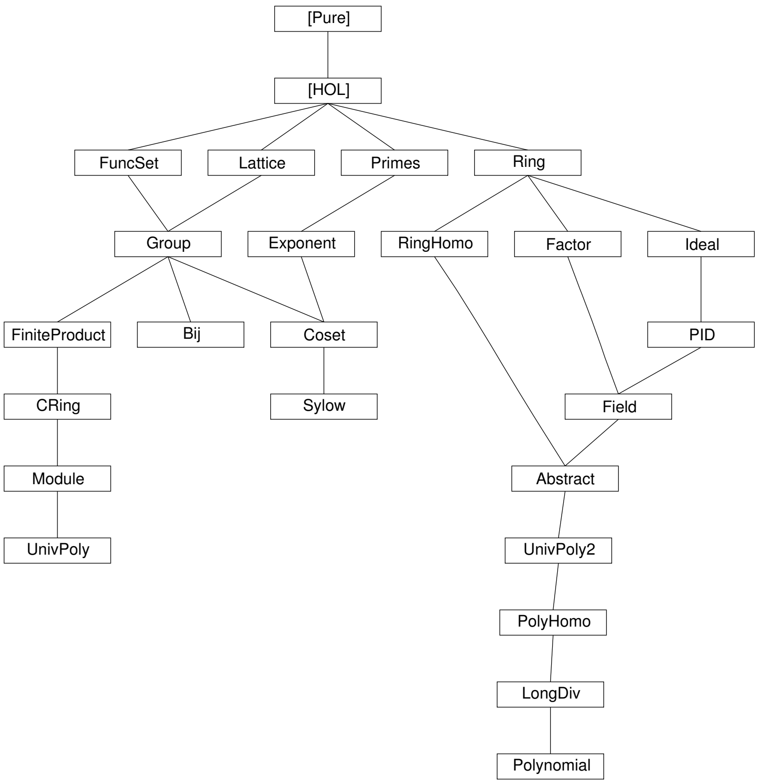
The Isabelle/HOL Algebra Library

Clemens Ballarin
Florian Kammuüller
Lawrence C Paulson

October 1, 2005

Contents


[Pure]
[HOL]
FuncSet
Lattice
Primes
Ring
Group
Exponent
RingHomo
Factor
Ideal
FiniteProduct
Bij
Coset
PID
CRing
Sylow
Field
Module
Abstract
UnivPoly
UnivPoly2
PolyHomo
LongDiv
Polynomial

1 Lattice: Orders and Lattices

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

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** $\sqsubseteq_\imath$ *50*)

locale *partial-order* = *struct L* +
  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*)
  less :: [-, *'a*, *'a*] => *bool* (**infixl** $\sqsubset_\imath$ *50*)
  $x \sqsubset y$ == $x \sqsubseteq y$ & x ~= 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* ($\bigsqcup_\imath$- [*90*] *90*)
  $\bigsqcup A$ == *THE x*. *least L x* (*Upper L A*)

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

  join :: [-, *'a*, *'a*] => *'a* (**infixl** $\sqcup_\imath$ *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*]:
includes *struct L*
shows [| $u \in$ *Upper L A*; $x \in A$; $A \subseteq$ *carrier L* |] ==> $x \sqsubseteq u$
⟨*proof*⟩

lemma *Upper-memI*:
includes *struct L*
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*]:
includes *struct L*
shows [| $l \in$ *Lower L A*; $x \in A$; $A \subseteq$ *carrier L* |] ==> $l \sqsubseteq x$
⟨*proof*⟩

lemma *Lower-memI*:
includes *struct L*
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*:
  includes *struct L*
  shows [| *least L x A*; $a \in A$ |] ==> $x \sqsubseteq a$
  ⟨*proof*⟩

lemma *least-UpperI*:
  includes *struct L*
  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*:
  includes *struct L*
  shows [| *greatest L x A*; $a \in A$ |] ==> $a \sqsubseteq x$
  ⟨*proof*⟩

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

lemma *greatest-Lower-above*:
includes *struct L*
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*]:
includes *struct L*
shows $x \in$ *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*:
[| *finite A*; $A \subseteq$ *carrier L*; A ~= {} |] ==> *least L* ($\bigsqcup A$) (*Upper L A*)
⟨*proof*⟩

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

assumes P: $!!l.$ *least* L l (*Upper* L (*insert* x A)) ==> P l
and xA: *finite* A $x \in$ *carrier* L $A \subseteq$ *carrier* L
shows P ($\bigsqcup$ (*insert* x A))
⟨*proof*⟩

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

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

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

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

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

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

lemma *join-comm*:
includes *struct* L
shows $x \sqcup y = y \sqcup x$
⟨*proof*⟩

lemma (**in** *lattice*) *join-assoc*:
assumes L: $x \in$ *carrier* L $y \in$ *carrier* L $z \in$ *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*]:
includes *struct* L
shows $x \in$ *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 $\{x, y\}$)
⟨*proof*⟩

lemma (**in** *lattice*) *meet-le*:

assumes *sub*: $z \sqsubseteq x \quad z \sqsubseteq y$
and *L*: $x \in$ *carrier* $L \quad y \in$ *carrier* $L \quad z \in$ *carrier* L
shows $z \sqsubseteq x \sqcap y$
⟨*proof*⟩

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

lemma *meet-comm*:
includes *struct L*
shows $x \sqcap y = y \sqcap x$
⟨*proof*⟩

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

1.3 Total Orders

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

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* |] ==> $x \sqsubseteq y$ | $y \sqsubseteq x$
shows *total-order L*
⟨*proof*⟩

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

constdefs (**structure** *L*)

top :: - => *'a* ($\top_1$)
$\top$ == *sup L* (*carrier L*)

bottom :: - => *'a* ($\perp_1$)
$\perp$ == *inf L* (*carrier L*)

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

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

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

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

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

lemma (**in** *complete-lattice*) *bottom-closed* [*simp*, *intro*]:
$\perp$ $\in$ *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* ⊆ |)
  (**is** *complete-lattice ?L*)
⟨*proof*⟩

An other example, that of the lattice of subgroups of a group, can be found in Group theory (Section 3.9).

end

2 Group: Groups

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

3 Monoids and Groups

Definitions follow [2].

3.1 Definitions

record *'a monoid* = *'a partial-object* +
  mult :: [*'a*, *'a*] ⇒ *'a* (**infixl** ⊗ı *70*)
  one :: *'a* (**1**ı)

constdefs (**structure** *G*)
  m-inv :: (*'a*, *'b*) *monoid-scheme* => *'a* => *'a* (*inv*ı - [*81*] *80*)
  inv x == (*THE y. y* ∈ *carrier G* & *x* ⊗ *y* = **1** & *y* ⊗ *x* = **1**)

  Units :: - => *'a set*
  — The set of invertible elements
  Units G == {*y. y* ∈ *carrier G* & (∃ *x* ∈ *carrier G*. *x* ⊗ *y* = **1** & *y* ⊗ *x* = **1**)}

consts
  pow :: [(*'a*, *'m*) *monoid-scheme*, *'a*, *'b*::*number*] => *'a* (**infixr** '(ˆ')ı *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* = *struct G* +
  assumes *m-closed* [*intro*, *simp*]:
    ⟦*x* ∈ *carrier G*; *y* ∈ *carrier G*⟧ ⟹ *x* ⊗ *y* ∈ *carrier G*

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

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

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

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

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

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

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

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

lemma (**in** *monoid*) *Units-inv-Units* [*intro*, *simp*]:
$x \in \textit{Units } G ==> \textit{inv } x \in \textit{Units } G$
⟨*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* ==> x (^) (n::*nat*) $\otimes$ x (^) m = x (^) ($n + m$)
⟨*proof*⟩

lemma (**in** *monoid*) *nat-pow-pow*:
$x \in$ *carrier G* ==> (x (^) n) (^) m = x (^) ($n * m$::*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*:
  includes *struct G*
  assumes *m-closed* [*simp*]:
    $!!x\ y.\ [|\ x \in carrier\ G;\ y \in carrier\ G\ |] ==> x \otimes y \in carrier\ G$
   and *one-closed* [*simp*]: $\mathbf{1} \in carrier\ G$
   and *m-assoc*:
    $!!x\ y\ z.\ [|\ x \in carrier\ G;\ y \in carrier\ G;\ z \in carrier\ G\ |] ==>$
    $(x \otimes y) \otimes z = x \otimes (y \otimes z)$
   and *l-one* [*simp*]: $!!x.\ x \in carrier\ G ==> \mathbf{1} \otimes x = x$
   and *l-inv-ex*: $!!x.\ x \in carrier\ G ==> \exists\, y \in carrier\ G.\ y \otimes x = \mathbf{1}$
  shows *group G*
⟨*proof*⟩

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

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

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

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

3.2 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*:
$a\ (\hat{}\ )\ (z::int) = (if\ neg\ z\ then\ inv\ (a\ (\hat{}\ )\ (nat\ (-z)))\ else\ a\ (\hat{}\ )\ (nat\ z))$
⟨*proof*⟩

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

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

3.3 Subgroups

locale *subgroup* = *var H* + *struct G* +
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*]: $x \in H \Longrightarrow inv\ x \in H$

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*:
  $\sim\ subgroup\ \{\}\ G$
  ⟨*proof*⟩

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

3.4 Direct Products

constdefs
  $DirProd :: \text{-} \Rightarrow \text{-} \Rightarrow ('a \times 'b)\ monoid$ (**infixr** $\times\times$ *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*⟩

3.5 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 |]

$\texttt{==>}\ h\ (x \otimes_G y) = h\ x \otimes_H h\ y$
$\langle proof \rangle$

lemma *hom-closed*:
$[|\ h \in hom\ G\ H;\ x \in carrier\ G\ |]\ \texttt{==>}\ h\ x \in carrier\ H$
$\langle proof \rangle$

lemma (**in** *group*) *hom-compose*:
$[|h \in hom\ G\ H;\ i \in hom\ H\ I|]\ \texttt{==>}\ compose\ (carrier\ G)\ i\ h \in hom\ G\ I$
$\langle proof \rangle$

3.6 Isomorphisms

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

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

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

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

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

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

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*]:
$h\ \mathbf{1} \in carrier\ H$
$\langle proof \rangle$

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*⟩

3.7 Commutative Structures

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

3.8 Definition

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*:
includes *struct G*
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*:
includes *struct G*
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*⟩

3.9 Lattice of subgroups of a group

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

end

4 FiniteProduct: Product Operator for Commutative Monoids

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

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.

consts
foldSetD :: ['*a set*, '*b* => '*a* => '*a*, '*a*] => ('*b set* * '*a*) *set*

inductive *foldSetD D f e*
intros
emptyI [*intro*]: $e \in D$ ==> ({}, *e*) ∈ *foldSetD D f e*
insertI [*intro*]: [| *x* ~: *A*; *f x y* ∈ *D*; (*A*, *y*) ∈ *foldSetD D f e* |] ==>
(*insert x A*, *f x y*) ∈ *foldSetD D f e*

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

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

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

lemma *Diff1-foldSetD*:

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

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

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

4.1 Left-commutative operations

locale *LCD* =
fixes $B :: 'b\ set$
and $D :: 'a\ set$
and $f :: 'b => 'a => 'a$ (**infixl** $\cdot$ *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*]:
$(A, z) \in \mathit{foldSetD}\ D\ f\ e ==> z \in D$
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

lemma *Int-mono2*:
[| *A* ⊆ *C*; *B* ⊆ *C* |] ==> *A Int B* ⊆ *C*
⟨*proof*⟩

lemma (**in** *LCD*) *foldD-nest-Un-Int*:
[| *finite A*; *finite C*; *e* ∈ *D*; *A* ⊆ *B*; *C* ⊆ *B* |] ==>
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* ∈ *D*; *A* ⊆ *B*; *C* ⊆ *B* |]
==> *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*]

4.2 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** · *70*)
    and e :: $'a$
  assumes *ident* [*simp*]: $x \in D$ ==> $x \cdot e = x$
    and *commute*: [| $x \in D$; $y \in D$ |] ==> $x \cdot y = y \cdot x$
    and *assoc*: [| $x \in D$; $y \in D$; $z \in D$ |] ==> $(x \cdot y) \cdot z = x \cdot (y \cdot z)$
    and *e-closed* [*simp*]: $e \in D$
    and *f-closed* [*simp*]: [| $x \in D$; $y \in D$ |] ==> $x \cdot y \in D$

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

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

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

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

4.3 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
⨂1*i*:*A*. *b* == *finprod* ⋄1 (%*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* ∉ *F*; *f* ∈ *F* −> *carrier G*; *f a* ∈ *carrier G* |] ==>
finprod G f (*insert a F*) = *f a* ⊗ *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* ∈ *A* −> *carrier G*
shows *finprod G f A* ∈ *carrier G*
⟨*proof*⟩

lemma *funcset-Int-left* [*simp*, *intro*]:
[| *f* ∈ *A* −> *C*; *f* ∈ *B* −> *C* |] ==> *f* ∈ *A Int B* −> *C*
⟨*proof*⟩

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

lemma (**in** *comm-monoid*) *finprod-Un-Int*:
[| *finite A*; *finite B*; *g* ∈ *A* −> *carrier G*; *g* ∈ *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 ∈ *A* −> *carrier G*; *g* ∈ *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. $f\ x \otimes g\ x$) A = (*finprod G f A* $\otimes$ *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*) $\otimes$ *finprod G f* {..*n*})
⟨*proof*⟩

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

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

end

5 Exponent: The Combinatorial Argument Underlying the First Sylow Theorem

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

constdefs
  exponent :: [*nat*, *nat*] => *nat*
  exponent p s == *if prime p then* (*GREATEST r*. *p*ˆ*r dvd s*) *else 0*

5.1 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*ˆ*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*
  ==> ∀ *m n*. *p*ˆ*c dvd m*∗*n* −−>
    (∀ *a b*. *a*+*b* = *Suc c* −−> *p*ˆ*a dvd m* | *p*ˆ*b dvd n*)
⟨*proof*⟩

lemma *div-combine*:
  [| *prime p*; ~ (*p* ˆ (*Suc r*) *dvd n*); *p*ˆ(*a*+*r*) *dvd n*∗*k* |]
  ==> *p* ˆ *a dvd k*
⟨*proof*⟩

lemma *Suc-le-power*: *Suc 0* < *p* ==> *Suc n* <= *p*ˆ*n*
⟨*proof*⟩

lemma *power-dvd-bound*: [|*p*ˆ*n dvd a*; *Suc 0* < *p*; *0* < *a*|] ==> *n* < *a*

⟨*proof*⟩

5.2 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*: $0<s ==> (p\ \hat{}\ exponent\ p\ s)\ dvd\ s$
⟨*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*:
  $[|\ 0 < a;\ 0 < b\ |]$
  $==> (exponent\ p\ a) + (exponent\ p\ b) <= exponent\ p\ (a * b)$
⟨*proof*⟩

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

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

5.3 Lemmas for the Main Combinatorial Argument

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

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

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

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

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

$==> (p\hat{}r)\ dvd\ (p\hat{}a)*m - k$
⟨*proof*⟩

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

lemma *const-p-fac*:

0 < m ==> exponent p (((p ^a) ∗ m) choose p ^a) = exponent p m
⟨*proof*⟩

end

6 Coset: Cosets and Quotient Groups

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

constdefs (**structure** *G*)
r-coset :: [-, *'a set*, *'a*] ⇒ *'a set* (**infixl** #>ı *60*)
H #> *a* ≡ ⋃*h*∈*H*. {*h* ⊗ *a*}

l-coset :: [-, *'a*, *'a set*] ⇒ *'a set* (**infixl** <#ı *60*)
a <# *H* ≡ ⋃*h*∈*H*. {*a* ⊗ *h*}

RCOSETS :: [-, *'a set*] ⇒ (*'a set*)*set* (*rcosets*ı - [*81*] *80*)
rcosets H ≡ ⋃*a*∈*carrier G*. {*H* #> *a*}

set-mult :: [-, *'a set* ,*'a set*] ⇒ *'a set* (**infixl** <#>ı *60*)
H <#> *K* ≡ ⋃*h*∈*H*. ⋃*k*∈*K*. {*h* ⊗ *k*}

SET-INV :: [-,*'a set*] ⇒ *'a set* (*set'-inv*ı - [*81*] *80*)
set-inv H ≡ ⋃*h*∈*H*. {*inv h*}

locale *normal* = *subgroup* + *group* +
assumes *coset-eq*: (∀ *x* ∈ *carrier G*. *H* #> *x* = *x* <# *H*)

syntax
@*normal* :: [*'a set*, (*'a*, *'b*) *monoid-scheme*] ⇒ *bool* (**infixl** ⊲ *60*)

translations
H ⊲ *G* == *normal H G*

6.1 Basic Properties of Cosets

lemma (**in** *group*) *coset-mult-assoc*:
[| *M* ⊆ *carrier G*; *g* ∈ *carrier G*; *h* ∈ *carrier G* |]
==> (*M* #> *g*) #> *h* = *M* #> (*g* ⊗ *h*)
⟨*proof*⟩

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

lemma (**in** *group*) *coset-mult-inv1*:

$[| \ M \ \#> (x \otimes (inv \ y)) = M; \ \ x \in carrier \ G \ ; \ y \in carrier \ G;$
$\quad M \subseteq carrier \ G \ |] ==> M \ \#> x = M \ \#> y$
$\langle proof \rangle$

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

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

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

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

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

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

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

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*: $[| \ x \in carrier \ G; \ subgroup \ H \ G \ |] ==> x \in H \ \#> x$
$\langle proof \rangle$

6.2 Normal subgroups

lemma *normal-imp-subgroup*: $H \lhd G \Longrightarrow subgroup\ H\ G$
⟨*proof*⟩

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

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

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*⟩

6.3 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: $x \in \mathit{carrier}\ G$ **and** sb: $\mathit{subgroup}\ H\ G$
shows $x <\# H = y <\# H$
⟨*proof*⟩

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

lemma (**in** *group*) *subgroup-mult-id*: $\mathit{subgroup}\ H\ G \Longrightarrow H <\#> H = H$
⟨*proof*⟩

6.3.1 Set of inverses of an *r-coset*.

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

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

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

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

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

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

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

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

⟨*proof*⟩

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

6.3.3 An Equivalence Relation

constdefs (**structure** G)
r-congruent :: $[('a,'b)monoid\text{-}scheme,\ 'a\ set] \Rightarrow ('a{*}'a)set$
(*rcong*ı -)
$rcong\ H \equiv \{(x,y).\ x \in carrier\ G\ \&\ y \in carrier\ G\ \&\ inv\ x \otimes y \in H\}$

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*: $a \in carrier\ G$
shows $a <\#\ H = rcong\ H\ ``\ \{a\}$
⟨*proof*⟩

6.3.4 Two distinct right cosets are disjoint

lemma (**in** *group*) *rcos-equation*:
includes *subgroup H G*
shows
$[\![ha \otimes a = h \otimes b;\ a \in carrier\ G;\ b \in carrier\ G;$
$h \in H;\ ha \in H;\ hb \in H]\!]$
$\Longrightarrow hb \otimes a \in (\bigcup h{\in}H.\ \{h \otimes b\})$
⟨*proof*⟩

lemma (**in** *group*) *rcos-disjoint*:
includes *subgroup H G*
shows $[\![a \in rcosets\ H;\ b \in rcosets\ H;\ a{\neq}b]\!] \Longrightarrow a \cap b = \{\}$
⟨*proof*⟩

6.4 Order of a Group and Lagrange's Theorem

constdefs
order :: $('a,\ 'b)\ monoid\text{-}scheme \Rightarrow nat$
$order\ S \equiv card\ (carrier\ S)$

lemma (**in** *group*) *rcos-self*:

includes *subgroup*
shows $x \in carrier\ G \Longrightarrow x \in H\ \#{>}\ x$
⟨*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*⟩

6.5 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*:
  $K \in rcosets\ H \Longrightarrow set\text{-}inv\ K \in rcosets\ H$
$\langle proof \rangle$

lemma (**in** *normal*) *rcosets-assoc*:
  $[\![M1 \in rcosets\ H;\ M2 \in rcosets\ H;\ M3 \in rcosets\ H]\!]$
  $\Longrightarrow M1 <\#> M2 <\#> M3 = M1 <\#> (M2 <\#> M3)$
$\langle proof \rangle$

lemma (**in** *subgroup*) *subgroup-in-rcosets*:
 includes *group G*
 shows $H \in rcosets\ H$
$\langle proof \rangle$

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

theorem (**in** *normal*) *factorgroup-is-group*:
 $group\ (G\ Mod\ H)$
$\langle proof \rangle$

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

lemma (**in** *normal*) *inv-FactGroup*:
  $X \in carrier\ (G\ Mod\ H) \Longrightarrow inv_{G\ Mod\ H}\ X = set\text{-}inv\ X$
$\langle proof \rangle$

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

lemma (**in** *normal*) *r-coset-hom-Mod*:
 $(\lambda a.\ H \#> a) \in hom\ G\ (G\ Mod\ H)$
 $\langle proof \rangle$

6.6 The First Isomorphism Theorem

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

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

lemma (**in** *group-hom*) *subgroup-kernel*: $subgroup\ (kernel\ G\ H\ h)\ G$
$\langle proof \rangle$

The kernel of a homomorphism is a normal subgroup

lemma (**in** *group-hom*) *normal-kernel*: $(kernel\ G\ H\ h) \lhd G$
⟨*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*:
$inj\text{-}on\ (\lambda X.\ contents\ (h\ `\ X))\ (carrier\ (G\ Mod\ kernel\ G\ H\ h))$
⟨*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

7 Sylow: Sylow's theorem

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

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*⟩

7.1 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*⟩

7.2 Discharging the Assumptions of *sylow-central*

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

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

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

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

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

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

7.2.1 Introduction and Destruct Rules for *H*

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

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

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

7.3 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*:
$(\%x{:}M.\ H\ \#>\ (SOME\ g.\ g \in carrier\ G\ \&\ M1\ \#>\ g = x)) \in M \rightarrow rcosets\ H$
⟨*proof*⟩

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

7.3.1 The opposite injection

lemma (**in** *sylow-central*) *H-elem-map*:
$H1 \in rcosets\ H ==> \exists g.\ g \in carrier\ G\ \&\ H\ \#>\ g = H1$
$\langle proof \rangle$

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*:
$[|equiv\ A\ r;\ M \in A//r;\ M1{\in}M;\ (M1,M2) \in r\ |] ==> M2 \in M$
$\langle proof \rangle$

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

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

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

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

lemma (**in** *sylow-central*) *finite-M*: *finite M*
$\langle proof \rangle$

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

lemma (**in** *sylow-central*) *index-lem*: $card(M) * card(H) = order(G)$
$\langle proof \rangle$

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

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

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

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

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

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

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

end

8 Bij: Bijections of a Set, Permutation Groups, Automorphism Groups

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

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*: $f \in Bij\ S \Longrightarrow f \in extensional\ S$
$\langle proof \rangle$

lemma *Bij-imp-funcset*: $f \in Bij\ S \Longrightarrow f \in S \to S$
$\langle proof \rangle$

8.1 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*⟩

8.2 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

9 CRing: Abelian Groups

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

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* |)

 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* = *struct G* +
 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* |)

9.1 Basic Properties

lemma *abelian-monoidI*:
 includes *struct R*
 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*:
includes *struct R*
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*]:
$[|\ x \in carrier\ G;\ y \in carrier\ G;\ z \in carrier\ G\ |] ==>$

$(x \oplus y = x \oplus z) = (y = z)$
$\langle proof \rangle$

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

lemma (**in** *abelian-monoid*) *a-assoc*:
$[\![x \in carrier\ G;\ y \in carrier\ G;\ z \in carrier\ G]\!] \Longrightarrow$
$(x \oplus y) \oplus z = x \oplus (y \oplus z)$
$\langle proof \rangle$

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

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

lemma (**in** *abelian-monoid*) *a-comm*:
$[\![x \in carrier\ G;\ y \in carrier\ G]\!] \Longrightarrow x \oplus y = y \oplus x$
$\langle proof \rangle$

lemma (**in** *abelian-monoid*) *a-lcomm*:
$[\![x \in carrier\ G;\ y \in carrier\ G;\ z \in carrier\ G]\!] \Longrightarrow$
$x \oplus (y \oplus z) = y \oplus (x \oplus z)$
$\langle proof \rangle$

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

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

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

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

lemma (**in** *abelian-group*) *a-inv-inj*:
$inj\text{-}on\ (a\text{-}inv\ G)\ (carrier\ G)$
$\langle proof \rangle$

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

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

9.2 Sums over Finite Sets

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

constdefs
finsum :: $[('b,\ 'm)\ ring\text{-}scheme,\ 'a => 'b,\ 'a\ set] => 'b$
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*
$((3\bigoplus$ --∈-. -) [1000, 0, 51, 10] 10)
syntax (*HTML* **output**)
-finsum :: *index* => *idt* => *'a set* => *'b* => *'b*
$((3\bigoplus$ --∈-. -) [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 {} = **0**
$\langle proof \rangle$

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

lemma (**in** *abelian-monoid*) *finsum-zero* [*simp*]:
finite A ==> ($\bigoplus i \in A.$ **0**) = **0**
$\langle proof \rangle$

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

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.

10 The Algebraic Hierarchy of Rings

10.1 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* $R - \{\mathbf{0}\}$

10.2 Basic Facts of Rings

lemma *ringI*:
  includes *struct* R
  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 *cringI*:
  includes *struct* R
  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*⟩

10.3 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** *ring*) *minus-eq*:
$[\| x \in carrier\ R; y \in carrier\ R \|] ==> x \ominus y = x \oplus \ominus y$
⟨*proof*⟩

lemmas (**in** *ring*) *ring-simprules* =
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*) *cring-simprules* =
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

⟨*ML*⟩

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

10.4 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*⟩

10.5 Facts of Integral Domains

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

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

lemma (**in** *domain*) *m-lcancel*:
assumes *prem*: a ~= **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 ~= **0**

  and R: $a \in carrier\ R\ b \in carrier\ R\ c \in carrier\ R$
 shows *conc*: $(b \otimes a = c \otimes a) = (b = c)$
$\langle proof \rangle$

10.6 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*:
 includes *struct* R + *struct* S
 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 $h \in ring\text{-}hom\ R\ S$
 $\langle proof \rangle$

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

lemma *ring-hom-mult*:
 includes *struct* R + *struct* S
 shows
  $[|\ h \in ring\text{-}hom\ R\ S; x \in carrier\ R; y \in carrier\ R\ |] ==>$
  $h\ (x \otimes y) = h\ x \otimes_S h\ y$
  $\langle proof \rangle$

lemma *ring-hom-add*:
 includes *struct* R + *struct* S
 shows
  $[|\ h \in ring\text{-}hom\ R\ S; x \in carrier\ R; y \in carrier\ R\ |] ==>$
  $h\ (x \oplus y) = h\ x \oplus_S h\ y$
  $\langle proof \rangle$

lemma *ring-hom-one*:
 includes *struct* R + *struct* S
 shows $h \in ring\text{-}hom\ R\ S ==> h\ \mathbf{1} = \mathbf{1}_S$
 $\langle proof \rangle$

locale *ring-hom-cring* = *cring* R + *cring* S + *var* h +
 assumes *homh* [*simp*, *intro*]: $h \in ring\text{-}hom\ R\ S$

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*]:
$x \in carrier\ R ==> h\ (\ominus\ x) = \ominus_S\ h\ x$
⟨*proof*⟩

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

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

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

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

end

11 Module: Modules over an Abelian Group

theory *Module* **imports** *CRing* **begin**

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*:
includes *struct R* + *struct M*
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*:
includes *struct R* + *struct M*
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*⟩

11.1 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

12 UnivPoly: Univariate Polynomials

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

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.

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

Properties of the set of polynomials *up*.

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

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

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

lemma (**in** *cring*) *up-one-closed*:
   (%*n*. *if n* = *0 then* **1** *else* **0**) ∈ *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*⟩

12.2 Effect of operations on coefficients

locale *UP* = *struct R* + *struct P* +
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*⟩

12.3 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*⟩

12.4 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*⟩

12.5 Further lemmas involving monomials

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

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

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

lemma (**in** *UP-cring*) *monom-one-Suc*:
monom P **1** (*Suc n*) = *monom P* **1** *n* $\otimes_P$ *monom P* **1** *1*
⟨*proof*⟩

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

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

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

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

lemma (**in** *UP-cring*) *monom-a-inv* [*simp*]:
a ∈ *carrier R* ==> *monom P* (⊖ *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*⟩

12.6 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*:
 [| *deg* R p < m; $p \in$ *carrier* P |] ==> *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 ∈ *carrier R* ==> *deg R* (*monom P a n*) <= *n*
⟨*proof*⟩

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

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

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

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

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

lemma (**in** *UP-domain*) *deg-mult* [*simp*]:
[| *p* ~= $\mathbf{0}_P$; *q* ~= $\mathbf{0}_P$; *p* ∈ *carrier P*; *q* ∈ *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* ∈ *A* −> *carrier P* ==>
coeff P (*finsum P p A*) *k* = ($\bigoplus$ *i* ∈ *A*. *coeff P* (*p i*) *k*)
⟨*proof*⟩

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

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

12.7 Polynomials over an integral domain form an integral domain

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*
  $\langle proof \rangle$

lemma (**in** *UP-domain*) *UP-one-not-zero*:
  $\mathbf{1}_P\ \tilde{}= \mathbf{0}_P$
$\langle proof \rangle$

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

theorem (**in** *UP-domain*) *UP-domain*:
  domain P
  $\langle proof \rangle$

Interpretation of theorems from *domain*.

interpretation *UP-domain* < *domain P*
  $\langle proof \rangle$

12.8 Evaluation Homomorphism and Universal Property

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

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

theorem (**in** *cring*) *cauchy-product*:
assumes *bf*: *bound* **0** *n f* **and** *bg*: *bound* **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*) ∈ *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*).
$\bigoplus i \in \{..deg\ R\ p\}.\ phi\ (coeff\ (UP\ R)\ p\ i) \otimes s\ (\hat{}\ )\ i$

lemma (**in** *UP*) *eval-on-carrier*:
includes *struct S*
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 ∈ *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* + *var s* + *var Eval* +
assumes *indet-img-carrier* [*simp*, *intro*]: *s* ∈ *carrier S*
defines *Eval-def*: *Eval* == *eval R S h s*

theorem (**in** *UP-pre-univ-prop*) *eval-ring-hom*:
assumes *S*: *s* ∈ *carrier S*
shows *eval R S h s* ∈ *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.

lemma (**in** *UP-pre-univ-prop*) *eval-const*:
[| *s* ∈ *carrier S*; *r* ∈ *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 $eval\ R\ S\ h\ s\ (monom\ P\ \mathbf{1}\ 1) = s$
⟨*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*⟩

12.9 Sample application of evaluation homomorphism

lemma *UP-pre-univ-propI*:
assumes *cring R*
and *cring S*
and $h \in ring\text{-}hom\ R\ S$
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*]:
$pow\ INTEG\ z\ n = z\ \hat{}\ n$
⟨*proof*⟩

lemma $eval\ INTEG\ INTEG\ id\ 10\ (monom\ (UP\ INTEG)\ 5\ 2) = 500$
$\langle proof \rangle$

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.