Computational p-Adics

Jeremy Sylvestre University of Alberta (Augustana Campus)

October 13, 2025

Abstract

We develop a framework for reasoning computationally about p-adic fields via formal Laurent series with integer coefficients. We define a ring-class quotient type consisting of equivalence classes of prime-indexed sequences of formal Laurent series, in which every p-adic field is contained as a subring. Via a further quotient of the ring of p-adic integers (that is, the elements of depth at least 0) by the prime ideal (that is, the subring of elements of depth at least 1), we define a ring-class type in which every prime-order finite field is contained as a subring. Finally, some topological properties are developed using depth as a proxy for a metric, Hensel's Lemma is proved, and the compactness of local rings of integers is established.

Contents

theory *Distinguished-Quotients*

imports *Main*

begin

1 Distinguished Algebraic Quotients

We define quotient groups and quotient rings modulo distinguished normal subgroups and ideals, respectively, leading to the construction of a field as a commutative ring with 1 modulo a maximal ideal.

class *distinguished-subset* =
  fixes *distinguished-subset* :: $'a\ set$ ($\mathcal{N}$)

hide-const *distinguished-subset*

1.1 Quotient groups

1.1.1 Class definitions

Here we set up subclasses of *group-add* with a distinguished subgroup.

class *group-add-with-distinguished-subgroup* = *group-add* + *distinguished-subset* +
  assumes *distset-nonempty*: $\mathcal{N} \neq \{\}$
    and *distset-diff-closed*:
    $g \in \mathcal{N} \Longrightarrow h \in \mathcal{N} \Longrightarrow g - h \in \mathcal{N}$
begin

lemma *distset-zero-closed* : $0 \in \mathcal{N}$
⟨*proof*⟩

lemma *distset-uminus-closed*: $a \in \mathcal{N} \Longrightarrow -a \in \mathcal{N}$
  ⟨*proof*⟩

lemma *distset-add-closed*:
  $a \in \mathcal{N} \Longrightarrow b \in \mathcal{N} \Longrightarrow a + b \in \mathcal{N}$
  ⟨*proof*⟩

lemma *distset-uminus-plus-closed*:
  $a \in \mathcal{N} \Longrightarrow b \in \mathcal{N} \Longrightarrow -a + b \in \mathcal{N}$
  ⟨*proof*⟩

lemma *distset-lcoset-closed1* :

$a \in \mathcal{N} \Longrightarrow -a + b \in \mathcal{N} \Longrightarrow b \in \mathcal{N}$
⟨*proof*⟩

lemma *distset-lcoset-closed2*:
$b \in \mathcal{N} \Longrightarrow -a + b \in \mathcal{N} \Longrightarrow a \in \mathcal{N}$
⟨*proof*⟩

lemma *reflp-lcosetrel*: $- x + x \in \mathcal{N}$
⟨*proof*⟩

lemma *symp-lcosetrel*: $- a + b \in \mathcal{N} \Longrightarrow - b + a \in \mathcal{N}$
⟨*proof*⟩

lemma *transp-lcosetrel*:
$-x + z \in \mathcal{N}$ **if** $-x + y \in \mathcal{N}$ **and** $-y + z \in \mathcal{N}$
⟨*proof*⟩

end

class *group-add-with-distinguished-normal-subgroup* = *group-add-with-distinguished-subgroup* +
assumes *distset-normal*: $((+)\ g)$ ‘ $\mathcal{N} = (\lambda x.\ x + g)$ ‘ $\mathcal{N}$

class *ab-group-add-with-distinguished-subgroup* =
ab-group-add + *group-add-with-distinguished-subgroup*
begin

subclass *group-add-with-distinguished-normal-subgroup*
⟨*proof*⟩

end

1.1.2 The quotient group type

Here we define a quotient type relative to the left-coset relation, and instantiate it as a *group-add*.

quotient-type (**overloaded**) *'a coset-by-dist-sbgrp* =
'a::*group-add-with-distinguished-subgroup* / $\lambda g\ h$ :: *'a*. $-g + h \in (\mathcal{N}$::*'a set*)
morphisms *distinguished-quotient-coset-rep distinguished-quotient-coset*
⟨*proof*⟩

lemmas *distinguished-quotient-coset-rep-inverse* =
Quotient-abs-rep[*OF Quotient-coset-by-dist-sbgrp*]
and *coset-by-dist-sbgrp-eq-iff* =
Quotient-rel-rep[*OF Quotient-coset-by-dist-sbgrp*]
and *distinguished-quotient-coset-eqI* =
Quotient-rel-abs[*OF Quotient-coset-by-dist-sbgrp*]
and *eq-to-distinguished-quotient-cosetI* =
Quotient-rel-abs2[*OF Quotient-coset-by-dist-sbgrp*]

lemma *coset-coset-by-dist-sbgrp-cases*
[*case-names coset*, *cases type*: *coset-by-dist-sbgrp*]:
$(\bigwedge y.\ z = \textit{distinguished-quotient-coset}\ y \Longrightarrow P) \Longrightarrow P$
⟨*proof*⟩

lemmas *distinguished-quotient-coset-eq-iff* = *coset-by-dist-sbgrp.abs-eq-iff*

lemma *distinguished-quotient-coset-rep-coset-equiv*:
$-$ *distinguished-quotient-coset-rep* (*distinguished-quotient-coset* r) $+\ r \in \mathcal{N}$
⟨*proof*⟩

instantiation *coset-by-dist-sbgrp* ::
(*group-add-with-distinguished-normal-subgroup*) *group-add*
begin

lift-definition *zero-coset-by-dist-sbgrp* :: *'a coset-by-dist-sbgrp*
is *0::'a* ⟨*proof*⟩

lemmas *zero-distinguished-quotient* [*simp*] = *zero-coset-by-dist-sbgrp.abs-eq*[*symmetric*]

lemma *zero-distinguished-quotient-eq-iffs*:
(*distinguished-quotient-coset* $x = 0$) = ($x \in \mathcal{N}$)
($X = 0$) = (*distinguished-quotient-coset-rep* $X \in \mathcal{N}$)
⟨*proof*⟩

lift-definition *plus-coset-by-dist-sbgrp* ::
'a coset-by-dist-sbgrp ⇒ *'a coset-by-dist-sbgrp* ⇒
'a coset-by-dist-sbgrp
is $\lambda x\ y{::}'a.\ x + y$
⟨*proof*⟩

lemmas *plus-coset-by-dist-sbgrp* [*simp*] = *plus-coset-by-dist-sbgrp.abs-eq*

lift-definition *uminus-coset-by-dist-sbgrp* ::
'a coset-by-dist-sbgrp ⇒ *'a coset-by-dist-sbgrp*
is $\lambda x{::}'a.\ -x$
⟨*proof*⟩

lemmas *uminus-coset-by-dist-sbgrp* [*simp*] =
uminus-coset-by-dist-sbgrp.abs-eq

lift-definition *minus-coset-by-dist-sbgrp* ::
'a coset-by-dist-sbgrp ⇒ *'a coset-by-dist-sbgrp* ⇒
'a coset-by-dist-sbgrp
is $\lambda x\ y{::}'a.\ x - y$
⟨*proof*⟩

lemmas *minus-coset-by-dist-sbgrp* [*simp*] = *minus-coset-by-dist-sbgrp.abs-eq*

instance ⟨*proof*⟩

end

instance *coset-by-dist-sbgrp* ::
(*ab-group-add-with-distinguished-subgroup*) *ab-group-add*
⟨*proof*⟩

1.2 Quotient rings

1.2.1 Class definitions

Now we set up subclasses of *ring* with a distinguished ideal.

class *ring-with-distinguished-ideal* = *ring* + *ab-group-add-with-distinguished-subgroup* +
assumes *distset-leftideal* : $a \in \mathcal{N} \Longrightarrow r * a \in \mathcal{N}$
and *distset-rightideal*: $a \in \mathcal{N} \Longrightarrow a * r \in \mathcal{N}$

class *ring-1-with-distinguished-ideal* = *ring-1* + *ring-with-distinguished-ideal* +
assumes *distset-no1*: $1 \notin \mathcal{N}$
— For unitary rings we do not allow the quotient to be trivial to ensure that *0* and *1* will be different.

1.2.2 The quotient ring type

We just reuse the *'a coset-by-dist-sbgrp* type to define a quotient ring, and instantiate it as a *ring*.

type-synonym *'a distinguished-quotient-ring* = *'a coset-by-dist-sbgrp*
— This is intended to be used with *ring-with-distinguished-ideal* or one of its subclasses.

instantiation *coset-by-dist-sbgrp* :: (*ring-with-distinguished-ideal*) *ring*
begin

lift-definition *times-coset-by-dist-sbgrp* ::
'a distinguished-quotient-ring ⇒ *'a distinguished-quotient-ring* ⇒
'a distinguished-quotient-ring
is $\lambda x\ y::'a.\ x * y$
⟨*proof*⟩

lemmas *times-coset-by-dist-sbgrp* [*simp*] = *times-coset-by-dist-sbgrp.abs-eq*

instance ⟨*proof*⟩

end

instantiation *coset-by-dist-sbgrp* :: (*ring-1-with-distinguished-ideal*) *ring-1*

begin

lift-definition *one-coset-by-dist-sbgrp* :: *'a distinguished-quotient-ring*
is *1*::*'a* ⟨*proof*⟩

lemmas *one-coset-by-dist-sbgrp* [*simp*] = *one-coset-by-dist-sbgrp.abs-eq*[*symmetric*]

instance ⟨*proof*⟩

end

1.3 Quotients of commutative rings

For a commutative ring, the quotient modulo a prime ideal is an integral domain, and the quotient modulo a maximal ideal is a field.

1.3.1 Class definitions

class *comm-ring-with-distinguished-ideal* = *comm-ring* + *ab-group-add-with-distinguished-subgroup*
+
assumes *distset-ideal* : $a \in \mathcal{N} \Longrightarrow r * a \in \mathcal{N}$
begin

subclass *ring-with-distinguished-ideal*
⟨*proof*⟩

definition *distideal-extension* :: *'a* ⇒ *'a set*
where *distideal-extension* $a \equiv \{x.\ \exists\, r\ z.\ z \in \mathcal{N} \wedge x = r * a + z\}$

lemma *distideal-extension*: $\mathcal{N} \subseteq$ *distideal-extension* a
⟨*proof*⟩

lemma *distideal-extension-diff-closed*:
$r \in$ *distideal-extension* $a \Longrightarrow s \in$ *distideal-extension* $a \Longrightarrow$
$r - s \in$ *distideal-extension* a
⟨*proof*⟩

lemma *distideal-extension-ideal*:
$x \in$ *distideal-extension* $a \Longrightarrow r * x \in$ *distideal-extension* a
⟨*proof*⟩

end

class *comm-ring-1-with-distinguished-ideal* = *ring-1* + *comm-ring-with-distinguished-ideal*
+
assumes *distset-no1*:
$1 \notin \mathcal{N}$
— Don't allow the quotient to be trivial.
begin

subclass *ring-1-with-distinguished-ideal* ⟨*proof*⟩

lemma *distideal-extension-by*: $a \in$ *distideal-extension* a
⟨*proof*⟩

end

class *comm-ring-1-with-distinguished-pideal* = *comm-ring-1-with-distinguished-ideal* +
assumes *distset-pideal*: $r * s \in \mathcal{N} \Longrightarrow r \in \mathcal{N} \vee s \in \mathcal{N}$

class *comm-ring-1-with-distinguished-maxideal* = *comm-ring-1-with-distinguished-ideal* +
assumes *distset-maxideal*:
$[\![$
$\mathcal{N} \subseteq I$; $1 \notin I$;
$\forall\, r\, s.\ r \in I \wedge s \in I \longrightarrow r - s \in I$;
$\forall\, a\, r.\ a \in I \longrightarrow r * a \in I$
$]\!] \Longrightarrow I = \mathcal{N}$
begin

lemma *distset-maxideal-vs-distideal-extension*:
$1 \notin$ *distideal-extension* $a \Longrightarrow$ *distideal-extension* $a = \mathcal{N}$
⟨*proof*⟩

subclass *comm-ring-1-with-distinguished-pideal*
⟨*proof*⟩

end

1.3.2 Instances and instantiations

instance *coset-by-dist-sbgrp* :: (*comm-ring-with-distinguished-ideal*) *comm-ring*
⟨*proof*⟩

instance *coset-by-dist-sbgrp* :: (*comm-ring-1-with-distinguished-ideal*) *comm-ring-1*
⟨*proof*⟩

instance *coset-by-dist-sbgrp* :: (*comm-ring-1-with-distinguished-pideal*) *idom*
⟨*proof*⟩

lemma *inverse-in-quotient-mod-maxideal*:
$\exists\, y.\ -(y * x) + 1 \in \mathcal{N}$ **if** $x \notin \mathcal{N}$
for x :: *'a::comm-ring-1-with-distinguished-maxideal*
⟨*proof*⟩

instantiation *coset-by-dist-sbgrp* :: (*comm-ring-1-with-distinguished-maxideal*) *field*
begin

lift-definition *inverse-coset-by-dist-sbgrp* ::
$'a$ *distinguished-quotient-ring* $\Rightarrow$ $'a$ *distinguished-quotient-ring*
is $\lambda x{::}'a.$ *if* $x{\in}\mathcal{N}$ *then 0 else* (*SOME* $y{::}'a.$ $-(y{*}x) + 1 \in \mathcal{N}$)
⟨*proof*⟩

lemma *inverse-coset-by-dist-sbgrp-eq0* [*simp*]:
$x \in \mathcal{N} \Longrightarrow$ *inverse* (*distinguished-quotient-coset* x) $= 0$
⟨*proof*⟩

lemma *coset-by-dist-sbgrp-inverse-rep*:
$x \notin \mathcal{N} \Longrightarrow -(y * x) + 1 \in \mathcal{N} \Longrightarrow$
inverse (*distinguished-quotient-coset* x) = *distinguished-quotient-coset* y
⟨*proof*⟩

definition *divide-coset-by-dist-sbgrp-def* [*simp*]:
x *div* y = ($x{::}'a$ *distinguished-quotient-ring*) $*$ *inverse* y

lemma *coset-by-dist-sbgrp-divide-rep*:
$y \notin \mathcal{N} \Longrightarrow -(z * y) + 1 \in \mathcal{N} \Longrightarrow$
distinguished-quotient-coset x *div* *distinguished-quotient-coset* y
= *distinguished-quotient-coset* ($x * z$)
⟨*proof*⟩

instance ⟨*proof*⟩

end

end

theory *Polynomial-Extras*

imports *HOL−Computational-Algebra.Polynomial*

begin

2 Additional facts about polynomials

2.1 General facts

lemma *set-strip-while*: *set* (*strip-while P xs*) $\subseteq$ *set xs*
⟨*proof*⟩

lemma *pCons-induct′* [*case-names 0 pCons0 pCons*]:
P p
if *zero* : *P 0*

and *pCons0* : $\bigwedge a.\ a \neq 0 \Longrightarrow P\ (pCons\ a\ 0)$
and *pCons-n0* : $\bigwedge a\ p.\ p \neq 0 \Longrightarrow P\ p \Longrightarrow P\ (pCons\ a\ p)$
⟨*proof*⟩

lemma *pCons-decomp*: $pCons\ a\ p = pCons\ a\ 0 + pCons\ 0\ p$
⟨*proof*⟩

lemma *coeffs-smult-eq-smult-coeffs*:
$coeffs\ (smult\ x\ p) = map\ ((*)\ x)\ (strip\text{-}while\ (\lambda c.\ x * c = 0)\ (coeffs\ p))$
⟨*proof*⟩

lemma *coeffs-smult-eq-smult-coeffs-no-zero-divisors*:
$coeffs\ (smult\ x\ p) = map\ ((*)\ x)\ (coeffs\ p)$ **if** $x \neq 0$
for $x :: 'a::\{comm\text{-}semiring\text{-}0, semiring\text{-}no\text{-}zero\text{-}divisors\}$ **and** $p :: 'a\ poly$
⟨*proof*⟩

2.2 Derivatives

The type sort on *pderiv* is overly restrictive for algebraic purposes, so here is the relevant material with the type sort relaxed.

function *polyderiv* :: $('a :: comm\text{-}monoid\text{-}add)\ poly \Rightarrow 'a\ poly$
where $polyderiv\ (pCons\ a\ p) = (if\ p = 0\ then\ 0\ else\ p + pCons\ 0\ (polyderiv\ p))$
⟨*proof*⟩

termination *polyderiv*
⟨*proof*⟩

declare *polyderiv.simps*[*simp del*]

lemma *polyderiv-0* [*simp*]: $polyderiv\ 0 = 0$
⟨*proof*⟩

lemma *polyderiv-pCons*: $polyderiv\ (pCons\ a\ p) = p + pCons\ 0\ (polyderiv\ p)$
⟨*proof*⟩

lemma *polyderiv-1* [*simp*]: $polyderiv\ 1 = 0$
⟨*proof*⟩

2.3 Additive polynomials

The Binomial Theorem implies that to every polynomial can be associated a finite list of polynomials that describe how the original polynomial evaluates on binomial arguments.

function *additive-poly-poly* :: $'a::comm\text{-}semiring\text{-}1\ poly \Rightarrow 'a\ poly\ poly$
where
$additive\text{-}poly\text{-}poly\ (pCons\ a\ p) =$
$(if\ p = 0$
$then\ [:[:a:]:]$

else smult (*pCons 0 1*) (*additive-poly-poly p*) + *pCons* [:*a*:] (*additive-poly-poly p*)
)
⟨*proof*⟩

termination *additive-poly-poly*
⟨*proof*⟩

declare *additive-poly-poly.simps*[*simp del*]

lemma *additive-poly-poly-0* [*simp*]: *additive-poly-poly 0* = *0*
⟨*proof*⟩

lemma *additive-poly-poly-pCons0*:
$a \neq 0 \Longrightarrow$ *additive-poly-poly* [:*a*:] = [:[:*a*:]:]
⟨*proof*⟩

lemma *additive-poly-poly-pCons*:
$p \neq 0 \Longrightarrow$
additive-poly-poly (*pCons a p*) =
smult (*pCons 0 1*) (*additive-poly-poly p*) + *pCons* [:*a*:] (*additive-poly-poly p*)
⟨*proof*⟩

lemma *additive-poly-poly*: *poly p* (*x* + *y*) = *poly* (*poly* (*additive-poly-poly p*) [:*x*:]) *y*
⟨*proof*⟩

lemma *additive-poly-poly-coeff0*: *coeff* (*additive-poly-poly p*) *0* = *p*
⟨*proof*⟩

lemma *additive-poly-poly-coeff1*: *coeff* (*additive-poly-poly p*) *1* = *polyderiv p*
⟨*proof*⟩

lemma *additive-poly-poly-eq0-iff*: *additive-poly-poly p* = *0* ⟷ *p* = *0*
⟨*proof*⟩

lemma *additive-poly-poly-degree*: *degree* (*additive-poly-poly p*) = *degree p*
⟨*proof*⟩

lemma *poly-additive-poly-poly-pCons*:
poly (*additive-poly-poly* (*pCons a p*)) [:*x*:] =
pCons a (*poly* (*additive-poly-poly p*) [:*x*:]) + *smult x* (*poly* (*additive-poly-poly p*) [:*x*:])
if $p \neq 0$
⟨*proof*⟩

lemma *poly-additive-poly-poly-eq0-iff*:
poly (*additive-poly-poly p*) [:*x*:] = *0* ⟷ *p* = *0*
⟨*proof*⟩

lemma *degree-poly-additive-poly-poly*:
  degree (*poly* (*additive-poly-poly* p) [:x:]) = *degree* p
⟨*proof*⟩

lemma *coeff-poly-additive-poly-poly*:
  coeff (*poly* (*additive-poly-poly* p) [:x:]) n = *poly* (*coeff* (*additive-poly-poly* p) n) x
⟨*proof*⟩

lemma *additive-poly-poly-decomp*:
  poly p $(x + y) = (\sum i{\le}degree\ p.\ (poly\ (coeff\ (additive\text{-}poly\text{-}poly\ p)\ i)\ x) * y \,\widehat{}\, i)$
  ⟨*proof*⟩

end

theory *Parametric-Equiv-Depth*

imports
  HOL.Rat
  HOL−Computational-Algebra.Fraction-Field
  HOL−Computational-Algebra.Primes
  HOL−Library.Function-Algebras
  HOL−Library.Set-Algebras
  HOL.Topological-Spaces
  Polynomial-Extras

begin

3 Common structures for adelic types

3.1 Preliminaries

lemma *ex-least-nat-le′*:
  fixes $P :: nat \Rightarrow bool$
  assumes $P\ n$
  shows $\exists k{\le}n.\ (\forall i{<}k.\ \neg\ P\ i) \wedge P\ k$
  ⟨*proof*⟩

lemma *ex-ex1-least-nat-le*:
  fixes $P :: nat \Rightarrow bool$
  assumes $\exists n.\ P\ n$
  shows $\exists!k.\ P\ k \wedge (\forall i.\ P\ i \longrightarrow k \le i)$
  ⟨*proof*⟩

lemma *least-le-least*:
  Least $Q \le$ *Least* P

if P : $\exists!x.\ P\ x \wedge (\forall\, y.\ P\ y \longrightarrow x \leq y)$
and Q : $\exists!x.\ Q\ x \wedge (\forall\, y.\ Q\ y \longrightarrow x \leq y)$
and PQ: $\forall\, x.\ P\ x \longrightarrow Q\ x$
for $P\ Q$ :: $'a$::*order* $\Rightarrow$ *bool*
$\langle proof \rangle$

lemma *linorder-wlog'* [*case-names le sym*]:
$[\![$
$\bigwedge a\ b.\ f\ a \leq f\ b \Longrightarrow P\ a\ b;$
$\bigwedge a\ b.\ P\ b\ a \Longrightarrow P\ a\ b$
$]\!] \Longrightarrow P\ a\ b$
for f :: $'a \Rightarrow\ 'b$::*linorder*
$\langle proof \rangle$

lemma *log-powi-cancel* [*simp*]:
$a > 0 \Longrightarrow a \neq 1 \Longrightarrow log\ a\ (a\ powi\ b) = b$
$\langle proof \rangle$

3.2 Existence of primes

class *factorial-comm-ring* = *factorial-semiring* + *comm-ring*
begin subclass *comm-ring-1* $\langle proof \rangle$ **end**

class *factorial-idom* = *factorial-semiring* + *idom*
begin subclass *factorial-comm-ring* $\langle proof \rangle$ **end**

class *nontrivial-factorial-semiring* = *factorial-semiring* +
assumes *nontrivial-elem-exists*: $\exists\, x.\ x \neq 0 \wedge normalize\ x \neq 1$
begin

lemma *primeE*:
obtains p **where** *prime* p
$\langle proof \rangle$

lemma *primes-exist*: $\exists\, p.\ prime\ p$
$\langle proof \rangle$

end

class *nontrivial-factorial-comm-ring* = *nontrivial-factorial-semiring* + *comm-ring*
begin
subclass *factorial-comm-ring* $\langle proof \rangle$
end

class *nontrivial-factorial-idom* = *nontrivial-factorial-semiring* + *idom*
begin
subclass *nontrivial-factorial-comm-ring* $\langle proof \rangle$
subclass *factorial-idom* $\langle proof \rangle$
end

instance *int* :: *nontrivial-factorial-idom*
⟨*proof*⟩

typedef (**overloaded**) *'a prime* =
{*p*::*'a*::*nontrivial-factorial-semiring*. *prime p*}
⟨*proof*⟩

lemma *Rep-prime-n0*: *Rep-prime p* ≠ *0*
⟨*proof*⟩

lemma *Rep-prime-not-unit*: ¬ *is-unit* (*Rep-prime p*)
⟨*proof*⟩

abbreviation *pdvd* ::
'a::*nontrivial-factorial-semiring prime* ⇒ *'a* ⇒ *bool* (**infix** *pdvd 50*)
where *p pdvd a* ≡ (*Rep-prime p*) *dvd a*
abbreviation *pmultiplicity p* ≡ *multiplicity* (*Rep-prime p*)

lemma *multiplicity-distinct-primes*:
p ≠ *q* ⟹ *pmultiplicity p* (*Rep-prime q*) = *0*
⟨*proof*⟩

3.3 Generic algebraic equality

context
fixes *R*
assumes *sympR* : *symp R*
and *transpR*: *transp R*
begin

lemma *transp-left*: *R x z* ⟹ *R y z* ⟹ *R x y*
⟨*proof*⟩

lemma *transp-right*: *R x y* ⟹ *R x z* ⟹ *R y z*
⟨*proof*⟩

lemma *transp-iffs*:
assumes *R x y* **shows** *R x z* ⟷ *R y z* **and** *R z x* ⟷ *R z y*
⟨*proof*⟩

lemma *transp-cong*: *R w z* **if** *R x w* **and** *R y z* **and** *R x y*
⟨*proof*⟩

lemma *transp-cong-sym*: *R x y* **if** *R x w* **and** *R y z* **and** *R w z*
⟨*proof*⟩

end

lemma *equivp-imp-symp*: *equivp R* $\Longrightarrow$ *symp R*
⟨*proof*⟩

locale *generic-alg-equality* =
fixes *alg-eq* :: *'a* ⇒ *'a* ⇒ *bool*
assumes *equivp*: *equivp alg-eq*
begin

lemmas *refl* = *equivp-reflp* [*OF equivp*]
and *sym* [*sym*] = *equivp-symp* [*OF equivp*]
and *trans* [*trans*] = *equivp-transp* [*OF equivp*]
and *trans-left* = *transp-left* [*OF equivp-imp-symp equivp-imp-transp*, *OF equivp equivp*]
and *trans-right* = *transp-right* [*OF equivp-imp-symp equivp-imp-transp*, *OF equivp equivp*]
and *trans-iffs* = *transp-iffs* [*OF equivp-imp-symp equivp-imp-transp*, *OF equivp equivp*]
and *cong* = *transp-cong* [*OF equivp-imp-symp equivp-imp-transp*, *OF equivp equivp*]
and *cong-sym* = *transp-cong-sym* [*OF equivp-imp-symp equivp-imp-transp*, *OF equivp equivp*]

lemma *sym-iff*: *alg-eq x y* = *alg-eq y x*
⟨*proof*⟩

end

locale *ab-group-alg-equality* = *generic-alg-equality alg-eq*
for *alg-eq* :: *'a*::*ab-group-add* ⇒ *'a* ⇒ *bool*
+
assumes *conv-0*: *alg-eq x y* $\longleftrightarrow$ *alg-eq* (*x* − *y*) *0*
begin

lemmas *trans0-iff* = *trans-iffs*(*1*)[*of - - 0*]

lemma *add-equiv0-left*: *alg-eq* (*x* + *y*) *y* **if** *alg-eq x 0*
⟨*proof*⟩

lemma *add-equiv0-right*: *alg-eq* (*x* + *y*) *x* **if** *alg-eq y 0*
⟨*proof*⟩

lemma *add*:
alg-eq (*x1* + *x2*) (*y1* + *y2*) **if** *alg-eq x1 y1* **and** *alg-eq x2 y2*
⟨*proof*⟩

lemma *add-left*: *alg-eq* (*x* + *y*) (*x* + *y'*) **if** *alg-eq y y'*
⟨*proof*⟩

lemma *add-right*: *alg-eq* $(x + y)$ $(x' + y)$ **if** *alg-eq* x x'
⟨*proof*⟩

lemma *add-0-left*: *alg-eq* $(x + y)$ y **if** *alg-eq* x *0*
⟨*proof*⟩

lemma *add-0-right*: *alg-eq* $(x + y)$ x **if** *alg-eq* y *0*
⟨*proof*⟩

lemma *sum-zeros*:
finite $A \Longrightarrow \forall a \in A.$ *alg-eq* $(f\ a)$ *0* $\Longrightarrow$ *alg-eq* (*sum* f A) *0*
⟨*proof*⟩

lemma *uminus*: *alg-eq* x $y \Longrightarrow$ *alg-eq* $(-x)$ $(-y)$
⟨*proof*⟩

lemma *uminus-iff*:
alg-eq x $y \longleftrightarrow$ *alg-eq* $(-x)$ $(-y)$
⟨*proof*⟩

lemma *add-0*: *alg-eq* $(x + y)$ *0* **if** *alg-eq* x *0* **and** *alg-eq* y *0*
⟨*proof*⟩

lemma *minus*:
alg-eq *x1* *y1* $\Longrightarrow$ *alg-eq* *x2* *y2* $\Longrightarrow$
alg-eq (*x1* $-$ *x2*) (*y1* $-$ *y2*)
⟨*proof*⟩

lemma *minus-0*: *alg-eq* $(x - y)$ *0* **if** *alg-eq* x *0* **and** *alg-eq* y *0*
⟨*proof*⟩

lemma *minus-left*: *alg-eq* $(x - y)$ $(x - y')$ **if** *alg-eq* y y'
⟨*proof*⟩

lemma *minus-right*: *alg-eq* $(x - y)$ $(x' - y)$ **if** *alg-eq* x x'
⟨*proof*⟩

lemma *minus-0-left*: *alg-eq* $(x - y)$ $(-y)$ **if** *alg-eq* x *0*
⟨*proof*⟩

lemma *minus-0-right*: *alg-eq* $(x - y)$ x **if** *alg-eq* y *0*
⟨*proof*⟩

end

locale *ring-alg-equality* = *ab-group-alg-equality alg-eq*
for *alg-eq* :: *'a::comm-ring* $\Rightarrow$ *'a* $\Rightarrow$ *bool*

+
assumes *mult-0-right*: *alg-eq y 0* $\Longrightarrow$ *alg-eq* $(x * y)$ *0*
begin

lemma *mult-0-left*: *alg-eq x 0* $\Longrightarrow$ *alg-eq* $(x * y)$ *0*
⟨*proof*⟩

lemma *mult*:
alg-eq x1 y1 $\Longrightarrow$ *alg-eq x2 y2* $\Longrightarrow$ *alg-eq* $(x1 * x2)$ $(y1 * y2)$
⟨*proof*⟩

lemma *mult-left*: *alg-eq y y'* $\Longrightarrow$ *alg-eq* $(x * y)$ $(x * y')$
⟨*proof*⟩

lemma *mult-right*: *alg-eq x x'* $\Longrightarrow$ *alg-eq* $(x * y)$ $(x' * y)$
⟨*proof*⟩

end

locale *ring-1-alg-equality* = *ring-alg-equality alg-eq*
for *alg-eq* :: *'a::comm-ring-1* ⇒ *'a* ⇒ *bool*
begin

lemma *mult-one-left*: *alg-eq x 1* $\Longrightarrow$ *alg-eq* $(x * y)$ *y*
⟨*proof*⟩

lemma *mult-one-right*: *alg-eq y 1* $\Longrightarrow$ *alg-eq* $(x * y)$ *x*
⟨*proof*⟩

lemma *prod-ones*:
finite A $\Longrightarrow$ ∀ *a*∈*A*. *alg-eq* (*f a*) *1* $\Longrightarrow$
alg-eq (*prod f A*) *1*
⟨*proof*⟩

lemma *prod-with-zero*:
alg-eq (*prod f* (*insert a A*)) *0* **if** *finite A* **and** *alg-eq* (*f a*) *0*
⟨*proof*⟩

lemma *pow*: *alg-eq* (*x* ^ *n*) (*y* ^ *n*) **if** *alg-eq x y*
⟨*proof*⟩

lemma *pow-equiv0*: *alg-eq* (*x* ^ *n*) *0* **if** *n* > *0* **and** *alg-eq x 0*
⟨*proof*⟩

end

3.4 Indexed equality

3.4.1 General structure

locale *p-equality* =
  fixes *p-equal* :: *'a ⇒ 'b::comm-ring ⇒ 'b ⇒ bool*
  assumes *p-alg-equality*: *ring-alg-equality (p-equal p)*
begin

lemmas *p-group-alg-equality* = *ring-alg-equality.axioms(1)[OF p-alg-equality]*
lemmas *p-generic-alg-equality* = *ab-group-alg-equality.axioms(1)[OF p-group-alg-equality]*

lemmas *equivp* = *generic-alg-equality.equivp [OF p-generic-alg-equality]*
  and *refl[simp]* = *generic-alg-equality.refl [OF p-generic-alg-equality]*
  and *sym [sym]* = *generic-alg-equality.sym [OF p-generic-alg-equality]*
  and *trans [trans]* = *generic-alg-equality.trans [OF p-generic-alg-equality]*
  and *sym-iff* = *generic-alg-equality.sym-iff [OF p-generic-alg-equality]*
  and *trans-left* = *generic-alg-equality.trans-left [OF p-generic-alg-equality]*
  and *trans-right* = *generic-alg-equality.trans-right [OF p-generic-alg-equality]*
  and *trans-iffs* = *generic-alg-equality.trans-iffs [OF p-generic-alg-equality]*
  and *cong* = *generic-alg-equality.cong [OF p-generic-alg-equality]*
  and *cong-sym* = *generic-alg-equality.cong-sym [OF p-generic-alg-equality]*

lemmas *conv-0* = *ab-group-alg-equality.conv-0 [OF p-group-alg-equality]*
  and *trans0-iff* = *ab-group-alg-equality.trans0-iff [OF p-group-alg-equality]*
  and *add-equiv0-left* = *ab-group-alg-equality.add-equiv0-left [OF p-group-alg-equality]*
  and *add-equiv0-right* = *ab-group-alg-equality.add-equiv0-right [OF p-group-alg-equality]*
  and *add* = *ab-group-alg-equality.add [OF p-group-alg-equality]*
  and *add-left* = *ab-group-alg-equality.add-left [OF p-group-alg-equality]*
  and *add-right* = *ab-group-alg-equality.add-right [OF p-group-alg-equality]*
  and *add-0-left* = *ab-group-alg-equality.add-0-left [OF p-group-alg-equality]*
  and *add-0-right* = *ab-group-alg-equality.add-0-right [OF p-group-alg-equality]*
  and *sum-zeros* = *ab-group-alg-equality.sum-zeros [OF p-group-alg-equality]*
  and *uminus* = *ab-group-alg-equality.uminus [OF p-group-alg-equality]*
  and *uminus-iff* = *ab-group-alg-equality.uminus-iff [OF p-group-alg-equality]*
  and *add-0* = *ab-group-alg-equality.add-0 [OF p-group-alg-equality]*
  and *minus* = *ab-group-alg-equality.minus [OF p-group-alg-equality]*
  and *minus-0* = *ab-group-alg-equality.minus-0 [OF p-group-alg-equality]*
  and *minus-left* = *ab-group-alg-equality.minus-left [OF p-group-alg-equality]*
  and *minus-right* = *ab-group-alg-equality.minus-right [OF p-group-alg-equality]*
  and *minus-0-left* = *ab-group-alg-equality.minus-0-left [OF p-group-alg-equality]*
  and *minus-0-right* = *ab-group-alg-equality.minus-0-right [OF p-group-alg-equality]*

lemmas *mult-0-left* = *ring-alg-equality.mult-0-left [OF p-alg-equality]*
  and *mult-0-right* = *ring-alg-equality.mult-0-right [OF p-alg-equality]*
  and *mult* = *ring-alg-equality.mult [OF p-alg-equality]*
  and *mult-left* = *ring-alg-equality.mult-left [OF p-alg-equality]*
  and *mult-right* = *ring-alg-equality.mult-right [OF p-alg-equality]*

definition *globally-p-equal* :: *'b ⇒ 'b ⇒ bool*

where *globally-p-equal-def*[*simp*]: *globally-p-equal x y* = ($\forall$ *p*::'*a*. *p-equal p x y*)

lemma *globally-p-equalI*[*intro*]: *globally-p-equal x y* **if** $\bigwedge$*p*::'*a*. *p-equal p x y*
⟨*proof*⟩

lemma *globally-p-equalD*: *globally-p-equal x y* $\Longrightarrow$ *p-equal p x y*
⟨*proof*⟩

lemma *globally-p-nequalE*:
assumes ¬ *globally-p-equal x y*
obtains *p* **where** ¬ *p-equal p x y*
⟨*proof*⟩

sublocale *globally-p-equality*: *ring-alg-equality globally-p-equal*
⟨*proof*⟩

lemmas *globally-p-equal-equiv p* = *globally-p-equality.equivp*
and *globally-p-equal-refl*[*simp*] = *globally-p-equality.refl*
and *globally-p-equal-sym* = *globally-p-equality.sym*
and *globally-p-equal-trans* = *globally-p-equality.trans*
and *globally-p-equal-trans-left* = *globally-p-equality.trans-left*
and *globally-p-equal-trans-right* = *globally-p-equality.trans-right*
and *globally-p-equal-trans-iffs* = *globally-p-equality.trans-iffs*
and *globally-p-equal-cong* = *globally-p-equality.cong*
and *globally-p-equal-conv-0* = *globally-p-equality.conv-0*
and *globally-p-equal-trans0-iff* = *globally-p-equality.trans0-iff*
and *globally-p-equal-add-equiv0-left* = *globally-p-equality.add-equiv0-left*
and *globally-p-equal-add-equiv0-right* = *globally-p-equality.add-equiv0-right*
and *globally-p-equal-add* = *globally-p-equality.add*
and *globally-p-equal-uminus* = *globally-p-equality.uminus*
and *globally-p-equal-uminus-iff* = *globally-p-equality.uminus-iff*
and *globally-p-equal-add-0* = *globally-p-equality.add-0*
and *globally-p-equal-minus* = *globally-p-equality.minus*
and *globally-p-equal-mult-0-left* = *globally-p-equality.mult-0-left*
and *globally-p-equal-mult-0-right* = *globally-p-equality.mult-0-right*
and *globally-p-equal-mult* = *globally-p-equality.mult*
and *globally-p-equal-mult-left* = *globally-p-equality.mult-left*
and *globally-p-equal-mult-right* = *globally-p-equality.mult-right*

lemma *globally-p-equal-transfer-equals-rsp*:
p-equal p x1 y1 $\longleftrightarrow$ *p-equal p x2 y2*
if *globally-p-equal x1 x2* **and** *globally-p-equal y1 y2*
⟨*proof*⟩

end

locale *p-equality-1* = *p-equality p-equal*
for *p-equal* :: '*a* $\Rightarrow$ '*b*::*comm-ring-1* $\Rightarrow$ '*b* $\Rightarrow$ *bool*

begin

lemma *p-alg-equality-1*: *ring-1-alg-equality* (*p-equal p*)
⟨*proof*⟩

lemmas *mult-one-left* = *ring-1-alg-equality.mult-one-left* [*OF p-alg-equality-1*]
and *mult-one-right* = *ring-1-alg-equality.mult-one-right* [*OF p-alg-equality-1*]
and *prod-ones* = *ring-1-alg-equality.prod-ones* [*OF p-alg-equality-1*]
and *pow* = *ring-1-alg-equality.pow* [*OF p-alg-equality-1*]
and *pow-equiv0* = *ring-1-alg-equality.pow-equiv0* [*OF p-alg-equality-1*]

end

locale *p-equality-no-zero-divisors* = *p-equality*
+
assumes *no-zero-divisors*:
¬ *p-equal p x 0* ⟹ ¬ *p-equal p y 0* ⟹
¬ *p-equal p* (*x* ∗ *y*) *0*
begin

lemma *mult-equiv-0-iff*:
p-equal p (*x* ∗ *y*) *0* ⟷
p-equal p x 0 ∨ *p-equal p y 0*
⟨*proof*⟩

end

locale *p-equality-no-zero-divisors-1* = *p-equality-no-zero-divisors p-equal*
for *p-equal* ::
′*a* ⇒ ′*b*::*comm-ring-1* ⇒ ′*b* ⇒ *bool*
+
assumes *nontrivial*: ∃ *x*. ¬ *p-equal p x 0*
begin

sublocale *p-equality-1* ⟨*proof*⟩

lemma *one-p-nequal-zero*: ¬ *p-equal p* (*1*::′*b*) (*0*::′*b*)
⟨*proof*⟩

lemma *zero-p-nequal-one*: ¬ *p-equal p* (*0*::′*b*) (*1*::′*b*)
⟨*proof*⟩

lemma *neg-one-p-nequal-zero*: ¬ *p-equal p* (−*1*::′*b*) (*0*::′*b*)
⟨*proof*⟩

lemma *pow-equiv-0-iff*: *p-equal p* (*x* ^ *n*) *0* ⟷ *p-equal p x 0* ∧ *n* ≠ *0*
⟨*proof*⟩

lemma *pow-equiv-base*: *p-equal p* $(x \mathbin{\widehat{}} n)$ $(y \mathbin{\widehat{}} n)$ **if** *p-equal p x y*
⟨*proof*⟩

end

locale *p-equality-div-inv* = *p-equality-no-zero-divisors-1 p-equal*
for *p-equal* ::
$'a \Rightarrow 'b$::{*comm-ring-1*, *inverse*, *divide-trivial*} ⇒
$'b \Rightarrow$ *bool*
\+
assumes *divide-inverse* : $\bigwedge x\ y :: 'b.\ x\ /\ y = x * \mathit{inverse}\ y$
and *inverse-inverse* : $\bigwedge x::'b.\ \mathit{inverse}\ (\mathit{inverse}\ x) = x$
and *inverse-mult* : $\bigwedge x\ y :: 'b.\ \mathit{inverse}\ (x * y) = \mathit{inverse}\ x * \mathit{inverse}\ y$
and *inverse-equiv0-iff*: *p-equal p* (*inverse x*) *0* ⟷ *p-equal p x 0*
and *divide-self-equiv* : ¬ *p-equal p x 0* ⟹ *p-equal p* $(x\ /\ x)$ *1*
begin

Global algebra facts

lemma *inverse-eq-divide*: *inverse* $x = 1\ /\ x$ **for** $x :: 'b$
⟨*proof*⟩

lemma *inverse-div-inverse*: *inverse x* / *inverse y* = $y\ /\ x$ **for** $x\ y :: 'b$
⟨*proof*⟩

lemma *inverse-0*[*simp*]: *inverse* $(0::'b) = 0$
⟨*proof*⟩

lemma *inverse-1*[*simp*]: *inverse* $(1::'b) = 1$
⟨*proof*⟩

lemma *inverse-pow*: *inverse* $(x \mathbin{\widehat{}} n)$ = *inverse* $x \mathbin{\widehat{}} n$ **for** $x :: 'b$
⟨*proof*⟩

lemma *power-int-minus*: *power-int x* $(-n)$ = *inverse* (*power-int x n*) **for** $x :: 'b$
⟨*proof*⟩

lemma *power-int-minus-divide*: *power-int x* $(-n) = 1\ /$ (*power-int x n*) **for** $x :: 'b$
⟨*proof*⟩

lemma *power-int-0-left-if*: *power-int* $(0 :: 'b)$ *m* = (*if* $m = 0$ *then 1 else 0*)
⟨*proof*⟩

lemma *power-int-0-left* [*simp*]: $m \neq 0 \Longrightarrow$ *power-int* $(0 :: 'b)$ $m = 0$
⟨*proof*⟩

lemma *power-int-1-left* [*simp*]: *power-int* $(1::'b)$ $n = 1$
⟨*proof*⟩

lemma *inverse-power-int*: *inverse* (*power-int* x n) = *power-int* x ($-n$) **for** x :: $'b$
⟨*proof*⟩

lemma *power-int-inverse*:
power-int (*inverse* x) n = *inverse* (*power-int* x n) **for** x :: $'b$
⟨*proof*⟩

lemma *power-int-mult-distrib*:
power-int ($x * y$) m = *power-int* x m $*$ *power-int* y m **for** x y :: $'b$
⟨*proof*⟩

lemma *inverse-divide*: *inverse* (a / b) = b / a **for** a b :: $'b$
⟨*proof*⟩

lemma *minus-divide-left*: $-$ (a / b) = ($-$ a) / b **for** a b :: $'b$
⟨*proof*⟩

lemma *times-divide-times-eq*: (a / b) $*$ (c / d) = ($a * c$) / ($b * d$) **for** a b :: $'b$
⟨*proof*⟩

lemma *divide-pow*: (x ^ n) / (y ^ n) = (x / y) ^ n **for** x y :: $'b$
⟨*proof*⟩

lemma *power-int-divide-distrib*:
power-int (x / y) m = *power-int* x m / *power-int* y m **for** x y :: $'b$
⟨*proof*⟩

lemma *power-int-one-over*: *power-int* (*1* / x) n = *1* / *power-int* x n **for** x :: $'b$
⟨*proof*⟩

lemma *add-divide-distrib*: ($a + b$) / c = (a / c) + (b / c) **for** a b c :: $'b$
⟨*proof*⟩

lemma *diff-divide-distrib*: ($a - b$) / c = (a / c) $-$ (b / c) **for** a b c :: $'b$
⟨*proof*⟩

lemma *times-divide-eq-right*: a $*$ (b / c) = ($a * b$) / c **for** a b c :: $'b$
⟨*proof*⟩

lemma *times-divide-eq-left*: (b / c) $*$ a = ($b * a$) / c **for** a b c :: $'b$
⟨*proof*⟩

lemma *divide-divide-eq-left*: (a / b) / c = a / ($b * c$) **for** a b c :: $'b$
⟨*proof*⟩

lemma *swap-numerator*: a $*$ (b / c) = (a / c) $*$ b **for** a b c :: $'b$
⟨*proof*⟩

lemma *divide-divide-times-eq*: $(a \ / \ b) \ / \ (c \ / \ d) = (a * d) \ / \ (b * c)$ **for** $a\ b\ c\ d$ :: $'b$
⟨*proof*⟩

Algebra facts by place

lemma *right-inverse-equiv*: *p-equal p* $(x * \textit{inverse } x)$ *1* **if** $\neg$ *p-equal p x 0*
⟨*proof*⟩

lemma *left-inverse-equiv*: *p-equal p* $(\textit{inverse } x * x)$ *1* **if** $\neg$ *p-equal p x 0*
⟨*proof*⟩

lemma *inverse-p-unique*:
p-equal p (*inverse x*) *y* **if** *p-equal p* $(x * y)$ *1*
⟨*proof*⟩

lemma *inverse-equiv-imp-equiv*:
p-equal p (*inverse a*) (*inverse b*) $\Longrightarrow$ *p-equal p a b*
⟨*proof*⟩

lemma *inverse-pcong*: *p-equal p* (*inverse x*) (*inverse y*) **if** *p-equal p x y*
⟨*proof*⟩

lemma *inverse-equiv-iff-equiv*:
p-equal p (*inverse a*) (*inverse b*) $\longleftrightarrow$ *p-equal p a b*
⟨*proof*⟩

lemma *power-int-equiv-0-iff*:
p-equal p (*power-int x n*) *0* $\longleftrightarrow$ *p-equal p x 0* $\wedge\ n \neq 0$
⟨*proof*⟩

lemma *power-diff-conv-inverse-equiv*:
$m \leq n \Longrightarrow$ *p-equal p* $(x \mathbin{\hat{}} (n - m))$ $(x \mathbin{\hat{}} n * \textit{inverse } x \mathbin{\hat{}} m)$
if $\neg$ *p-equal p x 0*
⟨*proof*⟩

lemma *power-int-add-1-equiv*:
p-equal p (*power-int x* $(m + 1)$) (*power-int x m* $* \ x$)
if $\neg$ *p-equal p x 0* $\vee\ m \neq -1$
⟨*proof*⟩

lemma *power-int-add-equiv*:
$\neg$ *p-equal p x 0* $\vee\ m + n \neq 0 \Longrightarrow$
p-equal p (*power-int x* $(m + n)$) (*power-int x m* $*$ *power-int x n*)
⟨*proof*⟩

lemma *power-int-diff-equiv*:
p-equal p (*power-int x* $(m - n)$) (*power-int x m* / *power-int x n*)
if $\neg$ *p-equal p x 0* $\vee\ m \neq n$
⟨*proof*⟩

lemma *power-int-minus-mult-equiv*:
$p\text{-}equal\ p\ (power\text{-}int\ x\ (n - 1) * x)\ (power\text{-}int\ x\ n)$
if $\neg\ p\text{-}equal\ p\ x\ 0 \lor n \neq 0$
⟨*proof*⟩

lemma *power-int-not-equiv-zero*:
$\neg\ p\text{-}equal\ p\ x\ 0 \lor n = 0 \Longrightarrow \neg\ p\text{-}equal\ p\ (power\text{-}int\ x\ n)\ 0$
⟨*proof*⟩

lemma *power-int-equiv-base*: $p\text{-}equal\ p\ (power\text{-}int\ x\ n)\ (power\text{-}int\ y\ n)$ **if** $p\text{-}equal\ p$
$x\ y$
⟨*proof*⟩

lemma *divide-equiv-0-iff*:
$p\text{-}equal\ p\ (x\ /\ y)\ 0 \longleftrightarrow p\text{-}equal\ p\ x\ 0 \lor p\text{-}equal\ p\ y\ 0$
⟨*proof*⟩

lemma *divide-pcong*: $p\text{-}equal\ p\ (w\ /\ x)\ (y\ /\ z)$ **if** $p\text{-}equal\ p\ w\ y$ **and** $p\text{-}equal\ p\ x\ z$
⟨*proof*⟩

lemma *divide-left-equiv*: $p\text{-}equal\ p\ x\ y \Longrightarrow p\text{-}equal\ p\ (x\ /\ z)\ (y\ /\ z)$
⟨*proof*⟩

lemma *divide-right-equiv*: $p\text{-}equal\ p\ x\ y \Longrightarrow p\text{-}equal\ p\ (z\ /\ x)\ (z\ /\ y)$
⟨*proof*⟩

lemma *add-frac-equiv*:
$p\text{-}equal\ p\ ((a\ /\ b) + (c\ /\ d))\ ((a * d + c * b)\ /\ (b * d))$
if $\neg\ p\text{-}equal\ p\ b\ 0$ **and** $\neg\ p\text{-}equal\ p\ d\ 0$
⟨*proof*⟩

lemma *mult-divide-mult-cancel-right*:
$p\text{-}equal\ p\ ((a * b)\ /\ (c * b))\ (a\ /\ c)$ **if** $\neg\ p\text{-}equal\ p\ b\ 0$
⟨*proof*⟩

lemma *mult-divide-mult-cancel-right2*:
$p\text{-}equal\ p\ ((a * c)\ /\ (c * b))\ (a\ /\ b)$ **if** $\neg\ p\text{-}equal\ p\ c\ 0$
⟨*proof*⟩

lemma *mult-divide-mult-cancel-left*:
$p\text{-}equal\ p\ ((c * a)\ /\ (c * b))\ (a\ /\ b)$ **if** $\neg\ p\text{-}equal\ p\ c\ 0$
⟨*proof*⟩

lemma *mult-divide-mult-cancel-left2*:
$p\text{-}equal\ p\ ((c * a)\ /\ (b * c))\ (a\ /\ b)$ **if** $\neg\ p\text{-}equal\ p\ c\ 0$
⟨*proof*⟩

lemma *mult-divide-cancel-right*: $p\text{-}equal\ p\ ((a * b)\ /\ b)\ a$ **if** $\neg\ p\text{-}equal\ p\ b\ 0$

⟨*proof*⟩

lemma *mult-divide-cancel-left*: *p-equal p* $((a * b) \ / \ a)$ *b* **if** ¬ *p-equal p a 0*
⟨*proof*⟩

lemma *divide-equiv-equiv*: (*p-equal p* $(b \ / \ c)$ *a*) = (*p-equal p b* $(a * c)$) **if** ¬ *p-equal p c 0*
⟨*proof*⟩

lemma *neg-divide-equiv-equiv*:
p-equal p $(- \ (b \ / \ c))$ *a* ⟷ *p-equal p* $(-b)$ $(a * c)$ **if** ¬ *p-equal p c 0*
⟨*proof*⟩

lemma *equiv-divide-imp*:
p-equal p $(a * c)$ *b* ⟹ *p-equal p a* $(b \ / \ c)$ **if** ¬ *p-equal p c 0*
⟨*proof*⟩

lemma *add-divide-equiv-iff*:
p-equal p $(x + y \ / \ z)$ $((x * z + y) \ / \ z)$ **if** ¬ *p-equal p z 0*
⟨*proof*⟩

lemma *divide-add-equiv-iff*: *p-equal p* $(x \ / \ z + y)$ $((x + y * z) \ / \ z)$ **if** ¬ *p-equal p z 0*
⟨*proof*⟩

lemma *diff-divide-equiv-iff*: *p-equal p* $(x - y \ / \ z)$ $((x * z - y) \ / \ z)$ **if** ¬ *p-equal p z 0*
⟨*proof*⟩

lemma *minus-divide-add-equiv-iff*:
p-equal p $(- \ (x \ / \ z) + y)$ $((- \ x + y * z) \ / \ z)$ **if** ¬ *p-equal p z 0*
⟨*proof*⟩

lemma *divide-diff-equiv-iff*: *p-equal p* $(x \ / \ z - y)$ $((x - y * z) \ / \ z)$ **if** ¬ *p-equal p z 0*
⟨*proof*⟩

lemma *minus-divide-diff-equiv-iff*:
p-equal p $(- \ (x \ / \ z) - y)$ $((- \ x - y * z) \ / \ z)$ **if** ¬ *p-equal p z 0*
⟨*proof*⟩

lemma *divide-mult-cancel-left*: *p-equal p* $(a \ / \ (a * b))$ $(1 \ / \ b)$ **if** ¬ *p-equal p a 0*
⟨*proof*⟩

lemma *divide-mult-cancel-right*: *p-equal p* $(b \ / \ (a * b))$ $(1 \ / \ a)$ **if** ¬ *p-equal p b 0*
⟨*proof*⟩

lemma *diff-frac-equiv*:
p-equal p $(x \ / \ y - w \ / \ z)$ $((x * z - w * y) \ / \ (y * z))$

if $\neg$ *p-equal p y 0* **and** $\neg$ *p-equal p z 0*
⟨*proof*⟩

lemma *frac-equiv-equiv*:
p-equal p $(x\ /\ y)$ $(w\ /\ z)$ $\longleftrightarrow$ *p-equal p* $(x * z)$ $(w * y)$
if $\neg$ *p-equal p y 0* **and** $\neg$ *p-equal p z 0*
⟨*proof*⟩

lemma *inverse-equiv-1-iff*:
p-equal p (*inverse x*) *1* $\longleftrightarrow$ *p-equal p x 1*
⟨*proof*⟩

lemma *inverse-neg-1-equiv*: *p-equal p* (*inverse* (-1)) (-1)
⟨*proof*⟩

lemma *globally-inverse-neg-1*: *globally-p-equal* (*inverse* (-1)) (-1)
⟨*proof*⟩

lemma *inverse-uminus-equiv*: *p-equal p* (*inverse* $(-x)$) ($-$ *inverse x*)
⟨*proof*⟩

lemma *minus-divide-right-equiv*: *p-equal p* $(a\ /\ (-\ b))$ $(-\ (a\ /\ b))$
⟨*proof*⟩

lemma *minus-divide-divide-equiv*: *p-equal p* $((-\ a)\ /\ (-\ b))$ $(a\ /\ b)$
⟨*proof*⟩

lemma *divide-minus1-equiv*: *p-equal p* $(x\ /\ (-1))$ $(-\ x)$
⟨*proof*⟩

lemma *divide-cancel-right*:
p-equal p $(a\ /\ c)$ $(b\ /\ c)$ $\longleftrightarrow$ *p-equal p c 0* $\vee$ *p-equal p a b*
⟨*proof*⟩

lemma *divide-cancel-left*:
p-equal p $(c\ /\ a)$ $(c\ /\ b)$ $\longleftrightarrow$ *p-equal p c 0* $\vee$ *p-equal p a b*
⟨*proof*⟩

lemma *divide-equiv-1-iff*:
p-equal p $(a\ /\ b)$ *1* $\longleftrightarrow$ $\neg$ *p-equal p b 0* $\wedge$ *p-equal p a b*
⟨*proof*⟩

lemma *divide-equiv-minus-1-iff*:
p-equal p $(a\ /\ b)$ $(-\ 1)$ $\longleftrightarrow$ $\neg$ *p-equal p b 0* $\wedge$ *p-equal p a* $(-\ b)$
⟨*proof*⟩

end

3.5 Indexed depth functions

3.5.1 General structure

locale *p-equality-depth* = *p-equality* +
  fixes *p-depth* :: $'a \Rightarrow 'b$::*comm-ring* $\Rightarrow$ *int*
  assumes *depth-of-0*[*simp*]: *p-depth p 0* = *0*
  and *depth-equiv*: *p-equal p x y* $\Longrightarrow$ *p-depth p x* = *p-depth p y*
  and *depth-uminus*: *p-depth p* $(-x)$ = *p-depth p x*
  and *depth-pre-nonarch*:
    $\neg$ *p-equal p x 0* $\Longrightarrow$ *p-depth p x* < *p-depth p y* $\Longrightarrow$
      p-depth p $(x + y)$ = *p-depth p x*
    $[\![$ $\neg$ *p-equal p x 0*; $\neg$ *p-equal p* $(x + y)$ *0*; *p-depth p x* = *p-depth p y* $]\!]$
      $\Longrightarrow$ *p-depth p* $(x + y) \geq$ *p-depth p x*
begin

lemma *depth-equiv-0*: *p-equal p x 0* $\Longrightarrow$ *p-depth p x* = *0*
  ⟨*proof*⟩

lemma *depth-equiv-uminus*: *p-equal p y* $(-x)$ $\Longrightarrow$ *p-depth p y* = *p-depth p x*
  ⟨*proof*⟩

lemma *depth-add-equiv0-left*: *p-equal p x 0* $\Longrightarrow$ *p-depth p* $(x + y)$ = *p-depth p y*
  ⟨*proof*⟩

lemma *depth-add-equiv0-right*: *p-equal p y 0* $\Longrightarrow$ *p-depth p* $(x + y)$ = *p-depth p x*
  ⟨*proof*⟩

lemma *depth-add-equiv*:
  p-depth p $(x + y)$ = *p-depth p* $(w + z)$ **if** *p-equal p x w* **and** *p-equal p y z*
  ⟨*proof*⟩

lemma *depth-diff*: *p-depth p* $(x - y)$ = *p-depth p* $(y - x)$
  ⟨*proof*⟩

lemma *depth-diff-equiv0-left*: *p-equal p x 0* $\Longrightarrow$ *p-depth p* $(x - y)$ = *p-depth p y*
  ⟨*proof*⟩

lemma *depth-diff-equiv0-right*: *p-equal p y 0* $\Longrightarrow$ *p-depth p* $(x - y)$ = *p-depth p x*
  ⟨*proof*⟩

lemma *depth-diff-equiv*:
  p-depth p $(x - y)$ = *p-depth p* $(w - z)$ **if** *p-equal p x w* **and** *p-equal p y z*
  ⟨*proof*⟩

lemma *depth-pre-nonarch-diff-left*:
  p-depth p $(x - y)$ = *p-depth p x* **if** $\neg$ *p-equal p x 0* **and** *p-depth p x* < *p-depth p*
y
  ⟨*proof*⟩

lemma *depth-pre-nonarch-diff-right*:
$p\text{-}depth\ p\ (x - y) = p\text{-}depth\ p\ y$ **if** $\neg\ p\text{-}equal\ p\ y\ 0$ **and** $p\text{-}depth\ p\ y < p\text{-}depth\ p\ x$
⟨*proof*⟩

lemma *depth-nonarch*:
$[\![\ \neg\ p\text{-}equal\ p\ x\ 0;\ \neg\ p\text{-}equal\ p\ y\ 0;\ \neg\ p\text{-}equal\ p\ (x + y)\ 0\ ]\!]$
$\Longrightarrow p\text{-}depth\ p\ (x + y) \geq min\ (p\text{-}depth\ p\ x)\ (p\text{-}depth\ p\ y)$
⟨*proof*⟩

lemma *depth-nonarch-diff*:
$[\![\ \neg\ p\text{-}equal\ p\ x\ 0;\ \neg\ p\text{-}equal\ p\ y\ 0;\ \neg\ p\text{-}equal\ p\ (x - y)\ 0\ ]\!]$
$\Longrightarrow p\text{-}depth\ p\ (x - y) \geq min\ (p\text{-}depth\ p\ x)\ (p\text{-}depth\ p\ y)$
⟨*proof*⟩

lemma *depth-sum-nonarch*:
$finite\ A \Longrightarrow \neg\ p\text{-}equal\ p\ (sum\ f\ A)\ 0 \Longrightarrow$
$p\text{-}depth\ p\ (sum\ f\ A) \geq Min\ \{p\text{-}depth\ p\ (f\ a)\ |\ a.\ a \in A \wedge \neg\ p\text{-}equal\ p\ (f\ a)\ 0\ \}$
⟨*proof*⟩

lemma *obtains-depth-sum-nonarch-witness*:
assumes $finite\ A$ **and** $\neg\ p\text{-}equal\ p\ (sum\ f\ A)\ 0$
obtains a
where $a \in A$ **and** $\neg\ p\text{-}equal\ p\ (f\ a)\ 0$ **and** $p\text{-}depth\ p\ (sum\ f\ A) \geq p\text{-}depth\ p\ (f\ a)$
⟨*proof*⟩

lemma *depth-add-cancel-imp-eq-depth*:
$p\text{-}depth\ p\ x = p\text{-}depth\ p\ y$ **if** $\neg\ p\text{-}equal\ p\ x\ 0$ **and** $p\text{-}depth\ p\ (x + y) > p\text{-}depth\ p\ x$
⟨*proof*⟩

lemma *depth-diff-cancel-imp-eq-depth-left*:
$p\text{-}depth\ p\ x = p\text{-}depth\ p\ y$ **if** $\neg\ p\text{-}equal\ p\ x\ 0$ **and** $p\text{-}depth\ p\ (x - y) > p\text{-}depth\ p\ x$
⟨*proof*⟩

lemma *depth-diff-cancel-imp-eq-depth-right*:
$p\text{-}depth\ p\ x = p\text{-}depth\ p\ y$ **if** $\neg\ p\text{-}equal\ p\ y\ 0$ **and** $p\text{-}depth\ p\ (x - y) > p\text{-}depth\ p\ y$
⟨*proof*⟩

lemma *level-closed-add*:
$p\text{-}depth\ p\ (x + y) \geq n$
if $\neg\ p\text{-}equal\ p\ (x + y)\ 0$ **and** $p\text{-}depth\ p\ x \geq n$ **and** $p\text{-}depth\ p\ y \geq n$
⟨*proof*⟩

lemma *level-closed-diff*:
$p\text{-}depth\ p\ (x - y) \geq n$

if $\neg$ *p-equal p* $(x - y)$ *0* **and** *p-depth p x* $\geq n$ **and** *p-depth p y* $\geq n$
$\langle proof \rangle$

definition *p-depth-set* :: $'a \Rightarrow int \Rightarrow 'b\ set$
where *p-depth-set p n* = $\{x.\ \neg$ *p-equal p x 0* $\longrightarrow$ *p-depth p x* $\geq n\}$

definition *global-depth-set* :: $int \Rightarrow 'b\ set$
where
global-depth-set n =
$\{x.\ \forall p.\ \neg$ *p-equal p x 0* $\longrightarrow$ *p-depth p x* $\geq n\}$

lemma *global-depth-set*: *global-depth-set n* = $(\bigcap p.$ *p-depth-set p n*$)$
$\langle proof \rangle$

lemma *p-depth-setD*:
$\neg$ *p-equal p x 0* $\Longrightarrow x \in$ *p-depth-set p n* $\Longrightarrow$
p-depth p x $\geq n$
$\langle proof \rangle$

lemma *nonpos-p-depth-setD*:
$n \leq 0 \Longrightarrow x \in$ *p-depth-set p n* $\Longrightarrow$ *p-depth p x* $\geq n$
$\langle proof \rangle$

lemma *global-depth-setD*:
$\neg$ *p-equal p x 0* $\Longrightarrow x \in$ *global-depth-set n* $\Longrightarrow$
p-depth p x $\geq n$
$\langle proof \rangle$

lemma *nonpos-global-depth-setD*:
$n \leq 0 \Longrightarrow x \in$ *global-depth-set n* $\Longrightarrow$ *p-depth p x* $\geq n$
$\langle proof \rangle$

lemma *p-depth-set-0*: $0 \in$ *p-depth-set p n*
$\langle proof \rangle$

lemma *global-depth-set-0*: $0 \in$ *global-depth-set n*
$\langle proof \rangle$

lemma *p-depth-set-equiv0*: *p-equal p x 0* $\Longrightarrow x \in$ *p-depth-set p n*
$\langle proof \rangle$

lemma *p-depth-setI*:
$x \in$ *p-depth-set p n* **if** $\neg$ *p-equal p x 0* $\longrightarrow$ *p-depth p x* $\geq n$
$\langle proof \rangle$

lemma *p-depth-set-by-equivI*:
$x \in$ *p-depth-set p n* **if** *p-equal p x y* **and** $y \in$ *p-depth-set p n*
$\langle proof \rangle$

lemma *global-depth-setI*:
$x \in$ *global-depth-set n*
if $\bigwedge p.\ \neg$ *p-equal p x 0* $\Longrightarrow$ *p-depth p x* $\geq n$
⟨*proof*⟩

lemma *global-depth-setI′*: $x \in$ *global-depth-set n* **if** $\bigwedge p.$ *p-depth p x* $\geq n$
⟨*proof*⟩

lemma *global-depth-setI′′*: $x \in$ *global-depth-set n* **if** $\bigwedge p.\ x \in$ *p-depth-set p n*
⟨*proof*⟩

lemma *p-depth-set-antimono*:
$n \leq m \Longrightarrow$ *p-depth-set p n* $\supseteq$ *p-depth-set p m*
⟨*proof*⟩

lemma *global-depth-set-antimono*:
$n \leq m \Longrightarrow$ *global-depth-set n* $\supseteq$ *global-depth-set m*
⟨*proof*⟩

lemma *p-depth-set-add*:
$x + y \in$ *p-depth-set p n* **if** $x \in$ *p-depth-set p n* **and** $y \in$ *p-depth-set p n*
⟨*proof*⟩

lemma *global-depth-set-add*:
$x + y \in$ *global-depth-set n* **if** $x \in$ *global-depth-set n* **and** $y \in$ *global-depth-set n*
⟨*proof*⟩

lemma *p-depth-set-uminus*: $-x \in$ *p-depth-set p n* **if** $x \in$ *p-depth-set p n*
⟨*proof*⟩

lemma *global-depth-set-uminus*: $-x \in$ *global-depth-set n* **if** $x \in$ *global-depth-set n*
⟨*proof*⟩

lemma *p-depth-set-minus*:
$x - y \in$ *p-depth-set p n* **if** $x \in$ *p-depth-set p n* **and** $y \in$ *p-depth-set p n*
⟨*proof*⟩

lemma *global-depth-set-minus*:
$x - y \in$ *global-depth-set n* **if** $x \in$ *global-depth-set n* **and** $y \in$ *global-depth-set n*
⟨*proof*⟩

lemma *p-depth-set-elt-set-plus*:
x *+o p-depth-set p n* $=$ *p-depth-set p n* **if** $x \in$ *p-depth-set p n*
⟨*proof*⟩

lemma *global-depth-set-elt-set-plus*:
x *+o global-depth-set n* $=$ *global-depth-set n* **if** $x \in$ *global-depth-set n*
⟨*proof*⟩

lemma *p-depth-set-elt-set-plus-closed*:
  x *+o* *p-depth-set p m* $\subseteq$ *p-depth-set p n* **if** $m \geq n$ **and** $x \in$ *p-depth-set p n*
  ⟨*proof*⟩

lemma *global-depth-set-elt-set-plus-closed*:
  x *+o* *global-depth-set m* $\subseteq$ *global-depth-set n*
  if $m \geq n$ **and** $x \in$ *global-depth-set n*
  ⟨*proof*⟩

lemma *p-depth-set-plus-coeffs*:
  set xs $\subseteq$ *p-depth-set p n* $\Longrightarrow$
    set ys $\subseteq$ *p-depth-set p n* $\Longrightarrow$
    set (*plus-coeffs xs ys*) $\subseteq$ *p-depth-set p n*
⟨*proof*⟩

lemma *global-depth-set-plus-coeffs*:
  set (*plus-coeffs xs ys*) $\subseteq$ *global-depth-set n*
  if *set xs* $\subseteq$ *global-depth-set n* **and** *set ys* $\subseteq$ *global-depth-set n*
  ⟨*proof*⟩

lemma *p-depth-set-poly-pCons*:
  set (*coeffs* (*pCons a f*)) $\subseteq$ *p-depth-set p n*
  if $a \in$ *p-depth-set p n* **and** *set* (*coeffs f*) $\subseteq$ *p-depth-set p n*
  ⟨*proof*⟩

lemma *p-depth-set-poly-add*:
  set (*coeffs* (f + g)) $\subseteq$ *p-depth-set p n*
  if *set* (*coeffs f*) $\subseteq$ *p-depth-set p n* **and** *set* (*coeffs g*) $\subseteq$ *p-depth-set p n*
  ⟨*proof*⟩

lemma *global-depth-set-poly-add*:
  set (*coeffs* (f + g)) $\subseteq$ *global-depth-set n*
  if *set* (*coeffs f*) $\subseteq$ *global-depth-set n*
  and *set* (*coeffs g*) $\subseteq$ *global-depth-set n*
  ⟨*proof*⟩

end

locale *p-equality-depth-no-zero-divisors* =
  p-equality-no-zero-divisors + *p-equality-depth*
+ **assumes** *depth-mult-additive*:
    $\neg$ *p-equal p* ($x * y$) *0* $\Longrightarrow$ *p-depth p* ($x * y$) = *p-depth p x* + *p-depth p y*
begin

lemma *depth-mult3-additive*:
  p-depth p ($x * y * z$) = *p-depth p x* + *p-depth p y* + *p-depth p z*
  if $\neg$ *p-equal p* ($x * y * z$) *0*
  ⟨*proof*⟩

lemma *p-depth-set-times*:
  $x * y \in$ *p-depth-set p n*
  if $n \geq 0$ **and** $x \in$ *p-depth-set p n* **and** $y \in$ *p-depth-set p n*
⟨*proof*⟩

lemma *global-depth-set-times*:
  $x * y \in$ *global-depth-set n*
  if $n \geq 0$ **and** $x \in$ *global-depth-set n* **and** $y \in$ *global-depth-set n*
  ⟨*proof*⟩

lemma *poly-p-depth-set-poly*:
  set (*coeffs f*) $\subseteq$ *p-depth-set p n* $\Longrightarrow$ *poly f x* $\in$ *p-depth-set p n*
  if $n \geq 0$ **and** $x \in$ *p-depth-set p n*
  ⟨*proof*⟩

lemma *poly-global-depth-set-poly*:
  poly f x $\in$ *global-depth-set n*
  if $n \geq 0$
  and $x \in$ *global-depth-set n*
  and *set* (*coeffs f*) $\subseteq$ *global-depth-set n*
  ⟨*proof*⟩

lemma *p-depth-set-ideal*:
  $x * y \in$ *p-depth-set p n*
  if $x \in$ *p-depth-set p 0* **and** $y \in$ *p-depth-set p n*
⟨*proof*⟩

lemma *global-depth-set-ideal*:
  $x * y \in$ *global-depth-set n*
  if $x \in$ *global-depth-set 0* **and** $y \in$ *global-depth-set n*
  ⟨*proof*⟩

lemma *poly-p-depth-set-poly-ideal*:
  set (*coeffs f*) $\subseteq$ *p-depth-set p n* $\Longrightarrow$ *poly f x* $\in$ *p-depth-set p n*
  if $x \in$ *p-depth-set p 0*
  ⟨*proof*⟩

lemma *poly-global-depth-set-poly-ideal*:
  poly f x $\in$ *global-depth-set n*
  if $x \in$ *global-depth-set 0* **and** *set* (*coeffs f*) $\subseteq$ *global-depth-set n*
  ⟨*proof*⟩

lemma *p-depth-set-poly-smult*:
  set (*coeffs* (*smult x f*)) $\subseteq$ *p-depth-set p n*
  if $n \geq 0$ **and** $x \in$ *p-depth-set p n* **and** *set* (*coeffs f*) $\subseteq$ *p-depth-set p n*
⟨*proof*⟩

lemma *global-depth-set-poly-smult*:

set (*coeffs* (*smult x f*)) ⊆ *global-depth-set n*
if *n* ≥ *0*
and *x* ∈ *global-depth-set n*
and *set* (*coeffs f*) ⊆ *global-depth-set n*
⟨*proof*⟩

lemma *p-depth-set-poly-smult-ideal*:
set (*coeffs* (*smult x f*)) ⊆ *p-depth-set p n*
if *x* ∈ *p-depth-set p 0* **and** *set* (*coeffs f*) ⊆ *p-depth-set p n*
⟨*proof*⟩

lemma *global-depth-set-poly-smult-ideal*:
set (*coeffs* (*smult x f*)) ⊆ *global-depth-set n*
if *x* ∈ *global-depth-set 0* **and** *set* (*coeffs f*) ⊆ *global-depth-set n*
⟨*proof*⟩

lemma *p-depth-set-poly-times*:
set (*coeffs f*) ⊆ *p-depth-set p n* ⟹
set (*coeffs g*) ⊆ *p-depth-set p n* ⟹
set (*coeffs* (*f* ∗ *g*)) ⊆ *p-depth-set p n*
if *n* ≥ *0*
⟨*proof*⟩

lemma *global-depth-set-poly-times*:
set (*coeffs* (*f* ∗ *g*)) ⊆ *global-depth-set n*
if *n* ≥ *0*
and *set* (*coeffs g*) ⊆ *global-depth-set n*
and *set* (*coeffs f*) ⊆ *global-depth-set n*
⟨*proof*⟩

lemma *p-depth-set-poly-times-ideal*:
set (*coeffs f*) ⊆ *p-depth-set p 0* ⟹
set (*coeffs g*) ⊆ *p-depth-set p n* ⟹
set (*coeffs* (*f* ∗ *g*)) ⊆ *p-depth-set p n*
⟨*proof*⟩

lemma *global-depth-set-poly-times-ideal*:
set (*coeffs* (*f* ∗ *g*)) ⊆ *global-depth-set n*
if *set* (*coeffs f*) ⊆ *global-depth-set 0*
and *set* (*coeffs g*) ⊆ *global-depth-set n*
⟨*proof*⟩

end

locale *p-equality-depth-no-zero-divisors-1* =
p-equality-no-zero-divisors-1 + *p-equality-depth-no-zero-divisors*
begin

lemma *depth-of-1* [*simp*]: *p-depth p 1 = 0*
⟨*proof*⟩

lemma *depth-of-neg1*: *p-depth p* (-1) *= 0*
⟨*proof*⟩

lemma *depth-pow-additive*: *p-depth p* $(x \mathbin{\hat{}} n)$ = *int n* $*$ *p-depth p x*
⟨*proof*⟩

lemma *depth-prod-summative*:
finite X $\Longrightarrow \neg$ *p-equal p* $(\prod X)$ *0* $\Longrightarrow$
p-depth p $(\prod X)$ = *sum* (*p-depth p*) *X*
⟨*proof*⟩

lemma *p-depth-set-1*: *1* $\in$ *p-depth-set p 0*
⟨*proof*⟩

lemma *pos-p-depth-set-1*: *n* $>$ *0* $\Longrightarrow$ *1* $\notin$ *p-depth-set p n*
⟨*proof*⟩

lemma *global-depth-set-1*: *1* $\in$ *global-depth-set 0*
⟨*proof*⟩

lemma *pos-global-depth-set-1*: *n* $>$ *0* $\Longrightarrow$ *1* $\notin$ *global-depth-set n*
⟨*proof*⟩

lemma *p-depth-set-neg1*: -1 $\in$ *p-depth-set p 0*
⟨*proof*⟩

lemma *global-depth-set-neg1*: -1 $\in$ *global-depth-set 0*
⟨*proof*⟩

lemma *p-depth-set-pow*:
$x \mathbin{\hat{}}$ *Suc k* $\in$ *p-depth-set p n* **if** *n* $\geq$ *0* **and** *x* $\in$ *p-depth-set p n*
⟨*proof*⟩

lemma *global-depth-set-pow*:
$x \mathbin{\hat{}}$ *Suc k* $\in$ *global-depth-set n* **if** *n* $\geq$ *0* **and** *x* $\in$ *global-depth-set n*
⟨*proof*⟩

lemma *p-depth-set-0-pow*: $x \mathbin{\hat{}} k$ $\in$ *p-depth-set p 0* **if** *x* $\in$ *p-depth-set p 0*
⟨*proof*⟩

lemma *global-depth-set-0-pow*: $x \mathbin{\hat{}} k$ $\in$ *global-depth-set 0* **if** *x* $\in$ *global-depth-set 0*
⟨*proof*⟩

lemma *p-depth-set-of-nat*: *of-nat n* $\in$ *p-depth-set p 0*
⟨*proof*⟩

lemma *global-depth-set-of-nat*: *of-nat n* $\in$ *global-depth-set 0*
⟨*proof*⟩

lemma *p-depth-set-of-int*: *of-int n* $\in$ *p-depth-set p 0*
⟨*proof*⟩

lemma *global-depth-set-of-int*: *of-int n* $\in$ *global-depth-set 0*
⟨*proof*⟩

lemma *p-depth-set-polyderiv*:
set (*coeffs f*) $\subseteq$ *p-depth-set p n* $\Longrightarrow$
set (*coeffs* (*polyderiv f*)) $\subseteq$ *p-depth-set p n*
⟨*proof*⟩

lemma *global-depth-set-polyderiv*:
set (*coeffs* (*polyderiv f*)) $\subseteq$ *global-depth-set n*
if *set* (*coeffs f*) $\subseteq$ *global-depth-set n*
⟨*proof*⟩

lemma *poly-p-depth-set-polyderiv*:
poly (*polyderiv f*) *x* $\in$ *p-depth-set p n*
if *n* $\geq$ *0* **and** *set* (*coeffs f*) $\subseteq$ *p-depth-set p n* **and** *x* $\in$ *p-depth-set p n*
⟨*proof*⟩

lemma *poly-global-depth-set-polyderiv*:
poly (*polyderiv f*) *x* $\in$ *global-depth-set n*
if *n* $\geq$ *0*
and *set* (*coeffs f*) $\subseteq$ *global-depth-set n*
and *x* $\in$ *global-depth-set n*
⟨*proof*⟩

lemma *p-set-depth-additive-poly-poly*:
set (*coeffs f*) $\subseteq$ *p-depth-set p m* $\Longrightarrow$
set (*coeffs* (*coeff* (*additive-poly-poly f*) *n*)) $\subseteq$ *p-depth-set p m*
if *m* $\geq$ *0*
⟨*proof*⟩

lemma *global-depth-set-additive-poly-poly*:
set (*coeffs* (*coeff* (*additive-poly-poly f*) *n*)) $\subseteq$ *global-depth-set m*
if *m* $\geq$ *0* **and** *set* (*coeffs f*) $\subseteq$ *global-depth-set m*
⟨*proof*⟩

lemma *poly-p-depth-set-additive-poly-poly*:
poly (*coeff* (*additive-poly-poly f*) *n*) *x* $\in$ *p-depth-set p m*
if *m* $\geq$ *0* **and** *set* (*coeffs f*) $\subseteq$ *p-depth-set p m* **and** *x* $\in$ *p-depth-set p m*
⟨*proof*⟩

lemma *poly-global-depth-set-additive-poly-poly*:
poly (*coeff* (*additive-poly-poly f*) *n*) *x* $\in$ *global-depth-set m*

if $m \geq 0$
and $set\ (coeffs\ f) \subseteq global\text{-}depth\text{-}set\ m$
and $x \in global\text{-}depth\text{-}set\ m$
$\langle proof \rangle$

lemma *sum-poly-additive-poly-poly-depth-bound*:
fixes $p :: 'a$ **and** $f :: 'b\ poly$ **and** $x\ y :: 'b$ **and** $a\ b\ n :: nat$
defines $g \equiv \lambda i.\ poly\ (coeff\ (additive\text{-}poly\text{-}poly\ f)\ (Suc\ i))\ x * y\ \hat{}\ (Suc\ i)$
defines $S \equiv sum\ g\ \{a..b\}$
assumes *coeffs*: $set\ (coeffs\ f) \subseteq p\text{-}depth\text{-}set\ p\ 0$
and *arg* : $x \in p\text{-}depth\text{-}set\ p\ 0$
and *sum* : $\neg\ p\text{-}equal\ p\ S\ 0$
obtains j **where** $j \in \{a..b\}$ **and** $p\text{-}depth\ p\ S \geq int\ (Suc\ j) * p\text{-}depth\ p\ y$
$\langle proof \rangle$

end

3.5.2 Depth of inverses and division

locale *p-equality-depth-div-inv* =
p-equality-div-inv + *p-equality-depth-no-zero-divisors-1*
begin

lemma *inverse-depth*: $p\text{-}depth\ p\ (inverse\ x) = -\ p\text{-}depth\ p\ x$
$\langle proof \rangle$

lemma *depth-powi-additive*: $p\text{-}depth\ p\ (power\text{-}int\ x\ n) = n * p\text{-}depth\ p\ x$
$\langle proof \rangle$

lemma *p-depth-set-inverse*: $inverse\ x \in p\text{-}depth\text{-}set\ p\ 0$ **if** $p\text{-}depth\ p\ x = 0$
$\langle proof \rangle$

lemma *divide-depth*: $p\text{-}depth\ p\ (x\ /\ y) = p\text{-}depth\ p\ x - p\text{-}depth\ p\ y$ **if** $\neg\ p\text{-}equal\ p\ (x\ /\ y)\ 0$
$\langle proof \rangle$

lemma *p-depth-set-divide*:
$x\ /\ y \in p\text{-}depth\text{-}set\ p\ n$ **if** $x \in p\text{-}depth\text{-}set\ p\ n$ **and** $p\text{-}depth\ p\ y = 0$
$\langle proof \rangle$

lemma *p-depth-poly-deriv-quotient*:
$p\text{-}depth\ p\ ((poly\ f\ a)\ /\ (poly\ (polyderiv\ f)\ a)) \geq n$
if $n \geq 0$
and $set\ (coeffs\ f) \subseteq p\text{-}depth\text{-}set\ p\ n$
and $a \in p\text{-}depth\text{-}set\ p\ n$
and $\neg\ p\text{-}equal\ p\ (poly\ f\ a)\ 0$
and $\neg\ p\text{-}equal\ p\ (poly\ (polyderiv\ f)\ a)\ 0$
and $p\text{-}depth\ p\ (poly\ f\ a) \geq 2 * p\text{-}depth\ p\ (poly\ (polyderiv\ f)\ a)$
$\langle proof \rangle$

lemma *p-depth-set-poly-deriv-quotient*:
  $(poly\ f\ a)\ /\ (poly\ (polyderiv\ f)\ a) \in p\text{-}depth\text{-}set\ p\ n$
  if $n \geq 0$
  and $set\ (coeffs\ f) \subseteq p\text{-}depth\text{-}set\ p\ n$
  and $a \in p\text{-}depth\text{-}set\ p\ n$
  and $\neg\ p\text{-}equal\ p\ (poly\ f\ a)\ 0$
  and $\neg\ p\text{-}equal\ p\ (poly\ (polyderiv\ f)\ a)\ 0$
  and $p\text{-}depth\ p\ (poly\ f\ a) \geq 2 * p\text{-}depth\ p\ (poly\ (polyderiv\ f)\ a)$
  ⟨*proof*⟩

end

3.6 Global equality and restriction of places

3.6.1 Locales

locale *global-p-equality* = *p-equality*
+
  fixes $p\text{-}restrict :: 'b::comm\text{-}ring \Rightarrow ('a \Rightarrow bool) \Rightarrow 'b$
  assumes *p-restrict-equiv* : $P\ p \Longrightarrow p\text{-}equal\ p\ (p\text{-}restrict\ x\ P)\ x$
  and *p-restrict-equiv0* : $\neg\ P\ p \Longrightarrow p\text{-}equal\ p\ (p\text{-}restrict\ x\ P)\ 0$
begin

lemma *p-restrict-equiv-sym*: $P\ p \Longrightarrow p\text{-}equal\ p\ x\ (p\text{-}restrict\ x\ P)$
  ⟨*proof*⟩

lemma *p-restrict-equiv0-iff*:
  $p\text{-}equal\ p\ (p\text{-}restrict\ x\ P)\ 0 \longleftrightarrow p\text{-}equal\ p\ x\ 0 \vee \neg\ P\ p$
  ⟨*proof*⟩

lemma *p-restrict-restrict-equiv-conj*:
  $p\text{-}equal\ p\ (p\text{-}restrict\ (p\text{-}restrict\ x\ P)\ Q)\ (p\text{-}restrict\ x\ (\lambda p.\ P\ p \wedge Q\ p))$
⟨*proof*⟩

lemma *p-restrict-restrict-equiv*: $p\text{-}equal\ p\ (p\text{-}restrict\ (p\text{-}restrict\ x\ P)\ P)\ (p\text{-}restrict\ x\ P)$
  ⟨*proof*⟩

lemma *p-restrict-0-equiv0*: $p\text{-}equal\ p\ (p\text{-}restrict\ 0\ P)\ 0$
  ⟨*proof*⟩

lemma *p-restrict-add-equiv*: $p\text{-}equal\ p\ (p\text{-}restrict\ (x + y)\ P)\ (p\text{-}restrict\ x\ P + p\text{-}restrict\ y\ P)$
  ⟨*proof*⟩

lemma *p-restrict-add-mixed-equiv*:
  $p\text{-}equal\ p\ (p\text{-}restrict\ x\ P + p\text{-}restrict\ y\ Q)\ (x + y)$ **if** $P\ p$ **and** $Q\ p$
  ⟨*proof*⟩

lemma *p-restrict-add-mixed-equiv-drop-right*:
p-equal p (*p-restrict x P* + *p-restrict y Q*) *x* **if** *P p* **and** ¬ *Q p*
⟨*proof*⟩

lemma *p-restrict-add-mixed-equiv-drop-left*:
p-equal p (*p-restrict x P* + *p-restrict y Q*) *y* **if** ¬ *P p* **and** *Q p*
⟨*proof*⟩

lemma *p-restrict-add-mixed-equiv-drop-both*:
p-equal p (*p-restrict x P* + *p-restrict y Q*) *0* **if** ¬ *P p* **and** ¬ *Q p*
⟨*proof*⟩

lemma *p-restrict-uminus-equiv*: *p-equal p* (*p-restrict* (−*x*) *P*) (− *p-restrict x P*)
⟨*proof*⟩

lemma *p-restrict-minus-equiv*: *p-equal p* (*p-restrict* (*x* − *y*) *P*) (*p-restrict x P* − *p-restrict y P*)
⟨*proof*⟩

lemma *p-restrict-minus-mixed-equiv*:
p-equal p (*p-restrict x P* − *p-restrict y Q*) (*x* − *y*) **if** *P p* **and** *Q p*
⟨*proof*⟩

lemma *p-restrict-minus-mixed-equiv-drop-right*:
p-equal p (*p-restrict x P* − *p-restrict y Q*) *x* **if** *P p* **and** ¬ *Q p*
⟨*proof*⟩

lemma *p-restrict-minus-mixed-equiv-drop-left*:
p-equal p (*p-restrict x P* − *p-restrict y Q*) (−*y*) **if** ¬ *P p* **and** *Q p*
⟨*proof*⟩

lemma *p-restrict-minus-mixed-equiv-drop-both*:
p-equal p (*p-restrict x P* − *p-restrict y Q*) *0* **if** ¬ *P p* **and** ¬ *Q p*
⟨*proof*⟩

lemma *p-restrict-times-equiv*:
p-equal p ((*p-restrict x P*) ∗ (*p-restrict y Q*))
(*p-restrict* (*x* ∗ *y*) (λ*p*. *P p* ∧ *Q p*))
⟨*proof*⟩

lemma *p-restrict-times-equiv*′:
p-equal p ((*p-restrict x P*) ∗ (*p-restrict y P*)) (*p-restrict* (*x* ∗ *y*) *P*)
⟨*proof*⟩

lemma *p-restrict-p-equalI*:
p-equal p (*p-restrict x P*) *y*
if *P p* ⟶ *p-equal p x y*
and ¬ *P p* ⟶ *p-equal p y 0*
⟨*proof*⟩

lemma *p-restrict-p-equal-p-restrictI*:
p-equal p (*p-restrict x P*) (*p-restrict y Q*)
if ($P\ p \wedge Q\ p$) $\longrightarrow$ *p-equal p x y*
and ($P\ p \wedge \neg\ Q\ p$) $\longrightarrow$ *p-equal p x 0*
and ($\neg\ P\ p \wedge Q\ p$) $\longrightarrow$ *p-equal p y 0*
⟨*proof*⟩

lemma *p-restrict-p-equal-p-restrict-by-sameI*:
p-equal p (*p-restrict x P*) (*p-restrict y P*) **if** *P p* $\longrightarrow$ *p-equal p x y*
⟨*proof*⟩

end

locale *global-p-equality-div-inv* =
p-equality-div-inv p-equal + *global-p-equality p-equal p-restrict*
for *p-equal* ::
$'a \Rightarrow 'b$::{*comm-ring-1*, *inverse*, *divide-trivial*} $\Rightarrow$
$'b \Rightarrow$ *bool*
and *p-restrict* :: $'b \Rightarrow ('a \Rightarrow$ *bool*$) \Rightarrow 'b$
begin

lemma *inverse-p-restrict-equiv*: *p-equal p* (*inverse* (*p-restrict x P*)) (*p-restrict* (*inverse x*) *P*)
⟨*proof*⟩

end

locale *global-p-equal* = *global-p-equality*
+ **assumes** *global-imp-eq*: *globally-p-equal x y* $\Longrightarrow x = y$
begin

lemma *global-eq-iff*: *globally-p-equal x y* $\longleftrightarrow x = y$
⟨*proof*⟩

lemma *p-restrict-true*[*simp*] : *p-restrict x* ($\lambda x.$ *True*) = *x*
⟨*proof*⟩

lemma *p-restrict-false*[*simp*]: *p-restrict x* ($\lambda x.$ *False*) = *0*
⟨*proof*⟩

lemma *p-restrict-restrict*:
p-restrict (*p-restrict x P*) *Q* = *p-restrict x* ($\lambda p.\ P\ p \wedge Q\ p$)
⟨*proof*⟩

lemma *p-restrict-restrict'*[*simp*]: *p-restrict* (*p-restrict x P*) *P* = *p-restrict x P*
⟨*proof*⟩

lemma *p-restrict-image-restrict*: *p-restrict x P* = *x* **if** $x \in (\lambda z.\ \textit{p-restrict } z\ P)$ ‘ *B*
⟨*proof*⟩

lemma *p-restrict-zero*: *p-restrict 0 P* = *0*
⟨*proof*⟩

lemma *p-restrict-add*: *p-restrict* (*x* + *y*) *P* = *p-restrict x P* + *p-restrict y P*
⟨*proof*⟩

lemma *p-restrict-uminus*: *p-restrict* (−*x*) *P* = − *p-restrict x P*
⟨*proof*⟩

lemma *p-restrict-minus*: *p-restrict* (*x* − *y*) *P* = *p-restrict x P* − *p-restrict y P*
⟨*proof*⟩

lemma *p-restrict-times*:
(*p-restrict x P*) ∗ (*p-restrict y Q*) = *p-restrict* (*x* ∗ *y*) (λ*p*. *P p* ∧ *Q p*)
⟨*proof*⟩

lemma *times-p-restrict*: *x* ∗ (*p-restrict y P*) = *p-restrict* (*x* ∗ *y*) *P*
⟨*proof*⟩

lemmas *p-restrict-pre-finite-adele-simps* =
p-restrict-zero p-restrict-add p-restrict-uminus p-restrict-minus p-restrict-times

lemma *p-equal-0-iff-p-restrict-eq-0*: *p-equal p x 0* ⟷ *p-restrict x* ((=) *p*) = *0*
⟨*proof*⟩

lemma *p-equal-iff-p-restrict-eq*:
p-equal p x y ⟷ *p-restrict x* ((=) *p*) = *p-restrict y* ((=) *p*)
⟨*proof*⟩

lemma *p-restrict-eqI*:
y = *p-restrict x P*
if *P*: ⋀*p*. *P p* ⟹ *p-equal p y x*
and *not-P*: ⋀*p*. ¬ *P p* ⟹ *p-equal p y 0*
⟨*proof*⟩

lemma *p-restrict-eq-p-restrict-mixedI*:
p-restrict x P = *p-restrict y Q*
if *both*: ⋀*p*::′*a*. *P p* ⟹ *Q p* ⟹ *p-equal p x y*
and *P* : ⋀*p*::′*a*. *P p* ⟹ ¬ *Q p* ⟹ *p-equal p x 0*
and *Q* : ⋀*p*::′*a*. ¬ *P p* ⟹ *Q p* ⟹ *p-equal p y 0*
⟨*proof*⟩

lemma *p-restrict-eq-p-restrictI*:
p-restrict x P = *p-restrict y P* **if** ⋀*p*::′*a*. *P p* ⟹ *p-equal p x y*
⟨*proof*⟩

lemma *p-restrict-eq-p-restrict-iff*:
  p-restrict x P = *p-restrict y P* $\longleftrightarrow$
    ($\forall$ *p*::'*a*. *P p* $\longrightarrow$ *p-equal p x y*)
  ⟨*proof*⟩

lemma *p-restrict-eq-zero-iff*:
  p-restrict x P = *0* $\longleftrightarrow$ ($\forall$ *p*. *P p* $\longrightarrow$ *p-equal p x 0*)
⟨*proof*⟩

lemma *p-restrict-decomp*:
  x = *p-restrict x P* + *p-restrict x* (λ*p*. $\neg$ *P p*)
⟨*proof*⟩

lemma *restrict-complement*:
  range (λ*x*. *p-restrict x P*) + *range* (λ*x*. *p-restrict x* (λ*p*. $\neg$ *P p*))
    = *UNIV*
  ⟨*proof*⟩

end

locale *global-p-equal-no-zero-divisors* =
  global-p-equal p-equal p-restrict + *p-equality-no-zero-divisors p-equal*
  for *p-equal* :: '*a* ⇒ '*b*::*comm-ring* ⇒ '*b* ⇒ *bool*
  and *p-restrict* :: '*b* ⇒ ('*a* ⇒ *bool*) ⇒ '*b*

locale *global-p-equal-no-zero-divisors-1* =
  global-p-equal-no-zero-divisors p-equal p-restrict + *p-equality-no-zero-divisors-1*
p-equal
  for *p-equal* :: '*a* ⇒ '*b*::*comm-ring-1* ⇒ '*b* ⇒ *bool*
  and *p-restrict* :: '*b* ⇒ ('*a* ⇒ *bool*) ⇒ '*b*
begin

lemma *pow-eq-0-iff*: *x* ^ *n* = *0* $\longleftrightarrow$ *x* = *0* $\wedge$ *n* $\neq$ *0* **for** *x* :: '*b*
  ⟨*proof*⟩

end

locale *global-p-equal-div-inv* =
  global-p-equality-div-inv p-equal p-restrict
+ *global-p-equal p-equal p-restrict*
  for *p-equal* ::
    '*a* ⇒ '*b*::{*comm-ring-1*, *inverse*, *divide-trivial*} ⇒
      '*b* ⇒ *bool*
  and *p-restrict* :: '*b* ⇒ ('*a* ⇒ *bool*) ⇒ '*b*
begin

lemma *power-int-eq-0-iff*:
power-int x n = *0* $\longleftrightarrow$ x = *0* $\wedge$ $n \neq$ *0* **for** x :: $'b$
⟨*proof*⟩

lemma *inverse-eq0-iff*: *inverse* x = *0* $\longleftrightarrow$ x = *0* **for** x :: $'b$
⟨*proof*⟩

lemma *inverse-neg-1* [*simp*]: *inverse* ($-$*1*::$'b$) = $-$*1*
⟨*proof*⟩

lemma *inverse-uminus*: *inverse* ($-x$) = $-$ *inverse* x **for** x :: $'b$
⟨*proof*⟩

lemma *global-right-inverse*: x $*$ *inverse* x = *1* **if** $\forall$ p. $\neg$ *p-equal* p x *0*
⟨*proof*⟩

lemma *right-inverse*: x $*$ *inverse* x = *p-restrict* *1* (λp. $\neg$ *p-equal* p x *0*)
⟨*proof*⟩

lemma *global-left-inverse*: *inverse* x $*$ x = *1* **if** $\forall$ p. $\neg$ *p-equal* p x *0*
⟨*proof*⟩

lemma *left-inverse*: *inverse* x $*$ x = *p-restrict* *1* (λp. $\neg$ *p-equal* p x *0*)
⟨*proof*⟩

lemma *inverse-unique*: *inverse* x = y **if** x $*$ y = *1* **for** x y :: $'b$
⟨*proof*⟩

lemma *inverse-p-restrict*: *inverse* (*p-restrict* x P) = *p-restrict* (*inverse* x) P
⟨*proof*⟩

lemma *power-diff-conv-inverse*:
x ^ (n $-$ m) = x ^ n $*$ *inverse* x ^ m **if** $m < n$
for x :: $'b$
⟨*proof*⟩

lemma *power-int-add-1*: *power-int* x (m + *1*) = *power-int* x m $*$ x **if** $m \neq -$*1* **for** x :: $'b$
⟨*proof*⟩

lemma *power-int-add*:
power-int x (m + n) = *power-int* x m $*$ *power-int* x n
if ($\forall$ p. $\neg$ *p-equal* p x *0*) $\vee$ $m + n \neq$ *0* **for** x :: $'b$
⟨*proof*⟩

lemma *power-int-diff*:
power-int x (m $-$ n) = *power-int* x m / *power-int* x n **if** $m \neq n$ **for** x :: $'b$
⟨*proof*⟩

lemma *power-int-minus-mult*: *power-int* x $(n - 1) * x$ = *power-int* x n **if** $n \neq 0$
for x :: $'b$
⟨*proof*⟩

lemma *power-int-not-eq-zero*:
$x \neq 0 \vee n = 0 \Longrightarrow$ *power-int* x $n \neq 0$ **for** x :: $'b$
⟨*proof*⟩

lemma *minus-divide-right*: $a / (- b) = - (a / b)$ **for** a b :: $'b$
⟨*proof*⟩

lemma *minus-divide-divide*: $(- a) / (- b) = a / b$ **for** a b :: $'b$
⟨*proof*⟩

lemma *divide-minus1* [*simp*]: $x / (-1) = - x$ **for** x :: $'b$
⟨*proof*⟩

lemma *divide-self*: x / x = *p-restrict* 1 (λp. $\neg$ *p-equal* p x 0)
⟨*proof*⟩

lemma *global-divide-self*: $x / x = 1$ **if** $\forall p$. $\neg$ *p-equal* p x 0
⟨*proof*⟩

lemma *global-mult-divide-mult-cancel-right*:
$(a * b) / (c * b) = a / c$ **if** $\forall p$. $\neg$ *p-equal* p b 0
⟨*proof*⟩

lemma *global-mult-divide-mult-cancel-left*:
$(c * a) / (c * b) = a / b$ **if** $\forall p$. $\neg$ *p-equal* p c 0
⟨*proof*⟩

lemma *divide-p-restrict-left*: (*p-restrict* x P) / y = *p-restrict* (x / y) P
⟨*proof*⟩

lemma *divide-p-restrict-right*: x / (*p-restrict* y P) = *p-restrict* (x / y) P
⟨*proof*⟩

lemma *inj-divide-right*:
inj (λb. b / a) $\longleftrightarrow$ ($\forall p$. $\neg$ *p-equal* p a 0)
⟨*proof*⟩

end

locale *global-p-equality-depth* = *global-p-equality* + *p-equality-depth*
begin

lemma *globally-p-equal-p-depth*: *globally-p-equal* x y $\Longrightarrow$ *p-depth* p x = *p-depth* p

y
  ⟨*proof*⟩

lemma *p-restrict-depth*:
  $P\ p \Longrightarrow p\text{-}depth\ p\ (p\text{-}restrict\ x\ P) = p\text{-}depth\ p\ x$
  $\neg\ P\ p \Longrightarrow p\text{-}depth\ p\ (p\text{-}restrict\ x\ P) = 0$
  ⟨*proof*⟩

lemma *p-restrict-add-mixed-depth-drop-right*:
  $p\text{-}depth\ p\ (p\text{-}restrict\ x\ P + p\text{-}restrict\ y\ Q) = p\text{-}depth\ p\ x$ **if** $P\ p$ **and** $\neg\ Q\ p$
  ⟨*proof*⟩

lemma *p-restrict-add-mixed-depth-drop-left*:
  $p\text{-}depth\ p\ (p\text{-}restrict\ x\ P + p\text{-}restrict\ y\ Q) = p\text{-}depth\ p\ y$ **if** $\neg\ P\ p$ **and** $Q\ p$
  ⟨*proof*⟩

lemma *p-depth-set-p-restrict*:
  $p\text{-}restrict\ x\ P \in p\text{-}depth\text{-}set\ p\ n$ **if** $P\ p \longrightarrow x \in p\text{-}depth\text{-}set\ p\ n$
⟨*proof*⟩

lemma *global-depth-set-p-restrict*:
  $p\text{-}restrict\ x\ P \in global\text{-}depth\text{-}set\ n$ **if** $x \in global\text{-}depth\text{-}set\ n$
  ⟨*proof*⟩

lemma *p-depth-set-closed-under-p-restrict-image*:
  $(\lambda x.\ p\text{-}restrict\ x\ P)$ ‘ $p\text{-}depth\text{-}set\ p\ n \subseteq p\text{-}depth\text{-}set\ p\ n$
  ⟨*proof*⟩

lemma *global-depth-set-closed-under-p-restrict-image*:
  $(\lambda x.\ p\text{-}restrict\ x\ P)$ ‘ $global\text{-}depth\text{-}set\ n \subseteq global\text{-}depth\text{-}set\ n$
  ⟨*proof*⟩

lemma *global-depth-set-p-restrictI*:
  $p\text{-}restrict\ x\ P \in global\text{-}depth\text{-}set\ n$
  if $\bigwedge p.\ P\ p \Longrightarrow x \in p\text{-}depth\text{-}set\ p\ n$
  ⟨*proof*⟩

lemma *p-depth-set-image-under-one-p-restrict-image*:
  $(\lambda x.\ p\text{-}restrict\ x\ ((=)\ p))$ ‘ $p\text{-}depth\text{-}set\ p\ n \subseteq global\text{-}depth\text{-}set\ n$
  ⟨*proof*⟩

end

locale *global-p-equality-depth-no-zero-divisors-1* =
  p-equality-depth-no-zero-divisors-1 p-equal p-depth
+ *global-p-equality-depth p-equal p-restrict p-depth*
  for *p-equal* ::
    $'a \Rightarrow 'b{::}comm\text{-}ring\text{-}1 \Rightarrow 'b \Rightarrow bool$

and *p-restrict* :: $'b \Rightarrow ('a \Rightarrow bool) \Rightarrow 'b$
and *p-depth* :: $'a \Rightarrow 'b \Rightarrow int$

locale *global-p-equality-depth-div-inv* =
p-equality-depth-div-inv + *global-p-equality-depth*
begin

sublocale *global-p-equality-div-inv* ⟨*proof*⟩

lemma *global-depth-set-inverse*:
p-restrict (*inverse x*) (λp. *p-depth p x* = *0*) ∈ *global-depth-set 0*
⟨*proof*⟩

lemma *global-depth-set-divide*:
p-restrict (*x* / *y*) (λp. *p-depth p y* = *0*) ∈ *global-depth-set n*
if *x* ∈ *global-depth-set n*
⟨*proof*⟩

lemma *global-depth-set-divideI*:
p-restrict (*x* / *y*) (λp. *p-depth p y* = *0*) ∈ *global-depth-set n*
if $\bigwedge p$. *p-depth p y* = *0* ⟹ *x* ∈ *p-depth-set p n*
⟨*proof*⟩

end

locale *global-p-equal-depth* = *global-p-equal* + *global-p-equality-depth*
begin

lemma *p-depth-set-vs-global-depth-set-image-under-one-p-restrict-image*:
(λx. *p-restrict x* ((=) *p*)) ‘ *p-depth-set p n* =
(λx. *p-restrict x* ((=) *p*)) ‘*global-depth-set n*
⟨*proof*⟩

end

locale *global-p-equal-depth-no-zero-divisors* =
global-p-equal-no-zero-divisors + *p-equality-depth-no-zero-divisors*
begin
sublocale *global-p-equal-depth* ⟨*proof*⟩
end

locale *global-p-equal-depth-div-inv* =
global-p-equality-depth-div-inv + *global-p-equal-depth-no-zero-divisors*
begin

sublocale *global-p-equal-div-inv* ⟨*proof*⟩

lemma *p-depth-set-eq-coset*:
p-depth-set p n = (*w powi n*) ∗*o* (*p-depth-set p 0*)
if *p-depth p w* = *1* **and** ∀ *p*. ¬ *p-equal p w 0*
⟨*proof*⟩

lemma *global-depth-set-eq-coset*:
global-depth-set n = (*w powi n*) ∗*o* (*global-depth-set 0*) **if** ∀ *p*. *p-depth p w* = *1*
⟨*proof*⟩

lemma *p-depth-set-eq-coset′*:
p-depth-set p 0 = (*w powi* (−*n*)) ∗*o* (*p-depth-set p n*)
if *p-depth p w* = *1* **and** ∀ *p*. ¬ *p-equal p w 0*
⟨*proof*⟩

lemma *global-depth-set-eq-coset′*:
global-depth-set 0 = (*w powi* (−*n*)) ∗*o* (*global-depth-set n*) **if** ∀ *p*. *p-depth p w* = *1*
⟨*proof*⟩

end

3.6.2 Embedding of base type constants

locale *global-p-equality-w-consts* = *global-p-equality p-equal p-restrict*
for *p-equal* ::
′a ⇒ *′b*::*comm-ring-1* ⇒ *′b* ⇒ *bool*
and *p-restrict* :: *′b* ⇒ (*′a* ⇒ *bool*) ⇒ *′b*
+
fixes *func-of-consts* :: (*′a* ⇒ *′c*::*ring-1*) ⇒ *′b*
assumes *consts-1* [*simp*]: *func-of-consts 1* = *1*
and *consts-diff* [*simp*]: *func-of-consts* (*f* − *g*) = *func-of-consts f* − *func-of-consts g*
and *consts-mult* [*simp*]:
func-of-consts (*f* ∗ *g*) = *func-of-consts f* ∗ *func-of-consts g*
begin

abbreviation *consts* ≡ *func-of-consts*
abbreviation *const a* ≡ *consts* (λ*p*. *a*)

lemma *consts-0*[*simp*]: *consts 0* = *0*
⟨*proof*⟩

lemma *const-0*[*simp*]: *const 0* = *0*
⟨*proof*⟩

lemma *const-1*[*simp*]: *const 1* = *1*
⟨*proof*⟩

lemma *consts-uminus* [*simp*]: *consts* (− *f*) = − *consts f*
⟨*proof*⟩

lemma *const-uminus* [*simp*]: *const* (−*a*) = − *const a*
⟨*proof*⟩

lemma *const-diff* [*simp*]: *const* (*a* − *b*) = *const a* − *const b*
⟨*proof*⟩

lemma *consts-add* [*simp*]: *consts* (*f* + *g*) = *consts f* + *consts g*
⟨*proof*⟩

lemma *const-add* [*simp*]: *const* (*a* + *b*) = *const a* + *const b*
⟨*proof*⟩

lemma *const-mult* [*simp*]: *const* (*a* ∗ *b*) = *const a* ∗ *const b*
⟨*proof*⟩

lemma *const-of-nat*: *const* (*of-nat n*) = *of-nat n*
⟨*proof*⟩

lemma *const-of-int*: *const* (*of-int z*) = *of-int z*
⟨*proof*⟩

end

locale *global-p-equal-w-consts* =
global-p-equal p-equal p-restrict
+ *global-p-equality-w-consts p-equal p-restrict func-of-consts*
for *p-equal* ::
′*a* ⇒ ′*b*::*comm-ring-1* ⇒ ′*b* ⇒ *bool*
and *p-restrict* :: ′*b* ⇒ (′*a* ⇒ *bool*) ⇒ ′*b*
and *func-of-consts* :: (′*a* ⇒ ′*c*::*ring-1*) ⇒ ′*b*

locale *global-p-equality-w-inj-consts* = *global-p-equality-w-consts*
+
assumes *p-equal-func-of-consts-0*: *p-equal p* (*consts f*) *0* ⟷ *f p* = *0*
begin

lemmas *p-equal-func-of-const-0* = *p-equal-func-of-consts-0*[*of* - *λp*. -]

lemma *p-equal-func-of-consts*:
p-equal p (*consts f*) (*consts g*) ⟷ *f p* = *g p*
⟨*proof*⟩

lemma *globally-p-equal-func-of-consts*:
globally-p-equal (*func-of-consts f*) (*func-of-consts g*) ⟷

$(\forall\, p.\ f\ p = g\ p)$
⟨*proof*⟩

lemma *const-p-equal*: $p\text{-}equal\ p\ (const\ a)\ (const\ b) \Longrightarrow a = b$
⟨*proof*⟩

end

locale *global-p-equality-w-inj-consts-char-0* =
global-p-equality-w-inj-consts p-equal p-restrict func-of-consts
for *p-equal* ::
$'a \Rightarrow 'b{::}comm\text{-}ring\text{-}1 \Rightarrow 'b \Rightarrow bool$
and *p-restrict* :: $'b \Rightarrow ('a \Rightarrow bool) \Rightarrow 'b$
and *func-of-consts* :: $('a \Rightarrow 'c{::}ring\text{-}char\text{-}0) \Rightarrow 'b$
begin

lemma *const-of-nat-p-equal-0-iff*: $p\text{-}equal\ p\ (of\text{-}nat\ n)\ 0 \longleftrightarrow n = 0$
⟨*proof*⟩

lemma *const-of-int-p-equal-0-iff*: $p\text{-}equal\ p\ (of\text{-}int\ z)\ 0 \longleftrightarrow z = 0$
⟨*proof*⟩

end

locale *global-p-equality-div-inv-w-inj-consts* =
p-equality-div-inv + *global-p-equality-w-inj-consts*
begin

lemma *consts-divide-self-equiv*: $p\text{-}equal\ p\ ((consts\ f)\ /\ (consts\ f))\ 1$ **if** $f\ p \neq 0$
⟨*proof*⟩

lemma *const-divide-self-equiv*: $p\text{-}equal\ p\ ((const\ c)\ /\ (const\ c))\ 1$ **if** $c \neq 0$
⟨*proof*⟩

end

locale *global-p-equal-w-inj-consts* =
global-p-equal p-equal p-restrict
+ *global-p-equality-w-inj-consts p-equal p-restrict*
for *p-equal* ::
$'a \Rightarrow 'b{::}comm\text{-}ring\text{-}1 \Rightarrow 'b \Rightarrow bool$
and *p-restrict* :: $'b \Rightarrow ('a \Rightarrow bool) \Rightarrow 'b$
begin

lemma *consts-eq-iff*: $consts\ f = consts\ g \longleftrightarrow f = g$
⟨*proof*⟩

lemma *consts-eq-0-iff*: $consts\ f = 0 \longleftrightarrow f = 0$
⟨*proof*⟩

lemma *inj-consts*: *inj consts*
⟨*proof*⟩

lemma *const-eq-iff*: $const\ a = const\ b \longleftrightarrow a = b$
⟨*proof*⟩

lemma *const-eq-0-iff*: $const\ a = 0 \longleftrightarrow a = 0$
⟨*proof*⟩

lemma *inj-const*: *inj const*
⟨*proof*⟩

end

locale *global-p-equal-div-inv-w-inj-consts* =
global-p-equal-div-inv + *global-p-equal-w-inj-consts*
begin

sublocale *global-p-equality-div-inv-w-inj-consts* ⟨*proof*⟩

lemma *consts-divide-self*:
$(consts\ f)\ /\ (consts\ f) = p\text{-}restrict\ 1\ (\lambda p{::}'a.\ f\ p \neq 0)$
⟨*proof*⟩

lemma *const-right-inverse* [*simp*]: $const\ a * inverse\ (const\ a) = 1$ **if** $a \neq 0$
⟨*proof*⟩

lemma *const-left-inverse* [*simp*]: $inverse\ (const\ a) * const\ a = 1$ **if** $a \neq 0$
⟨*proof*⟩

lemma *const-divide-self*: $(const\ c)\ /\ (const\ c) = 1$ **if** $c \neq 0$
⟨*proof*⟩

lemma *mult-const-divide-mult-const-cancel-right*:
$(y * const\ a)\ /\ (z * const\ a) = y\ /\ z$ **if** $a \neq 0$
⟨*proof*⟩

lemma *mult-const-divide-mult-const-cancel-left*:
$(const\ a * y)\ /\ (const\ a * z) = y\ /\ z$ **if** $a \neq 0$
⟨*proof*⟩

lemma *mult-const-divide-mult-const-cancel-left-right*:
$(const\ a * y)\ /\ (z * const\ a) = y\ /\ z$ **if** $a \neq 0$
⟨*proof*⟩

lemma *mult-const-divide-mult-const-cancel-right-left*:
$(y * const\ a) / (const\ a * z) = y / z$ **if** $a \neq 0$
⟨*proof*⟩

end

locale *global-p-equal-div-inv-w-inj-consts-char-0* =
global-p-equality-w-inj-consts-char-0 p-equal p-restrict func-of-consts
+ *global-p-equal-div-inv-w-inj-consts p-equal p-restrict func-of-consts*
for *p-equal* ::
$'a \Rightarrow 'b$::{*comm-ring-1*, *ring-char-0*, *inverse*, *divide-trivial*} ⇒
$'b \Rightarrow bool$
and *p-restrict* :: $'b \Rightarrow ('a \Rightarrow bool) \Rightarrow 'b$
and *func-of-consts* :: $('a \Rightarrow 'c$::*ring-char-0*$) \Rightarrow 'b$
begin

context
fixes n :: *nat*
assumes $n \neq 0$
begin

lemma *left-inverse-const-of-nat*[*simp*]: *inverse* (*of-nat* n :: $'b$) ∗ *of-nat* $n = 1$
⟨*proof*⟩

lemma *right-inverse-const-of-nat*[*simp*]: *of-nat* n ∗ *inverse* (*of-nat* n :: $'b$) = 1
⟨*proof*⟩

lemma *divide-self-const-of-nat*[*simp*]: (*of-nat* n :: $'b$) / *of-nat* $n = 1$
⟨*proof*⟩

lemma *mult-const-of-nat-divide-const-mult-of-nat-cancel-right*:
(a ∗ *of-nat* n) / (b ∗ *of-nat* n) = a / b **for** $a\ b$ :: $'b$
⟨*proof*⟩

lemma *mult-const-of-nat-divide-const-mult-of-nat-cancel-left*:
(*of-nat* $n * a$) / (*of-nat* $n * b$) = a / b **for** $a\ b$ :: $'b$
⟨*proof*⟩

end

context
fixes z :: *int*
assumes $z \neq 0$
begin

lemma *left-inverse-const-of-int*[*simp*]: *inverse* (*of-int* z :: $'b$) ∗ *of-int* $z = 1$
⟨*proof*⟩

lemma *right-inverse-const-of-int*[*simp*]: *of-int z* ∗ *inverse* (*of-int z* :: ′*b*) = *1*
⟨*proof*⟩

lemma *divide-self-const-of-int*[*simp*]: (*of-int z* :: ′*b*) / *of-int z* = *1*
⟨*proof*⟩

lemma *mult-const-of-int-divide-const-mult-of-int-cancel-right*:
(*a* ∗ *of-int z*) / (*b* ∗ *of-int z*) = *a* / *b* **for** *a b* :: ′*b*
⟨*proof*⟩

lemma *mult-const-of-int-divide-const-mult-of-int-cancel-left*:
(*of-int z* ∗ *a*) / (*of-int z* ∗ *b*) = *a* / *b* **for** *a b* :: ′*b*
⟨*proof*⟩

end

context
includes *rat.lifting*
begin

lift-definition *const-of-rat* :: *rat* ⇒ ′*b*
is λ*x*. *of-int* (*fst x*) / *of-int* (*snd x*)
⟨*proof*⟩

lemma *inj-const-of-rat*: *inj const-of-rat*
⟨*proof*⟩

lemma *const-of-rat-rat*: *const-of-rat* (*Rat.Fract a b*) = (*of-int a* :: ′*b*) / *of-int b*
⟨*proof*⟩

lemma *const-of-rat-0* [*simp*]: *const-of-rat 0* = *0*
⟨*proof*⟩

lemma *const-of-rat-1* [*simp*]: *const-of-rat 1* = *1*
⟨*proof*⟩

lemma *const-of-rat-minus*: *const-of-rat* (− *a*) = − *const-of-rat a*
⟨*proof*⟩

lemma *const-of-rat-add*: *const-of-rat* (*a* + *b*) = *const-of-rat a* + *const-of-rat b*
⟨*proof*⟩

lemma *const-of-rat-mult*: *const-of-rat* (*a* ∗ *b*) = *const-of-rat a* ∗ *const-of-rat b*
⟨*proof*⟩

lemma *const-of-rat-p-equal-0-iff*: *p-equal p* (*const-of-rat a*) *0* ⟷ *a* = *0*
⟨*proof*⟩

end

lemma *const-of-rat-eq-0-iff* [*simp*]: *const-of-rat a* = *0* $\longleftrightarrow$ *a* = *0*
⟨*proof*⟩

lemma *const-of-rat-neg-one* [*simp*]: *const-of-rat* (− *1*) = −*1*
⟨*proof*⟩

lemma *const-of-rat-diff*: *const-of-rat* (*a* − *b*) = *const-of-rat a* − *const-of-rat b*
⟨*proof*⟩

lemma *const-of-rat-sum*: *const-of-rat* ($\sum$ *a*∈*A*. *f a*) = ($\sum$ *a*∈*A*. *const-of-rat* (*f a*))
⟨*proof*⟩

lemma *const-of-rat-prod*: *const-of-rat* ($\prod$ *a*∈*A*. *f a*) = ($\prod$ *a*∈*A*. *const-of-rat* (*f a*))
⟨*proof*⟩

lemma *const-of-rat-inverse*: *const-of-rat* (*inverse a*) = *inverse* (*const-of-rat a*)
⟨*proof*⟩

lemma *left-inverse-const-of-rat*[*simp*]:
inverse (*const-of-rat a*) ∗ *const-of-rat a* = *1* **if** *a* ≠ *0*
⟨*proof*⟩

lemma *right-inverse-const-of-rat*[*simp*]:
const-of-rat a ∗ *inverse* (*const-of-rat a*) = *1* **if** *a* ≠ *0*
⟨*proof*⟩

lemma *const-of-rat-divide*: *const-of-rat* (*a* / *b*) = *const-of-rat a* / *const-of-rat b*
⟨*proof*⟩

lemma *divide-self-const-of-rat*[*simp*]: (*const-of-rat a*) / (*const-of-rat a*) = *1* **if** *a* ≠ *0*
⟨*proof*⟩

lemma *const-of-rat-power*: *const-of-rat* (*a* ^ *n*) = (*const-of-rat a*) ^ *n*
⟨*proof*⟩

lemma *const-of-rat-eq-iff* [*simp*]: *const-of-rat a* = *const-of-rat b* $\longleftrightarrow$ *a* = *b*
⟨*proof*⟩

lemma *zero-eq-const-of-rat-iff* [*simp*]: *0* = *const-of-rat a* $\longleftrightarrow$ *0* = *a*
⟨*proof*⟩

lemma *const-of-rat-eq-1-iff* [*simp*]: *const-of-rat a* = *1* $\longleftrightarrow$ *a* = *1*
⟨*proof*⟩

lemma *one-eq-const-of-rat-iff* [*simp*]: *1* = *const-of-rat a* $\longleftrightarrow$ *1* = *a*
⟨*proof*⟩

Collapse nested embeddings.

lemma *const-of-rat-of-nat-eq* [*simp*]: *const-of-rat* (*of-nat n*) = *of-nat n*
  ⟨*proof*⟩

lemma *const-of-rat-of-int-eq* [*simp*]: *const-of-rat* (*of-int z*) = *of-int z*
  ⟨*proof*⟩

Product of all p-adic embeddings of the field of rationals.

definition *range-const-of-rat* :: *'b set* ($\mathbb{Q}_{\forall\, p}$)
  where $\mathbb{Q}_{\forall\, p}$ = *range const-of-rat*

lemma *range-const-of-rat-cases* [*cases set*: *range-const-of-rat*]:
  assumes $q \in \mathbb{Q}_{\forall\, p}$
  obtains (*const-of-rat*) *r* **where** *q* = *const-of-rat r*
⟨*proof*⟩

lemma *range-const-of-rat-cases'*:
  assumes $x \in \mathbb{Q}_{\forall\, p}$
  obtains *a b* **where** *b* > *0 coprime a b x* = *of-int a* / *of-int b*
⟨*proof*⟩

lemma *range-const-of-rat-of-rat* [*simp*]: *const-of-rat r* $\in \mathbb{Q}_{\forall\, p}$
  ⟨*proof*⟩

lemma *range-const-of-rat-of-int* [*simp*]: *of-int z* $\in \mathbb{Q}_{\forall\, p}$
  ⟨*proof*⟩

lemma *Ints-subset-range-const-of-rat*: $\mathbb{Z} \subseteq \mathbb{Q}_{\forall\, p}$
  ⟨*proof*⟩

lemma *range-const-of-rat-of-nat* [*simp*]: *of-nat n* $\in \mathbb{Q}_{\forall\, p}$
  ⟨*proof*⟩

lemma *Nats-subset-range-const-of-rat*: $\mathbb{N} \subseteq \mathbb{Q}_{\forall\, p}$
  ⟨*proof*⟩

lemma *range-const-of-rat-0* [*simp*]: *0* $\in \mathbb{Q}_{\forall\, p}$
  ⟨*proof*⟩

lemma *range-const-of-rat-1* [*simp*]: *1* $\in \mathbb{Q}_{\forall\, p}$
  ⟨*proof*⟩

lemma *range-const-of-rat-add* [*simp*]:
  $a \in \mathbb{Q}_{\forall\, p} \Longrightarrow$
    $b \in \mathbb{Q}_{\forall\, p} \Longrightarrow$
    $a + b \in \mathbb{Q}_{\forall\, p}$
  ⟨*proof*⟩

lemma *range-const-of-rat-minus-iff* [*simp*]:

$- \ a \in \mathbb{Q}_{\forall\ p} \longleftrightarrow$
$a \in \mathbb{Q}_{\forall\ p}$
⟨*proof*⟩

lemma *range-const-of-rat-diff* [*simp*]:
$a \in \mathbb{Q}_{\forall\ p} \Longrightarrow$
$b \in \mathbb{Q}_{\forall\ p} \Longrightarrow$
$a - b \in \mathbb{Q}_{\forall\ p}$
⟨*proof*⟩

lemma *range-const-of-rat-mult* [*simp*]:
$a \in \mathbb{Q}_{\forall\ p} \Longrightarrow$
$b \in \mathbb{Q}_{\forall\ p} \Longrightarrow$
$a * b \in \mathbb{Q}_{\forall\ p}$
⟨*proof*⟩

lemma *range-const-of-rat-inverse* [*simp*]:
$a \in \mathbb{Q}_{\forall\ p} \Longrightarrow$
$\mathit{inverse}\ a \in \mathbb{Q}_{\forall\ p}$
⟨*proof*⟩

lemma *range-const-of-rat-divide* [*simp*]:
$a \in \mathbb{Q}_{\forall\ p} \Longrightarrow$
$b \in \mathbb{Q}_{\forall\ p} \Longrightarrow$
$a\ /\ b \in \mathbb{Q}_{\forall\ p}$
⟨*proof*⟩

lemma *range-const-of-rat-power* [*simp*]:
$a \in \mathbb{Q}_{\forall\ p} \Longrightarrow$
$a\ \widehat{}\ n \in \mathbb{Q}_{\forall\ p}$
⟨*proof*⟩

lemma *range-const-of-rat-sum* [*intro*]:
$(\bigwedge x.\ x \in A \Longrightarrow f\ x \in \mathbb{Q}_{\forall\ p}) \Longrightarrow$
$\mathit{sum}\ f\ A \in \mathbb{Q}_{\forall\ p}$
⟨*proof*⟩

lemma *range-const-of-rat-prod* [*intro*]:
$(\bigwedge x.\ x \in A \Longrightarrow f\ x \in \mathbb{Q}_{\forall\ p})$
$\Longrightarrow \mathit{prod}\ f\ A \in \mathbb{Q}_{\forall\ p}$
⟨*proof*⟩

lemma *range-const-of-rat-induct* [*case-names const-of-rat*, *induct set*: *range-const-of-rat*]:
$q \in \mathbb{Q}_{\forall\ p} \Longrightarrow$
$(\bigwedge r.\ P\ (\mathit{const\text{-}of\text{-}rat}\ r)) \Longrightarrow P\ q$
⟨*proof*⟩

lemma *p-adic-Rats-p-equal-0-iff*:
$a \in \mathbb{Q}_{\forall\ p} \Longrightarrow$

p-equal p a 0 $\longleftrightarrow$ *a* = *0*
⟨*proof*⟩

lemma *range-const-of-rat-infinite*: ¬ *finite* $\mathbb{Q}_{\forall\, p}$
⟨*proof*⟩

lemma *left-inverse-range-const-of-rat*:
$a \in \mathbb{Q}_{\forall\, p} \Longrightarrow a \neq 0 \Longrightarrow$
inverse $a * a = 1$
⟨*proof*⟩

lemma *right-inverse-range-const-of-rat*:
$a \in \mathbb{Q}_{\forall\, p} \Longrightarrow a \neq 0 \Longrightarrow$
$a *$ *inverse* $a = 1$
⟨*proof*⟩

lemma *divide-self-range-const-of-rat*:
$a \in \mathbb{Q}_{\forall\, p} \Longrightarrow a \neq 0 \Longrightarrow$
$a \mathbin{/} a = 1$
⟨*proof*⟩

lemma *range-const-of-rat-add-iff*:
$a \in \mathbb{Q}_{\forall\, p} \vee$
$b \in \mathbb{Q}_{\forall\, p} \Longrightarrow$
$a + b \in \mathbb{Q}_{\forall\, p} \longleftrightarrow$
$a \in \mathbb{Q}_{\forall\, p} \wedge b \in \mathbb{Q}_{\forall\, p}$
⟨*proof*⟩

lemma *range-const-of-rat-diff-iff*:
$a \in \mathbb{Q}_{\forall\, p} \vee$
$b \in \mathbb{Q}_{\forall\, p} \Longrightarrow$
$a - b \in \mathbb{Q}_{\forall\, p} \longleftrightarrow$
$a \in \mathbb{Q}_{\forall\, p} \wedge b \in \mathbb{Q}_{\forall\, p}$
⟨*proof*⟩

lemma *range-const-of-rat-mult-iff*:
$a * b \in \mathbb{Q}_{\forall\, p} \longleftrightarrow$
$a \in \mathbb{Q}_{\forall\, p} \wedge b \in \mathbb{Q}_{\forall\, p}$
if
$a \in \mathbb{Q}_{\forall\, p} - \{0\} \vee$
$b \in \mathbb{Q}_{\forall\, p} - \{0\}$
⟨*proof*⟩

lemma *range-const-of-rat-inverse-iff* [*simp*]:
inverse $a \in \mathbb{Q}_{\forall\, p} \longleftrightarrow$
$a \in \mathbb{Q}_{\forall\, p}$
⟨*proof*⟩

lemma *range-const-of-rat-divide-iff*:

$a\ /\ b \in \mathbb{Q}_{\forall\ p} \longleftrightarrow$
$a \in \mathbb{Q}_{\forall\ p} \wedge b \in \mathbb{Q}_{\forall\ p}$
if
$a \in \mathbb{Q}_{\forall\ p} - \{0\} \vee$
$b \in \mathbb{Q}_{\forall\ p} - \{0\}$
⟨*proof*⟩

end

lemma *Fract-eq0-iff*: *Fraction-Field.Fract a b* = *0* ⟷ *a* = *0* ∨ *b* = *0*
⟨*proof*⟩

locale *global-p-equal-div-inv-w-inj-idom-consts* =
global-p-equal-div-inv-w-inj-consts p-equal p-restrict func-of-consts
\+ *global-p-equality-w-inj-consts p-equal p-restrict func-of-consts*
for *p-equal* ::
′*a* ⇒ ′*b*::{*comm-ring-1*, *inverse*, *divide-trivial*} ⇒
′*b* ⇒ *bool*
and *p-restrict* :: ′*b* ⇒ (′*a* ⇒ *bool*) ⇒ ′*b*
and *func-of-consts* :: (′*a* ⇒ ′*c*::*idom*) ⇒ ′*b*
begin

lift-definition *const-of-fract* :: ′*c fract* ⇒ ′*b*
is λ*x*::′*c* × ′*c*. *const* (*fst x*) / *const* (*snd x*)
⟨*proof*⟩

lemma *inj-const-of-fract*: *inj const-of-fract*
⟨*proof*⟩

lemma *const-of-fract-0* [*simp*]: *const-of-fract 0* = *0*
⟨*proof*⟩

lemma *const-of-fract-1* [*simp*]: *const-of-fract 1* = *1*
⟨*proof*⟩

lemma *const-of-fract-Fract*: *const-of-fract* (*Fraction-Field.Fract a b*) = *const a* / *const b*
⟨*proof*⟩

lemma *const-of-fract-p-eq-0-iff* [*simp*]: *p-equal p* (*const-of-fract x*) *0* ⟷ *x* = *0*
⟨*proof*⟩

lemma *const-of-fract-equal-0-iff* [*simp*]: *const-of-fract x* = *0* ⟷ *x* = *0*
⟨*proof*⟩

lemma *const-of-fract-minus*: *const-of-fract* (− *a*) = − *const-of-fract a*
⟨*proof*⟩

lemma *const-of-fract-add*: *const-of-fract* (*a* + *b*) = *const-of-fract a* + *const-of-fract*

b
⟨*proof*⟩

lemma *const-of-fract-mult*: $\textit{const-of-fract}\ (a * b) = \textit{const-of-fract}\ a * \textit{const-of-fract}\ b$
⟨*proof*⟩

lemma *const-of-fract-neg-one* [*simp*]: $\textit{const-of-fract}\ (-\ 1) = -1$
⟨*proof*⟩

lemma *const-of-fract-diff*: $\textit{const-of-fract}\ (a - b) = \textit{const-of-fract}\ a - \textit{const-of-fract}\ b$
⟨*proof*⟩

lemma *const-of-fract-sum*:
$\textit{const-of-fract}\ (\sum a{\in}A.\ f\ a) = (\sum a{\in}A.\ \textit{const-of-fract}\ (f\ a))$
⟨*proof*⟩

lemma *const-of-fract-prod*:
$\textit{const-of-fract}\ (\prod a{\in}A.\ f\ a) = (\prod a{\in}A.\ \textit{const-of-fract}\ (f\ a))$
⟨*proof*⟩

lemma *const-of-fract-inverse*: $\textit{const-of-fract}\ (\textit{inverse}\ a) = \textit{inverse}\ (\textit{const-of-fract}\ a)$
⟨*proof*⟩

lemma *left-inverse-const-of-fract*[*simp*]:
$\textit{inverse}\ (\textit{const-of-fract}\ a) * \textit{const-of-fract}\ a = 1$ **if** $a \neq 0$
⟨*proof*⟩

lemma *right-inverse-const-of-fract*[*simp*]:
$(\textit{const-of-fract}\ a) * \textit{inverse}\ (\textit{const-of-fract}\ a) = 1$ **if** $a \neq 0$
⟨*proof*⟩

lemma *const-of-fract-divide*: $\textit{const-of-fract}\ (a\ /\ b) = \textit{const-of-fract}\ a\ /\ \textit{const-of-fract}\ b$
⟨*proof*⟩

lemma *p-adic-prod-divide-self-of-fract*[*simp*]:
$(\textit{const-of-fract}\ a)\ /\ (\textit{const-of-fract}\ a) = 1$ **if** $a \neq 0$
⟨*proof*⟩

lemma *const-of-fract-power*: $\textit{const-of-fract}\ (a\ \hat{}\ n) = (\textit{const-of-fract}\ a)\ \hat{}\ n$
⟨*proof*⟩

lemma *const-of-fract-eq-iff* [*simp*]:
$\textit{const-of-fract}\ a = \textit{const-of-fract}\ b \longleftrightarrow a = b$
⟨*proof*⟩

lemma *zero-eq-const-of-fract-iff* [*simp*]: $0 = \textit{const-of-fract } a \longleftrightarrow 0 = a$
  ⟨*proof*⟩

lemma *const-of-fract-eq-1-iff* [*simp*]: $\textit{const-of-fract } a = 1 \longleftrightarrow a = 1$
  ⟨*proof*⟩

lemma *one-eq-const-of-fract-iff* [*simp*]: $1 = \textit{const-of-fract } a \longleftrightarrow 1 = a$
  ⟨*proof*⟩

Collapse nested embeddings.

lemma *const-of-fract-numer-eq-const* [*simp*]: *const-of-fract* (*Fraction-Field.Fract a 1*) = *const a*
  ⟨*proof*⟩

lemma *const-of-fract-of-nat-eq* [*simp*]: *const-of-fract* (*of-nat n*) = *of-nat n*
  ⟨*proof*⟩

lemma *const-of-fract-of-int-eq* [*simp*]: *const-of-fract* (*of-int z*) = *of-int z*
  ⟨*proof*⟩

Product of all p-adic embeddings of the field of fractions of the base ring.

definition *range-const-of-fract* :: *'b set* ($\mathcal{Q}_{\forall\, p}$)
  where $\mathcal{Q}_{\forall\, p}$ = *range const-of-fract*

lemma *range-const-of-fract-cases* [*cases set*: *range-const-of-fract*]:
  assumes $q \in \mathcal{Q}_{\forall\, p}$
  obtains (*const-of-fract*) *r* **where** *q* = *const-of-fract r*
⟨*proof*⟩

lemma *range-const-of-fract-cases'*:
  assumes $x \in \mathcal{Q}_{\forall\, p}$
  obtains *a b* **where** $b \neq 0$ *x* = *const a* / *const b*
⟨*proof*⟩

lemma *range-const-of-fract-of-fract* [*simp*]:
  const-of-fract r $\in \mathcal{Q}_{\forall\, p}$
  ⟨*proof*⟩

lemma *range-const-of-fract-const* [*simp*]: *const a* $\in \mathcal{Q}_{\forall\, p}$
  ⟨*proof*⟩

lemma *range-const-of-fract-of-int* [*simp*]: *of-int z* $\in \mathcal{Q}_{\forall\, p}$
  ⟨*proof*⟩

lemma *Ints-subset-range-const-of-fract*: $\mathbb{Z} \subseteq \mathcal{Q}_{\forall\, p}$
  ⟨*proof*⟩

lemma *range-const-of-fract-of-nat* [*simp*]: *of-nat n* $\in \mathcal{Q}_{\forall\, p}$
  ⟨*proof*⟩

lemma *Nats-subset-range-const-of-fract*: $\mathbb{N} \subseteq \mathcal{Q}_{\forall\ p}$
  ⟨*proof*⟩

lemma *range-const-of-fract-0* [*simp*]: $0 \in \mathcal{Q}_{\forall\ p}$
  ⟨*proof*⟩

lemma *range-const-of-fract-1* [*simp*]: $1 \in \mathcal{Q}_{\forall\ p}$
  ⟨*proof*⟩

lemma *range-const-of-fract-add* [*simp*]:
  $a \in \mathcal{Q}_{\forall\ p} \Longrightarrow$
    $b \in \mathcal{Q}_{\forall\ p} \Longrightarrow$
    $a + b \in \mathcal{Q}_{\forall\ p}$
  ⟨*proof*⟩

lemma *range-const-of-fract-minus-iff* [*simp*]:
  $- a \in \mathcal{Q}_{\forall\ p} \longleftrightarrow$
    $a \in \mathcal{Q}_{\forall\ p}$
  ⟨*proof*⟩

lemma *range-const-of-fract-diff* [*simp*]:
  $a \in \mathcal{Q}_{\forall\ p} \Longrightarrow$
    $b \in \mathcal{Q}_{\forall\ p} \Longrightarrow$
    $a - b \in \mathcal{Q}_{\forall\ p}$
  ⟨*proof*⟩

lemma *range-const-of-fract-mult* [*simp*]:
  $a \in \mathcal{Q}_{\forall\ p} \Longrightarrow$
    $b \in \mathcal{Q}_{\forall\ p} \Longrightarrow$
    $a * b \in \mathcal{Q}_{\forall\ p}$
  ⟨*proof*⟩

lemma *range-const-of-fract-inverse* [*simp*]:
  $a \in \mathcal{Q}_{\forall\ p} \Longrightarrow$
    inverse $a \in \mathcal{Q}_{\forall\ p}$
  ⟨*proof*⟩

lemma *range-const-of-fract-divide* [*simp*]:
  $a \in \mathcal{Q}_{\forall\ p} \Longrightarrow$
    $b \in \mathcal{Q}_{\forall\ p} \Longrightarrow$
    $a\ /\ b \in \mathcal{Q}_{\forall\ p}$
  ⟨*proof*⟩

lemma *range-const-of-fract-power* [*simp*]:
  $a \in \mathcal{Q}_{\forall\ p} \Longrightarrow$
    $a \mathbin{\widehat{}} n \in \mathcal{Q}_{\forall\ p}$
  ⟨*proof*⟩

lemma *range-const-of-fract-sum* [*intro*]:
$(\bigwedge x.\ x \in A \Longrightarrow$
$f\ x \in \mathcal{Q}_{\forall\ p}) \Longrightarrow$
$sum\ f\ A \in \mathcal{Q}_{\forall\ p}$
⟨*proof*⟩

lemma *range-const-of-fract* [*intro*]:
$(\bigwedge x.\ x \in A \Longrightarrow f\ x \in \mathcal{Q}_{\forall\ p})$
$\Longrightarrow prod\ f\ A \in \mathcal{Q}_{\forall\ p}$
⟨*proof*⟩

lemma *range-const-of-fract-induct* [*case-names const-of-fract*, *induct set*: *range-const-of-fract*]:
$q \in \mathcal{Q}_{\forall\ p} \Longrightarrow$
$(\bigwedge r.\ P\ (const\text{-}of\text{-}fract\ r)) \Longrightarrow P\ q$
⟨*proof*⟩

lemma *p-adic-Fracts-p-equal-0-iff*:
$a \in \mathcal{Q}_{\forall\ p} \Longrightarrow$
$p\text{-}equal\ p\ a\ 0 \longleftrightarrow a = 0$
⟨*proof*⟩

lemma *range-const-of-fract-infinite*:
infinite $\mathcal{Q}_{\forall\ p}$ **if** *infinite* $(UNIV :: 'c\ set)$
⟨*proof*⟩

lemma *left-inverse-range-const-of-fract*:
$a \in \mathcal{Q}_{\forall\ p} \Longrightarrow a \neq 0 \Longrightarrow$
$inverse\ a * a = 1$
⟨*proof*⟩

lemma *right-inverse-range-const-of-fract*:
$a \in \mathcal{Q}_{\forall\ p} \Longrightarrow a \neq 0 \Longrightarrow$
$a * inverse\ a = 1$
⟨*proof*⟩

lemma *divide-self-range-const-of-fract*:
$a \in \mathcal{Q}_{\forall\ p} \Longrightarrow a \neq 0 \Longrightarrow$
$a\ /\ a = 1$
⟨*proof*⟩

lemma *range-const-of-fract-add-iff*:
$a \in \mathcal{Q}_{\forall\ p} \vee$
$b \in \mathcal{Q}_{\forall\ p} \Longrightarrow$
$a + b \in \mathcal{Q}_{\forall\ p} \longleftrightarrow$
$a \in \mathcal{Q}_{\forall\ p} \wedge b \in \mathcal{Q}_{\forall\ p}$
⟨*proof*⟩

lemma *range-const-of-fract-diff-iff*:
$a \in \mathcal{Q}_{\forall\ p} \vee$

$b \in \mathcal{Q}_{\forall\, p} \Longrightarrow$
$a - b \in \mathcal{Q}_{\forall\, p} \longleftrightarrow$
$a \in \mathcal{Q}_{\forall\, p} \wedge b \in \mathcal{Q}_{\forall\, p}$
⟨*proof*⟩

lemma *range-const-of-fract-mult-iff*:
$a * b \in \mathcal{Q}_{\forall\, p} \longleftrightarrow$
$a \in \mathcal{Q}_{\forall\, p} \wedge b \in \mathcal{Q}_{\forall\, p}$
if
$a \in \mathcal{Q}_{\forall\, p} - \{0\} \vee$
$b \in \mathcal{Q}_{\forall\, p} - \{0\}$
⟨*proof*⟩

lemma *range-const-of-fract-inverse-iff* [*simp*]:
inverse $a \in \mathcal{Q}_{\forall\, p} \longleftrightarrow$
$a \in \mathcal{Q}_{\forall\, p}$
⟨*proof*⟩

lemma *range-const-of-fract-divide-iff*:
$a\ /\ b \in \mathcal{Q}_{\forall\, p} \longleftrightarrow$
$a \in \mathcal{Q}_{\forall\, p} \wedge b \in \mathcal{Q}_{\forall\, p}$
if
$a \in \mathcal{Q}_{\forall\, p} - \{0\} \vee$
$b \in \mathcal{Q}_{\forall\, p} - \{0\}$
⟨*proof*⟩

end

locale *global-p-equality-w-inj-index-consts* = *global-p-equality-w-inj-consts*
+
fixes *index-inj* :: $'a \Rightarrow 'c$
assumes *index-inj* : *inj index-inj*
and *index-inj-nzero*: *index-inj p* $\neq$ *0*
begin

abbreviation *global-uniformizer* $\equiv$ *consts index-inj*
abbreviation *index-const p* $\equiv$ *const* (*index-inj p*)

lemma *global-uniformizer-locally-uniformizes*: *p-equal p global-uniformizer* (*index-const p*)
⟨*proof*⟩

lemma *index-const-globally-nequiv-0*: ¬ *p-equal q* (*index-const p*) *0*
⟨*proof*⟩

lemma *global-uniformizer-globally-nequiv-0*: ¬ *p-equal p global-uniformizer 0*
⟨*proof*⟩

end

locale *global-p-equality-no-zero-divisors-1-w-inj-index-consts* =
global-p-equality-w-inj-index-consts + *p-equality-no-zero-divisors-1*
begin

lemma *index-const-pow-globally-nequiv-0*: ¬ *p-equal q* (*index-const p* ^ *n*) *0*
⟨*proof*⟩

lemma *global-uniformizer-pow-globally-nequiv-0*: ¬ *p-equal q* (*global-uniformizer* ^ *n*) *0*
⟨*proof*⟩

end

locale *global-p-equality-depth-w-inj-index-consts* =
p-equality-depth p-equal p-depth
+ *global-p-equality-w-inj-index-consts p-equal p-restrict func-of-consts index-inj*
for *p-equal* :: *'a* ⇒ *'b*::*comm-ring-1* ⇒ *'b* ⇒ *bool*
and *p-restrict* :: *'b* ⇒ (*'a* ⇒ *bool*) ⇒ *'b*
and *p-depth* :: *'a* ⇒ *'b* ⇒ *int*
and *func-of-consts* :: (*'a* ⇒ *'c*::{*factorial-semiring*,*ring-1*}) ⇒ *'b*
and *index-inj* :: *'a* ⇒ *'c*
+
assumes *p-depth-func-of-consts*:
p-depth p (*func-of-consts f*) = *int* (*multiplicity* (*index-inj p*) (*f p*))
and *index-inj-nunit* : ¬ *is-unit* (*index-inj p*)
and *index-inj-coprime*:
p ≠ *q* ⟹ *multiplicity* (*index-inj p*) (*index-inj q*) = *0*
begin

lemma *p-depth-set-consts*: *func-of-consts f* ∈ *p-depth-set p 0*
⟨*proof*⟩

lemma *global-depth-set-consts*: *func-of-consts f* ∈ *global-depth-set 0*
⟨*proof*⟩

lemma *p-depth-1*[*simp*]: *p-depth p* (*1*::*'b*) = *0*
⟨*proof*⟩

lemma *p-depth-index-const*: *p-depth p* (*index-const p*) = *1*
⟨*proof*⟩

lemma *p-depth-index-const'*: *p-depth q* (*index-const p*) = *of-bool* (*q* = *p*)
⟨*proof*⟩

lemma *p-depth-global-uniformizer*: *p-depth p* (*global-uniformizer*::*'b*) = *1*
⟨*proof*⟩

lemma *p-depth-consts-func-ge0*: *p-depth p* (*consts f*) ≥ *0*

⟨*proof*⟩

end

locale *global-p-equal-depth-no-zero-divisors-w-inj-index-consts* =
global-p-equal p-equal p-restrict
\+ *global-p-equality-depth-no-zero-divisors-1 p-equal p-restrict p-depth*
\+ *global-p-equality-depth-w-inj-index-consts p-equal p-restrict p-depth func-of-consts index-inj*
for *p-equal* :: *'a* ⇒ *'b*::*comm-ring-1* ⇒ *'b* ⇒ *bool*
and *p-restrict* :: *'b* ⇒ (*'a* ⇒ *bool*) ⇒ *'b*
and *p-depth* :: *'a* ⇒ *'b* ⇒ *int*
and *func-of-consts* :: (*'a* ⇒ *'c*::{*factorial-semiring*,*ring-1*}) ⇒ *'b*
and *index-inj* :: *'a* ⇒ *'c*
begin

sublocale *global-p-equal-w-inj-consts* ⟨*proof*⟩
sublocale *global-p-equal-depth-no-zero-divisors* ⟨*proof*⟩

lemma *p-depth-index-const-pow*: *p-depth p* ((*index-const p*) ^ *n*) = *int n*
⟨*proof*⟩

lemma *p-depth-index-const-pow'*: *p-depth q* ((*index-const p*) ^ *n*) = *int n* ∗ *of-bool* (*q* = *p*)
⟨*proof*⟩

lemma *p-depth-global-uniformizer-pow*: *p-depth p* (*global-uniformizer* ^ *n*) = *int n*
⟨*proof*⟩

end

locale *global-p-equal-depth-div-inv-w-inj-index-consts* =
global-p-equal-depth-div-inv p-equal p-depth p-restrict
\+ *global-p-equal-depth-no-zero-divisors-w-inj-index-consts*
p-equal p-restrict p-depth func-of-consts index-inj
for *p-equal* ::
'a ⇒ *'b*::{*comm-ring-1*, *inverse*, *divide-trivial*} ⇒
'b ⇒ *bool*
and *p-depth* :: *'a* ⇒ *'b* ⇒ *int*
and *p-restrict* :: *'b* ⇒ (*'a* ⇒ *bool*) ⇒ *'b*
and *func-of-consts* :: (*'a* ⇒ *'c*::{*factorial-semiring*,*ring-1*}) ⇒ *'b*
and *index-inj* :: *'a* ⇒ *'c*
begin

lemma *index-const-powi-globally-nequiv-0*: ¬ *p-equal q* (*index-const p powi n*) *0*
⟨*proof*⟩

lemma *global-uniformizer-powi-globally-nequiv-0*: ¬ *p-equal q* (*global-uniformizer*

powi n) 0
⟨*proof*⟩

lemma *index-const-powi-add*:
($index$-$const$ p) $powi$ (m + n) = ($index$-$const$ p) $powi$ m ∗ ($index$-$const$ p) $powi$ n
⟨*proof*⟩

lemma *global-uniformizer-powi-add*:
global-uniformizer powi (m + n) = *global-uniformizer powi m* ∗ *global-uniformizer powi n*
⟨*proof*⟩

lemma *p-depth-index-const-powi*: *p-depth p* (*index-const p powi n*) = *n*
⟨*proof*⟩

lemma *p-depth-index-const-powi′*: *p-depth q* (*index-const p powi n*) = *n* ∗ *of-bool* (*q* = *p*)
⟨*proof*⟩

lemma *p-depth-global-uniformizer-powi*: *p-depth p* (*global-uniformizer powi n*) = *n*
⟨*proof*⟩

lemma *local-uniformizer-locally-uniformizes*:
p-depth p ((*index-const p*) *powi* (−*n*) ∗ *x*) = *0* **if** *p-depth p x* = *n*
⟨*proof*⟩

lemma *global-uniformizer-locally-uniformizes*:
p-depth p (*global-uniformizer powi* (−*n*) ∗ *x*) = *0* **if** *p-depth p x* = *n*
⟨*proof*⟩

lemma *p-depth-set-eq-index-const-coset*:
p-depth-set p n = ((*index-const p*) *powi n*) ∗*o* (*p-depth-set p 0*)
⟨*proof*⟩

lemma *p-depth-set-eq-index-const-coset′*:
p-depth-set p 0 = ((*index-const p*) *powi* (−*n*)) ∗*o* (*p-depth-set p n*)
⟨*proof*⟩

lemma *p-depth-set-eq-global-uniformizer-coset*:
p-depth-set p n = (*global-uniformizer powi n*) ∗*o* (*p-depth-set p 0*)
⟨*proof*⟩

lemma *p-depth-set-eq-global-uniformizer-coset′*:
p-depth-set p 0 = (*global-uniformizer powi* (−*n*)) ∗*o* (*p-depth-set p n*)
⟨*proof*⟩

lemma *global-depth-set-eq-global-uniformizer-coset*:
global-depth-set n = (*global-uniformizer powi n*) ∗*o* (*global-depth-set 0*)

$\langle proof \rangle$

lemma *global-depth-set-eq-global-uniformizer-coset'*:
global-depth-set 0 = (*global-uniformizer powi* (−*n*)) ∗*o* (*global-depth-set n*)
$\langle proof \rangle$

definition *shift-p-depth* :: *'a* ⇒ *int* ⇒ *'b* ⇒ *'b*
where *shift-p-depth p n x* = *x* ∗ (*consts* (λ*q*. *if q* = *p then index-inj p else 1*)) *powi n*

lemma *shift-p-depth-0*[*simp*]: *shift-p-depth p n 0* = *0*
$\langle proof \rangle$

lemma *shift-p-depth-by-0*[*simp*]: *shift-p-depth p 0 x* = *x*
$\langle proof \rangle$

lemma *shift-p-depth-add*[*simp*]:
shift-p-depth p n (*x* + *y*) = *shift-p-depth p n x* + *shift-p-depth p n y*
$\langle proof \rangle$

lemma *shift-p-depth-uminus*[*simp*]:
shift-p-depth p n (− *x*) = − *shift-p-depth p n x*
$\langle proof \rangle$

lemma *shift-p-depth-minus*[*simp*]:
shift-p-depth p n (*x* − *y*) = *shift-p-depth p n x* − *shift-p-depth p n y*
$\langle proof \rangle$

lemma *shift-p-depth-equiv-at-place*: *p-equal p* (*shift-p-depth p n x*) (*x* ∗ (*index-const p powi n*))
$\langle proof \rangle$

lemma *shift-p-depth-equiv-at-other-places*:
p-equal q (*shift-p-depth p n x*) *x* **if** *q* ≠ *p*
$\langle proof \rangle$

lemma *shift-p-depth-equiv0-iff*:
p-equal q (*shift-p-depth p n x*) *0* ⟷ *p-equal q x 0*
$\langle proof \rangle$

lemma *shift-p-depth-equiv-iff*:
p-equal q (*shift-p-depth p n x*) (*shift-p-depth p n y*) = *p-equal q x y*
$\langle proof \rangle$

lemma *shift-p-depth-trivial-iff*:
shift-p-depth p n x = *x* ⟷ *n* = *0* ∨ *p-equal p x 0*
$\langle proof \rangle$

lemma *shift-p-depth-times-right*: *shift-p-depth p n* (*x* ∗ *y*) = *x* ∗ *shift-p-depth p n*

y
⟨*proof*⟩

lemma *shift-p-depth-times-left*: $\textit{shift-p-depth}\ p\ n\ (x * y) = \textit{shift-p-depth}\ p\ n\ x * y$
⟨*proof*⟩

lemma *shift-shift-p-depth*: $\textit{shift-p-depth}\ p\ n\ (\textit{shift-p-depth}\ p\ m\ x) = \textit{shift-p-depth}\ p\ (m + n)\ x$
⟨*proof*⟩

lemma *shift-shift-p-depth-image*:
$\textit{shift-p-depth}\ p\ n\ `\ \textit{shift-p-depth}\ p\ m\ `\ A = \textit{shift-p-depth}\ p\ (m + n)\ `\ A$
⟨*proof*⟩

lemma *shift-p-depth-at-place*:
$\textit{p-depth}\ p\ (\textit{shift-p-depth}\ p\ n\ x) = \textit{p-depth}\ p\ x + n$ **if** $n = 0 \lor \neg\ \textit{p-equal}\ p\ x\ 0$
⟨*proof*⟩

lemma *shift-p-depth-at-other-places*:
$\textit{p-depth}\ q\ (\textit{shift-p-depth}\ p\ n\ x) = \textit{p-depth}\ q\ x$ **if** $q \neq p$
⟨*proof*⟩

lemma *p-depth-set-shift-p-depth-closed*:
$\textit{shift-p-depth}\ p\ n\ x \in \textit{p-depth-set}\ p\ m$ **if** $\textit{p-depth}\ p\ x \geq m - n$
⟨*proof*⟩

lemma *global-depth-set-shift-p-depth-closed*:
$\textit{shift-p-depth}\ p\ n\ x \in \textit{global-depth-set}\ m$
if $x \in \textit{global-depth-set}\ m$
and $\neg\ \textit{p-equal}\ p\ x\ 0 \longrightarrow \textit{p-depth}\ p\ x \geq m - n$
⟨*proof*⟩

lemma *shift-p-depth-mem-p-depth-set*:
$\textit{p-depth}\ p\ x \geq m - n$
if $\neg\ \textit{p-equal}\ p\ x\ 0$ **and** $\textit{shift-p-depth}\ p\ n\ x \in \textit{p-depth-set}\ p\ m$
⟨*proof*⟩

lemma *shift-p-depth-mem-global-depth-set*:
$\textit{p-depth}\ p\ x \geq m - n$
if $\neg\ \textit{p-equal}\ p\ x\ 0$ **and** $\textit{shift-p-depth}\ p\ n\ x \in \textit{global-depth-set}\ m$
⟨*proof*⟩

lemma *shift-p-depth-p-depth-set*: $\textit{shift-p-depth}\ p\ n\ `\ \textit{p-depth-set}\ p\ m = \textit{p-depth-set}\ p\ (m + n)$
⟨*proof*⟩

lemma *shift-p-depth-cong*:
$\textit{p-equal}\ q\ (\textit{shift-p-depth}\ p\ n\ x)\ (\textit{shift-p-depth}\ p\ n\ y)$ **if** $\textit{p-equal}\ q\ x\ y$
⟨*proof*⟩

lemma *shift-p-depth-p-restrict*:
  shift-p-depth p n (*p-restrict x P*) = *p-restrict* (*shift-p-depth p n x*) *P*
⟨*proof*⟩

lemma *shift-p-depth-p-restrict-global-depth-set-memI*:
  shift-p-depth p n (*p-restrict x* ((=) *p*)) ∈ *global-depth-set m*
  if ¬ *p-equal p x 0* ⟶ *p-depth p x* ≥ *m* − *n*
  ⟨*proof*⟩

lemma *shift-p-depth-p-restrict-global-depth-set-memI'*:
  shift-p-depth p n (*p-restrict x* ((=) *p*)) ∈ *global-depth-set* (*m* + *n*)
  if *x* ∈ *p-depth-set p m*
⟨*proof*⟩

lemma *shift-p-depth-p-restrict-global-depth-set-image*:
  shift-p-depth p n ‘ (λ*x*. *p-restrict x* ((=) *p*)) ‘ *global-depth-set m*
    = (λ*x*. *p-restrict x* ((=) *p*)) ‘ *global-depth-set* (*m* + *n*)
⟨*proof*⟩

end

3.7 Topological patterns

3.7.1 By place

Convergence of sequences as measured by depth

context *p-equality-depth*
begin

definition *p-limseq-condition* ::
  'a ⇒ (*nat* ⇒ *'b*) ⇒ *'b* ⇒ *int* ⇒
    nat ⇒ *bool*
  where
    p-limseq-condition p X x n K =
      (∀ *k*≥*K*. ¬ *p-equal p* (*X k*) *x* ⟶ *p-depth p* (*X k* − *x*) > *n*)

lemma *p-limseq-conditionI* [*intro*]:
  p-limseq-condition p X x n K
  if
    ⋀*k*. *k* ≥ *K* ⟹ ¬ *p-equal p* (*X k*) *x* ⟹
      p-depth p (*X k* − *x*) > *n*
  ⟨*proof*⟩

lemma *p-limseq-conditionD*:
  p-depth p (*X k* − *x*) > *n*
  if *p-limseq-condition p X x n K* **and** *k* ≥ *K* **and** ¬ *p-equal p* (*X k*) *x*
  ⟨*proof*⟩

abbreviation *p-limseq* ::
  $'a \Rightarrow (nat \Rightarrow 'b) \Rightarrow 'b \Rightarrow bool$
  where *p-limseq p X x* $\equiv$ ($\forall$ *n*. $\exists$ *K*. *p-limseq-condition p X x n K*)

lemma *p-limseq-p-cong*:
  p-limseq p X x
  if $\forall$ *n*$\geq$*N*. *p-equal p* (*X n*) (*Y n*) **and** *p-equal p x y* **and** *p-limseq p Y y*
⟨*proof*⟩

lemma *p-limseq-p-constant*: *p-limseq p X x* **if** $\forall$ *n*. *p-equal p* (*X n*) *x*
  ⟨*proof*⟩

lemma *p-limseq-constant*: *p-limseq p* (λ*n*. *x*) *x*
  ⟨*proof*⟩

lemma *p-limseq-p-eventually-constant*:
  p-limseq p X x **if** $\forall_F$ *k in sequentially*. *p-equal p* (*X k*) *x*
⟨*proof*⟩

lemma *p-limseq-0* [*simp*]: *p-limseq p 0 0*
  ⟨*proof*⟩

lemma *p-limseq-0-iff*: *p-limseq p 0 x* $\longleftrightarrow$ *p-equal p x 0*
⟨*proof*⟩

lemma *p-limseq-add*: *p-limseq p* (*X* + *Y*) (*x* + *y*) **if** *p-limseq p X x* **and** *p-limseq p Y y*
⟨*proof*⟩

lemma *p-limseq-uminus*: *p-limseq p* ($-$*X*) ($-$*x*) **if** *p-limseq p X x*
⟨*proof*⟩

lemma *p-limseq-diff*: *p-limseq p* (*X* $-$ *Y*) (*x* $-$ *y*) **if** *p-limseq p X x* **and** *p-limseq p Y y*
  ⟨*proof*⟩

lemma *p-limseq-conv-0*: *p-limseq p X x* $\longleftrightarrow$ *p-limseq p* (λ*n*. *X n* $-$ *x*) *0*
⟨*proof*⟩

lemma *p-limseq-unique*: *p-equal p x y* **if** *p-limseq p X x* **and** *p-limseq p X y*
  ⟨*proof*⟩

lemma *p-limseq-eventually-nequiv-0*:
  $\forall_F$ *k in sequentially*. $\neg$ *p-equal p* (*X k*) *0*
  if $\neg$ *p-equal p x 0* **and** *p-limseq p X x*
⟨*proof*⟩

lemma *p-limseq-bdd-depth*:
  assumes $\neg$ *p-equal p x 0* **and** *p-limseq p X x*

obtains d **where** $\forall k.\ \neg$ *p-equal* $p\ (X\ k)\ 0 \longrightarrow$ *p-depth* $p\ (X\ k) \leq d$
⟨*proof*⟩

lemma *p-limseq-eventually-bdd-depth*:
$\forall_F\ k$ *in sequentially*. *p-depth* $p\ (X\ k) \leq$ *p-depth* $p\ x$
if $\neg$ *p-equal* $p\ x\ 0$ **and** *p-limseq* $p\ X\ x$
⟨*proof*⟩

lemma *p-limseq-delayed-iff*:
p-limseq $p\ X\ x \longleftrightarrow$ *p-limseq* $p\ (\lambda n.\ X\ (N + n))\ x$
⟨*proof*⟩

lemma *p-depth-set-p-limseq-closed*:
$x \in$ *p-depth-set* $p\ n$
if *p-limseq* $p\ X\ x$ **and** $\forall_F\ k$ *in sequentially*. $X\ k \in$ *p-depth-set* $p\ n$
⟨*proof*⟩

Cauchy sequences by place

definition *p-cauchy-condition* ::
$'a \Rightarrow (nat \Rightarrow 'b) \Rightarrow int \Rightarrow nat \Rightarrow bool$
where
p-cauchy-condition $p\ X\ n\ K =$
$(\forall k1 \geq K.\ \forall k2 \geq K.$
$\neg$ *p-equal* $p\ (X\ k1)\ (X\ k2) \longrightarrow$ *p-depth* $p\ (X\ k1 - X\ k2) > n$
$)$

lemma *p-cauchy-conditionI*:
p-cauchy-condition $p\ X\ n\ K$
if
$\bigwedge k\ k'.\ k \geq K \Longrightarrow k' \geq K \Longrightarrow$
$\neg$ *p-equal* $p\ (X\ k)\ (X\ k') \Longrightarrow$ *p-depth* $p\ (X\ k - X\ k') > n$
⟨*proof*⟩

lemma *p-cauchy-conditionD*:
p-depth $p\ (X\ k - X\ k') > n$
if *p-cauchy-condition* $p\ X\ n\ K$ **and** $k \geq K$ **and** $k' \geq K$
and $\neg$ *p-equal* $p\ (X\ k)\ (X\ k')$
⟨*proof*⟩

lemma *p-cauchy-condition-mono-seq-tail*:
p-cauchy-condition $p\ X\ n\ K \Longrightarrow$ *p-cauchy-condition* $p\ X\ n\ K'$
if $K' \geq K$
⟨*proof*⟩

lemma *p-cauchy-condition-mono-depth*:
p-cauchy-condition $p\ X\ n\ K \Longrightarrow$ *p-cauchy-condition* $p\ X\ m\ K$
if $m \leq n$
⟨*proof*⟩

abbreviation *p-cauchy p X* $\equiv$ ($\forall$ *n*. $\exists$ *K*. *p-cauchy-condition p X n K*)

lemma *p-cauchy-condition-LEAST*:
p-cauchy-condition p X n (*LEAST K*. *p-cauchy-condition p X n K*)
if *p-cauchy p X*
⟨*proof*⟩

lemma *p-cauchy-condition-LEAST-mono*:
(*LEAST K*. *p-cauchy-condition p X m K*) $\leq$ (*LEAST K*. *p-cauchy-condition p X n K*)
if *p-cauchy p X* **and** $m \leq n$
⟨*proof*⟩

definition *p-consec-cauchy-condition* ::
'a $\Rightarrow$ (*nat* $\Rightarrow$ *'b*) $\Rightarrow$ *int* $\Rightarrow$ *nat* $\Rightarrow$ *bool*
where
p-consec-cauchy-condition p X n K =
($\forall$ *k*$\geq$*K*.
$\neg$ *p-equal p* (*X* (*Suc k*)) (*X k*) $\longrightarrow$ *p-depth p* (*X* (*Suc k*) $-$ *X k*) $>$ *n*
)

lemma *p-consec-cauchy-conditionI*:
p-consec-cauchy-condition p X n K
if
$\bigwedge$*k*. $k \geq K \Longrightarrow \neg$ *p-equal p* (*X* (*Suc k*)) (*X k*) $\Longrightarrow$
p-depth p (*X* (*Suc k*) $-$ *X k*) $>$ *n*
⟨*proof*⟩

lemma *p-consec-cauchy*:
p-cauchy p X = ($\forall$ *n*. $\exists$ *K*. *p-consec-cauchy-condition p X n K*)
⟨*proof*⟩

end

context *p-equality-depth-no-zero-divisors*
begin

lemma *p-limseq-mult-both-0*: *p-limseq p* (*X* $*$ *Y*) *0* **if** *p-limseq p X 0* **and** *p-limseq p Y 0*
⟨*proof*⟩

lemma *p-limseq-constant-mult-0*: *p-limseq p* (λ*n*. *x* $*$ *Y n*) *0* **if** *p-limseq p Y 0*
⟨*proof*⟩

lemma *p-limseq-mult-0-right*: *p-limseq p* (*X* $*$ *Y*) *0* **if** *p-limseq p X x* **and** *p-limseq p Y 0*
⟨*proof*⟩

lemma *p-limseq-mult*: *p-limseq p* (*X* $*$ *Y*) (*x* $*$ *y*) **if** *p-limseq p X x* **and** *p-limseq*

p Y y
⟨*proof*⟩

lemma *poly-continuous-p-open-nbhds*:
p-limseq p (*λn. poly f* (*X n*)) (*poly f x*) **if** *p-limseq p X x*
⟨*proof*⟩

end

context *p-equality-depth-no-zero-divisors-1*
begin

lemma *p-limseq-pow*: *p-limseq p* (*λk.* (*X k*)^*n*) (*x* ^ *n*) **if** *p-limseq p X x*
⟨*proof*⟩

lemma *p-limseq-poly*: *p-limseq p* (*λn. poly f* (*X n*)) (*poly f x*) **if** *p-limseq p X x*
⟨*proof*⟩

end

context *p-equality-depth-div-inv*
begin

lemma *p-limseq-inverse*:
p-limseq p (*λn. inverse* (*X n*)) (*inverse x*)
if ¬ *p-equal p x 0* **and** *p-limseq p X x*
⟨*proof*⟩

lemma *p-limseq-divide*:
p-limseq p (*λn. X n* / *Y n*) (*x* / *y*)
if ¬ *p-equal p y 0* **and** *p-limseq p X x* **and** *p-limseq p Y y*
⟨*proof*⟩

end

context *global-p-equality-depth*
begin

lemma *p-limseq-p-restrict-iff*:
p-limseq p (*λn. p-restrict* (*X n*) *P*) *x* ⟷ *p-limseq p X x* **if** *P p*
⟨*proof*⟩

lemma *p-limseq-p-restricted*: *p-limseq p* (*λn. p-restrict* (*X n*) *P*) *0* **if** ¬ *P p*
⟨*proof*⟩

end

Base of open sets at local places

context *global-p-equal-depth*

begin

abbreviation *p-open-nbhd* :: $'a \Rightarrow int \Rightarrow 'b \Rightarrow 'b$ *set*
  where *p-open-nbhd p n x* $\equiv$ *x +o p-depth-set p n*

abbreviation *local-p-open-nbhds p* $\equiv$ { *p-open-nbhd p n x* | *n x*. *True* }
abbreviation *p-open-nbhds* $\equiv$ ($\bigcup$*p*. *local-p-open-nbhds p*)

lemma *p-open-nbhd*: $x \in$ *p-open-nbhd p n x*
  ⟨*proof*⟩

lemma *local-p-open-nbhd-is-open*: *generate-topology* (*local-p-open-nbhds p*) (*p-open-nbhd p n x*)
  ⟨*proof*⟩

lemma *p-open-nbhd-is-open*: *generate-topology p-open-nbhds* (*p-open-nbhd p n x*)
  ⟨*proof*⟩

lemma *local-p-open-nbhds-are-coarser*:
  generate-topology (*local-p-open-nbhds p*) *S* $\Longrightarrow$ *generate-topology p-open-nbhds S*
⟨*proof*⟩

lemma *p-open-nbhd-diff-depth*:
  p-depth p $(y - x) \geq n$ **if** $\neg$ *p-equal p y x* **and** $y \in$ *p-open-nbhd p n x*
⟨*proof*⟩

lemma *p-open-nbhd-eq-circle*:
  p-open-nbhd p n x = {*y*. $\neg$ *p-equal p y x* $\longrightarrow$ *p-depth p* $(y - x) \geq n$}
⟨*proof*⟩

lemma *p-open-nbhd-circle-multicentre*:
  p-open-nbhd p n y = *p-open-nbhd p n x* **if** $y \in$ *p-open-nbhd p n x*
⟨*proof*⟩

lemma *p-open-nbhd-circle-multicentre'*:
  $x \in$ *p-open-nbhd p n y* $\longleftrightarrow$ $y \in$ *p-open-nbhd p n x*
  ⟨*proof*⟩

lemma *p-open-nbhd-p-restrict*:
  p-open-nbhd p n (*p-restrict x P*) = *p-open-nbhd p n x* **if** *P p*
⟨*proof*⟩

lemma *p-restrict-p-open-nbhd-mem-iff*:
  $y \in$ *p-open-nbhd p n x* $\longleftrightarrow$ *p-restrict y P* $\in$ *p-open-nbhd p n x* **if** *P p*
  ⟨*proof*⟩

lemma *p-open-nbhd-p-restrict-add-mixed-drop-right*:
  p-open-nbhd p n (*p-restrict x P* + *p-restrict y Q*) = *p-open-nbhd p n x*
  if *P p* **and** $\neg$ *Q p*

⟨*proof*⟩

lemma *p-open-nbhd-p-restrict-add-mixed-drop-left*:
p-open-nbhd p n (*p-restrict x P* + *p-restrict y Q*) = *p-open-nbhd p n y*
if ¬ *P p* **and** *Q p*
⟨*proof*⟩

lemma *p-open-nbhd-antimono*:
p-open-nbhd p n x ⊆ *p-open-nbhd p m x* **if** $m \leq n$
⟨*proof*⟩

lemma *p-open-nbhds-open-subopen*:
generate-topology (*local-p-open-nbhds p*) *A* =
(∀ *a*∈*A*. ∃ *n*. *p-open-nbhd p n a* ⊆ *A*)
⟨*proof*⟩

lemma *p-restrict-p-open-set-mem-iff*:
a ∈ *A* ⟷ *p-restrict a P* ∈ *A*
if *generate-topology* (*local-p-open-nbhds p*) *A* **and** *P p*
⟨*proof*⟩

lemma *hausdorff*:
∃ *p n*. (*p-open-nbhd p n x*) ∩ (*p-open-nbhd p n y*) = {} **if** $x \neq y$
⟨*proof*⟩

sublocale *t2-p-open-nbhds*: *t2-space generate-topology p-open-nbhds*
⟨*proof*⟩

abbreviation *p-open-nbhds-tendsto* ≡ *t2-p-open-nbhds.tendsto*
abbreviation *p-open-nbhds-convergent* ≡ *t2-p-open-nbhds.convergent*
abbreviation *p-open-nbhds-LIMSEQ* ≡ *t2-p-open-nbhds.LIMSEQ*

lemma *locally-limD*: ∃ *K*. *p-limseq-condition p X x n K* **if** *p-open-nbhds-LIMSEQ X x*
⟨*proof*⟩

lemma *globally-limseq-imp-locally-limseq*:
p-limseq p X x **if** *p-open-nbhds-LIMSEQ X x*
⟨*proof*⟩

lemma *globally-limseq-iff-locally-limseq*: *p-open-nbhds-LIMSEQ X x* = (∀ *p*. *p-limseq p X x*)
⟨*proof*⟩

lemma *p-open-nbhds-LIMSEQ-eventually-p-constant*:
p-equal p x c
if *p-open-nbhds-LIMSEQ X x* **and** $\forall_F$ *k in sequentially*. *p-equal p* (*X k*) *c*
⟨*proof*⟩

lemma *p-open-nbhds-LIMSEQ-p-restrict*:
p-equal p x 0 **if** *p-open-nbhds-LIMSEQ* (λn. *p-restrict* (*X n*) *P*) *x* **and** $\neg$ *P p*
⟨*proof*⟩

lemma *global-depth-set-p-open-nbhds-LIMSEQ-closed*:
x $\in$ *global-depth-set n*
if *p-open-nbhds-LIMSEQ X x*
and $\forall_F$ *k in sequentially. X k* $\in$ *global-depth-set n*
⟨*proof*⟩

end

context *global-p-equal-depth-div-inv-w-inj-index-consts*
begin

lemma *shift-p-depth-p-open-nbhd*:
shift-p-depth p n ‘ *p-open-nbhd p m x* = *p-open-nbhd p* (*m* + *n*) (*shift-p-depth p n x*)
⟨*proof*⟩

lemma *shift-p-depth-p-open-set*:
generate-topology (*local-p-open-nbhds p*) (*shift-p-depth p n* ‘ *A*)
if *generate-topology* (*local-p-open-nbhds p*) *A*
⟨*proof*⟩

end

locale *p-complete-global-p-equal-depth* = *global-p-equal-depth* +
assumes *complete*:
p-cauchy p X $\Longrightarrow$ *p-open-nbhds-convergent* (λn. *p-restrict* (*X n*) ((=) *p*))
begin

lemma *p-cauchyE*:
assumes *p-cauchy p X*
obtains *x* **where** *p-open-nbhds-LIMSEQ* (λn. *p-restrict* (*X n*) ((=) *p*)) *x*
⟨*proof*⟩

end

Hensel’s Lemma.

context *p-equality-div-inv*
begin

— A sequence to approach a root from within the ring of integers.
primrec *hensel-seq* :: *'a* ⇒ *'b poly* ⇒ *'b* ⇒ *nat* ⇒ *'b*
where *hensel-seq p f x 0* = *x* |
hensel-seq p f x (*Suc n*) = (
if p-equal p (*poly f* (*hensel-seq p f x n*)) *0 then hensel-seq p f x n*
else

hensel-seq p f x n − (*poly f* (*hensel-seq p f x n*)) / (*poly* (*polyderiv f*) (*hensel-seq p f x n*))
)

lemma *constant-hensel-seq-case*:
k ≥ *n* ⟹ *hensel-seq p f x k* = *hensel-seq p f x n*
if *p-equal p* (*poly f* (*hensel-seq p f x n*)) *0*
⟨*proof*⟩

end

context *p-equality-depth-div-inv*
begin

— Context for inductive steps leading to Hensel's Lemma.
context
fixes *p* :: *'a* **and** *f* :: *'b poly* **and** *x* :: *'b* **and** *n* :: *nat*
assumes *f-deg* : *degree f* > *0*
and *f-depth* : *set* (*coeffs f*) ⊆ *p-depth-set p 0*
and *d-hensel*: *hensel-seq p f x n* ∈ *p-depth-set p 0*
and *f* : ¬ *p-equal p* (*poly f* (*hensel-seq p f x n*)) *0*
and *deriv-f* : ¬ *p-equal p* (*poly* (*polyderiv f*) (*hensel-seq p f x n*)) *0*
and *df* :
p-depth p (*poly f* (*hensel-seq p f x n*)) >
2 ∗ *p-depth p* (*poly* (*polyderiv f*) (*hensel-seq p f x n*))
begin

lemma *hensel-seq-step-1*:
p-depth p (*poly f* (*hensel-seq p f x* (*Suc n*))) >
p-depth p (*poly f* ((*hensel-seq p f x n*)))
if *next-f*: ¬ *p-equal p* (*poly f* (*hensel-seq p f x* (*Suc n*))) *0*
⟨*proof*⟩

lemma *hensel-seq-step-2*:
¬ *p-equal p* (*poly* (*polyderiv f*) (*hensel-seq p f x* (*Suc n*))) *0*
p-depth p (*poly* (*polyderiv f*) (*hensel-seq p f x* (*Suc n*))) =
p-depth p (*poly* (*polyderiv f*) (*hensel-seq p f x n*))
⟨*proof*⟩

end

— Context for Hensel's Lemma.
context
fixes *p* :: *'a* **and** *f* :: *'b poly* **and** *x* :: *'b*
assumes *f-deg* : *degree f* > *0*
and *f-depth* : *set* (*coeffs f*) ⊆ *p-depth-set p 0*
and *d-init* : *x* ∈ *p-depth-set p 0*
and *init-deriv* :
¬ *p-equal p* (*poly f x*) *0* ⟶ ¬ *p-equal p* (*poly* (*polyderiv f*) *x*) *0*

and *d-init-deriv*:
$\neg$ *p-equal p* (*poly f x*) *0* $\longrightarrow$
p-depth p (*poly f x*) $>$ *2* $*$ *p-depth p* (*poly* (*polyderiv f*) *x*)
begin

lemma
hensel-seq-adelic-int : *hensel-seq p f x n* $\in$ *p-depth-set p 0* (**is** *?A*)
and *hensel-seq-poly-depth* :
$\neg$ *p-equal p* (*poly f* (*hensel-seq p f x n*)) *0* $\longrightarrow$
p-depth p (*poly f* (*hensel-seq p f x n*)) $\geq$ *p-depth p* (*poly f x*) + *int n*
(**is** *?B*)
and *hensel-seq-deriv* :
$\neg$ *p-equal p* (*poly f* (*hensel-seq p f x n*)) *0* $\longrightarrow$
$\neg$ *p-equal p* (*poly* (*polyderiv f*) (*hensel-seq p f x n*)) *0*
(**is** *?C*)
and *hensel-seq-deriv-depth*:
$\neg$ *p-equal p* (*poly f* (*hensel-seq p f x n*)) *0* $\longrightarrow$
p-depth p (*poly* (*polyderiv f*) (*hensel-seq p f x n*)) = *p-depth p* (*poly* (*polyderiv f*) *x*)
(**is** *?D*)
and *hensel-seq-depth-ineq*:
$\neg$ *p-equal p* (*poly f* (*hensel-seq p f x n*)) *0* $\longrightarrow$
p-depth p (*poly f* (*hensel-seq p f x n*)) $>$
2 $*$ *p-depth p* (*poly* (*polyderiv f*) (*hensel-seq p f x n*))
(**is** *?E*)
⟨*proof*⟩

lemma *hensel-seq-cauchy*: *p-cauchy p* (*hensel-seq p f x*)
⟨*proof*⟩

lemma *p-limseq-poly-hensel-seq*: *p-limseq p* (λ*n. poly f* (*hensel-seq p f x n*)) *0*
⟨*proof*⟩

end

end

locale *p-complete-global-p-equal-depth-div-inv* =
global-p-equal-depth-div-inv + *p-complete-global-p-equal-depth*
begin

lemma *hensel*:
fixes *p* :: *'a* **and** *f* :: *'b poly* **and** *x* :: *'b*
assumes *degree f* $>$ *0*
and *set* (*coeffs f*) $\subseteq$ *p-depth-set p 0*
and *x* $\in$ *p-depth-set p 0*
and
$\neg$ *p-equal p* (*poly f x*) *0* $\longrightarrow$ $\neg$ *p-equal p* (*poly* (*polyderiv f*) *x*) *0*

and
  ¬ *p-equal p (poly f x) 0* ⟶
    p-depth p (poly f x) > 2 ∗ p-depth p (poly (polyderiv f) x)
obtains *z* **where** *z* ∈ *p-depth-set p 0* **and** *p-equal p (poly f z) 0*
⟨*proof*⟩

end

3.7.2 Globally

We use bounded depth to create a global metric.

context *p-equality-depth*
begin

definition *bdd-global-depth* :: *'b* ⇒ *nat*
  where
    bdd-global-depth x =
      (*if globally-p-equal x 0 then 0 else Inf* {*nat* (*p-depth p x* + *1*) | *p*. ¬ *p-equal p x 0*})
— The plus-one is to distinguish depth-0 from negative depth.

lemma *bdd-global-depth-0*[*simp*]: *globally-p-equal x 0* ⟹ *bdd-global-depth x* = *0*
  ⟨*proof*⟩

lemma *bdd-global-depthD*:
  bdd-global-depth x = *Inf* {*nat* (*p-depth p x* + *1*) | *p*. ¬ *p-equal p x 0*}
  if ¬ *globally-p-equal x 0*
  ⟨*proof*⟩

lemma *bdd-global-depthD-as-image*:
  bdd-global-depth x = (*INF p*∈{*p*. ¬ *p-equal p x 0*}. *nat* (*p-depth p x* + *1*))
  if ¬ *globally-p-equal x 0*
  ⟨*proof*⟩

lemma *bdd-global-depth-cong*:
  bdd-global-depth x = *bdd-global-depth y* **if** *globally-p-equal x y*
⟨*proof*⟩

lemma *bdd-global-depth-le*:
  bdd-global-depth x ≤ *nat* (*p-depth p x* + *1*) **if** ¬ *p-equal p x 0*
  ⟨*proof*⟩

lemma *bdd-global-depth-greatest*:
  bdd-global-depth x ≥ *B*
  if ¬ *p-equal p x 0*
  and ∀ *q*. ¬ *p-equal q x 0* ⟶ *nat* (*p-depth q x* + *1*) ≥ *B*
  ⟨*proof*⟩

lemma *bdd-global-depth-witness*:

assumes $\neg$ *globally-p-equal x 0*
obtains *p* **where** $\neg$ *p-equal p x 0* **and** *bdd-global-depth x* = *nat* (*p-depth p x* + *1*)
⟨*proof*⟩

lemma *bdd-global-depth-eq-0*:
bdd-global-depth x = *0* **if** *p-depth p x* < *0*
⟨*proof*⟩

lemma *bdd-global-depth-eq-0-iff*:
bdd-global-depth x = *0* ⟷
(∃ *p*. *p-depth p x* < *0*) ∨ (*globally-p-equal x 0*)
(**is** *?L* = *?R*)
⟨*proof*⟩

lemma *bdd-global-depth-diff*: *bdd-global-depth* (*x* − *y*) = *bdd-global-depth* (*y* − *x*)
⟨*proof*⟩

lemma *bdd-global-depth-uminus*: *bdd-global-depth* (− *x*) = *bdd-global-depth x*
⟨*proof*⟩

lemma *bdd-global-depth-nonarch*:
bdd-global-depth (*x* + *y*) ≥ *min* (*bdd-global-depth x*) (*bdd-global-depth y*)
if $\neg$ *globally-p-equal* (*x* + *y*) *0*
⟨*proof*⟩

definition *bdd-global-dist* :: *'b* ⇒ *'b* ⇒ *real* **where**
bdd-global-dist x y =
(*if globally-p-equal x y then 0 else inverse* (*2* ^ *bdd-global-depth* (*x* − *y*)))

lemma *bdd-global-dist-eqD* [*simp*]: *bdd-global-dist x y* = *0* **if** *globally-p-equal x y*
⟨*proof*⟩

lemma *bdd-global-distD*:
bdd-global-dist x y = *inverse* (*2* ^ *bdd-global-depth* (*x* − *y*)) **if** $\neg$ *globally-p-equal x y*
⟨*proof*⟩

lemma *bdd-global-dist-cong*:
bdd-global-dist w x = *bdd-global-dist y z*
if *globally-p-equal w y* **and** *globally-p-equal x z*
⟨*proof*⟩

lemma *bdd-global-dist-nonneg*: *bdd-global-dist x y* ≥ *0*
⟨*proof*⟩

lemma *bdd-global-dist-bdd*: *bdd-global-dist x y* ≤ *1*
⟨*proof*⟩

lemma *bdd-global-dist-lessI*:
bdd-global-dist x y $<$ *e*
if *pos*: $e > 0$
and *depth*:
$\bigwedge p.\ \neg$ *p-equal p x y* $\Longrightarrow$
p-depth p $(x - y) \geq \lfloor log\ 2\ (inverse\ e) \rfloor$
$\langle proof \rangle$

lemma *bdd-global-dist-less-imp*:
p-depth p $(x - y) \geq \lfloor log\ 2\ (inverse\ e) \rfloor$
if $e > 0$ **and** $e \leq 1$ **and** $\neg$ *p-equal p x y* **and** *bdd-global-dist x y* $< e$
$\langle proof \rangle$

lemma *bdd-global-dist-less-pow2-iff*:
bdd-global-dist x y $<$ *inverse* $(2\ \hat{}\ n) \longleftrightarrow$
$(\forall p.\ \neg$ *p-equal p x y* $\longrightarrow$ *p-depth p* $(x - y) \geq int\ n)$
$\langle proof \rangle$

lemma *bdd-global-dist-sym*: *bdd-global-dist x y* $=$ *bdd-global-dist y x*
$\langle proof \rangle$

lemma *bdd-global-dist-conv0*: *bdd-global-dist x y* $=$ *bdd-global-dist* $(x - y)$ *0*
$\langle proof \rangle$

lemma *bdd-global-dist-conv0'*: *bdd-global-dist x y* $=$ *bdd-global-dist 0* $(x - y)$
$\langle proof \rangle$

lemma *bdd-global-dist-nonarch*:
bdd-global-dist x y $\leq$ *max* (*bdd-global-dist x z*) (*bdd-global-dist y z*)
$\langle proof \rangle$

lemma *bdd-global-dist-ball-multicentre*:
$\{z.$ *bdd-global-dist y z* $< e\} = \{z.$ *bdd-global-dist x z* $< e\}$
if *bdd-global-dist x y* $< e$
$\langle proof \rangle$

lemma *bdd-global-dist-ball-at-0*:
$\{z.$ *bdd-global-dist 0 z* $<$ *inverse* $(2\ \hat{}\ n)\} =$ *global-depth-set* (*int n*)
$\langle proof \rangle$

lemma *bdd-global-dist-ball-UNIV*:
$\{z.$ *bdd-global-dist x z* $< e\} =$ *UNIV* **if** $e > 1$
$\langle proof \rangle$

lemma *bdd-global-dist-ball-at-0-normalize*:
$\{z.$ *bdd-global-dist 0 z* $< e\} =$ *global-depth-set* $(\lfloor log\ 2\ (inverse\ e) \rfloor)$
if $e > 0$ **and** $e \leq 1$
$\langle proof \rangle$

lemma *bdd-global-dist-ball-translate*:
$\{z.\ bdd\text{-}global\text{-}dist\ x\ z < e\} = x +o\ \{z.\ bdd\text{-}global\text{-}dist\ 0\ z < e\}$
⟨*proof*⟩

lemma *bdd-global-dist-left-translate-continuous*:
$bdd\text{-}global\text{-}dist\ (x + y)\ (x + z) < e$ **if** $bdd\text{-}global\text{-}dist\ y\ z < e$
⟨*proof*⟩

lemma *bdd-global-dist-right-translate-continuous*:
$bdd\text{-}global\text{-}dist\ (y + x)\ (z + x) < e$ **if** $bdd\text{-}global\text{-}dist\ y\ z < e$
⟨*proof*⟩

definition *global-cauchy-condition* ::
$(nat \Rightarrow 'b) \Rightarrow nat \Rightarrow nat \Rightarrow bool$
where
$global\text{-}cauchy\text{-}condition\ X\ n\ K =$
$(\forall\, k1{\geq}K.\ \forall\, k2{\geq}K.\ \forall\, p.$
$\neg\ p\text{-}equal\ p\ (X\ k1)\ (X\ k2) \longrightarrow nat\ (p\text{-}depth\ p\ (X\ k1\ -\ X\ k2)) > n$
$)$

lemma *global-cauchy-conditionI*:
$global\text{-}cauchy\text{-}condition\ X\ n\ K$
if
$\bigwedge p\ k\ k'.\ k \geq K \Longrightarrow k' \geq K \Longrightarrow$
$\neg\ p\text{-}equal\ p\ (X\ k)\ (X\ k') \Longrightarrow nat\ (p\text{-}depth\ p\ (X\ k - X\ k')) > n$
⟨*proof*⟩

lemma *global-cauchy-conditionD*:
$nat\ (p\text{-}depth\ p\ (X\ k - X\ k')) > n$
if $global\text{-}cauchy\text{-}condition\ X\ n\ K$ **and** $k \geq K$ **and** $k' \geq K$
and $\neg\ p\text{-}equal\ p\ (X\ k)\ (X\ k')$
⟨*proof*⟩

lemma *global-cauchy-condition-mono-seq-tail*:
$global\text{-}cauchy\text{-}condition\ X\ n\ K \Longrightarrow global\text{-}cauchy\text{-}condition\ X\ n\ K'$
if $K' \geq K$
⟨*proof*⟩

lemma *global-cauchy-condition-mono-depth*:
$global\text{-}cauchy\text{-}condition\ X\ n\ K \Longrightarrow global\text{-}cauchy\text{-}condition\ X\ m\ K$
if $m \leq n$
⟨*proof*⟩

abbreviation $globally\text{-}cauchy\ X \equiv (\forall\, n.\ \exists\, K.\ global\text{-}cauchy\text{-}condition\ X\ n\ K)$

lemma *global-cauchy-condition-LEAST*:
$global\text{-}cauchy\text{-}condition\ X\ n\ (LEAST\ K.\ global\text{-}cauchy\text{-}condition\ X\ n\ K)$ **if** *globally-cauchy X*
⟨*proof*⟩

lemma *global-cauchy-condition-LEAST-mono*:
$(LEAST\ K.\ global\text{-}cauchy\text{-}condition\ X\ m\ K) \leq (LEAST\ K.\ global\text{-}cauchy\text{-}condition\ X\ n\ K)$
if *globally-cauchy X* **and** $m \leq n$
⟨*proof*⟩

definition *bdd-global-uniformity* :: $('b \times 'b)$ *filter*
where $bdd\text{-}global\text{-}uniformity = (INF\ e{\in}\{0 <..\}.\ principal\ \{(x,\ y).\ bdd\text{-}global\text{-}dist\ x\ y < e\})$

end

context *global-p-equal-depth*
begin

sublocale *metric-space-by-bdd-depth*:
metric-space bdd-global-dist bdd-global-uniformity
$\lambda U.\ \forall x{\in}U.$
$eventually\ (\lambda(x',\ y).\ x' = x \longrightarrow y \in U)\ bdd\text{-}global\text{-}uniformity$
⟨*proof*⟩

lemma *nonneg-global-depth-set-LIMSEQ-closed*:
$x \in global\text{-}depth\text{-}set\ (int\ n)$
if *lim*: *metric-space-by-bdd-depth.LIMSEQ X x*
and *seq*: $\forall_F\ k\ in\ sequentially.\ X\ k \in global\text{-}depth\text{-}set\ (int\ n)$
⟨*proof*⟩

lemma *globally-cauchy-vs-bdd-metric-Cauchy*:
globally-cauchy X = metric-space-by-bdd-depth.Cauchy X **for** $X :: nat \Rightarrow 'b$
⟨*proof*⟩

end

context *global-p-equal-depth-no-zero-divisors*
begin

lemma *bdd-global-dist-bdd-mult-continuous*:
$bdd\text{-}global\text{-}dist\ (w * x)\ (w * y) < e$
if *pos* : $e > 0$
and *bdd* : $\forall p.\ \neg\ p\text{-}equal\ p\ w\ 0 \longrightarrow p\text{-}depth\ p\ w \geq n$
and *input*: $bdd\text{-}global\text{-}dist\ x\ y < min\ (e * 2\ powi\ (n - 1))\ 1$
⟨*proof*⟩

lemma *bdd-global-dist-limseq-bdd-mult*:
$metric\text{-}space\text{-}by\text{-}bdd\text{-}depth.LIMSEQ\ (\lambda k.\ w * X\ k)\ (w * x)$
if *bdd*: $\forall p.\ \neg\ p\text{-}equal\ p\ w\ 0 \longrightarrow p\text{-}depth\ p\ w \geq n$
and *seq*: *metric-space-by-bdd-depth.LIMSEQ X x*
⟨*proof*⟩

end

context *global-p-equal-depth-div-inv-w-inj-index-consts*
begin

lemma *bdd-global-dist-limseq-global-uniformizer-powi-mult*:
  metric-space-by-bdd-depth.LIMSEQ
    (λk. *global-uniformizer powi n* $*$ *X k*) (*global-uniformizer powi n* $*$ *x*)
  if *metric-space-by-bdd-depth.LIMSEQ X x*
  ⟨*proof*⟩

lemma *global-depth-set-LIMSEQ-closed*:
  $x \in$ *global-depth-set n*
  if *lim* : *metric-space-by-bdd-depth.LIMSEQ X x*
  and *depth*: $\forall_F$ *k in sequentially. X k* $\in$ *global-depth-set n*
⟨*proof*⟩

end

3.8 Notation

consts
  p-equal :: $'a \Rightarrow 'b \Rightarrow 'b \Rightarrow$ *bool*
  globally-p-equal ::
    $'b \Rightarrow 'b \Rightarrow$ *bool*
    ((-/ $\simeq_{\forall p}$/ -) [*51*, *51*] *50*)
  p-restrict :: $'b \Rightarrow ('a \Rightarrow$ *bool*$) \Rightarrow 'b$ (**infixl** *prestrict 70*)
  p-depth :: $'a \Rightarrow 'b \Rightarrow$ *int*
  p-depth-set :: $'a \Rightarrow$ *int* $\Rightarrow 'b$ *set* ($\mathcal{P}_{@\text{-}}{}^{\text{-}}$)
  global-depth-set :: *int* $\Rightarrow 'b$ *set* ($\mathcal{P}_{\forall p}{}^{\text{-}}$)
  global-unfrmzr-pows :: $('a \Rightarrow 'c) \Rightarrow 'b$

abbreviation *p-equal-notation* :: $'b \Rightarrow 'a \Rightarrow 'b \Rightarrow$ *bool*
  ((-/ $\simeq_{\text{-}}$/ -) [*51*, *51*, *51*] *50*)
  where *p-equal-notation x p y* $\equiv$ *p-equal p x y*

abbreviation *p-nequal-notation* :: $'b \Rightarrow 'a \Rightarrow 'b \Rightarrow$ *bool*
  ((-/ $\neg\simeq_{\text{-}}$/ -) [*51*, *51*, *51*] *50*)
  where *p-nequal-notation x p y* $\equiv \neg$ *p-equal p x y*

abbreviation *p-depth-notation* :: $'b \Rightarrow 'a \Rightarrow$ *int*
  ((-$^{\circ\text{-}}$) [*51*, *51*] *50*)
  where *p-depth-notation x p* $\equiv$ *p-depth p x*

abbreviation *p-depth-set-0-notation* :: $'a \Rightarrow 'b$ *set* ($\mathcal{O}_{@\text{-}}$)
  where *p-depth-set-0-notation p* $\equiv$ *p-depth-set p 0*

abbreviation *p-depth-set-1-notation* :: $'a \Rightarrow 'b$ *set* ($\mathcal{P}_{@\text{-}}$)

where *p-depth-set-1-notation p* $\equiv$ *p-depth-set p 1*

abbreviation *global-depth-set-0-notation* :: *'b set* ($\mathcal{O}_{\forall\, p}$)
where *global-depth-set-0-notation* $\equiv$ *global-depth-set 0*

abbreviation *global-depth-set-1-notation* :: *'b set* ($\mathcal{P}_{\forall\, p}$)
where *global-depth-set-1-notation* $\equiv$ *global-depth-set 1*

abbreviation $\mathfrak{p}$ $\equiv$ *global-unfrmzr-pows*

end

theory *FLS-Prime-Equiv-Depth*

imports
Parametric-Equiv-Depth
HOL.Equiv-Relations
HOL−Computational-Algebra.Formal-Laurent-Series

begin

unbundle *fps-syntax*

4 Equivalence of sequences of formal Laurent series relative to divisibility of partial sums by a fixed prime

4.1 Preliminaries

4.1.1 Lift/transfer of equivalence relations.

lemma *relfp-transfer*:
reflp r $\Longrightarrow$ *reflp* $(\lambda x\ y.\ r\ (f\ x)\ (f\ y))$
⟨*proof*⟩

lemma *symp-transfer*:
symp r $\Longrightarrow$ *symp* $(\lambda x\ y.\ r\ (f\ x)\ (f\ y))$
⟨*proof*⟩

lemma *transp-transfer*:
transp r $\Longrightarrow$ *transp* $(\lambda x\ y.\ r\ (f\ x)\ (f\ y))$
⟨*proof*⟩

lemma *equivp-transfer*:
equivp r $\Longrightarrow$ *equivp* $(\lambda x\ y.\ r\ (f\ x)\ (f\ y))$
⟨*proof*⟩

4.1.2 An additional fact about products/sums

lemma (**in** *comm-monoid-set*) *atLeastAtMost-int-shift-bounds*:
$F\ g\ \{m..n\} = F\ (g \circ plus\ m \circ int)\ \{..nat\ (n - m)\}$
if $m \leq n$
for $m\ n :: int$
⟨*proof*⟩

4.1.3 An additional fact about algebraic powers of functions

lemma *pow-fun-apply*: $(f \ \hat{}\ n)\ x = (f\ x)\ \hat{}\ n$
⟨*proof*⟩

4.1.4 Some additional facts about formal power and Laurent series

lemma *fps-shift-fps-mult-by-fps-const*:
fixes $c :: 'a::\{comm\text{-}monoid\text{-}add,mult\text{-}zero\}$
shows $fps\text{-}shift\ n\ (fps\text{-}const\ c * f) = fps\text{-}const\ c * fps\text{-}shift\ n\ f$
and $fps\text{-}shift\ n\ (f * fps\text{-}const\ c) = fps\text{-}shift\ n\ f * fps\text{-}const\ c$
⟨*proof*⟩

lemma *fps-shift-mult-Suc*:
fixes $f\ g :: 'a::\{comm\text{-}monoid\text{-}add,mult\text{-}zero\}\ fps$
shows $fps\text{-}shift\ (Suc\ n)\ (f*g)$
$= fps\text{-}const\ (f\$0) * fps\text{-}shift\ (Suc\ n)\ g + fps\text{-}shift\ n\ (fps\text{-}shift\ 1\ f * g)$
and $fps\text{-}shift\ (Suc\ n)\ (f*g)$
$= fps\text{-}shift\ n\ (f * fps\text{-}shift\ 1\ g) + fps\text{-}shift\ (Suc\ n)\ f * fps\text{-}const\ (g\$0)$
⟨*proof*⟩

lemma *fls-subdegree-minus'*:
$fls\text{-}subdegree\ (f - g) = fls\text{-}subdegree\ f$
if $f \neq 0$ **and** $fls\text{-}subdegree\ g > fls\text{-}subdegree\ f$
⟨*proof*⟩

lemma *fls-regpart-times-fps-X*:
fixes $f :: 'a::\{comm\text{-}monoid\text{-}add,mult\text{-}zero,monoid\text{-}mult\}\ fls$
assumes $fls\text{-}subdegree\ f \geq 0$
shows $fls\text{-}regpart\ f * fps\text{-}X = fls\text{-}regpart\ (fls\text{-}shift\ (-1)\ f)$
and $fps\text{-}X * fls\text{-}regpart\ f = fls\text{-}regpart\ (fls\text{-}shift\ (-1)\ f)$
⟨*proof*⟩

lemma *fls-regpart-times-fps-X-power*:
fixes $f :: 'a::semiring\text{-}1\ fls$
assumes $fls\text{-}subdegree\ f \geq 0$
shows $fls\text{-}regpart\ f * fps\text{-}X \ \hat{}\ n = fls\text{-}regpart\ (fls\text{-}shift\ (-n)\ f)$
and $fps\text{-}X \ \hat{}\ n * fls\text{-}regpart\ f = fls\text{-}regpart\ (fls\text{-}shift\ (-n)\ f)$
⟨*proof*⟩

lemma *fls-base-factor-uminus*: $fls\text{-}base\text{-}factor\ (-f) = -\ fls\text{-}base\text{-}factor\ f$

⟨*proof*⟩

4.2 Partial evaluation of formal power series

Make a degree-n polynomial out of a formal power series.

definition *poly-of-fps*
where $poly\text{-}of\text{-}fps\ f\ n = Abs\text{-}poly\ (\lambda i.\ if\ i \leq n\ then\ f\$i\ else\ 0)$

lemma *if-then-MOST-zero*:
$\forall_\infty x.\ (if\ P\ x\ then\ f\ x\ else\ 0) = 0$
if $\forall_\infty x.\ \neg\ P\ x$
⟨*proof*⟩

lemma *truncated-fps-eventually-zero*:
$(\lambda i.\ if\ i \leq n\ then\ f\$i\ else\ 0) \in \{g.\ \forall_\infty n.\ g\ n = 0\}$
⟨*proof*⟩

lemma *coeff-poly-of-fps*[*simp*]:
$coeff\ (poly\text{-}of\text{-}fps\ f\ n)\ i = (if\ i \leq n\ then\ f\$i\ else\ 0)$
⟨*proof*⟩

lemma *poly-of-fps-0*[*simp*]: $poly\text{-}of\text{-}fps\ 0\ n = 0$
⟨*proof*⟩

lemma *poly-of-fps-0th-conv-pCons*[*simp*]: $poly\text{-}of\text{-}fps\ f\ 0 = [:f\$0:]$
⟨*proof*⟩

lemma *poly-of-fps-degree*: $degree\ (poly\text{-}of\text{-}fps\ f\ n) \leq n$
⟨*proof*⟩

lemma *poly-of-fps-Suc*:
$poly\text{-}of\text{-}fps\ f\ (Suc\ n) = poly\text{-}of\text{-}fps\ f\ n + monom\ (f\ \$\ Suc\ n)\ (Suc\ n)$
⟨*proof*⟩

lemma *poly-of-fps-Suc-nth-eq0*:
$poly\text{-}of\text{-}fps\ f\ (Suc\ n) = poly\text{-}of\text{-}fps\ f\ n$ **if** $f\ \$\ Suc\ n = 0$
⟨*proof*⟩

lemma *poly-of-fps-Suc-conv-pCons-shift*:
$poly\text{-}of\text{-}fps\ f\ (Suc\ n) = pCons\ (f\$0)\ (poly\text{-}of\text{-}fps\ (fps\text{-}shift\ 1\ f)\ n)$
⟨*proof*⟩

abbreviation $fps\text{-}parteval\ f\ x\ n \equiv poly\ (poly\text{-}of\text{-}fps\ f\ n)\ x$
abbreviation $fls\text{-}base\text{-}parteval\ f \equiv fps\text{-}parteval\ (fls\text{-}base\text{-}factor\text{-}to\text{-}fps\ f)$

lemma *fps-parteval-0*[*simp*]: $fps\text{-}parteval\ 0\ x\ n = 0$
⟨*proof*⟩

lemma *fps-parteval-0th*[*simp*]: $fps\text{-}parteval\ f\ x\ 0 = f\ \$\ 0$

⟨*proof*⟩

lemma *fps-parteval-Suc*:
fps-parteval f x (Suc n) = fps-parteval f x n + (f \$ Suc n) ∗ x ^ Suc n
for *f* :: *'a::comm-semiring-1 fps*
⟨*proof*⟩

lemma *fps-parteval-Suc-cons*:
fps-parteval f x (Suc n) = f\$0 + x ∗ fps-parteval (fps-shift 1 f) x n
⟨*proof*⟩

lemma *fps-parteval-at-0*[*simp*]: *fps-parteval f 0 n = f \$ 0*
⟨*proof*⟩

lemma *fps-parteval-conv-sum*:
fps-parteval f x n = (∑ k≤n. f\$k ∗ x^k) **for** *x* :: *'a::comm-semiring-1*
⟨*proof*⟩

lemma *fls-base-parteval-conv-sum*:
fls-base-parteval f x n = (∑ k≤n. f\$\$(fls-subdegree f + int k) ∗ x^k)
for *f* :: *'a::comm-semiring-1 fls* **and** *x* :: *'a* **and** *n* :: *nat*
⟨*proof*⟩

lemma *fps-parteval-fps-const*[*simp*]:
fps-parteval (fps-const c) x n = c
⟨*proof*⟩

lemma *fps-parteval-1*[*simp*]: *fps-parteval 1 x n = 1*
⟨*proof*⟩

lemma *fps-parteval-mult-by-fps-const*:
fps-parteval (fps-const c ∗ f) x n = c ∗ fps-parteval f x n
⟨*proof*⟩

lemma *fps-parteval-plus*[*simp*]:
fps-parteval (f + g) x n = fps-parteval f x n + fps-parteval g x n
⟨*proof*⟩

lemma *fps-parteval-uminus*[*simp*]:
fps-parteval (−f) x n = − fps-parteval f x n **for** *f* :: *'a::comm-ring fps*
⟨*proof*⟩

lemma *fps-parteval-minus*[*simp*]:
fps-parteval (f − g) x n = fps-parteval f x n − fps-parteval g x n
for *f* :: *'a::comm-ring fps*
⟨*proof*⟩

lemma *fps-parteval-mult-fps-X*:
fixes *f* :: *'a::comm-semiring-1 fps*

shows *fps-parteval* $(f * \textit{fps-X})\ x\ (\textit{Suc}\ n) = x * \textit{fps-parteval}\ f\ x\ n$
and *fps-parteval* $(\textit{fps-X} * f)\ x\ (\textit{Suc}\ n) = x * \textit{fps-parteval}\ f\ x\ n$
⟨*proof*⟩

lemma *fps-parteval-mult-fps-X-power*:
fixes f :: *'a::comm-semiring-1 fps*
assumes $n \geq m$
shows *fps-parteval* $(f * \textit{fps-X}\ \widehat{}\ m)\ x\ n = x\widehat{}m * \textit{fps-parteval}\ f\ x\ (n - m)$
and *fps-parteval* $(\textit{fps-X}\ \widehat{}\ m * f)\ x\ n = x\widehat{}m * \textit{fps-parteval}\ f\ x\ (n - m)$
⟨*proof*⟩

lemma *fps-parteval-mult-right*:
fps-parteval $(f * g)\ x\ n =$
fps-parteval $(\textit{Abs-fps}\ (\lambda i.\ f\$i * \textit{fps-parteval}\ g\ x\ (n - i)))\ x\ n$
⟨*proof*⟩

lemma *fps-parteval-mult-left*:
fps-parteval $(f * g)\ x\ n =$
fps-parteval $(\textit{Abs-fps}\ (\lambda i.\ \textit{fps-parteval}\ f\ x\ (n - i) * g\$i))\ x\ n$
⟨*proof*⟩

lemma *fps-parteval-mult-conv-sum*:
fixes $f\ g$ :: *'a::comm-semiring-1 fps*
shows *fps-parteval* $(f * g)\ x\ n = (\sum k{\leq}n.\ f\$k * \textit{fps-parteval}\ g\ x\ (n - k) * x\widehat{}k)$
and *fps-parteval* $(f * g)\ x\ n = (\sum k{\leq}n.\ \textit{fps-parteval}\ f\ x\ (n - k) * g\$k * x\widehat{}k)$
⟨*proof*⟩

lemma *fps-parteval-fls-base-factor-to-fps-diff*:
fls-base-parteval $(f - g)\ x\ n = \textit{fls-base-parteval}\ f\ x\ n$
if $f \neq 0$ **and** *fls-subdegree* $g >$ *fls-subdegree* $f + (\textit{int}\ n)$
for $f\ g$ :: *'a::comm-ring-1 fls*
⟨*proof*⟩

4.3 Vanishing of formal power series relative to divisibility of all partial sums

definition *fps-pvanishes* :: *'a::comm-semiring-1* ⇒ *'a fps* ⇒ *bool*
where *fps-pvanishes* $p\ f \equiv (\forall\ n{::}nat.\ p\ \widehat{}\ \textit{Suc}\ n\ \textit{dvd}\ (\textit{fps-parteval}\ f\ p\ n))$

lemma *fps-pvanishesD*: *fps-pvanishes* $p\ f \Longrightarrow p\ \widehat{}\ \textit{Suc}\ n\ \textit{dvd}\ (\textit{fps-parteval}\ f\ p\ n)$
⟨*proof*⟩

lemma *fps-pvanishesI*:
fps-pvanishes $p\ f$ **if** $\bigwedge n{::}nat.\ p\ \widehat{}\ \textit{Suc}\ n\ \textit{dvd}\ (\textit{fps-parteval}\ f\ p\ n)$
⟨*proof*⟩

lemma *fps-pvanishes-0*[*simp*]: *fps-pvanishes* $p\ 0$
⟨*proof*⟩

lemma *fps-pvanishes-at-0*[*simp*]: *fps-pvanishes 0 f* = (*f*\$*0* = *0*)
⟨*proof*⟩

lemma *fps-pvanishes-at-unit*: *fps-pvanishes p f* **if** *is-unit p*
⟨*proof*⟩

lemma *fps-pvanishes-one-iff*:
fps-pvanishes p 1 = *is-unit p*
⟨*proof*⟩

lemma *fps-pvanishes-uminus*[*simp*]:
fps-pvanishes p (−*f*) **if** *fps-pvanishes p f* **for** *f* :: ′*a*::*comm-ring-1 fps*
⟨*proof*⟩

lemma *fps-pvanishes-add*[*simp*]:
fps-pvanishes p (*f* + *g*) **if** *fps-pvanishes p f* **and** *fps-pvanishes p g*
for *f* :: ′*a*::*comm-semiring-1 fps*
⟨*proof*⟩

lemma *fps-pvanishes-minus*[*simp*]:
fps-pvanishes p (*f* − *g*) **if** *fps-pvanishes p f* **and** *fps-pvanishes p g*
for *f* :: ′*a*::*comm-ring-1 fps*
⟨*proof*⟩

lemma *fps-pvanishes-mult*: *fps-pvanishes p* (*f* ∗ *g*) **if** *fps-pvanishes p g*
⟨*proof*⟩

lemma *fps-pvanishes-mult-fps-X-iff*:
assumes (*p*::′*a*::*idom*) ≠ *0*
shows *fps-pvanishes p f* ⟷ *fps-pvanishes p* (*f* ∗ *fps-X*)
and *fps-pvanishes p f* ⟷ *fps-pvanishes p* (*fps-X* ∗ *f*)
⟨*proof*⟩

lemma *fps-pvanishes-mult-fps-X-power-iff*:
assumes (*p*::′*a*::*idom*) ≠ *0*
shows *fps-pvanishes p f* ⟷ *fps-pvanishes p* (*f* ∗ *fps-X* ^ *n*)
and *fps-pvanishes p f* ⟷ *fps-pvanishes p* (*fps-X* ^ *n* ∗ *f*)
⟨*proof*⟩

4.4 Equivalence and depth of formal Laurent series relative to a prime

4.4.1 Definition of equivalence and basic facts

abbreviation *fps-primevanishes p* ≡ *fps-pvanishes* (*Rep-prime p*)

overloading
p-equal-fls ≡
p-equal :: ′*a*::*nontrivial-factorial-comm-ring prime* ⇒
′*a fls* ⇒ ′*a fls* ⇒ *bool*

begin

definition *p-equal-fls* ::
  $'a$::*nontrivial-factorial-comm-ring prime* $\Rightarrow$
    $'a$ *fls* $\Rightarrow$ $'a$ *fls* $\Rightarrow$ *bool*
  where
    p-equal-fls p f g $\equiv$ *fps-primevanishes p* (*fls-base-factor-to-fps* $(f - g)$)

end

context
  fixes p :: $'a$ :: *nontrivial-factorial-comm-ring prime*
begin

lemma *p-equal-flsD*:
  $f \simeq_p g \Longrightarrow$ *fps-primevanishes p* (*fls-base-factor-to-fps* $(f - g)$)
  for f g :: $'a$ *fls*
  ⟨*proof*⟩

lemma *p-equal-fls-imp-p-dvd*: *p pdvd* $((f - g)$ $$ (*fls-subdegree* $(f - g)))$
  if $f \simeq_p g$
  for f g :: $'a$ *fls*
⟨*proof*⟩

lemma *p-equal-flsI*:
  fps-primevanishes p (*fls-base-factor-to-fps* $(f - g)$) $\Longrightarrow f \simeq_p g$
  for f g :: $'a$ *fls*
  ⟨*proof*⟩

lemma *p-equal-fls-conv-p-equal-0*:
  $f \simeq_p g \longleftrightarrow (f - g) \simeq_p 0$
  for f g :: $'a$ *fls*
  ⟨*proof*⟩

lemma *p-equal-fls-refl*: $f \simeq_p f$ **for** f:: $'a$ *fls*
  ⟨*proof*⟩

lemma *p-equal-fls-sym*:
  $f \simeq_p g \Longrightarrow g \simeq_p f$ **for** f g :: $'a$ *fls*
  ⟨*proof*⟩

lemma *p-equal-fls-shift-iff*:
  $f \simeq_p g \longleftrightarrow$ (*fls-shift n f*) $\simeq_p$ (*fls-shift n g*)
  for f g :: $'a$ *fls*
  ⟨*proof*⟩

lemma *p-equal-fls-extended-sum-iff*:
  $f \simeq_p g \longleftrightarrow$
    ($\forall$ n. (*Rep-prime p*) ^ *Suc n dvd*

$(\sum k \leq n.\ (f - g)\$\$(N + int\ k) * (Rep\text{-}prime\ p)\hat{}k))$
if $N \leq min\ (fls\text{-}subdegree\ f)\ (fls\text{-}subdegree\ g)$
⟨*proof*⟩

lemma *p-equal-fls-extended-intsum-iff*:
$f \simeq_p g \longleftrightarrow$
$(\forall n \geq N.\ (Rep\text{-}prime\ p)\ \hat{}\ Suc\ (nat\ (n - N))\ dvd$
$(\sum k{=}N..n.\ (f - g)\$\$k * (Rep\text{-}prime\ p)\ \hat{}\ nat\ (k - N)))$
if $N \leq min\ (fls\text{-}subdegree\ f)\ (fls\text{-}subdegree\ g)$
⟨*proof*⟩

lemma *fls-p-equal0-extended-sum-iff*:
$f \simeq_p 0 \longleftrightarrow$
$(\forall n.\ (Rep\text{-}prime\ p)\ \hat{}\ Suc\ n\ dvd$
$(\sum k \leq n.\ f\$\$(N + int\ k) * (Rep\text{-}prime\ p)\hat{}k))$
if $N \leq fls\text{-}subdegree\ f$
⟨*proof*⟩

lemma *fls-p-equal0-extended-intsum-iff*:
$f \simeq_p 0 \longleftrightarrow$
$(\forall n \geq N.\ (Rep\text{-}prime\ p)\ \hat{}\ Suc\ (nat\ (n - N))\ dvd$
$(\sum k{=}N..n.\ f\$\$k * (Rep\text{-}prime\ p)\ \hat{}\ nat\ (k - N)))$
if $N \leq fls\text{-}subdegree\ f$
⟨*proof*⟩

end

context
fixes $p :: 'a :: nontrivial\text{-}factorial\text{-}idom\ prime$
begin

lemma *fls-1-not-p-equal-0*: $(1::'a\ fls) \neg\simeq_p 0$
⟨*proof*⟩

lemma *p-equal-fls0-nonneg-subdegree*:
$f \simeq_p 0 \longleftrightarrow fps\text{-}primevanishes\ p\ (fls\text{-}regpart\ f)$
if $fls\text{-}subdegree\ f \geq 0$
⟨*proof*⟩

lemma *p-equal-fls-nonneg-subdegree*:
$f \simeq_p g \longleftrightarrow fps\text{-}primevanishes\ p\ (fls\text{-}regpart\ (f - g))$
if $fls\text{-}subdegree\ f \geq 0$ **and** $fls\text{-}subdegree\ g \geq 0$
⟨*proof*⟩

lemma *p-equal-fls-trans-to-0*:
$f \simeq_p g$ **if** $f \simeq_p 0$ **and** $g \simeq_p 0$ **for** $f\ g :: 'a\ fls$
⟨*proof*⟩

lemma *p-equal-fls-trans*:

$f \simeq_p g \Longrightarrow g \simeq_p h \Longrightarrow$
$f \simeq_p h$
for $f\ g\ h$ :: *'a fls*
⟨*proof*⟩

end

global-interpretation *p-equality-fls*:
p-equality-1
p-equal :: *'a*::*nontrivial-factorial-idom prime* ⇒
'a fls ⇒ *'a fls* ⇒ *bool*
⟨*proof*⟩

overloading
globally-p-equal-fls ≡
globally-p-equal ::
'a::*nontrivial-factorial-idom fls* ⇒ *'a fls* ⇒ *bool*
begin

definition *globally-p-equal-fls* ::
'a::*nontrivial-factorial-idom fls* ⇒ *'a fls* ⇒ *bool*
where *globally-p-equal-fls*[*simp*]:
globally-p-equal-fls = *p-equality-fls.globally-p-equal*

end

context
fixes *p* :: *'a* :: *nontrivial-factorial-idom prime*
begin

lemma *p-equal-fls-const-iff*:
(*fls-const c*) $\simeq_p$ (*fls-const d*) ⟷ $c = d$ **for** $c\ d$ :: *'a*
⟨*proof*⟩

lemma *p-equal-fls-const-0-iff*:
(*fls-const c*) $\simeq_p$ *0* ⟷ $c = 0$ **for** c :: *'a*
⟨*proof*⟩

lemma *p-equal-fls-X-intpow-iff*:
((*fls-X-intpow m* :: *'a fls*) $\simeq_p$ (*fls-X-intpow n*)) = ($m = n$)
⟨*proof*⟩

lemma *fls-X-intpow-nequiv0*: (*fls-X-intpow m* :: *'a fls*) $\neg\simeq_p$ *0*
⟨*proof*⟩

end

4.4.2 Depth as maximum subdegree of equivalent series

Each nonzero equivalence class will have a maximum subdegree.

lemma *no-greatest-p-equal-subdegree*:
  $f \simeq_p 0$
  if $\forall n :: int.\ (\exists g.\ \textit{fls-subdegree}\ g > n \wedge f \simeq_p g)$
  for $p :: 'a::\textit{nontrivial-factorial-comm-ring prime}$ **and** $f :: 'a\ \textit{fls}$
⟨*proof*⟩

abbreviation
  p-equal-fls-simplified-set p f ≡
    $\{g.\ f \simeq_p g \wedge \textit{fls-subdegree}\ g \geq \textit{fls-subdegree}\ f\}$
— We restrict to those equivalent series that represent a simplification of the given series so that the image of the collection under taking subdegrees is finite.

context
  fixes $p :: 'a::\textit{nontrivial-factorial-comm-ring prime}$
  and $f :: 'a\ \textit{fls}$
begin

lemma *p-equal-fls-simplifed-nempty*:
  fls-subdegree ‘ (*p-equal-fls-simplified-set p f*) $\neq \{\}$
  ⟨*proof*⟩

lemma *p-equal-fls-simplifed-finite*:
  finite (*fls-subdegree* ‘ (*p-equal-fls-simplified-set p f*)) **if** $f \neg\simeq_p 0$
⟨*proof*⟩

end

overloading
  p-depth-fls ≡
    $\textit{p-depth} :: 'a::\textit{nontrivial-factorial-comm-ring prime} \Rightarrow 'a\ \textit{fls} \Rightarrow \textit{int}$
  p-depth-const ≡
    $\textit{p-depth} :: 'a::\textit{nontrivial-factorial-comm-ring prime} \Rightarrow 'a \Rightarrow \textit{int}$
begin

definition
  $\textit{p-depth-fls} :: 'a::\textit{nontrivial-factorial-comm-ring prime} \Rightarrow 'a\ \textit{fls} \Rightarrow \textit{int}$
  where
  p-depth-fls p f ≡
    (*if* $f \simeq_p 0$ *then 0 else Max* (*fls-subdegree* ‘ (*p-equal-fls-simplified-set p f*)))

definition
  $\textit{p-depth-const} :: 'a::\textit{nontrivial-factorial-comm-ring prime} \Rightarrow 'a \Rightarrow \textit{int}$
  where
  p-depth-const p c ≡ *p-depth p* (*fls-const c*)

end

context
  fixes p :: $'a$::*nontrivial-factorial-comm-ring prime*
begin

lemma *p-depth-fls-eqI*:
  $f^{\circ p}$ = *fls-subdegree* g
  if $f \neg\simeq_p 0$ **and** $f \simeq_p g$
  and $\bigwedge h.\ f \simeq_p h \Longrightarrow$ *fls-subdegree* $h \leq$ *fls-subdegree* g
  for f g :: $'a$ *fls*
⟨*proof*⟩

lemma *p-depth-fls-p-equal0*[*simp*]:
  $(f^{\circ p}) = 0$ **if** $f \simeq_p 0$ **for** f :: $'a$ *fls*
  ⟨*proof*⟩

lemma *p-depth-fls0*[*simp*]: $(0::'a\ fls)^{\circ p} = 0$
  ⟨*proof*⟩

lemma *p-depth-fls-n0D*:
  $f^{\circ p}$ = *Max* (*fls-subdegree* ‘ (*p-equal-fls-simplified-set* p f))
  if $f \neg\simeq_p 0$
  for f :: $'a$ *fls*
  ⟨*proof*⟩

lemma *p-depth-flsE*:
  fixes f :: $'a$ *fls*
  assumes $f \neg\simeq_p 0$
  obtains g **where** $f \simeq_p g$ **and** *fls-subdegree* $g = (f^{\circ p})$
  and $\forall h.\ f \simeq_p h \longrightarrow$ *fls-subdegree* $h \leq$ *fls-subdegree* g
⟨*proof*⟩

lemma *p-depth-fls-ge-p-equal-subdegree*:
  $(f^{\circ p}) \geq$ *fls-subdegree* g
  if $f \neg\simeq_p 0$ **and** $f \simeq_p g$
  for f g :: $'a$ *fls*
⟨*proof*⟩

lemma *p-depth-fls-ge-p-equal-subdegree′*:
  $(f^{\circ p}) \geq$ *fls-subdegree* f **if** $f \neg\simeq_p 0$ **for** f g :: $'a$ *fls*
  ⟨*proof*⟩

lemma *p-equal-fls-simplified-set-shift*:
  p-equal-fls-simplified-set p (*fls-shift* n f) = *fls-shift* n ‘ *p-equal-fls-simplified-set* p f
  if $f \neg\simeq_p 0$
  for f :: $'a$ *fls*
⟨*proof*⟩

lemma *p-equal-fls-simplified-set-shift-subdegrees*:
fls-subdegree ‘ *p-equal-fls-simplified-set p* (*fls-shift n f*) =
(λ*x*. *x* − *n*) ‘ *fls-subdegree* ‘ *p-equal-fls-simplified-set p f*
if $f \neg\simeq_p 0$
for *f* :: ′*a fls*
⟨*proof*⟩

lemma *p-depth-fls-shift*:
$(\textit{fls-shift}\ n\ f)^{\circ p} = (f^{\circ p}) - n$ **if** $f \neg\simeq_p 0$
for *f* :: ′*a fls*
⟨*proof*⟩

end

context
fixes *p* :: ′*a*::*nontrivial-factorial-idom prime*
and *f g* :: ′*a fls*
begin

lemma *p-depth-fls-p-equal*: $(f^{\circ p}) = (g^{\circ p})$ **if** $f \simeq_p g$
⟨*proof*⟩

lemma *p-depth-fls-can-simplify-iff*:
p pdvd g $$ (*fls-subdegree g*) = (*fls-subdegree g* < $(f^{\circ p})$)
if *f-nequiv0*: $f \neg\simeq_p 0$
and *fg-equiv* : $f \simeq_p g$
⟨*proof*⟩

lemma *p-depth-fls-rep-iff*:
$(f^{\circ p})$ = *fls-subdegree g* ⟷
¬ *p pdvd g* $$ (*fls-subdegree g*)
if $f \neg\simeq_p 0$ **and** $f \simeq_p g$
⟨*proof*⟩

end

global-interpretation *p-depth-fls*:
p-equality-depth-no-zero-divisors-1
p-equal :: ′*a*::*nontrivial-factorial-idom prime* ⇒
′*a fls* ⇒ ′*a fls* ⇒ *bool*
p-depth :: ′*a prime* ⇒ ′*a fls* ⇒ *int*
⟨*proof*⟩

context
fixes *p* :: ′*a*::*nontrivial-factorial-idom prime*
begin

lemma *p-depth-fls1*[*simp*]: $(1::'a\ \textit{fls})^{\circ p} = 0$
⟨*proof*⟩

lemma *p-depth-fls-X*: $(\textit{fls-X}::'a\ \textit{fls})^{\circ p} = \textit{1}$
⟨*proof*⟩

lemma *p-depth-fls-X-inv*: $(\textit{fls-X-inv}::'a\ \textit{fls})^{\circ p} = -\textit{1}$
⟨*proof*⟩

lemma *p-depth-fls-X-intpow*: $((\textit{fls-X-intpow}\ n)::'a\ \textit{fls})^{\circ p} = n$
⟨*proof*⟩

lemma *p-depth-const*:
$c^{\circ p} = \textit{pmultiplicity}\ p\ c$ **for** $c :: 'a$
⟨*proof*⟩

lemma *p-depth-prime-const*: $(\textit{Rep-prime}\ p)^{\circ p} = \textit{1}$
⟨*proof*⟩

end

4.5 Reduction relative to a prime

4.5.1 Of scalars

class *nontrivial-euclidean-ring-cancel* = *nontrivial-factorial-comm-ring* + *euclidean-semiring-cancel*
begin
subclass *nontrivial-factorial-idom* ⟨*proof*⟩
subclass *euclidean-ring-cancel* ⟨*proof*⟩
end

instance *int* :: *nontrivial-euclidean-ring-cancel* ⟨*proof*⟩

abbreviation *pdiv* ::
$'a \Rightarrow 'a$::*nontrivial-euclidean-ring-cancel prime* $\Rightarrow 'a$ (**infix** *pdiv 70*)
where *a pdiv p* ≡ *a div* (*Rep-prime p*)

consts *p-modulo* :: $'a \Rightarrow 'b$ *prime* $\Rightarrow 'a$ (**infixl** *pmod 70*)

overloading *p-modulo-scalar* ≡ *p-modulo* :: $'a \Rightarrow 'a$ *prime* $\Rightarrow 'a$
begin
definition *p-modulo-scalar* ::
$'a$::*nontrivial-euclidean-ring-cancel* $\Rightarrow 'a$ *prime* $\Rightarrow 'a$
where *p-modulo-scalar-def*[*simp*]: *p-modulo-scalar a p* ≡ *a mod* (*Rep-prime p*)
end

class *nontrivial-unique-euclidean-ring* = *nontrivial-factorial-comm-ring* + *unique-euclidean-semiring*
+ **assumes** *division-segment-prime*: *prime p* $\Longrightarrow$ *division-segment p* = *1*
begin
subclass *nontrivial-euclidean-ring-cancel* ⟨*proof*⟩
end

instance *int* :: *nontrivial-unique-euclidean-ring*
⟨*proof*⟩

lemma *division-segment-1*: *division-segment* (*1*::′*a*::*unique-euclidean-semiring*) = *1*
⟨*proof*⟩

lemma *one-pdiv*: (*1*::′*a*) *pdiv p* = *0*
and *one-pmod*: (*1*::′*a*) *pmod p* = *1*
for *p* :: ′*a*::*nontrivial-unique-euclidean-ring prime*
⟨*proof*⟩

4.5.2 Inductive partial reduction of formal Laurent series

This inductive function reduces one term at a time. The *nat* argument specifies how far beyond the subdegree to reduce, so that all terms to *one less* than the *nat* argument beyond the subdegree are reduced, and the term at the *nat* argument beyond the subdegree is updated to include the carry.

primrec *fls-partially-p-modulo-rep* ::
′*a*::*nontrivial-euclidean-ring-cancel fls* ⇒
nat ⇒ ′*a prime* ⇒ (*int* ⇒ ′*a*)
where *fls-partially-p-modulo-rep f 0 p* = ($$) *f* |
fls-partially-p-modulo-rep f (*Suc N*) *p* =
(*fls-partially-p-modulo-rep f N p*)(
fls-subdegree f + *int N* :=
(*fls-partially-p-modulo-rep f N p* (*fls-subdegree f* + *int N*)) *pmod p*,
fls-subdegree f + *int N* + *1* :=
f $$ (*fls-subdegree f* + *int N* + *1*) +
(*fls-partially-p-modulo-rep f N p* (*fls-subdegree f* + *int N*)) *pdiv p*
)

lemma *fls-partially-p-modulo-rep-zero*[*simp*]:
fls-partially-p-modulo-rep 0 N p n = *0*
⟨*proof*⟩

lemma *fls-partially-p-modulo-rep-zeros-below*:
fls-partially-p-modulo-rep f N p n = *0* **if** *n* < *fls-subdegree f*
⟨*proof*⟩

lemma *fls-partially-p-modulo-rep-func-lower-bound*:
∀ *n*<*fls-subdegree f*. *fls-partially-p-modulo-rep f* (*nat* (*n* + *1* − *fls-subdegree f*)) *p n* = *0*
⟨*proof*⟩

lemma *fls-partially-p-modulo-rep-agrees*:
fls-partially-p-modulo-rep f N p n = *fls-partially-p-modulo-rep f M p n*
if *n* ≤ *fls-subdegree f* + *int N* − *1* **and** *M* ≥ *N*
⟨*proof*⟩

lemma *fls-partially-p-modulo-rep-func-partially*:
  $n > \textit{fls-subdegree } f + \textit{int } N \Longrightarrow \textit{fls-partially-p-modulo-rep } f\ N\ p\ n = f\ \$\$\ n$
  ⟨*proof*⟩

lemma *fls-partially-p-modulo-rep-cong*:
  $\textit{fls-partially-p-modulo-rep } f\ N\ p\ n = \textit{fls-partially-p-modulo-rep } g\ N\ p\ n$
  if $\textit{fls-subdegree } f = \textit{fls-subdegree } g$ **and** $\forall k{\le}n.\ f\ \$\$\ k = g\ \$\$\ k$
⟨*proof*⟩

lemma *fls-partially-reduced-nth-vs-rep*:
  $N \le n + 1 - \textit{fls-subdegree } f \Longrightarrow \textit{fls-partially-p-modulo-rep } f\ N\ p = (\$\$)\ f$
  if $\forall k{\le}n.\ (f\ \$\$\ k)\ \textit{pdiv}\ p = 0$
⟨*proof*⟩

lemma *fls-partially-p-modulo-rep-carry*:
  $(\sum k{=}\textit{fls-subdegree } f..N.$
    $(f\$\$k - \textit{fls-partially-p-modulo-rep } f\ (\textit{nat}\ (N - \textit{fls-subdegree } f))\ p\ k) *$
      $(\textit{Rep-prime } p)\ \widehat{}\ \textit{nat}\ (k - \textit{fls-subdegree } f)) = 0$
⟨*proof*⟩

4.5.3 Full reduction of formal Laurent series

Now we collect the reduced terms into a formal Laurent series.

overloading *p-modulo-fls* $\equiv$ *p-modulo* :: *'a fls* $\Rightarrow$ *'a prime* $\Rightarrow$ *'a fls*
begin
definition *p-modulo-fls* ::
  'a::nontrivial-euclidean-ring-cancel fls $\Rightarrow$ *'a prime* $\Rightarrow$ *'a fls*
  where
    p-modulo-fls f p $\equiv$
      $\textit{Abs-fls}\ (\lambda n.\ \textit{fls-partially-p-modulo-rep } f\ (\textit{nat}\ (n + 1 - \textit{fls-subdegree } f))\ p\ n)$
end

lemma *p-modulo-fls-nth*:
  $(f\ \textit{pmod}\ p)\ \$\$\ n = \textit{fls-partially-p-modulo-rep } f\ (\textit{nat}\ (n + 1 - \textit{fls-subdegree } f))\ p$
n
  ⟨*proof*⟩

lemma *p-modulo-fls-nth'*:
  $\textit{int } N \ge n + 1 - \textit{fls-subdegree } f \Longrightarrow$
    $(f\ \textit{pmod}\ p)\ \$\$\ n = \textit{fls-partially-p-modulo-rep } f\ N\ p\ n$
⟨*proof*⟩

context
  fixes *p* :: *'a::nontrivial-euclidean-ring-cancel prime*
begin

lemma *fls-pmod-reduced*[*simp*]: $((f\ \textit{pmod}\ p)\ \$\$\ n)\ \textit{pdiv}\ p = 0$
  ⟨*proof*⟩

lemma *fls-pmod-pmod-nth*[*simp*]: $((f\ pmod\ p)\ \$\$\ n)\ pmod\ p = (f\ pmod\ p)\ \$\$\ n$ **for** $f :: 'a\ fls$
⟨*proof*⟩

lemma *fls-pmod-carry*:
$(\sum k{=}fls\text{-}subdegree\ f..N.\ (f - (f\ pmod\ p))\ \$\$\ k * (Rep\text{-}prime\ p)\ \hat{}\ nat\ (k - fls\text{-}subdegree\ f)) =$
$(fls\text{-}partially\text{-}p\text{-}modulo\text{-}rep\ f\ (nat\ (N + 1 - fls\text{-}subdegree\ f))\ p\ (N + 1) - f\ \$\$\ (N + 1)) *$
$(Rep\text{-}prime\ p)\ \hat{}\ nat\ (N + 1 - fls\text{-}subdegree\ f)$
⟨*proof*⟩

lemma *fls-pmod-carry′*:
$(\sum k{=}fls\text{-}subdegree\ f..N.\ (f - (f\ pmod\ p))\ \$\$\ k * (Rep\text{-}prime\ p)\ \hat{}\ nat\ (k - fls\text{-}subdegree\ f)) =$
$(fls\text{-}partially\text{-}p\text{-}modulo\text{-}rep\ f\ (nat\ (N - fls\text{-}subdegree\ f))\ p\ N\ pdiv\ p) *$
$(Rep\text{-}prime\ p)\ \hat{}\ Suc\ (nat\ (N - fls\text{-}subdegree\ f))$
if $N \geq fls\text{-}subdegree\ f$
⟨*proof*⟩

lemma *fls-p-reduced-is-zero-iff*:
$f = 0 \longleftrightarrow f \simeq_p 0 \wedge (\forall n.\ f\$\$n\ pdiv\ p = 0)$
for $f :: 'a\ fls$
⟨*proof*⟩

lemma *p-modulo-fls-eq0*[*simp*]:
$f\ pmod\ p = 0$ **if** $f \simeq_p 0$ **for** $f :: 'a\ fls$
⟨*proof*⟩

lemma *fls-pmod-eq0-iff*:
$f\ pmod\ p = 0 \longleftrightarrow f \simeq_p 0$ **for** $f :: 'a\ fls$
⟨*proof*⟩

lemma *fls-pmod-equiv*: $f \simeq_p f\ pmod\ p$ **for** $f :: 'a\ fls$
⟨*proof*⟩

lemma *fls-p-reduced-subdegree-unique*:
$fls\text{-}subdegree\ g = (f^{\circ p})$
if $f \simeq_p g$ **and** $\forall n.\ (g\ \$\$\ n)\ pdiv\ p = 0$
for $f\ g :: 'a\ fls$
⟨*proof*⟩

lemma *fls-pmod-subdegree*:
$fls\text{-}subdegree\ (f\ pmod\ p) = (f^{\circ p})$ **for** $f :: 'a\ fls$
⟨*proof*⟩

lemma *fls-pmod-depth*:
$(f\ pmod\ p)^{\circ p} = (f^{\circ p})$ **for** $f :: 'a\ fls$

⟨*proof*⟩

lemma *fls-pmod-subdegree-ge*:
fls-subdegree (*f pmod p*) $\geq$ *fls-subdegree f* **if** $f \neg\simeq_p 0$
for *f* :: *'a fls*
⟨*proof*⟩

lemma *fls-pmod-nth-below-subdegree*:
(*f pmod p*) \$\$ *n* = *0* **if** *n* < *fls-subdegree f* **for** *f* :: *'a fls*
⟨*proof*⟩

lemma *fls-pmod-nth-cong*:
(*f pmod p*) \$\$ *n* = (*g pmod p*) \$\$ *n* **if** ∀ *k*≤*n*. *f* \$\$ *k* = *g* \$\$ *k* **for** *f g* :: *'a fls*
⟨*proof*⟩

lemma *fls-partially-reduced-nth-vs-pmod*:
(*f pmod p*) \$\$ *n* = *f* \$\$ *n* **if** ∀ *k*≤*n*. (*f* \$\$ *k*) *pdiv p* = *0*
⟨*proof*⟩

lemma *fls-pmod-eqI*:
f pmod p = *g* **if** $f \simeq_p g$ **and** ∀ *n*. (*g* \$\$ *n*) *pdiv p* = *0*
⟨*proof*⟩

lemma *fls-pmod-eq-pmod-iff*:
f pmod p = *g pmod p* ⟷ $f \simeq_p g$ **for** *f g* :: *'a fls*
⟨*proof*⟩

lemma *fls-pmod-equiv-pmod-iff*:
(*f pmod p*) $\simeq_p$ (*g pmod p*) ⟷ $f \simeq_p g$
for *f g* :: *'a fls*
⟨*proof*⟩

lemma *fls-p-modulo-iff*:
f pmod p = *g* ⟷ $f \simeq_p g$ ∧ (∀ *n*. (*g* \$\$ *n*) *pdiv p* = *0*)
⟨*proof*⟩

lemma *fls-pmod-pmod*[*simp*]: (*f pmod p*) *pmod p* = *f pmod p* **for** *f* :: *'a fls*
⟨*proof*⟩

lemma *fls-pmod-cong*:
f pmod p = *g pmod p* **if** $f \simeq_p g$ **for** *f g* :: *'a fls*
⟨*proof*⟩

end

4.5.4 Algebraic properties of reduction

lemma *fls-pmod-1*[*simp*]:
(*1*::*'a fls*) *pmod p* = *1* **for** *p* :: *'a*::*nontrivial-unique-euclidean-ring prime*

⟨*proof*⟩

context
fixes p :: $'a$::*nontrivial-euclidean-ring-cancel prime*
begin

lemma *fls-pmod-0*: $(0::'a\ fls)$ *pmod* $p = 0$
⟨*proof*⟩

lemma *fls-pmod-uminus-equiv*: $(-f)$ *pmod* $p \simeq_p -(f$ *pmod* $p)$ **for** f :: $'a$ *fls*
⟨*proof*⟩

lemma *fls-pmod-equiv-add*:
$(f + g)$ *pmod* $p = (f' + g')$ *pmod* p **if** $f \simeq_p f'$ **and** $g \simeq_p g'$
for $f\ f'\ g\ g'$ :: $'a$ *fls*
⟨*proof*⟩

lemma *fls-pmod-const-add*:
(*fls-const* $c + f$) *pmod* p = *fls-const* $c + f$ *pmod* p **if** *fls-subdegree* $f > 0$ **and** c *pdiv* $p = 0$
for c :: $'a$ **and** f :: $'a$ *fls*
⟨*proof*⟩

lemma *fls-pmod-add-equiv*:
$(f + g)$ *pmod* $p \simeq_p (f$ *pmod* $p) + (g$ *pmod* $p)$ **for** $f\ g$ :: $'a$ *fls*
⟨*proof*⟩

lemma *fls-pmod-mismatched-add-nth*:
$n <$ *fls-subdegree* $g \Longrightarrow ((f + g)$ *pmod* $p)$ \$\$ $n = (f$ *pmod* $p)$ \$\$ n
$((f + g)$ *pmod* $p)$ \$\$ (*fls-subdegree* g) =
$((f$ *pmod* $p)$ \$\$ (*fls-subdegree* g) + g \$\$ (*fls-subdegree* g)) *pmod* p
if *fls-subdegree* g > *fls-subdegree* f
for f :: $'a$ *fls*
⟨*proof*⟩

lemma *fls-pmod-equiv-minus*:
$(f - g)$ *pmod* $p = (f' - g')$ *pmod* p **if** $f \simeq_p f'$ **and** $g \simeq_p g'$
for $f\ f'\ g\ g'$ :: $'a$ *fls*
⟨*proof*⟩

lemma *fls-pmod-minus-equiv*:
$(f - g)$ *pmod* $p \simeq_p ((f$ *pmod* $p) - (g$ *pmod* $p))$ **for** $f\ g$ :: $'a$ *fls*
⟨*proof*⟩

lemma *fls-pmod-equiv-times*:
$(f * g)$ *pmod* $p = (f' * g')$ *pmod* p **if** $f \simeq_p f'$ **and** $g \simeq_p g'$
for $f\ f'\ g\ g'$ :: $'a$ *fls*
⟨*proof*⟩

lemma *fls-pmod-times-equiv*:
$(f * g)$ *pmod* $p \simeq_p (f$ *pmod* $p) * (g$ *pmod* $p)$ **for** $f\ g :: 'a$ *fls*
⟨*proof*⟩

lemma *fls-pmod-diff-cancel*:
$(f$ *pmod* $p)$ $$ $n = (g$ *pmod* $p)$ $$ n **if** $n < ((f - g)^{\circ p})$
for $f\ g :: 'a$ *fls*
⟨*proof*⟩

lemma *fls-pmod-diff-cancel-iff*:
$((f - g)^{\circ p}) > n \longleftrightarrow$
$(\forall k \le n. (f$ *pmod* $p)$ $$ $k = (g$ *pmod* $p)$ $$ $k)$
if $f \neg\simeq_p g$
for $f\ g :: 'a$ *fls*
⟨*proof*⟩

end

4.6 Inversion relative to a prime

4.6.1 Of scalars

class *bezout-semiring* = *semiring-gcd* +
assumes *bezout*: $\exists u\ v.\ u * x + v * y = gcd\ x\ y$

class *bezout-ring* = *ring-gcd* +
assumes *bezout*: $\exists u\ v.\ u * x + v * y = gcd\ x\ y$
begin
subclass *bezout-semiring* ⟨*proof*⟩
subclass *idom* ⟨*proof*⟩
end

instance *int* :: *bezout-ring* ⟨*proof*⟩

class *nontrivial-factorial-euclidean-bezout* = *nontrivial-euclidean-ring-cancel* + *bezout-semiring*
begin
subclass *bezout-ring* ⟨*proof*⟩
end

class *nontrivial-factorial-unique-euclidean-bezout* =
nontrivial-unique-euclidean-ring + *bezout-semiring*
begin
subclass *nontrivial-factorial-euclidean-bezout* ⟨*proof*⟩
end

instance *int* :: *nontrivial-factorial-unique-euclidean-bezout* ⟨*proof*⟩

lemma *prime-residue-inverse-ex1*:
$\exists!x.\ x$ *pdiv* $p = 0 \wedge (x * a)$ *pmod* $p = 1$ **if** $\neg\ p$ *pdvd* a

for a :: $'a$::*nontrivial-factorial-unique-euclidean-bezout* **and** p :: $'a$ *prime*
⟨*proof*⟩

consts *pinverse* :: $'a \Rightarrow 'b$ *prime* $\Rightarrow 'a$
((-$^{-1-}$) [*51*, *51*] *50*)

overloading *pinverse-scalar* $\equiv$ *pinverse* :: $'a \Rightarrow 'a$ *prime* $\Rightarrow 'a$
begin
definition *pinverse-scalar* ::
$'a$::*nontrivial-factorial-unique-euclidean-bezout* $\Rightarrow 'a$ *prime* $\Rightarrow 'a$
where
pinverse-scalar a p $\equiv$
(*if p pdvd a then 0 else*
(*THE x. x pdiv p* $= 0 \wedge (x * a)$ *pmod p* $= 1$))
end

context
fixes p :: $'a$::*nontrivial-factorial-unique-euclidean-bezout prime*
begin

lemma *pinverse-scalar*: $((a^{-1p}) * a)$ *pmod p* $= 1$ **if** $\neg$ *p pdvd a* **for** a :: $'a$
⟨*proof*⟩

lemma *pinverse-scalar-reduced*: (a^{-1p}) *pdiv p* $= 0$ **for** a :: $'a$
⟨*proof*⟩

lemma *pinverse-scalar-nzero*: $(a^{-1p}) \neq 0$ **if** $\neg$ *p pdvd a* **for** a :: $'a$
⟨*proof*⟩

lemma *pinverse-scalar-zero*[*simp*]: $((0::'a)^{-1p}) = 0$
⟨*proof*⟩

lemma *pinverse-scalar-one*[*simp*]: $((1::'a)^{-1p}) = 1$
⟨*proof*⟩

end

4.6.2 Inductive partial inversion of formal Laurent series

Inductively define a depth-zero partial inverse representative function. It is effectively a polynomial of degree equal to the *nat* argument.

primrec *fls-partially-pinverse-rep* ::
$'a$::*nontrivial-factorial-unique-euclidean-bezout fls* $\Rightarrow$ *nat* $\Rightarrow$
$'a$ *prime* $\Rightarrow$ (*int* $\Rightarrow 'a$)
where
fls-partially-pinverse-rep f 0 p n $=$
(*if* $n = 0$
then ((*f pmod p*) \$\$ $(f^{\circ p}))^{-1p}$
else *0*)

| *fls-partially-pinverse-rep f* (*Suc N*) *p* =
(*fls-partially-pinverse-rep f N p*)(
int (*Suc N*) := −(
(
(
fls-shift ($f^{\circ p}$) (*Abs-fls* (*fls-partially-pinverse-rep f N p*)) ∗
(*f pmod p*)
) *pmod p*
) $$ (*int* (*Suc N*)) ∗ (((*f pmod p*) $$ ($f^{\circ p}$))$^{-1p}$)
) *pmod p*
)

lemma *fls-partially-pinverse-rep-func-lower-bound*:
∀ *n*<*0*. *fls-partially-pinverse-rep f* (*nat n*) *p n* = *0*
⟨*proof*⟩

lemma *fls-partially-pinverse-rep-at-0*:
fls-partially-pinverse-rep f N p 0 = (((*f pmod p*) $$ ($f^{\circ p}$))$^{-1p}$)
⟨*proof*⟩

lemma *fls-partially-pinverse-rep-func-lower-bound'*:
∀ *n*<*0*. *fls-partially-pinverse-rep f N p n* = *0*
⟨*proof*⟩

lemmas *Abs-fls-partially-pinverse-rep-nth* =
nth-Abs-fls-lower-bound[*OF fls-partially-pinverse-rep-func-lower-bound'*]

lemma *fls-partially-pinverse-rep-func-truncates*:
n > *int N* ⟹ *fls-partially-pinverse-rep f N p n* = *0*
⟨*proof*⟩

context
fixes *p* :: *'a*::*nontrivial-factorial-unique-euclidean-bezout prime*
begin

lemma *pinverse-scalar-fls-pmod-base*:
(
(((*f pmod p*) $$ ($f^{\circ p}$))$^{-1p}$) ∗ (*f pmod p*) $$ ($f^{\circ p}$)
) *pmod p* = *1*
if $f \neg\simeq_p 0$
for *f* :: *'a fls*
⟨*proof*⟩

lemma *fls-partially-pinverse-rep-eq-zero*:
fls-partially-pinverse-rep f N p = *0* **if** $f \simeq_p 0$
for *f* :: *'a fls*
⟨*proof*⟩

lemma *fls-partially-pinverse-rep-nzero-at-0*:

fls-partially-pinverse-rep f N p $0 \neq 0$ **if** $f \neg\simeq_p 0$ **for** f :: *'a fls*
⟨*proof*⟩

lemma *fls-partially-pinverse-rep-eq-zero-iff*:
fls-partially-pinverse-rep f N $p = 0 \longleftrightarrow f \simeq_p 0$
for f :: *'a fls*
⟨*proof*⟩

lemma *Abs-fls-partially-pinverse-rep-nzero*:
Abs-fls (*fls-partially-pinverse-rep* f N p) $\neq 0$ **if** $f \neg\simeq_p 0$
for f :: *'a fls*
⟨*proof*⟩

lemma *Abs-fls-partially-pinverse-rep-eq-zero-iff*:
Abs-fls (*fls-partially-pinverse-rep* f N p) $= 0 \longleftrightarrow f \simeq_p 0$
for f :: *'a fls*
⟨*proof*⟩

lemma *Abs-fls-partially-pinverse-rep-subdegree*:
fls-subdegree (*Abs-fls* (*fls-partially-pinverse-rep* f N p)) $= 0$ **if** $f \neg\simeq_p 0$
for f :: *'a fls*
⟨*proof*⟩

lemma *fls-shift-pinv-rep-subdegree*:
fls-subdegree (*fls-shift* m (*Abs-fls* (*fls-partially-pinverse-rep* f N p))) $= -m$
if $f \neg\simeq_p 0$
for f :: *'a fls*
⟨*proof*⟩

lemma *fls-partially-pinverse-rep-cong*:
fls-partially-pinverse-rep f N p $(n - K)$ = *fls-partially-pinverse-rep* g N p $(n - K)$
if $f^{\circ p} = K$ **and** $g^{\circ p} = K$ **and** $\forall k \leq n.$ f \$\$ $k = g$ \$\$ k
for f g :: *'a fls*
⟨*proof*⟩

lemma *fls-pinv-rep-times-f-subdegree*:
fls-subdegree (
fls-shift $(f^{\circ p})$ (*Abs-fls* (*fls-partially-pinverse-rep* f N p)) $*$
(f *pmod* p)
) $= 0$
if $f \neg\simeq_p 0$
for f :: *'a fls*
⟨*proof*⟩

lemma *Abs-fls-pinv-rep-Suc*:
fixes f :: *'a fls* **and** N :: *nat*
defines
$c \equiv$

```
    −(
      (
        (
          fls-shift (f°p) (Abs-fls (fls-partially-pinverse-rep f N p)) ∗
          (f pmod p)
        ) pmod p
      ) $$ (int (Suc N)) ∗ (((f pmod p) $$ (f°p))⁻¹ᵖ)
    ) pmod p
  assumes f: f ¬≃p 0
  shows
    Abs-fls (fls-partially-pinverse-rep f (Suc N) p) =
      Abs-fls (fls-partially-pinverse-rep f N p) + fls-shift (− int (Suc N)) (fls-const
c)
⟨proof⟩

lemma fls-pinv-rep-times-f-nth:
  ((
    fls-shift (f°p) (Abs-fls (fls-partially-pinverse-rep f N p)) ∗
    (f pmod p)
  ) pmod p) $$ n = 0
  if  f: f ¬≃p 0 and n: n ≥ 1 n ≤ int N
  for f :: 'a fls and n :: int
⟨proof⟩

end
```

4.6.3 Full inversion of formal Laurent series

Now we collect the terms of the infinite sequence of partial inversion polynomials into a formal Laurent series, shifted to the opposite depth as the original.

```
overloading pinverse-fls ≡ pinverse :: 'a fls ⇒ 'a prime ⇒ 'a fls
begin
definition pinverse-fls ::
  'a::nontrivial-factorial-unique-euclidean-bezout fls ⇒ 'a prime ⇒ 'a fls
  where
    pinverse-fls f p ≡
      (if f ≃p 0 then 0 else
        fls-shift (f°p)
          (Abs-fls (λn. fls-partially-pinverse-rep f (nat n) p n)))
end

context
  fixes p :: 'a::nontrivial-factorial-unique-euclidean-bezout prime
begin

lemma pinverse-fls-nth:
  (f⁻¹ᵖ) $$ n =
    fls-partially-pinverse-rep f (nat (n + (f°p))) p (n + (f°p))
```

if $f \neg\simeq_p 0$
⟨*proof*⟩

lemma *pinverse-fls-nth'*:
int $N \geq n + (f^{\circ p}) \Longrightarrow$
(f^{-1p}) $$ n = *fls-partially-pinverse-rep* $f\ N\ p\ (n + (f^{\circ p}))$
if $f \neg\simeq_p 0$
⟨*proof*⟩

lemma *pinverse-fls-eq0*[*simp*]:
$(f^{-1p}) = 0$ **if** $f \simeq_p 0$ **for** $f :: 'a\ fls$
⟨*proof*⟩

lemma *fls-pinv-eq0-iff*:
$(f^{-1p}) = 0 \longleftrightarrow f \simeq_p 0$ **for** $f :: 'a\ fls$
⟨*proof*⟩

lemma *fls-pinv-subdegree*:
fls-subdegree $(f^{-1p}) = -(f^{\circ p})$ **for** $f :: 'a\ fls$
⟨*proof*⟩

lemma *fls-pinv-reduced*[*simp*]: $((f^{-1p})$ $$ $n)$ *pdiv* $p = 0$
⟨*proof*⟩

lemma *pinverse-times-p-modulo-fls*:
$(\ (f^{-1p}) * (f\ pmod\ p)\ )\ pmod\ p = 1$ **if** $f \neg\simeq_p 0$
for $f :: 'a\ fls$
⟨*proof*⟩

lemma *pinverse-fls*:
$((f^{-1p}) * f)\ pmod\ p = 1$ **if** $f \neg\simeq_p 0$ **for** $f :: 'a\ fls$
⟨*proof*⟩

lemma *pinverse-fls-mult-equiv1*:
$((f^{-1p}) * f) \simeq_p 1$ **if** $f \neg\simeq_p 0$ **for** $f :: 'a\ fls$
⟨*proof*⟩

lemma *pinverse-fls-mult-equiv1'*:
$(f * (f^{-1p})) \simeq_p 1$ **if** $f \neg\simeq_p 0$ **for** $f :: 'a\ fls$
⟨*proof*⟩

lemma *p-modulo-pinverse-fls*[*simp*]:
$(f^{-1p})\ pmod\ p = (f^{-1p})$ **for** $f :: 'a\ fls$
⟨*proof*⟩

lemma *fls-pinv-equiv0-iff*:
$(f^{-1p}) \simeq_p 0 \longleftrightarrow f \simeq_p 0$
for $f :: 'a\ fls$
⟨*proof*⟩

lemma *fls-pinv-depth*:
$(f^{-1p})^{\circ p} = -(f^{\circ p})$ **for** f :: *'a fls*
⟨*proof*⟩

lemma *fls-pinv-eqI*:
$(f^{-1p}) = g$ **if** $g * f \simeq_p 1$ **and** $\forall\, n.\ (g\ \$\$\ n)\ pdiv\ p = 0$
for $f\ g$ :: *'a fls*
⟨*proof*⟩

lemma *fls-pinv-cong*:
$(f^{-1p}) = (g^{-1p})$ **if** $f \simeq_p g$ **for** f :: *'a fls*
⟨*proof*⟩

lemma *pinverse-p-modulo-fls*:
$(f\ pmod\ p)^{-1p} = (f^{-1p})$ **for** f :: *'a fls*
⟨*proof*⟩

lemma *fls-pinv-X-intpow*: $(\mathit{fls\text{-}X\text{-}intpow}\ n$ :: *'a fls*$)^{-1p} = \mathit{fls\text{-}X\text{-}intpow}\ (-n)$
⟨*proof*⟩

lemma *fls-pinv-uminus*: $(-f)^{-1p} = (-\ (f^{-1p}))\ pmod\ p$ **for** f :: *'a fls*
⟨*proof*⟩

end

4.6.4 Algebraic properties of inversion

context
fixes p :: *'a::nontrivial-factorial-unique-euclidean-bezout prime*
begin

lemma *fls-pinv0*: $(0$ :: *'a fls*$)^{-1p} = 0$
⟨*proof*⟩

lemma *fls-pinv1* [*simp*]: $(1$ :: *'a fls*$)^{-1p} = 1$
⟨*proof*⟩

lemma *fls-pinv-pinv*:
$((f^{-1p})^{-1p}) = f\ pmod\ p$ **for** f :: *'a fls*
⟨*proof*⟩

lemma *fls-pinv-pinv-equiv*:
$((f^{-1p})^{-1p}) \simeq_p f$ **for** f :: *'a fls*
⟨*proof*⟩

lemma *fls-pinv-mult*:
$(f * g)^{-1p} = ((f^{-1p}) * (g^{-1p}))\ pmod\ p$
for $f\ g$ :: *'a fls*

⟨*proof*⟩

lemma *fls-pinv-mult-equiv*:
$((f * g)^{-1p}) \simeq_p (f^{-1p}) * (g^{-1p})$
for $f\ g$ :: $'a$ *fls*
⟨*proof*⟩

lemma *fls-pinv-pow*:
$(f \mathbin{\char`\^} n)^{-1p} = ((f^{-1p}) \mathbin{\char`\^} n)$ *pmod* p
for $f\ g$ :: $'a$ *fls*
⟨*proof*⟩

lemma *fls-pinv-pow-equiv*:
$((f \mathbin{\char`\^} n)^{-1p}) \simeq_p (f^{-1p}) \mathbin{\char`\^} n$ **for** f :: $'a$ *fls*
⟨*proof*⟩

lemma *fls-pinv-diff-cancel-lead-coeff*:
$(((f^{-1p}) - (g^{-1p}))^{\circ p}) > -n$
if f : $f \neg\simeq_p 0\ f^{\circ p} = n$
and g : $g \neg\simeq_p 0\ g^{\circ p} = n$
and *fg*: $f \neg\simeq_p g\ ((f - g)^{\circ p}) > n$
for $f\ g$ :: $'a$ *fls*
⟨*proof*⟩

end

4.7 Topology by place relative to indexed depth for formal Laurent series

4.7.1 General pattern for constructing sequence limits

definition *fls-condition-lim* ::
$'a$::*nontrivial-euclidean-ring-cancel prime* ⇒ (*nat* ⇒ $'a$ *fls*) ⇒
(*nat* ⇒ *nat* ⇒ *bool*) ⇒ $'a$ *fls*
where
fls-condition-lim $p\ F\ P$ =
Abs-fls (λn. (F (*LEAST* K. P (*nat* n) K) *pmod* p) \$\$ n)

lemma *fls-condition-lim-fun-ex-lower-bound*:
fixes p :: $'a$::*nontrivial-euclidean-ring-cancel prime*
and F :: *nat* ⇒ $'a$ *fls*
and P :: *int* ⇒ *nat* ⇒ *bool*
defines
f ≡ (λn. (F (*LEAST* K. P (*nat* n) K) *pmod* p) \$\$ n)
and
N ≡ *min* 0 (*fls-subdegree* ((F (*LEAST* K. P 0 K)) *pmod* p))
shows $\forall\, n{<}N.\ f\ n = 0$
⟨*proof*⟩

lemma *fls-condition-lim-nth*:

fls-condition-lim p F P $$ n = ((*F* (*LEAST K*. *P* (*nat n*) *K*)) *pmod p*) *$$ n*
⟨*proof*⟩

lemma *fls-pmod-condition-lim*: (*fls-condition-lim p F P*) *pmod p* = *fls-condition-lim p F P*
⟨*proof*⟩

4.7.2 Completeness

abbreviation *fls-p-limseq-condition* ≡ *p-depth-fls.p-limseq-condition*
abbreviation *fls-p-limseq* ≡ *p-depth-fls.p-limseq*
abbreviation *fls-p-cauchy-condition* ≡ *p-depth-fls.p-cauchy-condition*
abbreviation *fls-p-cauchy* ≡ *p-depth-fls.p-cauchy*

lemma *fls-p-cauchy-condition-pmod-uniformity*:
((*F k*) *pmod p*) *$$ m* = ((*F k'*) *pmod p*) *$$ m*
if *fls-p-cauchy-condition p F n K* **and** $k \geq K$ **and** $k' \geq K$ **and** $m \leq n$
for *p* :: *'a::nontrivial-euclidean-ring-cancel prime* **and** *F* :: *nat* ⇒ *'a fls*
⟨*proof*⟩

abbreviation
fls-p-cauchy-lim p X ≡
fls-condition-lim p X (λ*n*. *fls-p-cauchy-condition p X* (*int n*))

lemma *fls-p-cauchy-limseq*: *fls-p-limseq p F* (*fls-p-cauchy-lim p F*) **if** *fls-p-cauchy p F*
⟨*proof*⟩

lemma *fls-p-cauchy-lim-unique*:
fls-p-cauchy-lim p F = *fls-p-cauchy-lim p G*
if *fls-p-cauchy p G* **and** ∀ *n*≥*N*. (*F n*) $\simeq_p$ (*G n*)
⟨*proof*⟩

4.8 Transfer to prime-indexed sequences of formal Laurent series

4.8.1 Equivalence and depth

The equivalence is by divisibility of the difference of each pair of corresponding terms by the index for those terms.

type-synonym *'a fls-pseq* = *'a prime* ⇒ *'a fls*

overloading
p-equal-fls-pseq ≡
p-equal :: *'a::nontrivial-factorial-semiring prime* ⇒
'a fls-pseq ⇒ *'a fls-pseq* ⇒ *bool*
p-restrict-fls-pseq ≡
p-restrict ::
'a fls-pseq ⇒ (*'a prime* ⇒ *bool*) ⇒ *'a fls-pseq*

p-depth-fls-pseq ≡
p-depth :: *'a prime* ⇒ *'a fls-pseq* ⇒ *int*
global-unfrmzr-pows-fls-pseq ≡
global-unfrmzr-pows :: (*'a prime* ⇒ *int*) ⇒ *'a fls-pseq*
begin

definition *p-equal-fls-pseq* ::
'a::nontrivial-factorial-semiring prime ⇒
'a fls-pseq ⇒ *'a fls-pseq* ⇒ *bool*
where *p-equal-fls-pseq-def*[*simp*]: *p-equal-fls-pseq p X Y* ≡ $(X\ p) \simeq_p (Y\ p)$

definition *p-restrict-fls-pseq* ::
'a::nontrivial-factorial-semiring fls-pseq ⇒
(*'a prime* ⇒ *bool*) ⇒ *'a fls-pseq*
where
p-restrict-fls-pseq X P ≡ (λ*p*::*'a prime. if P p then X p else 0*)

definition *p-depth-fls-pseq* ::
'a::nontrivial-factorial-semiring prime ⇒ *'a fls-pseq* ⇒ *int*
where *p-depth-fls-pseq-def*[*simp*]: *p-depth-fls-pseq p X* ≡ $(X\ p)^{\circ p}$

definition *global-unfrmzr-pows-fls-pseq* ::
(*'a::nontrivial-factorial-semiring prime* ⇒ *int*) ⇒ *'a fls-pseq*
where *global-unfrmzr-pows-fls-pseq f* ≡ (λ*p*::*'a prime. fls-X-intpow* (*f p*))

end

global-interpretation *p-equality-depth-fls-pseq*:
global-p-equality-depth-no-zero-divisors-1
p-equal :: *'a::nontrivial-factorial-idom prime* ⇒
'a fls-pseq ⇒ *'a fls-pseq* ⇒ *bool*
p-restrict ::
'a fls-pseq ⇒ (*'a prime* ⇒ *bool*) ⇒ *'a fls-pseq*
p-depth :: *'a prime* ⇒ *'a fls-pseq* ⇒ *int*
⟨*proof*⟩

overloading
globally-p-equal-fls-pseq ≡
globally-p-equal :: *'a::nontrivial-factorial-idom fls-pseq* ⇒
'a fls-pseq ⇒ *bool*
p-depth-set-fls-pseq ≡
p-depth-set ::
'a::nontrivial-factorial-idom prime ⇒ *int* ⇒ *'a fls-pseq set*
global-depth-set-fls-pseq ≡
global-depth-set :: *int* ⇒ *'a fls-pseq set*
begin

definition *globally-p-equal-fls-pseq* ::
'a::nontrivial-factorial-idom fls-pseq ⇒ *'a fls-pseq* ⇒ *bool*

where *globally-p-equal-fls-pseq-def* [*simp*]:
globally-p-equal-fls-pseq = *p-equality-depth-fls-pseq.globally-p-equal*

definition *p-depth-set-fls-pseq* ::
'a::nontrivial-factorial-idom prime ⇒ *int* ⇒ *'a fls-pseq set*
where *p-depth-set-fls-pseq-def* [*simp*]:
p-depth-set-fls-pseq = *p-equality-depth-fls-pseq.p-depth-set*

definition *global-depth-set-fls-pseq* ::
int ⇒ *'a::nontrivial-factorial-idom fls-pseq set*
where *global-depth-set-fls-pseq-def* [*simp*]:
global-depth-set-fls-pseq = *p-equality-depth-fls-pseq.global-depth-set*

end

lemma *fls-pseq-globally-reduced*:
$X \simeq_{\forall\, p} (\lambda p.\ (X\ p)\ pmod\ p)$
for *X* :: *'a::nontrivial-euclidean-ring-cancel fls-pseq*
⟨*proof*⟩

lemma *global-unfrmzr-pows0-fls-pseq*:
$(\mathfrak{p}\ (0 :: 'a{::}nontrivial\text{-}factorial\text{-}idom\ prime \Rightarrow int) :: 'a\ fls\text{-}pseq) = 1$
⟨*proof*⟩

context
fixes *p* :: *'a::nontrivial-factorial-idom prime*
begin

lemma *global-unfrmzr-pows-fls-pseq-nequiv0*:
$(\mathfrak{p}\ f :: 'a\ fls\text{-}pseq) \neg\simeq_p 0$ **for** $f :: 'a\ prime \Rightarrow int$
⟨*proof*⟩

lemma *global-unfrmzr-pows-fls-pseq*:
$(\mathfrak{p}\ f :: 'a\ fls\text{-}pseq)^{\circ p} = f\ p$ **for** $f :: 'a\ prime \Rightarrow int$
⟨*proof*⟩

lemma *global-unfrmzr-pows-fls-pseq-pequiv-iff*:
$(\mathfrak{p}\ f{::}\ 'a\ fls\text{-}pseq) \simeq_p (\mathfrak{p}\ g) \longleftrightarrow f\ p = g\ p$
for $f\ g :: 'a\ prime \Rightarrow int$
⟨*proof*⟩

end

lemma *global-unfrmzr-pows-prod-fls-pseq*:
$(\mathfrak{p}\ f :: 'a\ fls\text{-}pseq) * (\mathfrak{p}\ g) = \mathfrak{p}\ (f + g)$
for $f\ g :: 'a{::}nontrivial\text{-}factorial\text{-}idom\ prime \Rightarrow int$
⟨*proof*⟩

context

fixes p :: $'a$::*nontrivial-factorial-idom prime*
begin

lemma *global-unfrmzr-pows-fls-pseq-nzero*:
$(\mathfrak{p}\ f :: 'a\ \mathit{fls\text{-}pseq}) \neg\simeq_p 0$ **for** $f :: 'a\ \mathit{prime} \Rightarrow \mathit{int}$
$\langle proof \rangle$

lemma *prod-w-global-unfrmzr-pows-fls-pseq*:
$(X * \mathfrak{p}\ f)^{\circ p} = (X^{\circ p}) + f\ p$ **if** $X \neg\simeq_p 0$
for $X :: 'a\ \mathit{fls\text{-}pseq}$ **and** $f :: 'a\ \mathit{prime} \Rightarrow \mathit{int}$
$\langle proof \rangle$

lemma *normalize-depth-fls-pseq*:
$(X * \mathfrak{p}\ (\lambda p{::}'a\ \mathit{prime}.\ -(X^{\circ p})))^{\circ p} = 0$
for $X :: 'a\ \mathit{fls\text{-}pseq}$
$\langle proof \rangle$

end

lemma *normalized-depth-fls-pseq-product-additive*:
$(X * \mathfrak{p}\ (\lambda p{::}'a\ \mathit{prime}.\ -(X^{\circ p}))) *$
$(Y * \mathfrak{p}\ (\lambda p{::}'a\ \mathit{prime}.\ -(Y^{\circ p}))) =$
$((X * Y) * \mathfrak{p}\ (\lambda p{::}'a\ \mathit{prime}.\ -((X^{\circ p}) + (Y^{\circ p}))))$
for $X\ Y :: 'a$::*nontrivial-factorial-idom fls-pseq*
$\langle proof \rangle$

context
fixes p :: $'a$::*nontrivial-factorial-idom prime*
begin

lemma *normalized-depth-fls-pseq-product-equiv*:
$(X * \mathfrak{p}\ (\lambda p{::}'a\ \mathit{prime}.\ -(X^{\circ p}))) *$
$(Y * \mathfrak{p}\ (\lambda p{::}'a\ \mathit{prime}.\ -(Y^{\circ p}))) \simeq_p$
$((X * Y) * \mathfrak{p}\ (\lambda p{::}'a\ \mathit{prime}.\ -((X * Y)^{\circ p})))$
for $X\ Y :: 'a\ \mathit{fls\text{-}pseq}$
$\langle proof \rangle$

lemma *trivial-global-unfrmzr-pows-fls-pseq*:
$(\mathfrak{p}\ f :: 'a\ \mathit{fls\text{-}pseq}) \simeq_p 1$ **if** $f\ p = 0$ **for** $f :: 'a\ \mathit{prime} \Rightarrow \mathit{int}$
$\langle proof \rangle$

lemma *prod-w-trivial-global-unfrmzr-pows-fls-pseq*:
$X * \mathfrak{p}\ f \simeq_p X$ **if** $f\ p = 0$
for $f :: 'a\ \mathit{prime} \Rightarrow \mathit{int}$ **and** $X :: 'a\ \mathit{fls\text{-}pseq}$
$\langle proof \rangle$

end

lemma *pow-global-unfrmzr-pows-fls-pseq*:

($\mathfrak{p}$ f :: $'a$ *fls-pseq*) ^ n = ($\mathfrak{p}$ (λp::$'a$ *prime*. *int* $n * f\ p$))
for f :: $'a$::*nontrivial-factorial-idom prime* $\Rightarrow$ *int*
⟨*proof*⟩

lemma *global-unfrmzr-pows-fls-pseq-inv*:
($\mathfrak{p}$ ($-f$) :: $'a$ *fls-pseq*) $*$ ($\mathfrak{p}$ f) = *1*
for f :: $'a$::*nontrivial-factorial-idom prime* $\Rightarrow$ *int*
⟨*proof*⟩

lemma *global-unfrmzr-pows-fls-pseq-decomp*:
X = (
($X * \mathfrak{p}$ (λp::$'a$ *prime*. $-(X^{\circ p})$)) $*$
$\mathfrak{p}$ (λp::$'a$ *prime*. $X^{\circ p}$)
)
for X :: $'a$::*nontrivial-factorial-idom fls-pseq*
⟨*proof*⟩

context
fixes p :: $'a$::*nontrivial-factorial-idom prime*
begin

lemma *global-unfrmzr-pth-fls-pseq*:
($\mathfrak{p}$ (*1* :: ($'a$ *prime* $\Rightarrow$ *int*)) p :: $'a$ *fls*) $\simeq_p$
(*fls-const* (*Rep-prime* p))
⟨*proof*⟩

lemma *global-unfrmzr-fls-pseq*:
($\mathfrak{p}$ (*1* :: ($'a$ *prime* $\Rightarrow$ *int*)) :: $'a$ *fls-pseq*) $\simeq_p$
(λp::$'a$ *prime*. *fls-const* (*Rep-prime* p))
⟨*proof*⟩

end

lemma *normalize-depth-fls-pseqE*:
fixes X :: $'a$::*nontrivial-factorial-idom fls-pseq*
obtains *X-0*
where $\forall p$::$'a$ *prime*. $X\text{-}0^{\circ p} = 0$
and $X = X\text{-}0 * \mathfrak{p}$ (λp::$'a$ *prime*. $X^{\circ p}$)
⟨*proof*⟩

4.8.2 Inversion

abbreviation *global-pinverse-fls-pseq* ::
$'a$::*nontrivial-factorial-comm-ring fls-pseq* $\Rightarrow$ $'a$ *fls-pseq*
(($-^{-1\forall p}$) [*51*] *50*)
where *global-pinverse-fls-pseq* $X \equiv (\lambda p.\ (X\ p)^{-1p})$

context
fixes X :: $'a$::*nontrivial-factorial-unique-euclidean-bezout fls-pseq*

begin

lemma *global-pinverse-fls-pseq-eq0*[*simp*]:
$(X^{-1\forall\, p}) = 0$ **if** $X \simeq_{\forall\, p} 0$
⟨*proof*⟩

lemma *global-pinverse-fls-pseq-eq0-iff*:
$(X^{-1\forall\, p}) = 0 \longleftrightarrow$
$X \simeq_{\forall\, p} 0$
⟨*proof*⟩

lemma *global-pinverse-fls-pseq-equiv0-iff*:
$(X^{-1\forall\, p}) \simeq_{\forall\, p} 0 \longleftrightarrow$
$X \simeq_{\forall\, p} 0$
⟨*proof*⟩

lemma *global-left-pinverse-fls-pseq*:
$(X^{-1\forall\, p}) * X \simeq_{\forall\, p}$
$(\lambda p.\ \mathit{of\text{-}bool}\ (X \neg\simeq_p 0))$
⟨*proof*⟩

lemma *global-right-pinverse-fls-pseq*:
$X * (X^{-1\forall\, p}) \simeq_{\forall\, p}$
$(\lambda p.\ \mathit{of\text{-}bool}\ (X \neg\simeq_p 0))$
⟨*proof*⟩

lemma *global-left-pinverse-fls-pseq'*:
$(X^{-1\forall\, p}) * X \simeq_p 1$ **if** $X \neg\simeq_p 0$
for p :: *'a prime*
⟨*proof*⟩

lemma *global-right-pinverse-fls-pseq'*:
$X * (X^{-1\forall\, p}) \simeq_p 1$ **if** $X \neg\simeq_p 0$
for p :: *'a prime*
⟨*proof*⟩

lemma *global-pinverse-pinverse-fls-pseq*:
$((X^{-1\forall\, p})^{-1\forall\, p})$
$\simeq_{\forall\, p} X$
⟨*proof*⟩

lemma *global-pinverse-mult-fls-pseq*:
$((X * Y)^{-1\forall\, p}) \simeq_{\forall\, p}$
$(X^{-1\forall\, p}) * (Y^{-1\forall\, p})$
⟨*proof*⟩

lemma *global-pinverse-pow-fls-pseq*:
$((X \frown n)^{-1\forall\, p}) \simeq_{\forall\, p}$
$(X^{-1\forall\, p}) \frown n$

$\langle proof \rangle$

lemma *global-pinverse-uminus-fls-pseq*:
$((-X)^{-1\forall\, p}) \simeq_{\forall\, p}$
$- (X^{-1\forall\, p})$
$\langle proof \rangle$

end

lemma *globally-pinverse-pcong*:
$(X^{-1\forall\, p}) \simeq_p (Y^{-1\forall\, p})$
if $X \simeq_p Y$
for p :: $'a$::*nontrivial-factorial-unique-euclidean-bezout prime* **and** X Y :: $'a$ *fls-pseq*
$\langle proof \rangle$

lemma *globally-pinverse-cong*:
$(X^{-1\forall\, p}) \simeq_{\forall\, p}$
$(Y^{-1\forall\, p})$
if $X \simeq_{\forall\, p} Y$
for X Y :: $'a$::*nontrivial-factorial-unique-euclidean-bezout fls-pseq*
$\langle proof \rangle$

lemma *globally-pinverse-global-unfrmzr-pows*:
$(\mathfrak{p}\ f$ :: $'a$ *fls-pseq*$)^{-1\forall\, p} = \mathfrak{p}\ (-\ f)$
for f :: $'a$::*nontrivial-factorial-unique-euclidean-bezout prime* $\Rightarrow$ *int*
$\langle proof \rangle$

lemma *global-pinverse-fls-pseq0*:
(
$(0$::$'a$::*nontrivial-factorial-unique-euclidean-bezout fls-pseq*$)^{-1\forall\, p}$
$) = 0$
$\langle proof \rangle$

lemma *global-pinverse-fls-pseq1*:
(
$(1$::$'a$::*nontrivial-factorial-unique-euclidean-bezout fls-pseq*$)^{-1\forall\, p}$
$) = 1$
$\langle proof \rangle$

lemma *global-pinverse-diff-cancel-lead-coeff*:
((
$(X^{-1\forall\, p}) -$
$(Y^{-1\forall\, p})$
$)^{\circ p}) > -n$
if $X \neg\simeq_p 0$ **and** $X^{\circ p} = n$
and $Y \neg\simeq_p 0$ **and** $Y^{\circ p} = n$
and $X \neg\simeq_p Y$ **and** $((X - Y)^{\circ p}) > n$
for p :: $'a$::*nontrivial-factorial-unique-euclidean-bezout prime*

and $X\ Y$ :: *'a fls-pseq*
⟨*proof*⟩

4.8.3 Topology via global bounded depth

abbreviation *fls-pseq-bdd-global-depth* ≡ *p-equality-depth-fls-pseq.bdd-global-depth*
abbreviation *fls-pseq-bdd-global-dist* ≡ *p-equality-depth-fls-pseq.bdd-global-dist*
abbreviation *fls-pseq-globally-cauchy* ≡ *p-equality-depth-fls-pseq.globally-cauchy*
abbreviation
fls-pseq-global-cauchy-condition ≡ *p-equality-depth-fls-pseq.global-cauchy-condition*

lemma *fls-pseq-global-cauchy-condition-pmod-uniformity*:
$((X\ k\ p)\ pmod\ p)$ \$\$ $m = ((X\ k'\ p)\ pmod\ p)$ \$\$ m
if *fls-pseq-global-cauchy-condition* $X\ n\ K$ **and** $k \geq K$ **and** $k' \geq K$ **and** $m \leq int$ n
for p :: *'a::nontrivial-euclidean-ring-cancel prime* **and** X :: $nat \Rightarrow$ *'a fls-pseq*
⟨*proof*⟩

abbreviation
fls-pseq-cauchy-lim X ≡
$\lambda p.$ *fls-condition-lim* p $(\lambda n.\ X\ n\ p)$ (*fls-pseq-global-cauchy-condition* X)

lemma *fls-pseq-cauchy-condition-lim*:
$\forall_F$ *k in sequentially. fls-pseq-bdd-global-dist* $(X\ k)$ (*fls-pseq-cauchy-lim* X) $< e$
if *pos*: $e > 0$ **and** *cauchy*: *fls-pseq-globally-cauchy* X
⟨*proof*⟩

end

theory *pAdic-Product*

imports
FLS-Prime-Equiv-Depth
HOL−Computational-Algebra.Fraction-Field
HOL−Analysis.Elementary-Metric-Spaces

begin

5 p-Adic fields as equivalence classes of sequences of formal Laurent series

5.1 Preliminaries

lemma *inj-on-vimage-image-eq*:
$(f\ -`\ (f\ `\ B)) \cap A = B$ **if** *inj-on f A* **and** $B \subseteq A$
⟨*proof*⟩

5.2 The type definition as a quotient

quotient-type (**overloaded**) $'a$ *p-adic-prod*
 $=$ $'a$::*nontrivial-factorial-idom fls-pseq* / *globally-p-equal*
 ⟨*proof*⟩

lemmas *rep-p-adic-prod-inverse* $=$ *Quotient3-abs-rep*[*OF Quotient3-p-adic-prod*]
 and *p-adic-prod-lift-globally-p-equal* $=$ *Quotient3-rel-abs*[*OF Quotient3-p-adic-prod*]
 and *globally-p-equal-fls-pseq-equals-rsp* $=$ *equals-rsp*[*OF Quotient3-p-adic-prod*]
 and *p-adic-prod-eq-iff*
 $=$ *Quotient3-rel-rep*[*OF Quotient3-p-adic-prod*, *symmetric*]
 and *globally-p-equal-fls-pseq-rep-p-adic-prod-refl*
 $=$ *Quotient3-rep-reflp*[*OF Quotient3-p-adic-prod*]
 and *globally-p-equal-fls-pseq-rep-abs-p-adic-prod*
 $=$ *rep-abs-rsp*[*OF Quotient3-p-adic-prod*] *rep-abs-rsp-left*[*OF Quotient3-p-adic-prod*]

lemma *p-adic-prod-globally-p-equal-self*:
 (*rep-p-adic-prod* (*abs-p-adic-prod* r)) $\simeq_{\forall\, p}$ r
 ⟨*proof*⟩

lemma *rep-p-adic-prod-p-equal-self*: *rep-p-adic-prod* (*abs-p-adic-prod* r) $\simeq_p$ r
 for p :: $'a$::*nontrivial-factorial-idom prime* **and** r :: $'a$ *fls-pseq*
 ⟨*proof*⟩

lemma *rep-p-adic-prod-set-inverse*: *abs-p-adic-prod* ‘ (*rep-p-adic-prod* ‘ A) $=$ A
⟨*proof*⟩

lemma *p-adic-prod-cases* [*case-names abs-p-adic-prod*, *cases type*: *p-adic-prod*]:
 C **if** $\bigwedge X.\ x =$ *abs-p-adic-prod* $X \Longrightarrow C$
 ⟨*proof*⟩

lemmas *two-p-adic-prod-cases* $=$ *p-adic-prod-cases*[*case-product p-adic-prod-cases*]
 and *three-p-adic-prod-cases* $=$
 p-adic-prod-cases[*case-product p-adic-prod-cases*[*case-product p-adic-prod-cases*]]

lemma *p-adic-prod-reduced-cases* [*case-names abs-p-adic-prod*]:
 fixes x :: $'a$::*nontrivial-euclidean-ring-cancel p-adic-prod*
 obtains X **where** $x =$ *abs-p-adic-prod* X **and** $\forall\, p.$ $(X\ p)$ *pmod* $p = X\ p$
⟨*proof*⟩

lemma *p-adic-prod-unique-rep*:
 $\exists!X.\ x =$ *abs-p-adic-prod* $X \wedge (\forall\, p.\ (X\ p)$ *pmod* $p = X\ p)$
 for x :: $'a$::*nontrivial-euclidean-ring-cancel p-adic-prod*
⟨*proof*⟩

abbreviation
 reduced-rep-p-adic-prod $x \equiv$
 (*THE* $X.\ x =$ *abs-p-adic-prod* $X \wedge (\forall\, p.\ (X\ p)$ *pmod* $p = X\ p)$)

lemma *reduced-rep-p-adic-prod-is-rep*:
$x = abs\text{-}p\text{-}adic\text{-}prod\ (reduced\text{-}rep\text{-}p\text{-}adic\text{-}prod\ x)$
for x :: $'a$::*nontrivial-euclidean-ring-cancel p-adic-prod*
⟨*proof*⟩

lemma *reduced-rep-p-adic-prod-is-reduced*:
$(reduced\text{-}rep\text{-}p\text{-}adic\text{-}prod\ x\ p)\ pmod\ p = reduced\text{-}rep\text{-}p\text{-}adic\text{-}prod\ x\ p$
for p :: $'a$::*nontrivial-euclidean-ring-cancel prime* **and** X :: $'a$ *p-adic-prod*
⟨*proof*⟩

lemma *abs-p-adic-prod-inverse*:
$reduced\text{-}rep\text{-}p\text{-}adic\text{-}prod\ (abs\text{-}p\text{-}adic\text{-}prod\ X) = X$ **if** $\forall p.\ (X\ p)\ pmod\ p = X\ p$
for X :: $'a$::*nontrivial-euclidean-ring-cancel fls-pseq*
⟨*proof*⟩

lemma *p-adic-prod-seq-cases* [*case-names abs-p-adic-prod*]:
C **if** $\bigwedge F.\ X = (\lambda n.\ abs\text{-}p\text{-}adic\text{-}prod\ (F\ n)) \Longrightarrow C$
⟨*proof*⟩

lemma *p-adic-prod-seq-reduced-cases* [*case-names abs-p-adic-prod*]:
fixes X :: $nat \Rightarrow$ $'a$::*nontrivial-euclidean-ring-cancel p-adic-prod*
obtains F
where $X = (\lambda n.\ abs\text{-}p\text{-}adic\text{-}prod\ (F\ n))$
and $\forall (n::nat)\ (p::'a\ prime).\ (F\ n\ p)\ pmod\ p = F\ n\ p$
⟨*proof*⟩

5.3 Algebraic instantiations

The product of p-adic fields form a ring.

instantiation *p-adic-prod* :: (*nontrivial-factorial-idom*) *comm-ring-1*
begin

lift-definition *zero-p-adic-prod* :: $'a$ *p-adic-prod* **is** 0::$'a$ *fls-pseq* ⟨*proof*⟩

lift-definition *one-p-adic-prod* :: $'a$ *p-adic-prod* **is** 1::$'a$ *fls-pseq* ⟨*proof*⟩

lift-definition *plus-p-adic-prod* ::
$'a$ *p-adic-prod* $\Rightarrow$ $'a$ *p-adic-prod* $\Rightarrow$ $'a$ *p-adic-prod*
is $\lambda X\ Y.\ X + Y$
⟨*proof*⟩

lift-definition *uminus-p-adic-prod* :: $'a$ *p-adic-prod* $\Rightarrow$ $'a$ *p-adic-prod*
is $\lambda X.\ - X$
⟨*proof*⟩

lift-definition *minus-p-adic-prod* ::
$'a$ *p-adic-prod* $\Rightarrow$ $'a$ *p-adic-prod* $\Rightarrow$ $'a$ *p-adic-prod*
is $\lambda X\ Y.\ X - Y$
⟨*proof*⟩

lift-definition *times-p-adic-prod* ::
  'a p-adic-prod ⇒ *'a p-adic-prod* ⇒ *'a p-adic-prod*
  is $\lambda X\ Y.\ X * Y$
  ⟨*proof*⟩

instance
⟨*proof*⟩

end

lemma *pow-p-adic-prod-abs-eq*:
  (*abs-p-adic-prod X*) ^ *n* = *abs-p-adic-prod* (*X* ^ *n*)
  for *X* :: *'a::nontrivial-factorial-idom fls-pseq*
⟨*proof*⟩

We can create inverses at the nonzero places.

instantiation *p-adic-prod* ::
  (*nontrivial-factorial-unique-euclidean-bezout*) {*divide-trivial*, *inverse*}
begin

lift-definition *divide-p-adic-prod* ::
  'a p-adic-prod ⇒ *'a p-adic-prod* ⇒ *'a p-adic-prod*
  is $\lambda X\ Y.\ X * (Y^{-1\forall p})$
⟨*proof*⟩

lift-definition *inverse-p-adic-prod* :: *'a p-adic-prod* ⇒ *'a p-adic-prod*
  is *global-pinverse-fls-pseq*
  ⟨*proof*⟩

instance
⟨*proof*⟩

end

5.4 Equivalence and depth relative to a prime

overloading
  p-equal-p-adic-prod ≡
    p-equal :: *'a::nontrivial-factorial-idom prime* ⇒ *'a p-adic-prod* ⇒
      'a p-adic-prod ⇒ *bool*
  p-restrict-p-adic-prod ≡
    p-restrict :: *'a p-adic-prod* ⇒
      (*'a prime* ⇒ *bool*) ⇒ *'a p-adic-prod*
  p-depth-p-adic-prod ≡ *p-depth* :: *'a prime* ⇒ *'a p-adic-prod* ⇒ *int*
  global-unfrmzr-pows-p-adic-prod ≡
    global-unfrmzr-pows :: (*'a prime* ⇒ *int*) ⇒ *'a p-adic-prod*
begin

lift-definition *p-equal-p-adic-prod* ::
  'a::nontrivial-factorial-idom prime $\Rightarrow$
    'a p-adic-prod $\Rightarrow$ *'a p-adic-prod* $\Rightarrow$ *bool*
  is *p-equal*
  ⟨*proof*⟩

lift-definition *p-restrict-p-adic-prod* ::
  'a::nontrivial-factorial-idom p-adic-prod $\Rightarrow$
    (*'a prime* $\Rightarrow$ *bool*) $\Rightarrow$ *'a p-adic-prod*
  is $\lambda X\ P\ p.$ *if* $P\ p$ *then* $X\ p$ *else* 0
  ⟨*proof*⟩

lift-definition *p-depth-p-adic-prod* ::
  'a::nontrivial-factorial-idom prime $\Rightarrow$ *'a p-adic-prod* $\Rightarrow$ *int*
  is *p-depth*
  ⟨*proof*⟩

lift-definition *global-unfrmzr-pows-p-adic-prod* ::
  (*'a::nontrivial-factorial-idom prime* $\Rightarrow$ *int*) $\Rightarrow$ *'a p-adic-prod*
  is *global-unfrmzr-pows* ⟨*proof*⟩

end

lemma *p-equal-p-adic-prod-abs-eq0*: (*abs-p-adic-prod* $X \simeq_p 0$) = ($X \simeq_p 0$)
  for p :: *'a::nontrivial-factorial-idom prime* **and** X :: *'a fls-pseq*
  ⟨*proof*⟩

lemma *p-equal-p-adic-prod-abs-eq1*: (*abs-p-adic-prod* $X \simeq_p 1$) = ($X \simeq_p 1$)
  for p :: *'a::nontrivial-factorial-idom prime* **and** X :: *'a fls-pseq*
  ⟨*proof*⟩

global-interpretation *p-equality-p-adic-prod*:
  p-equality-no-zero-divisors-1
    p-equal :: *'a::nontrivial-factorial-idom prime* $\Rightarrow$
      'a p-adic-prod $\Rightarrow$ *'a p-adic-prod* $\Rightarrow$ *bool*
⟨*proof*⟩

overloading
  globally-p-equal-p-adic-prod $\equiv$
    globally-p-equal ::
      'a::nontrivial-factorial-idom p-adic-prod $\Rightarrow$ *'a p-adic-prod* $\Rightarrow$ *bool*
begin

definition *globally-p-equal-p-adic-prod* ::
  'a::nontrivial-factorial-idom p-adic-prod $\Rightarrow$ *'a p-adic-prod* $\Rightarrow$ *bool*
  where *globally-p-equal-p-adic-prod-def*[*simp*]:
    globally-p-equal-p-adic-prod = *p-equality-p-adic-prod.globally-p-equal*

end

global-interpretation *global-p-depth-p-adic-prod*:
global-p-equal-depth-no-zero-divisors-w-inj-index-consts
p-equal :: *'a*::*nontrivial-factorial-idom prime* ⇒
'a p-adic-prod ⇒ *'a p-adic-prod* ⇒ *bool*
p-restrict ::
'a p-adic-prod ⇒ (*'a prime* ⇒ *bool*) ⇒ *'a p-adic-prod*
p-depth :: *'a prime* ⇒ *'a p-adic-prod* ⇒ *int*
λf. *abs-p-adic-prod* (λp. *fls-const* (*f p*))
Rep-prime
⟨*proof*⟩

lemma *p-depth-p-adic-prod-diff-abs-eq*:
((*abs-p-adic-prod X* − *abs-p-adic-prod Y*)$^{\circ p}$) = ((*X* − *Y*)$^{\circ p}$)
for *p* :: *'a*::*nontrivial-factorial-idom prime* **and** *X Y* :: *'a fls-pseq*
⟨*proof*⟩

lemma *p-depth-p-adic-prod-diff-abs-eq'*:
((*abs-p-adic-prod X* − *abs-p-adic-prod Y*)$^{\circ p}$) = ((*X p* − *Y p*)$^{\circ p}$)
for *p* :: *'a*::*nontrivial-factorial-idom prime* **and** *X Y* :: *'a fls-pseq*
⟨*proof*⟩

overloading
p-depth-set-p-adic-prod ≡
p-depth-set ::
'a::*nontrivial-factorial-idom prime* ⇒ *int* ⇒ *'a p-adic-prod set*
global-depth-set-p-adic-prod ≡
global-depth-set :: *int* ⇒ *'a p-adic-prod set*
begin

definition *p-depth-set-p-adic-prod* ::
'a::*nontrivial-factorial-idom prime* ⇒ *int* ⇒ *'a p-adic-prod set*
where *p-depth-set-p-adic-prod-def*[*simp*]:
p-depth-set-p-adic-prod = *global-p-depth-p-adic-prod.p-depth-set*

definition *global-depth-set-p-adic-prod* ::
int ⇒ *'a*::*nontrivial-factorial-idom p-adic-prod set*
where *global-depth-set-p-adic-prod-def*[*simp*]:
global-depth-set-p-adic-prod = *global-p-depth-p-adic-prod.global-depth-set*

end

lemma *p-depth-set-p-adic-prod-eq-projection*:
(($\mathcal{P}_{@p}{}^{n}$) :: *'a p-adic-prod set*) =
abs-p-adic-prod ‘ ($\mathcal{P}_{@p}{}^{n}$)
for *p* :: *'a*::*nontrivial-factorial-idom prime*
⟨*proof*⟩

lemma *global-depth-set-p-adic-prod-eq-projection*:

$((\mathcal{P}_{\forall\, p}{}^{n}) :: {'a}\ \textit{p-adic-prod set}) =$
$\quad \textit{abs-p-adic-prod}\ `\ (\mathcal{P}_{\forall\, p}{}^{n})$
for $p :: {'a}::\textit{nontrivial-factorial-idom prime}$
⟨*proof*⟩

context
fixes $x :: {'a}::\textit{nontrivial-factorial-idom p-adic-prod}$
begin

lemma *p-adic-prod-p-restrict-depth*:
$(x\ \textit{prestrict}\ P)^{\circ p} = (\textit{if}\ P\ p\ \textit{then}\ x^{\circ p}\ \textit{else}\ 0)$
for $P :: {'a}\ \textit{prime} \Rightarrow \textit{bool}$
⟨*proof*⟩

lemma *p-adic-prod-global-depth-setI*:
$x \in \mathcal{P}_{\forall\, p}{}^{n}$
if $\bigwedge p::{'a}\ \textit{prime}.\ x \neg\simeq_p 0 \Longrightarrow (x^{\circ p}) \geq n$
⟨*proof*⟩

lemma *p-adic-prod-global-depth-setI'*:
$x \in \mathcal{P}_{\forall\, p}{}^{n}$
if $\bigwedge p::{'a}\ \textit{prime}.\ (x^{\circ p}) \geq n$
⟨*proof*⟩

lemma *p-adic-prod-global-depth-set-p-restrictI*:
$\textit{p-restrict}\ x\ P \in \mathcal{P}_{\forall\, p}{}^{n}$
if $\bigwedge p::{'a}\ \textit{prime}.\ P\ p \Longrightarrow x \in \mathcal{P}_{@p}{}^{n}$
⟨*proof*⟩

lemma *p-adic-prod-p-depth-setI*:
$x \in \mathcal{P}_{@p}{}^{n}$
if $x \neg\simeq_p 0 \longrightarrow (x^{\circ p}) \geq n$
for $p :: {'a}\ \textit{prime}$
⟨*proof*⟩

end

context
fixes $p :: {'a}::\textit{nontrivial-euclidean-ring-cancel prime}$
begin

lemma *p-adic-prod-diff-depth-gt-equiv-trans*:
$((x - z)^{\circ p}) > n$
if *xy*: $x \simeq_p y$
and *yz*: $y \neg\simeq_p z\ ((y - z)^{\circ p}) > n$
for $x\ y\ z :: {'a}\ \textit{p-adic-prod}$
⟨*proof*⟩

lemma *p-adic-prod-diff-depth-gt0-equiv-trans*:

$((x - z)^{\circ p}) > n$
if $n \geq 0$ **and** $x \simeq_p y$ **and** $((y - z)^{\circ p}) > n$
for $x\ y\ z :: 'a\ p\text{-}adic\text{-}prod$
⟨*proof*⟩

lemma *p-adic-prod-diff-depth-gt-trans*:
$((x - z)^{\circ p}) > n$
if *xy*: $x \neg\simeq_p y\ ((x - y)^{\circ p}) > n$
and *yz*: $y \neg\simeq_p z\ ((y - z)^{\circ p}) > n$
and *xz*: $x \neg\simeq_p z$
for $x\ y\ z :: 'a\ p\text{-}adic\text{-}prod$
⟨*proof*⟩

lemma *p-adic-prod-diff-depth-gt0-trans*:
$((x - z)^{\circ p}) > n$
if $n \geq 0$
and $((x - y)^{\circ p}) > n$
and $((y - z)^{\circ p}) > n$
and $x \neg\simeq_p z$
for $x\ y\ z :: 'a\ p\text{-}adic\text{-}prod$
⟨*proof*⟩

end

lemma *p-adic-prod-diff-cancel-lead-coeff*:
$((inverse\ x - inverse\ y)^{\circ p}) > -n$
if $x : x \neg\simeq_p 0\ x^{\circ p} = n$
and $y : y \neg\simeq_p 0\ y^{\circ p} = n$
and *xy*: $x \neg\simeq_p y\ ((x - y)^{\circ p}) > n$
for $p :: 'a::nontrivial\text{-}factorial\text{-}unique\text{-}euclidean\text{-}bezout\ prime$
and $x\ y :: 'a\ p\text{-}adic\text{-}prod$
⟨*proof*⟩

lemma *global-unfrmzr-pows-p-adic-prod0*:
$(\mathfrak{p}\ (0 :: 'a::nontrivial\text{-}factorial\text{-}idom\ prime \Rightarrow int) :: 'a\ p\text{-}adic\text{-}prod) = 1$
⟨*proof*⟩

context
fixes $p :: 'a::nontrivial\text{-}factorial\text{-}idom\ prime$
begin

lemma *global-unfrmzr-pows-p-adic-prod-nequiv0*:
$(\mathfrak{p}\ f :: 'a\ p\text{-}adic\text{-}prod) \neg\simeq_p 0$ **for** $f :: 'a\ prime \Rightarrow int$
⟨*proof*⟩

lemma *global-unfrmzr-pows-p-adic-prod*:
$(\mathfrak{p}\ f :: 'a\ p\text{-}adic\text{-}prod)^{\circ p} = f\ p$ **for** $f :: 'a\ prime \Rightarrow int$
⟨*proof*⟩

lemma *global-unfrmzr-pows-p-adic-prod-depth-set*:
fixes $f :: 'a\ prime \Rightarrow int$
defines $n \equiv f\ p$
shows $(\mathfrak{p}\ f :: 'a\ p\text{-}adic\text{-}prod) \in \mathcal{P}_{@p}{}^{n}$
⟨*proof*⟩

lemma *global-unfrmzr-pows-p-adic-prod-pequiv-iff*:
$((\mathfrak{p}\ f :: 'a\ p\text{-}adic\text{-}prod) \simeq_p (\mathfrak{p}\ g)) = (f\ p = g\ p)$
for $f\ g :: 'a\ prime \Rightarrow int$
⟨*proof*⟩

end

lemma *global-unfrmzr-pows-p-adic-prod-global-depth-set*:
$(\mathfrak{p}\ f :: 'a\ p\text{-}adic\text{-}prod) \in \mathcal{P}_{\forall\ p}{}^{n}$
if $\forall\, p.\ f\ p \geq n$ **for** $f :: 'a::nontrivial\text{-}factorial\text{-}idom\ prime \Rightarrow int$
⟨*proof*⟩

lemma *global-unfrmzr-pows-p-adic-prod-global-depth-set-0*:
$(\mathfrak{p}\ (int \circ f) :: 'a\ p\text{-}adic\text{-}prod) \in \mathcal{O}_{\forall\ p}$
for $f :: 'a::nontrivial\text{-}factorial\text{-}idom\ prime \Rightarrow nat$
⟨*proof*⟩

lemma *global-unfrmzr-pows-prod-p-adic-prod*:
$(\mathfrak{p}\ f :: 'a\ p\text{-}adic\text{-}prod) * (\mathfrak{p}\ g) = \mathfrak{p}\ (f + g)$
for $f\ g :: 'a::nontrivial\text{-}factorial\text{-}idom\ prime \Rightarrow int$
⟨*proof*⟩

context
fixes $p :: 'a::nontrivial\text{-}factorial\text{-}idom\ prime$
begin

lemma *global-unfrmzr-pows-p-adic-prod-nzero*:
$(\mathfrak{p}\ f :: 'a\ p\text{-}adic\text{-}prod) \neg\simeq_p 0$ **for** $f :: 'a\ prime \Rightarrow int$
⟨*proof*⟩

lemma *prod-w-global-unfrmzr-pows-p-adic-prod*:
$x \neg\simeq_p 0 \Longrightarrow$
$(x * \mathfrak{p}\ f)^{\circ p} = (x^{\circ p}) + f\ p$
for $f :: 'a\ prime \Rightarrow int$ **and** $x :: 'a\ p\text{-}adic\text{-}prod$
⟨*proof*⟩

lemma *p-adic-prod-normalize-depth*:
$(x * \mathfrak{p}\ (\lambda p::'a\ prime.\ -(x^{\circ p})))^{\circ p} = 0$
for $x :: 'a\ p\text{-}adic\text{-}prod$
⟨*proof*⟩

end

lemma *p-adic-prod-normalized-depth-product-equiv*:
$(x * \mathfrak{p}\ (\lambda p{::}'a\ prime.\ -(x^{\circ p}))) *$
$(y * \mathfrak{p}\ (\lambda p{::}'a\ prime.\ -(y^{\circ p}))) =$
$((x * y) * \mathfrak{p}\ (\lambda p{::}'a\ prime.\ -((x * y)^{\circ p})))$
for $x\ y :: 'a{::}nontrivial\text{-}factorial\text{-}idom\ p\text{-}adic\text{-}prod$
$\langle proof \rangle$

context
fixes $p :: 'a{::}nontrivial\text{-}factorial\text{-}idom\ prime$
begin

lemma *trivial-global-unfrmzr-pows-p-adic-prod*:
$f\ p = 0 \Longrightarrow (\mathfrak{p}\ f :: 'a\ p\text{-}adic\text{-}prod) \simeq_p 1$
for $f :: 'a\ prime \Rightarrow int$
$\langle proof \rangle$

lemma *prod-w-trivial-global-unfrmzr-pows-p-adic-prod*:
$f\ p = 0 \Longrightarrow x * \mathfrak{p}\ f \simeq_p x$
for $f :: 'a\ prime \Rightarrow int$ **and** $x :: 'a\ p\text{-}adic\text{-}prod$
$\langle proof \rangle$

end

lemma *global-unfrmzr-pows-p-adic-prod-pow*:
$(\mathfrak{p}\ f :: 'a\ p\text{-}adic\text{-}prod)\ \widehat{}\ n = (\mathfrak{p}\ (\lambda p.\ int\ n * f\ p))$
for $f :: 'a{::}nontrivial\text{-}factorial\text{-}idom\ prime \Rightarrow int$
$\langle proof \rangle$
lemma *global-unfrmzr-pows-p-adic-prod-inv*:
$(\mathfrak{p}\ (-f) :: 'a\ p\text{-}adic\text{-}prod) * (\mathfrak{p}\ f) = 1$
for $f :: 'a{::}nontrivial\text{-}factorial\text{-}idom\ prime \Rightarrow int$
$\langle proof \rangle$

lemma *global-unfrmzr-pows-p-adic-prod-inverse*:
$inverse\ (\mathfrak{p}\ f :: 'a\ p\text{-}adic\text{-}prod) = \mathfrak{p}\ (-f)$
for $f :: 'a{::}nontrivial\text{-}factorial\text{-}unique\text{-}euclidean\text{-}bezout\ prime \Rightarrow int$
$\langle proof \rangle$

lemma *global-unfrmzr-pows-p-adic-prod-decomp*:
$x = ($
$(x * \mathfrak{p}\ (\lambda p{::}'a\ prime.\ -(x^{\circ p}))) *$
$\mathfrak{p}\ (\lambda p{::}'a\ prime.\ x^{\circ p})$
$)$
for $p :: 'a{::}nontrivial\text{-}factorial\text{-}idom\ prime$ **and** $x :: 'a\ p\text{-}adic\text{-}prod$
$\langle proof \rangle$

lemma *p-adic-prod-normalize-depthE*:
fixes $x :: 'a{::}nontrivial\text{-}factorial\text{-}idom\ p\text{-}adic\text{-}prod$
obtains $x\text{-}0$
where $\forall\, p{::}'a\ prime.\ x\text{-}0^{\circ p} = 0$

and $x = x\text{-}0 * \mathfrak{p}\ (\lambda p{::}'a\ prime.\ x^{\circ p})$
⟨*proof*⟩

global-interpretation *p-adic-prod-div-inv*:
global-p-equal-depth-div-inv
p-equal :: *'a::nontrivial-factorial-unique-euclidean-bezout prime* ⇒
'a p-adic-prod ⇒ *'a p-adic-prod* ⇒ *bool*
p-depth :: *'a prime* ⇒ *'a p-adic-prod* ⇒ *int*
p-restrict ::
'a p-adic-prod ⇒ (*'a prime* ⇒ *bool*) ⇒ *'a p-adic-prod*
⟨*proof*⟩

5.5 Embeddings of the base ring, *nat*, *int*, and *rat*

5.5.1 Embedding of the base ring

abbreviation *p-adic-prod-consts* ≡ *global-p-depth-p-adic-prod.consts*
abbreviation *p-adic-prod-const* ≡ *global-p-depth-p-adic-prod.const*

lemma *p-depth-p-adic-prod-consts*:
$(p\text{-}adic\text{-}prod\text{-}consts\ f)^{\circ p} = ((f\ p)^{\circ p})$
for *p* :: *'a::nontrivial-factorial-idom prime* **and** *f* :: *'a prime* ⇒ *'a*
⟨*proof*⟩

lemmas *p-depth-p-adic-prod-const* = *p-depth-p-adic-prod-consts*[*of* - λ*p*. -]

lemma *p-adic-prod-global-unfrmzr*:
($\mathfrak{p}$ (*1* :: *'a::nontrivial-factorial-idom prime* ⇒ *int*) :: *'a p-adic-prod*) =
p-adic-prod-consts Rep-prime
⟨*proof*⟩

5.5.2 Embedding of the field of fractions of the base ring

global-interpretation *p-adic-prod-embeds*:
global-p-equal-div-inv-w-inj-idom-consts
p-equal :: *'a::nontrivial-factorial-unique-euclidean-bezout prime* ⇒
'a p-adic-prod ⇒ *'a p-adic-prod* ⇒ *bool*
p-restrict ::
'a p-adic-prod ⇒ (*'a prime* ⇒ *bool*) ⇒ *'a p-adic-prod*
λ*f*. *abs-p-adic-prod* (λ*p*. *fls-const* (*f p*))
⟨*proof*⟩

global-interpretation *p-adic-prod-depth-embeds*:
global-p-equal-depth-div-inv-w-inj-index-consts
p-equal :: *'a::nontrivial-factorial-unique-euclidean-bezout prime* ⇒
'a p-adic-prod ⇒ *'a p-adic-prod* ⇒ *bool*
p-depth :: *'a prime* ⇒ *'a p-adic-prod* ⇒ *int*
p-restrict ::
'a p-adic-prod ⇒ (*'a prime* ⇒ *bool*) ⇒ *'a p-adic-prod*
λ*f*. *abs-p-adic-prod* (λ*p*. *fls-const* (*f p*))

Rep-prime
⟨*proof*⟩

abbreviation *p-adic-prod-shift-p-depth* ≡ *p-adic-prod-depth-embeds.shift-p-depth*
abbreviation *p-adic-prod-of-fract* ≡ *p-adic-prod-embeds.const-of-fract*

lemma *p-adic-prod-of-fract-depth*:
$(p\text{-}adic\text{-}prod\text{-}of\text{-}fract\ (Fraction\text{-}Field.Fract\ a\ b) :: 'a\ p\text{-}adic\text{-}prod)^{\circ p} =$
$(a^{\circ p}) - (b^{\circ p})$
if $a \neq 0$ **and** $b \neq 0$
for p :: *'a::nontrivial-factorial-unique-euclidean-bezout prime*
and $a\ b$ :: *'a*
⟨*proof*⟩

Product of all p-adic embeddings of the field of fractions of the base ring.

abbreviation *p-adic-Fracts-prod* ::
'a::nontrivial-factorial-unique-euclidean-bezout p-adic-prod set
$(\mathcal{Q}_{\forall\,p})$
where $\mathcal{Q}_{\forall\,p}$ ≡ *p-adic-prod-embeds.range-const-of-fract*

5.5.3 Characteristic zero embeddings

class *nontrivial-factorial-idom-char-0* = *nontrivial-factorial-idom* + *semiring-char-0*
begin
subclass *ring-char-0* ⟨*proof*⟩
end

instance *int* :: *nontrivial-factorial-idom-char-0* ⟨*proof*⟩

class *nontrivial-factorial-unique-euclidean-bezout-char-0* =
nontrivial-factorial-unique-euclidean-bezout + *nontrivial-factorial-idom-char-0*

instance *int* :: *nontrivial-factorial-unique-euclidean-bezout-char-0* ⟨*proof*⟩

instance *p-adic-prod* :: (*nontrivial-factorial-idom-char-0*) *ring-char-0*
⟨*proof*⟩

global-interpretation *p-adic-prod-consts-char-0*:
global-p-equality-w-inj-consts-char-0
p-equal :: *'a::nontrivial-factorial-idom-char-0 prime* ⇒
'a p-adic-prod ⇒ *'a p-adic-prod* ⇒ *bool*
p-restrict ::
'a p-adic-prod ⇒ (*'a prime* ⇒ *bool*) ⇒ *'a p-adic-prod*
$\lambda f.$ *abs-p-adic-prod* ($\lambda p.$ *fls-const* ($f\ p$))
⟨*proof*⟩

global-interpretation *p-adic-prod-div-inv-consts-char-0*:
global-p-equal-div-inv-w-inj-consts-char-0
p-equal :: *'a::nontrivial-factorial-unique-euclidean-bezout-char-0 prime* ⇒

$'a$ *p-adic-prod* $\Rightarrow$ $'a$ *p-adic-prod* $\Rightarrow$ *bool*
p-restrict ::
$'a$ *p-adic-prod* $\Rightarrow$ ($'a$ *prime* $\Rightarrow$ *bool*) $\Rightarrow$ $'a$ *p-adic-prod*
λf. *abs-p-adic-prod* (λp. *fls-const* (f p))
⟨*proof*⟩

lemma *p-adic-prod-of-nat-depth*:
(*of-nat* n :: $'a$ *p-adic-prod*)$^{\circ p}$ = ((*of-nat* n :: $'a$)$^{\circ p}$)
for p :: $'a$::*nontrivial-factorial-idom prime*
⟨*proof*⟩

lemma *p-adic-prod-of-int-depth*:
(*of-int* z :: $'a$ *p-adic-prod*)$^{\circ p}$ = ((*of-int* z :: $'a$)$^{\circ p}$)
for p :: $'a$::*nontrivial-factorial-idom prime*
⟨*proof*⟩

abbreviation *p-adic-prod-of-rat* $\equiv$ *p-adic-prod-div-inv-consts-char-0.const-of-rat*

lemma *p-adic-prod-of-rat-depth*:
(*p-adic-prod-of-rat* (*Rat.Fract* a b) :: $'a$ *p-adic-prod*)$^{\circ p}$ =
((*of-int* a :: $'a$)$^{\circ p}$) − ((*of-int* b :: $'a$)$^{\circ p}$)
if $a \neq 0$ **and** $b \neq 0$
for p :: $'a$::*nontrivial-factorial-unique-euclidean-bezout-char-0 prime*
and a b :: *int*
⟨*proof*⟩

5.6 Topologies on products of p-adic fields

5.6.1 By local place

Completeness

abbreviation *p-adic-prod-p-open-nbhd* $\equiv$ *global-p-depth-p-adic-prod.p-open-nbhd*
abbreviation *p-adic-prod-p-open-nbhds* $\equiv$ *global-p-depth-p-adic-prod.p-open-nbhds*
abbreviation
p-adic-prod-p-limseq-condition $\equiv$ *global-p-depth-p-adic-prod.p-limseq-condition*
abbreviation
p-adic-prod-p-cauchy-condition $\equiv$ *global-p-depth-p-adic-prod.p-cauchy-condition*
abbreviation *p-adic-prod-p-limseq* $\equiv$ *global-p-depth-p-adic-prod.p-limseq*
abbreviation *p-adic-prod-p-cauchy* $\equiv$ *global-p-depth-p-adic-prod.p-cauchy*
abbreviation
p-adic-prod-p-open-nbhds-limseq $\equiv$ *global-p-depth-p-adic-prod.p-open-nbhds-LIMSEQ*
abbreviation
p-adic-prod-local-p-open-nbhds $\equiv$ *global-p-depth-p-adic-prod.local-p-open-nbhds*

lemma *abs-p-adic-prod-seq-cases* [*case-names abs-p-adic-prod*]:
C **if** $\bigwedge F$. X = (λn. *abs-p-adic-prod* (F n)) $\Longrightarrow$ C
⟨*proof*⟩

lemma *abs-p-adic-prod-p-limseq-condition*:

p-adic-prod-p-limseq-condition p (*λn. abs-p-adic-prod* (*F n*)) (*abs-p-adic-prod X*)
n K =
fls-p-limseq-condition p (*λn. F n p*) (*X p*) *n K*
⟨*proof*⟩

lemma *abs-p-adic-prod-p-limseq*:
p-adic-prod-p-limseq p (*λn. abs-p-adic-prod* (*F n*)) (*abs-p-adic-prod X*) =
fls-p-limseq p (*λn. F n p*) (*X p*)
⟨*proof*⟩

lemma *abs-p-adic-prod-p-cauchy-condition*:
p-adic-prod-p-cauchy-condition p (*λn. abs-p-adic-prod* (*F n*)) *n K* =
fls-p-cauchy-condition p (*λn. F n p*) *n K*
⟨*proof*⟩

lemma *abs-p-adic-prod-p-cauchy*:
p-adic-prod-p-cauchy p (*λn. abs-p-adic-prod* (*F n*)) = *fls-p-cauchy p* (*λn. F n p*)
⟨*proof*⟩

lemma *p-adic-prod-p-restrict-cauchy-lim*:
p-adic-prod-p-limseq q
(*λn. p-restrict* (*X n*) ((=) *p*))
(*abs-p-adic-prod* (*λq.*
if q = p then fls-p-cauchy-lim p (*λn. rep-p-adic-prod* (*X n*) *p*) *else 0*
))
if *p-adic-prod-p-cauchy p X*
⟨*proof*⟩

global-interpretation *p-complete-p-adic-prod*:
p-complete-global-p-equal-depth
p-equal :: *'a::nontrivial-euclidean-ring-cancel prime* ⇒
'a p-adic-prod ⇒ *'a p-adic-prod* ⇒ *bool*
p-restrict ::
'a p-adic-prod ⇒ (*'a prime* ⇒ *bool*) ⇒ *'a p-adic-prod*
p-depth :: *'a prime* ⇒ *'a p-adic-prod* ⇒ *int*
⟨*proof*⟩

global-interpretation *p-complete-p-adic-prod-div-inv*:
p-complete-global-p-equal-depth-div-inv
p-equal :: *'a::nontrivial-factorial-unique-euclidean-bezout prime* ⇒
'a p-adic-prod ⇒ *'a p-adic-prod* ⇒ *bool*
p-depth :: *'a prime* ⇒ *'a p-adic-prod* ⇒ *int*
p-restrict ::
'a p-adic-prod ⇒ (*'a prime* ⇒ *bool*) ⇒ *'a p-adic-prod*
⟨*proof*⟩

lemmas *p-adic-prod-hensel* = *p-complete-p-adic-prod-div-inv.hensel*

5.6.2 By bounded global depth

The metric

instantiation *p-adic-prod* :: (*nontrivial-factorial-idom*) *metric-space*
begin

definition *dist* = *global-p-depth-p-adic-prod.bdd-global-dist*
declare *dist-p-adic-prod-def* [*simp*]

definition *uniformity* = *global-p-depth-p-adic-prod.bdd-global-uniformity*
declare *uniformity-p-adic-prod-def* [*simp*]

definition *open-p-adic-prod* :: $'a$ *p-adic-prod set* $\Rightarrow$ *bool*
 where
 open-p-adic-prod $U = (\forall\, x{\in}U.$
 eventually $(\lambda(x', y).\ x' = x \longrightarrow y \in U)$ *uniformity*
)

instance
 ⟨*proof*⟩

end

abbreviation *p-adic-prod-global-depth* ≡ *global-p-depth-p-adic-prod.bdd-global-depth*

lemma *bdd-global-depth-p-adic-prod-abs-eq*:
 p-adic-prod-global-depth (*abs-p-adic-prod* X) = *fls-pseq-bdd-global-depth* X
⟨*proof*⟩

lemma *dist-p-adic-prod-abs-eq*:
 dist (*abs-p-adic-prod* X) (*abs-p-adic-prod* Y) = *fls-pseq-bdd-global-dist* X Y
⟨*proof*⟩

lemma *p-adic-prod-metric-is-finer*:
 generate-topology p-adic-prod-p-open-nbhds $U \Longrightarrow$ *open* U
⟨*proof*⟩

lemma *p-adic-prod-metric-is-finer-than-purely-local*:
 generate-topology (*p-adic-prod-local-p-open-nbhds* p) $U \Longrightarrow$
 open U
 for p :: $'a$::*nontrivial-factorial-idom prime*
 and U :: $'a$ *p-adic-prod set*
 ⟨*proof*⟩

lemma *p-adic-prod-limseq-abs-eq*:
 $(\lambda n.$ *abs-p-adic-prod* $(X\ n)) \longlonglongrightarrow$ *abs-p-adic-prod* x
 $\longleftrightarrow$
 $(\forall\, e{>}0.\ \forall_F\ n$ *in sequentially. fls-pseq-bdd-global-dist* $(X\ n)\ x < e)$
⟨*proof*⟩

lemma *p-adic-prod-bdd-metric-LIMSEQ*:
$X \longrightarrow x = global\text{-}p\text{-}depth\text{-}p\text{-}adic\text{-}prod.metric\text{-}space\text{-}by\text{-}bdd\text{-}depth.LIMSEQ\ X\ x$
for $X :: nat \Rightarrow 'a::nontrivial\text{-}factorial\text{-}idom\ p\text{-}adic\text{-}prod$ **and** $x :: 'a\ p\text{-}adic\text{-}prod$
⟨*proof*⟩

lemma *p-adic-prod-global-depth-set-lim-closed*:
$x \in \mathcal{P}_{\forall\, p}{}^{n}$
if $lim\ : X \longrightarrow x$
and *depth*: $\forall_F\ k\ in\ sequentially.\ X\ k \in \mathcal{P}_{\forall\, p}{}^{n}$
for $X :: nat \Rightarrow 'a::nontrivial\text{-}factorial\text{-}unique\text{-}euclidean\text{-}bezout\ p\text{-}adic\text{-}prod$
and $x :: 'a\ p\text{-}adic\text{-}prod$
⟨*proof*⟩

lemma *p-adic-prod-metric-ball-multicentre*:
$ball\ y\ e = ball\ x\ e$ **if** $y \in ball\ x\ e$
for $x\ y :: 'a::nontrivial\text{-}factorial\text{-}idom\ p\text{-}adic\text{-}prod$
⟨*proof*⟩

lemma *p-adic-prod-metric-ball-at-0*:
$ball\ (0::'a::nontrivial\text{-}factorial\text{-}idom\ p\text{-}adic\text{-}prod)\ (inverse\ (2\ \widehat{}\ n)) = global\text{-}depth\text{-}set\ (int\ n)$
⟨*proof*⟩

lemma *p-adic-prod-metric-ball-UNIV*:
$ball\ x\ e = UNIV$ **if** $e > 1$
for $x :: 'a::nontrivial\text{-}factorial\text{-}idom\ p\text{-}adic\text{-}prod$
⟨*proof*⟩

lemma *p-adic-prod-metric-ball-at-0-normalize*:
$ball\ (0::'a::nontrivial\text{-}factorial\text{-}idom\ p\text{-}adic\text{-}prod)\ e =$
$global\text{-}depth\text{-}set\ \lfloor log\ 2\ (inverse\ e)\rfloor$
if $e > 0$ **and** $e \leq 1$
⟨*proof*⟩

lemma *p-adic-prod-metric-ball-translate*:
$ball\ x\ e = x +o\ ball\ 0\ e$ **for** $x :: 'a::nontrivial\text{-}factorial\text{-}idom\ p\text{-}adic\text{-}prod$
⟨*proof*⟩

lemma *p-adic-prod-left-translate-metric-continuous*:
$continuous\text{-}on\ UNIV\ ((+)\ x)$ **for** $x :: 'a::nontrivial\text{-}factorial\text{-}idom\ p\text{-}adic\text{-}prod$
⟨*proof*⟩

lemma *p-adic-prod-right-translate-metric-continuous*:
$continuous\text{-}on\ UNIV\ (\lambda z.\ z + x)$ **for** $x :: 'a::nontrivial\text{-}factorial\text{-}idom\ p\text{-}adic\text{-}prod$
⟨*proof*⟩

lemma *p-adic-prod-left-bdd-mult-bdd-metric-continuous*:
$continuous\text{-}on\ UNIV\ ((*)\ x)$

if *bdd*:
$\forall\, p$::$'a$ *prime*. $x \neg\simeq_p 0 \longrightarrow$
$(x^{\circ p}) \geq n$
for x :: $'a$::*nontrivial-factorial-idom p-adic-prod*
⟨*proof*⟩

lemma *p-adic-prod-bdd-metric-limseq-bdd-mult*:
$(\lambda k.\ w * X\ k) \longrightarrow (w * x)$
if *bdd*:
$\forall\, p$::$'a$ *prime*. $w \neg\simeq_p 0 \longrightarrow$
$(w^{\circ p}) \geq n$
and *seq*: $X \longrightarrow x$
for $w\ x$ :: $'a$::*nontrivial-factorial-idom p-adic-prod* **and** X :: *nat* ⇒ $'a$ *p-adic-prod*
⟨*proof*⟩

lemma *p-adic-prod-nonpos-global-depth-set-open*:
ball x *1* $\subseteq (\mathcal{P}_{\forall\, p}{}^n)$
if $n \leq 0$ **and** $x \in (\mathcal{P}_{\forall\, p}{}^n)$
for x :: $'a$::*nontrivial-factorial-idom p-adic-prod*
⟨*proof*⟩

lemma *p-adic-prod-global-depth-set-open*:
open $((\mathcal{P}_{\forall\, p}{}^n)$ :: $'a$::*nontrivial-factorial-idom p-adic-prod set*)
⟨*proof*⟩

lemma *p-adic-prod-ball-abs-eq*:
abs-p-adic-prod ‘ $\{Y.$ *fls-pseq-bdd-global-dist* $X\ Y < e\}$ =
ball (*abs-p-adic-prod* X) e
for x :: $'a$::*nontrivial-factorial-idom p-adic-prod*
⟨*proof*⟩

Completeness

abbreviation *p-adic-prod-globally-cauchy* ≡ *global-p-depth-p-adic-prod.globally-cauchy*
abbreviation
p-adic-prod-global-cauchy-condition ≡ *global-p-depth-p-adic-prod.global-cauchy-condition*

lemma *p-adic-prod-global-cauchy-condition-abs-eq*:
p-adic-prod-global-cauchy-condition (λn. *abs-p-adic-prod* ($X\ n$)) =
fls-pseq-global-cauchy-condition X
⟨*proof*⟩

lemma *p-adic-prod-globally-cauchy-abs-eq*:
p-adic-prod-globally-cauchy (λn. *abs-p-adic-prod* ($X\ n$)) = *fls-pseq-globally-cauchy* X
⟨*proof*⟩

lemma *p-adic-prod-globally-cauchy-vs-bdd-metric-Cauchy*:
p-adic-prod-globally-cauchy X = *Cauchy* X
for X :: *nat* ⇒ $'a$::*nontrivial-factorial-idom p-adic-prod*

⟨*proof*⟩

abbreviation
p-adic-prod-cauchy-lim X ≡
abs-p-adic-prod (λ*p*.
fls-condition-lim p
(λ*n*. *reduced-rep-p-adic-prod* (*X n*) *p*)
(*p-adic-prod-global-cauchy-condition X*)
)

lemma *p-adic-prod-bdd-metric-complete'*:
$X \longrightarrow$ *p-adic-prod-cauchy-lim X* **if** *Cauchy X*
⟨*proof*⟩

lemma *p-adic-prod-bdd-metric-complete*:
complete (*UNIV* :: *'a::nontrivial-euclidean-ring-cancel p-adic-prod set*)
⟨*proof*⟩

5.7 Hiding implementation details

lifting-update *p-adic-prod.lifting*
lifting-forget *p-adic-prod.lifting*

5.8 The subring of adelic integers

5.8.1 Type definition as a subtype of adeles

typedef (**overloaded**) *'a adelic-int* =
$\mathcal{O}_{\forall p}$ :: *'a::nontrivial-factorial-idom p-adic-prod set*
⟨*proof*⟩

lemma *Abs-adelic-int-inverse'*:
Rep-adelic-int (*Abs-adelic-int x*) = *x* **if** $x \in \mathcal{O}_{\forall p}$
⟨*proof*⟩

lemma *Abs-adelic-int-cases* [*case-names Abs-adelic-int*, *cases type*: *adelic-int*]:
P **if**
$\bigwedge x.\ z =$ *Abs-adelic-int x* $\Longrightarrow$
$x \in \mathcal{O}_{\forall p} \Longrightarrow P$
⟨*proof*⟩

lemmas *two-Abs-adelic-int-cases* = *Abs-adelic-int-cases*[*case-product Abs-adelic-int-cases*]
and *three-Abs-adelic-int-cases* =
Abs-adelic-int-cases[*case-product Abs-adelic-int-cases*[*case-product Abs-adelic-int-cases*]]

lemma *adelic-int-seq-cases* [*case-names Abs-adelic-int*]:
fixes *X* :: *nat* ⇒ *'a::nontrivial-factorial-idom adelic-int*
obtains *F* :: *nat* ⇒ *'a p-adic-prod*
where *X* = (λ*n*. *Abs-adelic-int* (*F n*))
and *range F* $\subseteq \mathcal{O}_{\forall p}$

⟨*proof*⟩

5.8.2 Algebraic instantiaions

instantiation *adelic-int* :: (*nontrivial-factorial-idom*) *comm-ring-1*
begin

definition $0 = Abs\text{-}adelic\text{-}int\ 0$

lemma *zero-adelic-int-rep-eq*: $Rep\text{-}adelic\text{-}int\ 0 = 0$
⟨*proof*⟩

definition $1 \equiv Abs\text{-}adelic\text{-}int\ 1$

lemma *one-adelic-int-rep-eq*: $Rep\text{-}adelic\text{-}int\ 1 = 1$
⟨*proof*⟩

definition $x + y = Abs\text{-}adelic\text{-}int\ (Rep\text{-}adelic\text{-}int\ x + Rep\text{-}adelic\text{-}int\ y)$

lemma *plus-adelic-int-rep-eq*: $Rep\text{-}adelic\text{-}int\ (a + b) = Rep\text{-}adelic\text{-}int\ a + Rep\text{-}adelic\text{-}int\ b$
⟨*proof*⟩

lemma *plus-adelic-int-abs-eq*:
$x \in \mathcal{O}_{\forall\, p} \Longrightarrow$
$y \in \mathcal{O}_{\forall\, p} \Longrightarrow$
$Abs\text{-}adelic\text{-}int\ x + Abs\text{-}adelic\text{-}int\ y = Abs\text{-}adelic\text{-}int\ (x + y)$
⟨*proof*⟩

definition $-x = Abs\text{-}adelic\text{-}int\ (-\ Rep\text{-}adelic\text{-}int\ x)$

lemma *uminus-adelic-int-rep-eq*: $Rep\text{-}adelic\text{-}int\ (-\ x) = -\ Rep\text{-}adelic\text{-}int\ x$
⟨*proof*⟩

lemma *uminus-adelic-int-abs-eq*:
$x \in \mathcal{O}_{\forall\, p} \Longrightarrow -\ Abs\text{-}adelic\text{-}int\ x = Abs\text{-}adelic\text{-}int\ (-\ x)$
⟨*proof*⟩

definition *minus-adelic-int* ::
$'a\ adelic\text{-}int \Rightarrow\ 'a\ adelic\text{-}int \Rightarrow\ 'a\ adelic\text{-}int$
where $minus\text{-}adelic\text{-}int \equiv (\lambda x\ y.\ x + -\ y)$

lemma *minus-adelic-int-rep-eq*: $Rep\text{-}adelic\text{-}int\ (x - y) = Rep\text{-}adelic\text{-}int\ x - Rep\text{-}adelic\text{-}int\ y$
⟨*proof*⟩

lemma *minus-adelic-int-abs-eq*:
$x \in \mathcal{O}_{\forall\, p} \Longrightarrow$
$y \in \mathcal{O}_{\forall\, p} \Longrightarrow$

$Abs\text{-}adelic\text{-}int\ x - Abs\text{-}adelic\text{-}int\ y = Abs\text{-}adelic\text{-}int\ (x - y)$
⟨*proof*⟩

definition $x * y = Abs\text{-}adelic\text{-}int\ (Rep\text{-}adelic\text{-}int\ x * Rep\text{-}adelic\text{-}int\ y)$

lemma *times-adelic-int-rep-eq*: $Rep\text{-}adelic\text{-}int\ (x * y) = Rep\text{-}adelic\text{-}int\ x * Rep\text{-}adelic\text{-}int$
y
⟨*proof*⟩

lemma *times-adelic-int-abs-eq*:
$x \in \mathcal{O}_{\forall\ p} \Longrightarrow$
$y \in \mathcal{O}_{\forall\ p} \Longrightarrow$
$Abs\text{-}adelic\text{-}int\ x * Abs\text{-}adelic\text{-}int\ y = Abs\text{-}adelic\text{-}int\ (x * y)$
⟨*proof*⟩

instance
⟨*proof*⟩

end

lemma *pow-adelic-int-rep-eq*: $Rep\text{-}adelic\text{-}int\ (x \mathbin{\widehat{}} n) = (Rep\text{-}adelic\text{-}int\ x) \mathbin{\widehat{}} n$
⟨*proof*⟩

lemma *pow-adelic-int-abs-eq*:
$(Abs\text{-}adelic\text{-}int\ x) \mathbin{\widehat{}} n = Abs\text{-}adelic\text{-}int\ (x \mathbin{\widehat{}} n)$ **if** $x \in \mathcal{O}_{\forall\ p}$
⟨*proof*⟩

5.8.3 Equivalence and depth relative to a prime

overloading
p-equal-adelic-int $\equiv$
p-equal :: $'a$::*nontrivial-factorial-idom prime* $\Rightarrow$
$'a$ *adelic-int* $\Rightarrow$ $'a$ *adelic-int* $\Rightarrow$ *bool*
p-restrict-adelic-int $\equiv$
p-restrict :: $'a$ *adelic-int* $\Rightarrow$
($'a$ *prime* $\Rightarrow$ *bool*) $\Rightarrow$ $'a$ *adelic-int*
p-depth-adelic-int $\equiv$
p-depth :: $'a$::*nontrivial-factorial-idom prime* $\Rightarrow$ $'a$ *adelic-int* $\Rightarrow$ *int*
global-unfrmzr-pows-adelic-int $\equiv$
global-unfrmzr-pows :: ($'a$ *prime* $\Rightarrow$ *nat*) $\Rightarrow$ $'a$ *adelic-int*
begin

definition *p-equal-adelic-int* ::
$'a$::*nontrivial-factorial-idom prime* $\Rightarrow$
$'a$ *adelic-int* $\Rightarrow$ $'a$ *adelic-int* $\Rightarrow$ *bool*
where *p-equal-adelic-int p x y* $\equiv$ $(Rep\text{-}adelic\text{-}int\ x) \simeq_p (Rep\text{-}adelic\text{-}int\ y)$

definition *p-restrict-adelic-int* ::
$'a$::*nontrivial-factorial-idom adelic-int* $\Rightarrow$

$('a\ prime \Rightarrow bool) \Rightarrow 'a\ adelic\text{-}int$
where
p-restrict-adelic-int x P $\equiv$ *Abs-adelic-int* ((*Rep-adelic-int x*) *prestrict P*)

definition *p-depth-adelic-int* ::
$'a$::*nontrivial-factorial-idom prime* $\Rightarrow$ $'a$ *adelic-int* $\Rightarrow$ *int*
where *p-depth-adelic-int p x* $\equiv$ ((*Rep-adelic-int x*)$^{\circ p}$)

definition *global-unfrmzr-pows-adelic-int* ::
($'a$::*nontrivial-factorial-idom prime* $\Rightarrow$ *nat*) $\Rightarrow$ $'a$ *adelic-int*
where
global-unfrmzr-pows-adelic-int f $\equiv$ *Abs-adelic-int* (*global-unfrmzr-pows* (*int* $\circ$ *f*))

end

context
fixes *p* :: $'a$::*nontrivial-factorial-idom prime*
begin

lemma *p-equal-adelic-int-abs-eq*:
(*Abs-adelic-int x*) $\simeq_p$ (*Abs-adelic-int y*) $\longleftrightarrow$
$x \simeq_p y$
if $x \in \mathcal{O}_{\forall\ p}$
and $y \in \mathcal{O}_{\forall\ p}$
for *x y* :: $'a$ *p-adic-prod*
⟨*proof*⟩

lemma *p-equal-adelic-int-abs-eq0*:
(*Abs-adelic-int x*) $\simeq_p$ *0* $\longleftrightarrow$ $x \simeq_p$ *0*
if $x \in \mathcal{O}_{\forall\ p}$ **for** *x* :: $'a$ *p-adic-prod*
⟨*proof*⟩

lemma *p-equal-adelic-int-rep-eq0*:
(*Rep-adelic-int x*) $\simeq_p$ *0* $\longleftrightarrow$ $x \simeq_p$ *0*
for *x* :: $'a$ *adelic-int*
⟨*proof*⟩

lemma *p-equal-adelic-int-abs-eq1*:
(*Abs-adelic-int x*) $\simeq_p$ *1* $\longleftrightarrow$ $x \simeq_p$ *1*
if $x \in \mathcal{O}_{\forall\ p}$ **for** *x* :: $'a$ *p-adic-prod*
⟨*proof*⟩

lemma *p-depth-adelic-int-abs-eq*:
(*Abs-adelic-int x*)$^{\circ p}$ = ($x^{\circ p}$)
if $x \in \mathcal{O}_{\forall\ p}$ **for** *x* :: $'a$ *p-adic-prod*
⟨*proof*⟩

lemma *adelic-int-depth*: ($x^{\circ p}$) $\geq$ *0* **for** *x* :: $'a$ *adelic-int*

$\langle proof \rangle$

end

lemma *p-restrict-adelic-int-rep-eq*:
Rep-adelic-int (x *prestrict* P) = (*Rep-adelic-int* x) *prestrict* P
for x :: *'a::nontrivial-factorial-idom adelic-int*
and P :: *'a prime* $\Rightarrow$ *bool*
$\langle proof \rangle$

lemma *p-restrict-adelic-int-abs-eq*:
(*Abs-adelic-int* x) *prestrict* P = *Abs-adelic-int* (x *prestrict* P)
if $x \in \mathcal{O}_{\forall\, p}$
for x :: *'a::nontrivial-factorial-idom p-adic-prod* **and** P :: *'a prime* $\Rightarrow$ *bool*
$\langle proof \rangle$

lemma *global-unfrmzr-pows-adelic-int-rep-eq*:
Rep-adelic-int ($\mathfrak{p}$ f :: *'a adelic-int*) = $\mathfrak{p}$ (*int* $\circ$ f)
for f :: *'a::nontrivial-factorial-idom prime* $\Rightarrow$ *nat*
$\langle proof \rangle$

global-interpretation *p-equality-adelic-int*:
p-equality-no-zero-divisors-1
p-equal :: *'a::nontrivial-factorial-idom prime* $\Rightarrow$
'a adelic-int $\Rightarrow$ *'a adelic-int* $\Rightarrow$ *bool*
$\langle proof \rangle$

overloading
globally-p-equal-adelic-int $\equiv$
globally-p-equal ::
'a::nontrivial-factorial-idom adelic-int $\Rightarrow$ *'a adelic-int* $\Rightarrow$ *bool*
begin

definition *globally-p-equal-adelic-int* ::
'a::nontrivial-factorial-idom adelic-int $\Rightarrow$ *'a adelic-int* $\Rightarrow$ *bool*
where *globally-p-equal-adelic-int-def*[*simp*]:
globally-p-equal-adelic-int = *p-equality-adelic-int.globally-p-equal*

end

global-interpretation *global-p-depth-adelic-int*:
global-p-equal-depth-no-zero-divisors-w-inj-index-consts
p-equal :: *'a::nontrivial-factorial-idom prime* $\Rightarrow$
'a adelic-int $\Rightarrow$ *'a adelic-int* $\Rightarrow$ *bool*
p-restrict ::
'a adelic-int $\Rightarrow$ (*'a prime* $\Rightarrow$ *bool*) $\Rightarrow$ *'a adelic-int*
p-depth :: *'a prime* $\Rightarrow$ *'a adelic-int* $\Rightarrow$ *int*
$\lambda f.$ *Abs-adelic-int* (*p-adic-prod-consts* f)
Rep-prime

⟨*proof*⟩

lemma *p-depth-adelic-int-diff-abs-eq*:
  $((\textit{Abs-adelic-int } x - \textit{Abs-adelic-int } y)^{\circ p}) = ((x - y)^{\circ p})$
  if $x \in \mathcal{O}_{\forall p}$ **and** $y \in \mathcal{O}_{\forall p}$
  for p :: $'a$::*nontrivial-factorial-idom prime* **and** $x\ y$ :: $'a$ *p-adic-prod*
  ⟨*proof*⟩

lemma *p-depth-adelic-int-diff-abs-eq'*:
  $((\textit{Abs-adelic-int } (\textit{abs-p-adic-prod } X) - \textit{Abs-adelic-int } (\textit{abs-p-adic-prod } Y))^{\circ p}) =$
    $((X - Y)^{\circ p})$
  if X: $X \in \mathcal{O}_{\forall p}$ **and** Y: $Y \in \mathcal{O}_{\forall p}$
  for p :: $'a$::*nontrivial-factorial-idom prime* **and** $X\ Y$ :: $'a$ *fls-pseq*
⟨*proof*⟩

context
  fixes p :: $'a$::*nontrivial-euclidean-ring-cancel prime*
begin

lemma *adelic-int-diff-depth-gt-equiv-trans*:
  $((a - c)^{\circ p}) > n$
  if $a \simeq_p b$ **and** $b \neg\simeq_p c$ **and** $((b - c)^{\circ p}) > n$
  for $a\ b\ c$ :: $'a$ *adelic-int*
  ⟨*proof*⟩

lemma *adelic-int-diff-depth-gt0-equiv-trans*:
  $((a - c)^{\circ p}) > n$
  if $n \geq 0$ **and** $a \simeq_p b$ **and** $((b - c)^{\circ p}) > n$
  for $a\ b\ c$ :: $'a$ *adelic-int*
  ⟨*proof*⟩

lemma *adelic-int-diff-depth-gt-trans*:
  $((a - c)^{\circ p}) > n$
  if $a \neg\simeq_p b$ **and** $((a - b)^{\circ p}) > n$
  and $b \neg\simeq_p c$ **and** $((b - c)^{\circ p}) > n$
  and $a \neg\simeq_p c$
  for $a\ b\ c$ :: $'a$ *adelic-int*
  ⟨*proof*⟩

lemma *adelic-int-diff-depth-gt0-trans*:
  $((a - c)^{\circ p}) > n$
  if $n \geq 0$
  and $((a - b)^{\circ p}) > n$
  and $((b - c)^{\circ p}) > n$
  and $a \neg\simeq_p c$
  for $a\ b\ c$ :: $'a$ *adelic-int*
  ⟨*proof*⟩

lemma *adelic-int-prestrict-zero-depth*:

$1 \le ((x\ prestrict\ (\lambda p{::}'a\ prime.\ (x^{\circ p}) = 0) - x)^{\circ p})$
if $x\ prestrict\ (\lambda p{::}'a\ prime.\ (x^{\circ p}) = 0) \neg\simeq_p x$
for $x :: 'a\ adelic\text{-}int$
⟨*proof*⟩

end

lemma *adelic-int-normalize-depth*:
$(x * \mathfrak{p}\ (\lambda p{::}'a\ prime.\ -(x^{\circ p}))) \in \mathcal{O}_{\forall\ p}$
for $x :: 'a{::}nontrivial\text{-}factorial\text{-}idom\ p\text{-}adic\text{-}prod$
⟨*proof*⟩

lemma *global-unfrmzr-pows0-adelic-int*:
$(\mathfrak{p}\ (0 :: 'a{::}nontrivial\text{-}factorial\text{-}idom\ prime \Rightarrow nat) :: 'a\ adelic\text{-}int) = 1$
⟨*proof*⟩

context
fixes $p :: 'a{::}nontrivial\text{-}factorial\text{-}idom\ prime$
begin

lemma *global-unfrmzr-pows-adelic-int*:
$(\mathfrak{p}\ f :: 'a\ adelic\text{-}int)^{\circ p} = int\ (f\ p)$ **for** $f :: 'a\ prime \Rightarrow nat$
⟨*proof*⟩

lemma *global-unfrmzr-pows-adelic-int-pequiv-iff*:
$((\mathfrak{p}\ f :: 'a\ adelic\text{-}int) \simeq_p (\mathfrak{p}\ g)) = (f\ p = g\ p)$
for $f\ g :: 'a\ prime \Rightarrow nat$
⟨*proof*⟩

end

lemma *global-unfrmzr-pows-prod-adelic-int*:
$(\mathfrak{p}\ f :: 'a\ adelic\text{-}int) * (\mathfrak{p}\ g) = \mathfrak{p}\ (f + g)$
for $f\ g :: 'a{::}nontrivial\text{-}factorial\text{-}idom\ prime \Rightarrow nat$
⟨*proof*⟩

context
fixes $p :: 'a{::}nontrivial\text{-}factorial\text{-}idom\ prime$
begin

lemma *global-unfrmzr-pows-adelic-int-nzero*:
$(\mathfrak{p}\ f :: 'a\ adelic\text{-}int) \neg\simeq_p 0$ **for** $f :: 'a\ prime \Rightarrow nat$
⟨*proof*⟩

lemma *prod-w-global-unfrmzr-pows-adelic-int*:
$(x * \mathfrak{p}\ f)^{\circ p} = (x^{\circ p}) + int\ (f\ p)$
if $x \neg\simeq_p 0$ **for** $f :: 'a\ prime \Rightarrow nat$ **and** $x :: 'a\ adelic\text{-}int$
⟨*proof*⟩

lemma *trivial-global-unfrmzr-pows-adelic-int*:
  $(\mathfrak{p}\ f :: 'a\ adelic\text{-}int) \simeq_p 1$ **if** $f\ p = 0$ **for** $f :: 'a\ prime \Rightarrow nat$
  ⟨*proof*⟩

lemma *prod-w-trivial-global-unfrmzr-pows-adelic-int*:
  $x * \mathfrak{p}\ f \simeq_p x$ **if** $f\ p = 0$
  for $f :: 'a\ prime \Rightarrow nat$ **and** $x :: 'a\ adelic\text{-}int$
  ⟨*proof*⟩

end

lemma *pow-global-unfrmzr-pows-adelic-int*:
  $(\mathfrak{p}\ f :: 'a\ adelic\text{-}int)\ \widehat{}\ n = (\mathfrak{p}\ ((\lambda p.\ n) * f))$
  for $f :: 'a{::}nontrivial\text{-}factorial\text{-}idom\ prime \Rightarrow nat$
⟨*proof*⟩

abbreviation *adelic-int-consts* ≡ *global-p-depth-adelic-int.consts*
abbreviation *adelic-int-const* ≡ *global-p-depth-adelic-int.const*

lemma *adelic-int-p-depth-consts*:
  $(adelic\text{-}int\text{-}consts\ f)^{\circ p} = ((f\ p)^{\circ p})$
  for $p :: 'a{::}nontrivial\text{-}factorial\text{-}idom\ prime$ **and** $f :: 'a\ prime \Rightarrow 'a$
  ⟨*proof*⟩

lemmas *adelic-int-p-depth-const* = *adelic-int-p-depth-consts*[*of* - $\lambda p.$ -]

lemma *adelic-int-global-unfrmzr*:
  $(\mathfrak{p}\ (1 :: 'a{::}nontrivial\text{-}factorial\text{-}idom\ prime \Rightarrow nat) :: 'a\ adelic\text{-}int)$
    $= adelic\text{-}int\text{-}consts\ Rep\text{-}prime$
⟨*proof*⟩

lemma *adelic-int-normalize-depthE*:
  fixes $x :: 'a{::}nontrivial\text{-}factorial\text{-}idom\ adelic\text{-}int$
  obtains *x-0*
  where $\forall p{::}'a\ prime.\ x\text{-}0^{\circ p} = 0$
  and $x = x\text{-}0 * \mathfrak{p}\ (\lambda p{::}'a\ prime.\ nat\ (x^{\circ p}))$
⟨*proof*⟩

5.8.4 Inverses

We can create inverses at depth-zero places.

instantiation *adelic-int* ::
  (*nontrivial-factorial-unique-euclidean-bezout*) {*divide-trivial*, *inverse*}
begin

definition *inverse-adelic-int* ::
  'a adelic-int ⇒ *'a adelic-int*
  where
    inverse-adelic-int x =

Abs-adelic-int (
  (*inverse* (*Rep-adelic-int* x)) *prestrict* (λp::$'a$ *prime*. $x^{\circ p} = 0$)
)

definition *divide-adelic-int* ::
$'a$ *adelic-int* $\Rightarrow$ $'a$ *adelic-int* $\Rightarrow$ $'a$ *adelic-int*
where *divide-adelic-int-def* [*simp*]: *divide-adelic-int* x y = x $*$ *inverse* y

instance
⟨*proof*⟩

end

lemma *inverse-adelic-int-abs-eq*:
inverse (*Abs-adelic-int* x) =
  Abs-adelic-int (*inverse* x *prestrict* (λp::$'a$ *prime*. $x^{\circ p} = 0$))
if $x \in \mathcal{O}_{\forall\, p}$
for x :: $'a$::*nontrivial-factorial-unique-euclidean-bezout p-adic-prod*
⟨*proof*⟩

lemma *divide-adelic-int-abs-eq*:
Abs-adelic-int x / *Abs-adelic-int* y =
  Abs-adelic-int ((x / y) *prestrict* (λp::$'a$ *prime*. $y^{\circ p} = 0$))
if $x \in \mathcal{O}_{\forall\, p}$ **and** $y \in \mathcal{O}_{\forall\, p}$
for x y :: $'a$::*nontrivial-factorial-unique-euclidean-bezout p-adic-prod*
⟨*proof*⟩

lemma *p-depth-adelic-int-inverse-diff-abs-eq*:
((*inverse* (*Abs-adelic-int* x) $-$ *inverse* (*Abs-adelic-int* y))$^{\circ p}$) =
  ((*inverse* x $-$ *inverse* y)$^{\circ p}$)
if $x \in \mathcal{O}_{\forall\, p}$ $x^{\circ p} = 0$
and $y \in \mathcal{O}_{\forall\, p}$ $y^{\circ p} = 0$
for p :: $'a$::*nontrivial-factorial-unique-euclidean-bezout prime* **and** x y :: $'a$ *p-adic-prod*
⟨*proof*⟩

context
fixes x :: $'a$::*nontrivial-factorial-unique-euclidean-bezout adelic-int*
begin

lemma *inverse-adelic-int-rep-eq*:
Rep-adelic-int (*inverse* x) =
  (*inverse* (*Rep-adelic-int* x)) *prestrict* (λp::$'a$ *prime*. $x^{\circ p} = 0$)
⟨*proof*⟩

lemma *adelic-int-inverse-equiv0-iff*:
inverse $x \simeq_p 0 \longleftrightarrow$
  ($x \simeq_p 0 \vee (x^{\circ p}) \neq 0$)
for p :: $'a$ *prime*
⟨*proof*⟩

lemma *adelic-int-inverse-equiv0-iff'*:
$\textit{inverse } x \simeq_p 0 \longleftrightarrow$
$(x \simeq_p 0 \vee (x^{\circ p}) \geq 1)$
for p :: *'a prime*
⟨*proof*⟩

end

lemma *adelic-int-inverse-eq0-iff*:
$\textit{inverse } x = 0 \longleftrightarrow$
$(\forall p{::}'a \textit{ prime}.\ x^{\circ p} = 0 \longrightarrow x \simeq_p 0)$
for x :: *'a::nontrivial-factorial-unique-euclidean-bezout adelic-int*
⟨*proof*⟩

lemma *divide-adelic-int-rep-eq*:
Rep-adelic-int $(x\ /\ y) =$
$(\textit{Rep-adelic-int } x\ /\ \textit{Rep-adelic-int } y)\ \textit{prestrict}\ (\lambda p{::}'a \textit{ prime}.\ y^{\circ p} = 0)$
for $x\ y$ :: *'a::nontrivial-factorial-unique-euclidean-bezout adelic-int*
⟨*proof*⟩

lemma *adelic-int-divide-by-pequiv0*:
$x\ /\ y \simeq_p 0$ **if** $y \simeq_p 0$
for p :: *'a::nontrivial-factorial-unique-euclidean-bezout prime* **and** $x\ y$ :: *'a adelic-int*
⟨*proof*⟩

lemma *adelic-int-divide-by-pos-depth*:
$x\ /\ y \simeq_p 0$ **if** $(y^{\circ p}) > 0$
for p :: *'a::nontrivial-factorial-unique-euclidean-bezout prime* **and** $x\ y$ :: *'a adelic-int*
⟨*proof*⟩

lemma *adelic-int-inverse0*[*simp*]:
inverse (*0::'a::nontrivial-factorial-unique-euclidean-bezout adelic-int*) = *0*
⟨*proof*⟩

lemma *adelic-int-inverse1*[*simp*]:
inverse (*1::'a::nontrivial-factorial-unique-euclidean-bezout adelic-int*) = *1*
⟨*proof*⟩

lemma *adelic-int-inverse-mult*:
$\textit{inverse } (x * y) = \textit{inverse } x * \textit{inverse } y$
for $x\ y$ :: *'a::nontrivial-factorial-unique-euclidean-bezout adelic-int*
⟨*proof*⟩

lemma *adelic-int-inverse-pcong*:
$(\textit{inverse } x) \simeq_p (\textit{inverse } y)$ **if** $x \simeq_p y$
for p :: *'a::nontrivial-factorial-unique-euclidean-bezout prime* **and** $x\ y$ :: *'a adelic-int*
⟨*proof*⟩

context
fixes x :: $'a$::*nontrivial-factorial-unique-euclidean-bezout adelic-int*
begin

lemma *adelic-int-inverse-uminus*: $inverse\ (-x) = -\ inverse\ x$
⟨*proof*⟩

lemma *adelic-int-inverse-inverse*:
$inverse\ (inverse\ x) = x\ prestrict\ (\lambda p::'a\ prime.\ x^{\circ p} = 0)$
⟨*proof*⟩

lemma *adelic-int-inverse-pow*: $inverse\ (x\ \widehat{}\ n) = (inverse\ x)\ \widehat{}\ n$
⟨*proof*⟩

lemma *adelic-int-divide-self'*:
$x\ /\ x \simeq_p 1$ **if** $x \neg\simeq_p 0$ **and** $x^{\circ p} = 0$
for p :: $'a\ prime$
⟨*proof*⟩

lemma *adelic-int-global-divide-self*:
$x\ /\ x = 1$
if $\forall p::'a\ prime.\ x \neg\simeq_p 0$
and $\forall p::'a\ prime.\ x^{\circ p} = 0$
⟨*proof*⟩

lemma *adelic-int-divide-self*:
$x\ /\ x =$
$1\ prestrict\ (\lambda p::'a\ prime.\ x \neg\simeq_p 0 \wedge x^{\circ p} = 0)$
⟨*proof*⟩

lemma *adelic-int-p-restrict-inverse*:
$(inverse\ x)\ prestrict\ P = inverse\ (x\ prestrict\ P)$ **for** P :: $'a\ prime \Rightarrow bool$
⟨*proof*⟩

lemma *adelic-int-inverse-depth*: $(inverse\ x)^{\circ p} = 0$ **for** p :: $'a\ prime$
⟨*proof*⟩

end

lemma *adelic-int-consts-divide-self*:
$(adelic\text{-}int\text{-}consts\ f)\ /\ (adelic\text{-}int\text{-}consts\ f) =$
$1\ prestrict\ (\lambda p::'a\ prime.\ f\ p \neq 0 \wedge (f\ p)^{\circ p} = 0)$
for f :: $'a$::*nontrivial-factorial-unique-euclidean-bezout prime* $\Rightarrow 'a$
⟨*proof*⟩

lemma *adelic-int-const-divide-self*:
$(adelic\text{-}int\text{-}const\ c)\ /\ (adelic\text{-}int\text{-}const\ c) =$
$1\ prestrict\ (\lambda p::'a\ prime.\ c^{\circ p} = 0)$
if $c \neq 0$ **for** c :: $'a$::*nontrivial-factorial-unique-euclidean-bezout*

$\langle proof \rangle$

lemma *adelic-int-consts-divide-self'*:
$(adelic\text{-}int\text{-}consts\ f)\ /\ (adelic\text{-}int\text{-}consts\ f) \simeq_p 1$
if $f\ p \neq 0$ **and** $(f\ p)^{\circ p} = 0$
for $f :: 'a::nontrivial\text{-}factorial\text{-}unique\text{-}euclidean\text{-}bezout\ prime \Rightarrow 'a$
$\langle proof \rangle$

lemma *adelic-int-divide-depth*:
$(x\ /\ y)^{\circ p} = (x^{\circ p})$ **if** $x\ /\ y\ \neg\simeq_p 0$
for $x\ y :: 'a::nontrivial\text{-}factorial\text{-}unique\text{-}euclidean\text{-}bezout\ adelic\text{-}int$ **and** $p :: 'a\ prime$
$\langle proof \rangle$

lemma *global-unfrmzr-pows-adelic-int-inverse*:
$inverse\ (\mathfrak{p}\ f :: 'a\ adelic\text{-}int) = 1\ prestrict\ (\lambda p{::}'a\ prime.\ f\ p = 0)$
for $f :: 'a::nontrivial\text{-}factorial\text{-}unique\text{-}euclidean\text{-}bezout\ prime \Rightarrow nat$
$\langle proof \rangle$

lemma *adelic-int-diff-cancel-lead-coeff*:
$((inverse\ x - inverse\ y)^{\circ p}) > 0$
if $x : x\ \neg\simeq_p 0\ x^{\circ p} = 0$
and $y : y\ \neg\simeq_p 0\ y^{\circ p} = 0$
and xy: $x\ \neg\simeq_p y\ ((x - y)^{\circ p}) > 0$
for $p :: 'a::nontrivial\text{-}factorial\text{-}unique\text{-}euclidean\text{-}bezout\ prime$
and $x\ y :: 'a\ adelic\text{-}int$
$\langle proof \rangle$

5.8.5 The ideal of primes

overloading
p-depth-set-adelic-int ≡
p-depth-set ::
$'a::nontrivial\text{-}factorial\text{-}idom\ prime \Rightarrow int \Rightarrow 'a\ adelic\text{-}int\ set$
global-depth-set-adelic-int ≡
global-depth-set :: $int \Rightarrow 'a\ adelic\text{-}int\ set$
begin

definition *p-depth-set-adelic-int* ::
$'a::nontrivial\text{-}factorial\text{-}idom\ prime \Rightarrow int \Rightarrow 'a\ adelic\text{-}int\ set$
where *p-depth-set-adelic-int-def*[*simp*]:
p-depth-set-adelic-int = *global-p-depth-adelic-int.p-depth-set*

definition *global-depth-set-adelic-int* ::
$int \Rightarrow 'a::nontrivial\text{-}factorial\text{-}idom\ adelic\text{-}int\ set$
where *global-depth-set-adelic-int-def*[*simp*]:
global-depth-set-adelic-int = *global-p-depth-adelic-int.global-depth-set*

end

context
  fixes x :: $'a$::*nontrivial-factorial-idom adelic-int*
begin

lemma *adelic-int-global-depth-setI*:
  $x \in \mathcal{P}_{\forall\, p}{}^n$
  if $\bigwedge p::'a\ prime.\ x \neg\simeq_p 0 \Longrightarrow (x^{\circ p}) \geq n$
  ⟨*proof*⟩

lemma *adelic-int-global-depth-setI'*:
  $x \in \mathcal{P}_{\forall\, p}{}^n$
  if $\bigwedge p::'a\ prime.\ (x^{\circ p}) \geq n$
  ⟨*proof*⟩

lemma *adelic-int-global-depth-set-p-restrictI*:
  $p\text{-}restrict\ x\ P \in \mathcal{P}_{\forall\, p}{}^n$
  if $\bigwedge p::'a\ prime.\ P\ p \Longrightarrow x \in \mathcal{P}_{@p}{}^n$
  ⟨*proof*⟩

end

context
  fixes p :: $'a$::*nontrivial-factorial-idom prime*
begin

lemma *adelic-int-p-depth-setI*:
  $x \in \mathcal{P}_{@p}{}^n$
  if $x \neg\simeq_p 0 \longrightarrow (x^{\circ p}) \geq n$
  for x :: $'a$ *adelic-int*
  ⟨*proof*⟩

lemma *nonpos-p-depth-set-adelic-int*:
  $(\mathcal{P}_{@p}{}^n) = (UNIV :: 'a::nontrivial\text{-}factorial\text{-}idom\ adelic\text{-}int\ set)$
  if $n \leq 0$
⟨*proof*⟩

lemma *p-depth-set-adelic-int-eq-projection*:
  $((\mathcal{P}_{@p}{}^n) :: 'a::nontrivial\text{-}factorial\text{-}idom\ adelic\text{-}int\ set) =$
    $Abs\text{-}adelic\text{-}int\ `\ ((\mathcal{P}_{@p}{}^n) \cap \mathcal{O}_{\forall\, p})$
⟨*proof*⟩

lemma *lift-p-depth-set-adelic-int*:
  Rep-adelic-int ‘
    $((\mathcal{P}_{@p}{}^n) :: 'a::nontrivial\text{-}factorial\text{-}idom\ adelic\text{-}int\ set)$
    $= (\mathcal{P}_{@p}{}^n) \cap \mathcal{O}_{\forall\, p}$
⟨*proof*⟩

end

lemma *nonpos-global-depth-set-adelic-int*:
  $(\mathcal{P}_{\forall\, p}{}^{n}) = (UNIV :: 'a::nontrivial\text{-}factorial\text{-}idom\ adelic\text{-}int\ set)$
  if $n \leq 0$
  ⟨*proof*⟩

lemma *nonneg-global-depth-set-adelic-int-eq-projection*:
  $((\mathcal{P}_{\forall\, p}{}^{n})\ :: 'a::nontrivial\text{-}factorial\text{-}idom\ adelic\text{-}int\ set) =$
    $Abs\text{-}adelic\text{-}int\ `\ \mathcal{P}_{\forall\, p}{}^{n}$
  if n: $n \geq 0$
⟨*proof*⟩

lemma *lift-nonneg-global-depth-set-adelic-int*:
  $Rep\text{-}adelic\text{-}int\ `$
    $((\mathcal{P}_{\forall\, p}{}^{n})\ :: 'a::nontrivial\text{-}factorial\text{-}idom\ adelic\text{-}int\ set)$
    $= \mathcal{P}_{\forall\, p}{}^{n}$
  if n: $n \geq 0$
⟨*proof*⟩

5.8.6 Topology p-open neighbourhoods

General properties of p-open sets

abbreviation *adelic-int-p-open-nbhd* $\equiv$ *global-p-depth-adelic-int.p-open-nbhd*
abbreviation *adelic-int-p-open-nbhds* $\equiv$ *global-p-depth-adelic-int.p-open-nbhds*
abbreviation *adelic-int-local-p-open-nbhds* $\equiv$ *global-p-depth-adelic-int.local-p-open-nbhds*

lemma *adelic-int-nonpos-p-open-nbhd*:
  adelic-int-p-open-nbhd p n x = *UNIV* **if** $n \leq 0$
  for $p :: 'a::nontrivial\text{-}factorial\text{-}idom\ prime$
⟨*proof*⟩

lemma *adelic-int-p-open-nbhd-bound*:
  adelic-int-p-open-nbhd p n x = *adelic-int-p-open-nbhd p* (*int* (*nat n*)) *x*
  ⟨*proof*⟩

lemma *adelic-int-p-open-nbhd-abs-eq*:
  adelic-int-p-open-nbhd p n (*Abs-adelic-int x*) =
    Abs-adelic-int ‘ (*p-adic-prod-p-open-nbhd p n x* $\cap \mathcal{O}_{\forall\, p}$)
  if $x \in \mathcal{O}_{\forall\, p}$
  for $p :: 'a::nontrivial\text{-}factorial\text{-}idom\ prime$
⟨*proof*⟩

lemma *adelic-int-p-open-nbhd-rep-eq*:
  Rep-adelic-int ‘ *adelic-int-p-open-nbhd p n x* =
    p-adic-prod-p-open-nbhd p n (*Rep-adelic-int x*) $\cap \mathcal{O}_{\forall\, p}$
  for $p :: 'a::nontrivial\text{-}factorial\text{-}idom\ prime$
⟨*proof*⟩

lemma *adelic-int-local-depth-ring-lift-cover*:

fixes n :: *int*
and p :: *′a::nontrivial-factorial-unique-euclidean-bezout prime*
and $\mathcal{A}$ :: *′a adelic-int set set*
defines
$(\mathcal{A}'$ :: *′a p-adic-prod set set*$) \equiv$
$(\lambda A.$
$(\lambda x.\ x$ *prestrict* $((=)\ p))$ ‘ *Rep-adelic-int* ‘ A +
range $(\lambda x.\ x$ *prestrict* $((\neq)\ p))$
) ‘ $\mathcal{A}$
assumes *nonneg*: $n \geq 0$
and *cover*:
$(\lambda x.\ x$ *prestrict* $((=)\ p))$ ‘ $(\mathcal{P}_{\forall\ p}{}^{n}) \subseteq$
$\bigcup\ \mathcal{A}$
shows
$(\lambda x.\ x$ *prestrict* $((=)\ p))$ ‘ $(\mathcal{P}_{\forall\ p}{}^{n}) \subseteq$
$\bigcup\ \mathcal{A}'$
⟨*proof*⟩

lemma *adelic-int-lift-local-p-open*:
fixes p :: *′a::nontrivial-factorial-unique-euclidean-bezout prime*
and A :: *′a adelic-int set*
defines
$A' \equiv$
$(\lambda x.\ x$ *prestrict* $((=)\ p))$ ‘ *Rep-adelic-int* ‘ A +
range $(\lambda x.\ x$ *prestrict* $((\neq)\ p))$
assumes *adelic-int-p-open*: *generate-topology* (*adelic-int-local-p-open-nbhds* p) A
shows *generate-topology* (*p-adic-prod-local-p-open-nbhds* p) A'
⟨*proof*⟩

Sequences

abbreviation
adelic-int-p-limseq-condition $\equiv$ *global-p-depth-adelic-int.p-limseq-condition*
abbreviation
adelic-int-p-cauchy-condition $\equiv$ *global-p-depth-adelic-int.p-cauchy-condition*
abbreviation *adelic-int-p-limseq* $\equiv$ *global-p-depth-adelic-int.p-limseq*
abbreviation *adelic-int-p-cauchy* $\equiv$ *global-p-depth-adelic-int.p-cauchy*
abbreviation *adelic-int-p-open-nbhds-limseq* $\equiv$ *global-p-depth-adelic-int.p-open-nbhds-LIMSEQ*

context
fixes p :: *′a::nontrivial-factorial-idom prime*
and X :: *nat* $\Rightarrow$ *′a p-adic-prod*
and Y :: *nat* $\Rightarrow$ *′a adelic-int*
defines $Y \equiv \lambda n.$ *Abs-adelic-int* $(X\ n)$
assumes *range-X*: *range* $X \subseteq \mathcal{O}_{\forall\ p}$
begin

lemma *adelic-int-p-limseq-condition-abs-eq*:
adelic-int-p-limseq-condition $p\ Y$ (*Abs-adelic-int* x) =
p-adic-prod-p-limseq-condition $p\ X\ x$

if $x \in \mathcal{O}_{\forall\, p}$
⟨*proof*⟩

lemma *adelic-int-p-limseq-abs-eq*:
adelic-int-p-limseq p Y (*Abs-adelic-int* x) = *p-adic-prod-p-limseq* p X x
if $x \in \mathcal{O}_{\forall\, p}$
⟨*proof*⟩

lemma *adelic-int-p-cauchy-condition-abs-eq*:
adelic-int-p-cauchy-condition p Y = *p-adic-prod-p-cauchy-condition* p X
⟨*proof*⟩

lemma *adelic-int-p-cauchy-abs-eq*: *adelic-int-p-cauchy* p Y = *p-adic-prod-p-cauchy* p X
⟨*proof*⟩

end

lemma *adelic-int-p-open-nbhds-limseq-abs-eq*:
adelic-int-p-open-nbhds-limseq (λn. *Abs-adelic-int* (X n)) (*Abs-adelic-int* x) =
p-adic-prod-p-open-nbhds-limseq X x
if *range* $X \subseteq \mathcal{O}_{\forall\, p}$ **and** $x \in \mathcal{O}_{\forall\, p}$
⟨*proof*⟩

5.8.7 Topology via global depth

The metric

instantiation *adelic-int* :: (*nontrivial-factorial-idom*) *metric-space*
begin

definition *dist* = *global-p-depth-adelic-int.bdd-global-dist*
declare *dist-adelic-int-def* [*simp*]

definition *uniformity* = *global-p-depth-adelic-int.bdd-global-uniformity*
declare *uniformity-adelic-int-def* [*simp*]

definition *open-adelic-int* :: *'a adelic-int set* ⇒ *bool*
where
open-adelic-int U =
($\forall\, x \in U$.
eventually ($\lambda(x', y)$. $x' = x \longrightarrow y \in U$) *uniformity*
)

instance
⟨*proof*⟩

end

abbreviation *adelic-int-global-depth* ≡ *global-p-depth-adelic-int.bdd-global-depth*

lemma *adelic-int-bdd-global-depth-abs-eq*:
adelic-int-global-depth (*Abs-adelic-int x*) = *p-adic-prod-global-depth x*
if $x \in \mathcal{O}_{\forall\, p}$ **for** *x* :: *'a::nontrivial-factorial-idom p-adic-prod*
⟨*proof*⟩

lemma *dist-adelic-int-abs-eq*:
dist (*Abs-adelic-int x*) (*Abs-adelic-int y*) = *dist x y*
if $x \in \mathcal{O}_{\forall\, p}$ **and** $y \in \mathcal{O}_{\forall\, p}$
for *x y* :: *'a::nontrivial-factorial-idom p-adic-prod*
⟨*proof*⟩

lemma *adelic-int-limseq-abs-eq*:
(λn. *Abs-adelic-int* (*X n*)) $\longrightarrow$ *Abs-adelic-int x*
$\longleftrightarrow$ *X* $\longrightarrow$ *x*
if *range X* $\subseteq \mathcal{O}_{\forall\, p}$ **and** $x \in \mathcal{O}_{\forall\, p}$
⟨*proof*⟩

lemma *adelic-int-bdd-metric-LIMSEQ*:
X $\longrightarrow$ *x* = *global-p-depth-adelic-int.metric-space-by-bdd-depth.LIMSEQ X x*
for *X* :: *nat* ⇒ *'a::nontrivial-factorial-idom adelic-int* **and** *x* :: *'a adelic-int*
⟨*proof*⟩

lemma *adelic-int-global-depth-set-lim-closed*:
$x \in \mathcal{P}_{\forall\, p}{}^{n}$
if *lim* : *X* $\longrightarrow$ *x*
and *depth*: $\forall_F$ *k in sequentially. X k* $\in \mathcal{P}_{\forall\, p}{}^{n}$
for *X* :: *nat* ⇒ *'a::nontrivial-factorial-unique-euclidean-bezout adelic-int*
and *x* :: *'a adelic-int*
⟨*proof*⟩

lemma *adelic-int-metric-ball-multicentre*:
ball y e = *ball x e* **if** *y* ∈ *ball x e* **for** *x y* :: *'a::nontrivial-factorial-idom adelic-int*
⟨*proof*⟩

lemma *adelic-int-metric-ball-at-0*:
ball (*0::'a::nontrivial-factorial-idom adelic-int*) (*inverse* (*2* ^ *n*)) = *global-depth-set* (*int n*)
⟨*proof*⟩

lemma *adelic-int-metric-ball-at-0-normalize*:
ball (*0::'a::nontrivial-factorial-idom adelic-int*) *e* =
global-depth-set ⌊*log 2* (*inverse e*)⌋
if $e > 0$ **and** $e \leq 1$
⟨*proof*⟩

lemma *adelic-int-metric-ball-translate*:
ball x e = *x* +*o* *ball 0 e* **for** *x* :: *'a::nontrivial-factorial-idom adelic-int*
⟨*proof*⟩

lemma *adelic-int-metric-ball-UNIV*:
ball x e = UNIV **if** $e \geq 1$ **for** x :: *'a::nontrivial-factorial-idom adelic-int*
⟨*proof*⟩

lemma *adelic-int-left-translate-metric-continuous*:
continuous-on UNIV $((+)\ x)$ **for** x :: *'a::nontrivial-factorial-idom adelic-int*
⟨*proof*⟩

lemma *adelic-int-right-translate-metric-continuous*:
continuous-on UNIV $(\lambda z.\ z + x)$ **for** x :: *'a::nontrivial-factorial-idom adelic-int*
⟨*proof*⟩

lemma *adelic-int-left-mult-bdd-metric-continuous*:
continuous-on UNIV $((*)\ x)$
for x :: *'a::nontrivial-factorial-idom adelic-int*
⟨*proof*⟩

lemma *adelic-int-bdd-metric-limseq-mult*:
$(\lambda k.\ w * X\ k) \longrightarrow (w * x)$ **if** $X \longrightarrow x$
for *w x* :: *'a::nontrivial-factorial-idom adelic-int* **and** *X* :: *nat* ⇒ *'a adelic-int*
⟨*proof*⟩

lemma *adelic-int-global-depth-set-open*:
open $((\mathcal{P}_{\forall\, p}{}^{n})$ :: *'a::nontrivial-factorial-idom adelic-int set*)
⟨*proof*⟩

lemma *adelic-int-ball-abs-eq*:
Abs-adelic-int ‘ ball x e = ball (Abs-adelic-int x) e
if $x \in \mathcal{O}_{\forall\, p}$ **and** $e \leq 1$
for x :: *'a::nontrivial-factorial-idom p-adic-prod*
⟨*proof*⟩

lemma *open-adelic-int-abs-eq*:
open U = open (Abs-adelic-int ‘ U) **if** $U \subseteq (\mathcal{O}_{\forall\, p})$
for U :: *'a::nontrivial-factorial-idom p-adic-prod set*
⟨*proof*⟩

Completeness

abbreviation *adelic-int-globally-cauchy* ≡ *global-p-depth-adelic-int.globally-cauchy*
abbreviation
adelic-int-global-cauchy-condition ≡ *global-p-depth-adelic-int.global-cauchy-condition*

lemma *adelic-int-global-cauchy-condition-abs-eq*:
adelic-int-global-cauchy-condition (*λn. Abs-adelic-int* (*X n*)) =
p-adic-prod-global-cauchy-condition X
if *range X* $\subseteq \mathcal{O}_{\forall\, p}$
⟨*proof*⟩

lemma *adelic-int-globally-cauchy-abs-eq*:
adelic-int-globally-cauchy (λn. *Abs-adelic-int* (X n)) = *p-adic-prod-globally-cauchy* X
if *range* $X \subseteq \mathcal{O}_{\forall\, p}$
⟨*proof*⟩

lemma *adelic-int-globally-cauchy-vs-bdd-metric-Cauchy*:
adelic-int-globally-cauchy X = *Cauchy* X
for X :: *nat* ⇒ *'a::nontrivial-factorial-idom adelic-int*
⟨*proof*⟩

lemma *adelic-int-Cauchy-abs-eq*:
Cauchy (λn. *Abs-adelic-int* (X n)) = *Cauchy* X
if *range* $X \subseteq \mathcal{O}_{\forall\, p}$
for X :: *nat* ⇒ *'a::nontrivial-factorial-idom p-adic-prod*
⟨*proof*⟩

abbreviation
adelic-int-cauchy-lim X ≡
Abs-adelic-int (*p-adic-prod-cauchy-lim* (λn. *Rep-adelic-int* (X n)))

lemma *adelic-int-bdd-metric-complete'*:
$X \longrightarrow$ *adelic-int-cauchy-lim* X **if** *Cauchy* X
for X :: *nat* ⇒ *'a::nontrivial-factorial-unique-euclidean-bezout adelic-int*
⟨*proof*⟩

lemma *adelic-int-bdd-metric-complete*:
complete (*UNIV* :: *'a::nontrivial-factorial-unique-euclidean-bezout adelic-int set*)
⟨*proof*⟩

end

theory *Fin-Field-Product*

imports
Distinguished-Quotients
pAdic-Product

begin

6 Finite fields of prime order as quotients of the ring of integers of p-adic fields

6.1 The type

6.1.1 The prime ideal as the distinguished ideal of the ring of adelic integers

instantiation *adelic-int* ::
  (*nontrivial-factorial-idom*) *comm-ring-1-with-distinguished-ideal*
begin

definition *distinguished-subset-adelic-int* :: *'a adelic-int set*
  where *distinguished-subset-adelic-int-def* [*simp*]:
    distinguished-subset-adelic-int $\equiv \mathcal{P}_{\forall\, p}$

instance
⟨*proof*⟩

end

lemma *distinguished-subset-adelic-int-inverse*:
  inverse $x = 0$ **if** $x \in \mathcal{P}_{\forall\, p}$
  for x :: *'a::nontrivial-factorial-unique-euclidean-bezout adelic-int*
  ⟨*proof*⟩

6.1.2 The finite field product type as a quotient

Create type wrapper *coset-by-dist-sbgrp* over *adelic-int*.

typedef (**overloaded**) *'a adelic-int-quot* =
  UNIV :: *'a::nontrivial-factorial-idom adelic-int coset-by-dist-sbgrp set*
  ⟨*proof*⟩

abbreviation
  adelic-int-quot $\equiv \lambda x.$ *Abs-adelic-int-quot* (*distinguished-quotient-coset* x)
abbreviation *p-adic-prod-int-quot* $\equiv \lambda x.$ *adelic-int-quot* (*Abs-adelic-int* x)

lemma *adelic-int-quot-cases* [*case-names adelic-int-quot*]:
  obtains y **where** $x =$ *adelic-int-quot* y
⟨*proof*⟩

lemma *Abs-adelic-int-quot-cases* [*case-names adelic-int-quot-Abs-adelic-int*]:
  obtains z **where** $x =$ *p-adic-prod-int-quot* z $z \in \mathcal{O}_{\forall\, p}$
⟨*proof*⟩

lemmas *two-adelic-int-quot-cases* = *adelic-int-quot-cases*[*case-product adelic-int-quot-cases*]
  and *three-adelic-int-quot-cases* =
  adelic-int-quot-cases[*case-product adelic-int-quot-cases*[*case-product adelic-int-quot-cases*]]

lemma *abs-adelic-int-eq-iff*:
adelic-int-quot x = *adelic-int-quot* $y \longleftrightarrow$
$y - x \in \mathcal{P}_{\forall\, p}$
⟨*proof*⟩

lemma *p-adic-prod-int-quot-eq-iff*:
p-adic-prod-int-quot x = *p-adic-prod-int-quot* $y \longleftrightarrow$
$y - x \in \mathcal{P}_{\forall\, p}$
if $x \in \mathcal{O}_{\forall\, p}$ **and** $y \in \mathcal{O}_{\forall\, p}$
⟨*proof*⟩

6.1.3 Algebraic instantiations

instantiation *adelic-int-quot* :: (*nontrivial-factorial-idom*) *comm-ring-1*
begin

definition *0* = *Abs-adelic-int-quot 0*

lemma *zero-adelic-int-quot-rep-eq*: *Rep-adelic-int-quot 0* = *0*
⟨*proof*⟩

lemma *zero-adelic-int-quot*: *0* = *adelic-int-quot 0*
⟨*proof*⟩

definition *1* ≡ *Abs-adelic-int-quot 1*

lemma *one-adelic-int-quot-rep-eq*: *Rep-adelic-int-quot 1* = *1*
⟨*proof*⟩

lemma *one-adelic-int-quot*: *1* = *adelic-int-quot 1*
⟨*proof*⟩

definition $x + y$ = *Abs-adelic-int-quot* (*Rep-adelic-int-quot* x + *Rep-adelic-int-quot* y)

lemma *plus-adelic-int-quot-rep-eq*:
Rep-adelic-int-quot $(a + b)$ = *Rep-adelic-int-quot* a + *Rep-adelic-int-quot* b
⟨*proof*⟩

lemma *plus-adelic-int-quot-abs-eq*:
Abs-adelic-int-quot x + *Abs-adelic-int-quot* y = *Abs-adelic-int-quot* $(x + y)$
⟨*proof*⟩

lemma *plus-adelic-int-quot*: *adelic-int-quot* x + *adelic-int-quot* y = *adelic-int-quot* $(x + y)$
⟨*proof*⟩

definition $-x$ = *Abs-adelic-int-quot* ($-$ *Rep-adelic-int-quot* x)

lemma *uminus-adelic-int-quot-rep-eq*: *Rep-adelic-int-quot* $(- x) = -$ *Rep-adelic-int-quot* x
⟨*proof*⟩

lemma *uminus-adelic-int-quot-abs-eq*: $-$ *Abs-adelic-int-quot* $x =$ *Abs-adelic-int-quot* $(- x)$
⟨*proof*⟩

lemma *uminus-adelic-int-quot*: $-$ *adelic-int-quot* $x =$ *adelic-int-quot* $(-x)$
⟨*proof*⟩

definition *minus-adelic-int-quot* ::
$'a$ *adelic-int-quot* $\Rightarrow$ $'a$ *adelic-int-quot* $\Rightarrow$ $'a$ *adelic-int-quot*
where *minus-adelic-int-quot* $\equiv (\lambda x\ y.\ x + - y)$

lemma *minus-adelic-int-quot-rep-eq*:
Rep-adelic-int-quot $(x - y) =$ *Rep-adelic-int-quot* $x -$ *Rep-adelic-int-quot* y
⟨*proof*⟩

lemma *minus-adelic-int-quot-abs-eq*:
Abs-adelic-int-quot $x -$ *Abs-adelic-int-quot* $y =$ *Abs-adelic-int-quot* $(x - y)$
⟨*proof*⟩

lemma *minus-adelic-int-quot*: *adelic-int-quot* $x -$ *adelic-int-quot* $y =$ *adelic-int-quot* $(x - y)$
⟨*proof*⟩

definition $x * y =$ *Abs-adelic-int-quot* (*Rep-adelic-int-quot* $x *$ *Rep-adelic-int-quot* y)

lemma *times-adelic-int-quot-rep-eq*:
Rep-adelic-int-quot $(x * y) =$ *Rep-adelic-int-quot* $x *$ *Rep-adelic-int-quot* y
⟨*proof*⟩

lemma *times-adelic-int-quot-abs-eq*:
Abs-adelic-int-quot $x *$ *Abs-adelic-int-quot* $y =$ *Abs-adelic-int-quot* $(x * y)$
⟨*proof*⟩

lemma *times-adelic-int-quot*: *adelic-int-quot* $x *$ *adelic-int-quot* $y =$ *adelic-int-quot* $(x * y)$
⟨*proof*⟩

instance
⟨*proof*⟩

end

instantiation *adelic-int-quot* ::
(*nontrivial-factorial-unique-euclidean-bezout*) {*divide-trivial*, *inverse*}

begin

lift-definition *inverse-adelic-int-coset* ::
 'a adelic-int coset-by-dist-sbgrp ⇒ *'a adelic-int coset-by-dist-sbgrp*
 is *inverse*
 ⟨*proof*⟩

definition *inverse-adelic-int-quot* :: *'a adelic-int-quot* ⇒ *'a adelic-int-quot*
 where
  inverse-adelic-int-quot x ≡
   Abs-adelic-int-quot (*inverse-adelic-int-coset* (*Rep-adelic-int-quot x*))

lemma *inverse-adelic-int-quot*: *inverse* (*adelic-int-quot x*) = *adelic-int-quot* (*inverse x*)
 ⟨*proof*⟩

lemma *inverse-adelic-int-quot-0*[*simp*]: *inverse* (*0* :: *'a adelic-int-quot*) = *0*
 ⟨*proof*⟩

lemma *inverse-adelic-int-quot-1*[*simp*]: *inverse* (*1* :: *'a adelic-int-quot*) = *1*
 ⟨*proof*⟩

definition *divide-adelic-int-quot* ::
 'a adelic-int-quot ⇒ *'a adelic-int-quot* ⇒ *'a adelic-int-quot*
 where *divide-adelic-int-quot-def*[*simp*]: *divide-adelic-int-quot x y* ≡ *x* ∗ *inverse y*

instance ⟨*proof*⟩

end

lemma *divide-adelic-int-quot*: *adelic-int-quot x* / *adelic-int-quot y* = *adelic-int-quot* (*x* / *y*)
 for *x y* :: *'a::nontrivial-factorial-unique-euclidean-bezout adelic-int*
 ⟨*proof*⟩

6.2 Equivalence relative to a prime

6.2.1 Equivalence

overloading
 p-equal-adelic-int-coset ≡
  p-equal :: *'a::nontrivial-factorial-idom prime* ⇒
   'a adelic-int coset-by-dist-sbgrp ⇒
   'a adelic-int coset-by-dist-sbgrp ⇒ *bool*
 p-equal-adelic-int-quot ≡
  p-equal :: *'a::nontrivial-factorial-idom prime* ⇒
   'a adelic-int-quot ⇒ *'a adelic-int-quot* ⇒ *bool*
 p-restrict-adelic-int-coset ≡
  p-restrict :: *'a adelic-int coset-by-dist-sbgrp* ⇒
   (*'a prime* ⇒ *bool*) ⇒ *'a adelic-int coset-by-dist-sbgrp*

p-restrict-adelic-int-quot ≡
p-restrict :: *'a adelic-int-quot* ⇒
(*'a prime* ⇒ *bool*) ⇒ *'a adelic-int-quot*
begin

lift-definition *p-equal-adelic-int-coset* ::
'a::nontrivial-factorial-idom prime ⇒
'a adelic-int coset-by-dist-sbgrp ⇒
'a adelic-int coset-by-dist-sbgrp ⇒ *bool*
is $\lambda p\ x\ y.\ x \simeq_p y \vee ((x - y)^{\circ p}) \geq 1$
⟨*proof*⟩

definition *p-equal-adelic-int-quot* ::
'a::nontrivial-factorial-idom prime ⇒
'a adelic-int-quot ⇒ *'a adelic-int-quot* ⇒ *bool*
where
p-equal-adelic-int-quot p x y ≡
(*Rep-adelic-int-quot x*) $\simeq_p$ (*Rep-adelic-int-quot y*)

lift-definition *p-restrict-adelic-int-coset* ::
'a::nontrivial-factorial-idom adelic-int coset-by-dist-sbgrp ⇒
(*'a prime* ⇒ *bool*) ⇒ *'a adelic-int coset-by-dist-sbgrp*
is $\lambda x\ P.\ x$ *prestrict P*
⟨*proof*⟩

definition *p-restrict-adelic-int-quot* ::
'a::nontrivial-factorial-idom adelic-int-quot ⇒
(*'a prime* ⇒ *bool*) ⇒ *'a adelic-int-quot*
where
p-restrict-adelic-int-quot x P ≡
Abs-adelic-int-quot ((*Rep-adelic-int-quot x*) *prestrict P*)

end

context
fixes *p* :: *'a::nontrivial-factorial-idom prime*
begin

lemma *p-equal-adelic-int-coset-abs-eq0*:
distinguished-quotient-coset $x \simeq_p 0 \longleftrightarrow$
$x \simeq_p 0 \vee (x^{\circ p}) \geq 1$
for *x* :: *'a adelic-int*
⟨*proof*⟩

lemma *p-equal-adelic-int-coset-abs-eq1*:
distinguished-quotient-coset $x \simeq_p 1 \longleftrightarrow$
$x \simeq_p 1 \vee ((x - 1)^{\circ p}) \geq 1$
for *x* :: *'a adelic-int*
⟨*proof*⟩

lemma *p-equal-adelic-int-quot-abs-eq*:
$(\textit{adelic-int-quot } x) \simeq_p (\textit{adelic-int-quot } y) \longleftrightarrow$
$x \simeq_p y \vee ((x - y)^{\circ p}) \geq 1$
for $x\ y :: {'a}\ \textit{adelic-int}$
⟨*proof*⟩

lemma *p-equal-adelic-int-quot-abs-eq0*:
$\textit{adelic-int-quot } x \simeq_p 0 \longleftrightarrow$
$x \simeq_p 0 \vee (x^{\circ p}) \geq 1$
for $x :: {'a}\ \textit{adelic-int}$
⟨*proof*⟩

lemma *p-equal-adelic-int-quot-abs-eq1*:
$\textit{adelic-int-quot } x \simeq_p 1 \longleftrightarrow$
$x \simeq_p 1 \vee ((x - 1)^{\circ p}) \geq 1$
for $x :: {'a}\ \textit{adelic-int}$
⟨*proof*⟩

end

lemma *p-restrict-adelic-int-quot-abs-eq*:
$(\textit{adelic-int-quot } x)\ \textit{prestrict } P = \textit{adelic-int-quot } (x\ \ \textit{prestrict } P)$
for $P :: {'a}::\textit{nontrivial-factorial-idom prime} \Rightarrow \textit{bool}$ **and** $x :: {'a}\ \textit{adelic-int}$
⟨*proof*⟩

global-interpretation *p-equality-adelic-int-quot*:
p-equality-no-zero-divisors-1
$\textit{p-equal} :: {'a}::\textit{nontrivial-euclidean-ring-cancel prime} \Rightarrow$
${'a}\ \textit{adelic-int-quot} \Rightarrow {'a}\ \textit{adelic-int-quot} \Rightarrow \textit{bool}$
⟨*proof*⟩

overloading
$\textit{globally-p-equal-adelic-int-quot} \equiv$
$\textit{globally-p-equal} ::$
${'a}::\textit{nontrivial-euclidean-ring-cancel adelic-int-quot} \Rightarrow$
${'a}\ \textit{adelic-int-quot} \Rightarrow \textit{bool}$
begin

definition $\textit{globally-p-equal-adelic-int-quot} ::$
${'a}::\textit{nontrivial-euclidean-ring-cancel adelic-int-quot} \Rightarrow$
${'a}\ \textit{adelic-int-quot} \Rightarrow \textit{bool}$
where *globally-p-equal-adelic-int-quot-def*[*simp*]:
$\textit{globally-p-equal-adelic-int-quot} = \textit{p-equality-adelic-int-quot.globally-p-equal}$

end

6.2.2 Embedding of constants

global-interpretation *global-p-equal-adelic-int-quot*:
  global-p-equal-w-consts
    p-equal :: $'a$::*nontrivial-euclidean-ring-cancel prime* ⇒
      $'a$ *adelic-int-quot* ⇒ $'a$ *adelic-int-quot* ⇒ *bool*
    p-restrict ::
      $'a$ *adelic-int-quot* ⇒ ($'a$ *prime* ⇒ *bool*) ⇒
        $'a$ *adelic-int-quot*
    λf. *adelic-int-quot* (*adelic-int-consts* f)
⟨*proof*⟩

abbreviation *adelic-int-quot-consts* ≡ *global-p-equal-adelic-int-quot.consts*
abbreviation *adelic-int-quot-const* ≡ *global-p-equal-adelic-int-quot.const*

lemma *p-equal-adelic-int-quot-consts-iff*:
  (*adelic-int-quot-consts* f) $\simeq_p$ (*adelic-int-quot-consts* g) ⟷
    (f p) *pmod* p = (g p) *pmod* p
  (**is** *?L* ⟷ *?R*)
⟨*proof*⟩

lemma *p-equal0-adelic-int-quot-consts-iff*:
  (*adelic-int-quot-consts* f) $\simeq_p$ *0* ⟷ (f p) *pmod* p = *0*
  (**is** *?L* ⟷ *?R*)
⟨*proof*⟩

lemma *adelic-int-quot-consts-eq-iff*:
  adelic-int-quot-consts f = *adelic-int-quot-consts* g ⟷
    (∀ p. (f p) *pmod* p = (g p) *pmod* p)
  ⟨*proof*⟩

lemma *adelic-int-quot-consts-eq0-iff*:
  (*adelic-int-quot-consts* f) = *0* ⟷
    (∀ p. (f p) *pmod* p = *0*)
  ⟨*proof*⟩

lemmas
  p-equal-adelic-int-quot-const-iff =
    p-equal-adelic-int-quot-consts-iff[*of* - λp. - λp. -]
and *p-equal0-adelic-int-quot-const-iff* = *p-equal0-adelic-int-quot-consts-iff*[*of* - λp.
-]
and *adelic-int-quot-const-eq-iff* = *adelic-int-quot-consts-eq-iff*[*of* λp. - λp. -]
and *adelic-int-quot-const-eq0-iff* = *adelic-int-quot-consts-eq0-iff*[*of* λp. -]

lemma *adelic-int-quot-UNIV-eq-consts-range*:
  (*UNIV*::$'a$::*nontrivial-euclidean-ring-cancel adelic-int-quot set*) =
    range (*adelic-int-quot-consts*)
⟨*proof*⟩

lemma *adelic-int-quot-constsE*:

fixes x :: $'a$::*nontrivial-euclidean-ring-cancel adelic-int-quot*
obtains f **where** x = *adelic-int-quot-consts f*
⟨*proof*⟩

lemma *adelic-int-quot-reduced-constsE*:
fixes x :: $'a$::*nontrivial-euclidean-ring-cancel adelic-int-quot*
obtains f
where x = *adelic-int-quot-consts f*
and ∀ p::$'a$ *prime*. (f p) *pdiv* p = *0*
⟨*proof*⟩

lemma *adelic-int-quot-prestrict-zero-depth-abs-eq*:
adelic-int-quot (x *prestrict* (λp::$'a$ *prime*. ($x^{\circ p}$) = *0*)) =
adelic-int-quot x
for x :: $'a$::*nontrivial-euclidean-ring-cancel adelic-int*
⟨*proof*⟩

6.2.3 Division and inverses

global-interpretation *p-equal-div-inv-adelic-int-quot*:
global-p-equal-div-inv
p-equal :: $'a$::*nontrivial-factorial-unique-euclidean-bezout prime* ⇒
$'a$ *adelic-int-quot* ⇒ $'a$ *adelic-int-quot* ⇒ *bool*
p-restrict ::
$'a$ *adelic-int-quot* ⇒ ($'a$ *prime* ⇒ *bool*) ⇒
$'a$ *adelic-int-quot*
⟨*proof*⟩

6.3 Special case of integer

context
fixes p :: *int prime*
begin

lemma *inj-on-const-prestrict-single-int-prime*:
inj-on (λk. *adelic-int-quot-const k prestrict* ((=) p)) {*0..Rep-prime p* − *1*}
⟨*proof*⟩

lemma *range-prestrict-single-int-prime*:
range (λx. x *prestrict* ((=) p)) =
(λk. *adelic-int-quot-const k prestrict* ((=) p)) ‘ {*0..Rep-prime p* − *1*}
⟨*proof*⟩

lemma *finite-range-prestrict-single-int-prime*:
finite (*range* (λx::*int adelic-int-quot*. x *prestrict* ((=) p)))
⟨*proof*⟩

lemma *card-range-prestrict-single-int-prime*:
card (*range* (λx::*int adelic-int-quot*. x *prestrict* ((=) p))) = *nat* (*Rep-prime p*)
⟨*proof*⟩

end

end

theory *More-pAdic-Product*

imports *Fin-Field-Product*

begin

7 Compactness of local ring of integers

7.1 Preliminaries

lemma *infinite-vimageE*:
  fixes $f :: 'a \Rightarrow 'b$
  assumes *infinite* (*UNIV* :: *'a set*) **and** *range f* $\subseteq B$ **and** *finite B*
  obtains b **where** $b \in B$ **and** *infinite* (f −‘ $\{b\}$)
⟨*proof*⟩
primrec *subset-subseq* ::
  ($nat \Rightarrow 'a$) $\Rightarrow$ *'a set* $\Rightarrow nat \Rightarrow nat$
  where
    subset-subseq f A 0 = (*LEAST k. f k* $\in A$) |
    subset-subseq f A (*Suc n*) = (*LEAST k. k* > *subset-subseq f A n* $\wedge$ *f k* $\in A$)

lemma
  assumes *infinite* (f −‘ A)
  shows *subset-subseq-strict-mono*: *strict-mono* (*subset-subseq f A*)
    and *subset-subseq-range*   : f (*subset-subseq f A n*) $\in A$
⟨*proof*⟩

7.2 Sequential compactness

7.2.1 Creating a Cauchy subsequence

abbreviation
  p-adic-prod-int-convergent-subseq-seq-step p X g n ≡
    λ*k*. *p-adic-prod-int-quot* (*p-adic-prod-shift-p-depth p* (− *int n*) ((*X* (*g k*) − *X* (*g 0*))))

primrec *p-adic-prod-int-convergent-subseq-seq* ::
  'a::nontrivial-factorial-unique-euclidean-bezout prime $\Rightarrow$
    ($nat \Rightarrow$ *'a p-adic-prod*) $\Rightarrow nat \Rightarrow$ ($nat \Rightarrow nat$)
  where *p-adic-prod-int-convergent-subseq-seq p X 0* = *id* |
  p-adic-prod-int-convergent-subseq-seq p X (*Suc n*) =
    p-adic-prod-int-convergent-subseq-seq p X n ∘
      subset-subseq

(
  p-adic-prod-int-convergent-subseq-seq-step p X (
    p-adic-prod-int-convergent-subseq-seq p X n
) *n*
)
{*SOME z.*
  infinite (
    p-adic-prod-int-convergent-subseq-seq-step p X (
      p-adic-prod-int-convergent-subseq-seq p X n
) *n*
    $-$‘ {*z*}
)
}

abbreviation
p-adic-prod-int-convergent-subseq p X n $\equiv$
p-adic-prod-int-convergent-subseq-seq p X n n

lemma *restricted-X-infinite-quot-vimage*:
$\exists\, z.$ *infinite* (($\lambda k.$ *p-adic-prod-int-quot* (*X k*)) $-$‘ {*z*})
if *fin-p-quot* : *finite* (*range* (λx::′*a adelic-int-quot. x prestrict* ((=) *p*)))
and *range-X* : *range X* $\subseteq \mathcal{O}_{\forall\, p}$
and *restricted-X*: $\forall\, n.$ *X n* = (*X n*) *prestrict* ((=) *p*)
for *p* :: ′*a*::*nontrivial-factorial-idom prime* **and** *X* :: *nat* $\Rightarrow$ ′*a p-adic-prod*
⟨*proof*⟩

context
fixes *p* :: ′*a*::*nontrivial-factorial-unique-euclidean-bezout prime*
and *X* :: *nat* $\Rightarrow$ ′*a p-adic-prod*
and *ss* :: *nat* $\Rightarrow$ (*nat* $\Rightarrow$ *nat*)
and *ss-step* :: *nat* $\Rightarrow$ (*nat* $\Rightarrow$ ′*a adelic-int-quot*)
and *nth-im* :: *nat* $\Rightarrow$ ′*a adelic-int-quot*
and *subss* :: *nat* $\Rightarrow$ (*nat* $\Rightarrow$ *nat*)
defines
ss $\equiv$ *p-adic-prod-int-convergent-subseq-seq p X*
and
ss-step $\equiv$ $\lambda n.$ *p-adic-prod-int-convergent-subseq-seq-step p X* (*ss n*) *n*
and *nth-im* $\equiv$ $\lambda n.$ (*SOME z. infinite* (*ss-step n* $-$‘ {*z*}))
and *subss* $\equiv$ $\lambda n.$ *subset-subseq* (*ss-step n*) {*nth-im n*}
assumes *fin-p-quot* : *finite* (*range* (λx::′*a adelic-int-quot. x prestrict* ((=) *p*)))
and *range-X* : *range X* $\subseteq \mathcal{O}_{\forall\, p}$
and *restricted-X*: $\forall\, n.$ *X n* = (*X n*) *prestrict* ((=) *p*)
begin

lemma *p-adic-prod-int-convergent-subseq-seq-step-infinite-quot-vimage*:
$\exists\, z.$ *infinite* (*ss-step n* $-$‘ {*z*}) (**is** *?A*)
and *p-adic-prod-int-convergent-subseq-seq-step-depth*:
$\forall\, k.$ (*X* (*ss n k*)) $\neg\simeq_p$ (*X* (*ss n 0*)) $\longrightarrow$
((X (*ss n k*) $-$ *X* (*ss n 0*))$^{\circ p}$) $\geq$ *int n*

(**is** *?B*)
⟨*proof*⟩

lemma *p-adic-prod-int-convergent-subset-subseq-strict-mono*:
strict-mono (*subset-subseq* (*ss-step* *n*) {*nth-im* *n*})
⟨*proof*⟩

lemma *p-adic-prod-int-convergent-subseq-strict-mono′*: *strict-mono* (*ss* *n*)
⟨*proof*⟩

lemma *p-adic-prod-int-convergent-subseq-strict-mono*:
strict-mono (*p-adic-prod-int-convergent-subseq* *p* *X*)
⟨*proof*⟩

lemma *p-adic-prod-int-convergent-subseq-Cauchy*:
p-adic-prod-p-cauchy *p* ($X \circ$ *p-adic-prod-int-convergent-subseq* *p* *X*)
⟨*proof*⟩

lemma *p-adic-prod-int-convergent-subseq-limseq*:
$x \in (\lambda z.\ z$ *prestrict* $((=)\ p))$ ‘ $\mathcal{O}_{\forall\, p}$
if *p-adic-prod-p-open-nbhds-limseq* $(X \circ g)\ x$
⟨*proof*⟩

end

7.2.2 Proving sequential compactness

context
fixes p :: ′*a*::*nontrivial-factorial-unique-euclidean-bezout prime*
assumes *fin-p-quot*: *finite* (*range* (λx::′*a adelic-int-quot*. *x prestrict* ((=) *p*)))
begin

lemma *p-adic-prod-local-int-ring-seq-compact′*:
$\exists\, g$::*nat*⇒*nat*. *strict-mono* $g \wedge$ *p-adic-prod-p-cauchy* p $(X \circ g)$
if *range-X*:
range $X \subseteq (\lambda x.\ x$ *prestrict* $((=)\ p))$ ‘ $(\mathcal{O}_{\forall\, p})$
for X :: *nat* ⇒ ′*a p-adic-prod*
⟨*proof*⟩

lemma *p-adic-prod-local-int-ring-seq-compact*:
$\exists\, x \in (\lambda x.\ x$ *prestrict* $((=)\ p))$ ‘ $(\mathcal{O}_{\forall\, p})$.
$\exists\, g$::*nat*⇒*nat*.
strict-mono $g \wedge$ *p-adic-prod-p-open-nbhds-limseq* $(X \circ g)\ x$
if *range-X*:
range $X \subseteq (\lambda x.\ x$ *prestrict* $((=)\ p))$ ‘ $(\mathcal{O}_{\forall\, p})$
for X :: *nat* ⇒ ′*a p-adic-prod*
⟨*proof*⟩

lemma *adelic-int-local-int-ring-seq-abs*:

range $X \subseteq (\lambda x.\ x$ *prestrict* $((=)\ p))$ ‘ $(\mathcal{O}_{\forall\ p})$
if *range-X*: *range* $X \subseteq \mathcal{O}_{\forall\ p}$
and *range-abs-X*:
range $(\lambda k.$ *Abs-adelic-int* $(X\ k)) \subseteq$ *range* $(\lambda x.\ x$ *prestrict* $((=)\ p))$
for $X :: nat \Rightarrow 'a$ *p-adic-prod*
⟨*proof*⟩

lemma *adelic-int-local-int-ring-seq-compact′*:
$\exists\, g{::}nat{\Rightarrow}nat.$ *strict-mono* $g \wedge$ *adelic-int-p-cauchy* $p\ (X \circ g)$
if *range-X*:
range $X \subseteq (\lambda x.\ x$ *prestrict* $((=)\ p))$ ‘ $(\mathcal{O}_{\forall\ p})$
for $X :: nat \Rightarrow 'a$ *adelic-int*
⟨*proof*⟩

lemma *adelic-int-local-int-ring-seq-compact*:
$\exists\, x{\in}$*range* $(\lambda x.\ x$ *prestrict* $((=)\ p))$.
$\exists\, g{::}nat{\Rightarrow}nat.$
strict-mono $g \wedge$ *adelic-int-p-open-nbhds-limseq* $(X \circ g)\ x$
if *range-X*:
range $X \subseteq$ *range* $(\lambda x.\ x$ *prestrict* $((=)\ p))$
for $X :: nat \Rightarrow 'a$ *adelic-int*
⟨*proof*⟩

end

lemma *int-adic-prod-local-int-ring-seq-compact*:
$\exists\, x{\in}(\lambda x.\ x$ *prestrict* $((=)\ p))$ ‘ $(\mathcal{O}_{\forall\ p})$.
$\exists\, g{::}nat{\Rightarrow}nat.$
strict-mono $g \wedge$ *p-adic-prod-p-open-nbhds-limseq* $(X \circ g)\ x$
if *range* $X \subseteq (\lambda x.\ x$ *prestrict* $((=)\ p))$ ‘ $(\mathcal{O}_{\forall\ p})$
for $p :: int$ *prime*
and $X :: nat \Rightarrow int$ *p-adic-prod*
⟨*proof*⟩

lemma *int-adelic-int-local-int-ring-seq-compact*:
$\exists\, x{\in}$*range* $(\lambda x.\ x$ *prestrict* $((=)\ p))$.
$\exists\, g{::}nat{\Rightarrow}nat.$
strict-mono $g \wedge$ *adelic-int-p-open-nbhds-limseq* $(X \circ g)\ x$
if *range* $X \subseteq$ *range* $(\lambda x.\ x$ *prestrict* $((=)\ p))$
for $p :: int$ *prime*
and $X :: nat \Rightarrow int$ *adelic-int*
⟨*proof*⟩

7.3 Finite-open-cover compactness

function *exclusion-sequence* ::
$'a\ set \Rightarrow ('a \Rightarrow 'a\ set) \Rightarrow$
$'a \Rightarrow nat \Rightarrow 'a$
where *exclusion-sequence* $A\ f\ a\ 0 = a$ |

exclusion-sequence A f a $(Suc\ n)$ =
  $(SOME\ x.\ x \in A \wedge x \notin (\bigcup k{\leq}n.\ f\ (exclusion\text{-}sequence\ A\ f\ a\ k)))$
⟨*proof*⟩
termination ⟨*proof*⟩

lemma
  assumes $\neg\ A \subseteq (\bigcup k{\leq}n.\ f\ (exclusion\text{-}sequence\ A\ f\ a\ k))$
  shows *exclusion-sequence-mem*: $exclusion\text{-}sequence\ A\ f\ a\ (Suc\ n) \in A$
    and *exclusion-sequence-excludes*:
      $exclusion\text{-}sequence\ A\ f\ a\ (Suc\ n) \notin (\bigcup k{\leq}n.\ f\ (exclusion\text{-}sequence\ A\ f\ a\ k))$
⟨*proof*⟩

context
  fixes p :: *'a::nontrivial-factorial-unique-euclidean-bezout prime*
  assumes *fin-p-quot*: $finite\ (range\ (\lambda x{::}'a\ adelic\text{-}int\text{-}quot.\ x\ prestrict\ ((=)\ p)))$
begin

lemma *p-adic-prod-local-int-ring-finite-cover*:
  $\exists\, C.\ finite\ C\ \wedge$
    $C \subseteq (\lambda x.\ x\ prestrict\ ((=)\ p))$ ‘ $\mathcal{O}_{\forall\ p}\ \wedge$
    $(\lambda x.\ x\ prestrict\ ((=)\ p))$ ‘ $\mathcal{O}_{\forall\ p} \subseteq$
      $(\bigcup c{\in}C.\ p\text{-}adic\text{-}prod\text{-}p\text{-}open\text{-}nbhd\ p\ n\ c)$
  for C :: *'a set*
⟨*proof*⟩

lemma *p-adic-prod-local-int-ring-lebesgue-number*:
  $\exists\, d.\ \forall\, x{\in}(\lambda x.\ x\ prestrict\ ((=)\ p))$ ‘ $\mathcal{O}_{\forall\ p}.$
    $\exists\, A{\in}\mathcal{A}.\ p\text{-}adic\text{-}prod\text{-}p\text{-}open\text{-}nbhd\ p\ d\ x \subseteq A$
  if *cover*:
    $(\lambda x.\ x\ prestrict\ ((=)\ p))$ ‘ $\mathcal{O}_{\forall\ p} \subseteq \bigcup\ \mathcal{A}$
  and *by-opens*:
    $\forall\, A{\in}\mathcal{A}.\ generate\text{-}topology\ (p\text{-}adic\text{-}prod\text{-}local\text{-}p\text{-}open\text{-}nbhds\ p)\ A$
⟨*proof*⟩

lemma *p-adic-prod-local-int-ring-compact*:
  $\exists \mathcal{B}{\subseteq}\mathcal{A}.\ finite\ \mathcal{B}\ \wedge$
    $(\lambda x.\ x\ prestrict\ ((=)\ p))$ ‘ $\mathcal{O}_{\forall\ p} \subseteq \bigcup\ \mathcal{B}$
  if *cover*:
    $(\lambda x.\ x\ prestrict\ ((=)\ p))$ ‘ $\mathcal{O}_{\forall\ p} \subseteq \bigcup\ \mathcal{A}$
  and *by-opens*:
    $\forall\, A{\in}\mathcal{A}.\ generate\text{-}topology\ (p\text{-}adic\text{-}prod\text{-}local\text{-}p\text{-}open\text{-}nbhds\ p)\ A$
⟨*proof*⟩

lemma *p-adic-prod-local-scaled-int-ring-compact*:
  $\exists \mathcal{B}{\subseteq}\mathcal{A}.\ finite\ \mathcal{B}\ \wedge$
    $(\lambda x.\ x\ prestrict\ ((=)\ p))$ ‘ $(\mathcal{P}_{\forall\ p}{}^{n})$
      $\subseteq \bigcup\ \mathcal{B}$
  if *cover*:
    $(\lambda x.\ x\ prestrict\ ((=)\ p))$ ‘ $(\mathcal{P}_{\forall\ p}{}^{n})$

$\subseteq \bigcup\ \mathcal{A}$
and *by-opens*:
$\forall A{\in}\mathcal{A}.$ *generate-topology* (*p-adic-prod-local-p-open-nbhds* p) A
for $\mathcal{A}$ :: *'a p-adic-prod set set*
⟨*proof*⟩

lemma *adelic-int-local-int-ring-compact*:
$\exists\,\mathcal{B}{\subseteq}\mathcal{A}.$ *finite* $\mathcal{B} \wedge$
range ($\lambda x.$ x *prestrict* ((=) p)) $\subseteq \bigcup\ \mathcal{B}$
if *fin-p-quot*: *finite* (*range* (λx::*'a adelic-int-quot*. x *prestrict* ((=) p)))
and *cover*: *range* ($\lambda x.$ x *prestrict* ((=) p)) $\subseteq \bigcup\ \mathcal{A}$
and *by-opens*:
$\forall A{\in}\mathcal{A}.$ *generate-topology* (*adelic-int-local-p-open-nbhds* p) A
for $\mathcal{A}$ :: *'a adelic-int set set*
⟨*proof*⟩

lemma *adelic-int-local-scaled-int-ring-compact*:
$\exists\,\mathcal{B}{\subseteq}\mathcal{A}.$ *finite* $\mathcal{B} \wedge$
($\lambda x.$ x *prestrict* ((=) p)) ‘ ($\mathcal{P}_{\forall\, p}{}^{n}$)
$\subseteq \bigcup\ \mathcal{B}$
if *fin-p-quot*: *finite* (*range* (λx::*'a adelic-int-quot*. x *prestrict* ((=) p)))
and *cover*:
($\lambda x.$ x *prestrict* ((=) p)) ‘ ($\mathcal{P}_{\forall\, p}{}^{n}$)
$\subseteq \bigcup\ \mathcal{A}$
and *by-opens*:
$\forall A{\in}\mathcal{A}.$ *generate-topology* (*adelic-int-local-p-open-nbhds* p) A
for $\mathcal{A}$ :: *'a adelic-int set set*
⟨*proof*⟩

end

lemma *int-adic-prod-local-int-ring-compact*:
$\exists\,\mathcal{B}{\subseteq}\mathcal{A}.$ *finite* $\mathcal{B} \wedge$
($\lambda x.$ x *prestrict* ((=) p)) ‘ $\mathcal{O}_{\forall\, p} \subseteq \bigcup\ \mathcal{B}$
if ($\lambda x.$ x *prestrict* ((=) p)) ‘ $\mathcal{O}_{\forall\, p} \subseteq \bigcup\ \mathcal{A}$
and $\forall A{\in}\mathcal{A}.$ *generate-topology* (*p-adic-prod-local-p-open-nbhds* p) A
for p :: *int prime*
and $\mathcal{A}$ :: *int p-adic-prod set set*
⟨*proof*⟩

lemma *int-adic-prod-local-scaled-int-ring-compact*:
$\exists\,\mathcal{B}{\subseteq}\mathcal{A}.$ *finite* $\mathcal{B} \wedge$
($\lambda x.$ x *prestrict* ((=) p)) ‘ ($\mathcal{P}_{\forall\, p}{}^{n}$)
$\subseteq \bigcup\ \mathcal{B}$
if
($\lambda x.$ x *prestrict* ((=) p)) ‘ ($\mathcal{P}_{\forall\, p}{}^{n}$)
$\subseteq \bigcup\ \mathcal{A}$
and $\forall A{\in}\mathcal{A}.$ *generate-topology* (*p-adic-prod-local-p-open-nbhds* p) A
for p :: *int prime*

and $\mathcal{A}$ :: *int p-adic-prod set set*
⟨*proof*⟩

lemma *int-adelic-int-local-int-ring-compact*:
$\exists\,\mathcal{B}\subseteq\mathcal{A}$. *finite* $\mathcal{B}$ $\wedge$
range (λx. *x prestrict* ((=) *p*)) $\subseteq \bigcup$ $\mathcal{B}$
if *range* (λx. *x prestrict* ((=) *p*)) $\subseteq \bigcup$ $\mathcal{A}$
and $\forall\, A\in\mathcal{A}$. *generate-topology* (*adelic-int-local-p-open-nbhds p*) *A*
for *p* :: *int prime*
and $\mathcal{A}$ :: *int adelic-int set set*
⟨*proof*⟩

lemma *int-adelic-int-local-scaled-int-ring-compact*:
$\exists\,\mathcal{B}\subseteq\mathcal{A}$. *finite* $\mathcal{B}$ $\wedge$
(λx. *x prestrict* ((=) *p*)) ‘ ($\mathcal{P}_{\forall\, p}{}^{n}$)
$\subseteq \bigcup$ $\mathcal{B}$
if
(λx. *x prestrict* ((=) *p*)) ‘ ($\mathcal{P}_{\forall\, p}{}^{n}$)
$\subseteq \bigcup$ $\mathcal{A}$
and $\forall\, A\in\mathcal{A}$. *generate-topology* (*adelic-int-local-p-open-nbhds p*) *A*
for *p* :: *int prime*
and $\mathcal{A}$ :: *int adelic-int set set*
⟨*proof*⟩

end

References

[1] J. W. S. Cassels. *Local Fields*, volume 3 of *London Mathematical Society Student Texts*. Cambridge University Press, 1986.

[2] D. S. Dummit and R. M. Foote. *Abstract Algebra*. Wiley, 3rd edition, 2004.

[3] J. R. Munkres. *Topology*. Prentice Hall, 2nd edition, 1975.