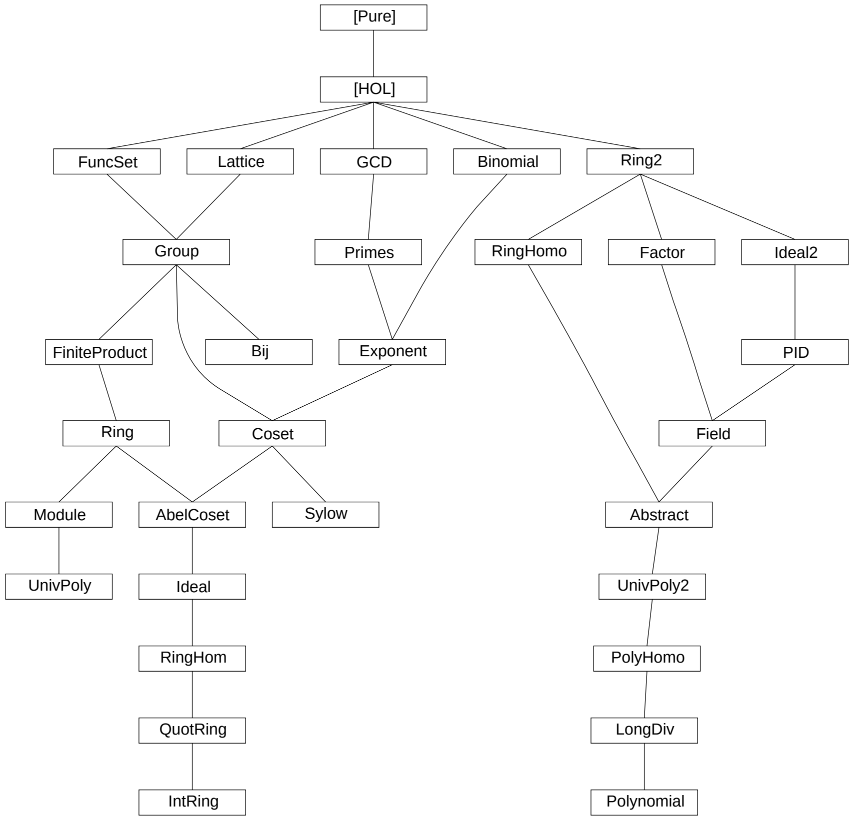
The Isabelle/HOL Algebra Library

Clemens Ballarin
Florian Kammüller
Lawrence C Paulson

November 22, 2007

Contents


[Pure]
[HOL]
FuncSet
Lattice
GCD
Binomial
Ring2
Group
Primes
RingHomo
Factor
Ideal2
FiniteProduct
Bij
Exponent
PID
Ring
Coset
Field
Module
AbelCoset
Sylow
Abstract
UnivPoly
Ideal
UnivPoly2
RingHom
PolyHomo
QuotRing
LongDiv
IntRing
Polynomial

theory *Lattice* **imports** *Main* **begin**

1 Orders and Lattices

Object with a carrier set.

record *'a partial-object* =
  carrier :: *'a set*

1.1 Partial Orders

record *'a order* = *'a partial-object* +
  le :: [*'a*, *'a*] => *bool* (**infixl** $\sqsubseteq_1$ *50*)

locale *partial-order* =
  fixes *L* (**structure**)
  assumes *refl* [*intro*, *simp*]:
    $x \in carrier\ L ==> x \sqsubseteq x$
   and *anti-sym* [*intro*]:
    $[|\ x \sqsubseteq y;\ y \sqsubseteq x;\ x \in carrier\ L;\ y \in carrier\ L\ |] ==> x = y$
   and *trans* [*trans*]:
    $[|\ x \sqsubseteq y;\ y \sqsubseteq z;$
    $x \in carrier\ L;\ y \in carrier\ L;\ z \in carrier\ L\ |] ==> x \sqsubseteq z$

constdefs (**structure** *L*)
  lless :: [-, *'a*, *'a*] => *bool* (**infixl** $\sqsubset_1$ *50*)
  $x \sqsubset y == x \sqsubseteq y\ \&\ x\ {}^{\sim}\!= y$

  — Upper and lower bounds of a set.
  Upper :: [-, *'a set*] => *'a set*
  $Upper\ L\ A == \{u.\ (ALL\ x.\ x \in A \cap carrier\ L\ --> x \sqsubseteq u)\} \cap$
    $carrier\ L$

  Lower :: [-, *'a set*] => *'a set*
  $Lower\ L\ A == \{l.\ (ALL\ x.\ x \in A \cap carrier\ L\ --> l \sqsubseteq x)\} \cap$
    $carrier\ L$

  — Least and greatest, as predicate.
  least :: [-, *'a*, *'a set*] => *bool*
  $least\ L\ l\ A == A \subseteq carrier\ L\ \&\ l \in A\ \&\ (ALL\ x : A.\ l \sqsubseteq x)$

  greatest :: [-, *'a*, *'a set*] => *bool*
  $greatest\ L\ g\ A == A \subseteq carrier\ L\ \&\ g \in A\ \&\ (ALL\ x : A.\ x \sqsubseteq g)$

  — Supremum and infimum
  sup :: [-, *'a set*] => *'a* ($\bigsqcup_1$- [*90*] *90*)
  $\bigsqcup A == THE\ x.\ least\ L\ x\ (Upper\ L\ A)$

inf :: [-, *'a set*] => *'a* ($\bigsqcap$1- [*90*] *90*)
$\bigsqcap$*A* == *THE x. greatest L x* (*Lower L A*)

join :: [-, *'a*, *'a*] => *'a* (**infixl** $\sqcup$1 *65*)
x $\sqcup$ *y* == *sup L* {*x*, *y*}

meet :: [-, *'a*, *'a*] => *'a* (**infixl** $\sqcap$1 *70*)
x $\sqcap$ *y* == *inf L* {*x*, *y*}

1.1.1 Upper

lemma *Upper-closed* [*intro*, *simp*]:
Upper L A $\subseteq$ *carrier L*
⟨*proof*⟩

lemma *UpperD* [*dest*]:
fixes *L* (**structure**)
shows [| *u* $\in$ *Upper L A*; *x* $\in$ *A*; *A* $\subseteq$ *carrier L* |] ==> *x* $\sqsubseteq$ *u*
⟨*proof*⟩

lemma *Upper-memI*:
fixes *L* (**structure**)
shows [| !! *y*. *y* $\in$ *A* ==> *y* $\sqsubseteq$ *x*; *x* $\in$ *carrier L* |] ==> *x* $\in$ *Upper L A*
⟨*proof*⟩

lemma *Upper-antimono*:
A $\subseteq$ *B* ==> *Upper L B* $\subseteq$ *Upper L A*
⟨*proof*⟩

1.1.2 Lower

lemma *Lower-closed* [*intro*, *simp*]:
Lower L A $\subseteq$ *carrier L*
⟨*proof*⟩

lemma *LowerD* [*dest*]:
fixes *L* (**structure**)
shows [| *l* $\in$ *Lower L A*; *x* $\in$ *A*; *A* $\subseteq$ *carrier L* |] ==> *l* $\sqsubseteq$ *x*
⟨*proof*⟩

lemma *Lower-memI*:
fixes *L* (**structure**)
shows [| !! *y*. *y* $\in$ *A* ==> *x* $\sqsubseteq$ *y*; *x* $\in$ *carrier L* |] ==> *x* $\in$ *Lower L A*
⟨*proof*⟩

lemma *Lower-antimono*:
A $\subseteq$ *B* ==> *Lower L B* $\subseteq$ *Lower L A*
⟨*proof*⟩

1.1.3 least

lemma *least-carrier* [*intro*, *simp*]:
  shows *least L l A* ==> $l \in$ *carrier L*
  ⟨*proof*⟩

lemma *least-mem*:
  least L l A ==> $l \in A$
  ⟨*proof*⟩

lemma (**in** *partial-order*) *least-unique*:
  [| *least L x A*; *least L y A* |] ==> $x = y$
  ⟨*proof*⟩

lemma *least-le*:
  fixes *L* (**structure**)
  shows [| *least L x A*; $a \in A$ |] ==> $x \sqsubseteq a$
  ⟨*proof*⟩

lemma *least-UpperI*:
  fixes *L* (**structure**)
  assumes *above*: !! *x*. $x \in A$ ==> $x \sqsubseteq s$
    and *below*: !! *y*. $y \in$ *Upper L A* ==> $s \sqsubseteq y$
    and *L*: $A \subseteq$ *carrier L* $s \in$ *carrier L*
  shows *least L s* (*Upper L A*)
⟨*proof*⟩

1.1.4 greatest

lemma *greatest-carrier* [*intro*, *simp*]:
  shows *greatest L l A* ==> $l \in$ *carrier L*
  ⟨*proof*⟩

lemma *greatest-mem*:
  greatest L l A ==> $l \in A$
  ⟨*proof*⟩

lemma (**in** *partial-order*) *greatest-unique*:
  [| *greatest L x A*; *greatest L y A* |] ==> $x = y$
  ⟨*proof*⟩

lemma *greatest-le*:
  fixes *L* (**structure**)
  shows [| *greatest L x A*; $a \in A$ |] ==> $a \sqsubseteq x$
  ⟨*proof*⟩

lemma *greatest-LowerI*:
  fixes *L* (**structure**)
  assumes *below*: !! *x*. $x \in A$ ==> $i \sqsubseteq x$
    and *above*: !! *y*. $y \in$ *Lower L A* ==> $y \sqsubseteq i$

and L: $A \subseteq$ *carrier* L $i \in$ *carrier* L
shows *greatest* L i (*Lower* L A)
⟨*proof*⟩

1.2 Lattices

locale *lattice* = *partial-order* +
assumes *sup-of-two-exists*:
[| $x \in$ *carrier* L; $y \in$ *carrier* L |] ==> *EX* s. *least* L s (*Upper* L $\{x, y\}$)
and *inf-of-two-exists*:
[| $x \in$ *carrier* L; $y \in$ *carrier* L |] ==> *EX* s. *greatest* L s (*Lower* L $\{x, y\}$)

lemma *least-Upper-above*:
fixes L (**structure**)
shows [| *least* L s (*Upper* L A); $x \in A$; $A \subseteq$ *carrier* L |] ==> $x \sqsubseteq s$
⟨*proof*⟩

lemma *greatest-Lower-above*:
fixes L (**structure**)
shows [| *greatest* L i (*Lower* L A); $x \in A$; $A \subseteq$ *carrier* L |] ==> $i \sqsubseteq x$
⟨*proof*⟩

1.2.1 Supremum

lemma (**in** *lattice*) *joinI*:
[| !!l. *least* L l (*Upper* L $\{x, y\}$) ==> P l; $x \in$ *carrier* L; $y \in$ *carrier* L |]
==> P $(x \sqcup y)$
⟨*proof*⟩

lemma (**in** *lattice*) *join-closed* [*simp*]:
[| $x \in$ *carrier* L; $y \in$ *carrier* L |] ==> $x \sqcup y \in$ *carrier* L
⟨*proof*⟩

lemma (**in** *partial-order*) *sup-of-singletonI*:
$x \in$ *carrier* L ==> *least* L x (*Upper* L $\{x\}$)
⟨*proof*⟩

lemma (**in** *partial-order*) *sup-of-singleton* [*simp*]:
$x \in$ *carrier* L ==> $\bigsqcup\{x\} = x$
⟨*proof*⟩

Condition on A: supremum exists.

lemma (**in** *lattice*) *sup-insertI*:
[| !!s. *least* L s (*Upper* L (*insert* x A)) ==> P s;
least L a (*Upper* L A); $x \in$ *carrier* L; $A \subseteq$ *carrier* L |]
==> P ($\bigsqcup$(*insert* x A))
⟨*proof*⟩

lemma (**in** *lattice*) *finite-sup-least*:

$[|\ \mathit{finite}\ A;\ A \subseteq \mathit{carrier}\ L;\ A\ \tilde{}{=}\ \{\}\ |] ==> \mathit{least}\ L\ (\bigsqcup A)\ (\mathit{Upper}\ L\ A)$
⟨*proof*⟩

lemma (**in** *lattice*) *finite-sup-insertI*:
assumes P: $!!l.\ \mathit{least}\ L\ l\ (\mathit{Upper}\ L\ (\mathit{insert}\ x\ A)) ==> P\ l$
and xA: $\mathit{finite}\ A\ \ x \in \mathit{carrier}\ L\ \ A \subseteq \mathit{carrier}\ L$
shows $P\ (\bigsqcup\ (\mathit{insert}\ x\ A))$
⟨*proof*⟩

lemma (**in** *lattice*) *finite-sup-closed*:
$[|\ \mathit{finite}\ A;\ A \subseteq \mathit{carrier}\ L;\ A\ \tilde{}{=}\ \{\}\ |] ==> \bigsqcup A \in \mathit{carrier}\ L$
⟨*proof*⟩

lemma (**in** *lattice*) *join-left*:
$[|\ x \in \mathit{carrier}\ L;\ y \in \mathit{carrier}\ L\ |] ==> x \sqsubseteq x \sqcup y$
⟨*proof*⟩

lemma (**in** *lattice*) *join-right*:
$[|\ x \in \mathit{carrier}\ L;\ y \in \mathit{carrier}\ L\ |] ==> y \sqsubseteq x \sqcup y$
⟨*proof*⟩

lemma (**in** *lattice*) *sup-of-two-least*:
$[|\ x \in \mathit{carrier}\ L;\ y \in \mathit{carrier}\ L\ |] ==> \mathit{least}\ L\ (\bigsqcup \{x,\ y\})\ (\mathit{Upper}\ L\ \{x,\ y\})$
⟨*proof*⟩

lemma (**in** *lattice*) *join-le*:
assumes *sub*: $x \sqsubseteq z\ \ y \sqsubseteq z$
and x: $x \in \mathit{carrier}\ L$ **and** y: $y \in \mathit{carrier}\ L$ **and** z: $z \in \mathit{carrier}\ L$
shows $x \sqcup y \sqsubseteq z$
⟨*proof*⟩

lemma (**in** *lattice*) *join-assoc-lemma*:
assumes L: $x \in \mathit{carrier}\ L\ \ y \in \mathit{carrier}\ L\ \ z \in \mathit{carrier}\ L$
shows $x \sqcup (y \sqcup z) = \bigsqcup \{x,\ y,\ z\}$
⟨*proof*⟩

lemma *join-comm*:
fixes L (**structure**)
shows $x \sqcup y = y \sqcup x$
⟨*proof*⟩

lemma (**in** *lattice*) *join-assoc*:
assumes L: $x \in \mathit{carrier}\ L\ \ y \in \mathit{carrier}\ L\ \ z \in \mathit{carrier}\ L$
shows $(x \sqcup y) \sqcup z = x \sqcup (y \sqcup z)$
⟨*proof*⟩

1.2.2 Infimum

lemma (**in** *lattice*) *meetI*:

[| !!i. *greatest L i* (*Lower L* $\{x, y\}$) ==> *P i*;
$x \in$ *carrier L*; $y \in$ *carrier L* |]
==> *P* ($x \sqcap y$)
⟨*proof*⟩

lemma (**in** *lattice*) *meet-closed* [*simp*]:
[| $x \in$ *carrier L*; $y \in$ *carrier L* |] ==> $x \sqcap y \in$ *carrier L*
⟨*proof*⟩

lemma (**in** *partial-order*) *inf-of-singletonI*:
$x \in$ *carrier L* ==> *greatest L x* (*Lower L* $\{x\}$)
⟨*proof*⟩

lemma (**in** *partial-order*) *inf-of-singleton* [*simp*]:
$x \in$ *carrier L* ==> $\bigsqcap$ $\{x\} = x$
⟨*proof*⟩

Condition on A: infimum exists.

lemma (**in** *lattice*) *inf-insertI*:
[| !!i. *greatest L i* (*Lower L* (*insert x A*)) ==> *P i*;
greatest L a (*Lower L A*); $x \in$ *carrier L*; $A \subseteq$ *carrier L* |]
==> *P* ($\bigsqcap$(*insert x A*))
⟨*proof*⟩

lemma (**in** *lattice*) *finite-inf-greatest*:
[| *finite A*; $A \subseteq$ *carrier L*; A ~= {} |] ==> *greatest L* ($\bigsqcap A$) (*Lower L A*)
⟨*proof*⟩

lemma (**in** *lattice*) *finite-inf-insertI*:
assumes *P*: !!i. *greatest L i* (*Lower L* (*insert x A*)) ==> *P i*
and *xA*: *finite A* $x \in$ *carrier L* $A \subseteq$ *carrier L*
shows *P* ($\bigsqcap$ (*insert x A*))
⟨*proof*⟩

lemma (**in** *lattice*) *finite-inf-closed*:
[| *finite A*; $A \subseteq$ *carrier L*; A ~= {} |] ==> $\bigsqcap A \in$ *carrier L*
⟨*proof*⟩

lemma (**in** *lattice*) *meet-left*:
[| $x \in$ *carrier L*; $y \in$ *carrier L* |] ==> $x \sqcap y \sqsubseteq x$
⟨*proof*⟩

lemma (**in** *lattice*) *meet-right*:
[| $x \in$ *carrier L*; $y \in$ *carrier L* |] ==> $x \sqcap y \sqsubseteq y$
⟨*proof*⟩

lemma (**in** *lattice*) *inf-of-two-greatest*:
[| $x \in$ *carrier L*; $y \in$ *carrier L* |] ==>
greatest L ($\bigsqcap$ $\{x, y\}$) (*Lower L* $\{x, y\}$)

$\langle proof \rangle$

lemma (**in** *lattice*) *meet-le*:
assumes *sub*: $z \sqsubseteq x \quad z \sqsubseteq y$
and *x*: $x \in carrier\ L$ **and** *y*: $y \in carrier\ L$ **and** *z*: $z \in carrier\ L$
shows $z \sqsubseteq x \sqcap y$
$\langle proof \rangle$

lemma (**in** *lattice*) *meet-assoc-lemma*:
assumes *L*: $x \in carrier\ L \quad y \in carrier\ L \quad z \in carrier\ L$
shows $x \sqcap (y \sqcap z) = \bigsqcap\{x, y, z\}$
$\langle proof \rangle$

lemma *meet-comm*:
fixes *L* (**structure**)
shows $x \sqcap y = y \sqcap x$
$\langle proof \rangle$

lemma (**in** *lattice*) *meet-assoc*:
assumes *L*: $x \in carrier\ L \quad y \in carrier\ L \quad z \in carrier\ L$
shows $(x \sqcap y) \sqcap z = x \sqcap (y \sqcap z)$
$\langle proof \rangle$

1.3 Total Orders

locale *total-order* = *partial-order* +
assumes *total*: $[| \ x \in carrier\ L;\ y \in carrier\ L \ |] ==> x \sqsubseteq y \mid y \sqsubseteq x$

Introduction rule: the usual definition of total order

lemma (**in** *partial-order*) *total-orderI*:
assumes *total*: $!!x\ y.\ [| \ x \in carrier\ L;\ y \in carrier\ L \ |] ==> x \sqsubseteq y \mid y \sqsubseteq x$
shows *total-order L*
$\langle proof \rangle$

Total orders are lattices.

interpretation *total-order* < *lattice*
$\langle proof \rangle$

1.4 Complete lattices

locale *complete-lattice* = *lattice* +
assumes *sup-exists*:
$[| \ A \subseteq carrier\ L \ |] ==> EX\ s.\ least\ L\ s\ (Upper\ L\ A)$
and *inf-exists*:
$[| \ A \subseteq carrier\ L \ |] ==> EX\ i.\ greatest\ L\ i\ (Lower\ L\ A)$

Introduction rule: the usual definition of complete lattice

lemma (**in** *partial-order*) *complete-latticeI*:
assumes *sup-exists*:

!!*A*. [| *A* ⊆ *carrier L* |] ==> *EX s*. *least L s* (*Upper L A*)
and *inf-exists*:
!!*A*. [| *A* ⊆ *carrier L* |] ==> *EX i*. *greatest L i* (*Lower L A*)
shows *complete-lattice L*
⟨*proof*⟩

constdefs (**structure** *L*)
top :: - => *'a* (⊤1)
⊤ == *sup L* (*carrier L*)

bottom :: - => *'a* (⊥1)
⊥ == *inf L* (*carrier L*)

lemma (**in** *complete-lattice*) *supI*:
[| !!*l*. *least L l* (*Upper L A*) ==> *P l*; *A* ⊆ *carrier L* |]
==> *P* ($\bigsqcup$*A*)
⟨*proof*⟩

lemma (**in** *complete-lattice*) *sup-closed* [*simp*]:
A ⊆ *carrier L* ==> $\bigsqcup$*A* ∈ *carrier L*
⟨*proof*⟩

lemma (**in** *complete-lattice*) *top-closed* [*simp*, *intro*]:
⊤ ∈ *carrier L*
⟨*proof*⟩

lemma (**in** *complete-lattice*) *infI*:
[| !!*i*. *greatest L i* (*Lower L A*) ==> *P i*; *A* ⊆ *carrier L* |]
==> *P* ($\bigsqcap$*A*)
⟨*proof*⟩

lemma (**in** *complete-lattice*) *inf-closed* [*simp*]:
A ⊆ *carrier L* ==> $\bigsqcap$*A* ∈ *carrier L*
⟨*proof*⟩

lemma (**in** *complete-lattice*) *bottom-closed* [*simp*, *intro*]:
⊥ ∈ *carrier L*
⟨*proof*⟩

Jacobson: Theorem 8.1

lemma *Lower-empty* [*simp*]:
Lower L {} = *carrier L*
⟨*proof*⟩

lemma *Upper-empty* [*simp*]:
Upper L {} = *carrier L*
⟨*proof*⟩

theorem (**in** *partial-order*) *complete-lattice-criterion1*:
  assumes *top-exists*: *EX g. greatest L g* (*carrier L*)
    and *inf-exists*:
      $!!A$. [| $A \subseteq$ *carrier L*; A ~= {} |] ==> *EX i. greatest L i* (*Lower L A*)
  shows *complete-lattice L*
⟨*proof*⟩

1.5 Examples

1.5.1 Powerset of a Set is a Complete Lattice

theorem *powerset-is-complete-lattice*:
  complete-lattice (| *carrier* = *Pow A*, *le* = *op* $\subseteq$ |)
  (**is** *complete-lattice ?L*)
⟨*proof*⟩

An other example, that of the lattice of subgroups of a group, can be found in Group theory (Section 2.7).

end

theory *Group* **imports** *FuncSet Lattice* **begin**

2 Monoids and Groups

2.1 Definitions

Definitions follow [2].

record *'a monoid* = *'a partial-object* +
  mult :: [*'a*, *'a*] $\Rightarrow$ *'a* (**infixl** $\otimes_1$ *70*)
  one :: *'a* ($\mathbf{1}_1$)

constdefs (**structure** *G*)
  m-inv :: (*'a*, *'b*) *monoid-scheme* => *'a* => *'a* (*inv*1 - [*81*] *80*)
  inv x == (*THE y*. $y \in$ *carrier G* & $x \otimes y = \mathbf{1}$ & $y \otimes x = \mathbf{1}$)

  Units :: - => *'a set*
  — The set of invertible elements
  Units G == {y. $y \in$ *carrier G* & ($\exists x \in$ *carrier G*. $x \otimes y = \mathbf{1}$ & $y \otimes x = \mathbf{1}$)}

consts
  pow :: [(*'a*, *'m*) *monoid-scheme*, *'a*, *'b::number*] => *'a* (**infixr** '(^')1 *75*)

defs (**overloaded**)
  nat-pow-def: *pow G a n* == *nat-rec* $\mathbf{1}_G$ (%*u b*. $b \otimes_G a$) *n*
  int-pow-def: *pow G a z* ==
    let p = *nat-rec* $\mathbf{1}_G$ (%*u b*. $b \otimes_G a$)

in if neg z then inv_G $(p\ (nat\ (-z)))$ *else* $p\ (nat\ z)$

locale *monoid* =
fixes G (**structure**)
assumes *m-closed* [*intro*, *simp*]:
$[\![x \in carrier\ G;\ y \in carrier\ G]\!] \Longrightarrow x \otimes y \in carrier\ G$
and *m-assoc*:
$[\![x \in carrier\ G;\ y \in carrier\ G;\ z \in carrier\ G]\!]$
$\Longrightarrow (x \otimes y) \otimes z = x \otimes (y \otimes z)$
and *one-closed* [*intro*, *simp*]: $\mathbf{1} \in carrier\ G$
and *l-one* [*simp*]: $x \in carrier\ G \Longrightarrow \mathbf{1} \otimes x = x$
and *r-one* [*simp*]: $x \in carrier\ G \Longrightarrow x \otimes \mathbf{1} = x$

lemma *monoidI*:
fixes G (**structure**)
assumes *m-closed*:
!!$x\ y$. [| $x \in carrier\ G$; $y \in carrier\ G$ |] ==> $x \otimes y \in carrier\ G$
and *one-closed*: $\mathbf{1} \in carrier\ G$
and *m-assoc*:
!!$x\ y\ z$. [| $x \in carrier\ G$; $y \in carrier\ G$; $z \in carrier\ G$ |] ==>
$(x \otimes y) \otimes z = x \otimes (y \otimes z)$
and *l-one*: !!x. $x \in carrier\ G$ ==> $\mathbf{1} \otimes x = x$
and *r-one*: !!x. $x \in carrier\ G$ ==> $x \otimes \mathbf{1} = x$
shows *monoid* G
⟨*proof*⟩

lemma (**in** *monoid*) *Units-closed* [*dest*]:
$x \in Units\ G$ ==> $x \in carrier\ G$
⟨*proof*⟩

lemma (**in** *monoid*) *inv-unique*:
assumes *eq*: $y \otimes x = \mathbf{1}$ $\ x \otimes y' = \mathbf{1}$
and G: $x \in carrier\ G$ $\ y \in carrier\ G$ $\ y' \in carrier\ G$
shows $y = y'$
⟨*proof*⟩

lemma (**in** *monoid*) *Units-one-closed* [*intro*, *simp*]:
$\mathbf{1} \in Units\ G$
⟨*proof*⟩

lemma (**in** *monoid*) *Units-inv-closed* [*intro*, *simp*]:
$x \in Units\ G$ ==> $inv\ x \in carrier\ G$
⟨*proof*⟩

lemma (**in** *monoid*) *Units-l-inv-ex*:
$x \in Units\ G$ ==> $\exists\, y \in carrier\ G.\ y \otimes x = \mathbf{1}$
⟨*proof*⟩

lemma (**in** *monoid*) *Units-r-inv-ex*:

$x \in$ *Units G* ==> $\exists\, y \in$ *carrier G*. $x \otimes y = \mathbf{1}$
⟨*proof*⟩

lemma (**in** *monoid*) *Units-l-inv*:
$x \in$ *Units G* ==> *inv* $x \otimes x = \mathbf{1}$
⟨*proof*⟩

lemma (**in** *monoid*) *Units-r-inv*:
$x \in$ *Units G* ==> $x \otimes$ *inv* $x = \mathbf{1}$
⟨*proof*⟩

lemma (**in** *monoid*) *Units-inv-Units* [*intro*, *simp*]:
$x \in$ *Units G* ==> *inv* $x \in$ *Units G*
⟨*proof*⟩

lemma (**in** *monoid*) *Units-l-cancel* [*simp*]:
[| $x \in$ *Units G*; $y \in$ *carrier G*; $z \in$ *carrier G* |] ==>
$(x \otimes y = x \otimes z) = (y = z)$
⟨*proof*⟩

lemma (**in** *monoid*) *Units-inv-inv* [*simp*]:
$x \in$ *Units G* ==> *inv* (*inv* x) $= x$
⟨*proof*⟩

lemma (**in** *monoid*) *inv-inj-on-Units*:
inj-on (*m-inv G*) (*Units G*)
⟨*proof*⟩

lemma (**in** *monoid*) *Units-inv-comm*:
assumes *inv*: $x \otimes y = \mathbf{1}$
and *G*: $x \in$ *Units G* $y \in$ *Units G*
shows $y \otimes x = \mathbf{1}$
⟨*proof*⟩

Power

lemma (**in** *monoid*) *nat-pow-closed* [*intro*, *simp*]:
$x \in$ *carrier G* ==> x (^) (n::*nat*) $\in$ *carrier G*
⟨*proof*⟩

lemma (**in** *monoid*) *nat-pow-0* [*simp*]:
x (^) (*0*::*nat*) $= \mathbf{1}$
⟨*proof*⟩

lemma (**in** *monoid*) *nat-pow-Suc* [*simp*]:
x (^) (*Suc* n) $= x$ (^) $n \otimes x$
⟨*proof*⟩

lemma (**in** *monoid*) *nat-pow-one* [*simp*]:
$\mathbf{1}$ (^) (n::*nat*) $= \mathbf{1}$

⟨*proof*⟩

lemma (**in** *monoid*) *nat-pow-mult*:
$x \in carrier\ G \Longrightarrow x\ (\hat{}\ )\ (n{::}nat) \otimes x\ (\hat{}\ )\ m = x\ (\hat{}\ )\ (n + m)$
⟨*proof*⟩

lemma (**in** *monoid*) *nat-pow-pow*:
$x \in carrier\ G \Longrightarrow (x\ (\hat{}\ )\ n)\ (\hat{}\ )\ m = x\ (\hat{}\ )\ (n * m{::}nat)$
⟨*proof*⟩

A group is a monoid all of whose elements are invertible.

locale *group* = *monoid* +
assumes *Units*: $carrier\ G <= Units\ G$

lemma (**in** *group*) *is-group*: *group G* ⟨*proof*⟩

theorem *groupI*:
fixes *G* (**structure**)
assumes *m-closed* [*simp*]:
$!!x\ y.\ [|\ x \in carrier\ G;\ y \in carrier\ G\ |] \Longrightarrow x \otimes y \in carrier\ G$
and *one-closed* [*simp*]: $\mathbf{1} \in carrier\ G$
and *m-assoc*:
$!!x\ y\ z.\ [|\ x \in carrier\ G;\ y \in carrier\ G;\ z \in carrier\ G\ |] \Longrightarrow$
$(x \otimes y) \otimes z = x \otimes (y \otimes z)$
and *l-one* [*simp*]: $!!x.\ x \in carrier\ G \Longrightarrow \mathbf{1} \otimes x = x$
and *l-inv-ex*: $!!x.\ x \in carrier\ G \Longrightarrow \exists\, y \in carrier\ G.\ y \otimes x = \mathbf{1}$
shows *group G*
⟨*proof*⟩

lemma (**in** *monoid*) *monoid-groupI*:
assumes *l-inv-ex*:
$!!x.\ x \in carrier\ G \Longrightarrow \exists\, y \in carrier\ G.\ y \otimes x = \mathbf{1}$
shows *group G*
⟨*proof*⟩

lemma (**in** *group*) *Units-eq* [*simp*]:
$Units\ G = carrier\ G$
⟨*proof*⟩

lemma (**in** *group*) *inv-closed* [*intro*, *simp*]:
$x \in carrier\ G \Longrightarrow inv\ x \in carrier\ G$
⟨*proof*⟩

lemma (**in** *group*) *l-inv-ex* [*simp*]:
$x \in carrier\ G \Longrightarrow \exists\, y \in carrier\ G.\ y \otimes x = \mathbf{1}$
⟨*proof*⟩

lemma (**in** *group*) *r-inv-ex* [*simp*]:

$x \in carrier\ G ==> \exists\, y \in carrier\ G.\ x \otimes y = \mathbf{1}$
⟨*proof*⟩

lemma (**in** *group*) *l-inv* [*simp*]:
$x \in carrier\ G ==> inv\ x \otimes x = \mathbf{1}$
⟨*proof*⟩

2.2 Cancellation Laws and Basic Properties

lemma (**in** *group*) *l-cancel* [*simp*]:
$[|\ x \in carrier\ G;\ y \in carrier\ G;\ z \in carrier\ G\ |] ==>$
$(x \otimes y = x \otimes z) = (y = z)$
⟨*proof*⟩

lemma (**in** *group*) *r-inv* [*simp*]:
$x \in carrier\ G ==> x \otimes inv\ x = \mathbf{1}$
⟨*proof*⟩

lemma (**in** *group*) *r-cancel* [*simp*]:
$[|\ x \in carrier\ G;\ y \in carrier\ G;\ z \in carrier\ G\ |] ==>$
$(y \otimes x = z \otimes x) = (y = z)$
⟨*proof*⟩

lemma (**in** *group*) *inv-one* [*simp*]:
$inv\ \mathbf{1} = \mathbf{1}$
⟨*proof*⟩

lemma (**in** *group*) *inv-inv* [*simp*]:
$x \in carrier\ G ==> inv\ (inv\ x) = x$
⟨*proof*⟩

lemma (**in** *group*) *inv-inj*:
inj-on (*m-inv G*) (*carrier G*)
⟨*proof*⟩

lemma (**in** *group*) *inv-mult-group*:
$[|\ x \in carrier\ G;\ y \in carrier\ G\ |] ==> inv\ (x \otimes y) = inv\ y \otimes inv\ x$
⟨*proof*⟩

lemma (**in** *group*) *inv-comm*:
$[|\ x \otimes y = \mathbf{1};\ x \in carrier\ G;\ y \in carrier\ G\ |] ==> y \otimes x = \mathbf{1}$
⟨*proof*⟩

lemma (**in** *group*) *inv-equality*:
$[|y \otimes x = \mathbf{1};\ x \in carrier\ G;\ y \in carrier\ G|] ==> inv\ x = y$
⟨*proof*⟩

Power

lemma (**in** *group*) *int-pow-def2*:

a (^) $(z::int) = (if\ neg\ z\ then\ inv\ (a$ (^) $(nat\ (-z)))\ else\ a$ (^) $(nat\ z))$
⟨*proof*⟩

lemma (**in** *group*) *int-pow-0* [*simp*]:
x (^) $(0::int) = \mathbf{1}$
⟨*proof*⟩

lemma (**in** *group*) *int-pow-one* [*simp*]:
$\mathbf{1}$ (^) $(z::int) = \mathbf{1}$
⟨*proof*⟩

2.3 Subgroups

locale *subgroup* =
fixes *H* **and** *G* (**structure**)
assumes *subset*: $H \subseteq carrier\ G$
and *m-closed* [*intro*, *simp*]: $[\![x \in H; y \in H]\!] \Longrightarrow x \otimes y \in H$
and *one-closed* [*simp*]: $\mathbf{1} \in H$
and *m-inv-closed* [*intro*,*simp*]: $x \in H \Longrightarrow inv\ x \in H$

lemma (**in** *subgroup*) *is-subgroup*:
subgroup H G ⟨*proof*⟩

declare (**in** *subgroup*) *group.intro* [*intro*]

lemma (**in** *subgroup*) *mem-carrier* [*simp*]:
$x \in H \Longrightarrow x \in carrier\ G$
⟨*proof*⟩

lemma *subgroup-imp-subset*:
$subgroup\ H\ G \Longrightarrow H \subseteq carrier\ G$
⟨*proof*⟩

lemma (**in** *subgroup*) *subgroup-is-group* [*intro*]:
includes *group G*
shows $group\ (G(\!|carrier := H|\!))$
⟨*proof*⟩

Since H is nonempty, it contains some element x. Since it is closed under inverse, it contains $inv\ x$. Since it is closed under product, it contains $x \otimes inv\ x = \mathbf{1}$.

lemma (**in** *group*) *one-in-subset*:
$[|\ H \subseteq carrier\ G; H \neq \{\}; \forall a \in H.\ inv\ a \in H; \forall a{\in}H.\ \forall b{\in}H.\ a \otimes b \in H\ |]$
$==> \mathbf{1} \in H$
⟨*proof*⟩

A characterization of subgroups: closed, non-empty subset.

lemma (**in** *group*) *subgroupI*:
assumes *subset*: $H \subseteq carrier\ G$ **and** *non-empty*: $H \neq \{\}$

  and *inv*: !!a. $a \in H \Longrightarrow$ *inv* $a \in H$
  and *mult*: !!a b. $[\![a \in H; b \in H]\!] \Longrightarrow a \otimes b \in H$
 shows *subgroup H G*
⟨*proof*⟩

declare *monoid.one-closed* [*iff*] *group.inv-closed* [*simp*]
 monoid.l-one [*simp*] *monoid.r-one* [*simp*] *group.inv-inv* [*simp*]

lemma *subgroup-nonempty*:
 ~ *subgroup* {} *G*
 ⟨*proof*⟩

lemma (**in** *subgroup*) *finite-imp-card-positive*:
 finite (*carrier G*) ==> *0* < *card H*
⟨*proof*⟩

2.4 Direct Products

constdefs
 DirProd :: - ⇒ - ⇒ ($'a \times 'b$) *monoid* (**infixr** ×× *80*)
 $G \times\times H \equiv (\!|$ *carrier* = *carrier* $G \times$ *carrier* H,
  mult = $(\lambda(g, h)\ (g', h').\ (g \otimes_G g', h \otimes_H h'))$,
  one = $(\mathbf{1}_G, \mathbf{1}_H)|\!)$

lemma *DirProd-monoid*:
 includes *monoid G* + *monoid H*
 shows *monoid* ($G \times\times H$)
⟨*proof*⟩

Does not use the previous result because it's easier just to use auto.

lemma *DirProd-group*:
 includes *group G* + *group H*
 shows *group* ($G \times\times H$)
 ⟨*proof*⟩

lemma *carrier-DirProd* [*simp*]:
  carrier ($G \times\times H$) = *carrier* $G \times$ *carrier* H
 ⟨*proof*⟩

lemma *one-DirProd* [*simp*]:
  $\mathbf{1}_{G \times\times H} = (\mathbf{1}_G, \mathbf{1}_H)$
 ⟨*proof*⟩

lemma *mult-DirProd* [*simp*]:
  $(g, h) \otimes_{(G \times\times H)} (g', h') = (g \otimes_G g', h \otimes_H h')$
 ⟨*proof*⟩

lemma *inv-DirProd* [*simp*]:
 includes *group G* + *group H*

assumes g: $g \in$ *carrier* G
and h: $h \in$ *carrier* H
shows *m-inv* $(G \times\times H)$ $(g, h) = (inv_G\ g,\ inv_H\ h)$
⟨*proof*⟩

This alternative proof of the previous result demonstrates interpret. It uses *Prod.inv-equality* (available after *interpret*) instead of *group.inv-equality* [*OF DirProd-group*].

lemma
includes *group* G + *group* H
assumes g: $g \in$ *carrier* G
and h: $h \in$ *carrier* H
shows *m-inv* $(G \times\times H)$ $(g, h) = (inv_G\ g,\ inv_H\ h)$
⟨*proof*⟩

2.5 Homomorphisms and Isomorphisms

constdefs (**structure** G **and** H)
hom :: - => - => ('*a* => '*b*) *set*
hom G H ==
{h. $h \in$ *carrier* G −> *carrier* H &
($\forall x \in$ *carrier* G. $\forall y \in$ *carrier* G. $h\ (x \otimes_G y) = h\ x \otimes_H h\ y$)}

lemma *hom-mult*:
[| $h \in$ *hom* G H; $x \in$ *carrier* G; $y \in$ *carrier* G |]
==> $h\ (x \otimes_G y) = h\ x \otimes_H h\ y$
⟨*proof*⟩

lemma *hom-closed*:
[| $h \in$ *hom* G H; $x \in$ *carrier* G |] ==> $h\ x \in$ *carrier* H
⟨*proof*⟩

lemma (**in** *group*) *hom-compose*:
[|$h \in$ *hom* G H; $i \in$ *hom* H I|] ==> *compose* (*carrier* G) i $h \in$ *hom* G I
⟨*proof*⟩

constdefs
iso :: - => - => ('*a* => '*b*) *set* (**infixr** ≅ *60*)
$G \cong H$ == {h. $h \in$ *hom* G H & *bij-betw* h (*carrier* G) (*carrier* H)}

lemma *iso-refl*: (%x. x) $\in G \cong G$
⟨*proof*⟩

lemma (**in** *group*) *iso-sym*:
$h \in G \cong H \Longrightarrow$ *Inv* (*carrier* G) $h \in H \cong G$
⟨*proof*⟩

lemma (**in** *group*) *iso-trans*:
[|$h \in G \cong H$; $i \in H \cong I$|] ==> (*compose* (*carrier* G) i h) $\in G \cong I$

⟨*proof*⟩

lemma *DirProd-commute-iso*:
shows $(\lambda(x,y).\ (y,x)) \in (G \times\times H) \cong (H \times\times G)$
⟨*proof*⟩

lemma *DirProd-assoc-iso*:
shows $(\lambda(x,y,z).\ (x,(y,z))) \in (G \times\times H \times\times I) \cong (G \times\times (H \times\times I))$
⟨*proof*⟩

Basis for homomorphism proofs: we assume two groups G and H, with a homomorphism h between them

locale *group-hom* = *group G* + *group H* + *var h* +
assumes *homh*: $h \in$ *hom G H*
notes *hom-mult* [*simp*] = *hom-mult* [*OF homh*]
and *hom-closed* [*simp*] = *hom-closed* [*OF homh*]

lemma (**in** *group-hom*) *one-closed* [*simp*]:
$h\ \mathbf{1} \in$ *carrier H*
⟨*proof*⟩

lemma (**in** *group-hom*) *hom-one* [*simp*]:
$h\ \mathbf{1} = \mathbf{1}_H$
⟨*proof*⟩

lemma (**in** *group-hom*) *inv-closed* [*simp*]:
$x \in$ *carrier G* ==> *h* (*inv x*) $\in$ *carrier H*
⟨*proof*⟩

lemma (**in** *group-hom*) *hom-inv* [*simp*]:
$x \in$ *carrier G* ==> *h* (*inv x*) = inv_H (*h x*)
⟨*proof*⟩

2.6 Commutative Structures

Naming convention: multiplicative structures that are commutative are called *commutative*, additive structures are called *Abelian*.

locale *comm-monoid* = *monoid* +
assumes *m-comm*: $[\![x \in carrier\ G;\ y \in carrier\ G]\!] \Longrightarrow x \otimes y = y \otimes x$

lemma (**in** *comm-monoid*) *m-lcomm*:
$[\![x \in carrier\ G;\ y \in carrier\ G;\ z \in carrier\ G]\!] \Longrightarrow$
$x \otimes (y \otimes z) = y \otimes (x \otimes z)$
⟨*proof*⟩

lemmas (**in** *comm-monoid*) *m-ac* = *m-assoc m-comm m-lcomm*

lemma *comm-monoidI*:

fixes G (**structure**)
assumes *m-closed*:
$!!x\ y.\ [|\ x \in carrier\ G;\ y \in carrier\ G\ |] ==> x \otimes y \in carrier\ G$
and *one-closed*: $\mathbf{1} \in carrier\ G$
and *m-assoc*:
$!!x\ y\ z.\ [|\ x \in carrier\ G;\ y \in carrier\ G;\ z \in carrier\ G\ |] ==>$
$(x \otimes y) \otimes z = x \otimes (y \otimes z)$
and *l-one*: $!!x.\ x \in carrier\ G ==> \mathbf{1} \otimes x = x$
and *m-comm*:
$!!x\ y.\ [|\ x \in carrier\ G;\ y \in carrier\ G\ |] ==> x \otimes y = y \otimes x$
shows *comm-monoid G*
⟨*proof*⟩

lemma (**in** *monoid*) *monoid-comm-monoidI*:
assumes *m-comm*:
$!!x\ y.\ [|\ x \in carrier\ G;\ y \in carrier\ G\ |] ==> x \otimes y = y \otimes x$
shows *comm-monoid G*
⟨*proof*⟩

lemma (**in** *comm-monoid*) *nat-pow-distr*:
$[|\ x \in carrier\ G;\ y \in carrier\ G\ |] ==>$
$(x \otimes y)\ (\hat{})\ (n::nat) = x\ (\hat{})\ n \otimes y\ (\hat{})\ n$
⟨*proof*⟩

locale *comm-group* = *comm-monoid* + *group*

lemma (**in** *group*) *group-comm-groupI*:
assumes *m-comm*: $!!x\ y.\ [|\ x \in carrier\ G;\ y \in carrier\ G\ |] ==>$
$x \otimes y = y \otimes x$
shows *comm-group G*
⟨*proof*⟩

lemma *comm-groupI*:
fixes G (**structure**)
assumes *m-closed*:
$!!x\ y.\ [|\ x \in carrier\ G;\ y \in carrier\ G\ |] ==> x \otimes y \in carrier\ G$
and *one-closed*: $\mathbf{1} \in carrier\ G$
and *m-assoc*:
$!!x\ y\ z.\ [|\ x \in carrier\ G;\ y \in carrier\ G;\ z \in carrier\ G\ |] ==>$
$(x \otimes y) \otimes z = x \otimes (y \otimes z)$
and *m-comm*:
$!!x\ y.\ [|\ x \in carrier\ G;\ y \in carrier\ G\ |] ==> x \otimes y = y \otimes x$
and *l-one*: $!!x.\ x \in carrier\ G ==> \mathbf{1} \otimes x = x$
and *l-inv-ex*: $!!x.\ x \in carrier\ G ==> \exists\, y \in carrier\ G.\ y \otimes x = \mathbf{1}$
shows *comm-group G*
⟨*proof*⟩

lemma (**in** *comm-group*) *inv-mult*:
[| $x \in$ *carrier* G; $y \in$ *carrier* G |] ==> *inv* ($x \otimes y$) = *inv* $x \otimes$ *inv* y
⟨*proof*⟩

2.7 The Lattice of Subgroups of a Group

theorem (**in** *group*) *subgroups-partial-order*:
partial-order (| *carrier* = {H. *subgroup* H G}, *le* = *op* $\subseteq$ |)
⟨*proof*⟩

lemma (**in** *group*) *subgroup-self*:
subgroup (*carrier* G) G
⟨*proof*⟩

lemma (**in** *group*) *subgroup-imp-group*:
subgroup H G ==> *group* (G(| *carrier* := H |))
⟨*proof*⟩

lemma (**in** *group*) *is-monoid* [*intro*, *simp*]:
monoid G
⟨*proof*⟩

lemma (**in** *group*) *subgroup-inv-equality*:
[| *subgroup* H G; $x \in H$ |] ==> *m-inv* (G (| *carrier* := H |)) x = *inv* x
⟨*proof*⟩

theorem (**in** *group*) *subgroups-Inter*:
assumes *subgr*: (!!H. $H \in A$ ==> *subgroup* H G)
and *not-empty*: A ~= {}
shows *subgroup* ($\bigcap A$) G
⟨*proof*⟩

theorem (**in** *group*) *subgroups-complete-lattice*:
complete-lattice (| *carrier* = {H. *subgroup* H G}, *le* = *op* $\subseteq$ |)
(**is** *complete-lattice* *?L*)
⟨*proof*⟩

end

theory *FiniteProduct* **imports** *Group* **begin**

3 Product Operator for Commutative Monoids

3.1 Inductive Definition of a Relation for Products over Sets

Instantiation of locale *LC* of theory *Finite-Set* is not possible, because here we have explicit typing rules like $x \in$ *carrier* G. We introduce an explicit argument for the domain D.

inductive-set
foldSetD :: ['*a set*, '*b* => '*a* => '*a*, '*a*] => ('*b set* * '*a*) *set*
for D :: '*a set* **and** f :: '*b* => '*a* => '*a* **and** e :: '*a*
where
emptyI [*intro*]: $e \in D$ ==> ({}, e) $\in$ *foldSetD D f e*
| *insertI* [*intro*]: [| x ~: A; $f\ x\ y \in D$; (A, y) $\in$ *foldSetD D f e* |] ==>
(*insert x A*, $f\ x\ y$) $\in$ *foldSetD D f e*

inductive-cases *empty-foldSetDE* [*elim*!]: ({}, x) $\in$ *foldSetD D f e*

constdefs
foldD :: ['*a set*, '*b* => '*a* => '*a*, '*a*, '*b set*] => '*a*
foldD D f e A == *THE* x. (A, x) $\in$ *foldSetD D f e*

lemma *foldSetD-closed*:
[| (A, z) $\in$ *foldSetD D f e* ; $e \in D$; !!$x\ y$. [| $x \in A$; $y \in D$ |] ==> $f\ x\ y \in D$
|] ==> $z \in D$
⟨*proof*⟩

lemma *Diff1-foldSetD*:
[| (A − {x}, y) $\in$ *foldSetD D f e*; $x \in A$; $f\ x\ y \in D$ |] ==>
(A, $f\ x\ y$) $\in$ *foldSetD D f e*
⟨*proof*⟩

lemma *foldSetD-imp-finite* [*simp*]: (A, x) $\in$ *foldSetD D f e* ==> *finite A*
⟨*proof*⟩

lemma *finite-imp-foldSetD*:
[| *finite A*; $e \in D$; !!$x\ y$. [| $x \in A$; $y \in D$ |] ==> $f\ x\ y \in D$ |] ==>
EX x. (A, x) $\in$ *foldSetD D f e*
⟨*proof*⟩

3.2 Left-Commutative Operations

locale *LCD* =
fixes B :: '*b set*
and D :: '*a set*
and f :: '*b* => '*a* => '*a* (**infixl** · *70*)
assumes *left-commute*:
[| $x \in B$; $y \in B$; $z \in D$ |] ==> $x \cdot (y \cdot z) = y \cdot (x \cdot z)$
and *f-closed* [*simp*, *intro*!]: !!$x\ y$. [| $x \in B$; $y \in D$ |] ==> $f\ x\ y \in D$

lemma (**in** *LCD*) *foldSetD-closed* [*dest*]:
(A, z) $\in$ *foldSetD D f e* ==> $z \in D$
⟨*proof*⟩

lemma (**in** *LCD*) *Diff1-foldSetD*:
[| (A − {x}, y) $\in$ *foldSetD D f e*; $x \in A$; $A \subseteq B$ |] ==>

$(A, f\ x\ y) \in foldSetD\ D\ f\ e$
⟨*proof*⟩

lemma (**in** *LCD*) *foldSetD-imp-finite* [*simp*]:
$(A, x) \in foldSetD\ D\ f\ e$ ==> *finite* A
⟨*proof*⟩

lemma (**in** *LCD*) *finite-imp-foldSetD*:
[| *finite* A; $A \subseteq B$; $e \in D$ |] ==> *EX* x. $(A, x) \in foldSetD\ D\ f\ e$
⟨*proof*⟩

lemma (**in** *LCD*) *foldSetD-determ-aux*:
$e \in D$ ==> $\forall A\ x.\ A \subseteq B$ & *card* $A < n$ --> $(A, x) \in foldSetD\ D\ f\ e$ -->
($\forall y.\ (A, y) \in foldSetD\ D\ f\ e$ --> $y = x$)
⟨*proof*⟩

lemma (**in** *LCD*) *foldSetD-determ*:
[| $(A, x) \in foldSetD\ D\ f\ e$; $(A, y) \in foldSetD\ D\ f\ e$; $e \in D$; $A \subseteq B$ |]
==> $y = x$
⟨*proof*⟩

lemma (**in** *LCD*) *foldD-equality*:
[| $(A, y) \in foldSetD\ D\ f\ e$; $e \in D$; $A \subseteq B$ |] ==> *foldD* $D\ f\ e\ A = y$
⟨*proof*⟩

lemma *foldD-empty* [*simp*]:
$e \in D$ ==> *foldD* $D\ f\ e\ \{\} = e$
⟨*proof*⟩

lemma (**in** *LCD*) *foldD-insert-aux*:
[| x ~: A; $x \in B$; $e \in D$; $A \subseteq B$ |] ==>
((*insert* $x\ A$, v) $\in foldSetD\ D\ f\ e$) =
(*EX* y. $(A, y) \in foldSetD\ D\ f\ e$ & $v = f\ x\ y$)
⟨*proof*⟩

lemma (**in** *LCD*) *foldD-insert*:
[| *finite* A; x ~: A; $x \in B$; $e \in D$; $A \subseteq B$ |] ==>
foldD $D\ f\ e$ (*insert* $x\ A$) = $f\ x$ (*foldD* $D\ f\ e\ A$)
⟨*proof*⟩

lemma (**in** *LCD*) *foldD-closed* [*simp*]:
[| *finite* A; $e \in D$; $A \subseteq B$ |] ==> *foldD* $D\ f\ e\ A \in D$
⟨*proof*⟩

lemma (**in** *LCD*) *foldD-commute*:
[| *finite* A; $x \in B$; $e \in D$; $A \subseteq B$ |] ==>
$f\ x$ (*foldD* $D\ f\ e\ A$) = *foldD* $D\ f$ $(f\ x\ e)$ A
⟨*proof*⟩

lemma *Int-mono2*:
  $[\![A \subseteq C; B \subseteq C]\!] \Longrightarrow A \ Int \ B \subseteq C$
  ⟨*proof*⟩

lemma (**in** *LCD*) *foldD-nest-Un-Int*:
  $[\![$ *finite* A; *finite* C; $e \in D$; $A \subseteq B$; $C \subseteq B$ $]\!] \Longrightarrow$
   foldD D f (*foldD* D f e C) A = *foldD* D f (*foldD* D f e (A *Int* C)) (A *Un* C)
  ⟨*proof*⟩

lemma (**in** *LCD*) *foldD-nest-Un-disjoint*:
  $[\![$ *finite* A; *finite* B; A *Int* B = {}; $e \in D$; $A \subseteq B$; $C \subseteq B$ $]\!]$
    $\Longrightarrow$ *foldD* D f e (A *Un* B) = *foldD* D f (*foldD* D f e B) A
  ⟨*proof*⟩

declare *foldSetD-imp-finite* [*simp del*]
  empty-foldSetDE [*rule del*]
  foldSetD.intros [*rule del*]
declare (**in** *LCD*)
  foldSetD-closed [*rule del*]

3.3 Commutative Monoids

We enter a more restrictive context, with $f :: 'a => 'a => 'a$ instead of $'b => 'a => 'a$.

locale *ACeD* =
  fixes D :: *'a set*
    and f :: $'a => 'a => 'a$ (**infixl** $\cdot$ *70*)
    and e :: $'a$
  assumes *ident* [*simp*]: $x \in D \Longrightarrow x \cdot e = x$
    and *commute*: $[\![x \in D; y \in D]\!] \Longrightarrow x \cdot y = y \cdot x$
    and *assoc*: $[\![x \in D; y \in D; z \in D]\!] \Longrightarrow (x \cdot y) \cdot z = x \cdot (y \cdot z)$
    and *e-closed* [*simp*]: $e \in D$
    and *f-closed* [*simp*]: $[\![x \in D; y \in D]\!] \Longrightarrow x \cdot y \in D$

lemma (**in** *ACeD*) *left-commute*:
  $[\![x \in D; y \in D; z \in D]\!] \Longrightarrow x \cdot (y \cdot z) = y \cdot (x \cdot z)$
⟨*proof*⟩

lemmas (**in** *ACeD*) *AC* = *assoc commute left-commute*

lemma (**in** *ACeD*) *left-ident* [*simp*]: $x \in D \Longrightarrow e \cdot x = x$
⟨*proof*⟩

lemma (**in** *ACeD*) *foldD-Un-Int*:
  $[\![$ *finite* A; *finite* B; $A \subseteq D$; $B \subseteq D$ $]\!] \Longrightarrow$
    foldD D f e A $\cdot$ *foldD* D f e B =
    foldD D f e (A *Un* B) $\cdot$ *foldD* D f e (A *Int* B)
  ⟨*proof*⟩

lemma (**in** *ACeD*) *foldD-Un-disjoint*:
[| *finite A*; *finite B*; *A Int B* = {}; $A \subseteq D$; $B \subseteq D$ |] ==>
foldD D f e (*A Un B*) = *foldD D f e A* · *foldD D f e B*
⟨*proof*⟩

3.4 Products over Finite Sets

constdefs (**structure** *G*)
finprod :: [(*'b*, *'m*) *monoid-scheme*, *'a* => *'b*, *'a set*] => *'b*
finprod G f A == *if finite A*
then foldD (*carrier G*) (*mult G o f*) **1** *A*
else arbitrary

syntax
-finprod :: *index* => *idt* => *'a set* => *'b* => *'b*
((*3*⨂ --:-. -) [*1000*, *0*, *51*, *10*] *10*)
syntax (*xsymbols*)
-finprod :: *index* => *idt* => *'a set* => *'b* => *'b*
((*3*⨂ --∈-. -) [*1000*, *0*, *51*, *10*] *10*)
syntax (*HTML* **output**)
-finprod :: *index* => *idt* => *'a set* => *'b* => *'b*
((*3*⨂ --∈-. -) [*1000*, *0*, *51*, *10*] *10*)
translations
⨂ ı*i*:*A*. *b* == *finprod* ◇ı (%*i*. *b*) *A*
— Beware of argument permutation!

lemma (**in** *comm-monoid*) *finprod-empty* [*simp*]:
finprod G f {} = **1**
⟨*proof*⟩

declare *funcsetI* [*intro*]
funcset-mem [*dest*]

lemma (**in** *comm-monoid*) *finprod-insert* [*simp*]:
[| *finite F*; $a \notin F$; $f \in F$ −> *carrier G*; $f\ a \in$ *carrier G* |] ==>
finprod G f (*insert a F*) = $f\ a \otimes$ *finprod G f F*
⟨*proof*⟩

lemma (**in** *comm-monoid*) *finprod-one* [*simp*]:
finite A ==> (⨂ *i*:*A*. **1**) = **1**
⟨*proof*⟩

lemma (**in** *comm-monoid*) *finprod-closed* [*simp*]:
fixes *A*
assumes *fin*: *finite A* **and** *f*: $f \in A$ −> *carrier G*
shows *finprod G f A* ∈ *carrier G*
⟨*proof*⟩

lemma *funcset-Int-left* [*simp*, *intro*]:

[| $f \in A$ −> C; $f \in B$ −> C |] ==> $f \in A$ *Int* B −> C
⟨*proof*⟩

lemma *funcset-Un-left* [*iff*]:
($f \in A$ *Un* B −> C) = ($f \in A$ −> C & $f \in B$ −> C)
⟨*proof*⟩

lemma (**in** *comm-monoid*) *finprod-Un-Int*:
[| *finite* A; *finite* B; $g \in A$ −> *carrier* G; $g \in B$ −> *carrier* G |] ==>
finprod G g (A *Un* B) ⊗ *finprod* G g (A *Int* B) =
finprod G g A ⊗ *finprod* G g B
— The reversed orientation looks more natural, but LOOPS as a simprule!
⟨*proof*⟩

lemma (**in** *comm-monoid*) *finprod-Un-disjoint*:
[| *finite* A; *finite* B; A *Int* B = {};
$g \in A$ −> *carrier* G; $g \in B$ −> *carrier* G |]
==> *finprod* G g (A *Un* B) = *finprod* G g A ⊗ *finprod* G g B
⟨*proof*⟩

lemma (**in** *comm-monoid*) *finprod-multf*:
[| *finite* A; $f \in A$ −> *carrier* G; $g \in A$ −> *carrier* G |] ==>
finprod G (%x. f x ⊗ g x) A = (*finprod* G f A ⊗ *finprod* G g A)
⟨*proof*⟩

lemma (**in** *comm-monoid*) *finprod-cong'*:
[| A = B; $g \in B$ −> *carrier* G;
!!i. $i \in B$ ==> f i = g i |] ==> *finprod* G f A = *finprod* G g B
⟨*proof*⟩

lemma (**in** *comm-monoid*) *finprod-cong*:
[| A = B; $f \in B$ −> *carrier* G = *True*;
!!i. $i \in B$ ==> f i = g i |] ==> *finprod* G f A = *finprod* G g B

⟨*proof*⟩

Usually, if this rule causes a failed congruence proof error, the reason is that the premise $g \in B$ −> *carrier* G cannot be shown. Adding *Pi-def* to the simpset is often useful. For this reason, *comm-monoid.finprod-cong* is not added to the simpset by default.

declare *funcsetI* [*rule del*]
funcset-mem [*rule del*]

lemma (**in** *comm-monoid*) *finprod-0* [*simp*]:
$f \in$ {*0::nat*} −> *carrier* G ==> *finprod* G f {..*0*} = f *0*
⟨*proof*⟩

lemma (**in** *comm-monoid*) *finprod-Suc* [*simp*]:
$f \in$ {..*Suc* n} −> *carrier* G ==>

finprod G f {..Suc n} = (f (Suc n) ⊗ finprod G f {..n})
⟨*proof*⟩

lemma (**in** *comm-monoid*) *finprod-Suc2*:
f ∈ {..Suc n} −> carrier G ==>
finprod G f {..Suc n} = (finprod G (%i. f (Suc i)) {..n} ⊗ f 0)
⟨*proof*⟩

lemma (**in** *comm-monoid*) *finprod-mult* [*simp*]:
[| f ∈ {..n} −> carrier G; g ∈ {..n} −> carrier G |] ==>
finprod G (%i. f i ⊗ g i) {..n::nat} =
finprod G f {..n} ⊗ finprod G g {..n}
⟨*proof*⟩

end

theory *Exponent* **imports** *Main Primes Binomial* **begin**

4 The Combinatorial Argument Underlying the First Sylow Theorem

definition *exponent* :: *nat => nat => nat* **where**
exponent p s == if prime p then (GREATEST r. p^r dvd s) else 0

4.1 Prime Theorems

lemma *prime-imp-one-less*: *prime p ==> Suc 0 < p*
⟨*proof*⟩

lemma *prime-iff*:
(prime p) = (Suc 0 < p & (∀ a b. p dvd a∗b −−> (p dvd a) | (p dvd b)))
⟨*proof*⟩

lemma *zero-less-prime-power*: *prime p ==> 0 < p^a*
⟨*proof*⟩

lemma *zero-less-card-empty*: *[| finite S; S ≠ {} |] ==> 0 < card(S)*
⟨*proof*⟩

lemma *prime-dvd-cases*:
[| p∗k dvd m∗n; prime p |]
==> (∃ x. k dvd x∗n & m = p∗x) | (∃ y. k dvd m∗y & n = p∗y)
⟨*proof*⟩

lemma *prime-power-dvd-cases* [*rule-format* (*no-asm*)]: *prime p*
$==> \forall\, m\ n.\ p\hat{\ }c\ dvd\ m{*}n\ -->$
$(\forall\, a\ b.\ a{+}b = Suc\ c\ --> p\hat{\ }a\ dvd\ m \mid p\hat{\ }b\ dvd\ n)$
⟨*proof*⟩

lemma *div-combine*:
$[|\ prime\ p;\ \sim\ (p\ \hat{\ }\ (Suc\ r)\ dvd\ n);\ \ p\hat{\ }(a{+}r)\ dvd\ n{*}k\ |]$
$==> p\ \hat{\ }\ a\ dvd\ k$
⟨*proof*⟩

lemma *Suc-le-power*: $Suc\ 0 < p ==> Suc\ n <= p\hat{\ }n$
⟨*proof*⟩

lemma *power-dvd-bound*: $[|p\hat{\ }n\ dvd\ a;\ \ Suc\ 0 < p;\ \ a > 0|] ==> n < a$
⟨*proof*⟩

4.2 Exponent Theorems

lemma *exponent-ge* [*rule-format*]:
$[|p\hat{\ }k\ dvd\ n;\ \ prime\ p;\ \ 0{<}n|] ==> k <= exponent\ p\ n$
⟨*proof*⟩

lemma *power-exponent-dvd*: $s{>}0 ==> (p\ \hat{\ }\ exponent\ p\ s)\ dvd\ s$
⟨*proof*⟩

lemma *power-Suc-exponent-Not-dvd*:
$[|(p * p\ \hat{\ }\ exponent\ p\ s)\ dvd\ s;\ \ prime\ p\ |] ==> s{=}0$
⟨*proof*⟩

lemma *exponent-power-eq* [*simp*]: $prime\ p ==> exponent\ p\ (p\hat{\ }a) = a$
⟨*proof*⟩

lemma *exponent-equalityI*:
$!r{::}nat.\ (p\hat{\ }r\ dvd\ a) = (p\hat{\ }r\ dvd\ b) ==> exponent\ p\ a = exponent\ p\ b$
⟨*proof*⟩

lemma *exponent-eq-0* [*simp*]: $\neg\ prime\ p ==> exponent\ p\ s = 0$
⟨*proof*⟩

lemma *exponent-mult-add1*: $[|\ a > 0;\ b > 0\ |]$
$==> (exponent\ p\ a) + (exponent\ p\ b) <= exponent\ p\ (a * b)$
⟨*proof*⟩

lemma *exponent-mult-add2*: $[| \ a > 0; \ b > 0 \ |]$
$==>$ *exponent p* $(a * b) <= ($*exponent p a*$) + ($*exponent p b*$)$
⟨*proof*⟩

lemma *exponent-mult-add*: $[| \ a > 0; \ b > 0 \ |]$
$==>$ *exponent p* $(a * b) = ($*exponent p a*$) + ($*exponent p b*$)$
⟨*proof*⟩

lemma *not-divides-exponent-0*: $\sim$ $(p \ dvd \ n)$ $==>$ *exponent p n* $= 0$
⟨*proof*⟩

lemma *exponent-1-eq-0* [*simp*]: *exponent p* $(Suc \ 0) = 0$
⟨*proof*⟩

4.3 Main Combinatorial Argument

lemma *le-extend-mult*: $[| \ c > 0; \ a <= b \ |] ==> a <= b * (c::nat)$
⟨*proof*⟩

lemma *p-fac-forw-lemma*:
$[| \ (m::nat) > 0; \ k > 0; \ k < p\hat{}a; \ (p\hat{}r) \ dvd \ (p\hat{}a)* \ m - k \ |] ==> r <= a$
⟨*proof*⟩

lemma *p-fac-forw*: $[| \ (m::nat) > 0; \ k>0; \ k < p\hat{}a; \ (p\hat{}r) \ dvd \ (p\hat{}a)* \ m - k \ |]$
$==> (p\hat{}r) \ dvd \ (p\hat{}a) - k$
⟨*proof*⟩

lemma *r-le-a-forw*:
$[| \ (k::nat) > 0; \ k < p\hat{}a; \ p>0; \ (p\hat{}r) \ dvd \ (p\hat{}a) - k \ |] ==> r <= a$
⟨*proof*⟩

lemma *p-fac-backw*: $[| \ m>0; \ k>0; \ (p::nat) \neq 0; \ \ k < p\hat{}a; \ \ (p\hat{}r) \ dvd \ p\hat{}a - k \ |]$
$==> (p\hat{}r) \ dvd \ (p\hat{}a)*m - k$
⟨*proof*⟩

lemma *exponent-p-a-m-k-equation*: $[| \ m>0; \ k>0; \ (p::nat) \neq 0; \ \ k < p\hat{}a \ |]$
$==>$ *exponent p* $(p\hat{}a * m - k) =$ *exponent p* $(p\hat{}a - k)$
⟨*proof*⟩

Suc rules that we have to delete from the simpset

lemmas *bad-Sucs* = *binomial-Suc-Suc mult-Suc mult-Suc-right*

lemma *p-not-div-choose-lemma* [*rule-format*]:
$[| \ \forall i. \ Suc \ i < K \ --> $ *exponent p* $(Suc \ i) =$ *exponent p* $(Suc(j+i))|]$

==> $k<K$ --> *exponent p* $((j+k)$ *choose* $k) = 0$
⟨*proof*⟩

lemma *p-not-div-choose*:
[| $k<K$; $k<=n$;
$\forall j.\ 0<j$ & $j<K$ --> *exponent p* $(n - k + (K - j))$ = *exponent p* $(K - j)$|]
==> *exponent p* $(n$ *choose* $k) = 0$
⟨*proof*⟩

lemma *const-p-fac-right*:
$m>0$ ==> *exponent p* $((p\hat{}a * m - Suc\ 0)$ *choose* $(p\hat{}a - Suc\ 0)) = 0$
⟨*proof*⟩

lemma *const-p-fac*:
$m>0$ ==> *exponent p* $(((p\hat{}a) * m)$ *choose* $p\hat{}a)$ = *exponent p m*
⟨*proof*⟩

end

theory *Coset* **imports** *Group Exponent* **begin**

5 Cosets and Quotient Groups

constdefs (**structure** *G*)
r-coset :: [-, 'a set, 'a] ⇒ 'a set (**infixl** #>ı *60*)
H #> $a \equiv \bigcup h{\in}H.\ \{h \otimes a\}$

l-coset :: [-, 'a, 'a set] ⇒ 'a set (**infixl** <#ı *60*)
a <# $H \equiv \bigcup h{\in}H.\ \{a \otimes h\}$

RCOSETS :: [-, 'a set] ⇒ ('a set)set (*rcosets*ı - [*81*] *80*)
rcosets $H \equiv \bigcup a{\in}carrier\ G.\ \{H$ #> $a\}$

set-mult :: [-, 'a set ,'a set] ⇒ 'a set (**infixl** <#>ı *60*)
H <#> $K \equiv \bigcup h{\in}H.\ \bigcup k{\in}K.\ \{h \otimes k\}$

SET-INV :: [-,'a set] ⇒ 'a set (*set'-inv*ı - [*81*] *80*)
set-inv $H \equiv \bigcup h{\in}H.\ \{inv\ h\}$

locale *normal* = *subgroup* + *group* +
assumes *coset-eq*: $(\forall x \in carrier\ G.\ H$ #> $x = x$ <# $H)$

abbreviation
normal-rel :: [$'a$ *set*, ($'a$, $'b$) *monoid-scheme*] $\Rightarrow$ *bool* (**infixl** $\lhd$ *60*) **where**
$H \lhd G \equiv$ *normal* H G

5.1 Basic Properties of Cosets

lemma (**in** *group*) *coset-mult-assoc*:
[| $M \subseteq$ *carrier* G; $g \in$ *carrier* G; $h \in$ *carrier* G |]
==> (M #> g) #> h = M #> ($g \otimes h$)
⟨*proof*⟩

lemma (**in** *group*) *coset-mult-one* [*simp*]: $M \subseteq$ *carrier* G ==> M #> **1** = M
⟨*proof*⟩

lemma (**in** *group*) *coset-mult-inv1*:
[| M #> ($x \otimes$ (*inv* y)) = M; $x \in$ *carrier* G ; $y \in$ *carrier* G;
$M \subseteq$ *carrier* G |] ==> M #> x = M #> y
⟨*proof*⟩

lemma (**in** *group*) *coset-mult-inv2*:
[| M #> x = M #> y; $x \in$ *carrier* G; $y \in$ *carrier* G; $M \subseteq$ *carrier* G |]
==> M #> ($x \otimes$ (*inv* y)) = M
⟨*proof*⟩

lemma (**in** *group*) *coset-join1*:
[| H #> x = H; $x \in$ *carrier* G; *subgroup* H G |] ==> $x \in H$
⟨*proof*⟩

lemma (**in** *group*) *solve-equation*:
$[\![$*subgroup* H G; $x \in H$; $y \in H]\!] \Longrightarrow \exists h \in H.\ y = h \otimes x$
⟨*proof*⟩

lemma (**in** *group*) *repr-independence*:
$[\![y \in H$ #> x; $x \in$ *carrier* G; *subgroup* H $G]\!] \Longrightarrow H$ #> x = H #> y
⟨*proof*⟩

lemma (**in** *group*) *coset-join2*:
$[\![x \in$ *carrier* G; *subgroup* H G; $x \in H]\!] \Longrightarrow H$ #> x = H
— Alternative proof is to put x = **1** in *repr-independence*.
⟨*proof*⟩

lemma (**in** *monoid*) *r-coset-subset-G*:
[| $H \subseteq$ *carrier* G; $x \in$ *carrier* G |] ==> H #> $x \subseteq$ *carrier* G
⟨*proof*⟩

lemma (**in** *group*) *rcosI*:
[| $h \in H$; $H \subseteq$ *carrier* G; $x \in$ *carrier* G|] ==> $h \otimes x \in H$ #> x
⟨*proof*⟩

lemma (**in** *group*) *rcosetsI*:
$[\![H \subseteq carrier\ G;\ x \in carrier\ G]\!] \Longrightarrow H\ \#>\ x \in rcosets\ H$
$\langle proof \rangle$

Really needed?

lemma (**in** *group*) *transpose-inv*:
[| $x \otimes y = z$; $x \in carrier\ G$; $y \in carrier\ G$; $z \in carrier\ G$ |]
==> $(inv\ x) \otimes z = y$
$\langle proof \rangle$

lemma (**in** *group*) *rcos-self*: [| $x \in carrier\ G$; *subgroup* $H\ G$ |] ==> $x \in H\ \#>\ x$
$\langle proof \rangle$

Opposite of *repr-independence*

lemma (**in** *group*) *repr-independenceD*:
includes *subgroup* $H\ G$
assumes *ycarr*: $y \in carrier\ G$
and *repr*: $H\ \#>\ x = H\ \#>\ y$
shows $y \in H\ \#>\ x$
$\langle proof \rangle$

Elements of a right coset are in the carrier

lemma (**in** *subgroup*) *elemrcos-carrier*:
includes *group*
assumes *acarr*: $a \in carrier\ G$
and a': $a' \in H\ \#>\ a$
shows $a' \in carrier\ G$
$\langle proof \rangle$

lemma (**in** *subgroup*) *rcos-const*:
includes *group*
assumes *hH*: $h \in H$
shows $H\ \#>\ h = H$
$\langle proof \rangle$

Step one for lemma *rcos-module*

lemma (**in** *subgroup*) *rcos-module-imp*:
includes *group*
assumes *xcarr*: $x \in carrier\ G$
and $x'cos$: $x' \in H\ \#>\ x$
shows $(x' \otimes inv\ x) \in H$
$\langle proof \rangle$

Step two for lemma *rcos-module*

lemma (**in** *subgroup*) *rcos-module-rev*:
includes *group*
assumes *carr*: $x \in carrier\ G\ x' \in carrier\ G$
and *xixH*: $(x' \otimes inv\ x) \in H$

shows $x' \in H \ \#> \ x$
$\langle proof \rangle$

Module property of right cosets

lemma (**in** *subgroup*) *rcos-module*:
includes *group*
assumes *carr*: $x \in carrier\ G\ x' \in carrier\ G$
shows $(x' \in H \ \#> \ x) = (x' \otimes inv\ x \in H)$
$\langle proof \rangle$

Right cosets are subsets of the carrier.

lemma (**in** *subgroup*) *rcosets-carrier*:
includes *group*
assumes *XH*: $X \in rcosets\ H$
shows $X \subseteq carrier\ G$
$\langle proof \rangle$

Multiplication of general subsets

lemma (**in** *monoid*) *set-mult-closed*:
assumes *Acarr*: $A \subseteq carrier\ G$
and *Bcarr*: $B \subseteq carrier\ G$
shows $A <\#> B \subseteq carrier\ G$
$\langle proof \rangle$

lemma (**in** *comm-group*) *mult-subgroups*:
assumes *subH*: *subgroup* H G
and *subK*: *subgroup* K G
shows *subgroup* $(H <\#> K)$ G
$\langle proof \rangle$

lemma (**in** *subgroup*) *lcos-module-rev*:
includes *group*
assumes *carr*: $x \in carrier\ G\ x' \in carrier\ G$
and *xixH*: $(inv\ x \otimes x') \in H$
shows $x' \in x <\# \ H$
$\langle proof \rangle$

5.2 Normal subgroups

lemma *normal-imp-subgroup*: $H \lhd G \Longrightarrow subgroup\ H\ G$
$\langle proof \rangle$

lemma (**in** *group*) *normalI*:
$subgroup\ H\ G \Longrightarrow (\forall x \in carrier\ G.\ H \ \#> \ x = x <\# \ H) \Longrightarrow H \lhd G$
$\langle proof \rangle$

lemma (**in** *normal*) *inv-op-closed1*:
$[\![x \in carrier\ G;\ h \in H]\!] \Longrightarrow (inv\ x) \otimes h \otimes x \in H$
$\langle proof \rangle$

lemma (**in** *normal*) *inv-op-closed2*:
$[\![x \in carrier\ G;\ h \in H]\!] \Longrightarrow x \otimes h \otimes (inv\ x) \in H$
⟨*proof*⟩

Alternative characterization of normal subgroups

lemma (**in** *group*) *normal-inv-iff*:
$(N \lhd G) =$
$(subgroup\ N\ G\ \&\ (\forall x \in carrier\ G.\ \forall h \in N.\ x \otimes h \otimes (inv\ x) \in N))$
(**is** - = *?rhs*)
⟨*proof*⟩

5.3 More Properties of Cosets

lemma (**in** *group*) *lcos-m-assoc*:
$[|\ M \subseteq carrier\ G;\ g \in carrier\ G;\ h \in carrier\ G\ |]$
$==> g <\#\ (h <\#\ M) = (g \otimes h) <\#\ M$
⟨*proof*⟩

lemma (**in** *group*) *lcos-mult-one*: $M \subseteq carrier\ G ==> \mathbf{1} <\#\ M = M$
⟨*proof*⟩

lemma (**in** *group*) *l-coset-subset-G*:
$[|\ H \subseteq carrier\ G;\ x \in carrier\ G\ |] ==> x <\#\ H \subseteq carrier\ G$
⟨*proof*⟩

lemma (**in** *group*) *l-coset-swap*:
$[\![y \in x <\#\ H;\ \ x \in carrier\ G;\ \ subgroup\ H\ G]\!] \Longrightarrow x \in y <\#\ H$
⟨*proof*⟩

lemma (**in** *group*) *l-coset-carrier*:
$[|\ y \in x <\#\ H;\ \ x \in carrier\ G;\ \ subgroup\ H\ G\ |] ==> y \in carrier\ G$
⟨*proof*⟩

lemma (**in** *group*) *l-repr-imp-subset*:
assumes *y*: $y \in x <\#\ H$ **and** *x*: $x \in carrier\ G$ **and** *sb*: *subgroup H G*
shows $y <\#\ H \subseteq x <\#\ H$
⟨*proof*⟩

lemma (**in** *group*) *l-repr-independence*:
assumes *y*: $y \in x <\#\ H$ **and** *x*: $x \in carrier\ G$ **and** *sb*: *subgroup H G*
shows $x <\#\ H = y <\#\ H$
⟨*proof*⟩

lemma (**in** *group*) *setmult-subset-G*:
$[\![H \subseteq carrier\ G;\ K \subseteq carrier\ G]\!] \Longrightarrow H <\#> K \subseteq carrier\ G$
⟨*proof*⟩

lemma (**in** *group*) *subgroup-mult-id*: $subgroup\ H\ G \Longrightarrow H <\#> H = H$

$\langle proof \rangle$

5.3.1 Set of Inverses of an *r-coset*.

lemma (**in** *normal*) *rcos-inv*:
assumes x: $x \in carrier\ G$
shows $set\text{-}inv\ (H\ \#{>}\ x) = H\ \#{>}\ (inv\ x)$
$\langle proof \rangle$

5.3.2 Theorems for <#> with #> or <#.

lemma (**in** *group*) *setmult-rcos-assoc*:
$[\![H \subseteq carrier\ G;\ K \subseteq carrier\ G;\ x \in carrier\ G]\!]$
$\Longrightarrow H\ {<}\#{>}\ (K\ \#{>}\ x) = (H\ {<}\#{>}\ K)\ \#{>}\ x$
$\langle proof \rangle$

lemma (**in** *group*) *rcos-assoc-lcos*:
$[\![H \subseteq carrier\ G;\ K \subseteq carrier\ G;\ x \in carrier\ G]\!]$
$\Longrightarrow (H\ \#{>}\ x)\ {<}\#{>}\ K = H\ {<}\#{>}\ (x\ {<}\#\ K)$
$\langle proof \rangle$

lemma (**in** *normal*) *rcos-mult-step1*:
$[\![x \in carrier\ G;\ y \in carrier\ G]\!]$
$\Longrightarrow (H\ \#{>}\ x)\ {<}\#{>}\ (H\ \#{>}\ y) = (H\ {<}\#{>}\ (x\ {<}\#\ H))\ \#{>}\ y$
$\langle proof \rangle$

lemma (**in** *normal*) *rcos-mult-step2*:
$[\![x \in carrier\ G;\ y \in carrier\ G]\!]$
$\Longrightarrow (H\ {<}\#{>}\ (x\ {<}\#\ H))\ \#{>}\ y = (H\ {<}\#{>}\ (H\ \#{>}\ x))\ \#{>}\ y$
$\langle proof \rangle$

lemma (**in** *normal*) *rcos-mult-step3*:
$[\![x \in carrier\ G;\ y \in carrier\ G]\!]$
$\Longrightarrow (H\ {<}\#{>}\ (H\ \#{>}\ x))\ \#{>}\ y = H\ \#{>}\ (x \otimes y)$
$\langle proof \rangle$

lemma (**in** *normal*) *rcos-sum*:
$[\![x \in carrier\ G;\ y \in carrier\ G]\!]$
$\Longrightarrow (H\ \#{>}\ x)\ {<}\#{>}\ (H\ \#{>}\ y) = H\ \#{>}\ (x \otimes y)$
$\langle proof \rangle$

lemma (**in** *normal*) *rcosets-mult-eq*: $M \in rcosets\ H \Longrightarrow H\ {<}\#{>}\ M = M$
— generalizes *subgroup-mult-id*
$\langle proof \rangle$

5.3.3 An Equivalence Relation

constdefs (**structure** G)
r-congruent :: $[('a,'b)monoid\text{-}scheme,\ 'a\ set] \Rightarrow ('a*'a)set$
$(rcong_1\ \text{-})$

rcong H ≡ {(*x*,*y*). *x* ∈ *carrier G* & *y* ∈ *carrier G* & *inv x* ⊗ *y* ∈ *H*}

lemma (**in** *subgroup*) *equiv-rcong*:
  includes *group G*
  shows *equiv* (*carrier G*) (*rcong H*)
⟨*proof*⟩

Equivalence classes of *rcong* correspond to left cosets. Was there a mistake in the definitions? I'd have expected them to correspond to right cosets.

lemma (**in** *subgroup*) *l-coset-eq-rcong*:
  includes *group G*
  assumes *a*: *a* ∈ *carrier G*
  shows *a* <# *H* = *rcong H* ‘‘ {*a*}
⟨*proof*⟩

5.3.4 Two Distinct Right Cosets are Disjoint

lemma (**in** *group*) *rcos-equation*:
  includes *subgroup H G*
  shows
    ⟦*ha* ⊗ *a* = *h* ⊗ *b*; *a* ∈ *carrier G*; *b* ∈ *carrier G*;
      h ∈ *H*; *ha* ∈ *H*; *hb* ∈ *H*⟧
    ⟹ *hb* ⊗ *a* ∈ (⋃*h*∈*H*. {*h* ⊗ *b*})
⟨*proof*⟩

lemma (**in** *group*) *rcos-disjoint*:
  includes *subgroup H G*
  shows ⟦*a* ∈ *rcosets H*; *b* ∈ *rcosets H*; *a*≠*b*⟧ ⟹ *a* ∩ *b* = {}
⟨*proof*⟩

5.4 Further lemmas for *r-congruent*

The relation is a congruence

lemma (**in** *normal*) *congruent-rcong*:
  shows *congruent2* (*rcong H*) (*rcong H*) (λ*a b*. *a* ⊗ *b* <# *H*)
⟨*proof*⟩

5.5 Order of a Group and Lagrange's Theorem

constdefs
  order :: ('*a*, '*b*) *monoid-scheme* ⇒ *nat*
  order S ≡ *card* (*carrier S*)

lemma (**in** *group*) *rcos-self*:
  includes *subgroup*
  shows *x* ∈ *carrier G* ⟹ *x* ∈ *H* #> *x*
⟨*proof*⟩

lemma (**in** *group*) *rcosets-part-G*:
includes *subgroup*
shows $\bigcup(rcosets\ H) = carrier\ G$
⟨*proof*⟩

lemma (**in** *group*) *cosets-finite*:
$[\![c \in rcosets\ H;\ \ H \subseteq carrier\ G;\ \ finite\ (carrier\ G)]\!] \Longrightarrow finite\ c$
⟨*proof*⟩

The next two lemmas support the proof of *card-cosets-equal*.

lemma (**in** *group*) *inj-on-f*:
$[\![H \subseteq carrier\ G;\ \ a \in carrier\ G]\!] \Longrightarrow inj\text{-}on\ (\lambda y.\ y \otimes inv\ a)\ (H\ \#{>}\ a)$
⟨*proof*⟩

lemma (**in** *group*) *inj-on-g*:
$[\![H \subseteq carrier\ G;\ \ a \in carrier\ G]\!] \Longrightarrow inj\text{-}on\ (\lambda y.\ y \otimes a)\ H$
⟨*proof*⟩

lemma (**in** *group*) *card-cosets-equal*:
$[\![c \in rcosets\ H;\ \ H \subseteq carrier\ G;\ finite(carrier\ G)]\!]$
$\Longrightarrow card\ c = card\ H$
⟨*proof*⟩

lemma (**in** *group*) *rcosets-subset-PowG*:
$subgroup\ H\ G\ \Longrightarrow rcosets\ H \subseteq Pow(carrier\ G)$
⟨*proof*⟩

theorem (**in** *group*) *lagrange*:
$[\![finite(carrier\ G);\ subgroup\ H\ G]\!]$
$\Longrightarrow card(rcosets\ H) * card(H) = order(G)$
⟨*proof*⟩

5.6 Quotient Groups: Factorization of a Group

constdefs
$FactGroup :: [('a,'b)\ monoid\text{-}scheme,\ 'a\ set] \Rightarrow ('a\ set)\ monoid$
(**infixl** *Mod 65*)
— Actually defined for groups rather than monoids
$FactGroup\ G\ H \equiv$
$(\!|carrier = rcosets_G\ H,\ mult = set\text{-}mult\ G,\ one = H|\!)$

lemma (**in** *normal*) *setmult-closed*:
$[\![K1 \in rcosets\ H;\ K2 \in rcosets\ H]\!] \Longrightarrow K1 <\#> K2 \in rcosets\ H$
⟨*proof*⟩

lemma (**in** *normal*) *setinv-closed*:
$K \in rcosets\ H \Longrightarrow set\text{-}inv\ K \in rcosets\ H$
⟨*proof*⟩

lemma (**in** *normal*) *rcosets-assoc*:
$[\![M1 \in \mathit{rcosets}\ H;\ M2 \in \mathit{rcosets}\ H;\ M3 \in \mathit{rcosets}\ H]\!]$
$\Longrightarrow M1 <\#> M2 <\#> M3 = M1 <\#> (M2 <\#> M3)$
⟨*proof*⟩

lemma (**in** *subgroup*) *subgroup-in-rcosets*:
includes *group G*
shows $H \in \mathit{rcosets}\ H$
⟨*proof*⟩

lemma (**in** *normal*) *rcosets-inv-mult-group-eq*:
$M \in \mathit{rcosets}\ H \Longrightarrow \mathit{set\text{-}inv}\ M <\#> M = H$
⟨*proof*⟩

theorem (**in** *normal*) *factorgroup-is-group*:
group (*G Mod H*)
⟨*proof*⟩

lemma *mult-FactGroup* [*simp*]: $X \otimes_{(G\ \mathit{Mod}\ H)} X' = X <\#>_G X'$
⟨*proof*⟩

lemma (**in** *normal*) *inv-FactGroup*:
$X \in \mathit{carrier}\ (G\ \mathit{Mod}\ H) \Longrightarrow \mathit{inv}_{G\ \mathit{Mod}\ H}\ X = \mathit{set\text{-}inv}\ X$
⟨*proof*⟩

The coset map is a homomorphism from G to the quotient group $G\ \mathit{Mod}\ H$

lemma (**in** *normal*) *r-coset-hom-Mod*:
$(\lambda a.\ H \#> a) \in \mathit{hom}\ G\ (G\ \mathit{Mod}\ H)$
⟨*proof*⟩

5.7 The First Isomorphism Theorem

The quotient by the kernel of a homomorphism is isomorphic to the range of that homomorphism.

constdefs
$\mathit{kernel} :: ('a, 'm)\ \mathit{monoid\text{-}scheme} \Rightarrow ('b, 'n)\ \mathit{monoid\text{-}scheme} \Rightarrow$
$('a \Rightarrow 'b) \Rightarrow 'a\ \mathit{set}$
— the kernel of a homomorphism
$\mathit{kernel}\ G\ H\ h \equiv \{x.\ x \in \mathit{carrier}\ G\ \&\ h\ x = \mathbf{1}_H\}$

lemma (**in** *group-hom*) *subgroup-kernel*: *subgroup* (*kernel G H h*) *G*
⟨*proof*⟩

The kernel of a homomorphism is a normal subgroup

lemma (**in** *group-hom*) *normal-kernel*: $(\mathit{kernel}\ G\ H\ h) \lhd G$
⟨*proof*⟩

lemma (**in** *group-hom*) *FactGroup-nonempty*:
  assumes X: $X \in$ *carrier* $(G\ Mod\ kernel\ G\ H\ h)$
  shows $X \neq \{\}$
⟨*proof*⟩

lemma (**in** *group-hom*) *FactGroup-contents-mem*:
  assumes X: $X \in$ *carrier* $(G\ Mod\ (kernel\ G\ H\ h))$
  shows *contents* $(h`X) \in$ *carrier* H
⟨*proof*⟩

lemma (**in** *group-hom*) *FactGroup-hom*:
  $(\lambda X.\ contents\ (h`X)) \in hom\ (G\ Mod\ (kernel\ G\ H\ h))\ H$
⟨*proof*⟩

Lemma for the following injectivity result

lemma (**in** *group-hom*) *FactGroup-subset*:
  $[\![g \in carrier\ G;\ g' \in carrier\ G;\ h\ g = h\ g']\!]$
  $\Longrightarrow\ kernel\ G\ H\ h\ \#{>}\ g \subseteq kernel\ G\ H\ h\ \#{>}\ g'$
⟨*proof*⟩

lemma (**in** *group-hom*) *FactGroup-inj-on*:
  inj-on $(\lambda X.\ contents\ (h\ `\ X))\ (carrier\ (G\ Mod\ kernel\ G\ H\ h))$
⟨*proof*⟩

If the homomorphism h is onto H, then so is the homomorphism from the quotient group

lemma (**in** *group-hom*) *FactGroup-onto*:
  assumes h: $h\ `\ carrier\ G = carrier\ H$
  shows $(\lambda X.\ contents\ (h\ `\ X))\ `\ carrier\ (G\ Mod\ kernel\ G\ H\ h) = carrier\ H$
⟨*proof*⟩

If h is a homomorphism from G onto H, then the quotient group $G\ Mod\ kernel\ G\ H\ h$ is isomorphic to H.

theorem (**in** *group-hom*) *FactGroup-iso*:
  $h\ `\ carrier\ G = carrier\ H$
  $\Longrightarrow (\lambda X.\ contents\ (h`X)) \in (G\ Mod\ (kernel\ G\ H\ h)) \cong H$
⟨*proof*⟩

end

theory *Sylow* **imports** *Coset* **begin**

6 Sylow's Theorem

See also [3].

The combinatorial argument is in theory Exponent

locale *sylow* = *group* +
  fixes *p* **and** *a* **and** *m* **and** *calM* **and** *RelM*
  assumes *prime-p*: *prime p*
    and *order-G*: *order*(*G*) = (*p* ^*a*) ∗ *m*
    and *finite-G* [*iff*]: *finite* (*carrier G*)
  defines *calM* == {*s*. *s* ⊆ *carrier*(*G*) & *card*(*s*) = *p* ^*a*}
    and *RelM* == {(*N1*,*N2*). *N1* ∈ *calM* & *N2* ∈ *calM* &
        (∃ *g* ∈ *carrier*(*G*). *N1* = (*N2* #> *g*))}

lemma (**in** *sylow*) *RelM-refl*: *refl calM RelM*
⟨*proof*⟩

lemma (**in** *sylow*) *RelM-sym*: *sym RelM*
⟨*proof*⟩

lemma (**in** *sylow*) *RelM-trans*: *trans RelM*
⟨*proof*⟩

lemma (**in** *sylow*) *RelM-equiv*: *equiv calM RelM*
⟨*proof*⟩

lemma (**in** *sylow*) *M-subset-calM-prep*: *M'* ∈ *calM* // *RelM* ==> *M'* ⊆ *calM*
⟨*proof*⟩

6.1 Main Part of the Proof

locale *sylow-central* = *sylow* +
  fixes *H* **and** *M1* **and** *M*
  assumes *M-in-quot*: *M* ∈ *calM* // *RelM*
    and *not-dvd-M*: ~(*p* ^ *Suc*(*exponent p m*) *dvd card*(*M*))
    and *M1-in-M*: *M1* ∈ *M*
  defines *H* == {*g*. *g*∈*carrier G* & *M1* #> *g* = *M1*}

lemma (**in** *sylow-central*) *M-subset-calM*: *M* ⊆ *calM*
⟨*proof*⟩

lemma (**in** *sylow-central*) *card-M1*: *card*(*M1*) = *p* ^*a*
⟨*proof*⟩

lemma *card-nonempty*: *0* < *card*(*S*) ==> *S* ≠ {}
⟨*proof*⟩

lemma (**in** *sylow-central*) *exists-x-in-M1*: ∃ *x*. *x*∈*M1*
⟨*proof*⟩

lemma (**in** *sylow-central*) *M1-subset-G* [*simp*]: $M1 \subseteq carrier\ G$
⟨*proof*⟩

lemma (**in** *sylow-central*) *M1-inj-H*: $\exists f \in H \rightarrow M1.\ inj\text{-}on\ f\ H$
⟨*proof*⟩

6.2 Discharging the Assumptions of *sylow-central*

lemma (**in** *sylow*) *EmptyNotInEquivSet*: $\{\} \notin calM\ //\ RelM$
⟨*proof*⟩

lemma (**in** *sylow*) *existsM1inM*: $M \in calM\ //\ RelM ==> \exists M1.\ M1 \in M$
⟨*proof*⟩

lemma (**in** *sylow*) *zero-less-o-G*: $0 < order(G)$
⟨*proof*⟩

lemma (**in** *sylow*) *zero-less-m*: $m > 0$
⟨*proof*⟩

lemma (**in** *sylow*) *card-calM*: $card(calM) = (p\hat{\ }a) * m\ choose\ p\hat{\ }a$
⟨*proof*⟩

lemma (**in** *sylow*) *zero-less-card-calM*: $card\ calM > 0$
⟨*proof*⟩

lemma (**in** *sylow*) *max-p-div-calM*:
$\sim (p\ \hat{}\ Suc(exponent\ p\ m)\ dvd\ card(calM))$
⟨*proof*⟩

lemma (**in** *sylow*) *finite-calM*: $finite\ calM$
⟨*proof*⟩

lemma (**in** *sylow*) *lemma-A1*:
$\exists M \in calM\ //\ RelM.\ \sim (p\ \hat{}\ Suc(exponent\ p\ m)\ dvd\ card(M))$
⟨*proof*⟩

6.2.1 Introduction and Destruct Rules for *H*

lemma (**in** *sylow-central*) *H-I*: $[|g \in carrier\ G;\ M1\ \#>\ g = M1|] ==> g \in H$
⟨*proof*⟩

lemma (**in** *sylow-central*) *H-into-carrier-G*: $x \in H ==> x \in carrier\ G$
⟨*proof*⟩

lemma (**in** *sylow-central*) *in-H-imp-eq*: $g : H ==> M1\ \#>\ g = M1$
⟨*proof*⟩

lemma (**in** *sylow-central*) *H-m-closed*: $[|\ x \in H;\ y \in H|] ==> x \otimes y \in H$

⟨*proof*⟩

lemma (**in** *sylow-central*) *H-not-empty*: $H \neq \{\}$
⟨*proof*⟩

lemma (**in** *sylow-central*) *H-is-subgroup*: *subgroup H G*
⟨*proof*⟩

lemma (**in** *sylow-central*) *rcosetGM1g-subset-G*:
  [| $g \in$ *carrier G*; $x \in$ *M1* #> g |] ==> $x \in$ *carrier G*
⟨*proof*⟩

lemma (**in** *sylow-central*) *finite-M1*: *finite M1*
⟨*proof*⟩

lemma (**in** *sylow-central*) *finite-rcosetGM1g*: $g \in$*carrier G* ==> *finite* (*M1* #> g)
⟨*proof*⟩

lemma (**in** *sylow-central*) *M1-cardeq-rcosetGM1g*:
  $g \in$ *carrier G* ==> *card*(*M1* #> g) = *card*(*M1*)
⟨*proof*⟩

lemma (**in** *sylow-central*) *M1-RelM-rcosetGM1g*:
  $g \in$ *carrier G* ==> (*M1*, *M1* #> g) $\in$ *RelM*
⟨*proof*⟩

6.3 Equal Cardinalities of M and the Set of Cosets

Injections between M and $rcosets_G\ H$ show that their cardinalities are equal.

lemma *ElemClassEquiv*:
  [| *equiv A r*; $C \in A$ // r |] ==> $\forall x \in C.\ \forall y \in C.\ (x,y) \in r$
⟨*proof*⟩

lemma (**in** *sylow-central*) *M-elem-map*:
  M2 $\in M$ ==> $\exists g.\ g \in$ *carrier G* & *M1* #> g = *M2*
⟨*proof*⟩

lemmas (**in** *sylow-central*) *M-elem-map-carrier* =
    M-elem-map [*THEN someI-ex*, *THEN conjunct1*]

lemmas (**in** *sylow-central*) *M-elem-map-eq* =
    M-elem-map [*THEN someI-ex*, *THEN conjunct2*]

lemma (**in** *sylow-central*) *M-funcset-rcosets-H*:
  (%x:M. H #> (*SOME* g. $g \in$ *carrier G* & *M1* #> g = x)) $\in M \to$ *rcosets H*
⟨*proof*⟩

lemma (**in** *sylow-central*) *inj-M-GmodH*: $\exists f \in M \to$*rcosets H*. *inj-on f M*

⟨*proof*⟩

6.3.1 The Opposite Injection

lemma (**in** *sylow-central*) *H-elem-map*:
  $H1 \in rcosets\ H \Longrightarrow \exists g.\ g \in carrier\ G\ \&\ H\ \#{>}\ g = H1$
⟨*proof*⟩

lemmas (**in** *sylow-central*) *H-elem-map-carrier* =
  H-elem-map [*THEN someI-ex*, *THEN conjunct1*]

lemmas (**in** *sylow-central*) *H-elem-map-eq* =
  H-elem-map [*THEN someI-ex*, *THEN conjunct2*]

lemma *EquivElemClass*:
  $[|equiv\ A\ r;\ M \in A//r;\ M1{\in}M;\ (M1,M2) \in r\ |] \Longrightarrow M2 \in M$
⟨*proof*⟩

lemma (**in** *sylow-central*) *rcosets-H-funcset-M*:
  $(\lambda C \in rcosets\ H.\ M1\ \#{>}\ (@g.\ g \in carrier\ G \wedge H\ \#{>}\ g = C)) \in rcosets\ H \rightarrow M$
⟨*proof*⟩

close to a duplicate of *inj-M-GmodH*

lemma (**in** *sylow-central*) *inj-GmodH-M*:
  $\exists g \in rcosets\ H{\rightarrow}M.\ inj\text{-}on\ g\ (rcosets\ H)$
⟨*proof*⟩

lemma (**in** *sylow-central*) *calM-subset-PowG*: $calM \subseteq Pow(carrier\ G)$
⟨*proof*⟩

lemma (**in** *sylow-central*) *finite-M*: *finite M*
⟨*proof*⟩

lemma (**in** *sylow-central*) *cardMeqIndexH*: $card(M) = card(rcosets\ H)$
⟨*proof*⟩

lemma (**in** *sylow-central*) *index-lem*: $card(M) * card(H) = order(G)$
⟨*proof*⟩

lemma (**in** *sylow-central*) *lemma-leq1*: $p\hat{}a \le card(H)$
⟨*proof*⟩

lemma (**in** *sylow-central*) *lemma-leq2*: $card(H) \le p\hat{}a$
⟨*proof*⟩

lemma (**in** *sylow-central*) *card-H-eq*: $card(H) = p\hat{\ }a$
⟨*proof*⟩

lemma (**in** *sylow*) *sylow-thm*: $\exists H.\ subgroup\ H\ G\ \&\ card(H) = p\hat{\ }a$
⟨*proof*⟩

Needed because the locale's automatic definition refers to *semigroup G* and *group-axioms G* rather than simply to *group G*.

lemma *sylow-eq*: $sylow\ G\ p\ a\ m = (group\ G\ \&\ sylow\text{-}axioms\ G\ p\ a\ m)$
⟨*proof*⟩

6.4 Sylow's Theorem

theorem *sylow-thm*:
$[|\ prime\ p;\ \ group(G);\ \ order(G) = (p\hat{\ }a) * m;\ finite\ (carrier\ G)|]$
$==> \exists H.\ subgroup\ H\ G\ \&\ card(H) = p\hat{\ }a$
⟨*proof*⟩

end

theory *Bij* **imports** *Group* **begin**

7 Bijections of a Set, Permutation Groups and Automorphism Groups

constdefs
$Bij :: {'a}\ set \Rightarrow ({'a} \Rightarrow {'a})\ set$
— Only extensional functions, since otherwise we get too many.
$Bij\ S \equiv extensional\ S \cap \{f.\ bij\text{-}betw\ f\ S\ S\}$

$BijGroup :: {'a}\ set \Rightarrow ({'a} \Rightarrow {'a})\ monoid$
$BijGroup\ S \equiv$
$(|carrier = Bij\ S,$
$mult = \lambda g \in Bij\ S.\ \lambda f \in Bij\ S.\ compose\ S\ g\ f,$
$one = \lambda x \in S.\ x|)$

declare *Id-compose* [*simp*] *compose-Id* [*simp*]

lemma *Bij-imp-extensional*: $f \in Bij\ S \Longrightarrow f \in extensional\ S$
⟨*proof*⟩

lemma *Bij-imp-funcset*: $f \in Bij\ S \Longrightarrow f \in S \rightarrow S$
⟨*proof*⟩

7.1 Bijections Form a Group

lemma *restrict-Inv-Bij*: $f \in Bij\ S \Longrightarrow (\lambda x \in S.\ (Inv\ S\ f)\ x) \in Bij\ S$
⟨*proof*⟩

lemma *id-Bij*: $(\lambda x{\in}S.\ x) \in Bij\ S$
⟨*proof*⟩

lemma *compose-Bij*: $[\![x \in Bij\ S;\ y \in Bij\ S]\!] \Longrightarrow compose\ S\ x\ y \in Bij\ S$
⟨*proof*⟩

lemma *Bij-compose-restrict-eq*:
$f \in Bij\ S \Longrightarrow compose\ S\ (restrict\ (Inv\ S\ f)\ S)\ f = (\lambda x{\in}S.\ x)$
⟨*proof*⟩

theorem *group-BijGroup*: $group\ (BijGroup\ S)$
⟨*proof*⟩

7.2 Automorphisms Form a Group

lemma *Bij-Inv-mem*: $[\![\ f \in Bij\ S;\ \ x \in S]\!] \Longrightarrow Inv\ S\ f\ x \in S$
⟨*proof*⟩

lemma *Bij-Inv-lemma*:
assumes *eq*: $\bigwedge x\ y.\ [\![x \in S;\ y \in S]\!] \Longrightarrow h(g\ x\ y) = g\ (h\ x)\ (h\ y)$
shows $[\![h \in Bij\ S;\ \ g \in S \to S \to S;\ \ x \in S;\ \ y \in S]\!]$
$\Longrightarrow Inv\ S\ h\ (g\ x\ y) = g\ (Inv\ S\ h\ x)\ (Inv\ S\ h\ y)$
⟨*proof*⟩

constdefs
$auto :: ('a,\ 'b)\ monoid\text{-}scheme \Rightarrow ('a \Rightarrow 'a)\ set$
$auto\ G \equiv hom\ G\ G \cap Bij\ (carrier\ G)$

$AutoGroup :: ('a,\ 'c)\ monoid\text{-}scheme \Rightarrow ('a \Rightarrow 'a)\ monoid$
$AutoGroup\ G \equiv BijGroup\ (carrier\ G)\ (\!|carrier := auto\ G|\!)$

lemma (**in** *group*) *id-in-auto*: $(\lambda x \in carrier\ G.\ x) \in auto\ G$
⟨*proof*⟩

lemma (**in** *group*) *mult-funcset*: $mult\ G \in carrier\ G \to carrier\ G \to carrier\ G$
⟨*proof*⟩

lemma (**in** *group*) *restrict-Inv-hom*:
$[\![h \in hom\ G\ G;\ h \in Bij\ (carrier\ G)]\!]$
$\Longrightarrow restrict\ (Inv\ (carrier\ G)\ h)\ (carrier\ G) \in hom\ G\ G$
⟨*proof*⟩

lemma *inv-BijGroup*:
$f \in Bij\ S \Longrightarrow m\text{-}inv\ (BijGroup\ S)\ f = (\lambda x \in S.\ (Inv\ S\ f)\ x)$

⟨*proof*⟩

lemma (**in** *group*) *subgroup-auto*:
  subgroup (*auto G*) (*BijGroup* (*carrier G*))
⟨*proof*⟩

theorem (**in** *group*) *AutoGroup*: *group* (*AutoGroup G*)
⟨*proof*⟩

end

theory *Ring* **imports** *FiniteProduct*
uses (*ringsimp.ML*) **begin**

8 Abelian Groups

record *'a ring* = *'a monoid* +
 zero :: *'a* ($\mathbf{0}_1$)
 add :: [*'a*, *'a*] => *'a* (**infixl** $\oplus_1$ *65*)

Derived operations.

constdefs (**structure** *R*)
 a-inv :: [(*'a*, *'m*) *ring-scheme*, *'a*] => *'a* ($\ominus_1$ - [*81*] *80*)
 a-inv R == *m-inv* (| *carrier* = *carrier R*, *mult* = *add R*, *one* = *zero R* |)

 a-minus :: [(*'a*, *'m*) *ring-scheme*, *'a*, *'a*] => *'a* (**infixl** $\ominus_1$ *65*)
 [| $x \in$ *carrier R*; $y \in$ *carrier R* |] ==> $x \ominus y$ == $x \oplus (\ominus\ y)$

locale *abelian-monoid* =
 fixes *G* (**structure**)
 assumes *a-comm-monoid*:
  comm-monoid (| *carrier* = *carrier G*, *mult* = *add G*, *one* = *zero G* |)

The following definition is redundant but simple to use.

locale *abelian-group* = *abelian-monoid* +
 assumes *a-comm-group*:
  comm-group (| *carrier* = *carrier G*, *mult* = *add G*, *one* = *zero G* |)

8.1 Basic Properties

lemma *abelian-monoidI*:
 fixes *R* (**structure**)
 assumes *a-closed*:
  !!*x y*. [| $x \in$ *carrier R*; $y \in$ *carrier R* |] ==> $x \oplus y \in$ *carrier R*
 and *zero-closed*: $\mathbf{0} \in$ *carrier R*
 and *a-assoc*:

$!!x\ y\ z.\ [|\ x \in carrier\ R;\ y \in carrier\ R;\ z \in carrier\ R\ |] ==>$
$(x \oplus y) \oplus z = x \oplus (y \oplus z)$
and *l-zero*: $!!x.\ x \in carrier\ R ==> \mathbf{0} \oplus x = x$
and *a-comm*:
$!!x\ y.\ [|\ x \in carrier\ R;\ y \in carrier\ R\ |] ==> x \oplus y = y \oplus x$
shows *abelian-monoid R*
⟨*proof*⟩

lemma *abelian-groupI*:
fixes *R* (**structure**)
assumes *a-closed*:
$!!x\ y.\ [|\ x \in carrier\ R;\ y \in carrier\ R\ |] ==> x \oplus y \in carrier\ R$
and *zero-closed*: $zero\ R \in carrier\ R$
and *a-assoc*:
$!!x\ y\ z.\ [|\ x \in carrier\ R;\ y \in carrier\ R;\ z \in carrier\ R\ |] ==>$
$(x \oplus y) \oplus z = x \oplus (y \oplus z)$
and *a-comm*:
$!!x\ y.\ [|\ x \in carrier\ R;\ y \in carrier\ R\ |] ==> x \oplus y = y \oplus x$
and *l-zero*: $!!x.\ x \in carrier\ R ==> \mathbf{0} \oplus x = x$
and *l-inv-ex*: $!!x.\ x \in carrier\ R ==> EX\ y : carrier\ R.\ y \oplus x = \mathbf{0}$
shows *abelian-group R*
⟨*proof*⟩

lemma (**in** *abelian-monoid*) *a-monoid*:
$monoid\ (|\ carrier = carrier\ G,\ mult = add\ G,\ one = zero\ G\ |)$
⟨*proof*⟩

lemma (**in** *abelian-group*) *a-group*:
$group\ (|\ carrier = carrier\ G,\ mult = add\ G,\ one = zero\ G\ |)$
⟨*proof*⟩

lemmas *monoid-record-simps* = *partial-object.simps monoid.simps*

lemma (**in** *abelian-monoid*) *a-closed* [*intro*, *simp*]:
$[\![\ x \in carrier\ G;\ y \in carrier\ G\ ]\!] \Longrightarrow x \oplus y \in carrier\ G$
⟨*proof*⟩

lemma (**in** *abelian-monoid*) *zero-closed* [*intro*, *simp*]:
$\mathbf{0} \in carrier\ G$
⟨*proof*⟩

lemma (**in** *abelian-group*) *a-inv-closed* [*intro*, *simp*]:
$x \in carrier\ G ==> \ominus\ x \in carrier\ G$
⟨*proof*⟩

lemma (**in** *abelian-group*) *minus-closed* [*intro*, *simp*]:
$[|\ x \in carrier\ G;\ y \in carrier\ G\ |] ==> x \ominus y \in carrier\ G$
⟨*proof*⟩

lemma (**in** *abelian-group*) *a-l-cancel* [*simp*]:
[| $x \in$ *carrier* G; $y \in$ *carrier* G; $z \in$ *carrier* G |] ==>
$(x \oplus y = x \oplus z) = (y = z)$
⟨*proof*⟩

lemma (**in** *abelian-group*) *a-r-cancel* [*simp*]:
[| $x \in$ *carrier* G; $y \in$ *carrier* G; $z \in$ *carrier* G |] ==>
$(y \oplus x = z \oplus x) = (y = z)$
⟨*proof*⟩

lemma (**in** *abelian-monoid*) *a-assoc*:
$[\![x \in \mathit{carrier}\ G;\ y \in \mathit{carrier}\ G;\ z \in \mathit{carrier}\ G]\!] \Longrightarrow$
$(x \oplus y) \oplus z = x \oplus (y \oplus z)$
⟨*proof*⟩

lemma (**in** *abelian-monoid*) *l-zero* [*simp*]:
$x \in$ *carrier* G ==> $\mathbf{0} \oplus x = x$
⟨*proof*⟩

lemma (**in** *abelian-group*) *l-neg*:
$x \in$ *carrier* G ==> $\ominus\ x \oplus x = \mathbf{0}$
⟨*proof*⟩

lemma (**in** *abelian-monoid*) *a-comm*:
$[\![x \in \mathit{carrier}\ G;\ y \in \mathit{carrier}\ G]\!] \Longrightarrow x \oplus y = y \oplus x$
⟨*proof*⟩

lemma (**in** *abelian-monoid*) *a-lcomm*:
$[\![x \in \mathit{carrier}\ G;\ y \in \mathit{carrier}\ G;\ z \in \mathit{carrier}\ G]\!] \Longrightarrow$
$x \oplus (y \oplus z) = y \oplus (x \oplus z)$
⟨*proof*⟩

lemma (**in** *abelian-monoid*) *r-zero* [*simp*]:
$x \in$ *carrier* G ==> $x \oplus \mathbf{0} = x$
⟨*proof*⟩

lemma (**in** *abelian-group*) *r-neg*:
$x \in$ *carrier* G ==> $x \oplus (\ominus\ x) = \mathbf{0}$
⟨*proof*⟩

lemma (**in** *abelian-group*) *minus-zero* [*simp*]:
$\ominus\ \mathbf{0} = \mathbf{0}$
⟨*proof*⟩

lemma (**in** *abelian-group*) *minus-minus* [*simp*]:
$x \in$ *carrier* G ==> $\ominus\ (\ominus\ x) = x$
⟨*proof*⟩

lemma (**in** *abelian-group*) *a-inv-inj*:

inj-on (*a-inv* G) (*carrier* G)
⟨*proof*⟩

lemma (**in** *abelian-group*) *minus-add*:
[| $x \in$ *carrier* G; $y \in$ *carrier* G |] ==> $\ominus$ ($x \oplus y$) = $\ominus x \oplus \ominus y$
⟨*proof*⟩

lemma (**in** *abelian-group*) *minus-equality*:
[| $x \in$ *carrier* G; $y \in$ *carrier* G; $y \oplus x = \mathbf{0}$ |] ==> $\ominus x = y$
⟨*proof*⟩

lemma (**in** *abelian-monoid*) *minus-unique*:
[| $x \in$ *carrier* G; $y \in$ *carrier* G; $y' \in$ *carrier* G;
$y \oplus x = \mathbf{0}$; $x \oplus y' = \mathbf{0}$ |] ==> $y = y'$
⟨*proof*⟩

lemmas (**in** *abelian-monoid*) *a-ac* = *a-assoc a-comm a-lcomm*

Derive an *abelian-group* from a *comm-group*

lemma *comm-group-abelian-groupI*:
fixes G (**structure**)
assumes *cg*: *comm-group* (|*carrier* = *carrier* G, *mult* = *add* G, *one* = *zero* G|)
shows *abelian-group* G
⟨*proof*⟩

8.2 Sums over Finite Sets

This definition makes it easy to lift lemmas from *finprod*.

constdefs
finsum :: [('b, 'm) *ring-scheme*, 'a => 'b, 'a *set*] => 'b
finsum G f A == *finprod* (| *carrier* = *carrier* G,
mult = *add* G, *one* = *zero* G |) f A

syntax
-finsum :: *index* => *idt* => 'a *set* => 'b => 'b
((*3*$\bigoplus$--:-. -) [*1000*, *0*, *51*, *10*] *10*)
syntax (*xsymbols*)
-finsum :: *index* => *idt* => 'a *set* => 'b => 'b
((*3*$\bigoplus$--$\in$-. -) [*1000*, *0*, *51*, *10*] *10*)
syntax (*HTML* **output**)
-finsum :: *index* => *idt* => 'a *set* => 'b => 'b
((*3*$\bigoplus$--$\in$-. -) [*1000*, *0*, *51*, *10*] *10*)
translations
$\bigoplus_{\imath} i$:A. b == *finsum* $\diamond_{\imath}$ (%i. b) A
— Beware of argument permutation!

lemma (**in** *abelian-monoid*) *finsum-empty* [*simp*]:

finsum G f $\{\} = \mathbf{0}$
⟨*proof*⟩

lemma (**in** *abelian-monoid*) *finsum-insert* [*simp*]:
$[|$ *finite F*; $a \notin F$; $f \in F$ $->$ *carrier G*; $f\ a \in$ *carrier G* $|]$
$==>$ *finsum G f* (*insert a F*) $= f\ a \oplus$ *finsum G f F*
⟨*proof*⟩

lemma (**in** *abelian-monoid*) *finsum-zero* [*simp*]:
finite A $==>$ $(\bigoplus i{\in}A.\ \mathbf{0}) = \mathbf{0}$
⟨*proof*⟩

lemma (**in** *abelian-monoid*) *finsum-closed* [*simp*]:
fixes *A*
assumes *fin*: *finite A* **and** *f*: $f \in A$ $->$ *carrier G*
shows *finsum G f A* $\in$ *carrier G*
⟨*proof*⟩

lemma (**in** *abelian-monoid*) *finsum-Un-Int*:
$[|$ *finite A*; *finite B*; $g \in A$ $->$ *carrier G*; $g \in B$ $->$ *carrier G* $|]$ $==>$
finsum G g (*A Un B*) $\oplus$ *finsum G g* (*A Int B*) $=$
finsum G g A $\oplus$ *finsum G g B*
⟨*proof*⟩

lemma (**in** *abelian-monoid*) *finsum-Un-disjoint*:
$[|$ *finite A*; *finite B*; *A Int B* $= \{\}$;
$g \in A$ $->$ *carrier G*; $g \in B$ $->$ *carrier G* $|]$
$==>$ *finsum G g* (*A Un B*) $=$ *finsum G g A* $\oplus$ *finsum G g B*
⟨*proof*⟩

lemma (**in** *abelian-monoid*) *finsum-addf*:
$[|$ *finite A*; $f \in A$ $->$ *carrier G*; $g \in A$ $->$ *carrier G* $|]$ $==>$
finsum G (%*x*. $f\ x \oplus g\ x$) $A =$ (*finsum G f A* $\oplus$ *finsum G g A*)
⟨*proof*⟩

lemma (**in** *abelian-monoid*) *finsum-cong'*:
$[|$ $A = B$; $g : B$ $->$ *carrier G*;
!!*i*. $i : B$ $==>$ $f\ i = g\ i$ $|]$ $==>$ *finsum G f A* $=$ *finsum G g B*
⟨*proof*⟩

lemma (**in** *abelian-monoid*) *finsum-0* [*simp*]:
$f : \{0{::}nat\}$ $->$ *carrier G* $==>$ *finsum G f* $\{..0\} = f\ 0$
⟨*proof*⟩

lemma (**in** *abelian-monoid*) *finsum-Suc* [*simp*]:
$f : \{..Suc\ n\}$ $->$ *carrier G* $==>$
finsum G f $\{..Suc\ n\} = (f\ (Suc\ n) \oplus$ *finsum G f* $\{..n\})$
⟨*proof*⟩

lemma (**in** *abelian-monoid*) *finsum-Suc2*:
  $f : \{..Suc\ n\}$ $->$ *carrier* G $==>$
   finsum $G\ f\ \{..Suc\ n\} = ($*finsum* $G\ (\%i.\ f\ (Suc\ i))\ \{..n\} \oplus f\ 0)$
  ⟨*proof*⟩

lemma (**in** *abelian-monoid*) *finsum-add* [*simp*]:
  [| $f : \{..n\}$ $->$ *carrier* G; $g : \{..n\}$ $->$ *carrier* G |] $==>$
    finsum $G\ (\%i.\ f\ i \oplus g\ i)\ \{..n::nat\} =$
    finsum $G\ f\ \{..n\} \oplus$ *finsum* $G\ g\ \{..n\}$
  ⟨*proof*⟩

lemma (**in** *abelian-monoid*) *finsum-cong*:
  [| $A = B$; $f : B$ $->$ *carrier* G;
    $!!i.\ i : B$ $=simp=>$ $f\ i = g\ i$ |] $==>$ *finsum* $G\ f\ A =$ *finsum* $G\ g\ B$
  ⟨*proof*⟩

Usually, if this rule causes a failed congruence proof error, the reason is that the premise $g \in B$ $->$ *carrier* G cannot be shown. Adding *Pi-def* to the simpset is often useful.

9 The Algebraic Hierarchy of Rings

9.1 Basic Definitions

locale *ring* = *abelian-group* R + *monoid* R +
  assumes *l-distr*: [| $x \in$ *carrier* R; $y \in$ *carrier* R; $z \in$ *carrier* R |]
    $==>$ $(x \oplus y) \otimes z = x \otimes z \oplus y \otimes z$
   and *r-distr*: [| $x \in$ *carrier* R; $y \in$ *carrier* R; $z \in$ *carrier* R |]
    $==>$ $z \otimes (x \oplus y) = z \otimes x \oplus z \otimes y$

locale *cring* = *ring* + *comm-monoid* R

locale *domain* = *cring* +
  assumes *one-not-zero* [*simp*]: $\mathbf{1}$ ~= $\mathbf{0}$
   and *integral*: [| $a \otimes b = \mathbf{0}$; $a \in$ *carrier* R; $b \in$ *carrier* R |] $==>$
        $a = \mathbf{0}$ | $b = \mathbf{0}$

locale *field* = *domain* +
  assumes *field-Units*: *Units* R = *carrier* $R - \{\mathbf{0}\}$

9.2 Rings

lemma *ringI*:
  fixes R (**structure**)
  assumes *abelian-group*: *abelian-group* R
   and *monoid*: *monoid* R
   and *l-distr*: $!!x\ y\ z.$ [| $x \in$ *carrier* R; $y \in$ *carrier* R; $z \in$ *carrier* R |]
    $==>$ $(x \oplus y) \otimes z = x \otimes z \oplus y \otimes z$
   and *r-distr*: $!!x\ y\ z.$ [| $x \in$ *carrier* R; $y \in$ *carrier* R; $z \in$ *carrier* R |]

    ==> $z \otimes (x \oplus y) = z \otimes x \oplus z \otimes y$
  shows *ring R*
  ⟨*proof*⟩

lemma (**in** *ring*) *is-abelian-group*:
  abelian-group R
  ⟨*proof*⟩

lemma (**in** *ring*) *is-monoid*:
  monoid R
  ⟨*proof*⟩

lemma (**in** *ring*) *is-ring*:
  ring R
  ⟨*proof*⟩

lemmas *ring-record-simps* = *monoid-record-simps ring.simps*

lemma *cringI*:
  fixes *R* (**structure**)
  assumes *abelian-group*: *abelian-group R*
    and *comm-monoid*: *comm-monoid R*
    and *l-distr*: !!*x y z*. [| $x \in$ *carrier R*; $y \in$ *carrier R*; $z \in$ *carrier R* |]
      ==> $(x \oplus y) \otimes z = x \otimes z \oplus y \otimes z$
  shows *cring R*
⟨*proof*⟩

lemma (**in** *cring*) *is-comm-monoid*:
  comm-monoid R
  ⟨*proof*⟩

lemma (**in** *cring*) *is-cring*:
  cring R ⟨*proof*⟩

9.2.1 Normaliser for Rings

lemma (**in** *abelian-group*) *r-neg2*:
  [| $x \in$ *carrier G*; $y \in$ *carrier G* |] ==> $x \oplus (\ominus x \oplus y) = y$
⟨*proof*⟩

lemma (**in** *abelian-group*) *r-neg1*:
  [| $x \in$ *carrier G*; $y \in$ *carrier G* |] ==> $\ominus x \oplus (x \oplus y) = y$
⟨*proof*⟩

The following proofs are from Jacobson, Basic Algebra I, pp. 88–89

lemma (**in** *ring*) *l-null* [*simp*]:
  $x \in$ *carrier R* ==> $\mathbf{0} \otimes x = \mathbf{0}$
⟨*proof*⟩

lemma (**in** *ring*) *r-null* [*simp*]:
$x \in$ *carrier R* ==> $x \otimes \mathbf{0} = \mathbf{0}$
⟨*proof*⟩

lemma (**in** *ring*) *l-minus*:
[| $x \in$ *carrier R*; $y \in$ *carrier R* |] ==> $\ominus\, x \otimes y = \ominus\, (x \otimes y)$
⟨*proof*⟩

lemma (**in** *ring*) *r-minus*:
[| $x \in$ *carrier R*; $y \in$ *carrier R* |] ==> $x \otimes \ominus\, y = \ominus\, (x \otimes y)$
⟨*proof*⟩

lemma (**in** *abelian-group*) *minus-eq*:
[| $x \in$ *carrier G*; $y \in$ *carrier G* |] ==> $x \ominus y = x \oplus \ominus\, y$
⟨*proof*⟩

Setup algebra method: compute distributive normal form in locale contexts

⟨*ML*⟩

lemmas (**in** *ring*) *ring-simprules*
[*algebra ring zero R add R a-inv R a-minus R one R mult R*] =
a-closed zero-closed a-inv-closed minus-closed m-closed one-closed
a-assoc l-zero l-neg a-comm m-assoc l-one l-distr minus-eq
r-zero r-neg r-neg2 r-neg1 minus-add minus-minus minus-zero
a-lcomm r-distr l-null r-null l-minus r-minus

lemmas (**in** *cring*)
[*algebra del*: *ring zero R add R a-inv R a-minus R one R mult R*] =
\-

lemmas (**in** *cring*) *cring-simprules*
[*algebra add*: *cring zero R add R a-inv R a-minus R one R mult R*] =
a-closed zero-closed a-inv-closed minus-closed m-closed one-closed
a-assoc l-zero l-neg a-comm m-assoc l-one l-distr m-comm minus-eq
r-zero r-neg r-neg2 r-neg1 minus-add minus-minus minus-zero
a-lcomm m-lcomm r-distr l-null r-null l-minus r-minus

lemma (**in** *cring*) *nat-pow-zero*:
(*n*::*nat*) ~= *0* ==> $\mathbf{0}$ (^) $n = \mathbf{0}$
⟨*proof*⟩

lemma (**in** *ring*) *one-zeroD*:
assumes *onezero*: $\mathbf{1} = \mathbf{0}$
shows *carrier R* = $\{\mathbf{0}\}$
⟨*proof*⟩

lemma (**in** *ring*) *one-zeroI*:
assumes *carrzero*: *carrier R* = $\{\mathbf{0}\}$

shows $\mathbf{1} = \mathbf{0}$
⟨*proof*⟩

lemma (**in** *ring*) *one-zero*:
shows $(carrier\ R = \{\mathbf{0}\}) = (\mathbf{1} = \mathbf{0})$
⟨*proof*⟩

lemma (**in** *ring*) *one-not-zero*:
shows $(carrier\ R \neq \{\mathbf{0}\}) = (\mathbf{1} \neq \mathbf{0})$
⟨*proof*⟩

Two examples for use of method algebra

lemma
includes *ring* R + *cring* S
shows [| $a \in carrier\ R$; $b \in carrier\ R$; $c \in carrier\ S$; $d \in carrier\ S$ |] ==>
$a \oplus \ominus (a \oplus \ominus b) = b$ & $c \otimes_S d = d \otimes_S c$
⟨*proof*⟩

lemma
includes *cring*
shows [| $a \in carrier\ R$; $b \in carrier\ R$ |] ==> $a \ominus (a \ominus b) = b$
⟨*proof*⟩

9.2.2 Sums over Finite Sets

lemma (**in** *cring*) *finsum-ldistr*:
[| *finite* A; $a \in carrier\ R$; $f \in A$ −> *carrier* R |] ==>
finsum $R\ f\ A \otimes a$ = *finsum* R (%i. $f\ i \otimes a$) A
⟨*proof*⟩

lemma (**in** *cring*) *finsum-rdistr*:
[| *finite* A; $a \in carrier\ R$; $f \in A$ −> *carrier* R |] ==>
$a \otimes$ *finsum* $R\ f\ A$ = *finsum* R (%i. $a \otimes f\ i$) A
⟨*proof*⟩

9.3 Integral Domains

lemma (**in** *domain*) *zero-not-one* [*simp*]:
$\mathbf{0}$ ~= $\mathbf{1}$
⟨*proof*⟩

lemma (**in** *domain*) *integral-iff*:
[| $a \in carrier\ R$; $b \in carrier\ R$ |] ==> $(a \otimes b = \mathbf{0}) = (a = \mathbf{0} \mid b = \mathbf{0})$
⟨*proof*⟩

lemma (**in** *domain*) *m-lcancel*:
assumes *prem*: a ~= $\mathbf{0}$
and R: $a \in carrier\ R$ $b \in carrier\ R$ $c \in carrier\ R$
shows $(a \otimes b = a \otimes c) = (b = c)$

⟨*proof*⟩

lemma (**in** *domain*) *m-rcancel*:
  assumes *prem*: $a \sim= \mathbf{0}$
    and *R*: $a \in carrier\ R\ b \in carrier\ R\ c \in carrier\ R$
  shows *conc*: $(b \otimes a = c \otimes a) = (b = c)$
⟨*proof*⟩

9.4 Fields

Field would not need to be derived from domain, the properties for domain follow from the assumptions of field

lemma (**in** *cring*) *cring-fieldI*:
  assumes *field-Units*: $Units\ R = carrier\ R - \{\mathbf{0}\}$
  shows *field R*
⟨*proof*⟩

Another variant to show that something is a field

lemma (**in** *cring*) *cring-fieldI2*:
  assumes *notzero*: $\mathbf{0} \neq \mathbf{1}$
  and *invex*: $\bigwedge a.\ [\![a \in carrier\ R;\ a \neq \mathbf{0}]\!] \Longrightarrow \exists\, b \in carrier\ R.\ a \otimes b = \mathbf{1}$
  shows *field R*
  ⟨*proof*⟩

9.5 Morphisms

constdefs (**structure** *R S*)
  $ring\text{-}hom :: [('a, 'm)\ ring\text{-}scheme, ('b, 'n)\ ring\text{-}scheme] => ('a => 'b)\ set$
  $ring\text{-}hom\ R\ S == \{h.\ h \in carrier\ R -> carrier\ S\ \&$
    $(ALL\ x\ y.\ x \in carrier\ R\ \&\ y \in carrier\ R -->$
      $h\ (x \otimes y) = h\ x \otimes_S h\ y\ \&\ h\ (x \oplus y) = h\ x \oplus_S h\ y)\ \&$
    $h\ \mathbf{1} = \mathbf{1}_S\}$

lemma *ring-hom-memI*:
  fixes *R* (**structure**) **and** *S* (**structure**)
  assumes *hom-closed*: $!!x.\ x \in carrier\ R ==> h\ x \in carrier\ S$
    and *hom-mult*: $!!x\ y.\ [|\ x \in carrier\ R;\ y \in carrier\ R\ |] ==>$
      $h\ (x \otimes y) = h\ x \otimes_S h\ y$
    and *hom-add*: $!!x\ y.\ [|\ x \in carrier\ R;\ y \in carrier\ R\ |] ==>$
      $h\ (x \oplus y) = h\ x \oplus_S h\ y$
    and *hom-one*: $h\ \mathbf{1} = \mathbf{1}_S$
  shows $h \in ring\text{-}hom\ R\ S$
  ⟨*proof*⟩

lemma *ring-hom-closed*:
  $[|\ h \in ring\text{-}hom\ R\ S;\ x \in carrier\ R\ |] ==> h\ x \in carrier\ S$
  ⟨*proof*⟩

lemma *ring-hom-mult*:

fixes R (**structure**) **and** S (**structure**)
shows
$[| \ h \in \mathit{ring\text{-}hom}\ R\ S;\ x \in \mathit{carrier}\ R;\ y \in \mathit{carrier}\ R\ |] ==>$
$h\ (x \otimes y) = h\ x \otimes_S h\ y$
⟨*proof*⟩

lemma *ring-hom-add*:
fixes R (**structure**) **and** S (**structure**)
shows
$[| \ h \in \mathit{ring\text{-}hom}\ R\ S;\ x \in \mathit{carrier}\ R;\ y \in \mathit{carrier}\ R\ |] ==>$
$h\ (x \oplus y) = h\ x \oplus_S h\ y$
⟨*proof*⟩

lemma *ring-hom-one*:
fixes R (**structure**) **and** S (**structure**)
shows $h \in \mathit{ring\text{-}hom}\ R\ S ==> h\ \mathbf{1} = \mathbf{1}_S$
⟨*proof*⟩

locale *ring-hom-cring* = *cring* R + *cring* S +
fixes h
assumes *homh* [*simp*, *intro*]: $h \in \mathit{ring\text{-}hom}\ R\ S$
notes *hom-closed* [*simp*, *intro*] = *ring-hom-closed* [*OF homh*]
and *hom-mult* [*simp*] = *ring-hom-mult* [*OF homh*]
and *hom-add* [*simp*] = *ring-hom-add* [*OF homh*]
and *hom-one* [*simp*] = *ring-hom-one* [*OF homh*]

lemma (**in** *ring-hom-cring*) *hom-zero* [*simp*]:
$h\ \mathbf{0} = \mathbf{0}_S$
⟨*proof*⟩

lemma (**in** *ring-hom-cring*) *hom-a-inv* [*simp*]:
$x \in \mathit{carrier}\ R ==> h\ (\ominus\ x) = \ominus_S\ h\ x$
⟨*proof*⟩

lemma (**in** *ring-hom-cring*) *hom-finsum* [*simp*]:
$[| \ \mathit{finite}\ A;\ f \in A\ -> \mathit{carrier}\ R\ |] ==>$
$h\ (\mathit{finsum}\ R\ f\ A) = \mathit{finsum}\ S\ (h\ o\ f)\ A$
⟨*proof*⟩

lemma (**in** *ring-hom-cring*) *hom-finprod*:
$[| \ \mathit{finite}\ A;\ f \in A\ -> \mathit{carrier}\ R\ |] ==>$
$h\ (\mathit{finprod}\ R\ f\ A) = \mathit{finprod}\ S\ (h\ o\ f)\ A$
⟨*proof*⟩

declare *ring-hom-cring.hom-finprod* [*simp*]

lemma *id-ring-hom* [*simp*]:
$\mathit{id} \in \mathit{ring\text{-}hom}\ R\ R$
⟨*proof*⟩

end

theory *Module* **imports** *Ring* **begin**

10 Modules over an Abelian Group

10.1 Definitions

record ('*a*, '*b*) *module* = '*b ring* +
smult :: ['*a*, '*b*] => '*b* (**infixl** ⊙ı *70*)

locale *module* = *cring R* + *abelian-group M* +
assumes *smult-closed* [*simp*, *intro*]:
$[| a \in carrier\ R; x \in carrier\ M |] ==> a \odot_M x \in carrier\ M$
and *smult-l-distr*:
$[| a \in carrier\ R; b \in carrier\ R; x \in carrier\ M |] ==>$
$(a \oplus b) \odot_M x = a \odot_M x \oplus_M b \odot_M x$
and *smult-r-distr*:
$[| a \in carrier\ R; x \in carrier\ M; y \in carrier\ M |] ==>$
$a \odot_M (x \oplus_M y) = a \odot_M x \oplus_M a \odot_M y$
and *smult-assoc1*:
$[| a \in carrier\ R; b \in carrier\ R; x \in carrier\ M |] ==>$
$(a \otimes b) \odot_M x = a \odot_M (b \odot_M x)$
and *smult-one* [*simp*]:
$x \in carrier\ M ==> \mathbf{1} \odot_M x = x$

locale *algebra* = *module R M* + *cring M* +
assumes *smult-assoc2*:
$[| a \in carrier\ R; x \in carrier\ M; y \in carrier\ M |] ==>$
$(a \odot_M x) \otimes_M y = a \odot_M (x \otimes_M y)$

lemma *moduleI*:
fixes *R* (**structure**) **and** *M* (**structure**)
assumes *cring*: *cring R*
and *abelian-group*: *abelian-group M*
and *smult-closed*:
$!!a\ x.\ [| a \in carrier\ R; x \in carrier\ M |] ==> a \odot_M x \in carrier\ M$
and *smult-l-distr*:
$!!a\ b\ x.\ [| a \in carrier\ R; b \in carrier\ R; x \in carrier\ M |] ==>$
$(a \oplus b) \odot_M x = (a \odot_M x) \oplus_M (b \odot_M x)$
and *smult-r-distr*:
$!!a\ x\ y.\ [| a \in carrier\ R; x \in carrier\ M; y \in carrier\ M |] ==>$
$a \odot_M (x \oplus_M y) = (a \odot_M x) \oplus_M (a \odot_M y)$
and *smult-assoc1*:
$!!a\ b\ x.\ [| a \in carrier\ R; b \in carrier\ R; x \in carrier\ M |] ==>$
$(a \otimes b) \odot_M x = a \odot_M (b \odot_M x)$

and *smult-one*:
!!x. $x \in$ *carrier* M ==> $\mathbf{1} \odot_M x = x$
shows *module R M*
⟨*proof*⟩

lemma *algebraI*:
fixes R (**structure**) **and** M (**structure**)
assumes *R-cring*: *cring R*
and *M-cring*: *cring M*
and *smult-closed*:
!!a x. [| $a \in$ *carrier* R; $x \in$ *carrier* M |] ==> $a \odot_M x \in$ *carrier* M
and *smult-l-distr*:
!!a b x. [| $a \in$ *carrier* R; $b \in$ *carrier* R; $x \in$ *carrier* M |] ==>
$(a \oplus b) \odot_M x = (a \odot_M x) \oplus_M (b \odot_M x)$
and *smult-r-distr*:
!!a x y. [| $a \in$ *carrier* R; $x \in$ *carrier* M; $y \in$ *carrier* M |] ==>
$a \odot_M (x \oplus_M y) = (a \odot_M x) \oplus_M (a \odot_M y)$
and *smult-assoc1*:
!!a b x. [| $a \in$ *carrier* R; $b \in$ *carrier* R; $x \in$ *carrier* M |] ==>
$(a \otimes b) \odot_M x = a \odot_M (b \odot_M x)$
and *smult-one*:
!!x. $x \in$ *carrier* M ==> (*one* R) $\odot_M x = x$
and *smult-assoc2*:
!!a x y. [| $a \in$ *carrier* R; $x \in$ *carrier* M; $y \in$ *carrier* M |] ==>
$(a \odot_M x) \otimes_M y = a \odot_M (x \otimes_M y)$
shows *algebra R M*
⟨*proof*⟩

lemma (**in** *algebra*) *R-cring*:
cring R
⟨*proof*⟩

lemma (**in** *algebra*) *M-cring*:
cring M
⟨*proof*⟩

lemma (**in** *algebra*) *module*:
module R M
⟨*proof*⟩

10.2 Basic Properties of Algebras

lemma (**in** *algebra*) *smult-l-null* [*simp*]:
$x \in$ *carrier* M ==> $\mathbf{0} \odot_M x = \mathbf{0}_M$
⟨*proof*⟩

lemma (**in** *algebra*) *smult-r-null* [*simp*]:
$a \in$ *carrier* R ==> $a \odot_M \mathbf{0}_M = \mathbf{0}_M$
⟨*proof*⟩

lemma (**in** *algebra*) *smult-l-minus*:
  $[| \; a \in carrier \; R; \; x \in carrier \; M \; |] \; ==> (\ominus a) \odot_M x = \ominus_M (a \odot_M x)$
⟨*proof*⟩

lemma (**in** *algebra*) *smult-r-minus*:
  $[| \; a \in carrier \; R; \; x \in carrier \; M \; |] \; ==> a \odot_M (\ominus_M x) = \ominus_M (a \odot_M x)$
⟨*proof*⟩

end

theory *UnivPoly* **imports** *Module* **begin**

11 Univariate Polynomials

Polynomials are formalised as modules with additional operations for extracting coefficients from polynomials and for obtaining monomials from coefficients and exponents (record *up-ring*). The carrier set is a set of bounded functions from Nat to the coefficient domain. Bounded means that these functions return zero above a certain bound (the degree). There is a chapter on the formalisation of polynomials in the PhD thesis [1], which was implemented with axiomatic type classes. This was later ported to Locales.

11.1 The Constructor for Univariate Polynomials

Functions with finite support.

locale *bound* =
  fixes $z :: 'a$
    and $n :: nat$
    and $f :: nat => 'a$
  assumes *bound*: $!!m. \; n < m \Longrightarrow f \; m = z$

declare *bound.intro* [*intro*!]
  and *bound.bound* [*dest*]

lemma *bound-below*:
  assumes *bound*: *bound z m f* **and** *nonzero*: $f \; n \neq z$ **shows** $n \leq m$
⟨*proof*⟩

record ('a, 'p) *up-ring* = ('a, 'p) *module* +
  monom :: ['a, *nat*] => 'p
  coeff :: ['p, *nat*] => 'a

constdefs (**structure** *R*)
  up :: ('a, 'm) *ring-scheme* => (*nat* => 'a) *set*

$up\ R == \{f.\ f \in UNIV\ {-}{>}\ carrier\ R\ \&\ (EX\ n.\ bound\ \mathbf{0}\ n\ f)\}$
$UP :: ('a,\ 'm)\ ring\text{-}scheme\ {=}{>}\ ('a,\ nat\ {=}{>}\ 'a)\ up\text{-}ring$
$UP\ R == (|$
$carrier = up\ R,$
$mult = (\%p{:}up\ R.\ \%q{:}up\ R.\ \%n.\ \bigoplus i \in \{..n\}.\ p\ i \otimes q\ (n-i)),$
$one = (\%i.\ if\ i{=}0\ then\ \mathbf{1}\ else\ \mathbf{0}),$
$zero = (\%i.\ \mathbf{0}),$
$add = (\%p{:}up\ R.\ \%q{:}up\ R.\ \%i.\ p\ i \oplus q\ i),$
$smult = (\%a{:}carrier\ R.\ \%p{:}up\ R.\ \%i.\ a \otimes p\ i),$
$monom = (\%a{:}carrier\ R.\ \%n\ i.\ if\ i{=}n\ then\ a\ else\ \mathbf{0}),$
$coeff = (\%p{:}up\ R.\ \%n.\ p\ n)\ |)$

Properties of the set of polynomials *up*.

lemma *mem-upI* [*intro*]:
$[|\ !!n.\ f\ n \in carrier\ R;\ EX\ n.\ bound\ (zero\ R)\ n\ f\ |] ==> f \in up\ R$
⟨*proof*⟩

lemma *mem-upD* [*dest*]:
$f \in up\ R ==> f\ n \in carrier\ R$
⟨*proof*⟩

lemma (**in** *cring*) *bound-upD* [*dest*]:
$f \in up\ R ==> EX\ n.\ bound\ \mathbf{0}\ n\ f$
⟨*proof*⟩

lemma (**in** *cring*) *up-one-closed*:
$(\%n.\ if\ n = 0\ then\ \mathbf{1}\ else\ \mathbf{0}) \in up\ R$
⟨*proof*⟩

lemma (**in** *cring*) *up-smult-closed*:
$[|\ a \in carrier\ R;\ p \in up\ R\ |] ==> (\%i.\ a \otimes p\ i) \in up\ R$
⟨*proof*⟩

lemma (**in** *cring*) *up-add-closed*:
$[|\ p \in up\ R;\ q \in up\ R\ |] ==> (\%i.\ p\ i \oplus q\ i) \in up\ R$
⟨*proof*⟩

lemma (**in** *cring*) *up-a-inv-closed*:
$p \in up\ R ==> (\%i.\ \ominus\ (p\ i)) \in up\ R$
⟨*proof*⟩

lemma (**in** *cring*) *up-mult-closed*:
$[|\ p \in up\ R;\ q \in up\ R\ |] ==>$
$(\%n.\ \bigoplus i \in \{..n\}.\ p\ i \otimes q\ (n-i)) \in up\ R$
⟨*proof*⟩

11.2 Effect of Operations on Coefficients

locale *UP* =

fixes R (**structure**) **and** P (**structure**)
defines *P-def*: $P == UP\ R$

locale *UP-cring* = *UP* + *cring R*

locale *UP-domain* = *UP-cring* + *domain R*

Temporarily declare $P \equiv UP\ R$ as simp rule.

declare (**in** *UP*) *P-def* [*simp*]

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

lemma (**in** *UP-cring*) *coeff-zero* [*simp*]:
$coeff\ P\ \mathbf{0}_P\ n = \mathbf{0}$
⟨*proof*⟩

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

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

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

lemma (**in** *UP-cring*) *coeff-mult* [*simp*]:
$[|\ p \in carrier\ P;\ q \in carrier\ P\ |] ==>$
$coeff\ P\ (p \otimes_P q)\ n = (\bigoplus i \in \{..n\}.\ coeff\ P\ p\ i \otimes coeff\ P\ q\ (n-i))$
⟨*proof*⟩

lemma (**in** *UP*) *up-eqI*:
assumes *prem*: $!!n.\ coeff\ P\ p\ n = coeff\ P\ q\ n$
and R: $p \in carrier\ P\ q \in carrier\ P$
shows $p = q$
⟨*proof*⟩

11.3 Polynomials Form a Commutative Ring.

Operations are closed over P.

lemma (**in** *UP-cring*) *UP-mult-closed* [*simp*]:
$[|\ p \in carrier\ P;\ q \in carrier\ P\ |] ==> p \otimes_P q \in carrier\ P$

⟨*proof*⟩

lemma (**in** *UP-cring*) *UP-one-closed* [*simp*]:
$\mathbf{1}_P \in$ *carrier P*
⟨*proof*⟩

lemma (**in** *UP-cring*) *UP-zero-closed* [*intro*, *simp*]:
$\mathbf{0}_P \in$ *carrier P*
⟨*proof*⟩

lemma (**in** *UP-cring*) *UP-a-closed* [*intro*, *simp*]:
[| $p \in$ *carrier P*; $q \in$ *carrier P* |] ==> $p \oplus_P q \in$ *carrier P*
⟨*proof*⟩

lemma (**in** *UP-cring*) *monom-closed* [*simp*]:
$a \in$ *carrier R* ==> *monom P a n* $\in$ *carrier P*
⟨*proof*⟩

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

lemma (**in** *UP*) *coeff-closed* [*simp*]:
$p \in$ *carrier P* ==> *coeff P p n* $\in$ *carrier R*
⟨*proof*⟩

declare (**in** *UP*) *P-def* [*simp del*]

Algebraic ring properties

lemma (**in** *UP-cring*) *UP-a-assoc*:
assumes *R*: $p \in$ *carrier P* $q \in$ *carrier P* $r \in$ *carrier P*
shows $(p \oplus_P q) \oplus_P r = p \oplus_P (q \oplus_P r)$
⟨*proof*⟩

lemma (**in** *UP-cring*) *UP-l-zero* [*simp*]:
assumes *R*: $p \in$ *carrier P*
shows $\mathbf{0}_P \oplus_P p = p$
⟨*proof*⟩

lemma (**in** *UP-cring*) *UP-l-neg-ex*:
assumes *R*: $p \in$ *carrier P*
shows *EX q* : *carrier P*. $q \oplus_P p = \mathbf{0}_P$
⟨*proof*⟩

lemma (**in** *UP-cring*) *UP-a-comm*:
assumes *R*: $p \in$ *carrier P* $q \in$ *carrier P*
shows $p \oplus_P q = q \oplus_P p$
⟨*proof*⟩

lemma (**in** *UP-cring*) *UP-m-assoc*:
assumes *R*: $p \in carrier\ P\ q \in carrier\ P\ r \in carrier\ P$
shows $(p \otimes_P q) \otimes_P r = p \otimes_P (q \otimes_P r)$
⟨*proof*⟩

lemma (**in** *UP-cring*) *UP-l-one* [*simp*]:
assumes *R*: $p \in carrier\ P$
shows $\mathbf{1}_P \otimes_P p = p$
⟨*proof*⟩

lemma (**in** *UP-cring*) *UP-l-distr*:
assumes *R*: $p \in carrier\ P\ q \in carrier\ P\ r \in carrier\ P$
shows $(p \oplus_P q) \otimes_P r = (p \otimes_P r) \oplus_P (q \otimes_P r)$
⟨*proof*⟩

lemma (**in** *UP-cring*) *UP-m-comm*:
assumes *R*: $p \in carrier\ P\ q \in carrier\ P$
shows $p \otimes_P q = q \otimes_P p$
⟨*proof*⟩

theorem (**in** *UP-cring*) *UP-cring*:
cring P
⟨*proof*⟩

lemma (**in** *UP-cring*) *UP-ring*:

ring P
⟨*proof*⟩

lemma (**in** *UP-cring*) *UP-a-inv-closed* [*intro*, *simp*]:
$p \in carrier\ P$ ==> $\ominus_P\ p \in carrier\ P$
⟨*proof*⟩

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

Interpretation of lemmas from *cring*. Saves lifting 43 lemmas manually.

interpretation *UP-cring* < *cring P*
⟨*proof*⟩

11.4 Polynomials Form an Algebra

lemma (**in** *UP-cring*) *UP-smult-l-distr*:
[| $a \in carrier\ R$; $b \in carrier\ R$; $p \in carrier\ P$ |] ==>
$(a \oplus b) \odot_P p = a \odot_P p \oplus_P b \odot_P p$
⟨*proof*⟩

lemma (**in** *UP-cring*) *UP-smult-r-distr*:
  [| $a \in$ *carrier* R; $p \in$ *carrier* P; $q \in$ *carrier* P |] ==>
  $a \odot_P (p \oplus_P q) = a \odot_P p \oplus_P a \odot_P q$
  ⟨*proof*⟩

lemma (**in** *UP-cring*) *UP-smult-assoc1*:
    [| $a \in$ *carrier* R; $b \in$ *carrier* R; $p \in$ *carrier* P |] ==>
    $(a \otimes b) \odot_P p = a \odot_P (b \odot_P p)$
  ⟨*proof*⟩

lemma (**in** *UP-cring*) *UP-smult-one* [*simp*]:
    $p \in$ *carrier* P ==> $\mathbf{1} \odot_P p = p$
  ⟨*proof*⟩

lemma (**in** *UP-cring*) *UP-smult-assoc2*:
  [| $a \in$ *carrier* R; $p \in$ *carrier* P; $q \in$ *carrier* P |] ==>
  $(a \odot_P p) \otimes_P q = a \odot_P (p \otimes_P q)$
  ⟨*proof*⟩

Interpretation of lemmas from *algebra*.

lemma (**in** *cring*) *cring*:
  cring R
  ⟨*proof*⟩

lemma (**in** *UP-cring*) *UP-algebra*:
  algebra R P
  ⟨*proof*⟩

interpretation *UP-cring* < *algebra R P*
  ⟨*proof*⟩

11.5 Further Lemmas Involving Monomials

lemma (**in** *UP-cring*) *monom-zero* [*simp*]:
  monom P $\mathbf{0}$ $n = \mathbf{0}_P$
  ⟨*proof*⟩

lemma (**in** *UP-cring*) *monom-mult-is-smult*:
  assumes R: $a \in$ *carrier* R $p \in$ *carrier* P
  shows *monom* P a $0 \otimes_P p = a \odot_P p$
⟨*proof*⟩

lemma (**in** *UP-cring*) *monom-add* [*simp*]:
  [| $a \in$ *carrier* R; $b \in$ *carrier* R |] ==>
  monom P $(a \oplus b)$ n = *monom* P a $n \oplus_P$ *monom* P b n
  ⟨*proof*⟩

lemma (**in** *UP-cring*) *monom-one-Suc*:
  monom P $\mathbf{1}$ (*Suc* n) = *monom* P $\mathbf{1}$ $n \otimes_P$ *monom* P $\mathbf{1}$ 1

⟨*proof*⟩

lemma (**in** *UP-cring*) *monom-mult-smult*:
[| $a \in carrier\ R$; $b \in carrier\ R$ |] ==> $monom\ P\ (a \otimes b)\ n = a \odot_P monom\ P\ b\ n$
⟨*proof*⟩

lemma (**in** *UP-cring*) *monom-one* [*simp*]:
$monom\ P\ \mathbf{1}\ 0 = \mathbf{1}_P$
⟨*proof*⟩

lemma (**in** *UP-cring*) *monom-one-mult*:
$monom\ P\ \mathbf{1}\ (n + m) = monom\ P\ \mathbf{1}\ n \otimes_P monom\ P\ \mathbf{1}\ m$
⟨*proof*⟩

lemma (**in** *UP-cring*) *monom-mult* [*simp*]:
assumes *R*: $a \in carrier\ R$ $b \in carrier\ R$
shows $monom\ P\ (a \otimes b)\ (n + m) = monom\ P\ a\ n \otimes_P monom\ P\ b\ m$
⟨*proof*⟩

lemma (**in** *UP-cring*) *monom-a-inv* [*simp*]:
$a \in carrier\ R$ ==> $monom\ P\ (\ominus\ a)\ n = \ominus_P\ monom\ P\ a\ n$
⟨*proof*⟩

lemma (**in** *UP-cring*) *monom-inj*:
inj-on (%*a*. *monom P a n*) (*carrier R*)
⟨*proof*⟩

11.6 The Degree Function

constdefs (**structure** *R*)
deg :: [('*a*, '*m*) *ring-scheme*, *nat* => '*a*] => *nat*
deg R p == *LEAST n*. *bound* **0** *n* (*coeff* (*UP R*) *p*)

lemma (**in** *UP-cring*) *deg-aboveI*:
[| (!!*m*. $n < m$ ==> *coeff P p m* = **0**); $p \in carrier\ P$ |] ==> *deg R p* <= *n*
⟨*proof*⟩

lemma (**in** *UP-cring*) *deg-aboveD*:
assumes *deg R p* < *m* **and** $p \in carrier\ P$
shows *coeff P p m* = **0**
⟨*proof*⟩

lemma (**in** *UP-cring*) *deg-belowI*:
assumes *non-zero*: *n* ~= *0* ==> *coeff P p n* ~= **0**
and *R*: $p \in carrier\ P$
shows *n* <= *deg R p*

— Logically, this is a slightly stronger version of *deg-aboveD*
⟨*proof*⟩

lemma (**in** *UP-cring*) *lcoeff-nonzero-deg*:
assumes *deg*: *deg R p* ~= *0* **and** *R*: $p \in$ *carrier P*
shows *coeff P p* (*deg R p*) ~= **0**
⟨*proof*⟩

lemma (**in** *UP-cring*) *lcoeff-nonzero-nonzero*:
assumes *deg*: *deg R p* = *0* **and** *nonzero*: *p* ~= $\mathbf{0}_P$ **and** *R*: $p \in$ *carrier P*
shows *coeff P p 0* ~= **0**
⟨*proof*⟩

lemma (**in** *UP-cring*) *lcoeff-nonzero*:
assumes *neq*: *p* ~= $\mathbf{0}_P$ **and** *R*: $p \in$ *carrier P*
shows *coeff P p* (*deg R p*) ~= **0**
⟨*proof*⟩

lemma (**in** *UP-cring*) *deg-eqI*:
[| !!*m*. *n* < *m* ==> *coeff P p m* = **0**;
!!*n*. *n* ~= *0* ==> *coeff P p n* ~= **0**; $p \in$ *carrier P* |] ==> *deg R p* = *n*
⟨*proof*⟩

Degree and polynomial operations

lemma (**in** *UP-cring*) *deg-add* [*simp*]:
assumes *R*: $p \in$ *carrier P* $q \in$ *carrier P*
shows *deg R* ($p \oplus_P q$) <= *max* (*deg R p*) (*deg R q*)
⟨*proof*⟩

lemma (**in** *UP-cring*) *deg-monom-le*:
$a \in$ *carrier R* ==> *deg R* (*monom P a n*) <= *n*
⟨*proof*⟩

lemma (**in** *UP-cring*) *deg-monom* [*simp*]:
[| *a* ~= **0**; $a \in$ *carrier R* |] ==> *deg R* (*monom P a n*) = *n*
⟨*proof*⟩

lemma (**in** *UP-cring*) *deg-const* [*simp*]:
assumes *R*: $a \in$ *carrier R* **shows** *deg R* (*monom P a 0*) = *0*
⟨*proof*⟩

lemma (**in** *UP-cring*) *deg-zero* [*simp*]:
deg R $\mathbf{0}_P$ = *0*
⟨*proof*⟩

lemma (**in** *UP-cring*) *deg-one* [*simp*]:
deg R $\mathbf{1}_P$ = *0*
⟨*proof*⟩

lemma (**in** *UP-cring*) *deg-uminus* [*simp*]:
assumes *R*: $p \in carrier\ P$ **shows** $deg\ R\ (\ominus_P\ p) = deg\ R\ p$
⟨*proof*⟩

lemma (**in** *UP-domain*) *deg-smult-ring*:
$[|\ a \in carrier\ R;\ p \in carrier\ P\ |] ==>$
$deg\ R\ (a \odot_P\ p) <= (if\ a = \mathbf{0}\ then\ 0\ else\ deg\ R\ p)$
⟨*proof*⟩

lemma (**in** *UP-domain*) *deg-smult* [*simp*]:
assumes *R*: $a \in carrier\ R\ p \in carrier\ P$
shows $deg\ R\ (a \odot_P\ p) = (if\ a = \mathbf{0}\ then\ 0\ else\ deg\ R\ p)$
⟨*proof*⟩

lemma (**in** *UP-cring*) *deg-mult-cring*:
assumes *R*: $p \in carrier\ P\ q \in carrier\ P$
shows $deg\ R\ (p \otimes_P\ q) <= deg\ R\ p + deg\ R\ q$
⟨*proof*⟩

lemma (**in** *UP-domain*) *deg-mult* [*simp*]:
$[|\ p\ \tilde{}= \mathbf{0}_P;\ q\ \tilde{}= \mathbf{0}_P;\ p \in carrier\ P;\ q \in carrier\ P\ |] ==>$
$deg\ R\ (p \otimes_P\ q) = deg\ R\ p + deg\ R\ q$
⟨*proof*⟩

lemma (**in** *UP-cring*) *coeff-finsum*:
assumes *fin*: *finite A*
shows $p \in A\ {-}{>}\ carrier\ P ==>$
$coeff\ P\ (finsum\ P\ p\ A)\ k = (\bigoplus i \in A.\ coeff\ P\ (p\ i)\ k)$
⟨*proof*⟩

lemma (**in** *UP-cring*) *up-repr*:
assumes *R*: $p \in carrier\ P$
shows $(\bigoplus_P\ i \in \{..deg\ R\ p\}.\ monom\ P\ (coeff\ P\ p\ i)\ i) = p$
⟨*proof*⟩

lemma (**in** *UP-cring*) *up-repr-le*:
$[|\ deg\ R\ p <= n;\ p \in carrier\ P\ |] ==>$
$(\bigoplus_P\ i \in \{..n\}.\ monom\ P\ (coeff\ P\ p\ i)\ i) = p$
⟨*proof*⟩

11.7 Polynomials over Integral Domains

lemma *domainI*:
assumes *cring*: *cring R*
and *one-not-zero*: $one\ R\ \tilde{}= zero\ R$
and *integral*: $!!a\ b.\ [|\ mult\ R\ a\ b = zero\ R;\ a \in carrier\ R;$
$b \in carrier\ R\ |] ==> a = zero\ R\ |\ b = zero\ R$
shows *domain R*
⟨*proof*⟩

lemma (**in** *UP-domain*) *UP-one-not-zero*:
$\mathbf{1}_P$ ~= $\mathbf{0}_P$
⟨*proof*⟩

lemma (**in** *UP-domain*) *UP-integral*:
[| $p \otimes_P q = \mathbf{0}_P$; $p \in$ *carrier* P; $q \in$ *carrier* P |] ==> $p = \mathbf{0}_P$ | $q = \mathbf{0}_P$
⟨*proof*⟩

theorem (**in** *UP-domain*) *UP-domain*:
domain P
⟨*proof*⟩

Interpretation of theorems from *domain*.

interpretation *UP-domain* < *domain P*
⟨*proof*⟩

11.8 The Evaluation Homomorphism and Universal Property

theorem (**in** *cring*) *diagonal-sum*:
[| $f \in$ {..n + m::*nat*} -> *carrier R*; $g \in$ {..n + m} -> *carrier R* |] ==>
($\bigoplus k \in$ {..n + m}. $\bigoplus i \in$ {..k}. $f\ i \otimes g\ (k - i)$) =
($\bigoplus k \in$ {..n + m}. $\bigoplus i \in$ {..$n + m - k$}. $f\ k \otimes g\ i$)
⟨*proof*⟩

lemma (**in** *abelian-monoid*) *boundD-carrier*:
[| *bound* $\mathbf{0}$ n f; $n < m$ |] ==> $f\ m \in$ *carrier G*
⟨*proof*⟩

theorem (**in** *cring*) *cauchy-product*:
assumes *bf*: *bound* $\mathbf{0}$ n f **and** *bg*: *bound* $\mathbf{0}$ m g
and *Rf*: $f \in$ {..n} -> *carrier R* **and** *Rg*: $g \in$ {..m} -> *carrier R*
shows ($\bigoplus k \in$ {..n + m}. $\bigoplus i \in$ {..k}. $f\ i \otimes g\ (k - i)$) =
($\bigoplus i \in$ {..n}. $f\ i$) $\otimes$ ($\bigoplus i \in$ {..m}. $g\ i$)
⟨*proof*⟩

lemma (**in** *UP-cring*) *const-ring-hom*:
(%*a*. *monom P a 0*) $\in$ *ring-hom R P*
⟨*proof*⟩

constdefs (**structure** *S*)
eval :: [('*a*, '*m*) *ring-scheme*, ('*b*, '*n*) *ring-scheme*,
'*a* => '*b*, '*b*, *nat* => '*a*] => '*b*
eval R S phi s == $\lambda p \in$ *carrier* (*UP R*).
$\bigoplus i \in$ {..*deg R p*}. *phi* (*coeff* (*UP R*) *p i*) $\otimes$ *s* (^) *i*

lemma (**in** *UP*) *eval-on-carrier*:

fixes S (**structure**)
shows $p \in carrier\ P$ ==>
$eval\ R\ S\ phi\ s\ p = (\bigoplus_S\ i \in \{..deg\ R\ p\}.\ phi\ (coeff\ P\ p\ i) \otimes_S s\ (\hat{\ })_S\ i)$
⟨*proof*⟩

lemma (**in** *UP*) *eval-extensional*:
eval R S phi p $\in$ *extensional* (*carrier P*)
⟨*proof*⟩

The universal property of the polynomial ring

locale *UP-pre-univ-prop* = *ring-hom-cring R S h* + *UP-cring R P*

locale *UP-univ-prop* = *UP-pre-univ-prop* +
fixes *s* **and** *Eval*
assumes *indet-img-carrier* [*simp*, *intro*]: *s* $\in$ *carrier S*
defines *Eval-def*: *Eval* == *eval R S h s*

theorem (**in** *UP-pre-univ-prop*) *eval-ring-hom*:
assumes *S*: *s* $\in$ *carrier S*
shows *eval R S h s* $\in$ *ring-hom P S*
⟨*proof*⟩

Interpretation of ring homomorphism lemmas.

interpretation *UP-univ-prop* < *ring-hom-cring P S Eval*
⟨*proof*⟩

Further properties of the evaluation homomorphism.

The following lemma could be proved in *UP-cring* with the additional assumption that h is closed.

lemma (**in** *UP-pre-univ-prop*) *eval-const*:
[| *s* $\in$ *carrier S*; *r* $\in$ *carrier R* |] ==> *eval R S h s* (*monom P r 0*) = *h r*
⟨*proof*⟩

The following proof is complicated by the fact that in arbitrary rings one might have $\mathbf{1}_R = \mathbf{0}_R$.

lemma (**in** *UP-pre-univ-prop*) *eval-monom1*:
assumes *S*: *s* $\in$ *carrier S*
shows *eval R S h s* (*monom P* **1** *1*) = *s*
⟨*proof*⟩

lemma (**in** *UP-cring*) *monom-pow*:
assumes *R*: *a* $\in$ *carrier R*
shows $(monom\ P\ a\ n)\ (\hat{\ })_P\ m = monom\ P\ (a\ (\hat{\ })\ m)\ (n * m)$
⟨*proof*⟩

lemma (**in** *ring-hom-cring*) *hom-pow* [*simp*]:
$x \in carrier\ R$ ==> $h\ (x\ (\hat{\ })\ n) = h\ x\ (\hat{\ })_S\ (n::nat)$

⟨*proof*⟩

lemma (**in** *UP-univ-prop*) *Eval-monom*:
$r \in carrier\ R ==> Eval\ (monom\ P\ r\ n) = h\ r \otimes_S s\ (\hat{\ })_S\ n$
⟨*proof*⟩

lemma (**in** *UP-pre-univ-prop*) *eval-monom*:
assumes *R*: $r \in carrier\ R$ **and** *S*: $s \in carrier\ S$
shows $eval\ R\ S\ h\ s\ (monom\ P\ r\ n) = h\ r \otimes_S s\ (\hat{\ })_S\ n$
⟨*proof*⟩

lemma (**in** *UP-univ-prop*) *Eval-smult*:
$[|\ r \in carrier\ R;\ p \in carrier\ P\ |] ==> Eval\ (r \odot_P p) = h\ r \otimes_S Eval\ p$
⟨*proof*⟩

lemma *ring-hom-cringI*:
assumes *cring R*
and *cring S*
and $h \in ring\text{-}hom\ R\ S$
shows *ring-hom-cring R S h*
⟨*proof*⟩

lemma (**in** *UP-pre-univ-prop*) *UP-hom-unique*:
includes *ring-hom-cring P S Phi*
assumes *Phi*: $Phi\ (monom\ P\ \mathbf{1}\ (Suc\ 0)) = s$
$!!r.\ r \in carrier\ R ==> Phi\ (monom\ P\ r\ 0) = h\ r$
includes *ring-hom-cring P S Psi*
assumes *Psi*: $Psi\ (monom\ P\ \mathbf{1}\ (Suc\ 0)) = s$
$!!r.\ r \in carrier\ R ==> Psi\ (monom\ P\ r\ 0) = h\ r$
and *P*: $p \in carrier\ P$ **and** *S*: $s \in carrier\ S$
shows $Phi\ p = Psi\ p$
⟨*proof*⟩

lemma (**in** *UP-pre-univ-prop*) *ring-homD*:
assumes *Phi*: $Phi \in ring\text{-}hom\ P\ S$
shows *ring-hom-cring P S Phi*
⟨*proof*⟩

theorem (**in** *UP-pre-univ-prop*) *UP-universal-property*:
assumes *S*: $s \in carrier\ S$
shows $EX!\ Phi.\ Phi \in ring\text{-}hom\ P\ S \cap extensional\ (carrier\ P)\ \&$
$Phi\ (monom\ P\ \mathbf{1}\ 1) = s\ \&$
$(ALL\ r : carrier\ R.\ Phi\ (monom\ P\ r\ 0) = h\ r)$
⟨*proof*⟩

11.9 Sample Application of Evaluation Homomorphism

lemma *UP-pre-univ-propI*:
assumes *cring R*

  and *cring S*
  and $h \in$ *ring-hom R S*
shows *UP-pre-univ-prop R S h*
⟨*proof*⟩

constdefs
INTEG :: *int ring*
INTEG == (| *carrier* = *UNIV*, *mult* = *op* ∗, *one* = *1*, *zero* = *0*, *add* = *op* + |)

lemma *INTEG-cring*:
cring INTEG
⟨*proof*⟩

lemma *INTEG-id-eval*:
UP-pre-univ-prop INTEG INTEG id
⟨*proof*⟩

Interpretation now enables to import all theorems and lemmas valid in the context of homomorphisms between *INTEG* and *UP INTEG* globally.

interpretation *INTEG*: *UP-pre-univ-prop* [*INTEG INTEG id*]
⟨*proof*⟩

lemma *INTEG-closed* [*intro*, *simp*]:
$z \in$ *carrier INTEG*
⟨*proof*⟩

lemma *INTEG-mult* [*simp*]:
mult INTEG z w = *z* ∗ *w*
⟨*proof*⟩

lemma *INTEG-pow* [*simp*]:
pow INTEG z n = *z* ˆ *n*
⟨*proof*⟩

lemma *eval INTEG INTEG id 10* (*monom* (*UP INTEG*) *5 2*) = *500*
⟨*proof*⟩

end

theory *AbelCoset*
imports *Coset Ring*
begin

12 More Lifting from Groups to Abelian Groups

12.1 Definitions

Hiding <+> from *Sum-Type* until I come up with better syntax here

hide *const Plus*

constdefs (**structure** *G*)
  a-r-coset :: [-, *'a set*, *'a*] ⇒ *'a set* (**infixl** +>ı *60*)
  a-r-coset G ≡ *r-coset* (|*carrier* = *carrier G*, *mult* = *add G*, *one* = *zero G*|)

  a-l-coset :: [-, *'a*, *'a set*] ⇒ *'a set* (**infixl** <+ı *60*)
  a-l-coset G ≡ *l-coset* (|*carrier* = *carrier G*, *mult* = *add G*, *one* = *zero G*|)

  A-RCOSETS :: [-, *'a set*] ⇒ (*'a set*)*set* (*a'-rcosets*ı - [*81*] *80*)
  A-RCOSETS G H ≡ *RCOSETS* (|*carrier* = *carrier G*, *mult* = *add G*, *one* = *zero G*|) *H*

  set-add :: [-, *'a set* ,*'a set*] ⇒ *'a set* (**infixl** <+>ı *60*)
  set-add G ≡ *set-mult* (|*carrier* = *carrier G*, *mult* = *add G*, *one* = *zero G*|)

  A-SET-INV :: [-,*'a set*] ⇒ *'a set* (*a'-set'-inv*ı - [*81*] *80*)
  A-SET-INV G H ≡ *SET-INV* (|*carrier* = *carrier G*, *mult* = *add G*, *one* = *zero G*|) *H*

constdefs (**structure** *G*)
  a-r-congruent :: [(*'a*,*'b*)*ring-scheme*, *'a set*] ⇒ (*'a*∗*'a*)*set*
    (*racong*ı -)
  a-r-congruent G ≡ *r-congruent* (|*carrier* = *carrier G*, *mult* = *add G*, *one* = *zero G*|)

constdefs
  A-FactGroup :: [(*'a*,*'b*) *ring-scheme*, *'a set*] ⇒ (*'a set*) *monoid*
    (**infixl** *A'-Mod 65*)
  — Actually defined for groups rather than monoids
  A-FactGroup G H ≡ *FactGroup* (|*carrier* = *carrier G*, *mult* = *add G*, *one* = *zero G*|) *H*

constdefs
  a-kernel :: (*'a*, *'m*) *ring-scheme* ⇒ (*'b*, *'n*) *ring-scheme* ⇒
    (*'a* ⇒ *'b*) ⇒ *'a set*
  — the kernel of a homomorphism (additive)
  a-kernel G H h ≡ *kernel* (|*carrier* = *carrier G*, *mult* = *add G*, *one* = *zero G*|)
    (|*carrier* = *carrier H*, *mult* = *add H*, *one* = *zero H*|) *h*

locale *abelian-group-hom* = *abelian-group G* + *abelian-group H* + *var h* +
  assumes *a-group-hom*: *group-hom* (| *carrier* = *carrier G*, *mult* = *add G*, *one* = *zero G* |)
    (| *carrier* = *carrier H*, *mult* = *add H*, *one* = *zero H* |) *h*

lemmas *a-r-coset-defs* =
a-r-coset-def r-coset-def

lemma *a-r-coset-def'*:
includes *struct G*
shows $H +> a \equiv \bigcup h{\in}H.\ \{h \oplus a\}$
⟨*proof*⟩

lemmas *a-l-coset-defs* =
a-l-coset-def l-coset-def

lemma *a-l-coset-def'*:
includes *struct G*
shows $a <+ H \equiv \bigcup h{\in}H.\ \{a \oplus h\}$
⟨*proof*⟩

lemmas *A-RCOSETS-defs* =
A-RCOSETS-def RCOSETS-def

lemma *A-RCOSETS-def'*:
includes *struct G*
shows $a\text{-}rcosets\ H \equiv \bigcup a{\in}carrier\ G.\ \{H +> a\}$
⟨*proof*⟩

lemmas *set-add-defs* =
set-add-def set-mult-def

lemma *set-add-def'*:
includes *struct G*
shows $H <+> K \equiv \bigcup h{\in}H.\ \bigcup k{\in}K.\ \{h \oplus k\}$
⟨*proof*⟩

lemmas *A-SET-INV-defs* =
A-SET-INV-def SET-INV-def

lemma *A-SET-INV-def'*:
includes *struct G*
shows $a\text{-}set\text{-}inv\ H \equiv \bigcup h{\in}H.\ \{\ominus\ h\}$
⟨*proof*⟩

12.2 Cosets

lemma (**in** *abelian-group*) *a-coset-add-assoc*:
$[|\ M \subseteq carrier\ G;\ g \in carrier\ G;\ h \in carrier\ G\ |]$
$==> (M +> g) +> h = M +> (g \oplus h)$
⟨*proof*⟩

lemma (**in** *abelian-group*) *a-coset-add-zero* [*simp*]:

$M \subseteq carrier\ G ==> M +> \mathbf{0} = M$
⟨proof⟩

lemma (**in** *abelian-group*) *a-coset-add-inv1*:
$[| M +> (x \oplus (\ominus y)) = M;\ x \in carrier\ G\ ;\ y \in carrier\ G;$
$M \subseteq carrier\ G\ |] ==> M +> x = M +> y$
⟨proof⟩

lemma (**in** *abelian-group*) *a-coset-add-inv2*:
$[| M +> x = M +> y;\ x \in carrier\ G;\ y \in carrier\ G;\ M \subseteq carrier\ G\ |]$
$==> M +> (x \oplus (\ominus y)) = M$
⟨proof⟩

lemma (**in** *abelian-group*) *a-coset-join1*:
$[| H +> x = H;\ x \in carrier\ G;\ subgroup\ H\ (|carrier = carrier\ G,\ mult = add\ G,\ one = zero\ G|)\ |] ==> x \in H$
⟨proof⟩

lemma (**in** *abelian-group*) *a-solve-equation*:
$[\![subgroup\ H\ (|carrier = carrier\ G,\ mult = add\ G,\ one = zero\ G|);\ x \in H;\ y \in H]\!] \Longrightarrow \exists h{\in}H.\ y = h \oplus x$
⟨proof⟩

lemma (**in** *abelian-group*) *a-repr-independence*:
$[\![y \in H +> x;\ x \in carrier\ G;\ subgroup\ H\ (\!|carrier = carrier\ G,\ mult = add\ G,\ one = zero\ G|\!)\ ]\!] \Longrightarrow H +> x = H +> y$
⟨proof⟩

lemma (**in** *abelian-group*) *a-coset-join2*:
$[\![x \in carrier\ G;\ subgroup\ H\ (\!|carrier = carrier\ G,\ mult = add\ G,\ one = zero\ G|\!);\ x{\in}H]\!] \Longrightarrow H +> x = H$
⟨proof⟩

lemma (**in** *abelian-monoid*) *a-r-coset-subset-G*:
$[| H \subseteq carrier\ G;\ x \in carrier\ G\ |] ==> H +> x \subseteq carrier\ G$
⟨proof⟩

lemma (**in** *abelian-group*) *a-rcosI*:
$[| h \in H;\ H \subseteq carrier\ G;\ x \in carrier\ G|] ==> h \oplus x \in H +> x$
⟨proof⟩

lemma (**in** *abelian-group*) *a-rcosetsI*:
$[\![H \subseteq carrier\ G;\ x \in carrier\ G]\!] \Longrightarrow H +> x \in a\text{-}rcosets\ H$
⟨proof⟩

Really needed?

lemma (**in** *abelian-group*) *a-transpose-inv*:
$[| x \oplus y = z;\ x \in carrier\ G;\ y \in carrier\ G;\ z \in carrier\ G\ |]$
$==> (\ominus x) \oplus z = y$

$\langle proof \rangle$

12.3 Subgroups

locale *additive-subgroup* = *var H* + *struct G* +
assumes *a-subgroup*: *subgroup H* (|*carrier* = *carrier G*, *mult* = *add G*, *one* = *zero G*|)

lemma (**in** *additive-subgroup*) *is-additive-subgroup*:
shows *additive-subgroup H G*
$\langle proof \rangle$

lemma *additive-subgroupI*:
includes *struct G*
assumes *a-subgroup*: *subgroup H* (|*carrier* = *carrier G*, *mult* = *add G*, *one* = *zero G*|)
shows *additive-subgroup H G*
$\langle proof \rangle$

lemma (**in** *additive-subgroup*) *a-subset*:
$H \subseteq carrier\ G$
$\langle proof \rangle$

lemma (**in** *additive-subgroup*) *a-closed* [*intro*, *simp*]:
$[\![x \in H; y \in H]\!] \Longrightarrow x \oplus y \in H$
$\langle proof \rangle$

lemma (**in** *additive-subgroup*) *zero-closed* [*simp*]:
$\mathbf{0} \in H$
$\langle proof \rangle$

lemma (**in** *additive-subgroup*) *a-inv-closed* [*intro*,*simp*]:
$x \in H \Longrightarrow \ominus\ x \in H$
$\langle proof \rangle$

12.4 Normal additive subgroups

12.4.1 Definition of *abelian-subgroup*

Every subgroup of an *abelian-group* is normal

locale *abelian-subgroup* = *additive-subgroup H G* + *abelian-group G* +
assumes *a-normal*: *normal H* (|*carrier* = *carrier G*, *mult* = *add G*, *one* = *zero G*|)

lemma (**in** *abelian-subgroup*) *is-abelian-subgroup*:
shows *abelian-subgroup H G*
$\langle proof \rangle$

lemma *abelian-subgroupI*:

assumes *a-normal*: *normal H* $(\\|$*carrier* = *carrier G*, *mult* = *add G*, *one* = *zero G*$\|)$
and *a-comm*: !!*x y*. [| *x* ∈ *carrier G*; *y* ∈ *carrier G* |] ==> $x \oplus_G y = y \oplus_G x$
shows *abelian-subgroup H G*
⟨*proof*⟩

lemma *abelian-subgroupI2*:
includes *struct G*
assumes *a-comm-group*: *comm-group* $(\\|$*carrier* = *carrier G*, *mult* = *add G*, *one* = *zero G*$\|)$
and *a-subgroup*: *subgroup H* $(\\|$*carrier* = *carrier G*, *mult* = *add G*, *one* = *zero G*$\|)$
shows *abelian-subgroup H G*
⟨*proof*⟩

lemma *abelian-subgroupI3*:
includes *struct G*
assumes *asg*: *additive-subgroup H G*
and *ag*: *abelian-group G*
shows *abelian-subgroup H G*
⟨*proof*⟩

lemma (**in** *abelian-subgroup*) *a-coset-eq*:
(∀ *x* ∈ *carrier G*. *H* +> *x* = *x* <+ *H*)
⟨*proof*⟩

lemma (**in** *abelian-subgroup*) *a-inv-op-closed1*:
shows $[\![x \in carrier\ G;\ h \in H]\!] \Longrightarrow (\ominus\ x) \oplus h \oplus x \in H$
⟨*proof*⟩

lemma (**in** *abelian-subgroup*) *a-inv-op-closed2*:
shows $[\![x \in carrier\ G;\ h \in H]\!] \Longrightarrow x \oplus h \oplus (\ominus\ x) \in H$
⟨*proof*⟩

Alternative characterization of normal subgroups

lemma (**in** *abelian-group*) *a-normal-inv-iff*:
(*N* ◁ $(\\|$*carrier* = *carrier G*, *mult* = *add G*, *one* = *zero G*$\|)$) =
(*subgroup N* $(\\|$*carrier* = *carrier G*, *mult* = *add G*, *one* = *zero G*$\|)$ & (∀ *x* ∈ *carrier G*. ∀ *h* ∈ *N*. $x \oplus h \oplus (\ominus\ x) \in N$))
(**is** - = *?rhs*)
⟨*proof*⟩

lemma (**in** *abelian-group*) *a-lcos-m-assoc*:
[| *M* ⊆ *carrier G*; *g* ∈ *carrier G*; *h* ∈ *carrier G* |]
==> *g* <+ (*h* <+ *M*) = ($g \oplus h$) <+ *M*
⟨*proof*⟩

lemma (**in** *abelian-group*) *a-lcos-mult-one*:

$M \subseteq \mathit{carrier}\ G ==> \mathbf{0} <+ M = M$
$\langle \mathit{proof} \rangle$

lemma (**in** *abelian-group*) *a-l-coset-subset-G*:
$[|\ H \subseteq \mathit{carrier}\ G;\ x \in \mathit{carrier}\ G\ |] ==> x <+ H \subseteq \mathit{carrier}\ G$
$\langle \mathit{proof} \rangle$

lemma (**in** *abelian-group*) *a-l-coset-swap*:
$[\![y \in x <+ H;\ \ x \in \mathit{carrier}\ G;\ \ \mathit{subgroup}\ H\ (\!|\mathit{carrier} = \mathit{carrier}\ G,\ \mathit{mult} = \mathit{add}\ G,\ \mathit{one} = \mathit{zero}\ G|\!)]\!] \Longrightarrow x \in y <+ H$
$\langle \mathit{proof} \rangle$

lemma (**in** *abelian-group*) *a-l-coset-carrier*:
$[|\ y \in x <+ H;\ \ x \in \mathit{carrier}\ G;\ \ \mathit{subgroup}\ H\ (\!|\mathit{carrier} = \mathit{carrier}\ G,\ \mathit{mult} = \mathit{add}\ G,\ \mathit{one} = \mathit{zero}\ G|\!)\ |] ==> y \in \mathit{carrier}\ G$
$\langle \mathit{proof} \rangle$

lemma (**in** *abelian-group*) *a-l-repr-imp-subset*:
assumes y: $y \in x <+ H$ **and** x: $x \in \mathit{carrier}\ G$ **and** sb: $\mathit{subgroup}\ H\ (\!|\mathit{carrier} = \mathit{carrier}\ G,\ \mathit{mult} = \mathit{add}\ G,\ \mathit{one} = \mathit{zero}\ G|\!)$
shows $y <+ H \subseteq x <+ H$
$\langle \mathit{proof} \rangle$

lemma (**in** *abelian-group*) *a-l-repr-independence*:
assumes y: $y \in x <+ H$ **and** x: $x \in \mathit{carrier}\ G$ **and** sb: $\mathit{subgroup}\ H\ (\!|\mathit{carrier} = \mathit{carrier}\ G,\ \mathit{mult} = \mathit{add}\ G,\ \mathit{one} = \mathit{zero}\ G|\!)$
shows $x <+ H = y <+ H$
$\langle \mathit{proof} \rangle$

lemma (**in** *abelian-group*) *setadd-subset-G*:
$[\![H \subseteq \mathit{carrier}\ G;\ K \subseteq \mathit{carrier}\ G]\!] \Longrightarrow H <+> K \subseteq \mathit{carrier}\ G$
$\langle \mathit{proof} \rangle$

lemma (**in** *abelian-group*) *subgroup-add-id*: $\mathit{subgroup}\ H\ (\!|\mathit{carrier} = \mathit{carrier}\ G,\ \mathit{mult} = \mathit{add}\ G,\ \mathit{one} = \mathit{zero}\ G|\!) \Longrightarrow H <+> H = H$
$\langle \mathit{proof} \rangle$

lemma (**in** *abelian-subgroup*) *a-rcos-inv*:
assumes x: $\quad x \in \mathit{carrier}\ G$
shows $\mathit{a\text{-}set\text{-}inv}\ (H +> x) = H +> (\ominus\ x)$
$\langle \mathit{proof} \rangle$

lemma (**in** *abelian-group*) *a-setmult-rcos-assoc*:
$[\![H \subseteq \mathit{carrier}\ G;\ K \subseteq \mathit{carrier}\ G;\ x \in \mathit{carrier}\ G]\!]$
$\Longrightarrow H <+> (K +> x) = (H <+> K) +> x$
$\langle \mathit{proof} \rangle$

lemma (**in** *abelian-group*) *a-rcos-assoc-lcos*:
$\llbracket H \subseteq carrier\ G;\ K \subseteq carrier\ G;\ x \in carrier\ G \rrbracket$
$\Longrightarrow (H\ {+}{>}\ x)\ {<}{+}{>}\ K = H\ {<}{+}{>}\ (x\ {<}{+}\ K)$
⟨*proof*⟩

lemma (**in** *abelian-subgroup*) *a-rcos-sum*:
$\llbracket x \in carrier\ G;\ y \in carrier\ G \rrbracket$
$\Longrightarrow (H\ {+}{>}\ x)\ {<}{+}{>}\ (H\ {+}{>}\ y) = H\ {+}{>}\ (x \oplus y)$
⟨*proof*⟩

lemma (**in** *abelian-subgroup*) *rcosets-add-eq*:
$M \in a\text{-}rcosets\ H \Longrightarrow H\ {<}{+}{>}\ M = M$
— generalizes *subgroup-mult-id*
⟨*proof*⟩

12.5 Congruence Relation

lemma (**in** *abelian-subgroup*) *a-equiv-rcong*:
shows *equiv* (*carrier G*) (*racong H*)
⟨*proof*⟩

lemma (**in** *abelian-subgroup*) *a-l-coset-eq-rcong*:
assumes *a*: $a \in carrier\ G$
shows $a\ {<}{+}\ H = racong\ H\ ``\ \{a\}$
⟨*proof*⟩

lemma (**in** *abelian-subgroup*) *a-rcos-equation*:
shows
$\llbracket ha \oplus a = h \oplus b;\ a \in carrier\ G;\ \ b \in carrier\ G;$
$h \in H;\ \ ha \in H;\ \ hb \in H \rrbracket$
$\Longrightarrow hb \oplus a \in (\bigcup h{\in}H.\ \{h \oplus b\})$
⟨*proof*⟩

lemma (**in** *abelian-subgroup*) *a-rcos-disjoint*:
shows $\llbracket a \in a\text{-}rcosets\ H;\ b \in a\text{-}rcosets\ H;\ a{\neq}b \rrbracket \Longrightarrow a \cap b = \{\}$
⟨*proof*⟩

lemma (**in** *abelian-subgroup*) *a-rcos-self*:
shows $x \in carrier\ G \Longrightarrow x \in H\ {+}{>}\ x$
⟨*proof*⟩

lemma (**in** *abelian-subgroup*) *a-rcosets-part-G*:
shows $\bigcup(a\text{-}rcosets\ H) = carrier\ G$
⟨*proof*⟩

lemma (**in** *abelian-subgroup*) *a-cosets-finite*:
$\llbracket c \in a\text{-}rcosets\ H;\ \ H \subseteq carrier\ G;\ \ finite\ (carrier\ G) \rrbracket \Longrightarrow finite\ c$
⟨*proof*⟩

lemma (**in** *abelian-group*) *a-card-cosets-equal*:
$[\![c \in \textit{a-rcosets } H;\ \ H \subseteq \textit{carrier } G;\ \textit{finite}(\textit{carrier } G)]\!]$
$\Longrightarrow \textit{card } c = \textit{card } H$
⟨*proof*⟩

lemma (**in** *abelian-group*) *rcosets-subset-PowG*:
$\textit{additive-subgroup } H\ G \ \Longrightarrow \textit{a-rcosets } H \subseteq \textit{Pow}(\textit{carrier } G)$
⟨*proof*⟩

theorem (**in** *abelian-group*) *a-lagrange*:
$[\![\textit{finite}(\textit{carrier } G);\ \textit{additive-subgroup } H\ G]\!]$
$\Longrightarrow \textit{card}(\textit{a-rcosets } H) * \textit{card}(H) = \textit{order}(G)$
⟨*proof*⟩

12.6 Factorization

lemmas *A-FactGroup-defs* = *A-FactGroup-def FactGroup-def*

lemma *A-FactGroup-def ′*:
includes *struct G*
shows $G\ \textit{A-Mod}\ H \equiv (\!|\textit{carrier} = \textit{a-rcosets}_G\ H,\ \textit{mult} = \textit{set-add}\ G,\ \textit{one} = H|\!)$
⟨*proof*⟩

lemma (**in** *abelian-subgroup*) *a-setmult-closed*:
$[\![K1 \in \textit{a-rcosets } H;\ K2 \in \textit{a-rcosets } H]\!] \Longrightarrow K1 <\!+\!> K2 \in \textit{a-rcosets } H$
⟨*proof*⟩

lemma (**in** *abelian-subgroup*) *a-setinv-closed*:
$K \in \textit{a-rcosets } H \Longrightarrow \textit{a-set-inv } K \in \textit{a-rcosets } H$
⟨*proof*⟩

lemma (**in** *abelian-subgroup*) *a-rcosets-assoc*:
$[\![M1 \in \textit{a-rcosets } H;\ M2 \in \textit{a-rcosets } H;\ M3 \in \textit{a-rcosets } H]\!]$
$\Longrightarrow M1 <\!+\!> M2 <\!+\!> M3 = M1 <\!+\!> (M2 <\!+\!> M3)$
⟨*proof*⟩

lemma (**in** *abelian-subgroup*) *a-subgroup-in-rcosets*:
$H \in \textit{a-rcosets } H$
⟨*proof*⟩

lemma (**in** *abelian-subgroup*) *a-rcosets-inv-mult-group-eq*:
$M \in \textit{a-rcosets } H \Longrightarrow \textit{a-set-inv } M <\!+\!> M = H$
⟨*proof*⟩

theorem (**in** *abelian-subgroup*) *a-factorgroup-is-group*:
group (*G A-Mod H*)
⟨*proof*⟩

Since the Factorization is based on an *abelian* subgroup, is results in a commutative group

theorem (**in** *abelian-subgroup*) *a-factorgroup-is-comm-group*:
comm-group (*G A-Mod H*)
⟨*proof*⟩

lemma *add-A-FactGroup* [*simp*]: $X \otimes_{(G\ A\text{-}Mod\ H)} X' = X <\!+\!>_G X'$
⟨*proof*⟩

lemma (**in** *abelian-subgroup*) *a-inv-FactGroup*:
$X \in carrier\ (G\ A\text{-}Mod\ H) \Longrightarrow inv_{G\ A\text{-}Mod\ H}\ X = a\text{-}set\text{-}inv\ X$
⟨*proof*⟩

The coset map is a homomorphism from G to the quotient group $G\ Mod\ H$

lemma (**in** *abelian-subgroup*) *a-r-coset-hom-A-Mod*:
$(\lambda a.\ H +\!> a) \in hom\ (\!|carrier = carrier\ G,\ mult = add\ G,\ one = zero\ G|\!)\ (G\ A\text{-}Mod\ H)$
⟨*proof*⟩

The isomorphism theorems have been omitted from lifting, at least for now

12.7 The First Isomorphism Theorem

The quotient by the kernel of a homomorphism is isomorphic to the range of that homomorphism.

lemmas *a-kernel-defs* =
a-kernel-def kernel-def

lemma *a-kernel-def'*:
$a\text{-}kernel\ R\ S\ h \equiv \{x \in carrier\ R.\ h\ x = \mathbf{0}_S\}$
⟨*proof*⟩

12.8 Homomorphisms

lemma *abelian-group-homI*:
includes *abelian-group G*
includes *abelian-group H*
assumes *a-group-hom*: $group\text{-}hom\ (|\ carrier = carrier\ G,\ mult = add\ G,\ one = zero\ G\ |)$
$(|\ carrier = carrier\ H,\ mult = add\ H,\ one = zero\ H\ |)\ h$
shows *abelian-group-hom G H h*
⟨*proof*⟩

lemma (**in** *abelian-group-hom*) *is-abelian-group-hom*:
abelian-group-hom G H h
⟨*proof*⟩

lemma (**in** *abelian-group-hom*) *hom-add* [*simp*]:
$[|\ x : carrier\ G;\ y : carrier\ G\ |]$

$==> h\ (x \oplus_G y) = h\ x \oplus_H h\ y$
$\langle proof \rangle$

lemma (**in** *abelian-group-hom*) *hom-closed* [*simp*]:
$x \in carrier\ G \Longrightarrow h\ x \in carrier\ H$
$\langle proof \rangle$

lemma (**in** *abelian-group-hom*) *zero-closed* [*simp*]:
$h\ \mathbf{0} \in carrier\ H$
$\langle proof \rangle$

lemma (**in** *abelian-group-hom*) *hom-zero* [*simp*]:
$h\ \mathbf{0} = \mathbf{0}_H$
$\langle proof \rangle$

lemma (**in** *abelian-group-hom*) *a-inv-closed* [*simp*]:
$x \in carrier\ G ==> h\ (\ominus x) \in carrier\ H$
$\langle proof \rangle$

lemma (**in** *abelian-group-hom*) *hom-a-inv* [*simp*]:
$x \in carrier\ G ==> h\ (\ominus x) = \ominus_H\ (h\ x)$
$\langle proof \rangle$

lemma (**in** *abelian-group-hom*) *additive-subgroup-a-kernel*:
additive-subgroup (*a-kernel G H h*) *G*
$\langle proof \rangle$

The kernel of a homomorphism is an abelian subgroup

lemma (**in** *abelian-group-hom*) *abelian-subgroup-a-kernel*:
abelian-subgroup (*a-kernel G H h*) *G*
$\langle proof \rangle$

lemma (**in** *abelian-group-hom*) *A-FactGroup-nonempty*:
assumes *X*: $X \in carrier\ (G\ A\text{-}Mod\ a\text{-}kernel\ G\ H\ h)$
shows $X \neq \{\}$
$\langle proof \rangle$

lemma (**in** *abelian-group-hom*) *FactGroup-contents-mem*:
assumes *X*: $X \in carrier\ (G\ A\text{-}Mod\ (a\text{-}kernel\ G\ H\ h))$
shows $contents\ (h`X) \in carrier\ H$
$\langle proof \rangle$

lemma (**in** *abelian-group-hom*) *A-FactGroup-hom*:
$(\lambda X.\ contents\ (h`X)) \in hom\ (G\ A\text{-}Mod\ (a\text{-}kernel\ G\ H\ h))$
$(\!|carrier = carrier\ H,\ mult = add\ H,\ one = zero\ H|\!)$
$\langle proof \rangle$

lemma (**in** *abelian-group-hom*) *A-FactGroup-inj-on*:
$inj\text{-}on\ (\lambda X.\ contents\ (h\ `\ X))\ (carrier\ (G\ A\text{-}Mod\ a\text{-}kernel\ G\ H\ h))$

⟨*proof*⟩

If the homomorphism h is onto H, then so is the homomorphism from the quotient group

lemma (**in** *abelian-group-hom*) *A-FactGroup-onto*:
  assumes *h*: h ‘ *carrier* G = *carrier* H
  shows $(\lambda X.\ contents\ (h\ ‘\ X))$ ‘ *carrier* $(G$ *A-Mod a-kernel* $G\ H\ h)$ = *carrier* H
⟨*proof*⟩

If h is a homomorphism from G onto H, then the quotient group G *Mod kernel* $G\ H\ h$ is isomorphic to H.

theorem (**in** *abelian-group-hom*) *A-FactGroup-iso*:
  h ‘ *carrier* G = *carrier* H
   $\Longrightarrow (\lambda X.\ contents\ (h‘X)) \in (G\ A\text{-}Mod\ (a\text{-}kernel\ G\ H\ h)) \cong$
      (| *carrier* = *carrier* H, *mult* = *add* H, *one* = *zero* H |)
⟨*proof*⟩

13 Lemmas Lifted from CosetExt.thy

Not eveything from `CosetExt.thy` is lifted here.

13.1 General Lemmas from `AlgebraExt.thy`

lemma (**in** *additive-subgroup*) *a-Hcarr* [*simp*]:
  assumes *hH*: $h \in H$
  shows $h \in$ *carrier* G
⟨*proof*⟩

13.2 Lemmas for Right Cosets

lemma (**in** *abelian-subgroup*) *a-elemrcos-carrier*:
  assumes *acarr*: $a \in$ *carrier* G
    and a': $a' \in H$ +> a
  shows $a' \in$ *carrier* G
⟨*proof*⟩

lemma (**in** *abelian-subgroup*) *a-rcos-const*:
  assumes *hH*: $h \in H$
  shows H +> $h = H$
⟨*proof*⟩

lemma (**in** *abelian-subgroup*) *a-rcos-module-imp*:
  assumes *xcarr*: $x \in$ *carrier* G
    and *x'cos*: $x' \in H$ +> x
  shows $(x' \oplus \ominus x) \in H$
⟨*proof*⟩

lemma (**in** *abelian-subgroup*) *a-rcos-module-rev*:

assumes $x \in$ *carrier* G $x' \in$ *carrier* G
and $(x' \oplus \ominus x) \in H$
shows $x' \in H$ +> x
⟨*proof*⟩

lemma (**in** *abelian-subgroup*) *a-rcos-module*:
assumes $x \in$ *carrier* G $x' \in$ *carrier* G
shows $(x' \in H$ +> $x) = (x' \oplus \ominus x \in H)$
⟨*proof*⟩
lemma (**in** *abelian-subgroup*) *a-rcos-module-minus*:
includes *ring* G
assumes *carr*: $x \in$ *carrier* G $x' \in$ *carrier* G
shows $(x' \in H$ +> $x) = (x' \ominus x \in H)$
⟨*proof*⟩

lemma (**in** *abelian-subgroup*) *a-repr-independence'*:
assumes *y*: $y \in H$ +> x
and *xcarr*: $x \in$ *carrier* G
shows H +> $x = H$ +> y
⟨*proof*⟩

lemma (**in** *abelian-subgroup*) *a-repr-independenceD*:
assumes *ycarr*: $y \in$ *carrier* G
and *repr*: H +> $x = H$ +> y
shows $y \in H$ +> x
⟨*proof*⟩

13.3 Lemmas for the Set of Right Cosets

lemma (**in** *abelian-subgroup*) *a-rcosets-carrier*:
$X \in$ *a-rcosets* $H \Longrightarrow X \subseteq$ *carrier* G
⟨*proof*⟩

13.4 Addition of Subgroups

lemma (**in** *abelian-monoid*) *set-add-closed*:
assumes *Acarr*: $A \subseteq$ *carrier* G
and *Bcarr*: $B \subseteq$ *carrier* G
shows A <+> $B \subseteq$ *carrier* G
⟨*proof*⟩

lemma (**in** *abelian-group*) *add-additive-subgroups*:
assumes *subH*: *additive-subgroup* H G
and *subK*: *additive-subgroup* K G
shows *additive-subgroup* $(H$ <+> $K)$ G
⟨*proof*⟩

end

theory *Ideal*
imports *Ring AbelCoset*
begin

14 Ideals

14.1 General definition

locale *ideal* = *additive-subgroup I R* + *ring R* +
  assumes *I-l-closed*: $[\![a \in I; x \in carrier\ R]\!] \Longrightarrow x \otimes a \in I$
    and *I-r-closed*: $[\![a \in I; x \in carrier\ R]\!] \Longrightarrow a \otimes x \in I$

interpretation *ideal* ⊆ *abelian-subgroup I R*
⟨*proof*⟩

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

lemma *idealI*:
  includes *ring*
  assumes *a-subgroup*: *subgroup I* (|*carrier* = *carrier R*, *mult* = *add R*, *one* = *zero R*|)
    and *I-l-closed*: $\bigwedge a\ x.\ [\![a \in I; x \in carrier\ R]\!] \Longrightarrow x \otimes a \in I$
    and *I-r-closed*: $\bigwedge a\ x.\ [\![a \in I; x \in carrier\ R]\!] \Longrightarrow a \otimes x \in I$
  shows *ideal I R*
  ⟨*proof*⟩

14.2 Ideals Generated by a Subset of *carrier R*

constdefs (**structure** *R*)
  genideal :: (*'a*, *'b*) *ring-scheme* ⇒ *'a set* ⇒ *'a set* (*Idl*ı - [*80*] *79*)
  genideal R S ≡ *Inter* {*I*. *ideal I R* ∧ *S* ⊆ *I*}

14.3 Principal Ideals

locale *principalideal* = *ideal* +
  assumes *generate*: ∃ *i* ∈ *carrier R*. *I* = *Idl* {*i*}

lemma (**in** *principalideal*) *is-principalideal*:
  shows *principalideal I R*
⟨*proof*⟩

lemma *principalidealI*:
  includes *ideal*
  assumes *generate*: ∃ *i* ∈ *carrier R*. *I* = *Idl* {*i*}
  shows *principalideal I R*
  ⟨*proof*⟩

14.4 Maximal Ideals

locale *maximalideal* = *ideal* +
assumes *I-notcarr*: *carrier* $R \neq I$
and *I-maximal*: $[\![$*ideal* J R; $I \subseteq J$; $J \subseteq$ *carrier* $R]\!] \Longrightarrow J = I \vee J =$ *carrier* R

lemma (**in** *maximalideal*) *is-maximalideal*:
shows *maximalideal* I R
⟨*proof*⟩

lemma *maximalidealI*:
includes *ideal*
assumes *I-notcarr*: *carrier* $R \neq I$
and *I-maximal*: $\bigwedge J$. $[\![$*ideal* J R; $I \subseteq J$; $J \subseteq$ *carrier* $R]\!] \Longrightarrow J = I \vee J =$ *carrier* R
shows *maximalideal* I R
⟨*proof*⟩

14.5 Prime Ideals

locale *primeideal* = *ideal* + *cring* +
assumes *I-notcarr*: *carrier* $R \neq I$
and *I-prime*: $[\![a \in$ *carrier* R; $b \in$ *carrier* R; $a \otimes b \in I]\!] \Longrightarrow a \in I \vee b \in I$

lemma (**in** *primeideal*) *is-primeideal*:
shows *primeideal* I R
⟨*proof*⟩

lemma *primeidealI*:
includes *ideal*
includes *cring*
assumes *I-notcarr*: *carrier* $R \neq I$
and *I-prime*: $\bigwedge a\ b$. $[\![a \in$ *carrier* R; $b \in$ *carrier* R; $a \otimes b \in I]\!] \Longrightarrow a \in I \vee b \in I$
shows *primeideal* I R
⟨*proof*⟩

lemma *primeidealI2*:
includes *additive-subgroup* I R
includes *cring*
assumes *I-l-closed*: $\bigwedge a\ x$. $[\![a \in I$; $x \in$ *carrier* $R]\!] \Longrightarrow x \otimes a \in I$
and *I-r-closed*: $\bigwedge a\ x$. $[\![a \in I$; $x \in$ *carrier* $R]\!] \Longrightarrow a \otimes x \in I$
and *I-notcarr*: *carrier* $R \neq I$
and *I-prime*: $\bigwedge a\ b$. $[\![a \in$ *carrier* R; $b \in$ *carrier* R; $a \otimes b \in I]\!] \Longrightarrow a \in I \vee b \in I$
shows *primeideal* I R
⟨*proof*⟩

15 Properties of Ideals

15.1 Special Ideals

lemma (**in** *ring*) *zeroideal*:
  shows *ideal* $\{\mathbf{0}\}$ *R*
⟨*proof*⟩

lemma (**in** *ring*) *oneideal*:
  shows *ideal* (*carrier R*) *R*
⟨*proof*⟩

lemma (**in** *domain*) *zeroprimeideal*:
 shows *primeideal* $\{\mathbf{0}\}$ *R*
⟨*proof*⟩

15.2 General Ideal Properies

lemma (**in** *ideal*) *one-imp-carrier*:
  assumes *I-one-closed*: $\mathbf{1} \in I$
  shows $I =$ *carrier R*
⟨*proof*⟩

lemma (**in** *ideal*) *Icarr*:
  assumes *iI*: $i \in I$
  shows $i \in$ *carrier R*
⟨*proof*⟩

15.3 Intersection of Ideals

Intersection of two ideals The intersection of any two ideals is again an ideal in R

lemma (**in** *ring*) *i-intersect*:
  includes *ideal I R*
  includes *ideal J R*
  shows *ideal* $(I \cap J)$ *R*
⟨*proof*⟩

15.3.1 Intersection of a Set of Ideals

The intersection of any Number of Ideals is again an Ideal in R

lemma (**in** *ring*) *i-Intersect*:
  assumes *Sideals*: $\bigwedge I.\ I \in S \Longrightarrow$ *ideal I R*
    and *notempty*: $S \neq \{\}$
  shows *ideal* (*Inter S*) *R*
⟨*proof*⟩

15.4 Addition of Ideals

lemma (**in** *ring*) *add-ideals*:
assumes *idealI*: *ideal I R*
and *idealJ*: *ideal J R*
shows *ideal* (*I* <+> *J*) *R*
⟨*proof*⟩

15.5 Ideals generated by a subset of *carrier R*

15.5.1 Generation of Ideals in General Rings

genideal generates an ideal

lemma (**in** *ring*) *genideal-ideal*:
assumes *Scarr*: $S \subseteq$ *carrier R*
shows *ideal* (*Idl S*) *R*
⟨*proof*⟩

lemma (**in** *ring*) *genideal-self*:
assumes $S \subseteq$ *carrier R*
shows $S \subseteq$ *Idl S*
⟨*proof*⟩

lemma (**in** *ring*) *genideal-self'*:
assumes *carr*: $i \in$ *carrier R*
shows $i \in$ *Idl* $\{i\}$
⟨*proof*⟩

genideal generates the minimal ideal

lemma (**in** *ring*) *genideal-minimal*:
assumes *a*: *ideal I R*
and *b*: $S \subseteq I$
shows *Idl* $S \subseteq I$
⟨*proof*⟩

Generated ideals and subsets

lemma (**in** *ring*) *Idl-subset-ideal*:
assumes *Iideal*: *ideal I R*
and *Hcarr*: $H \subseteq$ *carrier R*
shows (*Idl* $H \subseteq I$) = ($H \subseteq I$)
⟨*proof*⟩

lemma (**in** *ring*) *subset-Idl-subset*:
assumes *Icarr*: $I \subseteq$ *carrier R*
and *HI*: $H \subseteq I$
shows *Idl* $H \subseteq$ *Idl* I
⟨*proof*⟩

lemma (**in** *ring*) *Idl-subset-ideal'*:

assumes *acarr*: $a \in carrier\ R$ **and** *bcarr*: $b \in carrier\ R$
shows $(Idl\ \{a\} \subseteq Idl\ \{b\}) = (a \in Idl\ \{b\})$
⟨*proof*⟩

lemma (**in** *ring*) *genideal-zero*:
$Idl\ \{\mathbf{0}\} = \{\mathbf{0}\}$
⟨*proof*⟩

lemma (**in** *ring*) *genideal-one*:
$Idl\ \{\mathbf{1}\} = carrier\ R$
⟨*proof*⟩

15.5.2 Generation of Principal Ideals in Commutative Rings

constdefs (**structure** *R*)
cgenideal :: $('a, 'b)$ *monoid-scheme* $\Rightarrow$ $'a \Rightarrow 'a$ *set* (*PIdl*ı - [*80*] *79*)
$cgenideal\ R\ a \equiv \{\ x \otimes a \mid x.\ x \in carrier\ R\ \}$

genhideal (?) really generates an ideal

lemma (**in** *cring*) *cgenideal-ideal*:
assumes *acarr*: $a \in carrier\ R$
shows *ideal* (*PIdl* *a*) *R*
⟨*proof*⟩

lemma (**in** *ring*) *cgenideal-self*:
assumes *icarr*: $i \in carrier\ R$
shows $i \in PIdl\ i$
⟨*proof*⟩

cgenideal is minimal

lemma (**in** *ring*) *cgenideal-minimal*:
includes *ideal J R*
assumes *aJ*: $a \in J$
shows $PIdl\ a \subseteq J$
⟨*proof*⟩

lemma (**in** *cring*) *cgenideal-eq-genideal*:
assumes *icarr*: $i \in carrier\ R$
shows $PIdl\ i = Idl\ \{i\}$
⟨*proof*⟩

lemma (**in** *cring*) *cgenideal-eq-rcos*:
$PIdl\ i = carrier\ R\ \#\!>\ i$
⟨*proof*⟩

lemma (**in** *cring*) *cgenideal-is-principalideal*:
assumes *icarr*: $i \in carrier\ R$
shows *principalideal* (*PIdl* *i*) *R*
⟨*proof*⟩

15.6 Union of Ideals

lemma (**in** *ring*) *union-genideal*:
 assumes *idealI*: *ideal I R*
  and *idealJ*: *ideal J R*
 shows $Idl\ (I \cup J) = I <\!+\!> J$
⟨*proof*⟩

15.7 Properties of Principal Ideals

0 generates the zero ideal

lemma (**in** *ring*) *zero-genideal*:
 shows $Idl\ \{\mathbf{0}\} = \{\mathbf{0}\}$
⟨*proof*⟩

1 generates the unit ideal

lemma (**in** *ring*) *one-genideal*:
 shows $Idl\ \{\mathbf{1}\} = carrier\ R$
⟨*proof*⟩

The zero ideal is a principal ideal

corollary (**in** *ring*) *zeropideal*:
 shows $principalideal\ \{\mathbf{0}\}\ R$
⟨*proof*⟩

The unit ideal is a principal ideal

corollary (**in** *ring*) *onepideal*:
 shows $principalideal\ (carrier\ R)\ R$
⟨*proof*⟩

Every principal ideal is a right coset of the carrier

lemma (**in** *principalideal*) *rcos-generate*:
 includes *cring*
 shows $\exists x{\in}I.\ I = carrier\ R\ \#\!>\ x$
⟨*proof*⟩

15.8 Prime Ideals

lemma (**in** *ideal*) *primeidealCD*:
 includes *cring*
 assumes *notprime*: $\neg\ primeideal\ I\ R$
 shows $carrier\ R = I \lor (\exists\, a\ b.\ a \in carrier\ R \land b \in carrier\ R \land a \otimes b \in I \land a \notin I \land b \notin I)$
⟨*proof*⟩

lemma (**in** *ideal*) *primeidealCE*:
 includes *cring*
 assumes *notprime*: $\neg\ primeideal\ I\ R$

obtains *carrier* $R = I$
  | $\exists\, a\ b.\ a \in carrier\ R \wedge b \in carrier\ R \wedge a \otimes b \in I \wedge a \notin I \wedge b \notin I$
⟨*proof*⟩

If $\{\mathbf{0}\}$ is a prime ideal of a commutative ring, the ring is a domain

lemma (**in** *cring*) *zeroprimeideal-domainI*:
  assumes *pi*: *primeideal* $\{\mathbf{0}\}$ *R*
  shows *domain R*
⟨*proof*⟩

corollary (**in** *cring*) *domain-eq-zeroprimeideal*:
  shows *domain* $R =$ *primeideal* $\{\mathbf{0}\}$ *R*
⟨*proof*⟩

15.9 Maximal Ideals

lemma (**in** *ideal*) *helper-I-closed*:
  assumes *carr*: $a \in carrier\ R\ x \in carrier\ R\ y \in carrier\ R$
    and *axI*: $a \otimes x \in I$
  shows $a \otimes (x \otimes y) \in I$
⟨*proof*⟩

lemma (**in** *ideal*) *helper-max-prime*:
  includes *cring*
  assumes *acarr*: $a \in carrier\ R$
  shows *ideal* $\{x{\in}carrier\ R.\ a \otimes x \in I\}$ *R*
⟨*proof*⟩

In a cring every maximal ideal is prime

lemma (**in** *cring*) *maximalideal-is-prime*:
  includes *maximalideal*
  shows *primeideal I R*
⟨*proof*⟩

15.10 Derived Theorems Involving Ideals

— A non-zero cring that has only the two trivial ideals is a field
lemma (**in** *cring*) *trivialideals-fieldI*:
  assumes *carrnzero*: $carrier\ R \neq \{\mathbf{0}\}$
    and *haveideals*: $\{I.\ ideal\ I\ R\} = \{\{\mathbf{0}\},\ carrier\ R\}$
  shows *field R*
⟨*proof*⟩

lemma (**in** *field*) *all-ideals*:
  shows $\{I.\ ideal\ I\ R\} = \{\{\mathbf{0}\},\ carrier\ R\}$
⟨*proof*⟩
lemma (**in** *cring*) *trivialideals-eq-field*:
  assumes *carrnzero*: $carrier\ R \neq \{\mathbf{0}\}$
  shows $(\{I.\ ideal\ I\ R\} = \{\{\mathbf{0}\},\ carrier\ R\}) = field\ R$

⟨*proof*⟩

Like zeroprimeideal for domains

lemma (**in** *field*) *zeromaximalideal*:
  maximalideal {**0**} *R*
⟨*proof*⟩

lemma (**in** *cring*) *zeromaximalideal-fieldI*:
  assumes *zeromax*: *maximalideal* {**0**} *R*
  shows *field R*
⟨*proof*⟩

lemma (**in** *cring*) *zeromaximalideal-eq-field*:
  maximalideal {**0**} *R* = *field R*
⟨*proof*⟩

end

theory *RingHom*
imports *Ideal*
begin

16 Homomorphisms of Non-Commutative Rings

Lifting existing lemmas in a *ring-hom-ring* locale

locale *ring-hom-ring* = *ring R* + *ring S* + *var h* +
  assumes *homh*: *h* ∈ *ring-hom R S*
  notes *hom-mult* [*simp*] = *ring-hom-mult* [*OF homh*]
    and *hom-one* [*simp*] = *ring-hom-one* [*OF homh*]

interpretation *ring-hom-cring* ⊆ *ring-hom-ring*
  ⟨*proof*⟩

interpretation *ring-hom-ring* ⊆ *abelian-group-hom R S*
⟨*proof*⟩

lemma (**in** *ring-hom-ring*) *is-ring-hom-ring*:
  includes *struct R* + *struct S*
  shows *ring-hom-ring R S h*
⟨*proof*⟩

lemma *ring-hom-ringI*:
  includes *ring R* + *ring S*
  assumes
    hom-closed: !!*x*. *x* ∈ *carrier R* ==> *h x* ∈ *carrier S*

and *compatible-mult*: !!x y. [| x : *carrier* R; y : *carrier* R |] ==> h ($x \otimes y$) = h $x \otimes_S h$ y
and *compatible-add*: !!x y. [| x : *carrier* R; y : *carrier* R |] ==> h ($x \oplus y$) = h $x \oplus_S h$ y
and *compatible-one*: h $\mathbf{1} = \mathbf{1}_S$
shows *ring-hom-ring* R S h
⟨*proof*⟩

lemma *ring-hom-ringI2*:
includes *ring* R + *ring* S
assumes h: $h \in$ *ring-hom* R S
shows *ring-hom-ring* R S h
⟨*proof*⟩

lemma *ring-hom-ringI3*:
includes *abelian-group-hom* R S + *ring* R + *ring* S
assumes *compatible-mult*: !!x y. [| x : *carrier* R; y : *carrier* R |] ==> h ($x \otimes y$) = h $x \otimes_S h$ y
and *compatible-one*: h $\mathbf{1} = \mathbf{1}_S$
shows *ring-hom-ring* R S h
⟨*proof*⟩

lemma *ring-hom-cringI*:
includes *ring-hom-ring* R S h + *cring* R + *cring* S
shows *ring-hom-cring* R S h
⟨*proof*⟩

16.1 The kernel of a ring homomorphism

— the kernel of a ring homomorphism is an ideal
lemma (**in** *ring-hom-ring*) *kernel-is-ideal*:
shows *ideal* (*a-kernel* R S h) R
⟨*proof*⟩

Elements of the kernel are mapped to zero

lemma (**in** *abelian-group-hom*) *kernel-zero* [*simp*]:
$i \in$ *a-kernel* R S $h \Longrightarrow h\ i = \mathbf{0}_S$
⟨*proof*⟩

16.2 Cosets

Cosets of the kernel correspond to the elements of the image of the homomorphism

lemma (**in** *ring-hom-ring*) *rcos-imp-homeq*:
assumes *acarr*: $a \in$ *carrier* R
and *xrcos*: $x \in$ *a-kernel* R S h +> a
shows $h\ x = h\ a$
⟨*proof*⟩

lemma (**in** *ring-hom-ring*) *homeq-imp-rcos*:
  assumes *acarr*: $a \in carrier\ R$
    and *xcarr*: $x \in carrier\ R$
    and *hx*: $h\ x = h\ a$
  shows $x \in a\text{-}kernel\ R\ S\ h\ {+>}\ a$
$\langle proof \rangle$

corollary (**in** *ring-hom-ring*) *rcos-eq-homeq*:
  assumes *acarr*: $a \in carrier\ R$
  shows $(a\text{-}kernel\ R\ S\ h)\ {+>}\ a = \{x \in carrier\ R.\ h\ x = h\ a\}$
$\langle proof \rangle$

end

17 QuotRing: Quotient Rings

theory *QuotRing*
imports *RingHom*
begin

17.1 Multiplication on Cosets

constdefs (**structure** *R*)
  $rcoset\text{-}mult :: [('a, \text{-})\ ring\text{-}scheme, 'a\ set, 'a\ set, 'a\ set] \Rightarrow 'a\ set$
    $([mod\ \text{-}:]\ \text{-}\ \bigotimes\iota\ \text{-}\ [81,81,81]\ 80)$
  $rcoset\text{-}mult\ R\ I\ A\ B \equiv \bigcup a{\in}A.\ \bigcup b{\in}B.\ I\ {+>}\ (a \otimes b)$

rcoset-mult fulfils the properties required by congruences

lemma (**in** *ideal*) *rcoset-mult-add*:
  $[\![x \in carrier\ R;\ y \in carrier\ R]\!] \Longrightarrow [mod\ I:]\ (I\ {+>}\ x) \bigotimes (I\ {+>}\ y) = I\ {+>}\ (x \otimes y)$
$\langle proof \rangle$

17.2 Quotient Ring Definition

constdefs (**structure** *R*)
  $FactRing :: [('a,'b)\ ring\text{-}scheme, 'a\ set] \Rightarrow ('a\ set)\ ring$
    (**infixl** *Quot 65*)
  $FactRing\ R\ I \equiv$
    $(\!|carrier = a\text{-}rcosets\ I,\ mult = rcoset\text{-}mult\ R\ I,\ one = (I\ {+>}\ \mathbf{1}),\ zero = I,\ add = set\text{-}add\ R|\!)$

17.3 Factorization over General Ideals

The quotient is a ring

lemma (**in** *ideal*) *quotient-is-ring*:
  shows $ring\ (R\ Quot\ I)$

⟨*proof*⟩

This is a ring homomorphism

lemma (**in** *ideal*) *rcos-ring-hom*:
 $(op +> I) \in ring\text{-}hom\ R\ (R\ Quot\ I)$
⟨*proof*⟩

lemma (**in** *ideal*) *rcos-ring-hom-ring*:
 $ring\text{-}hom\text{-}ring\ R\ (R\ Quot\ I)\ (op +> I)$
⟨*proof*⟩

The quotient of a cring is also commutative

lemma (**in** *ideal*) *quotient-is-cring*:
 includes *cring*
 shows *cring* $(R\ Quot\ I)$
⟨*proof*⟩

Cosets as a ring homomorphism on crings

lemma (**in** *ideal*) *rcos-ring-hom-cring*:
 includes *cring*
 shows $ring\text{-}hom\text{-}cring\ R\ (R\ Quot\ I)\ (op +> I)$
⟨*proof*⟩

17.4 Factorization over Prime Ideals

The quotient ring generated by a prime ideal is a domain

lemma (**in** *primeideal*) *quotient-is-domain*:
 shows *domain* $(R\ Quot\ I)$
⟨*proof*⟩

Generating right cosets of a prime ideal is a homomorphism on commutative rings

lemma (**in** *primeideal*) *rcos-ring-hom-cring*:
 shows $ring\text{-}hom\text{-}cring\ R\ (R\ Quot\ I)\ (op +> I)$
⟨*proof*⟩

17.5 Factorization over Maximal Ideals

In a commutative ring, the quotient ring over a maximal ideal is a field. The proof follows "W. Adkins, S. Weintraub: Algebra – An Approach via Module Theory"

lemma (**in** *maximalideal*) *quotient-is-field*:
 includes *cring*
 shows *field* $(R\ Quot\ I)$
⟨*proof*⟩

end

theory *IntRing*
imports *QuotRing IntDef*
begin

18 The Ring of Integers

18.1 Some properties of *int*

lemma *dvds-imp-abseq*:
$[\![l \ dvd \ k;\ k \ dvd \ l]\!] \Longrightarrow abs\ l = abs\ (k{::}int)$
$\langle proof \rangle$

lemma *abseq-imp-dvd*:
assumes *a-lk*: $abs\ l = abs\ (k{::}int)$
shows $l\ dvd\ k$
$\langle proof \rangle$

lemma *dvds-eq-abseq*:
$(l\ dvd\ k \wedge k\ dvd\ l) = (abs\ l = abs\ (k{::}int))$
$\langle proof \rangle$

18.2 The Set of Integers as Algebraic Structure

18.2.1 Definition of $\mathcal{Z}$

constdefs
$int\text{-}ring :: int\ ring\ (\mathcal{Z})$
$int\text{-}ring \equiv (\!|carrier = UNIV,\ mult = op\ *,\ one = 1,\ zero = 0,\ add = op\ +|\!)$

lemma *int-Zcarr* [*intro*!, *simp*]:
$k \in carrier\ \mathcal{Z}$
$\langle proof \rangle$

lemma *int-is-cring*:
$cring\ \mathcal{Z}$
$\langle proof \rangle$

18.2.2 Interpretations

Since definitions of derived operations are global, their interpretation needs to be done as early as possible — that is, with as few assumptions as possible.

interpretation *int*: *monoid* $[\mathcal{Z}]$
where $carrier\ \mathcal{Z} = UNIV$
and $mult\ \mathcal{Z}\ x\ y = x * y$
and $one\ \mathcal{Z} = 1$
and $pow\ \mathcal{Z}\ x\ n = x\hat{}n$

⟨*proof*⟩

interpretation *int*: *comm-monoid* [$\mathcal{Z}$]
where *finprod* $\mathcal{Z}$ *f A* = (*if finite A then setprod f A else arbitrary*)
⟨*proof*⟩

interpretation *int*: *abelian-monoid* [$\mathcal{Z}$]
where *zero* $\mathcal{Z}$ = *0*
and *add* $\mathcal{Z}$ x y = $x + y$
and *finsum* $\mathcal{Z}$ *f A* = (*if finite A then setsum f A else arbitrary*)
⟨*proof*⟩

interpretation *int*: *abelian-group* [$\mathcal{Z}$]
where *a-inv* $\mathcal{Z}$ $x = -\ x$
and *a-minus* $\mathcal{Z}$ $x\ y = x\ -\ y$
⟨*proof*⟩

interpretation *int*: *domain* [$\mathcal{Z}$]
⟨*proof*⟩

Removal of occurrences of *UNIV* in interpretation result — experimental.

lemma *UNIV*:
$x \in$ *UNIV* = *True*
$A \subseteq$ *UNIV* = *True*
(*ALL x* : *UNIV*. *P x*) = (*ALL x*. *P x*)
(*EX x* : *UNIV*. *P x*) = (*EX x*. *P x*)
(*True* −−> *Q*) = *Q*
(*True* ==> *PROP R*) == *PROP R*
⟨*proof*⟩

interpretation *int* [*unfolded UNIV*]:
partial-order [(| *carrier* = *UNIV*::*int set*, *le* = *op* ≤ |)]
where *carrier* (| *carrier* = *UNIV*::*int set*, *le* = *op* ≤ |) = *UNIV*
and *le* (| *carrier* = *UNIV*::*int set*, *le* = *op* ≤ |) *x y* = (*x* ≤ *y*)
and *lless* (| *carrier* = *UNIV*::*int set*, *le* = *op* ≤ |) *x y* = (*x* < *y*)
⟨*proof*⟩

interpretation *int* [*unfolded UNIV*]:
lattice [(| *carrier* = *UNIV*::*int set*, *le* = *op* ≤ |)]
where *join* (| *carrier* = *UNIV*::*int set*, *le* = *op* ≤ |) *x y* = *max x y*
and *meet* (| *carrier* = *UNIV*::*int set*, *le* = *op* ≤ |) *x y* = *min x y*
⟨*proof*⟩

interpretation *int* [*unfolded UNIV*]:
total-order [(| *carrier* = *UNIV*::*int set*, *le* = *op* ≤ |)]
⟨*proof*⟩

18.2.3 Generated Ideals of $\mathcal{Z}$

lemma *int-Idl*:
$Idl_{\mathcal{Z}}\ \{a\} = \{x * a \mid x.\ True\}$
⟨*proof*⟩

lemma *multiples-principalideal*:
principalideal $\{x * a \mid x.\ True\ \}\ \mathcal{Z}$
⟨*proof*⟩

lemma *prime-primeideal*:
assumes *prime*: *prime* $(nat\ p)$
shows *primeideal* $(Idl_{\mathcal{Z}}\ \{p\})\ \mathcal{Z}$
⟨*proof*⟩

18.2.4 Ideals and Divisibility

lemma *int-Idl-subset-ideal*:
$Idl_{\mathcal{Z}}\ \{k\} \subseteq Idl_{\mathcal{Z}}\ \{l\} = (k \in Idl_{\mathcal{Z}}\ \{l\})$
⟨*proof*⟩

lemma *Idl-subset-eq-dvd*:
$(Idl_{\mathcal{Z}}\ \{k\} \subseteq Idl_{\mathcal{Z}}\ \{l\}) = (l\ dvd\ k)$
⟨*proof*⟩

lemma *dvds-eq-Idl*:
$(l\ dvd\ k \wedge k\ dvd\ l) = (Idl_{\mathcal{Z}}\ \{k\} = Idl_{\mathcal{Z}}\ \{l\})$
⟨*proof*⟩

lemma *Idl-eq-abs*:
$(Idl_{\mathcal{Z}}\ \{k\} = Idl_{\mathcal{Z}}\ \{l\}) = (abs\ l = abs\ k)$
⟨*proof*⟩

18.2.5 Ideals and the Modulus

constdefs
ZMod :: *int* => *int* => *int set*
$ZMod\ k\ r == (Idl_{\mathcal{Z}}\ \{k\})\ +>_{\mathcal{Z}}\ r$

lemmas *ZMod-defs* =
ZMod-def genideal-def

lemma *rcos-zfact*:
assumes *kIl*: $k \in ZMod\ l\ r$
shows $EX\ x.\ k = x * l + r$
⟨*proof*⟩

lemma *ZMod-imp-zmod*:
assumes *zmods*: $ZMod\ m\ a = ZMod\ m\ b$
shows $a\ mod\ m = b\ mod\ m$

⟨*proof*⟩

lemma *ZMod-mod*:
  shows *ZMod m a* = *ZMod m* (*a mod m*)
⟨*proof*⟩

lemma *zmod-imp-ZMod*:
  assumes *modeq*: *a mod m* = *b mod m*
  shows *ZMod m a* = *ZMod m b*
⟨*proof*⟩

corollary *ZMod-eq-mod*:
  shows (*ZMod m a* = *ZMod m b*) = (*a mod m* = *b mod m*)
⟨*proof*⟩

18.2.6 Factorization

constdefs
  ZFact :: *int* ⇒ *int set ring*
  ZFact k == $\mathcal{Z}$ *Quot* (*Idl*$_{\mathcal{Z}}$ {*k*})

lemmas *ZFact-defs* = *ZFact-def FactRing-def*

lemma *ZFact-is-cring*:
  shows *cring* (*ZFact k*)
⟨*proof*⟩

lemma *ZFact-zero*:
  carrier (*ZFact 0*) = ($\bigcup$ *a*. {{*a*}})
⟨*proof*⟩

lemma *ZFact-one*:
  carrier (*ZFact 1*) = {*UNIV*}
⟨*proof*⟩

lemma *ZFact-prime-is-domain*:
  assumes *pprime*: *prime* (*nat p*)
  shows *domain* (*ZFact p*)
⟨*proof*⟩

end

References

[1] C. Ballarin. *Computer Algebra and Theorem Proving.* PhD thesis, University of Cambridge, 1999. http://www4.in.tum.de/~ballarin/publications.html.

[2] N. Jacobson. *Basic Algebra I.* Freeman, 1985.

[3] F. Kammüller and L. C. Paulson. A formal proof of sylow's theorem: An experiment in abstract algebra with Isabelle HOL. *J. Automated Reasoning*, (23):235–264, 1999.