Lattices and Orders in Isabelle/HOL

Markus Wenzel
TU München

October 1, 2005

Abstract

We consider abstract structures of orders and lattices. Many fundamental concepts of lattice theory are developed, including dual structures, properties of bounds versus algebraic laws, lattice operations versus set-theoretic ones etc. We also give example instantiations of lattices and orders, such as direct products and function spaces. Well-known properties are demonstrated, like the Knaster-Tarski Theorem for complete lattices.

This formal theory development may serve as an example of applying Isabelle/HOL to the domain of mathematical reasoning about "axiomatic" structures. Apart from the simply-typed classical set-theory of HOL, we employ Isabelle's system of axiomatic type classes for expressing structures and functors in a light-weight manner. Proofs are expressed in the Isar language for readable formal proof, while aiming at its "best-style" of representing formal reasoning.

Contents

1 Orders

theory *Orders* **imports** *Main* **begin**

1.1 Ordered structures

We define several classes of ordered structures over some type $'a$ with relation $\sqsubseteq :: 'a \Rightarrow 'a \Rightarrow bool$. For a *quasi-order* that relation is required to be reflexive and transitive, for a *partial order* it also has to be anti-symmetric, while for a *linear order* all elements are required to be related (in either direction).

axclass *leq* < *type*
consts
  $leq :: 'a::leq \Rightarrow 'a \Rightarrow bool$ (**infixl** [= *50*)
syntax (*xsymbols*)
  $leq :: 'a::leq \Rightarrow 'a \Rightarrow bool$ (**infixl** $\sqsubseteq$ *50*)

axclass *quasi-order* < *leq*
  leq-refl [*intro?*]: $x \sqsubseteq x$
  leq-trans [*trans*]: $x \sqsubseteq y \Longrightarrow y \sqsubseteq z \Longrightarrow x \sqsubseteq z$

axclass *partial-order* < *quasi-order*
  leq-antisym [*trans*]: $x \sqsubseteq y \Longrightarrow y \sqsubseteq x \Longrightarrow x = y$

axclass *linear-order* < *partial-order*
  leq-linear: $x \sqsubseteq y \vee y \sqsubseteq x$

lemma *linear-order-cases*:
  $((x::'a::linear\text{-}order) \sqsubseteq y \Longrightarrow C) \Longrightarrow (y \sqsubseteq x \Longrightarrow C) \Longrightarrow C$
  by (*insert leq-linear*) *blast*

1.2 Duality

The *dual* of an ordered structure is an isomorphic copy of the underlying type, with the $\sqsubseteq$ relation defined as the inverse of the original one.

datatype $'a$ *dual* = *dual* $'a$

consts
  undual :: $'a$ *dual* $\Rightarrow$ $'a$
primrec
  undual-dual: *undual* (*dual* x) = x

instance *dual* :: (*leq*) *leq* **..**

defs (**overloaded**)
  leq-dual-def: $x' \sqsubseteq y' \equiv undual\ y' \sqsubseteq undual\ x'$

lemma *undual-leq* [*iff?*]: $(undual\ x' \sqsubseteq undual\ y') = (y' \sqsubseteq x')$
  by (*simp add*: *leq-dual-def*)

lemma *dual-leq* [*iff?*]: $(dual\ x \sqsubseteq dual\ y) = (y \sqsubseteq x)$
  by (*simp add*: *leq-dual-def*)

Functions *dual* and *undual* are inverse to each other; this entails the following fundamental properties.

lemma *dual-undual* [*simp*]: $dual\ (undual\ x') = x'$
  by (*cases* x') *simp*

lemma *undual-dual-id* [*simp*]: $undual\ o\ dual = id$
  by (*rule ext*) *simp*

lemma *dual-undual-id* [*simp*]: $dual\ o\ undual = id$
  by (*rule ext*) *simp*

Since *dual* (and *undual*) are both injective and surjective, the basic logical connectives (equality, quantification etc.) are transferred as follows.

lemma *undual-equality* [*iff?*]: $(undual\ x' = undual\ y') = (x' = y')$
  by (*cases* x', *cases* y') *simp*

lemma *dual-equality* [*iff?*]: $(dual\ x = dual\ y) = (x = y)$
  by *simp*

lemma *dual-ball* [*iff?*]: $(\forall x \in A.\ P\ (dual\ x)) = (\forall x' \in dual\ `\ A.\ P\ x')$
proof
  assume *a*: $\forall x \in A.\ P\ (dual\ x)$
  show $\forall x' \in dual\ `\ A.\ P\ x'$
  proof
    fix x' **assume** x': $x' \in dual\ `\ A$
    have $undual\ x' \in A$
    proof −
      from x' **have** $undual\ x' \in undual\ `\ dual\ `\ A$ **by** *simp*
      thus $undual\ x' \in A$ **by** (*simp add*: *image-compose* [*symmetric*])
    qed
    with *a* **have** $P\ (dual\ (undual\ x'))$ **..**
    also have $\ldots = x'$ **by** *simp*
    finally show $P\ x'$ **.**
  qed
next
  assume *a*: $\forall x' \in dual\ `\ A.\ P\ x'$
  show $\forall x \in A.\ P\ (dual\ x)$
  proof
    fix x **assume** $x \in A$
    hence $dual\ x \in dual\ `\ A$ **by** *simp*
    with *a* **show** $P\ (dual\ x)$ **..**
  qed
qed

lemma *range-dual* [*simp*]: *dual* ' *UNIV* = *UNIV*
proof (*rule surj-range*)
  have $\bigwedge x'.$ *dual* (*undual* x') = x' **by** *simp*
  thus *surj dual* **by** (*rule surjI*)
qed

lemma *dual-all* [*iff?*]: ($\forall x.$ P (*dual* x)) = ($\forall x'.$ P x')
proof −
  have ($\forall x \in$ *UNIV*. P (*dual* x)) = ($\forall x' \in$ *dual* ' *UNIV*. P x')
    by (*rule dual-ball*)
  thus *?thesis* **by** *simp*
qed

lemma *dual-ex*: ($\exists x.$ P (*dual* x)) = ($\exists x'.$ P x')
proof −
  have ($\forall x.$ $\neg$ P (*dual* x)) = ($\forall x'.$ $\neg$ P x')
    by (*rule dual-all*)
  thus *?thesis* **by** *blast*
qed

lemma *dual-Collect*: {*dual* x| x. P (*dual* x)} = {x'. P x'}
proof −
  have {*dual* x| x. P (*dual* x)} = {x'. $\exists x''.$ $x' = x'' \wedge P$ x''}
    by (*simp only*: *dual-ex* [*symmetric*])
  thus *?thesis* **by** *blast*
qed

1.3 Transforming orders

1.3.1 Duals

The classes of quasi, partial, and linear orders are all closed under formation of dual structures.

instance *dual* :: (*quasi-order*) *quasi-order*
proof
  fix x' y' z' :: '*a*::*quasi-order dual*
  have *undual* $x' \sqsubseteq$ *undual* x' **..** **thus** $x' \sqsubseteq x'$ **..**
  assume $y' \sqsubseteq z'$ **hence** *undual* $z' \sqsubseteq$ *undual* y' **..**
  also assume $x' \sqsubseteq y'$ **hence** *undual* $y' \sqsubseteq$ *undual* x' **..**
  finally show $x' \sqsubseteq z'$ **..**
qed

instance *dual* :: (*partial-order*) *partial-order*
proof
  fix x' y' :: '*a*::*partial-order dual*
  assume $y' \sqsubseteq x'$ **hence** *undual* $x' \sqsubseteq$ *undual* y' **..**
  also assume $x' \sqsubseteq y'$ **hence** *undual* $y' \sqsubseteq$ *undual* x' **..**
  finally show $x' = y'$ **..**
qed

instance *dual* :: (*linear-order*) *linear-order*
proof
  fix x' y' :: $'a$::*linear-order dual*
  show $x' \sqsubseteq y' \vee y' \sqsubseteq x'$
  proof (*rule linear-order-cases*)
    assume *undual* $y' \sqsubseteq$ *undual* x'
    hence $x' \sqsubseteq y'$ **..** **thus** *?thesis* **..**
  next
    assume *undual* $x' \sqsubseteq$ *undual* y'
    hence $y' \sqsubseteq x'$ **..** **thus** *?thesis* **..**
  qed
qed

1.3.2 Binary products

The classes of quasi and partial orders are closed under binary products. Note that the direct product of linear orders need *not* be linear in general.

instance $*$ :: (*leq*, *leq*) *leq* **..**

defs (**overloaded**)
  leq-prod-def: $p \sqsubseteq q \equiv$ *fst* $p \sqsubseteq$ *fst* $q \wedge$ *snd* $p \sqsubseteq$ *snd* q

lemma *leq-prodI* [*intro?*]:
    fst $p \sqsubseteq$ *fst* $q \Longrightarrow$ *snd* $p \sqsubseteq$ *snd* $q \Longrightarrow p \sqsubseteq q$
  by (*unfold leq-prod-def*) *blast*

lemma *leq-prodE* [*elim?*]:
    $p \sqsubseteq q \Longrightarrow$ (*fst* $p \sqsubseteq$ *fst* $q \Longrightarrow$ *snd* $p \sqsubseteq$ *snd* $q \Longrightarrow C) \Longrightarrow C$
  by (*unfold leq-prod-def*) *blast*

instance $*$ :: (*quasi-order*, *quasi-order*) *quasi-order*
proof
  fix p q r :: $'a$::*quasi-order* $\times$ $'b$::*quasi-order*
  show $p \sqsubseteq p$
  proof
    show *fst* $p \sqsubseteq$ *fst* p **..**
    show *snd* $p \sqsubseteq$ *snd* p **..**
  qed
  assume *pq*: $p \sqsubseteq q$ **and** *qr*: $q \sqsubseteq r$
  show $p \sqsubseteq r$
  proof
    from *pq* **have** *fst* $p \sqsubseteq$ *fst* q **..**
    also from *qr* **have** $\ldots \sqsubseteq$ *fst* r **..**
    finally show *fst* $p \sqsubseteq$ *fst* r **.**
    from *pq* **have** *snd* $p \sqsubseteq$ *snd* q **..**
    also from *qr* **have** $\ldots \sqsubseteq$ *snd* r **..**
    finally show *snd* $p \sqsubseteq$ *snd* r **.**
  qed

qed

instance $*$ :: (*partial-order*, *partial-order*) *partial-order*
proof
  fix p q :: $'a$::*partial-order* $\times$ $'b$::*partial-order*
  assume *pq*: $p \sqsubseteq q$ **and** *qp*: $q \sqsubseteq p$
  show $p = q$
  proof
    from *pq* **have** *fst* $p \sqsubseteq$ *fst* q **..**
    also from *qp* **have** $\ldots \sqsubseteq$ *fst* p **..**
    finally show *fst* $p =$ *fst* q **.**
    from *pq* **have** *snd* $p \sqsubseteq$ *snd* q **..**
    also from *qp* **have** $\ldots \sqsubseteq$ *snd* p **..**
    finally show *snd* $p =$ *snd* q **.**
  qed
qed

1.3.3 General products

The classes of quasi and partial orders are closed under general products (function spaces). Note that the direct product of linear orders need *not* be linear in general.

instance *fun* :: (*type*, *leq*) *leq* **..**

defs (**overloaded**)
  leq-fun-def: $f \sqsubseteq g \equiv \forall x.\ f\ x \sqsubseteq g\ x$

lemma *leq-funI* [*intro?*]: $(\bigwedge x.\ f\ x \sqsubseteq g\ x) \Longrightarrow f \sqsubseteq g$
  by (*unfold leq-fun-def*) *blast*

lemma *leq-funD* [*dest?*]: $f \sqsubseteq g \Longrightarrow f\ x \sqsubseteq g\ x$
  by (*unfold leq-fun-def*) *blast*

instance *fun* :: (*type*, *quasi-order*) *quasi-order*
proof
  fix f g h :: $'a \Rightarrow 'b$::*quasi-order*
  show $f \sqsubseteq f$
  proof
    fix x **show** $f\ x \sqsubseteq f\ x$ **..**
  qed
  assume *fg*: $f \sqsubseteq g$ **and** *gh*: $g \sqsubseteq h$
  show $f \sqsubseteq h$
  proof
    fix x **from** *fg* **have** $f\ x \sqsubseteq g\ x$ **..**
    also from *gh* **have** $\ldots \sqsubseteq h\ x$ **..**
    finally show $f\ x \sqsubseteq h\ x$ **.**
  qed
qed

instance *fun* :: (*type*, *partial-order*) *partial-order*
proof
  fix f g :: $'a \Rightarrow 'b$::*partial-order*
  assume *fg*: $f \sqsubseteq g$ **and** *gf*: $g \sqsubseteq f$
  show $f = g$
  proof
    fix x **from** *fg* **have** $f\ x \sqsubseteq g\ x$ **..**
    also from *gf* **have** $\ldots \sqsubseteq f\ x$ **..**
    finally show $f\ x = g\ x$ **.**
  qed
qed

end

2 Bounds

theory *Bounds* **imports** *Orders* **begin**

2.1 Infimum and supremum

Given a partial order, we define infimum (greatest lower bound) and supremum (least upper bound) wrt. $\sqsubseteq$ for two and for any number of elements.

constdefs
  is-inf :: $'a$::*partial-order* $\Rightarrow 'a \Rightarrow 'a \Rightarrow$ *bool*
  is-inf x y *inf* $\equiv$ *inf* $\sqsubseteq x \wedge$ *inf* $\sqsubseteq y \wedge (\forall z.\ z \sqsubseteq x \wedge z \sqsubseteq y \longrightarrow z \sqsubseteq$ *inf*$)$

  is-sup :: $'a$::*partial-order* $\Rightarrow 'a \Rightarrow 'a \Rightarrow$ *bool*
  is-sup x y *sup* $\equiv x \sqsubseteq$ *sup* $\wedge\ y \sqsubseteq$ *sup* $\wedge (\forall z.\ x \sqsubseteq z \wedge y \sqsubseteq z \longrightarrow$ *sup* $\sqsubseteq z)$

  is-Inf :: $'a$::*partial-order set* $\Rightarrow 'a \Rightarrow$ *bool*
  is-Inf A *inf* $\equiv (\forall x \in A.$ *inf* $\sqsubseteq x) \wedge (\forall z.\ (\forall x \in A.\ z \sqsubseteq x) \longrightarrow z \sqsubseteq$ *inf*$)$

  is-Sup :: $'a$::*partial-order set* $\Rightarrow 'a \Rightarrow$ *bool*
  is-Sup A *sup* $\equiv (\forall x \in A.\ x \sqsubseteq$ *sup*$) \wedge (\forall z.\ (\forall x \in A.\ x \sqsubseteq z) \longrightarrow$ *sup* $\sqsubseteq z)$

These definitions entail the following basic properties of boundary elements.

lemma *is-infI* [*intro?*]: *inf* $\sqsubseteq x \Longrightarrow$ *inf* $\sqsubseteq y \Longrightarrow$
    $(\bigwedge z.\ z \sqsubseteq x \Longrightarrow z \sqsubseteq y \Longrightarrow z \sqsubseteq$ *inf*$) \Longrightarrow$ *is-inf* x y *inf*
  by (*unfold is-inf-def*) *blast*

lemma *is-inf-greatest* [*elim?*]:
    is-inf x y *inf* $\Longrightarrow z \sqsubseteq x \Longrightarrow z \sqsubseteq y \Longrightarrow z \sqsubseteq$ *inf*
  by (*unfold is-inf-def*) *blast*

lemma *is-inf-lower* [*elim?*]:
    is-inf x y *inf* $\Longrightarrow$ (*inf* $\sqsubseteq x \Longrightarrow$ *inf* $\sqsubseteq y \Longrightarrow C) \Longrightarrow C$

by (*unfold is-inf-def*) *blast*

lemma *is-supI* [*intro?*]: $x \sqsubseteq sup \Longrightarrow y \sqsubseteq sup \Longrightarrow$
$(\bigwedge z.\ x \sqsubseteq z \Longrightarrow y \sqsubseteq z \Longrightarrow sup \sqsubseteq z) \Longrightarrow is\text{-}sup\ x\ y\ sup$
by (*unfold is-sup-def*) *blast*

lemma *is-sup-least* [*elim?*]:
$is\text{-}sup\ x\ y\ sup \Longrightarrow x \sqsubseteq z \Longrightarrow y \sqsubseteq z \Longrightarrow sup \sqsubseteq z$
by (*unfold is-sup-def*) *blast*

lemma *is-sup-upper* [*elim?*]:
$is\text{-}sup\ x\ y\ sup \Longrightarrow (x \sqsubseteq sup \Longrightarrow y \sqsubseteq sup \Longrightarrow C) \Longrightarrow C$
by (*unfold is-sup-def*) *blast*

lemma *is-InfI* [*intro?*]: $(\bigwedge x.\ x \in A \Longrightarrow inf \sqsubseteq x) \Longrightarrow$
$(\bigwedge z.\ (\forall x \in A.\ z \sqsubseteq x) \Longrightarrow z \sqsubseteq inf) \Longrightarrow is\text{-}Inf\ A\ inf$
by (*unfold is-Inf-def*) *blast*

lemma *is-Inf-greatest* [*elim?*]:
$is\text{-}Inf\ A\ inf \Longrightarrow (\bigwedge x.\ x \in A \Longrightarrow z \sqsubseteq x) \Longrightarrow z \sqsubseteq inf$
by (*unfold is-Inf-def*) *blast*

lemma *is-Inf-lower* [*dest?*]:
$is\text{-}Inf\ A\ inf \Longrightarrow x \in A \Longrightarrow inf \sqsubseteq x$
by (*unfold is-Inf-def*) *blast*

lemma *is-SupI* [*intro?*]: $(\bigwedge x.\ x \in A \Longrightarrow x \sqsubseteq sup) \Longrightarrow$
$(\bigwedge z.\ (\forall x \in A.\ x \sqsubseteq z) \Longrightarrow sup \sqsubseteq z) \Longrightarrow is\text{-}Sup\ A\ sup$
by (*unfold is-Sup-def*) *blast*

lemma *is-Sup-least* [*elim?*]:
$is\text{-}Sup\ A\ sup \Longrightarrow (\bigwedge x.\ x \in A \Longrightarrow x \sqsubseteq z) \Longrightarrow sup \sqsubseteq z$
by (*unfold is-Sup-def*) *blast*

lemma *is-Sup-upper* [*dest?*]:
$is\text{-}Sup\ A\ sup \Longrightarrow x \in A \Longrightarrow x \sqsubseteq sup$
by (*unfold is-Sup-def*) *blast*

2.2 Duality

Infimum and supremum are dual to each other.

theorem *dual-inf* [*iff?*]:
$is\text{-}inf\ (dual\ x)\ (dual\ y)\ (dual\ sup) = is\text{-}sup\ x\ y\ sup$
by (*simp add*: *is-inf-def is-sup-def dual-all* [*symmetric*] *dual-leq*)

theorem *dual-sup* [*iff?*]:

```
    is-sup (dual x) (dual y) (dual inf) = is-inf x y inf
  by (simp add: is-inf-def is-sup-def dual-all [symmetric] dual-leq)

theorem dual-Inf [iff?]:
    is-Inf (dual ‘ A) (dual sup) = is-Sup A sup
  by (simp add: is-Inf-def is-Sup-def dual-all [symmetric] dual-leq)

theorem dual-Sup [iff?]:
    is-Sup (dual ‘ A) (dual inf) = is-Inf A inf
  by (simp add: is-Inf-def is-Sup-def dual-all [symmetric] dual-leq)
```

2.3 Uniqueness

Infima and suprema on partial orders are unique; this is mainly due to antisymmetry of the underlying relation.

```
theorem is-inf-uniq: is-inf x y inf ⟹ is-inf x y inf' ⟹ inf = inf'
proof −
  assume inf: is-inf x y inf
  assume inf': is-inf x y inf'
  show ?thesis
  proof (rule leq-antisym)
    from inf' show inf ⊑ inf'
    proof (rule is-inf-greatest)
      from inf show inf ⊑ x ..
      from inf show inf ⊑ y ..
    qed
    from inf show inf' ⊑ inf
    proof (rule is-inf-greatest)
      from inf' show inf' ⊑ x ..
      from inf' show inf' ⊑ y ..
    qed
  qed
qed

theorem is-sup-uniq: is-sup x y sup ⟹ is-sup x y sup' ⟹ sup = sup'
proof −
  assume sup: is-sup x y sup and sup': is-sup x y sup'
  have dual sup = dual sup'
  proof (rule is-inf-uniq)
    from sup show is-inf (dual x) (dual y) (dual sup) ..
    from sup' show is-inf (dual x) (dual y) (dual sup') ..
  qed
  thus sup = sup' ..
qed

theorem is-Inf-uniq: is-Inf A inf ⟹ is-Inf A inf' ⟹ inf = inf'
proof −
  assume inf: is-Inf A inf
  assume inf': is-Inf A inf'
```

show *?thesis*
proof (*rule leq-antisym*)
from inf' **show** $inf \sqsubseteq inf'$
proof (*rule is-Inf-greatest*)
fix x **assume** $x \in A$
from inf **show** $inf \sqsubseteq x$ **..**
qed
from inf **show** $inf' \sqsubseteq inf$
proof (*rule is-Inf-greatest*)
fix x **assume** $x \in A$
from inf' **show** $inf' \sqsubseteq x$ **..**
qed
qed
qed

theorem *is-Sup-uniq*: *is-Sup A sup* $\Longrightarrow$ *is-Sup A sup'* $\Longrightarrow$ $sup = sup'$
proof −
assume *sup*: *is-Sup A sup* **and** *sup'*: *is-Sup A sup'*
have *dual sup* = *dual sup'*
proof (*rule is-Inf-uniq*)
from *sup* **show** *is-Inf* (*dual ‘ A*) (*dual sup*) **..**
from *sup'* **show** *is-Inf* (*dual ‘ A*) (*dual sup'*) **..**
qed
thus $sup = sup'$ **..**
qed

2.4 Related elements

The binary bound of related elements is either one of the argument.

theorem *is-inf-related* [*elim?*]: $x \sqsubseteq y \Longrightarrow$ *is-inf x y x*
proof −
assume $x \sqsubseteq y$
show *?thesis*
proof
show $x \sqsubseteq x$ **..**
show $x \sqsubseteq y$ **.**
fix z **assume** $z \sqsubseteq x$ **and** $z \sqsubseteq y$ **show** $z \sqsubseteq x$ **.**
qed
qed

theorem *is-sup-related* [*elim?*]: $x \sqsubseteq y \Longrightarrow$ *is-sup x y y*
proof −
assume $x \sqsubseteq y$
show *?thesis*
proof
show $x \sqsubseteq y$ **.**
show $y \sqsubseteq y$ **..**
fix z **assume** $x \sqsubseteq z$ **and** $y \sqsubseteq z$
show $y \sqsubseteq z$ **.**

```
  qed
qed
```

2.5 General versus binary bounds

General bounds of two-element sets coincide with binary bounds.

```
theorem is-Inf-binary: is-Inf {x, y} inf = is-inf x y inf
proof -
  let ?A = {x, y}
  show ?thesis
  proof
    assume is-Inf: is-Inf ?A inf
    show is-inf x y inf
    proof
      have x ∈ ?A by simp
      with is-Inf show inf ⊑ x ..
      have y ∈ ?A by simp
      with is-Inf show inf ⊑ y ..
      fix z assume zx: z ⊑ x and zy: z ⊑ y
      from is-Inf show z ⊑ inf
      proof (rule is-Inf-greatest)
        fix a assume a ∈ ?A
        hence a = x ∨ a = y by blast
        thus z ⊑ a
        proof
          assume a = x
          with zx show ?thesis by simp
        next
          assume a = y
          with zy show ?thesis by simp
        qed
      qed
    qed
  next
    assume is-inf: is-inf x y inf
    show is-Inf {x, y} inf
    proof
      fix a assume a ∈ ?A
      hence a = x ∨ a = y by blast
      thus inf ⊑ a
      proof
        assume a = x
        also from is-inf have inf ⊑ x ..
        finally show ?thesis .
      next
        assume a = y
        also from is-inf have inf ⊑ y ..
        finally show ?thesis .
      qed
```

```
    next
      fix z assume z: ∀ a ∈ ?A. z ⊑ a
      from is-inf show z ⊑ inf
      proof (rule is-inf-greatest)
        from z show z ⊑ x by blast
        from z show z ⊑ y by blast
      qed
    qed
  qed
qed
```

```
theorem is-Sup-binary: is-Sup {x, y} sup = is-sup x y sup
proof –
  have is-Sup {x, y} sup = is-Inf (dual ‘ {x, y}) (dual sup)
    by (simp only: dual-Inf)
  also have dual ‘ {x, y} = {dual x, dual y}
    by simp
  also have is-Inf ... (dual sup) = is-inf (dual x) (dual y) (dual sup)
    by (rule is-Inf-binary)
  also have ... = is-sup x y sup
    by (simp only: dual-inf)
  finally show ?thesis .
qed
```

2.6 Connecting general bounds

Either kind of general bounds is sufficient to express the other. The least upper bound (supremum) is the same as the the greatest lower bound of the set of all upper bounds; the dual statements holds as well; the dual statement holds as well.

```
theorem Inf-Sup: is-Inf {b. ∀ a ∈ A. a ⊑ b} sup ⟹ is-Sup A sup
proof –
  let ?B = {b. ∀ a ∈ A. a ⊑ b}
  assume is-Inf: is-Inf ?B sup
  show is-Sup A sup
  proof
    fix x assume x: x ∈ A
    from is-Inf show x ⊑ sup
    proof (rule is-Inf-greatest)
      fix y assume y ∈ ?B
      hence ∀ a ∈ A. a ⊑ y ..
      from this x show x ⊑ y ..
    qed
  next
    fix z assume ∀ x ∈ A. x ⊑ z
    hence z ∈ ?B ..
    with is-Inf show sup ⊑ z ..
  qed
```

qed

theorem *Sup-Inf*: *is-Sup* $\{b. \forall a \in A.\ b \sqsubseteq a\}$ *inf* $\Longrightarrow$ *is-Inf A inf*
proof –
  assume *is-Sup* $\{b. \forall a \in A.\ b \sqsubseteq a\}$ *inf*
  hence *is-Inf* (*dual* ‘ $\{b. \forall a \in A.$ *dual a* $\sqsubseteq$ *dual b*$\}$) (*dual inf*)
    by (*simp only*: *dual-Inf dual-leq*)
  also have *dual* ‘ $\{b. \forall a \in A.$ *dual a* $\sqsubseteq$ *dual b*$\}$ = $\{b'. \forall a' \in$ *dual* ‘ $A.\ a' \sqsubseteq b'\}$
    by (*auto iff*: *dual-ball dual-Collect simp add*: *image-Collect*)
  finally have *is-Inf* ... (*dual inf*) **.**
  hence *is-Sup* (*dual* ‘ *A*) (*dual inf*)
    by (*rule Inf-Sup*)
  thus *?thesis* **..**
qed

end

3 Lattices

theory *Lattice* **imports** *Bounds* **begin**

3.1 Lattice operations

A *lattice* is a partial order with infimum and supremum of any two elements (thus any *finite* number of elements have bounds as well).

axclass *lattice* $\subseteq$ *partial-order*
  ex-inf: $\exists$ *inf*. *is-inf x y inf*
  ex-sup: $\exists$ *sup*. *is-sup x y sup*

The $\sqcap$ (meet) and $\sqcup$ (join) operations select such infimum and supremum elements.

consts
  meet :: *'a::lattice* $\Rightarrow$ *'a* $\Rightarrow$ *'a* (**infixl** && *70*)
  join :: *'a::lattice* $\Rightarrow$ *'a* $\Rightarrow$ *'a* (**infixl** || *65*)
syntax (*xsymbols*)
  meet :: *'a::lattice* $\Rightarrow$ *'a* $\Rightarrow$ *'a* (**infixl** $\sqcap$ *70*)
  join :: *'a::lattice* $\Rightarrow$ *'a* $\Rightarrow$ *'a* (**infixl** $\sqcup$ *65*)
defs
  meet-def: *x* && *y* $\equiv$ *THE inf*. *is-inf x y inf*
  join-def: *x* || *y* $\equiv$ *THE sup*. *is-sup x y sup*

Due to unique existence of bounds, the lattice operations may be exhibited as follows.

lemma *meet-equality* [*elim?*]: *is-inf x y inf* $\Longrightarrow$ $x \sqcap y$ = *inf*
proof (*unfold meet-def*)
  assume *is-inf x y inf*

thus $(THE\ inf.\ is\text{-}inf\ x\ y\ inf) = inf$
by $(rule\ the\text{-}equality)\ (rule\ is\text{-}inf\text{-}uniq)$
qed

lemma *meetI* [*intro?*]:
$inf \sqsubseteq x \Longrightarrow inf \sqsubseteq y \Longrightarrow (\bigwedge z.\ z \sqsubseteq x \Longrightarrow z \sqsubseteq y \Longrightarrow z \sqsubseteq inf) \Longrightarrow x \sqcap y = inf$
by $(rule\ meet\text{-}equality,\ rule\ is\text{-}infI)\ blast+$

lemma *join-equality* [*elim?*]: $is\text{-}sup\ x\ y\ sup \Longrightarrow x \sqcup y = sup$
proof $(unfold\ join\text{-}def)$
assume $is\text{-}sup\ x\ y\ sup$
thus $(THE\ sup.\ is\text{-}sup\ x\ y\ sup) = sup$
by $(rule\ the\text{-}equality)\ (rule\ is\text{-}sup\text{-}uniq)$
qed

lemma *joinI* [*intro?*]: $x \sqsubseteq sup \Longrightarrow y \sqsubseteq sup \Longrightarrow$
$(\bigwedge z.\ x \sqsubseteq z \Longrightarrow y \sqsubseteq z \Longrightarrow sup \sqsubseteq z) \Longrightarrow x \sqcup y = sup$
by $(rule\ join\text{-}equality,\ rule\ is\text{-}supI)\ blast+$

The $\sqcap$ and $\sqcup$ operations indeed determine bounds on a lattice structure.

lemma *is-inf-meet* [*intro?*]: $is\text{-}inf\ x\ y\ (x \sqcap y)$
proof $(unfold\ meet\text{-}def)$
from *ex-inf* **obtain** *inf* **where** $is\text{-}inf\ x\ y\ inf$ **..**
thus $is\text{-}inf\ x\ y\ (THE\ inf.\ is\text{-}inf\ x\ y\ inf)$ **by** $(rule\ theI)\ (rule\ is\text{-}inf\text{-}uniq)$
qed

lemma *meet-greatest* [*intro?*]: $z \sqsubseteq x \Longrightarrow z \sqsubseteq y \Longrightarrow z \sqsubseteq x \sqcap y$
by $(rule\ is\text{-}inf\text{-}greatest)\ (rule\ is\text{-}inf\text{-}meet)$

lemma *meet-lower1* [*intro?*]: $x \sqcap y \sqsubseteq x$
by $(rule\ is\text{-}inf\text{-}lower)\ (rule\ is\text{-}inf\text{-}meet)$

lemma *meet-lower2* [*intro?*]: $x \sqcap y \sqsubseteq y$
by $(rule\ is\text{-}inf\text{-}lower)\ (rule\ is\text{-}inf\text{-}meet)$

lemma *is-sup-join* [*intro?*]: $is\text{-}sup\ x\ y\ (x \sqcup y)$
proof $(unfold\ join\text{-}def)$
from *ex-sup* **obtain** *sup* **where** $is\text{-}sup\ x\ y\ sup$ **..**
thus $is\text{-}sup\ x\ y\ (THE\ sup.\ is\text{-}sup\ x\ y\ sup)$ **by** $(rule\ theI)\ (rule\ is\text{-}sup\text{-}uniq)$
qed

lemma *join-least* [*intro?*]: $x \sqsubseteq z \Longrightarrow y \sqsubseteq z \Longrightarrow x \sqcup y \sqsubseteq z$
by $(rule\ is\text{-}sup\text{-}least)\ (rule\ is\text{-}sup\text{-}join)$

lemma *join-upper1* [*intro?*]: $x \sqsubseteq x \sqcup y$
by $(rule\ is\text{-}sup\text{-}upper)\ (rule\ is\text{-}sup\text{-}join)$

lemma *join-upper2* [*intro?*]: $y \sqsubseteq x \sqcup y$

by (*rule is-sup-upper*) (*rule is-sup-join*)

3.2 Duality

The class of lattices is closed under formation of dual structures. This means that for any theorem of lattice theory, the dualized statement holds as well; this important fact simplifies many proofs of lattice theory.

instance *dual* :: (*lattice*) *lattice*
proof
  fix x' y' :: $'a$::*lattice dual*
  show $\exists$ *inf'*. *is-inf* x' y' *inf'*
  proof −
    have $\exists$ *sup*. *is-sup* (*undual* x') (*undual* y') *sup* **by** (*rule ex-sup*)
    hence $\exists$ *sup*. *is-inf* (*dual* (*undual* x')) (*dual* (*undual* y')) (*dual sup*)
      by (*simp only*: *dual-inf*)
    thus *?thesis* **by** (*simp add*: *dual-ex* [*symmetric*])
  qed
  show $\exists$ *sup'*. *is-sup* x' y' *sup'*
  proof −
    have $\exists$ *inf*. *is-inf* (*undual* x') (*undual* y') *inf* **by** (*rule ex-inf*)
    hence $\exists$ *inf*. *is-sup* (*dual* (*undual* x')) (*dual* (*undual* y')) (*dual inf*)
      by (*simp only*: *dual-sup*)
    thus *?thesis* **by** (*simp add*: *dual-ex* [*symmetric*])
  qed
qed

Apparently, the $\sqcap$ and $\sqcup$ operations are dual to each other.

theorem *dual-meet* [*intro?*]: *dual* $(x \sqcap y)$ = *dual* $x \sqcup$ *dual* y
proof −
  from *is-inf-meet* **have** *is-sup* (*dual* x) (*dual* y) (*dual* $(x \sqcap y)$) **..**
  hence *dual* $x \sqcup$ *dual* y = *dual* $(x \sqcap y)$ **..**
  thus *?thesis* **..**
qed

theorem *dual-join* [*intro?*]: *dual* $(x \sqcup y)$ = *dual* $x \sqcap$ *dual* y
proof −
  from *is-sup-join* **have** *is-inf* (*dual* x) (*dual* y) (*dual* $(x \sqcup y)$) **..**
  hence *dual* $x \sqcap$ *dual* y = *dual* $(x \sqcup y)$ **..**
  thus *?thesis* **..**
qed

3.3 Algebraic properties

The $\sqcap$ and $\sqcup$ operations have the following characteristic algebraic properties: associative (A), commutative (C), and absorptive (AB).

theorem *meet-assoc*: $(x \sqcap y) \sqcap z = x \sqcap (y \sqcap z)$
proof

show $x \sqcap (y \sqcap z) \sqsubseteq x \sqcap y$
proof
show $x \sqcap (y \sqcap z) \sqsubseteq x$ **..**
show $x \sqcap (y \sqcap z) \sqsubseteq y$
proof $-$
have $x \sqcap (y \sqcap z) \sqsubseteq y \sqcap z$ **..**
also have $\ldots \sqsubseteq y$ **..**
finally show *?thesis* **.**
qed
qed
show $x \sqcap (y \sqcap z) \sqsubseteq z$
proof $-$
have $x \sqcap (y \sqcap z) \sqsubseteq y \sqcap z$ **..**
also have $\ldots \sqsubseteq z$ **..**
finally show *?thesis* **.**
qed
fix w **assume** $w \sqsubseteq x \sqcap y$ **and** $w \sqsubseteq z$
show $w \sqsubseteq x \sqcap (y \sqcap z)$
proof
show $w \sqsubseteq x$
proof $-$
have $w \sqsubseteq x \sqcap y$ **.**
also have $\ldots \sqsubseteq x$ **..**
finally show *?thesis* **.**
qed
show $w \sqsubseteq y \sqcap z$
proof
show $w \sqsubseteq y$
proof $-$
have $w \sqsubseteq x \sqcap y$ **.**
also have $\ldots \sqsubseteq y$ **..**
finally show *?thesis* **.**
qed
show $w \sqsubseteq z$ **.**
qed
qed
qed

theorem *join-assoc*: $(x \sqcup y) \sqcup z = x \sqcup (y \sqcup z)$
proof $-$
have $dual\ ((x \sqcup y) \sqcup z) = (dual\ x \sqcap dual\ y) \sqcap dual\ z$
by (*simp only*: *dual-join*)
also have $\ldots = dual\ x \sqcap (dual\ y \sqcap dual\ z)$
by (*rule meet-assoc*)
also have $\ldots = dual\ (x \sqcup (y \sqcup z))$
by (*simp only*: *dual-join*)
finally show *?thesis* **..**
qed

theorem *meet-commute*: $x \sqcap y = y \sqcap x$
proof
  show $y \sqcap x \sqsubseteq x$ **..**
  show $y \sqcap x \sqsubseteq y$ **..**
  fix z **assume** $z \sqsubseteq x$ **and** $z \sqsubseteq y$
  show $z \sqsubseteq y \sqcap x$ **..**
qed

theorem *join-commute*: $x \sqcup y = y \sqcup x$
proof $-$
  have $dual\ (x \sqcup y) = dual\ x \sqcap dual\ y$ **..**
  also have $\ldots = dual\ y \sqcap dual\ x$
    by (*rule meet-commute*)
  also have $\ldots = dual\ (y \sqcup x)$
    by (*simp only*: *dual-join*)
  finally show *?thesis* **..**
qed

theorem *meet-join-absorb*: $x \sqcap (x \sqcup y) = x$
proof
  show $x \sqsubseteq x$ **..**
  show $x \sqsubseteq x \sqcup y$ **..**
  fix z **assume** $z \sqsubseteq x$ **and** $z \sqsubseteq x \sqcup y$
  show $z \sqsubseteq x$ **.**
qed

theorem *join-meet-absorb*: $x \sqcup (x \sqcap y) = x$
proof $-$
  have $dual\ x \sqcap (dual\ x \sqcup dual\ y) = dual\ x$
    by (*rule meet-join-absorb*)
  hence $dual\ (x \sqcup (x \sqcap y)) = dual\ x$
    by (*simp only*: *dual-meet dual-join*)
  thus *?thesis* **..**
qed

Some further algebraic properties hold as well. The property idempotent (I) is a basic algebraic consequence of (AB).

theorem *meet-idem*: $x \sqcap x = x$
proof $-$
  have $x \sqcap (x \sqcup (x \sqcap x)) = x$ **by** (*rule meet-join-absorb*)
  also have $x \sqcup (x \sqcap x) = x$ **by** (*rule join-meet-absorb*)
  finally show *?thesis* **.**
qed

theorem *join-idem*: $x \sqcup x = x$
proof $-$
  have $dual\ x \sqcap dual\ x = dual\ x$
    by (*rule meet-idem*)
  hence $dual\ (x \sqcup x) = dual\ x$

by (*simp only*: *dual-join*)
thus *?thesis* **..**
qed

Meet and join are trivial for related elements.

theorem *meet-related* [*elim?*]: $x \sqsubseteq y \Longrightarrow x \sqcap y = x$
proof
assume $x \sqsubseteq y$
show $x \sqsubseteq x$ **..**
show $x \sqsubseteq y$ **.**
fix z **assume** $z \sqsubseteq x$ **and** $z \sqsubseteq y$ **show** $z \sqsubseteq x$ **.**
qed

theorem *join-related* [*elim?*]: $x \sqsubseteq y \Longrightarrow x \sqcup y = y$
proof −
assume $x \sqsubseteq y$ **hence** *dual* $y \sqsubseteq$ *dual* x **..**
hence *dual* $y \sqcap$ *dual* $x =$ *dual* y **by** (*rule meet-related*)
also have *dual* $y \sqcap$ *dual* $x =$ *dual* $(y \sqcup x)$ **by** (*simp only*: *dual-join*)
also have $y \sqcup x = x \sqcup y$ **by** (*rule join-commute*)
finally show *?thesis* **..**
qed

3.4 Order versus algebraic structure

The $\sqcap$ and $\sqcup$ operations are connected with the underlying $\sqsubseteq$ relation in a canonical manner.

theorem *meet-connection*: $(x \sqsubseteq y) = (x \sqcap y = x)$
proof
assume $x \sqsubseteq y$
hence *is-inf* $x\ y\ x$ **..**
thus $x \sqcap y = x$ **..**
next
have $x \sqcap y \sqsubseteq y$ **..**
also assume $x \sqcap y = x$
finally show $x \sqsubseteq y$ **.**
qed

theorem *join-connection*: $(x \sqsubseteq y) = (x \sqcup y = y)$
proof
assume $x \sqsubseteq y$
hence *is-sup* $x\ y\ y$ **..**
thus $x \sqcup y = y$ **..**
next
have $x \sqsubseteq x \sqcup y$ **..**
also assume $x \sqcup y = y$
finally show $x \sqsubseteq y$ **.**
qed

The most fundamental result of the meta-theory of lattices is as follows (we do not prove it here).

Given a structure with binary operations $\sqcap$ and $\sqcup$ such that (A), (C), and (AB) hold (cf. §3.3). This structure represents a lattice, if the relation $x \sqsubseteq y$ is defined as $Lattice.meet\ x\ y = x$ (alternatively as $Lattice.join\ x\ y = y$). Furthermore, infimum and supremum with respect to this ordering coincide with the original $\sqcap$ and $\sqcup$ operations.

3.5 Example instances

3.5.1 Linear orders

Linear orders with *minimum* and *maximum* operations are a (degenerate) example of lattice structures.

constdefs
 $minimum :: 'a::linear\text{-}order \Rightarrow 'a \Rightarrow 'a$
 $minimum\ x\ y \equiv (if\ x \sqsubseteq y\ then\ x\ else\ y)$
 $maximum :: 'a::linear\text{-}order \Rightarrow 'a \Rightarrow 'a$
 $maximum\ x\ y \equiv (if\ x \sqsubseteq y\ then\ y\ else\ x)$

lemma *is-inf-minimum*: *is-inf x y* (*minimum x y*)
proof
 let *?min* = *minimum x y*
 from *leq-linear* **show** *?min* $\sqsubseteq$ *x* **by** (*auto simp add*: *minimum-def*)
 from *leq-linear* **show** *?min* $\sqsubseteq$ *y* **by** (*auto simp add*: *minimum-def*)
 fix *z* **assume** *z* $\sqsubseteq$ *x* **and** *z* $\sqsubseteq$ *y*
 with *leq-linear* **show** *z* $\sqsubseteq$ *?min* **by** (*auto simp add*: *minimum-def*)
qed

lemma *is-sup-maximum*: *is-sup x y* (*maximum x y*)
proof
 let *?max* = *maximum x y*
 from *leq-linear* **show** *x* $\sqsubseteq$ *?max* **by** (*auto simp add*: *maximum-def*)
 from *leq-linear* **show** *y* $\sqsubseteq$ *?max* **by** (*auto simp add*: *maximum-def*)
 fix *z* **assume** *x* $\sqsubseteq$ *z* **and** *y* $\sqsubseteq$ *z*
 with *leq-linear* **show** *?max* $\sqsubseteq$ *z* **by** (*auto simp add*: *maximum-def*)
qed

instance *linear-order* $\subseteq$ *lattice*
proof
 fix $x\ y :: 'a::linear\text{-}order$
 from *is-inf-minimum* **show** $\exists\, inf.\ is\text{-}inf\ x\ y\ inf$ **..**
 from *is-sup-maximum* **show** $\exists\, sup.\ is\text{-}sup\ x\ y\ sup$ **..**
qed

The lattice operations on linear orders indeed coincide with *minimum* and *maximum*.

theorem *meet-mimimum*: $x \sqcap y = \mathit{minimum}\ x\ y$
  by (*rule meet-equality*) (*rule is-inf-minimum*)

theorem *meet-maximum*: $x \sqcup y = \mathit{maximum}\ x\ y$
  by (*rule join-equality*) (*rule is-sup-maximum*)

3.5.2 Binary products

The class of lattices is closed under direct binary products (cf. §1.3.2).

lemma *is-inf-prod*: *is-inf* $p\ q$ ($\mathit{fst}\ p \sqcap \mathit{fst}\ q$, $\mathit{snd}\ p \sqcap \mathit{snd}\ q$)
proof
  show ($\mathit{fst}\ p \sqcap \mathit{fst}\ q$, $\mathit{snd}\ p \sqcap \mathit{snd}\ q$) $\sqsubseteq p$
  proof −
    have $\mathit{fst}\ p \sqcap \mathit{fst}\ q \sqsubseteq \mathit{fst}\ p$ **..**
    moreover have $\mathit{snd}\ p \sqcap \mathit{snd}\ q \sqsubseteq \mathit{snd}\ p$ **..**
    ultimately show *?thesis* **by** (*simp add*: *leq-prod-def*)
  qed
  show ($\mathit{fst}\ p \sqcap \mathit{fst}\ q$, $\mathit{snd}\ p \sqcap \mathit{snd}\ q$) $\sqsubseteq q$
  proof −
    have $\mathit{fst}\ p \sqcap \mathit{fst}\ q \sqsubseteq \mathit{fst}\ q$ **..**
    moreover have $\mathit{snd}\ p \sqcap \mathit{snd}\ q \sqsubseteq \mathit{snd}\ q$ **..**
    ultimately show *?thesis* **by** (*simp add*: *leq-prod-def*)
  qed
  fix r **assume** *rp*: $r \sqsubseteq p$ **and** *rq*: $r \sqsubseteq q$
  show $r \sqsubseteq$ ($\mathit{fst}\ p \sqcap \mathit{fst}\ q$, $\mathit{snd}\ p \sqcap \mathit{snd}\ q$)
  proof −
    have $\mathit{fst}\ r \sqsubseteq \mathit{fst}\ p \sqcap \mathit{fst}\ q$
    proof
      from *rp* **show** $\mathit{fst}\ r \sqsubseteq \mathit{fst}\ p$ **by** (*simp add*: *leq-prod-def*)
      from *rq* **show** $\mathit{fst}\ r \sqsubseteq \mathit{fst}\ q$ **by** (*simp add*: *leq-prod-def*)
    qed
    moreover have $\mathit{snd}\ r \sqsubseteq \mathit{snd}\ p \sqcap \mathit{snd}\ q$
    proof
      from *rp* **show** $\mathit{snd}\ r \sqsubseteq \mathit{snd}\ p$ **by** (*simp add*: *leq-prod-def*)
      from *rq* **show** $\mathit{snd}\ r \sqsubseteq \mathit{snd}\ q$ **by** (*simp add*: *leq-prod-def*)
    qed
    ultimately show *?thesis* **by** (*simp add*: *leq-prod-def*)
  qed
qed

lemma *is-sup-prod*: *is-sup* $p\ q$ ($\mathit{fst}\ p \sqcup \mathit{fst}\ q$, $\mathit{snd}\ p \sqcup \mathit{snd}\ q$)
proof
  show $p \sqsubseteq$ ($\mathit{fst}\ p \sqcup \mathit{fst}\ q$, $\mathit{snd}\ p \sqcup \mathit{snd}\ q$)
  proof −
    have $\mathit{fst}\ p \sqsubseteq \mathit{fst}\ p \sqcup \mathit{fst}\ q$ **..**
    moreover have $\mathit{snd}\ p \sqsubseteq \mathit{snd}\ p \sqcup \mathit{snd}\ q$ **..**
    ultimately show *?thesis* **by** (*simp add*: *leq-prod-def*)
  qed
  show $q \sqsubseteq$ ($\mathit{fst}\ p \sqcup \mathit{fst}\ q$, $\mathit{snd}\ p \sqcup \mathit{snd}\ q$)

proof −
  have *fst q* $\sqsubseteq$ *fst p* $\sqcup$ *fst q* **..**
  moreover have *snd q* $\sqsubseteq$ *snd p* $\sqcup$ *snd q* **..**
  ultimately show *?thesis* **by** (*simp add*: *leq-prod-def*)
qed
fix *r* **assume** *pr*: $p \sqsubseteq r$ **and** *qr*: $q \sqsubseteq r$
show (*fst p* $\sqcup$ *fst q*, *snd p* $\sqcup$ *snd q*) $\sqsubseteq r$
proof −
  have *fst p* $\sqcup$ *fst q* $\sqsubseteq$ *fst r*
  proof
    from *pr* **show** *fst p* $\sqsubseteq$ *fst r* **by** (*simp add*: *leq-prod-def*)
    from *qr* **show** *fst q* $\sqsubseteq$ *fst r* **by** (*simp add*: *leq-prod-def*)
  qed
  moreover have *snd p* $\sqcup$ *snd q* $\sqsubseteq$ *snd r*
  proof
    from *pr* **show** *snd p* $\sqsubseteq$ *snd r* **by** (*simp add*: *leq-prod-def*)
    from *qr* **show** *snd q* $\sqsubseteq$ *snd r* **by** (*simp add*: *leq-prod-def*)
  qed
  ultimately show *?thesis* **by** (*simp add*: *leq-prod-def*)
qed
qed

instance $*$ :: (*lattice*, *lattice*) *lattice*
proof
  fix *p q* :: *'a*::*lattice* × *'b*::*lattice*
  from *is-inf-prod* **show** $\exists$ *inf*. *is-inf p q inf* **..**
  from *is-sup-prod* **show** $\exists$ *sup*. *is-sup p q sup* **..**
qed

The lattice operations on a binary product structure indeed coincide with the products of the original ones.

theorem *meet-prod*: $p \sqcap q$ = (*fst p* $\sqcap$ *fst q*, *snd p* $\sqcap$ *snd q*)
  by (*rule meet-equality*) (*rule is-inf-prod*)

theorem *join-prod*: $p \sqcup q$ = (*fst p* $\sqcup$ *fst q*, *snd p* $\sqcup$ *snd q*)
  by (*rule join-equality*) (*rule is-sup-prod*)

3.5.3 General products

The class of lattices is closed under general products (function spaces) as well (cf. §1.3.3).

lemma *is-inf-fun*: *is-inf f g* ($\lambda x.\ f\ x \sqcap g\ x$)
proof
  show ($\lambda x.\ f\ x \sqcap g\ x$) $\sqsubseteq f$
  proof
    fix *x* **show** $f\ x \sqcap g\ x \sqsubseteq f\ x$ **..**
  qed
  show ($\lambda x.\ f\ x \sqcap g\ x$) $\sqsubseteq g$

proof
fix x **show** $f\ x \sqcap g\ x \sqsubseteq g\ x$ **..**
qed
fix h **assume** hf: $h \sqsubseteq f$ **and** hg: $h \sqsubseteq g$
show $h \sqsubseteq (\lambda x.\ f\ x \sqcap g\ x)$
proof
fix x
show $h\ x \sqsubseteq f\ x \sqcap g\ x$
proof
from hf **show** $h\ x \sqsubseteq f\ x$ **..**
from hg **show** $h\ x \sqsubseteq g\ x$ **..**
qed
qed
qed

lemma *is-sup-fun*: *is-sup* $f\ g\ (\lambda x.\ f\ x \sqcup g\ x)$
proof
show $f \sqsubseteq (\lambda x.\ f\ x \sqcup g\ x)$
proof
fix x **show** $f\ x \sqsubseteq f\ x \sqcup g\ x$ **..**
qed
show $g \sqsubseteq (\lambda x.\ f\ x \sqcup g\ x)$
proof
fix x **show** $g\ x \sqsubseteq f\ x \sqcup g\ x$ **..**
qed
fix h **assume** fh: $f \sqsubseteq h$ **and** gh: $g \sqsubseteq h$
show $(\lambda x.\ f\ x \sqcup g\ x) \sqsubseteq h$
proof
fix x
show $f\ x \sqcup g\ x \sqsubseteq h\ x$
proof
from fh **show** $f\ x \sqsubseteq h\ x$ **..**
from gh **show** $g\ x \sqsubseteq h\ x$ **..**
qed
qed
qed

instance *fun* :: (*type*, *lattice*) *lattice*
proof
fix $f\ g$ :: $'a \Rightarrow\ 'b$::*lattice*
show $\exists\, inf.$ *is-inf* $f\ g$ *inf* **by** *rule* (*rule is-inf-fun*)
show $\exists\, sup.$ *is-sup* $f\ g$ *sup* **by** *rule* (*rule is-sup-fun*)
qed

The lattice operations on a general product structure (function space) indeed emerge by point-wise lifting of the original ones.

theorem *meet-fun*: $f \sqcap g = (\lambda x.\ f\ x \sqcap g\ x)$
by (*rule meet-equality*) (*rule is-inf-fun*)

theorem *join-fun*: $f \sqcup g = (\lambda x.\ f\ x \sqcup g\ x)$
  by (*rule join-equality*) (*rule is-sup-fun*)

3.6 Monotonicity and semi-morphisms

The lattice operations are monotone in both argument positions. In fact, monotonicity of the second position is trivial due to commutativity.

theorem *meet-mono*: $x \sqsubseteq z \Longrightarrow y \sqsubseteq w \Longrightarrow x \sqcap y \sqsubseteq z \sqcap w$
proof –
  {
    fix $a\ b\ c :: 'a::lattice$
    assume $a \sqsubseteq c$ **have** $a \sqcap b \sqsubseteq c \sqcap b$
    proof
      have $a \sqcap b \sqsubseteq a$ **..**
      also have $\ldots \sqsubseteq c$ **.**
      finally show $a \sqcap b \sqsubseteq c$ **.**
      show $a \sqcap b \sqsubseteq b$ **..**
    qed
  } **note** *this* [*elim?*]
  assume $x \sqsubseteq z$ **hence** $x \sqcap y \sqsubseteq z \sqcap y$ **..**
  also have $\ldots = y \sqcap z$ **by** (*rule meet-commute*)
  also assume $y \sqsubseteq w$ **hence** $y \sqcap z \sqsubseteq w \sqcap z$ **..**
  also have $\ldots = z \sqcap w$ **by** (*rule meet-commute*)
  finally show *?thesis* **.**
qed

theorem *join-mono*: $x \sqsubseteq z \Longrightarrow y \sqsubseteq w \Longrightarrow x \sqcup y \sqsubseteq z \sqcup w$
proof –
  assume $x \sqsubseteq z$ **hence** $dual\ z \sqsubseteq dual\ x$ **..**
  moreover assume $y \sqsubseteq w$ **hence** $dual\ w \sqsubseteq dual\ y$ **..**
  ultimately have $dual\ z \sqcap dual\ w \sqsubseteq dual\ x \sqcap dual\ y$
    by (*rule meet-mono*)
  hence $dual\ (z \sqcup w) \sqsubseteq dual\ (x \sqcup y)$
    by (*simp only*: *dual-join*)
  thus *?thesis* **..**
qed

A semi-morphisms is a function f that preserves the lattice operations in the following manner: $f\ (Lattice.meet\ x\ y) \sqsubseteq Lattice.meet\ (f\ x)\ (f\ y)$ and $Lattice.join\ (f\ x)\ (f\ y) \sqsubseteq f\ (Lattice.join\ x\ y)$, respectively. Any of these properties is equivalent with monotonicity.

theorem *meet-semimorph*:
  $(\bigwedge x\ y.\ f\ (x \sqcap y) \sqsubseteq f\ x \sqcap f\ y) \equiv (\bigwedge x\ y.\ x \sqsubseteq y \Longrightarrow f\ x \sqsubseteq f\ y)$
proof
  assume *morph*: $\bigwedge x\ y.\ f\ (x \sqcap y) \sqsubseteq f\ x \sqcap f\ y$
  fix $x\ y :: 'a::lattice$
  assume $x \sqsubseteq y$ **hence** $x \sqcap y = x$ **..**

hence $x = x \sqcap y$ **..**
also have $f \ \ldots \sqsubseteq f\ x \sqcap f\ y$ **by** (*rule morph*)
also have $\ldots \sqsubseteq f\ y$ **..**
finally show $f\ x \sqsubseteq f\ y$ **.**
next
assume *mono*: $\bigwedge x\ y.\ x \sqsubseteq y \Longrightarrow f\ x \sqsubseteq f\ y$
show $\bigwedge x\ y.\ f\ (x \sqcap y) \sqsubseteq f\ x \sqcap f\ y$
proof $-$
fix $x\ y$
show $f\ (x \sqcap y) \sqsubseteq f\ x \sqcap f\ y$
proof
have $x \sqcap y \sqsubseteq x$ **.. thus** $f\ (x \sqcap y) \sqsubseteq f\ x$ **by** (*rule mono*)
have $x \sqcap y \sqsubseteq y$ **.. thus** $f\ (x \sqcap y) \sqsubseteq f\ y$ **by** (*rule mono*)
qed
qed
qed

end

4 Complete lattices

theory *CompleteLattice* **imports** *Lattice* **begin**

4.1 Complete lattice operations

A *complete lattice* is a partial order with general (infinitary) infimum of any set of elements. General supremum exists as well, as a consequence of the connection of infinitary bounds (see §2.6).

axclass *complete-lattice* $\subseteq$ *partial-order*
ex-Inf: $\exists\, inf.\ is\text{-}Inf\ A\ inf$

theorem *ex-Sup*: $\exists\, sup::'a::complete\text{-}lattice.\ is\text{-}Sup\ A\ sup$
proof $-$
from *ex-Inf* **obtain** *sup* **where** *is-Inf* $\{b.\ \forall\, a{\in}A.\ a \sqsubseteq b\}$ *sup* **by** *blast*
hence *is-Sup A sup* **by** (*rule Inf-Sup*)
thus *?thesis* **..**
qed

The general $\bigsqcap$ (meet) and $\bigsqcup$ (join) operations select such infimum and supremum elements.

consts
Meet :: $'a::complete\text{-}lattice\ set \Rightarrow 'a$
Join :: $'a::complete\text{-}lattice\ set \Rightarrow 'a$
syntax (*xsymbols*)
Meet :: $'a::complete\text{-}lattice\ set \Rightarrow 'a$ $(\bigsqcap\text{-}\ [90]\ 90)$
Join :: $'a::complete\text{-}lattice\ set \Rightarrow 'a$ $(\bigsqcup\text{-}\ [90]\ 90)$

defs
  Meet-def: $\bigsqcap A \equiv$ *THE inf. is-Inf A inf*
  Join-def: $\bigsqcup A \equiv$ *THE sup. is-Sup A sup*

Due to unique existence of bounds, the complete lattice operations may be exhibited as follows.

lemma *Meet-equality* [*elim?*]: *is-Inf A inf* $\Longrightarrow \bigsqcap A = inf$
proof (*unfold Meet-def*)
  assume *is-Inf A inf*
  thus (*THE inf. is-Inf A inf*) = *inf*
    by (*rule the-equality*) (*rule is-Inf-uniq*)
qed

lemma *MeetI* [*intro?*]:
  $(\bigwedge a.\ a \in A \Longrightarrow inf \sqsubseteq a) \Longrightarrow$
    $(\bigwedge b.\ \forall a \in A.\ b \sqsubseteq a \Longrightarrow b \sqsubseteq inf) \Longrightarrow$
    $\bigsqcap A = inf$
  by (*rule Meet-equality*, *rule is-InfI*) *blast+*

lemma *Join-equality* [*elim?*]: *is-Sup A sup* $\Longrightarrow \bigsqcup A = sup$
proof (*unfold Join-def*)
  assume *is-Sup A sup*
  thus (*THE sup. is-Sup A sup*) = *sup*
    by (*rule the-equality*) (*rule is-Sup-uniq*)
qed

lemma *JoinI* [*intro?*]:
  $(\bigwedge a.\ a \in A \Longrightarrow a \sqsubseteq sup) \Longrightarrow$
    $(\bigwedge b.\ \forall a \in A.\ a \sqsubseteq b \Longrightarrow sup \sqsubseteq b) \Longrightarrow$
    $\bigsqcup A = sup$
  by (*rule Join-equality*, *rule is-SupI*) *blast+*

The $\bigsqcap$ and $\bigsqcup$ operations indeed determine bounds on a complete lattice structure.

lemma *is-Inf-Meet* [*intro?*]: *is-Inf A* ($\bigsqcap A$)
proof (*unfold Meet-def*)
  from *ex-Inf* **obtain** *inf* **where** *is-Inf A inf* **..**
  thus *is-Inf A* (*THE inf. is-Inf A inf*) **by** (*rule theI*) (*rule is-Inf-uniq*)
qed

lemma *Meet-greatest* [*intro?*]: $(\bigwedge a.\ a \in A \Longrightarrow x \sqsubseteq a) \Longrightarrow x \sqsubseteq \bigsqcap A$
  by (*rule is-Inf-greatest*, *rule is-Inf-Meet*) *blast*

lemma *Meet-lower* [*intro?*]: $a \in A \Longrightarrow \bigsqcap A \sqsubseteq a$
  by (*rule is-Inf-lower*) (*rule is-Inf-Meet*)

lemma *is-Sup-Join* [*intro?*]: *is-Sup A* ($\bigsqcup A$)

proof (*unfold Join-def*)
  from *ex-Sup* **obtain** *sup* **where** *is-Sup A sup* **..**
  thus *is-Sup A* (*THE sup. is-Sup A sup*) **by** (*rule theI*) (*rule is-Sup-uniq*)
qed

lemma *Join-least* [*intro?*]: $(\bigwedge a.\ a \in A \Longrightarrow a \sqsubseteq x) \Longrightarrow \bigsqcup A \sqsubseteq x$
  by (*rule is-Sup-least*, *rule is-Sup-Join*) *blast*
lemma *Join-lower* [*intro?*]: $a \in A \Longrightarrow a \sqsubseteq \bigsqcup A$
  by (*rule is-Sup-upper*) (*rule is-Sup-Join*)

4.2 The Knaster-Tarski Theorem

The Knaster-Tarski Theorem (in its simplest formulation) states that any monotone function on a complete lattice has a least fixed-point (see [2, pages 93–94] for example). This is a consequence of the basic boundary properties of the complete lattice operations.

theorem *Knaster-Tarski*:
  $(\bigwedge x\ y.\ x \sqsubseteq y \Longrightarrow f\ x \sqsubseteq f\ y) \Longrightarrow \exists\ a::'a::complete\text{-}lattice.\ f\ a = a$
proof
  assume *mono*: $\bigwedge x\ y.\ x \sqsubseteq y \Longrightarrow f\ x \sqsubseteq f\ y$
  let $?H = \{u.\ f\ u \sqsubseteq u\}$ **let** $?a = \bigsqcap ?H$
  have *ge*: $f\ ?a \sqsubseteq ?a$
  proof
    fix *x* **assume** *x*: $x \in ?H$
    hence $?a \sqsubseteq x$ **..**
    hence $f\ ?a \sqsubseteq f\ x$ **by** (*rule mono*)
    also from *x* **have** $\ldots \sqsubseteq x$ **..**
    finally show $f\ ?a \sqsubseteq x$ **.**
  qed
  also have $?a \sqsubseteq f\ ?a$
  proof
    from *ge* **have** $f\ (f\ ?a) \sqsubseteq f\ ?a$ **by** (*rule mono*)
    thus $f\ ?a \in ?H$ **..**
  qed
  finally show $f\ ?a = ?a$ **.**
qed

4.3 Bottom and top elements

With general bounds available, complete lattices also have least and greatest elements.

constdefs
  bottom :: *'a::complete-lattice* $(\bot)$
  $\bot \equiv \bigsqcap UNIV$
  top :: *'a::complete-lattice* $(\top)$
  $\top \equiv \bigsqcup UNIV$

lemma *bottom-least* [*intro?*]: $\bot \sqsubseteq x$
proof (*unfold bottom-def*)
  have $x \in UNIV$ **..**
  thus $\bigsqcap UNIV \sqsubseteq x$ **..**
qed

lemma *bottomI* [*intro?*]: $(\bigwedge a.\ x \sqsubseteq a) \Longrightarrow \bot = x$
proof (*unfold bottom-def*)
  assume $\bigwedge a.\ x \sqsubseteq a$
  show $\bigsqcap UNIV = x$
  proof
    fix a **show** $x \sqsubseteq a$ **.**
  next
    fix b :: *'a::complete-lattice*
    assume b: $\forall a \in UNIV.\ b \sqsubseteq a$
    have $x \in UNIV$ **..**
    with b **show** $b \sqsubseteq x$ **..**
  qed
qed

lemma *top-greatest* [*intro?*]: $x \sqsubseteq \top$
proof (*unfold top-def*)
  have $x \in UNIV$ **..**
  thus $x \sqsubseteq \bigsqcup UNIV$ **..**
qed

lemma *topI* [*intro?*]: $(\bigwedge a.\ a \sqsubseteq x) \Longrightarrow \top = x$
proof (*unfold top-def*)
  assume $\bigwedge a.\ a \sqsubseteq x$
  show $\bigsqcup UNIV = x$
  proof
    fix a **show** $a \sqsubseteq x$ **.**
  next
    fix b :: *'a::complete-lattice*
    assume b: $\forall a \in UNIV.\ a \sqsubseteq b$
    have $x \in UNIV$ **..**
    with b **show** $x \sqsubseteq b$ **..**
  qed
qed

4.4 Duality

The class of complete lattices is closed under formation of dual structures.

instance *dual* :: (*complete-lattice*) *complete-lattice*
proof
  fix A' :: *'a::complete-lattice dual set*
  show $\exists\, inf'.\ is\text{-}Inf\ A'\ inf'$
  proof −
    have $\exists\, sup.\ is\text{-}Sup\ (undual\ `\ A')\ sup$ **by** (*rule ex-Sup*)

hence $\exists$ *sup. is-Inf* (*dual* ' *undual* ' *A'*) (*dual sup*) **by** (*simp only*: *dual-Inf*)
thus *?thesis* **by** (*simp add*: *dual-ex* [*symmetric*] *image-compose* [*symmetric*])
qed
qed

Apparently, the $\bigsqcap$ and $\bigsqcup$ operations are dual to each other.

theorem *dual-Meet* [*intro?*]: *dual* ($\bigsqcap$ *A*) = $\bigsqcup$(*dual* ' *A*)
proof −
from *is-Inf-Meet* **have** *is-Sup* (*dual* ' *A*) (*dual* ($\bigsqcap$ *A*)) **..**
hence $\bigsqcup$(*dual* ' *A*) = *dual* ($\bigsqcap$ *A*) **..**
thus *?thesis* **..**
qed

theorem *dual-Join* [*intro?*]: *dual* ($\bigsqcup$ *A*) = $\bigsqcap$(*dual* ' *A*)
proof −
from *is-Sup-Join* **have** *is-Inf* (*dual* ' *A*) (*dual* ($\bigsqcup$ *A*)) **..**
hence $\bigsqcap$(*dual* ' *A*) = *dual* ($\bigsqcup$ *A*) **..**
thus *?thesis* **..**
qed

Likewise are $\bot$ and $\top$ duals of each other.

theorem *dual-bottom* [*intro?*]: *dual* $\bot = \top$
proof −
have $\top$ = *dual* $\bot$
proof
fix a' **have** $\bot \sqsubseteq$ *undual* a' **..**
hence *dual* (*undual* a') $\sqsubseteq$ *dual* $\bot$ **..**
thus $a' \sqsubseteq$ *dual* $\bot$ **by** *simp*
qed
thus *?thesis* **..**
qed

theorem *dual-top* [*intro?*]: *dual* $\top = \bot$
proof −
have $\bot$ = *dual* $\top$
proof
fix a' **have** *undual* $a' \sqsubseteq \top$ **..**
hence *dual* $\top \sqsubseteq$ *dual* (*undual* a') **..**
thus *dual* $\top \sqsubseteq a'$ **by** *simp*
qed
thus *?thesis* **..**
qed

4.5 Complete lattices are lattices

Complete lattices (with general bounds available) are indeed plain lattices as well. This holds due to the connection of general versus binary bounds that has been formally established in §2.5.

lemma *is-inf-binary*: *is-inf* x y ($\bigsqcap\{x, y\}$)
proof −
  have *is-Inf* $\{x, y\}$ ($\bigsqcap\{x, y\}$) **..**
  thus *?thesis* **by** (*simp only*: *is-Inf-binary*)
qed

lemma *is-sup-binary*: *is-sup* x y ($\bigsqcup\{x, y\}$)
proof −
  have *is-Sup* $\{x, y\}$ ($\bigsqcup\{x, y\}$) **..**
  thus *?thesis* **by** (*simp only*: *is-Sup-binary*)
qed

instance *complete-lattice* $\subseteq$ *lattice*
proof
  fix x y :: *'a::complete-lattice*
  from *is-inf-binary* **show** $\exists$ *inf*. *is-inf* x y *inf* **..**
  from *is-sup-binary* **show** $\exists$ *sup*. *is-sup* x y *sup* **..**
qed

theorem *meet-binary*: $x \sqcap y = \bigsqcap\{x, y\}$
  by (*rule meet-equality*) (*rule is-inf-binary*)

theorem *join-binary*: $x \sqcup y = \bigsqcup\{x, y\}$
  by (*rule join-equality*) (*rule is-sup-binary*)

4.6 Complete lattices and set-theory operations

The complete lattice operations are (anti) monotone wrt. set inclusion.

theorem *Meet-subset-antimono*: $A \subseteq B \Longrightarrow \bigsqcap B \sqsubseteq \bigsqcap A$
proof (*rule Meet-greatest*)
  fix a **assume** $a \in A$
  also assume $A \subseteq B$
  finally have $a \in B$ **.**
  thus $\bigsqcap B \sqsubseteq a$ **..**
qed

theorem *Join-subset-mono*: $A \subseteq B \Longrightarrow \bigsqcup A \sqsubseteq \bigsqcup B$
proof −
  assume $A \subseteq B$
  hence *dual* ‘ $A \subseteq$ *dual* ‘ B **by** *blast*
  hence $\bigsqcap$(*dual* ‘ B) $\sqsubseteq \bigsqcap$(*dual* ‘ A) **by** (*rule Meet-subset-antimono*)
  hence *dual* ($\bigsqcup B$) $\sqsubseteq$ *dual* ($\bigsqcup A$) **by** (*simp only*: *dual-Join*)
  thus *?thesis* **by** (*simp only*: *dual-leq*)
qed

Bounds over unions of sets may be obtained separately.

theorem *Meet-Un*: $\bigsqcap(A \cup B) = \bigsqcap A \sqcap \bigsqcap B$
proof

fix a **assume** $a \in A \cup B$
thus $\bigsqcap A \sqcap \bigsqcap B \sqsubseteq a$
proof
assume a: $a \in A$
have $\bigsqcap A \sqcap \bigsqcap B \sqsubseteq \bigsqcap A$ **..**
also from a **have** $\ldots \sqsubseteq a$ **..**
finally show *?thesis* **.**
next
assume a: $a \in B$
have $\bigsqcap A \sqcap \bigsqcap B \sqsubseteq \bigsqcap B$ **..**
also from a **have** $\ldots \sqsubseteq a$ **..**
finally show *?thesis* **.**
qed
next
fix b **assume** b: $\forall\, a \in A \cup B.\ b \sqsubseteq a$
show $b \sqsubseteq \bigsqcap A \sqcap \bigsqcap B$
proof
show $b \sqsubseteq \bigsqcap A$
proof
fix a **assume** $a \in A$
hence $a \in A \cup B$ **..**
with b **show** $b \sqsubseteq a$ **..**
qed
show $b \sqsubseteq \bigsqcap B$
proof
fix a **assume** $a \in B$
hence $a \in A \cup B$ **..**
with b **show** $b \sqsubseteq a$ **..**
qed
qed
qed

theorem *Join-Un*: $\bigsqcup (A \cup B) = \bigsqcup A \sqcup \bigsqcup B$
proof $-$
have $dual\ (\bigsqcup (A \cup B)) = \bigsqcap (dual\ \text{`}\ A \cup dual\ \text{`}\ B)$
by (*simp only*: *dual-Join image-Un*)
also have $\ldots = \bigsqcap (dual\ \text{`}\ A) \sqcap \bigsqcap (dual\ \text{`}\ B)$
by (*rule Meet-Un*)
also have $\ldots = dual\ (\bigsqcup A \sqcup \bigsqcup B)$
by (*simp only*: *dual-join dual-Join*)
finally show *?thesis* **..**
qed

Bounds over singleton sets are trivial.

theorem *Meet-singleton*: $\bigsqcap \{x\} = x$
proof
fix a **assume** $a \in \{x\}$
hence $a = x$ **by** *simp*
thus $x \sqsubseteq a$ **by** (*simp only*: *leq-refl*)

next
  fix b **assume** $\forall\, a \in \{x\}.\ b \sqsubseteq a$
  thus $b \sqsubseteq x$ **by** *simp*
qed

theorem *Join-singleton*: $\bigsqcup\{x\} = x$
proof –
  have $\mathit{dual}\ (\bigsqcup\{x\}) = \bigsqcap\{\mathit{dual}\ x\}$ **by** (*simp add*: *dual-Join*)
  also have $\ldots = \mathit{dual}\ x$ **by** (*rule Meet-singleton*)
  finally show *?thesis* **..**
qed

Bounds over the empty and universal set correspond to each other.

theorem *Meet-empty*: $\bigsqcap\{\} = \bigsqcup\ \mathit{UNIV}$
proof
  fix $a :: {'a}::\mathit{complete\text{-}lattice}$
  assume $a \in \{\}$
  hence *False* **by** *simp*
  thus $\bigsqcup\ \mathit{UNIV} \sqsubseteq a$ **..**
next
  fix $b :: {'a}::\mathit{complete\text{-}lattice}$
  have $b \in \mathit{UNIV}$ **..**
  thus $b \sqsubseteq \bigsqcup\ \mathit{UNIV}$ **..**
qed

theorem *Join-empty*: $\bigsqcup\{\} = \bigsqcap\ \mathit{UNIV}$
proof –
  have $\mathit{dual}\ (\bigsqcup\{\}) = \bigsqcap\{\}$ **by** (*simp add*: *dual-Join*)
  also have $\ldots = \bigsqcup\ \mathit{UNIV}$ **by** (*rule Meet-empty*)
  also have $\ldots = \mathit{dual}\ (\bigsqcap\ \mathit{UNIV})$ **by** (*simp add*: *dual-Meet*)
  finally show *?thesis* **..**
qed

end

References

[1] G. Bauer and M. Wenzel. Computer-assisted mathematics at work — the Hahn-Banach theorem in Isabelle/Isar. In T. Coquand, P. Dybjer, B. Nordström, and J. Smith, editors, *Types for Proofs and Programs: TYPES'99*, LNCS, 2000.

[2] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 1990.

[3] M. Wenzel. Isar — a generic interpretative approach to readable formal proof documents. In Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and

L. Thery, editors, *Theorem Proving in Higher Order Logics: TPHOLs '99*, volume 1690 of *LNCS*, 1999.

[4] M. Wenzel. *The Isabelle/Isar Reference Manual*, 2000. http://isabelle.in.tum.de/doc/isar-ref.pdf.

[5] M. Wenzel. *Using Axiomatic Type Classes in Isabelle*, 2000. http://isabelle.in.tum.de/doc/axclass.pdf.