

Lattices and Orders in Isabelle/HOL

Markus Wenzel
TU München

June 8, 2008

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	3
1.1	Ordered structures	3
1.2	Duality	3
1.3	Transforming orders	5
1.3.1	Duals	5
1.3.2	Binary products	6
1.3.3	General products	7
2	Bounds	8
2.1	Infimum and supremum	8
2.2	Duality	9
2.3	Uniqueness	10
2.4	Related elements	11
2.5	General versus binary bounds	12
2.6	Connecting general bounds	13

3	Lattices	14
3.1	Lattice operations	14
3.2	Duality	16
3.3	Algebraic properties	17
3.4	Order versus algebraic structure	19
3.5	Example instances	20
3.5.1	Linear orders	20
3.5.2	Binary products	21
3.5.3	General products	23
3.6	Monotonicity and semi-morphisms	24
4	Complete lattices	26
4.1	Complete lattice operations	26
4.2	The Knaster-Tarski Theorem	28
4.3	Bottom and top elements	29
4.4	Duality	30
4.5	Complete lattices are lattices	31
4.6	Complete lattices and set-theory operations	32

1 Orders

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

1.1 Ordered structures

We define several classes of ordered structures over some type $'a$ with relation $\sqsubseteq :: 'a \Rightarrow 'a \Rightarrow \text{bool}$. 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 :: $'a::\text{leq} \Rightarrow 'a \Rightarrow \text{bool}$ (**infixl** [= 50])

notation (*xsymbols*)

leq (**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*:

$((x::'a::\text{linear-order}) \sqsubseteq y \Longrightarrow C) \Longrightarrow (y \sqsubseteq x \Longrightarrow C) \Longrightarrow C$

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 :: $'a \text{ dual} \Rightarrow 'a$

primrec

undual-dual: *undual* (*dual* x) = x

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

defs (**overloaded**)

leq-dual-def: $x' \sqsubseteq y' \equiv \text{undual } y' \sqsubseteq \text{undual } x'$

lemma *undual-leq* [*iff?*]: $(\text{undual } x' \sqsubseteq \text{undual } y') = (y' \sqsubseteq x')$

by (*simp add: leq-dual-def*)

lemma *dual-leq* [iff?]: $(\text{dual } x \sqsubseteq \text{dual } y) = (y \sqsubseteq x)$
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]: $\text{dual } (\text{undual } x') = x'$
by (*cases x' simp*)

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

lemma *dual-undual-id* [simp]: $\text{dual } o \text{ undual} = \text{id}$
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?]: $(\text{undual } x' = \text{undual } y') = (x' = y')$
by (*cases x', cases y' simp*)

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

lemma *dual-ball* [iff?]: $(\forall x \in A. P (\text{dual } x)) = (\forall x' \in \text{dual } 'A. P x')$
proof

assume *a*: $\forall x \in A. P (\text{dual } x)$

show $\forall x' \in \text{dual } 'A. P x'$

proof

fix *x'* **assume** *x'*: $x' \in \text{dual } 'A$

have $\text{undual } x' \in A$

proof –

from *x'* **have** $\text{undual } x' \in \text{undual } ' \text{dual } 'A$ **by** *simp*

thus $\text{undual } x' \in A$ **by** (*simp add: image-compose [symmetric]*)

qed

with *a* **have** $P (\text{dual } (\text{undual } x'))$ **..**

also have $\dots = x'$ **by** *simp*

finally show $P x'$.

qed

next

assume *a*: $\forall x' \in \text{dual } 'A. P x'$

show $\forall x \in A. P (\text{dual } x)$

proof

fix *x* **assume** *x* $\in A$

hence $\text{dual } x \in \text{dual } 'A$ **by** *simp*

with *a* **show** $P (\text{dual } x)$ **..**

qed

qed

lemma *range-dual* [*simp*]: $\text{dual} \text{ ` } UNIV = UNIV$

proof (*rule surj-range*)

have $\bigwedge x'. \text{dual} (\text{undual } x') = x'$ **by** *simp*

thus *surj dual* **by** (*rule surjI*)

qed

lemma *dual-all* [*iff?*]: $(\forall x. P (\text{dual } x)) = (\forall x'. P x')$

proof –

have $(\forall x \in UNIV. P (\text{dual } x)) = (\forall x' \in \text{dual ` } UNIV. P x')$

by (*rule dual-ball*)

thus *?thesis* **by** *simp*

qed

lemma *dual-ex*: $(\exists x. P (\text{dual } x)) = (\exists x'. P x')$

proof –

have $(\forall x. \neg P (\text{dual } x)) = (\forall x'. \neg P x')$

by (*rule dual-all*)

thus *?thesis* **by** *blast*

qed

lemma *dual-Collect*: $\{\text{dual } x \mid x. P (\text{dual } x)\} = \{x'. P x'\}$

proof –

have $\{\text{dual } x \mid x. P (\text{dual } x)\} = \{x'. \exists x''. x' = x'' \wedge P x''\}$

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 $x' y' z' :: 'a::\text{quasi-order dual}$

have $\text{undual } x' \sqsubseteq \text{undual } x' ..$ **thus** $x' \sqsubseteq x' ..$

assume $y' \sqsubseteq z'$ **hence** $\text{undual } z' \sqsubseteq \text{undual } y' ..$

also assume $x' \sqsubseteq y'$ **hence** $\text{undual } y' \sqsubseteq \text{undual } x' ..$

finally **show** $x' \sqsubseteq z' ..$

qed

instance *dual* :: (*partial-order*) *partial-order*

proof

fix $x' y' :: 'a::\text{partial-order dual}$

assume $y' \sqsubseteq x'$ **hence** $\text{undual } x' \sqsubseteq \text{undual } y' ..$

also assume $x' \sqsubseteq y'$ **hence** $\text{undual } y' \sqsubseteq \text{undual } x' ..$

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  $\implies$  snd p  $\sqsubseteq$  snd q  $\implies$  p  $\sqsubseteq$  q

```

```

  by (unfold leq-prod-def) blast

```

lemma leq-prodE [elim?]:

```

  p  $\sqsubseteq$  q  $\implies$  (fst p  $\sqsubseteq$  fst q  $\implies$  snd p  $\sqsubseteq$  snd q  $\implies$  C)  $\implies$  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 ...  $\sqsubseteq$  fst r ..

```

```

  finally show fst p  $\sqsubseteq$  fst r .

```

```

  from pq have snd p  $\sqsubseteq$  snd q ..

```

```

  also from qr have ...  $\sqsubseteq$  snd r ..

```

```

  finally show snd p  $\sqsubseteq$  snd r .

```

qed

qed

instance * :: (*partial-order*, *partial-order*) *partial-order*

proof

fix $p\ q :: 'a::\text{partial-order} \times 'b::\text{partial-order}$

assume $pq: p \sqsubseteq q$ and $qp: q \sqsubseteq p$

show $p = q$

proof

from pq have $\text{fst } p \sqsubseteq \text{fst } q$..

also from qp have $\dots \sqsubseteq \text{fst } p$..

finally show $\text{fst } p = \text{fst } q$.

from pq have $\text{snd } p \sqsubseteq \text{snd } q$..

also from qp have $\dots \sqsubseteq \text{snd } p$..

finally show $\text{snd } p = \text{snd } q$.

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?*]: $(\bigwedge x. f\ x \sqsubseteq g\ x) \implies f \sqsubseteq g$

by (*unfold leq-fun-def*) *blast*

lemma *leq-funD* [*dest?*]: $f \sqsubseteq g \implies f\ x \sqsubseteq g\ x$

by (*unfold leq-fun-def*) *blast*

instance *fun* :: (*type*, *quasi-order*) *quasi-order*

proof

fix $f\ g\ h :: 'a \Rightarrow 'b::\text{quasi-order}$

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 $\dots \sqsubseteq h\ x$..

finally show $f\ x \sqsubseteq h\ x$.

qed

qed

```

instance fun :: (type, partial-order) partial-order
proof
  fix f g :: 'a ⇒ 'b::partial-order
  assume fg: f ⊆ g and gf: g ⊆ f
  show f = g
  proof
    fix x from fg have f x ⊆ g x ..
    also from gf have ... ⊆ f x ..
    finally show f x = g x .
  qed
qed

end

```

2 Bounds

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

hide const inf sup

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.

definition

```

is-inf :: 'a::partial-order ⇒ 'a ⇒ 'a ⇒ bool where
is-inf x y inf = (inf ⊆ x ∧ inf ⊆ y ∧ (∀ z. z ⊆ x ∧ z ⊆ y ⟶ z ⊆ inf))

```

definition

```

is-sup :: 'a::partial-order ⇒ 'a ⇒ 'a ⇒ bool where
is-sup x y sup = (x ⊆ sup ∧ y ⊆ sup ∧ (∀ z. x ⊆ z ∧ y ⊆ z ⟶ sup ⊆ z))

```

definition

```

is-Inf :: 'a::partial-order set ⇒ 'a ⇒ bool where
is-Inf A inf = ((∀ x ∈ A. inf ⊆ x) ∧ (∀ z. (∀ x ∈ A. z ⊆ x) ⟶ z ⊆ inf))

```

definition

```

is-Sup :: 'a::partial-order set ⇒ 'a ⇒ bool where
is-Sup A sup = ((∀ x ∈ A. x ⊆ sup) ∧ (∀ z. (∀ x ∈ A. x ⊆ z) ⟶ sup ⊆ z))

```

These definitions entail the following basic properties of boundary elements.

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

lemma is-inf-greatest [elim?]:

$is-inf\ x\ y\ inf \implies z \sqsubseteq x \implies z \sqsubseteq y \implies z \sqsubseteq inf$
by (unfold is-inf-def) blast

lemma *is-inf-lower* [elim?]:
 $is-inf\ x\ y\ inf \implies (inf \sqsubseteq x \implies inf \sqsubseteq y \implies C) \implies C$
by (unfold is-inf-def) blast

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

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

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

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

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

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

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

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

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

2.2 Duality

Infimum and supremum are dual to each other.

theorem *dual-inf* [iff?]:

is-inf (dual *x*) (dual *y*) (dual *sup*) = *is-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 anti-symmetry of the underlying relation.

theorem *is-inf-uniq*: *is-inf* *x y inf* \implies *is-inf* *x y inf'* \implies *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* \sqsubseteq *inf'*

proof (*rule is-inf-greatest*)

from *inf* **show** *inf* \sqsubseteq *x* ..

from *inf* **show** *inf* \sqsubseteq *y* ..

qed

from *inf* **show** *inf'* \sqsubseteq *inf*

proof (*rule is-inf-greatest*)

from *inf'* **show** *inf'* \sqsubseteq *x* ..

from *inf'* **show** *inf'* \sqsubseteq *y* ..

qed

qed

qed

theorem *is-sup-uniq*: *is-sup* *x y sup* \implies *is-sup* *x y sup'* \implies *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

then show *sup* = *sup'* ..

qed

theorem *is-Inf-uniq*: $is-Inf\ A\ inf \implies is-Inf\ A\ inf' \implies inf = inf'$

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** $x \in A$

with inf **show** $inf \sqsubseteq x$..

qed

from inf **show** $inf' \sqsubseteq inf$

proof (*rule is-Inf-greatest*)

fix x **assume** $x \in A$

with inf' **show** $inf' \sqsubseteq x$..

qed

qed

qed

theorem *is-Sup-uniq*: $is-Sup\ A\ sup \implies is-Sup\ A\ sup' \implies sup = sup'$

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

then **show** $sup = sup'$..

qed

2.4 Related elements

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

theorem *is-inf-related* [*elim?*]: $x \sqsubseteq y \implies is-inf\ x\ y\ x$

proof –

assume $x \sqsubseteq y$

show *?thesis*

proof

show $x \sqsubseteq x$..

show $x \sqsubseteq y$ **by** *fact*

fix z **assume** $z \sqsubseteq x$ **and** $z \sqsubseteq y$ **show** $z \sqsubseteq x$ **by** *fact*

qed

qed

theorem *is-sup-related* [*elim?*]: $x \sqsubseteq y \implies is-sup\ x\ y\ y$

proof –

assume $x \sqsubseteq y$

show *?thesis*

```

proof
  show  $x \sqsubseteq y$  by fact
  show  $y \sqsubseteq y$  ..
  fix  $z$  assume  $x \sqsubseteq z$  and  $y \sqsubseteq z$ 
  show  $y \sqsubseteq z$  by fact
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  $x \in ?A$  by simp
      with is-Inf show inf  $\sqsubseteq x$  ..
      have  $y \in ?A$  by simp
      with is-Inf show inf  $\sqsubseteq y$  ..
      fix  $z$  assume  $zx$ :  $z \sqsubseteq x$  and  $zy$ :  $z \sqsubseteq y$ 
      from is-Inf show  $z \sqsubseteq \text{inf}$ 
      proof (rule is-Inf-greatest)
        fix  $a$  assume  $a \in ?A$ 
        then have  $a = x \vee a = y$  by blast
        then show  $z \sqsubseteq a$ 
        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  $a \in ?A$ 
      then have  $a = x \vee a = y$  by blast
      then show inf  $\sqsubseteq a$ 
      proof
        assume  $a = x$ 
        also from is-inf have inf  $\sqsubseteq x$  ..
        finally show ?thesis .
      qed
    qed
  next

```

```

next
  assume  $a = y$ 
  also from is-inf have  $\inf \sqsubseteq y$  ..
  finally show ?thesis .
qed
next
fix  $z$  assume  $z: \forall a \in ?A. z \sqsubseteq a$ 
from is-inf show  $z \sqsubseteq \inf$ 
proof (rule is-inf-greatest)
  from  $z$  show  $z \sqsubseteq x$  by blast
  from  $z$  show  $z \sqsubseteq y$  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 ‘  $\{x, y\}$  =  $\{\textit{dual } x, \textit{dual } y\}$ 
  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  $\{b. \forall a \in A. a \sqsubseteq b\}$  sup  $\implies$  is-Sup  $A$  sup
proof -
  let  $?B = \{b. \forall a \in A. a \sqsubseteq b\}$ 
  assume is-Inf: is-Inf  $?B$  sup
  show is-Sup  $A$  sup
  proof
    fix  $x$  assume  $x: x \in A$ 
    from is-Inf show  $x \sqsubseteq \textit{sup}$ 
  proof (rule is-Inf-greatest)
    fix  $y$  assume  $y \in ?B$ 
    then have  $\forall a \in A. a \sqsubseteq y$  ..
    from this x show  $x \sqsubseteq y$  ..
  qed
qed

```

```

next
  fix z assume  $\forall x \in A. x \sqsubseteq z$ 
  then have  $z \in ?B$  ..
  with is-Inf show  $sup \sqsubseteq z$  ..
qed
qed

theorem Sup-Inf: is-Sup  $\{b. \forall a \in A. b \sqsubseteq a\}$  inf  $\implies$  is-Inf A inf
proof -
  assume is-Sup  $\{b. \forall a \in A. b \sqsubseteq a\}$  inf
  then have is-Inf (dual ‘  $\{b. \forall a \in A. \text{dual } a \sqsubseteq \text{dual } b\}$ ) (dual inf)
    by (simp only: dual-Inf dual-leq)
  also have dual ‘  $\{b. \forall a \in A. \text{dual } a \sqsubseteq \text{dual } b\} = \{b'. \forall a' \in \text{dual } 'A. a' \sqsubseteq b'\}$ 
    by (auto iff: dual-ball dual-Collect simp add: image-Collect)
  finally have is-Inf ... (dual inf) .
  then have is-Sup (dual ‘ A) (dual inf)
    by (rule Inf-Sup)
  then show ?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  $\sqsubseteq$  partial-order
  ex-inf:  $\exists \text{inf}. \text{is-inf } x \ y \ \text{inf}$ 
  ex-sup:  $\exists \text{sup}. \text{is-sup } x \ y \ \text{sup}$ 

```

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

definition

```

meet :: 'a::lattice  $\Rightarrow$  'a  $\Rightarrow$  'a (infixl && 70) where
  x && y = (THE inf. is-inf x y inf)

```

definition

```

join :: 'a::lattice  $\Rightarrow$  'a  $\Rightarrow$  'a (infixl || 65) where
  x || y = (THE sup. is-sup x y sup)

```

notation (*xsymbols*)

```

meet (infixl  $\sqcap$  70) and
join (infixl  $\sqcup$  65)

```

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

lemma *meet-equality* [elim?]: $is-inf\ x\ y\ inf \implies x \sqcap y = inf$

proof (*unfold meet-def*)

assume $is-inf\ x\ y\ inf$

then show ($THE\ inf.\ is-inf\ x\ y\ inf$) = inf

by (*rule the-equality*) (*rule is-inf-uniq* [$OF - \langle is-inf\ x\ y\ inf \rangle$])

qed

lemma *meetI* [intro?]:

$inf \sqsubseteq x \implies inf \sqsubseteq y \implies (\bigwedge z. z \sqsubseteq x \implies z \sqsubseteq y \implies z \sqsubseteq inf) \implies x \sqcap y = inf$

by (*rule meet-equality*, *rule is-infI*) *blast+*

lemma *join-equality* [elim?]: $is-sup\ x\ y\ sup \implies x \sqcup y = sup$

proof (*unfold join-def*)

assume $is-sup\ x\ y\ sup$

then show ($THE\ sup.\ is-sup\ x\ y\ sup$) = sup

by (*rule the-equality*) (*rule is-sup-uniq* [$OF - \langle is-sup\ x\ y\ sup \rangle$])

qed

lemma *joinI* [intro?]: $x \sqsubseteq sup \implies y \sqsubseteq sup \implies$

$(\bigwedge z. x \sqsubseteq z \implies y \sqsubseteq z \implies sup \sqsubseteq z) \implies x \sqcup y = sup$

by (*rule join-equality*, *rule is-supI*) *blast+*

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

lemma *is-inf-meet* [intro?]: $is-inf\ x\ y\ (x \sqcap y)$

proof (*unfold meet-def*)

from $ex-inf$ **obtain** inf **where** $is-inf\ x\ y\ inf$..

then show $is-inf\ x\ y\ (THE\ inf.\ is-inf\ x\ y\ inf)$

by (*rule theI*) (*rule is-inf-uniq* [$OF - \langle is-inf\ x\ y\ inf \rangle$])

qed

lemma *meet-greatest* [intro?]: $z \sqsubseteq x \implies z \sqsubseteq y \implies z \sqsubseteq x \sqcap y$

by (*rule is-inf-greatest*) (*rule is-inf-meet*)

lemma *meet-lower1* [intro?]: $x \sqcap y \sqsubseteq x$

by (*rule is-inf-lower*) (*rule is-inf-meet*)

lemma *meet-lower2* [intro?]: $x \sqcap y \sqsubseteq y$

by (*rule is-inf-lower*) (*rule is-inf-meet*)

lemma *is-sup-join* [intro?]: $is-sup\ x\ y\ (x \sqcup y)$

proof (*unfold join-def*)

from $ex-sup$ **obtain** sup **where** $is-sup\ x\ y\ sup$..

then show $is-sup\ x\ y\ (THE\ sup.\ is-sup\ x\ y\ sup)$

by (*rule theI*) (*rule is-sup-uniq* [$OF - \langle is-sup\ x\ y\ sup \rangle$])

qed

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

lemma *join-upper1* [intro?]: $x \sqsubseteq x \sqcup y$
by (rule *is-sup-upper*) (rule *is-sup-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 $x' y' :: 'a :: \text{lattice } \text{dual}$
show $\exists \text{inf}'. \text{is-inf } x' y' \text{inf}'$
proof –
have $\exists \text{sup}. \text{is-sup } (\text{undual } x') (\text{undual } y') \text{sup}$ **by** (rule *ex-sup*)
then have $\exists \text{sup}. \text{is-inf } (\text{dual } (\text{undual } x')) (\text{dual } (\text{undual } y')) (\text{dual } \text{sup})$
by (*simp only: dual-inf*)
then show *?thesis* **by** (*simp add: dual-ex [symmetric]*)
qed
show $\exists \text{sup}'. \text{is-sup } x' y' \text{sup}'$
proof –
have $\exists \text{inf}. \text{is-inf } (\text{undual } x') (\text{undual } y') \text{inf}$ **by** (rule *ex-inf*)
then have $\exists \text{inf}. \text{is-sup } (\text{dual } (\text{undual } x')) (\text{dual } (\text{undual } y')) (\text{dual } \text{inf})$
by (*simp only: dual-sup*)
then show *?thesis* **by** (*simp add: dual-ex [symmetric]*)
qed
qed

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

theorem *dual-meet* [intro?]: $\text{dual } (x \sqcap y) = \text{dual } x \sqcup \text{dual } y$
proof –
from *is-inf-meet* **have** $\text{is-sup } (\text{dual } x) (\text{dual } y) (\text{dual } (x \sqcap y)) \dots$
then have $\text{dual } x \sqcup \text{dual } y = \text{dual } (x \sqcap y) \dots$
then show *?thesis* **..**
qed

theorem *dual-join* [intro?]: $\text{dual } (x \sqcup y) = \text{dual } x \sqcap \text{dual } y$
proof –
from *is-sup-join* **have** $\text{is-inf } (\text{dual } x) (\text{dual } y) (\text{dual } (x \sqcup y)) \dots$
then have $\text{dual } x \sqcap \text{dual } y = \text{dual } (x \sqcup y) \dots$
then show *?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 ... $\sqsubseteq y$..

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 ... $\sqsubseteq z$..

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 $w \sqsubseteq x \sqcap y$ **by fact**

also have ... $\sqsubseteq x$..

finally show *?thesis* .

qed

show $w \sqsubseteq y \sqcap z$

proof

show $w \sqsubseteq y$

proof –

have $w \sqsubseteq x \sqcap y$ **by fact**

also have ... $\sqsubseteq y$..

finally show *?thesis* .

qed

show $w \sqsubseteq z$ **by fact**

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 ... $= dual x \sqcap (dual y \sqcap dual z)$

```

    by (rule meet-assoc)
  also have ... = 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  $\sqsubseteq$  y and z  $\sqsubseteq$  x
  then 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 ... = dual y  $\sqcap$  dual x
    by (rule meet-commute)
  also have ... = 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 by fact
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)
  then have dual (x  $\sqcup$  (x  $\sqcap$  y)) = dual x
    by (simp only: dual-meet dual-join)
  then show ?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  $\text{dual } x \sqcap \text{dual } x = \text{dual } x$ 
    by (rule meet-idem)
  then have  $\text{dual } (x \sqcup x) = \text{dual } x$ 
    by (simp only: dual-join)
  then show ?thesis ..
qed

```

Meet and join are trivial for related elements.

```

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

```

```

theorem join-related [elim?]:  $x \sqsubseteq y \implies x \sqcup y = y$ 
proof –
  assume  $x \sqsubseteq y$  then have  $\text{dual } y \sqsubseteq \text{dual } x$  ..
  then have  $\text{dual } y \sqcap \text{dual } x = \text{dual } y$  by (rule meet-related)
  also have  $\text{dual } y \sqcap \text{dual } x = \text{dual } (y \sqcup x)$  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  $x \sqsubseteq y$ 
  then have is-inf  $x \ y \ x$  ..
  then show  $x \sqcap y = x$  ..
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  $x \sqsubseteq y$ 
  then have is-sup  $x \ y \ y$  ..

```

```

    then show  $x \sqcup y = y$  ..
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 $x \sqcap y = x$ (alternatively as $x \sqcup y = y$). 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.

definition

```

minimum :: 'a::linear-order  $\Rightarrow$  'a  $\Rightarrow$  'a where
minimum x y = (if  $x \sqsubseteq y$  then x else y)

```

definition

```

maximum :: 'a::linear-order  $\Rightarrow$  'a  $\Rightarrow$  'a where
maximum x y = (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* \sqsubseteq *lattice*

```

proof
  fix  $x\ y :: 'a::linear-order$ 
  from is-inf-minimum show  $\exists\ inf. is-inf\ x\ y\ inf\ ..$ 
  from is-sup-maximum show  $\exists\ sup. is-sup\ x\ y\ sup\ ..$ 
qed

```

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

```

theorem meet-minimum:  $x \sqcap y = minimum\ x\ y$ 
  by (rule meet-equality) (rule is-inf-minimum)

```

```

theorem meet-maximum:  $x \sqcup y = maximum\ x\ y$ 
  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:  $is-inf\ p\ q\ (fst\ p \sqcap fst\ q,\ snd\ p \sqcap snd\ q)$ 

```

```

proof
  show  $(fst\ p \sqcap fst\ q,\ snd\ p \sqcap snd\ q) \sqsubseteq p$ 
  proof –
    have  $fst\ p \sqcap fst\ q \sqsubseteq fst\ p\ ..$ 
    moreover have  $snd\ p \sqcap snd\ q \sqsubseteq snd\ p\ ..$ 
    ultimately show ?thesis by (simp add: leq-prod-def)
  qed
  show  $(fst\ p \sqcap fst\ q,\ snd\ p \sqcap snd\ q) \sqsubseteq q$ 
  proof –
    have  $fst\ p \sqcap fst\ q \sqsubseteq fst\ q\ ..$ 
    moreover have  $snd\ p \sqcap snd\ q \sqsubseteq snd\ q\ ..$ 
    ultimately show ?thesis by (simp add: leq-prod-def)
  qed
  fix  $r$  assume  $rp: r \sqsubseteq p$  and  $rq: r \sqsubseteq q$ 
  show  $r \sqsubseteq (fst\ p \sqcap fst\ q,\ snd\ p \sqcap snd\ q)$ 
  proof –
    have  $fst\ r \sqsubseteq fst\ p \sqcap fst\ q$ 
    proof
      from  $rp$  show  $fst\ r \sqsubseteq fst\ p$  by (simp add: leq-prod-def)
      from  $rq$  show  $fst\ r \sqsubseteq fst\ q$  by (simp add: leq-prod-def)
    qed
    moreover have  $snd\ r \sqsubseteq snd\ p \sqcap snd\ q$ 
    proof
      from  $rp$  show  $snd\ r \sqsubseteq snd\ p$  by (simp add: leq-prod-def)
      from  $rq$  show  $snd\ r \sqsubseteq snd\ q$  by (simp add: leq-prod-def)
    qed
    ultimately show ?thesis by (simp add: leq-prod-def)
  qed
qed

```

```

lemma is-sup-prod: is-sup  $p\ q\ (fst\ p\ \sqcup\ fst\ q,\ snd\ p\ \sqcup\ snd\ q)$ 
proof
  show  $p\ \sqsubseteq\ (fst\ p\ \sqcup\ fst\ q,\ snd\ p\ \sqcup\ snd\ q)$ 
  proof –
    have  $fst\ p\ \sqsubseteq\ fst\ p\ \sqcup\ fst\ q\ ..$ 
    moreover have  $snd\ p\ \sqsubseteq\ snd\ p\ \sqcup\ snd\ q\ ..$ 
    ultimately show ?thesis by (simp add: leq-prod-def)
  qed
  show  $q\ \sqsubseteq\ (fst\ p\ \sqcup\ fst\ q,\ snd\ p\ \sqcup\ snd\ q)$ 
  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  $p\ q :: 'a::lattice \times 'b::lattice$ 
  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 is-inf-fun
  show  $\exists sup. is-sup\ f\ g\ sup$  by 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  $\dots \sqsubseteq c$  by fact
      finally show  $a \sqcap b \sqsubseteq c$  .
      show  $a \sqcap b \sqsubseteq b$  ..
    qed
  } note this [elim?]
  assume  $x \sqsubseteq z$  then have  $x \sqcap y \sqsubseteq z \sqcap y$  ..
  also have  $\dots = y \sqcap z$  by (rule meet-commute)
  also assume  $y \sqsubseteq w$  then have  $y \sqcap z \sqsubseteq w \sqcap z$  ..
  also have  $\dots = z \sqcap w$  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  $x \sqsubseteq z$  then have  $dual\ z \sqsubseteq dual\ x$  ..
  moreover assume  $y \sqsubseteq w$  then have  $dual\ w \sqsubseteq dual\ y$  ..
  ultimately have  $dual\ z \sqcap dual\ w \sqsubseteq dual\ x \sqcap dual\ y$ 
    by (rule meet-mono)
  then have  $dual\ (z \sqcup w) \sqsubseteq dual\ (x \sqcup y)$ 
    by (simp only: dual-join)
  then show ?thesis ..
qed

```


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

theorem *meet-semimorph*:

$$(\bigwedge x y. f (x \sqcap y) \sqsubseteq f x \sqcap f y) \equiv (\bigwedge x y. x \sqsubseteq y \implies f x \sqsubseteq f y)$$

proof

assume *morph*: $\bigwedge x y. f (x \sqcap y) \sqsubseteq f x \sqcap f y$

fix $x y :: 'a::\text{lattice}$

assume $x \sqsubseteq y$

then have $x \sqcap y = x$..

then have $x = x \sqcap y$..

also have $f \dots \sqsubseteq f x \sqcap f y$ **by** (*rule morph*)

also have $\dots \sqsubseteq f y$..

finally show $f x \sqsubseteq f y$.

next

assume *mono*: $\bigwedge x y. x \sqsubseteq y \implies f x \sqsubseteq f y$

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 $x \sqcap y \sqsubseteq x$.. **then show** $f (x \sqcap y) \sqsubseteq f x$ **by** (*rule mono*)

have $x \sqcap y \sqsubseteq y$.. **then show** $f (x \sqcap y) \sqsubseteq f y$ **by** (*rule mono*)

qed

qed

qed

lemma *join-semimorph*:

$$(\bigwedge x y. f x \sqcup f y \sqsubseteq f (x \sqcup y)) \equiv (\bigwedge x y. x \sqsubseteq y \implies f x \sqsubseteq f y)$$

proof

assume *morph*: $\bigwedge x y. f x \sqcup f y \sqsubseteq f (x \sqcup y)$

fix $x y :: 'a::\text{lattice}$

assume $x \sqsubseteq y$ **then have** $x \sqcup y = y$..

have $f x \sqsubseteq f x \sqcup f y$..

also have $\dots \sqsubseteq f (x \sqcup y)$ **by** (*rule morph*)

also from $\langle x \sqsubseteq y \rangle$ **have** $x \sqcup y = y$..

finally show $f x \sqsubseteq f y$.

next

assume *mono*: $\bigwedge x y. x \sqsubseteq y \implies f x \sqsubseteq f y$

show $\bigwedge x y. f x \sqcup f y \sqsubseteq f (x \sqcup y)$

proof –

fix $x y$

show $f x \sqcup f y \sqsubseteq f (x \sqcup y)$

proof

have $x \sqsubseteq x \sqcup y$.. **then show** $f x \sqsubseteq f (x \sqcup y)$ **by** (*rule mono*)

have $y \sqsubseteq x \sqcup y$.. **then show** $f y \sqsubseteq f (x \sqcup y)$ **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-Inf* *A inf*

theorem *ex-Sup*: \exists *sup*::'*a*::*complete-lattice*. *is-Sup* *A sup*

proof –

from *ex-Inf* **obtain** *sup* **where** *is-Inf* $\{b. \forall a \in A. a \sqsubseteq b\}$ *sup* **by** *blast*
then have *is-Sup* *A sup* **by** (rule *Inf-Sup*)
then show *?thesis* ..

qed

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

definition

Meet :: '*a*::*complete-lattice* *set* \Rightarrow '*a* **where**
Meet *A* = (*THE inf*. *is-Inf* *A inf*)

definition

Join :: '*a*::*complete-lattice* *set* \Rightarrow '*a* **where**
Join *A* = (*THE sup*. *is-Sup* *A sup*)

notation (*xsymbols*)

Meet (\sqcap - [90] 90) **and**
Join (\sqcup - [90] 90)

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

lemma *Meet-equality* [*elim?*]: *is-Inf* *A inf* $\Rightarrow \sqcap A = inf$

proof (*unfold Meet-def*)

assume *is-Inf* *A inf*

then show (*THE inf*. *is-Inf* *A inf*) = *inf*

by (rule *the-equality*) (rule *is-Inf-uniq* [*OF* - $\langle is-Inf A inf \rangle$])

qed

lemma *MeetI* [*intro?*]:

$(\bigwedge a. a \in A \Rightarrow inf \sqsubseteq a) \Rightarrow$

$(\bigwedge b. \forall a \in A. b \sqsubseteq a \implies b \sqsubseteq \inf) \implies$
 $\bigcap A = \inf$
by (rule Meet-equality, rule is-InfI) blast+

lemma Join-equality [elim?]: $\text{is-Sup } A \text{ sup} \implies \bigsqcup A = \text{sup}$
proof (unfold Join-def)
 assume $\text{is-Sup } A \text{ sup}$
 then show $(\text{THE sup. is-Sup } A \text{ sup}) = \text{sup}$
 by (rule the-equality) (rule is-Sup-uniq [OF - ⟨is-Sup A sup⟩])
qed

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

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

lemma is-Inf-Meet [intro?]: $\text{is-Inf } A (\bigcap A)$
proof (unfold Meet-def)
 from ex-Inf obtain inf where $\text{is-Inf } A \text{ inf} ..$
 then show $\text{is-Inf } A (\text{THE inf. is-Inf } A \text{ inf})$
 by (rule theI) (rule is-Inf-uniq [OF - ⟨is-Inf A inf⟩])
qed

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

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

lemma is-Sup-Join [intro?]: $\text{is-Sup } A (\bigsqcup A)$
proof (unfold Join-def)
 from ex-Sup obtain sup where $\text{is-Sup } A \text{ sup} ..$
 then show $\text{is-Sup } A (\text{THE sup. is-Sup } A \text{ sup})$
 by (rule theI) (rule is-Sup-uniq [OF - ⟨is-Sup A sup⟩])
qed

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

lemma Join-lower [intro?]: $a \in A \implies a \sqsubseteq \bigsqcup A$
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:*

assumes *mono*: $\bigwedge x y. x \sqsubseteq y \implies f x \sqsubseteq f y$

obtains $a :: 'a::\text{complete-lattice}$ **where**

$f a = a$ **and** $\bigwedge a'. f a' = a' \implies a \sqsubseteq a'$

proof

let $?H = \{u. f u \sqsubseteq u\}$

let $?a = \bigcap ?H$

show $f ?a = ?a$

proof –

have $ge: f ?a \sqsubseteq ?a$

proof

fix x **assume** $x: x \in ?H$

then have $?a \sqsubseteq x$..

then have $f ?a \sqsubseteq f x$ **by** (*rule mono*)

also from x **have** $\dots \sqsubseteq x$..

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*)

then show $f ?a \in ?H$..

qed

finally show $?thesis$.

qed

fix a'

assume $f a' = a'$

then have $f a' \sqsubseteq a'$ **by** (*simp only: leq-refl*)

then have $a' \in ?H$..

then show $?a \sqsubseteq a'$..

qed

theorem *Knaster-Tarski-dual:*

assumes *mono*: $\bigwedge x y. x \sqsubseteq y \implies f x \sqsubseteq f y$

obtains $a :: 'a::\text{complete-lattice}$ **where**

$f a = a$ **and** $\bigwedge a'. f a' = a' \implies a' \sqsubseteq a$

proof

let $?H = \{u. u \sqsubseteq f u\}$

let $?a = \bigcup ?H$

show $f ?a = ?a$

proof –

have $le: ?a \sqsubseteq f ?a$

proof

```

    fix x assume x: x ∈ ?H
    then have x ⊆ f x ..
    also from x have x ⊆ ?a ..
    then have f x ⊆ f ?a by (rule mono)
    finally show x ⊆ f ?a .
  qed
  have f ?a ⊆ ?a
  proof
    from le have f ?a ⊆ f (f ?a) by (rule mono)
    then show f ?a ∈ ?H ..
  qed
  from this and le show ?thesis by (rule leq-antisym)
  qed

  fix a'
  assume f a' = a'
  then have a' ⊆ f a' by (simp only: leq-refl)
  then have a' ∈ ?H ..
  then show a' ⊆ ?a ..
  qed

```

4.3 Bottom and top elements

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

definition

bottom :: 'a::complete-lattice (⊥) where
 $\perp = \bigcap UNIV$

definition

top :: 'a::complete-lattice (⊤) where
 $\top = \bigcup UNIV$

lemma *bottom-least* [intro?]: $\perp \subseteq x$

proof (*unfold bottom-def*)

have $x \in UNIV$..

then show $\bigcap UNIV \subseteq x$..

qed

lemma *bottomI* [intro?]: $(\bigwedge a. x \subseteq a) \implies \perp = x$

proof (*unfold bottom-def*)

assume $\bigwedge a. x \subseteq a$

show $\bigcap UNIV = x$

proof

fix a show $x \subseteq a$ by fact

next

fix b :: 'a::complete-lattice

assume b: $\forall a \in UNIV. b \subseteq a$

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 ..
  then show x  $\sqsubseteq$   $\bigsqcup$  UNIV ..
qed

```

```

lemma topI [intro?]: ( $\bigwedge a. a \sqsubseteq x$ )  $\implies \top = x$ 
proof (unfold top-def)
  assume  $\bigwedge a. a \sqsubseteq x$ 
  show  $\bigsqcup$  UNIV = x
  proof
    fix a show a  $\sqsubseteq$  x by fact
  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-Inf A' inf'
  proof -
    have  $\exists$  sup. is-Sup (undual ' A') sup by (rule ex-Sup)
    then have  $\exists$  sup. is-Inf (dual ' undual ' A') (dual sup) by (simp only: dual-Inf)
    then show ?thesis by (simp add: dual-ex [symmetric] image-compose [symmetric])
  qed
qed

```

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

```

theorem dual-Meet [intro?]: dual ( $\sqcap$  A) =  $\sqcup$  (dual ' A)
proof -
  from is-Inf-Meet have is-Sup (dual ' A) (dual ( $\sqcap$  A)) ..
  then have  $\sqcup$  (dual ' A) = dual ( $\sqcap$  A) ..
  then show ?thesis ..
qed

```

```

theorem dual-Join [intro?]: dual ( $\sqcup$  A) =  $\sqcap$  (dual ' A)
proof -

```

```

from is-Sup-Join have is-Inf (dual ‘ A) (dual ( $\sqcup$  A)) ..
then have  $\sqcap$  (dual ‘ A) = dual ( $\sqcup$  A) ..
then show ?thesis ..
qed

```

Likewise are \perp and \top duals of each other.

```

theorem dual-bottom [intro?]: dual  $\perp$  =  $\top$ 
proof –
  have  $\top$  = dual  $\perp$ 
  proof
    fix a' have  $\perp \sqsubseteq$  undual a' ..
    then have dual (undual a')  $\sqsubseteq$  dual  $\perp$  ..
    then show a'  $\sqsubseteq$  dual  $\perp$  by simp
  qed
  then show ?thesis ..
qed

```

```

theorem dual-top [intro?]: dual  $\top$  =  $\perp$ 
proof –
  have  $\perp$  = dual  $\top$ 
  proof
    fix a' have undual a'  $\sqsubseteq$   $\top$  ..
    then have dual  $\top$   $\sqsubseteq$  dual (undual a') ..
    then show dual  $\top$   $\sqsubseteq$  a' by simp
  qed
  then show ?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 ( $\sqcap$  {x, y})
proof –
  have is-Inf {x, y} ( $\sqcap$  {x, y}) ..
  then show ?thesis by (simp only: is-Inf-binary)
qed

```

```

lemma is-sup-binary: is-sup x y ( $\sqcup$  {x, y})
proof –
  have is-Sup {x, y} ( $\sqcup$  {x, y}) ..
  then show ?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  $\exists \text{ inf. } is\text{-inf } x \ y \ \text{inf} \ ..$ 
from is-sup-binary show  $\exists \text{ sup. } is\text{-sup } x \ y \ \text{sup} \ ..$ 
qed

```

```

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

```

```

theorem join-binary:  $x \sqcup y = \sqcup \{x, y\}$ 
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*: $A \subseteq B \implies \sqcap B \sqsubseteq \sqcap A$

proof (*rule Meet-greatest*)

```

  fix a assume  $a \in A$ 
  also assume  $A \subseteq B$ 
  finally have  $a \in B$  .
  then show  $\sqcap B \sqsubseteq a \ ..$ 

```

qed

theorem *Join-subset-mono*: $A \subseteq B \implies \sqcup A \sqsubseteq \sqcup B$

proof –

```

  assume  $A \subseteq B$ 
  then have  $\text{dual } 'A \subseteq \text{dual } 'B$  by blast
  then have  $\sqcap (\text{dual } 'B) \sqsubseteq \sqcap (\text{dual } 'A)$  by (rule Meet-subset-antimono)
  then have  $\text{dual } (\sqcup B) \sqsubseteq \text{dual } (\sqcup A)$  by (simp only: dual-Join)
  then show ?thesis by (simp only: dual-leq)

```

qed

Bounds over unions of sets may be obtained separately.

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

proof

```

  fix a assume  $a \in A \cup B$ 
  then show  $\sqcap A \sqcap \sqcap B \sqsubseteq a$ 
  proof
    assume  $a: a \in A$ 
    have  $\sqcap A \sqcap \sqcap B \sqsubseteq \sqcap A \ ..$ 
    also from a have  $\dots \sqsubseteq a \ ..$ 
    finally show ?thesis .
  
```

next

```

    assume  $a: a \in B$ 
    have  $\sqcap A \sqcap \sqcap B \sqsubseteq \sqcap B \ ..$ 
    also from a have  $\dots \sqsubseteq a \ ..$ 
    finally show ?thesis .
  
```

qed

next

```

  fix b assume  $b: \forall a \in A \cup B. b \sqsubseteq a$ 

```



```

show  $b \sqsubseteq \sqcap A \sqcap \sqcap B$ 
proof
  show  $b \sqsubseteq \sqcap A$ 
  proof
    fix  $a$  assume  $a \in A$ 
    then have  $a \in A \cup B$  ..
    with  $b$  show  $b \sqsubseteq a$  ..
  qed
  show  $b \sqsubseteq \sqcap B$ 
  proof
    fix  $a$  assume  $a \in B$ 
    then have  $a \in A \cup B$  ..
    with  $b$  show  $b \sqsubseteq a$  ..
  qed
qed
qed

```

```

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

```

Bounds over singleton sets are trivial.

```

theorem Meet-singleton:  $\sqcap \{x\} = x$ 
proof
  fix  $a$  assume  $a \in \{x\}$ 
  then have  $a = x$  by simp
  then show  $x \sqsubseteq a$  by (simp only: leq-refl)
next
  fix  $b$  assume  $\forall a \in \{x\}. b \sqsubseteq a$ 
  then show  $b \sqsubseteq x$  by simp
qed

```

```

theorem Join-singleton:  $\sqcup \{x\} = x$ 
proof -
  have  $\text{dual } (\sqcup \{x\}) = \sqcap \{\text{dual } x\}$  by (simp add: dual-Join)
  also have  $\dots = \text{dual } x$  by (rule Meet-singleton)
  finally show ?thesis ..
qed

```

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

```

theorem Meet-empty:  $\sqcap \{\} = \sqcup UNIV$ 
proof

```

```

fix a :: 'a::complete-lattice
assume a ∈ {}
then have False by simp
then show  $\bigsqcup UNIV \sqsubseteq a$  ..
next
fix b :: 'a::complete-lattice
have b ∈ UNIV ..
then show b  $\sqsubseteq \bigsqcup UNIV$  ..
qed

theorem Join-empty:  $\bigsqcup \{\} = \bigcap UNIV$ 
proof -
  have dual ( $\bigsqcup \{\}$ ) =  $\bigcap \{\}$  by (simp add: dual-Join)
  also have ... =  $\bigsqcup UNIV$  by (rule Meet-empty)
  also have ... = dual ( $\bigcap UNIV$ ) 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>.