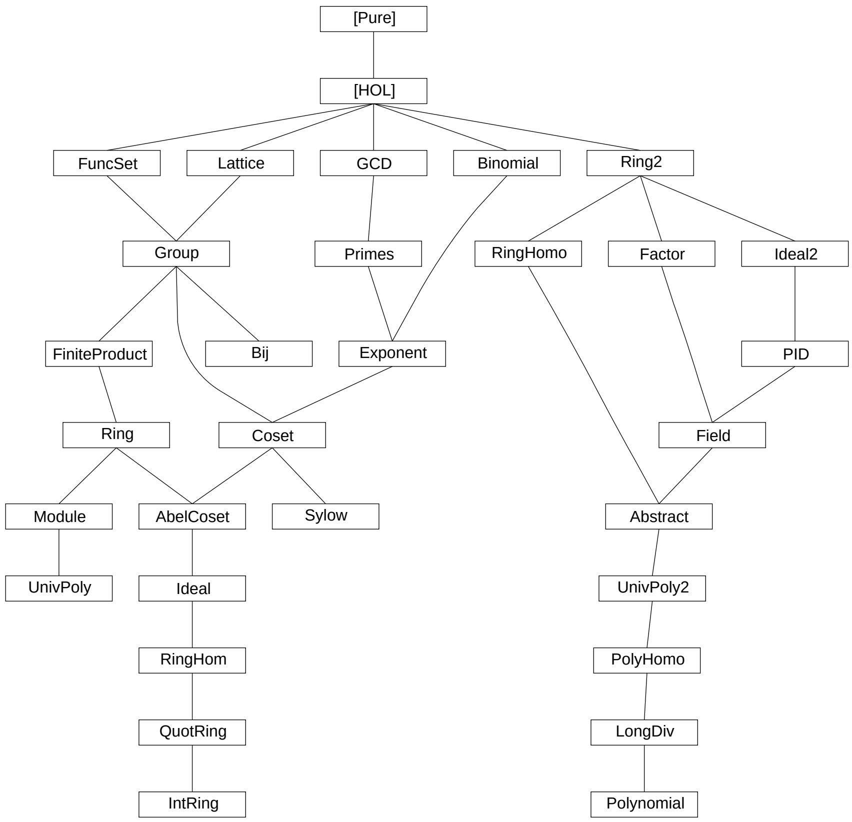
The Isabelle/HOL Algebra Library

Clemens Ballarin
Florian Kammüller
Lawrence C Paulson

November 22, 2007

Contents


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

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

1 Orders and Lattices

Object with a carrier set.

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

1.1 Partial Orders

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

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

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

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

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

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

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

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

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

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

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

1.1.1 Upper

lemma *Upper-closed* [*intro*, *simp*]:
Upper L A ⊆ *carrier L*
by (*unfold Upper-def*) *clarify*

lemma *UpperD* [*dest*]:
fixes *L* (**structure**)
shows [| *u* ∈ *Upper L A*; *x* ∈ *A*; *A* ⊆ *carrier L* |] ==> *x* ⊑ *u*
by (*unfold Upper-def*) *blast*

lemma *Upper-memI*:
fixes *L* (**structure**)
shows [| !! *y*. *y* ∈ *A* ==> *y* ⊑ *x*; *x* ∈ *carrier L* |] ==> *x* ∈ *Upper L A*
by (*unfold Upper-def*) *blast*

lemma *Upper-antimono*:
A ⊆ *B* ==> *Upper L B* ⊆ *Upper L A*
by (*unfold Upper-def*) *blast*

1.1.2 Lower

lemma *Lower-closed* [*intro*, *simp*]:
Lower L A ⊆ *carrier L*
by (*unfold Lower-def*) *clarify*

lemma *LowerD* [*dest*]:
fixes *L* (**structure**)
shows [| *l* ∈ *Lower L A*; *x* ∈ *A*; *A* ⊆ *carrier L* |] ==> *l* ⊑ *x*
by (*unfold Lower-def*) *blast*

lemma *Lower-memI*:
fixes *L* (**structure**)
shows [| !! *y*. *y* ∈ *A* ==> *x* ⊑ *y*; *x* ∈ *carrier L* |] ==> *x* ∈ *Lower L A*
by (*unfold Lower-def*) *blast*

lemma *Lower-antimono*:
A ⊆ *B* ==> *Lower L B* ⊆ *Lower L A*
by (*unfold Lower-def*) *blast*

1.1.3 least

lemma *least-carrier* [*intro*, *simp*]:
  shows *least L l A* ==> $l \in$ *carrier L*
  by (*unfold least-def*) *fast*

lemma *least-mem*:
  least L l A ==> $l \in A$
  by (*unfold least-def*) *fast*

lemma (**in** *partial-order*) *least-unique*:
  [| *least L x A*; *least L y A* |] ==> $x = y$
  by (*unfold least-def*) *blast*

lemma *least-le*:
  fixes *L* (**structure**)
  shows [| *least L x A*; $a \in A$ |] ==> $x \sqsubseteq a$
  by (*unfold least-def*) *fast*

lemma *least-UpperI*:
  fixes *L* (**structure**)
  assumes *above*: !! x. $x \in A$ ==> $x \sqsubseteq s$
    and *below*: !! y. $y \in$ *Upper L A* ==> $s \sqsubseteq y$
    and *L*: $A \subseteq$ *carrier L* $s \in$ *carrier L*
  shows *least L s* (*Upper L A*)
proof –
  have *Upper L A* $\subseteq$ *carrier L* **by** *simp*
  moreover from *above L* **have** $s \in$ *Upper L A* **by** (*simp add*: *Upper-def*)
  moreover from *below* **have** *ALL x* : *Upper L A*. $s \sqsubseteq x$ **by** *fast*
  ultimately show *?thesis* **by** (*simp add*: *least-def*)
qed

1.1.4 greatest

lemma *greatest-carrier* [*intro*, *simp*]:
  shows *greatest L l A* ==> $l \in$ *carrier L*
  by (*unfold greatest-def*) *fast*

lemma *greatest-mem*:
  greatest L l A ==> $l \in A$
  by (*unfold greatest-def*) *fast*

lemma (**in** *partial-order*) *greatest-unique*:
  [| *greatest L x A*; *greatest L y A* |] ==> $x = y$
  by (*unfold greatest-def*) *blast*

lemma *greatest-le*:
  fixes *L* (**structure**)
  shows [| *greatest L x A*; $a \in A$ |] ==> $a \sqsubseteq x$
  by (*unfold greatest-def*) *fast*

lemma *greatest-LowerI*:
  fixes L (**structure**)
  assumes *below*: !! x. $x \in A$ ==> $i \sqsubseteq x$
    and *above*: !! y. $y \in$ *Lower* L A ==> $y \sqsubseteq i$
    and L: $A \subseteq$ *carrier* L $i \in$ *carrier* L
  shows *greatest* L i (*Lower* L A)
proof −
  have *Lower* L $A \subseteq$ *carrier* L **by** *simp*
  moreover from *below* L **have** $i \in$ *Lower* L A **by** (*simp add*: *Lower-def*)
  moreover from *above* **have** *ALL* x : *Lower* L A. $x \sqsubseteq i$ **by** *fast*
  ultimately show *?thesis* **by** (*simp add*: *greatest-def*)
qed

1.2 Lattices

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

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

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

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 (*unfold join-def sup-def*)
  assume L: $x \in$ *carrier* L $y \in$ *carrier* L
    and P: !!l. *least* L l (*Upper* L $\{x, y\}$) ==> P l
  with *sup-of-two-exists* **obtain** s **where** *least* L s (*Upper* L $\{x, y\}$) **by** *fast*
  with L **show** P (*THE* l. *least* L l (*Upper* L $\{x, y\}$))
    by (*fast intro*: *theI2 least-unique* P)
qed

lemma (**in** *lattice*) *join-closed* [*simp*]:
  [| $x \in$ *carrier* L; $y \in$ *carrier* L |] ==> $x \sqcup y \in$ *carrier* L
  by (*rule joinI*) (*rule least-carrier*)

lemma (**in** *partial-order*) *sup-of-singletonI*:

```
  x ∈ carrier L ==> least L x (Upper L {x})
  by (rule least-UpperI) fast+

lemma (in partial-order) sup-of-singleton [simp]:
  x ∈ carrier L ==> ⨆{x} = x
  by (unfold sup-def) (blast intro: least-unique least-UpperI sup-of-singletonI)
```

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 ∈ carrier L; A ⊆ carrier L |]
  ==> P (⨆(insert x A))
proof (unfold sup-def)
  assume L: x ∈ carrier L  A ⊆ carrier L
    and P: !!l. least L l (Upper L (insert x A)) ==> P l
    and least-a: least L a (Upper L A)
  from L least-a have La: a ∈ carrier L by simp
  from L sup-of-two-exists least-a
  obtain s where least-s: least L s (Upper L {a, x}) by blast
  show P (THE l. least L l (Upper L (insert x A)))
  proof (rule theI2)
    show least L s (Upper L (insert x A))
    proof (rule least-UpperI)
      fix z
      assume z ∈ insert x A
      then show z ⊑ s
      proof
        assume z = x then show ?thesis
          by (simp add: least-Upper-above [OF least-s] L La)
      next
        assume z ∈ A
        with L least-s least-a show ?thesis
          by (rule-tac trans [where y = a]) (auto dest: least-Upper-above)
      qed
    next
      fix y
      assume y: y ∈ Upper L (insert x A)
      show s ⊑ y
      proof (rule least-le [OF least-s], rule Upper-memI)
        fix z
        assume z: z ∈ {a, x}
        then show z ⊑ y
        proof
          have y': y ∈ Upper L A
            apply (rule subsetD [where A = Upper L (insert x A)])
             apply (rule Upper-antimono)
             apply blast
            apply (rule y)
            done
```

```
          assume z = a
          with y' least-a show ?thesis by (fast dest: least-le)
        next
          assume z ∈ {x}
          with y L show ?thesis by blast
        qed
      qed (rule Upper-closed [THEN subsetD, OF y])
    next
      from L show insert x A ⊆ carrier L by simp
      from least-s show s ∈ carrier L by simp
    qed
  next
    fix l
    assume least-l: least L l (Upper L (insert x A))
    show l = s
    proof (rule least-unique)
      show least L s (Upper L (insert x A))
      proof (rule least-UpperI)
        fix z
        assume z ∈ insert x A
        then show z ⊑ s
        proof
          assume z = x then show ?thesis
            by (simp add: least-Upper-above [OF least-s] L La)
        next
          assume z ∈ A
          with L least-s least-a show ?thesis
            by (rule-tac trans [where y = a]) (auto dest: least-Upper-above)
        qed
      next
        fix y
        assume y: y ∈ Upper L (insert x A)
        show s ⊑ y
        proof (rule least-le [OF least-s], rule Upper-memI)
          fix z
          assume z: z ∈ {a, x}
          then show z ⊑ y
          proof
            have y': y ∈ Upper L A
              apply (rule subsetD [where A = Upper L (insert x A)])
              apply (rule Upper-antimono)
               apply blast
              apply (rule y)
              done
            assume z = a
            with y' least-a show ?thesis by (fast dest: least-le)
          next
            assume z ∈ {x}
            with y L show ?thesis by blast
```

```
        qed
      qed (rule Upper-closed [THEN subsetD, OF y])
    next
      from L show insert x A ⊆ carrier L by simp
      from least-s show s ∈ carrier L by simp
    qed
  qed (rule least-l)
 qed (rule P)
qed

lemma (in lattice) finite-sup-least:
 [| finite A; A ⊆ carrier L; A ~= {} |] ==> least L (⨆A) (Upper L A)
proof (induct set: finite)
 case empty
 then show ?case by simp
next
 case (insert x A)
 show ?case
 proof (cases A = {})
  case True
  with insert show ?thesis by (simp add: sup-of-singletonI)
 next
  case False
  with insert have least L (⨆A) (Upper L A) by simp
  with - show ?thesis
   by (rule sup-insertI) (simp-all add: insert [simplified])
 qed
qed

lemma (in lattice) finite-sup-insertI:
 assumes P: !!l. least L l (Upper L (insert x A)) ==> P l
  and xA: finite A  x ∈ carrier L  A ⊆ carrier L
 shows P (⨆ (insert x A))
proof (cases A = {})
 case True with P and xA show ?thesis
  by (simp add: sup-of-singletonI)
next
 case False with P and xA show ?thesis
  by (simp add: sup-insertI finite-sup-least)
qed

lemma (in lattice) finite-sup-closed:
 [| finite A; A ⊆ carrier L; A ~= {} |] ==> ⨆A ∈ carrier L
proof (induct set: finite)
 case empty then show ?case by simp
next
 case insert then show ?case
  by - (rule finite-sup-insertI, simp-all)
qed
```

lemma (**in** *lattice*) *join-left*:
  [| $x \in$ *carrier L*; $y \in$ *carrier L* |] ==> $x \sqsubseteq x \sqcup y$
  by (*rule joinI* [*folded join-def*]) (*blast dest*: *least-mem*)

lemma (**in** *lattice*) *join-right*:
  [| $x \in$ *carrier L*; $y \in$ *carrier L* |] ==> $y \sqsubseteq x \sqcup y$
  by (*rule joinI* [*folded join-def*]) (*blast dest*: *least-mem*)

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 (*unfold sup-def*)
  assume *L*: $x \in$ *carrier L* $y \in$ *carrier L*
  with *sup-of-two-exists* **obtain** *s* **where** *least L s* (*Upper L* $\{x, y\}$) **by** *fast*
  with *L* **show** *least L* (*THE xa. least L xa* (*Upper L* $\{x, y\}$)) (*Upper L* $\{x, y\}$)
  by (*fast intro*: *theI2 least-unique*)
qed

lemma (**in** *lattice*) *join-le*:
  assumes *sub*: $x \sqsubseteq z$ $y \sqsubseteq z$
    and *x*: $x \in$ *carrier L* **and** *y*: $y \in$ *carrier L* **and** *z*: $z \in$ *carrier L*
  shows $x \sqcup y \sqsubseteq z$
proof (*rule joinI* [*OF - x y*])
  fix *s*
  assume *least L s* (*Upper L* $\{x, y\}$)
  with *sub z* **show** $s \sqsubseteq z$ **by** (*fast elim*: *least-le intro*: *Upper-memI*)
qed

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 (*rule finite-sup-insertI*)
  — The textbook argument in Jacobson I, p 457
  fix *s*
  assume *sup*: *least L s* (*Upper L* $\{x, y, z\}$)
  show $x \sqcup (y \sqcup z) = s$
  proof (*rule anti-sym*)
    from *sup L* **show** $x \sqcup (y \sqcup z) \sqsubseteq s$
      by (*fastsimp intro*!: *join-le elim*: *least-Upper-above*)
  next
    from *sup L* **show** $s \sqsubseteq x \sqcup (y \sqcup z)$
    by (*erule-tac least-le*)
      (*blast intro*!: *Upper-memI intro*: *trans join-left join-right join-closed*)
  qed (*simp-all add*: *L least-carrier* [*OF sup*])
qed (*simp-all add*: *L*)

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

by (*unfold join-def*) (*simp add*: *insert-commute*)

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 −
have $(x \sqcup y) \sqcup z = z \sqcup (x \sqcup y)$ **by** (*simp only*: *join-comm*)
also from *L* **have** ... $= \bigsqcup \{z, x, y\}$ **by** (*simp add*: *join-assoc-lemma*)
also from *L* **have** ... $= \bigsqcup \{x, y, z\}$ **by** (*simp add*: *insert-commute*)
also from *L* **have** ... $= x \sqcup (y \sqcup z)$ **by** (*simp add*: *join-assoc-lemma*)
finally show *?thesis* **.**
qed

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 (*unfold meet-def inf-def*)
assume *L*: $x \in$ *carrier* *L* $y \in$ *carrier* *L*
and *P*: !!*g*. *greatest* *L* *g* (*Lower* *L* $\{x, y\}$) ==> *P* *g*
with *inf-of-two-exists* **obtain** *i* **where** *greatest* *L* *i* (*Lower* *L* $\{x, y\}$) **by** *fast*
with *L* **show** *P* (*THE* *g*. *greatest* *L* *g* (*Lower* *L* $\{x, y\}$))
by (*fast intro*: *theI2 greatest-unique P*)
qed

lemma (**in** *lattice*) *meet-closed* [*simp*]:
[| $x \in$ *carrier* *L*; $y \in$ *carrier* *L* |] ==> $x \sqcap y \in$ *carrier* *L*
by (*rule meetI*) (*rule greatest-carrier*)

lemma (**in** *partial-order*) *inf-of-singletonI*:
$x \in$ *carrier* *L* ==> *greatest* *L* *x* (*Lower* *L* $\{x\}$)
by (*rule greatest-LowerI*) *fast+*

lemma (**in** *partial-order*) *inf-of-singleton* [*simp*]:
$x \in$ *carrier* *L* ==> $\bigsqcap \{x\} = x$
by (*unfold inf-def*) (*blast intro*: *greatest-unique greatest-LowerI inf-of-singletonI*)

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 (*unfold inf-def*)
assume *L*: $x \in$ *carrier* *L* $A \subseteq$ *carrier* *L*
and *P*: !!*g*. *greatest* *L* *g* (*Lower* *L* (*insert* *x* *A*)) ==> *P* *g*
and *greatest-a*: *greatest* *L* *a* (*Lower* *L* *A*)
from *L* *greatest-a* **have** *La*: $a \in$ *carrier* *L* **by** *simp*

```
from L inf-of-two-exists greatest-a
obtain i where greatest-i: greatest L i (Lower L {a, x}) by blast
show P (THE g. greatest L g (Lower L (insert x A)))
proof (rule theI2)
  show greatest L i (Lower L (insert x A))
  proof (rule greatest-LowerI)
    fix z
    assume z ∈ insert x A
    then show i ⊑ z
    proof
      assume z = x then show ?thesis
        by (simp add: greatest-Lower-above [OF greatest-i] L La)
    next
      assume z ∈ A
      with L greatest-i greatest-a show ?thesis
        by (rule-tac trans [where y = a]) (auto dest: greatest-Lower-above)
    qed
  next
    fix y
    assume y: y ∈ Lower L (insert x A)
    show y ⊑ i
    proof (rule greatest-le [OF greatest-i], rule Lower-memI)
      fix z
      assume z: z ∈ {a, x}
      then show y ⊑ z
      proof
        have y': y ∈ Lower L A
          apply (rule subsetD [where A = Lower L (insert x A)])
          apply (rule Lower-antimono)
           apply blast
          apply (rule y)
          done
        assume z = a
        with y' greatest-a show ?thesis by (fast dest: greatest-le)
      next
        assume z ∈ {x}
        with y L show ?thesis by blast
      qed
    qed (rule Lower-closed [THEN subsetD, OF y])
  next
    from L show insert x A ⊆ carrier L by simp
    from greatest-i show i ∈ carrier L by simp
  qed
next
  fix g
  assume greatest-g: greatest L g (Lower L (insert x A))
  show g = i
  proof (rule greatest-unique)
    show greatest L i (Lower L (insert x A))
```

```
      proof (rule greatest-LowerI)
        fix z
        assume z ∈ insert x A
        then show i ⊑ z
        proof
          assume z = x then show ?thesis
            by (simp add: greatest-Lower-above [OF greatest-i] L La)
        next
          assume z ∈ A
          with L greatest-i greatest-a show ?thesis
            by (rule-tac trans [where y = a]) (auto dest: greatest-Lower-above)
        qed
      next
        fix y
        assume y: y ∈ Lower L (insert x A)
        show y ⊑ i
        proof (rule greatest-le [OF greatest-i], rule Lower-memI)
          fix z
          assume z: z ∈ {a, x}
          then show y ⊑ z
          proof
            have y': y ∈ Lower L A
              apply (rule subsetD [where A = Lower L (insert x A)])
              apply (rule Lower-antimono)
               apply blast
              apply (rule y)
              done
            assume z = a
            with y' greatest-a show ?thesis by (fast dest: greatest-le)
          next
            assume z ∈ {x}
            with y L show ?thesis by blast
          qed
        qed (rule Lower-closed [THEN subsetD, OF y])
      next
        from L show insert x A ⊆ carrier L by simp
        from greatest-i show i ∈ carrier L by simp
      qed
    qed (rule greatest-g)
  qed (rule P)
qed

lemma (in lattice) finite-inf-greatest:
  [| finite A; A ⊆ carrier L; A ~= {} |] ==> greatest L (⨅ A) (Lower L A)
proof (induct set: finite)
  case empty then show ?case by simp
next
  case (insert x A)
  show ?case
```

proof (*cases* $A = \{\}$)
case *True*
with *insert* **show** *?thesis* **by** (*simp add*: *inf-of-singletonI*)
next
case *False*
from *insert* **show** *?thesis*
proof (*rule-tac inf-insertI*)
from *False insert* **show** *greatest L* ($\bigsqcap A$) (*Lower L A*) **by** *simp*
qed *simp-all*
qed
qed

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 (*cases* $A = \{\}$)
case *True* **with** *P* **and** *xA* **show** *?thesis*
by (*simp add*: *inf-of-singletonI*)
next
case *False* **with** *P* **and** *xA* **show** *?thesis*
by (*simp add*: *inf-insertI finite-inf-greatest*)
qed

lemma (**in** *lattice*) *finite-inf-closed*:
[| *finite A*; $A \subseteq$ *carrier L*; *A* ~= {} |] ==> $\bigsqcap A \in$ *carrier L*
proof (*induct set*: *finite*)
case *empty* **then show** *?case* **by** *simp*
next
case *insert* **then show** *?case*
by (*rule-tac finite-inf-insertI*) (*simp-all*)
qed

lemma (**in** *lattice*) *meet-left*:
[| $x \in$ *carrier L*; $y \in$ *carrier L* |] ==> $x \sqcap y \sqsubseteq x$
by (*rule meetI* [*folded meet-def*]) (*blast dest*: *greatest-mem*)

lemma (**in** *lattice*) *meet-right*:
[| $x \in$ *carrier L*; $y \in$ *carrier L* |] ==> $x \sqcap y \sqsubseteq y$
by (*rule meetI* [*folded meet-def*]) (*blast dest*: *greatest-mem*)

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 (*unfold inf-def*)
assume *L*: $x \in$ *carrier L* $y \in$ *carrier L*
with *inf-of-two-exists* **obtain** *s* **where** *greatest L s* (*Lower L* $\{x, y\}$) **by** *fast*
with *L*
show *greatest L* (*THE xa*. *greatest L xa* (*Lower L* $\{x, y\}$)) (*Lower L* $\{x, y\}$)

by (*fast intro*: *theI2 greatest-unique*)
qed

lemma (**in** *lattice*) *meet-le*:
assumes *sub*: $z \sqsubseteq x \quad z \sqsubseteq y$
and *x*: $x \in$ *carrier* L **and** *y*: $y \in$ *carrier* L **and** *z*: $z \in$ *carrier* L
shows $z \sqsubseteq x \sqcap y$
proof (*rule meetI* [*OF* - *x y*])
fix *i*
assume *greatest* L i (*Lower* L $\{x, y\}$)
with *sub z* **show** $z \sqsubseteq i$ **by** (*fast elim*: *greatest-le intro*: *Lower-memI*)
qed

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 (*rule finite-inf-insertI*)

The textbook argument in Jacobson I, p 457

fix *i*
assume *inf*: *greatest* L i (*Lower* L $\{x, y, z\}$)
show $x \sqcap (y \sqcap z) = i$
proof (*rule anti-sym*)
from *inf L* **show** $i \sqsubseteq x \sqcap (y \sqcap z)$
by (*fastsimp intro*!: *meet-le elim*: *greatest-Lower-above*)
next
from *inf L* **show** $x \sqcap (y \sqcap z) \sqsubseteq i$
by (*erule-tac greatest-le*)
(*blast intro*!: *Lower-memI intro*: *trans meet-left meet-right meet-closed*)
qed (*simp-all add*: *L greatest-carrier* [*OF inf*])
qed (*simp-all add*: *L*)

lemma *meet-comm*:
fixes *L* (**structure**)
shows $x \sqcap y = y \sqcap x$
by (*unfold meet-def*) (*simp add*: *insert-commute*)

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 −
have $(x \sqcap y) \sqcap z = z \sqcap (x \sqcap y)$ **by** (*simp only*: *meet-comm*)
also from *L* **have** ... $= \bigsqcap \{z, x, y\}$ **by** (*simp add*: *meet-assoc-lemma*)
also from *L* **have** ... $= \bigsqcap \{x, y, z\}$ **by** (*simp add*: *insert-commute*)
also from *L* **have** ... $= x \sqcap (y \sqcap z)$ **by** (*simp add*: *meet-assoc-lemma*)
finally show *?thesis* **.**
qed

1.3 Total Orders

locale *total-order* = *partial-order* +
  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*
  by *unfold-locales* (*rule total*)

Total orders are lattices.

interpretation *total-order* < *lattice*
proof *unfold-locales*
  fix *x y*
  assume *L*: $x \in$ *carrier L* $y \in$ *carrier L*
  show *EX s. least L s* (*Upper L* {*x*, *y*})
  proof −
    note *total L*
    moreover
    {
      assume $x \sqsubseteq y$
      with *L* **have** *least L y* (*Upper L* {*x*, *y*})
        by (*rule-tac least-UpperI*) *auto*
    }
    moreover
    {
      assume $y \sqsubseteq x$
      with *L* **have** *least L x* (*Upper L* {*x*, *y*})
        by (*rule-tac least-UpperI*) *auto*
    }
    ultimately show *?thesis* **by** *blast*
  qed
next
  fix *x y*
  assume *L*: $x \in$ *carrier L* $y \in$ *carrier L*
  show *EX i. greatest L i* (*Lower L* {*x*, *y*})
  proof −
    note *total L*
    moreover
    {
      assume $y \sqsubseteq x$
      with *L* **have** *greatest L y* (*Lower L* {*x*, *y*})
        by (*rule-tac greatest-LowerI*) *auto*
    }
    moreover
    {
      assume $x \sqsubseteq y$
      with *L* **have** *greatest L x* (*Lower L* {*x*, *y*})
        by (*rule-tac greatest-LowerI*) *auto*

    }
    ultimately show *?thesis* **by** *blast*
  qed
qed

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 *intro-locales*
  show *lattice-axioms L*
    by (*rule lattice-axioms.intro*) (*blast intro*: *sup-exists inf-exists*)+
qed (*rule complete-lattice-axioms.intro sup-exists inf-exists* | *assumption*)+

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

  bottom :: - => *'a* ($\bot_1$)
  $\bot$ == *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 (*unfold sup-def*)
  assume *L*: $A \subseteq$ *carrier L*
    and *P*: !!*l*. *least L l* (*Upper L A*) ==> *P l*
  with *sup-exists* **obtain** *s* **where** *least L s* (*Upper L A*) **by** *blast*
  with *L* **show** *P* (*THE l. least L l* (*Upper L A*))
  by (*fast intro*: *theI2 least-unique P*)
qed

lemma (**in** *complete-lattice*) *sup-closed* [*simp*]:
  $A \subseteq$ *carrier L* ==> $\bigsqcup$*A* $\in$ *carrier L*
  by (*rule supI*) *simp-all*

lemma (**in** *complete-lattice*) *top-closed* [*simp*, *intro*]:

$\top \in$ *carrier L*
by (*unfold top-def*) *simp*

lemma (**in** *complete-lattice*) *infI*:
[| !!*i*. *greatest L i* (*Lower L A*) ==> *P i*; $A \subseteq$ *carrier L* |]
==> *P* ($\bigsqcap$*A*)
proof (*unfold inf-def*)
assume *L*: $A \subseteq$ *carrier L*
and *P*: !!*l*. *greatest L l* (*Lower L A*) ==> *P l*
with *inf-exists* **obtain** *s* **where** *greatest L s* (*Lower L A*) **by** *blast*
with *L* **show** *P* (*THE l*. *greatest L l* (*Lower L A*))
by (*fast intro*: *theI2 greatest-unique P*)
qed

lemma (**in** *complete-lattice*) *inf-closed* [*simp*]:
$A \subseteq$ *carrier L* ==> $\bigsqcap$*A* $\in$ *carrier L*
by (*rule infI*) *simp-all*

lemma (**in** *complete-lattice*) *bottom-closed* [*simp*, *intro*]:
$\bot \in$ *carrier L*
by (*unfold bottom-def*) *simp*

Jacobson: Theorem 8.1

lemma *Lower-empty* [*simp*]:
Lower L {} = *carrier L*
by (*unfold Lower-def*) *simp*

lemma *Upper-empty* [*simp*]:
Upper L {} = *carrier L*
by (*unfold Upper-def*) *simp*

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 (*rule complete-latticeI*)
from *top-exists* **obtain** *top* **where** *top*: *greatest L top* (*carrier L*) **..**
fix *A*
assume *L*: $A \subseteq$ *carrier L*
let *?B* = *Upper L A*
from *L top* **have** *top* $\in$ *?B* **by** (*fast intro*!: *Upper-memI intro*: *greatest-le*)
then have *B-non-empty*: *?B* ~= {} **by** *fast*
have *B-L*: *?B* $\subseteq$ *carrier L* **by** *simp*
from *inf-exists* [*OF B-L B-non-empty*]
obtain *b* **where** *b-inf-B*: *greatest L b* (*Lower L ?B*) **..**
have *least L b* (*Upper L A*)
apply (*rule least-UpperI*)
apply (*rule greatest-le* [**where** *A* = *Lower L ?B*])

apply (*rule b-inf-B*)
apply (*rule Lower-memI*)
apply (*erule UpperD*)
apply *assumption*
apply (*rule L*)
apply (*fast intro*: *L* [*THEN subsetD*])
apply (*erule greatest-Lower-above* [*OF b-inf-B*])
apply *simp*
apply (*rule L*)
apply (*rule greatest-carrier* [*OF b-inf-B*])
done
then show *EX s. least L s* (*Upper L A*) **..**
next
fix *A*
assume *L*: $A \subseteq$ *carrier L*
show *EX i. greatest L i* (*Lower L A*)
proof (*cases* $A = \{\}$)
case *True* **then show** *?thesis*
by (*simp add*: *top-exists*)
next
case *False* **with** *L* **show** *?thesis*
by (*rule inf-exists*)
qed
qed

1.5 Examples

1.5.1 Powerset of a Set is a Complete Lattice

theorem *powerset-is-complete-lattice*:
complete-lattice (| *carrier* = *Pow A*, *le* = *op* $\subseteq$ |)
(**is** *complete-lattice ?L*)
proof (*rule partial-order.complete-latticeI*)
show *partial-order ?L*
by (*rule partial-order.intro*) *auto*
next
fix *B*
assume $B \subseteq$ *carrier ?L*
then have *least ?L* ($\bigcup B$) (*Upper ?L B*)
by (*fastsimp intro*!: *least-UpperI simp*: *Upper-def*)
then show *EX s. least ?L s* (*Upper ?L B*) **..**
next
fix *B*
assume $B \subseteq$ *carrier ?L*
then have *greatest ?L* ($\bigcap B \cap A$) (*Lower ?L B*)

$\bigcap B$ is not the infimum of B: $\bigcap \{\} = UNIV$ which is in general bigger than A!

by (*fastsimp intro*!: *greatest-LowerI simp*: *Lower-def*)
then show *EX i. greatest ?L i* (*Lower ?L B*) **..**
qed

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

end

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

2 Monoids and Groups

2.1 Definitions

Definitions follow [2].

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

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

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

consts
  pow :: [(*'a*, *'m*) *monoid-scheme*, *'a*, *'b*::*number*] => *'a* (**infixr** '(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* =
  fixes *G* (**structure**)
  assumes *m-closed* [*intro*, *simp*]:
      $[\![x \in$ *carrier G*; $y \in$ *carrier G*$]\!] \Longrightarrow x \otimes y \in$ *carrier G*
    and *m-assoc*:
      $[\![x \in$ *carrier G*; $y \in$ *carrier G*; $z \in$ *carrier G*$]\!]$
      $\Longrightarrow (x \otimes y) \otimes z = x \otimes (y \otimes z)$
    and *one-closed* [*intro*, *simp*]: $\mathbf{1} \in$ *carrier G*
    and *l-one* [*simp*]: $x \in$ *carrier G* $\Longrightarrow \mathbf{1} \otimes x = x$
    and *r-one* [*simp*]: $x \in$ *carrier G* $\Longrightarrow x \otimes \mathbf{1} = x$

lemma *monoidI*:
  fixes *G* (**structure**)
  assumes *m-closed*:

$!!x\ y.\ [|\ x \in carrier\ G;\ y \in carrier\ G\ |] ==> x \otimes y \in carrier\ G$
and *one-closed*: $\mathbf{1} \in carrier\ G$
and *m-assoc*:
$!!x\ y\ z.\ [|\ x \in carrier\ G;\ y \in carrier\ G;\ z \in carrier\ G\ |] ==>$
$(x \otimes y) \otimes z = x \otimes (y \otimes z)$
and *l-one*: $!!x.\ x \in carrier\ G ==> \mathbf{1} \otimes x = x$
and *r-one*: $!!x.\ x \in carrier\ G ==> x \otimes \mathbf{1} = x$
shows *monoid G*
by (*fast intro*!: *monoid.intro intro*: *prems*)

lemma (**in** *monoid*) *Units-closed* [*dest*]:
$x \in Units\ G ==> x \in carrier\ G$
by (*unfold Units-def*) *fast*

lemma (**in** *monoid*) *inv-unique*:
assumes *eq*: $y \otimes x = \mathbf{1}\ \ x \otimes y' = \mathbf{1}$
and *G*: $x \in carrier\ G\ \ y \in carrier\ G\ \ y' \in carrier\ G$
shows $y = y'$
proof −
from *G eq* **have** $y = y \otimes (x \otimes y')$ **by** *simp*
also from *G* **have** ... $= (y \otimes x) \otimes y'$ **by** (*simp add*: *m-assoc*)
also from *G eq* **have** ... $= y'$ **by** *simp*
finally show *?thesis* **.**
qed

lemma (**in** *monoid*) *Units-one-closed* [*intro*, *simp*]:
$\mathbf{1} \in Units\ G$
by (*unfold Units-def*) *auto*

lemma (**in** *monoid*) *Units-inv-closed* [*intro*, *simp*]:
$x \in Units\ G ==> inv\ x \in carrier\ G$
apply (*unfold Units-def m-inv-def*, *auto*)
apply (*rule theI2*, *fast*)
apply (*fast intro*: *inv-unique*, *fast*)
done

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

lemma (**in** *monoid*) *Units-r-inv-ex*:
$x \in Units\ G ==> \exists\, y \in carrier\ G.\ x \otimes y = \mathbf{1}$
by (*unfold Units-def*) *auto*

lemma (**in** *monoid*) *Units-l-inv*:
$x \in Units\ G ==> inv\ x \otimes x = \mathbf{1}$
apply (*unfold Units-def m-inv-def*, *auto*)
apply (*rule theI2*, *fast*)
apply (*fast intro*: *inv-unique*, *fast*)

done

lemma (**in** *monoid*) *Units-r-inv*:
$x \in Units\ G ==> x \otimes inv\ x = \mathbf{1}$
apply (*unfold Units-def m-inv-def*, *auto*)
apply (*rule theI2*, *fast*)
apply (*fast intro*: *inv-unique*, *fast*)
done

lemma (**in** *monoid*) *Units-inv-Units* [*intro*, *simp*]:
$x \in Units\ G ==> inv\ x \in Units\ G$
proof −
assume *x*: $x \in Units\ G$
show $inv\ x \in Units\ G$
by (*auto simp add*: *Units-def*
intro: *Units-l-inv Units-r-inv x Units-closed* [*OF x*])
qed

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
assume *eq*: $x \otimes y = x \otimes z$
and *G*: $x \in Units\ G \quad y \in carrier\ G \quad z \in carrier\ G$
then have $(inv\ x \otimes x) \otimes y = (inv\ x \otimes x) \otimes z$
by (*simp add*: *m-assoc Units-closed*)
with *G* **show** $y = z$ **by** (*simp add*: *Units-l-inv*)
next
assume *eq*: $y = z$
and *G*: $x \in Units\ G \quad y \in carrier\ G \quad z \in carrier\ G$
then show $x \otimes y = x \otimes z$ **by** *simp*
qed

lemma (**in** *monoid*) *Units-inv-inv* [*simp*]:
$x \in Units\ G ==> inv\ (inv\ x) = x$
proof −
assume *x*: $x \in Units\ G$
then have $inv\ x \otimes inv\ (inv\ x) = inv\ x \otimes x$
by (*simp add*: *Units-l-inv Units-r-inv*)
with *x* **show** *?thesis* **by** (*simp add*: *Units-closed*)
qed

lemma (**in** *monoid*) *inv-inj-on-Units*:
inj-on (*m-inv G*) (*Units G*)
proof (*rule inj-onI*)
fix *x y*
assume *G*: $x \in Units\ G \quad y \in Units\ G$ **and** *eq*: $inv\ x = inv\ y$
then have $inv\ (inv\ x) = inv\ (inv\ y)$ **by** *simp*
with *G* **show** $x = y$ **by** *simp*

qed

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 −
  from *G* **have** $x \otimes y \otimes x = x \otimes \mathbf{1}$ **by** (*auto simp add*: *inv Units-closed*)
  with *G* **show** *?thesis* **by** (*simp del*: *r-one add*: *m-assoc Units-closed*)
qed

Power

lemma (**in** *monoid*) *nat-pow-closed* [*intro*, *simp*]:
  $x \in$ *carrier G* ==> x (ˆ) (n::*nat*) $\in$ *carrier G*
  by (*induct n*) (*simp-all add*: *nat-pow-def*)

lemma (**in** *monoid*) *nat-pow-0* [*simp*]:
  x (ˆ) (*0*::*nat*) $= \mathbf{1}$
  by (*simp add*: *nat-pow-def*)

lemma (**in** *monoid*) *nat-pow-Suc* [*simp*]:
  x (ˆ) (*Suc n*) $=$ x (ˆ) $n \otimes x$
  by (*simp add*: *nat-pow-def*)

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

lemma (**in** *monoid*) *nat-pow-mult*:
  $x \in$ *carrier G* ==> x (ˆ) (n::*nat*) $\otimes$ x (ˆ) m $=$ x (ˆ) ($n + m$)
  by (*induct m*) (*simp-all add*: *m-assoc* [*THEN sym*])

lemma (**in** *monoid*) *nat-pow-pow*:
  $x \in$ *carrier G* ==> (x (ˆ) n) (ˆ) m $=$ x (ˆ) ($n * m$::*nat*)
  by (*induct m*) (*simp*, *simp add*: *nat-pow-mult add-commute*)

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* **by** *fact*

theorem *groupI*:
  fixes *G* (**structure**)
  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 ∈ carrier G; y ∈ carrier G; z ∈ carrier G |] ==>
    (x ⊗ y) ⊗ z = x ⊗ (y ⊗ z)
  and l-one [simp]: !!x. x ∈ carrier G ==> 1 ⊗ x = x
  and l-inv-ex: !!x. x ∈ carrier G ==> ∃ y ∈ carrier G. y ⊗ x = 1
 shows group G
proof −
 have l-cancel [simp]:
  !!x y z. [| x ∈ carrier G; y ∈ carrier G; z ∈ carrier G |] ==>
  (x ⊗ y = x ⊗ z) = (y = z)
 proof
  fix x y z
  assume eq: x ⊗ y = x ⊗ z
   and G: x ∈ carrier G  y ∈ carrier G  z ∈ carrier G
  with l-inv-ex obtain x-inv where xG: x-inv ∈ carrier G
   and l-inv: x-inv ⊗ x = 1 by fast
  from G eq xG have (x-inv ⊗ x) ⊗ y = (x-inv ⊗ x) ⊗ z
   by (simp add: m-assoc)
  with G show y = z by (simp add: l-inv)
 next
  fix x y z
  assume eq: y = z
   and G: x ∈ carrier G  y ∈ carrier G  z ∈ carrier G
  then show x ⊗ y = x ⊗ z by simp
 qed
 have r-one:
  !!x. x ∈ carrier G ==> x ⊗ 1 = x
 proof −
  fix x
  assume x: x ∈ carrier G
  with l-inv-ex obtain x-inv where xG: x-inv ∈ carrier G
   and l-inv: x-inv ⊗ x = 1 by fast
  from x xG have x-inv ⊗ (x ⊗ 1) = x-inv ⊗ x
   by (simp add: m-assoc [symmetric] l-inv)
  with x xG show x ⊗ 1 = x by simp
 qed
 have inv-ex:
  !!x. x ∈ carrier G ==> ∃ y ∈ carrier G. y ⊗ x = 1 & x ⊗ y = 1
 proof −
  fix x
  assume x: x ∈ carrier G
  with l-inv-ex obtain y where y: y ∈ carrier G
   and l-inv: y ⊗ x = 1 by fast
  from x y have y ⊗ (x ⊗ y) = y ⊗ 1
   by (simp add: m-assoc [symmetric] l-inv r-one)
  with x y have r-inv: x ⊗ y = 1
   by simp
  from x y show ∃ y ∈ carrier G. y ⊗ x = 1 & x ⊗ y = 1
   by (fast intro: l-inv r-inv)
 qed
```

then have *carrier-subset-Units*: *carrier G* <= *Units G*
  by (*unfold Units-def*) *fast*
show *?thesis*
  by (*fast intro*!: *group.intro monoid.intro group-axioms.intro*
    carrier-subset-Units intro: *prems r-one*)
qed

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*
  by (*rule groupI*) (*auto intro*: *m-assoc l-inv-ex*)

lemma (**in** *group*) *Units-eq* [*simp*]:
  Units G = *carrier G*
proof
  show *Units G* <= *carrier G* **by** *fast*
next
  show *carrier G* <= *Units G* **by** (*rule Units*)
qed

lemma (**in** *group*) *inv-closed* [*intro*, *simp*]:
  $x \in$ *carrier G* ==> *inv x* $\in$ *carrier G*
  using *Units-inv-closed* **by** *simp*

lemma (**in** *group*) *l-inv-ex* [*simp*]:
  $x \in$ *carrier G* ==> $\exists\, y \in$ *carrier G*. $y \otimes x = \mathbf{1}$
  using *Units-l-inv-ex* **by** *simp*

lemma (**in** *group*) *r-inv-ex* [*simp*]:
  $x \in$ *carrier G* ==> $\exists\, y \in$ *carrier G*. $x \otimes y = \mathbf{1}$
  using *Units-r-inv-ex* **by** *simp*

lemma (**in** *group*) *l-inv* [*simp*]:
  $x \in$ *carrier G* ==> *inv x* $\otimes\, x = \mathbf{1}$
  using *Units-l-inv* **by** *simp*

2.2 Cancellation Laws and Basic Properties

lemma (**in** *group*) *l-cancel* [*simp*]:
  [| $x \in$ *carrier G*; $y \in$ *carrier G*; $z \in$ *carrier G* |] ==>
    $(x \otimes y = x \otimes z) = (y = z)$
  using *Units-l-inv* **by** *simp*

lemma (**in** *group*) *r-inv* [*simp*]:
  $x \in$ *carrier G* ==> $x \otimes$ *inv x* $= \mathbf{1}$
proof −
  assume *x*: $x \in$ *carrier G*
  then have *inv x* $\otimes$ ($x \otimes$ *inv x*) = *inv x* $\otimes\, \mathbf{1}$

```
    by (simp add: m-assoc [symmetric] l-inv)
  with x show ?thesis by (simp del: r-one)
qed

lemma (in group) r-cancel [simp]:
  [| x ∈ carrier G; y ∈ carrier G; z ∈ carrier G |] ==>
   (y ⊗ x = z ⊗ x) = (y = z)
proof
  assume eq: y ⊗ x = z ⊗ x
    and G: x ∈ carrier G  y ∈ carrier G  z ∈ carrier G
  then have y ⊗ (x ⊗ inv x) = z ⊗ (x ⊗ inv x)
    by (simp add: m-assoc [symmetric] del: r-inv)
  with G show y = z by simp
next
  assume eq: y = z
    and G: x ∈ carrier G  y ∈ carrier G  z ∈ carrier G
  then show y ⊗ x = z ⊗ x by simp
qed

lemma (in group) inv-one [simp]:
  inv 1 = 1
proof -
  have inv 1 = 1 ⊗ (inv 1) by (simp del: r-inv)
  moreover have ... = 1 by simp
  finally show ?thesis .
qed

lemma (in group) inv-inv [simp]:
  x ∈ carrier G ==> inv (inv x) = x
  using Units-inv-inv by simp

lemma (in group) inv-inj:
  inj-on (m-inv G) (carrier G)
  using inv-inj-on-Units by simp

lemma (in group) inv-mult-group:
  [| x ∈ carrier G; y ∈ carrier G |] ==> inv (x ⊗ y) = inv y ⊗ inv x
proof -
  assume G: x ∈ carrier G  y ∈ carrier G
  then have inv (x ⊗ y) ⊗ (x ⊗ y) = (inv y ⊗ inv x) ⊗ (x ⊗ y)
    by (simp add: m-assoc l-inv) (simp add: m-assoc [symmetric])
  with G show ?thesis by (simp del: l-inv)
qed

lemma (in group) inv-comm:
  [| x ⊗ y = 1; x ∈ carrier G; y ∈ carrier G |] ==> y ⊗ x = 1
  by (rule Units-inv-comm) auto

lemma (in group) inv-equality:
```

```
  [|y ⊗ x = 1; x ∈ carrier G; y ∈ carrier G|] ==> inv x = y
apply (simp add: m-inv-def)
apply (rule the-equality)
 apply (simp add: inv-comm [of y x])
apply (rule r-cancel [THEN iffD1], auto)
done
```

Power

```
lemma (in group) int-pow-def2:
  a (^) (z::int) = (if neg z then inv (a (^) (nat (−z))) else a (^) (nat z))
  by (simp add: int-pow-def nat-pow-def Let-def)

lemma (in group) int-pow-0 [simp]:
  x (^) (0::int) = 1
  by (simp add: int-pow-def2)

lemma (in group) int-pow-one [simp]:
  1 (^) (z::int) = 1
  by (simp add: int-pow-def2)
```

2.3 Subgroups

```
locale subgroup =
  fixes H and G (structure)
  assumes subset: H ⊆ carrier G
    and m-closed [intro, simp]: [[x ∈ H; y ∈ H]] ⟹ x ⊗ y ∈ H
    and one-closed [simp]: 1 ∈ H
    and m-inv-closed [intro,simp]: x ∈ H ⟹ inv x ∈ H

lemma (in subgroup) is-subgroup:
  subgroup H G by fact

declare (in subgroup) group.intro [intro]

lemma (in subgroup) mem-carrier [simp]:
  x ∈ H ⟹ x ∈ carrier G
  using subset by blast

lemma subgroup-imp-subset:
  subgroup H G ⟹ H ⊆ carrier G
  by (rule subgroup.subset)

lemma (in subgroup) subgroup-is-group [intro]:
  includes group G
  shows group (G(|carrier := H|))
  by (rule groupI) (auto intro: m-assoc l-inv mem-carrier)
```

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*:
$[| \ H \subseteq carrier\ G;\ H \neq \{\};\ \forall\, a \in H.\ inv\ a \in H;\ \forall\, a{\in}H.\ \forall\, b{\in}H.\ a \otimes b \in H\ |]$
$==> \mathbf{1} \in H$
by (*force simp add*: *l-inv*)

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 (*simp add*: *subgroup-def prems*)
show $\mathbf{1} \in H$ **by** (*rule one-in-subset*) (*auto simp only*: *prems*)
qed

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*
by (*blast dest*: *subgroup.one-closed*)

lemma (**in** *subgroup*) *finite-imp-card-positive*:
finite (*carrier G*) $==> 0 < card\ H$
proof (*rule classical*)
assume *finite* (*carrier G*) $\sim\ 0 < card\ H$
then have *finite H* **by** (*blast intro*: *finite-subset* [*OF subset*])
with *prems* **have** *subgroup* $\{\}$ *G* **by** *simp*
with *subgroup-nonempty* **show** *?thesis* **by** *contradiction*
qed

2.4 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 −
from *prems*
show *?thesis* **by** (*unfold monoid-def DirProd-def*, *auto*)
qed

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* ×× *H*)
by (*rule groupI*)
(*auto intro*: *G.m-assoc H.m-assoc G.l-inv H.l-inv*
simp add: *DirProd-def*)

lemma *carrier-DirProd* [*simp*]:
carrier (*G* ×× *H*) = *carrier G* × *carrier H*
by (*simp add*: *DirProd-def*)

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

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

lemma *inv-DirProd* [*simp*]:
includes *group G* + *group H*
assumes *g*: *g* ∈ *carrier G*
and *h*: *h* ∈ *carrier H*
shows *m-inv* (*G* ×× *H*) (*g*, *h*) = (inv_G *g*, inv_H *h*)
apply (*rule group.inv-equality* [*OF DirProd-group*])
apply (*simp-all add*: *prems group.l-inv*)
done

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* ∈ *carrier G*
and *h*: *h* ∈ *carrier H*
shows *m-inv* (*G* ×× *H*) (*g*, *h*) = (inv_G *g*, inv_H *h*)
proof −
interpret *Prod*: *group* [*G* ×× *H*]
by (*auto intro*: *DirProd-group group.intro group.axioms prems*)
show *?thesis* **by** (*simp add*: *Prod.inv-equality g h*)
qed

2.5 Homomorphisms and Isomorphisms

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

lemma *hom-mult*:
  [| $h \in hom\ G\ H$; $x \in carrier\ G$; $y \in carrier\ G$ |]
  ==> $h\ (x \otimes_G y) = h\ x \otimes_H h\ y$
  by (*simp add*: *hom-def*)

lemma *hom-closed*:
  [| $h \in hom\ G\ H$; $x \in carrier\ G$ |] ==> $h\ x \in carrier\ H$
  by (*auto simp add*: *hom-def funcset-mem*)

lemma (**in** *group*) *hom-compose*:
    [|$h \in hom\ G\ H$; $i \in hom\ H\ I$|] ==> *compose* (*carrier G*) $i\ h \in hom\ G\ I$
apply (*auto simp add*: *hom-def funcset-compose*)
apply (*simp add*: *compose-def funcset-mem*)
done

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

lemma *iso-refl*: (%*x*. *x*) $\in G \cong G$
by (*simp add*: *iso-def hom-def inj-on-def bij-betw-def Pi-def*)

lemma (**in** *group*) *iso-sym*:
    $h \in G \cong H \Longrightarrow$ *Inv* (*carrier G*) $h \in H \cong G$
apply (*simp add*: *iso-def bij-betw-Inv*)
apply (*subgoal-tac Inv* (*carrier G*) $h \in carrier\ H \to carrier\ G$)
 prefer *2* **apply** (*simp add*: *bij-betw-imp-funcset* [*OF bij-betw-Inv*])
apply (*simp add*: *hom-def bij-betw-def Inv-f-eq funcset-mem f-Inv-f*)
done

lemma (**in** *group*) *iso-trans*:
    [|$h \in G \cong H$; $i \in H \cong I$|] ==> (*compose* (*carrier G*) *i h*) $\in G \cong I$
by (*auto simp add*: *iso-def hom-compose bij-betw-compose*)

lemma *DirProd-commute-iso*:
  shows $(\lambda(x,y).\ (y,x)) \in (G \times\times H) \cong (H \times\times G)$
by (*auto simp add*: *iso-def hom-def inj-on-def bij-betw-def Pi-def*)

lemma *DirProd-assoc-iso*:
  shows $(\lambda(x,y,z).\ (x,(y,z))) \in (G \times\times H \times\times I) \cong (G \times\times (H \times\times I))$
by (*auto simp add*: *iso-def hom-def inj-on-def bij-betw-def Pi-def*)

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*]:
$h\ \mathbf{1} \in carrier\ H$
by *simp*

lemma (**in** *group-hom*) *hom-one* [*simp*]:
$h\ \mathbf{1} = \mathbf{1}_H$
proof −
have $h\ \mathbf{1} \otimes_H \mathbf{1}_H = h\ \mathbf{1} \otimes_H h\ \mathbf{1}$
by (*simp add*: *hom-mult* [*symmetric*] *del*: *hom-mult*)
then show *?thesis* **by** (*simp del*: *r-one*)
qed

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

lemma (**in** *group-hom*) *hom-inv* [*simp*]:
$x \in carrier\ G$ ==> $h\ (inv\ x) = inv_H\ (h\ x)$
proof −
assume x: $x \in carrier\ G$
then have $h\ x \otimes_H h\ (inv\ x) = \mathbf{1}_H$
by (*simp add*: *hom-mult* [*symmetric*] *del*: *hom-mult*)
also from x **have** ... $= h\ x \otimes_H inv_H\ (h\ x)$
by (*simp add*: *hom-mult* [*symmetric*] *del*: *hom-mult*)
finally have $h\ x \otimes_H h\ (inv\ x) = h\ x \otimes_H inv_H\ (h\ x)$ **.**
with x **show** *?thesis* **by** (*simp del*: *H.r-inv*)
qed

2.6 Commutative Structures

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

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

lemma (**in** *comm-monoid*) *m-lcomm*:
$[\![x \in carrier\ G;\ y \in carrier\ G;\ z \in carrier\ G]\!] \Longrightarrow$
$x \otimes (y \otimes z) = y \otimes (x \otimes z)$
proof −
assume *xyz*: $x \in carrier\ G\ \ y \in carrier\ G\ \ z \in carrier\ G$
from *xyz* **have** $x \otimes (y \otimes z) = (x \otimes y) \otimes z$ **by** (*simp add*: *m-assoc*)
also from *xyz* **have** ... $= (y \otimes x) \otimes z$ **by** (*simp add*: *m-comm*)
also from *xyz* **have** ... $= y \otimes (x \otimes z)$ **by** (*simp add*: *m-assoc*)
finally show *?thesis* **.**
qed

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

lemma *comm-monoidI*:

fixes G (**structure**)
assumes *m-closed*:
$!!x\ y.\ [|\ x \in carrier\ G;\ y \in carrier\ G\ |] ==> x \otimes y \in carrier\ G$
and *one-closed*: $\mathbf{1} \in carrier\ G$
and *m-assoc*:
$!!x\ y\ z.\ [|\ x \in carrier\ G;\ y \in carrier\ G;\ z \in carrier\ G\ |] ==>$
$(x \otimes y) \otimes z = x \otimes (y \otimes z)$
and *l-one*: $!!x.\ x \in carrier\ G ==> \mathbf{1} \otimes x = x$
and *m-comm*:
$!!x\ y.\ [|\ x \in carrier\ G;\ y \in carrier\ G\ |] ==> x \otimes y = y \otimes x$
shows *comm-monoid G*
using *l-one*
by (*auto intro*!: *comm-monoid.intro comm-monoid-axioms.intro monoid.intro intro*: *prems simp*: *m-closed one-closed m-comm*)

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*
by (*rule comm-monoidI*) (*auto intro*: *m-assoc m-comm*)

lemma (**in** *comm-monoid*) *nat-pow-distr*:
$[|\ x \in carrier\ G;\ y \in carrier\ G\ |] ==>$
$(x \otimes y)\ (\hat{})\ (n::nat) = x\ (\hat{})\ n \otimes y\ (\hat{})\ n$
by (*induct n*) (*simp*, *simp add*: *m-ac*)

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*
by *unfold-locales* (*simp-all add*: *m-comm*)

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

by (*fast intro*: *group.group-comm-groupI groupI prems*)

lemma (**in** *comm-group*) *inv-mult*:
[| $x \in$ *carrier G*; $y \in$ *carrier G* |] ==> *inv* $(x \otimes y)$ = *inv* $x \otimes$ *inv* y
by (*simp add*: *m-ac inv-mult-group*)

2.7 The Lattice of Subgroups of a Group

theorem (**in** *group*) *subgroups-partial-order*:
partial-order (| *carrier* = $\{H.$ *subgroup H G*$\}$, *le* = *op* $\subseteq$ |)
by (*rule partial-order.intro*) *simp-all*

lemma (**in** *group*) *subgroup-self*:
subgroup (*carrier G*) *G*
by (*rule subgroupI*) *auto*

lemma (**in** *group*) *subgroup-imp-group*:
subgroup H G ==> *group* (*G*(| *carrier* := *H* |))
by (*erule subgroup.subgroup-is-group*) (*rule* ‹*group G*›)

lemma (**in** *group*) *is-monoid* [*intro*, *simp*]:
monoid G
by (*auto intro*: *monoid.intro m-assoc*)

lemma (**in** *group*) *subgroup-inv-equality*:
[| *subgroup H G*; $x \in H$ |] ==> *m-inv* (*G* (| *carrier* := *H* |)) x = *inv* x
apply (*rule-tac inv-equality* [*THEN sym*])
apply (*rule group.l-inv* [*OF subgroup-imp-group*, *simplified*], *assumption+*)
apply (*rule subsetD* [*OF subgroup.subset*], *assumption+*)
apply (*rule subsetD* [*OF subgroup.subset*], *assumption*)
apply (*rule-tac group.inv-closed* [*OF subgroup-imp-group*, *simplified*], *assumption+*)
done

theorem (**in** *group*) *subgroups-Inter*:
assumes *subgr*: (!!H. $H \in A$ ==> *subgroup H G*)
and *not-empty*: A ~= {}
shows *subgroup* $(\bigcap A)$ *G*
proof (*rule subgroupI*)
from *subgr* [*THEN subgroup.subset*] **and** *not-empty*
show $\bigcap A \subseteq$ *carrier G* **by** *blast*
next
from *subgr* [*THEN subgroup.one-closed*]
show $\bigcap A$ ~= {} **by** *blast*
next
fix x **assume** $x \in \bigcap A$
with *subgr* [*THEN subgroup.m-inv-closed*]
show *inv* $x \in \bigcap A$ **by** *blast*
next
fix x y **assume** $x \in \bigcap A$ $y \in \bigcap A$

```
  with subgr [THEN subgroup.m-closed]
  show x ⊗ y ∈ ⋂A by blast
qed

theorem (in group) subgroups-complete-lattice:
  complete-lattice (| carrier = {H. subgroup H G}, le = op ⊆ |)
    (is complete-lattice ?L)
proof (rule partial-order.complete-lattice-criterion1)
  show partial-order ?L by (rule subgroups-partial-order)
next
  have greatest ?L (carrier G) (carrier ?L)
    by (unfold greatest-def)
      (simp add: subgroup.subset subgroup-self)
  then show ∃G. greatest ?L G (carrier ?L) ..
next
  fix A
  assume L: A ⊆ carrier ?L and non-empty: A ~= {}
  then have Int-subgroup: subgroup (⋂A) G
    by (fastsimp intro: subgroups-Inter)
  have greatest ?L (⋂A) (Lower ?L A)
    (is greatest - ?Int -)
  proof (rule greatest-LowerI)
    fix H
    assume H: H ∈ A
    with L have subgroupH: subgroup H G by auto
    from subgroupH have groupH: group (G (| carrier := H |)) (is group ?H)
      by (rule subgroup-imp-group)
    from groupH have monoidH: monoid ?H
      by (rule group.is-monoid)
    from H have Int-subset: ?Int ⊆ H by fastsimp
    then show le ?L ?Int H by simp
  next
    fix H
    assume H: H ∈ Lower ?L A
    with L Int-subgroup show le ?L H ?Int
      by (fastsimp simp: Lower-def intro: Inter-greatest)
  next
    show A ⊆ carrier ?L by (rule L)
  next
    show ?Int ∈ carrier ?L by simp (rule Int-subgroup)
  qed
  then show ∃I. greatest ?L I (Lower ?L A) ..
qed

end
```

```
theory FiniteProduct imports Group begin
```

3 Product Operator for Commutative Monoids

3.1 Inductive Definition of a Relation for Products over Sets

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

```
inductive-set
  foldSetD :: ['a set, 'b => 'a => 'a, 'a] => ('b set * 'a) set
  for D :: 'a set and f :: 'b => 'a => 'a and e :: 'a
  where
    emptyI [intro]: e ∈ 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
  by (erule foldSetD.cases) auto

lemma Diff1-foldSetD:
  [| (A − {x}, y) ∈ foldSetD D f e; x ∈ A; f x y ∈ D |] ==>
   (A, f x y) ∈ foldSetD D f e
  apply (erule insert-Diff [THEN subst], rule foldSetD.intros)
    apply auto
  done

lemma foldSetD-imp-finite [simp]: (A, x) ∈ foldSetD D f e ==> finite A
  by (induct set: foldSetD) auto

lemma finite-imp-foldSetD:
  [| finite A; e ∈ D; !!x y. [| x ∈ A; y ∈ D |] ==> f x y ∈ D |] ==>
   EX x. (A, x) ∈ foldSetD D f e
proof (induct set: finite)
  case empty then show ?case by auto
next
  case (insert x F)
  then obtain y where y: (F, y) ∈ foldSetD D f e by auto
  with insert have y ∈ D by (auto dest: foldSetD-closed)
  with y and insert have (insert x F, f x y) ∈ foldSetD D f e
    by (intro foldSetD.intros) auto
  then show ?case ..
qed
```

3.2 Left-Commutative Operations

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

lemma (in LCD) foldSetD-closed [dest]:
  (A, z) ∈ foldSetD D f e ==> z ∈ D
  by (erule foldSetD.cases) auto

lemma (in LCD) Diff1-foldSetD:
  [| (A − {x}, y) ∈ foldSetD D f e; x ∈ A; A ⊆ B |] ==>
  (A, f x y) ∈ foldSetD D f e
  apply (subgoal-tac x ∈ B)
   prefer 2 apply fast
  apply (erule insert-Diff [THEN subst], rule foldSetD.intros)
    apply auto
  done

lemma (in LCD) foldSetD-imp-finite [simp]:
  (A, x) ∈ foldSetD D f e ==> finite A
  by (induct set: foldSetD) auto

lemma (in LCD) finite-imp-foldSetD:
  [| finite A; A ⊆ B; e ∈ D |] ==> EX x. (A, x) ∈ foldSetD D f e
proof (induct set: finite)
  case empty then show ?case by auto
next
  case (insert x F)
  then obtain y where y: (F, y) ∈ foldSetD D f e by auto
  with insert have y ∈ D by auto
  with y and insert have (insert x F, f x y) ∈ foldSetD D f e
    by (intro foldSetD.intros) auto
  then show ?case ..
qed

lemma (in LCD) foldSetD-determ-aux:
  e ∈ D ==> ∀ A x. A ⊆ B & card A < n −−> (A, x) ∈ foldSetD D f e −−>
    (∀ y. (A, y) ∈ foldSetD D f e −−> y = x)
  apply (induct n)
   apply (auto simp add: less-Suc-eq)
  apply (erule foldSetD.cases)
   apply blast
  apply (erule foldSetD.cases)
   apply blast
  apply clarify
```

force simplification of *card* $A <$ *card* (*insert* ...).

apply (*erule rev-mp*)
apply (*simp add*: *less-Suc-eq-le*)
apply (*rule impI*)
apply (*rename-tac xa Aa ya xb Ab yb*, *case-tac xa* = *xb*)
apply (*subgoal-tac Aa* = *Ab*)
prefer *2* **apply** (*blast elim*!: *equalityE*)
apply *blast*

case $xa \notin xb$.

apply (*subgoal-tac* $Aa - \{xb\} = Ab - \{xa\}$ & $xb \in Aa$ & $xa \in Ab$)
prefer *2* **apply** (*blast elim*!: *equalityE*)
apply *clarify*
apply (*subgoal-tac Aa* = *insert xb Ab* $- \{xa\}$)
prefer *2* **apply** *blast*
apply (*subgoal-tac card Aa* $\leq$ *card Ab*)
prefer *2*
apply (*rule Suc-le-mono* [*THEN subst*])
apply (*simp add*: *card-Suc-Diff1*)
apply (*rule-tac A1* = $Aa - \{xb\}$ **in** *finite-imp-foldSetD* [*THEN exE*])
apply (*blast intro*: *foldSetD-imp-finite finite-Diff*)
apply *best*
apply *assumption*
apply (*frule* (*1*) *Diff1-foldSetD*)
apply *best*
apply (*subgoal-tac ya* = *f xb x*)
prefer *2*
apply (*subgoal-tac Aa* $\subseteq$ *B*)
prefer *2* **apply** *best*
apply (*blast del*: *equalityCE*)
apply (*subgoal-tac* ($Ab - \{xa\}$, x) $\in$ *foldSetD D f e*)
prefer *2* **apply** *simp*
apply (*subgoal-tac yb* = *f xa x*)
prefer *2*
apply (*blast del*: *equalityCE dest*: *Diff1-foldSetD*)
apply (*simp* (*no-asm-simp*))
apply (*rule left-commute*)
apply *assumption*
apply *best*
apply *best*
done

lemma (**in** *LCD*) *foldSetD-determ*:
[| (A, x) $\in$ *foldSetD D f e*; (A, y) $\in$ *foldSetD D f e*; $e \in D$; $A \subseteq B$ |]
==> $y = x$
by (*blast intro*: *foldSetD-determ-aux* [*rule-format*])

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

```
  by (unfold foldD-def) (blast intro: foldSetD-determ)

lemma foldD-empty [simp]:
  e ∈ D ==> foldD D f e {} = e
  by (unfold foldD-def) blast

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)
  apply auto
  apply (rule-tac A1 = A in finite-imp-foldSetD [THEN exE])
     apply (fastsimp dest: foldSetD-imp-finite)
    apply assumption
   apply assumption
  apply (blast intro: foldSetD-determ)
  done

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)
  apply (unfold foldD-def)
  apply (simp add: foldD-insert-aux)
  apply (rule the-equality)
   apply (auto intro: finite-imp-foldSetD
     cong add: conj-cong simp add: foldD-def [symmetric] foldD-equality)
  done

lemma (in LCD) foldD-closed [simp]:
  [| finite A; e ∈ D; A ⊆ B |] ==> foldD D f e A ∈ D
proof (induct set: finite)
  case empty then show ?case by (simp add: foldD-empty)
next
  case insert then show ?case by (simp add: foldD-insert)
qed

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
  apply (induct set: finite)
   apply simp
  apply (auto simp add: left-commute foldD-insert)
  done

lemma Int-mono2:
  [| A ⊆ C; B ⊆ C |] ==> A Int B ⊆ C
  by blast

lemma (in LCD) foldD-nest-Un-Int:
```

[| *finite* A; *finite* C; $e \in D$; $A \subseteq B$; $C \subseteq B$ |] ==>
foldD D f (*foldD* D f e C) A = *foldD* D f (*foldD* D f e (A *Int* C)) (A *Un* C)
apply (*induct set*: *finite*)
apply *simp*
apply (*simp add*: *foldD-insert foldD-commute Int-insert-left insert-absorb*
Int-mono2 Un-subset-iff)
done

lemma (**in** *LCD*) *foldD-nest-Un-disjoint*:
[| *finite* A; *finite* B; A *Int* $B = \{\}$; $e \in D$; $A \subseteq B$; $C \subseteq B$ |]
==> *foldD* D f e (A *Un* B) = *foldD* D f (*foldD* D f e B) A
by (*simp add*: *foldD-nest-Un-Int*)

— Delete rules to do with *foldSetD* relation.

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

3.3 Commutative Monoids

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

locale *ACeD* =
fixes D :: $'a$ *set*
and f :: $'a$ => $'a$ => $'a$ (**infixl** $\cdot$ *70*)
and e :: $'a$
assumes *ident* [*simp*]: $x \in D$ ==> $x \cdot e = x$
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 −
assume D: $x \in D$ $y \in D$ $z \in D$
then have $x \cdot (y \cdot z) = (y \cdot z) \cdot x$ **by** (*simp add*: *commute*)
also from D **have** ... $= y \cdot (z \cdot x)$ **by** (*simp add*: *assoc*)
also from D **have** $z \cdot x = x \cdot z$ **by** (*simp add*: *commute*)
finally show *?thesis* **.**
qed

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

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

assume $x \in D$
then have $x \cdot e = x$ **by** (*rule ident*)
with ‹$x \in D$› **show** *?thesis* **by** (*simp add*: *commute*)
qed

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*)
apply (*induct set*: *finite*)
apply (*simp add*: *left-commute LCD.foldD-closed* [*OF LCD.intro* [*of D*]])
apply (*simp add*: *AC insert-absorb Int-insert-left*
LCD.foldD-insert [*OF LCD.intro* [*of D*]]
LCD.foldD-closed [*OF LCD.intro* [*of D*]]
Int-mono2 Un-subset-iff)
done

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*
by (*simp add*: *foldD-Un-Int*
left-commute LCD.foldD-closed [*OF LCD.intro* [*of D*]] *Un-subset-iff*)

3.4 Products over Finite Sets

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

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

lemma (**in** *comm-monoid*) *finprod-empty* [*simp*]:
finprod G f {} = **1**
by (*simp add*: *finprod-def*)

declare *funcsetI* [*intro*]

funcset-mem [*dest*]

lemma (**in** *comm-monoid*) *finprod-insert* [*simp*]:
[| *finite F*; $a \notin F$; $f \in F$ −> *carrier G*; $f\ a \in$ *carrier G* |] ==>
finprod G f (*insert a F*) = $f\ a \otimes$ *finprod G f F*
apply (*rule trans*)
apply (*simp add*: *finprod-def*)
apply (*rule trans*)
apply (*rule LCD.foldD-insert* [*OF LCD.intro* [*of insert a F*]])
apply *simp*
apply (*rule m-lcomm*)
apply *fast*
apply *fast*
apply *assumption*
apply (*fastsimp intro*: *m-closed*)
apply *simp+*
apply *fast*
apply (*auto simp add*: *finprod-def*)
done

lemma (**in** *comm-monoid*) *finprod-one* [*simp*]:
finite A ==> ($\bigotimes$ *i*:*A*. **1**) = **1**
proof (*induct set*: *finite*)
case *empty* **show** *?case* **by** *simp*
next
case (*insert a A*)
have (%*i*. **1**) $\in$ *A* −> *carrier G* **by** *auto*
with *insert* **show** *?case* **by** *simp*
qed

lemma (**in** *comm-monoid*) *finprod-closed* [*simp*]:
fixes *A*
assumes *fin*: *finite A* **and** *f*: $f \in A$ −> *carrier G*
shows *finprod G f A* $\in$ *carrier G*
using *fin f*
proof *induct*
case *empty* **show** *?case* **by** *simp*
next
case (*insert a A*)
then have *a*: $f\ a \in$ *carrier G* **by** *fast*
from *insert* **have** *A*: $f \in A$ −> *carrier G* **by** *fast*
from *insert A a* **show** *?case* **by** *simp*
qed

lemma *funcset-Int-left* [*simp*, *intro*]:
[| $f \in A$ −> *C*; $f \in B$ −> *C* |] ==> $f \in A$ *Int B* −> *C*
by *fast*

lemma *funcset-Un-left* [*iff*]:

$(f \in A\ Un\ B\ {-}{>}\ C) = (f \in A\ {-}{>}\ C\ \&\ f \in B\ {-}{>}\ C)$
by *fast*

lemma (**in** *comm-monoid*) *finprod-Un-Int*:
[| *finite A*; *finite B*; $g \in A\ {-}{>}$ *carrier G*; $g \in B\ {-}{>}$ *carrier G* |] ==>
finprod G g (*A Un B*) $\otimes$ *finprod G g* (*A Int B*) =
finprod G g A $\otimes$ *finprod G g B*
— The reversed orientation looks more natural, but LOOPS as a simprule!
proof (*induct set*: *finite*)
case *empty* **then show** *?case* **by** (*simp add*: *finprod-closed*)
next
case (*insert a A*)
then have *a*: $g\ a \in$ *carrier G* **by** *fast*
from *insert* **have** *A*: $g \in A\ {-}{>}$ *carrier G* **by** *fast*
from *insert A a* **show** *?case*
by (*simp add*: *m-ac Int-insert-left insert-absorb finprod-closed*
Int-mono2 Un-subset-iff)
qed

lemma (**in** *comm-monoid*) *finprod-Un-disjoint*:
[| *finite A*; *finite B*; *A Int B* = {};
$g \in A\ {-}{>}$ *carrier G*; $g \in B\ {-}{>}$ *carrier G* |]
==> *finprod G g* (*A Un B*) = *finprod G g A* $\otimes$ *finprod G g B*
apply (*subst finprod-Un-Int* [*symmetric*])
apply (*auto simp add*: *finprod-closed*)
done

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 (*induct set*: *finite*)
case *empty* **show** *?case* **by** *simp*
next
case (*insert a A*) **then**
have *fA*: $f \in A\ {-}{>}$ *carrier G* **by** *fast*
from *insert* **have** *fa*: $f\ a \in$ *carrier G* **by** *fast*
from *insert* **have** *gA*: $g \in A\ {-}{>}$ *carrier G* **by** *fast*
from *insert* **have** *ga*: $g\ a \in$ *carrier G* **by** *fast*
from *insert* **have** *fgA*: (%*x*. *f x* $\otimes$ *g x*) $\in A\ {-}{>}$ *carrier G*
by (*simp add*: *Pi-def*)
show *?case*
by (*simp add*: *insert fA fa gA ga fgA m-ac*)
qed

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 −
assume *prems*: *A* = *B* $g \in B\ {-}{>}$ *carrier G*

```
    !!i. i ∈ B ==> f i = g i
  show ?thesis
  proof (cases finite B)
    case True
    then have !!A. [| A = B; g ∈ B -> carrier G;
      !!i. i ∈ B ==> f i = g i |] ==> finprod G f A = finprod G g B
    proof induct
      case empty thus ?case by simp
    next
      case (insert x B)
      then have finprod G f A = finprod G f (insert x B) by simp
      also from insert have ... = f x ⊗ finprod G f B
      proof (intro finprod-insert)
        show finite B by fact
      next
        show x ~: B by fact
      next
        assume x ~: B !!i. i ∈ insert x B ⟹ f i = g i
          g ∈ insert x B → carrier G
        thus f ∈ B -> carrier G by fastsimp
      next
        assume x ~: B !!i. i ∈ insert x B ⟹ f i = g i
          g ∈ insert x B → carrier G
        thus f x ∈ carrier G by fastsimp
      qed
      also from insert have ... = g x ⊗ finprod G g B by fastsimp
      also from insert have ... = finprod G g (insert x B)
      by (intro finprod-insert [THEN sym]) auto
      finally show ?case .
    qed
    with prems show ?thesis by simp
  next
    case False with prems show ?thesis by (simp add: finprod-def)
  qed
qed

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

  by (rule finprod-cong') force+
```

Usually, if this rule causes a failed congruence proof error, the reason is that the premise *g* ∈ *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 ∈ {0::nat} −> carrier G ==> finprod G f {..0} = f 0
by (simp add: Pi-def)

lemma (in comm-monoid) finprod-Suc [simp]:
  f ∈ {..Suc n} −> carrier G ==>
   finprod G f {..Suc n} = (f (Suc n) ⊗ finprod G f {..n})
by (simp add: Pi-def atMost-Suc)

lemma (in comm-monoid) finprod-Suc2:
  f ∈ {..Suc n} −> carrier G ==>
   finprod G f {..Suc n} = (finprod G (%i. f (Suc i)) {..n} ⊗ f 0)
proof (induct n)
  case 0 thus ?case by (simp add: Pi-def)
next
  case Suc thus ?case by (simp add: m-assoc Pi-def)
qed

lemma (in comm-monoid) finprod-mult [simp]:
  [| f ∈ {..n} −> carrier G; g ∈ {..n} −> carrier G |] ==>
     finprod G (%i. f i ⊗ g i) {..n::nat} =
     finprod G f {..n} ⊗ finprod G g {..n}
  by (induct n) (simp-all add: m-ac Pi-def)

end
```

```
theory Exponent imports Main Primes Binomial begin
```

4 The Combinatorial Argument Underlying the First Sylow Theorem

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

4.1 Prime Theorems

```
lemma prime-imp-one-less: prime p ==> Suc 0 < p
by (unfold prime-def, force)

lemma prime-iff:
  (prime p) = (Suc 0 < p & (∀ a b. p dvd a*b −−> (p dvd a) | (p dvd b)))
apply (auto simp add: prime-imp-one-less)
apply (blast dest!: prime-dvd-mult)
apply (auto simp add: prime-def)
apply (erule dvdE)
```

apply (*case-tac k=0*, *simp*)
apply (*drule-tac x* = *m* **in** *spec*)
apply (*drule-tac x* = *k* **in** *spec*)
apply (*simp add*: *dvd-mult-cancel1 dvd-mult-cancel2*)
done

lemma *zero-less-prime-power*: *prime p* ==> *0* < *p*ˆ*a*
by (*force simp add*: *prime-iff*)

lemma *zero-less-card-empty*: [| *finite S*; *S* ≠ {} |] ==> *0* < *card*(*S*)
by (*rule ccontr*, *simp*)

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*)
apply (*simp add*: *prime-iff*)
apply (*frule dvd-mult-left*)
apply (*subgoal-tac p dvd m* | *p dvd n*)
 prefer *2* **apply** *blast*
apply (*erule disjE*)
apply (*rule disjI1*)
apply (*rule-tac* [*2*] *disjI2*)
apply (*erule-tac n* = *m* **in** *dvdE*)
apply (*erule-tac* [*2*] *n* = *n* **in** *dvdE*, *auto*)
done

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*)
apply (*induct-tac c*)
 apply *clarify*
 apply (*case-tac a*)
 apply *simp*
 apply *simp*

apply *simp*
apply *clarify*
apply (*erule prime-dvd-cases* [*THEN disjE*], *assumption*, *auto*)

 apply (*case-tac a*)
 apply *simp*
 apply *clarify*
 apply (*drule spec*, *drule spec*, *erule* (*1*) *notE impE*)
 apply (*drule-tac x* = *nat* **in** *spec*)
 apply (*drule-tac x* = *b* **in** *spec*)
 apply *simp*

apply (*case-tac b*)
apply *simp*
apply *clarify*
apply (*drule spec*, *drule spec*, *erule* (*1*) *notE impE*)
apply (*drule-tac x* = *a* **in** *spec*)
apply (*drule-tac x* = *nat* **in** *spec*, *simp*)
done

lemma *div-combine*:
[| *prime p*; ~ (*p* ^ (*Suc r*) *dvd n*); *p*^(*a*+*r*) *dvd n*∗*k* |]
==> *p* ^ *a dvd k*
by (*drule-tac a* = *Suc r* **and** *b* = *a* **in** *prime-power-dvd-cases*, *assumption*, *auto*)

lemma *Suc-le-power*: *Suc 0* < *p* ==> *Suc n* <= *p*^*n*
apply (*induct-tac n*)
apply (*simp* (*no-asm-simp*))
apply *simp*
apply (*subgoal-tac 2* ∗ *n* + *2* <= *p* ∗ *p*^*n*, *simp*)
apply (*subgoal-tac 2* ∗ *p*^*n* <= *p* ∗ *p*^*n*)
apply *arith*
apply (*drule-tac k* = *2* **in** *mult-le-mono2*, *simp*)
done

lemma *power-dvd-bound*: [|*p*^*n dvd a*; *Suc 0* < *p*; *a* > *0*|] ==> *n* < *a*
apply (*drule dvd-imp-le*)
apply (*drule-tac* [*2*] *n* = *n* **in** *Suc-le-power*, *auto*)
done

4.2 Exponent Theorems

lemma *exponent-ge* [*rule-format*]:
[|*p*^*k dvd n*; *prime p*; *0*<*n*|] ==> *k* <= *exponent p n*
apply (*simp add*: *exponent-def*)
apply (*erule Greatest-le*)
apply (*blast dest*: *prime-imp-one-less power-dvd-bound*)
done

lemma *power-exponent-dvd*: *s*>*0* ==> (*p* ^ *exponent p s*) *dvd s*
apply (*simp add*: *exponent-def*)
apply *clarify*
apply (*rule-tac k* = *0* **in** *GreatestI*)
prefer *2* **apply** (*blast dest*: *prime-imp-one-less power-dvd-bound*, *simp*)
done

lemma *power-Suc-exponent-Not-dvd*:

[|(p ∗ p ˆ *exponent p s*) *dvd s*; *prime p* |] ==> *s=0*
apply (*subgoal-tac p* ˆ *Suc* (*exponent p s*) *dvd s*)
prefer *2* **apply** *simp*
apply (*rule ccontr*)
apply (*drule exponent-ge*, *auto*)
done

lemma *exponent-power-eq* [*simp*]: *prime p* ==> *exponent p* (*p*ˆ*a*) = *a*
apply (*simp* (*no-asm-simp*) *add*: *exponent-def*)
apply (*rule Greatest-equality*, *simp*)
apply (*simp* (*no-asm-simp*) *add*: *prime-imp-one-less power-dvd-imp-le*)
done

lemma *exponent-equalityI*:
!*r*::*nat*. (*p*ˆ*r dvd a*) = (*p*ˆ*r dvd b*) ==> *exponent p a* = *exponent p b*
by (*simp* (*no-asm-simp*) *add*: *exponent-def*)

lemma *exponent-eq-0* [*simp*]: ¬ *prime p* ==> *exponent p s* = *0*
by (*simp* (*no-asm-simp*) *add*: *exponent-def*)

lemma *exponent-mult-add1*: [| *a* > *0*; *b* > *0* |]
==> (*exponent p a*) + (*exponent p b*) <= *exponent p* (*a* ∗ *b*)
apply (*case-tac prime p*)
apply (*rule exponent-ge*)
apply (*auto simp add*: *power-add*)
apply (*blast intro*: *prime-imp-one-less power-exponent-dvd mult-dvd-mono*)
done

lemma *exponent-mult-add2*: [| *a* > *0*; *b* > *0* |]
==> *exponent p* (*a* ∗ *b*) <= (*exponent p a*) + (*exponent p b*)
apply (*case-tac prime p*)
apply (*rule leI*, *clarify*)
apply (*cut-tac p* = *p* **and** *s* = *a*∗*b* **in** *power-exponent-dvd*, *auto*)
apply (*subgoal-tac p* ˆ (*Suc* (*exponent p a* + *exponent p b*)) *dvd a* ∗ *b*)
apply (*rule-tac* [*2*] *le-imp-power-dvd* [*THEN dvd-trans*])
prefer *3* **apply** *assumption*
prefer *2* **apply** *simp*
apply (*frule-tac a* = *Suc* (*exponent p a*) **and** *b* = *Suc* (*exponent p b*) **in** *prime-power-dvd-cases*)
apply (*assumption*, *force*, *simp*)
apply (*blast dest*: *power-Suc-exponent-Not-dvd*)
done

lemma *exponent-mult-add*: [| *a* > *0*; *b* > *0* |]
==> *exponent p* (*a* ∗ *b*) = (*exponent p a*) + (*exponent p b*)
by (*blast intro*: *exponent-mult-add1 exponent-mult-add2 order-antisym*)

lemma *not-divides-exponent-0*: ~ (*p dvd n*) ==> *exponent p n* = *0*
apply (*case-tac exponent p n*, *simp*)
apply (*case-tac n*, *simp*)
apply (*cut-tac s* = *n* **and** *p* = *p* **in** *power-exponent-dvd*)
apply (*auto dest*: *dvd-mult-left*)
done

lemma *exponent-1-eq-0* [*simp*]: *exponent p* (*Suc 0*) = *0*
apply (*case-tac prime p*)
apply (*auto simp add*: *prime-iff not-divides-exponent-0*)
done

4.3 Main Combinatorial Argument

lemma *le-extend-mult*: [| *c* > *0*; *a* <= *b* |] ==> *a* <= *b* * (*c*::*nat*)
apply (*rule-tac P* = %*x*. *x* <= *b* * *c* **in** *subst*)
apply (*rule mult-1-right*)
apply (*rule mult-le-mono*, *auto*)
done

lemma *p-fac-forw-lemma*:
[| (*m*::*nat*) > *0*; *k* > *0*; *k* < *p*ˆ*a*; (*p*ˆ*r*) *dvd* (*p*ˆ*a*)* *m* − *k* |] ==> *r* <= *a*
apply (*rule notnotD*)
apply (*rule notI*)
apply (*drule contrapos-nn* [*OF* - *leI*, *THEN notnotD*], *assumption*)
apply (*drule less-imp-le* [*of a*])
apply (*drule le-imp-power-dvd*)
apply (*drule-tac n* = *p* ˆ *r* **in** *dvd-trans*, *assumption*)
apply(*metis dvd-diffD1 dvd-triv-right le-extend-mult linorder-linear linorder-not-less mult-commute nat-dvd-not-less neq0-conv*)
done

lemma *p-fac-forw*: [| (*m*::*nat*) > *0*; *k*>*0*; *k* < *p*ˆ*a*; (*p*ˆ*r*) *dvd* (*p*ˆ*a*)* *m* − *k* |]
==> (*p*ˆ*r*) *dvd* (*p*ˆ*a*) − *k*
apply (*frule-tac k1* = *k* **and** *i* = *p* **in** *p-fac-forw-lemma* [*THEN le-imp-power-dvd*], *auto*)
apply (*subgoal-tac p*ˆ*r dvd p*ˆ*a***m*)
prefer *2* **apply** (*blast intro*: *dvd-mult2*)
apply (*drule dvd-diffD1*)
apply *assumption*
prefer *2* **apply** (*blast intro*: *dvd-diff*)
apply (*drule gr0-implies-Suc*, *auto*)
done

lemma *r-le-a-forw*:
[| (*k*::*nat*) > *0*; *k* < *p*ˆ*a*; *p*>*0*; (*p*ˆ*r*) *dvd* (*p*ˆ*a*) − *k* |] ==> *r* <= *a*

by (*rule-tac m* = *Suc 0* **in** *p-fac-forw-lemma*, *auto*)

lemma *p-fac-backw*: [| *m*>*0*; *k*>*0*; (*p*::*nat*)≠*0*; *k* < *p* ^ *a*; (*p* ^ *r*) *dvd p* ^ *a* − *k* |]
==> (*p* ^ *r*) *dvd* (*p* ^ *a*)∗*m* − *k*
apply (*frule-tac k1* = *k* **and** *i* = *p* **in** *r-le-a-forw* [*THEN le-imp-power-dvd*], *auto*)
apply (*subgoal-tac p* ^ *r dvd p* ^ *a*∗*m*)
prefer *2* **apply** (*blast intro*: *dvd-mult2*)
apply (*drule dvd-diffD1*)
apply *assumption*
prefer *2* **apply** (*blast intro*: *dvd-diff*)
apply (*drule less-imp-Suc-add*, *auto*)
done

lemma *exponent-p-a-m-k-equation*: [| *m*>*0*; *k*>*0*; (*p*::*nat*)≠*0*; *k* < *p* ^ *a* |]
==> *exponent p* (*p* ^ *a* ∗ *m* − *k*) = *exponent p* (*p* ^ *a* − *k*)
apply (*blast intro*: *exponent-equalityI p-fac-forw p-fac-backw*)
done

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*]:
[| ∀ *i*. *Suc i* < *K* −−> *exponent p* (*Suc i*) = *exponent p* (*Suc*(*j*+*i*))|]
==> *k*<*K* −−> *exponent p* ((*j*+*k*) *choose k*) = *0*
apply (*case-tac prime p*)
prefer *2* **apply** *simp*
apply (*induct-tac k*)
apply (*simp* (*no-asm*))

apply (*subgoal-tac* (*Suc* (*j*+*n*) *choose Suc n*) > *0*)
prefer *2* **apply** (*simp add*: *zero-less-binomial-iff*, *clarify*)
apply (*subgoal-tac exponent p* ((*Suc* (*j*+*n*) *choose Suc n*) ∗ *Suc n*) =
exponent p (*Suc n*))

First, use the assumed equation. We simplify the LHS to *exponent p* (*Suc* (*j* + *n*) *choose Suc n*) + *exponent p* (*Suc n*) the common terms cancel, proving the conclusion.

apply (*simp del*: *bad-Sucs add*: *exponent-mult-add*)

Establishing the equation requires first applying *Suc-times-binomial-eq* ...

apply (*simp del*: *bad-Sucs add*: *Suc-times-binomial-eq* [*symmetric*])

...then *exponent-mult-add* and the quantified premise.

apply (*simp del*: *bad-Sucs add*: *zero-less-binomial-iff exponent-mult-add*)
done

lemma *p-not-div-choose*:

```
  [| k<K;  k<=n;
      ∀ j. 0<j & j<K --> exponent p (n - k + (K - j)) = exponent p (K -
j)|]
   ==> exponent p (n choose k) = 0
apply (cut-tac j = n-k and k = k and p = p in p-not-div-choose-lemma)
  prefer 3 apply simp
 prefer 2 apply assumption
apply (drule-tac x = K - Suc i in spec)
apply (simp add: Suc-diff-le)
done


lemma const-p-fac-right:
  m>0 ==> exponent p ((p^a * m - Suc 0) choose (p^a - Suc 0)) = 0
apply (case-tac prime p)
 prefer 2 apply simp
apply (frule-tac a = a in zero-less-prime-power)
apply (rule-tac K = p^a in p-not-div-choose)
   apply simp
  apply simp
 apply (case-tac m)
  apply (case-tac [2] p^a)
   apply auto

apply (subgoal-tac 0<p)
 prefer 2 apply (force dest!: prime-imp-one-less)
apply (subst exponent-p-a-m-k-equation, auto)
done

lemma const-p-fac:
  m>0 ==> exponent p (((p^a) * m) choose p^a) = exponent p m
apply (case-tac prime p)
 prefer 2 apply simp
apply (subgoal-tac 0 < p^a * m & p^a <= p^a * m)
 prefer 2 apply (force simp add: prime-iff)
```

A similar trick to the one used in *p-not-div-choose-lemma*: insert an equation; use *exponent-mult-add* on the LHS; on the RHS, first transform the binomial coefficient, then use *exponent-mult-add*.

```
apply (subgoal-tac exponent p ((( (p^a) * m) choose p^a) * p^a) =
                   a + exponent p m)
 apply (simp del: bad-Sucs add: zero-less-binomial-iff exponent-mult-add prime-iff)
```

one subgoal left!

```
apply (subst times-binomial-minus1-eq, simp, simp)
apply (subst exponent-mult-add, simp)
apply (simp (no-asm-simp) add: zero-less-binomial-iff)
apply arith
apply (simp del: bad-Sucs add: exponent-mult-add const-p-fac-right)
```

done

end

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

5 Cosets and Quotient Groups

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

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

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

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

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

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

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

5.1 Basic Properties of Cosets

lemma (**in** *group*) *coset-mult-assoc*:
  [| $M \subseteq$ *carrier* G; $g \in$ *carrier* G; $h \in$ *carrier* G |]
  ==> (M #> g) #> h = M #> ($g \otimes h$)
by (*force simp add*: *r-coset-def m-assoc*)

lemma (**in** *group*) *coset-mult-one* [*simp*]: $M \subseteq$ *carrier* G ==> M #> **1** = M
by (*force simp add*: *r-coset-def*)

lemma (**in** *group*) *coset-mult-inv1*:
  [| M #> ($x \otimes$ (*inv* y)) = M; $x \in$ *carrier* G ; $y \in$ *carrier* G;
    $M \subseteq$ *carrier* G |] ==> M #> x = M #> y
apply (*erule subst* [*of* **concl**: %z. M #> x = z #> y])

apply (*simp add*: *coset-mult-assoc m-assoc*)
done

lemma (**in** *group*) *coset-mult-inv2*:
$[| \ M \ \#> x = M \ \#> y; \ \ x \in carrier \ G; \ \ y \in carrier \ G; \ \ M \subseteq carrier \ G \ |]$
$==> M \ \#> (x \otimes (inv \ y)) = M$
apply (*simp add*: *coset-mult-assoc* [*symmetric*])
apply (*simp add*: *coset-mult-assoc*)
done

lemma (**in** *group*) *coset-join1*:
$[| \ H \ \#> x = H; \ \ x \in carrier \ G; \ \ subgroup \ H \ G \ |] ==> x \in H$
apply (*erule subst*)
apply (*simp add*: *r-coset-def*)
apply (*blast intro*: *l-one subgroup.one-closed sym*)
done

lemma (**in** *group*) *solve-equation*:
$[\![subgroup \ H \ G; x \in H; y \in H]\!] \Longrightarrow \exists \, h{\in}H. \ y = h \otimes x$
apply (*rule bexI* [*of* - $y \otimes (inv \ x)$])
apply (*auto simp add*: *subgroup.m-closed subgroup.m-inv-closed m-assoc*
subgroup.subset [*THEN subsetD*])
done

lemma (**in** *group*) *repr-independence*:
$[\![y \in H \ \#> x; \ \ x \in carrier \ G; subgroup \ H \ G]\!] \Longrightarrow H \ \#> x = H \ \#> y$
by (*auto simp add*: *r-coset-def m-assoc* [*symmetric*]
subgroup.subset [*THEN subsetD*]
subgroup.m-closed solve-equation)

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*.
by (*force simp add*: *subgroup.m-closed r-coset-def solve-equation*)

lemma (**in** *monoid*) *r-coset-subset-G*:
$[| \ H \subseteq carrier \ G; x \in carrier \ G \ |] ==> H \ \#> x \subseteq carrier \ G$
by (*auto simp add*: *r-coset-def*)

lemma (**in** *group*) *rcosI*:
$[| \ h \in H; H \subseteq carrier \ G; x \in carrier \ G|] ==> h \otimes x \in H \ \#> x$
by (*auto simp add*: *r-coset-def*)

lemma (**in** *group*) *rcosetsI*:
$[\![H \subseteq carrier \ G; x \in carrier \ G]\!] \Longrightarrow H \ \#> x \in rcosets \ H$
by (*auto simp add*: *RCOSETS-def*)

Really needed?

lemma (**in** *group*) *transpose-inv*:

```
  [| x ⊗ y = z;  x ∈ carrier G;  y ∈ carrier G;  z ∈ carrier G |]
   ==> (inv x) ⊗ z = y
by (force simp add: m-assoc [symmetric])

lemma (in group) rcos-self: [| x ∈ carrier G; subgroup H G |] ==> x ∈ H #> x
apply (simp add: r-coset-def)
apply (blast intro: sym l-one subgroup.subset [THEN subsetD]
                    subgroup.one-closed)
done
```

Opposite of *repr-independence*

```
lemma (in group) repr-independenceD:
  includes subgroup H G
  assumes ycarr: y ∈ carrier G
      and repr:  H #> x = H #> y
  shows y ∈ H #> x
  apply (subst repr)
  apply (intro rcos-self)
   apply (rule ycarr)
   apply (rule is-subgroup)
  done
```

Elements of a right coset are in the carrier

```
lemma (in subgroup) elemrcos-carrier:
  includes group
  assumes acarr: a ∈ carrier G
    and a': a' ∈ H #> a
  shows a' ∈ carrier G
proof -
  from subset and acarr
  have H #> a ⊆ carrier G by (rule r-coset-subset-G)
  from this and a'
  show a' ∈ carrier G
    by fast
qed

lemma (in subgroup) rcos-const:
  includes group
  assumes hH: h ∈ H
  shows H #> h = H
  apply (unfold r-coset-def)
  apply rule
   apply rule
   apply clarsimp
   apply (intro subgroup.m-closed)
     apply (rule is-subgroup)
    apply assumption
   apply (rule hH)
  apply rule
```

```
  apply simp
proof -
  fix h'
  assume h'H: h' ∈ H
  note carr = hH[THEN mem-carrier] h'H[THEN mem-carrier]
  from carr
  have a: h' = (h' ⊗ inv h) ⊗ h by (simp add: m-assoc)
  from h'H hH
  have h' ⊗ inv h ∈ H by simp
  from this and a
  show ∃x∈H. h' = x ⊗ h by fast
qed
```

Step one for lemma *rcos-module*

```
lemma (in subgroup) rcos-module-imp:
  includes group
  assumes xcarr: x ∈ carrier G
      and x'cos: x' ∈ H #> x
  shows (x' ⊗ inv x) ∈ H
proof -
  from xcarr x'cos
      have x'carr: x' ∈ carrier G
      by (rule elemrcos-carrier[OF is-group])
  from xcarr
      have ixcarr: inv x ∈ carrier G
      by simp
  from x'cos
      have ∃h∈H. x' = h ⊗ x
      unfolding r-coset-def
      by fast
  from this
      obtain h
        where hH: h ∈ H
        and x': x' = h ⊗ x
      by auto
  from hH and subset
      have hcarr: h ∈ carrier G by fast
  note carr = xcarr x'carr hcarr
  from x' and carr
      have x' ⊗ (inv x) = (h ⊗ x) ⊗ (inv x) by fast
  also from carr
      have ... = h ⊗ (x ⊗ inv x) by (simp add: m-assoc)
  also from carr
      have ... = h ⊗ 1 by simp
  also from carr
      have ... = h by simp
  finally
      have x' ⊗ (inv x) = h by simp
  from hH this
```

show $x' \otimes (inv\ x) \in H$ **by** *simp*
qed

Step two for lemma *rcos-module*

lemma (**in** *subgroup*) *rcos-module-rev*:
includes *group*
assumes *carr*: $x \in carrier\ G\ x' \in carrier\ G$
and *xixH*: $(x' \otimes inv\ x) \in H$
shows $x' \in H\ \#{>}\ x$
proof −
from *xixH*
have $\exists h \in H.\ x' \otimes (inv\ x) = h$ **by** *fast*
from *this*
obtain h
where *hH*: $h \in H$
and *hsym*: $x' \otimes (inv\ x) = h$
by *fast*
from *hH subset* **have** *hcarr*: $h \in carrier\ G$ **by** *simp*
note *carr* = *carr hcarr*
from *hsym*[*symmetric*] **have** $h \otimes x = x' \otimes (inv\ x) \otimes x$ **by** *fast*
also from *carr*
have $\ldots = x' \otimes ((inv\ x) \otimes x)$ **by** (*simp add*: *m-assoc*)
also from *carr*
have $\ldots = x' \otimes \mathbf{1}$ **by** (*simp add*: *l-inv*)
also from *carr*
have $\ldots = x'$ **by** *simp*
finally
have $h \otimes x = x'$ **by** *simp*
from *this*[*symmetric*] **and** *hH*
show $x' \in H\ \#{>}\ x$
unfolding *r-coset-def*
by *fast*
qed

Module property of right cosets

lemma (**in** *subgroup*) *rcos-module*:
includes *group*
assumes *carr*: $x \in carrier\ G\ x' \in carrier\ G$
shows $(x' \in H\ \#{>}\ x) = (x' \otimes inv\ x \in H)$
proof
assume $x' \in H\ \#{>}\ x$
from *this* **and** *carr*
show $x' \otimes inv\ x \in H$
by (*intro rcos-module-imp*[*OF is-group*])
next
assume $x' \otimes inv\ x \in H$
from *this* **and** *carr*
show $x' \in H\ \#{>}\ x$
by (*intro rcos-module-rev*[*OF is-group*])

qed

Right cosets are subsets of the carrier.

lemma (**in** *subgroup*) *rcosets-carrier*:
  includes *group*
  assumes *XH*: $X \in$ *rcosets H*
  shows $X \subseteq$ *carrier G*
proof $-$
  from *XH* **have** $\exists x\in$ *carrier G*. $X = H$ #> x
    unfolding *RCOSETS-def*
    by *fast*
  from *this*
    obtain x
      where *xcarr*: $x\in$ *carrier G*
      and *X*: $X = H$ #> x
    by *fast*
  from *subset* **and** *xcarr*
    show $X \subseteq$ *carrier G*
    unfolding *X*
    by (*rule r-coset-subset-G*)
qed

Multiplication of general subsets

lemma (**in** *monoid*) *set-mult-closed*:
  assumes *Acarr*: $A \subseteq$ *carrier G*
    and *Bcarr*: $B \subseteq$ *carrier G*
  shows A <#> $B \subseteq$ *carrier G*
apply *rule* **apply** (*simp add*: *set-mult-def*, *clarsimp*)
proof $-$
  fix *a b*
  assume $a \in A$
  from *this* **and** *Acarr*
    have *acarr*: $a \in$ *carrier G* **by** *fast*

  assume $b \in B$
  from *this* **and** *Bcarr*
    have *bcarr*: $b \in$ *carrier G* **by** *fast*

  from *acarr bcarr*
    show $a \otimes b \in$ *carrier G* **by** (*rule m-closed*)
qed

lemma (**in** *comm-group*) *mult-subgroups*:
  assumes *subH*: *subgroup H G*
    and *subK*: *subgroup K G*
  shows *subgroup* (H <#> K) *G*
apply (*rule subgroup.intro*)
  apply (*intro set-mult-closed subgroup.subset*[*OF subH*] *subgroup.subset*[*OF subK*])
  apply (*simp add*: *set-mult-def*) **apply** *clarsimp* **defer** *1*

apply (*simp add*: *set-mult-def*) **defer** *1*
apply (*simp add*: *set-mult-def*, *clarsimp*) **defer** *1*
proof −
fix *ha hb ka kb*
assume *haH*: $ha \in H$ **and** *hbH*: $hb \in H$ **and** *kaK*: $ka \in K$ **and** *kbK*: $kb \in K$
note *carr* = *haH*[*THEN subgroup.mem-carrier*[*OF subH*]] *hbH*[*THEN subgroup.mem-carrier*[*OF subH*]]
kaK[*THEN subgroup.mem-carrier*[*OF subK*]] *kbK*[*THEN subgroup.mem-carrier*[*OF subK*]]
from *carr*
have $(ha \otimes ka) \otimes (hb \otimes kb) = ha \otimes (ka \otimes hb) \otimes kb$ **by** (*simp add*: *m-assoc*)
also from *carr*
have $\ldots = ha \otimes (hb \otimes ka) \otimes kb$ **by** (*simp add*: *m-comm*)
also from *carr*
have $\ldots = (ha \otimes hb) \otimes (ka \otimes kb)$ **by** (*simp add*: *m-assoc*)
finally
have *eq*: $(ha \otimes ka) \otimes (hb \otimes kb) = (ha \otimes hb) \otimes (ka \otimes kb)$ **.**

from *haH hbH* **have** *hH*: $ha \otimes hb \in H$ **by** (*simp add*: *subgroup.m-closed*[*OF subH*])
from *kaK kbK* **have** *kK*: $ka \otimes kb \in K$ **by** (*simp add*: *subgroup.m-closed*[*OF subK*])

from *hH* **and** *kK* **and** *eq*
show $\exists h'{\in}H.\ \exists k'{\in}K.\ (ha \otimes ka) \otimes (hb \otimes kb) = h' \otimes k'$ **by** *fast*
next
have $\mathbf{1} = \mathbf{1} \otimes \mathbf{1}$ **by** *simp*
from *subgroup.one-closed*[*OF subH*] *subgroup.one-closed*[*OF subK*] *this*
show $\exists h{\in}H.\ \exists k{\in}K.\ \mathbf{1} = h \otimes k$ **by** *fast*
next
fix *h k*
assume *hH*: $h \in H$
and *kK*: $k \in K$

from *hH*[*THEN subgroup.mem-carrier*[*OF subH*]] *kK*[*THEN subgroup.mem-carrier*[*OF subK*]]
have $inv\ (h \otimes k) = inv\ h \otimes inv\ k$ **by** (*simp add*: *inv-mult-group m-comm*)

from *subgroup.m-inv-closed*[*OF subH hH*] **and** *subgroup.m-inv-closed*[*OF subK kK*] **and** *this*
show $\exists ha{\in}H.\ \exists ka{\in}K.\ inv\ (h \otimes k) = ha \otimes ka$ **by** *fast*
qed

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

from *xixH*
  have $\exists\, h{\in}H.\ (inv\ x) \otimes x' = h$ **by** *fast*
from *this*
  obtain *h*
    where *hH*: $h \in H$
    and *hsym*: $(inv\ x) \otimes x' = h$
  by *fast*

from *hH subset* **have** *hcarr*: $h \in carrier\ G$ **by** *simp*
note *carr = carr hcarr*
from *hsym*[*symmetric*] **have** $x \otimes h = x \otimes ((inv\ x) \otimes x')$ **by** *fast*
also from *carr*
  have $\ldots = (x \otimes (inv\ x)) \otimes x'$ **by** (*simp add*: *m-assoc*[*symmetric*])
also from *carr*
  have $\ldots = \mathbf{1} \otimes x'$ **by** *simp*
also from *carr*
  have $\ldots = x'$ **by** *simp*
finally
  have $x \otimes h = x'$ **by** *simp*

from *this*[*symmetric*] **and** *hH*
  show $x' \in x <\#\ H$
  unfolding *l-coset-def*
  by *fast*
qed

5.2 Normal subgroups

lemma *normal-imp-subgroup*: $H \lhd G \Longrightarrow subgroup\ H\ G$
  by (*simp add*: *normal-def subgroup-def*)

lemma (**in** *group*) *normalI*:
  $subgroup\ H\ G \Longrightarrow (\forall\, x \in carrier\ G.\ H \#{>}\ x = x <\#\ H) \Longrightarrow H \lhd G$
  by (*simp add*: *normal-def normal-axioms-def prems*)

lemma (**in** *normal*) *inv-op-closed1*:
  $[\![x \in carrier\ G;\ h \in H]\!] \Longrightarrow (inv\ x) \otimes h \otimes x \in H$
apply (*insert coset-eq*)
apply (*auto simp add*: *l-coset-def r-coset-def*)
apply (*drule bspec*, *assumption*)
apply (*drule equalityD1* [*THEN subsetD*], *blast*, *clarify*)
apply (*simp add*: *m-assoc*)
apply (*simp add*: *m-assoc* [*symmetric*])
done

lemma (**in** *normal*) *inv-op-closed2*:
  $[\![x \in carrier\ G;\ h \in H]\!] \Longrightarrow x \otimes h \otimes (inv\ x) \in H$
apply (*subgoal-tac inv* $(inv\ x) \otimes h \otimes (inv\ x) \in H$)
apply (*simp add*:)

```
apply (blast intro: inv-op-closed1)
done
```

Alternative characterization of normal subgroups

```
lemma (in group) normal-inv-iff:
   (N ◁ G) =
    (subgroup N G & (∀ x ∈ carrier G. ∀ h ∈ N. x ⊗ h ⊗ (inv x) ∈ N))
    (is - = ?rhs)
proof
  assume N: N ◁ G
  show ?rhs
    by (blast intro: N normal.inv-op-closed2 normal-imp-subgroup)
next
  assume ?rhs
  hence sg: subgroup N G
    and closed: ⋀x. x∈carrier G ⟹ ∀ h∈N. x ⊗ h ⊗ inv x ∈ N by auto
  hence sb: N ⊆ carrier G by (simp add: subgroup.subset)
  show N ◁ G
  proof (intro normalI [OF sg], simp add: l-coset-def r-coset-def, clarify)
    fix x
    assume x: x ∈ carrier G
    show (⋃h∈N. {h ⊗ x}) = (⋃h∈N. {x ⊗ h})
    proof
      show (⋃h∈N. {h ⊗ x}) ⊆ (⋃h∈N. {x ⊗ h})
      proof clarify
        fix n
        assume n: n ∈ N
        show n ⊗ x ∈ (⋃h∈N. {x ⊗ h})
        proof
          from closed [of inv x]
          show inv x ⊗ n ⊗ x ∈ N by (simp add: x n)
          show n ⊗ x ∈ {x ⊗ (inv x ⊗ n ⊗ x)}
            by (simp add: x n m-assoc [symmetric] sb [THEN subsetD])
        qed
      qed
    next
      show (⋃h∈N. {x ⊗ h}) ⊆ (⋃h∈N. {h ⊗ x})
      proof clarify
        fix n
        assume n: n ∈ N
        show x ⊗ n ∈ (⋃h∈N. {h ⊗ x})
        proof
          show x ⊗ n ⊗ inv x ∈ N by (simp add: x n closed)
          show x ⊗ n ∈ {x ⊗ n ⊗ inv x ⊗ x}
            by (simp add: x n m-assoc sb [THEN subsetD])
        qed
      qed
    qed
  qed
```

qed

5.3 More Properties of Cosets

lemma (**in** *group*) *lcos-m-assoc*:
  [| $M \subseteq$ *carrier* G; $g \in$ *carrier* G; $h \in$ *carrier* G |]
  ==> g <# (h <# M) = ($g \otimes h$) <# M
by (*force simp add*: *l-coset-def m-assoc*)

lemma (**in** *group*) *lcos-mult-one*: $M \subseteq$ *carrier* G ==> **1** <# M = M
by (*force simp add*: *l-coset-def*)

lemma (**in** *group*) *l-coset-subset-G*:
  [| $H \subseteq$ *carrier* G; $x \in$ *carrier* G |] ==> x <# $H \subseteq$ *carrier* G
by (*auto simp add*: *l-coset-def subsetD*)

lemma (**in** *group*) *l-coset-swap*:
  $[\![y \in x$ <# H; $x \in$ *carrier* G; *subgroup* H $G]\!] \Longrightarrow x \in y$ <# H
proof (*simp add*: *l-coset-def*)
 assume $\exists h \in H.\ y = x \otimes h$
  and x: $x \in$ *carrier* G
  and *sb*: *subgroup* H G
 then obtain h' **where** h': $h' \in H$ & $x \otimes h' = y$ **by** *blast*
 show $\exists h \in H.\ x = y \otimes h$
 proof
  show $x = y \otimes$ *inv* h' **using** h' x *sb*
   by (*auto simp add*: *m-assoc subgroup.subset* [*THEN subsetD*])
  show *inv* $h' \in H$ **using** h' *sb*
   by (*auto simp add*: *subgroup.subset* [*THEN subsetD*] *subgroup.m-inv-closed*)
 qed
qed

lemma (**in** *group*) *l-coset-carrier*:
  [| $y \in x$ <# H; $x \in$ *carrier* G; *subgroup* H G |] ==> $y \in$ *carrier* G
by (*auto simp add*: *l-coset-def m-assoc*
    subgroup.subset [*THEN subsetD*] *subgroup.m-closed*)

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 −
 from y
 obtain h' **where** $h' \in H$ $x \otimes h' = y$ **by** (*auto simp add*: *l-coset-def*)
 thus *?thesis* **using** x *sb*
  by (*auto simp add*: *l-coset-def m-assoc*
     subgroup.subset [*THEN subsetD*] *subgroup.m-closed*)
qed

lemma (**in** *group*) *l-repr-independence*:

assumes *y*: $y \in x$ <# H **and** *x*: $x \in$ *carrier* G **and** *sb*: *subgroup* H G
shows x <# H = y <# H
proof
show x <# $H \subseteq y$ <# H
by (*rule l-repr-imp-subset*,
(*blast intro*: *l-coset-swap l-coset-carrier y x sb*)+)
show y <# $H \subseteq x$ <# H **by** (*rule l-repr-imp-subset* [*OF y x sb*])
qed

lemma (**in** *group*) *setmult-subset-G*:
⟦$H \subseteq$ *carrier* G; $K \subseteq$ *carrier* G⟧ $\Longrightarrow$ H <#> $K \subseteq$ *carrier* G
by (*auto simp add*: *set-mult-def subsetD*)

lemma (**in** *group*) *subgroup-mult-id*: *subgroup* H G $\Longrightarrow$ H <#> $H = H$
apply (*auto simp add*: *subgroup.m-closed set-mult-def Sigma-def image-def*)
apply (*rule-tac* $x = x$ **in** *bexI*)
apply (*rule bexI* [*of* - **1**])
apply (*auto simp add*: *subgroup.m-closed subgroup.one-closed*
r-one subgroup.subset [*THEN subsetD*])
done

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

lemma (**in** *normal*) *rcos-inv*:
assumes *x*: $x \in$ *carrier* G
shows *set-inv* (H #> x) = H #> (*inv* x)
proof (*simp add*: *r-coset-def SET-INV-def x inv-mult-group*, *safe*)
fix h
assume $h \in H$
show *inv* $x \otimes$ *inv* $h \in (\bigcup j{\in}H.\ \{j \otimes \mathit{inv}\ x\})$
proof
show *inv* $x \otimes$ *inv* $h \otimes x \in H$
by (*simp add*: *inv-op-closed1 prems*)
show *inv* $x \otimes$ *inv* $h \in \{\mathit{inv}\ x \otimes \mathit{inv}\ h \otimes x \otimes \mathit{inv}\ x\}$
by (*simp add*: *prems m-assoc*)
qed
next
fix h
assume $h \in H$
show $h \otimes$ *inv* $x \in (\bigcup j{\in}H.\ \{\mathit{inv}\ x \otimes \mathit{inv}\ j\})$
proof
show $x \otimes$ *inv* $h \otimes$ *inv* $x \in H$
by (*simp add*: *inv-op-closed2 prems*)
show $h \otimes$ *inv* $x \in \{\mathit{inv}\ x \otimes \mathit{inv}\ (x \otimes \mathit{inv}\ h \otimes \mathit{inv}\ x)\}$
by (*simp add*: *prems m-assoc* [*symmetric*] *inv-mult-group*)
qed
qed

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

lemma (**in** *group*) *setmult-rcos-assoc*:
$[\![H \subseteq carrier\ G;\ K \subseteq carrier\ G;\ x \in carrier\ G]\!]$
$\Longrightarrow H <\#> (K \ \#> x) = (H <\#> K) \ \#> x$
by (*force simp add*: *r-coset-def set-mult-def m-assoc*)

lemma (**in** *group*) *rcos-assoc-lcos*:
$[\![H \subseteq carrier\ G;\ K \subseteq carrier\ G;\ x \in carrier\ G]\!]$
$\Longrightarrow (H \ \#> x) <\#> K = H <\#> (x <\# \ K)$
by (*force simp add*: *r-coset-def l-coset-def set-mult-def m-assoc*)

lemma (**in** *normal*) *rcos-mult-step1*:
$[\![x \in carrier\ G;\ y \in carrier\ G]\!]$
$\Longrightarrow (H \ \#> x) <\#> (H \ \#> y) = (H <\#> (x <\# \ H)) \ \#> y$
by (*simp add*: *setmult-rcos-assoc subset*
r-coset-subset-G l-coset-subset-G rcos-assoc-lcos)

lemma (**in** *normal*) *rcos-mult-step2*:
$[\![x \in carrier\ G;\ y \in carrier\ G]\!]$
$\Longrightarrow (H <\#> (x <\# \ H)) \ \#> y = (H <\#> (H \ \#> x)) \ \#> y$
by (*insert coset-eq*, *simp add*: *normal-def*)

lemma (**in** *normal*) *rcos-mult-step3*:
$[\![x \in carrier\ G;\ y \in carrier\ G]\!]$
$\Longrightarrow (H <\#> (H \ \#> x)) \ \#> y = H \ \#> (x \otimes y)$
by (*simp add*: *setmult-rcos-assoc coset-mult-assoc*
subgroup-mult-id normal.axioms subset prems)

lemma (**in** *normal*) *rcos-sum*:
$[\![x \in carrier\ G;\ y \in carrier\ G]\!]$
$\Longrightarrow (H \ \#> x) <\#> (H \ \#> y) = H \ \#> (x \otimes y)$
by (*simp add*: *rcos-mult-step1 rcos-mult-step2 rcos-mult-step3*)

lemma (**in** *normal*) *rcosets-mult-eq*: $M \in rcosets\ H \Longrightarrow H <\#> M = M$
— generalizes *subgroup-mult-id*
by (*auto simp add*: *RCOSETS-def subset*
setmult-rcos-assoc subgroup-mult-id normal.axioms prems)

5.3.3 An Equivalence Relation

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

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

```
proof (intro equiv.intro)
  show refl (carrier G) (rcong H)
    by (auto simp add: r-congruent-def refl-def)
next
  show sym (rcong H)
  proof (simp add: r-congruent-def sym-def, clarify)
    fix x y
    assume [simp]: x ∈ carrier G y ∈ carrier G
       and inv x ⊗ y ∈ H
    hence inv (inv x ⊗ y) ∈ H by (simp add: m-inv-closed)
    thus inv y ⊗ x ∈ H by (simp add: inv-mult-group)
  qed
next
  show trans (rcong H)
  proof (simp add: r-congruent-def trans-def, clarify)
    fix x y z
    assume [simp]: x ∈ carrier G y ∈ carrier G z ∈ carrier G
       and inv x ⊗ y ∈ H and inv y ⊗ z ∈ H
    hence (inv x ⊗ y) ⊗ (inv y ⊗ z) ∈ H by simp
    hence inv x ⊗ (y ⊗ inv y) ⊗ z ∈ H by (simp add: m-assoc del: r-inv)
    thus inv x ⊗ z ∈ H by simp
  qed
qed
```

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

```
lemma (in subgroup) l-coset-eq-rcong:
  includes group G
  assumes a: a ∈ carrier G
  shows a <# H = rcong H `` {a}
by (force simp add: r-congruent-def l-coset-def m-assoc [symmetric] a )
```

5.3.4 Two Distinct Right Cosets are Disjoint

```
lemma (in group) rcos-equation:
  includes subgroup H G
  shows
    ⟦ha ⊗ a = h ⊗ b; a ∈ carrier G;  b ∈ carrier G;
      h ∈ H;  ha ∈ H;  hb ∈ H⟧
    ⟹ hb ⊗ a ∈ (⋃h∈H. {h ⊗ b})
apply (rule UN-I [of hb ⊗ ((inv ha) ⊗ h)])
apply (simp add: )
apply (simp add: m-assoc transpose-inv)
done

lemma (in group) rcos-disjoint:
  includes subgroup H G
  shows ⟦a ∈ rcosets H; b ∈ rcosets H; a≠b⟧ ⟹ a ∩ b = {}
apply (simp add: RCOSETS-def r-coset-def)
```

```
apply (blast intro: rcos-equation prems sym)
done
```

5.4 Further lemmas for *r-congruent*

The relation is a congruence

```
lemma (in normal) congruent-rcong:
  shows congruent2 (rcong H) (rcong H) (λa b. a ⊗ b <# H)
proof (intro congruent2I[of carrier G - carrier G -] equiv-rcong is-group)
  fix a b c
  assume abrcong: (a, b) ∈ rcong H
    and ccarr: c ∈ carrier G

  from abrcong
      have acarr: a ∈ carrier G
        and bcarr: b ∈ carrier G
        and abH: inv a ⊗ b ∈ H
      unfolding r-congruent-def
      by fast+

  note carr = acarr bcarr ccarr

  from ccarr and abH
      have inv c ⊗ (inv a ⊗ b) ⊗ c ∈ H by (rule inv-op-closed1)
  moreover
      from carr and inv-closed
      have inv c ⊗ (inv a ⊗ b) ⊗ c = (inv c ⊗ inv a) ⊗ (b ⊗ c)
      by (force cong: m-assoc)
  moreover
      from carr and inv-closed
      have ... = (inv (a ⊗ c)) ⊗ (b ⊗ c)
      by (simp add: inv-mult-group)
  ultimately
      have (inv (a ⊗ c)) ⊗ (b ⊗ c) ∈ H by simp
  from carr and this
     have (b ⊗ c) ∈ (a ⊗ c) <# H
     by (simp add: lcos-module-rev[OF is-group])
  from carr and this and is-subgroup
     show (a ⊗ c) <# H = (b ⊗ c) <# H by (intro l-repr-independence, simp+)
next
  fix a b c
  assume abrcong: (a, b) ∈ rcong H
    and ccarr: c ∈ carrier G

  from ccarr have c ∈ Units G by (simp add: Units-eq)
  hence cinvc-one: inv c ⊗ c = 1 by (rule Units-l-inv)

  from abrcong
      have acarr: a ∈ carrier G
```

and *bcarr*: $b \in$ *carrier G*
and *abH*: *inv* $a \otimes b \in H$
by (*unfold r-congruent-def*, *fast+*)

note *carr* = *acarr bcarr ccarr*

from *carr* **and** *inv-closed*
have *inv* $a \otimes b =$ *inv* $a \otimes (\mathbf{1} \otimes b)$ **by** *simp*
also from *carr* **and** *inv-closed*
have $\ldots =$ *inv* $a \otimes$ (*inv* $c \otimes c) \otimes b$ **by** *simp*
also from *carr* **and** *inv-closed*
have $\ldots =$ (*inv* $a \otimes$ *inv* $c) \otimes (c \otimes b)$ **by** (*force cong*: *m-assoc*)
also from *carr* **and** *inv-closed*
have $\ldots =$ *inv* $(c \otimes a) \otimes (c \otimes b)$ **by** (*simp add*: *inv-mult-group*)
finally
have *inv* $a \otimes b =$ *inv* $(c \otimes a) \otimes (c \otimes b)$ **.**
from *abH* **and** *this*
have *inv* $(c \otimes a) \otimes (c \otimes b) \in H$ **by** *simp*

from *carr* **and** *this*
have $(c \otimes b) \in (c \otimes a)$ <# H
by (*simp add*: *lcos-module-rev*[*OF is-group*])
from *carr* **and** *this* **and** *is-subgroup*
show $(c \otimes a)$ <# $H = (c \otimes b)$ <# H **by** (*intro l-repr-independence*, *simp+*)
qed

5.5 Order of a Group and Lagrange's Theorem

constdefs
order :: ('*a*, '*b*) *monoid-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
apply (*simp add*: *r-coset-def*)
apply (*rule-tac* x=**1** **in** *bexI*)
apply (*auto simp add*:)
done

lemma (**in** *group*) *rcosets-part-G*:
includes *subgroup*
shows $\bigcup$(*rcosets H*) = *carrier G*
apply (*rule equalityI*)
apply (*force simp add*: *RCOSETS-def r-coset-def*)
apply (*auto simp add*: *RCOSETS-def intro*: *rcos-self prems*)
done

lemma (**in** *group*) *cosets-finite*:

$[\![c \in \mathit{rcosets}\ H;\ \ H \subseteq \mathit{carrier}\ G;\ \ \mathit{finite}\ (\mathit{carrier}\ G)]\!] \Longrightarrow \mathit{finite}\ c$
apply (*auto simp add*: *RCOSETS-def*)
apply (*simp add*: *r-coset-subset-G* [*THEN finite-subset*])
done

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

lemma (**in** *group*) *inj-on-f*:
$[\![H \subseteq \mathit{carrier}\ G;\ \ a \in \mathit{carrier}\ G]\!] \Longrightarrow \mathit{inj\text{-}on}\ (\lambda y.\ y \otimes \mathit{inv}\ a)\ (H\ \#{>}\ a)$
apply (*rule inj-onI*)
apply (*subgoal-tac* $x \in \mathit{carrier}\ G\ \&\ y \in \mathit{carrier}\ G$)
prefer *2* **apply** (*blast intro*: *r-coset-subset-G* [*THEN subsetD*])
apply (*simp add*: *subsetD*)
done

lemma (**in** *group*) *inj-on-g*:
$[\![H \subseteq \mathit{carrier}\ G;\ \ a \in \mathit{carrier}\ G]\!] \Longrightarrow \mathit{inj\text{-}on}\ (\lambda y.\ y \otimes a)\ H$
by (*force simp add*: *inj-on-def subsetD*)

lemma (**in** *group*) *card-cosets-equal*:
$[\![c \in \mathit{rcosets}\ H;\ \ H \subseteq \mathit{carrier}\ G;\ \mathit{finite}(\mathit{carrier}\ G)]\!]$
$\Longrightarrow \mathit{card}\ c = \mathit{card}\ H$
apply (*auto simp add*: *RCOSETS-def*)
apply (*rule card-bij-eq*)
apply (*rule inj-on-f*, *assumption+*)
apply (*force simp add*: *m-assoc subsetD r-coset-def*)
apply (*rule inj-on-g*, *assumption+*)
apply (*force simp add*: *m-assoc subsetD r-coset-def*)

The sets $H\ \#{>}\ a$ and H are finite.

apply (*simp add*: *r-coset-subset-G* [*THEN finite-subset*])
apply (*blast intro*: *finite-subset*)
done

lemma (**in** *group*) *rcosets-subset-PowG*:
$\mathit{subgroup}\ H\ G \ \Longrightarrow \mathit{rcosets}\ H \subseteq \mathit{Pow}(\mathit{carrier}\ G)$
apply (*simp add*: *RCOSETS-def*)
apply (*blast dest*: *r-coset-subset-G subgroup.subset*)
done

theorem (**in** *group*) *lagrange*:
$[\![\mathit{finite}(\mathit{carrier}\ G);\ \mathit{subgroup}\ H\ G]\!]$
$\Longrightarrow \mathit{card}(\mathit{rcosets}\ H) * \mathit{card}(H) = \mathit{order}(G)$
apply (*simp* (*no-asm-simp*) *add*: *order-def rcosets-part-G* [*symmetric*])
apply (*subst mult-commute*)
apply (*rule card-partition*)
apply (*simp add*: *rcosets-subset-PowG* [*THEN finite-subset*])
apply (*simp add*: *rcosets-part-G*)
apply (*simp add*: *card-cosets-equal subgroup.subset*)

apply (*simp add*: *rcos-disjoint*)
done

5.6 Quotient Groups: Factorization of a Group

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

lemma (**in** *normal*) *setmult-closed*:
⟦*K1* ∈ *rcosets H*; *K2* ∈ *rcosets H*⟧ ⟹ *K1* <#> *K2* ∈ *rcosets H*
by (*auto simp add*: *rcos-sum RCOSETS-def*)

lemma (**in** *normal*) *setinv-closed*:
K ∈ *rcosets H* ⟹ *set-inv K* ∈ *rcosets H*
by (*auto simp add*: *rcos-inv RCOSETS-def*)

lemma (**in** *normal*) *rcosets-assoc*:
⟦*M1* ∈ *rcosets H*; *M2* ∈ *rcosets H*; *M3* ∈ *rcosets H*⟧
⟹ *M1* <#> *M2* <#> *M3* = *M1* <#> (*M2* <#> *M3*)
by (*auto simp add*: *RCOSETS-def rcos-sum m-assoc*)

lemma (**in** *subgroup*) *subgroup-in-rcosets*:
includes *group G*
shows *H* ∈ *rcosets H*
proof −
from - ⟨*subgroup H G*⟩ **have** *H* #> **1** = *H*
by (*rule coset-join2*) *auto*
then show *?thesis*
by (*auto simp add*: *RCOSETS-def*)
qed

lemma (**in** *normal*) *rcosets-inv-mult-group-eq*:
M ∈ *rcosets H* ⟹ *set-inv M* <#> *M* = *H*
by (*auto simp add*: *RCOSETS-def rcos-inv rcos-sum subgroup.subset normal.axioms prems*)

theorem (**in** *normal*) *factorgroup-is-group*:
group (*G Mod H*)
apply (*simp add*: *FactGroup-def*)
apply (*rule groupI*)
apply (*simp add*: *setmult-closed*)
apply (*simp add*: *normal-imp-subgroup subgroup-in-rcosets* [*OF is-group*])
apply (*simp add*: *restrictI setmult-closed rcosets-assoc*)
apply (*simp add*: *normal-imp-subgroup*
subgroup-in-rcosets rcosets-mult-eq)

apply (*auto dest*: *rcosets-inv-mult-group-eq simp add*: *setinv-closed*)
done

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

lemma (**in** *normal*) *inv-FactGroup*:
$X \in carrier\ (G\ Mod\ H) \Longrightarrow inv_{G\ Mod\ H}\ X = set\text{-}inv\ X$
apply (*rule group.inv-equality* [*OF factorgroup-is-group*])
apply (*simp-all add*: *FactGroup-def setinv-closed rcosets-inv-mult-group-eq*)
done

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

lemma (**in** *normal*) *r-coset-hom-Mod*:
$(\lambda a.\ H\ \#>\ a) \in hom\ G\ (G\ Mod\ H)$
by (*auto simp add*: *FactGroup-def RCOSETS-def Pi-def hom-def rcos-sum*)

5.7 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-scheme* $\Rightarrow$ $('b, 'n)$ *monoid-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*
apply (*rule subgroup.intro*)
apply (*auto simp add*: *kernel-def group.intro prems*)
done

The kernel of a homomorphism is a normal subgroup

lemma (**in** *group-hom*) *normal-kernel*: $(kernel\ G\ H\ h) \lhd G$
apply (*simp add*: *G.normal-inv-iff subgroup-kernel*)
apply (*simp add*: *kernel-def*)
done

lemma (**in** *group-hom*) *FactGroup-nonempty*:
assumes X: $X \in carrier\ (G\ Mod\ kernel\ G\ H\ h)$
shows $X \neq \{\}$
proof −
from *X*
obtain *g* **where** $g \in carrier\ G$
and $X = kernel\ G\ H\ h\ \#>\ g$
by (*auto simp add*: *FactGroup-def RCOSETS-def*)
thus *?thesis*
by (*auto simp add*: *kernel-def r-coset-def image-def intro*: *hom-one*)

qed

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 −
  from X
  obtain g **where** g: $g \in$ *carrier G*
      and X = *kernel G H h* #> g
    by (*auto simp add*: *FactGroup-def RCOSETS-def*)
  hence h ‘ $X = \{h\ g\}$ **by** (*auto simp add*: *kernel-def r-coset-def image-def g*)
  thus *?thesis* **by** (*auto simp add*: *g*)
qed

lemma (**in** *group-hom*) *FactGroup-hom*:
  (λX. *contents* (h‘X)) $\in$ *hom* (*G Mod* (*kernel G H h*)) *H*
apply (*simp add*: *hom-def FactGroup-contents-mem normal.factorgroup-is-group* [*OF normal-kernel*] *group.axioms monoid.m-closed*)
proof (*simp add*: *hom-def funcsetI FactGroup-contents-mem*, *intro ballI*)
  fix X **and** X'
  assume X: $X \in$ *carrier* (*G Mod kernel G H h*)
    and X': $X' \in$ *carrier* (*G Mod kernel G H h*)
  then
  obtain g **and** g'
      where $g \in$ *carrier G* **and** $g' \in$ *carrier G*
      and X = *kernel G H h* #> g **and** X' = *kernel G H h* #> g'
    by (*auto simp add*: *FactGroup-def RCOSETS-def*)
  hence *all*: $\forall x \in X.\ h\ x = h\ g\ \forall x \in X'.\ h\ x = h\ g'$
    and *Xsub*: $X \subseteq$ *carrier G* **and** X'*sub*: $X' \subseteq$ *carrier G*
    by (*force simp add*: *kernel-def r-coset-def image-def*)+
  hence h ‘ (X <#> X') = $\{h\ g \otimes_H h\ g'\}$ **using** $X\ X'$
    by (*auto dest*!: *FactGroup-nonempty*
        simp add: *set-mult-def image-eq-UN*
            subsetD [*OF Xsub*] *subsetD* [*OF* X'*sub*])
  thus *contents* (h ‘ (X <#> X')) = *contents* (h ‘ X) $\otimes_H$ *contents* (h ‘ X')
    by (*simp add*: *all image-eq-UN FactGroup-nonempty* $X\ X'$)
qed

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'
apply (*clarsimp simp add*: *kernel-def r-coset-def image-def*)
apply (*rename-tac y*)
apply (*rule-tac* $x = y \otimes g \otimes$ *inv* g' **in** *exI*)
apply (*simp add*: *G.m-assoc*)
done

lemma (**in** *group-hom*) *FactGroup-inj-on*:
  inj-on (λX. *contents* (h ‘ X)) (*carrier* (*G Mod kernel G H h*))
proof (*simp add*: *inj-on-def*, *clarify*)
  fix X **and** X'
  assume X: $X \in$ *carrier* (*G Mod kernel G H h*)
   and X': $X' \in$ *carrier* (*G Mod kernel G H h*)
  then
  obtain g **and** g'
     where gX: $g \in$ *carrier G* $g' \in$ *carrier G*
      X = *kernel G H h* #> g X' = *kernel G H h* #> g'
   by (*auto simp add*: *FactGroup-def RCOSETS-def*)
  hence *all*: $\forall x \in X.\ h\ x = h\ g$ $\forall x \in X'.\ h\ x = h\ g'$
   by (*force simp add*: *kernel-def r-coset-def image-def*)+
  assume *contents* (h ‘ X) = *contents* (h ‘ X')
  hence h: $h\ g = h\ g'$
   by (*simp add*: *image-eq-UN all FactGroup-nonempty X X'*)
  show $X=X'$ **by** (*rule equalityI*) (*simp-all add*: *FactGroup-subset h gX*)
qed

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 (λX. *contents* (h ‘ X)) ‘ *carrier* (*G Mod kernel G H h*) = *carrier H*
proof
  show (λX. *contents* (h ‘ X)) ‘ *carrier* (*G Mod kernel G H h*) $\subseteq$ *carrier H*
   by (*auto simp add*: *FactGroup-contents-mem*)
  show *carrier H* $\subseteq$ (λX. *contents* (h ‘ X)) ‘ *carrier* (*G Mod kernel G H h*)
  proof
   fix y
   assume y: $y \in$ *carrier H*
   with h **obtain** g **where** g: $g \in$ *carrier G* $h\ g = y$
    by (*blast elim*: *equalityE*)
   hence ($\bigcup x\in$*kernel G H h* #> g. $\{h\ x\}$) = $\{y\}$
    by (*auto simp add*: *y kernel-def r-coset-def*)
   with g **show** $y \in$ (λX. *contents* (h ‘ X)) ‘ *carrier* (*G Mod kernel G H h*)
    by (*auto intro*!: *bexI simp add*: *FactGroup-def RCOSETS-def image-eq-UN*)
  qed
qed

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*
  $\Longrightarrow$ (λX. *contents* (h‘X)) $\in$ (*G Mod* (*kernel G H h*)) $\cong$ H
by (*simp add*: *iso-def FactGroup-hom FactGroup-inj-on bij-betw-def*
      FactGroup-onto)

end

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

6 Sylow's Theorem

See also [3].

The combinatorial argument is in theory Exponent

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

lemma (in sylow) RelM-refl: refl calM RelM
apply (auto simp add: refl-def RelM-def calM-def)
apply (blast intro!: coset-mult-one [symmetric])
done

lemma (in sylow) RelM-sym: sym RelM
proof (unfold sym-def RelM-def, clarify)
  fix y g
  assume   y ∈ calM
    and g: g ∈ carrier G
  hence y = y #> g #> (inv g) by (simp add: coset-mult-assoc calM-def)
  thus ∃ g'∈carrier G. y = y #> g #> g'
   by (blast intro: g inv-closed)
qed

lemma (in sylow) RelM-trans: trans RelM
by (auto simp add: trans-def RelM-def calM-def coset-mult-assoc)

lemma (in sylow) RelM-equiv: equiv calM RelM
apply (unfold equiv-def)
apply (blast intro: RelM-refl RelM-sym RelM-trans)
done

lemma (in sylow) M-subset-calM-prep: M' ∈ calM // RelM  ==> M' ⊆ calM
apply (unfold RelM-def)
apply (blast elim!: quotientE)
done
```

6.1 Main Part of the Proof

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

lemma (in sylow-central) M-subset-calM: M ⊆ calM
by (rule M-in-quot [THEN M-subset-calM-prep])

lemma (in sylow-central) card-M1: card(M1) = p^a
apply (cut-tac M-subset-calM M1-in-M)
apply (simp add: calM-def, blast)
done

lemma card-nonempty: 0 < card(S) ==> S ≠ {}
by force

lemma (in sylow-central) exists-x-in-M1: ∃x. x∈M1
apply (subgoal-tac 0 < card M1)
 apply (blast dest: card-nonempty)
apply (cut-tac prime-p [THEN prime-imp-one-less])
apply (simp (no-asm-simp) add: card-M1)
done

lemma (in sylow-central) M1-subset-G [simp]: M1 ⊆ carrier G
apply (rule subsetD [THEN PowD])
apply (rule-tac [2] M1-in-M)
apply (rule M-subset-calM [THEN subset-trans])
apply (auto simp add: calM-def)
done

lemma (in sylow-central) M1-inj-H: ∃f ∈ H→M1. inj-on f H
  proof -
    from exists-x-in-M1 obtain m1 where m1M: m1 ∈ M1..
   have m1G: m1 ∈ carrier G by (simp add: m1M M1-subset-G [THEN subsetD])
    show ?thesis
    proof
      show inj-on (λz∈H. m1 ⊗ z) H
        by (simp add: inj-on-def l-cancel [of m1 x y, THEN iffD1] H-def m1G)
      show restrict (op ⊗ m1) H ∈ H → M1
      proof (rule restrictI)
        fix z assume zH: z ∈ H
        show m1 ⊗ z ∈ M1
        proof -
          from zH
          have zG: z ∈ carrier G and M1zeq: M1 #> z = M1
            by (auto simp add: H-def)
```

show *?thesis*
by (*rule subst* [*OF M1zeq*], *simp add*: *m1M zG rcosI*)
qed
qed
qed
qed

6.2 Discharging the Assumptions of *sylow-central*

lemma (**in** *sylow*) *EmptyNotInEquivSet*: $\{\} \notin$ *calM // RelM*
by (*blast elim*!: *quotientE dest*: *RelM-equiv* [*THEN equiv-class-self*])

lemma (**in** *sylow*) *existsM1inM*: $M \in$ *calM // RelM* ==> $\exists$ *M1*. *M1* $\in$ *M*
apply (*subgoal-tac M* $\neq$ $\{\}$)
apply *blast*
apply (*cut-tac EmptyNotInEquivSet*, *blast*)
done

lemma (**in** *sylow*) *zero-less-o-G*: $0 <$ *order*(*G*)
apply (*unfold order-def*)
apply (*blast intro*: *one-closed zero-less-card-empty*)
done

lemma (**in** *sylow*) *zero-less-m*: $m > 0$
apply (*cut-tac zero-less-o-G*)
apply (*simp add*: *order-G*)
done

lemma (**in** *sylow*) *card-calM*: *card*(*calM*) = (*p* ^*a*) * *m choose p* ^*a*
by (*simp add*: *calM-def n-subsets order-G* [*symmetric*] *order-def*)

lemma (**in** *sylow*) *zero-less-card-calM*: *card calM* > 0
by (*simp add*: *card-calM zero-less-binomial le-extend-mult zero-less-m*)

lemma (**in** *sylow*) *max-p-div-calM*:
~ (*p* ^ *Suc*(*exponent p m*) *dvd card*(*calM*))
apply (*subgoal-tac exponent p m* = *exponent p* (*card calM*))
apply (*cut-tac zero-less-card-calM prime-p*)
apply (*force dest*: *power-Suc-exponent-Not-dvd*)
apply (*simp add*: *card-calM zero-less-m* [*THEN const-p-fac*])
done

lemma (**in** *sylow*) *finite-calM*: *finite calM*
apply (*unfold calM-def*)
apply (*rule-tac B* = *Pow* (*carrier G*) **in** *finite-subset*)
apply *auto*
done

lemma (**in** *sylow*) *lemma-A1*:

$\exists\, M \in calM$ // $RelM$. ~ (p ^ Suc($exponent\ p\ m$) dvd $card$(M))
apply (*rule max-p-div-calM* [*THEN contrapos-np*])
apply (*simp add*: *finite-calM equiv-imp-dvd-card* [*OF* - *RelM-equiv*])
done

6.2.1 Introduction and Destruct Rules for *H*

lemma (**in** *sylow-central*) *H-I*: [|$g \in$ *carrier G*; *M1* #> g = *M1*|] ==> $g \in H$
by (*simp add*: *H-def*)

lemma (**in** *sylow-central*) *H-into-carrier-G*: $x \in H$ ==> $x \in$ *carrier G*
by (*simp add*: *H-def*)

lemma (**in** *sylow-central*) *in-H-imp-eq*: g : H ==> *M1* #> g = *M1*
by (*simp add*: *H-def*)

lemma (**in** *sylow-central*) *H-m-closed*: [| $x{\in}H$; $y{\in}H$|] ==> $x \otimes y \in H$
apply (*unfold H-def*)
apply (*simp add*: *coset-mult-assoc* [*symmetric*] *m-closed*)
done

lemma (**in** *sylow-central*) *H-not-empty*: $H \neq \{\}$
apply (*simp add*: *H-def*)
apply (*rule exI* [*of* - **1**], *simp*)
done

lemma (**in** *sylow-central*) *H-is-subgroup*: *subgroup H G*
apply (*rule subgroupI*)
apply (*rule subsetI*)
apply (*erule H-into-carrier-G*)
apply (*rule H-not-empty*)
apply (*simp add*: *H-def*, *clarify*)
apply (*erule-tac P* = %*z*. ?*lhs*(*z*) = *M1* **in** *subst*)
apply (*simp add*: *coset-mult-assoc*)
apply (*blast intro*: *H-m-closed*)
done

lemma (**in** *sylow-central*) *rcosetGM1g-subset-G*:
[| $g \in$ *carrier G*; $x \in$ *M1* #> g |] ==> $x \in$ *carrier G*
by (*blast intro*: *M1-subset-G* [*THEN r-coset-subset-G*, *THEN subsetD*])

lemma (**in** *sylow-central*) *finite-M1*: *finite M1*
by (*rule finite-subset* [*OF M1-subset-G finite-G*])

lemma (**in** *sylow-central*) *finite-rcosetGM1g*: $g{\in}$*carrier G* ==> *finite* (*M1* #> g)
apply (*rule finite-subset*)
apply (*rule subsetI*)
apply (*erule rcosetGM1g-subset-G*, *assumption*)

apply (*rule finite-G*)
done

lemma (**in** *sylow-central*) *M1-cardeq-rcosetGM1g*:
$g \in carrier\ G ==> card(M1\ \#> g) = card(M1)$
by (*simp* (*no-asm-simp*) *add*: *M1-subset-G card-cosets-equal rcosetsI*)

lemma (**in** *sylow-central*) *M1-RelM-rcosetGM1g*:
$g \in carrier\ G ==> (M1,\ M1\ \#> g) \in RelM$
apply (*simp* (*no-asm*) *add*: *RelM-def calM-def card-M1 M1-subset-G*)
apply (*rule conjI*)
apply (*blast intro*: *rcosetGM1g-subset-G*)
apply (*simp* (*no-asm-simp*) *add*: *card-M1 M1-cardeq-rcosetGM1g*)
apply (*rule bexI* [*of - inv g*])
apply (*simp-all add*: *coset-mult-assoc M1-subset-G*)
done

6.3 Equal Cardinalities of M and the Set of Cosets

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

lemma *ElemClassEquiv*:
$[|\ equiv\ A\ r;\ C \in A\ //\ r\ |] ==> \forall x \in C.\ \forall y \in C.\ (x,y) \in r$
by (*unfold equiv-def quotient-def sym-def trans-def*, *blast*)

lemma (**in** *sylow-central*) *M-elem-map*:
$M2 \in M ==> \exists g.\ g \in carrier\ G\ \&\ M1\ \#> g = M2$
apply (*cut-tac M1-in-M M-in-quot* [*THEN RelM-equiv* [*THEN ElemClassEquiv*]])
apply (*simp add*: *RelM-def*)
apply (*blast dest*!: *bspec*)
done

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

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

lemma (**in** *sylow-central*) *M-funcset-rcosets-H*:
$(\%x{:}M.\ H\ \#> (SOME\ g.\ g \in carrier\ G\ \&\ M1\ \#> g = x)) \in M \to rcosets\ H$
apply (*rule rcosetsI* [*THEN restrictI*])
apply (*rule H-is-subgroup* [*THEN subgroup.subset*])
apply (*erule M-elem-map-carrier*)
done

lemma (**in** *sylow-central*) *inj-M-GmodH*: $\exists f \in M \to rcosets\ H.\ inj\text{-}on\ f\ M$
apply (*rule bexI*)
apply (*rule-tac* [*2*] *M-funcset-rcosets-H*)
apply (*rule inj-onI*, *simp*)
apply (*rule trans* [*OF - M-elem-map-eq*])

prefer *2* **apply** *assumption*
apply (*rule M-elem-map-eq* [*symmetric*, *THEN trans*], *assumption*)
apply (*rule coset-mult-inv1*)
apply (*erule-tac* [*2*] *M-elem-map-carrier*)+
apply (*rule-tac* [*2*] *M1-subset-G*)
apply (*rule coset-join1* [*THEN in-H-imp-eq*])
apply (*rule-tac* [*3*] *H-is-subgroup*)
prefer *2* **apply** (*blast intro*: *m-closed M-elem-map-carrier inv-closed*)
apply (*simp add*: *coset-mult-inv2 H-def M-elem-map-carrier subset-def*)
done

6.3.1 The Opposite Injection

lemma (**in** *sylow-central*) *H-elem-map*:
  H1 ∈ *rcosets H* ==> ∃ *g*. *g* ∈ *carrier G* & *H* #> *g* = *H1*
by (*auto simp add*: *RCOSETS-def*)

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* ∈ *A*//*r*; *M1*∈*M*; (*M1*,*M2*) ∈ *r* |] ==> *M2* ∈ *M*
by (*unfold equiv-def quotient-def sym-def trans-def*, *blast*)

lemma (**in** *sylow-central*) *rcosets-H-funcset-M*:
 (λ*C* ∈ *rcosets H*. *M1* #> (@*g*. *g* ∈ *carrier G* ∧ *H* #> *g* = *C*)) ∈ *rcosets H* →
M
apply (*simp add*: *RCOSETS-def*)
apply (*fast intro*: *someI2*
      intro!: *restrictI M1-in-M*
       EquivElemClass [*OF RelM-equiv M-in-quot* - *M1-RelM-rcosetGM1g*])
done

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

lemma (**in** *sylow-central*) *inj-GmodH-M*:
  ∃ *g* ∈ *rcosets H*→*M*. *inj-on g* (*rcosets H*)
apply (*rule bexI*)
apply (*rule-tac* [*2*] *rcosets-H-funcset-M*)
apply (*rule inj-onI*)
apply (*simp*)
apply (*rule trans* [*OF* - *H-elem-map-eq*])
prefer *2* **apply** *assumption*
apply (*rule H-elem-map-eq* [*symmetric*, *THEN trans*], *assumption*)
apply (*rule coset-mult-inv1*)

apply (*erule-tac* [*2*] *H-elem-map-carrier*)+
apply (*rule-tac* [*2*] *H-is-subgroup* [*THEN subgroup.subset*])
apply (*rule coset-join2*)
apply (*blast intro*: *m-closed inv-closed H-elem-map-carrier*)
apply (*rule H-is-subgroup*)
apply (*simp add*: *H-I coset-mult-inv2 M1-subset-G H-elem-map-carrier*)
done

lemma (**in** *sylow-central*) *calM-subset-PowG*: *calM* $\subseteq$ *Pow*(*carrier G*)
by (*auto simp add*: *calM-def*)

lemma (**in** *sylow-central*) *finite-M*: *finite M*
apply (*rule finite-subset*)
apply (*rule M-subset-calM* [*THEN subset-trans*])
apply (*rule calM-subset-PowG*, *blast*)
done

lemma (**in** *sylow-central*) *cardMeqIndexH*: *card*(*M*) = *card*(*rcosets H*)
apply (*insert inj-M-GmodH inj-GmodH-M*)
apply (*blast intro*: *card-bij finite-M H-is-subgroup*
rcosets-subset-PowG [*THEN finite-subset*]
finite-Pow-iff [*THEN iffD2*])
done

lemma (**in** *sylow-central*) *index-lem*: *card*(*M*) $*$ *card*(*H*) = *order*(*G*)
by (*simp add*: *cardMeqIndexH lagrange H-is-subgroup*)

lemma (**in** *sylow-central*) *lemma-leq1*: *p*ˆ*a* $\leq$ *card*(*H*)
apply (*rule dvd-imp-le*)
apply (*rule div-combine* [*OF prime-p not-dvd-M*])
prefer *2* **apply** (*blast intro*: *subgroup.finite-imp-card-positive H-is-subgroup*)
apply (*simp add*: *index-lem order-G power-add mult-dvd-mono power-exponent-dvd*
zero-less-m)
done

lemma (**in** *sylow-central*) *lemma-leq2*: *card*(*H*) $\leq$ *p*ˆ*a*
apply (*subst card-M1* [*symmetric*])
apply (*cut-tac M1-inj-H*)
apply (*blast intro*!: *M1-subset-G intro*:
card-inj H-into-carrier-G finite-subset [*OF - finite-G*])
done

lemma (**in** *sylow-central*) *card-H-eq*: *card*(*H*) = *p*ˆ*a*
by (*blast intro*: *le-anti-sym lemma-leq1 lemma-leq2*)

lemma (**in** *sylow*) *sylow-thm*: $\exists$ *H*. *subgroup H G* & *card*(*H*) = *p*ˆ*a*
apply (*cut-tac lemma-A1*, *clarify*)
apply (*frule existsM1inM*, *clarify*)

apply (*subgoal-tac sylow-central G p a m M1 M*)
apply (*blast dest*: *sylow-central.H-is-subgroup sylow-central.card-H-eq*)
apply (*simp add*: *sylow-central-def sylow-central-axioms-def prems*)
done

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-axioms G p a m*)
by (*simp add*: *sylow-def group-def*)

6.4 Sylow's Theorem

theorem *sylow-thm*:
  [| *prime p*; *group*(*G*); *order*(*G*) = (*p* ^ *a*) ∗ *m*; *finite* (*carrier G*)|]
  ==> ∃ *H*. *subgroup H G* & *card*(*H*) = *p* ^ *a*
apply (*rule sylow.sylow-thm* [*of G p a m*])
apply (*simp add*: *sylow-eq sylow-axioms-def*)
done

end

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

7 Bijections of a Set, Permutation Groups and Automorphism Groups

constdefs
 Bij :: *'a set* ⇒ (*'a* ⇒ *'a*) *set*
  — Only extensional functions, since otherwise we get too many.
 Bij S ≡ *extensional S* ∩ {*f*. *bij-betw f S S*}

 BijGroup :: *'a set* ⇒ (*'a* ⇒ *'a*) *monoid*
 BijGroup S ≡
  (|*carrier* = *Bij S*,
   mult = λ*g* ∈ *Bij S*. λ*f* ∈ *Bij S*. *compose S g f*,
   one = λ*x* ∈ *S*. *x*|)

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

lemma *Bij-imp-extensional*: *f* ∈ *Bij S* ⟹ *f* ∈ *extensional S*
 by (*simp add*: *Bij-def*)

lemma *Bij-imp-funcset*: *f* ∈ *Bij S* ⟹ *f* ∈ *S* → *S*
 by (*auto simp add*: *Bij-def bij-betw-imp-funcset*)

7.1 Bijections Form a Group

lemma *restrict-Inv-Bij*: $f \in Bij\ S \Longrightarrow (\lambda x \in S.\ (Inv\ S\ f)\ x) \in Bij\ S$
  by (*simp add*: *Bij-def bij-betw-Inv*)

lemma *id-Bij*: $(\lambda x{\in}S.\ x) \in Bij\ S$
  by (*auto simp add*: *Bij-def bij-betw-def inj-on-def*)

lemma *compose-Bij*: $[\![x \in Bij\ S;\ y \in Bij\ S]\!] \Longrightarrow compose\ S\ x\ y \in Bij\ S$
  by (*auto simp add*: *Bij-def bij-betw-compose*)

lemma *Bij-compose-restrict-eq*:
    $f \in Bij\ S \Longrightarrow compose\ S\ (restrict\ (Inv\ S\ f)\ S)\ f = (\lambda x{\in}S.\ x)$
  by (*simp add*: *Bij-def compose-Inv-id*)

theorem *group-BijGroup*: *group* (*BijGroup S*)
apply (*simp add*: *BijGroup-def*)
apply (*rule groupI*)
    apply (*simp add*: *compose-Bij*)
   apply (*simp add*: *id-Bij*)
  apply (*simp add*: *compose-Bij*)
  apply (*blast intro*: *compose-assoc* [*symmetric*] *Bij-imp-funcset*)
 apply (*simp add*: *id-Bij Bij-imp-funcset Bij-imp-extensional*, *simp*)
apply (*blast intro*: *Bij-compose-restrict-eq restrict-Inv-Bij*)
done

7.2 Automorphisms Form a Group

lemma *Bij-Inv-mem*: $[\![\ f \in Bij\ S;\ \ x \in S]\!] \Longrightarrow Inv\ S\ f\ x \in S$
by (*simp add*: *Bij-def bij-betw-def Inv-mem*)

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 $[\![h \in Bij\ S;\ \ g \in S \rightarrow S \rightarrow S;\ \ x \in S;\ \ y \in S]\!]$
    $\Longrightarrow Inv\ S\ h\ (g\ x\ y) = g\ (Inv\ S\ h\ x)\ (Inv\ S\ h\ y)$
apply (*simp add*: *Bij-def bij-betw-def*)
apply (*subgoal-tac* $\exists\, x'{\in}S.\ \exists\, y'{\in}S.\ x = h\ x'\ \&\ y = h\ y'$, *clarify*)
 apply (*simp add*: *eq* [*symmetric*] *Inv-f-f funcset-mem* [*THEN funcset-mem*], *blast*)
done

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*: $(\lambda x \in carrier\ G.\ x) \in auto\ G$
  by (*simp add*: *auto-def hom-def restrictI group.axioms id-Bij*)

lemma (**in** *group*) *mult-funcset*: *mult* $G \in$ *carrier* $G \to$ *carrier* $G \to$ *carrier* G
  by (*simp add*: *Pi-I group.axioms*)

lemma (**in** *group*) *restrict-Inv-hom*:
  $[\![h \in hom\ G\ G; h \in Bij\ (carrier\ G)]\!]$
  $\Longrightarrow restrict\ (Inv\ (carrier\ G)\ h)\ (carrier\ G) \in hom\ G\ G$
  by (*simp add*: *hom-def Bij-Inv-mem restrictI mult-funcset*
      group.axioms Bij-Inv-lemma)

lemma *inv-BijGroup*:
  $f \in Bij\ S \Longrightarrow m\text{-}inv\ (BijGroup\ S)\ f = (\lambda x \in S.\ (Inv\ S\ f)\ x)$
apply (*rule group.inv-equality*)
apply (*rule group-BijGroup*)
apply (*simp-all add*: *BijGroup-def restrict-Inv-Bij Bij-compose-restrict-eq*)
done

lemma (**in** *group*) *subgroup-auto*:
  subgroup (*auto* G) (*BijGroup* (*carrier* G))
proof (*rule subgroup.intro*)
  show *auto* $G \subseteq$ *carrier* (*BijGroup* (*carrier* G))
    by (*force simp add*: *auto-def BijGroup-def*)
next
  fix $x\ y$
  assume $x \in auto\ G\ y \in auto\ G$
  thus $x \otimes_{BijGroup\ (carrier\ G)} y \in auto\ G$
    by (*force simp add*: *BijGroup-def is-group auto-def Bij-imp-funcset*
        group.hom-compose compose-Bij)
next
  show $\mathbf{1}_{BijGroup\ (carrier\ G)} \in auto\ G$ **by** (*simp add*: *BijGroup-def id-in-auto*)
next
  fix x
  assume $x \in auto\ G$
  thus $inv_{BijGroup\ (carrier\ G)}\ x \in auto\ G$
    by (*simp del*: *restrict-apply*
      add: *inv-BijGroup auto-def restrict-Inv-Bij restrict-Inv-hom*)
qed

theorem (**in** *group*) *AutoGroup*: *group* (*AutoGroup* G)
by (*simp add*: *AutoGroup-def subgroup.subgroup-is-group subgroup-auto*
    group-BijGroup)

end

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

8 Abelian Groups

record *'a ring* = *'a monoid* +
  zero :: *'a* (**0**ı)
  add :: [*'a*, *'a*] => *'a* (**infixl** ⊕ı *65*)

Derived operations.

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

  a-minus :: [(*'a*, *'m*) *ring-scheme*, *'a*, *'a*] => *'a* (**infixl** ⊖ı *65*)
  [| *x* ∈ *carrier R*; *y* ∈ *carrier R* |] ==> *x* ⊖ *y* == *x* ⊕ (⊖ *y*)

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

The following definition is redundant but simple to use.

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

8.1 Basic Properties

lemma *abelian-monoidI*:
  fixes *R* (**structure**)
  assumes *a-closed*:
    !!*x y*. [| *x* ∈ *carrier R*; *y* ∈ *carrier R* |] ==> *x* ⊕ *y* ∈ *carrier R*
   and *zero-closed*: **0** ∈ *carrier R*
   and *a-assoc*:
    !!*x y z*. [| *x* ∈ *carrier R*; *y* ∈ *carrier R*; *z* ∈ *carrier R* |] ==>
    (*x* ⊕ *y*) ⊕ *z* = *x* ⊕ (*y* ⊕ *z*)
   and *l-zero*: !!*x*. *x* ∈ *carrier R* ==> **0** ⊕ *x* = *x*
   and *a-comm*:
    !!*x y*. [| *x* ∈ *carrier R*; *y* ∈ *carrier R* |] ==> *x* ⊕ *y* = *y* ⊕ *x*
  shows *abelian-monoid R*
  by (*auto intro*!: *abelian-monoid.intro comm-monoidI intro*: *prems*)

lemma *abelian-groupI*:
  fixes *R* (**structure**)
  assumes *a-closed*:
    !!*x y*. [| *x* ∈ *carrier R*; *y* ∈ *carrier R* |] ==> *x* ⊕ *y* ∈ *carrier R*
   and *zero-closed*: *zero R* ∈ *carrier R*
   and *a-assoc*:
    !!*x y z*. [| *x* ∈ *carrier R*; *y* ∈ *carrier R*; *z* ∈ *carrier R* |] ==>
    (*x* ⊕ *y*) ⊕ *z* = *x* ⊕ (*y* ⊕ *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*
by (*auto intro*!: *abelian-group.intro abelian-monoidI*
abelian-group-axioms.intro comm-monoidI comm-groupI
intro: *prems*)

lemma (**in** *abelian-monoid*) *a-monoid*:
monoid (| *carrier* = *carrier G*, *mult* = *add G*, *one* = *zero G* |)
by (*rule comm-monoid.axioms*, *rule a-comm-monoid*)

lemma (**in** *abelian-group*) *a-group*:
group (| *carrier* = *carrier G*, *mult* = *add G*, *one* = *zero G* |)
by (*simp add*: *group-def a-monoid*)
(*simp add*: *comm-group.axioms group.axioms a-comm-group*)

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

lemma (**in** *abelian-monoid*) *a-closed* [*intro*, *simp*]:
$[\![\ x \in carrier\ G;\ y \in carrier\ G\ ]\!] \Longrightarrow x \oplus y \in carrier\ G$
by (*rule monoid.m-closed* [*OF a-monoid*, *simplified monoid-record-simps*])

lemma (**in** *abelian-monoid*) *zero-closed* [*intro*, *simp*]:
$\mathbf{0} \in carrier\ G$
by (*rule monoid.one-closed* [*OF a-monoid*, *simplified monoid-record-simps*])

lemma (**in** *abelian-group*) *a-inv-closed* [*intro*, *simp*]:
$x \in carrier\ G ==> \ominus\ x \in carrier\ G$
by (*simp add*: *a-inv-def*
group.inv-closed [*OF a-group*, *simplified monoid-record-simps*])

lemma (**in** *abelian-group*) *minus-closed* [*intro*, *simp*]:
$[|\ x \in carrier\ G;\ y \in carrier\ G\ |] ==> x \ominus y \in carrier\ G$
by (*simp add*: *a-minus-def*)

lemma (**in** *abelian-group*) *a-l-cancel* [*simp*]:
$[|\ x \in carrier\ G;\ y \in carrier\ G;\ z \in carrier\ G\ |] ==>$
$(x \oplus y = x \oplus z) = (y = z)$
by (*rule group.l-cancel* [*OF a-group*, *simplified monoid-record-simps*])

lemma (**in** *abelian-group*) *a-r-cancel* [*simp*]:
$[|\ x \in carrier\ G;\ y \in carrier\ G;\ z \in carrier\ G\ |] ==>$
$(y \oplus x = z \oplus x) = (y = z)$
by (*rule group.r-cancel* [*OF a-group*, *simplified monoid-record-simps*])

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

by (*rule monoid.m-assoc* [*OF a-monoid*, *simplified monoid-record-simps*])

lemma (**in** *abelian-monoid*) *l-zero* [*simp*]:
$x \in carrier\ G ==> \mathbf{0} \oplus x = x$
by (*rule monoid.l-one* [*OF a-monoid*, *simplified monoid-record-simps*])

lemma (**in** *abelian-group*) *l-neg*:
$x \in carrier\ G ==> \ominus\ x \oplus x = \mathbf{0}$
by (*simp add*: *a-inv-def*
group.l-inv [*OF a-group*, *simplified monoid-record-simps*])

lemma (**in** *abelian-monoid*) *a-comm*:
$[\![x \in carrier\ G;\ y \in carrier\ G]\!] \Longrightarrow x \oplus y = y \oplus x$
by (*rule comm-monoid.m-comm* [*OF a-comm-monoid*,
simplified monoid-record-simps])

lemma (**in** *abelian-monoid*) *a-lcomm*:
$[\![x \in carrier\ G;\ y \in carrier\ G;\ z \in carrier\ G]\!] \Longrightarrow$
$x \oplus (y \oplus z) = y \oplus (x \oplus z)$
by (*rule comm-monoid.m-lcomm* [*OF a-comm-monoid*,
simplified monoid-record-simps])

lemma (**in** *abelian-monoid*) *r-zero* [*simp*]:
$x \in carrier\ G ==> x \oplus \mathbf{0} = x$
using *monoid.r-one* [*OF a-monoid*]
by *simp*

lemma (**in** *abelian-group*) *r-neg*:
$x \in carrier\ G ==> x \oplus (\ominus\ x) = \mathbf{0}$
using *group.r-inv* [*OF a-group*]
by (*simp add*: *a-inv-def*)

lemma (**in** *abelian-group*) *minus-zero* [*simp*]:
$\ominus\ \mathbf{0} = \mathbf{0}$
by (*simp add*: *a-inv-def*
group.inv-one [*OF a-group*, *simplified monoid-record-simps*])

lemma (**in** *abelian-group*) *minus-minus* [*simp*]:
$x \in carrier\ G ==> \ominus\ (\ominus\ x) = x$
using *group.inv-inv* [*OF a-group*, *simplified monoid-record-simps*]
by (*simp add*: *a-inv-def*)

lemma (**in** *abelian-group*) *a-inv-inj*:
inj-on (*a-inv G*) (*carrier G*)
using *group.inv-inj* [*OF a-group*, *simplified monoid-record-simps*]
by (*simp add*: *a-inv-def*)

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

```
  using comm-group.inv-mult [OF a-comm-group]
  by (simp add: a-inv-def)

lemma (in abelian-group) minus-equality:
  [| x ∈ carrier G; y ∈ carrier G; y ⊕ x = 0 |] ==> ⊖ x = y
  using group.inv-equality [OF a-group]
  by (auto simp add: a-inv-def)

lemma (in abelian-monoid) minus-unique:
  [| x ∈ carrier G; y ∈ carrier G; y' ∈ carrier G;
     y ⊕ x = 0; x ⊕ y' = 0 |] ==> y = y'
  using monoid.inv-unique [OF a-monoid]
  by (simp add: a-inv-def)

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

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

```
lemma comm-group-abelian-groupI:
  fixes G (structure)
  assumes cg: comm-group (|carrier = carrier G, mult = add G, one = zero G|)
  shows abelian-group G
proof −
  interpret comm-group [(|carrier = carrier G, mult = add G, one = zero G|)]
    by (rule cg)
  show abelian-group G by (unfold-locales)
qed
```

8.2 Sums over Finite Sets

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

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

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

lemma (**in** *abelian-monoid*) *finsum-empty* [*simp*]:
finsum G f {} = **0**
by (*rule comm-monoid.finprod-empty* [*OF a-comm-monoid*,
folded finsum-def, *simplified monoid-record-simps*])

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
by (*rule comm-monoid.finprod-insert* [*OF a-comm-monoid*,
folded finsum-def, *simplified monoid-record-simps*])

lemma (**in** *abelian-monoid*) *finsum-zero* [*simp*]:
finite A ==> ($\bigoplus i{\in}A$. **0**) = **0**
by (*rule comm-monoid.finprod-one* [*OF a-comm-monoid*, *folded finsum-def*,
simplified monoid-record-simps])

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
apply (*rule comm-monoid.finprod-closed* [*OF a-comm-monoid*,
folded finsum-def, *simplified monoid-record-simps*])
apply (*rule fin*)
apply (*rule* f)
done

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
by (*rule comm-monoid.finprod-Un-Int* [*OF a-comm-monoid*,
folded finsum-def, *simplified monoid-record-simps*])

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
by (*rule comm-monoid.finprod-Un-disjoint* [*OF a-comm-monoid*,
folded finsum-def, *simplified monoid-record-simps*])

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)
by (*rule comm-monoid.finprod-multf* [*OF a-comm-monoid*,
folded finsum-def, *simplified monoid-record-simps*])

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*
by (*rule comm-monoid.finprod-cong'* [*OF a-comm-monoid*,
folded finsum-def, *simplified monoid-record-simps*]) *auto*

lemma (**in** *abelian-monoid*) *finsum-0* [*simp*]:
f : {*0*::*nat*} −> *carrier G* ==> *finsum G f* {..*0*} = *f 0*
by (*rule comm-monoid.finprod-0* [*OF a-comm-monoid*, *folded finsum-def*,
simplified monoid-record-simps])

lemma (**in** *abelian-monoid*) *finsum-Suc* [*simp*]:
f : {..*Suc n*} −> *carrier G* ==>
finsum G f {..*Suc n*} = (*f* (*Suc n*) ⊕ *finsum G f* {..*n*})
by (*rule comm-monoid.finprod-Suc* [*OF a-comm-monoid*, *folded finsum-def*,
simplified monoid-record-simps])

lemma (**in** *abelian-monoid*) *finsum-Suc2*:
f : {..*Suc n*} −> *carrier G* ==>
finsum G f {..*Suc n*} = (*finsum G* (%*i*. *f* (*Suc i*)) {..*n*} ⊕ *f 0*)
by (*rule comm-monoid.finprod-Suc2* [*OF a-comm-monoid*, *folded finsum-def*,
simplified monoid-record-simps])

lemma (**in** *abelian-monoid*) *finsum-add* [*simp*]:
[| *f* : {..*n*} −> *carrier G*; *g* : {..*n*} −> *carrier G* |] ==>
finsum G (%*i*. *f i* ⊕ *g i*) {..*n*::*nat*} =
finsum G f {..*n*} ⊕ *finsum G g* {..*n*}
by (*rule comm-monoid.finprod-mult* [*OF a-comm-monoid*, *folded finsum-def*,
simplified monoid-record-simps])

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*
by (*rule comm-monoid.finprod-cong* [*OF a-comm-monoid*, *folded finsum-def*,
simplified monoid-record-simps]) (*auto simp add*: *simp-implies-def*)

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

9 The Algebraic Hierarchy of Rings

9.1 Basic Definitions

locale *ring* = *abelian-group R* + *monoid R* +
assumes *l-distr*: [| *x* ∈ *carrier R*; *y* ∈ *carrier R*; *z* ∈ *carrier R* |]
==> (*x* ⊕ *y*) ⊗ *z* = *x* ⊗ *z* ⊕ *y* ⊗ *z*
and *r-distr*: [| *x* ∈ *carrier R*; *y* ∈ *carrier R*; *z* ∈ *carrier R* |]
==> *z* ⊗ (*x* ⊕ *y*) = *z* ⊗ *x* ⊕ *z* ⊗ *y*

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

locale *domain* = *cring* +
  assumes *one-not-zero* [*simp*]: **1** ~= **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* − {**0**}

9.2 Rings

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

lemma (**in** *ring*) *is-abelian-group*:
  abelian-group R
  by (*auto intro*!: *abelian-groupI a-assoc a-comm l-neg*)

lemma (**in** *ring*) *is-monoid*:
  monoid R
  by (*auto intro*!: *monoidI m-assoc*)

lemma (**in** *ring*) *is-ring*:
  ring R
  by *fact*

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

lemma *cringI*:
  fixes *R* (**structure**)
  assumes *abelian-group*: *abelian-group R*
    and *comm-monoid*: *comm-monoid R*
    and *l-distr*: !!*x y z*. [| $x \in$ *carrier R*; $y \in$ *carrier R*; $z \in$ *carrier R* |]
      ==> $(x \oplus y) \otimes z = x \otimes z \oplus y \otimes z$
  shows *cring R*
proof (*intro cring.intro ring.intro*)
  show *ring-axioms R*
    — Right-distributivity follows from left-distributivity and commutativity.
  proof (*rule ring-axioms.intro*)
    fix *x y z*

assume *R*: $x \in$ *carrier R* $y \in$ *carrier R* $z \in$ *carrier R*
note [*simp*] = *comm-monoid.axioms* [*OF comm-monoid*]
abelian-group.axioms [*OF abelian-group*]
abelian-monoid.a-closed

from *R* **have** $z \otimes (x \oplus y) = (x \oplus y) \otimes z$
by (*simp add*: *comm-monoid.m-comm* [*OF comm-monoid.intro*])
also from *R* **have** ... $= x \otimes z \oplus y \otimes z$ **by** (*simp add*: *l-distr*)
also from *R* **have** ... $= z \otimes x \oplus z \otimes y$
by (*simp add*: *comm-monoid.m-comm* [*OF comm-monoid.intro*])
finally show $z \otimes (x \oplus y) = z \otimes x \oplus z \otimes y$ **.**
qed (*rule l-distr*)
qed (*auto intro*: *cring.intro*
abelian-group.axioms comm-monoid.axioms ring-axioms.intro prems)

lemma (**in** *cring*) *is-comm-monoid*:
comm-monoid R
by (*auto intro*!: *comm-monoidI m-assoc m-comm*)

lemma (**in** *cring*) *is-cring*:
cring R **by** *fact*

9.2.1 Normaliser for Rings

lemma (**in** *abelian-group*) *r-neg2*:
[| $x \in$ *carrier G*; $y \in$ *carrier G* |] ==> $x \oplus (\ominus x \oplus y) = y$
proof –
assume *G*: $x \in$ *carrier G* $y \in$ *carrier G*
then have $(x \oplus \ominus x) \oplus y = y$
by (*simp only*: *r-neg l-zero*)
with *G* **show** *?thesis*
by (*simp add*: *a-ac*)
qed

lemma (**in** *abelian-group*) *r-neg1*:
[| $x \in$ *carrier G*; $y \in$ *carrier G* |] ==> $\ominus x \oplus (x \oplus y) = y$
proof –
assume *G*: $x \in$ *carrier G* $y \in$ *carrier G*
then have $(\ominus x \oplus x) \oplus y = y$
by (*simp only*: *l-neg l-zero*)
with *G* **show** *?thesis* **by** (*simp add*: *a-ac*)
qed

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 –
assume *R*: $x \in$ *carrier R*
then have $\mathbf{0} \otimes x \oplus \mathbf{0} \otimes x = (\mathbf{0} \oplus \mathbf{0}) \otimes x$

by (*simp add*: *l-distr del*: *l-zero r-zero*)
also from R **have** ... $= \mathbf{0} \otimes x \oplus \mathbf{0}$ **by** *simp*
finally have $\mathbf{0} \otimes x \oplus \mathbf{0} \otimes x = \mathbf{0} \otimes x \oplus \mathbf{0}$ **.**
with R **show** *?thesis* **by** (*simp del*: *r-zero*)
qed

lemma (**in** *ring*) *r-null* [*simp*]:
$x \in carrier\ R ==> x \otimes \mathbf{0} = \mathbf{0}$
proof −
assume R: $x \in carrier\ R$
then have $x \otimes \mathbf{0} \oplus x \otimes \mathbf{0} = x \otimes (\mathbf{0} \oplus \mathbf{0})$
by (*simp add*: *r-distr del*: *l-zero r-zero*)
also from R **have** ... $= x \otimes \mathbf{0} \oplus \mathbf{0}$ **by** *simp*
finally have $x \otimes \mathbf{0} \oplus x \otimes \mathbf{0} = x \otimes \mathbf{0} \oplus \mathbf{0}$ **.**
with R **show** *?thesis* **by** (*simp del*: *r-zero*)
qed

lemma (**in** *ring*) *l-minus*:
$[|\ x \in carrier\ R;\ y \in carrier\ R\ |] ==> \ominus\ x \otimes y = \ominus\ (x \otimes y)$
proof −
assume R: $x \in carrier\ R\ y \in carrier\ R$
then have $(\ominus\ x) \otimes y \oplus x \otimes y = (\ominus\ x \oplus x) \otimes y$ **by** (*simp add*: *l-distr*)
also from R **have** ... $= \mathbf{0}$ **by** (*simp add*: *l-neg l-null*)
finally have $(\ominus\ x) \otimes y \oplus x \otimes y = \mathbf{0}$ **.**
with R **have** $(\ominus\ x) \otimes y \oplus x \otimes y \oplus \ominus\ (x \otimes y) = \mathbf{0} \oplus \ominus\ (x \otimes y)$ **by** *simp*
with R **show** *?thesis* **by** (*simp add*: *a-assoc r-neg*)
qed

lemma (**in** *ring*) *r-minus*:
$[|\ x \in carrier\ R;\ y \in carrier\ R\ |] ==> x \otimes \ominus\ y = \ominus\ (x \otimes y)$
proof −
assume R: $x \in carrier\ R\ y \in carrier\ R$
then have $x \otimes (\ominus\ y) \oplus x \otimes y = x \otimes (\ominus\ y \oplus y)$ **by** (*simp add*: *r-distr*)
also from R **have** ... $= \mathbf{0}$ **by** (*simp add*: *l-neg r-null*)
finally have $x \otimes (\ominus\ y) \oplus x \otimes y = \mathbf{0}$ **.**
with R **have** $x \otimes (\ominus\ y) \oplus x \otimes y \oplus \ominus\ (x \otimes y) = \mathbf{0} \oplus \ominus\ (x \otimes y)$ **by** *simp*
with R **show** *?thesis* **by** (*simp add*: *a-assoc r-neg*)
qed

lemma (**in** *abelian-group*) *minus-eq*:
$[|\ x \in carrier\ G;\ y \in carrier\ G\ |] ==> x \ominus y = x \oplus \ominus\ y$
by (*simp only*: *a-minus-def*)

Setup algebra method: compute distributive normal form in locale contexts

use *ringsimp.ML*

setup *Algebra.setup*

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

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

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

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

lemma (**in** *cring*) *nat-pow-zero*:
(*n*::*nat*) ~= *0* ==> **0** (^) *n* = **0**
by (*induct n*) *simp-all*

lemma (**in** *ring*) *one-zeroD*:
assumes *onezero*: **1** = **0**
shows *carrier R* = {**0**}
proof (*rule*, *rule*)
fix *x*
assume *xcarr*: $x \in$ *carrier R*
from *xcarr*
have $x = x \otimes$ **1** **by** *simp*
from *this* **and** *onezero*
have $x = x \otimes$ **0** **by** *simp*
from *this* **and** *xcarr*
have $x =$ **0** **by** *simp*
thus $x \in$ {**0**} **by** *fast*
qed *fast*

lemma (**in** *ring*) *one-zeroI*:
assumes *carrzero*: *carrier R* = {**0**}
shows **1** = **0**
proof −
from *one-closed* **and** *carrzero*
show **1** = **0** **by** *simp*
qed

lemma (**in** *ring*) *one-zero*:
shows (*carrier R* = {**0**}) = (**1** = **0**)
by (*rule*, *erule one-zeroI*, *erule one-zeroD*)

lemma (**in** *ring*) *one-not-zero*:
  shows $(carrier\ R \neq \{\mathbf{0}\}) = (\mathbf{1} \neq \mathbf{0})$
  by (*simp add*: *one-zero*)

Two examples for use of method algebra

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

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

9.2.2 Sums over Finite Sets

lemma (**in** *cring*) *finsum-ldistr*:
  [| *finite A*; $a \in carrier\ R$; $f \in A$ −> *carrier R* |] ==>
   finsum R f A $\otimes$ a = *finsum R* (%*i*. $f\ i \otimes a$) *A*
proof (*induct set*: *finite*)
  case *empty* **then show** *?case* **by** *simp*
next
  case (*insert x F*) **then show** *?case* **by** (*simp add*: *Pi-def l-distr*)
qed

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 (*induct set*: *finite*)
  case *empty* **then show** *?case* **by** *simp*
next
  case (*insert x F*) **then show** *?case* **by** (*simp add*: *Pi-def r-distr*)
qed

9.3 Integral Domains

lemma (**in** *domain*) *zero-not-one* [*simp*]:
  $\mathbf{0}$ ~= $\mathbf{1}$
  by (*rule not-sym*) *simp*

lemma (**in** *domain*) *integral-iff*:
  [| $a \in carrier\ R$; $b \in carrier\ R$ |] ==> $(a \otimes b = \mathbf{0}) = (a = \mathbf{0} \mid b = \mathbf{0})$
proof
  assume $a \in carrier\ R$ $b \in carrier\ R$ $a \otimes b = \mathbf{0}$
  then show $a = \mathbf{0} \mid b = \mathbf{0}$ **by** (*simp add*: *integral*)
next
  assume $a \in carrier\ R$ $b \in carrier\ R$ $a = \mathbf{0} \mid b = \mathbf{0}$

then show $a \otimes b = \mathbf{0}$ **by** *auto*
qed

lemma (**in** *domain*) *m-lcancel*:
assumes *prem*: $a \,\tilde{}{=}\, \mathbf{0}$
and *R*: $a \in$ *carrier R* $b \in$ *carrier R* $c \in$ *carrier R*
shows $(a \otimes b = a \otimes c) = (b = c)$
proof
assume *eq*: $a \otimes b = a \otimes c$
with *R* **have** $a \otimes (b \ominus c) = \mathbf{0}$ **by** *algebra*
with *R* **have** $a = \mathbf{0} \mid (b \ominus c) = \mathbf{0}$ **by** (*simp add*: *integral-iff*)
with *prem* **and** *R* **have** $b \ominus c = \mathbf{0}$ **by** *auto*
with *R* **have** $b = b \ominus (b \ominus c)$ **by** *algebra*
also from *R* **have** $b \ominus (b \ominus c) = c$ **by** *algebra*
finally show $b = c$ **.**
next
assume $b = c$ **then show** $a \otimes b = a \otimes c$ **by** *simp*
qed

lemma (**in** *domain*) *m-rcancel*:
assumes *prem*: $a \,\tilde{}{=}\, \mathbf{0}$
and *R*: $a \in$ *carrier R* $b \in$ *carrier R* $c \in$ *carrier R*
shows *conc*: $(b \otimes a = c \otimes a) = (b = c)$
proof −
from *prem* **and** *R* **have** $(a \otimes b = a \otimes c) = (b = c)$ **by** (*rule m-lcancel*)
with *R* **show** *?thesis* **by** *algebra*
qed

9.4 Fields

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

lemma (**in** *cring*) *cring-fieldI*:
assumes *field-Units*: *Units R* = *carrier R* − $\{\mathbf{0}\}$
shows *field R*
proof *unfold-locales*
from *field-Units*
have *a*: $\mathbf{0} \notin$ *Units R* **by** *fast*
have $\mathbf{1} \in$ *Units R* **by** *fast*
from *this* **and** *a*
show $\mathbf{1} \neq \mathbf{0}$ **by** *force*
next
fix *a b*
assume *acarr*: $a \in$ *carrier R*
and *bcarr*: $b \in$ *carrier R*
and *ab*: $a \otimes b = \mathbf{0}$
show $a = \mathbf{0} \vee b = \mathbf{0}$
proof (*cases* $a = \mathbf{0}$, *simp*)
assume $a \neq \mathbf{0}$

from *this* **and** *field-Units* **and** *acarr*
have *aUnit*: $a \in$ *Units R* **by** *fast*
from *bcarr*
have $b = \mathbf{1} \otimes b$ **by** *algebra*
also from *aUnit acarr*
have ... $=$ (*inv* $a \otimes a) \otimes b$ **by** (*simp add*: *Units-l-inv*)
also from *acarr bcarr aUnit*[*THEN Units-inv-closed*]
have ... $=$ (*inv* $a) \otimes (a \otimes b)$ **by** *algebra*
also from *ab* **and** *acarr bcarr aUnit*
have ... $=$ (*inv* $a) \otimes \mathbf{0}$ **by** *simp*
also from *aUnit*[*THEN Units-inv-closed*]
have ... $= \mathbf{0}$ **by** *algebra*
finally
have $b = \mathbf{0}$ **.**
thus $a = \mathbf{0} \vee b = \mathbf{0}$ **by** *simp*
qed
qed (*rule field-Units*)

Another variant to show that something is a field

lemma (**in** *cring*) *cring-fieldI2*:
assumes *notzero*: $\mathbf{0} \neq \mathbf{1}$
and *invex*: $\bigwedge a.\ [\![a \in \mathit{carrier}\ R;\ a \neq \mathbf{0}]\!] \Longrightarrow \exists\, b{\in}\mathit{carrier}\ R.\ a \otimes b = \mathbf{1}$
shows *field R*
apply (*rule cring-fieldI*, *simp add*: *Units-def*)
apply (*rule*, *clarsimp*)
apply (*simp add*: *notzero*)
proof (*clarsimp*)
fix x
assume *xcarr*: $x \in$ *carrier R*
and $x \neq \mathbf{0}$
from *this*
have $\exists\, y{\in}\mathit{carrier}\ R.\ x \otimes y = \mathbf{1}$ **by** (*rule invex*)
from *this*
obtain y
where *ycarr*: $y \in$ *carrier R*
and *xy*: $x \otimes y = \mathbf{1}$
by *fast*
from *xy xcarr ycarr* **have** $y \otimes x = \mathbf{1}$ **by** (*simp add*: *m-comm*)
from *ycarr* **and** *this* **and** *xy*
show $\exists\, y{\in}\mathit{carrier}\ R.\ y \otimes x = \mathbf{1} \wedge x \otimes y = \mathbf{1}$ **by** *fast*
qed

9.5 Morphisms

constdefs (**structure** *R S*)
ring-hom :: [(*'a*, *'m*) *ring-scheme*, (*'b*, *'n*) *ring-scheme*] => (*'a* => *'b*) *set*
ring-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 **1** = $\mathbf{1}_S$}

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

lemma *ring-hom-closed*:
[| $h \in$ *ring-hom* R S; $x \in$ *carrier* R |] ==> $h\ x \in$ *carrier* S
by (*auto simp add*: *ring-hom-def funcset-mem*)

lemma *ring-hom-mult*:
fixes R (**structure**) **and** S (**structure**)
shows
[| $h \in$ *ring-hom* R S; $x \in$ *carrier* R; $y \in$ *carrier* R |] ==>
$h\ (x \otimes y) = h\ x \otimes_S h\ y$
by (*simp add*: *ring-hom-def*)

lemma *ring-hom-add*:
fixes R (**structure**) **and** S (**structure**)
shows
[| $h \in$ *ring-hom* R S; $x \in$ *carrier* R; $y \in$ *carrier* R |] ==>
$h\ (x \oplus y) = h\ x \oplus_S h\ y$
by (*simp add*: *ring-hom-def*)

lemma *ring-hom-one*:
fixes R (**structure**) **and** S (**structure**)
shows $h \in$ *ring-hom* R S ==> h **1** = $\mathbf{1}_S$
by (*simp add*: *ring-hom-def*)

locale *ring-hom-cring* = *cring* R + *cring* S +
fixes h
assumes *homh* [*simp*, *intro*]: $h \in$ *ring-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 **0** = $\mathbf{0}_S$
proof −
have $h\ \mathbf{0} \oplus_S h\ \mathbf{0} = h\ \mathbf{0} \oplus_S \mathbf{0}_S$
by (*simp add*: *hom-add* [*symmetric*] *del*: *hom-add*)

then show *?thesis* **by** (*simp del*: *S.r-zero*)
qed

lemma (**in** *ring-hom-cring*) *hom-a-inv* [*simp*]:
$x \in$ *carrier R* ==> h ($\ominus$ x) = $\ominus_S$ h x
proof −
assume *R*: $x \in$ *carrier R*
then have $h\ x \oplus_S h\ (\ominus\ x) = h\ x \oplus_S (\ominus_S\ h\ x)$
by (*simp add*: *hom-add* [*symmetric*] *R.r-neg S.r-neg del*: *hom-add*)
with *R* **show** *?thesis* **by** *simp*
qed

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 (*induct set*: *finite*)
case *empty* **then show** *?case* **by** *simp*
next
case *insert* **then show** *?case* **by** (*simp add*: *Pi-def*)
qed

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 (*induct set*: *finite*)
case *empty* **then show** *?case* **by** *simp*
next
case *insert* **then show** *?case* **by** (*simp add*: *Pi-def*)
qed

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

lemma *id-ring-hom* [*simp*]:
id $\in$ *ring-hom R R*
by (*auto intro*!: *ring-hom-memI*)

end

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

10 Modules over an Abelian Group

10.1 Definitions

record ('*a*, '*b*) *module* = '*b ring* +
smult :: ['*a*, '*b*] => '*b* (**infixl** $\odot_1$ *70*)

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

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

lemma *moduleI*:
fixes *R* (**structure**) **and** *M* (**structure**)
assumes *cring*: *cring R*
and *abelian-group*: *abelian-group M*
and *smult-closed*:
$!!a\ x.\ [| \ a \in \mathit{carrier}\ R;\ x \in \mathit{carrier}\ M\ |] ==> a \odot_M x \in \mathit{carrier}\ M$
and *smult-l-distr*:
$!!a\ b\ x.\ [| \ a \in \mathit{carrier}\ R;\ b \in \mathit{carrier}\ R;\ x \in \mathit{carrier}\ M\ |] ==>$
$(a \oplus b) \odot_M x = (a \odot_M x) \oplus_M (b \odot_M x)$
and *smult-r-distr*:
$!!a\ x\ y.\ [| \ a \in \mathit{carrier}\ R;\ x \in \mathit{carrier}\ M;\ y \in \mathit{carrier}\ M\ |] ==>$
$a \odot_M (x \oplus_M y) = (a \odot_M x) \oplus_M (a \odot_M y)$
and *smult-assoc1*:
$!!a\ b\ x.\ [| \ a \in \mathit{carrier}\ R;\ b \in \mathit{carrier}\ R;\ x \in \mathit{carrier}\ M\ |] ==>$
$(a \otimes b) \odot_M x = a \odot_M (b \odot_M x)$
and *smult-one*:
$!!x.\ x \in \mathit{carrier}\ M ==> \mathbf{1} \odot_M x = x$
shows *module R M*
by (*auto intro*: *module.intro cring.axioms abelian-group.axioms*
module-axioms.intro prems)

lemma *algebraI*:
fixes *R* (**structure**) **and** *M* (**structure**)
assumes *R-cring*: *cring R*
and *M-cring*: *cring M*
and *smult-closed*:
$!!a\ x.\ [| \ a \in \mathit{carrier}\ R;\ x \in \mathit{carrier}\ M\ |] ==> a \odot_M x \in \mathit{carrier}\ M$
and *smult-l-distr*:
$!!a\ b\ x.\ [| \ a \in \mathit{carrier}\ R;\ b \in \mathit{carrier}\ R;\ x \in \mathit{carrier}\ M\ |] ==>$

$(a \oplus b) \odot_M x = (a \odot_M x) \oplus_M (b \odot_M x)$
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*
apply *intro-locales*
apply (*rule cring.axioms ring.axioms abelian-group.axioms comm-monoid.axioms prems*)+
apply (*rule module-axioms.intro*)
apply (*simp add*: *smult-closed*)
apply (*simp add*: *smult-l-distr*)
apply (*simp add*: *smult-r-distr*)
apply (*simp add*: *smult-assoc1*)
apply (*simp add*: *smult-one*)
apply (*rule cring.axioms ring.axioms abelian-group.axioms comm-monoid.axioms prems*)+
apply (*rule algebra-axioms.intro*)
apply (*simp add*: *smult-assoc2*)
done

lemma (**in** *algebra*) *R-cring*:
cring R
by *unfold-locales*

lemma (**in** *algebra*) *M-cring*:
cring M
by *unfold-locales*

lemma (**in** *algebra*) *module*:
module R M
by (*auto intro*: *moduleI R-cring is-abelian-group*
smult-l-distr smult-r-distr smult-assoc1)

10.2 Basic Properties of Algebras

lemma (**in** *algebra*) *smult-l-null* [*simp*]:
$x \in$ *carrier M* ==> $\mathbf{0} \odot_M x = \mathbf{0}_M$
proof −
assume *M*: $x \in$ *carrier M*
note *facts* = *M smult-closed* [*OF R.zero-closed*]
from *facts* **have** $\mathbf{0} \odot_M x = (\mathbf{0} \odot_M x \oplus_M \mathbf{0} \odot_M x) \oplus_M \ominus_M (\mathbf{0} \odot_M x)$ **by** *algebra*

also from M **have** ... $= (\mathbf{0} \oplus \mathbf{0}) \odot_M x \oplus_M \ominus_M (\mathbf{0} \odot_M x)$
by (*simp add*: *smult-l-distr del*: *R.l-zero R.r-zero*)
also from *facts* **have** ... $= \mathbf{0}_M$ **apply** *algebra* **apply** *algebra* **done**
finally show *?thesis* **.**
qed

lemma (**in** *algebra*) *smult-r-null* [*simp*]:
$a \in$ *carrier* R ==> $a \odot_M \mathbf{0}_M = \mathbf{0}_M$
proof –
assume R: $a \in$ *carrier* R
note *facts* = R *smult-closed*
from *facts* **have** $a \odot_M \mathbf{0}_M = (a \odot_M \mathbf{0}_M \oplus_M a \odot_M \mathbf{0}_M) \oplus_M \ominus_M (a \odot_M \mathbf{0}_M)$
by *algebra*
also from R **have** ... $= a \odot_M (\mathbf{0}_M \oplus_M \mathbf{0}_M) \oplus_M \ominus_M (a \odot_M \mathbf{0}_M)$
by (*simp add*: *smult-r-distr del*: *M.l-zero M.r-zero*)
also from *facts* **have** ... $= \mathbf{0}_M$ **by** *algebra*
finally show *?thesis* **.**
qed

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 –
assume *RM*: $a \in$ *carrier* R $x \in$ *carrier* M
from *RM* **have** *a-smult*: $a \odot_M x \in$ *carrier* M **by** *simp*
from *RM* **have** *ma-smult*: $\ominus a \odot_M x \in$ *carrier* M **by** *simp*
note *facts* = *RM a-smult ma-smult*
from *facts* **have** $(\ominus a) \odot_M x = (\ominus a \odot_M x \oplus_M a \odot_M x) \oplus_M \ominus_M(a \odot_M x)$
by *algebra*
also from *RM* **have** ... $= (\ominus a \oplus a) \odot_M x \oplus_M \ominus_M(a \odot_M x)$
by (*simp add*: *smult-l-distr*)
also from *facts smult-l-null* **have** ... $= \ominus_M(a \odot_M x)$
apply *algebra* **apply** *algebra* **done**
finally show *?thesis* **.**
qed

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 –
assume *RM*: $a \in$ *carrier* R $x \in$ *carrier* M
note *facts* = *RM smult-closed*
from *facts* **have** $a \odot_M (\ominus_M x) = (a \odot_M \ominus_M x \oplus_M a \odot_M x) \oplus_M \ominus_M(a \odot_M x)$
by *algebra*
also from *RM* **have** ... $= a \odot_M (\ominus_M x \oplus_M x) \oplus_M \ominus_M(a \odot_M x)$
by (*simp add*: *smult-r-distr*)
also from *facts smult-r-null* **have** ... $= \ominus_M(a \odot_M x)$ **by** *algebra*
finally show *?thesis* **.**
qed

end

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

11 Univariate Polynomials

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

11.1 The Constructor for Univariate Polynomials

Functions with finite support.

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

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

lemma *bound-below*:
  assumes *bound*: *bound* $z\ m\ f$ **and** *nonzero*: $f\ n \neq z$ **shows** $n \leq m$
proof (*rule classical*)
  assume ~ *?thesis*
  then have $m < n$ **by** *arith*
  with *bound* **have** $f\ n = z$ **..**
  with *nonzero* **show** *?thesis* **by** *contradiction*
qed

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

constdefs (**structure** R)
  up :: ($'a$, $'m$) *ring-scheme* => (*nat* => $'a$) *set*
  up R == {f. $f \in$ *UNIV* −> *carrier* R & (*EX* n. *bound* **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. $\bigoplus i \in \{..n\}.\ p\ i \otimes q\ (n-i)$),

```
  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
  by (simp add: up-def Pi-def)

lemma mem-upD [dest]:
  f ∈ up R ==> f n ∈ carrier R
  by (simp add: up-def Pi-def)

lemma (in cring) bound-upD [dest]:
  f ∈ up R ==> EX n. bound 0 n f
  by (simp add: up-def)

lemma (in cring) up-one-closed:
   (%n. if n = 0 then 1 else 0) ∈ up R
  using up-def by force

lemma (in cring) up-smult-closed:
  [| a ∈ carrier R; p ∈ up R |] ==> (%i. a ⊗ p i) ∈ up R
  by force

lemma (in cring) up-add-closed:
  [| p ∈ up R; q ∈ up R |] ==> (%i. p i ⊕ q i) ∈ up R
proof
  fix n
  assume p ∈ up R and q ∈ up R
  then show p n ⊕ q n ∈ carrier R
    by auto
next
  assume UP: p ∈ up R q ∈ up R
  show EX n. bound 0 n (%i. p i ⊕ q i)
  proof −
    from UP obtain n where boundn: bound 0 n p by fast
    from UP obtain m where boundm: bound 0 m q by fast
    have bound 0 (max n m) (%i. p i ⊕ q i)
    proof
      fix i
      assume max n m < i
      with boundn and boundm and UP show p i ⊕ q i = 0 by fastsimp
    qed
    then show ?thesis ..
  qed
```

qed

lemma (**in** *cring*) *up-a-inv-closed*:
  $p \in up\ R ==> (\%i.\ \ominus\ (p\ i)) \in up\ R$
proof
  assume *R*: $p \in up\ R$
  then obtain n **where** *bound* **0** $n\ p$ **by** *auto*
  then have *bound* **0** n $(\%i.\ \ominus\ p\ i)$ **by** *auto*
  then show *EX* n. *bound* **0** n $(\%i.\ \ominus\ p\ i)$ **by** *auto*
qed *auto*

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
  fix n
  assume $p \in up\ R\ q \in up\ R$
  then show $(\bigoplus i \in \{..n\}.\ p\ i \otimes q\ (n-i)) \in carrier\ R$
    by (*simp add*: *mem-upD funcsetI*)
next
  assume *UP*: $p \in up\ R\ q \in up\ R$
  show *EX* n. *bound* **0** n $(\%n.\ \bigoplus i \in \{..n\}.\ p\ i \otimes q\ (n-i))$
  proof $-$
    from *UP* **obtain** n **where** *boundn*: *bound* **0** $n\ p$ **by** *fast*
    from *UP* **obtain** m **where** *boundm*: *bound* **0** $m\ q$ **by** *fast*
    have *bound* **0** $(n + m)$ $(\%n.\ \bigoplus i \in \{..n\}.\ p\ i \otimes q\ (n - i))$
    proof
      fix k **assume** *bound*: $n + m < k$
      {
        fix i
        have $p\ i \otimes q\ (k-i) =$ **0**
        proof (*cases* $n < i$)
          case *True*
          with *boundn* **have** $p\ i =$ **0** **by** *auto*
          moreover from *UP* **have** $q\ (k-i) \in carrier\ R$ **by** *auto*
          ultimately show *?thesis* **by** *simp*
        next
          case *False*
          with *bound* **have** $m < k-i$ **by** *arith*
          with *boundm* **have** $q\ (k-i) =$ **0** **by** *auto*
          moreover from *UP* **have** $p\ i \in carrier\ R$ **by** *auto*
          ultimately show *?thesis* **by** *simp*
        qed
      }
      then show $(\bigoplus i \in \{..k\}.\ p\ i \otimes q\ (k-i)) =$ **0**
        by (*simp add*: *Pi-def*)
    qed
    then show *?thesis* **by** *fast*
  qed

qed

11.2 Effect of Operations on Coefficients

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

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

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

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

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

lemma (**in** *UP-cring*) *coeff-monom* [*simp*]:
  $a \in carrier\ R$ ==>
  coeff P (*monom P a m*) *n* = (*if m=n then a else* **0**)
proof −
  assume *R*: $a \in carrier\ R$
  then have (%*n*. *if n* = *m then a else* **0**) $\in$ *up R*
    using *up-def* **by** *force*
  with *R* **show** *?thesis* **by** (*simp add*: *UP-def*)
qed

lemma (**in** *UP-cring*) *coeff-zero* [*simp*]:
  coeff P $\mathbf{0}_P$ *n* = **0**
  by (*auto simp add*: *UP-def*)

lemma (**in** *UP-cring*) *coeff-one* [*simp*]:
  coeff P $\mathbf{1}_P$ *n* = (*if n=0 then* **1** *else* **0**)
  using *up-one-closed* **by** (*simp add*: *UP-def*)

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*
  by (*simp add*: *UP-def up-smult-closed*)

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*
  by (*simp add*: *UP-def up-add-closed*)

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* ($n-i$))
  by (*simp add*: *UP-def up-mult-closed*)

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
fix x
from *prem* **and** *R* **show** $p\ x = q\ x$ **by** (*simp add*: *UP-def*)
qed

11.3 Polynomials Form a Commutative Ring.

Operations are closed over P.

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

lemma (**in** *UP-cring*) *UP-one-closed* [*simp*]:
$\mathbf{1}_P \in$ *carrier P*
by (*simp add*: *UP-def up-one-closed*)

lemma (**in** *UP-cring*) *UP-zero-closed* [*intro*, *simp*]:
$\mathbf{0}_P \in$ *carrier P*
by (*auto simp add*: *UP-def*)

lemma (**in** *UP-cring*) *UP-a-closed* [*intro*, *simp*]:
[| $p \in$ *carrier P*; $q \in$ *carrier P* |] ==> $p \oplus_P q \in$ *carrier P*
by (*simp add*: *UP-def up-add-closed*)

lemma (**in** *UP-cring*) *monom-closed* [*simp*]:
$a \in$ *carrier R* ==> *monom P a n* $\in$ *carrier P*
by (*auto simp add*: *UP-def up-def Pi-def*)

lemma (**in** *UP-cring*) *UP-smult-closed* [*simp*]:
[| $a \in$ *carrier R*; $p \in$ *carrier P* |] ==> $a \odot_P p \in$ *carrier P*
by (*simp add*: *UP-def up-smult-closed*)

lemma (**in** *UP*) *coeff-closed* [*simp*]:
$p \in$ *carrier P* ==> *coeff P p n* $\in$ *carrier R*
by (*auto simp add*: *UP-def*)

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 $(p \oplus_P q) \oplus_P r = p \oplus_P (q \oplus_P r)$
by (*rule up-eqI*, *simp add*: *a-assoc R*, *simp-all add*: *R*)

lemma (**in** *UP-cring*) *UP-l-zero* [*simp*]:
assumes *R*: $p \in$ *carrier P*

```
  shows 0_P ⊕_P p = p
  by (rule up-eqI, simp-all add: R)

lemma (in UP-cring) UP-l-neg-ex:
  assumes R: p ∈ carrier P
  shows EX q : carrier P. q ⊕_P p = 0_P
proof −
  let ?q = %i. ⊖ (p i)
  from R have closed: ?q ∈ carrier P
    by (simp add: UP-def P-def up-a-inv-closed)
  from R have coeff: !!n. coeff P ?q n = ⊖ (coeff P p n)
    by (simp add: UP-def P-def up-a-inv-closed)
  show ?thesis
  proof
    show ?q ⊕_P p = 0_P
      by (auto intro!: up-eqI simp add: R closed coeff R.l-neg)
  qed (rule closed)
qed

lemma (in UP-cring) UP-a-comm:
  assumes R: p ∈ carrier P q ∈ carrier P
  shows p ⊕_P q = q ⊕_P p
  by (rule up-eqI, simp add: a-comm R, simp-all add: R)

lemma (in UP-cring) UP-m-assoc:
  assumes R: p ∈ carrier P q ∈ carrier P r ∈ carrier P
  shows (p ⊗_P q) ⊗_P r = p ⊗_P (q ⊗_P r)
proof (rule up-eqI)
  fix n
  {
    fix k and a b c :: nat=>'a
    assume R: a ∈ UNIV −> carrier R b ∈ UNIV −> carrier R
      c ∈ UNIV −> carrier R
    then have k <= n ==>
      (⨁ j ∈ {..k}. (⨁ i ∈ {..j}. a i ⊗ b (j−i)) ⊗ c (n−j)) =
      (⨁ j ∈ {..k}. a j ⊗ (⨁ i ∈ {..k−j}. b i ⊗ c (n−j−i)))
      (is - ⟹ ?eq k)
    proof (induct k)
      case 0 then show ?case by (simp add: Pi-def m-assoc)
    next
      case (Suc k)
      then have k <= n by arith
      from this R have ?eq k by (rule Suc)
      with R show ?case
        by (simp cong: finsum-cong
             add: Suc-diff-le Pi-def l-distr r-distr m-assoc)
          (simp cong: finsum-cong add: Pi-def a-ac finsum-ldistr m-assoc)
    qed
  }
```

```
  with R show coeff P ((p ⊗P q) ⊗P r) n = coeff P (p ⊗P (q ⊗P r)) n
    by (simp add: Pi-def)
qed (simp-all add: R)

lemma (in UP-cring) UP-l-one [simp]:
  assumes R: p ∈ carrier P
  shows 1P ⊗P p = p
proof (rule up-eqI)
  fix n
  show coeff P (1P ⊗P p) n = coeff P p n
  proof (cases n)
    case 0 with R show ?thesis by simp
  next
    case Suc with R show ?thesis
      by (simp del: finsum-Suc add: finsum-Suc2 Pi-def)
  qed
qed (simp-all add: R)

lemma (in UP-cring) UP-l-distr:
  assumes R: p ∈ carrier P q ∈ carrier P r ∈ carrier P
  shows (p ⊕P q) ⊗P r = (p ⊗P r) ⊕P (q ⊗P r)
  by (rule up-eqI) (simp add: l-distr R Pi-def, simp-all add: R)

lemma (in UP-cring) UP-m-comm:
  assumes R: p ∈ carrier P q ∈ carrier P
  shows p ⊗P q = q ⊗P p
proof (rule up-eqI)
  fix n
  {
    fix k and a b :: nat=>'a
    assume R: a ∈ UNIV -> carrier R b ∈ UNIV -> carrier R
    then have k <= n ==>
      (⨁ i ∈ {..k}. a i ⊗ b (n-i)) =
      (⨁ i ∈ {..k}. a (k-i) ⊗ b (i+n-k))
      (is - ⟹ ?eq k)
    proof (induct k)
      case 0 then show ?case by (simp add: Pi-def)
    next
      case (Suc k) then show ?case
        by (subst (2) finsum-Suc2) (simp add: Pi-def a-comm)+
    qed
  }
  note l = this
  from R show coeff P (p ⊗P q) n =  coeff P (q ⊗P p) n
    apply (simp add: Pi-def)
    apply (subst l)
    apply (auto simp add: Pi-def)
    apply (simp add: m-comm)
    done
```

qed (*simp-all add*: *R*)

theorem (**in** *UP-cring*) *UP-cring*:
cring P
by (*auto intro*!: *cringI abelian-groupI comm-monoidI UP-a-assoc UP-l-zero UP-l-neg-ex UP-a-comm UP-m-assoc UP-l-one UP-m-comm UP-l-distr*)

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

ring P
by (*auto intro*: *ring.intro cring.axioms UP-cring*)

lemma (**in** *UP-cring*) *UP-a-inv-closed* [*intro*, *simp*]:
$p \in$ *carrier P* ==> $\ominus_P$ $p \in$ *carrier P*
by (*rule abelian-group.a-inv-closed*
[*OF ring.is-abelian-group* [*OF UP-ring*]])

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 −
from *R coeff-closed UP-a-inv-closed* **have**
coeff P ($\ominus_P$ *p*) *n* = $\ominus$ *coeff P p n* $\oplus$ (*coeff P p n* $\oplus$ *coeff P* ($\ominus_P$ *p*) *n*)
by *algebra*
also from *R* **have** ... = $\ominus$ (*coeff P p n*)
by (*simp del*: *coeff-add add*: *coeff-add* [*THEN sym*]
abelian-group.r-neg [*OF ring.is-abelian-group* [*OF UP-ring*]])
finally show *?thesis* **.**
qed

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

interpretation *UP-cring* < *cring P*
by *intro-locales*
(*rule cring.axioms ring.axioms abelian-group.axioms comm-monoid.axioms UP-cring*)+

11.4 Polynomials Form an Algebra

lemma (**in** *UP-cring*) *UP-smult-l-distr*:
[| $a \in$ *carrier R*; $b \in$ *carrier R*; $p \in$ *carrier P* |] ==>
$(a \oplus b) \odot_P p = a \odot_P p \oplus_P b \odot_P p$
by (*rule up-eqI*) (*simp-all add*: *R.l-distr*)

lemma (**in** *UP-cring*) *UP-smult-r-distr*:
[| $a \in$ *carrier R*; $p \in$ *carrier P*; $q \in$ *carrier P* |] ==>
$a \odot_P (p \oplus_P q) = a \odot_P p \oplus_P a \odot_P q$
by (*rule up-eqI*) (*simp-all add*: *R.r-distr*)

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

$(a \otimes b) \odot_P p = a \odot_P (b \odot_P p)$
by (*rule up-eqI*) (*simp-all add*: *R.m-assoc*)

lemma (**in** *UP-cring*) *UP-smult-one* [*simp*]:
$p \in carrier\ P$ ==> $\mathbf{1} \odot_P p = p$
by (*rule up-eqI*) *simp-all*

lemma (**in** *UP-cring*) *UP-smult-assoc2*:
[| $a \in carrier\ R$; $p \in carrier\ P$; $q \in carrier\ P$ |] ==>
$(a \odot_P p) \otimes_P q = a \odot_P (p \otimes_P q)$
by (*rule up-eqI*) (*simp-all add*: *R.finsum-rdistr R.m-assoc Pi-def*)

Interpretation of lemmas from *algebra*.

lemma (**in** *cring*) *cring*:
cring R
by (*fast intro*: *cring.intro prems*)

lemma (**in** *UP-cring*) *UP-algebra*:
algebra R P
by (*auto intro*: *algebraI R.cring UP-cring UP-smult-l-distr UP-smult-r-distr UP-smult-assoc1 UP-smult-assoc2*)

interpretation *UP-cring* < *algebra R P*
by *intro-locales*
(*rule module.axioms algebra.axioms UP-algebra*)+

11.5 Further Lemmas Involving Monomials

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

lemma (**in** *UP-cring*) *monom-mult-is-smult*:
assumes *R*: $a \in carrier\ R$ $p \in carrier\ P$
shows $monom\ P\ a\ 0 \otimes_P p = a \odot_P p$
proof (*rule up-eqI*)
fix *n*
have $coeff\ P\ (p \otimes_P monom\ P\ a\ 0)\ n = coeff\ P\ (a \odot_P p)\ n$
proof (*cases n*)
case *0* **with** *R* **show** *?thesis* **by** (*simp add*: *R.m-comm*)
next
case *Suc* **with** *R* **show** *?thesis*
by (*simp cong*: *R.finsum-cong add*: *R.r-null Pi-def*)
(*simp add*: *R.m-comm*)
qed
with *R* **show** $coeff\ P\ (monom\ P\ a\ 0 \otimes_P p)\ n = coeff\ P\ (a \odot_P p)\ n$
by (*simp add*: *UP-m-comm*)
qed (*simp-all add*: *R*)

lemma (**in** *UP-cring*) *monom-add* [*simp*]:
  [| $a \in$ *carrier R*; $b \in$ *carrier R* |] ==>
  monom P $(a \oplus b)$ n = *monom P a n* $\oplus_P$ *monom P b n*
  by (*rule up-eqI*) *simp-all*

lemma (**in** *UP-cring*) *monom-one-Suc*:
  monom P **1** (*Suc n*) = *monom P* **1** *n* $\otimes_P$ *monom P* **1** *1*
proof (*rule up-eqI*)
  fix *k*
  show *coeff P* (*monom P* **1** (*Suc n*)) *k* = *coeff P* (*monom P* **1** *n* $\otimes_P$ *monom P* **1** *1*) *k*
  proof (*cases k* = *Suc n*)
    case *True* **show** *?thesis*
    proof –
      from *True* **have** *less-add-diff*:
        !!*i*. [| $n < i$; $i <= n + m$ |] ==> $n + m - i < m$ **by** *arith*
      from *True* **have** *coeff P* (*monom P* **1** (*Suc n*)) *k* = **1** **by** *simp*
      also from *True*
      have ... = ($\bigoplus i \in \{..<n\} \cup \{n\}$. *coeff P* (*monom P* **1** *n*) $i \otimes$
        coeff P (*monom P* **1** *1*) $(k - i)$)
        by (*simp cong*: *R.finsum-cong add*: *Pi-def*)
      also have ... = ($\bigoplus i \in \{..n\}$. *coeff P* (*monom P* **1** *n*) $i \otimes$
        coeff P (*monom P* **1** *1*) $(k - i)$)
        by (*simp only*: *ivl-disj-un-singleton*)
      also from *True*
      have ... = ($\bigoplus i \in \{..n\} \cup \{n<..k\}$. *coeff P* (*monom P* **1** *n*) $i \otimes$
        coeff P (*monom P* **1** *1*) $(k - i)$)
        by (*simp cong*: *R.finsum-cong add*: *R.finsum-Un-disjoint ivl-disj-int-one*
          order-less-imp-not-eq Pi-def)
      also from *True* **have** ... = *coeff P* (*monom P* **1** *n* $\otimes_P$ *monom P* **1** *1*) *k*
        by (*simp add*: *ivl-disj-un-one*)
      finally show *?thesis* **.**
    qed
  next
    case *False*
    note *neq* = *False*
    let *?s* =
      λi. (*if* $n = i$ *then* **1** *else* **0**) $\otimes$ (*if Suc 0* = $k - i$ *then* **1** *else* **0**)
    from *neq* **have** *coeff P* (*monom P* **1** (*Suc n*)) *k* = **0** **by** *simp*
    also have ... = ($\bigoplus i \in \{..k\}$. *?s i*)
    proof –
      have *f1*: ($\bigoplus i \in \{..<n\}$. *?s i*) = **0**
        by (*simp cong*: *R.finsum-cong add*: *Pi-def*)
      from *neq* **have** *f2*: ($\bigoplus i \in \{n\}$. *?s i*) = **0**
        by (*simp cong*: *R.finsum-cong add*: *Pi-def*) *arith*
      have *f3*: $n < k$ ==> ($\bigoplus i \in \{n<..k\}$. *?s i*) = **0**
        by (*simp cong*: *R.finsum-cong add*: *order-less-imp-not-eq Pi-def*)
      show *?thesis*
      proof (*cases* $k < n$)

```
      case True then show ?thesis by (simp cong: R.finsum-cong add: Pi-def)
    next
      case False then have n-le-k: n <= k by arith
      show ?thesis
      proof (cases n = k)
        case True
        then have 0 = (⨁ i ∈ {..<n} ∪ {n}. ?s i)
          by (simp cong: R.finsum-cong add: ivl-disj-int-singleton Pi-def)
        also from True have ... = (⨁ i ∈ {..k}. ?s i)
          by (simp only: ivl-disj-un-singleton)
        finally show ?thesis .
      next
        case False with n-le-k have n-less-k: n < k by arith
        with neq have 0 = (⨁ i ∈ {..<n} ∪ {n}. ?s i)
          by (simp add: R.finsum-Un-disjoint f1 f2
            ivl-disj-int-singleton Pi-def del: Un-insert-right)
        also have ... = (⨁ i ∈ {..n}. ?s i)
          by (simp only: ivl-disj-un-singleton)
        also from n-less-k neq have ... = (⨁ i ∈ {..n} ∪ {n<..k}. ?s i)
          by (simp add: R.finsum-Un-disjoint f3 ivl-disj-int-one Pi-def)
        also from n-less-k have ... = (⨁ i ∈ {..k}. ?s i)
          by (simp only: ivl-disj-un-one)
        finally show ?thesis .
      qed
    qed
  qed
  also have ... = coeff P (monom P 1 n ⊗P monom P 1 1) k by simp
  finally show ?thesis .
 qed
qed (simp-all)

lemma (in UP-cring) monom-mult-smult:
 [| a ∈ carrier R; b ∈ carrier R |] ==> monom P (a ⊗ b) n = a ⊙P monom P
b n
 by (rule up-eqI) simp-all

lemma (in UP-cring) monom-one [simp]:
 monom P 1 0 = 1P
 by (rule up-eqI) simp-all

lemma (in UP-cring) monom-one-mult:
 monom P 1 (n + m) = monom P 1 n ⊗P monom P 1 m
proof (induct n)
 case 0 show ?case by simp
next
 case Suc then show ?case
  by (simp only: add-Suc monom-one-Suc) (simp add: P.m-ac)
qed
```

lemma (**in** *UP-cring*) *monom-mult* [*simp*]:
 assumes *R*: $a \in$ *carrier R* $b \in$ *carrier R*
 shows *monom P* $(a \otimes b)$ $(n + m)$ = *monom P a n* $\otimes_P$ *monom P b m*
proof −
 from *R* **have** *monom P* $(a \otimes b)$ $(n + m)$ = *monom P* $(a \otimes b \otimes \mathbf{1})$ $(n + m)$
by *simp*
 also from *R* **have** ... = $a \otimes b \odot_P$ *monom P* **1** $(n + m)$
  by (*simp add*: *monom-mult-smult del*: *R.r-one*)
 also have ... = $a \otimes b \odot_P$ (*monom P* **1** *n* $\otimes_P$ *monom P* **1** *m*)
  by (*simp only*: *monom-one-mult*)
 also from *R* **have** ... = $a \odot_P$ ($b \odot_P$ (*monom P* **1** *n* $\otimes_P$ *monom P* **1** *m*))
  by (*simp add*: *UP-smult-assoc1*)
 also from *R* **have** ... = $a \odot_P$ ($b \odot_P$ (*monom P* **1** *m* $\otimes_P$ *monom P* **1** *n*))
  by (*simp add*: *P.m-comm*)
 also from *R* **have** ... = $a \odot_P$ (($b \odot_P$ *monom P* **1** *m*) $\otimes_P$ *monom P* **1** *n*)
  by (*simp add*: *UP-smult-assoc2*)
 also from *R* **have** ... = $a \odot_P$ (*monom P* **1** *n* $\otimes_P$ ($b \odot_P$ *monom P* **1** *m*))
  by (*simp add*: *P.m-comm*)
 also from *R* **have** ... = ($a \odot_P$ *monom P* **1** *n*) $\otimes_P$ ($b \odot_P$ *monom P* **1** *m*)
  by (*simp add*: *UP-smult-assoc2*)
 also from *R* **have** ... = *monom P* $(a \otimes \mathbf{1})$ *n* $\otimes_P$ *monom P* $(b \otimes \mathbf{1})$ *m*
  by (*simp add*: *monom-mult-smult del*: *R.r-one*)
 also from *R* **have** ... = *monom P a n* $\otimes_P$ *monom P b m* **by** *simp*
 finally show *?thesis* **.**
qed

lemma (**in** *UP-cring*) *monom-a-inv* [*simp*]:
 $a \in$ *carrier R* ==> *monom P* $(\ominus a)$ *n* = $\ominus_P$ *monom P a n*
 by (*rule up-eqI*) *simp-all*

lemma (**in** *UP-cring*) *monom-inj*:
 inj-on (%*a*. *monom P a n*) (*carrier R*)
proof (*rule inj-onI*)
 fix *x y*
 assume *R*: $x \in$ *carrier R* $y \in$ *carrier R* **and** *eq*: *monom P x n* = *monom P y n*
 then have *coeff P* (*monom P x n*) *n* = *coeff P* (*monom P y n*) *n* **by** *simp*
 with *R* **show** $x = y$ **by** *simp*
qed

11.6 The Degree Function

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

lemma (**in** *UP-cring*) *deg-aboveI*:
 [| (!!*m*. $n < m$ ==> *coeff P p m* = **0**); $p \in$ *carrier P* |] ==> *deg R p* <= *n*
 by (*unfold deg-def P-def*) (*fast intro*: *Least-le*)

```
lemma (in UP-cring) deg-aboveD:
  assumes deg R p < m and p ∈ carrier P
  shows coeff P p m = 0
proof –
  from ‹p ∈ carrier P› obtain n where bound 0 n (coeff P p)
    by (auto simp add: UP-def P-def)
  then have bound 0 (deg R p) (coeff P p)
    by (auto simp: deg-def P-def dest: LeastI)
  from this and ‹deg R p < m› show ?thesis ..
qed

lemma (in UP-cring) deg-belowI:
  assumes non-zero: n ~= 0 ==> coeff P p n ~= 0
    and R: p ∈ carrier P
  shows n <= deg R p
— Logically, this is a slightly stronger version of deg-aboveD
proof (cases n=0)
  case True then show ?thesis by simp
next
  case False then have coeff P p n ~= 0 by (rule non-zero)
  then have ~ deg R p < n by (fast dest: deg-aboveD intro: R)
  then show ?thesis by arith
qed

lemma (in UP-cring) lcoeff-nonzero-deg:
  assumes deg: deg R p ~= 0 and R: p ∈ carrier P
  shows coeff P p (deg R p) ~= 0
proof –
  from R obtain m where deg R p <= m and m-coeff: coeff P p m ~= 0
  proof –
    have minus: !!(n::nat) m. n ~= 0 ==> (n – Suc 0 < m) = (n <= m)
      by arith

    from deg have deg R p – 1 < (LEAST n. bound 0 n (coeff P p))
      by (unfold deg-def P-def) arith
    then have ~ bound 0 (deg R p – 1) (coeff P p) by (rule not-less-Least)
    then have EX m. deg R p – 1 < m & coeff P p m ~= 0
      by (unfold bound-def) fast
     then have EX m. deg R p <= m & coeff P p m ~= 0 by (simp add: deg
minus)
    then show ?thesis by (auto intro: that)
  qed
  with deg-belowI R have deg R p = m by fastsimp
  with m-coeff show ?thesis by simp
qed

lemma (in UP-cring) lcoeff-nonzero-nonzero:
```

assumes *deg*: *deg R p = 0* **and** *nonzero*: *p* ~= $\mathbf{0}_P$ **and** *R*: *p* ∈ *carrier P*
shows *coeff P p 0* ~= **0**
proof −
have *EX m. coeff P p m* ~= **0**
proof (*rule classical*)
assume ~ *?thesis*
with *R* **have** *p* = $\mathbf{0}_P$ **by** (*auto intro*: *up-eqI*)
with *nonzero* **show** *?thesis* **by** *contradiction*
qed
then obtain *m* **where** *coeff*: *coeff P p m* ~= **0 ..**
from *this* **and** *R* **have** *m* <= *deg R p* **by** (*rule deg-belowI*)
then have *m = 0* **by** (*simp add*: *deg*)
with *coeff* **show** *?thesis* **by** *simp*
qed

lemma (**in** *UP-cring*) *lcoeff-nonzero*:
assumes *neq*: *p* ~= $\mathbf{0}_P$ **and** *R*: *p* ∈ *carrier P*
shows *coeff P p* (*deg R p*) ~= **0**
proof (*cases deg R p = 0*)
case *True* **with** *neq R* **show** *?thesis* **by** (*simp add*: *lcoeff-nonzero-nonzero*)
next
case *False* **with** *neq R* **show** *?thesis* **by** (*simp add*: *lcoeff-nonzero-deg*)
qed

lemma (**in** *UP-cring*) *deg-eqI*:
[| !!*m. n* < *m* ==> *coeff P p m* = **0**;
!!*n. n* ~= *0* ==> *coeff P p n* ~= **0**; *p* ∈ *carrier P* |] ==> *deg R p* = *n*
by (*fast intro*: *le-anti-sym deg-aboveI deg-belowI*)

Degree and polynomial operations

lemma (**in** *UP-cring*) *deg-add* [*simp*]:
assumes *R*: *p* ∈ *carrier P q* ∈ *carrier P*
shows *deg R* ($p \oplus_P q$) <= *max* (*deg R p*) (*deg R q*)
proof (*cases deg R p* <= *deg R q*)
case *True* **show** *?thesis*
by (*rule deg-aboveI*) (*simp-all add*: *True R deg-aboveD*)
next
case *False* **show** *?thesis*
by (*rule deg-aboveI*) (*simp-all add*: *False R deg-aboveD*)
qed

lemma (**in** *UP-cring*) *deg-monom-le*:
a ∈ *carrier R* ==> *deg R* (*monom P a n*) <= *n*
by (*intro deg-aboveI*) *simp-all*

lemma (**in** *UP-cring*) *deg-monom* [*simp*]:
[| *a* ~= **0**; *a* ∈ *carrier R* |] ==> *deg R* (*monom P a n*) = *n*
by (*fastsimp intro*: *le-anti-sym deg-aboveI deg-belowI*)

lemma (**in** *UP-cring*) *deg-const* [*simp*]:
  assumes *R*: $a \in$ *carrier R* **shows** *deg R* (*monom P a 0*) = *0*
proof (*rule le-anti-sym*)
  show *deg R* (*monom P a 0*) <= *0* **by** (*rule deg-aboveI*) (*simp-all add*: *R*)
next
  show *0* <= *deg R* (*monom P a 0*) **by** (*rule deg-belowI*) (*simp-all add*: *R*)
qed

lemma (**in** *UP-cring*) *deg-zero* [*simp*]:
  deg R $\mathbf{0}_P$ = *0*
proof (*rule le-anti-sym*)
  show *deg R* $\mathbf{0}_P$ <= *0* **by** (*rule deg-aboveI*) *simp-all*
next
  show *0* <= *deg R* $\mathbf{0}_P$ **by** (*rule deg-belowI*) *simp-all*
qed

lemma (**in** *UP-cring*) *deg-one* [*simp*]:
  deg R $\mathbf{1}_P$ = *0*
proof (*rule le-anti-sym*)
  show *deg R* $\mathbf{1}_P$ <= *0* **by** (*rule deg-aboveI*) *simp-all*
next
  show *0* <= *deg R* $\mathbf{1}_P$ **by** (*rule deg-belowI*) *simp-all*
qed

lemma (**in** *UP-cring*) *deg-uminus* [*simp*]:
  assumes *R*: $p \in$ *carrier P* **shows** *deg R* ($\ominus_P$ *p*) = *deg R p*
proof (*rule le-anti-sym*)
  show *deg R* ($\ominus_P$ *p*) <= *deg R p* **by** (*simp add*: *deg-aboveI deg-aboveD R*)
next
  show *deg R p* <= *deg R* ($\ominus_P$ *p*)
    by (*simp add*: *deg-belowI lcoeff-nonzero-deg*
      inj-on-iff [*OF R.a-inv-inj*, *of* - $\mathbf{0}$, *simplified*] *R*)
qed

lemma (**in** *UP-domain*) *deg-smult-ring*:
  [| $a \in$ *carrier R*; $p \in$ *carrier P* |] ==>
  deg R (*a* $\odot_P$ *p*) <= (*if a* = $\mathbf{0}$ *then 0 else deg R p*)
  by (*cases a* = $\mathbf{0}$) (*simp add*: *deg-aboveI deg-aboveD*)+

lemma (**in** *UP-domain*) *deg-smult* [*simp*]:
  assumes *R*: $a \in$ *carrier R* $p \in$ *carrier P*
  shows *deg R* (*a* $\odot_P$ *p*) = (*if a* = $\mathbf{0}$ *then 0 else deg R p*)
proof (*rule le-anti-sym*)
  show *deg R* (*a* $\odot_P$ *p*) <= (*if a* = $\mathbf{0}$ *then 0 else deg R p*)
    using *R* **by** (*rule deg-smult-ring*)
next
  show (*if a* = $\mathbf{0}$ *then 0 else deg R p*) <= *deg R* (*a* $\odot_P$ *p*)
  proof (*cases a* = $\mathbf{0}$)
  qed (*simp*, *simp add*: *deg-belowI lcoeff-nonzero-deg integral-iff R*)

qed

lemma (**in** *UP-cring*) *deg-mult-cring*:
  assumes *R*: $p \in$ *carrier P* $q \in$ *carrier P*
  shows *deg R* ($p \otimes_P q$) <= *deg R p* + *deg R q*
proof (*rule deg-aboveI*)
  fix *m*
  assume *boundm*: *deg R p* + *deg R q* < *m*
  {
    fix *k i*
    assume *boundk*: *deg R p* + *deg R q* < *k*
    then have *coeff P p i* $\otimes$ *coeff P q* ($k - i$) = **0**
    proof (*cases deg R p* < *i*)
      case *True* **then show** *?thesis* **by** (*simp add*: *deg-aboveD R*)
    next
      case *False* **with** *boundk* **have** *deg R q* < $k - i$ **by** *arith*
      then show *?thesis* **by** (*simp add*: *deg-aboveD R*)
    qed
  }
  with *boundm R* **show** *coeff P* ($p \otimes_P q$) *m* = **0** **by** *simp*
qed (*simp add*: *R*)

lemma (**in** *UP-domain*) *deg-mult* [*simp*]:
  [| *p* ~= $\mathbf{0}_P$; *q* ~= $\mathbf{0}_P$; $p \in$ *carrier P*; $q \in$ *carrier P* |] ==>
  deg R ($p \otimes_P q$) = *deg R p* + *deg R q*
proof (*rule le-anti-sym*)
  assume $p \in$ *carrier P* $q \in$ *carrier P*
  then show *deg R* ($p \otimes_P q$) <= *deg R p* + *deg R q* **by** (*rule deg-mult-cring*)
next
  let *?s* = (%*i*. *coeff P p i* $\otimes$ *coeff P q* (*deg R p* + *deg R q* − *i*))
  assume *R*: $p \in$ *carrier P* $q \in$ *carrier P* **and** *nz*: *p* ~= $\mathbf{0}_P$ *q* ~= $\mathbf{0}_P$
  have *less-add-diff*: !!(*k*::*nat*) *n m*. *k* < *n* ==> *m* < *n* + *m* − *k* **by** *arith*
  show *deg R p* + *deg R q* <= *deg R* ($p \otimes_P q$)
  proof (*rule deg-belowI*, *simp add*: *R*)
    have ($\bigoplus i \in$ {.. *deg R p* + *deg R q*}. *?s i*)
      = ($\bigoplus i \in$ {..< *deg R p*} $\cup$ {*deg R p* .. *deg R p* + *deg R q*}. *?s i*)
      by (*simp only*: *ivl-disj-un-one*)
    also have ... = ($\bigoplus i \in$ {*deg R p* .. *deg R p* + *deg R q*}. *?s i*)
      by (*simp cong*: *R.finsum-cong add*: *R.finsum-Un-disjoint ivl-disj-int-one*
        deg-aboveD less-add-diff R Pi-def)
    also have ...= ($\bigoplus i \in$ {*deg R p*} $\cup$ {*deg R p* <.. *deg R p* + *deg R q*}. *?s i*)
      by (*simp only*: *ivl-disj-un-singleton*)
    also have ... = *coeff P p* (*deg R p*) $\otimes$ *coeff P q* (*deg R q*)
      by (*simp cong*: *R.finsum-cong*
        add: *ivl-disj-int-singleton deg-aboveD R Pi-def*)
    finally have ($\bigoplus i \in$ {.. *deg R p* + *deg R q*}. *?s i*)
      = *coeff P p* (*deg R p*) $\otimes$ *coeff P q* (*deg R q*) **.**
    with *nz* **show** ($\bigoplus i \in$ {.. *deg R p* + *deg R q*}. *?s i*) ~= **0**
      by (*simp add*: *integral-iff lcoeff-nonzero R*)

```
    qed (simp add: R)
  qed

lemma (in UP-cring) coeff-finsum:
  assumes fin: finite A
  shows p ∈ A -> carrier P ==>
    coeff P (finsum P p A) k = (⨁ i ∈ A. coeff P (p i) k)
  using fin by induct (auto simp: Pi-def)

lemma (in UP-cring) up-repr:
  assumes R: p ∈ carrier P
  shows (⨁ P i ∈ {..deg R p}. monom P (coeff P p i) i) = p
proof (rule up-eqI)
  let ?s = (%i. monom P (coeff P p i) i)
  fix k
  from R have RR: !!i. (if i = k then coeff P p i else 0) ∈ carrier R
    by simp
  show coeff P (⨁ P i ∈ {..deg R p}. ?s i) k = coeff P p k
  proof (cases k <= deg R p)
    case True
    hence coeff P (⨁ P i ∈ {..deg R p}. ?s i) k =
          coeff P (⨁ P i ∈ {..k} ∪ {k<..deg R p}. ?s i) k
      by (simp only: ivl-disj-un-one)
    also from True
    have ... = coeff P (⨁ P i ∈ {..k}. ?s i) k
      by (simp cong: R.finsum-cong add: R.finsum-Un-disjoint
        ivl-disj-int-one order-less-imp-not-eq2 coeff-finsum R RR Pi-def)
    also
    have ... = coeff P (⨁ P i ∈ {..<k} ∪ {k}. ?s i) k
      by (simp only: ivl-disj-un-singleton)
    also have ... = coeff P p k
      by (simp cong: R.finsum-cong
        add: ivl-disj-int-singleton coeff-finsum deg-aboveD R RR Pi-def)
    finally show ?thesis .
  next
    case False
    hence coeff P (⨁ P i ∈ {..deg R p}. ?s i) k =
          coeff P (⨁ P i ∈ {..<deg R p} ∪ {deg R p}. ?s i) k
      by (simp only: ivl-disj-un-singleton)
    also from False have ... = coeff P p k
      by (simp cong: R.finsum-cong
        add: ivl-disj-int-singleton coeff-finsum deg-aboveD R Pi-def)
    finally show ?thesis .
  qed
qed (simp-all add: R Pi-def)

lemma (in UP-cring) up-repr-le:
  [| deg R p <= n; p ∈ carrier P |] ==>
  (⨁ P i ∈ {..n}. monom P (coeff P p i) i) = p
```

proof –
let *?s* = (%*i*. *monom P* (*coeff P p i*) *i*)
assume *R*: *p* ∈ *carrier P* **and** *deg R p* <= *n*
then have *finsum P ?s* {..*n*} = *finsum P ?s* ({..*deg R p*} ∪ {*deg R p*<..*n*})
by (*simp only*: *ivl-disj-un-one*)
also have ... = *finsum P ?s* {..*deg R p*}
by (*simp cong*: *P.finsum-cong add*: *P.finsum-Un-disjoint ivl-disj-int-one*
deg-aboveD R Pi-def)
also have ... = *p* **using** *R* **by** (*rule up-repr*)
finally show *?thesis* **.**
qed

11.7 Polynomials over Integral Domains

lemma *domainI*:
assumes *cring*: *cring R*
and *one-not-zero*: *one R* ~= *zero R*
and *integral*: !!*a b*. [| *mult R a b* = *zero R*; *a* ∈ *carrier R*;
b ∈ *carrier R* |] ==> *a* = *zero R* | *b* = *zero R*
shows *domain R*
by (*auto intro*!: *domain.intro domain-axioms.intro cring.axioms prems*
del: *disjCI*)

lemma (**in** *UP-domain*) *UP-one-not-zero*:
$\mathbf{1}_P$ ~= $\mathbf{0}_P$
proof
assume $\mathbf{1}_P = \mathbf{0}_P$
hence *coeff P* $\mathbf{1}_P$ *0* = (*coeff P* $\mathbf{0}_P$ *0*) **by** *simp*
hence **1** = **0** **by** *simp*
with *one-not-zero* **show** *False* **by** *contradiction*
qed

lemma (**in** *UP-domain*) *UP-integral*:
[| $p \otimes_P q = \mathbf{0}_P$; *p* ∈ *carrier P*; *q* ∈ *carrier P* |] ==> $p = \mathbf{0}_P$ | $q = \mathbf{0}_P$
proof –
fix *p q*
assume *pq*: $p \otimes_P q = \mathbf{0}_P$ **and** *R*: *p* ∈ *carrier P q* ∈ *carrier P*
show $p = \mathbf{0}_P$ | $q = \mathbf{0}_P$
proof (*rule classical*)
assume *c*: ~ ($p = \mathbf{0}_P$ | $q = \mathbf{0}_P$)
with *R* **have** *deg R p* + *deg R q* = *deg R* ($p \otimes_P q$) **by** *simp*
also from *pq* **have** ... = *0* **by** *simp*
finally have *deg R p* + *deg R q* = *0* **.**
then have *f1*: *deg R p* = *0* & *deg R q* = *0* **by** *simp*
from *f1 R* **have** *p* = ($\bigoplus_P$ *i* ∈ {..*0*}. *monom P* (*coeff P p i*) *i*)
by (*simp only*: *up-repr-le*)
also from *R* **have** ... = *monom P* (*coeff P p 0*) *0* **by** *simp*
finally have *p*: *p* = *monom P* (*coeff P p 0*) *0* **.**
from *f1 R* **have** *q* = ($\bigoplus_P$ *i* ∈ {..*0*}. *monom P* (*coeff P q i*) *i*)

by (*simp only*: *up-repr-le*)
also from *R* **have** ... = *monom P* (*coeff P q 0*) *0* **by** *simp*
finally have *q*: *q* = *monom P* (*coeff P q 0*) *0* **.**
from *R* **have** *coeff P p 0* ⊗ *coeff P q 0* = *coeff P* (*p* $\otimes_P$ *q*) *0* **by** *simp*
also from *pq* **have** ... = **0** **by** *simp*
finally have *coeff P p 0* ⊗ *coeff P q 0* = **0** **.**
with *R* **have** *coeff P p 0* = **0** | *coeff P q 0* = **0**
by (*simp add*: *R.integral-iff*)
with *p q* **show** *p* = $\mathbf{0}_P$ | *q* = $\mathbf{0}_P$ **by** *fastsimp*
qed
qed

theorem (**in** *UP-domain*) *UP-domain*:
domain P
by (*auto intro*!: *domainI UP-cring UP-one-not-zero UP-integral del*: *disjCI*)

Interpretation of theorems from *domain*.

interpretation *UP-domain* < *domain P*
by *intro-locales* (*rule domain.axioms UP-domain*)+

11.8 The Evaluation Homomorphism and Universal Property

theorem (**in** *cring*) *diagonal-sum*:
[| *f* ∈ {..*n* + *m*::*nat*} −> *carrier R*; *g* ∈ {..*n* + *m*} −> *carrier R* |] ==>
(⨁ *k* ∈ {..*n* + *m*}. ⨁ *i* ∈ {..*k*}. *f i* ⊗ *g* (*k* − *i*)) =
(⨁ *k* ∈ {..*n* + *m*}. ⨁ *i* ∈ {..*n* + *m* − *k*}. *f k* ⊗ *g i*)
proof −
assume *Rf*: *f* ∈ {..*n* + *m*} −> *carrier R* **and** *Rg*: *g* ∈ {..*n* + *m*} −> *carrier R*
{
fix *j*
have *j* <= *n* + *m* ==>
(⨁ *k* ∈ {..*j*}. ⨁ *i* ∈ {..*k*}. *f i* ⊗ *g* (*k* − *i*)) =
(⨁ *k* ∈ {..*j*}. ⨁ *i* ∈ {..*j* − *k*}. *f k* ⊗ *g i*)
proof (*induct j*)
case *0* **from** *Rf Rg* **show** *?case* **by** (*simp add*: *Pi-def*)
next
case (*Suc j*)
have *R6*: !!*i k*. [| *k* <= *j*; *i* <= *Suc j* − *k* |] ==> *g i* ∈ *carrier R*
using *Suc* **by** (*auto intro*!: *funcset-mem* [*OF Rg*])
have *R8*: !!*i k*. [| *k* <= *Suc j*; *i* <= *k* |] ==> *g* (*k* − *i*) ∈ *carrier R*
using *Suc* **by** (*auto intro*!: *funcset-mem* [*OF Rg*])
have *R9*: !!*i k*. [| *k* <= *Suc j* |] ==> *f k* ∈ *carrier R*
using *Suc* **by** (*auto intro*!: *funcset-mem* [*OF Rf*])
have *R10*: !!*i k*. [| *k* <= *Suc j*; *i* <= *Suc j* − *k* |] ==> *g i* ∈ *carrier R*
using *Suc* **by** (*auto intro*!: *funcset-mem* [*OF Rg*])
have *R11*: *g 0* ∈ *carrier R*
using *Suc* **by** (*auto intro*!: *funcset-mem* [*OF Rg*])
from *Suc* **show** *?case*

```
      by (simp cong: finsum-cong add: Suc-diff-le a-ac
        Pi-def R6 R8 R9 R10 R11)
    qed
  }
  then show ?thesis by fast
qed

lemma (in abelian-monoid) boundD-carrier:
  [| bound 0 n f; n < m |] ==> f m ∈ carrier G
  by auto

theorem (in cring) cauchy-product:
  assumes bf: bound 0 n f and bg: bound 0 m g
    and Rf: f ∈ {..n} -> carrier R and Rg: g ∈ {..m} -> carrier R
  shows (⨁ k ∈ {..n + m}. ⨁ i ∈ {..k}. f i ⊗ g (k - i)) =
    (⨁ i ∈ {..n}. f i) ⊗ (⨁ i ∈ {..m}. g i)
proof -
  have f: !!x. f x ∈ carrier R
  proof -
    fix x
    show f x ∈ carrier R
      using Rf bf boundD-carrier by (cases x <= n) (auto simp: Pi-def)
  qed
  have g: !!x. g x ∈ carrier R
  proof -
    fix x
    show g x ∈ carrier R
      using Rg bg boundD-carrier by (cases x <= m) (auto simp: Pi-def)
  qed
  from f g have (⨁ k ∈ {..n + m}. ⨁ i ∈ {..k}. f i ⊗ g (k - i)) =
      (⨁ k ∈ {..n + m}. ⨁ i ∈ {..n + m - k}. f k ⊗ g i)
    by (simp add: diagonal-sum Pi-def)
  also have ... = (⨁ k ∈ {..n} ∪ {n<..n + m}. ⨁ i ∈ {..n + m - k}. f k ⊗ g i)
    by (simp only: ivl-disj-un-one)
  also from f g have ... = (⨁ k ∈ {..n}. ⨁ i ∈ {..n + m - k}. f k ⊗ g i)
    by (simp cong: finsum-cong
      add: bound.bound [OF bf] finsum-Un-disjoint ivl-disj-int-one Pi-def)
  also from f g
  have ... = (⨁ k ∈ {..n}. ⨁ i ∈ {..m} ∪ {m<..n + m - k}. f k ⊗ g i)
    by (simp cong: finsum-cong add: ivl-disj-un-one le-add-diff Pi-def)
  also from f g have ... = (⨁ k ∈ {..n}. ⨁ i ∈ {..m}. f k ⊗ g i)
    by (simp cong: finsum-cong
      add: bound.bound [OF bg] finsum-Un-disjoint ivl-disj-int-one Pi-def)
  also from f g have ... = (⨁ i ∈ {..n}. f i) ⊗ (⨁ i ∈ {..m}. g i)
    by (simp add: finsum-ldistr diagonal-sum Pi-def,
      simp cong: finsum-cong add: finsum-rdistr Pi-def)
  finally show ?thesis .
qed
```

lemma (**in** *UP-cring*) *const-ring-hom*:
  (%*a*. *monom P a 0*) $\in$ *ring-hom R P*
  by (*auto intro*!: *ring-hom-memI intro*: *up-eqI simp*: *monom-mult-is-smult*)

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*:
  fixes *S* (**structure**)
  shows $p \in$ *carrier P* ==>
  $eval\ R\ S\ phi\ s\ p = (\bigoplus_S i \in \{..deg\ R\ p\}.\ phi\ (coeff\ P\ p\ i) \otimes_S s\ (\hat{\ })_S\ i)$
  by (*unfold eval-def*, *fold P-def*) *simp*

lemma (**in** *UP*) *eval-extensional*:
  eval R S phi p $\in$ *extensional* (*carrier P*)
  by (*unfold eval-def*, *fold P-def*) *simp*

The universal property of the polynomial ring

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

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

theorem (**in** *UP-pre-univ-prop*) *eval-ring-hom*:
  assumes *S*: $s \in$ *carrier S*
  shows *eval R S h s* $\in$ *ring-hom P S*
proof (*rule ring-hom-memI*)
  fix *p*
  assume *R*: $p \in$ *carrier P*
  then show *eval R S h s p* $\in$ *carrier S*
    by (*simp only*: *eval-on-carrier*) (*simp add*: *S Pi-def*)
next
  fix *p q*
  assume *R*: $p \in$ *carrier P* $q \in$ *carrier P*
  then show $eval\ R\ S\ h\ s\ (p \otimes_P q) = eval\ R\ S\ h\ s\ p \otimes_S eval\ R\ S\ h\ s\ q$
  proof (*simp only*: *eval-on-carrier UP-mult-closed*)
    from *R S* **have**
      $(\bigoplus_S i \in \{..deg\ R\ (p \otimes_P q)\}.\ h\ (coeff\ P\ (p \otimes_P q)\ i) \otimes_S s\ (\hat{\ })_S\ i) =$
      $(\bigoplus_S i \in \{..deg\ R\ (p \otimes_P q)\} \cup \{deg\ R\ (p \otimes_P q)<..deg\ R\ p + deg\ R\ q\}.$
        $h\ (coeff\ P\ (p \otimes_P q)\ i) \otimes_S s\ (\hat{\ })_S\ i)$
      by (*simp cong*: *S.finsum-cong*
        add: *deg-aboveD S.finsum-Un-disjoint ivl-disj-int-one Pi-def*
        del: *coeff-mult*)

also from R **have** ... =
$(\bigoplus_S i \in \{..deg\ R\ p + deg\ R\ q\}.\ h\ (coeff\ P\ (p \otimes_P q)\ i) \otimes_S s\ (\hat{\ })_S\ i)$
by (*simp only*: *ivl-disj-un-one deg-mult-cring*)
also from $R\ S$ **have** ... =
$(\bigoplus_S i \in \{..deg\ R\ p + deg\ R\ q\}.$
$\bigoplus_S k \in \{..i\}.$
$h\ (coeff\ P\ p\ k) \otimes_S h\ (coeff\ P\ q\ (i - k)) \otimes_S$
$(s\ (\hat{\ })_S\ k \otimes_S s\ (\hat{\ })_S\ (i - k)))$
by (*simp cong*: *S.finsum-cong add*: *S.nat-pow-mult Pi-def*
S.m-ac S.finsum-rdistr)
also from $R\ S$ **have** ... =
$(\bigoplus_S i{\in}\{..deg\ R\ p\}.\ h\ (coeff\ P\ p\ i) \otimes_S s\ (\hat{\ })_S\ i) \otimes_S$
$(\bigoplus_S i{\in}\{..deg\ R\ q\}.\ h\ (coeff\ P\ q\ i) \otimes_S s\ (\hat{\ })_S\ i)$
by (*simp add*: *S.cauchy-product* [*THEN sym*] *bound.intro deg-aboveD S.m-ac*
Pi-def)
finally show
$(\bigoplus_S i \in \{..deg\ R\ (p \otimes_P q)\}.\ h\ (coeff\ P\ (p \otimes_P q)\ i) \otimes_S s\ (\hat{\ })_S\ i) =$
$(\bigoplus_S i \in \{..deg\ R\ p\}.\ h\ (coeff\ P\ p\ i) \otimes_S s\ (\hat{\ })_S\ i) \otimes_S$
$(\bigoplus_S i \in \{..deg\ R\ q\}.\ h\ (coeff\ P\ q\ i) \otimes_S s\ (\hat{\ })_S\ i)$ **.**
qed
next
fix $p\ q$
assume R: $p \in carrier\ P\ q \in carrier\ P$
then show $eval\ R\ S\ h\ s\ (p \oplus_P q) = eval\ R\ S\ h\ s\ p \oplus_S eval\ R\ S\ h\ s\ q$
proof (*simp only*: *eval-on-carrier P.a-closed*)
from $S\ R$ **have**
$(\bigoplus_S i{\in}\{..deg\ R\ (p \oplus_P q)\}.\ h\ (coeff\ P\ (p \oplus_P q)\ i) \otimes_S s\ (\hat{\ })_S\ i) =$
$(\bigoplus_S i{\in}\{..deg\ R\ (p \oplus_P q)\} \cup \{deg\ R\ (p \oplus_P q){<}..max\ (deg\ R\ p)\ (deg\ R\ q)\}.$
$h\ (coeff\ P\ (p \oplus_P q)\ i) \otimes_S s\ (\hat{\ })_S\ i)$
by (*simp cong*: *S.finsum-cong*
add: *deg-aboveD S.finsum-Un-disjoint ivl-disj-int-one Pi-def*
del: *coeff-add*)
also from R **have** ... =
$(\bigoplus_S i \in \{..max\ (deg\ R\ p)\ (deg\ R\ q)\}.$
$h\ (coeff\ P\ (p \oplus_P q)\ i) \otimes_S s\ (\hat{\ })_S\ i)$
by (*simp add*: *ivl-disj-un-one*)
also from $R\ S$ **have** ... =
$(\bigoplus_S i{\in}\{..max\ (deg\ R\ p)\ (deg\ R\ q)\}.\ h\ (coeff\ P\ p\ i) \otimes_S s\ (\hat{\ })_S\ i) \oplus_S$
$(\bigoplus_S i{\in}\{..max\ (deg\ R\ p)\ (deg\ R\ q)\}.\ h\ (coeff\ P\ q\ i) \otimes_S s\ (\hat{\ })_S\ i)$
by (*simp cong*: *S.finsum-cong*
add: *S.l-distr deg-aboveD ivl-disj-int-one Pi-def*)
also have ... =
$(\bigoplus_S i \in \{..deg\ R\ p\} \cup \{deg\ R\ p{<}..max\ (deg\ R\ p)\ (deg\ R\ q)\}.$
$h\ (coeff\ P\ p\ i) \otimes_S s\ (\hat{\ })_S\ i) \oplus_S$
$(\bigoplus_S i \in \{..deg\ R\ q\} \cup \{deg\ R\ q{<}..max\ (deg\ R\ p)\ (deg\ R\ q)\}.$
$h\ (coeff\ P\ q\ i) \otimes_S s\ (\hat{\ })_S\ i)$
by (*simp only*: *ivl-disj-un-one le-maxI1 le-maxI2*)
also from $R\ S$ **have** ... =
$(\bigoplus_S i \in \{..deg\ R\ p\}.\ h\ (coeff\ P\ p\ i) \otimes_S s\ (\hat{\ })_S\ i) \oplus_S$

$(\bigoplus_S\ i \in \{..deg\ R\ q\}.\ h\ (coeff\ P\ q\ i) \otimes_S s\ (\hat{}\,)_S\ i)$
by (*simp cong*: *S.finsum-cong*
add: *deg-aboveD S.finsum-Un-disjoint ivl-disj-int-one Pi-def*)
finally show
$(\bigoplus_S i \in \{..deg\ R\ (p \oplus_P q)\}.\ h\ (coeff\ P\ (p \oplus_P q)\ i) \otimes_S s\ (\hat{}\,)_S\ i) =$
$(\bigoplus_S i \in \{..deg\ R\ p\}.\ h\ (coeff\ P\ p\ i) \otimes_S s\ (\hat{}\,)_S\ i) \oplus_S$
$(\bigoplus_S i \in \{..deg\ R\ q\}.\ h\ (coeff\ P\ q\ i) \otimes_S s\ (\hat{}\,)_S\ i)$ **.**
qed
next
show *eval R S h s* $\mathbf{1}_P = \mathbf{1}_S$
by (*simp only*: *eval-on-carrier UP-one-closed*) *simp*
qed

Interpretation of ring homomorphism lemmas.

interpretation *UP-univ-prop* < *ring-hom-cring P S Eval*
apply (*unfold Eval-def*)
apply *intro-locales*
apply (*rule ring-hom-cring.axioms*)
apply (*rule ring-hom-cring.intro*)
apply *unfold-locales*
apply (*rule eval-ring-hom*)
apply *rule*
done

Further properties of the evaluation homomorphism.

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

lemma (**in** *UP-pre-univ-prop*) *eval-const*:
$[|\ s \in carrier\ S;\ r \in carrier\ R\ |] ==> eval\ R\ S\ h\ s\ (monom\ P\ r\ 0) = h\ r$
by (*simp only*: *eval-on-carrier monom-closed*) *simp*

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 (*simp only*: *eval-on-carrier monom-closed R.one-closed*)
from *S* **have**
$(\bigoplus_S\ i{\in}\{..deg\ R\ (monom\ P\ \mathbf{1}\ 1)\}.\ h\ (coeff\ P\ (monom\ P\ \mathbf{1}\ 1)\ i) \otimes_S s\ (\hat{}\,)_S\ i)$
$=$
$(\bigoplus_S\ i{\in}\{..deg\ R\ (monom\ P\ \mathbf{1}\ 1)\} \cup \{deg\ R\ (monom\ P\ \mathbf{1}\ 1){<}..1\}.$
$h\ (coeff\ P\ (monom\ P\ \mathbf{1}\ 1)\ i) \otimes_S s\ (\hat{}\,)_S\ i)$
by (*simp cong*: *S.finsum-cong del*: *coeff-monom*
add: *deg-aboveD S.finsum-Un-disjoint ivl-disj-int-one Pi-def*)
also have ... =
$(\bigoplus_S\ i \in \{..1\}.\ h\ (coeff\ P\ (monom\ P\ \mathbf{1}\ 1)\ i) \otimes_S s\ (\hat{}\,)_S\ i)$
by (*simp only*: *ivl-disj-un-one deg-monom-le R.one-closed*)
also have ... = *s*

proof (*cases* $s = \mathbf{0}_S$)
  case *True* **then show** *?thesis* **by** (*simp add*: *Pi-def*)
next
  case *False* **then show** *?thesis* **by** (*simp add*: *S Pi-def*)
qed
finally show ($\bigoplus_S i \in \{..deg\ R\ (monom\ P\ \mathbf{1}\ 1)\}.$
  $h\ (coeff\ P\ (monom\ P\ \mathbf{1}\ 1)\ i) \otimes_S s\ (\hat{\ })_S\ i) = s$ **.**
qed

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 (*induct m*)
  case *0* **from** *R* **show** *?case* **by** *simp*
next
  case *Suc* **with** *R* **show** *?case*
    by (*simp del*: *monom-mult add*: *monom-mult* [*THEN sym*] *add-commute*)
qed

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

lemma (**in** *UP-univ-prop*) *Eval-monom*:
  $r \in carrier\ R \Longrightarrow Eval\ (monom\ P\ r\ n) = h\ r \otimes_S s\ (\hat{\ })_S\ n$
proof −
  assume *R*: $r \in carrier\ R$
  from *R* **have** $Eval\ (monom\ P\ r\ n) = Eval\ (monom\ P\ r\ 0 \otimes_P (monom\ P\ \mathbf{1}\ 1)$
$(\hat{\ })_P\ n)$
    by (*simp del*: *monom-mult add*: *monom-mult* [*THEN sym*] *monom-pow*)
  also
  from *R* *eval-monom1* [**where** $s = s$, *folded Eval-def*]
  have $... = h\ r \otimes_S s\ (\hat{\ })_S\ n$
    by (*simp add*: *eval-const* [**where** $s = s$, *folded Eval-def*])
  finally show *?thesis* **.**
qed

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 −
  interpret *UP-univ-prop* [*R S h P s* -]
    using ⟨*UP-pre-univ-prop R S h*⟩ *P-def R S*
    by (*auto intro*: *UP-univ-prop.intro UP-univ-prop-axioms.intro*)
  from *R*
  show *?thesis* **by** (*rule Eval-monom*)
qed

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 –
  assume *R*: $r \in$ *carrier R* **and** *P*: $p \in$ *carrier P*
  then show *?thesis*
    by (*simp add*: *monom-mult-is-smult* [*THEN sym*]
      eval-const [**where** $s = s$, *folded Eval-def*])
qed

lemma *ring-hom-cringI*:
  assumes *cring R*
    and *cring S*
    and $h \in$ *ring-hom R S*
  shows *ring-hom-cring R S h*
  by (*fast intro*: *ring-hom-cring.intro ring-hom-cring-axioms.intro*
    cring.axioms prems)

lemma (**in** *UP-pre-univ-prop*) *UP-hom-unique*:
  includes *ring-hom-cring P S Phi*
  assumes *Phi*: *Phi* (*monom P* **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* **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 –
  have *Phi p* =
      Phi ($\bigoplus_P$ $i \in \{..deg\ R\ p\}$. *monom P* (*coeff P p i*) *0* $\otimes_P$ *monom P* **1** *1* $(\hat{\ })_P$
i)
   by (*simp add*: *up-repr P monom-mult* [*THEN sym*] *monom-pow del*: *monom-mult*)
  also
  have ... =
      Psi ($\bigoplus_P$ $i\in\{..deg\ R\ p\}$. *monom P* (*coeff P p i*) *0* $\otimes_P$ *monom P* **1** *1* $(\hat{\ })_P$ *i*)
    by (*simp add*: *Phi Psi P Pi-def comp-def*)
  also have ... = *Psi p*
   by (*simp add*: *up-repr P monom-mult* [*THEN sym*] *monom-pow del*: *monom-mult*)
  finally show *?thesis* **.**
qed

lemma (**in** *UP-pre-univ-prop*) *ring-homD*:
  assumes *Phi*: $Phi \in$ *ring-hom P S*
  shows *ring-hom-cring P S Phi*
proof (*rule ring-hom-cring.intro*)
  show *ring-hom-cring-axioms P S Phi*
  by (*rule ring-hom-cring-axioms.intro*) (*rule Phi*)
qed *unfold-locales*

theorem (**in** *UP-pre-univ-prop*) *UP-universal-property*:
  assumes *S*: $s \in$ *carrier S*

shows *EX*! *Phi*. *Phi* ∈ *ring-hom P S* ∩ *extensional* (*carrier P*) &
Phi (*monom P* **1** *1*) = *s* &
(*ALL r* : *carrier R*. *Phi* (*monom P r 0*) = *h r*)
using *S eval-monom1*
apply (*auto intro*: *eval-ring-hom eval-const eval-extensional*)
apply (*rule extensionalityI*)
apply (*auto intro*: *UP-hom-unique ring-homD*)
done

11.9 Sample Application of Evaluation Homomorphism

lemma *UP-pre-univ-propI*:
assumes *cring R*
and *cring S*
and *h* ∈ *ring-hom R S*
shows *UP-pre-univ-prop R S h*
using *assms*
by (*auto intro*!: *UP-pre-univ-prop.intro ring-hom-cring.intro*
ring-hom-cring-axioms.intro UP-cring.intro)

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

lemma *INTEG-cring*:
cring INTEG
by (*unfold INTEG-def*) (*auto intro*!: *cringI abelian-groupI comm-monoidI*
zadd-zminus-inverse2 zadd-zmult-distrib)

lemma *INTEG-id-eval*:
UP-pre-univ-prop INTEG INTEG id
by (*fast intro*: *UP-pre-univ-propI INTEG-cring id-ring-hom*)

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*]
apply *simp*
using *INTEG-id-eval*
apply *simp*
done

lemma *INTEG-closed* [*intro*, *simp*]:
z ∈ *carrier INTEG*
by (*unfold INTEG-def*) *simp*

lemma *INTEG-mult* [*simp*]:
mult INTEG z w = *z* ∗ *w*
by (*unfold INTEG-def*) *simp*

lemma *INTEG-pow* [*simp*]:
pow INTEG z n = *z* ^ *n*
by (*induct n*) (*simp-all add*: *INTEG-def nat-pow-def*)

lemma *eval INTEG INTEG id 10* (*monom* (*UP INTEG*) *5 2*) = *500*
by (*simp add*: *INTEG.eval-monom*)

end

theory *AbelCoset*
imports *Coset Ring*
begin

12 More Lifting from Groups to Abelian Groups

12.1 Definitions

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

hide *const Plus*

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

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

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

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

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

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

constdefs

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

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

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

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

lemma *a-r-coset-def'*:
includes *struct G*
shows $H +> a \equiv \bigcup h\in H.\ \{h \oplus a\}$
unfolding *a-r-coset-defs*
by *simp*

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

lemma *a-l-coset-def'*:
includes *struct G*
shows $a <+ H \equiv \bigcup h\in H.\ \{a \oplus h\}$
unfolding *a-l-coset-defs*
by *simp*

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

lemma *A-RCOSETS-def'*:
includes *struct G*
shows $a\text{-}rcosets\ H \equiv \bigcup a\in carrier\ G.\ \{H +> a\}$
unfolding *A-RCOSETS-defs*
by (*fold a-r-coset-def*, *simp*)

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

lemma *set-add-def'*:

includes *struct G*
shows $H <+> K \equiv \bigcup h{\in}H.\ \bigcup k{\in}K.\ \{h \oplus k\}$
unfolding *set-add-defs*
by *simp*

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

lemma *A-SET-INV-def'*:
includes *struct G*
shows $a\text{-}set\text{-}inv\ H \equiv \bigcup h{\in}H.\ \{\ominus\ h\}$
unfolding *A-SET-INV-defs*
by (*fold a-inv-def*)

12.2 Cosets

lemma (**in** *abelian-group*) *a-coset-add-assoc*:
$[|\ M \subseteq carrier\ G;\ g \in carrier\ G;\ h \in carrier\ G\ |]$
$==> (M +> g) +> h = M +> (g \oplus h)$
by (*rule group.coset-mult-assoc* [*OF a-group*,
folded a-r-coset-def, *simplified monoid-record-simps*])

lemma (**in** *abelian-group*) *a-coset-add-zero* [*simp*]:
$M \subseteq carrier\ G ==> M +> \mathbf{0} = M$
by (*rule group.coset-mult-one* [*OF a-group*,
folded a-r-coset-def, *simplified monoid-record-simps*])

lemma (**in** *abelian-group*) *a-coset-add-inv1*:
$[|\ M +> (x \oplus (\ominus\ y)) = M;\ \ x \in carrier\ G\ ;\ y \in carrier\ G;$
$M \subseteq carrier\ G\ |] ==> M +> x = M +> y$
by (*rule group.coset-mult-inv1* [*OF a-group*,
folded a-r-coset-def a-inv-def, *simplified monoid-record-simps*])

lemma (**in** *abelian-group*) *a-coset-add-inv2*:
$[|\ M +> x = M +> y;\ \ x \in carrier\ G;\ \ y \in carrier\ G;\ \ M \subseteq carrier\ G\ |]$
$==> M +> (x \oplus (\ominus\ y)) = M$
by (*rule group.coset-mult-inv2* [*OF a-group*,
folded a-r-coset-def a-inv-def, *simplified monoid-record-simps*])

lemma (**in** *abelian-group*) *a-coset-join1*:
$[|\ H +> x = H;\ \ x \in carrier\ G;\ \ subgroup\ H\ (|carrier = carrier\ G,\ mult = add\ G,\ one = zero\ G|)\ |] ==> x \in H$
by (*rule group.coset-join1* [*OF a-group*,
folded a-r-coset-def, *simplified monoid-record-simps*])

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

folded a-r-coset-def, *simplified monoid-record-simps*])

lemma (**in** *abelian-group*) *a-repr-independence*:
$[\![y \in H +> x;\ \ x \in carrier\ G;\ subgroup\ H\ (\!|carrier = carrier\ G,\ mult = add\ G,\ one = zero\ G|\!)\ ]\!] \Longrightarrow H +> x = H +> y$
by (*rule group.repr-independence* [*OF a-group*,
folded a-r-coset-def, *simplified monoid-record-simps*])

lemma (**in** *abelian-group*) *a-coset-join2*:
$[\![x \in carrier\ G;\ \ subgroup\ H\ (\!|carrier = carrier\ G,\ mult = add\ G,\ one = zero\ G|\!);\ x{\in}H]\!] \Longrightarrow H +> x = H$
by (*rule group.coset-join2* [*OF a-group*,
folded a-r-coset-def, *simplified monoid-record-simps*])

lemma (**in** *abelian-monoid*) *a-r-coset-subset-G*:
$[|\ H \subseteq carrier\ G;\ x \in carrier\ G\ |] ==> H +> x \subseteq carrier\ G$
by (*rule monoid.r-coset-subset-G* [*OF a-monoid*,
folded a-r-coset-def, *simplified monoid-record-simps*])

lemma (**in** *abelian-group*) *a-rcosI*:
$[|\ h \in H;\ H \subseteq carrier\ G;\ x \in carrier\ G|] ==> h \oplus x \in H +> x$
by (*rule group.rcosI* [*OF a-group*,
folded a-r-coset-def, *simplified monoid-record-simps*])

lemma (**in** *abelian-group*) *a-rcosetsI*:
$[\![H \subseteq carrier\ G;\ x \in carrier\ G]\!] \Longrightarrow H +> x \in a\text{-}rcosets\ H$
by (*rule group.rcosetsI* [*OF a-group*,
folded a-r-coset-def A-RCOSETS-def, *simplified monoid-record-simps*])

Really needed?

lemma (**in** *abelian-group*) *a-transpose-inv*:
$[|\ x \oplus y = z;\ \ x \in carrier\ G;\ \ y \in carrier\ G;\ \ z \in carrier\ G\ |]$
$==> (\ominus\ x) \oplus z = y$
by (*rule group.transpose-inv* [*OF a-group*,
folded a-r-coset-def a-inv-def, *simplified monoid-record-simps*])

12.3 Subgroups

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

lemma (**in** *additive-subgroup*) *is-additive-subgroup*:
shows *additive-subgroup H G*
by *fact*

lemma *additive-subgroupI*:
includes *struct G*
assumes *a-subgroup*: $subgroup\ H\ (\!|carrier = carrier\ G,\ mult = add\ G,\ one =$

zero G⦈
  shows *additive-subgroup H G*
by (*rule additive-subgroup.intro*) (*rule a-subgroup*)

lemma (**in** *additive-subgroup*) *a-subset*:
  $H \subseteq carrier\ G$
by (*rule subgroup.subset*[*OF a-subgroup*,
  simplified monoid-record-simps])

lemma (**in** *additive-subgroup*) *a-closed* [*intro*, *simp*]:
  $[\![x \in H;\ y \in H]\!] \Longrightarrow x \oplus y \in H$
by (*rule subgroup.m-closed*[*OF a-subgroup*,
  simplified monoid-record-simps])

lemma (**in** *additive-subgroup*) *zero-closed* [*simp*]:
  $\mathbf{0} \in H$
by (*rule subgroup.one-closed*[*OF a-subgroup*,
  simplified monoid-record-simps])

lemma (**in** *additive-subgroup*) *a-inv-closed* [*intro*,*simp*]:
  $x \in H \Longrightarrow \ominus\ x \in H$
by (*rule subgroup.m-inv-closed*[*OF a-subgroup*,
  folded a-inv-def, *simplified monoid-record-simps*])

12.4 Normal additive subgroups

12.4.1 Definition of *abelian-subgroup*

Every subgroup of an *abelian-group* is normal

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

lemma (**in** *abelian-subgroup*) *is-abelian-subgroup*:
  shows *abelian-subgroup H G*
by *fact*

lemma *abelian-subgroupI*:
  assumes *a-normal*: *normal H* (|*carrier* = *carrier G*, *mult* = *add G*, *one* = *zero G*|)
    and *a-comm*: !!*x y*. [| $x \in carrier\ G$; $y \in carrier\ G$ |] ==> $x \oplus_G y = y \oplus_G x$
  shows *abelian-subgroup H G*
proof −
  interpret *normal* [*H* (|*carrier* = *carrier G*, *mult* = *add G*, *one* = *zero G*|)]
  by (*rule a-normal*)

  show *abelian-subgroup H G*
  by (*unfold-locales*, *simp add*: *a-comm*)

qed

lemma *abelian-subgroupI2*:
 includes *struct G*
 assumes *a-comm-group*: *comm-group* (|*carrier* = *carrier G*, *mult* = *add G*, *one* = *zero G*|)
 and *a-subgroup*: *subgroup H* (|*carrier* = *carrier G*, *mult* = *add G*, *one* = *zero G*|)
 shows *abelian-subgroup H G*
proof −
 interpret *comm-group* [(|*carrier* = *carrier G*, *mult* = *add G*, *one* = *zero G*|)]
 by (*rule a-comm-group*)
 interpret *subgroup* [*H* (|*carrier* = *carrier G*, *mult* = *add G*, *one* = *zero G*|)]
 by (*rule a-subgroup*)

 show *abelian-subgroup H G*
 apply *unfold-locales*
 proof (*simp add*: *r-coset-def l-coset-def*, *clarsimp*)
 fix *x*
 assume *xcarr*: $x \in carrier\ G$
 from *a-subgroup*
 have *Hcarr*: $H \subseteq carrier\ G$ **by** (*unfold subgroup-def*, *simp*)
 from *xcarr Hcarr*
 show $(\bigcup h{\in}H.\ \{h \oplus_G x\}) = (\bigcup h{\in}H.\ \{x \oplus_G h\})$
 using *m-comm*[*simplified*]
 by *fast*
 qed
qed

lemma *abelian-subgroupI3*:
 includes *struct G*
 assumes *asg*: *additive-subgroup H G*
 and *ag*: *abelian-group G*
 shows *abelian-subgroup H G*
apply (*rule abelian-subgroupI2*)
apply (*rule abelian-group.a-comm-group*[*OF ag*])
apply (*rule additive-subgroup.a-subgroup*[*OF asg*])
done

lemma (**in** *abelian-subgroup*) *a-coset-eq*:
 $(\forall x \in carrier\ G.\ H +> x = x <+ H)$
by (*rule normal.coset-eq*[*OF a-normal*,
 folded a-r-coset-def a-l-coset-def, *simplified monoid-record-simps*])

lemma (**in** *abelian-subgroup*) *a-inv-op-closed1*:
 shows $[\![x \in carrier\ G;\ h \in H]\!] \Longrightarrow (\ominus\ x) \oplus h \oplus x \in H$
by (*rule normal.inv-op-closed1* [*OF a-normal*,
 folded a-inv-def, *simplified monoid-record-simps*])

lemma (**in** *abelian-subgroup*) *a-inv-op-closed2*:
shows $[\![x \in \mathit{carrier}\ G;\ h \in H]\!] \Longrightarrow x \oplus h \oplus (\ominus\ x) \in H$
by (*rule normal.inv-op-closed2* [*OF a-normal*,
folded a-inv-def, *simplified monoid-record-simps*])

Alternative characterization of normal subgroups

lemma (**in** *abelian-group*) *a-normal-inv-iff*:
$(N \lhd (\!|\mathit{carrier} = \mathit{carrier}\ G,\ \mathit{mult} = \mathit{add}\ G,\ \mathit{one} = \mathit{zero}\ G|\!)) =$
$(\mathit{subgroup}\ N\ (\!|\mathit{carrier} = \mathit{carrier}\ G,\ \mathit{mult} = \mathit{add}\ G,\ \mathit{one} = \mathit{zero}\ G|\!)\ \&\ (\forall\, x \in \mathit{carrier}\ G.\ \forall\, h \in N.\ x \oplus h \oplus (\ominus\ x) \in N))$
(**is** - = *?rhs*)
by (*rule group.normal-inv-iff* [*OF a-group*,
folded a-inv-def, *simplified monoid-record-simps*])

lemma (**in** *abelian-group*) *a-lcos-m-assoc*:
$[|\ M \subseteq \mathit{carrier}\ G;\ g \in \mathit{carrier}\ G;\ h \in \mathit{carrier}\ G\ |]$
$==> g <\!+ (h <\!+ M) = (g \oplus h) <\!+ M$
by (*rule group.lcos-m-assoc* [*OF a-group*,
folded a-l-coset-def, *simplified monoid-record-simps*])

lemma (**in** *abelian-group*) *a-lcos-mult-one*:
$M \subseteq \mathit{carrier}\ G ==> \mathbf{0} <\!+ M = M$
by (*rule group.lcos-mult-one* [*OF a-group*,
folded a-l-coset-def, *simplified monoid-record-simps*])

lemma (**in** *abelian-group*) *a-l-coset-subset-G*:
$[|\ H \subseteq \mathit{carrier}\ G;\ x \in \mathit{carrier}\ G\ |] ==> x <\!+ H \subseteq \mathit{carrier}\ G$
by (*rule group.l-coset-subset-G* [*OF a-group*,
folded a-l-coset-def, *simplified monoid-record-simps*])

lemma (**in** *abelian-group*) *a-l-coset-swap*:
$[\![y \in x <\!+ H;\ \ x \in \mathit{carrier}\ G;\ \ \mathit{subgroup}\ H\ (\!|\mathit{carrier} = \mathit{carrier}\ G,\ \mathit{mult} = \mathit{add}\ G,\ \mathit{one} = \mathit{zero}\ G|\!)]\!] \Longrightarrow x \in y <\!+ H$
by (*rule group.l-coset-swap* [*OF a-group*,
folded a-l-coset-def, *simplified monoid-record-simps*])

lemma (**in** *abelian-group*) *a-l-coset-carrier*:
$[|\ y \in x <\!+ H;\ \ x \in \mathit{carrier}\ G;\ \ \mathit{subgroup}\ H\ (\!|\mathit{carrier} = \mathit{carrier}\ G,\ \mathit{mult} = \mathit{add}\ G,\ \mathit{one} = \mathit{zero}\ G|\!)\ |] ==> y \in \mathit{carrier}\ G$
by (*rule group.l-coset-carrier* [*OF a-group*,
folded a-l-coset-def, *simplified monoid-record-simps*])

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

folded a-l-coset-def, *simplified monoid-record-simps*])
apply (*rule y*)
apply (*rule x*)
apply (*rule sb*)
done

lemma (**in** *abelian-group*) *a-l-repr-independence*:
assumes *y*: $y \in x <+ H$ **and** *x*: $x \in$ *carrier G* **and** *sb*: *subgroup H* $(\!|$*carrier* = *carrier G*, *mult* = *add G*, *one* = *zero G*$|\!)$
shows $x <+ H = y <+ H$
apply (*rule group.l-repr-independence* [*OF a-group*,
folded a-l-coset-def, *simplified monoid-record-simps*])
apply (*rule y*)
apply (*rule x*)
apply (*rule sb*)
done

lemma (**in** *abelian-group*) *setadd-subset-G*:
$[\![H \subseteq carrier\ G;\ K \subseteq carrier\ G]\!] \Longrightarrow H <+> K \subseteq carrier\ G$
by (*rule group.setmult-subset-G* [*OF a-group*,
folded set-add-def, *simplified monoid-record-simps*])

lemma (**in** *abelian-group*) *subgroup-add-id*: *subgroup H* $(\!|$*carrier* = *carrier G*, *mult* = *add G*, *one* = *zero G*$|\!) \Longrightarrow H <+> H = H$
by (*rule group.subgroup-mult-id* [*OF a-group*,
folded set-add-def, *simplified monoid-record-simps*])

lemma (**in** *abelian-subgroup*) *a-rcos-inv*:
assumes *x*: $x \in carrier\ G$
shows $a\text{-}set\text{-}inv\ (H +> x) = H +> (\ominus\ x)$
by (*rule normal.rcos-inv* [*OF a-normal*,
folded a-r-coset-def a-inv-def A-SET-INV-def, *simplified monoid-record-simps*]) (*rule x*)

lemma (**in** *abelian-group*) *a-setmult-rcos-assoc*:
$[\![H \subseteq carrier\ G;\ K \subseteq carrier\ G;\ x \in carrier\ G]\!]$
$\Longrightarrow H <+> (K +> x) = (H <+> K) +> x$
by (*rule group.setmult-rcos-assoc* [*OF a-group*,
folded set-add-def a-r-coset-def, *simplified monoid-record-simps*])

lemma (**in** *abelian-group*) *a-rcos-assoc-lcos*:
$[\![H \subseteq carrier\ G;\ K \subseteq carrier\ G;\ x \in carrier\ G]\!]$
$\Longrightarrow (H +> x) <+> K = H <+> (x <+ K)$
by (*rule group.rcos-assoc-lcos* [*OF a-group*,
folded set-add-def a-r-coset-def a-l-coset-def, *simplified monoid-record-simps*])

lemma (**in** *abelian-subgroup*) *a-rcos-sum*:
$[\![x \in carrier\ G;\ y \in carrier\ G]\!]$
$\Longrightarrow (H +> x) <+> (H +> y) = H +> (x \oplus y)$

by (*rule normal.rcos-sum* [*OF a-normal*,
folded set-add-def a-r-coset-def, *simplified monoid-record-simps*])

lemma (**in** *abelian-subgroup*) *rcosets-add-eq*:
$M \in$ *a-rcosets* $H \Longrightarrow H$ <+> $M = M$
— generalizes *subgroup-mult-id*
by (*rule normal.rcosets-mult-eq* [*OF a-normal*,
folded set-add-def A-RCOSETS-def, *simplified monoid-record-simps*])

12.5 Congruence Relation

lemma (**in** *abelian-subgroup*) *a-equiv-rcong*:
shows *equiv* (*carrier G*) (*racong H*)
by (*rule subgroup.equiv-rcong* [*OF a-subgroup a-group*,
folded a-r-congruent-def, *simplified monoid-record-simps*])

lemma (**in** *abelian-subgroup*) *a-l-coset-eq-rcong*:
assumes *a*: $a \in$ *carrier G*
shows a <+ H = *racong H* ‘‘ $\{a\}$
by (*rule subgroup.l-coset-eq-rcong* [*OF a-subgroup a-group*,
folded a-r-congruent-def a-l-coset-def, *simplified monoid-record-simps*]) (*rule a*)

lemma (**in** *abelian-subgroup*) *a-rcos-equation*:
shows
$[\![ha \oplus a = h \oplus b;\ a \in carrier\ G;\ \ b \in carrier\ G;$
$h \in H;\ \ ha \in H;\ \ hb \in H]\!]$
$\Longrightarrow hb \oplus a \in (\bigcup h{\in}H.\ \{h \oplus b\})$
by (*rule group.rcos-equation* [*OF a-group a-subgroup*,
folded a-r-congruent-def a-l-coset-def, *simplified monoid-record-simps*])

lemma (**in** *abelian-subgroup*) *a-rcos-disjoint*:
shows $[\![a \in$ *a-rcosets* $H;\ b \in$ *a-rcosets* $H;\ a{\neq}b]\!] \Longrightarrow a \cap b = \{\}$
by (*rule group.rcos-disjoint* [*OF a-group a-subgroup*,
folded A-RCOSETS-def, *simplified monoid-record-simps*])

lemma (**in** *abelian-subgroup*) *a-rcos-self*:
shows $x \in$ *carrier G* $\Longrightarrow x \in H$ +> x
by (*rule group.rcos-self* [*OF a-group a-subgroup*,
folded a-r-coset-def, *simplified monoid-record-simps*])

lemma (**in** *abelian-subgroup*) *a-rcosets-part-G*:
shows $\bigcup$(*a-rcosets H*) = *carrier G*
by (*rule group.rcosets-part-G* [*OF a-group a-subgroup*,
folded A-RCOSETS-def, *simplified monoid-record-simps*])

lemma (**in** *abelian-subgroup*) *a-cosets-finite*:
$[\![c \in$ *a-rcosets* $H;\ \ H \subseteq$ *carrier G*; *finite* (*carrier G*)$]\!] \Longrightarrow$ *finite c*
by (*rule group.cosets-finite* [*OF a-group*,
folded A-RCOSETS-def, *simplified monoid-record-simps*])

lemma (**in** *abelian-group*) *a-card-cosets-equal*:
$[\![c \in \textit{a-rcosets } H;\ \ H \subseteq \textit{carrier } G;\ \textit{finite}(\textit{carrier } G)]\!]$
$\Longrightarrow \textit{card } c = \textit{card } H$
by (*rule group.card-cosets-equal* [*OF a-group*,
folded A-RCOSETS-def, *simplified monoid-record-simps*])

lemma (**in** *abelian-group*) *rcosets-subset-PowG*:
$\textit{additive-subgroup } H\ G \Longrightarrow \textit{a-rcosets } H \subseteq \textit{Pow}(\textit{carrier } G)$
by (*rule group.rcosets-subset-PowG* [*OF a-group*,
folded A-RCOSETS-def, *simplified monoid-record-simps*],
rule additive-subgroup.a-subgroup)

theorem (**in** *abelian-group*) *a-lagrange*:
$[\![\textit{finite}(\textit{carrier } G);\ \textit{additive-subgroup } H\ G]\!]$
$\Longrightarrow \textit{card}(\textit{a-rcosets } H) * \textit{card}(H) = \textit{order}(G)$
by (*rule group.lagrange* [*OF a-group*,
folded A-RCOSETS-def, *simplified monoid-record-simps order-def*, *folded order-def*])
(*fast intro*!: *additive-subgroup.a-subgroup*)+

12.6 Factorization

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

lemma *A-FactGroup-def'*:
includes *struct G*
shows $G\ \textit{A-Mod}\ H \equiv (\!|\textit{carrier} = \textit{a-rcosets}_G\ H,\ \textit{mult} = \textit{set-add } G,\ \textit{one} = H|\!)$
unfolding *A-FactGroup-defs*
by (*fold A-RCOSETS-def set-add-def*)

lemma (**in** *abelian-subgroup*) *a-setmult-closed*:
$[\![K1 \in \textit{a-rcosets } H;\ K2 \in \textit{a-rcosets } H]\!] \Longrightarrow K1 <+> K2 \in \textit{a-rcosets } H$
by (*rule normal.setmult-closed* [*OF a-normal*,
folded A-RCOSETS-def set-add-def, *simplified monoid-record-simps*])

lemma (**in** *abelian-subgroup*) *a-setinv-closed*:
$K \in \textit{a-rcosets } H \Longrightarrow \textit{a-set-inv } K \in \textit{a-rcosets } H$
by (*rule normal.setinv-closed* [*OF a-normal*,
folded A-RCOSETS-def A-SET-INV-def, *simplified monoid-record-simps*])

lemma (**in** *abelian-subgroup*) *a-rcosets-assoc*:
$[\![M1 \in \textit{a-rcosets } H;\ M2 \in \textit{a-rcosets } H;\ M3 \in \textit{a-rcosets } H]\!]$
$\Longrightarrow M1 <+> M2 <+> M3 = M1 <+> (M2 <+> M3)$
by (*rule normal.rcosets-assoc* [*OF a-normal*,
folded A-RCOSETS-def set-add-def, *simplified monoid-record-simps*])

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

by (*rule subgroup.subgroup-in-rcosets* [*OF a-subgroup a-group*,
folded A-RCOSETS-def, *simplified monoid-record-simps*])

lemma (**in** *abelian-subgroup*) *a-rcosets-inv-mult-group-eq*:
$M \in$ *a-rcosets* $H \Longrightarrow$ *a-set-inv* M <+> $M = H$
by (*rule normal.rcosets-inv-mult-group-eq* [*OF a-normal*,
folded A-RCOSETS-def A-SET-INV-def set-add-def, *simplified monoid-record-simps*])

theorem (**in** *abelian-subgroup*) *a-factorgroup-is-group*:
group (G *A-Mod* H)
by (*rule normal.factorgroup-is-group* [*OF a-normal*,
folded A-FactGroup-def, *simplified monoid-record-simps*])

Since the Factorization is based on an *abelian* subgroup, is results in a commutative group

theorem (**in** *abelian-subgroup*) *a-factorgroup-is-comm-group*:
comm-group (G *A-Mod* H)
apply (*intro comm-group.intro comm-monoid.intro*) **prefer** *3*
apply (*rule a-factorgroup-is-group*)
apply (*rule group.axioms*[*OF a-factorgroup-is-group*])
apply (*rule comm-monoid-axioms.intro*)
apply (*unfold A-FactGroup-def FactGroup-def RCOSETS-def*, *fold set-add-def a-r-coset-def*, *clarsimp*)
apply (*simp add*: *a-rcos-sum a-comm*)
done

lemma *add-A-FactGroup* [*simp*]: $X \otimes_{(G\ A\text{-}Mod\ H)} X' = X$ <+>$_G$ X'
by (*simp add*: *A-FactGroup-def set-add-def*)

lemma (**in** *abelian-subgroup*) *a-inv-FactGroup*:
$X \in$ *carrier* (G *A-Mod* H) $\Longrightarrow$ $inv_{G\ A\text{-}Mod\ H}\ X =$ *a-set-inv* X
by (*rule normal.inv-FactGroup* [*OF a-normal*,
folded A-FactGroup-def A-SET-INV-def, *simplified monoid-record-simps*])

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

lemma (**in** *abelian-subgroup*) *a-r-coset-hom-A-Mod*:
(λa. H +> a) $\in$ *hom* (|*carrier* = *carrier* G, *mult* = *add* G, *one* = *zero* G|) (G *A-Mod* H)
by (*rule normal.r-coset-hom-Mod* [*OF a-normal*,
folded A-FactGroup-def a-r-coset-def, *simplified monoid-record-simps*])

The isomorphism theorems have been omitted from lifting, at least for now

12.7 The First Isomorphism Theorem

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

lemmas *a-kernel-defs* =

a-kernel-def kernel-def

lemma *a-kernel-def'*:
a-kernel R S h $\equiv$ {*x* $\in$ *carrier R*. *h x* = $\mathbf{0}_S$}
by (*rule a-kernel-def*[*unfolded kernel-def*, *simplified ring-record-simps*])

12.8 Homomorphisms

lemma *abelian-group-homI*:
includes *abelian-group G*
includes *abelian-group H*
assumes *a-group-hom*: *group-hom* (| *carrier* = *carrier G*, *mult* = *add G*, *one* = *zero G* |)
(| *carrier* = *carrier H*, *mult* = *add H*, *one* = *zero H* |) *h*
shows *abelian-group-hom G H h*
apply (*intro abelian-group-hom.intro abelian-group-hom-axioms.intro*)
apply (*rule G.abelian-group-axioms*)
apply (*rule H.abelian-group-axioms*)
apply (*rule a-group-hom*)
done

lemma (**in** *abelian-group-hom*) *is-abelian-group-hom*:
abelian-group-hom G H h
by (*unfold-locales*)

lemma (**in** *abelian-group-hom*) *hom-add* [*simp*]:
[| *x* : *carrier G*; *y* : *carrier G* |]
==> *h* ($x \oplus_G y$) = *h x* $\oplus_H$ *h y*
by (*rule group-hom.hom-mult*[*OF a-group-hom*,
simplified ring-record-simps])

lemma (**in** *abelian-group-hom*) *hom-closed* [*simp*]:
x $\in$ *carrier G* $\Longrightarrow$ *h x* $\in$ *carrier H*
by (*rule group-hom.hom-closed*[*OF a-group-hom*,
simplified ring-record-simps])

lemma (**in** *abelian-group-hom*) *zero-closed* [*simp*]:
h $\mathbf{0}$ $\in$ *carrier H*
by (*rule group-hom.one-closed*[*OF a-group-hom*,
simplified ring-record-simps])

lemma (**in** *abelian-group-hom*) *hom-zero* [*simp*]:
h $\mathbf{0} = \mathbf{0}_H$
by (*rule group-hom.hom-one*[*OF a-group-hom*,
simplified ring-record-simps])

lemma (**in** *abelian-group-hom*) *a-inv-closed* [*simp*]:
x $\in$ *carrier G* ==> *h* ($\ominus x$) $\in$ *carrier H*
by (*rule group-hom.inv-closed*[*OF a-group-hom*,

folded a-inv-def, *simplified ring-record-simps*])

lemma (**in** *abelian-group-hom*) *hom-a-inv* [*simp*]:
$x \in carrier\ G \implies h\ (\ominus x) = \ominus_H\ (h\ x)$
by (*rule group-hom.hom-inv*[*OF a-group-hom*,
folded a-inv-def, *simplified ring-record-simps*])

lemma (**in** *abelian-group-hom*) *additive-subgroup-a-kernel*:
additive-subgroup (*a-kernel G H h*) *G*
apply (*rule additive-subgroup.intro*)
apply (*rule group-hom.subgroup-kernel*[*OF a-group-hom*,
folded a-kernel-def, *simplified ring-record-simps*])
done

The kernel of a homomorphism is an abelian subgroup

lemma (**in** *abelian-group-hom*) *abelian-subgroup-a-kernel*:
abelian-subgroup (*a-kernel G H h*) *G*
apply (*rule abelian-subgroupI*)
apply (*rule group-hom.normal-kernel*[*OF a-group-hom*,
folded a-kernel-def, *simplified ring-record-simps*])
apply (*simp add*: *G.a-comm*)
done

lemma (**in** *abelian-group-hom*) *A-FactGroup-nonempty*:
assumes *X*: $X \in carrier\ (G\ A\text{-}Mod\ a\text{-}kernel\ G\ H\ h)$
shows $X \neq \{\}$
by (*rule group-hom.FactGroup-nonempty*[*OF a-group-hom*,
folded a-kernel-def A-FactGroup-def, *simplified ring-record-simps*]) (*rule X*)

lemma (**in** *abelian-group-hom*) *FactGroup-contents-mem*:
assumes *X*: $X \in carrier\ (G\ A\text{-}Mod\ (a\text{-}kernel\ G\ H\ h))$
shows *contents* (*h*‘*X*) $\in$ *carrier H*
by (*rule group-hom.FactGroup-contents-mem*[*OF a-group-hom*,
folded a-kernel-def A-FactGroup-def, *simplified ring-record-simps*]) (*rule X*)

lemma (**in** *abelian-group-hom*) *A-FactGroup-hom*:
(λX. *contents* (*h*‘*X*)) $\in$ *hom* (*G A-Mod* (*a-kernel G H h*))
(|*carrier* = *carrier H*, *mult* = *add H*, *one* = *zero H*|)
by (*rule group-hom.FactGroup-hom*[*OF a-group-hom*,
folded a-kernel-def A-FactGroup-def, *simplified ring-record-simps*])

lemma (**in** *abelian-group-hom*) *A-FactGroup-inj-on*:
inj-on (λX. *contents* (*h* ‘ *X*)) (*carrier* (*G A-Mod a-kernel G H h*))
by (*rule group-hom.FactGroup-inj-on*[*OF a-group-hom*,
folded a-kernel-def A-FactGroup-def, *simplified ring-record-simps*])

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

lemma (**in** *abelian-group-hom*) *A-FactGroup-onto*:

assumes h: h ` $carrier\ G = carrier\ H$
shows $(\lambda X.\ contents\ (h\ `\ X))$ ` $carrier\ (G\ A\text{-}Mod\ a\text{-}kernel\ G\ H\ h) = carrier\ H$
by (*rule group-hom.FactGroup-onto*[*OF a-group-hom*,
folded a-kernel-def A-FactGroup-def, *simplified ring-record-simps*]) (*rule h*)

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

theorem (**in** *abelian-group-hom*) *A-FactGroup-iso*:
h ` $carrier\ G = carrier\ H$
$\Longrightarrow (\lambda X.\ contents\ (h`X)) \in (G\ A\text{-}Mod\ (a\text{-}kernel\ G\ H\ h)) \cong$
$(|\ carrier = carrier\ H,\ mult = add\ H,\ one = zero\ H\ |)$
by (*rule group-hom.FactGroup-iso*[*OF a-group-hom*,
folded a-kernel-def A-FactGroup-def, *simplified ring-record-simps*])

13 Lemmas Lifted from CosetExt.thy

Not eveything from `CosetExt.thy` is lifted here.

13.1 General Lemmas from `AlgebraExt.thy`

lemma (**in** *additive-subgroup*) *a-Hcarr* [*simp*]:
assumes hH: $h \in H$
shows $h \in carrier\ G$
by (*rule subgroup.mem-carrier* [*OF a-subgroup*,
simplified monoid-record-simps]) (*rule hH*)

13.2 Lemmas for Right Cosets

lemma (**in** *abelian-subgroup*) *a-elemrcos-carrier*:
assumes *acarr*: $a \in carrier\ G$
and a': $a' \in H \mathbin{+>} a$
shows $a' \in carrier\ G$
by (*rule subgroup.elemrcos-carrier* [*OF a-subgroup a-group*,
folded a-r-coset-def, *simplified monoid-record-simps*]) (*rule acarr*, *rule* a')

lemma (**in** *abelian-subgroup*) *a-rcos-const*:
assumes hH: $h \in H$
shows $H \mathbin{+>} h = H$
by (*rule subgroup.rcos-const* [*OF a-subgroup a-group*,
folded a-r-coset-def, *simplified monoid-record-simps*]) (*rule hH*)

lemma (**in** *abelian-subgroup*) *a-rcos-module-imp*:
assumes *xcarr*: $x \in carrier\ G$
and $x'cos$: $x' \in H \mathbin{+>} x$
shows $(x' \oplus \ominus x) \in H$
by (*rule subgroup.rcos-module-imp* [*OF a-subgroup a-group*,
folded a-r-coset-def a-inv-def, *simplified monoid-record-simps*]) (*rule xcarr*, *rule* $x'cos$)

lemma (**in** *abelian-subgroup*) *a-rcos-module-rev*:
assumes $x \in$ *carrier G* $x' \in$ *carrier G*
and $(x' \oplus \ominus x) \in H$
shows $x' \in H \mathrel{+>} x$
using *assms*
by (*rule subgroup.rcos-module-rev* [*OF a-subgroup a-group*,
folded a-r-coset-def a-inv-def, *simplified monoid-record-simps*])

lemma (**in** *abelian-subgroup*) *a-rcos-module*:
assumes $x \in$ *carrier G* $x' \in$ *carrier G*
shows $(x' \in H \mathrel{+>} x) = (x' \oplus \ominus x \in H)$
using *assms*
by (*rule subgroup.rcos-module* [*OF a-subgroup a-group*,
folded a-r-coset-def a-inv-def, *simplified monoid-record-simps*])

— variant
lemma (**in** *abelian-subgroup*) *a-rcos-module-minus*:
includes *ring G*
assumes *carr*: $x \in$ *carrier G* $x' \in$ *carrier G*
shows $(x' \in H \mathrel{+>} x) = (x' \ominus x \in H)$
proof −
from *carr*
have $(x' \in H \mathrel{+>} x) = (x' \oplus \ominus x \in H)$ **by** (*rule a-rcos-module*)
with *carr*
show $(x' \in H \mathrel{+>} x) = (x' \ominus x \in H)$
by (*simp add*: *minus-eq*)
qed

lemma (**in** *abelian-subgroup*) *a-repr-independence'*:
assumes *y*: $y \in H \mathrel{+>} x$
and *xcarr*: $x \in$ *carrier G*
shows $H \mathrel{+>} x = H \mathrel{+>} y$
apply (*rule a-repr-independence*)
apply (*rule y*)
apply (*rule xcarr*)
apply (*rule a-subgroup*)
done

lemma (**in** *abelian-subgroup*) *a-repr-independenceD*:
assumes *ycarr*: $y \in$ *carrier G*
and *repr*: $H \mathrel{+>} x = H \mathrel{+>} y$
shows $y \in H \mathrel{+>} x$
by (*rule group.repr-independenceD* [*OF a-group a-subgroup*,
folded a-r-coset-def, *simplified monoid-record-simps*]) (*rule ycarr*, *rule repr*)

13.3 Lemmas for the Set of Right Cosets

lemma (**in** *abelian-subgroup*) *a-rcosets-carrier*:

X ∈ *a-rcosets H* ⟹ *X* ⊆ *carrier G*
by (*rule subgroup.rcosets-carrier* [*OF a-subgroup a-group*,
folded A-RCOSETS-def, *simplified monoid-record-simps*])

13.4 Addition of Subgroups

lemma (**in** *abelian-monoid*) *set-add-closed*:
assumes *Acarr*: *A* ⊆ *carrier G*
and *Bcarr*: *B* ⊆ *carrier G*
shows *A* <+> *B* ⊆ *carrier G*
by (*rule monoid.set-mult-closed* [*OF a-monoid*,
folded set-add-def, *simplified monoid-record-simps*]) (*rule Acarr*, *rule Bcarr*)

lemma (**in** *abelian-group*) *add-additive-subgroups*:
assumes *subH*: *additive-subgroup H G*
and *subK*: *additive-subgroup K G*
shows *additive-subgroup* (*H* <+> *K*) *G*
apply (*rule additive-subgroup.intro*)
apply (*unfold set-add-def*)
apply (*intro comm-group.mult-subgroups*)
apply (*rule a-comm-group*)
apply (*rule additive-subgroup.a-subgroup*[*OF subH*])
apply (*rule additive-subgroup.a-subgroup*[*OF subK*])
done

end

theory *Ideal*
imports *Ring AbelCoset*
begin

14 Ideals

14.1 General definition

locale *ideal* = *additive-subgroup I R* + *ring R* +
assumes *I-l-closed*: ⟦*a* ∈ *I*; *x* ∈ *carrier R*⟧ ⟹ *x* ⊗ *a* ∈ *I*
and *I-r-closed*: ⟦*a* ∈ *I*; *x* ∈ *carrier R*⟧ ⟹ *a* ⊗ *x* ∈ *I*

interpretation *ideal* ⊆ *abelian-subgroup I R*
apply (*intro abelian-subgroupI3 abelian-group.intro*)
apply (*rule ideal.axioms*, *rule ideal-axioms*)
apply (*rule abelian-group.axioms*, *rule ring.axioms*, *rule ideal.axioms*, *rule ideal-axioms*)
apply (*rule abelian-group.axioms*, *rule ring.axioms*, *rule ideal.axioms*, *rule ideal-axioms*)
done

lemma (**in** *ideal*) *is-ideal*:

ideal I R
by *fact*

lemma *idealI*:
includes *ring*
assumes *a-subgroup*: *subgroup I* $(\!|$*carrier* = *carrier R*, *mult* = *add R*, *one* = *zero R*$|\!)$
and *I-l-closed*: $\bigwedge a\ x.\ [\![a \in I;\ x \in carrier\ R]\!] \Longrightarrow x \otimes a \in I$
and *I-r-closed*: $\bigwedge a\ x.\ [\![a \in I;\ x \in carrier\ R]\!] \Longrightarrow a \otimes x \in I$
shows *ideal I R*
apply (*intro ideal.intro ideal-axioms.intro additive-subgroupI*)
apply (*rule a-subgroup*)
apply (*rule is-ring*)
apply (*erule* (*1*) *I-l-closed*)
apply (*erule* (*1*) *I-r-closed*)
done

14.2 Ideals Generated by a Subset of *carrier R*

constdefs (**structure** *R*)
genideal :: (*'a*, *'b*) *ring-scheme* ⇒ *'a set* ⇒ *'a set* (*Idl*ı - [*80*] *79*)
genideal R S ≡ *Inter* {*I*. *ideal I R* ∧ *S* ⊆ *I*}

14.3 Principal Ideals

locale *principalideal* = *ideal* +
assumes *generate*: ∃ *i* ∈ *carrier R*. *I* = *Idl* {*i*}

lemma (**in** *principalideal*) *is-principalideal*:
shows *principalideal I R*
by *fact*

lemma *principalidealI*:
includes *ideal*
assumes *generate*: ∃ *i* ∈ *carrier R*. *I* = *Idl* {*i*}
shows *principalideal I R*
by (*intro principalideal.intro principalideal-axioms.intro*) (*rule is-ideal*, *rule generate*)

14.4 Maximal Ideals

locale *maximalideal* = *ideal* +
assumes *I-notcarr*: *carrier R* ≠ *I*
and *I-maximal*: $[\![ideal\ J\ R;\ I \subseteq J;\ J \subseteq carrier\ R]\!] \Longrightarrow J = I \lor J = carrier\ R$

lemma (**in** *maximalideal*) *is-maximalideal*:
shows *maximalideal I R*
by *fact*

lemma *maximalidealI*:
includes *ideal*
assumes *I-notcarr*: *carrier* $R \neq I$
and *I-maximal*: $\bigwedge J$. $[\![$*ideal* J R; $I \subseteq J$; $J \subseteq$ *carrier* $R]\!] \Longrightarrow J = I \vee J =$ *carrier* R
shows *maximalideal* I R
by (*intro maximalideal.intro maximalideal-axioms.intro*)
(*rule is-ideal*, *rule I-notcarr*, *rule I-maximal*)

14.5 Prime Ideals

locale *primeideal* = *ideal* + *cring* +
assumes *I-notcarr*: *carrier* $R \neq I$
and *I-prime*: $[\![a \in$ *carrier* R; $b \in$ *carrier* R; $a \otimes b \in I]\!] \Longrightarrow a \in I \vee b \in I$

lemma (**in** *primeideal*) *is-primeideal*:
shows *primeideal* I R
by *fact*

lemma *primeidealI*:
includes *ideal*
includes *cring*
assumes *I-notcarr*: *carrier* $R \neq I$
and *I-prime*: $\bigwedge a$ b. $[\![a \in$ *carrier* R; $b \in$ *carrier* R; $a \otimes b \in I]\!] \Longrightarrow a \in I \vee b \in I$
shows *primeideal* I R
by (*intro primeideal.intro primeideal-axioms.intro*)
(*rule is-ideal*, *rule is-cring*, *rule I-notcarr*, *rule I-prime*)

lemma *primeidealI2*:
includes *additive-subgroup* I R
includes *cring*
assumes *I-l-closed*: $\bigwedge a$ x. $[\![a \in I$; $x \in$ *carrier* $R]\!] \Longrightarrow x \otimes a \in I$
and *I-r-closed*: $\bigwedge a$ x. $[\![a \in I$; $x \in$ *carrier* $R]\!] \Longrightarrow a \otimes x \in I$
and *I-notcarr*: *carrier* $R \neq I$
and *I-prime*: $\bigwedge a$ b. $[\![a \in$ *carrier* R; $b \in$ *carrier* R; $a \otimes b \in I]\!] \Longrightarrow a \in I \vee b \in I$
shows *primeideal* I R
apply (*intro-locales*)
apply (*intro ideal-axioms.intro*)
apply (*erule* (*1*) *I-l-closed*)
apply (*erule* (*1*) *I-r-closed*)
apply (*intro primeideal-axioms.intro*)
apply (*rule I-notcarr*)
apply (*erule* (*2*) *I-prime*)
done

15 Properties of Ideals

15.1 Special Ideals

```
lemma (in ring) zeroideal:
  shows ideal {0} R
apply (intro idealI subgroup.intro)
      apply (rule is-ring)
     apply simp+
  apply (fold a-inv-def, simp)
 apply simp+
done

lemma (in ring) oneideal:
  shows ideal (carrier R) R
apply (intro idealI  subgroup.intro)
      apply (rule is-ring)
     apply simp+
  apply (fold a-inv-def, simp)
 apply simp+
done

lemma (in domain) zeroprimeideal:
 shows primeideal {0} R
apply (intro primeidealI)
   apply (rule zeroideal)
  apply (rule domain.axioms, rule domain-axioms)
 defer 1
 apply (simp add: integral)
proof (rule ccontr, simp)
  assume carrier R = {0}
  from this have 1 = 0 by (rule one-zeroI)
  from this and one-not-zero
      show False by simp
qed
```

15.2 General Ideal Properies

```
lemma (in ideal) one-imp-carrier:
  assumes I-one-closed: 1 ∈ I
  shows I = carrier R
apply (rule)
apply (rule)
apply (rule a-Hcarr, simp)
proof
  fix x
  assume xcarr: x ∈ carrier R
  from I-one-closed and this
      have x ⊗ 1 ∈ I by (intro I-l-closed)
  from this and xcarr
```

show $x \in I$ **by** *simp*
qed

lemma (**in** *ideal*) *Icarr*:
assumes *iI*: $i \in I$
shows $i \in$ *carrier R*
using *iI* **by** (*rule a-Hcarr*)

15.3 Intersection of Ideals

Intersection of two ideals The intersection of any two ideals is again an ideal in R

lemma (**in** *ring*) *i-intersect*:
includes *ideal I R*
includes *ideal J R*
shows *ideal* $(I \cap J)$ R
apply (*intro idealI subgroup.intro*)
apply (*rule is-ring*)
apply (*force simp add*: *a-subset*)
apply (*simp add*: *a-inv-def* [*symmetric*])
apply *simp*
apply (*simp add*: *a-inv-def* [*symmetric*])
apply (*clarsimp*, *rule*)
apply (*fast intro*: *ideal.I-l-closed ideal.intro prems*)+
apply (*clarsimp*, *rule*)
apply (*fast intro*: *ideal.I-r-closed ideal.intro prems*)+
done

15.3.1 Intersection of a Set of Ideals

The intersection of any Number of Ideals is again an Ideal in R

lemma (**in** *ring*) *i-Intersect*:
assumes *Sideals*: $\bigwedge I.\ I \in S \Longrightarrow$ *ideal I R*
and *notempty*: $S \neq \{\}$
shows *ideal* (*Inter S*) *R*
apply (*unfold-locales*)
apply (*simp-all add*: *Inter-def INTER-def*)
apply (*rule*, *simp*) **defer** *1*
apply *rule* **defer** *1*
apply *rule* **defer** *1*
apply (*fold a-inv-def*, *rule*) **defer** *1*
apply *rule* **defer** *1*
apply *rule* **defer** *1*
proof −
fix x
assume $\forall I \in S.\ x \in I$
hence *xI*: $\bigwedge I.\ I \in S \Longrightarrow x \in I$ **by** *simp*

from *notempty* **have** $\exists\, I0.\ I0 \in S$ **by** *blast*
from *this* **obtain** *I0* **where** *I0S*: $I0 \in S$ **by** *auto*

interpret *ideal* [*I0 R*] **by** (*rule Sideals*[*OF I0S*])

from *xI*[*OF I0S*] **have** $x \in I0$ **.**
from *this* **and** *a-subset* **show** $x \in carrier\ R$ **by** *fast*
next
fix *x y*
assume $\forall\, I{\in}S.\ x \in I$
hence *xI*: $\bigwedge I.\ I \in S \Longrightarrow x \in I$ **by** *simp*
assume $\forall\, I{\in}S.\ y \in I$
hence *yI*: $\bigwedge I.\ I \in S \Longrightarrow y \in I$ **by** *simp*

fix *J*
assume *JS*: $J \in S$
interpret *ideal* [*J R*] **by** (*rule Sideals*[*OF JS*])
from *xI*[*OF JS*] **and** *yI*[*OF JS*]
show $x \oplus y \in J$ **by** (*rule a-closed*)
next
fix *J*
assume *JS*: $J \in S$
interpret *ideal* [*J R*] **by** (*rule Sideals*[*OF JS*])
show $\mathbf{0} \in J$ **by** *simp*
next
fix *x*
assume $\forall\, I{\in}S.\ x \in I$
hence *xI*: $\bigwedge I.\ I \in S \Longrightarrow x \in I$ **by** *simp*

fix *J*
assume *JS*: $J \in S$
interpret *ideal* [*J R*] **by** (*rule Sideals*[*OF JS*])

from *xI*[*OF JS*]
show $\ominus\ x \in J$ **by** (*rule a-inv-closed*)
next
fix *x y*
assume $\forall\, I{\in}S.\ x \in I$
hence *xI*: $\bigwedge I.\ I \in S \Longrightarrow x \in I$ **by** *simp*
assume *ycarr*: $y \in carrier\ R$

fix *J*
assume *JS*: $J \in S$
interpret *ideal* [*J R*] **by** (*rule Sideals*[*OF JS*])

from *xI*[*OF JS*] **and** *ycarr*
show $y \otimes x \in J$ **by** (*rule I-l-closed*)
next
fix *x y*

```
  assume ∀ I∈S. x ∈ I
  hence xI: ⋀I. I ∈ S ⟹ x ∈ I by simp
  assume ycarr: y ∈ carrier R

  fix J
  assume JS: J ∈ S
  interpret ideal [J R] by (rule Sideals[OF JS])

  from xI[OF JS] and ycarr
      show x ⊗ y ∈ J by (rule I-r-closed)
qed
```

15.4 Addition of Ideals

```
lemma (in ring) add-ideals:
  assumes idealI: ideal I R
      and idealJ: ideal J R
  shows ideal (I <+> J) R
apply (rule ideal.intro)
  apply (rule add-additive-subgroups)
   apply (intro ideal.axioms[OF idealI])
  apply (intro ideal.axioms[OF idealJ])
 apply (rule is-ring)
apply (rule ideal-axioms.intro)
 apply (simp add: set-add-defs, clarsimp) defer 1
 apply (simp add: set-add-defs, clarsimp) defer 1
proof −
  fix x i j
  assume xcarr: x ∈ carrier R
     and iI: i ∈ I
     and jJ: j ∈ J
  from xcarr ideal.Icarr[OF idealI iI] ideal.Icarr[OF idealJ jJ]
      have c: (i ⊕ j) ⊗ x = (i ⊗ x) ⊕ (j ⊗ x) by algebra
  from xcarr and iI
      have a: i ⊗ x ∈ I by (simp add: ideal.I-r-closed[OF idealI])
  from xcarr and jJ
      have b: j ⊗ x ∈ J by (simp add: ideal.I-r-closed[OF idealJ])
  from a b c
      show ∃ ha∈I. ∃ ka∈J. (i ⊕ j) ⊗ x = ha ⊕ ka by fast
next
  fix x i j
  assume xcarr: x ∈ carrier R
     and iI: i ∈ I
     and jJ: j ∈ J
  from xcarr ideal.Icarr[OF idealI iI] ideal.Icarr[OF idealJ jJ]
      have c: x ⊗ (i ⊕ j) = (x ⊗ i) ⊕ (x ⊗ j) by algebra
  from xcarr and iI
      have a: x ⊗ i ∈ I by (simp add: ideal.I-l-closed[OF idealI])
  from xcarr and jJ
```

have b: $x \otimes j \in J$ **by** $(simp\ add: ideal.I\text{-}l\text{-}closed[OF\ idealJ])$
from $a\ b\ c$
show $\exists ha{\in}I.\ \exists ka{\in}J.\ x \otimes (i \oplus j) = ha \oplus ka$ **by** *fast*
qed

15.5 Ideals generated by a subset of *carrier R*

15.5.1 Generation of Ideals in General Rings

genideal generates an ideal

lemma (**in** *ring*) *genideal-ideal*:
assumes *Scarr*: $S \subseteq carrier\ R$
shows $ideal\ (Idl\ S)\ R$
unfolding *genideal-def*
proof (*rule i-Intersect*, *fast*, *simp*)
from *oneideal* **and** *Scarr*
show $\exists I.\ ideal\ I\ R \wedge S \leq I$ **by** *fast*
qed

lemma (**in** *ring*) *genideal-self*:
assumes $S \subseteq carrier\ R$
shows $S \subseteq Idl\ S$
unfolding *genideal-def*
by *fast*

lemma (**in** *ring*) *genideal-self'*:
assumes *carr*: $i \in carrier\ R$
shows $i \in Idl\ \{i\}$
proof −
from *carr*
have $\{i\} \subseteq Idl\ \{i\}$ **by** (*fast intro*!: *genideal-self*)
thus $i \in Idl\ \{i\}$ **by** *fast*
qed

genideal generates the minimal ideal

lemma (**in** *ring*) *genideal-minimal*:
assumes a: $ideal\ I\ R$
and b: $S \subseteq I$
shows $Idl\ S \subseteq I$
unfolding *genideal-def*
by (*rule*, *elim InterD*, *simp add*: $a\ b$)

Generated ideals and subsets

lemma (**in** *ring*) *Idl-subset-ideal*:
assumes *Iideal*: $ideal\ I\ R$
and *Hcarr*: $H \subseteq carrier\ R$
shows $(Idl\ H \subseteq I) = (H \subseteq I)$
proof
assume a: $Idl\ H \subseteq I$

from *Hcarr* **have** $H \subseteq$ *Idl* H **by** (*rule genideal-self*)
from *this* **and** *a*
  show $H \subseteq I$ **by** *simp*
next
fix x
assume *HI*: $H \subseteq I$

from *Iideal* **and** *HI*
  have $I \in \{I.$ *ideal* I $R \wedge H \subseteq I\}$ **by** *fast*
from *this*
  show *Idl* $H \subseteq I$
  unfolding *genideal-def*
  by *fast*
qed

lemma (**in** *ring*) *subset-Idl-subset*:
  assumes *Icarr*: $I \subseteq$ *carrier* R
    and *HI*: $H \subseteq I$
  shows *Idl* $H \subseteq$ *Idl* I
proof −
  from *HI* **and** *genideal-self*[*OF Icarr*]
    have *HIdlI*: $H \subseteq$ *Idl* I **by** *fast*

  from *Icarr*
    have *Iideal*: *ideal* (*Idl* I) R **by** (*rule genideal-ideal*)
  from *HI* **and** *Icarr*
    have $H \subseteq$ *carrier* R **by** *fast*
  from *Iideal* **and** *this*
    have ($H \subseteq$ *Idl* I) = (*Idl* $H \subseteq$ *Idl* I)
    by (*rule Idl-subset-ideal*[*symmetric*])

  from *HIdlI* **and** *this*
    show *Idl* $H \subseteq$ *Idl* I **by** *simp*
qed

lemma (**in** *ring*) *Idl-subset-ideal′*:
  assumes *acarr*: $a \in$ *carrier* R **and** *bcarr*: $b \in$ *carrier* R
  shows (*Idl* $\{a\} \subseteq$ *Idl* $\{b\}$) = ($a \in$ *Idl* $\{b\}$)
apply (*subst Idl-subset-ideal*[*OF genideal-ideal*[*of* $\{b\}$], *of* $\{a\}$])
  apply (*fast intro*: *bcarr*, *fast intro*: *acarr*)
apply *fast*
done

lemma (**in** *ring*) *genideal-zero*:
  Idl $\{\mathbf{0}\} = \{\mathbf{0}\}$
apply *rule*
 apply (*rule genideal-minimal*[*OF zeroideal*], *simp*)
apply (*simp add*: *genideal-self′*)
done

lemma (**in** *ring*) *genideal-one*:
  Idl $\{\mathbf{1}\}$ = *carrier R*
proof −
  interpret *ideal* [*Idl* $\{\mathbf{1}\}$ *R*] **by** (*rule genideal-ideal*, *fast intro*: *one-closed*)
  show *Idl* $\{\mathbf{1}\}$ = *carrier R*
  apply (*rule*, *rule a-subset*)
  apply (*simp add*: *one-imp-carrier genideal-self′*)
  done
qed

15.5.2 Generation of Principal Ideals in Commutative Rings

constdefs (**structure** *R*)
  cgenideal :: ($'a$, $'b$) *monoid-scheme* $\Rightarrow$ $'a$ $\Rightarrow$ $'a$ *set* (*PIdl*ı - [*80*] *79*)
  cgenideal R a $\equiv \{ x \otimes a \mid x.\ x \in$ *carrier R* $\}$

genhideal (?) really generates an ideal

lemma (**in** *cring*) *cgenideal-ideal*:
  assumes *acarr*: $a \in$ *carrier R*
  shows *ideal* (*PIdl a*) *R*
apply (*unfold cgenideal-def*)
apply (*rule idealI*[*OF is-ring*])
   apply (*rule subgroup.intro*)
      apply (*simp-all add*: *monoid-record-simps*)
      apply (*blast intro*: *acarr m-closed*)
      apply *clarsimp* **defer** *1*
      defer *1*
      apply (*fold a-inv-def*, *clarsimp*) **defer** *1*
      apply *clarsimp* **defer** *1*
      apply *clarsimp* **defer** *1*
proof −
  fix *x y*
  assume *xcarr*: $x \in$ *carrier R*
    and *ycarr*: $y \in$ *carrier R*
  note *carr* = *acarr xcarr ycarr*

  from *carr*
      have $x \otimes a \oplus y \otimes a = (x \oplus y) \otimes a$ **by** (*simp add*: *l-distr*)
  from *this* **and** *carr*
      show $\exists z.\ x \otimes a \oplus y \otimes a = z \otimes a \land z \in$ *carrier R* **by** *fast*
next
  from *l-null*[*OF acarr*, *symmetric*] **and** *zero-closed*
      show $\exists x.\ \mathbf{0} = x \otimes a \land x \in$ *carrier R* **by** *fast*
next
  fix *x*
  assume *xcarr*: $x \in$ *carrier R*
  note *carr* = *acarr xcarr*

from *carr*
have $\ominus$ $(x \otimes a) = (\ominus\ x) \otimes a$ **by** (*simp add*: *l-minus*)
from *this* **and** *carr*
show $\exists z.\ \ominus\ (x \otimes a) = z \otimes a \wedge z \in$ *carrier R* **by** *fast*
next
fix *x y*
assume *xcarr*: $x \in$ *carrier R*
and *ycarr*: $y \in$ *carrier R*
note *carr* = *acarr xcarr ycarr*

from *carr*
have $y \otimes a \otimes x = (y \otimes x) \otimes a$ **by** (*simp add*: *m-assoc*, *simp add*: *m-comm*)
from *this* **and** *carr*
show $\exists z.\ y \otimes a \otimes x = z \otimes a \wedge z \in$ *carrier R* **by** *fast*
next
fix *x y*
assume *xcarr*: $x \in$ *carrier R*
and *ycarr*: $y \in$ *carrier R*
note *carr* = *acarr xcarr ycarr*

from *carr*
have $x \otimes (y \otimes a) = (x \otimes y) \otimes a$ **by** (*simp add*: *m-assoc*)
from *this* **and** *carr*
show $\exists z.\ x \otimes (y \otimes a) = z \otimes a \wedge z \in$ *carrier R* **by** *fast*
qed

lemma (**in** *ring*) *cgenideal-self*:
assumes *icarr*: $i \in$ *carrier R*
shows $i \in$ *PIdl i*
unfolding *cgenideal-def*
proof *simp*
from *icarr*
have $i = \mathbf{1} \otimes i$ **by** *simp*
from *this* **and** *icarr*
show $\exists x.\ i = x \otimes i \wedge x \in$ *carrier R* **by** *fast*
qed

cgenideal is minimal

lemma (**in** *ring*) *cgenideal-minimal*:
includes *ideal J R*
assumes *aJ*: $a \in J$
shows *PIdl a* $\subseteq J$
unfolding *cgenideal-def*
apply *rule*
apply *clarify*
using *aJ*
apply (*erule I-l-closed*)
done

```
lemma (in cring) cgenideal-eq-genideal:
  assumes icarr: i ∈ carrier R
  shows PIdl i = Idl {i}
apply rule
 apply (intro cgenideal-minimal)
  apply (rule genideal-ideal, fast intro: icarr)
 apply (rule genideal-self', fast intro: icarr)
apply (intro genideal-minimal)
 apply (rule cgenideal-ideal [OF icarr])
apply (simp, rule cgenideal-self [OF icarr])
done

lemma (in cring) cgenideal-eq-rcos:
 PIdl i = carrier R #> i
unfolding cgenideal-def r-coset-def
by fast

lemma (in cring) cgenideal-is-principalideal:
  assumes icarr: i ∈ carrier R
  shows principalideal (PIdl i) R
apply (rule principalidealI)
apply (rule cgenideal-ideal [OF icarr])
proof −
  from icarr
      have PIdl i = Idl {i} by (rule cgenideal-eq-genideal)
  from icarr and this
      show ∃ i'∈carrier R. PIdl i = Idl {i'} by fast
qed
```

15.6 Union of Ideals

```
lemma (in ring) union-genideal:
  assumes idealI: ideal I R
      and idealJ: ideal J R
  shows Idl (I ∪ J) = I <+> J
apply rule
 apply (rule ring.genideal-minimal)
   apply (rule R.is-ring)
  apply (rule add-ideals[OF idealI idealJ])
 apply (rule)
 apply (simp add: set-add-defs) apply (elim disjE) defer 1 defer 1
 apply (rule) apply (simp add: set-add-defs genideal-def) apply clarsimp defer
1
proof −
  fix x
  assume xI: x ∈ I
  have ZJ: 0 ∈ J
      by (intro additive-subgroup.zero-closed, rule ideal.axioms[OF idealJ])
  from ideal.Icarr[OF idealI xI]
```

have $x = x \oplus \mathbf{0}$ **by** *algebra*
from *xI* **and** *ZJ* **and** *this*
show $\exists h{\in}I.\ \exists k{\in}J.\ x = h \oplus k$ **by** *fast*
next
fix x
assume *xJ*: $x \in J$
have *ZI*: $\mathbf{0} \in I$
by (*intro additive-subgroup.zero-closed*, *rule ideal.axioms*[*OF idealI*])
from *ideal.Icarr*[*OF idealJ xJ*]
have $x = \mathbf{0} \oplus x$ **by** *algebra*
from *ZI* **and** *xJ* **and** *this*
show $\exists h{\in}I.\ \exists k{\in}J.\ x = h \oplus k$ **by** *fast*
next
fix $i\ j\ K$
assume *iI*: $i \in I$
and *jJ*: $j \in J$
and *idealK*: *ideal K R*
and *IK*: $I \subseteq K$
and *JK*: $J \subseteq K$
from *iI* **and** *IK*
have *iK*: $i \in K$ **by** *fast*
from *jJ* **and** *JK*
have *jK*: $j \in K$ **by** *fast*
from *iK* **and** *jK*
show $i \oplus j \in K$ **by** (*intro additive-subgroup.a-closed*) (*rule ideal.axioms*[*OF idealK*])
qed

15.7 Properties of Principal Ideals

0 generates the zero ideal

lemma (**in** *ring*) *zero-genideal*:
shows *Idl* $\{\mathbf{0}\} = \{\mathbf{0}\}$
apply *rule*
apply (*simp add*: *genideal-minimal zeroideal*)
apply (*fast intro*!: *genideal-self*)
done

1 generates the unit ideal

lemma (**in** *ring*) *one-genideal*:
shows *Idl* $\{\mathbf{1}\}$ = *carrier R*
proof −
have $\mathbf{1} \in$ *Idl* $\{\mathbf{1}\}$ **by** (*simp add*: *genideal-self'*)
thus *Idl* $\{\mathbf{1}\}$ = *carrier R* **by** (*intro ideal.one-imp-carrier*, *fast intro*: *genideal-ideal*)
qed

The zero ideal is a principal ideal

corollary (**in** *ring*) *zeropideal*:

shows *principalideal* $\{\mathbf{0}\}$ *R*
apply (*rule principalidealI*)
apply (*rule zeroideal*)
apply (*blast intro*!: *zero-closed zero-genideal*[*symmetric*])
done

The unit ideal is a principal ideal

corollary (**in** *ring*) *onepideal*:
shows *principalideal* (*carrier R*) *R*
apply (*rule principalidealI*)
apply (*rule oneideal*)
apply (*blast intro*!: *one-closed one-genideal*[*symmetric*])
done

Every principal ideal is a right coset of the carrier

lemma (**in** *principalideal*) *rcos-generate*:
includes *cring*
shows $\exists x \in I.\ I = carrier\ R\ \#{>}\ x$
proof −
from *generate*
obtain *i*
where *icarr*: $i \in carrier\ R$
and *I1*: $I = Idl\ \{i\}$
by *fast+*

from *icarr* **and** *genideal-self*[*of* $\{i\}$]
have $i \in Idl\ \{i\}$ **by** *fast*
hence *iI*: $i \in I$ **by** (*simp add*: *I1*)

from *I1 icarr*
have *I2*: $I = PIdl\ i$ **by** (*simp add*: *cgenideal-eq-genideal*)

have $PIdl\ i = carrier\ R\ \#{>}\ i$
unfolding *cgenideal-def r-coset-def*
by *fast*

from *I2* **and** *this*
have $I = carrier\ R\ \#{>}\ i$ **by** *simp*

from *iI* **and** *this*
show $\exists x \in I.\ I = carrier\ R\ \#{>}\ x$ **by** *fast*
qed

15.8 Prime Ideals

lemma (**in** *ideal*) *primeidealCD*:
includes *cring*
assumes *notprime*: $\neg$ *primeideal I R*

shows *carrier R* $= I \lor (\exists\, a\ b.\ a \in$ *carrier R* $\land\ b \in$ *carrier R* $\land\ a \otimes b \in I \land a \notin I \land b \notin I)$
proof (*rule ccontr*, *clarsimp*)
assume *InR*: *carrier R* $\neq I$
and $\forall\, a.\ a \in$ *carrier R* $\longrightarrow (\forall\, b.\ a \otimes b \in I \longrightarrow b \in$ *carrier R* $\longrightarrow a \in I \lor b \in I)$
hence *I-prime*: $\bigwedge a\ b.\ [\![a \in$ *carrier R*; $b \in$ *carrier R*; $a \otimes b \in I]\!] \Longrightarrow a \in I \lor b \in I$ **by** *simp*
have *primeideal I R*
apply (*rule primeideal.intro* [*OF is-ideal is-cring*])
apply (*rule primeideal-axioms.intro*)
apply (*rule InR*)
apply (*erule* (*2*) *I-prime*)
done
from *this* **and** *notprime*
show *False* **by** *simp*
qed

lemma (**in** *ideal*) *primeidealCE*:
includes *cring*
assumes *notprime*: $\neg$ *primeideal I R*
obtains *carrier R* $= I$
| $\exists\, a\ b.\ a \in$ *carrier R* $\land\ b \in$ *carrier R* $\land\ a \otimes b \in I \land a \notin I \land b \notin I$
using *primeidealCD* [*OF R.is-cring notprime*] **by** *blast*

If $\{\mathbf{0}\}$ is a prime ideal of a commutative ring, the ring is a domain

lemma (**in** *cring*) *zeroprimeideal-domainI*:
assumes *pi*: *primeideal* $\{\mathbf{0}\}$ *R*
shows *domain R*
apply (*rule domain.intro*, *rule is-cring*)
apply (*rule domain-axioms.intro*)
proof (*rule ccontr*, *simp*)
interpret *primeideal* [$\{\mathbf{0}\}$ *R*] **by** (*rule pi*)
assume $\mathbf{1} = \mathbf{0}$
hence *carrier R* $= \{\mathbf{0}\}$ **by** (*rule one-zeroD*)
from *this*[*symmetric*] **and** *I-notcarr*
show *False* **by** *simp*
next
interpret *primeideal* [$\{\mathbf{0}\}$ *R*] **by** (*rule pi*)
fix *a b*
assume *ab*: $a \otimes b = \mathbf{0}$
and *carr*: $a \in$ *carrier R* $b \in$ *carrier R*
from *ab*
have *abI*: $a \otimes b \in \{\mathbf{0}\}$ **by** *fast*
from *carr* **and** *this*
have $a \in \{\mathbf{0}\} \lor b \in \{\mathbf{0}\}$ **by** (*rule I-prime*)
thus $a = \mathbf{0} \lor b = \mathbf{0}$ **by** *simp*
qed

corollary (**in** *cring*) *domain-eq-zeroprimeideal*:
shows *domain R* = *primeideal* {**0**} *R*
apply *rule*
apply (*erule domain.zeroprimeideal*)
apply (*erule zeroprimeideal-domainI*)
done

15.9 Maximal Ideals

lemma (**in** *ideal*) *helper-I-closed*:
assumes *carr*: $a \in carrier\ R$ $x \in carrier\ R$ $y \in carrier\ R$
and *axI*: $a \otimes x \in I$
shows $a \otimes (x \otimes y) \in I$
proof −
from *axI* **and** *carr*
have $(a \otimes x) \otimes y \in I$ **by** (*simp add*: *I-r-closed*)
also from *carr*
have $(a \otimes x) \otimes y = a \otimes (x \otimes y)$ **by** (*simp add*: *m-assoc*)
finally
show $a \otimes (x \otimes y) \in I$ **.**
qed

lemma (**in** *ideal*) *helper-max-prime*:
includes *cring*
assumes *acarr*: $a \in carrier\ R$
shows *ideal* $\{x \in carrier\ R.\ a \otimes x \in I\}$ *R*
apply (*rule idealI*)
apply (*rule cring.axioms*[*OF is-cring*])
apply (*rule subgroup.intro*)
apply (*simp*, *fast*)
apply *clarsimp* **apply** (*simp add*: *r-distr acarr*)
apply (*simp add*: *acarr*)
apply (*simp add*: *a-inv-def*[*symmetric*], *clarify*) **defer** *1*
apply *clarsimp* **defer** *1*
apply (*fast intro*!: *helper-I-closed acarr*)
proof −
fix *x*
assume *xcarr*: $x \in carrier\ R$
and *ax*: $a \otimes x \in I$
from *ax* **and** *acarr xcarr*
have $\ominus(a \otimes x) \in I$ **by** *simp*
also from *acarr xcarr*
have $\ominus(a \otimes x) = a \otimes (\ominus x)$ **by** *algebra*
finally
show $a \otimes (\ominus x) \in I$ **.**
from *acarr*
have $a \otimes \mathbf{0} = \mathbf{0}$ **by** *simp*
next
fix *x y*

```
  assume xcarr: x ∈ carrier R
     and ycarr: y ∈ carrier R
     and ayI: a ⊗ y ∈ I
  from ayI and acarr xcarr ycarr
      have a ⊗ (y ⊗ x) ∈ I by (simp add: helper-I-closed)
  moreover from xcarr ycarr
      have y ⊗ x = x ⊗ y by (simp add: m-comm)
  ultimately
      show a ⊗ (x ⊗ y) ∈ I by simp
qed
```

In a cring every maximal ideal is prime

```
lemma (in cring) maximalideal-is-prime:
  includes maximalideal
  shows primeideal I R
apply (rule ccontr)
apply (rule primeidealCE)
   apply (rule is-cring)
  apply assumption
 apply (simp add: I-notcarr)
proof −
  assume ∃ a b. a ∈ carrier R ∧ b ∈ carrier R ∧ a ⊗ b ∈ I ∧ a ∉ I ∧ b ∉ I
  from this
      obtain a b
        where acarr: a ∈ carrier R
        and bcarr: b ∈ carrier R
        and abI: a ⊗ b ∈ I
        and anI: a ∉ I
        and bnI: b ∉ I
      by fast
  def J ≡ {x∈carrier R. a ⊗ x ∈ I}

  from R.is-cring and acarr
  have idealJ: ideal J R unfolding J-def by (rule helper-max-prime)

  have IsubJ: I ⊆ J
  proof
    fix x
    assume xI: x ∈ I
    from this and acarr
    have a ⊗ x ∈ I by (intro I-l-closed)
    from xI[THEN a-Hcarr] this
    show x ∈ J unfolding J-def by fast
  qed

  from abI and acarr bcarr
      have b ∈ J unfolding J-def by fast
  from bnI and this
      have JnI: J ≠ I by fast
```

from *acarr*
  have $a = a \otimes \mathbf{1}$ **by** *algebra*
from *this* **and** *anI*
  have $a \otimes \mathbf{1} \notin I$ **by** *simp*
from *one-closed* **and** *this*
  have $\mathbf{1} \notin J$ **unfolding** *J-def* **by** *fast*
hence *Jncarr*: $J \neq$ *carrier R* **by** *fast*

interpret *ideal* [*J R*] **by** (*rule idealJ*)

have $J = I \lor J =$ *carrier R*
  apply (*intro I-maximal*)
  apply (*rule idealJ*)
  apply (*rule IsubJ*)
  apply (*rule a-subset*)
  done

from *this* **and** *JnI* **and** *Jncarr*
  show *False* **by** *simp*
qed

15.10 Derived Theorems Involving Ideals

— A non-zero cring that has only the two trivial ideals is a field
lemma (**in** *cring*) *trivialideals-fieldI*:
  assumes *carrnzero*: *carrier R* $\neq \{\mathbf{0}\}$
    and *haveideals*: $\{I.$ *ideal I R*$\} = \{\{\mathbf{0}\}$, *carrier R*$\}$
  shows *field R*
apply (*rule cring-fieldI*)
apply (*rule*, *rule*, *rule*)
apply (*erule Units-closed*)
defer *1*
  apply *rule*
defer *1*
proof (*rule ccontr*, *simp*)
  assume *zUnit*: $\mathbf{0} \in$ *Units R*
  hence *a*: $\mathbf{0} \otimes$ *inv* $\mathbf{0} = \mathbf{1}$ **by** (*rule Units-r-inv*)
  from *zUnit*
    have $\mathbf{0} \otimes$ *inv* $\mathbf{0} = \mathbf{0}$ **by** (*intro l-null*, *rule Units-inv-closed*)
  from *a*[*symmetric*] **and** *this*
    have $\mathbf{1} = \mathbf{0}$ **by** *simp*
  hence *carrier R* $= \{\mathbf{0}\}$ **by** (*rule one-zeroD*)
  from *this* **and** *carrnzero*
    show *False* **by** *simp*
next
  fix *x*
  assume *xcarr'*: $x \in$ *carrier R* $- \{\mathbf{0}\}$
  hence *xcarr*: $x \in$ *carrier R* **by** *fast*
  from *xcarr'*

have *xnZ*: $x \neq \mathbf{0}$ **by** *fast*
from *xcarr*
have *xIdl*: *ideal* (*PIdl* *x*) *R* **by** (*intro cgenideal-ideal*, *fast*)

from *xcarr*
have $x \in$ *PIdl* *x* **by** (*intro cgenideal-self*, *fast*)
from *this* **and** *xnZ*
have *PIdl* $x \neq \{\mathbf{0}\}$ **by** *fast*
from *haveideals* **and** *this*
have *PIdl* $x =$ *carrier R*
by (*blast intro*!: *xIdl*)
hence $\mathbf{1} \in$ *PIdl* *x* **by** *simp*
hence $\exists\, y.\ \mathbf{1} = y \otimes x \wedge y \in$ *carrier R* **unfolding** *cgenideal-def* **by** *blast*
from *this*
obtain *y*
where *ycarr*: $y \in$ *carrier R*
and *ylinv*: $\mathbf{1} = y \otimes x$
by *fast*+
from *ylinv* **and** *xcarr ycarr*
have *yrinv*: $\mathbf{1} = x \otimes y$ **by** (*simp add*: *m-comm*)
from *ycarr* **and** *ylinv*[*symmetric*] **and** *yrinv*[*symmetric*]
have $\exists\, y \in$ *carrier R*. $y \otimes x = \mathbf{1} \wedge x \otimes y = \mathbf{1}$ **by** *fast*
from *this* **and** *xcarr*
show $x \in$ *Units R*
unfolding *Units-def*
by *fast*
qed

lemma (**in** *field*) *all-ideals*:
shows $\{I.$ *ideal I R*$\} = \{\{\mathbf{0}\},$ *carrier R*$\}$
apply (*rule*, *rule*)
proof −
fix *I*
assume *a*: $I \in \{I.$ *ideal I R*$\}$
with *this*
interpret *ideal* [*I R*] **by** *simp*

show $I \in \{\{\mathbf{0}\},$ *carrier R*$\}$
proof (*cases* $\exists\, a.\ a \in I - \{\mathbf{0}\}$)
assume $\exists\, a.\ a \in I - \{\mathbf{0}\}$
from *this*
obtain *a*
where *aI*: $a \in I$
and *anZ*: $a \neq \mathbf{0}$
by *fast*+
from *aI*[*THEN a-Hcarr*] *anZ*
have *aUnit*: $a \in$ *Units R* **by** (*simp add*: *field-Units*)
hence *a*: $a \otimes$ *inv* $a = \mathbf{1}$ **by** (*rule Units-r-inv*)
from *aI* **and** *aUnit*

```
      have a ⊗ inv a ∈ I by (simp add: I-r-closed)
    hence oneI: 1 ∈ I by (simp add: a[symmetric])

    have carrier R ⊆ I
    proof
      fix x
      assume xcarr: x ∈ carrier R
      from oneI and this
          have 1 ⊗ x ∈ I by (rule I-r-closed)
      from this and xcarr
          show x ∈ I by simp
    qed
    from this and a-subset
        have I = carrier R by fast
    thus I ∈ {{0}, carrier R} by fast
  next
    assume ¬ (∃ a. a ∈ I − {0})
    hence IZ: ⋀a. a ∈ I ⟹ a = 0 by simp

    have a: I ⊆ {0}
    proof
      fix x
      assume x ∈ I
      hence x = 0 by (rule IZ)
      thus x ∈ {0} by fast
    qed

    have 0 ∈ I by simp
    hence {0} ⊆ I by fast

    from this and a
        have I = {0} by fast
    thus I ∈ {{0}, carrier R} by fast
  qed
qed (simp add: zeroideal oneideal)

— Jacobson Theorem 2.2
lemma (in cring) trivialideals-eq-field:
  assumes carrnzero: carrier R ≠ {0}
  shows ({I. ideal I R} = {{0}, carrier R}) = field R
by (fast intro!: trivialideals-fieldI[OF carrnzero] field.all-ideals)
```

Like zeroprimeideal for domains

```
lemma (in field) zeromaximalideal:
  maximalideal {0} R
apply (rule maximalidealI)
  apply (rule zeroideal)
proof−
  from one-not-zero
```

have $\mathbf{1} \notin \{\mathbf{0}\}$ **by** *simp*
from *this* **and** *one-closed*
show *carrier* $R \neq \{\mathbf{0}\}$ **by** *fast*
next
fix J
assume *Jideal*: *ideal* J R
hence $J \in \{I.\ ideal\ I\ R\}$
by *fast*

from *this* **and** *all-ideals*
show $J = \{\mathbf{0}\} \vee J = carrier\ R$ **by** *simp*
qed

lemma (**in** *cring*) *zeromaximalideal-fieldI*:
assumes *zeromax*: *maximalideal* $\{\mathbf{0}\}$ R
shows *field* R
apply (*rule trivialideals-fieldI*, *rule maximalideal.I-notcarr*[*OF zeromax*])
apply *rule* **apply** *clarsimp* **defer** *1*
apply (*simp add*: *zeroideal oneideal*)
proof −
fix J
assume *Jn0*: $J \neq \{\mathbf{0}\}$
and *idealJ*: *ideal* J R
interpret *ideal* [J R] **by** (*rule idealJ*)
have $\{\mathbf{0}\} \subseteq J$ **by** (*rule ccontr*, *simp*)
from *zeromax* **and** *idealJ* **and** *this* **and** *a-subset*
have $J = \{\mathbf{0}\} \vee J = carrier\ R$ **by** (*rule maximalideal.I-maximal*)
from *this* **and** *Jn0*
show $J = carrier\ R$ **by** *simp*
qed

lemma (**in** *cring*) *zeromaximalideal-eq-field*:
maximalideal $\{\mathbf{0}\}$ R = *field* R
apply *rule*
apply (*erule zeromaximalideal-fieldI*)
apply (*erule field.zeromaximalideal*)
done

end

theory *RingHom*
imports *Ideal*
begin

16 Homomorphisms of Non-Commutative Rings

Lifting existing lemmas in a *ring-hom-ring* locale

locale *ring-hom-ring* = *ring R* + *ring S* + *var h* +
  assumes *homh*: $h \in$ *ring-hom R S*
  notes *hom-mult* [*simp*] = *ring-hom-mult* [*OF homh*]
    and *hom-one* [*simp*] = *ring-hom-one* [*OF homh*]

interpretation *ring-hom-cring* $\subseteq$ *ring-hom-ring*
  by (*unfold-locales*, *rule homh*)

interpretation *ring-hom-ring* $\subseteq$ *abelian-group-hom R S*
apply (*rule abelian-group-homI*)
  apply (*rule R.is-abelian-group*)
 apply (*rule S.is-abelian-group*)
apply (*intro group-hom.intro group-hom-axioms.intro*)
  apply (*rule R.a-group*)
 apply (*rule S.a-group*)
apply (*insert homh*, *unfold hom-def ring-hom-def*)
apply *simp*
done

lemma (**in** *ring-hom-ring*) *is-ring-hom-ring*:
  includes *struct R* + *struct S*
  shows *ring-hom-ring R S h*
by *fact*

lemma *ring-hom-ringI*:
  includes *ring R* + *ring S*
  assumes
      hom-closed: !!x. $x \in$ *carrier R* ==> $h\ x \in$ *carrier S*
    and *compatible-mult*: !!$x\ y$. [| x : *carrier R*; y : *carrier R* |] ==> $h\ (x \otimes y)$ = $h\ x \otimes_S h\ y$
    and *compatible-add*: !!$x\ y$. [| x : *carrier R*; y : *carrier R* |] ==> $h\ (x \oplus y)$ = $h\ x \oplus_S h\ y$
    and *compatible-one*: $h\ \mathbf{1} = \mathbf{1}_S$
  shows *ring-hom-ring R S h*
apply *unfold-locales*
apply (*unfold ring-hom-def*, *safe*)
   apply (*simp add*: *hom-closed Pi-def*)
  apply (*erule* (*1*) *compatible-mult*)
 apply (*erule* (*1*) *compatible-add*)
apply (*rule compatible-one*)
done

lemma *ring-hom-ringI2*:
  includes *ring R* + *ring S*
  assumes *h*: $h \in$ *ring-hom R S*
  shows *ring-hom-ring R S h*

apply (*intro ring-hom-ring.intro ring-hom-ring-axioms.intro*)
apply (*rule R.is-ring*)
apply (*rule S.is-ring*)
apply (*rule h*)
done

lemma *ring-hom-ringI3*:
includes *abelian-group-hom R S + ring R + ring S*
assumes *compatible-mult*: !!*x y*. [| *x* : *carrier R*; *y* : *carrier R* |] ==> *h* ($x \otimes y$) = *h* $x \otimes_S h\ y$
and *compatible-one*: $h\ \mathbf{1} = \mathbf{1}_S$
shows *ring-hom-ring R S h*
apply (*intro ring-hom-ring.intro ring-hom-ring-axioms.intro*, *rule R.is-ring*, *rule S.is-ring*)
apply (*insert group-hom.homh*[*OF a-group-hom*])
apply (*unfold hom-def ring-hom-def*, *simp*)
apply *safe*
apply (*erule* (*1*) *compatible-mult*)
apply (*rule compatible-one*)
done

lemma *ring-hom-cringI*:
includes *ring-hom-ring R S h + cring R + cring S*
shows *ring-hom-cring R S h*
by (*intro ring-hom-cring.intro ring-hom-cring-axioms.intro*)
(*rule R.is-cring*, *rule S.is-cring*, *rule homh*)

16.1 The kernel of a ring homomorphism

— the kernel of a ring homomorphism is an ideal
lemma (**in** *ring-hom-ring*) *kernel-is-ideal*:
shows *ideal* (*a-kernel R S h*) *R*
apply (*rule idealI*)
apply (*rule R.is-ring*)
apply (*rule additive-subgroup.a-subgroup*[*OF additive-subgroup-a-kernel*])
apply (*unfold a-kernel-def′*, *simp+*)
done

Elements of the kernel are mapped to zero

lemma (**in** *abelian-group-hom*) *kernel-zero* [*simp*]:
$i \in$ *a-kernel R S h* $\Longrightarrow h\ i = \mathbf{0}_S$
by (*simp add*: *a-kernel-defs*)

16.2 Cosets

Cosets of the kernel correspond to the elements of the image of the homomorphism

lemma (**in** *ring-hom-ring*) *rcos-imp-homeq*:
assumes *acarr*: $a \in$ *carrier R*

    and *xrcos*: $x \in$ *a-kernel R S h* +> *a*
  shows $h\ x = h\ a$
proof −
  interpret *ideal* [*a-kernel R S h R*] **by** (*rule kernel-is-ideal*)

  from *xrcos*
    have $\exists\, i \in$ *a-kernel R S h*. $x = i \oplus a$ **by** (*simp add*: *a-r-coset-defs*)
  from *this* **obtain** *i*
    where *iker*: $i \in$ *a-kernel R S h*
      and *x*: $x = i \oplus a$
    by *fast*+
  note *carr* = *acarr iker*[*THEN a-Hcarr*]

  from *x*
    have $h\ x = h\ (i \oplus a)$ **by** *simp*
  also from *carr*
    have $\ldots = h\ i \oplus_S h\ a$ **by** *simp*
  also from *iker*
    have $\ldots = \mathbf{0}_S \oplus_S h\ a$ **by** *simp*
  also from *carr*
    have $\ldots = h\ a$ **by** *simp*
  finally
    show $h\ x = h\ a$ **.**
qed

lemma (**in** *ring-hom-ring*) *homeq-imp-rcos*:
  assumes *acarr*: $a \in$ *carrier R*
    and *xcarr*: $x \in$ *carrier R*
    and *hx*: $h\ x = h\ a$
  shows $x \in$ *a-kernel R S h* +> *a*
proof −
  interpret *ideal* [*a-kernel R S h R*] **by** (*rule kernel-is-ideal*)

  note *carr* = *acarr xcarr*
  note *hcarr* = *acarr*[*THEN hom-closed*] *xcarr*[*THEN hom-closed*]

  from *hx* **and** *hcarr*
    have *a*: $h\ x \oplus_S \ominus_S h\ a = \mathbf{0}_S$ **by** *algebra*
  from *carr*
    have $h\ x \oplus_S \ominus_S h\ a = h\ (x \oplus \ominus a)$ **by** *simp*
  from *a* **and** *this*
    have *b*: $h\ (x \oplus \ominus a) = \mathbf{0}_S$ **by** *simp*

  from *carr* **have** $x \oplus \ominus a \in$ *carrier R* **by** *simp*
  from *this* **and** *b*
    have $x \oplus \ominus a \in$ *a-kernel R S h*
    unfolding *a-kernel-def′*
    by *fast*

from *this* **and** *carr*
show $x \in$ *a-kernel R S h* $+>$ *a* **by** (*simp add*: *a-rcos-module-rev*)
qed

corollary (**in** *ring-hom-ring*) *rcos-eq-homeq*:
assumes *acarr*: $a \in$ *carrier R*
shows (*a-kernel R S h*) $+>$ $a = \{x \in$ *carrier R*. $h\ x = h\ a\}$
apply *rule* **defer** *1*
apply *clarsimp* **defer** *1*
proof
interpret *ideal* [*a-kernel R S h R*] **by** (*rule kernel-is-ideal*)

fix *x*
assume *xrcos*: $x \in$ *a-kernel R S h* $+>$ *a*
from *acarr* **and** *this*
have *xcarr*: $x \in$ *carrier R*
by (*rule a-elemrcos-carrier*)

from *xrcos*
have $h\ x = h\ a$ **by** (*rule rcos-imp-homeq*[*OF acarr*])
from *xcarr* **and** *this*
show $x \in \{x \in$ *carrier R*. $h\ x = h\ a\}$ **by** *fast*
next
interpret *ideal* [*a-kernel R S h R*] **by** (*rule kernel-is-ideal*)

fix *x*
assume *xcarr*: $x \in$ *carrier R*
and *hx*: $h\ x = h\ a$
from *acarr xcarr hx*
show $x \in$ *a-kernel R S h* $+>$ *a* **by** (*rule homeq-imp-rcos*)
qed

end

17 QuotRing: Quotient Rings

theory *QuotRing*
imports *RingHom*
begin

17.1 Multiplication on Cosets

constdefs (**structure** *R*)
rcoset-mult :: [(*'a*, -) *ring-scheme*, *'a set*, *'a set*, *'a set*] $\Rightarrow$ *'a set*
([*mod* -:] - $\bigotimes\imath$ - [*81*,*81*,*81*] *80*)
rcoset-mult R I A B $\equiv \bigcup a \in A.\ \bigcup b \in B.\ I +> (a \otimes b)$

rcoset-mult fulfils the properties required by congruences

lemma (**in** *ideal*) *rcoset-mult-add*:
$[\![x \in carrier\ R;\ y \in carrier\ R]\!] \Longrightarrow [mod\ I:]\ (I\ {+}{>}\ x) \bigotimes (I\ {+}{>}\ y) = I\ {+}{>}\ (x \otimes y)$
apply *rule*
apply (*rule*, *simp add*: *rcoset-mult-def*, *clarsimp*)
defer *1*
apply (*rule*, *simp add*: *rcoset-mult-def*)
defer *1*
proof −
fix $z\ x'\ y'$
assume *carr*: $x \in carrier\ R\ y \in carrier\ R$
and *x'rcos*: $x' \in I\ {+}{>}\ x$
and *y'rcos*: $y' \in I\ {+}{>}\ y$
and *zrcos*: $z \in I\ {+}{>}\ x' \otimes y'$

from *x'rcos*
have $\exists h \in I.\ x' = h \oplus x$ **by** (*simp add*: *a-r-coset-def r-coset-def*)
from *this* **obtain** *hx*
where *hxI*: $hx \in I$
and *x'*: $x' = hx \oplus x$
by *fast+*

from *y'rcos*
have $\exists h \in I.\ y' = h \oplus y$ **by** (*simp add*: *a-r-coset-def r-coset-def*)
from *this*
obtain *hy*
where *hyI*: $hy \in I$
and *y'*: $y' = hy \oplus y$
by *fast+*

from *zrcos*
have $\exists h \in I.\ z = h \oplus (x' \otimes y')$ **by** (*simp add*: *a-r-coset-def r-coset-def*)
from *this*
obtain *hz*
where *hzI*: $hz \in I$
and *z*: $z = hz \oplus (x' \otimes y')$
by *fast+*

note *carr* = *carr hxI*[*THEN a-Hcarr*] *hyI*[*THEN a-Hcarr*] *hzI*[*THEN a-Hcarr*]

from *z* **have** $z = hz \oplus (x' \otimes y')$ **.**
also from $x'\ y'$
have $\ldots = hz \oplus ((hx \oplus x) \otimes (hy \oplus y))$ **by** *simp*
also from *carr*
have $\ldots = (hz \oplus (hx \otimes (hy \oplus y)) \oplus x \otimes hy) \oplus x \otimes y$ **by** *algebra*
finally
have *z2*: $z = (hz \oplus (hx \otimes (hy \oplus y)) \oplus x \otimes hy) \oplus x \otimes y$ **.**

from *hxI hyI hzI carr*

have $hz \oplus (hx \otimes (hy \oplus y)) \oplus x \otimes hy \in I$ **by** (*simp add*: *I-l-closed I-r-closed*)

from *this* **and** *z2*
have $\exists h{\in}I.\ z = h \oplus x \otimes y$ **by** *fast*
thus $z \in I +> x \otimes y$ **by** (*simp add*: *a-r-coset-def r-coset-def*)
next
fix z
assume *xcarr*: $x \in$ *carrier* R
and *ycarr*: $y \in$ *carrier* R
and *zrcos*: $z \in I +> x \otimes y$
from *xcarr*
have *xself*: $x \in I +> x$ **by** (*intro a-rcos-self*)
from *ycarr*
have *yself*: $y \in I +> y$ **by** (*intro a-rcos-self*)

from *xself* **and** *yself* **and** *zrcos*
show $\exists a{\in}I +> x.\ \exists b{\in}I +> y.\ z \in I +> a \otimes b$ **by** *fast*
qed

17.2 Quotient Ring Definition

constdefs (**structure** R)
FactRing :: [('a,'b) *ring-scheme*, 'a *set*] ⇒ ('a *set*) *ring*
(**infixl** *Quot 65*)
FactRing R I ≡
(|*carrier* = *a-rcosets* I, *mult* = *rcoset-mult* R I, *one* = $(I +> \mathbf{1})$, *zero* = I, *add* = *set-add* R|)

17.3 Factorization over General Ideals

The quotient is a ring

lemma (**in** *ideal*) *quotient-is-ring*:
shows *ring* (R *Quot* I)
apply (*rule ringI*)
— abelian group
apply (*rule comm-group-abelian-groupI*)
apply (*simp add*: *FactRing-def*)
apply (*rule a-factorgroup-is-comm-group*[*unfolded A-FactGroup-def'*])
— mult monoid
apply (*rule monoidI*)
apply (*simp-all add*: *FactRing-def A-RCOSETS-def RCOSETS-def a-r-coset-def*[*symmetric*])
— mult closed
apply (*clarify*)
apply (*simp add*: *rcoset-mult-add*, *fast*)
— mult *one-closed*
apply (*force intro*: *one-closed*)
— mult assoc
apply *clarify*

```
   apply (simp add: rcoset-mult-add m-assoc)
  — mult one
  apply clarify
  apply (simp add: rcoset-mult-add l-one)
 apply clarify
 apply (simp add: rcoset-mult-add r-one)
— distr
apply clarify
apply (simp add: rcoset-mult-add a-rcos-sum l-distr)
apply clarify
apply (simp add: rcoset-mult-add a-rcos-sum r-distr)
done
```

This is a ring homomorphism

```
lemma (in ideal) rcos-ring-hom:
  (op +> I) ∈ ring-hom R (R Quot I)
apply (rule ring-hom-memI)
   apply (simp add: FactRing-def a-rcosetsI[OF a-subset])
  apply (simp add: FactRing-def rcoset-mult-add)
 apply (simp add: FactRing-def a-rcos-sum)
apply (simp add: FactRing-def)
done

lemma (in ideal) rcos-ring-hom-ring:
  ring-hom-ring R (R Quot I) (op +> I)
apply (rule ring-hom-ringI)
     apply (rule is-ring, rule quotient-is-ring)
   apply (simp add: FactRing-def a-rcosetsI[OF a-subset])
  apply (simp add: FactRing-def rcoset-mult-add)
 apply (simp add: FactRing-def a-rcos-sum)
apply (simp add: FactRing-def)
done
```

The quotient of a cring is also commutative

```
lemma (in ideal) quotient-is-cring:
  includes cring
  shows cring (R Quot I)
apply (intro cring.intro comm-monoid.intro comm-monoid-axioms.intro)
  apply (rule quotient-is-ring)
 apply (rule ring.axioms[OF quotient-is-ring])
apply (simp add: FactRing-def A-RCOSETS-defs a-r-coset-def[symmetric])
apply clarify
apply (simp add: rcoset-mult-add m-comm)
done
```

Cosets as a ring homomorphism on crings

```
lemma (in ideal) rcos-ring-hom-cring:
  includes cring
  shows ring-hom-cring R (R Quot I) (op +> I)
```

apply (*rule ring-hom-cringI*)
  apply (*rule rcos-ring-hom-ring*)
 apply (*rule R.is-cring*)
apply (*rule quotient-is-cring*)
apply (*rule R.is-cring*)
done

17.4 Factorization over Prime Ideals

The quotient ring generated by a prime ideal is a domain

lemma (**in** *primeideal*) *quotient-is-domain*:
  shows *domain* (*R Quot I*)
apply (*rule domain.intro*)
 apply (*rule quotient-is-cring*, *rule is-cring*)
apply (*rule domain-axioms.intro*)
 apply (*simp add*: *FactRing-def*) **defer** *1*
 apply (*simp add*: *FactRing-def A-RCOSETS-defs a-r-coset-def*[*symmetric*], *clarify*)
 apply (*simp add*: *rcoset-mult-add*) **defer** *1*
proof (*rule ccontr*, *clarsimp*)
  assume $I +> \mathbf{1} = I$
  hence $\mathbf{1} \in I$ **by** (*simp only*: *a-coset-join1 one-closed a-subgroup*)
  hence *carrier R* $\subseteq I$ **by** (*subst one-imp-carrier*, *simp*, *fast*)
  from *this* **and** *a-subset*
    have $I =$ *carrier R* **by** *fast*
  from *this* **and** *I-notcarr*
    show *False* **by** *fast*
next
  fix x y
  assume *carr*: $x \in$ *carrier R* $y \in$ *carrier R*
    and *a*: $I +> x \otimes y = I$
    and *b*: $I +> y \neq I$

  have *ynI*: $y \notin I$
  proof (*rule ccontr*, *simp*)
    assume $y \in I$
    hence $I +> y = I$ **by** (*rule a-rcos-const*)
    from *this* **and** *b*
      show *False* **by** *simp*
  qed

  from *carr*
    have $x \otimes y \in I +> x \otimes y$ **by** (*simp add*: *a-rcos-self*)
  from *this*
    have *xyI*: $x \otimes y \in I$ **by** (*simp add*: *a*)

  from *xyI* **and** *carr*
    have *xI*: $x \in I \vee y \in I$ **by** (*simp add*: *I-prime*)
  from *this* **and** *ynI*

    have $x \in I$ **by** *fast*
  thus I +> $x = I$ **by** (*rule a-rcos-const*)
qed

Generating right cosets of a prime ideal is a homomorphism on commutative rings

lemma (**in** *primeideal*) *rcos-ring-hom-cring*:
  shows *ring-hom-cring R* (*R Quot I*) (*op* +> *I*)
by (*rule rcos-ring-hom-cring*, *rule is-cring*)

17.5 Factorization over Maximal Ideals

In a commutative ring, the quotient ring over a maximal ideal is a field. The proof follows "W. Adkins, S. Weintraub: Algebra – An Approach via Module Theory"

lemma (**in** *maximalideal*) *quotient-is-field*:
  includes *cring*
  shows *field* (*R Quot I*)
apply (*intro cring.cring-fieldI2*)
  apply (*rule quotient-is-cring*, *rule is-cring*)
 defer *1*
 apply (*simp add*: *FactRing-def A-RCOSETS-defs a-r-coset-def*[*symmetric*], *clarsimp*)
 apply (*simp add*: *rcoset-mult-add*) **defer** *1*
proof (*rule ccontr*, *simp*)
  — Quotient is not empty
  assume $\mathbf{0}_{R\ Quot\ I} = \mathbf{1}_{R\ Quot\ I}$
  hence *II1*: $I = I$ +> **1** **by** (*simp add*: *FactRing-def*)
  from *a-rcos-self*[*OF one-closed*]
  have $\mathbf{1} \in I$ **by** (*simp add*: *II1*[*symmetric*])
  hence I = *carrier R* **by** (*rule one-imp-carrier*)
  from *this* **and** *I-notcarr*
  show *False* **by** *simp*
next
  — Existence of Inverse
  fix *a*
  assume *IanI*: I +> $a \neq I$
    and *acarr*: $a \in$ *carrier R*

  — Helper ideal J
  def $J \equiv$ (*carrier R* #> *a*) <+> I :: *'a set*
  have *idealJ*: *ideal J R*
    apply (*unfold J-def*, *rule add-ideals*)
    apply (*simp only*: *cgenideal-eq-rcos*[*symmetric*], *rule cgenideal-ideal*, *rule acarr*)
    apply (*rule is-ideal*)
    done

  — Showing J not smaller than I

```
  have IinJ: I ⊆ J
  proof (rule, simp add: J-def r-coset-def set-add-defs)
    fix x
    assume xI: x ∈ I
    have Zcarr: 0 ∈ carrier R by fast
    from xI[THEN a-Hcarr] acarr
    have x = 0 ⊗ a ⊕ x by algebra

    from Zcarr and xI and this
    show ∃xa∈carrier R. ∃k∈I. x = xa ⊗ a ⊕ k by fast
  qed

  — Showing J ≠ I
  have anI: a ∉ I
  proof (rule ccontr, simp)
    assume a ∈ I
    hence I +> a = I by (rule a-rcos-const)
    from this and IanI
    show False by simp
  qed

  have aJ: a ∈ J
  proof (simp add: J-def r-coset-def set-add-defs)
    from acarr
    have a = 1 ⊗ a ⊕ 0 by algebra
      from one-closed and additive-subgroup.zero-closed[OF is-additive-subgroup]
and this
    show ∃x∈carrier R. ∃k∈I. a = x ⊗ a ⊕ k by fast
  qed

  from aJ and anI
  have JnI: J ≠ I by fast

  — Deducing J = carrier R because I is maximal
  from idealJ and IinJ
  have J = I ∨ J = carrier R
  proof (rule I-maximal, unfold J-def)
    have carrier R #> a ⊆ carrier R
      using subset-refl acarr
      by (rule r-coset-subset-G)
    from this and a-subset
    show carrier R #> a <+> I ⊆ carrier R by (rule set-add-closed)
  qed

  from this and JnI
  have Jcarr: J = carrier R by simp

  — Calculating an inverse for a
  from one-closed[folded Jcarr]
```

```
  have ∃r∈carrier R. ∃i∈I. 1 = r ⊗ a ⊕ i
    by (simp add: J-def r-coset-def set-add-defs)
  from this
  obtain r i
    where rcarr: r ∈ carrier R
      and iI: i ∈ I
      and one: 1 = r ⊗ a ⊕ i
    by fast
  from one and rcarr and acarr and iI[THEN a-Hcarr]
  have rai1: a ⊗ r = ⊖i ⊕ 1 by algebra

  — Lifting to cosets
  from iI
  have ⊖i ⊕ 1 ∈ I +> 1
    by (intro a-rcosI, simp, intro a-subset, simp)
  from this and rai1
  have a ⊗ r ∈ I +> 1 by simp
  from this have I +> 1 = I +> a ⊗ r
    by (rule a-repr-independence, simp) (rule a-subgroup)

  from rcarr and this[symmetric]
  show ∃r∈carrier R. I +> a ⊗ r = I +> 1 by fast
qed

end



theory IntRing
imports QuotRing IntDef
begin
```

18 The Ring of Integers

18.1 Some properties of *int*

```
lemma dvds-imp-absеq:
  [[l dvd k; k dvd l]] ⟹ abs l = abs (k::int)
apply (subst abs-split, rule conjI)
 apply (clarsimp, subst abs-split, rule conjI)
  apply (clarsimp)
  apply (cases k=0, simp)
  apply (cases l=0, simp)
  apply (simp add: zdvd-anti-sym)
 apply clarsimp
 apply (cases k=0, simp)
 apply (simp add: zdvd-anti-sym)
apply (clarsimp, subst abs-split, rule conjI)
 apply (clarsimp)
```

apply (*cases* $l=0$, *simp*)
apply (*simp add*: *zdvd-anti-sym*)
apply (*clarsimp*)
apply (*subgoal-tac* $-l = -k$, *simp*)
apply (*intro zdvd-anti-sym*, *simp*+)
done

lemma *abseq-imp-dvd*:
assumes *a-lk*: *abs* l = *abs* (k::*int*)
shows l *dvd* k
proof −
from *a-lk*
have *nat* (*abs* l) = *nat* (*abs* k) **by** *simp*
hence *nat* (*abs* l) *dvd* *nat* (*abs* k) **by** *simp*
hence *int* (*nat* (*abs* l)) *dvd* k **by** (*subst int-dvd-iff*)
hence *abs* l *dvd* k **by** *simp*
thus l *dvd* k
apply (*unfold dvd-def*, *cases* $l<0$)
defer *1* **apply** *clarsimp*
proof (*clarsimp*)
fix k
assume *l0*: $l < 0$
have $-(l * k) = l * (-k)$ **by** *simp*
thus $\exists ka. -(l * k) = l * ka$ **by** *fast*
qed
qed

lemma *dvds-eq-abseq*:
(l *dvd* k $\wedge$ k *dvd* l) = (*abs* l = *abs* (k::*int*))
apply *rule*
apply (*simp add*: *dvds-imp-abseq*)
apply (*rule conjI*)
apply (*simp add*: *abseq-imp-dvd*)+
done

18.2 The Set of Integers as Algebraic Structure

18.2.1 Definition of $\mathcal{Z}$

constdefs
int-ring :: *int ring* ($\mathcal{Z}$)
int-ring $\equiv$ (| *carrier* = *UNIV*, *mult* = *op* $*$, *one* = *1*, *zero* = *0*, *add* = *op* + |)

lemma *int-Zcarr* [*intro*!, *simp*]:
$k \in$ *carrier* $\mathcal{Z}$
by (*simp add*: *int-ring-def*)

lemma *int-is-cring*:
cring $\mathcal{Z}$
unfolding *int-ring-def*

apply (*rule cringI*)
  apply (*rule abelian-groupI*, *simp-all*)
  defer *1*
  apply (*rule comm-monoidI*, *simp-all*)
 apply (*rule zadd-zmult-distrib*)
apply (*fast intro*: *zadd-zminus-inverse2*)
done

18.2.2 Interpretations

Since definitions of derived operations are global, their interpretation needs to be done as early as possible — that is, with as few assumptions as possible.

interpretation *int*: *monoid* [$\mathcal{Z}$]
  where *carrier* $\mathcal{Z}$ = *UNIV*
    and *mult* $\mathcal{Z}$ x y = $x * y$
    and *one* $\mathcal{Z}$ = *1*
    and *pow* $\mathcal{Z}$ x n = $x\hat{}n$
proof −
  — Specification
  show *monoid* $\mathcal{Z}$ **by** (*unfold-locales*) (*auto simp*: *int-ring-def*)
  then interpret *int*: *monoid* [$\mathcal{Z}$] **.**

  — Carrier
  show *carrier* $\mathcal{Z}$ = *UNIV* **by** (*simp add*: *int-ring-def*)

  — Operations
  { fix x y **show** *mult* $\mathcal{Z}$ x y = $x * y$ **by** (*simp add*: *int-ring-def*) **}**
  note *mult* = *this*
  show *one*: *one* $\mathcal{Z}$ = *1* **by** (*simp add*: *int-ring-def*)
  show *pow* $\mathcal{Z}$ x n = $x\hat{}n$ **by** (*induct n*) (*simp*, *simp add*: *int-ring-def*)+
qed

interpretation *int*: *comm-monoid* [$\mathcal{Z}$]
  where *finprod* $\mathcal{Z}$ f A = (*if finite A then setprod f A else arbitrary*)
proof −
  — Specification
  show *comm-monoid* $\mathcal{Z}$ **by** (*unfold-locales*) (*auto simp*: *int-ring-def*)
  then interpret *int*: *comm-monoid* [$\mathcal{Z}$] **.**

  — Operations
  { fix x y **have** *mult* $\mathcal{Z}$ x y = $x * y$ **by** (*simp add*: *int-ring-def*) **}**
  note *mult* = *this*
  have *one*: *one* $\mathcal{Z}$ = *1* **by** (*simp add*: *int-ring-def*)
  show *finprod* $\mathcal{Z}$ f A = (*if finite A then setprod f A else arbitrary*)
  proof (*cases finite A*)
    case *True* **then show** *?thesis* **proof** *induct*
      case *empty* **show** *?case* **by** (*simp add*: *one*)
    next
      case *insert* **then show** *?case* **by** (*simp add*: *Pi-def mult*)

```
    qed
  next
    case False then show ?thesis by (simp add: finprod-def)
  qed
qed

interpretation int: abelian-monoid [Z]
  where zero Z = 0
    and add Z x y = x + y
    and finsum Z f A = (if finite A then setsum f A else arbitrary)
proof -
  — Specification
  show abelian-monoid Z by (unfold-locales) (auto simp: int-ring-def)
  then interpret int: abelian-monoid [Z] .

  — Operations
  { fix x y show add Z x y = x + y by (simp add: int-ring-def) }
  note add = this
  show zero: zero Z = 0 by (simp add: int-ring-def)
  show finsum Z f A = (if finite A then setsum f A else arbitrary)
  proof (cases finite A)
    case True then show ?thesis proof induct
      case empty show ?case by (simp add: zero)
    next
      case insert then show ?case by (simp add: Pi-def add)
    qed
  next
    case False then show ?thesis by (simp add: finsum-def finprod-def)
  qed
qed

interpretation int: abelian-group [Z]
  where a-inv Z x = - x
    and a-minus Z x y = x - y
proof -
  — Specification
  show abelian-group Z
  proof (rule abelian-groupI)
    show !!x. x ∈ carrier Z ==> EX y : carrier Z. y ⊕_Z x = 𝟎_Z
      by (simp add: int-ring-def) arith
  qed (auto simp: int-ring-def)
  then interpret int: abelian-group [Z] .

  — Operations
  { fix x y have add Z x y = x + y by (simp add: int-ring-def) }
  note add = this
  have zero: zero Z = 0 by (simp add: int-ring-def)
  { fix x
    have add Z (-x) x = zero Z by (simp add: add zero)
```

```
    then show a-inv Z x = - x by (simp add: int.minus-equality) }
  note a-inv = this
  show a-minus Z x y = x - y by (simp add: int.minus-eq add a-inv)
qed

interpretation int: domain [Z]
  by (unfold-locales) (auto simp: int-ring-def left-distrib right-distrib)
```

Removal of occurrences of *UNIV* in interpretation result — experimental.

```
lemma UNIV:
  x ∈ UNIV = True
  A ⊆ UNIV = True
  (ALL x : UNIV. P x) = (ALL x. P x)
  (EX x : UNIV. P x) = (EX x. P x)
  (True --> Q) = Q
  (True ==> PROP R) == PROP R
  by simp-all

interpretation int [unfolded UNIV]:
  partial-order [(| carrier = UNIV::int set, le = op ≤ |)]
  where carrier (| carrier = UNIV::int set, le = op ≤ |) = UNIV
    and le (| carrier = UNIV::int set, le = op ≤ |) x y = (x ≤ y)
    and lless (| carrier = UNIV::int set, le = op ≤ |) x y = (x < y)
proof -
  show partial-order (| carrier = UNIV::int set, le = op ≤ |)
    by unfold-locales simp-all
  show carrier (| carrier = UNIV::int set, le = op ≤ |) = UNIV
    by simp
  show le (| carrier = UNIV::int set, le = op ≤ |) x y = (x ≤ y)
    by simp
  show lless (| carrier = UNIV::int set, le = op ≤ |) x y = (x < y)
    by (simp add: lless-def) auto
qed

interpretation int [unfolded UNIV]:
  lattice [(| carrier = UNIV::int set, le = op ≤ |)]
  where join (| carrier = UNIV::int set, le = op ≤ |) x y = max x y
    and meet (| carrier = UNIV::int set, le = op ≤ |) x y = min x y
proof -
  let ?Z = (| carrier = UNIV::int set, le = op ≤ |)
  show lattice ?Z
    apply unfold-locales
    apply (simp add: least-def Upper-def)
    apply arith
    apply (simp add: greatest-def Lower-def)
    apply arith
    done
  then interpret int: lattice [?Z] .
  show join ?Z x y = max x y
```

apply (*rule int.joinI*)
apply (*simp-all add*: *least-def Upper-def*)
apply *arith*
done
show *meet ?Z x y* = *min x y*
apply (*rule int.meetI*)
apply (*simp-all add*: *greatest-def Lower-def*)
apply *arith*
done
qed

interpretation *int* [*unfolded UNIV*]:
total-order [(| *carrier* = *UNIV*::*int set*, *le* = *op* ≤ |)]
by *unfold-locales clarsimp*

18.2.3 Generated Ideals of $\mathcal{Z}$

lemma *int-Idl*:
$Idl_{\mathcal{Z}}$ {*a*} = {*x* ∗ *a* | *x*. *True*}
apply (*subst int.cgenideal-eq-genideal*[*symmetric*]) **apply** (*simp add*: *int-ring-def*)
apply (*simp add*: *cgenideal-def int-ring-def*)
done

lemma *multiples-principalideal*:
principalideal {*x* ∗ *a* | *x*. *True* } $\mathcal{Z}$
apply (*subst int-Idl*[*symmetric*], *rule principalidealI*)
apply (*rule int.genideal-ideal*, *simp*)
apply *fast*
done

lemma *prime-primeideal*:
assumes *prime*: *prime* (*nat p*)
shows *primeideal* ($Idl_{\mathcal{Z}}$ {*p*}) $\mathcal{Z}$
apply (*rule primeidealI*)
apply (*rule int.genideal-ideal*, *simp*)
apply (*rule int-is-cring*)
apply (*simp add*: *int.cgenideal-eq-genideal*[*symmetric*] *cgenideal-def*)
apply (*simp add*: *int-ring-def*)
apply *clarsimp* **defer** *1*
apply (*simp add*: *int.cgenideal-eq-genideal*[*symmetric*] *cgenideal-def*)
apply (*simp add*: *int-ring-def*)
apply (*elim exE*)
proof −
fix *a b x*

from *prime*
have *ppos*: *0* <= *p* **by** (*simp add*: *prime-def*)
have *unnat*: !!*x*. *nat p dvd nat* (*abs x*) ==> *p dvd x*
proof −

```
    fix x
    assume nat p dvd nat (abs x)
    hence int (nat p) dvd x by (simp add: int-dvd-iff[symmetric])
    thus p dvd x by (simp add: ppos)
  qed


  assume a * b = x * p
  hence p dvd a * b by simp
  hence nat p dvd nat (abs (a * b))
  apply (subst nat-dvd-iff, clarsimp)
  apply (rule conjI, clarsimp, simp add: zabs-def)
  proof (clarsimp)
    assume a: ~ 0 <= p
    from prime
        have 0 < p by (simp add: prime-def)
    from a and this
        have False by simp
    thus nat (abs (a * b)) = 0 ..
  qed
  hence nat p dvd (nat (abs a) * nat (abs b)) by (simp add: nat-abs-mult-distrib)
  hence nat p dvd nat (abs a) | nat p dvd nat (abs b) by (rule prime-dvd-mult[OF
prime])
  hence p dvd a | p dvd b by (fast intro: unnat)
  thus (EX x. a = x * p) | (EX x. b = x * p)
  proof
    assume p dvd a
    hence EX x. a = p * x by (simp add: dvd-def)
    from this obtain x
        where a = p * x by fast
    hence a = x * p by simp
    hence EX x. a = x * p by simp
    thus (EX x. a = x * p) | (EX x. b = x * p) ..
  next
    assume p dvd b
    hence EX x. b = p * x by (simp add: dvd-def)
    from this obtain x
        where b = p * x by fast
    hence b = x * p by simp
    hence EX x. b = x * p by simp
    thus (EX x. a = x * p) | (EX x. b = x * p) ..
  qed
next
  assume UNIV = {uu. EX x. uu = x * p}
  from this obtain x
      where 1 = x * p by fast
  from this [symmetric]
      have p * x = 1 by (subst zmult-commute)
  hence |p * x| = 1 by simp
```

hence $|p| = 1$ **by** (*rule abs-zmult-eq-1*)
from *this* **and** *prime*
show *False* **by** (*simp add*: *prime-def*)
qed

18.2.4 Ideals and Divisibility

lemma *int-Idl-subset-ideal*:
$Idl_{\mathcal{Z}}\ \{k\} \subseteq Idl_{\mathcal{Z}}\ \{l\} = (k \in Idl_{\mathcal{Z}}\ \{l\})$
by (*rule int.Idl-subset-ideal′*, *simp+*)

lemma *Idl-subset-eq-dvd*:
$(Idl_{\mathcal{Z}}\ \{k\} \subseteq Idl_{\mathcal{Z}}\ \{l\}) = (l\ dvd\ k)$
apply (*subst int-Idl-subset-ideal*, *subst int-Idl*, *simp*)
apply (*rule*, *clarify*)
apply (*simp add*: *dvd-def*, *clarify*)
apply (*simp add*: *int.m-comm*)
done

lemma *dvds-eq-Idl*:
$(l\ dvd\ k \wedge k\ dvd\ l) = (Idl_{\mathcal{Z}}\ \{k\} = Idl_{\mathcal{Z}}\ \{l\})$
proof −
have a: $l\ dvd\ k = (Idl_{\mathcal{Z}}\ \{k\} \subseteq Idl_{\mathcal{Z}}\ \{l\})$ **by** (*rule Idl-subset-eq-dvd*[*symmetric*])
have b: $k\ dvd\ l = (Idl_{\mathcal{Z}}\ \{l\} \subseteq Idl_{\mathcal{Z}}\ \{k\})$ **by** (*rule Idl-subset-eq-dvd*[*symmetric*])

have $(l\ dvd\ k \wedge k\ dvd\ l) = ((Idl_{\mathcal{Z}}\ \{k\} \subseteq Idl_{\mathcal{Z}}\ \{l\}) \wedge (Idl_{\mathcal{Z}}\ \{l\} \subseteq Idl_{\mathcal{Z}}\ \{k\}))$
by (*subst a*, *subst b*, *simp*)
also have $((Idl_{\mathcal{Z}}\ \{k\} \subseteq Idl_{\mathcal{Z}}\ \{l\}) \wedge (Idl_{\mathcal{Z}}\ \{l\} \subseteq Idl_{\mathcal{Z}}\ \{k\})) = (Idl_{\mathcal{Z}}\ \{k\} = Idl_{\mathcal{Z}}\ \{l\})$ **by** (*rule*, *fast+*)
finally
show *?thesis* **.**
qed

lemma *Idl-eq-abs*:
$(Idl_{\mathcal{Z}}\ \{k\} = Idl_{\mathcal{Z}}\ \{l\}) = (abs\ l = abs\ k)$
apply (*subst dvds-eq-abseq*[*symmetric*])
apply (*rule dvds-eq-Idl*[*symmetric*])
done

18.2.5 Ideals and the Modulus

constdefs
ZMod :: *int* => *int* => *int set*
$ZMod\ k\ r == (Idl_{\mathcal{Z}}\ \{k\}) +>_{\mathcal{Z}} r$

lemmas *ZMod-defs* =
ZMod-def genideal-def

lemma *rcos-zfact*:
assumes *kIl*: $k \in ZMod\ l\ r$

```
  shows EX x. k = x * l + r
proof -
  from kIl[unfolded ZMod-def]
    have ∃xl∈Idl_Z {l}. k = xl + r by (simp add: a-r-coset-defs int-ring-def)
  from this obtain xl
    where xl: xl ∈ Idl_Z {l}
    and k: k = xl + r
    by auto
  from xl obtain x
    where xl = x * l
    by (simp add: int-Idl, fast)
  from k and this
    have k = x * l + r by simp
  thus ∃x. k = x * l + r ..
qed

lemma ZMod-imp-zmod:
  assumes zmods: ZMod m a = ZMod m b
  shows a mod m = b mod m
proof -
  interpret ideal [Idl_Z {m} Z] by (rule int.genideal-ideal, fast)
  from zmods
    have b ∈ ZMod m a
    unfolding ZMod-def
    by (simp add: a-repr-independenceD)
  from this
    have EX x. b = x * m + a by (rule rcos-zfact)
  from this obtain x
    where b = x * m + a
    by fast

  hence b mod m = (x * m + a) mod m by simp
  also
    have ... = ((x * m) mod m) + (a mod m) by (simp add: zmod-zadd1-eq)
  also
    have ... = a mod m by simp
  finally
    have b mod m = a mod m .
  thus a mod m = b mod m ..
qed

lemma ZMod-mod:
  shows ZMod m a = ZMod m (a mod m)
proof -
  interpret ideal [Idl_Z {m} Z] by (rule int.genideal-ideal, fast)
  show ?thesis
    unfolding ZMod-def
  apply (rule a-repr-independence'[symmetric])
  apply (simp add: int-Idl a-r-coset-defs)
```

apply (*simp add*: *int-ring-def*)
proof −
have $a = m * (a \; div \; m) + (a \; mod \; m)$ **by** (*simp add*: *zmod-zdiv-equality*)
hence $a = (a \; div \; m) * m + (a \; mod \; m)$ **by** *simp*
thus $\exists h. (\exists x. h = x * m) \wedge a = h + a \; mod \; m$ **by** *fast*
qed *simp*
qed

lemma *zmod-imp-ZMod*:
assumes *modeq*: $a \; mod \; m = b \; mod \; m$
shows *ZMod* $m \; a$ = *ZMod* $m \; b$
proof −
have *ZMod* $m \; a$ = *ZMod* $m \; (a \; mod \; m)$ **by** (*rule ZMod-mod*)
also have ... = *ZMod* $m \; (b \; mod \; m)$ **by** (*simp add*: *modeq*[*symmetric*])
also have ... = *ZMod* $m \; b$ **by** (*rule ZMod-mod*[*symmetric*])
finally show *?thesis* **.**
qed

corollary *ZMod-eq-mod*:
shows (*ZMod* $m \; a$ = *ZMod* $m \; b$) = ($a \; mod \; m = b \; mod \; m$)
by (*rule*, *erule ZMod-imp-zmod*, *erule zmod-imp-ZMod*)

18.2.6 Factorization

constdefs
ZFact :: *int* ⇒ *int set ring*
ZFact k == $\mathcal{Z}$ *Quot* (*Idl*$_{\mathcal{Z}}$ $\{k\}$)

lemmas *ZFact-defs* = *ZFact-def FactRing-def*

lemma *ZFact-is-cring*:
shows *cring* (*ZFact* k)
apply (*unfold ZFact-def*)
apply (*rule ideal.quotient-is-cring*)
apply (*intro ring.genideal-ideal*)
apply (*simp add*: *cring.axioms*[*OF int-is-cring*] *ring.intro*)
apply *simp*
apply (*rule int-is-cring*)
done

lemma *ZFact-zero*:
carrier (*ZFact 0*) = $(\bigcup a. \{\{a\}\})$
apply (*insert int.genideal-zero*)
apply (*simp add*: *ZFact-defs A-RCOSETS-defs r-coset-def int-ring-def ring-record-simps*)
done

lemma *ZFact-one*:
carrier (*ZFact 1*) = {*UNIV*}
apply (*simp only*: *ZFact-defs A-RCOSETS-defs r-coset-def int-ring-def ring-record-simps*)

apply (*subst int.genideal-one*[*unfolded int-ring-def*, *simplified ring-record-simps*])
apply (*rule*, *rule*, *clarsimp*)
 apply (*rule*, *rule*, *clarsimp*)
 apply (*rule*, *clarsimp*, *arith*)
apply (*rule*, *clarsimp*)
apply (*rule exI*[*of - 0*], *clarsimp*)
done

lemma *ZFact-prime-is-domain*:
  assumes *pprime*: *prime* (*nat p*)
  shows *domain* (*ZFact p*)
apply (*unfold ZFact-def*)
apply (*rule primeideal.quotient-is-domain*)
apply (*rule prime-primeideal*[*OF pprime*])
done

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.