a n*) = *a*
⟨*proof*⟩

end

Function which truncates a polynomial by removing the leading term

definition *truncate* **where**
truncate R f = *f* ⊖$_{(UP\ R)}$ (*leading-term R f*)

context *UP-ring*
begin

abbreviation(*input*) *trunc* **where**
trunc ≡ *truncate R*

lemma *trunc-closed*:
assumes *f* ∈ *carrier P*
shows *trunc f* ∈ *carrier P*
⟨*proof*⟩

lemma *trunc-simps*:
assumes *f* ∈ *carrier P*
shows *f* = (*trunc f*) ⊕$_P$ (*ltrm f*)
f ⊖$_P$ (*trunc f*) = *ltrm f*
⟨*proof*⟩

lemma *trunc-zero*:
assumes *f* ∈ *carrier P*
assumes *degree f* = *0*
shows *trunc f* = **0**$_P$
⟨*proof*⟩

lemma *trunc-degree*:
assumes *f* ∈ *carrier P*
assumes *degree f* > *0*
shows *degree* (*trunc f*) < *degree f*

⟨*proof*⟩

The coefficients of trunc agree with f for small degree

lemma *trunc-cfs*:
assumes $p \in carrier\ P$
assumes $n < degree\ p$
shows $(trunc\ p)\ n = p\ n$
⟨*proof*⟩

monomial predicate

definition *is-UP-monom* **where**
$is\text{-}UP\text{-}monom = (\lambda f.\ f \in carrier\ (UP\ R) \wedge f = ltrm\ f)$

lemma *is-UP-monomI*:
assumes $a \in carrier\ R$
assumes $p = monom\ P\ a\ n$
shows *is-UP-monom p*
⟨*proof*⟩

lemma *is-UP-monomI'*:
assumes $f \in carrier\ (UP\ R)$
assumes $f = ltrm\ f$
shows *is-UP-monom f*
⟨*proof*⟩

lemma *monom-is-UP-monom*:
assumes $a \in carrier\ R$
shows $is\text{-}UP\text{-}monom\ (monom\ P\ a\ n)\ is\text{-}UP\text{-}monom\ (monom\ (UP\ R)\ a\ n)$
⟨*proof*⟩

lemma *is-UP-monomE*:
assumes *is-UP-monom f*
shows $f \in carrier\ P\ f = monom\ P\ (lcf\ f)\ (degree\ f)\ \ f = monom\ (UP\ R)\ (lcf\ f)\ (degree\ f)$
⟨*proof*⟩

lemma *ltrm-is-UP-monom*:
assumes $p \in carrier\ P$
shows $is\text{-}UP\text{-}monom\ (ltrm\ p)$
⟨*proof*⟩

lemma *is-UP-monom-mult*:
assumes *is-UP-monom p*
assumes *is-UP-monom q*
shows $is\text{-}UP\text{-}monom\ (p \otimes_P q)$
⟨*proof*⟩
end

3.4 Properties of Leading Terms and Leading Coefficients in Commutative Rings and Domains

context *UP-cring*
begin

lemma *cring-deg-mult*:
  assumes $q \in$ *carrier* P
  assumes $p \in$ *carrier* P
  assumes *lcf* $q \otimes$ *lcf* $p \neq \mathbf{0}$
  shows *degree* $(q \otimes_P p) =$ *degree* $p +$ *degree* q
⟨*proof*⟩

leading term is multiplicative

lemma *ltrm-of-sum-diff-deg*:
  assumes $q \in$ *carrier* P
  assumes $a \in$ *carrier* R
  assumes $a \neq \mathbf{0}$
  assumes *degree* $q < n$
  assumes $p = q \oplus_P$ (*monom* P a n)
  shows *ltrm* $p =$ (*monom* P a n)
⟨*proof*⟩

lemma(**in** *UP-cring*) *ltrm-smult-cring*:
  assumes $p \in$ *carrier* P
  assumes $a \in$ *carrier* R
  assumes *lcf* $p \otimes a \neq \mathbf{0}$
  shows *ltrm* $(a \odot_P p) = a \odot_P$(*ltrm* p)
  ⟨*proof*⟩

lemma(**in** *UP-cring*) *deg-zero-ltrm-smult-cring*:
  assumes $p \in$ *carrier* P
  assumes $a \in$ *carrier* R
  assumes *degree* $p = 0$
  shows *ltrm* $(a \odot_P p) = a \odot_P$(*ltrm* p)
  ⟨*proof*⟩

lemma(**in** *UP-domain*) *ltrm-smult*:
  assumes $p \in$ *carrier* P
  assumes $a \in$ *carrier* R
  shows *ltrm* $(a \odot_P p) = a \odot_P$(*ltrm* p)
  ⟨*proof*⟩

lemma(**in** *UP-cring*) *cring-ltrm-mult*:
  assumes $p \in$ *carrier* P
  assumes $q \in$ *carrier* P
  assumes *lcf* $p \otimes$ *lcf* $q \neq \mathbf{0}$
  shows *ltrm* $(p \otimes_P q) =$ (*ltrm* p) $\otimes_P$ (*ltrm* q)
⟨*proof*⟩

lemma(**in** *UP-domain*) *ltrm-mult*:
assumes $p \in$ *carrier* P
assumes $q \in$ *carrier* P
shows *ltrm* $(p \otimes_P q) = (ltrm\ p) \otimes_P (ltrm\ q)$
⟨*proof*⟩

lemma *lcf-deg-0*:
assumes *degree* $p = 0$
assumes $p \in$ *carrier* P
assumes $q \in$ *carrier* P
shows $(p \otimes_P q) = (lcf\ p) \odot_P q$
⟨*proof*⟩

leading term powers

lemma (**in** *domain*) *nonzero-pow-nonzero*:
assumes $a \in$ *carrier* R
assumes $a \neq \mathbf{0}$
shows $a[\hat{}](n::nat) \neq \mathbf{0}$
⟨*proof*⟩

lemma (**in** *UP-cring*) *cring-monom-degree*:
assumes $a \in (carrier\ R)$
assumes $p = monom\ P\ a\ m$
assumes $a[\hat{}]n \neq \mathbf{0}$
shows *degree* $(p[\hat{}]_P\ n) = n*m$
⟨*proof*⟩

lemma (**in** *UP-domain*) *monom-degree*:
assumes $a \neq \mathbf{0}$
assumes $a \in (carrier\ R)$
assumes $p = monom\ P\ a\ m$
shows *degree* $(p[\hat{}]_P\ n) = n*m$
⟨*proof*⟩

lemma(**in** *UP-cring*) *cring-pow-ltrm*:
assumes $p \in$ *carrier* P
assumes $lcf\ p\ [\hat{}]n \neq \mathbf{0}$
shows $ltrm\ (p[\hat{}]_P(n::nat)) = (ltrm\ p)[\hat{}]_P n$
⟨*proof*⟩

lemma(**in** *UP-cring*) *cring-pow-deg*:
assumes $p \in$ *carrier* P
assumes $lcf\ p\ [\hat{}]n \neq \mathbf{0}$
shows $degree\ (p[\hat{}]_P(n::nat)) = n*degree\ p$
⟨*proof*⟩

lemma(**in** *UP-cring*) *cring-pow-deg-bound*:
assumes $p \in$ *carrier* P
shows $degree\ (p[\hat{}]_P(n::nat)) \leq n*degree\ p$

⟨*proof*⟩

lemma(**in** *UP-cring*) *deg-smult*:
assumes $a \in carrier\ R$
assumes $f \in carrier\ (UP\ R)$
assumes $a \otimes lcf\ f \neq \mathbf{0}$
shows $deg\ R\ (a \odot_{UP\ R} f) = deg\ R\ \ f$
⟨*proof*⟩

lemma(**in** *UP-cring*) *deg-smult'*:
assumes $a \in Units\ R$
assumes $f \in carrier\ (UP\ R)$
shows $deg\ R\ (a \odot_{UP\ R} f) = deg\ R\ \ f$
⟨*proof*⟩

lemma(**in** *UP-domain*) *pow-sum0*:
$\bigwedge p\ q.\ p \in carrier\ P \Longrightarrow q \in carrier\ P \Longrightarrow degree\ q < degree\ p \Longrightarrow degree\ ((p \oplus_P q\ )[\hat{}]_P n) = (degree\ p)*n$
⟨*proof*⟩

lemma(**in** *UP-domain*) *pow-sum*:
assumes $p \in carrier\ P$
assumes $q \in carrier\ P$
assumes $degree\ q < degree\ p$
shows $degree\ ((p \oplus_P q\ )[\hat{}]_P n) = (degree\ p)*n$
⟨*proof*⟩

lemma(**in** *UP-domain*) *deg-pow0*:
$\bigwedge p.\ p \in carrier\ P \Longrightarrow n \geq degree\ p \Longrightarrow degree\ (p\ [\hat{}]_P\ m) = m*(degree\ p)$
⟨*proof*⟩

lemma(**in** *UP-domain*) *deg-pow*:
assumes $p \in carrier\ P$
shows $degree\ (p\ [\hat{}]_P\ m) = m*(degree\ p)$
⟨*proof*⟩

lemma(**in** *UP-domain*) *ltrm-pow0*:
$\bigwedge f.\ f \in carrier\ P \Longrightarrow ltrm\ (f\ [\hat{}]_P\ (n::nat)) = (ltrm\ f)\ [\hat{}]_P\ n$
⟨*proof*⟩

lemma(**in** *UP-domain*) *ltrm-pow*:
assumes $f \in carrier\ P$
shows $ltrm\ (f\ [\hat{}]_P\ (n::nat)) = (ltrm\ f)\ [\hat{}]_P\ n$
⟨*proof*⟩

lemma on the leading coefficient

lemma *lcf-eq*:
assumes $f \in carrier\ P$
shows $lcf\ f = lcf\ (ltrm\ f)$

⟨*proof*⟩

lemma *lcf-eq-deg-eq-imp-ltrm-eq*:
assumes $p \in$ *carrier* P
assumes $q \in$ *carrier* P
assumes *degree* $p > 0$
assumes *degree* $p =$ *degree* q
assumes *lcf* $p =$ *lcf* q
shows *ltrm* $p =$ *ltrm* q
⟨*proof*⟩

lemma *ltrm-eq-imp-lcf-eq*:
assumes $p \in$ *carrier* P
assumes $q \in$ *carrier* P
assumes *ltrm* $p =$ *ltrm* q
shows *lcf* $p =$ *lcf* q
⟨*proof*⟩

lemma *ltrm-eq-imp-deg-drop*:
assumes $p \in$ *carrier* P
assumes $q \in$ *carrier* P
assumes *ltrm* $p =$ *ltrm* q
assumes *degree* $p > 0$
shows *degree* $(p \ominus_P q) <$ *degree* p
⟨*proof*⟩

lemma(**in** *UP-cring*) *cring-lcf-scalar-mult*:
assumes $p \in$ *carrier* P
assumes $a \in$ *carrier* R
assumes $a \otimes$ (*lcf* p) $\neq \mathbf{0}$
shows *lcf* $(a \odot_P p) = a \otimes$ (*lcf* p)
⟨*proof*⟩

lemma(**in** *UP-domain*) *lcf-scalar-mult*:
assumes $p \in$ *carrier* P
assumes $a \in$ *carrier* R
shows *lcf* $(a \odot_P p) = a \otimes$ (*lcf* p)
⟨*proof*⟩

lemma(**in** *UP-cring*) *cring-lcf-mult*:
assumes $p \in$ *carrier* P
assumes $q \in$ *carrier* P
assumes (*lcf* p) $\otimes$ (*lcf* q) $\neq \mathbf{0}$
shows *lcf* $(p \otimes_P q) =$ (*lcf* p) $\otimes$ (*lcf* q)
⟨*proof*⟩

lemma(**in** *UP-domain*) *lcf-mult*:
assumes $p \in$ *carrier* P
assumes $q \in$ *carrier* P

shows *lcf* $(p \otimes_P q) = (lcf\ p) \otimes (lcf\ q)$
⟨*proof*⟩

lemma(**in** *UP-cring*) *cring-lcf-pow*:
assumes $p \in carrier\ P$
assumes $(lcf\ p)[\hat{}]n \neq \mathbf{0}$
shows *lcf* $(p[\hat{}]_P(n::nat)) = (lcf\ p)[\hat{}]n$
⟨*proof*⟩

lemma(**in** *UP-domain*) *lcf-pow*:
assumes $p \in carrier\ P$
shows *lcf* $(p[\hat{}]_P(n::nat)) = (lcf\ p)[\hat{}]n$
⟨*proof*⟩
end

3.5 Constant Terms and Constant Coefficients

Constant term and coefficient function

definition *zcf* **where**
$zcf\ f = (f\ 0)$

abbreviation(**in** *UP-cring*)(*input*) *ctrm* **where**
$ctrm\ f \equiv monom\ P\ (f\ 0)\ 0$

context *UP-cring*
begin

lemma *ctrm-is-poly*:
assumes $p \in carrier\ P$
shows $ctrm\ p \in carrier\ P$
⟨*proof*⟩

lemma *ctrm-degree*:
assumes $p \in carrier\ P$
shows $degree\ (ctrm\ p) = 0$
⟨*proof*⟩

lemma *ctrm-zcf*:
assumes $f \in carrier\ P$
assumes $zcf\ f = \mathbf{0}$
shows $ctrm\ f = \mathbf{0}_P$
⟨*proof*⟩

lemma *zcf-degree-zero*:
assumes $f \in carrier\ P$
assumes $degree\ f = 0$
shows $lcf\ f = zcf\ f$
⟨*proof*⟩

lemma *zcf-zero-degree-zero*:
assumes $f \in$ *carrier* P
assumes *degree* $f = 0$
assumes *zcf* $f = \mathbf{0}$
shows $f = \mathbf{0}_P$
⟨*proof*⟩

lemma *zcf-ctrm*:
assumes $p \in$ *carrier* P
shows *zcf* (*ctrm* p) = *zcf* p
⟨*proof*⟩

lemma *ctrm-trunc*:
assumes $p \in$ *carrier* P
assumes *degree* $p > 0$
shows *zcf*(*trunc* p) = *zcf* p
⟨*proof*⟩

Constant coefficient function is a ring homomorphism

lemma *zcf-add*:
assumes $p \in$ *carrier* P
assumes $q \in$ *carrier* P
shows *zcf*($p \oplus_P q$) = (*zcf* p) $\oplus$ (*zcf* q)
⟨*proof*⟩

lemma *coeff-ltrm*[*simp*]:
assumes $p \in$ *carrier* P
assumes *degree* $p > 0$
shows *zcf*(*ltrm* p) = $\mathbf{0}$
⟨*proof*⟩

lemma *zcf-zero*[*simp*]:
zcf $\mathbf{0}_P = \mathbf{0}$
⟨*proof*⟩

lemma *zcf-one*[*simp*]:
zcf $\mathbf{1}_P = \mathbf{1}$
⟨*proof*⟩

lemma *ctrm-smult*:
assumes $f \in$ *carrier* P
assumes $a \in$ *carrier* R
shows *ctrm* ($a \odot_P f$) = $a \odot_P$(*ctrm* f)
⟨*proof*⟩

lemma *ctrm-monom*[*simp*]:
assumes $a \in$ *carrier* R
shows *ctrm* (*monom* P a (*Suc* k)) = $\mathbf{0}_P$
⟨*proof*⟩

end

3.6 Polynomial Induction Rules

context *UP-ring*
begin

Rule for strong induction on polynomial degree

lemma *poly-induct*:
assumes $p \in carrier\ P$
assumes *Deg-0*: $\bigwedge p.\ p \in carrier\ P \Longrightarrow degree\ p = 0 \Longrightarrow Q\ p$
assumes *IH*: $\bigwedge p.\ (\bigwedge q.\ q \in carrier\ P \Longrightarrow degree\ q < degree\ p \Longrightarrow Q\ q) \Longrightarrow p \in carrier\ P \Longrightarrow degree\ p > 0 \Longrightarrow Q\ p$
shows $Q\ p$
⟨*proof*⟩

Variant on induction on degree

lemma *poly-induct2*:
assumes $p \in carrier\ P$
assumes *Deg-0*: $\bigwedge p.\ p \in carrier\ P \Longrightarrow degree\ p = 0 \Longrightarrow Q\ p$
assumes *IH*: $\bigwedge p.\ degree\ p > 0 \Longrightarrow p \in carrier\ P \Longrightarrow Q\ (trunc\ p) \Longrightarrow Q\ p$
shows $Q\ p$
⟨*proof*⟩

Additive properties which are true for all monomials are true for all polynomials

lemma *poly-induct3*:
assumes $p \in carrier\ P$
assumes *add*: $\bigwedge p\ q.\ q \in carrier\ P \Longrightarrow p \in carrier\ P \Longrightarrow Q\ p \Longrightarrow Q\ q \Longrightarrow Q\ (p \oplus_P q)$
assumes *monom*: $\bigwedge a\ n.\ a \in carrier\ R \Longrightarrow Q\ (monom\ P\ a\ n)$
shows $Q\ p$
⟨*proof*⟩

lemma *poly-induct4*:
assumes $p \in carrier\ P$
assumes *add*: $\bigwedge p\ q.\ q \in carrier\ P \Longrightarrow p \in carrier\ P \Longrightarrow Q\ p \Longrightarrow Q\ q \Longrightarrow Q\ (p \oplus_P q)$
assumes *monom-zero*: $\bigwedge a.\ a \in carrier\ R \Longrightarrow Q\ (monom\ P\ a\ 0)$
assumes *monom-Suc*: $\bigwedge a\ n.\ a \in carrier\ R \Longrightarrow Q\ (monom\ P\ a\ (Suc\ n))$
shows $Q\ p$
⟨*proof*⟩

lemma *monic-monom-smult*:
assumes $a \in carrier\ R$
shows $a \odot_P monom\ P\ \mathbf{1}\ n = monom\ P\ a\ n$
⟨*proof*⟩

lemma *poly-induct5*:

assumes $p \in carrier\ P$
assumes *add*: $\bigwedge p\ q.\ q \in carrier\ P \Longrightarrow p \in carrier\ P \Longrightarrow Q\ p \Longrightarrow Q\ q\ \Longrightarrow Q\ (p \oplus_P q)$
assumes *monic-monom*: $\bigwedge n.\ Q\ (monom\ P\ \mathbf{1}\ n)$
assumes *smult*: $\bigwedge p\ a\ .\ a \in carrier\ R \Longrightarrow p \in carrier\ P \Longrightarrow Q\ p \Longrightarrow Q\ (a \odot_P p)$
shows $Q\ p$
⟨*proof*⟩

lemma *poly-induct6*:
assumes $p \in carrier\ P$
assumes *monom*: $\bigwedge a\ n.\ a \in carrier\ R \Longrightarrow Q\ (monom\ P\ a\ 0)$
assumes *plus-monom*: $\bigwedge a\ n\ p.\ a \in carrier\ R \Longrightarrow a \neq \mathbf{0} \Longrightarrow p \in carrier\ P \Longrightarrow degree\ p < n \Longrightarrow Q\ p \Longrightarrow Q(p \oplus_P monom\ P\ a\ n)$
shows $Q\ p$
⟨*proof*⟩

end

4 Mapping a Polynomial to its Associated Ring Function

Turning a polynomial into a function on R:

definition *to-function* **where**
$to\text{-}function\ S\ f\ = (\lambda s \in carrier\ S.\ eval\ S\ S\ (\lambda x.\ x)\ s\ f)$

context *UP-cring*
begin

definition *to-fun* **where**
$to\text{-}fun\ f \equiv to\text{-}function\ R\ f$

Explicit formula for evaluating a polynomial function:

lemma *to-fun-eval*:
assumes $f \in carrier\ P$
assumes $x \in carrier\ R$
shows $to\text{-}fun\ f\ x\ = eval\ R\ R\ (\lambda x.\ x)\ x\ f$
⟨*proof*⟩

lemma *to-fun-formula*:
assumes $f \in carrier\ P$
assumes $x \in carrier\ R$
shows $to\text{-}fun\ f\ x = (\bigoplus i \in \{..degree\ f\}.\ (f\ i) \otimes x\ [\hat{}]\ i)$
⟨*proof*⟩

lemma *eval-ring-hom*:
assumes $a \in$ *carrier* R
shows *eval* R R $(\lambda x.\ x)$ $a \in$ *ring-hom* P R
⟨*proof*⟩

lemma *to-fun-closed*:
assumes $f \in$ *carrier* P
assumes $x \in$ *carrier* R
shows *to-fun* f x $\in$ *carrier* R
⟨*proof*⟩

lemma *to-fun-plus*:
assumes $g \in$ *carrier* P
assumes $f \in$ *carrier* P
assumes $x \in$ *carrier* R
shows *to-fun* $(f \oplus_P g)$ $x = (\textit{to-fun}\ f\ x) \oplus (\textit{to-fun}\ g\ x)$
⟨*proof*⟩

lemma *to-fun-mult*:
assumes $g \in$ *carrier* P
assumes $f \in$ *carrier* P
assumes $x \in$ *carrier* R
shows *to-fun* $(f \otimes_P g)$ $x = (\textit{to-fun}\ f\ x) \otimes (\textit{to-fun}\ g\ x)$
⟨*proof*⟩

lemma *to-fun-ring-hom*:
assumes $a \in$ *carrier* R
shows $(\lambda p.\ \textit{to-fun}\ p\ a) \in$ *ring-hom* P R
⟨*proof*⟩

lemma *ring-hom-uminus*:
assumes *ring* S
assumes $f \in$ (*ring-hom* S R)
assumes $a \in$ *carrier* S
shows $f\ (\ominus_S\ a) = \ominus\ (f\ a)$
⟨*proof*⟩

lemma *to-fun-minus*:
assumes $f \in$ *carrier* P
assumes $x \in$ *carrier* R
shows *to-fun* $(\ominus_P f)$ $x = \ominus\ (\textit{to-fun}\ f\ x)$
⟨*proof*⟩

lemma *id-is-hom*:
ring-hom-cring R R $(\lambda x.\ x)$
⟨*proof*⟩

lemma *UP-pre-univ-prop-fact*:
UP-pre-univ-prop R R $(\lambda x.\ x)$

⟨*proof*⟩

end

4.1 to-fun is a Ring Homomorphism from Polynomials to Functions

context *UP-cring*
begin

lemma *to-fun-is-Fun*:
assumes $x \in$ *carrier* P
shows *to-fun* $x \in$ *carrier* (*Fun* R)
⟨*proof*⟩

lemma *to-fun-Fun-mult*:
assumes $x \in$ *carrier* P
assumes $y \in$ *carrier* P
shows *to-fun* $(x \otimes_P y) =$ *to-fun* $x \otimes_{\textit{function-ring}\ (\textit{carrier}\ R)\ R}$ *to-fun* y
⟨*proof*⟩

lemma *to-fun-Fun-add*:
assumes $x \in$ *carrier* P
assumes $y \in$ *carrier* P
shows *to-fun* $(x \oplus_P y) =$ *to-fun* $x \oplus_{\textit{function-ring}\ (\textit{carrier}\ R)\ R}$ *to-fun* y
⟨*proof*⟩

lemma *to-fun-Fun-one*:
to-fun $\mathbf{1}_P = \mathbf{1}_{\textit{Fun}\ R}$
⟨*proof*⟩

lemma *to-fun-Fun-zero*:
to-fun $\mathbf{0}_P = \mathbf{0}_{\textit{Fun}\ R}$
⟨*proof*⟩

lemma *to-fun-function-ring-hom*:
to-fun $\in$ *ring-hom* P (*Fun* R)
⟨*proof*⟩

lemma(**in** *UP-cring*) *to-fun-one*:
assumes $a \in$ *carrier* R
shows *to-fun* $\mathbf{1}_P\ a = \mathbf{1}$
⟨*proof*⟩

lemma(**in** *UP-cring*) *to-fun-zero*:
assumes $a \in$ *carrier* R
shows *to-fun* $\mathbf{0}_P\ a = \mathbf{0}$
⟨*proof*⟩

lemma(**in** *UP-cring*) *to-fun-nat-pow*:
assumes $h \in$ *carrier* (*UP R*)
assumes $a \in$ *carrier R*
shows *to-fun* ($h[\hat{}]_{UP\ R}(n::nat)$) a = (*to-fun h a*)[^]n
⟨*proof*⟩

lemma(**in** *UP-cring*) *to-fun-finsum*:
assumes *finite* ($Y::'d$ *set*)
assumes $f \in UNIV \to$ *carrier* (*UP R*)
assumes $t \in$ *carrier R*
shows *to-fun* (*finsum* (*UP R*) *f Y*) *t* = *finsum R* (λ*i*. (*to-fun* (*f i*) *t*)) *Y*
⟨*proof*⟩

end

4.2 Inclusion of a Ring into its Polynomials Ring via Constants

definition *to-polynomial* **where**
to-polynomial R = (λ*a*. *monom* (*UP R*) *a* 0)

context *UP-cring*
begin

abbreviation(*input*) *to-poly* **where**
to-poly ≡ *to-polynomial R*

lemma *to-poly-mult-simp*:
assumes $b \in$ *carrier R*
assumes $f \in$ *carrier* (*UP R*)
shows (*to-polynomial R b*) $\otimes_{UP\ R}$ f = $b \odot_{UP\ R}$ f
$f \otimes_{UP\ R}$ (*to-polynomial R b*) = $b \odot_{UP\ R}$ f
⟨*proof*⟩

lemma *to-fun-to-poly*:
assumes $a \in$ *carrier R*
assumes $b \in$ *carrier R*
shows *to-fun* (*to-poly a*) *b* = *a*
⟨*proof*⟩

lemma *to-poly-inverse*:
assumes $f \in$ *carrier P*
assumes *degree f* = 0
shows *f* = *to-poly* (*f* 0)
⟨*proof*⟩

lemma *to-poly-closed*:
assumes $a \in$ *carrier R*
shows *to-poly a* ∈ *carrier P*

⟨*proof*⟩

lemma *degree-to-poly*[*simp*]:
assumes $a \in$ *carrier* R
shows *degree* (*to-poly* a) = *0*
⟨*proof*⟩

lemma *to-poly-is-ring-hom*:
to-poly ∈ *ring-hom* R P
⟨*proof*⟩

lemma *to-poly-add*:
assumes $a \in$ *carrier* R
assumes $b \in$ *carrier* R
shows *to-poly* $(a \oplus b)$ = *to-poly* $a \oplus_P$ *to-poly* b
⟨*proof*⟩

lemma *to-poly-mult*:
assumes $a \in$ *carrier* R
assumes $b \in$ *carrier* R
shows *to-poly* $(a \otimes b)$ = *to-poly* $a \otimes_P$ *to-poly* b
⟨*proof*⟩

lemma *to-poly-minus*:
assumes $a \in$ *carrier* R
assumes $b \in$ *carrier* R
shows *to-poly* $(a \ominus b)$ = *to-poly* $a \ominus_P$ *to-poly* b
⟨*proof*⟩

lemma *to-poly-a-inv*:
assumes $a \in$ *carrier* R
shows *to-poly* $(\ominus a) = \ominus_P$ *to-poly* a
⟨*proof*⟩

lemma *to-poly-nat-pow*:
assumes $a \in$ *carrier* R
shows (*to-poly* a) $[\hat{}]_P$ (n::*nat*)= *to-poly* $(a[\hat{}]n)$
⟨*proof*⟩

end

5 Polynomial Substitution

definition *compose* **where**
compose R f g = *eval* R (*UP* R) (*to-polynomial* R) g f

abbreviation(**in** *UP-cring*)(*input*) *sub* (**infixl** *of 70*) **where**
sub f $g \equiv$ *compose* R f g

definition *rev-compose* **where**
rev-compose R = *eval R* (*UP R*) (*to-polynomial R*)

abbreviation(**in** *UP-cring*)(*input*) *rev-sub* **where**
rev-sub ≡ *rev-compose R*

context *UP-cring*
begin

lemma *sub-rev-sub*:
sub f g = *rev-sub g f*
⟨*proof*⟩

lemma(**in** *UP-cring*) *to-poly-UP-pre-univ-prop*:
UP-pre-univ-prop R P to-poly
⟨*proof*⟩

lemma *rev-sub-is-hom*:
assumes *g* ∈ *carrier P*
shows *rev-sub g* ∈ *ring-hom P P*
⟨*proof*⟩

lemma *rev-sub-closed*:
assumes *p* ∈ *carrier P*
assumes *q* ∈ *carrier P*
shows *rev-sub q p* ∈ *carrier P*
⟨*proof*⟩

lemma *sub-closed*:
assumes *p* ∈ *carrier P*
assumes *q* ∈ *carrier P*
shows *sub q p* ∈ *carrier P*
⟨*proof*⟩

lemma *rev-sub-add*:
assumes *g* ∈ *carrier P*
assumes *f* ∈ *carrier P*
assumes *h* ∈*carrier P*
shows *rev-sub g* (*f* ⊕$_P$ *h*) = (*rev-sub g f*) ⊕$_P$ (*rev-sub g h*)
⟨*proof*⟩

lemma *sub-add*:
assumes *g* ∈ *carrier P*
assumes *f* ∈ *carrier P*
assumes *h* ∈*carrier P*
shows ((*f* ⊕$_P$ *h*) *of g*) = ((*f of g*) ⊕$_P$ (*h of g*))
⟨*proof*⟩

lemma *rev-sub-mult*:
assumes $g \in carrier\ P$
assumes $f \in carrier\ P$
assumes $h \in carrier\ P$
shows $rev\text{-}sub\ g\ (f \otimes_P h) = (rev\text{-}sub\ g\ f) \otimes_P (rev\text{-}sub\ g\ h)$
⟨*proof*⟩

lemma *sub-mult*:
assumes $g \in carrier\ P$
assumes $f \in carrier\ P$
assumes $h \in carrier\ P$
shows $((f \otimes_P h)\ of\ g) = ((f\ of\ g) \otimes_P (h\ of\ g))$
⟨*proof*⟩

lemma *sub-monom*:
assumes $g \in carrier\ (UP\ R)$
assumes $a \in carrier\ R$
shows $sub\ (monom\ (UP\ R)\ a\ n)\ g = to\text{-}poly\ a\ \otimes_{UP\ R} (g[\hat{}]_{UP\ R}\ (n::nat))$
$sub\ (monom\ (UP\ R)\ a\ n)\ g = a \odot_{UP\ R} (g[\hat{}]_{UP\ R}\ (n::nat))$
⟨*proof*⟩

Subbing into a constant does nothing

lemma *rev-sub-to-poly*:
assumes $g \in carrier\ P$
assumes $a \in carrier\ R$
shows $rev\text{-}sub\ g\ (to\text{-}poly\ a) = to\text{-}poly\ a$
⟨*proof*⟩

lemma *sub-to-poly*:
assumes $g \in carrier\ P$
assumes $a \in carrier\ R$
shows $(to\text{-}poly\ a)\ of\ g\ = to\text{-}poly\ a$
⟨*proof*⟩

lemma *sub-const*:
assumes $g \in carrier\ P$
assumes $f \in carrier\ P$
assumes $degree\ f = 0$
shows $f\ of\ g = f$
⟨*proof*⟩

Substitution into a monomial

lemma *monom-sub*:
assumes $a \in carrier\ R$
assumes $g \in carrier\ P$
shows $(monom\ P\ a\ n)\ of\ g = a \odot_P g[\hat{}]_P\ n$
⟨*proof*⟩

lemma(**in** *UP-cring*) *cring-sub-monom-bound*:

assumes $a \in$ *carrier* R
assumes $a \neq \mathbf{0}$
assumes $f =$ *monom* $P\ a\ n$
assumes $g \in$ *carrier* P
shows *degree* $(f$ *of* $g) \leq n*($*degree* $g)$
⟨*proof*⟩

lemma(**in** *UP-cring*) *cring-sub-monom*:
assumes $a \in$ *carrier* R
assumes $a \neq \mathbf{0}$
assumes $f =$ *monom* $P\ a\ n$
assumes $g \in$ *carrier* P
assumes $a \otimes (lcf\ g\ [\widehat{}]\ n) \neq \mathbf{0}$
shows *degree* $(f$ *of* $g) = n*($*degree* $g)$
⟨*proof*⟩

lemma(**in** *UP-domain*) *sub-monom*:
assumes $a \in$ *carrier* R
assumes $a \neq \mathbf{0}$
assumes $f =$ *monom* $P\ a\ n$
assumes $g \in$ *carrier* P
shows *degree* $(f$ *of* $g) = n*($*degree* $g)$
⟨*proof*⟩

Subbing a constant into a polynomial yields a constant

lemma *sub-in-const*:
assumes $g \in$ *carrier* P
assumes $f \in$ *carrier* P
assumes *degree* $g = 0$
shows *degree* $(f$ *of* $g) = 0$
⟨*proof*⟩

lemma (**in** *UP-cring*) *cring-sub-deg-bound*:
assumes $g \in$ *carrier* P
assumes $f \in$ *carrier* P
shows *degree* $(f$ *of* $g) \leq$ *degree* $f *$ *degree* g
⟨*proof*⟩

lemma (**in** *UP-cring*) *cring-sub-deg*:
assumes $g \in$ *carrier* P
assumes $f \in$ *carrier* P
assumes $lcf\ f \otimes (lcf\ g\ [\widehat{}]\ (\text{degree } f)) \neq \mathbf{0}$
shows *degree* $(f$ *of* $g) =$ *degree* $f *$ *degree* g
⟨*proof*⟩

lemma (**in** *UP-domain*) *sub-deg0*:
assumes $g \in$ *carrier* P
assumes $f \in$ *carrier* P
assumes $g \neq \mathbf{0}_P$

assumes $f \neq \mathbf{0}_P$
shows *degree* (*f of g*) = *degree f* ∗ *degree g*
⟨*proof*⟩

lemma(**in** *UP-domain*) *sub-deg*:
assumes $g \in$ *carrier P*
assumes $f \in$ *carrier P*
assumes $g \neq \mathbf{0}_P$
shows *degree* (*f of g*) = *degree f* ∗ *degree g*
⟨*proof*⟩

lemma(**in** *UP-cring*) *cring-ltrm-sub*:
assumes $g \in$ *carrier P*
assumes $f \in$ *carrier P*
assumes *degree g* > *0*
assumes *lcf f* ⊗ (*lcf g* [^] (*degree f*)) ≠ **0**
shows *ltrm* (*f of g*) = *ltrm* ((*ltrm f*) *of g*)
⟨*proof*⟩

lemma(**in** *UP-domain*) *ltrm-sub*:
assumes $g \in$ *carrier P*
assumes $f \in$ *carrier P*
assumes *degree g* > *0*
shows *ltrm* (*f of g*) = *ltrm* ((*ltrm f*) *of g*)
⟨*proof*⟩

lemma(**in** *UP-cring*) *cring-lcf-of-sub-in-ltrm*:
assumes $g \in$ *carrier P*
assumes $f \in$ *carrier P*
assumes *degree f* = *n*
assumes *degree g* > *0*
assumes (*lcf f*) ⊗ ((*lcf g*)[^]*n*) ≠**0**
shows *lcf* ((*ltrm f*) *of g*) = (*lcf f*) ⊗ ((*lcf g*)[^]*n*)
⟨*proof*⟩

lemma(**in** *UP-domain*) *lcf-of-sub-in-ltrm*:
assumes $g \in$ *carrier P*
assumes $f \in$ *carrier P*
assumes *degree f* = *n*
assumes *degree g* > *0*
shows *lcf* ((*ltrm f*) *of g*) = (*lcf f*) ⊗ ((*lcf g*)[^]*n*)
⟨*proof*⟩

lemma(**in** *UP-cring*) *cring-ltrm-of-sub-in-ltrm*:
assumes $g \in$ *carrier P*
assumes $f \in$ *carrier P*
assumes *degree f* = *n*
assumes *degree g* > *0*
assumes (*lcf f*) ⊗ ((*lcf g*)[^]*n*) ≠**0**

shows *ltrm* ((*ltrm f*) *of g*) = (*lcf f*) $\odot_P$ ((*ltrm g*)[^]$_P$*n*)
⟨*proof*⟩

lemma(**in** *UP-domain*) *ltrm-of-sub-in-ltrm*:
assumes $g \in$ *carrier P*
assumes $f \in$ *carrier P*
assumes *degree f* = *n*
assumes *degree g* > *0*
shows *ltrm* ((*ltrm f*) *of g*) = (*lcf f*) $\odot_P$ ((*ltrm g*)[^]$_P$*n*)
⟨*proof*⟩

formula for the leading term of a composition

lemma(**in** *UP-domain*) *cring-ltrm-of-sub*:
assumes $g \in$ *carrier P*
assumes $f \in$ *carrier P*
assumes *degree f* = *n*
assumes *degree g* > *0*
assumes (*lcf f*) $\otimes$ ((*lcf g*)[^]*n*) $\neq$**0**
shows *ltrm* (*f of g*) = (*lcf f*) $\odot_P$ ((*ltrm g*)[^]$_P$*n*)
⟨*proof*⟩

lemma(**in** *UP-domain*) *ltrm-of-sub*:
assumes $g \in$ *carrier P*
assumes $f \in$ *carrier P*
assumes *degree f* = *n*
assumes *degree g* > *0*
shows *ltrm* (*f of g*) = (*lcf f*) $\odot_P$ ((*ltrm g*)[^]$_P$*n*)
⟨*proof*⟩

subtitution is associative

lemma *sub-assoc-monom*:
assumes $f \in$ *carrier P*
assumes $q \in$ *carrier P*
assumes $r \in$ *carrier P*
shows (*ltrm f*) *of* (*q of r*) = ((*ltrm f*) *of q*) *of r*
⟨*proof*⟩

lemma *sub-assoc*:
assumes $f \in$ *carrier P*
assumes $q \in$ *carrier P*
assumes $r \in$ *carrier P*
shows *f of* (*q of r*) = (*f of q*) *of r*
⟨*proof*⟩

lemma *sub-smult*:
assumes $f \in$ *carrier P*
assumes $q \in$ *carrier P*
assumes $a \in$ *carrier R*
shows (*a*$\odot_P$*f*) *of q* = *a*$\odot_P$(*f of q*)

⟨*proof*⟩

lemma *to-fun-sub-monom*:
assumes *is-UP-monom f*
assumes $g \in$ *carrier P*
assumes $a \in$ *carrier R*
shows *to-fun* (*f of g*) $a =$ *to-fun f* (*to-fun g a*)
⟨*proof*⟩

lemma *to-fun-sub*:
assumes $g \in$ *carrier P*
assumes $f \in$ *carrier P*
assumes $a \in$ *carrier R*
shows *to-fun* (*f of g*) $a =$ (*to-fun f*) (*to-fun g a*)
⟨*proof*⟩
end

More material on constant terms and constant coefficients

context *UP-cring*
begin

lemma *to-fun-ctrm*:
assumes $f \in$ *carrier P*
assumes $b \in$ *carrier R*
shows *to-fun* (*ctrm f*) $b =$ (*f 0*)
⟨*proof*⟩

lemma *to-fun-smult*:
assumes $f \in$ *carrier P*
assumes $b \in$ *carrier R*
assumes $c \in$ *carrier R*
shows *to-fun* $(c \odot_P f)$ $b = c \otimes$(*to-fun f b*)
⟨*proof*⟩

lemma *to-fun-monom*:
assumes $c \in$ *carrier R*
assumes $x \in$ *carrier R*
shows *to-fun* (*monom P c n*) $x = c \otimes x\ [\hat{}]\ n$
⟨*proof*⟩

lemma *zcf-monom*:
assumes $a \in$ *carrier R*
shows *zcf* (*monom P a n*) = *to-fun* (*monom P a n*) **0**
⟨*proof*⟩

lemma *zcf-to-fun*:
assumes $p \in$ *carrier P*
shows *zcf p* = *to-fun p* **0**
⟨*proof*⟩

lemma *zcf-to-poly*[*simp*]:
assumes $a \in carrier\ R$
shows $zcf\ (to\text{-}poly\ a) = a$
⟨*proof*⟩

lemma *zcf-ltrm-mult*:
assumes $p \in carrier\ P$
assumes $q \in carrier\ P$
assumes $degree\ p > 0$
shows $zcf((ltrm\ p) \otimes_P q) = \mathbf{0}$
⟨*proof*⟩

lemma *zcf-mult*:
assumes $p \in carrier\ P$
assumes $q \in carrier\ P$
shows $zcf(p \otimes_P q) = (zcf\ p) \otimes (zcf\ q)$
⟨*proof*⟩

lemma *zcf-is-ring-hom*:
$zcf \in ring\text{-}hom\ P\ R$
⟨*proof*⟩

lemma *ctrm-is-ring-hom*:
$ctrm \in ring\text{-}hom\ P\ P$
⟨*proof*⟩

6 Describing the Image of (UP R) in the Ring of Functions from R to R

lemma *to-fun-diff*:
assumes $p \in carrier\ P$
assumes $q \in carrier\ P$
assumes $a \in carrier\ R$
shows $to\text{-}fun\ (p \ominus_P q)\ a = to\text{-}fun\ p\ a \ominus to\text{-}fun\ q\ a$
⟨*proof*⟩

lemma *to-fun-const*:
assumes $a \in carrier\ R$
assumes $b \in carrier\ R$
shows $to\text{-}fun\ (monom\ P\ a\ 0)\ b = a$
⟨*proof*⟩

lemma *to-fun-monic-monom*:
assumes $b \in carrier\ R$
shows $to\text{-}fun\ (monom\ P\ \mathbf{1}\ n)\ b = b[\hat{}]n$
⟨*proof*⟩

Constant polynomials map to constant polynomials

lemma *const-to-constant*:
assumes $a \in$ *carrier* R
shows *to-fun* (*monom* P a *0*) = *constant-function* (*carrier* R) a
⟨*proof*⟩

Monomial polynomials map to monomial functions

lemma *monom-to-monomial*:
assumes $a \in$ *carrier* R
shows *to-fun* (*monom* P a n) = *monomial-function* R a n
⟨*proof*⟩
end

7 Taylor Expansions

7.1 Monic Linear Polynomials

The polynomial representing the variable X

definition *X-poly* **where**
X-poly R = *monom* (*UP* R) $\mathbf{1}_R$ *1*

context *UP-cring*
begin

abbreviation(*input*) X **where**
$X \equiv$ *X-poly* R

lemma *X-closed*:
$X \in$ *carrier* P
⟨*proof*⟩

lemma *degree-X*[*simp*]:
assumes 1 $\neq$**0**
shows*degree* X = *1*
⟨*proof*⟩

lemma *X-not-zero*:
assumes 1 $\neq$**0**
shows$X \neq \mathbf{0}_P$
⟨*proof*⟩

lemma *sub-X*[*simp*]:
assumes $p \in$ *carrier* P
shows X *of* p = p
⟨*proof*⟩

lemma *sub-monom-deg-one*:
assumes $p \in$ *carrier* P
assumes $a \in$ *carrier* R

shows *monom P a 1 of p* = *a* $\odot_P$ *p*
⟨*proof*⟩

lemma *monom-rep-X-pow*:
assumes *a* ∈ *carrier R*
shows *monom P a n* = *a*$\odot_P$(*X*[^]$_P$*n*)
⟨*proof*⟩

lemma *X-sub*[*simp*]:
assumes *p* ∈ *carrier P*
shows *p of X* = *p*
⟨*proof*⟩

representation of monomials as scalar multiples of powers of X

lemma *ltrm-rep-X-pow*:
assumes *p* ∈ *carrier P*
shows *ltrm p* = (*lcf p*)$\odot_P$(*X*[^]$_P$(*degree p*))
⟨*proof*⟩

lemma *to-fun-monom′*:
assumes *c* ∈ *carrier R*
assumes *c* ≠**0**
assumes *x* ∈ *carrier R*
shows *to-fun* (*c* $\odot_P$ *X*[^]$_P$(*n*::*nat*)) *x* = *c* ⊗ *x* [^] *n*
⟨*proof*⟩

lemma *to-fun-X-pow*:
assumes *x* ∈ *carrier R*
shows *to-fun* (*X*[^]$_P$(*n*::*nat*)) *x* = *x* [^] *n*
⟨*proof*⟩
end

Monic linear polynomials

definition *X-poly-plus* **where**
X-poly-plus R a = (*X-poly R*) $\oplus_{(UP\ R)}$ *to-polynomial R a*

definition *X-poly-minus* **where**
X-poly-minus R a = (*X-poly R*) $\ominus_{(UP\ R)}$ *to-polynomial R a*

context *UP-cring*
begin

abbreviation(*input*) *X-plus* **where**
X-plus ≡ *X-poly-plus R*

abbreviation(*input*) *X-minus* **where**
X-minus ≡ *X-poly-minus R*

lemma *X-plus-closed*:

assumes $a \in$ *carrier* R
shows (*X-plus* a) $\in$ *carrier* P
⟨*proof*⟩

lemma *X-minus-closed*:
assumes $a \in$ *carrier* R
shows (*X-minus* a) $\in$ *carrier* P
⟨*proof*⟩

lemma *X-minus-plus*:
assumes $a \in$ *carrier* R
shows (*X-minus* a) = *X-plus* ($\ominus a$)
⟨*proof*⟩

lemma *degree-of-X-plus*:
assumes $a \in$ *carrier* R
assumes 1 $\neq$**0**
shows *degree* (*X-plus* a) = *1*
⟨*proof*⟩

lemma *degree-of-X-minus*:
assumes $a \in$ *carrier* R
assumes 1 $\neq$**0**
shows *degree* (*X-minus* a) = *1*
⟨*proof*⟩

lemma *ltrm-of-X*:
shows*ltrm* $X = X$
⟨*proof*⟩

lemma *ltrm-of-X-plus*:
assumes $a \in$ *carrier* R
assumes 1 $\neq$**0**
shows *ltrm* (*X-plus* a) = X
⟨*proof*⟩

lemma *ltrm-of-X-minus*:
assumes $a \in$ *carrier* R
assumes 1 $\neq$**0**
shows *ltrm* (*X-minus* a) = X
⟨*proof*⟩

lemma *lcf-of-X-minus*:
assumes $a \in$ *carrier* R
assumes 1 $\neq$**0**
shows *lcf* (*X-minus* a) = **1**
⟨*proof*⟩

lemma *lcf-of-X-plus*:

assumes $a \in$ *carrier* R
assumes $\mathbf{1} \neq \mathbf{0}$
shows *lcf* (*X-plus* a) $= \mathbf{1}$
⟨*proof*⟩

lemma *to-fun-X*[*simp*]:
assumes $a \in$ *carrier* R
shows *to-fun* X $a = a$
⟨*proof*⟩

lemma *to-fun-X-plus*[*simp*]:
assumes $a \in$ *carrier* R
assumes $b \in$ *carrier* R
shows *to-fun* (*X-plus* a) $b = b \oplus a$
⟨*proof*⟩

lemma *to-fun-X-minus*[*simp*]:
assumes $a \in$ *carrier* R
assumes $b \in$ *carrier* R
shows *to-fun* (*X-minus* a) $b = b \ominus a$
⟨*proof*⟩

lemma *cfs-X-plus*:
assumes $a \in$ *carrier* R
shows *X-plus* a n = (*if* $n = 0$ *then* a *else* (*if* $n = 1$ *then* $\mathbf{1}$ *else* $\mathbf{0}$))
⟨*proof*⟩

lemma *cfs-X-minus*:
assumes $a \in$ *carrier* R
shows *X-minus* a n = (*if* $n = 0$ *then* $\ominus$ a *else* (*if* $n = 1$ *then* $\mathbf{1}$ *else* $\mathbf{0}$))
⟨*proof*⟩

Linear substituions

lemma *X-plus-sub-deg*:
assumes $a \in$ *carrier* R
assumes $f \in$ *carrier* P
shows *degree* (*f of* (*X-plus* a)) = *degree* f
⟨*proof*⟩

lemma *X-minus-sub-deg*:
assumes $a \in$ *carrier* R
assumes $f \in$ *carrier* P
shows *degree* (*f of* (*X-minus* a)) = *degree* f
⟨*proof*⟩

lemma *plus-minus-sub*:
assumes $a \in$ *carrier* R
shows *X-plus a of X-minus a* $= X$
⟨*proof*⟩

lemma *minus-plus-sub*:
assumes $a \in$ *carrier* R
shows *X-minus a of X-plus a* $= X$
⟨*proof*⟩

lemma *ltrm-times-X*:
assumes $p \in$ *carrier* P
shows *ltrm* $(X \otimes_P p) = X \otimes_P$ (*ltrm* p)
⟨*proof*⟩

lemma *times-X-not-zero*:
assumes $p \in$ *carrier* P
assumes $p \neq \mathbf{0}_P$
shows $(X \otimes_P p) \neq \mathbf{0}_P$
⟨*proof*⟩

lemma *degree-times-X*:
assumes $p \in$ *carrier* P
assumes $p \neq \mathbf{0}_P$
shows *degree* $(X \otimes_P p) =$ *degree* $p + 1$
⟨*proof*⟩

end

7.2 Basic Facts About Taylor Expansions

definition *taylor-expansion* **where**
taylor-expansion R a p $=$ *compose R p* (*X-poly-plus R a*)

definition(**in** *UP-cring*) *taylor* **where**
taylor $\equiv$ *taylor-expansion R*

context *UP-cring*
begin

lemma *taylor-expansion-ring-hom*:
assumes $c \in$ *carrier* R
shows *taylor-expansion R c* $\in$ *ring-hom P P*
⟨*proof*⟩

notation *taylor* $(T\_)$

lemma(**in** *UP-cring*) *taylor-closed*:
assumes $f \in$ *carrier* P
assumes $a \in$ *carrier* R
shows $T_a\ f \in$ *carrier* P
⟨*proof*⟩

lemma *taylor-deg*:
assumes $a \in carrier\ R$
assumes $p \in carrier\ P$
shows $degree\ (T_a\ p) = degree\ p$
⟨*proof*⟩

lemma *taylor-id*:
assumes $a \in carrier\ R$
assumes $p \in carrier\ P$
shows $p = (T_a\ p)\ of\ (X\text{-}minus\ a)$
⟨*proof*⟩

lemma *taylor-eval*:
assumes $a \in carrier\ R$
assumes $f \in carrier\ P$
assumes $b \in carrier\ R$
shows $to\text{-}fun\ (T_a\ f)\ b = to\text{-}fun\ f\ (b \oplus a)$
⟨*proof*⟩

lemma *taylor-eval'*:
assumes $a \in carrier\ R$
assumes $f \in carrier\ P$
assumes $b \in carrier\ R$
shows $to\text{-}fun\ f\ (b) = to\text{-}fun\ (T_a\ f)\ (b \ominus a)$
⟨*proof*⟩

lemma(**in** *UP-cring*) *degree-monom*:
assumes $a \in carrier\ R$
shows $degree\ (a \odot_{UP\ R}\ (X\text{-}poly\ R)[\hat{}]_{UP\ R} n) = (if\ a = \mathbf{0}\ then\ 0\ else\ n)$
⟨*proof*⟩

lemma(**in** *UP-cring*) *poly-comp-finsum*:
assumes $\bigwedge i{::}nat.\ i \leq n \Longrightarrow g\ i \in carrier\ P$
assumes $q \in carrier\ P$
assumes $p = (\bigoplus_P\ i \in \{..n\}.\ g\ i)$
shows $p\ of\ q = (\bigoplus_P\ i \in \{..n\}.\ (g\ i)\ of\ q)$
⟨*proof*⟩

lemma(**in** *UP-cring*) *poly-comp-expansion*:
assumes $p \in carrier\ P$
assumes $q \in carrier\ P$
assumes $degree\ p \leq n$
shows $p\ of\ q = (\bigoplus_P\ i \in \{..n\}.\ (p\ i) \odot_P\ q[\hat{}]_P i)$
⟨*proof*⟩

lemma(**in** *UP-cring*) *taylor-sum*:
assumes $p \in carrier\ P$
assumes $degree\ p \leq n$
assumes $a \in carrier\ R$

shows $p = (\bigoplus_P i \in \{..n\}.\ T_a\ p\ i \odot_P (X\text{-}minus\ a)[\hat{}]_P i)$
⟨*proof*⟩

The i^{th} term in the taylor expansion

definition *taylor-term* **where**
taylor-term c p i = (*taylor-expansion R c p i*) $\odot_{UP\ R}$ (*UP-cring.X-minus R c*) $[\hat{}]_{UP\ R} i$

lemma (**in** *UP-cring*) *taylor-term-closed*:
assumes $p \in carrier\ P$
assumes $a \in carrier\ R$
shows *taylor-term a p i* ∈ *carrier* (*UP R*)
⟨*proof*⟩

lemma(**in** *UP-cring*) *taylor-term-sum*:
assumes $p \in carrier\ P$
assumes $degree\ p \leq n$
assumes $a \in carrier\ R$
shows $p = (\bigoplus_P i \in \{..n\}.\ taylor\text{-}term\ a\ p\ i)$
⟨*proof*⟩

lemma (**in** *UP-cring*) *taylor-expansion-add*:
assumes $p \in carrier\ P$
assumes $q \in carrier\ P$
assumes $c \in carrier\ R$
shows *taylor-expansion R c* (*p* $\oplus_{UP\ R}$ *q*) = (*taylor-expansion R c p*) $\oplus_{UP\ R}$ (*taylor-expansion R c q*)
⟨*proof*⟩

lemma (**in** *UP-cring*) *taylor-term-add*:
assumes $p \in carrier\ P$
assumes $q \in carrier\ P$
assumes $a \in carrier\ R$
shows *taylor-term a* (*p* $\oplus_{UP\ R}$*q*) *i* = *taylor-term a p i* $\oplus_{UP\ R}$ *taylor-term a q i*
⟨*proof*⟩

lemma (**in** *UP-cring*) *to-fun-taylor-term*:
assumes $p \in carrier\ P$
assumes $a \in carrier\ R$
assumes $c \in carrier\ R$
shows *to-fun* (*taylor-term c p i*) *a* = (T_c *p i*) ⊗ (*a* ⊖ *c*)[^]*i*
⟨*proof*⟩

end

7.3 Defining the (Scalar-Valued) Derivative of a Polynomial Using the Taylor Expansion

definition *derivative* **where**
derivative R f a = (*taylor-expansion R a f*) *1*

context *UP-cring*
begin

abbreviation(**in** *UP-cring*) *deriv* **where**
deriv ≡ *derivative R*

lemma(**in** *UP-cring*) *deriv-closed*:
assumes *f* ∈ *carrier P*
assumes *a* ∈ *carrier R*
shows (*derivative R f a*) ∈ *carrier R*
⟨*proof*⟩

lemma(**in** *UP-cring*) *deriv-add*:
assumes *f* ∈ *carrier P*
assumes *g* ∈ *carrier P*
assumes *a* ∈ *carrier R*
shows *deriv* (*f* ⊕$_P$ *g*) *a* = *deriv f a* ⊕ *deriv g a*
⟨*proof*⟩

end

8 The Polynomial-Valued Derivative Operator

context *UP-cring*
begin

8.1 Operator Which Shifts Coefficients

lemma *cfs-times-X*:
assumes *g* ∈ *carrier P*
shows (*X* ⊗$_P$ *g*) (*Suc n*) = *g n*
⟨*proof*⟩

lemma *times-X-pow-coeff*:
assumes *g* ∈ *carrier P*
shows (*monom P* **1** *k* ⊗$_P$ *g*) (*n* + *k*) = *g n*
⟨*proof*⟩

lemma *zcf-eq-zero-unique*:
assumes *f* ∈ *carrier P*
assumes *g* ∈ *carrier P* ∧ (*f* = *X* ⊗$_P$ *g*)
shows ⋀ *h*. *h* ∈ *carrier P* ∧ (*f* = *X* ⊗$_P$ *h*) ⟹ *h* = *g*
⟨*proof*⟩

lemma *f-minus-ctrm*:
assumes $f \in carrier\ P$
shows $zcf(f \ominus_P ctrm\ f) = \mathbf{0}$
⟨*proof*⟩

definition *poly-shift* **where**
$poly\text{-}shift\ f\ n = f\ (Suc\ n)$

lemma *poly-shift-closed*:
assumes $f \in carrier\ P$
shows $poly\text{-}shift\ f \in carrier\ P$
⟨*proof*⟩

lemma *poly-shift-eq-0*:
assumes $f \in carrier\ P$
shows $f\ \ n = (ctrm\ f \oplus_P X \otimes_P poly\text{-}shift\ f)\ n$
⟨*proof*⟩

lemma *poly-shift-eq*:
assumes $f \in carrier\ P$
shows $f = (ctrm\ f \oplus_P X \otimes_P poly\text{-}shift\ f)$
⟨*proof*⟩

lemma *poly-shift-id*:
assumes $f \in carrier\ P$
shows $f \ominus_P ctrm\ f = X \otimes_P poly\text{-}shift\ f$
⟨*proof*⟩

lemma *poly-shift-degree-zero*:
assumes $p \in carrier\ P$
assumes $degree\ p = 0$
shows $poly\text{-}shift\ p = \mathbf{0}_P$
⟨*proof*⟩

lemma *poly-shift-degree*:
assumes $p \in carrier\ P$
assumes $degree\ p > 0$
shows $degree\ (poly\text{-}shift\ p) = degree\ p - 1$
⟨*proof*⟩

lemma *poly-shift-monom*:
assumes $a \in carrier\ R$
shows $poly\text{-}shift\ (monom\ P\ a\ (Suc\ k)) = (monom\ P\ a\ k)$
⟨*proof*⟩

lemma(**in** *UP-cring*) *poly-shift-add*:
assumes $f \in carrier\ P$
assumes $g \in carrier\ P$

shows *poly-shift* $(f \oplus_P g) = (\textit{poly-shift}\ f) \oplus_P (\textit{poly-shift}\ g)$
⟨*proof*⟩

lemma(**in** *UP-cring*) *poly-shift-s-mult*:
assumes $f \in \textit{carrier}\ P$
assumes $s \in \textit{carrier}\ R$
shows *poly-shift* $(s \odot_P f) = s \odot_P (\textit{poly-shift}\ f)$
⟨*proof*⟩

lemma *zcf-poly-shift*:
assumes $f \in \textit{carrier}\ P$
shows $\textit{zcf}\ (\textit{poly-shift}\ f) = f\ 1$
⟨*proof*⟩

fun *poly-shift-iter* (*shift*) **where**
Base:*poly-shift-iter 0 f* = *f*|
Step:*poly-shift-iter* (*Suc n*) *f* = *poly-shift* (*poly-shift-iter n f*)

lemma *shift-closed*:
assumes $f \in \textit{carrier}\ P$
shows $\textit{shift}\ n\ f \in \textit{carrier}\ P$
⟨*proof*⟩

8.2 Operator Which Multiplies Coefficients by Their Degree

definition *n-mult* **where**
n-mult f = (λ*n*. $[n]\cdot_R(f\ n)$)

lemma(**in** *UP-cring*) *n-mult-closed*:
assumes $f \in \textit{carrier}\ P$
shows $\textit{n-mult}\ f \in \textit{carrier}\ P$
⟨*proof*⟩

Facts about the shift function

lemma *shift-one*:
shift (*Suc 0*) = *poly-shift*
⟨*proof*⟩

lemma *shift-factor0*:
assumes $f \in \textit{carrier}\ P$
shows $\textit{degree}\ f \geq (\textit{Suc}\ k) \Longrightarrow \textit{degree}\ (f \ominus_P ((\textit{shift}\ (\textit{Suc}\ k)\ f) \otimes_P (X[\hat{}]_P(\textit{Suc}\ k)))) < (\textit{Suc}\ k)$
⟨*proof*⟩

lemma(**in** *UP-cring*) *shift-degree0*:
assumes $f \in \textit{carrier}\ P$
shows $\textit{degree}\ f > n \Longrightarrow \textit{Suc}\ (\textit{degree}\ (\textit{shift}\ (\textit{Suc}\ n)\ f)) = \textit{degree}\ (\textit{shift}\ n\ f)$
⟨*proof*⟩

lemma(**in** *UP-cring*) *shift-degree*:
assumes $f \in$ *carrier* P
shows *degree* $f \geq n \Longrightarrow$ *degree* (*shift* n f) + n = *degree* f
⟨*proof*⟩

lemma(**in** *UP-cring*) *shift-degree'*:
assumes $f \in$ *carrier* P
shows *degree* (*shift* (*degree* f) f) = *0*
⟨*proof*⟩

lemma(**in** *UP-cring*) *shift-above-degree*:
assumes $f \in$ *carrier* P
assumes $k >$ *degree* f
shows (*shift* k f) = $\mathbf{0}_P$
⟨*proof*⟩

lemma(**in** *UP-domain*) *shift-cfs0*:
assumes $f \in$ *carrier* P
shows *zcf*(*shift 1 f*) = *f 1*
⟨*proof*⟩

lemma(**in** *UP-cring*) *X-mult-cf*:
assumes $p \in$ *carrier* P
shows ($p \otimes_P X$) (*k+1*) = p k
⟨*proof*⟩

lemma(**in** *UP-cring*) *X-pow-cf*:
assumes $p \in$ *carrier* P
shows ($p \otimes_P X[\hat{}]_P$(*n*::*nat*)) (n + k) = p k
⟨*proof*⟩

lemma *poly-shift-cfs*:
assumes $f \in$ *carrier* P
shows *poly-shift* f n = f (*Suc* n)
⟨*proof*⟩

lemma(**in** *UP-cring*) *shift-cfs*:
assumes $p \in$ *carrier* P
shows (*shift* k p) n = p (k + n)
⟨*proof*⟩

8.3 The Derivative Operator

definition *pderiv* **where**
pderiv p = *poly-shift* (*n-mult* p)

lemma *pderiv-closed*:
assumes $p \in$ *carrier* P
shows *pderiv* $p \in$ *carrier* P

⟨*proof*⟩

Function which obtains the first n+1 terms of f, in ascending order of degree:

definition *trms-of-deg-leq* **where**
trms-of-deg-leq n f ≡ *f* ⊖$_{(UP\ R)}$ ((*shift* (*Suc n*) *f*) ⊗$_{UP\ R}$ *monom P* **1** (*Suc n*))

lemma *trms-of-deg-leq-closed*:
assumes *f* ∈ *carrier P*
shows *trms-of-deg-leq n f* ∈ *carrier P*
⟨*proof*⟩

lemma *trms-of-deg-leq-id*:
assumes *f* ∈ *carrier P*
shows *f* ⊖$_P$ (*trms-of-deg-leq k f*) = *shift* (*Suc k*) *f* ⊗$_P$ *monom P* **1** (*Suc k*)
⟨*proof*⟩

lemma *trms-of-deg-leq-id'*:
assumes *f* ∈ *carrier P*
shows *f* = (*trms-of-deg-leq k f*) ⊕$_P$ *shift* (*Suc k*) *f* ⊗$_P$ *monom P* **1** (*Suc k*)
⟨*proof*⟩

lemma *deg-leqI*:
assumes *p* ∈ *carrier P*
assumes ⋀*n*. *n* > *k* ⟹ *p n* = **0**
shows *degree p* ≤ *k*
⟨*proof*⟩

lemma *deg-leE*:
assumes *p* ∈ *carrier P*
assumes *degree p* < *k*
shows *p k* = **0**
⟨*proof*⟩

lemma *trms-of-deg-leq-deg*:
assumes *f* ∈ *carrier P*
shows *degree* (*trms-of-deg-leq k f*) ≤ *k*
⟨*proof*⟩

lemma *trms-of-deg-leq-zero-is-ctrm*:
assumes *f* ∈ *carrier P*
assumes *degree f* > *0*
shows *trms-of-deg-leq 0 f* = *ctrm f*
⟨*proof*⟩

lemma *cfs-monom-mult*:
assumes *p* ∈ *carrier P*
assumes *a* ∈ *carrier R*
assumes *k* < *n*
shows (*p* ⊗$_P$ (*monom P a n*)) *k* = **0**

⟨*proof*⟩

lemma(**in** *UP-cring*) *cfs-monom-mult-2*:
assumes $f \in$ *carrier* P
assumes $a \in$ *carrier* R
assumes $m < n$
shows $((monom\ P\ a\ n) \otimes_P f)\ m = \mathbf{0}$
⟨*proof*⟩

lemma *trms-of-deg-leq-cfs*:
assumes $f \in$ *carrier* P
shows *trms-of-deg-leq* $n\ f\ k = (if\ k \leq n\ then\ (f\ k)\ else\ \mathbf{0})$
⟨*proof*⟩

lemma *trms-of-deg-leq-iter*:
assumes $f \in$ *carrier* P
shows *trms-of-deg-leq* $(Suc\ k)\ f = (trms\text{-}of\text{-}deg\text{-}leq\ k\ f) \oplus_P monom\ P\ (f\ (Suc\ k))\ (Suc\ k)$
⟨*proof*⟩

lemma *trms-of-deg-leq-0*:
assumes $f \in$ *carrier* P
shows *trms-of-deg-leq* $0\ f = ctrm\ f$
⟨*proof*⟩

lemma *trms-of-deg-leq-degree-f*:
assumes $f \in$ *carrier* P
shows *trms-of-deg-leq* $(degree\ f)\ f = f$
⟨*proof*⟩

definition(**in** *UP-cring*) *lin-part* **where**
lin-part $f =$ *trms-of-deg-leq* $1\ f$

lemma(**in** *UP-cring*) *lin-part-id*:
assumes $f \in$ *carrier* P
shows *lin-part* $f = (ctrm\ f) \oplus_P monom\ P\ (f\ 1)\ 1$
⟨*proof*⟩

lemma(**in** *UP-cring*) *lin-part-eq*:
assumes $f \in$ *carrier* P
shows $f =$ *lin-part* $f \oplus_P (shift\ 2\ f) \otimes_P monom\ P\ \mathbf{1}\ 2$
⟨*proof*⟩

Constant term of a substitution:

lemma *zcf-eval*:
assumes $f \in$ *carrier* P
shows $zcf\ f =$ *to-fun* $f\ \mathbf{0}$
⟨*proof*⟩

lemma *ctrm-of-sub*:
assumes $f \in carrier\ P$
assumes $g \in carrier\ P$
shows $zcf(f\ of\ g) = to\text{-}fun\ f\ (zcf\ g)$
⟨*proof*⟩

Evaluation of linear part:

lemma *to-fun-lin-part*:
assumes $f \in carrier\ P$
assumes $b \in carrier\ R$
shows $to\text{-}fun\ (lin\text{-}part\ f)\ b = (f\ 0) \oplus (f\ 1) \otimes b$
⟨*proof*⟩

Constant term of taylor expansion:

lemma *taylor-zcf*:
assumes $f \in carrier\ P$
assumes $a \in carrier\ R$
shows $zcf(T_a\ f) = to\text{-}fun\ f\ a$
⟨*proof*⟩

lemma(**in** *UP-cring*) *taylor-eq-1*:
assumes $f \in carrier\ P$
assumes $a \in carrier\ R$
shows $(T_a\ f) \ominus_P (trms\text{-}of\text{-}deg\text{-}leq\ 1\ (T_a\ f)) = (shift\ (2{::}nat)\ (T_a\ f)) \otimes_P (X[\widehat{\ }]_P(2{::}nat))$
⟨*proof*⟩

lemma(**in** *UP-cring*) *taylor-deg-1*:
assumes $f \in carrier\ P$
assumes $a \in carrier\ R$
shows $f\ of\ (X\text{-}plus\ a) = (lin\text{-}part\ (T_a\ f)) \oplus_P (shift\ (2{::}nat)\ (T_a\ f)) \otimes_P (X[\widehat{\ }]_P(2{::}nat))$
⟨*proof*⟩

lemma(**in** *UP-cring*) *taylor-deg-1-eval*:
assumes $f \in carrier\ P$
assumes $a \in carrier\ R$
assumes $b \in carrier\ R$
assumes $c = to\text{-}fun\ (shift\ (2{::}nat)\ (T_a\ f))\ b$
assumes $fa = to\text{-}fun\ f\ a$
assumes $f'a = deriv\ f\ a$
shows $to\text{-}fun\ f\ (b \oplus a) = fa \oplus (f'a \otimes b) \oplus (c \otimes b[\widehat{\ }](2{::}nat))$
⟨*proof*⟩

lemma(**in** *UP-cring*) *taylor-deg-1-eval'*:
assumes $f \in carrier\ P$
assumes $a \in carrier\ R$
assumes $b \in carrier\ R$
assumes $c = to\text{-}fun\ (shift\ (2{::}nat)\ (T_a\ f))\ b$

assumes $fa = to\text{-}fun\ f\ a$
assumes $f'a = deriv\ f\ a$
shows $to\text{-}fun\ f\ (a \oplus b) = fa \oplus (f'a \otimes b) \oplus (c \otimes b[\hat{}](2::nat))$
⟨*proof*⟩

lemma(**in** *UP-cring*) *taylor-deg-1-eval''*:
assumes $f \in carrier\ P$
assumes $a \in carrier\ R$
assumes $b \in carrier\ R$
assumes $c = to\text{-}fun\ (shift\ (2::nat)\ (T_a\ f))\ (\ominus b)$
shows $to\text{-}fun\ f\ (a \ominus b) = (to\text{-}fun\ f\ a) \ominus (deriv\ f\ a \otimes b) \oplus (c \otimes b[\hat{}](2::nat))$
⟨*proof*⟩

lemma(**in** *UP-cring*) *taylor-deg-1-expansion*:
assumes $f \in carrier\ P$
assumes $a \in carrier\ R$
assumes $b \in carrier\ R$
assumes $c = to\text{-}fun\ (shift\ (2::nat)\ (T_a\ f))\ (b \ominus a)$
assumes $fa = to\text{-}fun\ f\ a$
assumes $f'a = deriv\ f\ a$
shows $to\text{-}fun\ f\ (b) = fa \oplus f'a \otimes (b \ominus a) \oplus (c \otimes (b \ominus a)[\hat{}](2::nat))$
⟨*proof*⟩

lemma(**in** *UP-cring*) *Taylor-deg-1-expansion'*:
assumes $f \in carrier\ (UP\ R)$
assumes $a \in carrier\ R$
assumes $b \in carrier\ R$
shows $\exists c \in carrier\ R.\ to\text{-}fun\ f\ (b) = (to\text{-}fun\ f\ a) \oplus (deriv\ f\ a) \otimes (b \ominus a) \oplus (c \otimes (b \ominus a)[\hat{}](2::nat))$
⟨*proof*⟩

Basic Properties of deriv and pderiv:

lemma *n-mult-degree-bound*:
assumes $f \in carrier\ P$
shows $degree\ (n\text{-}mult\ f) \leq degree\ f$
⟨*proof*⟩

lemma *pderiv-deg-0*[*simp*]:
assumes $f \in carrier\ P$
assumes $degree\ f = 0$
shows $pderiv\ f = \mathbf{0}_P$
⟨*proof*⟩

lemma *deriv-deg-0*:
assumes $f \in carrier\ P$
assumes $degree\ f = 0$
assumes $a \in carrier\ R$
shows $deriv\ f\ a = \mathbf{0}$
⟨*proof*⟩

lemma *poly-shift-monom′*:
assumes $a \in carrier\ R$
shows $poly\text{-}shift\ (a \odot_P (X[\hat{}]_P(Suc\ n))) = a \odot_P (X[\hat{}]_P n)$
⟨*proof*⟩

lemma *monom-coeff*:
assumes $a \in carrier\ R$
shows $(a \odot_P X\ [\hat{}]_P\ (n::nat))\ k = (if\ (k = n)\ then\ a\ else\ \mathbf{0})$
⟨*proof*⟩

lemma *cfs-n-mult*:
assumes $p \in carrier\ P$
shows $n\text{-}mult\ p\ n = [n]\cdot(p\ n)$
⟨*proof*⟩

lemma *cfs-add-nat-pow*:
assumes $p \in carrier\ P$
shows $([(n::nat)]\cdot_P p)\ k = [n]\cdot(p\ k)$
⟨*proof*⟩

lemma *cfs-add-int-pow*:
assumes $p \in carrier\ P$
shows $([(n::int)]\cdot_P p)\ k = [n]\cdot(p\ k)$
⟨*proof*⟩

lemma *add-nat-pow-monom*:
assumes $a \in carrier\ R$
shows $[(n::nat)]\cdot_P monom\ P\ a\ k = monom\ P\ ([n]\cdot a)\ k$
⟨*proof*⟩

lemma *add-int-pow-monom*:
assumes $a \in carrier\ R$
shows $[(n::int)]\cdot_P monom\ P\ a\ k = monom\ P\ ([n]\cdot a)\ k$
⟨*proof*⟩

lemma *n-mult-monom*:
assumes $a \in carrier\ R$
shows $n\text{-}mult\ (monom\ P\ a\ (Suc\ n)) = monom\ P\ ([Suc\ n]\cdot a)\ (Suc\ n)$
⟨*proof*⟩

lemma *pderiv-monom*:
assumes $a \in carrier\ R$
shows $pderiv\ (monom\ P\ a\ n) = monom\ P\ ([n]\cdot a)\ (n-1)$
⟨*proof*⟩

lemma *pderiv-monom′*:
assumes $a \in carrier\ R$
shows $pderiv\ (a \odot_P X[\hat{}]_P(n::nat)) = ([n]\cdot a) \odot_P X[\hat{}]_P(n-1)$

⟨*proof*⟩

lemma *n-mult-add*:
assumes $p \in carrier\ P$
assumes $q \in carrier\ P$
shows $n\text{-}mult\ (p \oplus_P q) = n\text{-}mult\ p \oplus_P n\text{-}mult\ q$
⟨*proof*⟩

lemma *pderiv-add*:
assumes $p \in carrier\ P$
assumes $q \in carrier\ P$
shows $pderiv\ (p \oplus_P q) = pderiv\ p \oplus_P pderiv\ q$
⟨*proof*⟩

lemma *zcf-monom-sub*:
assumes $p \in carrier\ P$
shows $zcf\ ((monom\ P\ \mathbf{1}\ (Suc\ n))\ of\ \ p) = zcf\ p\ [\widehat{\ }]\ (Suc\ n)$
⟨*proof*⟩

lemma *zcf-monom-sub'*:
assumes $p \in carrier\ P$
assumes $a \in carrier\ R$
shows $zcf\ ((monom\ P\ a\ (Suc\ n))\ of\ \ p) = a \otimes zcf\ p\ [\widehat{\ }]\ (Suc\ n)$
⟨*proof*⟩

lemma *deriv-monom*:
assumes $a \in carrier\ R$
assumes $b \in carrier\ R$
shows $deriv\ (monom\ P\ a\ n)\ b = \ ([n]\cdot a)\otimes(b[\widehat{\ }](n-1))$
⟨*proof*⟩

lemma *deriv-smult*:
assumes $a \in carrier\ R$
assumes $b \in carrier\ R$
assumes $g \in carrier\ P$
shows $deriv\ (a \odot_P g)\ b = a \otimes (deriv\ g\ b)$
⟨*proof*⟩

lemma *deriv-const*:
assumes $a \in carrier\ R$
assumes $b \in carrier\ R$
shows $deriv\ (monom\ P\ a\ 0)\ b = \mathbf{0}$
⟨*proof*⟩

lemma *deriv-monom-deg-one*:
assumes $a \in carrier\ R$
assumes $b \in carrier\ R$
shows $deriv\ (monom\ P\ a\ 1)\ b = a$
⟨*proof*⟩

lemma *monom-Suc*:
assumes $a \in carrier\ R$
shows $monom\ P\ a\ (Suc\ n) = monom\ P\ \mathbf{1}\ 1 \otimes_P monom\ P\ a\ n$
$monom\ P\ a\ (Suc\ n) = monom\ P\ a\ n \otimes_P monom\ P\ \mathbf{1}\ 1$
⟨*proof*⟩

8.4 The Product Rule

lemma(**in** *UP-cring*) *times-x-product-rule*:
assumes $f \in carrier\ P$
shows $pderiv\ (f \otimes_P up\text{-}ring.monom\ P\ \mathbf{1}\ 1) = f \oplus_P pderiv\ f \otimes_P up\text{-}ring.monom\ P\ \mathbf{1}\ 1$
⟨*proof*⟩

lemma(**in** *UP-cring*) *deg-one-eval*:
assumes $g \in carrier\ (UP\ R)$
assumes $deg\ R\ g = 1$
shows $\bigwedge t.\ t \in carrier\ R \Longrightarrow to\text{-}fun\ g\ t = g\ 0 \oplus (g\ 1) \otimes t$
⟨*proof*⟩

lemma *nmult-smult*:
assumes $a \in carrier\ R$
assumes $f \in carrier\ P$
shows $n\text{-}mult\ (a \odot_P f) = a \odot_P (n\text{-}mult\ f)$
⟨*proof*⟩

lemma *pderiv-smult*:
assumes $a \in carrier\ R$
assumes $f \in carrier\ P$
shows $pderiv\ (a \odot_P f) = a \odot_P (pderiv\ f)$
⟨*proof*⟩

lemma(**in** *UP-cring*) *pderiv-minus*:
assumes $a \in carrier\ P$
assumes $b \in carrier\ P$
shows $pderiv\ (a \ominus_P b) = pderiv\ a \ominus_P pderiv\ b$
⟨*proof*⟩

lemma(**in** *UP-cring*) *pderiv-const*:
assumes $b \in carrier\ R$
shows $pderiv\ (up\text{-}ring.monom\ P\ b\ 0) = \mathbf{0}_P$
⟨*proof*⟩

lemma(**in** *UP-cring*) *pderiv-minus-const*:
assumes $a \in carrier\ P$
assumes $b \in carrier\ R$
shows $pderiv\ (a \ominus_P up\text{-}ring.monom\ P\ b\ 0) = pderiv\ a$
⟨*proof*⟩

lemma(**in** *UP-cring*) *monom-product-rule*:
assumes $f \in carrier\ P$
assumes $a \in carrier\ R$
shows $pderiv\ (f \otimes_P up\text{-}ring.monom\ P\ a\ n) = f \otimes_P pderiv\ (up\text{-}ring.monom\ P\ a\ n) \oplus_P pderiv\ f \otimes_P up\text{-}ring.monom\ P\ a\ n$
⟨*proof*⟩

lemma(**in** *UP-cring*) *product-rule*:
assumes $f \in carrier\ (UP\ R)$
assumes $g \in carrier\ (UP\ R)$
shows $pderiv\ (f \otimes_{UP\ R} g) = (pderiv\ f \otimes_{UP\ R}\ g) \oplus_{UP\ R} (f \otimes_{UP\ R}\ pderiv\ g)$
⟨*proof*⟩

8.5 The Chain Rule

lemma(**in** *UP-cring*) *chain-rule*:
assumes $f \in carrier\ P$
assumes $g \in carrier\ P$
shows $pderiv\ (compose\ R\ f\ g) = compose\ R\ (pderiv\ f)\ g \otimes_{UP\ R} pderiv\ g$
⟨*proof*⟩

lemma *deriv-prod-rule-times-monom*:
assumes $a \in carrier\ R$
assumes $b \in carrier\ R$
assumes $q \in carrier\ P$
shows $deriv\ ((monom\ P\ a\ n) \otimes_P q)\ b = (deriv\ (monom\ P\ a\ n)\ b) \otimes (to\text{-}fun\ q\ b) \oplus (to\text{-}fun\ (monom\ P\ a\ n)\ b) \otimes deriv\ q\ b$
⟨*proof*⟩

lemma *deriv-prod-rule*:
assumes $p \in carrier\ P$
assumes $q \in carrier\ P$
assumes $a \in carrier\ R$
shows $deriv\ (p \otimes_P q)\ a = deriv\ p\ a \otimes (to\text{-}fun\ q\ a) \oplus (to\text{-}fun\ p\ a) \otimes deriv\ q\ a$
⟨*proof*⟩

lemma *pderiv-eval-deriv-monom*:
assumes $a \in carrier\ R$
assumes $b \in carrier\ R$
shows $to\text{-}fun\ (pderiv\ (monom\ P\ a\ n))\ b = deriv\ (monom\ P\ a\ n)\ b$
⟨*proof*⟩

lemma *pderiv-eval-deriv*:
assumes $f \in carrier\ P$
assumes $a \in carrier\ R$
shows $deriv\ f\ a = to\text{-}fun\ (pderiv\ f)\ a$
⟨*proof*⟩

Taking taylor expansions commutes with taking derivatives:

lemma(**in** *UP-cring*) *taylor-expansion-pderiv-comm*:
assumes $f \in carrier\ (UP\ R)$
assumes $c \in carrier\ R$
shows $pderiv\ (taylor\text{-}expansion\ R\ c\ f) = taylor\text{-}expansion\ R\ c\ (pderiv\ f)$
⟨*proof*⟩

8.6 Linear Substitutions

lemma(**in** *UP-ring*) *lcoeff-Lcf*:
assumes $f \in carrier\ P$
shows $lcoeff\ f = lcf\ f$
⟨*proof*⟩

lemma(**in** *UP-cring*) *linear-sub-cfs*:
assumes $f \in carrier\ (UP\ R)$
assumes $d \in carrier\ R$
assumes $g = compose\ R\ f\ (up\text{-}ring.monom\ (UP\ R)\ d\ 1)$
shows $g\ i = d[\hat{}]i \otimes f\ i$
⟨*proof*⟩

lemma(**in** *UP-cring*) *linear-sub-deriv*:
assumes $f \in carrier\ (UP\ R)$
assumes $d \in carrier\ R$
assumes $g = compose\ R\ f\ (up\text{-}ring.monom\ (UP\ R)\ d\ 1)$
assumes $c \in carrier\ R$
shows $pderiv\ g = d \odot_{UP\ R} compose\ R\ (pderiv\ f)\ (up\text{-}ring.monom\ (UP\ R)\ d\ 1)$
⟨*proof*⟩

lemma(**in** *UP-cring*) *linear-sub-deriv'*:
assumes $f \in carrier\ (UP\ R)$
assumes $d \in carrier\ R$
assumes $g = compose\ R\ f\ (up\text{-}ring.monom\ (UP\ R)\ d\ 1)$
assumes $c \in carrier\ R$
shows $pderiv\ g = compose\ R\ (d \odot_{UP\ R} pderiv\ f)\ (up\text{-}ring.monom\ (UP\ R)\ d\ 1)$
⟨*proof*⟩

lemma(**in** *UP-cring*) *linear-sub-inv*:
assumes $f \in carrier\ (UP\ R)$
assumes $d \in Units\ R$
assumes $g = compose\ R\ f\ (up\text{-}ring.monom\ (UP\ R)\ d\ 1)$
shows $compose\ R\ g\ (up\text{-}ring.monom\ (UP\ R)\ (inv\ d)\ 1) = f$
⟨*proof*⟩

lemma(**in** *UP-cring*) *linear-sub-deg*:
assumes $f \in carrier\ (UP\ R)$
assumes $d \in Units\ R$
assumes $g = compose\ R\ f\ (up\text{-}ring.monom\ (UP\ R)\ d\ 1)$
shows $deg\ R\ g = deg\ R\ f$
⟨*proof*⟩

end

9 Lemmas About Polynomial Division

context *UP-cring*
begin

9.1 Division by Linear Terms

definition *UP-root-div* **where**
UP-root-div f a = (*poly-shift* (*T*$_a$ *f*)) *of* (*X-minus a*)

definition *UP-root-rem* **where**
UP-root-rem f a = *ctrm* (*T*$_a$ *f*)

lemma *UP-root-div-closed*:
assumes *f* ∈ *carrier P*
assumes *a* ∈ *carrier R*
shows *UP-root-div f a* ∈ *carrier P*
⟨*proof*⟩

lemma *rem-closed*:
assumes *f* ∈ *carrier P*
assumes *a* ∈ *carrier R*
shows *UP-root-rem f a* ∈ *carrier P*
⟨*proof*⟩

lemma *rem-deg*:
assumes *f* ∈ *carrier P*
assumes *a* ∈ *carrier R*
shows *degree* (*UP-root-rem f a*) = *0*
⟨*proof*⟩

lemma *remainder-theorem*:
assumes *f* ∈ *carrier P*
assumes *a* ∈ *carrier R*
assumes *g* = *UP-root-div f a*
assumes *r* = *UP-root-rem f a*
shows *f* = *r* ⊕$_P$ (*X-minus a*) ⊗$_P$ *g*
⟨*proof*⟩

lemma *remainder-theorem′*:
assumes *f* ∈ *carrier P*
assumes *a* ∈ *carrier R*
shows *f* = *UP-root-rem f a* ⊕$_P$ (*X-minus a*) ⊗$_P$ *UP-root-div f a*
⟨*proof*⟩

lemma *factor-theorem*:

assumes $f \in$ *carrier* P
assumes $a \in$ *carrier* R
assumes $g =$ *UP-root-div* $f\ a$
assumes *to-fun* $f\ a = \mathbf{0}$
shows $f = (X\text{-}minus\ a) \otimes_P g$
⟨*proof*⟩

lemma *factor-theorem'*:
assumes $f \in$ *carrier* P
assumes $a \in$ *carrier* R
assumes *to-fun* $f\ a = \mathbf{0}$
shows $f = (X\text{-}minus\ a) \otimes_P$ *UP-root-div* $f\ a$
⟨*proof*⟩

9.2 Geometric Sums

lemma *geom-quot*:
assumes $a \in$ *carrier* R
assumes $b \in$ *carrier* R
assumes $p =$ *monom* $P\ \mathbf{1}\ (Suc\ n) \ominus_P$ *monom* $P\ (b[\hat{}](Suc\ n))\ 0$
assumes $g =$ *UP-root-div* $p\ b$
shows $a[\hat{}](Suc\ n) \ominus b[\hat{}]\ (Suc\ n) = (a \ominus b) \otimes (to\text{-}fun\ g\ a)$
⟨*proof*⟩

end

context *UP-cring*
begin

definition *geometric-series* **where**
geometric-series $n\ a\ b =$ *to-fun* (*UP-root-div* (*monom* $P\ \mathbf{1}\ (Suc\ n) \ominus_{UP\ R}$ (*monom* $P\ (b[\hat{}](Suc\ n))\ 0))\ b)\ a$

lemma *geometric-series-id*:
assumes $a \in$ *carrier* R
assumes $b \in$ *carrier* R
shows $a[\hat{}](Suc\ n) \ominus b[\hat{}]\ (Suc\ n) = (a \ominus b) \otimes (geometric\text{-}series\ n\ a\ b)$
⟨*proof*⟩

lemma *geometric-series-closed*:
assumes $a \in$ *carrier* R
assumes $b \in$ *carrier* R
shows $(geometric\text{-}series\ n\ a\ b) \in$ *carrier* R
⟨*proof*⟩

Shows that $a^n - b^n$ has $a - b$ as a factor:

lemma *to-fun-monic-monom-diff*:
assumes $a \in$ *carrier* R
assumes $b \in$ *carrier* R

shows $\exists\, c.\ c \in \mathit{carrier}\ R \wedge \mathit{to\text{-}fun}\ (\mathit{monom}\ P\ \mathbf{1}\ n)\ a \ominus \mathit{to\text{-}fun}\ (\mathit{monom}\ P\ \mathbf{1}\ n)\ b = (a \ominus b) \otimes c$
⟨*proof*⟩

lemma *to-fun-diff-factor*:
assumes $a \in \mathit{carrier}\ R$
assumes $b \in \mathit{carrier}\ R$
assumes $f \in \mathit{carrier}\ P$
shows $\exists\, c.\ c \in \mathit{carrier}\ R \wedge (\mathit{to\text{-}fun}\ f\ a) \ominus (\mathit{to\text{-}fun}\ f\ b) = (a \ominus b) \otimes c$
⟨*proof*⟩

Any finite set over a domain is the zero set of a polynomial:

lemma(**in** *UP-domain*) *fin-set-poly-roots*:
assumes $F \subseteq \mathit{carrier}\ R$
assumes *finite F*
shows $\exists\ P \in \mathit{carrier}\ (\mathit{UP}\ R).\ \forall\ x \in \mathit{carrier}\ R.\ \mathit{to\text{-}fun}\ P\ x = \mathbf{0} \longleftrightarrow x \in F$
⟨*proof*⟩

9.3 Polynomial Evaluation at Multiplicative Inverses

For every polynomial $p(x)$ of degree n, there is a unique polynomial $q(x)$ which satisfies the equation $q(x) = x^n p(1/x)$. This section defines this polynomial and proves this identity.

definition(**in** *UP-cring*) *one-over-poly* **where**
$\mathit{one\text{-}over\text{-}poly}\ p = (\lambda\ n.\ \mathit{if}\ n \leq \mathit{degree}\ p\ \mathit{then}\ p\ ((\mathit{degree}\ \ p) - n)\ \mathit{else}\ \mathbf{0})$

lemma(**in** *UP-cring*) *one-over-poly-closed*:
assumes $p \in \mathit{carrier}\ P$
shows $\mathit{one\text{-}over\text{-}poly}\ p \in \mathit{carrier}\ P$
⟨*proof*⟩

lemma(**in** *UP-cring*) *one-over-poly-monom*:
assumes $a \in \mathit{carrier}\ R$
shows $\mathit{one\text{-}over\text{-}poly}\ (\mathit{monom}\ P\ a\ n) = \mathit{monom}\ P\ a\ 0$
⟨*proof*⟩

lemma(**in** *UP-cring*) *one-over-poly-monom-add*:
assumes $a \in \mathit{carrier}\ R$
assumes $a \neq \mathbf{0}$
assumes $p \in \mathit{carrier}\ P$
assumes $\mathit{degree}\ p < n$
shows $\mathit{one\text{-}over\text{-}poly}\ (p \oplus_P \mathit{monom}\ P\ a\ n) = \mathit{monom}\ P\ a\ 0 \oplus_P \mathit{monom}\ P\ \mathbf{1}\ (n - \mathit{degree}\ p) \otimes_P \mathit{one\text{-}over\text{-}poly}\ p$
⟨*proof*⟩

lemma(**in** *UP-cring*) *one-over-poly-eval*:
assumes $p \in \mathit{carrier}\ P$
assumes $x \in \mathit{carrier}\ R$

assumes $x \in$ *Units* R
shows *to-fun* (*one-over-poly* p) $x = (x[\hat{}](\textit{degree}\ p)) \otimes (\textit{to-fun}\ p\ (\textit{inv}_R\ x))$
⟨*proof*⟩

end

10 Lifting Homomorphisms of Rings to Polynomial Rings by Application to Coefficients

definition *poly-lift-hom* **where**
poly-lift-hom $R\ S\ \varphi =$ *eval* R (*UP* S) (*to-polynomial* $S \circ \varphi$) (*X-poly* S)

context *UP-ring*
begin

lemma(**in** *UP-cring*) *pre-poly-lift-hom-is-hom*:
assumes *cring* S
assumes $\varphi \in$ *ring-hom* $R\ S$
shows *ring-hom-ring* R (*UP* S) (*to-polynomial* $S \circ \varphi$)
⟨*proof*⟩

lemma(**in** *UP-cring*) *poly-lift-hom-is-hom*:
assumes *cring* S
assumes $\varphi \in$ *ring-hom* $R\ S$
shows *poly-lift-hom* $R\ S\ \varphi \in$ *ring-hom* (*UP* R) (*UP* S)
⟨*proof*⟩

lemma(**in** *UP-cring*) *poly-lift-hom-closed*:
assumes *cring* S
assumes $\varphi \in$ *ring-hom* $R\ S$
assumes $p \in$ *carrier* (*UP* R)
shows *poly-lift-hom* $R\ S\ \varphi\ p \in$ *carrier* (*UP* S)
⟨*proof*⟩

lemma(**in** *UP-cring*) *poly-lift-hom-add*:
assumes *cring* S
assumes $\varphi \in$ *ring-hom* $R\ S$
assumes $p \in$ *carrier* (*UP* R)
assumes $q \in$ *carrier* (*UP* R)
shows *poly-lift-hom* $R\ S\ \varphi\ (p \oplus_{UP\ R} q) =$ *poly-lift-hom* $R\ S\ \varphi\ p \oplus_{UP\ S}$ *poly-lift-hom* $R\ S\ \varphi\ q$
⟨*proof*⟩

lemma(**in** *UP-cring*) *poly-lift-hom-mult*:
assumes *cring* S
assumes $\varphi \in$ *ring-hom* $R\ S$
assumes $p \in$ *carrier* (*UP* R)
assumes $q \in$ *carrier* (*UP* R)

shows *poly-lift-hom R S φ (p ⊗$_{UP\ R}$ q) = poly-lift-hom R S φ p ⊗$_{UP\ S}$ poly-lift-hom R S φ q*
⟨*proof*⟩

lemma(**in** *UP-cring*) *poly-lift-hom-extends-hom*:
assumes *cring S*
assumes *φ ∈ ring-hom R S*
assumes *r ∈ carrier R*
shows *poly-lift-hom R S φ (to-polynomial R r) = to-polynomial S (φ r)*
⟨*proof*⟩

lemma(**in** *UP-cring*) *poly-lift-hom-extends-hom′*:
assumes *cring S*
assumes *φ ∈ ring-hom R S*
assumes *r ∈ carrier R*
shows *poly-lift-hom R S φ (monom P r 0) = monom (UP S) (φ r) 0*
⟨*proof*⟩

lemma(**in** *UP-cring*) *poly-lift-hom-smult*:
assumes *cring S*
assumes *φ ∈ ring-hom R S*
assumes *p ∈ carrier (UP R)*
assumes *a ∈ carrier R*
shows *poly-lift-hom R S φ (a ⊙$_{UP\ R}$ p) = φ a ⊙$_{UP\ S}$ (poly-lift-hom R S φ p)*
⟨*proof*⟩

lemma(**in** *UP-cring*) *poly-lift-hom-monom*:
assumes *cring S*
assumes *φ ∈ ring-hom R S*
assumes *r ∈ carrier R*
shows *poly-lift-hom R S φ (monom (UP R) r n) = (monom (UP S) (φ r) n)*
⟨*proof*⟩

lemma(**in** *UP-cring*) *poly-lift-hom-X-var*:
assumes *cring S*
assumes *φ ∈ ring-hom R S*
shows *poly-lift-hom R S φ (monom (UP R) **1**$_R$ 1) = (monom (UP S) **1**$_S$ 1)*
⟨*proof*⟩

lemma(**in** *UP-cring*) *poly-lift-hom-X-var′*:
assumes *cring S*
assumes *φ ∈ ring-hom R S*
shows *poly-lift-hom R S φ (X-poly R) = (X-poly S)*
⟨*proof*⟩

lemma(**in** *UP-cring*) *poly-lift-hom-X-var″*:
assumes *cring S*
assumes *φ ∈ ring-hom R S*
shows *poly-lift-hom R S φ (monom (UP R) **1**$_R$ n) = (monom (UP S) **1**$_S$ n)*

⟨*proof*⟩

lemma(**in** *UP-cring*) *poly-lift-hom-X-var'''*:
assumes *cring S*
assumes $\varphi \in$ *ring-hom R S*
shows *poly-lift-hom R S* φ (*X-poly R* $[\hat{}]_{UP\ R}$ (*n*::*nat*)) = (*X-poly S*) $[\hat{}]_{UP\ S}$ (*n*::*nat*)
⟨*proof*⟩

lemma(**in** *UP-cring*) *poly-lift-hom-X-plus*:
assumes *cring S*
assumes $\varphi \in$ *ring-hom R S*
assumes *a* ∈ *carrier R*
shows *poly-lift-hom R S* φ (*X-poly-plus R a*) = *X-poly-plus S* (φ *a*)
⟨*proof*⟩

lemma(**in** *UP-cring*) *poly-lift-hom-X-plus-nat-pow*:
assumes *cring S*
assumes $\varphi \in$ *ring-hom R S*
assumes *a* ∈ *carrier R*
shows *poly-lift-hom R S* φ (*X-poly-plus R a* $[\hat{}]_{UP\ R}$ (*n*::*nat*)) = *X-poly-plus S* (φ *a*) $[\hat{}]_{UP\ S}$ (*n*::*nat*)
⟨*proof*⟩

lemma(**in** *UP-cring*) *X-poly-plus-nat-pow-closed*:
assumes *a* ∈ *carrier R*
shows *X-poly-plus R a* $[\hat{}]_{UP\ R}$ (*n*::*nat*) ∈ *carrier* (*UP R*)
⟨*proof*⟩

lemma(**in** *UP-cring*) *poly-lift-hom-X-plus-nat-pow-smult*:
assumes *cring S*
assumes $\varphi \in$ *ring-hom R S*
assumes *a* ∈ *carrier R*
assumes *b* ∈ *carrier R*
shows *poly-lift-hom R S* φ (*b* $\odot_{UP\ R}$ *X-poly-plus R a* $[\hat{}]_{UP\ R}$ (*n*::*nat*)) = φ *b* $\odot_{UP\ S}$ *X-poly-plus S* (φ *a*) $[\hat{}]_{UP\ S}$ (*n*::*nat*)
⟨*proof*⟩

lemma(**in** *UP-cring*) *poly-lift-hom-X-minus*:
assumes *cring S*
assumes $\varphi \in$ *ring-hom R S*
assumes *a* ∈ *carrier R*
shows *poly-lift-hom R S* φ (*X-poly-minus R a*) = *X-poly-minus S* (φ *a*)
⟨*proof*⟩

lemma(**in** *UP-cring*) *poly-lift-hom-X-minus-nat-pow*:
assumes *cring S*
assumes $\varphi \in$ *ring-hom R S*
assumes *a* ∈ *carrier R*

shows *poly-lift-hom R S φ (X-poly-minus R a [^]$_{UP\ R}$ (n::nat)) = X-poly-minus S (φ a) [^]$_{UP\ S}$ (n::nat)*
⟨*proof*⟩

lemma(**in** *UP-cring*) *X-poly-minus-nat-pow-closed*:
assumes *a ∈ carrier R*
shows *X-poly-minus R a [^]$_{UP\ R}$ (n::nat) ∈ carrier (UP R)*
⟨*proof*⟩

lemma(**in** *UP-cring*) *poly-lift-hom-X-minus-nat-pow-smult*:
assumes *cring S*
assumes *φ ∈ ring-hom R S*
assumes *a ∈ carrier R*
assumes *b ∈ carrier R*
shows *poly-lift-hom R S φ (b ⊙$_{UP\ R}$ X-poly-minus R a [^]$_{UP\ R}$ (n::nat)) = φ b ⊙$_{UP\ S}$ X-poly-minus S (φ a) [^]$_{UP\ S}$ (n::nat)*
⟨*proof*⟩

lemma(**in** *UP-cring*) *poly-lift-hom-cf*:
assumes *cring S*
assumes *φ ∈ ring-hom R S*
assumes *p ∈ carrier P*
shows *poly-lift-hom R S φ p k = φ (p k)*
⟨*proof*⟩

lemma(**in** *ring*) *ring-hom-monom-term*:
assumes *a ∈ carrier R*
assumes *c ∈ carrier R*
assumes *ring S*
assumes *h ∈ ring-hom R S*
shows *h (a ⊗ c[^](n::nat)) = h a ⊗$_S$ (h c)[^]$_S$n*
⟨*proof*⟩

lemma(**in** *UP-cring*) *poly-lift-hom-eval*:
assumes *cring S*
assumes *φ ∈ ring-hom R S*
assumes *p ∈ carrier P*
assumes *a ∈ carrier R*
shows *UP-cring.to-fun S (poly-lift-hom R S φ p) (φ a) = φ (to-fun p a)*
⟨*proof*⟩

lemma(**in** *UP-cring*) *poly-lift-hom-sub*:
assumes *cring S*
assumes *φ ∈ ring-hom R S*
assumes *p ∈ carrier P*
assumes *q ∈ carrier P*
shows *poly-lift-hom R S φ (compose R p q) = compose S (poly-lift-hom R S φ p) (poly-lift-hom R S φ q)*
⟨*proof*⟩

lemma(**in** *UP-cring*) *poly-lift-hom-comm-taylor-expansion*:
assumes *cring S*
assumes $\varphi \in$ *ring-hom R S*
assumes $p \in$ *carrier P*
assumes $a \in$ *carrier R*
shows *poly-lift-hom R S* φ (*taylor-expansion R a p*) = *taylor-expansion S* (φ *a*) (*poly-lift-hom R S* φ *p*)
⟨*proof*⟩

lemma(**in** *UP-cring*) *poly-lift-hom-comm-taylor-expansion-cf*:
assumes *cring S*
assumes $\varphi \in$ *ring-hom R S*
assumes $p \in$ *carrier* (*UP R*)
assumes $a \in$ *carrier R*
shows φ (*taylor-expansion R a p i*) = *taylor-expansion S* (φ *a*) (*poly-lift-hom R S* φ *p*) *i*
⟨*proof*⟩

lemma(**in** *UP-cring*) *taylor-expansion-cf-closed*:
assumes $p \in$ *carrier P*
assumes $a \in$ *carrier R*
shows *taylor-expansion R a p i* $\in$ *carrier R*
⟨*proof*⟩

lemma(**in** *UP-cring*) *poly-lift-hom-comm-taylor-term*:
assumes *cring S*
assumes $\varphi \in$ *ring-hom R S*
assumes $p \in$ *carrier* (*UP R*)
assumes $a \in$ *carrier R*
shows *poly-lift-hom R S* φ (*taylor-term a p i*) = *UP-cring.taylor-term S* (φ *a*) (*poly-lift-hom R S* φ *p*) *i*
⟨*proof*⟩

lemma(**in** *UP-cring*) *poly-lift-hom-degree-bound*:
assumes *cring S*
assumes $h \in$ *ring-hom R S*
assumes $f \in$ *carrier* (*UP R*)
shows *deg S* (*poly-lift-hom R S h f*) $\leq$ *deg R f*
⟨*proof*⟩

lemma(**in** *UP-cring*) *deg-eqI*:
assumes $f \in$ *carrier* (*UP R*)
assumes *deg R f* $\leq n$
assumes $f\ n \neq \mathbf{0}$
shows *deg R f* = n
⟨*proof*⟩

lemma(**in** *UP-cring*) *poly-lift-hom-degree-eq*:

assumes *cring S*
assumes $h \in$ *ring-hom R S*
assumes h (*lcf f*) $\neq \mathbf{0}_S$
assumes $f \in$ *carrier* (*UP R*)
shows *deg S* (*poly-lift-hom R S h f*) = *deg R f*
⟨*proof*⟩

lemma(**in** *UP-cring*) *poly-lift-hom-lcoeff*:
assumes *cring S*
assumes $h \in$ *ring-hom R S*
assumes h (*lcf f*) $\neq \mathbf{0}_S$
assumes $f \in$ *carrier* (*UP R*)
shows *UP-ring.lcf S* (*poly-lift-hom R S h f*) = h (*lcf f*)
⟨*proof*⟩

end

11 Coefficient List Constructor for Polynomials

definition *list-to-poly* **where**
list-to-poly R as n = (*if n* < *length as then as*!*n else* $\mathbf{0}_R$)

context *UP-ring*
begin

lemma(**in** *UP-ring*) *list-to-poly-closed*:
assumes *set as* ⊆ *carrier R*
shows *list-to-poly R as* ∈ *carrier P*
⟨*proof*⟩

lemma(**in** *UP-ring*) *list-to-poly-zero*[*simp*]:
list-to-poly R [] = $\mathbf{0}_{UP\ R}$
⟨*proof*⟩

lemma(**in** *UP-domain*) *list-to-poly-singleton*:
assumes *a* ∈ *carrier R*
shows *list-to-poly R* [*a*] = *monom P a 0*
⟨*proof*⟩
end

definition *cf-list* **where**
cf-list R p = *map p* [(*0*::*nat*)..< *Suc* (*deg R p*)]

lemma *cf-list-length*:
length (*cf-list R p*) = *Suc* (*deg R p*)
⟨*proof*⟩

lemma *cf-list-entries*:
assumes *i* ≤ *deg R p*

shows (*cf-list R p*)!*i* = *p i*
⟨*proof*⟩

lemma(**in** *UP-ring*) *list-to-poly-cf-list-inv*:
assumes *p* ∈ *carrier* (*UP R*)
shows *list-to-poly R* (*cf-list R p*) = *p*
⟨*proof*⟩

12 Polynomial Rings over a Subring

12.1 Characterizing the Carrier of a Polynomial Ring over a Subring

lemma(**in** *ring*) *carrier-update*:
carrier (*R*⦇*carrier* := *S*⦈) = *S*
$\mathbf{0}_{(R⦇carrier := S⦈)} = \mathbf{0}$
$\mathbf{1}_{(R⦇carrier := S⦈)} = \mathbf{1}$
$(\oplus_{(R⦇carrier := S⦈)}) = (\oplus)$
$(\otimes_{(R⦇carrier := S⦈)}) = (\otimes)$
⟨*proof*⟩

lemma(**in** *UP-cring*) *poly-cfs-subring*:
assumes *subring S R*
assumes *g* ∈ *carrier* (*UP R*)
assumes ⋀*n*. *g n* ∈ *S*
shows *g* ∈ *carrier* (*UP* (*R* ⦇ *carrier* := *S* ⦈))
⟨*proof*⟩

lemma(**in** *UP-cring*) *UP-ring-subring*:
assumes *subring S R*
shows *UP-cring* (*R* ⦇ *carrier* := *S* ⦈) *UP-ring* (*R* ⦇ *carrier* := *S* ⦈)
⟨*proof*⟩

lemma(**in** *UP-cring*) *UP-ring-subring-is-ring*:
assumes *subring S R*
shows *cring* (*UP* (*R* ⦇ *carrier* := *S* ⦈))
⟨*proof*⟩

lemma(**in** *UP-cring*) *UP-ring-subring-add-closed*:
assumes *subring S R*
assumes *g* ∈ *carrier* (*UP* (*R* ⦇ *carrier* := *S* ⦈))
assumes *f* ∈ *carrier* (*UP* (*R* ⦇ *carrier* := *S* ⦈))
shows *f* $\oplus_{UP\ (R ⦇ carrier := S ⦈)}$ *g* ∈ *carrier* (*UP* (*R* ⦇ *carrier* := *S* ⦈))
⟨*proof*⟩

lemma(**in** *UP-cring*) *UP-ring-subring-mult-closed*:
assumes *subring S R*

assumes $g \in carrier\ (UP\ (R \lparr\ carrier := S\ \rparr))$
assumes $f \in carrier\ (UP\ (R \lparr\ carrier := S\ \rparr))$
shows $f \otimes_{UP\ (R \lparr\ carrier := S\ \rparr)} g \in carrier\ (UP\ (R \lparr\ carrier := S\ \rparr))$
⟨*proof*⟩

lemma(**in** *UP-cring*) *UP-ring-subring-car*:
assumes *subring S R*
shows $carrier\ (UP\ (R \lparr\ carrier := S\ \rparr)) = \{h \in carrier\ (UP\ R).\ \forall n.\ h\ n \in S\}$
⟨*proof*⟩

lemma(**in** *UP-cring*) *UP-ring-subring-car-subset*:
assumes *subring S R*
shows $carrier\ (UP\ (R \lparr\ carrier := S\ \rparr)) \subseteq carrier\ (UP\ R)$
⟨*proof*⟩

lemma(**in** *UP-cring*) *UP-ring-subring-car-subset'*:
assumes *subring S R*
assumes $h \in carrier\ (UP\ (R \lparr\ carrier := S\ \rparr))$
shows $h \in\ carrier\ (UP\ R)$
⟨*proof*⟩

lemma(**in** *UP-cring*) *UP-ring-subring-add*:
assumes *subring S R*
assumes $g \in carrier\ (UP\ (R \lparr\ carrier := S\ \rparr))$
assumes $f \in carrier\ (UP\ (R \lparr\ carrier := S\ \rparr))$
shows $g \oplus_{UP\ R} f = g \oplus_{UP\ (R \lparr\ carrier := S\ \rparr)} f$
⟨*proof*⟩

lemma(**in** *UP-cring*) *UP-ring-subring-deg*:
assumes *subring S R*
assumes $g \in carrier\ (UP\ (R \lparr\ carrier := S\ \rparr))$
shows $deg\ R\ g = deg\ (R \lparr\ carrier := S\ \rparr)\ g$
⟨*proof*⟩

lemma(**in** *UP-cring*) *UP-subring-monom*:
assumes *subring S R*
assumes $a \in S$
shows $up\text{-}ring.monom\ (UP\ R)\ a\ n = up\text{-}ring.monom\ (UP\ (R \lparr\ carrier := S\ \rparr))\ a\ n$
⟨*proof*⟩

lemma(**in** *UP-cring*) *UP-ring-subring-mult*:
assumes *subring S R*
assumes $g \in carrier\ (UP\ (R \lparr\ carrier := S\ \rparr))$
assumes $f \in carrier\ (UP\ (R \lparr\ carrier := S\ \rparr))$
shows $g \otimes_{UP\ R} f = g \otimes_{UP\ (R \lparr\ carrier := S\ \rparr)} f$
⟨*proof*⟩

lemma(**in** *UP-cring*) *UP-ring-subring-one*:
assumes *subring S R*
shows $\mathbf{1}_{UP\ R} = \mathbf{1}_{UP\ (R\ (\!|\ carrier := S\ |\!))}$
⟨*proof*⟩

lemma(**in** *UP-cring*) *UP-ring-subring-zero*:
assumes *subring S R*
shows $\mathbf{0}_{UP\ R} = \mathbf{0}_{UP\ (R\ (\!|\ carrier := S\ |\!))}$
⟨*proof*⟩

lemma(**in** *UP-cring*) *UP-ring-subring-nat-pow*:
assumes *subring S R*
assumes $g \in carrier\ (UP\ (R\ (\!|\ carrier := S\ |\!)))$
shows $g[\hat{}]_{UP\ R} n = g[\hat{}]_{UP\ (R\ (\!|\ carrier := S\ |\!))} (n::nat)$
⟨*proof*⟩

lemma(**in** *UP-cring*) *UP-subring-compose-monom*:
assumes *subring S R*
assumes $g \in carrier\ (UP\ (R\ (\!|\ carrier := S\ |\!)))$
assumes $a \in S$
shows *compose R* (*up-ring.monom* (*UP R*) *a n*) *g* = *compose* (*R* (|| *carrier* := *S* ||)) (*up-ring.monom* (*UP* (*R* (|| *carrier* := *S* ||))) *a n*) *g*
⟨*proof*⟩

lemma(**in** *UP-cring*) *UP-subring-compose*:
assumes *subring S R*
assumes $g \in carrier\ (UP\ R)$
assumes $f \in carrier\ (UP\ R)$
assumes $\bigwedge n.\ g\ n \in S$
assumes $\bigwedge n.\ f\ n \in S$
shows *compose R f g* = *compose* (*R* (|| *carrier* := *S* ||)) *f g*
⟨*proof*⟩

12.2 Evaluation over a Subring

lemma(**in** *UP-cring*) *UP-subring-eval*:
assumes *subring S R*
assumes $g \in carrier\ (UP\ (R\ (\!|\ carrier := S\ |\!)))$
assumes $a \in S$
shows *to-function R g a* = *to-function* (*R* (|| *carrier* := *S* ||)) *g a*
⟨*proof*⟩

lemma(**in** *UP-cring*) *UP-subring-eval'*:
assumes *subring S R*
assumes $g \in carrier\ (UP\ (R\ (\!|\ carrier := S\ |\!)))$
assumes $a \in S$
shows *to-fun g a* = *to-function* (*R* (|| *carrier* := *S* ||)) *g a*
⟨*proof*⟩

lemma(**in** *UP-cring*) *UP-subring-eval-closed*:
assumes *subring S R*
assumes $g \in$ *carrier* (*UP* (*R* (| *carrier* := *S* |)))
assumes $a \in S$
shows *to-fun g a* $\in S$
⟨*proof*⟩

12.3 Derivatives and Taylor Expansions over a Subring

lemma(**in** *UP-cring*) *UP-subring-taylor*:
assumes *subring S R*
assumes $g \in$ *carrier* (*UP R*)
assumes $\bigwedge n.\ g\ n \in S$
assumes $a \in S$
shows *taylor-expansion R a g* = *taylor-expansion* (*R* (| *carrier* := *S* |)) *a g*
⟨*proof*⟩

lemma(**in** *UP-cring*) *UP-subring-taylor-closed*:
assumes *subring S R*
assumes $g \in$ *carrier* (*UP R*)
assumes $\bigwedge n.\ g\ n \in S$
assumes $a \in S$
shows *taylor-expansion R a g* $\in$ *carrier* (*UP* (*R* (| *carrier* := *S* |)))
⟨*proof*⟩

lemma(**in** *UP-cring*) *UP-subring-taylor-closed'*:
assumes *subring S R*
assumes $g \in$ *carrier* (*UP* (*R* (| *carrier* := *S* |)))
assumes $a \in S$
shows *taylor-expansion R a g* $\in$ *carrier* (*UP* (*R* (| *carrier* := *S* |)))
⟨*proof*⟩

lemma(**in** *UP-cring*) *UP-subring-taylor'*:
assumes *subring S R*
assumes $g \in$ *carrier* (*UP R*)
assumes $\bigwedge n.\ g\ n \in S$
assumes $a \in S$
shows *taylor-expansion R a g n* $\in S$
⟨*proof*⟩

lemma(**in** *UP-cring*) *UP-subring-deriv*:
assumes *subring S R*
assumes $g \in$ *carrier* (*UP* (*R* (| *carrier* := *S* |)))
assumes $a \in S$
shows *deriv g a*= *UP-cring.deriv* (*R* (| *carrier* := *S* |)) *g a*
⟨*proof*⟩

lemma(**in** *UP-cring*) *UP-subring-deriv-closed*:

assumes *subring S R*
assumes $g \in$ *carrier* (*UP* (*R* (| *carrier* := *S* |)))
assumes $a \in S$
shows *deriv g a* $\in S$
⟨*proof*⟩

lemma(**in** *UP-cring*) *poly-shift-subring-closed*:
assumes *subring S R*
assumes $g \in$ *carrier* (*UP* (*R* (| *carrier* := *S* |)))
shows *poly-shift g* $\in$ *carrier* (*UP* (*R* (| *carrier* := *S* |)))
⟨*proof*⟩

lemma(**in** *UP-cring*) *UP-subring-taylor-appr*:
assumes *subring S R*
assumes $g \in$ *carrier* (*UP* (*R* (| *carrier* := *S* |)))
assumes $a \in S$
assumes $b \in S$
shows $\exists c \in S$. *to-fun g a*= *to-fun g b* $\oplus$ (*deriv g b*)$\otimes$ ($a \ominus b$) $\oplus$ ($c \otimes (a \ominus b)[\widehat{}](2::nat)$)
⟨*proof*⟩

lemma(**in** *UP-cring*) *UP-subring-taylor-appr′*:
assumes *subring S R*
assumes $g \in$ *carrier* (*UP* (*R* (| *carrier* := *S* |)))
assumes $a \in S$
assumes $b \in S$
shows $\exists c\ c'\ c''.\ c \in S \wedge c' \in S \wedge c'' \in S \wedge$ *to-fun g a*= $c \oplus c'\otimes (a \ominus b) \oplus (c'' \otimes (a \ominus b)[\widehat{}](2::nat))$
⟨*proof*⟩

lemma (**in** *UP-cring*) *pderiv-cfs*:
assumes*g* $\in$ *carrier* (*UP R*)
shows *pderiv g n* = [*Suc n*]·(*g* (*Suc n*))
⟨*proof*⟩

lemma(**in** *ring*) *subring-add-pow*:
assumes *subring S R*
assumes $a \in S$
shows $[(n::nat)] \cdot_{R(|carrier := S|)} a = [(n::nat)] \cdot a$
⟨*proof*⟩

lemma(**in** *UP-cring*) *UP-subring-pderiv-equal*:
assumes *subring S R*
assumes $g \in$ *carrier* (*UP* (*R* (| *carrier* := *S* |)))
shows *pderiv g* = *UP-cring.pderiv* (*R*(|*carrier* := *S*|)) *g*
⟨*proof*⟩

lemma(**in** *UP-cring*) *UP-subring-pderiv-closed*:
assumes *subring S R*

assumes $g \in carrier\ (UP\ (R\ (\!|\ carrier := S\ |\!)))$
shows $pderiv\ g \in carrier\ (UP\ (R\ (\!|\ carrier := S\ |\!)))$
⟨*proof*⟩

lemma(**in** *UP-cring*) *UP-subring-pderiv-closed'*:
assumes *subring S R*
assumes $g \in carrier\ (UP\ R)$
assumes $\bigwedge n.\ g\ n \in S$
shows $\bigwedge n.\ pderiv\ g\ n \in S$
⟨*proof*⟩

lemma(**in** *UP-cring*) *taylor-deg-one-expansion-subring*:
assumes $f \in carrier\ (UP\ R)$
assumes *subring S R*
assumes $\bigwedge i.\ f\ i \in S$
assumes $a \in S$
assumes $b \in S$
shows $\exists c \in S.\ to\text{-}fun\ f\ b = (to\text{-}fun\ f\ a) \oplus (deriv\ f\ a) \otimes (b \ominus a) \oplus (c \otimes (b \ominus a)[\widehat{}](2::nat))$
⟨*proof*⟩

lemma(**in** *UP-cring*) *taylor-deg-one-expansion-subring'*:
assumes $f \in carrier\ (UP\ R)$
assumes *subring S R*
assumes $\bigwedge i.\ f\ i \in S$
assumes $a \in S$
assumes $b \in S$
shows $\exists c \in S.\ to\text{-}fun\ f\ b = (to\text{-}fun\ f\ a) \oplus (to\text{-}fun\ (pderiv\ f)\ a) \otimes (b \ominus a) \oplus (c \otimes (b \ominus a)[\widehat{}](2::nat))$
⟨*proof*⟩

end
theory *Supplementary-Ring-Facts*
imports *HOL−Algebra.Ring*
HOL−Algebra.UnivPoly
HOL−Algebra.Subrings

begin

13 Supplementary Ring Facts

The nonzero elements of a ring.

definition *nonzero* :: $('a, 'b)\ ring\text{-}scheme \Rightarrow 'a\ set$ **where**
$nonzero\ R = \{a \in carrier\ R.\ a \neq \mathbf{0}_R\}$

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

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

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

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

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

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

lemma(**in** *domain*) *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*⟩

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** *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*⟩

end

14 Extended integers (i.e. with infinity)

This section formalizes the extended integers, which serve as the codomain for the p-adic valuation. The element ∞ is added to the integers to serve as a maximxal element in the order, which is the valuation of 0.

theory *Extended-Int*
imports *Main HOL−Library.Countable HOL−Library.Order-Continuity HOL−Library.Extended-Nat*
begin

The following is based very closely on the theory *HOL−Library.Extended-Nat* from the standard Isabelle distribution, with adaptations made to formalize the integers extended with an element for infinity. This is the standard range for the valuation function on a discretely valued ring such as the field of p-adic numbers, such as in [4].

context
fixes f :: *nat* ⇒ 'a::{*canonically-ordered-monoid-add*, *linorder-topology*, *complete-linorder*}
begin

lemma *sums-SUP*[*simp*, *intro*]: f *sums* (*SUP* n. $\sum i{<}n$. $f\ i$)
⟨*proof*⟩

lemma *suminf-eq-SUP*: *suminf* f = (*SUP* n. $\sum i{<}n$. $f\ i$)
⟨*proof*⟩

end

14.1 Type definition

We extend the standard natural numbers by a special value indicating infinity.

typedef *eint* = *UNIV* :: *int option set* ⟨*proof*⟩

definition *eint* :: *int* ⇒ *eint* **where**
eint n = *Abs-eint* (*Some n*)

instantiation *eint* :: *infinity*
begin

definition ∞ = *Abs-eint None*
instance ⟨*proof*⟩

end

fun *int-option-enumeration* :: *int option* ⇒ *nat* **where**
int-option-enumeration (*Some n*) = (*if n* ≥ *0 then nat* (*2*∗(*n* + *1*)) *else nat* (*2*∗(−*n*) + *1*))|
int-option-enumeration None = (*0*::*nat*)

lemma *int-option-enumeration-inj*:
inj int-option-enumeration
⟨*proof*⟩

definition *eint-enumeration* **where**
eint-enumeration = *int-option-enumeration* ∘ *Rep-eint*

lemma *eint-enumeration-inj*:
inj eint-enumeration
⟨*proof*⟩

instance *eint* :: *countable*
⟨*proof*⟩

old-rep-datatype *eint* ∞ :: *eint*
⟨*proof*⟩

declare [[*coercion eint*::*int*⇒*eint*]]

lemmas *eint2-cases* = *eint.exhaust*[*case-product eint.exhaust*]
lemmas *eint3-cases* = *eint.exhaust*[*case-product eint.exhaust eint.exhaust*]

lemma *not-infinity-eq* [*iff*]: (*x* ≠ ∞) = (∃ *i*. *x* = *eint i*)
⟨*proof*⟩

lemma *not-eint-eq* [*iff*]: $(\forall\, y.\ x \neq \mathit{eint}\ y) = (x = \infty)$
⟨*proof*⟩

lemma *eint-ex-split*: $(\exists\, c{::}\mathit{eint}.\ P\ c) \longleftrightarrow P\ \infty \vee (\exists\, c{::}\mathit{int}.\ P\ c)$
⟨*proof*⟩

primrec *the-eint* :: *eint* ⇒ *int*
where *the-eint* (*eint n*) = *n*

14.2 Constructors and numbers

instantiation *eint* :: *zero-neq-one*
begin

definition
0 = *eint 0*

definition
1 = *eint 1*

instance
⟨*proof*⟩

end

lemma *eint-0* [*code-post*]: *eint 0* = *0*
⟨*proof*⟩

lemma *eint-1* [*code-post*]: *eint 1* = *1*
⟨*proof*⟩

lemma *eint-0-iff*: *eint x* = *0* ⟷ *x* = *0 0* = *eint x* ⟷ *x* = *0*
⟨*proof*⟩

lemma *eint-1-iff*: *eint x* = *1* ⟷ *x* = *1 1* = *eint x* ⟷ *x* = *1*
⟨*proof*⟩

lemma *infinity-ne-i0* [*simp*]: $(\infty{::}\mathit{eint}) \neq 0$
⟨*proof*⟩

lemma *i0-ne-infinity* [*simp*]: $0 \neq (\infty{::}\mathit{eint})$
⟨*proof*⟩

lemma *zero-one-eint-neq*:
¬ *0* = (*1*::*eint*)
¬ *1* = (*0*::*eint*)
⟨*proof*⟩

lemma *infinity-ne-i1* [*simp*]: $(\infty::eint) \neq 1$
⟨*proof*⟩

lemma *i1-ne-infinity* [*simp*]: $1 \neq (\infty::eint)$
⟨*proof*⟩

14.3 Addition

instantiation *eint* :: *comm-monoid-add*
begin

definition [*nitpick-simp*]:
$m + n = (case\ m\ of\ \infty \Rightarrow \infty \mid eint\ m \Rightarrow (case\ n\ of\ \infty \Rightarrow \infty \mid eint\ n \Rightarrow eint\ (m + n)))$

lemma *plus-eint-simps* [*simp*, *code*]:
fixes q :: *eint*
shows $eint\ m + eint\ n = eint\ (m + n)$
and $\infty + q = \infty$
and $q + \infty = \infty$
⟨*proof*⟩

instance
⟨*proof*⟩

end

lemma *eSuc-eint*: $(eint\ n) + 1 = eint\ (n + 1)$
⟨*proof*⟩

lemma *eSuc-infinity* [*simp*]: $\infty + (1::eint) = \infty$
⟨*proof*⟩

lemma *eSuc-inject* [*simp*]: $m + (1::eint) = n + 1 \longleftrightarrow m = n$
⟨*proof*⟩

lemma *eSuc-eint-iff*: $x + 1 = eint\ y \longleftrightarrow (\exists n.\ y = n + 1 \wedge x = eint\ n)$
⟨*proof*⟩

lemma *enat-eSuc-iff*: $eint\ y = x + 1 \longleftrightarrow (\exists n.\ y = n + 1 \wedge eint\ n = x)$
⟨*proof*⟩

lemma *iadd-Suc*: $((m::eint) + 1) + n = (m + n) + 1$
⟨*proof*⟩

lemma *iadd-Suc-right*: $(m::eint) + (n + 1) = (m + n) + 1$
⟨*proof*⟩

14.4 Multiplication

instantiation *eint* :: {*comm-semiring*}
begin

definition *times-eint-def* [*nitpick-simp*]:
$m * n = (case\ m\ of\ \infty \Rightarrow\ \infty \mid eint\ m \Rightarrow$
$(case\ n\ of\ \infty \Rightarrow \infty \mid eint\ n \Rightarrow eint\ (m * n)))$

lemma *times-eint-simps* [*simp*, *code*]:
$eint\ m * eint\ n = eint\ (m * n)$
$\infty * \infty = (\infty::eint)$
$\infty * eint\ n = \infty$
$eint\ m * \infty = \infty$
⟨*proof*⟩

lemma *sum-infinity-imp-summand-infinity*:
assumes $a + b = (\infty::eint)$
shows $a = \infty \vee b = \infty$
⟨*proof*⟩

lemma *sum-finite-imp-summands-finite*:
assumes $a + b \neq (\infty::eint)$
shows $a \neq \infty\ b \neq \infty$
⟨*proof*⟩

instance
⟨*proof*⟩

end

lemma *mult-one-right*[*simp*]:
$(n::eint)*1 = n$
⟨*proof*⟩

lemma *mult-one-left*[*simp*]:
$1*(n::eint) = n$
⟨*proof*⟩

lemma *mult-eSuc*: $((m::eint) + 1) * n = m * n + n$
⟨*proof*⟩

lemma *mult-eSuc'*: $((m::eint) + 1) * n = n + m * n$
⟨*proof*⟩

lemma *mult-eSuc-right*: $(m::eint) * (n + 1) = m * n + m$
⟨*proof*⟩

lemma *mult-eSuc-right'*: $(m::eint) * (n + 1) = m + m * n$
⟨*proof*⟩

14.5 Numerals

lemma *numeral-eq-eint*:
$numeral\ k = eint\ (numeral\ k)$
⟨*proof*⟩

lemma *eint-numeral* [*code-abbrev*]:
$eint\ (numeral\ k) = numeral\ k$
⟨*proof*⟩

lemma *infinity-ne-numeral* [*simp*]: $(\infty::eint) \neq numeral\ k$
⟨*proof*⟩

lemma *numeral-ne-infinity* [*simp*]: $numeral\ k \neq (\infty::eint)$
⟨*proof*⟩

14.6 Subtraction

instantiation *eint* :: *minus*
begin

definition *diff-eint-def*:
$a - b = (case\ a\ of\ (eint\ x) \Rightarrow (case\ b\ of\ (eint\ y) \Rightarrow eint\ (x - y) \mid \infty \Rightarrow \infty) \mid \infty \Rightarrow \infty)$

instance ⟨*proof*⟩

end

lemma *idiff-eint-eint* [*simp*, *code*]: $eint\ a - eint\ b = eint\ (a - b)$
⟨*proof*⟩

lemma *idiff-infinity* [*simp*, *code*]: $\infty - n = (\infty::eint)$
⟨*proof*⟩

lemma *idiff-infinity-right* [*simp*, *code*]: $eint\ a - \infty = \infty$
⟨*proof*⟩

lemma *idiff-0* [*simp*]: $(0::eint) - n = -n$
⟨*proof*⟩

lemmas *idiff-eint-0* [*simp*] = *idiff-0* [*unfolded zero-eint-def*]

lemma *idiff-0-right* [*simp*]: $(n::eint) - 0 = n$

⟨*proof*⟩

lemmas *idiff-eint-0-right* [*simp*] = *idiff-0-right* [*unfolded zero-eint-def*]

lemma *idiff-self* [*simp*]: $n \neq \infty \Longrightarrow (n::eint) - n = 0$
⟨*proof*⟩

lemma *eSuc-minus-eSuc* [*simp*]: $((n::eint) + 1) - (m + 1) = n - m$
⟨*proof*⟩

lemma *eSuc-minus-1* [*simp*]: $((n::eint) + 1) - 1 = n$
⟨*proof*⟩

14.7 Ordering

instantiation *eint* :: *linordered-ab-semigroup-add*
begin

definition [*nitpick-simp*]:
$m \leq n = (case\ n\ of\ eint\ n1 \Rightarrow (case\ m\ of\ eint\ m1 \Rightarrow m1 \leq n1 \mid \infty \Rightarrow False) \mid \infty \Rightarrow True)$

definition [*nitpick-simp*]:
$m < n = (case\ m\ of\ eint\ m1 \Rightarrow (case\ n\ of\ eint\ n1 \Rightarrow m1 < n1 \mid \infty \Rightarrow True) \mid \infty \Rightarrow False)$

lemma *eint-ord-simps* [*simp*]:
$eint\ m \leq eint\ n \longleftrightarrow m \leq n$
$eint\ m < eint\ n \longleftrightarrow m < n$
$q \leq (\infty::eint)$
$q < (\infty::eint) \longleftrightarrow q \neq \infty$
$(\infty::eint) \leq q \longleftrightarrow q = \infty$
$(\infty::eint) < q \longleftrightarrow False$
⟨*proof*⟩

lemma *numeral-le-eint-iff* [*simp*]:
shows $numeral\ m \leq eint\ n \longleftrightarrow numeral\ m \leq n$
⟨*proof*⟩

lemma *numeral-less-eint-iff* [*simp*]:
shows $numeral\ m < eint\ n \longleftrightarrow numeral\ m < n$
⟨*proof*⟩

lemma *eint-ord-code* [*code*]:
$eint\ m \leq eint\ n \longleftrightarrow m \leq n$
$eint\ m < eint\ n \longleftrightarrow m < n$
$q \leq (\infty::eint) \longleftrightarrow True$
$eint\ m < \infty \longleftrightarrow True$

$\infty \leq \mathit{eint}\ n \longleftrightarrow \mathit{False}$
$(\infty{::}\mathit{eint}) < q \longleftrightarrow \mathit{False}$
⟨*proof*⟩

lemma *eint-ord-plus-one*[*simp*]:
assumes $\mathit{eint}\ n \leq x$
assumes $x < y$
shows $\mathit{eint}\ (n + 1) \leq y$
⟨*proof*⟩

instance
⟨*proof*⟩

end

instance *eint* :: {*strict-ordered-comm-monoid-add*}
⟨*proof*⟩

lemma *add-diff-assoc-eint*: $z \leq y \Longrightarrow x + (y - z) = x + y - (z{::}\mathit{eint})$
⟨*proof*⟩

lemma *eint-ord-number* [*simp*]:
$(\mathit{numeral}\ m :: \mathit{eint}) \leq \mathit{numeral}\ n \longleftrightarrow (\mathit{numeral}\ m :: \mathit{nat}) \leq \mathit{numeral}\ n$
$(\mathit{numeral}\ m :: \mathit{eint}) < \mathit{numeral}\ n \longleftrightarrow (\mathit{numeral}\ m :: \mathit{nat}) < \mathit{numeral}\ n$
⟨*proof*⟩

lemma *infinity-ileE* [*elim*!]: $\infty \leq \mathit{eint}\ m \Longrightarrow R$
⟨*proof*⟩

lemma *infinity-ilessE* [*elim*!]: $\infty < \mathit{eint}\ m \Longrightarrow R$
⟨*proof*⟩

lemma *imult-infinity*: $(0{::}\mathit{eint}) < n \Longrightarrow \infty * n = \infty$
⟨*proof*⟩

lemma *imult-infinity-right*: $(0{::}\mathit{eint}) < n \Longrightarrow n * \infty = \infty$
⟨*proof*⟩

lemma *min-eint-simps* [*simp*]:
$\mathit{min}\ (\mathit{eint}\ m)\ (\mathit{eint}\ n) = \mathit{eint}\ (\mathit{min}\ m\ n)$
$\mathit{min}\ q\ (\infty{::}\mathit{eint}) = q$
$\mathit{min}\ (\infty{::}\mathit{eint})\ q = q$
⟨*proof*⟩

lemma *max-eint-simps* [*simp*]:
$\mathit{max}\ (\mathit{eint}\ m)\ (\mathit{eint}\ n) = \mathit{eint}\ (\mathit{max}\ m\ n)$
$\mathit{max}\ q\ \infty = (\infty{::}\mathit{eint})$

$max\ \infty\ q = (\infty::eint)$
⟨*proof*⟩

lemma *eint-ile*: $n \leq eint\ m \Longrightarrow \exists\, k.\ n = eint\ k$
⟨*proof*⟩

lemma *eint-iless*: $n < eint\ m \Longrightarrow \exists\, k.\ n = eint\ k$
⟨*proof*⟩

lemma *iadd-le-eint-iff*:
$x + y \leq eint\ n \longleftrightarrow (\exists\, y'\ x'.\ x = eint\ x' \wedge y = eint\ y' \wedge x' + y' \leq n)$
⟨*proof*⟩

lemma *chain-incr*: $\forall\, i.\ \exists\, j.\ Y\ i < Y\ j \Longrightarrow \exists\, j.\ eint\ k < Y\ j$
⟨*proof*⟩

lemma *eint-ord-Suc*:
assumes $(x::eint) < y$
shows $x + 1 < y + 1$
⟨*proof*⟩

lemma *eSuc-ile-mono* [*simp*]: $(n::eint) + 1 \leq m + 1 \longleftrightarrow n \leq m$
⟨*proof*⟩

lemma *eSuc-mono* [*simp*]: $(n::eint) + 1 < m + 1 \longleftrightarrow n < m$
⟨*proof*⟩

lemma *ile-eSuc* [*simp*]: $(n::eint) \leq n + 1$
⟨*proof*⟩

lemma *ileI1*: $(m::eint) < n \Longrightarrow m + 1 \leq n$
⟨*proof*⟩

lemma *Suc-ile-eq*: $eint\ (m + 1) \leq n \longleftrightarrow eint\ m < n$
⟨*proof*⟩

lemma *iless-Suc-eq* [*simp*]: $eint\ m < n + 1 \longleftrightarrow eint\ m \leq n$
⟨*proof*⟩

lemma *eSuc-max*: $(max\ (x::eint)\ y) + 1 = max\ (x+1)\ (y+1)$
⟨*proof*⟩

lemma *eSuc-Max*:
assumes *finite A* $A \neq (\{\}::eint\ set)$
shows $(Max\ A) + 1 = Max\ ((+)1\ `\ A)$
⟨*proof*⟩

instantiation *eint* :: {*order-top*}
begin

definition *top-eint* :: *eint* **where** *top-eint* = ∞

instance
⟨*proof*⟩

end

lemma *finite-eint-bounded*:
assumes *le-fin*: $\bigwedge y.\ y \in A \Longrightarrow eint\ m \leq y \wedge y \leq eint\ n$
shows *finite A*
⟨*proof*⟩

14.8 Cancellation simprocs

lemma *add-diff-cancel-eint*[*simp*]: $x \neq \infty \Longrightarrow x + y - x = (y::eint)$
⟨*proof*⟩

lemma *eint-add-left-cancel*: $a + b = a + c \longleftrightarrow a = (\infty::eint) \vee b = c$
⟨*proof*⟩

lemma *eint-add-left-cancel-le*: $a + b \leq a + c \longleftrightarrow a = (\infty::eint) \vee b \leq c$
⟨*proof*⟩

lemma *eint-add-left-cancel-less*: $a + b < a + c \longleftrightarrow a \neq (\infty::eint) \wedge b < c$
⟨*proof*⟩

lemma *plus-eq-infty-iff-eint*: $(m::eint) + n = \infty \longleftrightarrow m{=}\infty \vee n{=}\infty$
⟨*proof*⟩

⟨*ML*⟩

TODO: add regression tests for these simprocs

TODO: add simprocs for combining and cancelling numerals

14.9 Well-ordering

lemma *less-eintE*:
[| $n < eint\ m$; !!k. $n = eint\ k$ ==> $k < m$ ==> P |] ==> P
⟨*proof*⟩

lemma *less-infinityE*:
[| $n < \infty$; !!k. $n = eint\ k$ ==> P |] ==> P
⟨*proof*⟩

14.10 Traditional theorem names

lemmas *eint-defs* = *zero-eint-def one-eint-def*
plus-eint-def less-eq-eint-def less-eint-def

instantiation *eint* :: *uminus*
begin

definition
$-\ b = (case\ b\ of\ \infty \Rightarrow \infty \mid eint\ m \Rightarrow eint\ (-m))$

lemma *eint-uminus-eq*:
$(a::eint) + (-a) = a - a$
⟨*proof*⟩

instance⟨*proof*⟩
end

15 Additional Lemmas (Useful for the Proof of Hensel's Lemma)

lemma *eint-mult-mono*:
assumes $(c::eint) > 0 \wedge c \neq \infty$
assumes $k > n$
shows $k*c > n*c$
⟨*proof*⟩

lemma *eint-mult-mono'*:
assumes $(c::eint) \geq 0 \wedge c \neq \infty$
assumes $k > n$
shows $k*c \geq n*c$
⟨*proof*⟩

lemma *eint-minus-le*:
assumes $(b::eint) < c$
shows $c - b > 0$
⟨*proof*⟩

lemma *eint-nat-times*:
assumes $(c::eint) > 0$
shows $(Suc\ n)*(c::eint) > 0$
⟨*proof*⟩

lemma *eint-pos-times-is-pos*:
assumes $(c::eint) > 0$
assumes $b > 0$
shows $b*c > 0$
⟨*proof*⟩

lemma *eint-nat-is-pos*:
$eint\ (Suc\ n) > 0$
⟨*proof*⟩

lemma *eint-pow-int-is-pos*:
assumes $n > 0$
shows $eint\ n > 0$
⟨*proof*⟩

lemma *eint-nat-times'*:
assumes $(c::eint) \geq 0$
shows $(Suc\ n)*c \geq 0$
⟨*proof*⟩

lemma *eint-pos-int-times-ge*:
assumes $(c::eint) \geq 0$
assumes $n > 0$
shows $eint\ n * c \geq c$
⟨*proof*⟩

lemma *eint-pos-int-times-gt*:
assumes $(c::eint) > 0$
assumes $c \neq \infty$
assumes $n > 1$
shows $eint\ n * c > c$
⟨*proof*⟩

lemma *eint-add-cancel-fact*[*simp*]:
assumes $(c::eint) \neq \infty$
shows $c + (b - c) = b$
⟨*proof*⟩

lemma *nat-mult-not-infty*[*simp*]:
assumes $c \neq \infty$
shows $(eint\ n) * c \neq \infty$
⟨*proof*⟩

lemma *eint-minus-distl*:
assumes $(b::eint) \neq d$
shows $b*c - d*c = (b-d)*c$
⟨*proof*⟩

lemma *eint-minus-distr*:
assumes $(b::eint) \neq d$
shows $c*(b - d) = c*b - c*d$
⟨*proof*⟩

lemma *eint-int-minus-distr*:
$(eint\ n)*c - (eint\ m)*c = eint\ (n - m) * c$

⟨*proof*⟩

lemma *eint-2-minus-1-mult*[*simp*]:
$2*(b::eint) - b = b$
⟨*proof*⟩

lemma *eint-minus-comm*:
$(d::eint) + b - c = d - c + b$
⟨*proof*⟩

lemma *ge-plus-pos-imp-gt*:
assumes $(c::eint) \neq \infty$
assumes $(b::eint) > 0$
assumes $d \geq c + b$
shows $d > c$
⟨*proof*⟩

lemma *eint-minus-ineq*:
assumes $(c::eint) \neq \infty$
assumes $b \geq d$
shows $b - c \geq d - c$
⟨*proof*⟩

lemma *eint-minus-ineq'*:
assumes $(c::eint) \neq \infty$
assumes $b \geq d$
assumes $(e::eint) > 0$
assumes $e \neq \infty$
shows $e*(b - c) \geq e*(d - c)$
⟨*proof*⟩

lemma *eint-minus-ineq''*:
assumes $(c::eint) \neq \infty$
assumes $b \geq d$
assumes $(e::eint) > 0$
assumes $e \neq \infty$
shows $e*(b - c) \geq e*d - e*c$
⟨*proof*⟩

lemma *eint-min-ineq*:
assumes $(b::eint) \geq min\ c\ d$
assumes $c > e$
assumes $d > e$
shows $b > e$
⟨*proof*⟩

lemma *eint-plus-times*:
assumes $(d::eint) \geq 0$
assumes $(b::eint) \geq c + (eint\ k)*d$

assumes $k \geq l$
shows $b \geq c + l*d$
⟨*proof*⟩
end
theory *Padic-Construction*
imports *HOL−Number-Theory.Residues HOL−Algebra.RingHom HOL−Algebra.IntRing*
begin

type-synonym *padic-int* = *nat* ⇒ *int*

16 Inverse Limit Construction of the p-adic Integers

This section formalizes the standard construction of the p-adic integers as the inverse limit of the finite rings $\mathbb{Z}/p^n\mathbb{Z}$ along the residue maps $\mathbb{Z}/p^n\mathbb{Z} \mapsto \mathbb{Z}/p^n\mathbb{Z}$ defined by $x \mapsto x \mod p^m$ when $n \geq m$. This is exposited, for example, in section 7.6 of [3]. The other main route for formalization is to first define the p-adic absolute value $|\cdot|_p$ on the rational numbers, and then define the field $\mathbb{Q}_p$ of p-adic numbers as the completion of the rationals under this absolute value. One can then define the ring of p-adic integers $\mathbb{Z}_p$ as the unit ball in $\mathbb{Q}_p$ using the unique extension of $|\cdot|_p$. There exist advantages and disadvantages to both approaches. The primary advantage to the absolute value approach is that the construction can be done generically using existing libraries for completions of normed fields. There are difficulties associated with performing such a construction in Isabelle using existing HOL formalizations. The chief issue is that the tools in HOL-Analysis require that a metric space be a type. If one then wanted to construct the fields $\mathbb{Q}_p$ as metric spaces, one would have to circumvent the apparent dependence on the parameter p, as Isabelle does not support dependent types. A workaround to this proposed by José Manuel Rodríguez Caballero on the Isabelle mailing list is to define a typeclass for fields $\mathbb{Q}_p$ as the completions of the rational numbers with a non-Archimedean absolute value. By Ostrowski's Theorem, any such absolute value must be a p-adic absolute value. We can recover the parameter p from a completion under one of these absolute values as the cardinality of the residue field.

Our approach uses HOL-Algebra, where algebraic structures are constructed as records which carry the data of the underlying carrier set plus other algebraic operations, and assumptions about these structures can be organized into locales. This approach is practical for abstract algebraic reasoning where definitions of structures which are dependent on object-level parameters are ubiquitous. Using this approach, we define $\mathbb{Z}_p$ directly as an inverse limit of rings, from which $\mathbb{Q}_p$ can later be defined as the field of fractions.

16.1 Canonical Projection Maps Between Residue Rings

definition *residue* :: *int* ⇒ *int* ⇒ *int* **where**
residue n m = *m mod n*

lemma *residue-is-hom-0*:
assumes *n* > *1*
shows *residue n* ∈ *ring-hom* $\mathcal{Z}$ (*residue-ring n*)
⟨*proof*⟩

The residue map is a ring homomorphism from $\mathbb{Z}/m\mathbb{Z} \to \mathbb{Z}/n\mathbb{Z}$ when n divides m

lemma *residue-is-hom-1*:
assumes *n* > *1*
assumes *m* > *1*
assumes *n dvd m*
shows *residue n* ∈ *ring-hom* (*residue-ring m*) (*residue-ring n*)
⟨*proof*⟩

lemma *residue-id*:
assumes *x* ∈ *carrier* (*residue-ring n*)
assumes *n* ≥*0*
shows *residue n x* = *x*
⟨*proof*⟩

The residue map is a ring homomorphism from $\mathbb{Z}/p^n\mathbb{Z} \to \mathbb{Z}/p^m\mathbb{Z}$ when $n \geq m$:

lemma *residue-hom-p*:
assumes (*n*::*nat*) ≥ *m*
assumes *m* >*0*
assumes *prime* (*p*::*int*)
shows *residue* (*p*^*m*) ∈ *ring-hom* (*residue-ring* (*p*^*n*)) (*residue-ring* (*p*^*m*))
⟨*proof*⟩

16.2 Defining the Set of p-adic Integers

The set of p-adic integers is the set of all maps $f : \mathbb{N} \to \mathbb{Z}$ which maps $n \to \{0, ..., p^n - 1\}$ such that $fm \mod p^n = fn$ when $m \geq n$. A p-adic integer x consists of the data of a residue map $x \mapsto x \mod p^n$ which commutes with further reduction $\mod p^m$. This formalization is specialized to just the p-adics, but this definition would work essentially as-is for any family of rings and residue maps indexed by a partially ordered type.

definition *padic-set* :: *int* ⇒ *padic-int set* **where**
padic-set p = {*f*::*nat* ⇒ *int* .(∀ *m*::*nat*. (*f m*) ∈ *carrier* (*residue-ring* (*p*^*m*)))

∧(∀ (*n*::*nat*) (*m*::*nat*). *n* > *m* ⟶ *residue* (*p*^*m*) (*f n*) = (*f m*)) }

lemma *padic-set-res-closed*:
assumes $f \in$ *padic-set p*
shows $(f\ m) \in (carrier\ (residue\text{-}ring\ (p\hat{}m)))$
⟨*proof*⟩

lemma *padic-set-res-coherent*:
assumes $f \in$ *padic-set p*
assumes $n \geq m$
assumes *prime p*
shows $residue\ (p\hat{}m)\ (f\ n) = (f\ m)$
⟨*proof*⟩

A consequence of this formalization is that each p-adic number is trivially defined to take a value of 0 at 0:

lemma *padic-set-zero-res*:
assumes *prime p*
assumes $f \in (padic\text{-}set\ p)$
shows $f\ 0 = 0$
⟨*proof*⟩

lemma *padic-set-memI*:
fixes f :: *padic-int*
assumes $\bigwedge m.\ (f\ m) \in (carrier\ (residue\text{-}ring\ (p\hat{}m)))$
assumes $(\bigwedge (m::nat)\ n.\ (n > m \Longrightarrow (residue\ (p\hat{}m)\ (f\ n) = (f\ m))))$
shows $f \in padic\text{-}set\ (p::int)$
⟨*proof*⟩

lemma *padic-set-memI'*:
fixes f :: *padic-int*
assumes $\bigwedge m.\ (f\ m) \in \{0..<p\hat{}m\}$
assumes $\bigwedge (m::nat)\ n.\ n > m \Longrightarrow (f\ n)\ mod\ p\hat{}m = (f\ m)$
shows $f \in padic\text{-}set\ (p::int)$
⟨*proof*⟩

17 The standard operations on the p-adic integers

17.1 Addition

Addition and multiplication are defined componentwise on residue rings:

definition *padic-add* :: *int* ⇒ *padic-int* ⇒ *padic-int* ⇒ *padic-int*
where $padic\text{-}add\ p\ f\ g \equiv (\lambda\ n.\ (f\ n)\ \oplus_{(residue\text{-}ring\ (p\hat{}n))}\ (g\ n))$

lemma *padic-add-res*:
$(padic\text{-}add\ p\ f\ g)\ n = (f\ n)\ \oplus_{(residue\text{-}ring\ (p\hat{}n))}\ (g\ n)$
⟨*proof*⟩

Definition of the p-adic additive unit:

definition *padic-zero* :: *int* ⇒ *padic-int* **where**

padic-zero p ≡ (λ*n*. *0*)

lemma *padic-zero-simp*:
padic-zero p n = $\mathbf{0}_{\textit{residue-ring } (p\hat{}n)}$
padic-zero p n = *0*
⟨*proof*⟩

lemma *padic-zero-in-padic-set*:
assumes *p* > *0*
shows *padic-zero p* ∈ *padic-set p*
⟨*proof*⟩

p-adic additive inverses:

definition *padic-a-inv* :: *int* ⇒ *padic-int* ⇒ *padic-int* **where**
padic-a-inv p f ≡ λ *n*. $\ominus_{\textit{residue-ring } (p\hat{}n)}$ (*f n*)

lemma *padic-a-inv-simp*:
padic-a-inv p f n≡ $\ominus_{\textit{residue-ring } (p\hat{}n)}$ (*f n*)
⟨*proof*⟩

lemma *padic-a-inv-simp′*:
assumes *prime p*
assumes *f* ∈ *padic-set p*
assumes *n* >*0*
shows *padic-a-inv p f n* = (*if n*=*0 then 0 else* (− (*f n*)) *mod* (*p*^*n*))
⟨*proof*⟩

We show that *padic-set* is closed under additive inverses. Note that we have to treat the case of residues at 0 separately.

lemma *residue-1-prop*:
$\ominus_{\textit{residue-ring } 1}$ $\mathbf{0}_{\textit{residue-ring } 1}$ = $\mathbf{0}_{\textit{residue-ring } 1}$
⟨*proof*⟩

lemma *residue-1-zero*:
residue 1 n = *0*
⟨*proof*⟩

lemma *padic-a-inv-in-padic-set*:
assumes *f* ∈ *padic-set p*
assumes *prime* (*p*::*int*)
shows (*padic-a-inv p f*) ∈ *padic-set p*
⟨*proof*⟩

17.2 Multiplication

definition *padic-mult* :: *int* ⇒ *padic-int* ⇒ *padic-int* ⇒ *padic-int*
where *padic-mult p f g* ≡ (λ *n*. (*f n*) $\otimes_{(\textit{residue-ring } (p\hat{}n))}$ (*g n*))

lemma *padic-mult-res*:

$(padic\text{-}mult\ p\ f\ g)\ n = (f\ n) \otimes_{(residue\text{-}ring\ (p\hat{}n))}\ (g\ n)$
⟨*proof*⟩

Definition of the p-adic multiplicative unit:

definition *padic-one* :: *int* ⇒ *padic-int* **where**
padic-one p ≡ (λ*n*.(*if n=0 then 0 else 1*))

lemma *padic-one-simp*:
assumes *n* >*0*
shows *padic-one p n* = $\mathbf{1}_{residue\text{-}ring\ (p\hat{}n)}$
padic-one p n = 1
⟨*proof*⟩

lemma *padic-one-in-padic-set*:
assumes *prime p*
shows *padic-one p* ∈ *padic-set p*
⟨*proof*⟩

lemma *padic-simps*:
padic-zero p n = $\mathbf{0}_{residue\text{-}ring\ (p\hat{}n)}$
padic-a-inv p f n ≡ $\ominus_{residue\text{-}ring\ (p\hat{}n)}\ (f\ n)$
$(padic\text{-}mult\ p\ f\ g)\ n = (f\ n) \otimes_{(residue\text{-}ring\ (p\hat{}n))}\ (g\ n)$
$(padic\text{-}add\ p\ f\ g)\ n = (f\ n) \oplus_{(residue\text{-}ring\ (p\hat{}n))}\ (g\ n)$
n>*0* ⟹*padic-one p n* = $\mathbf{1}_{residue\text{-}ring\ (p\hat{}n)}$
⟨*proof*⟩

lemma *residue-1-mult*:
assumes *x* ∈ *carrier* (*residue-ring 1*)
assumes *y* ∈ *carrier* (*residue-ring 1*)
shows $x \otimes_{residue\text{-}ring\ 1}\ y = 0$
⟨*proof*⟩

lemma *padic-mult-in-padic-set*:
assumes *f* ∈ (*padic-set p*)
assumes *g* ∈ (*padic-set p*)
assumes *prime p*
shows (*padic-mult p f g*)∈ (*padic-set p*)
⟨*proof*⟩

18 The p-adic Valuation

This section defines the integer-valued p-adic valuation. Maps 0 to -1 for now, otherwise is correct. We want the valuation to be integer-valued, but in practice we know it will always be positive. When we extend the valuation from the p-adic integers to the p-adic field we will have elements of negative valuation.

definition *padic-val* :: *int* ⇒ *padic-int* ⇒ *int* **where**
padic-val p f ≡ *if* (*f* = *padic-zero p*) *then* −*1 else int* (*LEAST k*::*nat*. (*f* (*Suc k*)) ≠ $\mathbf{0}_{\textit{residue-ring}\ (p\hat{}(\textit{Suc}\ k))}$)

Characterization of *padic_val* on nonzero elements

lemma *val-of-nonzero*:
assumes *f* ∈ *padic-set p*
assumes *f* ≠ *padic-zero p*
assumes *prime p*
shows *f* (*nat* (*padic-val p f*) + *1*) ≠ $\mathbf{0}_{\textit{residue-ring}\ (p\hat{}((\textit{nat}\ (\textit{padic-val}\ p\ f) + 1)))}$
f (*nat* (*padic-val p f*)) = $\mathbf{0}_{\textit{residue-ring}\ (p\hat{}((\textit{nat}\ (\textit{padic-val}\ p\ f))))}$
f (*nat* (*padic-val p f*) + *1*) ≠ *0*
f (*nat* (*padic-val p f*)) = *0*
⟨*proof*⟩

If $x \mod p^{n+1} \neq 0$, then $n \geq valx$.

lemma *below-val-zero*:
assumes *prime p*
assumes *x* ∈ (*padic-set p*)
assumes *x* (*Suc n*) ≠ $\mathbf{0}_{\textit{residue-ring}\ (p\hat{}(\textit{Suc}\ n))}$
shows *int n* ≥ (*padic-val p x*)
⟨*proof*⟩

If $n < valx$ then $x \mod p^n = 0$:

lemma *zero-below-val*:
assumes *prime p*
assumes *x* ∈ *padic-set p*
assumes *n* ≤ *padic-val p x*
shows *x n* = $\mathbf{0}_{\textit{residue-ring}\ (p\hat{}n)}$
x n = *0*
⟨*proof*⟩

Zero is the only element with valuation equal to −1:

lemma *val-zero*:
assumes *P*: *f* ∈ (*padic-set p*)
shows *padic-val p f* = −*1* ⟷ (*f* = (*padic-zero p*))
⟨*proof*⟩

The valuation turns multiplication into integer addition on nonzero elements. Note that this i the first instance where we need to explicity use the fact that p is a prime.

lemma *val-prod*:
assumes *prime p*
assumes *f* ∈ (*padic-set p*)
assumes *g* ∈ (*padic-set p*)
assumes *f* ≠ *padic-zero p*
assumes *g* ≠ *padic-zero p*

shows *padic-val p* (*padic-mult p f g*) = *padic-val p f* + *padic-val p g*
⟨*proof*⟩

19 Defining the Ring of p-adic Integers:

definition *padic-int* :: *int* ⇒ *padic-int ring*
where *padic-int p* ≡ ⦇*carrier* = (*padic-set p*),
Group.monoid.mult = (*padic-mult p*), *one* = (*padic-one p*),
zero = (*padic-zero p*), *add* = (*padic-add p*)⦈

lemma *padic-int-simps*:
$\mathbf{1}_{\text{padic-int } p}$ = *padic-one p*
$\mathbf{0}_{\text{padic-int } p}$ = *padic-zero p*
$(\oplus_{\text{padic-int } p})$ = *padic-add p*
$(\otimes_{\text{padic-int } p})$ = *padic-mult p*
carrier (*padic-int p*) = *padic-set p*
⟨*proof*⟩

lemma *residues-n*:
assumes *n* ≠ *0*
assumes *prime p*
shows *residues* (*p*^*n*)
⟨*proof*⟩

p-adic multiplication is associative

lemma *padic-mult-assoc*:
assumes *prime p*
shows ⋀*x y z*.
x ∈ *carrier* (*padic-int p*) ⟹
y ∈ *carrier* (*padic-int p*) ⟹
z ∈ *carrier* (*padic-int p*) ⟹
$x \otimes_{\text{padic-int } p} y \otimes_{\text{padic-int } p} z = x \otimes_{\text{padic-int } p} (y \otimes_{\text{padic-int } p} z)$
⟨*proof*⟩

The p-adic integers are closed under addition:

lemma *padic-add-closed*:
assumes *prime p*
shows ⋀*x y*.
x ∈ *carrier* (*padic-int p*) ⟹
y ∈ *carrier* (*padic-int p*) ⟹
$x \oplus_{(\text{padic-int } p)} y \in$ *carrier* (*padic-int p*)
⟨*proof*⟩

p-adic addition is associative:

lemma *padic-add-assoc*:
assumes *prime p*
shows ⋀*x y z*.
x ∈ *carrier* (*padic-int p*) ⟹

$y \in$ *carrier* (*padic-int p*) $\Longrightarrow$ $z \in$ *carrier* (*padic-int p*)
$\Longrightarrow$ $x \oplus_{padic\text{-}int\ p} y \oplus_{padic\text{-}int\ p} z = x \oplus_{padic\text{-}int\ p} (y \oplus_{padic\text{-}int\ p} z)$
⟨*proof*⟩

p-adic addition is commutative:

lemma *padic-add-comm*:
assumes *prime p*
shows $\bigwedge x\ y.$
$x \in$ *carrier* (*padic-int p*) $\Longrightarrow$
$y \in$ *carrier* (*padic-int p*) $\Longrightarrow$
$x \oplus_{padic\text{-}int\ p} y = y \oplus_{padic\text{-}int\ p} x$
⟨*proof*⟩

padic_zero is an additive identity:

lemma *padic-add-zero*:
assumes *prime p*
shows $\bigwedge x.\ x \in$ *carrier* (*padic-int p*) $\Longrightarrow \mathbf{0}_{padic\text{-}int\ p} \oplus_{padic\text{-}int\ p} x = x$
⟨*proof*⟩

Closure under additive inverses:

lemma *padic-add-inv*:
assumes *prime p*
shows $\bigwedge x.\ x \in$ *carrier* (*padic-int p*) $\Longrightarrow$
$\exists\, y \in$ *carrier* (*padic-int p*). $y \oplus_{padic\text{-}int\ p} x = \mathbf{0}_{padic\text{-}int\ p}$
⟨*proof*⟩

The ring of padic integers forms an abelian group under addition:

lemma *padic-is-abelian-group*:
assumes *prime p*
shows *abelian-group* (*padic-int p*)
⟨*proof*⟩

One is a multiplicative identity:

lemma *padic-one-id*:
assumes *prime p*
assumes $x \in$ *carrier* (*padic-int p*)
shows $\mathbf{1}_{padic\text{-}int\ p} \otimes_{padic\text{-}int\ p} x = x$
⟨*proof*⟩

p-adic multiplication is commutative:

lemma *padic-mult-comm*:
assumes *prime p*
assumes $x \in$ *carrier* (*padic-int p*)
assumes $y \in$ *carrier* (*padic-int p*)
shows $x \otimes_{padic\text{-}int\ p} y = y \otimes_{padic\text{-}int\ p} x$
⟨*proof*⟩

lemma *padic-is-comm-monoid*:
assumes *prime p*
shows *Group.comm-monoid* (*padic-int p*)
⟨*proof*⟩

lemma *padic-int-is-cring*:
  assumes *prime p*
  shows *cring* (*padic-int p*)
⟨*proof*⟩

The p-adic ring has no nontrivial zero divisors. Note that this argument is short because we have proved that the valuation is multiplicative on nonzero elements, which is where the primality assumption is used.

lemma *padic-no-zero-divisors*:
assumes *prime p*
assumes *a* ∈ *carrier* (*padic-int p*)
assumes *b* ∈*carrier* (*padic-int p*)
assumes *a* ≠$\mathbf{0}_{padic\text{-}int\ p}$
assumes *b* ≠$\mathbf{0}_{padic\text{-}int\ p}$
shows *a* $\otimes_{padic\text{-}int\ p}$ *b* ≠ $\mathbf{0}_{padic\text{-}int\ p}$
⟨*proof*⟩

lemma *padic-int-is-domain*:
  assumes *prime p*
  shows *domain* (*padic-int p*)
⟨*proof*⟩

20 The Ultrametric Inequality:

lemma *padic-val-ultrametric*:
  assumes *prime p*
  assumes *a* ∈ *carrier* (*padic-int p*)
  assumes *b* ∈ *carrier* (*padic-int p*)
  assumes *a* ≠ $\mathbf{0}_{(padic\text{-}int\ p)}$
  assumes *b* ≠ $\mathbf{0}_{(padic\text{-}int\ p)}$
  assumes *a* $\oplus_{(padic\text{-}int\ p)}$ *b* ≠ $\mathbf{0}_{(padic\text{-}int\ p)}$
  shows *padic-val p* (*a* $\oplus_{(padic\text{-}int\ p)}$ *b*) ≥ *min* (*padic-val p a*) (*padic-val p b*)
⟨*proof*⟩

lemma *padic-a-inv*:
  assumes *prime p*
  assumes *a* ∈ *carrier* (*padic-int p*)
  shows $\ominus_{padic\text{-}int\ p}$ *a* = (λ *n*. $\ominus_{residue\text{-}ring\ (p\hat{}n)}$ (*a n*))
⟨*proof*⟩

lemma *padic-val-a-inv*:
  assumes *prime p*
  assumes *a* ∈ *carrier* (*padic-int p*)

shows *padic-val p a* = *padic-val p* ($\ominus_{padic\text{-}int\ p}\ a$)
⟨*proof*⟩

end
theory *Padic-Integers*
imports *Padic-Construction*
Extended-Int
Supplementary-Ring-Facts
HOL−Algebra.Subrings
HOL−Number-Theory.Residues
HOL−Algebra.Multiplicative-Group

begin

In what follows we establish a locale for reasoning about the ring of p-adic integers for a fixed prime p. We will elaborate on the basic metric properties of $\mathbb{Z}_p$ and construct the angular component maps to the residue rings.

21 A Locale for p-adic Integer Rings

locale *padic-integers* =
fixes *Zp*:: - *ring* (**structure**)
fixes *p*
defines *Zp* ≡ *padic-int p*
assumes *prime*: *prime p*

sublocale *padic-integers* < *UPZ?*: *UP Zp UP Zp*
⟨*proof*⟩

sublocale *padic-integers* < *Zp?*:*UP-cring Zp UP Zp*
⟨*proof*⟩

sublocale *padic-integers* < *Zp?*:*ring Zp*
⟨*proof*⟩

sublocale *padic-integers* < *Zp?*:*cring Zp*
⟨*proof*⟩

sublocale *padic-integers* < *Zp?*:*domain Zp*
⟨*proof*⟩

context *padic-integers*
begin

lemma *Zp-defs*:
1 = *padic-one p*
0 = *padic-zero p*

carrier Zp = padic-set p
(⊗) = padic-mult p
(⊕) = padic-add p
⟨*proof*⟩

end

22 Residue Rings

lemma(**in** *field*) *field-inv*:
assumes *a* ∈ *carrier R*
assumes *a* ≠**0**
shows $inv_R\ a \otimes a = \mathbf{1}$
$a \otimes inv_R\ a = \mathbf{1}$
$inv_R\ a \in carrier\ R$
⟨*proof*⟩

$p_residue$ defines the standard projection maps between residue rings:

definition(**in** *padic-integers*) *p-residue*:: *nat* ⇒ *int* ⇒ - **where**
p-residue m n ≡ *residue* (*p*^*m*) *n*

lemma(**in** *padic-integers*) *p-residue-alt-def*:
p-residue m n = *n mod* (*p*^*m*)
⟨*proof*⟩

lemma(**in** *padic-integers*) *p-residue-range*:
p-residue m n ∈ {*0*..<*p*^*m*}
⟨*proof*⟩

lemma(**in** *padic-integers*) *p-residue-mod*:
assumes *m* > *k*
shows *p-residue k* (*p-residue m n*) = *p-residue k n*
⟨*proof*⟩

Compatibility of p_residue with elements of $\mathbb{Z}_p$:

lemma(**in** *padic-integers*) *p-residue-padic-int*:
assumes *x* ∈ *carrier Zp*
assumes *m* ≥ *k*
shows *p-residue k* (*x m*) = *x k*
⟨*proof*⟩

Definition of residue rings:

abbreviation(**in** *padic-integers*) (*input*) *Zp-res-ring*:: *nat* ⇒ - *ring* **where**
(*Zp-res-ring n*) ≡ *residue-ring* (*p*^*n*)

lemma (**in** *padic-integers*) *p-res-ring-zero*:
$\mathbf{0}_{Zp\text{-}res\text{-}ring\ k} = 0$
⟨*proof*⟩

lemma (**in** *padic-integers*) *p-res-ring-one*:
assumes $k > 0$
shows $\mathbf{1}_{Zp\text{-}res\text{-}ring\ k} = 1$
⟨*proof*⟩

lemma (**in** *padic-integers*) *p-res-ring-car*:
carrier (*Zp-res-ring k*) = {*0*..<*p*^*k*}
⟨*proof*⟩

lemma(**in** *padic-integers*) *p-residue-range'*:
p-residue m n ∈ *carrier* (*Zp-res-ring m*)
⟨*proof*⟩

First residue ring is a field:

lemma(**in** *padic-integers*) *p-res-ring-1-field*:
field (*Zp-res-ring 1*)
⟨*proof*⟩

0^{th} residue ring is the zero ring:

lemma(**in** *padic-integers*) *p-res-ring-0*:
carrier (*Zp-res-ring 0*) = {*0*}
⟨*proof*⟩

lemma(**in** *padic-integers*) *p-res-ring-0'*:
assumes $x \in$ *carrier* (*Zp-res-ring 0*)
shows $x = 0$
⟨*proof*⟩

If $m > 0$ then Zp_res_ringm is an instance of the residues locale:

lemma(**in** *padic-integers*) *p-residues*:
assumes $m > 0$
shows *residues* (*p*^*m*)
⟨*proof*⟩

If $m > 0$ then Zp_res_ringm is a commutative ring:

lemma(**in** *padic-integers*) *R-cring*:
assumes $m > 0$
shows *cring* (*Zp-res-ring m*)
⟨*proof*⟩

lemma(**in** *padic-integers*) *R-comm-monoid*:
assumes $m > 0$
shows *comm-monoid* (*Zp-res-ring m*)
⟨*proof*⟩

lemma(**in** *padic-integers*) *zero-rep*:
0 = (λ*m*. (*p-residue m 0*))

⟨*proof*⟩

The operations on residue rings are just the standard operations on the integers mod p^n. This means that the basic closure properties and algebraic properties of operations on these rings hold for all integers, not just elements of the ring carrier:

lemma(**in** *padic-integers*) *residue-add*:
shows $(x \oplus_{Zp\text{-}res\text{-}ring\ k} y) = (x + y)\ mod\ p\hat{}k$
⟨*proof*⟩

lemma(**in** *padic-integers*) *residue-add-closed*:
shows $(x \oplus_{Zp\text{-}res\text{-}ring\ k} y) \in carrier\ (Zp\text{-}res\text{-}ring\ k)$
⟨*proof*⟩

lemma(**in** *padic-integers*) *residue-add-closed'*:
shows $(x \oplus_{Zp\text{-}res\text{-}ring\ k} y) \in \{0..<p\hat{}k\}$
⟨*proof*⟩

lemma(**in** *padic-integers*) *residue-mult*:
shows $(x \otimes_{Zp\text{-}res\text{-}ring\ k} y) = (x * y)\ mod\ p\hat{}k$
⟨*proof*⟩

lemma(**in** *padic-integers*) *residue-mult-closed*:
shows $(x \otimes_{Zp\text{-}res\text{-}ring\ k} y) \in carrier\ (Zp\text{-}res\text{-}ring\ k)$
⟨*proof*⟩

lemma(**in** *padic-integers*) *residue-mult-closed'*:
shows $(x \otimes_{Zp\text{-}res\text{-}ring\ k} y) \in \{0..<p\hat{}k\}$
⟨*proof*⟩

lemma(**in** *padic-integers*) *residue-add-assoc*:
shows $(x \oplus_{Zp\text{-}res\text{-}ring\ k} y) \oplus_{Zp\text{-}res\text{-}ring\ k} z = x \oplus_{Zp\text{-}res\text{-}ring\ k} (y \oplus_{Zp\text{-}res\text{-}ring\ k} z)$
⟨*proof*⟩

lemma(**in** *padic-integers*) *residue-mult-comm*:
shows $x \otimes_{Zp\text{-}res\text{-}ring\ k} y = y \otimes_{Zp\text{-}res\text{-}ring\ k} x$
⟨*proof*⟩

lemma(**in** *padic-integers*) *residue-mult-assoc*:
shows $(x \otimes_{Zp\text{-}res\text{-}ring\ k} y) \otimes_{Zp\text{-}res\text{-}ring\ k} z = x \otimes_{Zp\text{-}res\text{-}ring\ k} (y \otimes_{Zp\text{-}res\text{-}ring\ k} z)$
⟨*proof*⟩

lemma(**in** *padic-integers*) *residue-add-comm*:
shows $x \oplus_{Zp\text{-}res\text{-}ring\ k} y = y \oplus_{Zp\text{-}res\text{-}ring\ k} x$
⟨*proof*⟩

lemma(**in** *padic-integers*) *residue-minus-car*:
assumes $y \in$ *carrier* (*Zp-res-ring k*)
shows $(x \ominus_{\textit{Zp-res-ring } k} y) = (x - y) \bmod p\hat{}k$
⟨*proof*⟩

lemma(**in** *padic-integers*) *residue-a-inv*:
shows $\ominus_{\textit{Zp-res-ring } k} y = \ominus_{\textit{Zp-res-ring } k} (y \bmod p\hat{}k)$
⟨*proof*⟩

lemma(**in** *padic-integers*) *residue-a-inv-closed*:
$\ominus_{\textit{Zp-res-ring } k} y \in$ *carrier* (*Zp-res-ring k*)
⟨*proof*⟩

lemma(**in** *padic-integers*) *residue-minus*:
$(x \ominus_{\textit{Zp-res-ring } k} y) = (x - y) \bmod p\hat{}k$
⟨*proof*⟩

lemma(**in** *padic-integers*) *residue-minus-closed*:
$(x \ominus_{\textit{Zp-res-ring } k} y) \in$ *carrier* (*Zp-res-ring k*)
⟨*proof*⟩

lemma (**in** *padic-integers*) *residue-plus-zero-r*:
$0 \oplus_{\textit{Zp-res-ring } k} y = y \bmod p\hat{}k$
⟨*proof*⟩

lemma (**in** *padic-integers*) *residue-plus-zero-l*:
$y \oplus_{\textit{Zp-res-ring } k} 0 = y \bmod p\hat{}k$
⟨*proof*⟩

lemma (**in** *padic-integers*) *residue-times-zero-r*:
$0 \otimes_{\textit{Zp-res-ring } k} y = 0$
⟨*proof*⟩

lemma (**in** *padic-integers*) *residue-times-zero-l*:
$y \otimes_{\textit{Zp-res-ring } k} 0 = 0$
⟨*proof*⟩

lemma (**in** *padic-integers*) *residue-times-one-r*:
$1 \otimes_{\textit{Zp-res-ring } k} y = y \bmod p\hat{}k$
⟨*proof*⟩

lemma (**in** *padic-integers*) *residue-times-one-l*:
$y \otimes_{\textit{Zp-res-ring } k} 1 = y \bmod p\hat{}k$
⟨*proof*⟩

Similarly to the previous lemmas, many identities about taking residues of p-adic integers hold even for elements which lie outside the carrier of $\mathbb{Z}_p$:

lemma (**in** *padic-integers*) *residue-of-sum*:

$(a \oplus b)\ k = (a\ k) \oplus_{Zp\text{-}res\text{-}ring\ k} (b\ k)$
⟨*proof*⟩

lemma (**in** *padic-integers*) *residue-of-sum′*:
$(a \oplus b)\ k = ((a\ k) + (b\ k))\ mod\ p\hat{}k$
⟨*proof*⟩

lemma (**in** *padic-integers*) *residue-closed*[*simp*]:
assumes $b \in carrier\ Zp$
shows $b\ k \in carrier\ (Zp\text{-}res\text{-}ring\ k)$
⟨*proof*⟩

lemma (**in** *padic-integers*) *residue-of-diff*:
assumes $b \in carrier\ Zp$
shows $(a \ominus b)\ k = (a\ k) \ominus_{Zp\text{-}res\text{-}ring\ k} (b\ k)$
⟨*proof*⟩

lemma (**in** *padic-integers*) *residue-of-prod*:
$(a \otimes b)\ k = (a\ k) \otimes_{Zp\text{-}res\text{-}ring\ k} (b\ k)$
⟨*proof*⟩

lemma (**in** *padic-integers*) *residue-of-prod′*:
$(a \otimes b)\ k = ((a\ k) * (b\ k))\ mod\ (p\hat{}k)$
⟨*proof*⟩

lemma (**in** *padic-integers*) *residue-of-one*:
assumes $k > 0$
shows $\mathbf{1}\ k = \mathbf{1}_{Zp\text{-}res\text{-}ring\ k}$
$\mathbf{1}\ k = 1$
⟨*proof*⟩

lemma (**in** *padic-integers*) *residue-of-zero*:
shows $\mathbf{0}\ k = \mathbf{0}_{Zp\text{-}res\text{-}ring\ k}$
$\mathbf{0}\ k = 0$
⟨*proof*⟩

lemma(**in** *padic-integers*) *Zp-residue-mult-zero*:
assumes $a\ k = 0$
shows $(a \otimes b)\ k = 0\ (b \otimes a)\ k = 0$
⟨*proof*⟩

lemma(**in** *padic-integers*) *Zp-residue-add-zero*:
assumes $b \in carrier\ Zp$
assumes $(a::\ padic\text{-}int)\ k = 0$
shows $(a \oplus b)\ k = b\ k\ (b \oplus a)\ k = b\ k$
⟨*proof*⟩

P-adic addition and multiplication are globally additive and associative:

lemma *padic-add-assoc0*:

assumes *prime p*
shows *padic-add p* (*padic-add p x y*) *z* = *padic-add p x* (*padic-add p y z*)
⟨*proof*⟩

lemma(**in** *padic-integers*) *add-assoc*:
$a \oplus b \oplus c = a \oplus (b \oplus c)$
⟨*proof*⟩

lemma *padic-add-comm0*:
assumes *prime p*
shows (*padic-add p x y*)= (*padic-add p y x*)
⟨*proof*⟩

lemma(**in** *padic-integers*) *add-comm*:
$a \oplus b = b \oplus a$
⟨*proof*⟩

lemma *padic-mult-assoc0*:
assumes *prime p*
shows *padic-mult p* (*padic-mult p x y*) *z* = *padic-mult p x* (*padic-mult p y z*)
⟨*proof*⟩

lemma(**in** *padic-integers*) *mult-assoc*:
$a \otimes b \otimes c = a \otimes (b \otimes c)$
⟨*proof*⟩

lemma *padic-mult-comm0*:
assumes *prime p*
shows (*padic-mult p x y*)= (*padic-mult p y x*)
⟨*proof*⟩

lemma(**in** *padic-integers*) *mult-comm*:
$a \otimes b = b \otimes a$
⟨*proof*⟩

lemma(**in** *padic-integers*) *mult-zero-l*:
$a \otimes \mathbf{0} = \mathbf{0}$
⟨*proof*⟩

lemma(**in** *padic-integers*) *mult-zero-r*:
$\mathbf{0} \otimes a = \mathbf{0}$
⟨*proof*⟩

lemma (**in** *padic-integers*) *p-residue-ring-car-memI*:
assumes (*m*::*int*) ≥*0*
assumes $m < p\hat{}k$
shows *m* ∈ *carrier* (*Zp-res-ring k*)
⟨*proof*⟩

lemma (**in** *padic-integers*) *p-residue-ring-car-memE*:
assumes $m \in$ *carrier* (*Zp-res-ring k*)
shows $m < p\hat{}k$ $m \geq 0$
⟨*proof*⟩

lemma (**in** *padic-integers*) *residues-closed*:
assumes $a \in$ *carrier Zp*
shows $a\ k \in$ *carrier* (*Zp-res-ring k*)
⟨*proof*⟩

lemma (**in** *padic-integers*) *mod-in-carrier*:
a *mod* $(p\hat{}n) \in$ *carrier* (*Zp-res-ring n*)
⟨*proof*⟩

lemma (**in** *padic-integers*) *Zp-residue-a-inv*:
assumes $a \in$ *carrier Zp*
shows $(\ominus\ a)\ k = \ominus_{Zp\text{-}res\text{-}ring\ k}\ (a\ k)$
$(\ominus\ a)\ k = (-\ (a\ k))$ *mod* $(p\hat{}k)$
⟨*proof*⟩

lemma (**in** *padic-integers*) *residue-of-diff'*:
assumes $b \in$ *carrier Zp*
shows $(a \ominus b)\ k = ((a\ k) - (b\ k))$ *mod* $(p\hat{}k)$
⟨*proof*⟩

lemma (**in** *padic-integers*) *residue-UnitsI*:
assumes $n \geq 1$
assumes $(k::int) > 0$
assumes $k < p\hat{}n$
assumes *coprime k p*
shows $k \in$ *Units* (*Zp-res-ring n*)
⟨*proof*⟩

lemma (**in** *padic-integers*) *residue-UnitsE*:
assumes $n \geq 1$
assumes $k \in$ *Units* (*Zp-res-ring n*)
shows *coprime k p*
⟨*proof*⟩

lemma(**in** *padic-integers*) *residue-units-nilpotent*:
assumes $m > 0$
assumes $k =$ *card* (*Units* (*Zp-res-ring m*))
assumes $x \in$ (*Units* (*Zp-res-ring m*))
shows $x[\hat{}]_{Zp\text{-}res\text{-}ring\ m}\ k = 1$
⟨*proof*⟩

lemma (**in** *padic-integers*) *residue-1-unit*:
assumes $m > 0$
shows $1 \in$ *Units* (*Zp-res-ring m*)

$\mathbf{1}_{Zp\text{-}res\text{-}ring\ m} \in Units\ (Zp\text{-}res\text{-}ring\ m)$
⟨*proof*⟩

lemma (**in** *padic-integers*) *zero-not-in-residue-units*:
assumes $n \geq 1$
shows $(0::int) \notin Units\ (Zp\text{-}res\text{-}ring\ n)$
⟨*proof*⟩

Cardinality bounds on the units of residue rings:

lemma (**in** *padic-integers*) *residue-units-card-geq-2*:
assumes $n \geq 2$
shows $card\ (Units\ (Zp\text{-}res\text{-}ring\ n)) \geq 2$
⟨*proof*⟩

lemma (**in** *padic-integers*) *residue-ring-card*:
$finite\ (carrier\ (Zp\text{-}res\text{-}ring\ n)) \wedge card\ (carrier\ (Zp\text{-}res\text{-}ring\ n)) = nat\ (p\hat{}n)$
⟨*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** *padic-integers*) *residue-of-Units*:
assumes $k > 0$
assumes $a \in Units\ Zp$
shows $a\ k \in Units\ (Zp\text{-}res\text{-}ring\ k)$
⟨*proof*⟩

23 *int* and *nat* inclusions in $\mathbb{Z}_p$.

lemma(**in** *ring*) *int-inc-zero*:
$[(0::int)]\cdot \mathbf{1} = \mathbf{0}$
⟨*proof*⟩

lemma(**in** *ring*) *int-inc-zero'*:
assumes $x \in carrier\ R$
shows $[(0::int)] \cdot x = \mathbf{0}$
⟨*proof*⟩

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

lemma(**in** *ring*) *nat-mult-zero*:
$[(0::nat)]\cdot x = \mathbf{0}$
⟨*proof*⟩

lemma(**in** *ring*) *nat-inc-closed*:
fixes $n::nat$
shows $[n] \cdot \mathbf{1} \in carrier\ R$
⟨*proof*⟩

lemma(**in** *ring*) *nat-mult-closed*:
fixes $n::nat$
assumes $x \in carrier\ R$
shows $[n] \cdot x \in carrier\ R$
⟨*proof*⟩

lemma(**in** *ring*) *int-inc-closed*:
fixes $n::int$
shows $[n] \cdot \mathbf{1} \in carrier\ R$
⟨*proof*⟩

lemma(**in** *ring*) *int-mult-closed*:
fixes $n::int$
assumes $x \in carrier\ R$
shows $[n] \cdot x \in carrier\ R$
⟨*proof*⟩

lemma(**in** *ring*) *nat-inc-prod*:
fixes $n::nat$
fixes $m::nat$
shows $[m]\cdot([n] \cdot \mathbf{1}) = [(m*n)]\cdot\mathbf{1}$
⟨*proof*⟩

lemma(**in** *ring*) *nat-inc-prod′*:
fixes $n::nat$
fixes $m::nat$
shows $[(m*n)]\cdot\mathbf{1} = [m]\cdot \mathbf{1} \otimes ([n] \cdot \mathbf{1})$
⟨*proof*⟩

lemma(**in** *padic-integers*) *Zp-nat-inc-zero*:
shows $[(0::nat)] \cdot x = \mathbf{0}$
⟨*proof*⟩

lemma(**in** *padic-integers*) *Zp-int-inc-zero′*:
shows $[(0::int)] \cdot x = \mathbf{0}$
⟨*proof*⟩

lemma(**in** *padic-integers*) *Zp-nat-inc-closed*:
fixes *n*::*nat*
shows $[n] \cdot \mathbf{1} \in$ *carrier Zp*
⟨*proof*⟩

lemma(**in** *padic-integers*) *Zp-nat-mult-closed*:
fixes *n*::*nat*
assumes $x \in$ *carrier Zp*
shows $[n] \cdot x \in$ *carrier Zp*
⟨*proof*⟩

lemma(**in** *padic-integers*) *Zp-int-inc-closed*:
fixes *n*::*int*
shows $[n] \cdot \mathbf{1} \in$ *carrier Zp*
⟨*proof*⟩

lemma(**in** *padic-integers*) *Zp-int-mult-closed*:
fixes *n*::*int*
assumes $x \in$ *carrier Zp*
shows $[n] \cdot x \in$ *carrier Zp*
⟨*proof*⟩

The following lemmas give a concrete description of the inclusion of integers and natural numbers into $\mathbb{Z}_p$:

lemma(**in** *padic-integers*) *Zp-nat-inc-rep*:
fixes *n*::*nat*
shows $[n] \cdot \mathbf{1} = (\lambda\ m.\ p\text{-}residue\ m\ n)$
⟨*proof*⟩

lemma(**in** *padic-integers*) *Zp-nat-inc-res*:
fixes *n*::*nat*
shows $([n] \cdot \mathbf{1})\ k = n\ mod\ (p\hat{}k)$
⟨*proof*⟩

lemma(**in** *padic-integers*) *Zp-int-inc-rep*:
fixes *n*::*int*
shows $[n] \cdot \mathbf{1} = (\lambda\ m.\ p\text{-}residue\ m\ n\ )$
⟨*proof*⟩

lemma(**in** *padic-integers*) *Zp-int-inc-res*:
fixes *n*::*int*
shows $([n] \cdot \mathbf{1})\ k = n\ mod\ (p\hat{}k)$
⟨*proof*⟩

abbreviation(**in** *padic-integers*)(*input*) p **where**
p ≡ $[p] \cdot \mathbf{1}$

lemma(**in** *padic-integers*) *p-natpow-prod*:

p[^](*n*::*nat*) ⊗ p[^](*m*::*nat*) = p[^](*n* + *m*)
⟨*proof*⟩

lemma(**in** *padic-integers*) *p-natintpow-prod*:
assumes (*m*::*int*) ≥ *0*
shows p[^](*n*::*nat*) ⊗ p[^]*m* = p[^](*n* + *m*)
⟨*proof*⟩

lemma(**in** *padic-integers*) *p-intnatpow-prod*:
assumes (*n*::*int*) ≥ *0*
shows p[^]*n* ⊗ p[^](*m*::*nat*) = p[^](*m* + *n*)
⟨*proof*⟩

lemma(**in** *padic-integers*) *p-int-pow-prod*:
assumes (*n*::*int*) ≥ *0*
assumes (*m*::*int*) ≥ *0*
shows p[^]*n* ⊗ p[^]*m* = p[^](*m* + *n*)
⟨*proof*⟩

lemma(**in** *padic-integers*) *p-natpow-prod-Suc*:
p ⊗ p[^](*m*::*nat*) = p[^](*Suc m*)
p[^](*m*::*nat*) ⊗ p = p[^](*Suc m*)
⟨*proof*⟩

lemma(**in** *padic-integers*) *power-residue*:
assumes *a* ∈ *carrier Zp*
assumes *k* > *0*
shows (*a*[^]$_{Zp}$ (*n*::*nat*)) *k* = (*a k*)^*n mod* (*p*^*k*)
⟨*proof*⟩

24 The Valuation on $\mathbb{Z}_p$

24.1 The Integer-Valued and Extended Integer-Valued Valuations

fun *fromeint* :: *eint* ⇒ *int* **where**
fromeint (*eint x*) = *x*

The extended-integer-valued *p*-adic valuation on $\mathbb{Z}_p$:

definition(**in** *padic-integers*) *val-Zp* **where**
val-Zp = (λ*x*. (*if* (*x* = **0**) *then* (∞::*eint*) *else* (*eint* (*padic-val p x*))))

We also define an integer-valued valuation on the nonzero elements of $\mathbb{Z}_p$, for simplified reasoning

definition(**in** *padic-integers*) *ord-Zp* **where**
ord-Zp = *padic-val p*

Ord of additive inverse

lemma(**in** *padic-integers*) *ord-Zp-of-a-inv*:
assumes $a \in$ *nonzero Zp*
shows *ord-Zp a* = *ord-Zp* ($\ominus a$)
⟨*proof*⟩

lemma(**in** *padic-integers*) *val-Zp-of-a-inv*:
assumes $a \in$ *carrier Zp*
shows *val-Zp a* = *val-Zp* ($\ominus a$)
⟨*proof*⟩

Ord-based criterion for being nonzero:

lemma(**in** *padic-integers*) *ord-of-nonzero*:
assumes $x \in$*carrier Zp*
assumes *ord-Zp x* $\geq$*0*
shows $x \neq \mathbf{0}$
$x \in$ *nonzero Zp*
⟨*proof*⟩

lemma(**in** *padic-integers*) *not-nonzero-Zp*:
assumes $x \in$ *carrier Zp*
assumes $x \notin$ *nonzero Zp*
shows $x = \mathbf{0}$
⟨*proof*⟩

lemma(**in** *padic-integers*) *not-nonzero-Qp*:
assumes $x \in$ *carrier* Q_p
assumes $x \notin$ *nonzero* Q_p
shows $x = \mathbf{0}_{Q_p}$
⟨*proof*⟩

Relationship between val and ord

lemma(**in** *padic-integers*) *val-ord-Zp*:
assumes $a \neq \mathbf{0}$
shows *val-Zp a* = *eint* (*ord-Zp a*)
⟨*proof*⟩

lemma(**in** *padic-integers*) *ord-pos*:
assumes $x \in$ *carrier Zp*
assumes $x \neq \mathbf{0}$
shows *ord-Zp x* $\geq$ *0*
⟨*proof*⟩

lemma(**in** *padic-integers*) *val-pos*:
assumes $x \in$ *carrier Zp*
shows *val-Zp x* $\geq$ *0*
⟨*proof*⟩

For passing between nat and int castings of ord

lemma(**in** *padic-integers*) *ord-nat*:

assumes $x \in$ *carrier Zp*
assumes $x \neq \mathbf{0}$
shows *int* (*nat* (*ord-Zp x*)) = *ord-Zp x*
⟨*proof*⟩

lemma(**in** *padic-integers*) *zero-below-ord*:
assumes $x \in$ *carrier Zp*
assumes $n \leq$ *ord-Zp x*
shows *x n* = *0*
⟨*proof*⟩

lemma(**in** *padic-integers*) *zero-below-val-Zp*:
assumes $x \in$ *carrier Zp*
assumes $n \leq$ *val-Zp x*
shows *x n* = *0*
⟨*proof*⟩

lemma(**in** *padic-integers*) *below-ord-zero*:
assumes $x \in$ *carrier Zp*
assumes *x* (*Suc n*) ≠ *0*
shows $n \geq$ *ord-Zp x*
⟨*proof*⟩

lemma(**in** *padic-integers*) *below-val-Zp-zero*:
assumes $x \in$ *carrier Zp*
assumes *x* (*Suc n*) ≠ *0*
shows $n \geq$ *val-Zp x*
⟨*proof*⟩

lemma(**in** *padic-integers*) *nonzero-imp-ex-nonzero-res*:
assumes $x \in$ *carrier Zp*
assumes $x \neq \mathbf{0}$
shows ∃ *k*. *x* (*Suc k*) ≠ *0*
⟨*proof*⟩

lemma(**in** *padic-integers*) *ord-suc-nonzero*:
assumes $x \in$ *carrier Zp*
assumes $x \neq \mathbf{0}$
assumes *ord-Zp x* = *n*
shows *x* (*Suc n*) ≠ *0*
⟨*proof*⟩

lemma(**in** *padic-integers*) *above-ord-nonzero*:
assumes $x \in$ *carrier Zp*
assumes $x \neq \mathbf{0}$
assumes *n* > *ord-Zp x*
shows *x n* ≠ *0*
⟨*proof*⟩

lemma(**in** *padic-integers*) *ord-Zp-geq*:
assumes $x \in$ *carrier Zp*
assumes $x\ n = 0$
assumes $x \neq \mathbf{0}$
shows *ord-Zp* $x \geq n$
⟨*proof*⟩

lemma(**in** *padic-integers*) *ord-equals*:
assumes $x \in$ *carrier Zp*
assumes $x\ (Suc\ n) \neq 0$
assumes $x\ n = 0$
shows *ord-Zp* $x = n$
⟨*proof*⟩

lemma(**in** *padic-integers*) *ord-Zp-p*:
ord-Zp p = (*1*::*int*)
⟨*proof*⟩

lemma(**in** *padic-integers*) *ord-Zp-one*:
ord-Zp $\mathbf{1} = 0$
⟨*proof*⟩

ord is multiplicative on nonzero elements of Zp

lemma(**in** *padic-integers*) *ord-Zp-mult*:
assumes $x \in$ *nonzero Zp*
assumes $y \in$ *nonzero Zp*
shows (*ord-Zp* ($x \otimes_{Zp} y$)) = (*ord-Zp* x) + (*ord-Zp* y)
⟨*proof*⟩

lemma(**in** *padic-integers*) *ord-Zp-pow*:
assumes $x \in$ *nonzero Zp*
shows *ord-Zp* ($x[\hat{}](n::nat)$) = $n*$(*ord-Zp* x)
⟨*proof*⟩

lemma(**in** *padic-integers*) *val-Zp-pow*:
assumes $x \in$ *nonzero Zp*
shows *val-Zp* ($x[\hat{}](n::nat)$) = ($n*$(*ord-Zp* x))
⟨*proof*⟩

lemma(**in** *padic-integers*) *val-Zp-pow'*:
assumes $x \in$ *nonzero Zp*
shows *val-Zp* ($x[\hat{}](n::nat)$) = $n*$(*val-Zp* x)
⟨*proof*⟩

lemma(**in** *padic-integers*) *ord-Zp-p-pow*:
ord-Zp (p[^](*n*::*nat*)) = *n*
⟨*proof*⟩

lemma(**in** *padic-integers*) *ord-Zp-p-int-pow*:

assumes $n \geq 0$
shows *ord-Zp* (p[^](*n*::*int*)) = *n*
⟨*proof*⟩

lemma(**in** *padic-integers*) *val-Zp-p*:
(*val-Zp* p) = *1*
⟨*proof*⟩

lemma(**in** *padic-integers*) *val-Zp-p-pow*:
val-Zp (p[^](*n*::*nat*)) = *eint n*
⟨*proof*⟩

lemma(**in** *padic-integers*) *p-pow-res*:
assumes (*n*::*nat*) ≥ *m*
shows (p[^]*n*) *m* = *0*
⟨*proof*⟩

lemma(**in** *padic-integers*) *p-pow-factor*:
assumes (*n*::*nat*) ≥ *m*
shows (*h* ⊗ (p[^]*n*)) *m* = *0* (*h* ⊗ (p[^]*n*)) *m* = $\mathbf{0}_{Zp\text{-}res\text{-}ring\ n}$
⟨*proof*⟩

24.2 The Ultrametric Inequality

Ultrametric inequality for ord

lemma(**in** *padic-integers*) *ord-Zp-ultrametric*:
assumes *x* ∈ *nonzero Zp*
assumes *y* ∈ *nonzero Zp*
assumes *x* ⊕ *y* ∈ *nonzero Zp*
shows *ord-Zp* (*x* ⊕ *y*) ≥ *min* (*ord-Zp x*) (*ord-Zp y*)
⟨*proof*⟩

Variants of the ultrametric inequality

lemma (**in** *padic-integers*) *ord-Zp-ultrametric-diff*:
assumes *x* ∈ *nonzero Zp*
assumes *y* ∈ *nonzero Zp*
assumes *x* ≠ *y*
shows *ord-Zp* (*x* ⊖ *y*) ≥ *min* (*ord-Zp x*) (*ord-Zp y*)
⟨*proof*⟩

lemma(**in** *padic-integers*) *ord-Zp-not-equal-imp-notequal*:
assumes *x* ∈ *nonzero Zp*
assumes *y* ∈ *nonzero Zp*
assumes *ord-Zp x* ≠ (*ord-Zp y*)
shows *x* ≠ *y* *x* ⊖ *y* ≠**0** *x* ⊕ *y* ≠**0**
⟨*proof*⟩

lemma(**in** *padic-integers*) *ord-Zp-ultrametric-eq*:

assumes $x \in$ *nonzero Zp*
assumes $y \in$ *nonzero Zp*
assumes *ord-Zp* $x >$ (*ord-Zp* y)
shows *ord-Zp* $(x \oplus y) =$ *ord-Zp* y
⟨*proof*⟩

lemma(**in** *padic-integers*) *ord-Zp-ultrametric-eq′*:
assumes $x \in$ *nonzero Zp*
assumes $y \in$ *nonzero Zp*
assumes *ord-Zp* $x >$ (*ord-Zp* y)
shows *ord-Zp* $(x \ominus y) =$ *ord-Zp* y
⟨*proof*⟩

lemma(**in** *padic-integers*) *ord-Zp-ultrametric-eq″*:
assumes $x \in$ *nonzero Zp*
assumes $y \in$ *nonzero Zp*
assumes *ord-Zp* $x >$ (*ord-Zp* y)
shows *ord-Zp* $(y \ominus x) =$ *ord-Zp* y
⟨*proof*⟩

lemma(**in** *padic-integers*) *ord-Zp-not-equal-ord-plus-minus*:
assumes $x \in$ *nonzero Zp*
assumes $y \in$ *nonzero Zp*
assumes *ord-Zp* $x \neq$ (*ord-Zp* y)
shows *ord-Zp* $(x \ominus y) =$ *ord-Zp* $(x \oplus y)$
⟨*proof*⟩

val is multiplicative on nonzero elements

lemma(**in** *padic-integers*) *val-Zp-mult0*:
assumes $x \in$ *carrier Zp*
assumes $x \neq \mathbf{0}$
assumes $y \in$ *carrier Zp*
assumes $y \neq \mathbf{0}$
shows (*val-Zp* $(x \otimes_{Zp} y)) =$ (*val-Zp* x) + (*val-Zp* y)
⟨*proof*⟩

val is multiplicative everywhere

lemma(**in** *padic-integers*) *val-Zp-mult*:
assumes $x \in$ *carrier Zp*
assumes $y \in$ *carrier Zp*
shows (*val-Zp* $(x \otimes_{Zp} y)) =$ (*val-Zp* x) + (*val-Zp* y)
⟨*proof*⟩

lemma(**in** *padic-integers*) *val-Zp-ultrametric0*:
assumes $x \in$ *carrier Zp*
assumes $x \neq \mathbf{0}$
assumes $y \in$ *carrier Zp*
assumes $y \neq \mathbf{0}$
assumes $x \oplus y \neq \mathbf{0}$

shows *min* (*val-Zp x*) (*val-Zp y*) ≤ *val-Zp* (*x* ⊕ *y*)
⟨*proof*⟩

Unconstrained ultrametric inequality

lemma(**in** *padic-integers*) *val-Zp-ultrametric*:
assumes *x* ∈ *carrier Zp*
assumes *y* ∈ *carrier Zp*
shows *min* (*val-Zp x*) (*val-Zp y*) ≤ *val-Zp* (*x* ⊕ *y*)
⟨*proof*⟩

Variants of the ultrametric inequality

lemma (**in** *padic-integers*) *val-Zp-ultrametric-diff*:
assumes *x* ∈ *carrier Zp*
assumes *y* ∈ *carrier Zp*
shows *val-Zp* (*x* ⊖ *y*) ≥ *min* (*val-Zp x*) (*val-Zp y*)
⟨*proof*⟩

lemma(**in** *padic-integers*) *val-Zp-not-equal-imp-notequal*:
assumes *x* ∈ *carrier Zp*
assumes *y* ∈ *carrier Zp*
assumes *val-Zp x* ≠ *val-Zp y*
shows *x* ≠ *y x* ⊖ *y* ≠**0** *x* ⊕ *y* ≠**0**
⟨*proof*⟩

lemma(**in** *padic-integers*) *val-Zp-ultrametric-eq*:
assumes *x* ∈ *carrier Zp*
assumes *y* ∈ *carrier Zp*
assumes *val-Zp x* > *val-Zp y*
shows *val-Zp* (*x* ⊕ *y*) = *val-Zp y*
⟨*proof*⟩

lemma(**in** *padic-integers*) *val-Zp-ultrametric-eq′*:
assumes *x* ∈ *carrier Zp*
assumes *y* ∈ *carrier Zp*
assumes *val-Zp x* > (*val-Zp y*)
shows *val-Zp* (*x* ⊖ *y*) = *val-Zp y*
⟨*proof*⟩

lemma(**in** *padic-integers*) *val-Zp-ultrametric-eq″*:
assumes *x* ∈ *carrier Zp*
assumes *y* ∈ *carrier Zp*
assumes *val-Zp x* > (*val-Zp y*)
shows *val-Zp* (*y* ⊖ *x*) = *val-Zp y*
⟨*proof*⟩

lemma(**in** *padic-integers*) *val-Zp-not-equal-ord-plus-minus*:
assumes *x* ∈ *carrier Zp*
assumes *y* ∈ *carrier Zp*
assumes *val-Zp x* ≠ (*val-Zp y*)

shows *val-Zp* $(x \ominus y) =$ *val-Zp* $(x \oplus y)$
⟨*proof*⟩

24.3 Units of $\mathbb{Z}_p$

Elements with valuation 0 in Zp are the units

lemma(**in** *padic-integers*) *val-Zp-0-criterion*:
assumes $x \in$ *carrier Zp*
assumes $x\ 1 \neq 0$
shows *val-Zp* $x = 0$
⟨*proof*⟩

Units in Zp have val 0

lemma(**in** *padic-integers*) *unit-imp-val-Zp0*:
assumes $x \in$ *Units Zp*
shows *val-Zp* $x = 0$
⟨*proof*⟩

Elements in Zp with ord 0 are units

lemma(**in** *padic-integers*) *val-Zp0-imp-unit0*:
assumes *val-Zp* $x = 0$
assumes $x \in$ *carrier Zp*
fixes n::*nat*
shows $(x\ (Suc\ n)) \in$ *Units* (*Zp-res-ring* (*Suc n*))
⟨*proof*⟩

lemma(**in** *padic-integers*) *val-Zp0-imp-unit0'*:
assumes *val-Zp* $x = 0$
assumes $x \in$ *carrier Zp*
assumes $(n$::*nat*$) > 0$
shows $(x\ n) \in$ *Units* (*Zp-res-ring n*)
⟨*proof*⟩

lemma(**in** *cring*) *ring-hom-Units-inv*:
assumes $a \in$ *Units R*
assumes *cring S*
assumes $h \in$ *ring-hom R S*
shows $h\ (inv\ a) = inv_S\ h\ a\ h\ a \in$ *Units S*
⟨*proof*⟩

lemma(**in** *padic-integers*) *val-Zp-0-imp-unit*:
assumes *val-Zp* $x = 0$
assumes $x \in$ *carrier Zp*
shows $x \in$ *Units Zp*
⟨*proof*⟩

Definition of ord on a fraction is independent of the choice of representatives

lemma(**in** *padic-integers*) *ord-Zp-eq-frac*:

assumes $a \in$ *nonzero Zp*
assumes $b \in$ *nonzero Zp*
assumes $c \in$ *nonzero Zp*
assumes $d \in$ *nonzero Zp*
assumes $a \otimes d = b \otimes c$
shows (*ord-Zp a*) − (*ord-Zp b*) = (*ord-Zp c*) − (*ord-Zp d*)
⟨*proof*⟩

lemma(**in** *padic-integers*) *val-Zp-eq-frac-0*:
assumes $a \in$ *nonzero Zp*
assumes $b \in$ *nonzero Zp*
assumes $c \in$ *nonzero Zp*
assumes $d \in$ *nonzero Zp*
assumes $a \otimes d = b \otimes c$
shows (*val-Zp a*) − (*val-Zp b*) = (*val-Zp c*) − (*val-Zp d*)
⟨*proof*⟩

25 Angular Component Maps on $\mathbb{Z}_p$

The angular component map on $\mathbb{Z}_p$ is just the map which normalizes a point $x \in \mathbb{Z}_p$ by mapping it to a point with valuation 0. It is explicitly defined as the mapping $x \mapsto p^{-ord(p)} * x$ for nonzero x, and $0 \mapsto 0$. By composing these maps with reductions mod p^n we get maps which are equal to the standard residue maps on units of $\mathbb{Z}_p$, but in general unequal elsewhere. Both the angular component map and the angular component map mod p^n are homomorpshims from the multiplicative group of units of $\mathbb{Z}_p$ to the multiplicative group of units of the residue rings, and play a key role in first-order model-theoretic formalizations of the p-adics (see, for example [5], or [2]).

lemma(**in** *cring*) *int-nat-pow-rep*:
[(*k*::*int*)]·**1** [^] (*n*::*nat*) = [(*k*^*n*)]·**1**
⟨*proof*⟩

lemma(**in** *padic-integers*) *p-pow-rep0*:
fixes *n*::*nat*
shows p[^]*n* = [(*p*^*n*)]·**1**
⟨*proof*⟩

lemma(**in** *padic-integers*) *p-pow-nonzero*:
shows (p[^](*n*::*nat*)) ∈ *carrier Zp*
(p[^](*n*::*nat*)) ≠ **0**
⟨*proof*⟩

lemma(**in** *padic-integers*) *p-pow-nonzero′*:
shows (p[^](*n*::*nat*)) ∈ *nonzero Zp*
⟨*proof*⟩

lemma(**in** *padic-integers*) *p-pow-rep*:
fixes *n*::*nat*
shows (p[^]*n*) *k* = (*p*^*n*) *mod* (*p*^*k*)
⟨*proof*⟩

lemma(**in** *padic-integers*) *p-pow-car*:
assumes (*n*::*int*)≥ *0*
shows (p[^]*n*) ∈ *carrier Zp*
⟨*proof*⟩

lemma(**in** *padic-integers*) *p-int-pow-nonzero*:
assumes (*n*::*int*) ≥*0*
shows (p[^]*n*) ∈ *nonzero Zp*
⟨*proof*⟩

lemma(**in** *padic-integers*) *p-nonzero*:
shows p ∈ *nonzero Zp*
⟨*proof*⟩

Every element of Zp is a unit times a power of p.

lemma(**in** *padic-integers*) *residue-factor-unique*:
assumes *k>0*
assumes *x* ∈ *carrier Zp*
assumes *u* ∈ *carrier* (*Zp-res-ring k*) ∧ (*u*∗ *p*^*m*) = (*x* (*m*+*k*))
shows *u* = (*THE u*. *u* ∈ *carrier* (*Zp-res-ring k*) ∧ (*u*∗ *p*^*m*) = (*x* (*m*+*k*)))
⟨*proof*⟩

lemma(**in** *padic-integers*) *residue-factor-exists*:
assumes *m* = *nat* (*ord-Zp x*)
assumes *k* > *0*
assumes *x* ∈ *carrier Zp*
assumes *x* ≠**0**
obtains *u* **where** *u* ∈ *carrier* (*Zp-res-ring k*) ∧ (*u*∗ *p*^*m*) = (*x* (*m*+*k*))
⟨*proof*⟩

definition(**in** *padic-integers*) *normalizer* **where**
normalizer m x = (λ*k*. *if* (*k*=*0*) *then 0 else* (*THE u*. *u* ∈ *carrier* (*Zp-res-ring k*)
∧ (*u*∗ *p*^*m*) = (*x* (*m* + *k*))))

definition(**in** *padic-integers*) *ac-Zp* **where**
ac-Zp x = *normalizer* (*nat* (*ord-Zp x*)) *x*

lemma(**in** *padic-integers*) *ac-Zp-equation*:
assumes *x* ∈ *carrier Zp*
assumes *x* ≠**0**
assumes *k* > *0*
assumes *m* = *nat* (*ord-Zp x*)
shows (*ac-Zp x k*) ∈ *carrier* (*Zp-res-ring k*) ∧ (*ac-Zp x k*)∗(*p*^*m*) = (*x* (*m*+*k*))
⟨*proof*⟩

lemma(**in** *padic-integers*) *ac-Zp-res*:
assumes $m > k$
assumes $x \in$ *carrier Zp*
assumes $x \neq \mathbf{0}$
shows *p-residue k* (*ac-Zp x m*) = (*ac-Zp x k*)
⟨*proof*⟩

lemma(**in** *padic-integers*) *ac-Zp-in-Zp*:
assumes $x \in$ *carrier Zp*
assumes $x \neq \mathbf{0}$
shows *ac-Zp x* $\in$ *carrier Zp*
⟨*proof*⟩

lemma(**in** *padic-integers*) *ac-Zp-is-Unit*:
assumes $x \in$ *carrier Zp*
assumes $x \neq \mathbf{0}$
shows *ac-Zp x* $\in$ *Units Zp*
⟨*proof*⟩

The typical defining equation for the angular component map.

lemma(**in** *padic-integers*) *ac-Zp-factors-x*:
assumes $x \in$ *carrier Zp*
assumes $x \neq \mathbf{0}$
shows $x = (\mathrm{p}[\hat{}](nat\ (ord\text{-}Zp\ x))) \otimes (ac\text{-}Zp\ x)\ x = (\mathrm{p}[\hat{}](ord\text{-}Zp\ x)) \otimes (ac\text{-}Zp\ x)$
⟨*proof*⟩

lemma(**in** *padic-integers*) *ac-Zp-factors'*:
assumes $x \in$ *nonzero Zp*
shows $x = [p] \cdot \mathbf{1}\ [\hat{}]\ ord\text{-}Zp\ x \otimes ac\text{-}Zp\ x$
⟨*proof*⟩

lemma(**in** *padic-integers*) *ac-Zp-mult*:
assumes $x \in$ *nonzero Zp*
assumes $y \in$ *nonzero Zp*
shows *ac-Zp* ($x \otimes y$) = (*ac-Zp x*) $\otimes$ (*ac-Zp y*)
⟨*proof*⟩

lemma(**in** *padic-integers*) *ac-Zp-one*:
ac-Zp $\mathbf{1} = \mathbf{1}$
⟨*proof*⟩

lemma(**in** *padic-integers*) *ac-Zp-inv*:
assumes $x \in$ *Units Zp*
shows *ac-Zp* ($inv_{Zp}\ x$) = inv_{Zp} (*ac-Zp x*)
⟨*proof*⟩

lemma(**in** *padic-integers*) *ac-Zp-of-Unit*:
assumes *val-Zp x* = *0*

assumes $x \in$ *carrier Zp*
shows *ac-Zp* $x = x$
⟨*proof*⟩

lemma(**in** *padic-integers*) *ac-Zp-p*:
(*ac-Zp* p) = **1**
⟨*proof*⟩

lemma(**in** *padic-integers*) *ac-Zp-p-nat-pow*:
(*ac-Zp* (p [^] (*n*::*nat*))) = **1**
⟨*proof*⟩

Facts for reasoning about integer powers in an arbitrary commutative monoid:

lemma(**in** *monoid*) *int-pow-add*:
fixes *n*::*int*
fixes *m*::*int*
assumes $a \in$ *Units G*
shows a [^] $(n + m) = (a$ [^] $n) \otimes (a$ [^] $m)$
⟨*proof*⟩

lemma(**in** *monoid*) *int-pow-unit-closed*:
fixes *n*::*int*
assumes $a \in$ *Units G*
shows a[^] $n \in$ *Units G*
⟨*proof*⟩

lemma(**in** *monoid*) *nat-pow-of-inv*:
fixes *n*::*nat*
assumes $a \in$ *Units G*
shows *inv* a[^] $n =$ *inv* $(a$[^] $n)$
⟨*proof*⟩

lemma(**in** *monoid*) *int-pow-of-inv*:
fixes *n*::*int*
assumes $a \in$ *Units G*
shows *inv* a[^] $n =$ *inv* $(a$[^] $n)$
⟨*proof*⟩

lemma(**in** *monoid*) *int-pow-inv*:
fixes *n*::*int*
assumes $a \in$ *Units G*
shows a[^] $-n =$ *inv* a[^] n
⟨*proof*⟩

lemma(**in** *monoid*) *int-pow-inv'*:
fixes *n*::*int*
assumes $a \in$ *Units G*
shows a[^] $-n =$ *inv* $(a$[^] $n)$
⟨*proof*⟩

lemma(**in** *comm-monoid*) *inv-of-prod*:
assumes $a \in$ *Units G*
assumes $b \in$ *Units G*
shows *inv* $(a \otimes\ b) = (inv\ \ a) \otimes (inv\ \ b)$
⟨*proof*⟩

26 Behaviour of *val_Zp* and *ord_Zp* on Natural Numbers and Integers

If f and g have an equal residue at k, then they differ by a multiple of p^k.

lemma(**in** *padic-integers*) *eq-residue-mod*:
assumes $f \in$ *carrier Zp*
assumes $g \in$ *carrier Zp*
assumes $f\ k = g\ k$
shows $\exists h.\ h \in$ *carrier Zp* $\wedge\ f = g \oplus (\mathrm{p}[\hat{}]k) \otimes h$
⟨*proof*⟩

lemma(**in** *padic-integers*) *eq-residue-mod'*:
assumes $f \in$ *carrier Zp*
assumes $g \in$ *carrier Zp*
assumes $f\ k = g\ k$
obtains h **where** $h \in$ *carrier Zp* $\wedge\ f = g \oplus (\mathrm{p}[\hat{}]k) \otimes h$
⟨*proof*⟩

Valuations of integers which do not divide p:

lemma(**in** *padic-integers*) *ord-Zp-p-nat-unit*:
assumes $(n{::}nat)$ *mod p* $\neq$ *0*
shows *ord-Zp* $([n] \cdot \mathbf{1}) =$ *0*
⟨*proof*⟩

lemma(**in** *padic-integers*) *val-Zp-p-nat-unit*:
assumes $(n{::}nat)$ *mod p* $\neq$ *0*
shows *val-Zp* $([n] \cdot \mathbf{1}) =$ *0*
⟨*proof*⟩

lemma(**in** *padic-integers*) *nat-unit*:
assumes $(n{::}nat)$ *mod p* $\neq$ *0*
shows $([n] \cdot \mathbf{1}) \in$ *Units Zp*
⟨*proof*⟩

lemma(**in** *padic-integers*) *ord-Zp-p-int-unit*:
assumes $(n{::}int)$ *mod p* $\neq$ *0*
shows *ord-Zp* $([n] \cdot \mathbf{1}) =$ *0*
⟨*proof*⟩

lemma(**in** *padic-integers*) *val-Zp-p-int-unit*:

assumes (*n*::*int*) *mod p* ≠ *0*
shows *val-Zp* ([*n*]·**1**) = *0*
⟨*proof*⟩

lemma(**in** *padic-integers*) *int-unit*:
assumes (*n*::*int*) *mod p* ≠ *0*
shows ([*n*]·**1**) ∈ *Units Zp*
⟨*proof*⟩

lemma(**in** *padic-integers*) *int-decomp-ord*:
assumes *n* = *l*∗(*p*^*k*)
assumes *l mod p* ≠ *0*
shows *ord-Zp* ([*n*]·**1**) = *k*
⟨*proof*⟩

lemma(**in** *padic-integers*) *int-decomp-val*:
assumes *n* = *l*∗(*p*^*k*)
assumes *l mod p* ≠ *0*
shows *val-Zp* ([*n*]·**1**) = *k*
⟨*proof*⟩

$\mathbb{Z}_p$ has characteristic zero:

lemma(**in** *padic-integers*) *Zp-char-0*:
assumes (*n*::*int*) > *0*
shows [*n*]·**1** ≠ **0**
⟨*proof*⟩

lemma(**in** *padic-integers*) *Zp-char-0′*:
assumes (*n*::*nat*) > *0*
shows [*n*]·**1** ≠ **0**
⟨*proof*⟩

lemma (**in** *domain*) *not-eq-diff-nonzero*:
assumes *a* ≠ *b*
assumes *a* ∈ *carrier R*
assumes *b* ∈*carrier R*
shows *a* ⊖ *b* ∈ *nonzero R*
⟨*proof*⟩

lemma (**in** *domain*) *minus-a-inv*:
assumes *a* ∈ *carrier R*
assumes *b* ∈ *carrier R*
shows *a* ⊖ *b* = ⊖ (*b* ⊖ *a*)
⟨*proof*⟩

lemma(**in** *ring*) *plus-diff-simp*:
assumes *a* ∈ *carrier R*
assumes *b* ∈ *carrier R*
assumes *c* ∈ *carrier R*

assumes $X = a \ominus b$
assumes $Y = c \ominus a$
shows $X \oplus Y = c \ominus b$
⟨*proof*⟩

lemma (**in** *padic-integers*) *Zp-residue-eq*:
assumes $a \in$ *carrier Zp*
assumes $b \in$ *carrier Zp*
assumes *val-Zp* $(a \ominus b) > k$
shows $(a\ k) = (b\ k)$
⟨*proof*⟩

lemma (**in** *padic-integers*) *Zp-residue-eq2*:
assumes $a \in$ *carrier Zp*
assumes $b \in$ *carrier Zp*
assumes $(a\ k) = (b\ k)$
assumes $a \neq b$
shows *val-Zp* $(a \ominus b) \geq k$
⟨*proof*⟩

lemma (**in** *padic-integers*) *equal-val-Zp*:
assumes $a \in$ *carrier Zp*
assumes $b \in$ *carrier Zp*
assumes $c \in$ *carrier Zp*
assumes *val-Zp* $a =$ *val-Zp* b
assumes *val-Zp* $(c \ominus a) >$ *val-Zp* b
shows *val-Zp* $c =$ *val-Zp* b
⟨*proof*⟩

lemma (**in** *padic-integers*) *equal-val-Zp'*:
assumes $a \in$ *carrier Zp*
assumes $b \in$ *carrier Zp*
assumes $c \in$ *carrier Zp*
assumes *val-Zp* $a =$ *val-Zp* b
assumes *val-Zp* $c >$ *val-Zp* b
shows *val-Zp* $(a \oplus c) =$ *val-Zp* b
⟨*proof*⟩

lemma (**in** *padic-integers*) *val-Zp-of-minus*:
assumes $a \in$ *carrier Zp*
shows *val-Zp* $a =$ *val-Zp* $(\ominus a)$
⟨*proof*⟩

end
theory *Padic-Int-Topology*
imports *Padic-Integers Function-Ring*
begin

type-synonym *padic-int-seq* $=$ *nat* $\Rightarrow$ *padic-int*

type-synonym *padic-int-fun* = *padic-int* ⇒ *padic-int*

sublocale *padic-integers* < *FunZp?*: *U-function-ring Zp*
⟨*proof*⟩

context *padic-integers*
begin

27 Sequences over Zp

The p-adic valuation can be thought of as equivalent to the p-adic absolute value, but with the notion of size inverted so that small numbers have large valuation, and zero has maximally large valuation. The p-adic distance between two points is just the valuation of the difference of those points, and is thus equivalent to the metric induced by the p-adic absolute value. For background on valuations and absolute values for p-adic rings see [4]. In what follows, we develop the topology of the p-adic from a valuative perspective rather than a metric perspective. Though equivalent to the metric approach in the p-adic case, this approach is more general in that there exist valued rings whose valuations take values in non-Archimedean ordered ablelian groups which do not embed into the real numbers.

27.1 The Valuative Distance Function on $\mathbb{Z}_p$

The following lemmas establish that the p-adic distance function satifies the standard properties of an ultrametric. It is symmetric, obeys the ultrametric inequality, and only identical elements are infinitely close.

definition *val-Zp-dist* :: *padic-int* ⇒ *padic-int* ⇒ *eint* **where**
val-Zp-dist a b ≡ *val-Zp* (*a* ⊖ *b*)

lemma *val-Zp-dist-sym*:
assumes *a* ∈ *carrier Zp*
assumes *b* ∈ *carrier Zp*
shows *val-Zp-dist a b* = *val-Zp-dist b a*
⟨*proof*⟩

lemma *val-Zp-dist-ultrametric*:
assumes *a* ∈ *carrier Zp*
assumes *b* ∈ *carrier Zp*
assumes *c* ∈ *carrier Zp*
shows *val-Zp-dist b c* ≥ *min* (*val-Zp-dist a c*) (*val-Zp-dist a b*)
⟨*proof*⟩

lemma *val-Zp-dist-infty*:
assumes *a* ∈ *carrier Zp*

assumes $b \in$ *carrier Zp*
assumes *val-Zp-dist a b* $= \infty$
shows $a = b$
⟨*proof*⟩

lemma *val-Zp-dist-infty'*:
assumes $a \in$ *carrier Zp*
assumes $b \in$ *carrier Zp*
assumes $a = b$
shows *val-Zp-dist a b* $= \infty$
⟨*proof*⟩

The following property will be useful in the proof of Hensel's Lemma: two p-adic integers are close together if and only if their residues are equal at high orders.

lemma *val-Zp-dist-res-eq*:
assumes $a \in$ *carrier Zp*
assumes $b \in$ *carrier Zp*
assumes *val-Zp-dist a b* $> k$
shows $(a\ k) = (b\ k)$
⟨*proof*⟩

lemma *val-Zp-dist-res-eq2*:
assumes $a \in$ *carrier Zp*
assumes $b \in$ *carrier Zp*
assumes $(a\ k) = (b\ k)$
shows *val-Zp-dist a b* $\geq k$
⟨*proof*⟩

lemma *val-Zp-dist-triangle-eqs*:
assumes $a \in$ *carrier Zp*
assumes $b \in$ *carrier Zp*
assumes $c \in$ *carrier Zp*
assumes *val-Zp-dist a b* $> n$
assumes *val-Zp-dist a c* $> n$
assumes $(k::nat) < n$
shows $a\ k = b\ k$
$a\ k = c\ k$
$b\ k = c\ k$
⟨*proof*⟩

27.2 Cauchy Sequences

The definition of Cauchy sequence here is equivalent to standard the metric notion, and is identical to the one found on page 50 of [4].

lemma *closed-seqs-diff-closed*:
assumes $s \in$ *closed-seqs Zp*
assumes $a \in$ *carrier Zp*

shows $s\ m \ominus a \in$ *carrier Zp*
⟨*proof*⟩

definition *is-Zp-cauchy* :: *padic-int-seq* ⇒ *bool* **where**
is-Zp-cauchy s = (($s \in$ *closed-seqs Zp*) ∧ (∀ (*n*::*int*). ∃ (*N*::*nat*). ∀ *m k*::*nat*.

$$(m{>}N \wedge k{>}N \longrightarrow (\textit{val-Zp-dist}\ (s\ m)\ (s\ k)) > \textit{eint}\ n)))$$

Relation for a sequence which converges to a point:

definition *Zp-converges-to* :: *padic-int-seq* ⇒ *padic-int* ⇒ *bool* **where**
Zp-converges-to s a = (($a \in$ *carrier Zp* ∧ $s \in$ *closed-seqs Zp*)
∧ (∀ (*n*::*int*). (∃ (*k*:: *nat*). (∀ (*m*::*nat*).
($m > k \longrightarrow$ (*val-Zp* ((*s m*) ⊖ *a*)) > *eint n*)))))

lemma *is-Zp-cauchy-imp-closed*:
assumes *is-Zp-cauchy s*
shows $s \in$ *closed-seqs Zp*
⟨*proof*⟩

Analogous to the lemmas about residues and p-adic distances, we can characterize Cauchy sequences without reference to a distance function: a sequence is Cauchy if and only if for every natural number k, the k^{th} residues of the elements in the sequence are eventually all equal.

lemma *is-Zp-cauchy-imp-res-eventually-const-0*:
assumes *is-Zp-cauchy s*
fixes *n*::*nat*
obtains *N* **where** ⋀ *n0 n1*. *n0* > *N* ∧ *n1* > *N* ⟹ (*s n0*) *n* = (*s n1*) *n*
⟨*proof*⟩

lemma *is-Zp-cauchyI*:
assumes $s \in$ *closed-seqs Zp*
assumes ⋀ *n*. (∃ *N*. (∀ *n0 n1*. *n0* > *N* ∧ *n1* > *N* ⟶ (*s n0*) *n* = (*s n1*) *n*))
shows *is-Zp-cauchy s*
⟨*proof*⟩

lemma *is-Zp-cauchy-imp-res-eventually-const*:
assumes *is-Zp-cauchy s*
fixes *n*::*nat*
obtains *N r* **where** *r* ∈ *carrier* (*Zp-res-ring n*) **and** ⋀ *m*. *m* > *N* ⟹ (*s m*) *n* = *r*
⟨*proof*⟩

This function identifies the eventual residues of the elements of a cauchy sequence. Since a p-adic integer is defined to be the map which identifies its residues, this map will turn out to be precisely the limit of a cauchy sequence.

definition *res-lim* :: *padic-int-seq* ⇒ *padic-int* **where**
res-lim s = (λ *k*. (*THE r*. (∃ *N*. (∀ *m*. *m* > *N* ⟶ (*s m*) *k* = *r*))))

lemma *res-lim-Zp-cauchy-0*:
assumes *is-Zp-cauchy s*
assumes $f = (\textit{res-lim}\ s)\ k$
shows $(\exists N.\ (\forall\ m.\ (m > N \longrightarrow (s\ m)\ k = f)))$
$f \in \textit{carrier}\ (\textit{Zp-res-ring}\ k)$
⟨*proof*⟩

lemma *res-lim-Zp-cauchy*:
assumes *is-Zp-cauchy s*
obtains N **where** $\bigwedge m.\ (m > N \longrightarrow (s\ m)\ k = (\textit{res-lim}\ s)\ k)$
⟨*proof*⟩

lemma *res-lim-in-Zp*:
assumes *is-Zp-cauchy s*
shows *res-lim s* ∈ *carrier Zp*
⟨*proof*⟩

27.3 Completeness of $\mathbb{Z}_p$

We can use the developments above to show that a sequence of p-adic integers is convergent if and only if it is cauchy, and that limits of convergent sequences are always unique.

lemma *is-Zp-cauchy-imp-has-limit*:
assumes *is-Zp-cauchy s*
assumes *a* = *res-lim s*
shows *Zp-converges-to s a*
⟨*proof*⟩

lemma *convergent-imp-Zp-cauchy*:
assumes *s* ∈ *closed-seqs Zp*
assumes *a* ∈ *carrier Zp*
assumes *Zp-converges-to s a*
shows *is-Zp-cauchy s*
⟨*proof*⟩

lemma *unique-limit*:
assumes *s* ∈ *closed-seqs Zp*
assumes *a* ∈ *carrier Zp*
assumes *b* ∈ *carrier Zp*
assumes *Zp-converges-to s a*
assumes *Zp-converges-to s b*
shows *a* = *b*
⟨*proof*⟩

lemma *unique-limit′*:
assumes *s* ∈ *closed-seqs Zp*
assumes *a* ∈ *carrier Zp*
assumes *Zp-converges-to s a*

shows $a = \textit{res-lim}\ s$
⟨*proof*⟩

lemma *Zp-converges-toE*:
assumes $s \in \textit{closed-seqs}\ Zp$
assumes $a \in \textit{carrier}\ Zp$
assumes $\textit{Zp-converges-to}\ s\ a$
shows $\exists N.\ \forall k > N.\ s\ k\ n = a\ n$
⟨*proof*⟩

lemma *Zp-converges-toI*:
assumes $s \in \textit{closed-seqs}\ Zp$
assumes $a \in \textit{carrier}\ Zp$
assumes $\bigwedge n.\ \exists N.\ \forall k > N.\ s\ k\ n = a\ n$
shows $\textit{Zp-converges-to}\ s\ a$
⟨*proof*⟩

Sums and products of cauchy sequences are cauchy:

lemma *sum-of-Zp-cauchy-is-Zp-cauchy*:
assumes $\textit{is-Zp-cauchy}\ s$
assumes $\textit{is-Zp-cauchy}\ t$
shows $\textit{is-Zp-cauchy}\ (s \oplus_{Zp^\omega}\ t)$
⟨*proof*⟩

lemma *prod-of-Zp-cauchy-is-Zp-cauchy*:
assumes $\textit{is-Zp-cauchy}\ s$
assumes $\textit{is-Zp-cauchy}\ t$
shows $\textit{is-Zp-cauchy}\ (s \otimes_{Zp^\omega}\ t)$
⟨*proof*⟩

Constant sequences are cauchy:

lemma *constant-is-Zp-cauchy*:
assumes $\textit{is-constant-seq}\ Zp\ s$
shows $\textit{is-Zp-cauchy}\ s$
⟨*proof*⟩

Scalar multiplies of cauchy sequences are cauchy:

lemma *smult-is-Zp-cauchy*:
assumes $\textit{is-Zp-cauchy}\ s$
assumes $a \in \textit{carrier}\ Zp$
shows $\textit{is-Zp-cauchy}\ (a \odot_{Zp^\omega}\ s)$
⟨*proof*⟩

lemma *Zp-cauchy-imp-approaches-res-lim*:
assumes $\textit{is-Zp-cauchy}\ s$
assumes $a = \textit{res-lim}\ s$
obtains N **where** $\bigwedge n.\ n > N \Longrightarrow \textit{val-Zp}\ (a \ominus (s\ n)) > \textit{eint}\ k$
⟨*proof*⟩

28 Continuous Functions

For convenience, we will use a slightly unorthodox definition of continuity here. Since $\mathbb{Z}_p$ is complete, a function is continuous if and only if its compositions with cauchy sequences are cauchy sequences. Thus we can define a continuous function on $\mathbb{Z}_p$ as a function which carries cauchy sequences to cauchy sequences under composition. Note that this does not generalize to a definition of continuity for functions defined on incomplete subsets of $\mathbb{Z}_p$. For example, the function $1/x$ defined on $\mathbb{Z}_p - \{0\}$ clearly does not have this property but is continuous. However, towards a proof of Hensel's Lemma we will only need to consider polynomial functions and so this definition suffices for our purposes.

28.1 Defining Continuous Functions and Basic Examples

abbreviation *Zp-constant-function* ($\mathfrak{c}_{Zp}$) **where**
$\mathfrak{c}_{Zp}\ a \equiv$ *constant-function* (*carrier Zp*) *a*

definition *is-Zp-continuous* ::*padic-int-fun* ⇒ *bool* **where**
is-Zp-continuous f = (*f* ∈ *carrier* (*Fun Zp*) ∧(∀ *s*. *is-Zp-cauchy s* ⟶ *is-Zp-cauchy*(*f* ∘ *s*)))

lemma *Zp-continuous-is-Zp-closed*:
assumes *is-Zp-continuous f*
shows *f* ∈ *carrier* (*Fun Zp*)
⟨*proof*⟩

lemma *is-Zp-continuousI*:
assumes *f* ∈ *carrier* (*Fun Zp*)
assumes ⋀*s*. *is-Zp-cauchy s* ⟹ *is-Zp-cauchy* (*f* ∘ *s*)
shows *is-Zp-continuous f*
⟨*proof*⟩

lemma *Zp-continuous-is-closed*:
assumes *is-Zp-continuous f*
shows *f* ∈ *carrier* (*Fun Zp*)
⟨*proof*⟩

lemma *constant-is-Zp-continuous*:
assumes *a* ∈ *carrier Zp*
shows *is-Zp-continuous* (*const a*)
⟨*proof*⟩

lemma *sum-of-cont-is-cont*:
assumes *is-Zp-continuous f*
assumes *is-Zp-continuous g*
shows *is-Zp-continuous* (*f* ⊕$_{Fun\ Zp}$ *g*)
⟨*proof*⟩

lemma *prod-of-cont-is-cont*:
assumes *is-Zp-continuous f*
assumes *is-Zp-continuous g*
shows *is-Zp-continuous* ($f \otimes_{Fun\ Zp} g$)
⟨*proof*⟩

lemma *smult-is-continuous*:
assumes *is-Zp-continuous f*
assumes $a \in carrier\ Zp$
shows *is-Zp-continuous* ($a \odot_{Fun\ Zp} f$)
⟨*proof*⟩

lemma *id-Zp-is-Zp-continuous*:
is-Zp-continuous ring-id
⟨*proof*⟩

lemma *nat-pow-is-Zp-continuous*:
assumes *is-Zp-continuous f*
shows *is-Zp-continuous* ($f[\widehat{}]_{Fun\ Zp}(n::nat)$)
⟨*proof*⟩

lemma *ring-id-pow-closed*:
$(ring\text{-}id)[\widehat{}]_{Fun\ Zp}\ (n::nat) \in carrier\ (Fun\ Zp)$
⟨*proof*⟩

lemma *monomial-equation*:
assumes $c \in carrier\ Zp$
shows $monomial\ c\ n = c \odot_{Fun\ Zp} (ring\text{-}id)[\widehat{}]_{Fun\ Zp} n$
⟨*proof*⟩

lemma *monomial-is-Zp-continuous*:
assumes $c \in carrier\ Zp$
shows *is-Zp-continuous* (*monomial c n*)
⟨*proof*⟩

28.2 Composition by a Continuous Function Commutes with Taking Limits of Sequences

Due to our choice of definition for continuity, a little bit of care is required to show that taking the limit of a cauchy sequence commutes with composition by a continuous function. For a sequence $(s_n)_{n\in\mathbb{N}}$ converging to a point t, we can consider the alternating sequence $(s_0, t, s_1, t, s_2, t, \dots)$ which is also cauchy. Clearly composition with f yields $(f(s_0), f(t), f(s_1), f(t), f(s_2), f(t), \dots)$ from which we can see that the limit must be $f(t)$.

definition *alt-seq* **where**
alt-seq s = (λ*k*. (*if* (*even k*) *then* (*s k*) *else* (*res-lim s*)))

lemma *alt-seq-Zp-cauchy*:
assumes *is-Zp-cauchy s*
shows *is-Zp-cauchy* (*alt-seq s*)
⟨*proof*⟩

lemma *alt-seq-limit*:
assumes *is-Zp-cauchy s*
shows *res-lim*(*alt-seq s*) = *res-lim s*
⟨*proof*⟩

lemma *res-lim-pushforward*:
assumes *is-Zp-continuous f*
assumes *is-Zp-cauchy s*
assumes *t* = *alt-seq s*
shows *res-lim* (*f* ∘ *t*) = *f* (*res-lim t*)
⟨*proof*⟩

lemma *res-lim-pushforward′*:
assumes *is-Zp-continuous f*
assumes *is-Zp-cauchy s*
assumes *t* = *alt-seq s*
shows *res-lim* (*f* ∘ *s*) = *res-lim* (*f* ∘ *t*)
⟨*proof*⟩

lemma *continuous-limit*:
assumes *is-Zp-continuous f*
assumes *is-Zp-cauchy s*
shows *Zp-converges-to* (*f* ∘ *s*) (*f* (*res-lim s*))
⟨*proof*⟩

end
end
theory *Padic-Int-Polynomials*
imports *Padic-Int-Topology Cring-Poly*
begin

context *padic-integers*
begin

This theory states and proves basic lemmas connecting the topology on $\mathbb{Z}_p$ with the functions induced by polynomial evaluation over $\mathbb{Z}_p$. This will imply that polynomial evaluation applied to a Cauchy Sequence will always produce a cauchy sequence, which is a key fact in the proof of Hensel's Lemma.

type-synonym *padic-int-poly* = *nat* ⇒ *padic-int*

lemma *monom-term-car*:
assumes *c* ∈ *carrier Zp*

assumes $x \in$ *carrier Zp*
shows $c \otimes x[\widehat{}](n::nat) \in$ *carrier Zp*
⟨*proof*⟩

Univariate polynomial ring over Zp

abbreviation(*input*) *Zp-x* **where**
Zp-x ≡ *UP Zp*

lemma *Zp-x-is-UP-cring*:
UP-cring Zp
⟨*proof*⟩

lemma *Zp-x-is-UP-domain*:
UP-domain Zp
⟨*proof*⟩

lemma *Zp-x-domain*:
domain Zp-x
⟨*proof*⟩

lemma *pow-closed*:
assumes $a \in$ *carrier Zp*
shows $a[\widehat{}](n::nat) \in$ *carrier Zp*
⟨*proof*⟩

lemma(**in** *ring*) *pow-zero*:
assumes $(n::nat)>0$
shows $\mathbf{0}[\widehat{}]\ n = \mathbf{0}$
⟨*proof*⟩

lemma *sum-closed*:
assumes $f \in$ *carrier Zp*
assumes $g \in$ *carrier Zp*
shows $f \oplus g \in$ *carrier Zp*
⟨*proof*⟩

lemma *mult-zero*:
assumes $f \in$ *carrier Zp*
shows $f \otimes \mathbf{0} = \mathbf{0}$
$\mathbf{0} \otimes f = \mathbf{0}$
⟨*proof*⟩

lemma *monom-poly-is-Zp-continuous*:
assumes $c \in$ *carrier Zp*
assumes $f =$ *monom Zp-x c n*
shows *is-Zp-continuous* (*to-fun f*)
⟨*proof*⟩

lemma *degree-0-is-Zp-continuous*:

assumes $f \in$ *carrier Zp-x*
assumes *degree* $f = 0$
shows *is-Zp-continuous* (*to-fun* f)
⟨*proof*⟩

lemma *UP-sum-is-Zp-continuous*:
assumes $a \in$ *carrier Zp-x*
assumes $b \in$ *carrier Zp-x*
assumes *is-Zp-continuous* (*to-fun* a)
assumes *is-Zp-continuous* (*to-fun* b)
shows *is-Zp-continuous* (*to-fun* ($a \oplus_{Zp\text{-}x} b$))
⟨*proof*⟩

lemma *polynomial-is-Zp-continuous*:
assumes $f \in$ *carrier Zp-x*
shows *is-Zp-continuous* (*to-fun* f)
⟨*proof*⟩
end

Notation for polynomial function application

context *padic-integers*
begin
notation *to-fun* (**infixl**‹•› *70*)

Evaluating polynomials in the residue rings

lemma *res-to-fun-monic-monom*:
assumes $a \in$ *carrier Zp*
assumes $b \in$ *carrier Zp*
assumes $a\ k = b\ k$
shows (*monom Zp-x* **1** $n \cdot a$) $k =$ (*monom Zp-x* **1** $n \cdot b$) k
⟨*proof*⟩

lemma *res-to-fun-monom*:
assumes $a \in$ *carrier Zp*
assumes $b \in$ *carrier Zp*
assumes $c \in$ *carrier Zp*
assumes $a\ k = b\ k$
shows (*monom Zp-x* $c\ n \cdot a$) $k =$ (*monom Zp-x* $c\ n \cdot b$) k
⟨*proof*⟩

lemma *to-fun-res-ltrm*:
assumes $a \in$ *carrier Zp*
assumes $b \in$ *carrier Zp*
assumes $f \in$ *carrier Zp-x*
assumes $a\ k = b\ k$
shows ((*ltrm* f)•a) $k =$ ((*ltrm* f)•b) k
⟨*proof*⟩

Polynomial application commutes with taking residues

lemma *to-fun-res*:
assumes $f \in$ *carrier Zp-x*
assumes $a \in$ *carrier Zp*
assumes $b \in$ *carrier Zp*
assumes $a\ k = b\ k$
shows $(f \cdot a)\ k = (f \cdot b)\ k$
⟨*proof*⟩

If a and b have equal kth residues, then so do f'(a) and f'(b)

lemma *deriv-res*:
assumes $f \in$ *carrier Zp-x*
assumes $a \in$ *carrier Zp*
assumes $b \in$ *carrier Zp*
assumes $a\ k = b\ k$
shows $(deriv\ f\ a)\ k = (deriv\ f\ b)\ k$
⟨*proof*⟩

Propositions about evaluation:

lemma *to-fun-monom-plus*:
assumes $a \in$ *carrier Zp*
assumes $g \in$ *carrier Zp-x*
assumes $c \in$ *carrier Zp*
shows $(monom\ Zp\text{-}x\ a\ n \oplus_{Zp\text{-}x} g) \cdot c = a \otimes c[\hat{}]n \oplus (g \cdot c)$
⟨*proof*⟩

lemma *to-fun-monom-minus*:
assumes $a \in$ *carrier Zp*
assumes $g \in$ *carrier Zp-x*
assumes $c \in$ *carrier Zp*
shows $(monom\ Zp\text{-}x\ a\ n \ominus_{Zp\text{-}x} g) \cdot c = a \otimes c[\hat{}]n \ominus (g \cdot c)$
⟨*proof*⟩

end

end
theory *Hensels-Lemma*
imports *Padic-Int-Polynomials*
begin

The following proof of Hensel's Lemma is directly adapted from Keith Conrad's proof which is given in an online note [1]. The same note was used as the basis for a formalization of Hensel's Lemma by Robert Lewis in the Lean proof assistant [**?**].

29 Auxiliary Lemmas for Hensel's Lemma

lemma(**in** *ring*) *minus-sum*:
assumes $a \in$ *carrier R*

assumes $b \in$ *carrier* R
shows $\ominus\ (a \oplus b) = \ominus\ a \oplus \ominus\ b$
⟨*proof*⟩

context *padic-integers*
begin

lemma *poly-diff-val*:
assumes $f \in$ *carrier Zp-x*
assumes $a \in$ *carrier Zp*
assumes $b \in$ *carrier Zp*
shows *val-Zp* $(f \bullet a \ominus f \bullet b) \geq$ *val-Zp* $(a \ominus b)$
⟨*proof*⟩

Restricted p-adic division

definition *divide* **where**
divide $x\ y =$ (*if* $x = \mathbf{0}$ *then* $\mathbf{0}$ *else*
(p[ˆ](*nat* (*ord-Zp* $x -$ *ord-Zp* y)) $\otimes$ *ac-Zp* $x \otimes$ (*inv ac-Zp* y)))

lemma *divide-closed*:
assumes $x \in$ *carrier Zp*
assumes $y \in$ *carrier Zp*
assumes $y \neq \mathbf{0}$
shows *divide* $x\ y \in$ *carrier Zp*
⟨*proof*⟩

lemma *divide-formula*:
assumes $x \in$ *carrier Zp*
assumes $y \in$ *carrier Zp*
assumes $y \neq \mathbf{0}$
assumes *val-Zp* $x \geq$ *val-Zp* y
shows $y \otimes$ *divide* $x\ y = x$
⟨*proof*⟩

lemma *divide-nonzero*:
assumes $x \in$ *nonzero Zp*
assumes $y \in$ *nonzero Zp*
assumes *val-Zp* $x \geq$ *val-Zp* y
shows *divide* $x\ y \in$ *nonzero Zp*
⟨*proof*⟩

lemma *val-of-divide*:
assumes $x \in$ *carrier Zp*
assumes $y \in$ *nonzero Zp*
assumes *val-Zp* $x \geq$ *val-Zp* y
shows *val-Zp* (*divide* $x\ y$) = *val-Zp* $x -$ *val-Zp* y
⟨*proof*⟩

lemma *val-of-divide′*:
assumes $x \in$ *carrier Zp*
assumes $y \in$ *carrier Zp*
assumes $y \neq \mathbf{0}$
assumes *val-Zp* $x \geq$ *val-Zp* y
shows *val-Zp* (*divide* x y) = *val-Zp* $x -$ *val-Zp* y
⟨*proof*⟩
end

lemma(**in** *UP-cring*) *taylor-deg-1-eval′′′*:
assumes $f \in$ *carrier* P
assumes $a \in$ *carrier* R
assumes $b \in$ *carrier* R
assumes $c =$ *to-fun* (*shift* (*2*::*nat*) (T_a f)) ($\ominus b$)
assumes $b \otimes$ (*deriv* f a) = (*to-fun* f a)
shows *to-fun* f ($a \ominus b$) = ($c \otimes b[\widehat{\ }](2::nat)$)
⟨*proof*⟩

lemma(**in** *padic-integers*) *res-diff-zero-fact*:
assumes $a \in$ *carrier Zp*
assumes $b \in$ *carrier Zp*
assumes ($a \ominus b$) k = *0*
shows $a\ k = b\ k\ a\ k \ominus_{Zp\text{-}res\text{-}ring\ k} b\ k$ = *0*
⟨*proof*⟩

lemma(**in** *padic-integers*) *res-diff-zero-fact′*:
assumes $a \in$ *carrier Zp*
assumes $b \in$ *carrier Zp*
assumes $a\ k = b\ k$
shows $a\ k \ominus_{Zp\text{-}res\text{-}ring\ k} b\ k$ = *0*
⟨*proof*⟩

lemma(**in** *padic-integers*) *res-diff-zero-fact′′*:
assumes $a \in$ *carrier Zp*
assumes $b \in$ *carrier Zp*
assumes $a\ k = b\ k$
shows ($a \ominus b$) k = *0*
⟨*proof*⟩

lemma(**in** *padic-integers*) *is-Zp-cauchyI′*:
assumes $s \in$ *closed-seqs Zp*
assumes $\forall\, n$::*nat*. $\exists$ k::*int*.$\forall\, m$. $m \geq k \longrightarrow$ *val-Zp* (s (*Suc* m) $\ominus$ s m) $\geq n$
shows *is-Zp-cauchy* s
⟨*proof*⟩

30 The Proof of Hensel's Lemma

30.1 Building a Locale for the Proof of Hensel's Lemma

locale *hensel* = *padic-integers*+
  fixes *f*::*padic-int-poly*
  fixes *a*::*padic-int*
  assumes *f-closed*[*simp*]: *f* ∈ *carrier Zp-x*
  assumes *a-closed*[*simp*]: *a* ∈ *carrier Zp*
  assumes *fa-nonzero*[*simp*]: *f*•*a* ≠**0**
  assumes *hensel-hypothesis*[*simp*]: (*val-Zp* (*f*•*a*) > 2∗ *val-Zp* ((*pderiv f*)•*a*))

sublocale *hensel* < *cring Zp*
  ⟨*proof*⟩

context *hensel*
begin

abbreviation *f*′ **where**
f′ ≡ *pderiv f*

lemma *f*′*-closed*:
f′ ∈ *carrier Zp-x*
  ⟨*proof*⟩

lemma *f*′*-vals-closed*:
  assumes *a* ∈ *carrier Zp*
  shows *f*′•*a* ∈ *carrier Zp*
  ⟨*proof*⟩

lemma *fa-closed*:
(*f*•*a*) ∈ *carrier Zp*
  ⟨*proof*⟩

lemma *f*′*a-closed*:
(*f*′•*a*) ∈ *carrier Zp*
⟨*proof*⟩

lemma *fa-nonzero*′:
(*f*•*a*) ∈ *nonzero Zp*
  ⟨*proof*⟩

lemma *f*′*a-nonzero*[*simp*]:
(*f*′•*a*) ≠ **0**
⟨*proof*⟩

lemma *f*′*a-nonzero*′:
(*f*′•*a*) ∈ *nonzero Zp*
  ⟨*proof*⟩

lemma *f'a-not-infinite*[*simp*]:
val-Zp (*f'•a*) ≠ ∞
⟨*proof*⟩

lemma *f'a-nonneg-val*[*simp*]:
val-Zp ((*f'•a*)) ≥ *0*
⟨*proof*⟩

lemma *hensel-hypothesis-weakened*:
val-Zp (*f•a*) > *val-Zp* (*f'•a*)
⟨*proof*⟩

30.2 Constructing the Newton Sequence

definition *newton-step* :: *padic-int* ⇒ *padic-int* **where**
newton-step x = *x* ⊖ (*divide* (*f•x*) (*f'•x*))

lemma *newton-step-closed*:
newton-step a ∈ *carrier Zp*
⟨*proof*⟩

fun *newton-seq* :: *padic-int-seq* (*ns*) **where**
newton-seq 0 = *a*|
newton-seq (*Suc n*) = *newton-step* (*newton-seq n*)

30.3 Key Properties of the Newton Sequence

lemma *hensel-factor-id*:
(*divide* (*f•a*) (*f'•a*)) ⊗ ((*f'•a*)) = (*f•a*)
⟨*proof*⟩

definition *hensel-factor* (*t*) **where**
hensel-factor = *val-Zp* (*f•a*) − *2*∗(*val-Zp* (*f'•a*))

lemma *t-pos*[*simp*]:
t > *0*
⟨*proof*⟩

lemma *t-neq-infty*[*simp*]:
t ≠ ∞
⟨*proof*⟩

lemma *t-times-pow-pos*[*simp*]:
(*2*^(*n*::*nat*))∗*t* > *0*
⟨*proof*⟩

lemma *newton-seq-props-induct*:
shows ⋀*k*. *k* ≤ *n* ⟹ (*ns k*) ∈ *carrier Zp*
∧ *val-Zp* (*f'•*(*ns k*)) = *val-Zp* ((*f'•a*))
∧ *val-Zp* (*f•*(*ns k*)) ≥ *2*∗(*val-Zp* (*f'•a*)) + (*2*^*k*)∗*t*

⟨*proof*⟩

lemma *newton-seq-closed*:
shows *ns m* ∈ *carrier Zp*
⟨*proof*⟩

lemma *f-of-newton-seq-closed*:
shows *f* · *ns m* ∈ *carrier Zp*
⟨*proof*⟩

lemma *newton-seq-fact1*[*simp*]:
val-Zp (*f'*·(*ns k*)) = *val-Zp* ((*f'*·*a*))
⟨*proof*⟩

lemma *newton-seq-fact2*:
⋀*k*. *val-Zp* (*f*·(*ns k*)) ≥ *2*∗(*val-Zp* (*f'*·*a*)) + (*2*^*k*)∗*t*
⟨*proof*⟩

lemma *newton-seq-fact3*:
val-Zp (*f*·(*ns l*)) ≥ *val-Zp* (*f'*·(*ns l*))
⟨*proof*⟩

lemma *newton-seq-fact4*[*simp*]:
assumes *f*·(*ns l*) ≠**0**
shows *val-Zp* (*f*·(*ns l*)) ≥ *val-Zp* (*f'*·(*ns l*))
⟨*proof*⟩

lemma *newton-seq-fact5*:
divide (*f* · *ns l*) (*f'* · *ns l*) ∈ *carrier Zp*
⟨*proof*⟩

lemma *newton-seq-fact6*:
(*f'*·(*ns l*)) ∈ *nonzero Zp*
⟨*proof*⟩

lemma *newton-seq-fact7*:
(*ns* (*Suc n*)) ⊖ (*ns n*) = ⊖*divide* (*f*·(*ns n*)) (*f'*·(*ns n*))
⟨*proof*⟩

lemma *newton-seq-fact8*:
assumes *f*·(*ns l*) ≠**0**
shows *divide* (*f* · *ns l*) (*f'* · *ns l*) ∈ *nonzero Zp*
⟨*proof*⟩

lemma *newton-seq-fact9*:
assumes *f*·(*ns n*) ≠**0**
shows *val-Zp*((*ns* (*Suc n*)) ⊖ (*ns n*)) = *val-Zp* (*f*·(*ns n*)) − *val-Zp* (*f'*·(*ns n*))
⟨*proof*⟩

Assuming no element of the Newton sequence is a root of f, the Newton

sequence is Cauchy.

lemma *newton-seq-is-Zp-cauchy-0*:
assumes $\bigwedge k.\ f`(ns\ k) \neq \mathbf{0}$
shows *is-Zp-cauchy ns*
⟨*proof*⟩

lemma *eventually-zero*:
$f \cdot ns\ (k + m) = \mathbf{0} \Longrightarrow f \cdot ns\ (k + Suc\ m) = \mathbf{0}$
⟨*proof*⟩

The Newton Sequence is Cauchy:

lemma *newton-seq-is-Zp-cauchy*:
is-Zp-cauchy ns
⟨*proof*⟩

30.4 The Proof of Hensel's Lemma

lemma *pre-hensel*:
$val\text{-}Zp\ (a \ominus (ns\ n)) >\ val\text{-}Zp\ (f'`a)$
$\exists N.\ \forall n.\ n > N \longrightarrow (val\text{-}Zp\ (a \ominus (ns\ n)) = val\text{-}Zp\ (divide\ (f`a)\ (f'`a)))$
$val\text{-}Zp\ (f'`(ns\ n)) = val\text{-}Zp\ (f'`a)$
⟨*proof*⟩

lemma *hensel-seq-comp-f*:
$res\text{-}lim\ ((to\text{-}fun\ f) \circ ns) = \mathbf{0}$
⟨*proof*⟩

lemma *full-hensels-lemma*:
obtains α **where**
$f`\alpha = \mathbf{0}$ **and** $\alpha \in carrier\ Zp$
$val\text{-}Zp\ (a \ominus \alpha) > val\text{-}Zp\ (f'`a)$
$(val\text{-}Zp\ (a \ominus \alpha) = val\text{-}Zp\ (divide\ (f`a)\ (f'`a)))$
$val\text{-}Zp\ (f'`\alpha) = val\text{-}Zp\ (f'`a)$
⟨*proof*⟩

end

31 Removing Hensel's Lemma from the Hensel Locale

context *padic-integers*
begin

lemma *hensels-lemma*:
assumes $f \in carrier\ Zp\text{-}x$
assumes $a \in carrier\ Zp$

assumes $(pderiv\ f)\cdot a \neq \mathbf{0}$
assumes $f\cdot a \neq \mathbf{0}$
assumes $val\text{-}Zp\ (f\cdot a) > 2 * val\text{-}Zp\ ((pderiv\ f)\cdot a)$
obtains α **where**
$f\cdot\alpha = \mathbf{0}$ **and** $\alpha \in carrier\ Zp$
$val\text{-}Zp\ (a \ominus \alpha) > val\text{-}Zp\ ((pderiv\ f)\cdot a)$
$val\text{-}Zp\ (a \ominus \alpha) = val\text{-}Zp\ (divide\ (f\cdot a)\ ((pderiv\ f)\cdot a))$
$val\text{-}Zp\ ((pderiv\ f)\cdot\alpha) = val\text{-}Zp\ ((pderiv\ f)\cdot a)$
⟨*proof*⟩

Uniqueness of the root found in Hensel's lemma

lemma *hensels-lemma-unique-root*:
assumes $f \in carrier\ Zp\text{-}x$
assumes $a \in carrier\ Zp$
assumes $(pderiv\ f)\cdot a \neq \mathbf{0}$
assumes $f\cdot a \neq \mathbf{0}$
assumes $(val\text{-}Zp\ (f\cdot a) > 2 * val\text{-}Zp\ ((pderiv\ f)\cdot a))$
assumes $f\cdot\alpha = \mathbf{0}$
assumes $\alpha \in carrier\ Zp$
assumes $val\text{-}Zp\ (a \ominus \alpha) > val\text{-}Zp\ ((pderiv\ f)\cdot a)$
assumes $f\cdot\beta = \mathbf{0}$
assumes $\beta \in carrier\ Zp$
assumes $val\text{-}Zp\ (a \ominus \beta) > val\text{-}Zp\ ((pderiv\ f)\cdot a)$
assumes $val\text{-}Zp\ ((pderiv\ f)\cdot\alpha) = val\text{-}Zp\ ((pderiv\ f)\cdot a)$
shows $\alpha = \beta$
⟨*proof*⟩

lemma *hensels-lemma'*:
assumes $f \in carrier\ Zp\text{-}x$
assumes $a \in carrier\ Zp$
assumes $val\text{-}Zp\ (f\cdot a) > 2 * val\text{-}Zp\ ((pderiv\ f)\cdot a)$
shows $\exists!\alpha \in carrier\ Zp.\ f\cdot\alpha = \mathbf{0} \wedge val\text{-}Zp\ (a \ominus \alpha) > val\text{-}Zp\ ((pderiv\ f)\cdot a)$
⟨*proof*⟩

32 Some Applications of Hensel's Lemma to Root Finding for Polynomials over $\mathbb{Z}_p$

lemma *Zp-square-root-criterion*:
assumes $p \neq 2$
assumes $a \in carrier\ Zp$
assumes $b \in carrier\ Zp$
assumes $val\text{-}Zp\ b \geq val\text{-}Zp\ a$
assumes $a \neq \mathbf{0}$
assumes $b \neq \mathbf{0}$
shows $\exists\, y \in carrier\ Zp.\ a[\hat{}](2::nat) \oplus \mathrm{p}\otimes b[\hat{}](2::nat) = (y\ [\hat{}]_{Zp}\ (2::nat))$
⟨*proof*⟩

lemma *Zp-semialg-eq*:

assumes $a \in$ *nonzero Zp*
shows $\exists\, y \in$ *carrier Zp*. $\mathbf{1} \oplus (\mathrm{p}\ [\widehat{}]\ (3{::}nat)) \otimes (a\ [\widehat{}]\ (4{::}nat)) = (y\ [\widehat{}]\ (2{::}nat))$
⟨*proof*⟩

lemma *Zp-nth-root-lemma*:
assumes $a \in$ *carrier Zp*
assumes $a \neq \mathbf{1}$
assumes $n > 1$
assumes *val-Zp* $(\mathbf{1} \ominus a) >$ *2∗val-Zp* $([(n{::}nat)]\cdot \mathbf{1})$
shows $\exists\ b \in$ *carrier Zp*. $b[\widehat{}]n = a$
⟨*proof*⟩

end
end
theory *Zp-Compact*
imports *Padic-Int-Topology*
begin

context *padic-integers*
begin

lemma *res-ring-car*:
carrier (*Zp-res-ring k*) = {*0..p* ^ *k* − *1*}
⟨*proof*⟩

The refinement of a sequence by a function $nat \Rightarrow nat$

definition *take-subseq* :: $(nat \Rightarrow 'a) \Rightarrow (nat \Rightarrow nat) \Rightarrow (nat \Rightarrow 'a)$ **where**
take-subseq s f = $(\lambda k.\ s\ (f\ k))$

Predicate for increasing function on the natural numbers

definition *is-increasing* :: $(nat \Rightarrow nat) \Rightarrow bool$ **where**
is-increasing f = $(\forall\ n\ m{::}nat.\ n{>}m \longrightarrow (f\ n) > (f\ m))$

Elimination and introduction lemma for increasing functions

lemma *is-increasingI*:
assumes $\bigwedge\ n\ m{::}nat.\ n{>}m \Longrightarrow (f\ n) > (f\ m)$
shows *is-increasing f*
⟨*proof*⟩

lemma *is-increasingE*:
assumes *is-increasing f*
assumes $n{>}\ m$
shows $f\ n > f\ m$
⟨*proof*⟩

The subsequence predicate

definition *is-subseq-of* :: $(nat \Rightarrow 'a) \Rightarrow (nat \Rightarrow 'a) \Rightarrow bool$ **where**
is-subseq-of s s′ = $(\exists (f{::}nat \Rightarrow nat).\ is\text{-}increasing\ f \wedge s' = take\text{-}subseq\ s\ f)$

Subsequence introduction lemma

lemma *is-subseqI*:
assumes *is-increasing f*
assumes $s' = take\text{-}subseq\ s\ f$
shows *is-subseq-of s s'*
⟨*proof*⟩

lemma *is-subseq-ind*:
assumes *is-subseq-of s s'*
shows $\exists\ l.\ s'\ k = s\ l$
⟨*proof*⟩

lemma *is-subseq-closed*:
assumes $s \in closed\text{-}seqs\ Zp$
assumes *is-subseq-of s s'*
shows $s' \in closed\text{-}seqs\ Zp$
⟨*proof*⟩

Given a sequence and a predicate, returns the function from nat to nat which represents the increasing sequences of indices n on which P (s n) holds.

primrec *seq-filter* :: $(nat \Rightarrow 'a) \Rightarrow ('a \Rightarrow bool) \Rightarrow nat \Rightarrow nat$ **where**
$seq\text{-}filter\ s\ P\ (0::nat) = (LEAST\ k::nat.\ P\ (s\ k))|$
$seq\text{-}filter\ s\ P\ (Suc\ n) = (LEAST\ k::\ nat.\ (P\ (s\ k)) \wedge k > (seq\text{-}filter\ s\ P\ n))$

lemma *seq-filter-pre-increasing*:
assumes $\forall\, n::nat.\ \exists\, m.\ m > n \wedge P\ (s\ m)$
shows $seq\text{-}filter\ s\ P\ n < seq\text{-}filter\ s\ P\ (Suc\ n)$
⟨*proof*⟩

lemma *seq-filter-increasing*:
assumes $\forall\, n::nat.\ \exists\, m.\ m > n \wedge P\ (s\ m)$
shows $is\text{-}increasing\ (seq\text{-}filter\ s\ P)$
⟨*proof*⟩

definition *filtered-seq* :: $(nat \Rightarrow 'a) \Rightarrow ('a \Rightarrow bool) \Rightarrow (nat \Rightarrow 'a)$ **where**
$filtered\text{-}seq\ s\ P = take\text{-}subseq\ s\ (seq\text{-}filter\ s\ P)$

lemma *filter-exist*:
assumes $s \in closed\text{-}seqs\ Zp$
assumes $\forall\, n::nat.\ \exists\, m.\ m > n \wedge P\ (s\ m)$
shows $\bigwedge m.\ n \leq m \Longrightarrow P\ (s\ (seq\text{-}filter\ s\ P\ n))$
⟨*proof*⟩

In a filtered sequence, every element satisfies the filtering predicate

lemma *fil-seq-pred*:
assumes $s \in closed\text{-}seqs\ Zp$
assumes $s' = filtered\text{-}seq\ s\ P$
assumes $\forall\, n::nat.\ \exists\, m.\ m > n \wedge P\ (s\ m)$

shows $\bigwedge m{::}nat.\ P\ (s'\ m)$
⟨*proof*⟩

definition *kth-res-equals* :: *nat* ⇒ *int* ⇒ (*padic-int* ⇒ *bool*) **where**
kth-res-equals k n a = (*a k* = *n*)

definition *indicator*:: (*nat* ⇒ *'a*) ⇒ (*'a* ⇒ *bool*) **where**
indicator s a = (∃ *n*::*nat*. *s n* = *a*)

Choice function for a subsequence with constant kth residue. Could be made constructive by choosing the LEAST n if we wanted.

definition *const-res-subseq* :: *nat* ⇒ *padic-int-seq* ⇒ *padic-int-seq* **where**
const-res-subseq k s = (*SOME s'*::(*padic-int-seq*). (∃ *n*. *is-subseq-of s s'* ∧ *s'*
= (*filtered-seq s* (*kth-res-equals k n*)) ∧ (∀ *m*. *s' m k* = *n*)))

The constant kth residue value for the sequence obtained by the previous function

definition *const-res* :: *nat* ⇒ *padic-int-seq* ⇒ *int* **where**
const-res k s = (*THE n*. (∀ *m*. (*const-res-subseq k s*) *m k* = *n*))

definition *maps-to-n*:: *int* ⇒ (*nat* ⇒ *int*) ⇒ *bool* **where**
maps-to-n n f = (∀ (*k*::*nat*). *f k* ∈ {*0*..*n*})

definition *drop-res* :: *int* ⇒ (*nat* ⇒ *int*) ⇒ (*nat* ⇒ *int*) **where**
drop-res k f n = (*if* (*f n*) = *k then 0 else f n*)

lemma *maps-to-nE*:
assumes *maps-to-n n f*
shows (*f k*) ∈ {*0*..*n*}
⟨*proof*⟩

lemma *maps-to-nI*:
assumes $\bigwedge n.\ f\ n \in \{0\ ..\ k\}$
shows *maps-to-n k f*
⟨*proof*⟩

lemma *maps-to-n-drop-res*:
assumes *maps-to-n* (*Suc n*) *f*
shows *maps-to-n n* (*drop-res* (*Suc n*) *f*)
⟨*proof*⟩

lemma *drop-res-eq-f*:
assumes *maps-to-n* (*Suc n*) *f*
assumes ¬ (∀ *m*. ∃ *n*. *n*>*m* ∧ (*f n* = (*Suc k*)))
shows ∃ *N*. ∀ *n*. *n*>*N* ⟶ *f n* = *drop-res* (*Suc k*) *f n*
⟨*proof*⟩

lemma *maps-to-n-infinite-seq*:
shows $\bigwedge f.$ *maps-to-n* (*k*::*nat*) *f* $\Longrightarrow \exists\, l$::*int*. $\forall\, m.\ \exists\, n.\ n{>}m \wedge$ (*f n* = *l*)
⟨*proof*⟩

lemma *int-nat-p-pow-minus*:
int (*nat* (*p* ^ *k* − *1*)) = *p* ^ *k* − *1*
⟨*proof*⟩

lemma *maps-to-n-infinite-seq-res-ring*:
$\bigwedge f.$ *f* ∈ (*UNIV*::*nat set*) → *carrier* (*Zp-res-ring k*) $\Longrightarrow \exists\, l.\ \forall\, m.\ \exists\, n.\ n{>}m \wedge$ (*f n* = *l*)
⟨*proof*⟩

definition *index-to-residue* :: *padic-int-seq* ⇒ *nat* ⇒ *nat* ⇒ *int* **where**
index-to-residue s k m = ((*s m*) *k*)

lemma *seq-maps-to-n*:
assumes *s* ∈ *closed-seqs Zp*
shows (*index-to-residue s k*) ∈ *UNIV* → *carrier* (*Zp-res-ring k*)
⟨*proof*⟩

lemma *seq-pr-inc*:
assumes *s* ∈ *closed-seqs Zp*
shows $\exists\, l.\ \forall\, m.\ \exists\, n > m.$ (*kth-res-equals k l*) (*s n*)
⟨*proof*⟩

lemma *kth-res-equals-subseq*:
assumes *s* ∈ *closed-seqs Zp*
shows $\exists\, n.$ *is-subseq-of s* (*filtered-seq s* (*kth-res-equals k n*)) ∧ (∀ *m.* (*filtered-seq s* (*kth-res-equals k n*)) *m k* = *n*)
⟨*proof*⟩

lemma *const-res-subseq-prop-0*:
assumes *s* ∈ *closed-seqs Zp*
shows $\exists\, l.$ (((*const-res-subseq k s*) = *filtered-seq s* (*kth-res-equals k l*)) ∧ (*is-subseq-of s* (*const-res-subseq k s*)) ∧ (∀ *m.*(*const-res-subseq k s*) *m k* = *l*))
⟨*proof*⟩

lemma *const-res-subseq-prop-1*:
assumes *s* ∈ *closed-seqs Zp*
shows (∀ *m.*(*const-res-subseq k s*) *m k* = (*const-res k s*))
⟨*proof*⟩

lemma *const-res-subseq*:
assumes *s* ∈ *closed-seqs Zp*
shows *is-subseq-of s* (*const-res-subseq k s*)
⟨*proof*⟩

lemma *const-res-range*:

assumes $s \in$ *closed-seqs Zp*
assumes $k > 0$
shows *const-res k s* $\in$ *carrier* (*Zp-res-ring k*)
⟨*proof*⟩

fun *res-seq* ::*padic-int-seq* ⇒ *nat* ⇒ *padic-int-seq* **where**
res-seq s 0 = *s*|
res-seq s (*Suc k*) = *const-res-subseq* (*Suc k*) (*res-seq s k*)

lemma *res-seq-res*:
assumes $s \in$ *closed-seqs Zp*
shows (*res-seq s k*) $\in$ *closed-seqs Zp*
⟨*proof*⟩

lemma *res-seq-res′*:
assumes $s \in$ *closed-seqs Zp*
shows $\bigwedge n$. *res-seq s* (*Suc k*) *n* (*Suc k*) = *const-res* (*Suc k*) (*res-seq s k*)
⟨*proof*⟩

lemma *res-seq-subseq*:
assumes $s \in$ *closed-seqs Zp*
shows *is-subseq-of* (*res-seq s k*) (*res-seq s* (*Suc k*))
⟨*proof*⟩

lemma *is-increasing-id*:
is-increasing (λ *n*. *n*)
⟨*proof*⟩

lemma *is-increasing-comp*:
assumes *is-increasing f*
assumes *is-increasing g*
shows *is-increasing* (*f* ∘ *g*)
⟨*proof*⟩

lemma *is-increasing-imp-geq-id*[*simp*]:
assumes *is-increasing f*
shows *f n* ≥*n*
⟨*proof*⟩

lemma *is-subseq-ofE*:
assumes $s \in$ *closed-seqs Zp*
assumes *is-subseq-of s s′*
shows ∃ *k*. $k \geq n \wedge s'\ n = s\ k$
⟨*proof*⟩

lemma *is-subseq-of-id*:
assumes $s \in$ *closed-seqs Zp*
shows *is-subseq-of s s*

⟨*proof*⟩

lemma *is-subseq-of-trans*:
assumes $s \in$ *closed-seqs Zp*
assumes *is-subseq-of s s'*
assumes *is-subseq-of s' s''*
shows *is-subseq-of s s''*
⟨*proof*⟩

lemma *res-seq-subseq'*:
assumes $s \in$ *closed-seqs Zp*
shows *is-subseq-of s* (*res-seq s k*)
⟨*proof*⟩

lemma *res-seq-subseq''*:
assumes $s \in$ *closed-seqs Zp*
shows *is-subseq-of* (*res-seq s n*) (*res-seq s* ($n + k$))
⟨*proof*⟩

definition *acc-point* :: *padic-int-seq* ⇒ *padic-int* **where**
acc-point s k = (*if* ($k = 0$) *then* (*0*::*int*) *else* ((*res-seq s k*) *0 k*))

lemma *res-seq-res-1*:
assumes $s \in$ *closed-seqs Zp*
shows *res-seq s* (*Suc k*) *0 k* = *res-seq s k 0 k*
⟨*proof*⟩

lemma *acc-point-cres*:
assumes $s \in$ *closed-seqs Zp*
shows (*acc-point s* (*Suc k*)) = (*const-res* (*Suc k*) (*res-seq s k*))
⟨*proof*⟩

lemma *acc-point-res*:
assumes $s \in$ *closed-seqs Zp*
shows *residue* ($p \hat{} k$) (*acc-point s* (*Suc k*)) = *acc-point s k*
⟨*proof*⟩

lemma *acc-point-closed*:
assumes $s \in$ *closed-seqs Zp*
shows *acc-point s* ∈ *carrier Zp*
⟨*proof*⟩

Choice function for a subsequence of s which converges to a, if it exists

fun *convergent-subseq-fun* :: *padic-int-seq* ⇒ *padic-int* ⇒ (*nat* ⇒ *nat*) **where**
convergent-subseq-fun s a 0 = *0*|
convergent-subseq-fun s a (*Suc n*) = (*SOME k*. $k >$ (*convergent-subseq-fun s a n*)
∧ (*s k* (*Suc n*)) = *a* (*Suc n*))

definition *convergent-subseq* :: *padic-int-seq* ⇒ *padic-int-seq* **where**
convergent-subseq s = *take-subseq s* (*convergent-subseq-fun s* (*acc-point s*))

lemma *increasing-conv-induction-0-pre*:
assumes $s \in$ *closed-seqs Zp*
assumes a = *acc-point s*
shows $\exists k >$ *convergent-subseq-fun s a n*. (*s k* (*Suc n*)) = *a* (*Suc n*)
⟨*proof*⟩

lemma *increasing-conv-subseq-fun-0*:
assumes $s \in$ *closed-seqs Zp*
assumes $\exists s'$. s' = *convergent-subseq s*
assumes a = *acc-point s*
shows *convergent-subseq-fun s a* (*Suc n*) > *convergent-subseq-fun s a n*
⟨*proof*⟩

lemma *increasing-conv-subseq-fun*:
assumes $s \in$ *closed-seqs Zp*
assumes a = *acc-point s*
assumes $\exists s'$. s' = *convergent-subseq s*
shows *is-increasing* (*convergent-subseq-fun s a*)
⟨*proof*⟩

lemma *convergent-subseq-is-subseq*:
assumes $s \in$ *closed-seqs Zp*
shows *is-subseq-of s* (*convergent-subseq s*)
⟨*proof*⟩

lemma *is-closed-seq-conv-subseq*:
assumes $s \in$ *closed-seqs Zp*
shows (*convergent-subseq s*) ∈ *closed-seqs Zp*
⟨*proof*⟩

lemma *convergent-subseq-res*:
assumes $s \in$ *closed-seqs Zp*
assumes a = *acc-point s*
shows *convergent-subseq s l l* = *residue* (*p* ^ *l*) (*acc-point s l*)
⟨*proof*⟩

lemma *convergent-subseq-res′*:
assumes $s \in$ *closed-seqs Zp*
assumes $n > l$
shows *convergent-subseq s n l* = *convergent-subseq s l l*
⟨*proof*⟩

lemma *convergent-subsequence-is-convergent*:
assumes $s \in$ *closed-seqs Zp*
assumes a = *acc-point s*
shows *Zp-converges-to* (*convergent-subseq s*) (*acc-point s*)

⟨*proof*⟩

theorem *Zp-is-compact*:
  assumes $s \in$ *closed-seqs Zp*
  shows $\exists\, s'$. *is-subseq-of s s'* $\wedge$ (*Zp-converges-to s'* (*acc-point s*))
  ⟨*proof*⟩

end
end

References

[1] K. Conrad. Hensel's lemma.

[2] J. Denef. p-adic semi-algebraic sets and cell decomposition. *Journal für die reine und angewandte Mathematik*, 369:154–166, 1986.

[3] D. Dummit. *Abstract algebra*. John Wiley & Sons, Inc, Hoboken, NJ, 2004.

[4] A. Engler. *Valued fields*. Springer, Berlin New York, 2005.

[5] J. Pas. On the angular component map modulo p. *The Journal of Symbolic Logic*, 55(3):1125–1129, 1990.