

# Pseudo-hoops

George Georgescu and Laurențiu Leuștean and Viorel Preoteasa

March 29, 2023

## Abstract

Pseudo-hoops are algebraic structures introduced in [1, 2] by B. Bosbach under the name of complementary semigroups. This is a formalization of the paper [4]. Following [4] we prove some properties of pseudo-hoops and we define the basic concepts of filter and normal filter. The lattice of normal filters is isomorphic with the lattice of congruences of a pseudo-hoop. We also study some important classes of pseudo-hoops. Bounded Wajsberg pseudo-hoops are equivalent to pseudo-Wajsberg algebras and bounded basic pseudo-hoops are equivalent to pseudo-BL algebras. Some examples of pseudo-hoops are given in the last section of the formalization.

## Contents

<b>1</b>	<b>Overview</b>	<b>1</b>
<b>2</b>	<b>Operations</b>	<b>2</b>
<b>3</b>	<b>Left Complemented Monoid</b>	<b>2</b>
<b>4</b>	<b>Right Complemented Monoid</b>	<b>6</b>
<b>5</b>	<b>Pseudo-Hoops</b>	<b>8</b>
<b>6</b>	<b>Filters and Congruences</b>	<b>27</b>
<b>7</b>	<b>Pseudo Wajsberg Algebra</b>	<b>50</b>
<b>8</b>	<b>Some Classes of Pseudo-Hoops</b>	<b>59</b>
<b>9</b>	<b>Examples of Pseudo-Hoops</b>	<b>78</b>

## 1 Overview

Section 2 introduces some operations and their infix syntax. Section 3 and 4 introduces some facts about residuated and complemented monoids. Section

5 introduces the pseudo-hoops and some of their properties. Section 6 introduces filters and normal filters and proves that the lattice of normal filters and the lattice of congruences are isomorphic. Following [3], section 7 introduces pseudo-Waisberg algebras and some of their properties. In Section 8 we investigate some classes of pseudo-hoops. Finally section 9 presents some examples of pseudo-hoops and normal filters.

## 2 Operations

```

theory Operations
imports Main
begin

class left-imp =
  fixes imp-l :: 'a  $\Rightarrow$  'a  $\Rightarrow$  'a (infixr l  $\rightarrow$  65)

class right-imp =
  fixes imp-r :: 'a  $\Rightarrow$  'a  $\Rightarrow$  'a (infixr r  $\rightarrow$  65)

class left-uminus =
  fixes uminus-l :: 'a  $\Rightarrow$  'a (-l - [81] 80)

class right-uminus =
  fixes uminus-r :: 'a  $\Rightarrow$  'a (-r - [81] 80)

end

```

## 3 Left Complemented Monoid

```

theory LeftComplementedMonoid
  imports Operations LatticeProperties.Lattice-Prop
begin

class right-pordered-monoid-mult = order + monoid-mult +
  assumes mult-right-mono:  $a \leq b \implies a * c \leq b * c$ 

class one-greatest = one + ord +
  assumes one-greatest [simp]:  $a \leq 1$ 

class integral-right-pordered-monoid-mult = right-pordered-monoid-mult + one-greatest

class left-residuated = ord + times + left-imp +
  assumes left-residual:  $(x * a \leq b) = (x \leq a \mathit{l}\rightarrow b)$ 

class left-residuated-pordered-monoid = integral-right-pordered-monoid-mult + left-residuated

class left-inf = inf + times + left-imp +
  assumes inf-l-def:  $(a \sqcap b) = (a \mathit{l}\rightarrow b) * a$ 

```

```

class left-complemented-monoid = left-residuated-pordered-monoid + left-inf +
  assumes right-divisibility:  $(a \leq b) = (\exists c . a = c * b)$ 
begin
lemma lcm-D:  $a \rightarrow a = 1$ 
  apply (rule order.antisym, simp)
  by (unfold left-residual [THEN sym], simp)

subclass semilattice-inf
  apply unfold-locales
  apply (metis inf-l-def right-divisibility)
  apply (metis inf-l-def left-residual order-refl)
  by (metis inf-l-def left-residual mult-right-mono right-divisibility)

lemma left-one-inf [simp]:  $1 \sqcap a = a$ 
  by (rule order.antisym, simp-all)

lemma left-one-impl [simp]:  $1 \rightarrow a = a$ 

  proof -
    have  $1 \rightarrow a = 1 \sqcap a$  by (unfold inf-l-def, simp)
    also have  $1 \sqcap a = a$  by simp
    finally show ?thesis .
  qed

lemma lcm-A:  $(a \rightarrow b) * a = (b \rightarrow a) * b$ 
  apply (unfold inf-l-def [THEN sym])
  by (simp add: inf-commute)

lemma lcm-B:  $((a * b) \rightarrow c) = (a \rightarrow (b \rightarrow c))$ 
  apply (rule order.antisym)
  apply (simp add: left-residual [THEN sym] mult.assoc)
  apply (simp add: left-residual)

  apply (simp add: left-residual [THEN sym])
  apply (rule-tac  $y=(b \rightarrow c) * b$  in order-trans)
  apply (simp add: mult.assoc [THEN sym])
  apply (rule mult-right-mono)
  by (simp-all add: left-residual)

lemma lcm-C:  $(a \leq b) = ((a \rightarrow b) = 1)$ 

  proof -
    have  $(a \leq b) = (1 * a \leq b)$  by simp
    also have  $\dots = (1 \leq a \rightarrow b)$  by (unfold left-residual, simp)
    also have  $\dots = (a \rightarrow b = 1)$  apply safe by (rule order.antisym, simp-all)
    finally show ?thesis .
  qed

```

```

end

class less-def = ord +
  assumes less-def:  $(a < b) = ((a \leq b) \wedge \neg (b \leq a))$ 

class one-times = one + times +
  assumes one-left [simp]:  $1 * a = a$ 
  and one-right [simp]:  $a * 1 = a$ 

class left-complemented-monoid-algebra = left-imp + one-times + left-inf + less-def
+
  assumes left-impl-one [simp]:  $a \rightarrow a = 1$ 
  and left-impl-times:  $(a \rightarrow b) * a = (b \rightarrow a) * b$ 
  and left-impl-ded:  $((a * b) \rightarrow c) = (a \rightarrow (b \rightarrow c))$ 
  and left-lesseq:  $(a \leq b) = ((a \rightarrow b) = 1)$ 
begin
  lemma A:  $(1 \rightarrow a) \rightarrow 1 = 1$ 
  proof -
    have  $(1 \rightarrow a) \rightarrow 1 = (1 \rightarrow a) * 1 \rightarrow 1$  by simp
    also have  $\dots = (a \rightarrow 1) * a \rightarrow 1$  by (subst left-impl-times, simp)
    also have  $\dots = 1$  by (simp add: left-impl-ded)
    finally show ?thesis .
  qed

subclass order
proof
  fix x y show  $(x < y) = (x \leq y \wedge \neg y \leq x)$  by (unfold less-def, simp)
next
  fix x show  $x \leq x$  by (unfold left-lesseq, simp)
next
  fix x y z assume a:  $x \leq y$  assume b:  $y \leq z$ 
  have  $x \rightarrow z = 1 * x \rightarrow z$  by simp
  also have  $\dots = (x \rightarrow y) * x \rightarrow z$  by (cut-tac a, simp add: left-lesseq)
  also have  $\dots = (y \rightarrow x) * y \rightarrow z$  by (simp add: left-impl-times)
  also have  $\dots = (y \rightarrow x) \rightarrow (y \rightarrow z)$  by (simp add: left-impl-ded)
  also have  $\dots = (y \rightarrow x) \rightarrow 1$  by (cut-tac b, simp add: left-lesseq)
  also have  $\dots = (1 * y \rightarrow x) \rightarrow 1$  by simp
  also have  $\dots = (1 \rightarrow (y \rightarrow x)) \rightarrow 1$  by (subst left-impl-ded, simp)
  also have  $\dots = 1$  by (simp add: A)
  finally show  $x \leq z$  by (simp add: left-lesseq)
next
  fix x y z assume a:  $x \leq y$  assume b:  $y \leq x$ 
  have  $x = (x \rightarrow y) * x$  by (cut-tac a, simp add: left-lesseq)
  also have  $\dots = (y \rightarrow x) * y$  by (simp add: left-impl-times)
  also have  $\dots = y$  by (cut-tac b, simp add: left-lesseq)
  finally show  $x = y$  .
qed

```

**lemma B:**  $(1 \text{ l}\rightarrow a) \leq 1$   
**apply** (*unfold left-lesseq*)  
**by** (*rule A*)

**lemma C:**  $a \leq (1 \text{ l}\rightarrow a)$   
**by** (*simp add: left-lesseq left-impl-ded [THEN sym]*)

**lemma D:**  $a \leq 1$   
**apply** (*rule-tac y = 1 l\rightarrow a in order-trans*)  
**by** (*simp-all add: C B*)

**lemma less-eq:**  $(a \leq b) = (\exists c . a = (c * b))$

**apply** *safe*  
**apply** (*unfold left-lesseq*)  
**apply** (*rule-tac x = b l\rightarrow a in exI*)  
**apply** (*simp add: left-impl-times*)  
**apply** (*simp add: left-impl-ded*)  
**apply** (*case-tac c \leq 1*)  
**apply** (*simp add: left-lesseq*)  
**by** (*simp add: D*)

**lemma F:**  $(a * b) * c \text{ l}\rightarrow z = a * (b * c) \text{ l}\rightarrow z$   
**by** (*simp add: left-impl-ded*)

**lemma associativity:**  $(a * b) * c = a * (b * c)$

**apply** (*rule order.antisym*)  
**apply** (*unfold left-lesseq*)  
**apply** (*simp add: F*)  
**by** (*simp add: F [THEN sym]*)

**lemma H:**  $a * b \leq b$   
**apply** (*simp add: less-eq*)  
**by** *auto*

**lemma I:**  $a * b \text{ l}\rightarrow b = 1$   
**apply** (*case-tac a \* b \leq b*)  
**apply** (*simp add: left-lesseq*)  
**by** (*simp add: H*)

**lemma K:**  $a \leq b \implies a * c \leq b * c$   
**apply** (*unfold less-eq*)  
**apply** *clarify*  
**apply** (*rule-tac x = ca in exI*)  
**by** (*simp add: associativity*)

**lemma L:**  $(x * a \leq b) = (x \leq a \text{ l}\rightarrow b)$

by (simp add: left-lesseq left-impl-ded)

```
subclass left-complemented-monoid
  apply unfold-locales
  apply (simp-all add: less-def D associativity K)
  apply (simp add: L)
  by (simp add: less-eq)
end
```

```
lemma (in left-complemented-monoid) left-complemented-monoid:
  class.left-complemented-monoid-algebra (*) inf (l→) (≤) (<) 1
  by (unfold-locales, simp-all add: less-le-not-le lcm-A lcm-B lcm-C lcm-D)
```

end

## 4 Right Complemented Monoid

```
theory RightComplementedMonoid
imports LeftComplementedMonoid
begin
```

```
class left-pordered-monoid-mult = order + monoid-mult +
  assumes mult-left-mono:  $a \leq b \implies c * a \leq c * b$ 
```

```
class integral-left-pordered-monoid-mult = left-pordered-monoid-mult + one-greatest
```

```
class right-residuated = ord + times + right-imp +
  assumes right-residual:  $(a * x \leq b) = (x \leq a \ r \rightarrow b)$ 
```

```
class right-residuated-pordered-monoid = integral-left-pordered-monoid-mult + right-residuated
```

```
class right-inf = inf + times + right-imp +
  assumes inf-r-def:  $(a \sqcap b) = a * (a \ r \rightarrow b)$ 
```

```
class right-complemented-monoid = right-residuated-pordered-monoid + right-inf
+
  assumes left-divisibility:  $(a \leq b) = (\exists c . a = b * c)$ 
```

```
sublocale right-complemented-monoid < dual: left-complemented-monoid  $\lambda a b .$ 
 $b * a \ (\sqcap) \ (r \rightarrow) \ 1 \ (\leq) \ (<)$ 
  apply unfold-locales
  apply (simp-all add: inf-r-def mult.assoc mult-left-mono)
  apply (simp add: right-residual)
  by (simp add: left-divisibility)
```

```
context right-complemented-monoid begin
```

```
lemma rcm-D:  $a \ r \rightarrow a = 1$ 
```

```

    by (rule dual.lcm-D)

subclass semilattice-inf
  by unfold-locales

lemma right-semilattice-inf: class.semilattice-inf inf ( $\leq$ ) ( $<$ )
  by unfold-locales

lemma right-one-inf [simp]:  $1 \sqcap a = a$ 
  by simp

lemma right-one-impl [simp]:  $1 r \rightarrow a = a$ 
  by simp

lemma rcm-A:  $a * (a r \rightarrow b) = b * (b r \rightarrow a)$ 
  by (rule dual.lcm-A)

lemma rcm-B:  $((b * a) r \rightarrow c) = (a r \rightarrow (b r \rightarrow c))$ 
  by (rule dual.lcm-B)

lemma rcm-C:  $(a \leq b) = ((a r \rightarrow b) = 1)$ 
  by (rule dual.lcm-C)
end

class right-complemented-monoid-nole-algebra = right-imp + one-times + right-inf
+ less-def +
  assumes right-impl-one [simp]:  $a r \rightarrow a = 1$ 
  and right-impl-times:  $a * (a r \rightarrow b) = b * (b r \rightarrow a)$ 
  and right-impl-ded:  $((a * b) r \rightarrow c) = (b r \rightarrow (a r \rightarrow c))$ 

class right-complemented-monoid-algebra = right-complemented-monoid-nole-algebra
+
  assumes right-lesseq:  $(a \leq b) = ((a r \rightarrow b) = 1)$ 
begin
end

sublocale right-complemented-monoid-algebra < dual-algebra: left-complemented-monoid-algebra
 $\lambda a b . b * a \text{ inf } (r \rightarrow) (\leq) (<) 1$ 
  apply (unfold-locales, simp-all)
  by (rule inf-r-def, rule right-impl-times, rule right-impl-ded, rule right-lesseq)

context right-complemented-monoid-algebra begin

subclass right-complemented-monoid
  apply unfold-locales
  apply simp-all
  apply (simp add: dual-algebra.mult.assoc)
  apply (simp add: dual-algebra.mult-right-mono)
  apply (simp add: dual-algebra.left-residual)

```

**by** (*simp add: dual-algebra.right-divisibility*)  
**end**

**lemma** (*in right-complemented-monoid*) *right-complemented-monoid: class.right-complemented-monoid-algebra*  
( $\leq$ ) ( $<$ )  $1$  ( $*$ ) *inf* ( $r \rightarrow$ )  
**by** (*unfold-locales, simp-all add: less-le-not-le rcm-A rcm-B rcm-C rcm-D*)

**end**

## 5 Pseudo-Hoops

**theory** *PseudoHoops*  
**imports** *RightComplementedMonoid*  
**begin**

**lemma** *drop-assumption:*  
 $p \implies \text{True}$   
**by** *simp*

**class** *pseudo-hoop-algebra* = *left-complemented-monoid-algebra* + *right-complemented-monoid-nole-algebra*  
+  
**assumes** *left-right-impl-times: (a l $\rightarrow$  b) \* a = a \* (a r $\rightarrow$  b)*

**begin**

**definition**

$\text{inf-rr } a \ b = a * (a \ r \rightarrow \ b)$

**definition**

$\text{lesseq-r } a \ b = (a \ r \rightarrow \ b = 1)$

**definition**

$\text{less-r } a \ b = (\text{lesseq-r } a \ b \wedge \neg \text{lesseq-r } b \ a)$

**end**

**context** *pseudo-hoop-algebra* **begin**

**lemma** *right-complemented-monoid-algebra: class.right-complemented-monoid-algebra*  
 $\text{lesseq-r } \text{less-r } 1$  ( $*$ ) *inf-rr* ( $r \rightarrow$ )

**apply** *unfold-locales*

**apply** *simp-all*

**apply** (*simp add: less-r-def*)

**apply** (*simp add: inf-rr-def*)

**apply** (*rule right-impl-times, rule right-impl-ded*)

**by** (*simp add: lesseq-r-def*)



```

lemma inf-rr-inf [simp]: inf-rr = ( $\sqcap$ )
  by (unfold fun-eq-iff, simp add: inf-rr-def inf-l-def left-right-impl-times)

lemma lesseq-lesseq-r: lesseq-r a b = ( $a \leq b$ )
proof –
  interpret A: right-complemented-monoid-algebra lesseq-r less-r 1 (*) inf-rr
(r→)
  by (rule right-complemented-monoid-algebra)
  show lesseq-r a b = ( $a \leq b$ )
  apply (subst le-iff-inf)
  apply (subst A.dual-algebra.inf.absorb-iff1 [of a b])
  apply (unfold inf-rr-inf [THEN sym])
  by simp
qed
lemma [simp]: lesseq-r = ( $\leq$ )
  apply (unfold fun-eq-iff, simp add: lesseq-lesseq-r) done

lemma [simp]: less-r = ( $<$ )
  by (unfold fun-eq-iff, simp add: less-r-def less-le-not-le)

subclass right-complemented-monoid-algebra
  apply (cut-tac right-complemented-monoid-algebra)
  by simp
end

sublocale pseudo-hoop-algebra <
  pseudo-hoop-dual: pseudo-hoop-algebra  $\lambda a b . b * a$  ( $\sqcap$ ) (r→) ( $\leq$ ) ( $<$ ) 1 (l→)
  apply unfold-locales
  apply (simp add: inf-l-def)
  apply simp
  apply (simp add: left-impl-times)
  apply (simp add: left-impl-ded)
  by (simp add: left-right-impl-times)

context pseudo-hoop-algebra begin
lemma commutative-ps: ( $\forall a b . a * b = b * a$ ) = ((l→) = (r→))
  apply (simp add: fun-eq-iff)
  apply safe
  apply (rule order.antisym)
  apply (simp add: right-residual [THEN sym])
  apply (subgoal-tac  $x * (x \text{ l} \rightarrow xa) = (x \text{ l} \rightarrow xa) * x$ )
  apply (drule drop-assumption)
  apply simp
  apply (simp add: left-residual)
  apply simp
  apply (simp add: left-residual [THEN sym])
  apply (simp add: right-residual)

```

**apply** (*rule order.antisym*)  
**apply** (*simp add: left-residual*)  
**apply** (*simp add: right-residual [THEN sym]*)  
**apply** (*simp add: left-residual*)  
**by** (*simp add: right-residual [THEN sym]*)

**lemma** *lemma-2-4-5*:  $a \multimap b \leq (c \multimap a) \multimap (c \multimap b)$   
**apply** (*simp add: left-residual [THEN sym] mult.assoc*)  
**apply** (*rule-tac y = (a \multimap b) \* a in order-trans*)  
**apply** (*rule mult-left-mono*)  
**by** (*simp-all add: left-residual*)

**end**

**context** *pseudo-hoop-algebra* **begin**

**lemma** *lemma-2-4-6*:  $a \multimap b \leq (c \multimap a) \multimap (c \multimap b)$   
**by** (*rule pseudo-hoop-dual.lemma-2-4-5*)

**primrec**

*imp-power-l*::  $'a \Rightarrow \text{nat} \Rightarrow 'a \Rightarrow 'a ((-) \multimap (-) \rightarrow (-) [65,0,65] 65)$  **where**  
 $a \multimap 0 \rightarrow b = b$  |  
 $a \multimap (\text{Suc } n) \rightarrow b = (a \multimap (a \multimap n \rightarrow b))$

**primrec**

*imp-power-r*::  $'a \Rightarrow \text{nat} \Rightarrow 'a \Rightarrow 'a ((-) \multimap (-) \rightarrow (-) [65,0,65] 65)$  **where**  
 $a \multimap 0 \rightarrow b = b$  |  
 $a \multimap (\text{Suc } n) \rightarrow b = (a \multimap (a \multimap n \rightarrow b))$

**lemma** *lemma-2-4-7-a*:  $a \multimap n \rightarrow b = a \wedge n \multimap b$

**apply** (*induct-tac n*)  
**by** (*simp-all add: left-impl-ded*)

**lemma** *lemma-2-4-7-b*:  $a \multimap n \rightarrow b = a \wedge n \multimap b$

**apply** (*induct-tac n*)  
**by** (*simp-all add: right-impl-ded [THEN sym] power-commutes*)

**lemma** *lemma-2-5-8-a* [*simp*]:  $a * b \leq a$

**by** (*rule dual-algebra.H*)

**lemma** *lemma-2-5-8-b* [*simp*]:  $a * b \leq b$

**by** (*rule H*)

**lemma** *lemma-2-5-9-a*:  $a \leq b \multimap a$

**by** (*simp add: left-residual [THEN sym] dual-algebra.H*)

**lemma** *lemma-2-5-9-b*:  $a \leq b \multimap a$

**by** (*simp add: right-residual [THEN sym] H*)

**lemma** *lemma-2-5-11*:  $a * b \leq a \sqcap b$

**by** *simp*

**lemma** *lemma-2-5-12-a*:  $a \leq b \implies c \text{ l} \rightarrow a \leq c \text{ l} \rightarrow b$

**apply** (*subst left-residual [THEN sym]*)

**apply** (*subst left-impl-times*)

**apply** (*rule-tac y = (a l → c) \* b in order-trans*)

**apply** (*simp add: mult-left-mono*)

**by** (*rule H*)

**lemma** *lemma-2-5-13-a*:  $a \leq b \implies b \text{ l} \rightarrow c \leq a \text{ l} \rightarrow c$

**apply** (*simp add: left-residual [THEN sym]*)

**apply** (*rule-tac y = (b l → c) \* b in order-trans*)

**apply** (*simp add: mult-left-mono*)

**by** (*simp add: left-residual*)

**lemma** *lemma-2-5-14*:  $(b \text{ l} \rightarrow c) * (a \text{ l} \rightarrow b) \leq a \text{ l} \rightarrow c$

**apply** (*simp add: left-residual [THEN sym]*)

**apply** (*simp add: mult.assoc*)

**apply** (*subst left-impl-times*)

**apply** (*subst mult.assoc [THEN sym]*)

**apply** (*subst left-residual*)

**by** (*rule dual-algebra.H*)

**lemma** *lemma-2-5-16*:  $(a \text{ l} \rightarrow b) \leq (b \text{ l} \rightarrow c) \text{ r} \rightarrow (a \text{ l} \rightarrow c)$

**apply** (*simp add: right-residual [THEN sym]*)

**by** (*rule lemma-2-5-14*)

**lemma** *lemma-2-5-18*:  $(a \text{ l} \rightarrow b) \leq a * c \text{ l} \rightarrow b * c$

**apply** (*simp add: left-residual [THEN sym]*)

**apply** (*subst mult.assoc [THEN sym]*)

**apply** (*rule mult-right-mono*)

**apply** (*subst left-impl-times*)

**by** (*rule H*)

**end**

**context** *pseudo-hoop-algebra* **begin**

**lemma** *lemma-2-5-12-b*:  $a \leq b \implies c \text{ r} \rightarrow a \leq c \text{ r} \rightarrow b$

**by** (*rule pseudo-hoop-dual.lemma-2-5-12-a*)

**lemma** *lemma-2-5-13-b*:  $a \leq b \implies b \text{ r} \rightarrow c \leq a \text{ r} \rightarrow c$

**by** (*rule pseudo-hoop-dual.lemma-2-5-13-a*)

**lemma** *lemma-2-5-15*:  $(a \text{ r} \rightarrow b) * (b \text{ r} \rightarrow c) \leq a \text{ r} \rightarrow c$

**by** (*rule pseudo-hoop-dual.lemma-2-5-14*)

**lemma** *lemma-2-5-17*:  $(a \text{ r} \rightarrow b) \leq (b \text{ r} \rightarrow c) \text{ l} \rightarrow (a \text{ r} \rightarrow c)$   
**by** (*rule pseudo-hoop-dual.lemma-2-5-16*)

**lemma** *lemma-2-5-19*:  $(a \text{ r} \rightarrow b) \leq c * a \text{ r} \rightarrow c * b$   
**by** (*rule pseudo-hoop-dual.lemma-2-5-18*)

**definition**

*lower-bound*  $A = \{a . \forall x \in A . a \leq x\}$

**definition**

*infimum*  $A = \{a \in \text{lower-bound } A . (\forall x \in \text{lower-bound } A . x \leq a)\}$

**lemma** *infimum-unique*:  $(\text{infimum } A = \{x\}) = (x \in \text{infimum } A)$   
**apply** *safe*  
**apply** *simp*  
**apply** (*rule order.antisym*)  
**by** (*simp-all add: infimum-def*)

**lemma** *lemma-2-6-20*:

$a \in \text{infimum } A \implies b \text{ l} \rightarrow a \in \text{infimum } (((\text{l} \rightarrow) b) 'A)$   
**apply** (*subst infimum-def*)  
**apply** *safe*  
**apply** (*simp add: infimum-def lower-bound-def lemma-2-5-12-a*)  
**by** (*simp add: infimum-def lower-bound-def left-residual [THEN sym]*)

**end**

**context** *pseudo-hoop-algebra* **begin**

**lemma** *lemma-2-6-21*:

$a \in \text{infimum } A \implies b \text{ r} \rightarrow a \in \text{infimum } (((\text{r} \rightarrow) b) 'A)$   
**by** (*rule pseudo-hoop-dual.lemma-2-6-20*)

**lemma** *infimum-pair*:  $a \in \text{infimum } \{x . x = b \vee x = c\} = (a = b \sqcap c)$   
**apply** (*simp add: infimum-def lower-bound-def*)  
**apply** *safe*  
**apply** (*rule order.antisym*)  
**by** *simp-all*

**lemma** *lemma-2-6-20-a*:

$a \text{ l} \rightarrow (b \sqcap c) = (a \text{ l} \rightarrow b) \sqcap (a \text{ l} \rightarrow c)$   
**apply** (*subgoal-tac b \sqcap c \in infimum \{x . x = b \vee x = c\}*)  
**apply** (*drule-tac b = a in lemma-2-6-20*)  
**apply** (*case-tac ((\text{l} \rightarrow) a) ' \{x . ((x = b) \vee (x = c))\} = \{x . x = a \text{ l} \rightarrow b \vee x = a \text{ l} \rightarrow c\}*)  
**apply** (*simp-all add: infimum-pair*)  
**by** *auto*  
**end**

**context** *pseudo-hoop-algebra* **begin**

**lemma** *lemma-2-6-21-a*:

$a \rightarrow (b \sqcap c) = (a \rightarrow b) \sqcap (a \rightarrow c)$   
**by** (*rule pseudo-hoop-dual.lemma-2-6-20-a*)

**lemma** *mult-mono*:  $a \leq b \implies c \leq d \implies a * c \leq b * d$

**apply** (*rule-tac y = a \* d in order-trans*)  
**by** (*simp-all add: mult-right-mono mult-left-mono*)

**lemma** *lemma-2-7-22*:  $(a \rightarrow b) * (c \rightarrow d) \leq (a \sqcap c) \rightarrow (b \sqcap d)$

**apply** (*rule-tac y = (a \sqcap c \rightarrow b) \* (a \sqcap c \rightarrow d) in order-trans*)  
**apply** (*rule mult-mono*)  
**apply** (*simp-all add: lemma-2-5-13-a*)  
**apply** (*rule-tac y = (a \sqcap c \rightarrow b) \sqcap (a \sqcap c \rightarrow d) in order-trans*)  
**apply** (*rule lemma-2-5-11*)  
**by** (*simp add: lemma-2-6-20-a*)

**end**

**context** *pseudo-hoop-algebra* **begin**

**lemma** *lemma-2-7-23*:  $(a \rightarrow b) * (c \rightarrow d) \leq (a \sqcap c) \rightarrow (b \sqcap d)$

**apply** (*rule-tac y = (c \sqcap a) \rightarrow (d \sqcap b) in order-trans*)  
**apply** (*rule pseudo-hoop-dual.lemma-2-7-22*)  
**by** (*simp add: inf-commute*)

**definition**

*upper-bound*  $A = \{a . \forall x \in A . x \leq a\}$

**definition**

*supremum*  $A = \{a \in \text{upper-bound } A . (\forall x \in \text{upper-bound } A . a \leq x)\}$

**lemma** *supremum-unique*:

$a \in \text{supremum } A \implies b \in \text{supremum } A \implies a = b$   
**apply** (*simp add: supremum-def upper-bound-def*)  
**apply** (*rule order.antisym*)  
**by** *auto*

**lemma** *lemma-2-8-i*:

$a \in \text{supremum } A \implies a \rightarrow b \in \text{infimum } ((\lambda x . x \rightarrow b)'A)$   
**apply** (*subst infimum-def*)  
**apply** *safe*  
**apply** (*simp add: supremum-def upper-bound-def lower-bound-def lemma-2-5-13-a*)  
**apply** (*simp add: supremum-def upper-bound-def lower-bound-def left-residual*  
[*THEN sym*])  
**by** (*simp add: right-residual*)

**end**

**context** *pseudo-hoop-algebra* **begin**

**lemma** *lemma-2-8-i1*:

$a \in \text{supremum } A \implies a \text{ r} \rightarrow b \in \text{infimum } ((\lambda x . x \text{ r} \rightarrow b) 'A)$   
**by** (*fact pseudo-hoop-dual.lemma-2-8-i*)

**definition**

*times-set* :: 'a set  $\Rightarrow$  'a set  $\Rightarrow$  'a set (**infixl** \*\* 70) **where**  
 $(A ** B) = \{a . \exists x \in A . \exists y \in B . a = x * y\}$

**lemma** *times-set-assoc*:  $A ** (B ** C) = (A ** B) ** C$

**apply** (*simp add: times-set-def*)  
**apply** *safe*  
**apply** (*rule-tac x = xa \* xb in exI*)  
**apply** *safe*  
**apply** (*rule-tac x = xa in beXI*)  
**apply** (*rule-tac x = xb in beXI*)  
**apply** *simp-all*  
**apply** (*subst mult.assoc*)  
**apply** (*rule-tac x = ya in beXI*)  
**apply** *simp-all*  
**apply** (*rule-tac x = xb in beXI*)  
**apply** *simp-all*  
**apply** (*rule-tac x = ya \* y in exI*)  
**apply** (*subst mult.assoc*)  
**apply** *simp*  
**by** *auto*

**primrec** *power-set* :: 'a set  $\Rightarrow$  nat  $\Rightarrow$  'a set (**infixr** \*<sup>^</sup> 80) **where**

*power-set-0*:  $(A *^{\wedge} 0) = \{1\}$   
*power-set-Suc*:  $(A *^{\wedge} (\text{Suc } n)) = (A ** (A *^{\wedge} n))$

**lemma** *infimum-singleton* [*simp*]:  $\text{infimum } \{a\} = \{a\}$

**apply** (*simp add: infimum-def lower-bound-def*)  
**by** *auto*

**lemma** *lemma-2-8-ii*:

$a \in \text{supremum } A \implies (a \wedge n) \text{ l} \rightarrow b \in \text{infimum } ((\lambda x . x \text{ l} \rightarrow b) '(A *^{\wedge} n))$   
**apply** (*induct-tac n*)  
**apply** *simp*  
**apply** (*simp add: left-impl-ded*)  
**apply** (*drule-tac a = a  $\wedge$  n l  $\rightarrow$  b and b = a in lemma-2-6-20*)  
**apply** (*simp add: infimum-def lower-bound-def times-set-def*)  
**apply** *safe*  
**apply** (*drule-tac b = y l  $\rightarrow$  b in lemma-2-8-i*)

```

apply (simp add: infimum-def lower-bound-def times-set-def left-impl-ded)
apply (rule-tac  $y = a \rightarrow y \rightarrow b$  in order-trans)
apply simp-all
apply (subgoal-tac ( $\forall xa \in A *^{\wedge} n. x \leq a \rightarrow xa \rightarrow b$ ))
apply simp
apply safe
apply (drule-tac  $b = xa \rightarrow b$  in lemma-2-8-i)
apply (simp add: infimum-def lower-bound-def)
apply safe
apply (subgoal-tac ( $\forall xb \in A. x \leq xb \rightarrow xa \rightarrow b$ ))
apply simp
apply safe
apply (subgoal-tac  $x \leq xb * xa \rightarrow b$ )
apply (simp add: left-impl-ded)
apply (subgoal-tac ( $\exists x \in A. \exists y \in A *^{\wedge} n. xb * xa = x * y$ ))
by auto

```

```

lemma power-set-Suc2:
 $A *^{\wedge} (\text{Suc } n) = A *^{\wedge} n ** A$ 
apply (induct-tac  $n$ )
apply (simp add: times-set-def)
apply simp
apply (subst times-set-assoc)
by simp

```

```

lemma power-set-add:
 $A *^{\wedge} (n + m) = (A *^{\wedge} n) ** (A *^{\wedge} m)$ 
apply (induct-tac  $m$ )
apply simp
apply (simp add: times-set-def)
apply simp
apply (subst times-set-assoc)
apply (subst times-set-assoc)
apply (subst power-set-Suc2 [THEN sym])
by simp
end

```

**context** pseudo-hoop-algebra **begin**

```

lemma lemma-2-8-ii1:
 $a \in \text{supremum } A \implies (a \wedge n) r \rightarrow b \in \text{infimum } ((\lambda x . x r \rightarrow b) (A *^{\wedge} n))$ 
apply (induct-tac  $n$ )
apply simp
apply (subst power-Suc2)
apply (subst power-set-Suc2)
apply (simp add: right-impl-ded)
apply (drule-tac  $a = a \wedge n r \rightarrow b$  and  $b = a$  in lemma-2-6-21)
apply (simp add: infimum-def lower-bound-def times-set-def)
apply safe

```

**apply** (*drule-tac*  $b = xa \ r \rightarrow b$  **in** *lemma-2-8-i1*)  
**apply** (*simp add: infimum-def lower-bound-def times-set-def right-impl-ded*)  
**apply** (*rule-tac*  $y = a \ r \rightarrow xa \ r \rightarrow b$  **in** *order-trans*)  
**apply** *simp-all*  
**apply** (*subgoal-tac* ( $\forall xa \in A \ * \wedge \ n. x \leq a \ r \rightarrow xa \ r \rightarrow b$ ))  
**apply** *simp*  
**apply** *safe*  
**apply** (*drule-tac*  $b = xa \ r \rightarrow b$  **in** *lemma-2-8-i1*)  
**apply** (*simp add: infimum-def lower-bound-def*)  
**apply** *safe*  
**apply** (*subgoal-tac* ( $\forall xb \in A. x \leq xb \ r \rightarrow xa \ r \rightarrow b$ ))  
**apply** *simp*  
**apply** *safe*  
**apply** (*subgoal-tac*  $x \leq xa \ * \ xb \ r \rightarrow b$ )  
**apply** (*simp add: right-impl-ded*)  
**apply** (*subgoal-tac* ( $\exists x \in A \ * \wedge \ n. \exists y \in A. xa \ * \ xb = x \ * \ y$ ))  
**by** *auto*

**lemma** *lemma-2-9-i:*

$b \in \text{supremum } A \implies a \ * \ b \in \text{supremum } ((*) \ a \ ' \ A)$   
**apply** (*simp add: supremum-def upper-bound-def*)  
**apply** *safe*  
**apply** (*simp add: mult-left-mono*)  
**by** (*simp add: right-residual*)

**lemma** *lemma-2-9-i1:*

$b \in \text{supremum } A \implies b \ * \ a \in \text{supremum } ((\lambda x. x \ * \ a) \ ' \ A)$   
**apply** (*simp add: supremum-def upper-bound-def*)  
**apply** *safe*  
**apply** (*simp add: mult-right-mono*)  
**by** (*simp add: left-residual*)

**lemma** *lemma-2-9-ii:*

$b \in \text{supremum } A \implies a \ \sqcap \ b \in \text{supremum } ((\sqcap) \ a \ ' \ A)$   
**apply** (*subst supremum-def*)  
**apply** *safe*  
**apply** (*simp add: supremum-def upper-bound-def*)  
**apply** *safe*  
**apply** (*rule-tac*  $y = x$  **in** *order-trans*)  
**apply** *simp-all*  
**apply** (*subst inf-commute*)  
**apply** (*subst inf-l-def*)  
**apply** (*subst left-right-impl-times*)  
**apply** (*frule-tac*  $a = (b \ r \rightarrow a)$  **in** *lemma-2-9-i1*)  
**apply** (*simp add: right-residual*)  
**apply** (*simp add: supremum-def upper-bound-def*)  
**apply** (*simp add: right-residual*)  
**apply** *safe*



```

apply (subgoal-tac ( $\forall xa \in A. b \ r \rightarrow a \leq xa \ r \rightarrow x$ ))
apply simp
apply safe
apply (simp add: inf-l-def)
apply (simp add: left-right-impl-times)
apply (rule-tac  $y = xa \ r \rightarrow b * (b \ r \rightarrow a)$  in order-trans)
apply simp
apply (rule lemma-2-5-12-b)
apply (subst left-residual)
apply (subgoal-tac ( $\forall xa \in A. xa \leq (b \ r \rightarrow a) \ l \rightarrow x$ ))
apply simp
apply safe
apply (subst left-residual [THEN sym])
apply (rule-tac  $y = xaa * (xaa \ r \rightarrow a)$  in order-trans)
apply (rule mult-left-mono)
apply (rule lemma-2-5-13-b)
apply simp
apply (subst right-impl-times)
by simp

```

**lemma** lemma-2-10-24:

```

 $a \leq (a \ l \rightarrow b) \ r \rightarrow b$ 
by (simp add: right-residual [THEN sym] inf-l-def [THEN sym])

```

**lemma** lemma-2-10-25:

```

 $a \leq (a \ l \rightarrow b) \ r \rightarrow a$ 
by (rule lemma-2-5-9-b)
end

```

**context** pseudo-hoop-algebra **begin**

**lemma** lemma-2-10-26:

```

 $a \leq (a \ r \rightarrow b) \ l \rightarrow b$ 
by (rule pseudo-hoop-dual.lemma-2-10-24)

```

**lemma** lemma-2-10-27:

```

 $a \leq (a \ r \rightarrow b) \ l \rightarrow a$ 
by (rule lemma-2-5-9-a)

```

**lemma** lemma-2-10-28:

```

 $b \ l \rightarrow ((a \ l \rightarrow b) \ r \rightarrow a) = b \ l \rightarrow a$ 
apply (rule order.antisym)
apply (subst left-residual [THEN sym])
apply (rule-tac  $y = ((a \ l \rightarrow b) \ r \rightarrow a) \sqcap a$  in order-trans)
apply (subst inf-l-def)
apply (rule-tac  $y = (((a \ l \rightarrow b) \ r \rightarrow a) \ l \rightarrow b) * ((a \ l \rightarrow b) \ r \rightarrow a)$  in order-trans)
apply (subst left-impl-times)
apply simp-all
apply (rule mult-right-mono)
apply (rule-tac  $y = a \ l \rightarrow b$  in order-trans)

```

```

apply (rule lemma-2-5-13-a)
apply (fact lemma-2-10-25)
apply (fact lemma-2-10-26)
apply (rule lemma-2-5-12-a)
by (fact lemma-2-10-25)

```

**end**

**context** *pseudo-hoop-algebra* **begin**

```

lemma lemma-2-10-29:
   $b \ r \rightarrow ((a \ r \rightarrow b) \ l \rightarrow a) = b \ r \rightarrow a$ 
by (rule pseudo-hoop-dual.lemma-2-10-28)

```

```

lemma lemma-2-10-30:
   $((b \ l \rightarrow a) \ r \rightarrow a) \ l \rightarrow a = b \ l \rightarrow a$ 
apply (rule order.antisym)
apply (simp-all add: lemma-2-10-26)
apply (rule lemma-2-5-13-a)
by (rule lemma-2-10-24)

```

**end**

**context** *pseudo-hoop-algebra* **begin**

```

lemma lemma-2-10-31:
   $((b \ r \rightarrow a) \ l \rightarrow a) \ r \rightarrow a = b \ r \rightarrow a$ 
by (rule pseudo-hoop-dual.lemma-2-10-30)

```

```

lemma lemma-2-10-32:
   $((((b \ l \rightarrow a) \ r \rightarrow a) \ l \rightarrow b) \ l \rightarrow (b \ l \rightarrow a)) = b \ l \rightarrow a$ 
proof -
  have  $((((b \ l \rightarrow a) \ r \rightarrow a) \ l \rightarrow b) \ l \rightarrow (b \ l \rightarrow a)) = (((b \ l \rightarrow a) \ r \rightarrow a) \ l \rightarrow b) \ l \rightarrow$ 
 $((b \ l \rightarrow a) \ r \rightarrow a) \ l \rightarrow a)$ 
  by (simp add: lemma-2-10-30)
  also have  $\dots = (((b \ l \rightarrow a) \ r \rightarrow a) \ l \rightarrow b) * ((b \ l \rightarrow a) \ r \rightarrow a) \ l \rightarrow a$ 
  by (simp add: left-impl-ded)
  also have  $\dots = (((b \ l \rightarrow a) \ r \rightarrow a) \sqcap b) \ l \rightarrow a$ 
  by (simp add: inf-l-def)
  also have  $\dots = b \ l \rightarrow a$  apply (subgoal-tac  $((b \ l \rightarrow a) \ r \rightarrow a) \sqcap b = b$ , simp,
rule order.antisym)
  by (simp-all add: lemma-2-10-24)
  finally show ?thesis .
qed

```

**end**

**context** *pseudo-hoop-algebra* **begin**

```

lemma lemma-2-10-33:
  (((b r→ a) l→ a) r→ b) r→ (b r→ a) = b r→ a
  by (rule pseudo-hoop-dual.lemma-2-10-32)
end

class pseudo-hoop-sup-algebra = pseudo-hoop-algebra + sup +
  assumes
    sup-comute: a ⊔ b = b ⊔ a
    and sup-le [simp]: a ≤ a ⊔ b
    and le-sup-equiv: (a ≤ b) = (a ⊔ b = b)
begin
  lemma sup-le-2 [simp]:
    b ≤ a ⊔ b
    by (subst sup-comute, simp)

  lemma le-sup-equiv-r:
    (a ⊔ b = b) = (a ≤ b)
    by (simp add: le-sup-equiv)

  lemma sup-idemp [simp]:
    a ⊔ a = a
    by (simp add: le-sup-equiv-r)
end

class pseudo-hoop-sup1-algebra = pseudo-hoop-algebra + sup +
  assumes
    sup-def: a ⊔ b = ((a l→ b) r→ b) ⊓ ((b l→ a) r→ a)
begin

  lemma sup-comute1: a ⊔ b = b ⊔ a
    apply (simp add: sup-def)
    apply (rule order.antisym)
    by simp-all

  lemma sup-le1 [simp]: a ≤ a ⊔ b
    by (simp add: sup-def lemma-2-10-24 lemma-2-5-9-b)

  lemma le-sup-equiv1: (a ≤ b) = (a ⊔ b = b)
    apply safe
    apply (simp add: left-lesseq)
    apply (rule order.antisym)
    apply (simp add: sup-def)
    apply (simp add: sup-def)
    apply (simp-all add: lemma-2-10-24)
    apply (simp add: le-iff-inf)
    apply (subgoal-tac (a ⊓ b = a ⊓ (a ⊔ b)) ∧ (a ⊓ (a ⊔ b) = a))
    apply simp

```

```

apply safe
apply simp
apply (rule order.antisym)
apply simp
apply (drule drop-assumption)
by (simp add: sup-comute1)

subclass pseudo-hoop-sup-algebra
apply unfold-locales
apply (simp add: sup-comute1)
apply simp
by (simp add: le-sup-equiv1)
end

class pseudo-hoop-sup2-algebra = pseudo-hoop-algebra + sup +
assumes
sup-2-def:  $a \sqcup b = ((a \rightarrow b) \rightarrow b) \sqcap ((b \rightarrow a) \rightarrow a)$ 

context pseudo-hoop-sup1-algebra begin end

sublocale pseudo-hoop-sup2-algebra < sup1-dual: pseudo-hoop-sup1-algebra ( $\sqcup$ )  $\lambda$ 
a b . b * a ( $\sqcap$ ) ( $\rightarrow$ ) ( $\leq$ ) ( $<$ ) 1 ( $\rightarrow$ )
apply unfold-locales
by (simp add: sup-2-def)

context pseudo-hoop-sup2-algebra begin

lemma sup-comute-2:  $a \sqcup b = b \sqcup a$ 
by (rule sup1-dual.sup-comute)

lemma sup-le2 [simp]:  $a \leq a \sqcup b$ 
by (rule sup1-dual.sup-le)

lemma le-sup-equiv2:  $(a \leq b) = (a \sqcup b = b)$ 
by (rule sup1-dual.le-sup-equiv)

subclass pseudo-hoop-sup-algebra
apply unfold-locales
apply (simp add: sup-comute-2)
apply simp
by (simp add: le-sup-equiv2)

end

class pseudo-hoop-lattice-a = pseudo-hoop-sup-algebra +
assumes sup-inf-le-distr:  $a \sqcup (b \sqcap c) \leq (a \sqcup b) \sqcap (a \sqcup c)$ 
begin
lemma sup-lower-upper-bound [simp]:

```

```

 $a \leq c \implies b \leq c \implies a \sqcup b \leq c$ 
apply (subst le-iff-inf)
apply (subgoal-tac ( $a \sqcup b$ )  $\sqcap c = (a \sqcup b) \sqcap (a \sqcup c) \wedge a \sqcup (b \sqcap c) \leq (a \sqcup b)$ )
 $\sqcap (a \sqcup c) \wedge a \sqcup (b \sqcap c) = a \sqcup b$ 
apply (rule order.antisym)
apply simp
apply safe
apply auto[1]
apply (simp add: le-sup-equiv)
apply (rule sup-inf-le-distr)
by (simp add: le-iff-inf)
end

```

```

sublocale pseudo-hoop-lattice-a < lattice ( $\sqcap$ ) ( $\leq$ ) ( $<$ ) ( $\sqcup$ )
by (unfold-locales, simp-all)

```

```

class pseudo-hoop-lattice-b = pseudo-hoop-sup-algebra +
assumes le-sup-cong:  $a \leq b \implies a \sqcup c \leq b \sqcup c$ 

```

```

begin

```

```

lemma sup-lower-upper-bound-b [simp]:
 $a \leq c \implies b \leq c \implies a \sqcup b \leq c$ 
proof –
assume A:  $a \leq c$ 
assume B:  $b \leq c$ 
have  $a \sqcup b \leq c \sqcup b$  by (cut-tac A, simp add: le-sup-cong)
also have  $\dots = b \sqcup c$  by (simp add: sup-comute)
also have  $\dots \leq c \sqcup c$  by (cut-tac B, rule le-sup-cong, simp)
also have  $\dots = c$  by simp
finally show ?thesis .
qed

```

```

lemma sup-inf-le-distr-b:
 $a \sqcup (b \sqcap c) \leq (a \sqcup b) \sqcap (a \sqcup c)$ 
apply (rule sup-lower-upper-bound-b)
apply simp
apply simp
apply safe
apply (subst sup-comute)
apply (rule-tac y = b in order-trans)
apply simp-all
apply (rule-tac y = c in order-trans)
by simp-all

```

```

end

```

```

context pseudo-hoop-lattice-a begin end

```

```

sublocale pseudo-hoop-lattice-b < pseudo-hoop-lattice-a ( $\sqcup$ ) ( $*$ ) ( $\sqcap$ ) ( $l \rightarrow$ ) ( $\leq$ ) ( $<$ )
1 ( $r \rightarrow$ )

```

```

by (unfold-locales, rule sup-inf-le-distr-b)

class pseudo-hoop-lattice = pseudo-hoop-sup-algebra +
  assumes sup-assoc-1:  $a \sqcup (b \sqcup c) = (a \sqcup b) \sqcup c$ 
begin
  lemma le-sup-cong-c:
     $a \leq b \implies a \sqcup c \leq b \sqcup c$ 
  proof -
    assume A:  $a \leq b$ 
    have  $a \sqcup c \sqcup (b \sqcup c) = a \sqcup (c \sqcup (b \sqcup c))$  by (simp add: sup-assoc-1)
    also have  $\dots = a \sqcup ((b \sqcup c) \sqcup c)$  by (simp add: sup-comute)
    also have  $\dots = a \sqcup (b \sqcup (c \sqcup c))$  by (simp add: sup-assoc-1 [THEN sym])
    also have  $\dots = (a \sqcup b) \sqcup c$  by (simp add: sup-assoc-1)
    also have  $\dots = b \sqcup c$  by (cut-tac A, simp add: le-sup-equiv)
    finally show ?thesis by (simp add: le-sup-equiv)
  qed
end

sublocale pseudo-hoop-lattice < pseudo-hoop-lattice-b ( $\sqcup$ ) ( $*$ ) ( $\sqcap$ ) ( $l \rightarrow$ ) ( $\leq$ ) ( $<$ )
1 ( $r \rightarrow$ )
by (unfold-locales, rule le-sup-cong-c)

sublocale pseudo-hoop-lattice < semilattice-sup ( $\sqcup$ ) ( $\leq$ ) ( $<$ )
by (unfold-locales, simp-all)

sublocale pseudo-hoop-lattice < lattice ( $\sqcap$ ) ( $\leq$ ) ( $<$ ) ( $\sqcup$ )
by unfold-locales

lemma (in pseudo-hoop-lattice-a) supremum-pair [simp]:
  supremum  $\{a, b\} = \{a \sqcup b\}$ 
  apply (simp add: supremum-def upper-bound-def)
  apply safe
  apply simp-all
  apply (rule order.antisym)
  by simp-all

sublocale pseudo-hoop-lattice < distrib-lattice ( $\sqcap$ ) ( $\leq$ ) ( $<$ ) ( $\sqcup$ )
  apply unfold-locales
  apply (rule distrib-imp1)
  apply (case-tac xa  $\sqcap$  (ya  $\sqcup$  za)  $\in$  supremum (( $\sqcap$ ) xa ‘ $\{ya, za\}$ ’))
  apply (simp add: supremum-pair)
  apply (erule notE)
  apply (rule lemma-2-9-ii)
  by (simp add: supremum-pair)

class bounded-semilattice-inf-top = semilattice-inf + order-top
begin

```

**lemma** *inf-eq-top-iff* [simp]:  
 $(\text{inf } x \ y = \text{top}) = (x = \text{top} \wedge y = \text{top})$   
**by** (*simp add: order.eq-iff*)  
**end**

**sublocale** *pseudo-hoop-algebra* < *bounded-semilattice-inf-top* ( $\sqcap$ ) ( $\leq$ ) ( $<$ ) 1  
**by** *unfold-locale simp*

**definition** (**in** *pseudo-hoop-algebra*)  
 $\text{sup1}::'a \Rightarrow 'a \Rightarrow 'a$  (**infixl**  $\sqcup 1$  70) **where**  
 $a \sqcup 1 \ b = ((a \ l \rightarrow \ b) \ r \rightarrow \ b) \sqcap ((b \ l \rightarrow \ a) \ r \rightarrow \ a)$

**sublocale** *pseudo-hoop-algebra* < *sup1: pseudo-hoop-sup1-algebra* ( $\sqcup 1$ ) ( $*$ ) ( $\sqcap$ )  
 $(l \rightarrow)$  ( $\leq$ ) ( $<$ ) 1 ( $r \rightarrow$ )  
**apply** *unfold-locale*  
**by** (*simp add: sup1-def*)

**definition** (**in** *pseudo-hoop-algebra*)  
 $\text{sup2}::'a \Rightarrow 'a \Rightarrow 'a$  (**infixl**  $\sqcup 2$  70) **where**  
 $a \sqcup 2 \ b = ((a \ r \rightarrow \ b) \ l \rightarrow \ b) \sqcap ((b \ r \rightarrow \ a) \ l \rightarrow \ a)$

**sublocale** *pseudo-hoop-algebra* < *sup2: pseudo-hoop-sup2-algebra* ( $\sqcup 2$ ) ( $*$ ) ( $\sqcap$ )  
 $(l \rightarrow)$  ( $\leq$ ) ( $<$ ) 1 ( $r \rightarrow$ )  
**apply** *unfold-locale*  
**by** (*simp add: sup2-def*)

**context** *pseudo-hoop-algebra*  
**begin**

**lemma** *lemma-2-15-i*:  
 $1 \in \text{supremum } \{a, b\} \Longrightarrow a * b = a \sqcap b$   
**apply** (*rule order.antisym*)  
**apply** (*rule lemma-2-5-11*)  
**apply** (*simp add: inf-l-def*)  
**apply** (*subst left-impl-times*)  
**apply** (*rule mult-right-mono*)  
**apply** (*subst right-lesseq*)  
**apply** (*subgoal-tac a  $\sqcup 1$  b = 1*)  
**apply** (*simp add: sup1-def*)  
**apply** (*rule order.antisym*)  
**apply** *simp*  
**by** (*simp add: supremum-def upper-bound-def*)

**lemma** *lemma-2-15-ii*:  
 $1 \in \text{supremum } \{a, b\} \Longrightarrow a \leq c \Longrightarrow b \leq d \Longrightarrow 1 \in \text{supremum } \{c, d\}$   
**apply** (*simp add: supremum-def upper-bound-def*)  
**apply** *safe*

**apply** (*drule-tac*  $x = x$  **in** *spec*)  
**apply** *safe*  
**by** *simp-all*

**lemma** *sup-union*:

$a \in \text{supremum } A \implies b \in \text{supremum } B \implies \text{supremum } \{a, b\} = \text{supremum } (A \cup B)$

**apply** *safe*  
**apply** (*simp-all add: supremum-def upper-bound-def*)  
**apply** *safe*  
**apply** *auto*  
**apply** (*subgoal-tac* ( $\forall x \in A \cup B. x \leq xa$ ))  
**by** *auto*

**lemma** *sup-singleton* [*simp*]:  $a \in \text{supremum } \{a\}$

**by** (*simp add: supremum-def upper-bound-def*)

**lemma** *sup-union-singleton*:  $a \in \text{supremum } X \implies \text{supremum } \{a, b\} = \text{supremum } (X \cup \{b\})$

**apply** (*rule-tac*  $B = \{b\}$  **in** *sup-union*)  
**by** *simp-all*

**lemma** *sup-le-union* [*simp*]:  $a \leq b \implies \text{supremum } (A \cup \{a, b\}) = \text{supremum } (A \cup \{b\})$

**apply** (*simp add: supremum-def upper-bound-def*)  
**by** *auto*

**lemma** *sup-sup-union*:  $a \in \text{supremum } A \implies b \in \text{supremum } (B \cup \{a\}) \implies b \in \text{supremum } (A \cup B)$

**apply** (*simp add: supremum-def upper-bound-def*)  
**by** *auto*

**end**

**lemma** [*simp*]:

$n \leq 2 \wedge n$   
**apply** (*induct-tac*  $n$ )  
**apply** *auto*  
**apply** (*rule-tac*  $y = 1 + 2 \wedge n$  **in** *order-trans*)  
**by** *auto*

**context** *pseudo-hoop-algebra*

**begin**

**lemma** *sup-le-union-2*:



```

a ≤ b ⇒ a ∈ A ⇒ b ∈ A ⇒ supremum A = supremum ((A - {a}) ∪ {b})
apply (case-tac supremum ((A - {a, b}) ∪ {a, b}) = supremum ((A - {a, b})
∪ {b}))
apply (case-tac ((A - {a, b}) ∪ {a, b} = A) ∧ ((A - {a, b}) ∪ {b} = (A -
{a}) ∪ {b}))
apply safe[1]
apply simp
apply simp
apply (erule notE)
apply safe [1]
apply (erule notE)
apply (rule sup-le-union)
by simp

```

**lemma** lemma-2-15-iii-0:

```

1 ∈ supremum {a, b} ⇒ 1 ∈ supremum {a ^ 2, b ^ 2}
apply (frule-tac a = a in lemma-2-9-i)
apply simp
apply (frule-tac a = a and b = b in sup-union-singleton)
apply (subgoal-tac supremum ({a * a, a * b} ∪ {b}) = supremum ({a * a, b}))
apply simp-all
apply (frule-tac a = b in lemma-2-9-i)
apply simp
apply (drule-tac a = b and A = {b * (a * a), b * b} and b = 1 and B = {a
* a} in sup-sup-union)
apply simp
apply (case-tac {a * a, b} = {b, a * a})
apply simp
apply auto [1]
apply simp
apply (subgoal-tac supremum {a * a, b * (a * a), b * b} = supremum {a * a,
b * b})
apply (simp add: power2-eq-square)
apply (case-tac b * (a * a) = b * b)
apply auto[1]
apply (cut-tac A = {a * a, b * (a * a), b * b} and a = b * (a * a) and b =
a * a in sup-le-union-2)
apply simp
apply simp
apply simp
apply (subgoal-tac ({a * a, b * (a * a), b * b} - {b * (a * a)}) ∪ {a * a} =
{a * a, b * b})
apply simp
apply auto[1]
apply (case-tac a * a = a * b)
apply (subgoal-tac {b, a * a, a * b} = {a * a, b})
apply simp
apply auto[1]

```

```

    apply (cut-tac A = {b, a * a, a * b} and a = a * b and b = b in
sup-le-union-2)
    apply simp
    apply simp
    apply simp
    apply (subgoal-tac {b, a * a, a * b} - {a * b} ∪ {b} = {a * a, b})
    apply simp
    by auto

```

```

lemma [simp]: m ≤ n ⇒ a ^ n ≤ a ^ m
  apply (subgoal-tac a ^ n = (a ^ m) * (a ^ (n-m)))
  apply simp
  apply (cut-tac a = a and m = m and n = n - m in power-add)
  by simp

```

```

lemma [simp]: a ^ (2 ^ n) ≤ a ^ n
  by simp

```

```

lemma lemma-2-15-iii-1: 1 ∈ supremum {a, b} ⇒ 1 ∈ supremum {a ^ (2 ^ n),
b ^ (2 ^ n)}
  apply (induct-tac n)
  apply auto[1]
  apply (drule drop-assumption)
  apply (drule lemma-2-15-iii-0)
  apply (subgoal-tac ∀ a . (a ^ (2::nat) ^ n)2 = a ^ (2::nat) ^ Suc n)
  apply simp
  apply safe
  apply (cut-tac a = aa and m = 2 ^ n and n = 2 in power-mult)
  apply auto
  apply (subgoal-tac ((2::nat) ^ n * (2::nat)) = ((2::nat) * (2::nat) ^ n))
  by simp-all

```

```

lemma lemma-2-15-iii:
  1 ∈ supremum {a, b} ⇒ 1 ∈ supremum {a ^ n, b ^ n}
  apply (drule-tac n = n in lemma-2-15-iii-1)
  apply (simp add: supremum-def upper-bound-def)
  apply safe
  apply (drule-tac x = x in spec)
  apply safe
  apply (rule-tac y = a ^ n in order-trans)
  apply simp-all
  apply (rule-tac y = b ^ n in order-trans)
  by simp-all

```

end

end

## 6 Filters and Congruences

```
theory PseudoHoopFilters
imports PseudoHoops
begin
```

```
context pseudo-hoop-algebra
```

```
begin
```

```
definition
```

```
filters = {F . F ≠ {} ∧ (∀ a b . a ∈ F ∧ b ∈ F → a * b ∈ F) ∧ (∀ a b . a ∈ F ∧ a ≤ b → b ∈ F)}
```

```
definition
```

```
properfilters = {F . F ∈ filters ∧ F ≠ UNIV}
```

```
definition
```

```
maximalfilters = {F . F ∈ filters ∧ (∀ A . A ∈ filters ∧ F ⊆ A → A = F ∨ A = UNIV)}
```

```
definition
```

```
ultrafilters = properfilters ∩ maximalfilters
```

```
lemma filter-i: F ∈ filters ⇒ a ∈ F ⇒ b ∈ F ⇒ a * b ∈ F
by (simp add: filters-def)
```

```
lemma filter-ii: F ∈ filters ⇒ a ∈ F ⇒ a ≤ b ⇒ b ∈ F
by (simp add: filters-def)
```

```
lemma filter-iii [simp]: F ∈ filters ⇒ 1 ∈ F
by (auto simp add: filters-def)
```

```
lemma filter-left-impl:
```

```
(F ∈ filters) = ((1 ∈ F) ∧ (∀ a b . a ∈ F ∧ a l→ b ∈ F → b ∈ F))
```

```
apply safe
```

```
apply simp
```

```
apply (frule-tac a = a l→ b and b = a in filter-i)
```

```
apply simp
```

```
apply simp
```

```
apply (rule-tac a = (a l→ b) * a in filter-ii)
```

```
apply simp
```

```
apply simp
```

```
apply (simp add: inf-l-def [THEN sym])
```

```
apply (subst filters-def)
```

```
apply safe
```

```
apply (subgoal-tac a l→ (b l→ a * b) ∈ F)
```

```
apply blast
```

```
apply (subst left-impl-ded [THEN sym])
```

```
apply (subst left-impl-one)
```

```
apply safe
```

**apply** (*subst (asm) left-lesseq*)  
**by** *blast*

**lemma** *filter-right-impl*:  
 $(F \in \text{filters}) = ((1 \in F) \wedge (\forall a b . a \in F \wedge a r \rightarrow b \in F \longrightarrow b \in F))$   
**apply** *safe*  
**apply** *simp*  
**apply** (*frule-tac a = a and b = a r → b in filter-i*)  
**apply** *simp*  
**apply** *simp*  
**apply** (*rule-tac a = a \* (a r → b) in filter-ii*)  
**apply** *simp*  
**apply** *simp*  
**apply** (*simp add: inf-r-def [THEN sym]*)  
**apply** (*subst filters-def*)  
**apply** *safe*  
**apply** (*subgoal-tac b r → (a r → a \* b) ∈ F*)  
**apply** *blast*  
**apply** (*subst right-impl-ded [THEN sym]*)  
**apply** (*subst right-impl-one*)  
**apply** *safe*  
**apply** (*subst (asm) right-lesseq*)  
**by** *blast*

**lemma** [*simp*]:  $A \subseteq \text{filters} \implies \bigcap A \in \text{filters}$   
**apply** (*simp add: filters-def*)  
**apply** *safe*  
**apply** (*simp add: Inter-eq*)  
**apply** (*drule-tac x = 1 in spec*)  
**apply** *safe*  
**apply** (*erule notE*)  
**apply** (*subgoal-tac x ∈ filters*)  
**apply** *simp*  
**apply** (*simp add: filters-def*)  
**apply** *blast*  
**apply** (*frule rev-subsetD*)  
**apply** *simp*  
**apply** *simp*  
**apply** (*frule rev-subsetD*)  
**apply** *simp*  
**apply** (*subgoal-tac a ∈ X*)  
**apply** *blast*  
**by** *blast*

**definition**

$\text{filterof } X = \bigcap \{F . F \in \text{filters} \wedge X \subseteq F\}$

**lemma** [*simp*]:  $\text{filterof } X \in \text{filters}$   
**by** (*auto simp add: filterof-def*)

**lemma** *times-le-mono* [*simp*]:  $x \leq y \implies u \leq v \implies x * u \leq y * v$   
**apply** (*rule-tac*  $y = x * v$  **in** *order-trans*)  
**by** (*simp-all add: mult-left-mono mult-right-mono*)

**lemma** *prop-3-2-i*:

*filterof*  $X = \{a . \exists n x . x \in X * ^n \wedge x \leq a\}$   
**apply** *safe*  
**apply** (*subgoal-tac*  $\{a . \exists n x . x \in X * ^n \wedge x \leq a\} \in \text{filters}$ )  
**apply** (*simp add: filterof-def*)  
**apply** (*drule-tac*  $x = \{a::'a . \exists (n::nat) x::'a . x \in X * ^n \wedge x \leq a\}$  **in** *spec*)  
**apply** *safe*  
**apply** (*rule-tac*  $x = 1::nat$  **in** *exI*)  
**apply** (*rule-tac*  $x = xa$  **in** *exI*)  
**apply** (*simp add: times-set-def*)  
**apply** (*drule drop-assumption*)  
**apply** (*simp add: filters-def*)  
**apply** *safe*  
**apply** (*rule-tac*  $x = 1$  **in** *exI*)  
**apply** (*rule-tac*  $x = 0$  **in** *exI*)  
**apply** (*rule-tac*  $x = 1$  **in** *exI*)  
**apply** *simp*  
**apply** (*rule-tac*  $x = n + na$  **in** *exI*)  
**apply** (*rule-tac*  $x = x * xa$  **in** *exI*)  
**apply** *safe*  
**apply** (*simp add: power-set-add times-set-def*)  
**apply** *blast*  
**apply** *simp*  
**apply** (*rule-tac*  $x = n$  **in** *exI*)  
**apply** (*rule-tac*  $x = x$  **in** *exI*)  
**apply** *simp*  
**apply** (*simp add: filterof-def*)  
**apply** *safe*  
**apply** (*rule filter-ii*)  
**apply** *simp-all*  
**apply** (*subgoal-tac*  $\forall x . x \in X * ^n \longrightarrow x \in xb$ )  
**apply** *simp*  
**apply** (*induct-tac*  $n$ )  
**apply** (*simp add: power-set-0*)  
**apply** (*simp add: power-set-Suc times-set-def*)  
**apply** *safe*  
**apply** (*rule filter-i*)  
**apply** *simp-all*  
**by** *blast*

**lemma** *ultrafilter-union*:

*ultrafilters* =  $\{F . F \in \text{filters} \wedge F \neq \text{UNIV} \wedge (\forall x . x \notin F \longrightarrow \text{filterof} (F \cup \{x\}) = \text{UNIV})\}$   
**apply** (*simp add: ultrafilters-def maximalfilters-def properfilters-def filterof-def*)

by *auto*

**lemma** *filterof-sub*:  $F \in \text{filters} \implies X \subseteq F \implies \text{filterof } X \subseteq F$   
 apply (*simp add: filterof-def*)  
 by *blast*

**lemma** *filterof-elem* [*simp*]:  $x \in X \implies x \in \text{filterof } X$   
 apply (*simp add: filterof-def*)  
 by *blast*

**lemma** [*simp*]:  $\text{filterof } X \in \text{filters}$   
 apply (*simp add: filters-def prop-3-2-i*)  
 apply *safe*  
 apply (*rule-tac x = 1 in exI*)  
 apply (*rule-tac x = 0 in exI*)  
 apply (*rule-tac x = 1 in exI*)  
 apply *auto* [1]  
 apply (*rule-tac x = n + na in exI*)  
 apply (*rule-tac x = x \* xa in exI*)  
 apply *safe*  
 apply (*unfold power-set-add*)  
 apply (*simp add: times-set-def*)  
 apply *auto* [1]  
 apply (*rule-tac y = x \* b in order-trans*)  
 apply (*rule mult-left-mono*)  
 apply *simp*  
 apply (*simp add: mult-right-mono*)  
 apply (*rule-tac x = n in exI*)  
 apply (*rule-tac x = x in exI*)  
 by *simp*

**lemma** *singleton-power* [*simp*]:  $\{a\} *^{\wedge} n = \{b \mid b = a^{\wedge} n\}$   
 apply (*induct-tac n*)  
 apply *auto* [1]  
 by (*simp add: times-set-def*)

**lemma** *power-pair*:  $x \in \{a, b\} *^{\wedge} n \implies \exists i j . i + j = n \wedge x \leq a^{\wedge} i \wedge x \leq b^{\wedge} j$   
 apply (*subgoal-tac*  $\forall x . x \in \{a, b\} *^{\wedge} n \longrightarrow (\exists i j . i + j = n \wedge x \leq a^{\wedge} i \wedge x \leq b^{\wedge} j)$ )  
 apply *auto*[1]  
 apply (*drule drop-assumption*)  
 apply (*induct-tac n*)  
 apply *auto* [1]  
 apply *safe*  
 apply (*simp add: times-set-def*)  
 apply *safe*  
 apply (*drule-tac x = y in spec*)

**apply** *safe*  
**apply** (*rule-tac*  $x = i + 1$  **in** *exI*)  
**apply** (*rule-tac*  $x = j$  **in** *exI*)  
**apply** *simp*  
**apply** (*rule-tac*  $y = y$  **in** *order-trans*)  
**apply** *simp-all*  
**apply** (*drule-tac*  $x = y$  **in** *spec*)  
**apply** *safe*  
**apply** (*rule-tac*  $x = i$  **in** *exI*)  
**apply** (*rule-tac*  $x = j+1$  **in** *exI*)  
**apply** *simp*  
**apply** (*rule-tac*  $y = y$  **in** *order-trans*)  
**by** *simp-all*

**lemma** *filterof-supremum*:

$c \in \text{supremum } \{a, b\} \implies \text{filterof } \{c\} = \text{filterof } \{a\} \cap \text{filterof } \{b\}$

**apply** *safe*  
**apply** (*cut-tac*  $X = \{c\}$  **and**  $F = \text{filterof } \{a\}$  **in** *filterof-sub*)  
**apply** *simp-all*  
**apply** (*simp add: supremum-def upper-bound-def*)  
**apply** *safe*  
**apply** (*rule-tac*  $a = a$  **in** *filter-ii*)  
**apply** *simp-all*  
**apply** *blast*  
**apply** (*cut-tac*  $X = \{c\}$  **and**  $F = \text{filterof } \{b\}$  **in** *filterof-sub*)  
**apply** *simp-all*  
**apply** (*simp add: supremum-def upper-bound-def*)  
**apply** *safe*  
**apply** (*rule-tac*  $a = b$  **in** *filter-ii*)  
**apply** *simp-all*  
**apply** *blast*  
**apply** (*subst (asm) prop-3-2-i*)  
**apply** *simp*  
**apply** (*subst (asm) prop-3-2-i*)  
**apply** *simp*  
**apply** *safe*  
**apply** (*cut-tac*  $A = \{a, b\}$  **and**  $a = c$  **and**  $b = x$  **and**  $n = n + na$  **in**  
*lemma-2-8-ii1*)  
**apply** *simp*  
**apply** (*subst prop-3-2-i*)  
**apply** *simp*  
**apply** (*rule-tac*  $x = n + na$  **in** *exI*)  
**apply** (*subgoal-tac infimum* ( $(\lambda xa::'a. xa \ r \rightarrow x) \text{ ` } (\{a, b\} \text{ * } \wedge (n + na)) = \{1\}$ ))  
**apply** *simp*  
**apply** (*simp add: right-lesseq*)  
**apply** (*subst infimum-unique*)  
**apply** (*subst infimum-def lower-bound-def*)  
**apply** (*subst lower-bound-def*)  
**apply** *safe*

**apply** *simp-all*  
**apply** (*drule power-pair*)  
**apply** *safe*  
**apply** (*subst right-residual [THEN sym]*)  
**apply** *simp*  
**apply** (*case-tac n ≤ i*)  
**apply** (*rule-tac y = a ^ n in order-trans*)  
**apply** (*rule-tac y = a ^ i in order-trans*)  
**apply** *simp-all*  
**apply** (*subgoal-tac na ≤ j*)  
**apply** (*rule-tac y = b ^ na in order-trans*)  
**apply** (*rule-tac y = b ^ j in order-trans*)  
**by** *simp-all*

**definition** *d1*  $a\ b = (a\ l\rightarrow\ b) * (b\ l\rightarrow\ a)$

**definition** *d2*  $a\ b = (a\ r\rightarrow\ b) * (b\ r\rightarrow\ a)$

**definition** *d3*  $a\ b = d1\ b\ a$

**definition** *d4*  $a\ b = d2\ b\ a$

**lemma** [*simp*]:  $(a * b = 1) = (a = 1 \wedge b = 1)$

**apply** (*rule iffI*)  
**apply** (*rule conjI*)  
**apply** (*rule order.antisym*)  
**apply** *simp*  
**apply** (*rule-tac y = a\*b in order-trans*)  
**apply** *simp*  
**apply** (*drule drop-assumption*)  
**apply** *simp*  
**apply** (*rule order.antisym*)  
**apply** *simp*  
**apply** (*rule-tac y = a\*b in order-trans*)  
**apply** *simp*  
**apply** (*drule drop-assumption*)  
**apply** *simp*  
**by** *simp*

**lemma** *lemma-3-5-i-1*:  $(d1\ a\ b = 1) = (a = b)$

**apply** (*simp add: d1-def left-lesseq [THEN sym]*)  
**by** *auto*

**lemma** *lemma-3-5-i-2*:  $(d2\ a\ b = 1) = (a = b)$

**apply** (*simp add: d2-def right-lesseq [THEN sym]*)  
**by** *auto*

**lemma** *lemma-3-5-i-3*:  $(d3\ a\ b = 1) = (a = b)$

**apply** (*simp add: d3-def lemma-3-5-i-1*)



```

    by auto

lemma lemma-3-5-i-4: (d4 a b = 1) = (a = b)
  apply (simp add: d4-def lemma-3-5-i-2)
  by auto

lemma lemma-3-5-ii-1 [simp]: d1 a a = 1
  apply (subst lemma-3-5-i-1)
  by simp

lemma lemma-3-5-ii-2 [simp]: d2 a a = 1
  apply (subst lemma-3-5-i-2)
  by simp

lemma lemma-3-5-ii-3 [simp]: d3 a a = 1
  apply (subst lemma-3-5-i-3)
  by simp

lemma lemma-3-5-ii-4 [simp]: d4 a a = 1
  apply (subst lemma-3-5-i-4)
  by simp

lemma [simp]: (a l→ 1) = 1
  by (simp add: left-lesseq [THEN sym])

end

context pseudo-hoop-algebra begin

lemma [simp]: (a r→ 1) = 1
  by simp

lemma lemma-3-5-iii-1 [simp]: d1 a 1 = a
  by (simp add: d1-def)

lemma lemma-3-5-iii-2 [simp]: d2 a 1 = a
  by (simp add: d2-def)

lemma lemma-3-5-iii-3 [simp]: d3 a 1 = a
  by (simp add: d3-def d1-def)

lemma lemma-3-5-iii-4 [simp]: d4 a 1 = a
  by (simp add: d4-def d2-def)

lemma lemma-3-5-iv-1: (d1 b c) * (d1 a b) * (d1 b c) ≤ d1 a c
  apply (simp add: d1-def)
  apply (subgoal-tac (b l→ c) * (c l→ b) * ((a l→ b) * (b l→ a)) * ((b l→ c) * (c
l→ b)) =
    ((b l→ c) * (c l→ b) * (a l→ b)) * ((b l→ a) * (b l→ c) * (c l→ b)))

```

```

apply simp
apply (rule mult-mono)
apply (rule-tac y = (b l→ c) * (a l→ b) in order-trans)
apply (rule mult-right-mono)
apply simp
apply (simp add: lemma-2-5-14)
apply (rule-tac y = (b l→ a) * (c l→ b) in order-trans)
apply (rule mult-right-mono)
apply simp
apply (simp add: lemma-2-5-14)
by (simp add: mult.assoc)

lemma lemma-3-5-iv-2:  $(d2\ a\ b) * (d2\ b\ c) * (d2\ a\ b) \leq d2\ a\ c$ 
apply (simp add: d2-def)
apply (subgoal-tac (a r→ b) * (b r→ a) * ((b r→ c) * (c r→ b)) * ((a r→ b) * (b r→ a)) =
   $((a\ r\rightarrow\ b) * (b\ r\rightarrow\ a) * (b\ r\rightarrow\ c)) * ((c\ r\rightarrow\ b) * (a\ r\rightarrow\ b) * (b\ r\rightarrow\ a))$ )
apply simp
apply (rule mult-mono)
apply (rule-tac y = (a r→ b) * (b r→ c) in order-trans)
apply (rule mult-right-mono)
apply simp
apply (simp add: lemma-2-5-15)
apply (rule-tac y = (c r→ b) * (b r→ a) in order-trans)
apply (rule mult-right-mono)
apply simp
apply (simp add: lemma-2-5-15)
by (simp add: mult.assoc)

lemma lemma-3-5-iv-3:  $(d3\ a\ b) * (d3\ b\ c) * (d3\ a\ b) \leq d3\ a\ c$ 
by (simp add: d3-def lemma-3-5-iv-1)

lemma lemma-3-5-iv-4:  $(d4\ b\ c) * (d4\ a\ b) * (d4\ b\ c) \leq d4\ a\ c$ 
by (simp add: d4-def lemma-3-5-iv-2)

definition
  cong-r F a b ≡ d1 a b ∈ F

definition
  cong-l F a b ≡ d2 a b ∈ F

lemma cong-r-filter:  $F \in filters \implies (cong-r\ F\ a\ b) = (a\ l\rightarrow\ b \in F \wedge b\ l\rightarrow\ a \in F)$ 
apply (simp add: cong-r-def d1-def)
apply safe
apply (rule filter-ii)
apply simp-all
apply simp

```

```

apply (rule filter-ii)
apply simp-all
apply simp
by (simp add: filter-i)

lemma cong-r-symmetric:  $F \in \text{filters} \implies (\text{cong-r } F \ a \ b) = (\text{cong-r } F \ b \ a)$ 
apply (simp add: cong-r-filter)
by blast

lemma cong-r-d3:  $F \in \text{filters} \implies (\text{cong-r } F \ a \ b) = (d3 \ a \ b \in F)$ 
apply (simp add: d3-def)
apply (subst cong-r-symmetric)
by (simp-all add: cong-r-def)

lemma cong-l-filter:  $F \in \text{filters} \implies (\text{cong-l } F \ a \ b) = (a \ r \rightarrow b \in F \wedge b \ r \rightarrow a \in F)$ 
apply (simp add: cong-l-def d2-def)
apply safe
apply (rule filter-ii)
apply simp-all
apply simp
apply (rule filter-ii)
apply simp-all
apply simp
by (simp add: filter-i)

lemma cong-l-symmetric:  $F \in \text{filters} \implies (\text{cong-l } F \ a \ b) = (\text{cong-l } F \ b \ a)$ 
apply (simp add: cong-l-filter)
by blast

lemma cong-l-d4:  $F \in \text{filters} \implies (\text{cong-l } F \ a \ b) = (d4 \ a \ b \in F)$ 
apply (simp add: d4-def)
apply (subst cong-l-symmetric)
by (simp-all add: cong-l-def)

lemma cong-r-reflexive:  $F \in \text{filters} \implies \text{cong-r } F \ a \ a$ 
by (simp add: cong-r-def)

lemma cong-r-transitive:  $F \in \text{filters} \implies \text{cong-r } F \ a \ b \implies \text{cong-r } F \ b \ c \implies \text{cong-r } F \ a \ c$ 
apply (simp add: cong-r-filter)
apply safe
apply (rule-tac  $a = (b \ l \rightarrow c) * (a \ l \rightarrow b)$  in filter-ii)
apply simp-all
apply (rule filter-i)
apply simp-all
apply (simp add: lemma-2-5-14)
apply (rule-tac  $a = (b \ l \rightarrow a) * (c \ l \rightarrow b)$  in filter-ii)

```

**apply** *simp-all*  
**apply** (*rule filter-i*)  
**apply** *simp-all*  
**by** (*simp add: lemma-2-5-14*)

**lemma** *cong-l-reflexive*:  $F \in \text{filters} \implies \text{cong-l } F \ a \ a$   
**by** (*simp add: cong-l-def*)

**lemma** *cong-l-transitive*:  $F \in \text{filters} \implies \text{cong-l } F \ a \ b \implies \text{cong-l } F \ b \ c \implies \text{cong-l } F \ a \ c$

**apply** (*simp add: cong-l-filter*)  
**apply** *safe*  
**apply** (*rule-tac a = (a r→ b) \* (b r→ c) in filter-ii*)  
**apply** *simp-all*  
**apply** (*rule filter-i*)  
**apply** *simp-all*  
**apply** (*simp add: lemma-2-5-15*)  
**apply** (*rule-tac a = (c r→ b) \* (b r→ a) in filter-ii*)  
**apply** *simp-all*  
**apply** (*rule filter-i*)  
**apply** *simp-all*  
**by** (*simp add: lemma-2-5-15*)

**lemma** *lemma-3-7-i*:  $F \in \text{filters} \implies F = \{a . \text{cong-r } F \ a \ 1\}$   
**by** (*simp add: cong-r-def*)

**lemma** *lemma-3-7-ii*:  $F \in \text{filters} \implies F = \{a . \text{cong-l } F \ a \ 1\}$   
**by** (*simp add: cong-l-def*)

**lemma** *lemma-3-8-i*:  $F \in \text{filters} \implies (\text{cong-r } F \ a \ b) = (\exists \ x \ y . x \in F \wedge y \in F \wedge x * a = y * b)$

**apply** (*subst cong-r-filter*)  
**apply** *safe*  
**apply** (*rule-tac x = a l→ b in exI*)  
**apply** (*rule-tac x = b l→ a in exI*)  
**apply** (*simp add: left-impl-times*)  
**apply** (*subgoal-tac x ≤ a l→ b*)  
**apply** (*simp add: filter-ii*)  
**apply** (*simp add: left-residual [THEN sym]*)  
**apply** (*subgoal-tac y ≤ b l→ a*)  
**apply** (*simp add: filter-ii*)  
**apply** (*simp add: left-residual [THEN sym]*)  
**apply** (*subgoal-tac y \* b = x \* a*)  
**by** *simp-all*

**lemma** *lemma-3-8-ii*:  $F \in \text{filters} \implies (\text{cong-l } F \ a \ b) = (\exists \ x \ y . x \in F \wedge y \in F \wedge a * x = b * y)$

**apply** (*subst cong-l-filter*)  
**apply** *safe*  
**apply** (*rule-tac*  $x = a \ r \rightarrow b$  **in** *exI*)  
**apply** (*rule-tac*  $x = b \ r \rightarrow a$  **in** *exI*)  
**apply** (*simp add: right-impl-times*)  
**apply** (*subgoal-tac*  $x \leq a \ r \rightarrow b$ )  
**apply** (*simp add: filter-ii*)  
**apply** (*simp add: right-residual [THEN sym]*)  
**apply** (*subgoal-tac*  $y \leq b \ r \rightarrow a$ )  
**apply** (*simp add: filter-ii*)  
**apply** (*simp add: right-residual [THEN sym]*)  
**apply** (*subgoal-tac*  $b * y = a * x$ )  
**by** *simp-all*

**lemma** *lemma-3-9-i*:  $F \in \text{filters} \implies \text{cong-r } F \ a \ b \implies \text{cong-r } F \ c \ d \implies (a \ l \rightarrow c \in F) = (b \ l \rightarrow d \in F)$

**apply** (*simp add: cong-r-filter*)  
**apply** *safe*  
**apply** (*rule-tac*  $a = (a \ l \rightarrow d) * (b \ l \rightarrow a)$  **in** *filter-ii*)  
**apply** (*simp-all add: lemma-2-5-14*)  
**apply** (*rule-tac*  $a = ((c \ l \rightarrow d) * (a \ l \rightarrow c)) * (b \ l \rightarrow a)$  **in** *filter-ii*)  
**apply** *simp-all*  
**apply** (*simp add: filter-i*)  
**apply** (*rule mult-right-mono*)  
**apply** (*simp-all add: lemma-2-5-14*)

**apply** (*rule-tac*  $a = (b \ l \rightarrow c) * (a \ l \rightarrow b)$  **in** *filter-ii*)  
**apply** (*simp-all add: lemma-2-5-14*)  
**apply** (*rule-tac*  $a = ((d \ l \rightarrow c) * (b \ l \rightarrow d)) * (a \ l \rightarrow b)$  **in** *filter-ii*)  
**apply** *simp-all*  
**apply** (*simp add: filter-i*)  
**apply** (*rule mult-right-mono*)  
**by** (*simp-all add: lemma-2-5-14*)

**lemma** *lemma-3-9-ii*:  $F \in \text{filters} \implies \text{cong-l } F \ a \ b \implies \text{cong-l } F \ c \ d \implies (a \ r \rightarrow c \in F) = (b \ r \rightarrow d \in F)$

**apply** (*simp add: cong-l-filter*)  
**apply** *safe*  
**apply** (*rule-tac*  $a = (b \ r \rightarrow a) * (a \ r \rightarrow d)$  **in** *filter-ii*)  
**apply** (*simp-all add: lemma-2-5-15*)  
**apply** (*rule-tac*  $a = (b \ r \rightarrow a) * ((a \ r \rightarrow c) * (c \ r \rightarrow d))$  **in** *filter-ii*)  
**apply** *simp-all*  
**apply** (*simp add: filter-i*)  
**apply** (*rule mult-left-mono*)  
**apply** (*simp-all add: lemma-2-5-15*)

**apply** (*rule-tac*  $a = (a \ r \rightarrow b) * (b \ r \rightarrow c)$  **in** *filter-ii*)  
**apply** (*simp-all add: lemma-2-5-15*)  
**apply** (*rule-tac*  $a = (a \ r \rightarrow b) * ((b \ r \rightarrow d) * (d \ r \rightarrow c))$  **in** *filter-ii*)

**apply** *simp-all*  
**apply** (*simp add: filter-i*)  
**apply** (*rule mult-left-mono*)  
**by** (*simp-all add: lemma-2-5-15*)

**definition**

$normalfilters = \{F . F \in filters \wedge (\forall a b . (a l \rightarrow b \in F) = (a r \rightarrow b \in F))\}$

**lemma** *normalfilter-i*:

$H \in normalfilters \implies a l \rightarrow b \in H \implies a r \rightarrow b \in H$   
**by** (*simp add: normalfilters-def*)

**lemma** *normalfilter-ii*:

$H \in normalfilters \implies a r \rightarrow b \in H \implies a l \rightarrow b \in H$   
**by** (*simp add: normalfilters-def*)

**lemma** [*simp*]:  $H \in normalfilters \implies H \in filters$

**by** (*simp add: normalfilters-def*)

**lemma** *lemma-3-10-i-ii*:

$H \in normalfilters \implies \{a\} ** H = H ** \{a\}$

**apply** (*simp add: times-set-def*)

**apply** *safe*

**apply** *simp*

**apply** (*rule-tac x = a l → a \* y in bexI*)

**apply** (*simp add: inf-l-def [THEN sym]*)

**apply** (*rule order.antisym*)

**apply** *simp*

**apply** *simp*

**apply** (*rule normalfilter-ii*)

**apply** *simp-all*

**apply** (*rule-tac a = y in filter-ii*)

**apply** *simp-all*

**apply** (*simp add: right-residual [THEN sym]*)

**apply** (*rule-tac x = a r → xa \* a in bexI*)

**apply** (*simp add: inf-r-def [THEN sym]*)

**apply** (*rule order.antisym*)

**apply** *simp*

**apply** *simp*

**apply** (*rule normalfilter-i*)

**apply** *simp-all*

**apply** (*rule-tac a = xa in filter-ii*)

**apply** *simp-all*

**by** (*simp add: left-residual [THEN sym]*)

**lemma** *lemma-3-10-ii-iii*:

```

H ∈ filters ⇒ (∀ a . {a} ** H = H ** {a}) ⇒ cong-r H = cong-l H
apply (subst fun-eq-iff)
apply (subst fun-eq-iff)
apply safe
apply (subst (asm) lemma-3-8-i)
apply simp-all
apply safe
apply (subst lemma-3-8-ii)
apply simp-all
apply (subgoal-tac xb * x ∈ {x} ** H)
apply (subgoal-tac y * xa ∈ {xa} ** H)
apply (drule drop-assumption)
apply (drule drop-assumption)
apply (simp add: times-set-def)
apply safe
apply (rule-tac x = ya in exI)
apply simp
apply (rule-tac x = yb in exI)
apply simp
apply (drule-tac x = xa in spec)
apply (simp add: times-set-def)
apply auto[1]
apply (drule-tac x = x in spec)
apply simp
apply (simp add: times-set-def)
apply (rule-tac x = xb in bexI)
apply simp-all

apply (subst (asm) lemma-3-8-ii)
apply simp-all
apply safe
apply (subst lemma-3-8-i)
apply simp-all
apply (subgoal-tac x * xb ∈ H ** {x})
apply (subgoal-tac xa * y ∈ H ** {xa})
apply (drule drop-assumption)
apply (drule drop-assumption)
apply (simp add: times-set-def)
apply safe
apply (rule-tac x = xc in exI)
apply simp
apply (rule-tac x = xd in exI)
apply simp
apply (drule-tac x = xa in spec)
apply (simp add: times-set-def)
apply auto[1]
apply (drule-tac x = x in spec)
apply (subgoal-tac x * xb ∈ {x} ** H)
apply simp

```

**apply** (*subst times-set-def*)  
**by** *blast*

**lemma** *lemma-3-10-i-iii*:  
 $H \in \text{normalfilters} \implies \text{cong-r } H = \text{cong-l } H$   
**by** (*simp add: lemma-3-10-i-ii lemma-3-10-ii-iii*)

**lemma** *order-impl-l [simp]*:  $a \leq b \implies a \text{ l} \rightarrow b = 1$   
**by** (*simp add: left-lesseq*)

**end**

**context** *pseudo-hoop-algebra* **begin**

**lemma** *impl-l-d1*:  $(a \text{ l} \rightarrow b) = d1 \ a \ (a \sqcap b)$   
**by** (*simp add: d1-def lemma-2-6-20-a*)

**lemma** *impl-r-d2*:  $(a \text{ r} \rightarrow b) = d2 \ a \ (a \sqcap b)$   
**by** (*simp add: d2-def lemma-2-6-21-a*)

**lemma** *lemma-3-10-iii-i*:  
 $H \in \text{filters} \implies \text{cong-r } H = \text{cong-l } H \implies H \in \text{normalfilters}$   
**apply** (*unfold normalfilters-def*)  
**apply** (*simp add: impl-l-d1 impl-r-d2*)  
**apply** *safe*  
**apply** (*subgoal-tac cong-r H a (a \sqcap b)*)  
**apply** *simp*  
**apply** (*subst (asm) cong-l-def*)  
**apply** *simp*  
**apply** (*subst cong-r-def*)  
**apply** *simp*  
**apply** (*subgoal-tac cong-r H a (a \sqcap b)*)  
**apply** (*subst (asm) cong-r-def*)  
**apply** *simp*  
**apply** *simp*  
**apply** (*subst cong-l-def*)  
**by** *simp*

**lemma** *lemma-3-10-ii-i*:  
 $H \in \text{filters} \implies (\forall a . \{a\} ** H = H ** \{a\}) \implies H \in \text{normalfilters}$   
**apply** (*rule lemma-3-10-iii-i*)  
**apply** *simp*  
**apply** (*rule lemma-3-10-ii-iii*)  
**by** *simp-all*

**definition**

*allpowers*  $x \ n = \{y . \exists i . i < n \wedge y = x \wedge i\}$



**lemma** *times-set-in*:  $a \in A \implies b \in B \implies c = a * b \implies c \in A ** B$   
**apply** (*simp add: times-set-def*)  
**by** *auto*

**lemma** *power-set-power*:  $a \in A \implies a \wedge n \in A * \wedge n$   
**apply** (*induct-tac n*)  
**apply** *simp*  
**apply** *simp*  
**apply** (*rule-tac a = a and b = a \wedge n in times-set-in*)  
**by** *simp-all*

**lemma** *normal-filter-union*:  $H \in \text{normalfilters} \implies (H \cup \{x\}) * \wedge n = (H ** (\text{allpowers } x \ n)) \cup \{x \wedge n\}$   
**apply** (*induct-tac n*)  
**apply** (*simp add: times-set-def allpowers-def*)  
**apply** *safe*  
**apply** *simp*  
**apply** (*simp add: times-set-def*)  
**apply** *safe*  
**apply** (*simp add: allpowers-def*)  
**apply** *safe*  
**apply** (*subgoal-tac x \* xa \in H \*\* \{x\}*)  
**apply** (*simp add: times-set-def*)  
**apply** *safe*  
**apply** (*drule-tac x = xb in bspec*)  
**apply** *simp*  
**apply** (*drule-tac x = x \wedge (i + 1) in spec*)  
**apply** *simp*  
**apply** *safe*  
**apply** (*erule notE*)  
**apply** (*rule-tac x = i + 1 in exI*)  
**apply** *simp*  
**apply** (*erule notE*)  
**apply** (*simp add: mult.assoc [THEN sym]*)  
**apply** (*drule-tac a = x in lemma-3-10-i-ii*)  
**apply** (*subgoal-tac H \*\* \{x\} = \{x\} \*\* H*)  
**apply** *simp*  
**apply** (*simp add: times-set-def*)  
**apply** *auto[1]*  
**apply** *simp*  
**apply** (*drule-tac x = xaa in bspec*)  
**apply** *simp*  
**apply** (*drule-tac x = x \wedge n in bspec*)  
**apply** (*simp add: allpowers-def*)  
**apply** *blast*  
**apply** *simp*  
**apply** (*drule-tac x = xaa \* xb in bspec*)  
**apply** (*simp add: filter-i*)  
**apply** (*simp add: mult.assoc*)

```

apply (drule-tac  $x = ya$  in bspec)
apply (simp add: allpowers-def)
apply safe
apply (rule-tac  $x = i$  in exI)
apply simp
apply simp
apply (subst (asm) times-set-def)
apply (subst (asm) times-set-def)
apply simp
apply safe
apply (subst (asm) allpowers-def)
apply (subst (asm) allpowers-def)
apply safe
apply (case-tac  $i = 0$ )
apply simp
apply (rule-tac  $a = xa$  and  $b = 1$  in times-set-in)
apply blast
apply (simp add: allpowers-def times-set-def)
apply safe
apply simp
apply (drule-tac  $x = 1$  in bspec)
apply simp
apply (drule-tac  $x = 1$  in spec)
apply simp
apply (drule-tac  $x = 0$  in spec)
apply auto[1]
apply simp
apply (rule-tac  $a = xaa$  and  $b = x \wedge i$  in times-set-in)
apply blast
apply (case-tac  $i = n$ )
apply simp
apply (simp add: allpowers-def)
apply safe
apply (subgoal-tac  $x \wedge i \in H ** \{y . \exists i < n. y = x \wedge i\}$ )
apply simp
apply (rule-tac  $a = 1$  and  $b = x \wedge i$  in times-set-in)
apply simp
apply simp
apply (rule-tac  $x = i$  in exI)
apply simp
apply simp
apply (rule power-set-power)
by simp

```

**lemma** *lemma-3-11-i*:  $H \in \text{normalfilters} \implies \text{filterof } (H \cup \{x\}) = \{a . \exists n h . h \in H \wedge h * x \wedge n \leq a\}$   
**apply** (*subst prop-3-2-i*)  
**apply** (*subst normal-filter-union*)

```

apply simp-all
apply safe
apply (rule-tac  $x = n$  in exI)
apply (rule-tac  $x = 1$  in exI)
apply simp
apply (simp-all add: allpowers-def times-set-def)
apply safe
apply (rule-tac  $x = i$  in exI)
apply (rule-tac  $x = xb$  in exI)
apply simp
apply (rule-tac  $x = n + 1$  in exI)
apply (rule-tac  $x = h * x ^ n$  in exI)
apply safe
apply (erule notE)
apply (rule-tac  $x = h$  in beXI)
apply (rule-tac  $x = x ^ n$  in exI)
by auto

```

**lemma** *lemma-3-11-ii*:  $H \in \text{normalfilters} \implies \text{filterof } (H \cup \{x\}) = \{a . \exists n h . h \in H \wedge (x ^ n) * h \leq a\}$

```

apply (subst lemma-3-11-i)
apply simp-all
apply safe
apply (rule-tac  $x = n$  in exI)
apply (subgoal-tac  $h * x ^ n \in \{x ^ n\} ** H$ )
apply (simp add: times-set-def)
apply auto[1]
apply (drule-tac  $a = x ^ n$  in lemma-3-10-i-ii)
apply simp
apply (simp add: times-set-def)
apply auto[1]
apply (rule-tac  $x = n$  in exI)
apply (subgoal-tac  $(x ^ n) * h \in H ** \{x ^ n\}$ )
apply (simp add: times-set-def)
apply auto[1]
apply (drule-tac  $a = x ^ n$  in lemma-3-10-i-ii)
apply (drule-tac sym)
apply simp
apply (simp add: times-set-def)
by auto

```

**lemma** *lemma-3-12-i-ii*:

$H \in \text{normalfilters} \implies H \in \text{ultrafilters} \implies x \notin H \implies (\exists n . x ^ n l \rightarrow a \in H)$

```

apply (subst (asm) ultrafilter-union)
apply clarify
apply (drule-tac  $x = x$  in spec)
apply clarify
apply (subst (asm) lemma-3-11-i)
apply assumption

```

**apply** (*subgoal-tac*  $a \in \{a::'a. \exists (n::nat) h::'a. h \in H \wedge h * x \wedge n \leq a\}$ )  
**apply** *clarify*  
**apply** (*rule-tac*  $x = n$  **in** *exI*)  
**apply** (*simp add: left-residual*)  
**apply** (*rule filter-ii*)  
**by** *simp-all*

**lemma** *lemma-3-12-ii-i:*

$H \in \text{normalfilters} \implies H \in \text{properfilters} \implies (\forall x a . x \notin H \longrightarrow (\exists n . x \wedge n \text{ l}\rightarrow a \in H)) \implies H \in \text{maximalfilters}$   
**apply** (*subgoal-tac*  $H \in \text{ultrafilters}$ )  
**apply** (*simp add: ultrafilters-def*)  
**apply** (*subst ultrafilter-union*)  
**apply** *clarify*  
**apply** (*subst (asm) properfilters-def*)  
**apply** *clarify*  
**apply** (*subst lemma-3-11-i*)  
**apply** *simp-all*  
**apply** *safe*  
**apply** *simp-all*  
**apply** (*drule-tac*  $x = x$  **in** *spec*)  
**apply** *clarify*  
**apply** (*drule-tac*  $x = xb$  **in** *spec*)  
**apply** *clarify*  
**apply** (*rule-tac*  $x = n$  **in** *exI*)  
**apply** (*rule-tac*  $x = x \wedge n \text{ l}\rightarrow xb$  **in** *exI*)  
**apply** *clarify*  
**apply** (*subst inf-l-def [THEN sym]*)  
**by** *simp*

**lemma** *lemma-3-12-i-iii:*

$H \in \text{normalfilters} \implies H \in \text{ultrafilters} \implies x \notin H \implies (\exists n . x \wedge n \text{ r}\rightarrow a \in H)$   
**apply** (*subst (asm) ultrafilter-union*)  
**apply** *clarify*  
**apply** (*drule-tac*  $x = x$  **in** *spec*)  
**apply** *clarify*  
**apply** (*subst (asm) lemma-3-11-ii*)  
**apply** *assumption*  
**apply** (*subgoal-tac*  $a \in \{a::'a. \exists (n::nat) h::'a. h \in H \wedge (x \wedge n) * h \leq a\}$ )  
**apply** *clarify*  
**apply** (*rule-tac*  $x = n$  **in** *exI*)  
**apply** (*simp add: right-residual*)  
**apply** (*rule filter-ii*)  
**by** *simp-all*

**lemma** *lemma-3-12-iii-i:*

$H \in \text{normalfilters} \implies H \in \text{properfilters} \implies (\forall x a . x \notin H \longrightarrow (\exists n . x \wedge n$

$r \rightarrow a \in H) \implies H \in \text{maximalfilters}$   
**apply** (*subgoal-tac*  $H \in \text{ultrafilters}$ )  
**apply** (*simp add: ultrafilters-def*)  
**apply** (*subst ultrafilter-union*)  
**apply** *clarify*  
**apply** (*subst (asm) properfilters-def*)  
**apply** *clarify*  
**apply** (*subst lemma-3-11-ii*)  
**apply** *simp-all*  
**apply** *safe*  
**apply** *simp-all*  
**apply** (*drule-tac*  $x = x$  **in** *spec*)  
**apply** *clarify*  
**apply** (*drule-tac*  $x = xb$  **in** *spec*)  
**apply** *clarify*  
**apply** (*rule-tac*  $x = n$  **in** *exI*)  
**apply** (*rule-tac*  $x = x \wedge n r \rightarrow xb$  **in** *exI*)  
**apply** *clarify*  
**apply** (*subst inf-r-def* [*THEN sym*])  
**by** *simp*

**definition**

$\text{cong } H = (\lambda x y . \text{cong-l } H x y \wedge \text{cong-r } H x y)$

**definition**

$\text{congruences} = \{R . \text{equivp } R \wedge (\forall a b c d . R a b \wedge R c d \longrightarrow R (a * c) (b * d) \wedge R (a \mapsto c) (b \mapsto d) \wedge R (a \mapsto c) (b \mapsto d))\}$

**lemma** *cong-l*:  $H \in \text{normalfilters} \implies \text{cong } H = \text{cong-l } H$

**by** (*simp add: cong-def lemma-3-10-i-iii*)

**lemma** *cong-r*:  $H \in \text{normalfilters} \implies \text{cong } H = \text{cong-r } H$

**by** (*simp add: cong-def lemma-3-10-i-iii*)

**lemma** *cong-equiv*:  $H \in \text{normalfilters} \implies \text{equivp } (\text{cong } H)$

**apply** (*simp add: cong-l*)

**apply** (*simp add: equivp-reflp-symp-transp reflp-def refl-on-def cong-l-reflexive cong-l-symmetric symp-def sym-def transp-def trans-def*)

**apply** *safe*

**apply** (*rule cong-l-transitive*)

**by** *simp-all*

**lemma** *cong-trans*:  $H \in \text{normalfilters} \implies \text{cong } H x y \implies \text{cong } H y z \implies \text{cong } H x z$

**apply** (*drule cong-equiv*)

**apply** (*drule equivp-transp*)

**by** *simp-all*

**lemma** *lemma-3-13* [*simp*]:

$H \in \text{normalfilters} \implies \text{cong } H \in \text{congruences}$   
**apply** (*subst congruences-def*)  
**apply** *safe*  
**apply** (*simp add: cong-equiv*)  
**apply** (*rule-tac y = b \* c in cong-trans*)  
**apply** *simp-all*  
**apply** (*simp add: cong-r lemma-3-8-i*)  
**apply** *safe*  
**apply** (*rule-tac x = x in exI*)  
**apply** *simp*  
**apply** (*rule-tac x = y in exI*)  
**apply** (*simp add: mult.assoc [THEN sym]*)  
**apply** (*simp add: cong-l lemma-3-8-ii*)  
**apply** *safe*  
**apply** (*rule-tac x = xa in exI*)  
**apply** *simp*  
**apply** (*rule-tac x = ya in exI*)  
**apply** (*simp add: mult.assoc*)  
**apply** (*rule-tac y = a l $\rightarrow$  d in cong-trans*)  
**apply** *simp*  
**apply** (*simp add: cong-r cong-r-filter*)  
**apply** *safe*  
**apply** (*rule-tac a = c l $\rightarrow$  d in filter-ii*)  
**apply** *simp-all*  
**apply** (*subst left-residual [THEN sym]*)  
**apply** (*simp add: lemma-2-5-14*)  
**apply** (*rule-tac a = d l $\rightarrow$  c in filter-ii*)  
**apply** *simp-all*  
**apply** (*subst left-residual [THEN sym]*)  
**apply** (*simp add: lemma-2-5-14*)  
**apply** (*subst cong-l*)  
**apply** *simp*  
**apply** (*simp add: cong-r cong-r-filter cong-l-filter*)  
**apply** *safe*  
**apply** (*rule-tac a = b l $\rightarrow$  a in filter-ii*)  
**apply** *simp-all*  
**apply** (*subst right-residual [THEN sym]*)  
**apply** (*simp add: lemma-2-5-14*)  
**apply** (*rule-tac a = a l $\rightarrow$  b in filter-ii*)  
**apply** *simp-all*  
**apply** (*subst right-residual [THEN sym]*)  
**apply** (*simp add: lemma-2-5-14*)

**apply** (*rule-tac y = a r $\rightarrow$  d in cong-trans*)  
**apply** *simp*  
**apply** (*simp add: cong-l cong-l-filter*)  
**apply** *safe*  
**apply** (*rule-tac a = c r $\rightarrow$  d in filter-ii*)  
**apply** *simp-all*

**apply** (*subst right-residual* [*THEN sym*])  
**apply** (*simp add: lemma-2-5-15*)  
**apply** (*rule-tac a = d r → c in filter-ii*)  
**apply** *simp-all*  
**apply** (*subst right-residual* [*THEN sym*])  
**apply** (*simp add: lemma-2-5-15*)

**apply** (*subst cong-r*)  
**apply** *simp*  
**apply** (*simp add: cong-l cong-l-filter cong-r-filter*)  
**apply** *safe*  
**apply** (*rule-tac a = b r → a in filter-ii*)  
**apply** *simp-all*  
**apply** (*subst left-residual* [*THEN sym*])  
**apply** (*simp add: lemma-2-5-15*)  
**apply** (*rule-tac a = a r → b in filter-ii*)  
**apply** *simp-all*  
**apply** (*subst left-residual* [*THEN sym*])  
**by** (*simp add: lemma-2-5-15*)

**lemma** *cong-times*:  $R \in \text{congruences} \implies R a b \implies R c d \implies R (a * c) (b * d)$   
**by** (*simp add: congruences-def*)

**lemma** *cong-impl-l*:  $R \in \text{congruences} \implies R a b \implies R c d \implies R (a l \rightarrow c) (b l \rightarrow d)$   
**by** (*simp add: congruences-def*)

**lemma** *cong-impl-r*:  $R \in \text{congruences} \implies R a b \implies R c d \implies R (a r \rightarrow c) (b r \rightarrow d)$   
**by** (*simp add: congruences-def*)

**lemma** *cong-refl* [*simp*]:  $R \in \text{congruences} \implies R a a$   
**by** (*simp add: congruences-def equivp-reflp*)

**lemma** *cong-trans-a*:  $R \in \text{congruences} \implies R a b \implies R b c \implies R a c$   
**apply** (*simp add: congruences-def*)  
**apply** (*rule-tac y = b in equivp-transp*)  
**by** *simp-all*

**lemma** *cong-sym*:  $R \in \text{congruences} \implies R a b \implies R b a$   
**by** (*simp add: congruences-def equivp-symp*)

#### definition

*normalfilter*  $R = \{a . R a 1\}$

**lemma** *lemma-3-14* [*simp*]:  
 $R \in \text{congruences} \implies (\text{normalfilter } R) \in \text{normalfilters}$   
**apply** (*unfold normalfilters-def*)  
**apply** *safe*

```

apply (simp add: filters-def)
apply safe
apply (simp add: normalfilter-def)
apply (drule-tac x = 1 in spec)
apply (simp add: congruences-def equivp-reflp)
apply (simp add: normalfilter-def)
apply (drule-tac a = a and c = b and b = 1 and d = 1 and R = R in
cong-times)
apply simp-all
apply (simp add: normalfilter-def)
apply (simp add: left-lesseq)
apply (cut-tac R = R and a = a and b = 1 and c = b and d = b in cong-impl-l)
apply simp-all
apply (simp add: cong-sym)
apply (simp-all add: normalfilter-def)
apply (cut-tac R = R and a = a l→ b and b = 1 and c = a and d = a in
cong-times)
apply simp-all
apply (simp add: inf-l-def [THEN sym])
apply (cut-tac R = R and a = a and b = a  $\sqcap$  b and c = b and d = b in
cong-impl-r)
apply simp-all
apply (simp add: cong-sym)
apply (cut-tac R = R and c = a r→ b and d = 1 and a = a and b = a in
cong-times)
apply simp-all
apply (simp add: inf-r-def [THEN sym])
apply (cut-tac R = R and a = a and b = a  $\sqcap$  b and c = b and d = b in
cong-impl-l)
apply simp-all
by (simp add: cong-sym)

```

**lemma** lemma-3-15-i:

```

H ∈ normalfilters ⇒ normalfilter (cong H) = H
by (simp add: normalfilter-def cong-r cong-r-filter)

```

**lemma** lemma-3-15-ii:

```

R ∈ congruences ⇒ cong (normalfilter R) = R
apply (simp add: fun-eq-iff cong-r cong-r-filter)
apply (simp add: normalfilter-def)
apply safe
apply (cut-tac R = R and a = x l→ xa and b = 1 and c = x and d = x in
cong-times)
apply simp-all
apply (cut-tac R = R and a = xa l→ x and b = 1 and c = xa and d = xa in
cong-times)
apply simp-all
apply (simp add: inf-l-def [THEN sym])
apply (rule-tac b = x  $\sqcap$  xa in cong-trans-a)

```



**apply** *simp-all*  
**apply** (*subst cong-sym*)  
**apply** *simp-all*  
**apply** (*subst inf.commute*)  
**apply** *simp-all*  
**apply** (*cut-tac R = R and a = x and b = xa and c = xa and d = xa in cong-impl-l*)  
**apply** *simp-all*  
**apply** (*cut-tac R = R and a = xa and b = xa and c = x and d = xa in cong-impl-l*)  
**by** *simp-all*

**lemma** *lemma-3-15-iii: H1 ∈ normalfilters ⇒ H2 ∈ normalfilters ⇒ (H1 ⊆ H2) = (cong H1 ≤ cong H2)*  
**apply** *safe*  
**apply** (*simp add: cong-l cong-l-filter*)  
**apply** *blast*  
**apply** (*subgoal-tac cong H2 x 1*)  
**apply** (*simp add: cong-l cong-l-def*)  
**apply** (*subgoal-tac cong H1 x 1*)  
**apply** *blast*  
**by** (*simp add: cong-l cong-l-def*)

**definition**

$p\ x\ y\ z = ((x\ l\rightarrow\ y)\ r\rightarrow\ z) \sqcap ((z\ l\rightarrow\ y)\ r\rightarrow\ x)$

**lemma** *lemma-3-16-i: p x x y = y ∧ p x y y = x*

**apply** *safe*  
**apply** (*simp-all add: p-def*)  
**apply** (*rule order.antisym*)  
**apply** (*simp-all add: lemma-2-10-24*)  
**apply** (*rule order.antisym*)  
**by** (*simp-all add: lemma-2-10-24*)

**definition**  $M\ x\ y\ z = ((y\ l\rightarrow\ x)\ r\rightarrow\ x) \sqcap ((z\ l\rightarrow\ y)\ r\rightarrow\ y) \sqcap ((x\ l\rightarrow\ z)\ r\rightarrow\ z)$

**lemma**  $M\ x\ x\ y = x \wedge M\ x\ y\ x = x \wedge M\ y\ x\ x = x$

**apply** (*simp add: M-def*)  
**apply** *safe*  
**apply** (*rule order.antisym*)  
**apply** (*simp-all add: lemma-2-10-24 lemma-2-5-9-b*)  
**apply** (*rule order.antisym*)  
**apply** (*simp-all add: lemma-2-10-24 lemma-2-5-9-b*)  
**apply** (*rule order.antisym*)  
**by** (*simp-all add: lemma-2-10-24 lemma-2-5-9-b*)

**end**

**end**

## 7 Pseudo Waisberg Algebra

```

theory PseudoWaisbergAlgebra
imports Operations
begin

class impl-lr-algebra = one + left-imp + right-imp +
  assumes W1a [simp]:  $1 \text{ l} \rightarrow a = a$ 
  and W1b [simp]:  $1 \text{ r} \rightarrow a = a$ 

  and W2a:  $(a \text{ l} \rightarrow b) \text{ r} \rightarrow b = (b \text{ l} \rightarrow a) \text{ r} \rightarrow a$ 
  and W2b:  $(b \text{ l} \rightarrow a) \text{ r} \rightarrow a = (b \text{ r} \rightarrow a) \text{ l} \rightarrow a$ 
  and W2c:  $(b \text{ r} \rightarrow a) \text{ l} \rightarrow a = (a \text{ r} \rightarrow b) \text{ l} \rightarrow b$ 

  and W3a:  $(a \text{ l} \rightarrow b) \text{ l} \rightarrow ((b \text{ l} \rightarrow c) \text{ r} \rightarrow (a \text{ l} \rightarrow c)) = 1$ 
  and W3b:  $(a \text{ r} \rightarrow b) \text{ r} \rightarrow ((b \text{ r} \rightarrow c) \text{ l} \rightarrow (a \text{ r} \rightarrow c)) = 1$ 

begin

lemma P1-a [simp]:  $x \text{ l} \rightarrow x = 1$ 
  apply (cut-tac  $a = 1$  and  $b = 1$  and  $c = x$  in W3b)
  by simp

lemma P1-b [simp]:  $x \text{ r} \rightarrow x = 1$ 
  apply (cut-tac  $a = 1$  and  $b = 1$  and  $c = x$  in W3a)
  by simp

lemma P2-a:  $x \text{ l} \rightarrow y = 1 \implies y \text{ l} \rightarrow x = 1 \implies x = y$ 
  apply (subgoal-tac  $(y \text{ l} \rightarrow x) \text{ r} \rightarrow x = y$ )
  apply simp
  apply (subgoal-tac  $(x \text{ l} \rightarrow y) \text{ r} \rightarrow y = y$ )
  apply (unfold W2a)
  by simp-all

lemma P2-b:  $x \text{ r} \rightarrow y = 1 \implies y \text{ r} \rightarrow x = 1 \implies x = y$ 
  apply (subgoal-tac  $(y \text{ r} \rightarrow x) \text{ l} \rightarrow x = y$ )
  apply simp
  apply (subgoal-tac  $(x \text{ r} \rightarrow y) \text{ l} \rightarrow y = y$ )
  apply (unfold W2c)
  by simp-all

lemma P2-c:  $x \text{ l} \rightarrow y = 1 \implies y \text{ r} \rightarrow x = 1 \implies x = y$ 
  apply (subgoal-tac  $(y \text{ r} \rightarrow x) \text{ l} \rightarrow x = y$ )
  apply simp
  apply (subgoal-tac  $(x \text{ l} \rightarrow y) \text{ r} \rightarrow y = y$ )
  apply (unfold W2b) [1]
  apply (unfold W2c) [1]
  by simp-all

```

**lemma P3-a:**  $(x \text{ l}\rightarrow 1) \text{ r}\rightarrow 1 = 1$   
**apply** (*unfold W2a*)  
**by** *simp*

**lemma P3-b:**  $(x \text{ r}\rightarrow 1) \text{ l}\rightarrow 1 = 1$   
**apply** (*unfold W2c*)  
**by** *simp*

**lemma P4-a** [*simp*]:  $x \text{ l}\rightarrow 1 = 1$   
**apply** (*subgoal-tac x l\rightarrow ((x l\rightarrow 1) r\rightarrow 1) = 1*)  
**apply** (*simp add: P3-a*)  
**apply** (*cut-tac a = 1 and b = x and c = 1 in W3a*)  
**by** *simp*

**lemma P4-b** [*simp*]:  $x \text{ r}\rightarrow 1 = 1$   
**apply** (*subgoal-tac x r\rightarrow ((x r\rightarrow 1) l\rightarrow 1) = 1*)  
**apply** (*simp add: P3-b*)  
**apply** (*cut-tac a = 1 and b = x and c = 1 in W3b*)  
**by** *simp*

**lemma P5-a:**  $x \text{ l}\rightarrow y = 1 \implies y \text{ l}\rightarrow z = 1 \implies x \text{ l}\rightarrow z = 1$   
**apply** (*cut-tac a = x and b = y and c = z in W3a*)  
**by** *simp*

**lemma P5-b:**  $x \text{ r}\rightarrow y = 1 \implies y \text{ r}\rightarrow z = 1 \implies x \text{ r}\rightarrow z = 1$   
**apply** (*cut-tac a = x and b = y and c = z in W3b*)  
**by** *simp*

**lemma P6-a:**  $x \text{ l}\rightarrow (y \text{ r}\rightarrow x) = 1$   
**apply** (*cut-tac a = y and b = 1 and c = x in W3b*)  
**by** *simp*

**lemma P6-b:**  $x \text{ r}\rightarrow (y \text{ l}\rightarrow x) = 1$   
**apply** (*cut-tac a = y and b = 1 and c = x in W3a*)  
**by** *simp*

**lemma P7:**  $(x \text{ l}\rightarrow (y \text{ r}\rightarrow z) = 1) = (y \text{ r}\rightarrow (x \text{ l}\rightarrow z) = 1)$   
**proof**

**fix**  $x\ y\ z$  **assume**  $A: x \text{ l}\rightarrow y \text{ r}\rightarrow z = 1$  **show**  $y \text{ r}\rightarrow x \text{ l}\rightarrow z = 1$

**apply** (*rule-tac y = (z r\rightarrow y) l\rightarrow y in P5-b*)

**apply** (*simp add: P6-b*)

**apply** (*unfold W2c*)

**apply** (*subgoal-tac (x l\rightarrow (y r\rightarrow z)) l\rightarrow (((y r\rightarrow z) l\rightarrow z) r\rightarrow x l\rightarrow z) = 1*)

**apply** (*unfold A*) [1]

**apply** *simp*

**by** (*simp add: W3a*)

**next**

**fix**  $x\ y\ z$  **assume**  $A: y \text{ r}\rightarrow x \text{ l}\rightarrow z = 1$  **show**  $x \text{ l}\rightarrow y \text{ r}\rightarrow z = 1$

**apply** (*rule-tac y = (z l\rightarrow x) r\rightarrow x in P5-a*)

**apply** (*simp add: P6-a*)  
**apply** (*unfold W2a*)  
**apply** (*subgoal-tac (y r→ x l→ z) r→ (((x l→ z) r→ z) l→ y r→ z) = 1*)  
**apply** (*unfold A [1]*)  
**apply** *simp*  
**by** (*simp add: W3b*)  
**qed**

**lemma P8-a:**  $(x l\rightarrow y) r\rightarrow ((z l\rightarrow x) l\rightarrow (z l\rightarrow y)) = 1$   
**by** (*simp add: W3a P7 [THEN sym]*)

**lemma P8-b:**  $(x r\rightarrow y) l\rightarrow ((z r\rightarrow x) r\rightarrow (z r\rightarrow y)) = 1$   
**by** (*simp add: W3b P7*)

**lemma P9:**  $x l\rightarrow (y r\rightarrow z) = y r\rightarrow (x l\rightarrow z)$   
**apply** (*rule P2-c*)  
**apply** (*subst P7*)  
**apply** (*rule-tac y = (z r→ y) l→ y in P5-b*)  
**apply** (*simp add: P6-b*)  
**apply** (*subst W2c*)  
**apply** (*rule P8-a*)  
**apply** (*subst P7 [THEN sym]*)  
**apply** (*rule-tac y = (z l→ x) r→ x in P5-a*)  
**apply** (*simp add: P6-a*)  
**apply** (*subst W2a*)  
**by** (*simp add: P8-b*)

**definition**  
 $lesseq-a\ a\ b = (a\ l\rightarrow\ b = 1)$

**definition**  
 $less-a\ a\ b = (lesseq-a\ a\ b \wedge \neg lesseq-a\ b\ a)$

**definition**  
 $lesseq-b\ a\ b = (a\ r\rightarrow\ b = 1)$

**definition**  
 $less-b\ a\ b = (lesseq-b\ a\ b \wedge \neg lesseq-b\ b\ a)$

**definition**  
 $sup-a\ a\ b = (a\ l\rightarrow\ b) r\rightarrow\ b$

**end**

**sublocale** *impl-lr-algebra* < *order-a:order lesseq-a less-a*  
**apply** *unfold-locales*  
**apply** (*simp add: less-a-def*)  
**apply** (*simp-all add: lesseq-a-def*)

```

apply (rule P5-a)
apply simp-all
apply (rule P2-a)
by simp-all

sublocale impl-lr-algebra < order-b:order lesseq-b less-b
apply unfold-locales
apply (simp add: less-b-def)
apply (simp-all add: lesseq-b-def)
apply (rule P5-b)
apply simp-all
apply (rule P2-b)
by simp-all

sublocale impl-lr-algebra < slattice-a:semilattice-sup sup-a lesseq-a less-a
apply unfold-locales
apply (simp-all add: lesseq-a-def sup-a-def)
apply (simp add: P9)
apply (simp add: P9)
apply (subst W2a)
apply (subgoal-tac ((z l→ y) r→ y) l→ ((y l→ x) r→ x) = 1)
apply simp
apply (subgoal-tac ((z l→ y) r→ y) l→ ((x l→ y) r→ y) = 1)
apply (simp add: W2a)
apply (subgoal-tac ((z l→ y) r→ y) l→ (x l→ y) r→ y = ((x l→ y) r→ (z l→
y)) r→ (((z l→ y) r→ y) l→ (x l→ y) r→ y))
apply (simp add: W3b)
apply (subgoal-tac (x l→ y) r→ z l→ y = 1)
apply simp
apply (cut-tac a = z and b = x and c = y in W3a)
by simp

sublocale impl-lr-algebra < slattice-b:semilattice-sup sup-a lesseq-b less-b
apply unfold-locales
apply (simp-all add: lesseq-b-def sup-a-def)
apply (simp-all add: W2b)
apply (simp add: P9 [THEN sym])
apply (simp add: P9 [THEN sym])
apply (subst W2c)
apply (subgoal-tac ((z r→ y) l→ y) r→ ((y r→ x) l→ x) = 1)
apply simp
apply (subgoal-tac ((z r→ y) l→ y) r→ ((x r→ y) l→ y) = 1)
apply (simp add: W2c)
apply (subgoal-tac ((z r→ y) l→ y) r→ (x r→ y) l→ y = ((x r→ y) l→ (z r→
y)) l→ (((z r→ y) l→ y) r→ (x r→ y) l→ y))
apply (simp add: W3a)
apply (subgoal-tac (x r→ y) l→ z r→ y = 1)
apply simp
apply (cut-tac a = z and b = x and c = y in W3b)

```

```

by simp

context impl-lr-algebra
begin
lemma lesseq-a-b: lesseq-b = lesseq-a
  apply (simp add: fun-eq-iff)
  apply clarify
  apply (cut-tac x = x and y = xa in slattice-a.le-iff-sup)
  apply (cut-tac x = x and y = xa in slattice-b.le-iff-sup)
  by simp

lemma P10: (a l→ b = 1) = (a r→ b = 1)
  apply (cut-tac lesseq-a-b)
  by (simp add: fun-eq-iff lesseq-a-def lesseq-b-def)
end

class one-ord = one + ord

class impl-lr-ord-algebra = impl-lr-algebra + one-ord +
  assumes
    order: a ≤ b = (a l→ b = 1)
  and
    strict: a < b = (a ≤ b ∧ ¬ b ≤ a)
begin
lemma order-l: (a ≤ b) = (a l→ b = 1)
  by (simp add: order)

lemma order-r: (a ≤ b) = (a r→ b = 1)
  by (simp add: order P10)

lemma P11-a: a ≤ b l→ a
  by (simp add: order-r P6-b)

lemma P11-b: a ≤ b r→ a
  by (simp add: order-l P6-a)

lemma P12: (a ≤ b l→ c) = (b ≤ a r→ c)
  apply (subst order-r)
  apply (subst order-l)
  by (simp add: P7)

lemma P13-a: a ≤ b ⇒ b l→ c ≤ a l→ c
  apply (subst order-r)
  apply (simp add: order-l)
  apply (cut-tac a = a and b = b and c = c in W3a)
  by simp

lemma P13-b: a ≤ b ⇒ b r→ c ≤ a r→ c

```

```

apply (subst order-l)
apply (simp add: order-r)
apply (cut-tac a = a and b = b and c = c in W3b)
by simp

lemma P14-a: a ≤ b ⇒ c l→ a ≤ c l→ b
apply (simp add: order-l)
apply (cut-tac x = a and y = b and z = c in P8-a)
by simp

lemma P14-b: a ≤ b ⇒ c r→ a ≤ c r→ b
apply (simp add: order-r)
apply (cut-tac x = a and y = b and z = c in P8-b)
by simp

subclass order
apply (subgoal-tac (≤) = lesseq-a ∧ (<) = less-a)
apply simp
apply unfold-locales
apply safe
by (simp-all add: fun-eq-iff lesseq-a-def less-a-def order-l strict)

end

class one-zero-uminus = one + zero + left-uminus + right-uminus

class impl-neg-lr-algebra = impl-lr-ord-algebra + one-zero-uminus +
assumes
  W4: -l 1 = -r 1
and W5a: (-l a r→ -l b) l→ (b l→ a) = 1
and W5b: (-r a l→ -r b) l→ (b r→ a) = 1
and zero-def: 0 = -l 1
begin

lemma zero-r-def: 0 = -r 1
by (simp add: zero-def W4)

lemma C1-a [simp]: (-l x r→ 0) l→ x = 1
apply (unfold zero-def)
apply (cut-tac a = x and b = 1 in W5a)
by simp

lemma C1-b [simp]: (-r x l→ 0) r→ x = 1
apply (unfold zero-r-def)
apply (cut-tac a = x and b = 1 in W5b)
by (simp add: P10)

lemma C2-b [simp]: 0 r→ x = 1
apply (cut-tac x = -r x l→ 0 and y = x and z = 0 in P8-b)

```

by (*simp add: P6-b*)

**lemma** *C2-a* [*simp*]:  $0 \text{ l} \rightarrow x = 1$   
by (*simp add: P10*)

**lemma** *C3-a*:  $x \text{ l} \rightarrow 0 = -l \ x$

**proof** –

have *A*:  $-l \ x \text{ l} \rightarrow (x \text{ l} \rightarrow 0) = 1$

apply (*cut-tac x = -l x and y = -l (-r 1) in P6-a*)

apply (*cut-tac a = -r 1 and b = x in W5a*)

apply (*unfold zero-r-def*)

apply (*rule-tac y = -l (-r (1::'a)) r \rightarrow -l x in P5-a*)

by *simp-all*

have *B*:  $(x \text{ l} \rightarrow 0) \text{ r} \rightarrow -l \ x = 1$

apply (*cut-tac a = -l x r \rightarrow 0 and b = x and c = 0 in W3a*)

apply *simp*

apply (*cut-tac b = -l x and a = 0 in W2c*)

by *simp*

show  $x \text{ l} \rightarrow 0 = -l \ x$

apply (*rule order.antisym*)

apply (*simp add: order-r B*)

by (*simp add: order-l A*)

**qed**

**lemma** *C3-b*:  $x \text{ r} \rightarrow 0 = -r \ x$

apply (*rule order.antisym*)

apply (*simp add: order-l*)

apply (*cut-tac x = x in C1-b*)

apply (*cut-tac a = -r x l \rightarrow 0 and b = x and c = 0 in W3b*)

apply *simp*

apply (*cut-tac b = -r x and a = 0 in W2a*)

apply *simp*

apply (*cut-tac x = -r x and y = -r (-l 1) in P6-b*)

apply (*cut-tac a = -l 1 and b = x in W5b*)

apply (*unfold zero-def order-r*)

apply (*rule-tac y = -r (-l (1::'a)) l \rightarrow -r x in P5-b*)

by (*simp-all add: P10*)

**lemma** *C4-a* [*simp*]:  $-r \ (-l \ x) = x$

apply (*unfold C3-b [THEN sym] C3-a [THEN sym]*)

apply (*subst W2a*)

by *simp*

**lemma** *C4-b* [*simp*]:  $-l \ (-r \ x) = x$

apply (*unfold C3-b [THEN sym] C3-a [THEN sym]*)

apply (*subst W2c*)

by *simp*

**lemma** *C5-a*:  $-r \ x \text{ l} \rightarrow -r \ y = y \text{ r} \rightarrow x$



```

apply (rule order.antisym)
apply (simp add: order-l W5b)
apply (cut-tac a = -r y and b = -r x in W5a)
by (simp add: order-l)

lemma C5-b:  $-l x r \rightarrow -l y = y l \rightarrow x$ 
apply (rule order.antisym)
apply (simp add: order-l W5a)
apply (cut-tac a = -l y and b = -l x in W5b)
by (simp add: order-l)

lemma C6:  $-r x l \rightarrow y = -l y r \rightarrow x$ 
apply (cut-tac x = x and y = -l y in C5-a)
by simp

lemma C7-a:  $(x \leq y) = (-l y \leq -l x)$ 
apply (subst order-l)
apply (subst order-r)
by (simp add: C5-b)

lemma C7-b:  $(x \leq y) = (-r y \leq -r x)$ 
apply (subst order-r)
apply (subst order-l)
by (simp add: C5-a)

end

class pseudo-wajsberg-algebra = impl-neg-lr-algebra +
assumes
  W6:  $-r (a l \rightarrow -l b) = -l (b r \rightarrow -r a)$ 
begin
definition
  mult a b = -r (a l  $\rightarrow$  -l b)

definition
  inf-a a b = -l (a r  $\rightarrow$  -r (a l  $\rightarrow$  b))

definition
  inf-b a b = -r (b l  $\rightarrow$  -l (b r  $\rightarrow$  a))

end

sublocale pseudo-wajsberg-algebra < slattice-inf-a:semilattice-inf inf-a ( $\leq$ ) ( $<$ )
apply unfold-locales
apply (simp-all add: inf-a-def)
apply (subst C7-b)
apply (simp add: order-l P9 C5-a P10 [THEN sym] P6-a)
apply (subst C7-b)

```

```

apply (simp add: order-l P9 C5-a P10 [THEN sym] P6-a)
apply (subst W6 [THEN sym])
apply (subst C7-a)
apply simp
proof -
  fix x y z
  assume A:  $x \leq y$ 
  assume B:  $x \leq z$ 
  have C:  $x \rightarrow y = 1$  by (simp add: order-l [THEN sym] A)
  have E:  $((y \rightarrow z) \rightarrow -l y) \rightarrow -l x = ((y \rightarrow z) \rightarrow -l y) \rightarrow ((x \rightarrow y) \rightarrow -l x)$ 
  by (simp add: C)
  have F:  $((y \rightarrow z) \rightarrow -l y) \rightarrow ((x \rightarrow y) \rightarrow -l x) = ((y \rightarrow z) \rightarrow -l y) \rightarrow ((-l y \rightarrow -l x) \rightarrow -l x)$ 
  by (simp add: C5-b)
  have G:  $((y \rightarrow z) \rightarrow -l y) \rightarrow ((-l y \rightarrow -l x) \rightarrow -l x) = ((y \rightarrow z) \rightarrow -l y) \rightarrow ((-l x \rightarrow -l y) \rightarrow -l y)$ 
  by (simp add: W2c)
  have H:  $((y \rightarrow z) \rightarrow -l y) \rightarrow ((-l x \rightarrow -l y) \rightarrow -l y) = ((y \rightarrow z) \rightarrow -l y) \rightarrow ((y \rightarrow x) \rightarrow -l y)$ 
  by (simp add: C5-b)
  have I:  $((y \rightarrow z) \rightarrow -l y) \rightarrow ((y \rightarrow x) \rightarrow -l y) = 1$ 
  apply (simp add: order-l [THEN sym] P14-a)
  apply (rule P13-a)
  apply (rule P14-a)
  by (simp add: B)
  show  $(y \rightarrow z) \rightarrow -l y \leq -l x$ 
  by (simp add: order-l E F G H I)
next
qed

```

```

sublocale pseudo-wajsberg-algebra < slattice-inf-b:semilattice-inf inf-b ( $\leq$ ) ( $<$ )
apply unfold-locales
apply (simp-all add: inf-b-def)
apply (subst C7-a)
apply (simp add: order-r P9 [THEN sym] C5-b P10 P6-b)
apply (subst C7-a)
apply (simp add: order-r P9 [THEN sym] C5-b P10 P6-b)
apply (subst W6)
apply (subst C7-b)
apply simp
proof -
  fix x y z
  assume A:  $x \leq y$ 
  assume B:  $x \leq z$ 
  have C:  $x \rightarrow z = 1$  by (simp add: order-r [THEN sym] B)
  have E:  $((z \rightarrow y) \rightarrow -r z) \rightarrow -r x = ((z \rightarrow y) \rightarrow -r z) \rightarrow ((x \rightarrow z) \rightarrow -r x)$ 

```

```

    by (simp add: C)
  have F:  $((z \ r \rightarrow y) \ r \rightarrow -r \ z) \ r \rightarrow ((x \ r \rightarrow z) \ r \rightarrow -r \ x) = ((z \ r \rightarrow y) \ r \rightarrow -r \ z) \ r \rightarrow ((-r \ z \ l \rightarrow -r \ x) \ r \rightarrow -r \ x)$ 
    by (simp add: C5-a)
  have G:  $((z \ r \rightarrow y) \ r \rightarrow -r \ z) \ r \rightarrow ((-r \ z \ l \rightarrow -r \ x) \ r \rightarrow -r \ x) = ((z \ r \rightarrow y) \ r \rightarrow -r \ z) \ r \rightarrow ((-r \ x \ l \rightarrow -r \ z) \ r \rightarrow -r \ z)$ 
    by (simp add: W2a)
  have H:  $((z \ r \rightarrow y) \ r \rightarrow -r \ z) \ r \rightarrow ((-r \ x \ l \rightarrow -r \ z) \ r \rightarrow -r \ z) = ((z \ r \rightarrow y) \ r \rightarrow -r \ z) \ r \rightarrow ((z \ r \rightarrow x) \ r \rightarrow -r \ z)$ 
    by (simp add: C5-a)
  have I:  $((z \ r \rightarrow y) \ r \rightarrow -r \ z) \ r \rightarrow ((z \ r \rightarrow x) \ r \rightarrow -r \ z) = 1$ 
    apply (simp add: order-r [THEN sym])
    apply (rule P13-b)
    apply (rule P14-b)
    by (simp add: A)
  show  $(z \ r \rightarrow y) \ r \rightarrow -r \ z \leq -r \ x$ 
    by (simp add: order-r E F G H I)
next
qed

```

```

context pseudo-wajsberg-algebra
begin
lemma inf-a-b:  $inf\ a = inf\ b$ 
  apply (simp add: fun-eq-iff)
  apply clarify
  apply (rule order.antisym)
  by simp-all

```

```

end
end

```

## 8 Some Classes of Pseudo-Hoops

```

theory SpecialPseudoHoops
imports PseudoHoopFilters PseudoWaisbergAlgebra
begin

class cancel-pseudo-hoop-algebra = pseudo-hoop-algebra +
  assumes mult-cancel-left:  $a * b = a * c \implies b = c$ 
  and mult-cancel-right:  $b * a = c * a \implies b = c$ 
begin
lemma cancel-left-a:  $b \ l \rightarrow (a * b) = a$ 
  apply (rule-tac  $a = b$  in mult-cancel-right)
  apply (subst inf-l-def [THEN sym])
  apply (rule order.antisym)
  by simp-all

```

```

lemma cancel-right-a:  $b \rightarrow (b * a) = a$ 
  apply (rule-tac  $a = b$  in mult-cancel-left)
  apply (subst inf-r-def [THEN sym])
  apply (rule order.antisym)
  by simp-all

end

class cancel-pseudo-hoop-algebra-2 = pseudo-hoop-algebra +
  assumes cancel-left:  $b \rightarrow (a * b) = a$ 
  and cancel-right:  $b \rightarrow (b * a) = a$ 

begin
subclass cancel-pseudo-hoop-algebra
  apply unfold-locales
  apply (subgoal-tac  $b = a \rightarrow (a * b) \wedge a \rightarrow (a * b) = a \rightarrow (a * c) \wedge a \rightarrow (a * c) = c$ )
  apply simp
  apply (rule conjI)
  apply (subst cancel-right)
  apply simp
  apply (rule conjI)
  apply simp
  apply (subst cancel-right)
  apply simp
  apply (subgoal-tac  $b = a \rightarrow (b * a) \wedge a \rightarrow (b * a) = a \rightarrow (c * a) \wedge a \rightarrow (c * a) = c$ )
  apply simp
  apply (rule conjI)
  apply (subst cancel-left)
  apply simp
  apply (rule conjI)
  apply simp
  apply (subst cancel-left)
  by simp

end

context cancel-pseudo-hoop-algebra
begin

lemma lemma-4-2-i:  $a \rightarrow b = (a * c) \rightarrow (b * c)$ 
  apply (subgoal-tac  $a \rightarrow b = a \rightarrow (c \rightarrow (b * c)) \wedge a \rightarrow (c \rightarrow (b * c)) = (a * c) \rightarrow (b * c)$ )
  apply simp
  apply (rule conjI)
  apply (simp add: cancel-left-a)
  by (simp add: left-impl-ded)

```

```

lemma lemma-4-2-ii:  $a \rightarrow b = (c * a) \rightarrow (c * b)$ 
  apply (subgoal-tac  $a \rightarrow b = a \rightarrow (c \rightarrow (c * b)) \wedge a \rightarrow (c \rightarrow (c * b)) = (c$ 
 $* a) \rightarrow (c * b)$ )
  apply simp
  apply (rule conjI)
  apply (simp add: cancel-right-a)
  by (simp add: right-impl-ded)

```

```

lemma lemma-4-2-iii:  $(a * c \leq b * c) = (a \leq b)$ 
  by (simp add: left-lesseq lemma-4-2-i [THEN sym])

```

```

lemma lemma-4-2-iv:  $(c * a \leq c * b) = (a \leq b)$ 
  by (simp add: right-lesseq lemma-4-2-ii [THEN sym])

```

**end**

```

class wajsberg-pseudo-hoop-algebra = pseudo-hoop-algebra +
  assumes wajsberg1:  $(a \rightarrow b) \rightarrow b = (b \rightarrow a) \rightarrow a$ 
  and wajsberg2:  $(a \rightarrow b) \rightarrow b = (b \rightarrow a) \rightarrow a$ 

```

```

context wajsberg-pseudo-hoop-algebra
begin

```

```

lemma lemma-4-3-i-a:  $a \sqcup 1 b = (a \rightarrow b) \rightarrow b$ 
  by (simp add: sup1-def wajsberg1)

```

```

lemma lemma-4-3-i-b:  $a \sqcup 1 b = (b \rightarrow a) \rightarrow a$ 
  by (simp add: sup1-def wajsberg1)

```

```

lemma lemma-4-3-ii-a:  $a \sqcup 2 b = (a \rightarrow b) \rightarrow b$ 
  by (simp add: sup2-def wajsberg2)

```

```

lemma lemma-4-3-ii-b:  $a \sqcup 2 b = (b \rightarrow a) \rightarrow a$ 
  by (simp add: sup2-def wajsberg2)

```

**end**

```

sublocale wajsberg-pseudo-hoop-algebra < lattice1:pseudo-hoop-lattice-b ( $\sqcup 1$ ) (*)
( $\sqcap$ ) ( $\rightarrow$ ) ( $\leq$ ) ( $<$ ) 1 ( $\rightarrow$ )
  apply unfold-locales
  apply (simp add: lemma-4-3-i-a)
  by (simp add: lemma-2-5-13-b lemma-2-5-13-a)

```

```

class zero-one = zero + one

```

```

class bounded-wajsberg-pseudo-hoop-algebra = zero-one + wajsberg-pseudo-hoop-algebra

```

```

+
  assumes zero-smallest [simp]:  $0 \leq a$ 
begin
end

sublocale wajsberg-pseudo-hoop-algebra < lattice2:pseudo-hoop-lattice-b ( $\sqcup 2$ ) (*)
( $\sqcap$ ) ( $l \rightarrow$ ) ( $\leq$ ) ( $<$ ) 1 ( $r \rightarrow$ )
  apply unfold-locales
  apply (simp add: lemma-4-3-ii-a)
  by (simp add: lemma-2-5-13-b lemma-2-5-13-a)

lemma (in wajsberg-pseudo-hoop-algebra) sup1-eq-sup2: ( $\sqcup 1$ ) = ( $\sqcup 2$ )
  apply (simp add: fun-eq-iff)
  apply safe
  apply (cut-tac a = x and b = xa in lattice1.supremum-pair)
  apply (cut-tac a = x and b = xa in lattice2.supremum-pair)
  by blast

context bounded-wajsberg-pseudo-hoop-algebra
begin
definition
  negl a = a  $l \rightarrow$  0

definition
  negr a = a  $r \rightarrow$  0

lemma [simp]:  $0 l \rightarrow a = 1$ 
  by (simp add: order [THEN sym])

lemma [simp]:  $0 r \rightarrow a = 1$ 
  by (simp add: order-r [THEN sym])
end

sublocale bounded-wajsberg-pseudo-hoop-algebra < wajsberg: pseudo-wajsberg-algebra
1 ( $l \rightarrow$ ) ( $r \rightarrow$ ) ( $\leq$ ) ( $<$ ) 0 negl negr
  apply unfold-locales
  apply simp-all
  apply (simp add: lemma-4-3-i-a [THEN sym])
  apply (rule order.antisym)
  apply simp-all
  apply (simp add: lemma-4-3-i-a [THEN sym] lemma-4-3-ii-a [THEN sym])
  apply (rule order.antisym)
  apply simp-all
  apply (simp add: lemma-4-3-i-a [THEN sym] lemma-4-3-ii-a [THEN sym])
  apply (rule order.antisym)
  apply simp-all
  apply (subst left-lesseq [THEN sym])
  apply (simp add: lemma-2-5-16)

```

```

apply (subst right-lesseq [THEN sym])
apply (simp add: lemma-2-5-17)
apply (simp add: left-lesseq)
apply (simp add: less-def)
apply (simp-all add: negl-def negr-def)
apply (subst left-lesseq [THEN sym])
apply (subgoal-tac b  $l \rightarrow a = ((b \rightarrow 0) \rightarrow 0) \rightarrow ((a \rightarrow 0) \rightarrow 0)$ )
apply (simp add: lemma-2-5-17)
apply (subst wajsberg1)
apply simp
apply (subst wajsberg1)
apply simp
apply (subst left-lesseq [THEN sym])
apply (subgoal-tac b  $r \rightarrow a = ((b \rightarrow 0) \rightarrow 0) \rightarrow ((a \rightarrow 0) \rightarrow 0)$ )
apply (simp add: lemma-2-5-16)
apply (subst wajsberg2)
apply simp
apply (subst wajsberg2)
apply simp
apply (simp add: left-impl-ded [THEN sym])
apply (simp add: right-impl-ded [THEN sym])
apply (simp add: lemma-4-3-i-a [THEN sym] lemma-4-3-ii-a [THEN sym])
apply (rule order.antisym)
by simp-all

```

**context** pseudo-wajsberg-algebra

**begin**

```

lemma class.bounded-wajsberg-pseudo-hoop-algebra mult inf-a ( $l \rightarrow$ ) ( $\leq$ ) ( $<$ ) 1
( $r \rightarrow$ ) ( $0 :: 'a$ )
apply unfold-locales
apply (simp add: inf-a-def mult-def W6)
apply (simp add: strict)
apply (simp-all add: mult-def order-l strict)
apply (simp add: zero-def [THEN sym] C3-a)
apply (simp add: W6 inf-a-def [THEN sym])
apply (rule order.antisym)
apply simp-all
apply (simp add: C6 P9 [THEN sym] C5-b)
apply (simp add: inf-b-def [THEN sym])
apply (rule order.antisym)
apply simp-all
apply (simp add: inf-b-def [THEN sym])
apply (rule order.antisym)
apply simp-all
apply (simp add: W6)
apply (simp add: C6 [THEN sym])
apply (simp add: P9 C5-a)
apply (simp add: inf-b-def [THEN sym])

```

```

apply (simp add: W6 inf-a-def [THEN sym])
apply (rule order.antisym)
apply simp-all
apply (simp add: W2a)
by (simp add: W2c)

end

class basic-pseudo-hoop-algebra = pseudo-hoop-algebra +
  assumes B1: (a l→ b) l→ c ≤ ((b l→ a) l→ c) l→ c
  and B2: (a r→ b) r→ c ≤ ((b r→ a) r→ c) r→ c
begin
lemma lemma-4-5-i: (a l→ b) ⊔1 (b l→ a) = 1
  apply (cut-tac a = a and b = b and c = (a l→ b) ⊔1 (b l→ a) in B1)
  apply (subgoal-tac (a l→ b) l→ (a l→ b) ⊔1 (b l→ a) = 1 ∧ ((b l→ a) l→ (a
l→ b) ⊔1 (b l→ a)) = 1)
  apply (erule conjE)
  apply simp
  apply (rule order.antisym)
  apply simp
  apply simp
  apply safe
  apply (subst left-lesseq [THEN sym])
  apply simp
  apply (subst left-lesseq [THEN sym])
  by simp

lemma lemma-4-5-ii: (a r→ b) ⊔2 (b r→ a) = 1
  apply (cut-tac a = a and b = b and c = (a r→ b) ⊔2 (b r→ a) in B2)
  apply (subgoal-tac (a r→ b) r→ (a r→ b) ⊔2 (b r→ a) = 1 ∧ ((b r→ a) r→
(a r→ b) ⊔2 (b r→ a)) = 1)
  apply (erule conjE)
  apply simp
  apply (rule order.antisym)
  apply simp
  apply simp
  apply safe
  apply (subst right-lesseq [THEN sym])
  apply simp
  apply (subst right-lesseq [THEN sym])
  by simp

lemma lemma-4-5-iii: a l→ b = (a ⊔1 b) l→ b
  apply (rule order.antisym)
  apply (rule-tac y = ((a l→ b) r→ b) l→ b in order-trans)
  apply (rule lemma-2-10-26)
  apply (rule lemma-2-5-13-a)
  apply (simp add: sup1-def)

```



```

apply (rule lemma-2-5-13-a)
by simp

lemma lemma-4-5-iv:  $a r \rightarrow b = (a \sqcup 2 b) r \rightarrow b$ 
apply (rule order.antisym)
apply (rule-tac  $y = ((a r \rightarrow b) l \rightarrow b) r \rightarrow b$  in order-trans)
apply (rule lemma-2-10-24)
apply (rule lemma-2-5-13-b)
apply (simp add: sup2-def)
apply (rule lemma-2-5-13-b)
by simp

lemma lemma-4-5-v:  $(a \sqcup 1 b) l \rightarrow c = (a l \rightarrow c) \sqcap (b l \rightarrow c)$ 
apply (rule order.antisym)
apply simp
apply safe
apply (rule lemma-2-5-13-a)
apply simp
apply (rule lemma-2-5-13-a)
apply simp
apply (subst right-lesseq)
apply (rule order.antisym)
apply simp
apply (rule-tac  $y = (a l \rightarrow b) l \rightarrow ((a l \rightarrow c) \sqcap (b l \rightarrow c) r \rightarrow a \sqcup 1 b l \rightarrow c)$  in
order-trans)
apply (subst left-residual [THEN sym])
apply simp
apply (subst lemma-4-5-iii)
apply (subst right-residual [THEN sym])
apply (subst left-residual [THEN sym])
apply (rule-tac  $y = b \sqcap c$  in order-trans)
apply (subst (2) inf-l-def)
apply (rule-tac  $y = ((a l \rightarrow c) \sqcap (b l \rightarrow c)) * ((a \sqcup 1 b) \sqcap b)$  in order-trans)
apply (subst (3) inf-l-def)
apply (simp add: mult.assoc)
apply (subgoal-tac  $(a \sqcup 1 b \sqcap b) = b$ )
apply simp
apply (rule order.antisym, simp)
apply simp
apply simp
apply (rule-tac  $y = ((b l \rightarrow a) l \rightarrow ((a l \rightarrow c) \sqcap (b l \rightarrow c) r \rightarrow a \sqcup 1 b l \rightarrow c)) l \rightarrow$ 
 $((a l \rightarrow c) \sqcap (b l \rightarrow c) r \rightarrow a \sqcup 1 b l \rightarrow c)$  in order-trans)
apply (rule B1)
apply (subgoal-tac  $(b l \rightarrow a) l \rightarrow ((a l \rightarrow c) \sqcap (b l \rightarrow c) r \rightarrow a \sqcup 1 b l \rightarrow c) = 1$ )
apply simp
apply (rule order.antisym)
apply simp
apply (subst left-residual [THEN sym])

```

```

apply simp
apply (subst lemma-4-5-iii)
apply (subst right-residual [THEN sym])
apply (subst left-residual [THEN sym])
apply (rule-tac y = a  $\sqcap$  c in order-trans)
apply (subst (2) inf-l-def)
apply (rule-tac y = ((a  $l \rightarrow$  c)  $\sqcap$  (b  $l \rightarrow$  c)) * ((a  $\sqcup 1$  b)  $\sqcap$  a) in order-trans)
apply (subst (3) inf-l-def)
apply (subst sup1.sup-comute1)
apply (simp add: mult.assoc)
apply (subgoal-tac (a  $\sqcup 1$  b  $\sqcap$  a) = a)
apply simp
apply (rule order.antisym, simp)
apply simp
by simp

```

```

lemma lemma-4-5-vi: (a  $\sqcup 2$  b)  $r \rightarrow$  c = (a  $r \rightarrow$  c)  $\sqcap$  (b  $r \rightarrow$  c)
apply (rule order.antisym)
apply simp
apply safe
apply (rule lemma-2-5-13-b)
apply simp
apply (rule lemma-2-5-13-b)
apply simp
apply (subst left-lesseq)
apply (rule order.antisym)
apply simp
apply (rule-tac y = (a  $r \rightarrow$  b)  $r \rightarrow$  ((a  $r \rightarrow$  c)  $\sqcap$  (b  $r \rightarrow$  c)  $l \rightarrow$  a  $\sqcup 2$  b  $r \rightarrow$  c) in
order-trans)
apply (subst right-residual [THEN sym])
apply simp
apply (subst lemma-4-5-iv)
apply (subst left-residual [THEN sym])
apply (subst right-residual [THEN sym])
apply (rule-tac y = b  $\sqcap$  c in order-trans)
apply (subst (2) inf-r-def)
apply (rule-tac y = ((a  $\sqcup 2$  b)  $\sqcap$  b) * ((a  $r \rightarrow$  c)  $\sqcap$  (b  $r \rightarrow$  c)) in order-trans)
apply (subst (2) inf-r-def)
apply (simp add: mult.assoc)
apply (subgoal-tac (a  $\sqcup 2$  b  $\sqcap$  b) = b)
apply simp
apply (rule order.antisym, simp)
apply simp
apply simp
apply (rule-tac y = ((b  $r \rightarrow$  a)  $r \rightarrow$  ((a  $r \rightarrow$  c)  $\sqcap$  (b  $r \rightarrow$  c)  $l \rightarrow$  a  $\sqcup 2$  b  $r \rightarrow$  c))  $r \rightarrow$ 
((a  $r \rightarrow$  c)  $\sqcap$  (b  $r \rightarrow$  c)  $l \rightarrow$  a  $\sqcup 2$  b  $r \rightarrow$  c) in order-trans)
apply (rule B2)
apply (subgoal-tac (b  $r \rightarrow$  a)  $r \rightarrow$  ((a  $r \rightarrow$  c)  $\sqcap$  (b  $r \rightarrow$  c)  $l \rightarrow$  a  $\sqcup 2$  b  $r \rightarrow$  c) = 1)

```

```

apply simp
apply (rule order.antisym)
apply simp
apply (subst right-residual [THEN sym])
apply simp
apply (subst lemma-4-5-iv)
apply (subst left-residual [THEN sym])
apply (subst right-residual [THEN sym])
apply (rule-tac y = a  $\sqcap$  c in order-trans)
apply (subst (2) inf-r-def)
apply (rule-tac y = ((a  $\sqcup$ 2 b)  $\sqcap$  a) * ((a  $r \rightarrow$  c)  $\sqcap$  (b  $r \rightarrow$  c)) in order-trans)
apply (subst (2) inf-r-def)
apply (subst (2) sup2.sup-comute)
apply (simp add: mult.assoc)
apply (subgoal-tac (a  $\sqcup$ 2 b  $\sqcap$  a) = a)
apply simp
apply (rule order.antisym, simp)
apply simp
by simp

lemma lemma-4-5-a: a  $\leq$  c  $\implies$  b  $\leq$  c  $\implies$  a  $\sqcup$ 1 b  $\leq$  c
apply (subst left-lesseq)
apply (subst lemma-4-5-v)
by simp

lemma lemma-4-5-b: a  $\leq$  c  $\implies$  b  $\leq$  c  $\implies$  a  $\sqcup$ 2 b  $\leq$  c
apply (subst right-lesseq)
apply (subst lemma-4-5-vi)
by simp

lemma lemma-4-5: a  $\sqcup$ 1 b = a  $\sqcup$ 2 b
apply (rule order.antisym)
by (simp-all add: lemma-4-5-a lemma-4-5-b)
end

sublocale basic-pseudo-hoop-algebra < basic-lattice:lattice ( $\sqcap$ ) ( $\leq$ ) ( $<$ ) ( $\sqcup$ 1)
apply unfold-locales
by (simp-all add: lemma-4-5-a)

context pseudo-hoop-lattice begin end

sublocale basic-pseudo-hoop-algebra < pseudo-hoop-lattice ( $\sqcup$ 1) ( $*$ ) ( $\sqcap$ ) ( $l \rightarrow$ ) ( $\leq$ )
( $<$ ) 1 ( $r \rightarrow$ )
apply unfold-locales
by (simp-all add: basic-lattice.sup-assoc)

class sup-assoc-pseudo-hoop-algebra = pseudo-hoop-algebra +
assumes sup1-assoc: a  $\sqcup$ 1 (b  $\sqcup$ 1 c) = (a  $\sqcup$ 1 b)  $\sqcup$ 1 c
and sup2-assoc: a  $\sqcup$ 2 (b  $\sqcup$ 2 c) = (a  $\sqcup$ 2 b)  $\sqcup$ 2 c

```

**sublocale** *sup-assoc-pseudo-hoop-algebra* < *sup1-lattice: pseudo-hoop-lattice* ( $\sqcup 1$ )  
 (\*) ( $\sqcap$ ) ( $l \rightarrow$ ) ( $\leq$ ) ( $<$ )  $1$  ( $r \rightarrow$ )  
**apply** *unfold-locales*  
**by** (*simp add: sup1-assoc*)

**sublocale** *sup-assoc-pseudo-hoop-algebra* < *sup2-lattice: pseudo-hoop-lattice* ( $\sqcup 2$ )  
 (\*) ( $\sqcap$ ) ( $l \rightarrow$ ) ( $\leq$ ) ( $<$ )  $1$  ( $r \rightarrow$ )  
**apply** *unfold-locales*  
**by** (*simp add: sup2-assoc*)

**class** *sup-assoc-pseudo-hoop-algebra-1* = *sup-assoc-pseudo-hoop-algebra* +  
**assumes** *union-impl*:  $(a \rightarrow b) \sqcup 1 (b \rightarrow a) = 1$   
**and** *union-impr*:  $(a \rightarrow b) \sqcup 1 (b \rightarrow a) = 1$

**lemma** (**in** *pseudo-hoop-algebra*) [*simp*]: *infimum*  $\{a, b\} = \{a \sqcap b\}$   
**apply** (*simp add: infimum-def lower-bound-def*)  
**apply** *safe*  
**apply** (*rule order.antisym*)  
**by** *simp-all*

**lemma** (**in** *pseudo-hoop-lattice*) *sup-impl-inf*:  
 $(a \sqcup b) \rightarrow c = (a \rightarrow c) \sqcap (b \rightarrow c)$   
**apply** (*cut-tac A = {a, b} and a = a  $\sqcup$  b and b = c in lemma-2-8-i*)  
**by** *simp-all*

**lemma** (**in** *pseudo-hoop-lattice*) *sup-impr-inf*:  
 $(a \sqcup b) \rightarrow c = (a \rightarrow c) \sqcap (b \rightarrow c)$   
**apply** (*cut-tac A = {a, b} and a = a  $\sqcup$  b and b = c in lemma-2-8-i1*)  
**by** *simp-all*

**lemma** (**in** *pseudo-hoop-lattice*) *sup-times*:  
 $a * (b \sqcup c) = (a * b) \sqcup (a * c)$   
**apply** (*cut-tac A = {b, c} and b = b  $\sqcup$  c and a = a in lemma-2-9-i*)  
**by** *simp-all*

**lemma** (**in** *pseudo-hoop-lattice*) *sup-times-right*:  
 $(b \sqcup c) * a = (b * a) \sqcup (c * a)$   
**apply** (*cut-tac A = {b, c} and b = b  $\sqcup$  c and a = a in lemma-2-9-i1*)  
**by** *simp-all*

**context** *basic-pseudo-hoop-algebra* **begin end**

**sublocale** *sup-assoc-pseudo-hoop-algebra-1* < *basic-1: basic-pseudo-hoop-algebra*  
 (\*) ( $\sqcap$ ) ( $l \rightarrow$ ) ( $\leq$ ) ( $<$ )  $1$  ( $r \rightarrow$ )  
**apply** *unfold-locales*  
**apply** (*subst left-residual [THEN sym]*)  
**apply** (*rule-tac y = (a  $\rightarrow$  b)  $\sqcup$  1 (b  $\rightarrow$  a)  $l \rightarrow$  c in order-trans*)

```

apply (subst sup1-lattice.sup-impl-inf)
apply (simp add: lemma-2-5-11)
apply (simp add: union-impl)
  apply (subst right-residual [THEN sym])
apply (rule-tac y = (b r→ a) ⊔1 (a r→ b) r→ c in order-trans)
apply (subst sup1-lattice.sup-impr-inf)
apply (simp add: lemma-2-5-11)
by (simp add: union-impr)

```

```

context basic-pseudo-hoop-algebra
begin

```

```

lemma lemma-4-8-i: a * (b ⊔ c) = (a * b) ⊔ (a * c)
  apply (rule order.antisym)
  apply simp
  apply (subgoal-tac a * (b ⊔ c) = (a * (b * (b r→ c))) ⊔1 (a * (c * (c r→ b))))
  apply simp
  apply (drule drop-assumption)
  apply (rule-tac y = (((a * b) ⊔ (a * c)) * (b r→ c)) ⊔1 (((a * b) ⊔ (a * c)) *
(c r→ b)) in order-trans)
  apply (subst sup-times [THEN sym])
  apply (simp add: lemma-4-5 lemma-4-5-ii)
  apply (simp add: mult.assoc [THEN sym])
  apply safe
  apply (rule-tac y = a * b * (b r→ c) in order-trans)
  apply simp
  apply simp
  apply (rule-tac y = a * c * (c r→ b) in order-trans)
  apply simp
  apply simp
  apply (simp add: inf-r-def [THEN sym])
  apply (subgoal-tac b ⊔ c = c ⊔ b)
  apply simp
  apply (rule order.antisym)
by simp-all

```

```

lemma lemma-4-8-ii: (b ⊔ c) * a = (b * a) ⊔ (c * a)
  apply (rule order.antisym)
  apply simp
  apply (subgoal-tac (b ⊔ c) * a = (((b l→ c) * b) * a) ⊔1 (((c l→ b) * c) * a))
  apply simp
  apply (drule drop-assumption)
  apply (rule-tac y = ((b l→ c) * ((b * a) ⊔ (c * a))) ⊔1 ((c l→ b) * ((b * a) ⊔
(c * a))) in order-trans)
  apply (subst sup-times-right [THEN sym])
  apply (simp add: lemma-4-5-i)
  apply (simp add: mult.assoc)
  apply safe

```

```

apply (rule-tac  $y = (b \rightarrow c) * (b * a)$  in order-trans)
apply simp-all
apply (rule-tac  $y = (c \rightarrow b) * (c * a)$  in order-trans)
apply simp-all
apply (simp add: inf-l-def [THEN sym])
apply (subgoal-tac  $b \sqcap c = c \sqcap b$ )
apply simp
apply (rule order.antisym)
by simp-all

lemma lemma-4-8-iii:  $(a \rightarrow b) \rightarrow (b \rightarrow a) = b \rightarrow a$ 
apply (rule order.antisym)
apply (cut-tac  $a = a$  and  $b = b$  in lemma-4-5-i)
apply (unfold sup1-def right-lesseq, simp)
by (simp add: lemma-2-5-9-a)

lemma lemma-4-8-iv:  $(a \rightarrow b) \rightarrow (b \rightarrow a) = b \rightarrow a$ 
apply (rule order.antisym)
apply (cut-tac  $a = a$  and  $b = b$  in lemma-4-5-ii)
apply (unfold sup2-def left-lesseq, simp)
by (simp add: lemma-2-5-9-b)

end

context wajsberg-pseudo-hoop-algebra
begin
subclass sup-assoc-pseudo-hoop-algebra-1
apply unfold-locales
apply (simp add: lattice1.sup-assoc)
apply (simp add: lattice2.sup-assoc)
apply (simp add: lemma-4-3-i-a)
apply (subgoal-tac  $(a \rightarrow b) \rightarrow (b \rightarrow a) = b \rightarrow a$ )
apply simp
apply (subst lemma-2-10-30 [THEN sym])
apply (subst wajsberg1)
apply (simp add: lemma-2-10-32)
apply (subst sup1-eq-sup2)
apply (simp add: lemma-4-3-ii-a)
apply (subgoal-tac  $(a \rightarrow b) \rightarrow (b \rightarrow a) = b \rightarrow a$ )
apply simp
apply (subst lemma-2-10-31 [THEN sym])
apply (subst wajsberg2)
by (simp add: lemma-2-10-33)
end

class bounded-basic-pseudo-hoop-algebra = zero-one + basic-pseudo-hoop-algebra
+
assumes zero-smallest [simp]:  $0 \leq a$ 

```

```

class inf-a =
  fixes inf-a :: 'a => 'a => 'a (infixl  $\sqcap$  65)

class pseudo-bl-algebra = zero + sup + inf + monoid-mult + ord + left-imp +
right-imp +
  assumes bounded-lattice: class.bounded-lattice inf ( $\leq$ ) ( $<$ ) sup 0 1
  and left-residual-bl:  $(x * a \leq b) = (x \leq a \text{ l}\rightarrow b)$ 
  and right-residual-bl:  $(a * x \leq b) = (x \leq a \text{ r}\rightarrow b)$ 
  and inf-l-bl-def:  $a \sqcap b = (a \text{ l}\rightarrow b) * a$ 
  and inf-r-bl-def:  $a \sqcap b = a * (a \text{ r}\rightarrow b)$ 
  and impl-sup-bl:  $(a \text{ l}\rightarrow b) \sqcup (b \text{ l}\rightarrow a) = 1$ 
  and impr-sup-bl:  $(a \text{ r}\rightarrow b) \sqcup (b \text{ r}\rightarrow a) = 1$ 

sublocale bounded-basic-pseudo-hoop-algebra < basic: pseudo-bl-algebra 1 (*) 0
( $\sqcap$ ) ( $\sqcup$ ) ( $\text{l}\rightarrow$ ) ( $\text{r}\rightarrow$ ) ( $\leq$ ) ( $<$ )
  apply unfold-locales
  apply (rule zero-smallest)
  apply (rule left-residual)
  apply (rule right-residual)
  apply (rule inf-l-def)
  apply (simp add: inf-r-def [THEN sym])
  apply (rule lemma-4-5-i)
  apply (simp add: lemma-4-5)
  by (rule lemma-4-5-ii)

sublocale pseudo-bl-algebra < bounded-lattice: bounded-lattice inf ( $\leq$ ) ( $<$ ) sup 0
1
  by (rule bounded-lattice)

context pseudo-bl-algebra
begin
  lemma impl-one-bl [simp]:  $a \text{ l}\rightarrow a = 1$ 
  apply (rule bounded-lattice.order.antisym)
  apply simp-all
  apply (subst left-residual-bl [THEN sym])
  by simp

  lemma impr-one-bl [simp]:  $a \text{ r}\rightarrow a = 1$ 
  apply (rule bounded-lattice.order.antisym)
  apply simp-all
  apply (subst right-residual-bl [THEN sym])
  by simp

  lemma impl-ded-bl:  $((a * b) \text{ l}\rightarrow c) = (a \text{ l}\rightarrow (b \text{ l}\rightarrow c))$ 
  apply (rule bounded-lattice.order.antisym)
  apply (case-tac  $(a * b \text{ l}\rightarrow c \leq a \text{ l}\rightarrow b \text{ l}\rightarrow c) = ((a * b \text{ l}\rightarrow c) * a \leq b \text{ l}\rightarrow c)$ 
 $\wedge ((a * b \text{ l}\rightarrow c) * a \leq b \text{ l}\rightarrow c) = (((a * b \text{ l}\rightarrow c) * a) * b \leq c)$ 
 $\wedge (((a * b \text{ l}\rightarrow c) * a) * b \leq c) = ((a * b \text{ l}\rightarrow c) * (a * b) \leq c)$ 
 $\wedge ((a * b \text{ l}\rightarrow c) * (a * b) \leq c) = ((a * b \text{ l}\rightarrow c) \leq (a * b \text{ l}\rightarrow c))$ )

```

```

apply simp
apply (erule notE)
apply (rule conjI)
apply (simp add: left-residual-bl)
apply (rule conjI)
apply (simp add: left-residual-bl)
apply (rule conjI)
apply (simp add: mult.assoc)
apply (simp add: left-residual-bl)
apply (simp add: left-residual-bl [THEN sym])
apply (rule-tac y=(b l→ c) * b in bounded-lattice.order-trans)
apply (simp add: mult.assoc [THEN sym])
apply (subst inf-l-bl-def [THEN sym])
apply (subst bounded-lattice.inf-commute)
apply (subst inf-l-bl-def)
apply (subst mult.assoc)
apply (subst left-residual-bl)
apply simp
apply (subst left-residual-bl)
by simp

```

**lemma** *impr-ded-bl*:  $(b * a \rightarrow c) = (a \rightarrow (b \rightarrow c))$

```

apply (rule bounded-lattice.order.antisym)
apply (case-tac (b * a r→ c ≤ a r→ b r→ c) = (a * (b * a r→ c) ≤ b r→ c)
   $\wedge (a * (b * a \rightarrow c) \leq b \rightarrow c) = (b * (a * (b * a \rightarrow c)) \leq c)$ 
   $\wedge (b * (a * (b * a \rightarrow c)) \leq c) = ((b * a) * (b * a \rightarrow c) \leq c)$ 
   $\wedge ((b * a) * (b * a \rightarrow c) \leq c) = ((b * a \rightarrow c) \leq (b * a \rightarrow c))$ )
apply simp
apply (erule notE)
apply (rule conjI)
apply (simp add: right-residual-bl)
apply (rule conjI)
apply (simp add: right-residual-bl)
apply (rule conjI)
apply (simp add: mult.assoc)
apply (simp add: right-residual-bl)
apply (simp add: right-residual-bl [THEN sym])
apply (rule-tac y=b * (b r→ c) in bounded-lattice.order-trans)
apply (simp add: mult.assoc)
apply (subst inf-r-bl-def [THEN sym])
apply (subst bounded-lattice.inf-commute)
apply (subst inf-r-bl-def)
apply (subst mult.assoc [THEN sym])
apply (subst right-residual-bl)
apply simp
apply (subst right-residual-bl)
by simp

```

**lemma** *lesseq-impl-bl*:  $(a \leq b) = (a \rightarrow b = 1)$



```

apply (rule iffI)
apply (rule bounded-lattice.order.antisym)
apply simp
apply (simp add: left-residual-bl [THEN sym])
apply (subgoal-tac  $1 \leq a \rightarrow b$ )
apply (subst (asm) left-residual-bl [THEN sym])
by simp-all

```

**end**

```

context pseudo-bl-algebra
begin
subclass pseudo-hoop-lattice
  apply unfold-locales
  apply (rule inf-l-bl-def)
  apply (simp add: bounded-lattice.less-le-not-le)
  apply (simp add: mult-1-left)
  apply (simp add: mult-1-right)
  apply (simp add: impl-one-bl)
  apply (simp add: inf-l-bl-def [THEN sym])
  apply (rule bounded-lattice.inf-commute)
  apply (rule impl-ded-bl)
  apply (rule lesseq-impl-bl)
  apply (rule inf-r-bl-def)
  apply (simp add: impr-one-bl)
  apply (simp add: inf-r-bl-def [THEN sym])
  apply (rule bounded-lattice.inf-commute)
  apply (rule impr-ded-bl)
  apply (simp add: inf-r-bl-def [THEN sym] inf-l-bl-def [THEN sym])
  apply (rule bounded-lattice.sup-commute)
  apply simp
  apply safe
  apply (rule bounded-lattice.order.antisym)
  apply simp-all
  apply (subgoal-tac  $a \leq a \sqcup b$ )
  apply simp
  apply (drule drop-assumption)
  apply simp
  by (simp add: bounded-lattice.sup-assoc)

subclass bounded-basic-pseudo-hoop-algebra
  apply unfold-locales
  apply simp-all
  apply (simp add: left-residual [THEN sym])
  apply (rule-tac  $y = ((a \rightarrow b) \sqcup (b \rightarrow a)) \rightarrow c$  in bounded-lattice.order-trans)
  apply (simp add: sup-impl-inf)
  apply (simp add: impl-sup-bl)

```

```

apply (simp add: right-residual [THEN sym])
apply (rule-tac y = ((a r→ b) ⊔ (b r→ a)) r→ c in bounded-lattice.order-trans)
apply (simp add: sup-impr-inf)
by (simp add: impr-sup-bl)

end

class product-pseudo-hoop-algebra = basic-pseudo-hoop-algebra +
  assumes P1: b l→ b * b ≤ (a ⊔ (a l→ b)) l→ b
  and P2: b r→ b * b ≤ (a ⊔ (a r→ b)) r→ b
  and P3: ((a l→ b) l→ b) * (c * a l→ d * a) * (c * b l→ d * b) ≤ c l→ d
  and P4: ((a r→ b) r→ b) * (a * c r→ a * d) * (b * c r→ b * d) ≤ c r→ d

class cancel-basic-pseudo-hoop-algebra = basic-pseudo-hoop-algebra + cancel-pseudo-hoop-algebra
begin
subclass product-pseudo-hoop-algebra
  apply unfold-locales
  apply (rule-tac y = 1 l→ b in order-trans)
  apply (cut-tac a = 1 and b = b and c = b in lemma-4-2-i)
  apply simp
  apply (simp add: lemma-2-5-9-a)

  apply (rule-tac y = 1 r→ b in order-trans)
  apply (cut-tac a = 1 and b = b and c = b in lemma-4-2-ii)
  apply simp
  apply (simp add: lemma-2-5-9-b)
  apply (simp add: lemma-4-2-i [THEN sym])
  by (simp add: lemma-4-2-ii [THEN sym])

end

class simple-pseudo-hoop-algebra = pseudo-hoop-algebra +
  assumes simple: normalfilters ∩ properfilters = {{1}}

class proper = one +
  assumes proper: ∃ a . a ≠ 1

class strong-simple-pseudo-hoop-algebra = pseudo-hoop-algebra +
  assumes strong-simple: properfilters = {{1}}
begin

subclass proper
  apply unfold-locales
  apply (cut-tac strong-simple)
  apply (simp add: properfilters-def)
  apply (case-tac {1} = UNIV)
  apply blast
  by blast

```

```

lemma lemma-4-12-i-ii:  $a \neq 1 \implies \text{filterof}(\{a\}) = UNIV$ 
  apply (cut-tac strong-simple)
  apply (simp add: properfilters-def)
  apply (subgoal-tac filterof {a}  $\notin \{F \in \text{filters}. F \neq UNIV\}$ )
  apply (drule drop-assumption)
  apply (drule drop-assumption)
  apply simp
  apply simp
  apply safe
  apply (subgoal-tac  $a \in \text{filterof} \{a\}$ )
  apply simp
  apply (subst filterof-def)
  by simp

```

```

lemma lemma-4-12-i-iii:  $a \neq 1 \implies (\exists n . a \wedge^n \leq b)$ 
  apply (drule lemma-4-12-i-ii)
  apply (simp add: prop-3-2-i)
  by blast

```

```

lemma lemma-4-12-i-iv:  $a \neq 1 \implies (\exists n . a \text{ l-}n \rightarrow b = 1)$ 
  apply (subst lemma-2-4-7-a)
  apply (subst left-lesseq [THEN sym])
  by (simp add: lemma-4-12-i-iii)

```

```

lemma lemma-4-12-i-v:  $a \neq 1 \implies (\exists n . a \text{ r-}n \rightarrow b = 1)$ 
  apply (subst lemma-2-4-7-b)
  apply (subst right-lesseq [THEN sym])
  by (simp add: lemma-4-12-i-iii)

```

**end**

```

lemma (in pseudo-hoop-algebra) [simp]:  $\{1\} \in \text{filters}$ 
  apply (simp add: filters-def)
  apply safe
  apply (rule order.antisym)
  by simp-all

```

```

class strong-simple-pseudo-hoop-algebra-a = pseudo-hoop-algebra + proper +
assumes strong-simple-a:  $a \neq 1 \implies \text{filterof}(\{a\}) = UNIV$ 

```

```

begin
  subclass strong-simple-pseudo-hoop-algebra
    apply unfold-locales
    apply (simp add: properfilters-def)
    apply safe
    apply simp-all
    apply (case-tac  $xb = 1$ )
    apply simp
    apply (cut-tac  $a = xb$  in strong-simple-a)
    apply simp

```

```

    apply (simp add: filterof-def)
    apply blast
    apply (cut-tac proper)
    by blast
end

sublocale strong-simple-pseudo-hoop-algebra < strong-simple-pseudo-hoop-algebra-a
  apply unfold-locales
  by (simp add: lemma-4-12-i-ii)

lemma (in pseudo-hoop-algebra) power-impl:  $b \multimap a = a \implies b \wedge^n \multimap a = a$ 
  apply (induct-tac n)
  apply simp
  by (simp add: left-impl-ded)

lemma (in pseudo-hoop-algebra) power-impr:  $b \multimap a = a \implies b \wedge^n \multimap a = a$ 
  apply (induct-tac n)
  apply simp
  by (simp add: right-impl-ded)

context strong-simple-pseudo-hoop-algebra
begin

lemma lemma-4-13-i:  $b \multimap a = a \implies a = 1 \vee b = 1$ 
  apply safe
  apply (drule-tac a = b and b = a in lemma-4-12-i-iii)
  apply safe
  apply (subst (asm) left-lesseq)
  apply (drule-tac n = n in power-impl)
  by simp

lemma lemma-4-13-ii:  $b \multimap a = a \implies a = 1 \vee b = 1$ 
  apply safe
  apply (drule-tac a = b and b = a in lemma-4-12-i-iii)
  apply safe
  apply (subst (asm) right-lesseq)
  apply (drule-tac n = n in power-impr)
  by simp
end

class basic-pseudo-hoop-algebra-A = basic-pseudo-hoop-algebra +
  assumes left-impl-one:  $b \multimap a = a \implies a = 1 \vee b = 1$ 
  and right-impl-one:  $b \multimap a = a \implies a = 1 \vee b = 1$ 
begin
subclass linorder
  apply unfold-locales
  apply (cut-tac a = x and b = y in lemma-4-8-iii)
  apply (drule left-impl-one)
  apply (simp add: left-lesseq)

```

```

    by blast

lemma [simp]: (a l→ b) r→ b ≤ (b l→ a) r→ a
  apply (case-tac a = b)
  apply simp
  apply (cut-tac x = a and y = b in linear)
  apply safe
  apply (subst (asm) left-lesseq)
  apply (simp add: lemma-2-10-24)
  apply (subst (asm) left-lesseq)
  apply simp
  apply (subst left-lesseq)
  apply (cut-tac b = ((a l→ b) r→ b) l→ a and a = a l→ b in left-impl-one)
  apply (simp add: lemma-2-10-32)
  apply (simp add: left-lesseq [THEN sym])
  apply safe
  apply (erule notE)
  by simp

end

context basic-pseudo-hoop-algebra-A begin

lemma [simp]: (a r→ b) l→ b ≤ (b r→ a) l→ a
  apply (case-tac a = b)
  apply simp
  apply (cut-tac x = a and y = b in linear)
  apply safe
  apply (subst (asm) right-lesseq)
  apply simp
  apply (simp add: lemma-2-10-26)
  apply (unfold right-lesseq)
  apply (cut-tac b = ((a r→ b) l→ b) r→ a and a = a r→ b in right-impl-one)
  apply (simp add: lemma-2-10-33)
  apply (simp add: right-lesseq [THEN sym])
  apply safe
  apply (erule notE)
  by simp

subclass wajsberg-pseudo-hoop-algebra
  apply unfold-locales
  apply (rule order.antisym)
  apply simp-all
  apply (rule order.antisym)
  by simp-all

end

class strong-simple-basic-pseudo-hoop-algebra = strong-simple-pseudo-hoop-algebra

```

```

+ basic-pseudo-hoop-algebra
begin
subclass basic-pseudo-hoop-algebra-A
  apply unfold-locales
  apply (simp add: lemma-4-13-i)
  by (simp add: lemma-4-13-ii)

subclass wajsberg-pseudo-hoop-algebra
  by unfold-locales

end

end

```

## 9 Examples of Pseudo-Hoops

```

theory Examples
imports SpecialPseudoHoops LatticeProperties.Lattice-Ordered-Group
begin

```

```

declare add-uminus-conv-diff [simp del] right-minus [simp]
lemmas diff-minus = diff-conv-add-uminus

```

```

context lgroup
begin
lemma (in lgroup) less-eq-inf-2:  $(x \leq y) = (\text{inf } y \ x = x)$ 
  by (simp add: le-iff-inf inf-commute)
end

```

```

class lgroup-with-const = lgroup +
  fixes  $u :: 'a$ 
  assumes [simp]:  $0 \leq u$ 

```

```

definition  $G = \{a :: 'a :: \text{lgroup-with-const}. (0 \leq a \wedge a \leq u)\}$ 
typedef (overloaded) 'a  $G = G :: 'a :: \text{lgroup-with-const}$  set
proof
  show  $0 \in G$  by (simp add: G-def)
qed

```

```

instantiation  $G :: (\text{lgroup-with-const}) \text{ bounded-wajsberg-pseudo-hoop-algebra}$ 
begin

```

```

definition
  times-def:  $a * b \equiv \text{Abs-}G (\text{sup } (\text{Rep-}G \ a - u + \text{Rep-}G \ b) \ 0)$ 

```

```

lemma [simp]:  $\text{sup } (\text{Rep-}G \ a - u + \text{Rep-}G \ b) \ 0 \in G$ 
  apply (cut-tac x = a in Rep-G)

```

**apply** (*cut-tac*  $x = b$  **in** *Rep-G*)  
**apply** (*unfold* *G-def*)  
**apply** *safe*  
**apply** (*simp-all* *add: diff-minus*)  
**apply** (*rule* *right-move-to-right*)  
**apply** (*rule-tac*  $y = 0$  **in** *order-trans*)  
**apply** (*rule* *right-move-to-right*)  
**apply** *simp*  
**apply** (*rule* *right-move-to-left*)  
**by** *simp*

**definition**

*impl-def*:  $a \mapsto b \equiv \text{Abs-}G ((\text{Rep-}G\ b - \text{Rep-}G\ a + u) \sqcap u)$

**lemma** [*simp*]:  $\text{inf} (\text{Rep-}G\ (b::'a\ G) - \text{Rep-}G\ a + u) \ u \in G$

**apply** (*cut-tac*  $x = a$  **in** *Rep-G*)  
**apply** (*cut-tac*  $x = b$  **in** *Rep-G*)  
**apply** (*unfold* *G-def*)  
**apply** (*simp-all* *add: diff-minus*)  
**apply** *safe*  
**apply** (*rule* *right-move-to-left*)  
**apply** (*rule* *right-move-to-left*)  
**apply** *simp*  
**apply** (*rule-tac*  $y = 0$  **in** *order-trans*)  
**apply** (*rule* *left-move-to-right*)  
**by** *simp-all*

**definition**

*impr-def*:  $a \mapsto b \equiv \text{Abs-}G (\text{inf} (u - \text{Rep-}G\ a + \text{Rep-}G\ b) \ u)$

**lemma** [*simp*]:  $\text{inf} (u - \text{Rep-}G\ a + \text{Rep-}G\ b) \ u \in G$

**apply** (*cut-tac*  $x = a$  **in** *Rep-G*)  
**apply** (*cut-tac*  $x = b$  **in** *Rep-G*)  
**apply** (*unfold* *G-def*)  
**apply** (*simp-all* *add: diff-minus*)  
**apply** *safe*  
**apply** (*rule* *right-move-to-left*)  
**apply** (*rule* *right-move-to-left*)  
**apply** *simp*  
**apply** (*rule* *left-move-to-right*)  
**apply** (*rule-tac*  $y = u$  **in** *order-trans*)  
**apply** *simp-all*  
**apply** (*rule* *right-move-to-left*)  
**by** *simp-all*

**definition**

*one-def*:  $1 \equiv \text{Abs-}G\ u$

**definition**

*zero-def*:  $0 \equiv \text{Abs-G } 0$

**definition**

*order-def*:  $((a::'a \ G) \leq b) \equiv (a \ l \rightarrow b = 1)$

**definition**

*strict-order-def*:  $(a::'a \ G) < b \equiv (a \leq b \wedge \neg b \leq a)$

**definition**

*inf-def*:  $(a::'a \ G) \sqcap b = ((a \ l \rightarrow b) * a)$

**lemma** [*simp*]:  $(u::'a) \in G$

**by** (*simp add: G-def*)

**lemma** [*simp*]:  $(1::'a \ G) * a = a$

**apply** (*simp add: one-def times-def*)

**apply** (*cut-tac y = u::'a in Abs-G-inverse*)

**apply** *simp-all*

**apply** (*subgoal-tac sup (Rep-G a) (0::'a) = Rep-G a*)

**apply** (*simp add: Rep-G-inverse*)

**apply** (*cut-tac x = a in Rep-G*)

**apply** (*rule antisym*)

**apply** (*simp add: G-def*)

**by** *simp*

**lemma** [*simp*]:  $a * (1::'a \ G) = a$

**apply** (*simp add: one-def times-def*)

**apply** (*cut-tac y = u::'a in Abs-G-inverse*)

**apply** (*simp-all add: diff-minus add.assoc*)

**apply** (*subgoal-tac sup (Rep-G a) (0::'a) = Rep-G a*)

**apply** (*simp add: Rep-G-inverse*)

**apply** (*cut-tac x = a in Rep-G*)

**apply** (*rule antisym*)

**by** (*simp-all add: G-def*)

**lemma** [*simp*]:  $a \ l \rightarrow a = (1::'a \ G)$

**by** (*simp add: one-def impl-def*)

**lemma** [*simp*]:  $a \ r \rightarrow a = (1::'a \ G)$

**by** (*simp add: one-def impr-def diff-minus add.assoc*)

**lemma** [*simp*]:  $a \in G \implies \text{Rep-G } (\text{Abs-G } a) = a$

**apply** (*rule Abs-G-inverse*)

**by** *simp*

**lemma** *inf-def-1*:  $((a::'a \ G) \ l \rightarrow b) * a = \text{Abs-G } (\text{inf } (\text{Rep-G } a) (\text{Rep-G } b))$

**apply** (*simp add: times-def impl-def*)

**apply** (*subgoal-tac sup (inf (Rep-G b) (Rep-G a)) 0 = inf (Rep-G a) (Rep-G b)*)



**apply** *simp*  
**apply** (*rule antisym*)  
**apply** (*cut-tac x = a in Rep-G*)  
**apply** (*cut-tac x = b in Rep-G*)  
**apply** (*simp add: G-def*)  
**apply** (*subgoal-tac inf (Rep-G a) (Rep-G b) = inf (Rep-G b) (Rep-G a)*)  
**apply** *simp*  
**apply** (*rule antisym*)  
**by** *simp-all*

**lemma** *inf-def-2*:  $(a::'a G) * (a r \rightarrow b) = \text{Abs-G } (\text{inf } (\text{Rep-G } a) (\text{Rep-G } b))$   
**apply** (*simp add: times-def impr-def*)  
**apply** (*simp add: diff-minus add.assoc [THEN sym]*)  
**apply** (*simp add: add.assoc*)  
**apply** (*subgoal-tac sup (inf (Rep-G b) (Rep-G a)) 0 = inf (Rep-G a) (Rep-G b)*)  
**apply** *simp*  
**apply** (*rule antisym*)  
**apply** (*cut-tac x = a in Rep-G*)  
**apply** (*cut-tac x = b in Rep-G*)  
**apply** (*simp add: G-def*)  
**apply** (*subgoal-tac inf (Rep-G a) (Rep-G b) = inf (Rep-G b) (Rep-G a)*)  
**apply** *simp*  
**apply** (*rule antisym*)  
**by** *simp-all*

**lemma** *Rep-G-order*:  $(a \leq b) = (\text{Rep-G } a \leq \text{Rep-G } b)$   
**apply** (*simp add: order-def impl-def one-def*)  
**apply** *safe*  
**apply** (*subgoal-tac Rep-G (Abs-G (inf (Rep-G b - Rep-G a + u) u)) = Rep-G (Abs-G u)*)  
**apply** (*drule drop-assumption*)  
**apply** *simp*  
**apply** (*subst (asm) less-eq-inf-2 [THEN sym]*)  
**apply** (*simp add: diff-minus*)  
**apply** (*drule-tac a = u and b = Rep-G b + - Rep-G a + u and v = -u in add-order-preserving-right*)  
**apply** (*simp add: add.assoc*)  
**apply** (*drule-tac a = 0 and b = Rep-G b + - Rep-G a and v = Rep-G a in add-order-preserving-right*)  
**apply** (*simp add: add.assoc*)  
**apply** *simp*  
**apply** (*subgoal-tac Rep-G (Abs-G (inf (Rep-G b - Rep-G a + u) u)) = Rep-G (Abs-G u)*)  
**apply** *simp*  
**apply** *simp*  
**apply** (*subst less-eq-inf-2 [THEN sym]*)  
**apply** (*rule right-move-to-left*)  
**apply** *simp*  
**apply** (*simp add: diff-minus*)

**apply** (*rule right-move-to-left*)  
**by** *simp*

**lemma** *ded-left*:  $((a::'a\ G) * b) l \rightarrow c = a l \rightarrow b l \rightarrow c$   
**apply** (*simp add: times-def impl-def*)  
**apply** (*simp add: diff-minus minus-add*)  
**apply** (*simp add: add.assoc [THEN sym]*)  
**apply** (*simp add: inf-assoc*)  
**apply** (*subgoal-tac inf (Rep-G c + u) u = u*)  
**apply** (*subgoal-tac inf (u + - Rep-G a + u) u = u*)  
**apply** *simp*  
**apply** (*rule antisym*)  
**apply** *simp*  
**apply** *simp*  
**apply** (*simp add: add.assoc*)  
**apply** (*rule add-pos*)  
**apply** (*cut-tac x = a in Rep-G*)  
**apply** (*simp add: G-def*)  
**apply** (*rule left-move-to-left*)  
**apply** *simp*  
**apply** (*rule antisym*)  
**apply** *simp*  
**apply** *simp*  
**apply** (*rule add-pos-left*)  
**apply** (*cut-tac x = c in Rep-G*)  
**by** (*simp add: G-def*)

**lemma** *ded-right*:  $((a::'a\ G) * b) r \rightarrow c = b r \rightarrow a r \rightarrow c$   
**apply** (*simp add: times-def impr-def*)  
**apply** (*simp add: diff-minus minus-add*)  
**apply** (*simp add: add.assoc [THEN sym]*)  
**apply** (*simp add: inf-assoc*)  
**apply** (*subgoal-tac inf (u + Rep-G c) u = u*)  
**apply** (*subgoal-tac inf (u + - Rep-G b + u) u = u*)  
**apply** *simp*  
**apply** (*rule antisym*)  
**apply** *simp*  
**apply** *simp*  
**apply** (*simp add: add.assoc*)  
**apply** (*rule add-pos*)  
**apply** (*cut-tac x = b in Rep-G*)  
**apply** (*simp add: G-def*)  
**apply** (*rule left-move-to-left*)  
**apply** *simp*  
**apply** (*rule antisym*)  
**apply** *simp*  
**apply** *simp*  
**apply** (*rule add-pos*)  
**apply** (*cut-tac x = c in Rep-G*)

**by** (*simp add: G-def*)

**lemma** [*simp*]:  $0 \in G$   
**by** (*simp add: G-def*)

**lemma** [*simp*]:  $0 \leq (a::'a G)$   
**apply** (*simp add: order-def impl-def zero-def one-def diff-minus*)  
**apply** (*subgoal-tac inf (Rep-G a + u) u = u*)  
**apply** *simp*  
**apply** (*rule antisym*)  
**apply** *simp*  
**apply** (*cut-tac x = a in Rep-G*)  
**apply** (*unfold G-def*)  
**apply** *simp*  
**apply** (*rule add-pos-left*)  
**by** *simp*

**lemma** *lemma-W1*:  $((a::'a G) l \rightarrow b) r \rightarrow b = (b l \rightarrow a) r \rightarrow a$   
**apply** (*simp add: impl-def impr-def*)  
**apply** (*simp add: diff-minus minus-add*)  
**apply** (*simp add: add.assoc*)  
**apply** (*subgoal-tac Rep-G a  $\sqcup$  Rep-G b = Rep-G b  $\sqcup$  Rep-G a*)  
**apply** *simp*  
**apply** (*rule antisym*)  
**by** *simp-all*

**lemma** *lemma-W2*:  $((a::'a G) r \rightarrow b) l \rightarrow b = (b r \rightarrow a) l \rightarrow a$   
**apply** (*simp add: impl-def impr-def*)  
**apply** (*simp add: diff-minus minus-add*)  
**apply** (*simp add: add.assoc*)  
**apply** (*subgoal-tac Rep-G a  $\sqcup$  Rep-G b = Rep-G b  $\sqcup$  Rep-G a*)  
**apply** *simp*  
**apply** (*rule antisym*)  
**by** *simp-all*

**instance proof**

**fix** *a* **show**  $(1::'a G) * a = a$  **by** *simp*  
**fix** *a* **show**  $a * (1::'a G) = a$  **by** *simp*  
**fix** *a* **show**  $a l \rightarrow a = (1::'a G)$  **by** *simp*  
**fix** *a* **show**  $a r \rightarrow a = (1::'a G)$  **by** *simp*

**next**

**fix** *a b* **have** *a*:  $((a::'a G) l \rightarrow b) * a = (b l \rightarrow a) * b$   
**by** (*simp add: inf-def-1 inf-commute*)  
**show**  $((a::'a G) l \rightarrow b) * a = (b l \rightarrow a) * b$  **by** (*rule a*)

**next**

**fix** *a b* **have** *a*:  $a * ((a::'a G) r \rightarrow b) = b * (b r \rightarrow a)$  **by** (*simp add: inf-def-2*)

```

inf-commute)
  show  $a * ((a::'a G) r \rightarrow b) = b * (b r \rightarrow a)$  by (rule a)
next
  fix a b have !!a b .  $((a::'a G) l \rightarrow b) * a = a * (a r \rightarrow b)$  by (simp add: inf-def-2
inf-def-1)
  from this show  $((a::'a G) l \rightarrow b) * a = a * (a r \rightarrow b)$  by simp
next
  fix a b c show  $(a::'a G) * b l \rightarrow c = a l \rightarrow b l \rightarrow c$  by (rule ded-left)
next
  fix a b c show  $(a::'a G) * b r \rightarrow c = b r \rightarrow a r \rightarrow c$  by (rule ded-right)
next
  fix  $a::'a G$  have  $0 \leq a$  by simp
  from this show  $0 \leq a$  by simp
next
  fix a  $b::'a G$  show  $(a \leq b) = (a l \rightarrow b = 1)$  by (simp add: order-def)
next
  fix a  $b::'a G$  show  $(a < b) = (a \leq b \wedge \neg b \leq a)$  by (simp add: strict-order-def)
next
  fix a  $b::'a G$  show  $(a l \rightarrow b) r \rightarrow b = (b l \rightarrow a) r \rightarrow a$  by (rule lemma-W1)
next
  fix a  $b::'a G$  show  $(a r \rightarrow b) l \rightarrow b = (b r \rightarrow a) l \rightarrow a$  by (rule lemma-W2)
next
  fix a  $b::'a G$  show  $a \sqcap b = (a l \rightarrow b) * a$  by (rule inf-def)
next
  fix a  $b::'a G$  show  $a \sqcap b = a * (a r \rightarrow b)$  by (simp add: inf-def inf-def-2 inf-def-1)

```

qed

end

context order

begin

definition

*closed-interval*:: $'a \Rightarrow 'a \Rightarrow 'a$  set ( $|| \cdot, \cdot || [0, 0]$  900) where  
*closed-interval* a b =  $\{c . a \leq c \wedge c \leq b\}$

definition

*convex* =  $\{A . \forall a b . a \in A \wedge b \in A \longrightarrow |[a, b]| \subseteq A\}$

end

context group-add

begin

definition

*subgroup* =  $\{A . 0 \in A \wedge (\forall a b . a \in A \wedge b \in A \longrightarrow a + b \in A \wedge -a \in A)\}$

lemma [simp]:  $A \in \text{subgroup} \implies 0 \in A$

by (simp add: subgroup-def)

**lemma** [simp]:  $A \in \text{subgroup} \implies a \in A \implies b \in A \implies a + b \in A$   
**apply** (subst (asm) subgroup-def)  
**by** simp

**lemma** minus-subgroup:  $A \in \text{subgroup} \implies -a \in A \implies a \in A$   
**apply** (subst (asm) subgroup-def)  
**apply** safe  
**apply** (drule-tac  $x = -a$  in spec)  
**by** simp

**definition**

$\text{add-set}:: 'a \text{ set} \Rightarrow 'a \text{ set} \Rightarrow 'a \text{ set}$  (**infixl** +++ 70) **where**  
 $\text{add-set } A \ B = \{c . \exists a \in A . \exists b \in B . c = a + b\}$

**definition**

$\text{normal} = \{A . (\forall a . A \text{ +++ } \{a\} = \{a\} \text{ +++ } A)\}$   
**end**

**context** lgroup

**begin**

**definition**

$\text{lsubgroup} = \{A . A \in \text{subgroup} \wedge (\forall a \ b . a \in A \wedge b \in A \longrightarrow \text{inf } a \ b \in A \wedge \text{sup } a \ b \in A)\}$

**lemma** inf-lsubgroup:

$A \in \text{lsubgroup} \implies a \in A \implies b \in A \implies \text{inf } a \ b \in A$   
**by** (simp add: lsubgroup-def)

**lemma** sup-lsubgroup:

$A \in \text{lsubgroup} \implies a \in A \implies b \in A \implies \text{sup } a \ b \in A$   
**by** (simp add: lsubgroup-def)

**end**

**definition**

$F \ K = \{a:: 'a \ G . (u:: 'a:: \text{lgroup-with-const}) - \text{Rep-G } a \in K\}$

**lemma** F-def2:  $K \in \text{normal} \implies F \ K = \{a:: 'a \ G . - \text{Rep-G } a + (u:: 'a:: \text{lgroup-with-const}) \in K\}$

**apply** (simp add: normal-def F-def)  
**apply** safe  
**apply** (drule-tac  $x = \text{Rep-G } x$  in spec)  
**apply** (subgoal-tac  $u \in K \text{ +++ } \{\text{Rep-G } x\}$ )  
**apply** simp  
**apply** (drule drop-assumption)  
**apply** (drule drop-assumption)  
**apply** (simp add: add-set-def)  
**apply** safe

```

apply (subgoal-tac  $- \text{Rep-G } x + u = - \text{Rep-G } x + \text{Rep-G } x + b$ )
apply simp
apply (subst add.assoc)
apply simp
apply (subst add-set-def)
apply simp
apply (rule-tac  $x = u - \text{Rep-G } x$  in bexI)
apply (simp add: diff-minus add.assoc)
apply simp
apply (drule-tac  $x = \text{Rep-G } x$  in spec)
apply (subgoal-tac  $u \in K \text{ +++ } \{\text{Rep-G } x\}$ )
apply (drule drop-assumption)
apply (drule drop-assumption)
apply (simp add: add-set-def)
apply safe
apply (subgoal-tac  $u - \text{Rep-G } x = a + (\text{Rep-G } x - \text{Rep-G } x)$ )
apply simp
apply (subst diff-minus)
apply (subst diff-minus)
apply (subst add.assoc [THEN sym])
apply simp
apply simp
apply (subst add-set-def)
apply simp
apply (rule-tac  $x = - \text{Rep-G } x + u$  in bexI)
apply (simp add: add.assoc [THEN sym])
by simp

```

**context** *lgroup* **begin**

**lemma** *sup-assoc-lgroup*:  $a \sqcup b \sqcup c = a \sqcup (b \sqcup c)$   
**by** (rule *sup-assoc*)

**end**

**lemma** *normal-1*:  $K \in \text{normal} \implies K \in \text{convex} \implies K \in \text{lsubgroup} \implies x \in \{a\}$

```

** F K  $\implies x \in F K$  ** {a}
apply (subst (asm) times-set-def)
apply simp
apply safe
apply (subst (asm) F-def2)
apply simp-all
apply (subgoal-tac  $-u + \text{Rep-G } y \in K$ )
apply (subgoal-tac  $\text{Rep-G } a - u + \text{Rep-G } y \in K \text{ +++ } \{\text{Rep-G } a\}$ )
apply (subst (asm) add-set-def)
apply simp
apply safe
apply (simp add: times-set-def)
apply (rule-tac  $x = \text{Abs-G } (\text{inf } (\text{sup } (aa + u) 0) u)$  in bexI)
apply (subgoal-tac  $aa = \text{Rep-G } a - u + \text{Rep-G } y - \text{Rep-G } a$ )

```

**apply** (*subgoal-tac inf (sup (aa + u) (0::'a)) u ∈ G*)  
**apply** *safe*  
**apply** *simp*  
**apply** (*simp add: times-def*)  
**apply** (*subgoal-tac (sup (Rep-G a - u + Rep-G y) 0) = (sup (inf (sup (Rep-G a - u + Rep-G y - Rep-G a + u - u + Rep-G a) (- u + Rep-G a)) (Rep-G a)) 0)*)  
**apply** *simp*  
**apply** (*simp add: diff-minus add.assoc*)  
**apply** (*subgoal-tac inf (sup (Rep-G a + (- u + Rep-G y)) (- u + Rep-G a)) (Rep-G a) = (sup (Rep-G a + (- u + Rep-G y)) (- u + Rep-G a))*)  
**apply** *simp*

**apply** (*subst sup-assoc-lgroup*)  
**apply** (*subgoal-tac (sup (- u + Rep-G a) (0::'a)) = 0*)  
**apply** *simp*  
**apply** (*rule antisym*)  
**apply** *simp*  
**apply** (*rule left-move-to-right*)  
**apply** *simp*  
**apply** (*cut-tac x = a in Rep-G*)  
**apply** (*simp add: G-def*)  
**apply** *simp*  
**apply** (*rule antisym*)  
**apply** *simp*  
**apply** *simp*  
**apply** *safe*  
**apply** (*rule left-move-to-right*)  
**apply** *simp*  
**apply** (*rule left-move-to-right*)  
**apply** *simp*  
**apply** (*cut-tac x = y in Rep-G*)  
**apply** (*simp add: G-def*)  
**apply** (*rule left-move-to-right*)  
**apply** *simp*  
**apply** (*rule right-move-to-left*)  
**apply** *simp*  
**apply** (*simp add: G-def*)  
**apply** (*simp add: diff-minus*)  
**apply** (*simp add: add.assoc*)  
**apply** (*simp add: F-def*)  
**apply** (*subgoal-tac inf (sup (aa + u) (0::'a)) u ∈ G*)  
**apply** *simp*  
**apply** (*simp add: diff-minus minus-add add.assoc [THEN sym]*)  
**apply** (*subst (asm) convex-def*)  
**apply** *simp*  
**apply** (*drule-tac x = 0 in spec*)  
**apply** (*drule-tac x = sup (- aa) 0 in spec*)  
**apply** *safe*

```

apply (subst (asm) lsubgroup-def)
apply simp
apply (rule sup-lsubgroup)
apply simp
apply (rule minus-subgroup)
apply (subst (asm) lsubgroup-def)
apply simp
apply simp
apply (subst (asm) lsubgroup-def)
apply simp
apply (subgoal-tac sup (inf (- aa) u) (0::'a) ∈ [| 0::'a , sup (- aa) (0::'a) |])
apply blast
apply (subst closed-interval-def)
apply safe
apply simp
apply simp

```

```

apply (simp add: G-def)
apply (subst (asm) normal-def)
apply simp
apply (drule drop-assumption)
apply (simp add: add-set-def)
apply (rule-tac x = -u + Rep-G y in beXI)
apply (simp add: diff-minus add.assoc)
apply simp
apply (rule minus-subgroup)
apply (simp add: lsubgroup-def)
by (simp add: minus-add)

```

```

lemma normal-2:  $K \in \text{normal} \implies K \in \text{convex} \implies K \in \text{lsubgroup} \implies x \in F K$ 
** {a}  $\implies x \in \{a\}$  ** F K
apply (subst (asm) times-set-def)
apply simp
apply safe
apply (subst (asm) F-def)
apply simp-all
apply hypsubst-thin
apply (subgoal-tac Rep-G x - u ∈ K)
apply (subgoal-tac Rep-G x - u + Rep-G a ∈ {Rep-G a} +++ K)
apply (subst (asm) add-set-def)
apply simp
apply safe
apply (simp add: times-set-def)
apply (rule-tac x = Abs-G (inf (sup (u + b) 0) u) in beXI)
apply (subgoal-tac b = - Rep-G a + Rep-G x - u + Rep-G a)
apply (subgoal-tac inf (sup (u + b) 0) u ∈ G)
apply safe
apply simp
apply (simp add: times-def)

```



```

apply (simp add: diff-minus add.assoc)
apply (simp add: add.assoc [THEN sym])
apply (subgoal-tac sup (Rep-G x + - u + Rep-G a) 0 = sup (inf (sup (Rep-G
x + - u + Rep-G a) (Rep-G a + - u)) (Rep-G a)) 0))
apply simp
apply (subgoal-tac inf (sup (Rep-G x + - u + Rep-G a) (Rep-G a + - u))
(Rep-G a) = sup (Rep-G x + - u + Rep-G a) (Rep-G a + - u))
apply simp

apply (subst sup-assoc-lgroup)
apply (subgoal-tac (sup (Rep-G a + - u) (0::'a)) = 0))
apply simp
apply (rule antisym)
apply simp
apply (rule right-move-to-right)
apply simp
apply (cut-tac x = a in Rep-G)
apply (simp add: G-def)
apply simp
apply (rule antisym)
apply simp
apply simp
apply safe
apply (rule right-move-to-right)
apply simp
apply (rule right-move-to-right)
apply simp
apply (cut-tac x = x in Rep-G)
apply (simp add: G-def)
apply (rule right-move-to-right)
apply simp
apply (rule left-move-to-left)
apply simp
apply (simp add: G-def)
apply (simp add: diff-minus)
apply (simp add: add.assoc)
apply (simp add: F-def2)
apply (subgoal-tac inf (sup (u + b) (0::'a)) u ∈ G)
apply simp
apply (simp add: diff-minus minus-add add.assoc [THEN sym])
apply (subst (asm) convex-def)
apply simp
apply (drule-tac x = 0 in spec)
apply (drule-tac x = sup (- b) 0 in spec)
apply safe
apply (subst (asm) lsubgroup-def)
apply simp
apply (rule sup-lsubgroup)
apply simp

```

```

apply (rule minus-subgroup)
apply (subst (asm) lsubgroup-def)
apply simp
apply simp
apply (subst (asm) lsubgroup-def)
apply simp
apply (simp add: add.assoc)
apply (subgoal-tac sup (inf (- b) u) (0::'a) ∈ |[ 0::'a , sup (-b) 0]|)
apply blast
apply (subst closed-interval-def)
apply safe
apply simp
apply simp

```

```

apply (simp add: G-def)
apply (subgoal-tac {Rep-G a} +++ K = K +++ {Rep-G a})
apply simp
apply (simp add: add-set-def)
apply (subst (asm) normal-def)
apply simp
apply (rule minus-subgroup)
apply (simp add: lsubgroup-def)
by (simp add: diff-minus minus-add)

```

**lemma**  $K \in \text{normal} \implies K \in \text{convex} \implies K \in \text{lsubgroup} \implies F K \in \text{normalfilters}$

```

apply (rule lemma-3-10-ii-i)
apply (subgoal-tac K ∈ subgroup)
apply (subst filters-def)
apply safe
apply (subgoal-tac 1 ∈ F K)
apply simp
apply (subst F-def)
apply safe
apply (subst one-def)
apply simp
apply (simp add: F-def)
apply (simp add: convex-def)
apply (drule-tac x = 0 in spec)
apply (drule-tac x = (u - Rep-G b) + (u - Rep-G a) in spec)
apply simp
apply (subgoal-tac u - Rep-G (a * b) ∈ |[ 0::'a , u - Rep-G b + (u - Rep-G
a) ]|)
apply blast
apply (simp add: closed-interval-def)
apply safe
apply (cut-tac x = a * b in Rep-G)
apply (simp add: G-def diff-minus)
apply (rule right-move-to-left)
apply simp

```

```

apply (simp add: times-def)
apply (subgoal-tac ( $u - (\text{Rep-G } a - u + \text{Rep-G } b) = u - \text{Rep-G } b + (u - \text{Rep-G } a)$ ))
apply simp
apply (simp add: diff-minus add.assoc minus-add)
apply (subst (asm) Rep-G-order)

```

```

apply (simp add: F-def)
apply (subst (asm) convex-def)
apply simp
apply (drule-tac  $x = 0$  in spec)
apply (drule-tac  $x = u - \text{Rep-G } a$  in spec)
apply simp
apply (subgoal-tac  $u - \text{Rep-G } b \in [| 0::'a, u - \text{Rep-G } a |]$ )
apply blast
apply (subst closed-interval-def)
apply simp
apply safe
apply (cut-tac  $x = b$  in Rep-G)
apply (simp add: G-def)
apply (safe)
apply (simp add: diff-minus)
apply (rule right-move-to-left)
apply simp
apply (simp add: diff-minus)
apply (rule add-order-preserving-left)
apply (rule minus-order)
apply simp
apply (simp add: lsubgroup-def)
apply (rule normal-1)
apply simp-all
apply (rule normal-2)
by simp-all

```

```

definition  $N = \{a::'a::lgroup. a \leq 0\}$ 
typedef (overloaded) ( $'a::lgroup$ )  $N = N :: 'a::lgroup$  set
proof
  show  $0 \in N$  by (simp add: N-def)
qed

```

```

class cancel-product-pseudo-hoop-algebra = cancel-pseudo-hoop-algebra + product-pseudo-hoop-algebra

```

```

context group-add
begin
subclass cancel-semigroup-add
proof qed

```

```

end

```

**instantiation**  $N :: (\text{lgroup}) \text{ pseudo-hoop-algebra}$   
**begin**

**definition**

*times-N-def*:  $a * b \equiv \text{Abs-N } (\text{Rep-N } a + \text{Rep-N } b)$

**lemma** [*simp*]:  $\text{Rep-N } a + \text{Rep-N } b \in N$

**apply** (*cut-tac*  $x = a$  **in** *Rep-N*)

**apply** (*cut-tac*  $x = b$  **in** *Rep-N*)

**apply** (*simp add: N-def*)

**apply** (*rule-tac left-move-to-right*)

**apply** (*rule-tac*  $y = 0$  **in** *order-trans*)

**apply** *simp-all*

**apply** (*rule-tac minus-order*)

**by** *simp*

**definition**

*impl-N-def*:  $a \text{ l} \rightarrow b \equiv \text{Abs-N } (\text{inf } (\text{Rep-N } b - \text{Rep-N } a) 0)$

**definition**

*inf-N-def*:  $(a :: 'a N) \sqcap b = (a \text{ l} \rightarrow b) * a$

**lemma** [*simp*]:  $\text{inf } (\text{Rep-N } b - \text{Rep-N } a) 0 \in N$

**apply** (*cut-tac*  $x = a$  **in** *Rep-N*)

**apply** (*cut-tac*  $x = b$  **in** *Rep-N*)

**by** (*simp add: N-def*)

**definition**

*impr-N-def*:  $a \text{ r} \rightarrow b \equiv \text{Abs-N } (\text{inf } (- \text{Rep-N } a + \text{Rep-N } b) 0)$

**lemma** [*simp*]:  $\text{inf } (- \text{Rep-N } a + \text{Rep-N } b) 0 \in N$

**apply** (*cut-tac*  $x = a$  **in** *Rep-N*)

**apply** (*cut-tac*  $x = b$  **in** *Rep-N*)

**by** (*simp add: N-def*)

**definition**

*one-N-def*:  $1 \equiv \text{Abs-N } 0$

**lemma** [*simp*]:  $0 \in N$

**by** (*simp add: N-def*)

**definition**

*order-N-def*:  $((a :: 'a N) \leq b) \equiv (a \text{ l} \rightarrow b = 1)$

**definition**

*strict-order-N-def*:  $(a :: 'a N) < b \equiv (a \leq b \wedge \neg b \leq a)$

**lemma** *order-Rep-N*:  
 $((a::'a\ N) \leq b) = (\text{Rep-}N\ a \leq \text{Rep-}N\ b)$   
**apply** (*subst order-N-def*)  
**apply** (*simp add: impl-N-def one-N-def*)  
**apply** (*subgoal-tac (Abs-N (inf (Rep-N b - Rep-N a) (0::'a)) = Abs-N (0::'a))*)  
 $= ((\text{Rep-}N\ (\text{Abs-}N\ (\text{inf}\ (\text{Rep-}N\ b - \text{Rep-}N\ a)\ (0::'a)))) = \text{Rep-}N\ (\text{Abs-}N\ (0::'a))))$   
**apply** *simp*  
**apply** (*drule drop-assumption*)  
**apply** (*simp add: Abs-N-inverse*)  
**apply** *safe*  
**apply** (*subgoal-tac  $0 \leq \text{Rep-}N\ b - \text{Rep-}N\ a$* )  
**apply** (*drule-tac  $v = \text{Rep-}N\ a$  in add-order-preserving-right*)  
**apply** (*simp add: diff-minus add.assoc*)  
**apply** (*rule-tac  $y = \text{inf}\ (\text{Rep-}N\ b - \text{Rep-}N\ a)\ (0::'a)$  in order-trans*)  
**apply** *simp*  
**apply** (*drule drop-assumption*)  
**apply** *simp*  
**apply** (*rule antisym*)  
**apply** *simp*  
**apply** *simp*  
**apply** (*simp add: diff-minus*)  
**apply** (*rule right-move-to-left*)  
**apply** *simp*  
**apply** *simp*  
**by** (*simp add: Abs-N-inverse*)

**lemma** *order-Abs-N*:  
 $a \in N \implies b \in N \implies (a \leq b) = (\text{Abs-}N\ a \leq \text{Abs-}N\ b)$   
**apply** (*subst order-N-def*)  
**apply** (*simp add: impl-N-def one-N-def*)  
**apply** (*simp add: Abs-N-inverse*)  
**apply** (*subgoal-tac  $\text{inf}\ (b - a)\ 0 \in N$* )  
**apply** (*subgoal-tac (Abs-N (inf (b - a) (0::'a)) = Abs-N (0::'a)) = (Rep-N (Abs-N (inf (b - a) (0::'a)))) = Rep-N (Abs-N (0::'a)))*)  
**apply** *simp*  
**apply** (*simp add: Abs-N-inverse*)  
**apply** (*drule drop-assumption*)  
**apply** (*drule drop-assumption*)  
**apply** (*drule drop-assumption*)  
**apply** (*drule drop-assumption*)  
**apply** *simp*  
**apply** *safe*  
**apply** (*rule antisym*)  
**apply** *simp-all*  
**apply** (*simp add: diff-minus*)  
**apply** (*rule right-move-to-left*)  
**apply** *simp*  
**apply** (*subgoal-tac  $0 \leq b - a$* )

**apply** (*drule-tac*  $v = a$  **in** *add-order-preserving-right*)  
**apply** (*simp add: diff-minus add.assoc*)  
**apply** (*rule-tac*  $y = \text{inf } (b - a) (0::'a)$  **in** *order-trans*)  
**apply** *simp*  
**apply** (*drule drop-assumption*)  
**apply** *simp*  
**apply** (*simp add: Abs-N-inverse*)  
**by** (*simp add: N-def*)

**lemma** [*simp*]:  $(1::'a N) * a = a$   
**by** (*simp add: one-N-def times-N-def Abs-N-inverse Rep-N-inverse*)

**lemma** [*simp*]:  $a * (1::'a N) = a$   
**by** (*simp add: one-N-def times-N-def Abs-N-inverse Rep-N-inverse*)

**lemma** [*simp*]:  $a \text{ l}\rightarrow a = (1::'a N)$   
**by** (*simp add: impl-N-def one-N-def Abs-N-inverse Rep-N-inverse*)

**lemma** [*simp*]:  $a \text{ r}\rightarrow a = (1::'a N)$   
**by** (*simp add: impr-N-def one-N-def Abs-N-inverse Rep-N-inverse*)

**lemma** *impl-times*:  $(a \text{ l}\rightarrow b) * a = (b \text{ l}\rightarrow a) * (b::'a N)$   
**apply** (*simp add: impl-N-def impr-N-def times-N-def Abs-N-inverse Rep-N-inverse*)

**apply** (*subgoal-tac*  $\text{inf } (\text{Rep-N } b - \text{Rep-N } a + \text{Rep-N } a) (\text{Rep-N } a) = \text{inf } (\text{Rep-N } a - \text{Rep-N } b + \text{Rep-N } b) (\text{Rep-N } b)$ )  
**apply** *simp*  
**apply** (*subgoal-tac*  $\text{Rep-N } b - \text{Rep-N } a + \text{Rep-N } a = \text{Rep-N } b$ )  
**apply** *simp*  
**apply** (*subgoal-tac*  $\text{Rep-N } a - \text{Rep-N } b + \text{Rep-N } b = \text{Rep-N } a$ )  
**apply** *simp*  
**apply** (*rule antisym*)  
**by** *simp-all*

**lemma** *impr-times*:  $a * (a \text{ r}\rightarrow b) = (b::'a N) * (b \text{ r}\rightarrow a)$   
**apply** (*simp add: impr-N-def times-N-def Abs-N-inverse Rep-N-inverse*)  
**apply** (*subgoal-tac*  $\text{inf } (\text{Rep-N } a + (- \text{Rep-N } a + \text{Rep-N } b)) (\text{Rep-N } a) = \text{inf } (\text{Rep-N } b + (- \text{Rep-N } b + \text{Rep-N } a)) (\text{Rep-N } b)$ )  
**apply** *simp*  
**apply** (*simp add: add.assoc [THEN sym]*)  
**apply** (*rule antisym*)  
**by** *simp-all*

**lemma** *impr-impl-times*:  $(a \text{ l}\rightarrow b) * a = (a::'a N) * (a \text{ r}\rightarrow b)$   
**by** (*simp add: impl-N-def impr-N-def times-N-def Abs-N-inverse Rep-N-inverse*)

**lemma** *impl-ded*:  $(a::'a N) * b \text{ l}\rightarrow c = a \text{ l}\rightarrow b \text{ l}\rightarrow c$   
**apply** (*simp add: impl-N-def impr-N-def times-N-def Abs-N-inverse Rep-N-inverse*)

**apply** (*subgoal-tac* *inf* ( $\text{Rep-N } c - (\text{Rep-N } a + \text{Rep-N } b)$ ) ( $0::'a$ ) = *inf* (*inf* ( $\text{Rep-N } c - \text{Rep-N } b - \text{Rep-N } a$ ) ( $-\text{Rep-N } a$ )) ( $0::'a$ ))  
**apply** *simp*  
**apply** (*rule antisym*)  
**apply** *simp-all*  
**apply** *safe*  
**apply** (*rule-tac*  $y = \text{Rep-N } c - (\text{Rep-N } a + \text{Rep-N } b)$  **in** *order-trans*)  
**apply** *simp*  
**apply** (*simp add: diff-minus minus-add add.assoc*)  
**apply** (*rule-tac*  $y = 0$  **in** *order-trans*)  
**apply** *simp*  
**apply** (*cut-tac*  $x = a$  **in** *Rep-N*)  
**apply** (*simp add: N-def*)  
**apply** (*drule-tac*  $u = 0$  **and**  $v = -\text{Rep-N } a$  **in** *add-order-preserving*)  
**apply** *simp*  
**apply** (*rule-tac*  $y = \text{inf } (\text{Rep-N } c - \text{Rep-N } b - \text{Rep-N } a)$  ( $-\text{Rep-N } a$ ) **in** *order-trans*)  
**apply** (*rule inf-le1*)  
**apply** (*rule-tac*  $y = \text{Rep-N } c - \text{Rep-N } b - \text{Rep-N } a$  **in** *order-trans*)  
**apply** *simp*  
**by** (*simp add: diff-minus minus-add add.assoc*)

**lemma impr-ded:** ( $a::'a \text{ N}$ ) \*  $b \text{ r} \rightarrow c = b \text{ r} \rightarrow a \text{ r} \rightarrow c$   
**apply** (*simp add: impr-N-def impr-N-def times-N-def Abs-N-inverse Rep-N-inverse*)

**apply** (*subgoal-tac* *inf* ( $-(\text{Rep-N } a + \text{Rep-N } b) + \text{Rep-N } c$ ) ( $0::'a$ ) = *inf* (*inf* ( $-\text{Rep-N } b + (-\text{Rep-N } a + \text{Rep-N } c)$ ) ( $-\text{Rep-N } b$ )) ( $0::'a$ ))  
**apply** *simp*  
**apply** (*rule antisym*)  
**apply** *simp-all*  
**apply** *safe*  
**apply** (*rule-tac*  $y = -(\text{Rep-N } a + \text{Rep-N } b) + \text{Rep-N } c$  **in** *order-trans*)  
**apply** *simp*  
**apply** (*simp add: diff-minus minus-add add.assoc*)  
**apply** (*rule-tac*  $y = 0$  **in** *order-trans*)  
**apply** *simp*  
**apply** (*cut-tac*  $x = b$  **in** *Rep-N*)  
**apply** (*simp add: N-def*)  
**apply** (*drule-tac*  $u = 0$  **and**  $v = -\text{Rep-N } b$  **in** *add-order-preserving*)  
**apply** *simp*  
**apply** (*rule-tac*  $y = \text{inf } (-\text{Rep-N } b + (-\text{Rep-N } a + \text{Rep-N } c))$  ( $-\text{Rep-N } b$ ) **in** *order-trans*)  
**apply** (*rule inf-le1*)  
**apply** (*rule-tac*  $y = -\text{Rep-N } b + (-\text{Rep-N } a + \text{Rep-N } c)$  **in** *order-trans*)  
**apply** *simp*  
**by** (*simp add: diff-minus minus-add add.assoc*)

**instance proof**

```

fix a show (1::a N) * a = a by simp
fix a show a * (1::a N) = a by simp
fix a show a  $l \rightarrow$  a = (1::a N) by simp
fix a show a  $r \rightarrow$  a = (1::a N) by simp
next
fix a b::a N show (a  $l \rightarrow$  b) * a = (b  $l \rightarrow$  a) * b by (simp add: impl-times)
next
fix a b::a N show a * (a  $r \rightarrow$  b) = b * (b  $r \rightarrow$  a) by (simp add: impr-times)
next
fix a b::a N show (a  $l \rightarrow$  b) * a = a * (a  $r \rightarrow$  b) by (simp add: impr-impl-times)
next
fix a b c::a N show a * b  $l \rightarrow$  c = a  $l \rightarrow$  b  $l \rightarrow$  c by (simp add: impl-ded)
fix a b c::a N show a * b  $r \rightarrow$  c = b  $r \rightarrow$  a  $r \rightarrow$  c by (simp add: impr-ded)
next
fix a b::a N show (a  $\leq$  b) = (a  $l \rightarrow$  b = 1) by (simp add: order-N-def)
next
fix a b::a N show (a < b) = (a  $\leq$  b  $\wedge$   $\neg$  b  $\leq$  a) by (simp add: strict-order-N-def)
next
fix a b::a N show a  $\sqcap$  b = (a  $l \rightarrow$  b) * a by (simp add: inf-N-def)
next
fix a b::a N show a  $\sqcap$  b = a * (a  $r \rightarrow$  b) by (simp add: inf-N-def impr-impl-times)
qed

end

```

```

lemma Rep-N-inf: Rep-N ((a::a::lgroup N)  $\sqcap$  b) = (Rep-N a)  $\sqcap$  (Rep-N b)
apply (rule antisym)
apply simp-all
apply safe
apply (simp add: order-Rep-N [THEN sym])
apply (simp add: order-Rep-N [THEN sym])
apply (subgoal-tac inf (Rep-N a) (Rep-N b)  $\in$  N)
apply (subst order-Abs-N)
apply simp-all
apply (cut-tac x = a  $\sqcap$  b in Rep-N)
apply (simp add: N-def)
apply (simp add: Rep-N-inverse)
apply safe
apply (subst order-Rep-N)
apply (simp add: Abs-N-inverse)
apply (subst order-Rep-N)
apply (simp add: Abs-N-inverse)
apply (simp add: N-def)
apply (rule-tac y = Rep-N a in order-trans)
apply simp
apply (cut-tac x = a in Rep-N)
by (simp add: N-def)

```

```

context lgroup begin

```



**lemma** *sup-inf-distrib2-lgroup*:  $(b \sqcap c) \sqcup a = (b \sqcup a) \sqcap (c \sqcup a)$   
**by** (*rule sup-inf-distrib2*)

**lemma** *inf-sup-distrib2-lgroup*:  $(b \sqcup c) \sqcap a = (b \sqcap a) \sqcup (c \sqcap a)$   
**by** (*rule inf-sup-distrib2*)  
**end**

**instantiation**  $N :: (\text{lgroup})$  *cancel-product-pseudo-hoop-algebra*  
**begin**

**lemma** *cancel-times-left*:  $(a::'a N) * b = a * c \implies b = c$   
**apply** (*simp add: times-N-def Abs-N-inverse Rep-N-inverse*)  
**apply** (*subgoal-tac Rep-N (Abs-N (Rep-N a + Rep-N b)) = Rep-N (Abs-N (Rep-N a + Rep-N c))*)  
**apply** (*drule drop-assumption*)  
**apply** (*simp add: Abs-N-inverse*)  
**apply** (*subgoal-tac Abs-N (Rep-N b) = Abs-N (Rep-N c)*)  
**apply** (*drule drop-assumption*)  
**apply** (*simp add: Rep-N-inverse*)  
**by** *simp-all*

**lemma** *cancel-times-right*:  $b * (a::'a N) = c * a \implies b = c$   
**apply** (*simp add: times-N-def Abs-N-inverse Rep-N-inverse*)  
**apply** (*subgoal-tac Rep-N (Abs-N (Rep-N b + Rep-N a)) = Rep-N (Abs-N (Rep-N c + Rep-N a))*)  
**apply** (*drule drop-assumption*)  
**apply** (*simp add: Abs-N-inverse*)  
**apply** (*subgoal-tac Abs-N (Rep-N b) = Abs-N (Rep-N c)*)  
**apply** (*drule drop-assumption*)  
**apply** (*simp add: Rep-N-inverse*)  
**by** *simp-all*

**lemma** *prod-1*:  $((a::'a N) \text{ l}\rightarrow b) \text{ l}\rightarrow c \leq ((b \text{ l}\rightarrow a) \text{ l}\rightarrow c) \text{ l}\rightarrow c$   
**apply** (*unfold impl-N-def times-N-def Abs-N-inverse Rep-N-inverse order-N-def one-N-def*)  
**apply** (*subst Abs-N-inverse*)  
**apply** *simp*  
**apply** (*subst Abs-N-inverse*)  
**apply** *simp*  
**apply** (*subst Abs-N-inverse*)  
**apply** *simp*  
**apply** (*subst Abs-N-inverse*)  
**apply** *simp*  
**apply** (*subst Abs-N-inverse*)  
**apply** *simp*  
**apply** (*subgoal-tac inf (inf (Rep-N c - inf (Rep-N c - inf (Rep-N a - Rep-N b) 0) 0) 0 - inf (Rep-N c - inf (Rep-N b - Rep-N a) 0) 0) 0 = 0*)  
**apply** *simp*

**apply** (*rule antisym*)  
**apply** *simp*  
**apply** (*rule inf-greatest*)  
**apply** (*subst diff-minus*)  
**apply** (*subst diff-minus*)  
**apply** (*subst diff-minus*)  
**apply** (*subst diff-minus*)  
**apply** (*rule right-move-to-left*)  
**apply** *simp-all*  
**apply** (*simp add: diff-minus minus-add*)

**apply** (*subst sup-inf-distrib2-lgroup*)  
**apply** *simp*

**apply** (*subst inf-sup-distrib2-lgroup*)  
**apply** *simp*

**apply** (*rule-tac y=Rep-N c + (Rep-N a + - Rep-N b + - Rep-N c) in order-trans*)  
**apply** *simp-all*  
**apply** (*rule-tac y=Rep-N c + (Rep-N a + - Rep-N b) in order-trans*)  
**apply** *simp-all*  
**apply** (*rule add-order-preserving-left*)  
**apply** (*simp add: add.assoc*)  
**apply** (*rule add-order-preserving-left*)  
**apply** (*rule left-move-to-left*)  
**apply** *simp*  
**apply** (*cut-tac x = c in Rep-N*)  
**apply** (*simp add: N-def*)  
**apply** (*rule minus-order*)  
**by** *simp*

**lemma prod-2:**  $((a::'a N) r \rightarrow b) r \rightarrow c \leq ((b r \rightarrow a) r \rightarrow c) r \rightarrow c$   
**apply** (*unfold impr-N-def times-N-def Abs-N-inverse Rep-N-inverse right-lesseq one-N-def*)  
**apply** (*subst Abs-N-inverse*)  
**apply** *simp*  
**apply** (*subst Abs-N-inverse*)  
**apply** *simp*  
**apply** (*subst Abs-N-inverse*)  
**apply** *simp*  
**apply** (*subst Abs-N-inverse*)  
**apply** *simp*  
**apply** (*subst Abs-N-inverse*)  
**apply** *simp*  
**apply** (*subgoal-tac inf (- inf (- inf (- Rep-N a + Rep-N b) (0::'a) + Rep-N c) (0::'a) + inf (- inf (- inf (- Rep-N b + Rep-N a) (0::'a) + Rep-N c) (0::'a)*)

```

+ Rep-N c) (0::'a))
  (0::'a) = 0)
apply simp
apply (rule antisym)
apply simp
apply (rule inf-greatest)
apply (rule minus-order)
apply (subst minus-add)
apply (subst minus-minus)
apply (subst minus-zero)
apply (rule left-move-to-right)
apply (subst minus-minus)
apply simp
apply (simp add: minus-add)
apply simp-all

apply (subst sup-inf-distrib2-lgroup)
apply simp

apply (subst inf-sup-distrib2-lgroup)
apply simp

apply (rule-tac y = - Rep-N c + (- Rep-N b + Rep-N a) + Rep-N c in
order-trans)
apply simp-all
apply (rule-tac y = - Rep-N b + Rep-N a + Rep-N c in order-trans)
apply simp-all
apply (rule add-order-preserving-right)
apply (simp add: add.assoc [THEN sym])
apply (rule add-order-preserving-right)
apply (rule left-move-to-left)
apply (rule right-move-to-right)
apply simp
apply (cut-tac x = c in Rep-N)
by (simp add: N-def)

lemma prod-3: (b::'a N) l→ b * b ≤ a □ (a l→ b) l→ b
apply (simp add: impl-N-def times-N-def Abs-N-inverse Rep-N-inverse order-N-def
one-N-def Rep-N-inf)
apply (subst Abs-N-inverse)
apply (simp add: add.assoc N-def)
apply (subst Abs-N-inverse)
apply (simp add: add.assoc N-def)
apply (subgoal-tac inf (inf (sup (Rep-N b - Rep-N a) (sup (Rep-N b - (Rep-N
b - Rep-N a)) (Rep-N b))) (0::'a) - inf (Rep-N b + Rep-N b - Rep-N b) (0::'a))
(0::'a) = 0)
apply simp

```

**apply** (*rule antisym*)  
**apply** (*simp*)  
**apply** (*subst diff-minus*)  
**apply** (*subst diff-minus*)  
**apply** (*subst diff-minus*)  
**apply** (*subst diff-minus*)  
**apply** (*subst diff-minus*)  
**apply** (*rule inf-greatest*)  
**apply** (*rule right-move-to-left*)  
**apply** (*subst minus-minus*)  
**apply** (*simp-all*)  
**apply** (*simp add: add.assoc*)  
**apply** (*rule-tac y = Rep-N b in order-trans*)  
**by** (*simp-all*)

**lemma prod-4:**  $(b::'a N) r \rightarrow b * b \leq a \sqcap (a r \rightarrow b) r \rightarrow b$   
**apply** (*simp add: impr-N-def times-N-def Abs-N-inverse Rep-N-inverse Rep-N-inf minus-add*)  
**apply** (*subst order-Abs-N [THEN sym]*)  
**apply** (*simp add: N-def*)  
**apply** (*simp add: N-def*)  
**apply** (*simp*)  
**apply** (*rule-tac y = - Rep-N a + Rep-N b in order-trans*)  
**apply** (*simp-all*)  
**apply** (*rule-tac y = Rep-N b in order-trans*)  
**apply** (*simp*)  
**apply** (*rule right-move-to-left*)  
**apply** (*simp*)  
**apply** (*rule minus-order*)  
**apply** (*simp*)  
**apply** (*cut-tac x = a in Rep-N*)  
**by** (*simp add: N-def*)

**lemma prod-5:**  $((a::'a N) l \rightarrow b) l \rightarrow b) * (c * a l \rightarrow f * a) * (c * b l \rightarrow f * b) \leq c l \rightarrow f$   
**apply** (*simp add: impl-N-def times-N-def Abs-N-inverse Rep-N-inverse Rep-N-inf minus-add*)  
**apply** (*subst Abs-N-inverse*)  
**apply** (*simp add: N-def*)  
**apply** (*subst Abs-N-inverse*)  
**apply** (*simp add: N-def*)  
**apply** (*subst Abs-N-inverse*)  
**apply** (*simp add: N-def*)  
**apply** (*subst order-Abs-N [THEN sym]*)  
**apply** (*simp add: N-def inf-assoc [THEN sym]*)  
**apply** (*simp add: N-def*)  
**apply** (*simp only: diff-minus minus-add minus-minus add.assoc*)  
**apply** (*subst (4) add.assoc [THEN sym]*)  
**apply** (*subst (5) add.assoc [THEN sym]*)

```

apply (simp only: right-minus add-0-left)
apply (rule right-move-to-right)
apply (simp only: minus-add add.assoc [THEN sym] add-0-left right-minus)
by (simp add: minus-add)

```

**lemma** *prod-6*:  $((a::'a\ N)\ r\ \rightarrow\ b)\ r\ \rightarrow\ b) * (a * c\ r\ \rightarrow\ a * f) * (b * c\ r\ \rightarrow\ b * f) \leq c\ r\ \rightarrow\ f$

```

apply (simp add: impr-N-def times-N-def Abs-N-inverse Rep-N-inverse Rep-N-inf
minus-add)
apply (subst Abs-N-inverse)
apply (simp add: N-def)
apply (subst Abs-N-inverse)
apply (simp add: N-def)
apply (subst Abs-N-inverse)
apply (simp add: N-def)
apply (subst order-Abs-N [THEN sym])
apply (simp add: N-def inf-assoc [THEN sym])
apply (simp add: N-def)
apply (simp only: diff-minus minus-add minus-minus add.assoc)
apply (subst (4) add.assoc [THEN sym])
apply (subst (5) add.assoc [THEN sym])
apply (simp only: left-minus add-0-left)
apply (rule right-move-to-right)
apply (simp only: minus-add add.assoc [THEN sym] add-0-left right-minus)
by (simp add: minus-add)

```

**instance**

**apply** *intro-classes*

**by** (fact cancel-times-left cancel-times-right prod-1 prod-2 prod-3 prod-4 prod-5 prod-6)+

**end**

**definition** *OrdSum* =

$\{x. (\exists a::'a::pseudo\text{-hoop}\text{-algebra}. x = (a, 1::'b::pseudo\text{-hoop}\text{-algebra})) \vee (\exists b::'b. x = (1::'a, b))\}$

**typedef** (**overloaded**) ('a, 'b) *OrdSum* = *OrdSum* :: ('a::pseudo-hoop-algebra  $\times$  'b::pseudo-hoop-algebra) set

**proof**

**show**  $(1, 1) \in \text{OrdSum}$  **by** (simp add: *OrdSum-def*)

**qed**

**lemma** [*simp*]:  $(1, b) \in \text{OrdSum}$

**by** (simp add: *OrdSum-def*)

**lemma** [*simp*]:  $(a, 1) \in \text{OrdSum}$

**by** (simp add: *OrdSum-def*)

**definition**

*first*  $x = \text{fst } (\text{Rep-OrdSum } x)$

**definition**

*second*  $x = \text{snd } (\text{Rep-OrdSum } x)$

**lemma** *if-unfold-left*:  $((\text{if } a \text{ then } b \text{ else } c) = d) = ((a \longrightarrow b = d) \wedge (\neg a \longrightarrow c = d))$

**apply** *auto*

**done**

**lemma** *if-unfold-right*:  $(d = (\text{if } a \text{ then } b \text{ else } c)) = ((a \longrightarrow d = b) \wedge (\neg a \longrightarrow d = c))$

**apply** *auto*

**done**

**lemma** *fst-snd-eq*:  $\text{fst } a = x \implies \text{snd } a = y \implies (x, y) = a$

**apply** (*subgoal-tac*  $x = \text{fst } a$ )

**apply** (*subgoal-tac*  $y = \text{snd } a$ )

**apply** (*drule drop-assumption*)

**apply** (*drule drop-assumption*)

**by** *simp-all*

**instantiation** *OrdSum* ::  $(\text{pseudo-hoop-algebra}, \text{pseudo-hoop-algebra}) \text{ pseudo-hoop-algebra}$   
**begin**

**definition**

*times-OrdSum-def*:  $a * b \equiv ($   
 $\text{if } \text{second } a = 1 \wedge \text{second } b = 1 \text{ then}$   
 $\text{Abs-OrdSum } (\text{first } a * \text{first } b, 1)$   
 $\text{else if } \text{first } a = 1 \wedge \text{first } b = 1 \text{ then}$   
 $\text{Abs-OrdSum } (1, \text{second } a * \text{second } b)$   
 $\text{else if } \text{first } a = 1 \wedge \text{second } b = 1 \text{ then}$   
 $b$   
 $\text{else}$   
 $a)$

**definition**

*one-OrdSum-def*:  $1 \equiv \text{Abs-OrdSum } (1, 1)$

**definition**

*impl-OrdSum-def*:  $a \text{ l}\rightarrow b \equiv ($   
 $\text{if } \text{second } a = 1 \wedge \text{second } b = 1 \text{ then}$   
 $\text{Abs-OrdSum } (\text{first } a \text{ l}\rightarrow \text{first } b, 1)$   
 $\text{else if } \text{first } a = 1 \wedge \text{first } b = 1 \text{ then}$   
 $\text{Abs-OrdSum } (1, \text{second } a \text{ l}\rightarrow \text{second } b)$   
 $\text{else if } \text{first } a = 1 \wedge \text{second } b = 1 \text{ then}$   
 $b$   
 $\text{else}$

1)

**definition**

*impr-OrdSum-def*:  $a \ r \rightarrow \ b \equiv$   
(if *second*  $a = 1 \wedge$  *second*  $b = 1$  then  
  *Abs-OrdSum* (*first*  $a \ r \rightarrow$  *first*  $b, 1$ )  
else if *first*  $a = 1 \wedge$  *first*  $b = 1$  then  
  *Abs-OrdSum* ( $1, \textit{second } a \ r \rightarrow \textit{second } b$ )  
else if *first*  $a = 1 \wedge$  *second*  $b = 1$  then  
   $b$   
else  
   $1$ )

**definition**

*order-OrdSum-def*:  $((a::('a, 'b) \textit{OrdSum}) \leq b) \equiv (a \ l \rightarrow \ b = 1)$

**definition**

*inf-OrdSum-def*:  $(a::('a, 'b) \textit{OrdSum}) \sqcap b = (a \ l \rightarrow \ b) * a$

**definition**

*strict-order-OrdSum-def*:  $(a::('a, 'b) \textit{OrdSum}) < b \equiv (a \leq b \wedge \neg b \leq a)$

**lemma** *OrdSum-first* [*simp*]:  $(a, 1) \in \textit{OrdSum}$

by (*simp add: OrdSum-def*)

**lemma** *OrdSum-second* [*simp*]:  $(1, b) \in \textit{OrdSum}$

by (*simp add: OrdSum-def*)

**lemma** *Rep-OrdSum-eq*:  $\textit{Rep-OrdSum } x = \textit{Rep-OrdSum } y \implies x = y$

**apply** (*subgoal-tac Abs-OrdSum (Rep-OrdSum x) = Abs-OrdSum (Rep-OrdSum y)*)

**apply** (*drule drop-assumption*)

**apply** (*simp add: Rep-OrdSum-inverse*)

by *simp*

**lemma** *Abs-OrdSum-eq*:  $x \in \textit{OrdSum} \implies y \in \textit{OrdSum} \implies \textit{Abs-OrdSum } x = \textit{Abs-OrdSum } y \implies x = y$

**apply** (*subgoal-tac Rep-OrdSum (Abs-OrdSum x) = Rep-OrdSum (Abs-OrdSum y)*)

**apply** (*unfold Abs-OrdSum-inverse*) [1]

by *simp-all*

**lemma** [*simp*]:  $\textit{fst } (\textit{Rep-OrdSum } a) \neq 1 \implies (\textit{snd } (\textit{Rep-OrdSum } a) \neq 1 = \textit{False})$

**apply** (*cut-tac x = a in Rep-OrdSum*)

**apply** (*simp add: OrdSum-def*)

by *auto*

**lemma** *fst-not-one-snd*:  $\textit{fst } (\textit{Rep-OrdSum } a) \neq 1 \implies (\textit{snd } (\textit{Rep-OrdSum } a) = 1)$

**apply** (*cut-tac x = a in Rep-OrdSum*)

```

apply (simp add: OrdSum-def)
by auto

lemma snd-not-one-fst: snd (Rep-OrdSum a) ≠ 1 ⇒ (fst (Rep-OrdSum a) = 1)
apply (cut-tac x = a in Rep-OrdSum)
apply (simp add: OrdSum-def)
by auto

lemma fst-not-one-simp [simp]: fst (Rep-OrdSum c) ≠ 1 ⇒ Abs-OrdSum (fst
(Rep-OrdSum c), 1) = c
apply (rule Rep-OrdSum-eq)
apply (simp add: Abs-OrdSum-inverse)
apply (rule fst-snd-eq)
apply simp-all
by (simp add: fst-not-one-snd)

lemma snd-not-one-simp [simp]: snd (Rep-OrdSum c) ≠ 1 ⇒ Abs-OrdSum (1,
snd (Rep-OrdSum c)) = c
apply (rule Rep-OrdSum-eq)
apply (simp add: Abs-OrdSum-inverse)
apply (rule fst-snd-eq)
apply simp-all
by (simp add: snd-not-one-fst)

lemma A: fixes a b::('a, 'b) OrdSum shows (a l→ b) * a = a * (a r→ b)
apply (simp add: one-OrdSum-def impr-OrdSum-def impl-OrdSum-def second-def
first-def Abs-OrdSum-inverse Rep-OrdSum-inverse)
apply safe
apply (simp-all add: fst-snd-eq times-OrdSum-def left-right-impl-times first-def
second-def Abs-OrdSum-inverse Rep-OrdSum-inverse )
apply safe
by simp-all

instance
proof
fix a::('a, 'b) OrdSum show 1 * a = a
by (simp add: fst-snd-eq one-OrdSum-def times-OrdSum-def first-def second-def
Abs-OrdSum-inverse Rep-OrdSum-inverse)
next
fix a::('a, 'b) OrdSum show a * 1 = a
by (simp add: fst-snd-eq one-OrdSum-def times-OrdSum-def first-def second-def
Abs-OrdSum-inverse Rep-OrdSum-inverse)
next
fix a::('a, 'b) OrdSum show a l→ a = 1
by (simp add: one-OrdSum-def impl-OrdSum-def)
next
fix a::('a, 'b) OrdSum show a r→ a = 1
by (simp add: one-OrdSum-def impr-OrdSum-def)

```



```

next
fix a b::('a, 'b) OrdSum show (a l→ b) * a = (b l→ a) * b
  apply (unfold one-OrdSum-def impl-OrdSum-def second-def first-def Abs-OrdSum-inverse
  Rep-OrdSum-inverse)
  apply simp
  apply safe
  by (simp-all add: times-OrdSum-def left-impl-times first-def second-def Abs-OrdSum-inverse
  Rep-OrdSum-inverse )
next
fix a b::('a, 'b) OrdSum show a * (a r→ b) = b * (b r→ a)
  apply (unfold one-OrdSum-def impr-OrdSum-def second-def first-def Abs-OrdSum-inverse
  Rep-OrdSum-inverse)
  apply (simp)
  apply safe
  by (simp-all add: fst-snd-eq times-OrdSum-def right-impl-times first-def second-def
  Abs-OrdSum-inverse Rep-OrdSum-inverse )
next
fix a b::('a, 'b) OrdSum show (a l→ b) * a = a * (a r→ b) by (rule A)
next
fix a b c::('a, 'b) OrdSum show a * b l→ c = a l→ b l→ c
  apply (unfold times-OrdSum-def)
  apply simp apply safe
  apply (simp-all add: impl-OrdSum-def)
  apply (simp-all add: first-def second-def)
  apply (simp-all add: Abs-OrdSum-inverse Rep-OrdSum-inverse)
  apply (simp-all add: fst-snd-eq)
  apply (simp-all add: Abs-OrdSum-inverse Rep-OrdSum-inverse)
  apply (simp-all add: left-impl-ded)
  apply (simp-all add: fst-snd-eq one-OrdSum-def times-OrdSum-def left-impl-ded
  impl-OrdSum-def second-def first-def Abs-OrdSum-inverse Rep-OrdSum-inverse)
  by auto
next
fix a b c::('a, 'b) OrdSum show a * b r→ c = b r→ a r→ c
  apply (simp add: right-impl-ded impr-OrdSum-def second-def first-def one-OrdSum-def
  times-OrdSum-def second-def first-def Abs-OrdSum-inverse Rep-OrdSum-inverse)
  by auto
next
fix a b::('a, 'b) OrdSum show (a ≤ b) = (a l→ b = 1)
  by (simp add: order-OrdSum-def)
next
fix a b::('a, 'b) OrdSum show (a < b) = (a ≤ b ∧ ¬ b ≤ a)
  by (simp add: strict-order-OrdSum-def)
next
fix a b::('a, 'b) OrdSum show a ⊓ b = (a l→ b) * a by (simp add: inf-OrdSum-def)
next
fix a b::('a, 'b) OrdSum show a ⊓ b = a * (a r→ b) by (simp add: inf-OrdSum-def
  A)
qed

```

**definition**

$Second = \{x . \exists b . x = Abs-OrdSum(1::'a, b::'b)\}$

**end**

**lemma**  $Second \in normalfilters$

**apply** (*unfold normalfilters-def*)  
**apply** *safe*  
**apply** (*unfold filters-def*)  
**apply** *safe*  
**apply** (*unfold Second-def*)  
**apply** *auto*  
**apply** (*rule-tac*  $x = ba * bb$  **in**  $exI$ )  
**apply** (*simp* *add: times-OrdSum-def second-def first-def Abs-OrdSum-inverse*  
*Rep-OrdSum-inverse*)  
**apply** (*rule-tac*  $x = second\ b$  **in**  $exI$ )  
**apply** (*subgoal-tac*  $Abs-OrdSum(1::'a, second\ b) = Abs-OrdSum(first\ b, second\ b)$ )  
**apply** *simp*  
**apply** (*simp* *add: first-def second-def Rep-OrdSum-inverse*)  
**apply** (*subgoal-tac*  $first\ b = 1$ )  
**apply** *simp*  
**apply** (*simp* *add: order-OrdSum-def one-OrdSum-def impl-OrdSum-def second-def*  
*first-def Abs-OrdSum-inverse Rep-OrdSum-inverse*)  
**apply** (*unfold second-def first-def*)  
**apply** (*case-tac*  $ba = (1::'b) \wedge snd(Rep-OrdSum\ b) = (1::'b)$ )  
**apply** *simp*  
**apply** (*simp* *add: Abs-OrdSum-inverse Rep-OrdSum-inverse*)  
**apply** (*subgoal-tac*  $Rep-OrdSum(Abs-OrdSum(fst(Rep-OrdSum\ b), 1::'b)) =$   
 $Rep-OrdSum(Abs-OrdSum(1::'a, 1::'b))$ )  
**apply** (*drule drop-assumption*)  
**apply** (*simp* *add: Abs-OrdSum-inverse Rep-OrdSum-inverse*)  
**apply** *simp*  
**apply** *simp*  
**apply** (*simp* *add: Abs-OrdSum-inverse Rep-OrdSum-inverse*)  
**apply** (*case-tac*  $fst(Rep-OrdSum\ b) = (1::'a)$ )  
**apply** *simp*  
**apply** *simp*  
**apply** (*simp* *add: Abs-OrdSum-inverse Rep-OrdSum-inverse*)  
**apply** (*case-tac*  $snd(Rep-OrdSum\ b) = (1::'b)$ )  
**apply** *simp-all*  
**apply** (*simp* *add: Abs-OrdSum-inverse Rep-OrdSum-inverse*)  
**apply** (*simp* *add: impr-OrdSum-def impl-OrdSum-def second-def first-def Abs-OrdSum-inverse*  
*Rep-OrdSum-inverse*)  
**apply** *safe*  
**apply** (*unfold second-def first-def*)  
**apply** (*simp-all* *add: second-def first-def Abs-OrdSum-inverse Rep-OrdSum-inverse*)  
**apply** (*case-tac*  $snd(Rep-OrdSum\ a) = (1::'b)$ )

```

apply simp-all
apply auto
apply (case-tac snd (Rep-OrdSum a) = (1::'b))
apply auto
apply (rule-tac x = 1 in exI)
apply (rule Rep-OrdSum-eq)
apply (simp-all add: second-def first-def Abs-OrdSum-inverse Rep-OrdSum-inverse)
apply (subgoal-tac Rep-OrdSum (Abs-OrdSum (fst (Rep-OrdSum a) l → fst (Rep-OrdSum
b), 1::'b)) = Rep-OrdSum (Abs-OrdSum (1::'a, ba))))
apply (drule drop-assumption)
apply (simp add: second-def first-def Abs-OrdSum-inverse Rep-OrdSum-inverse)
apply (simp add: left-lesseq [THEN sym] right-lesseq [THEN sym])
apply simp
apply (rule-tac x = 1 in exI)
apply (rule Rep-OrdSum-eq)
apply (simp-all add: second-def first-def Abs-OrdSum-inverse Rep-OrdSum-inverse)
apply (subgoal-tac Rep-OrdSum (Abs-OrdSum (fst (Rep-OrdSum a) l → fst (Rep-OrdSum
b), 1::'b)) = Rep-OrdSum (Abs-OrdSum (1::'a, ba))))
apply (drule drop-assumption)
apply (simp add: second-def first-def Abs-OrdSum-inverse Rep-OrdSum-inverse)
apply (simp add: left-lesseq [THEN sym] right-lesseq [THEN sym])
apply simp

apply (simp add: impr-OrdSum-def impl-OrdSum-def second-def first-def Abs-OrdSum-inverse
Rep-OrdSum-inverse)
apply safe
apply (unfold second-def first-def)
apply (simp-all add: second-def first-def Abs-OrdSum-inverse Rep-OrdSum-inverse)
apply (case-tac snd (Rep-OrdSum a) = (1::'b))
apply simp-all
apply auto
apply (case-tac snd (Rep-OrdSum a) = (1::'b))
apply auto
apply (rule-tac x = 1 in exI)
apply (rule Rep-OrdSum-eq)
apply (simp-all add: second-def first-def Abs-OrdSum-inverse Rep-OrdSum-inverse)
apply (subgoal-tac Rep-OrdSum (Abs-OrdSum (fst (Rep-OrdSum a) r → fst (Rep-OrdSum
b), 1::'b)) = Rep-OrdSum (Abs-OrdSum (1::'a, ba))))
apply (drule drop-assumption)
apply (simp add: second-def first-def Abs-OrdSum-inverse Rep-OrdSum-inverse)
apply (simp add: left-lesseq [THEN sym] right-lesseq [THEN sym])
apply simp
apply (rule-tac x = 1 in exI)
apply (rule Rep-OrdSum-eq)
apply (simp-all add: second-def first-def Abs-OrdSum-inverse Rep-OrdSum-inverse)
apply (subgoal-tac Rep-OrdSum (Abs-OrdSum (fst (Rep-OrdSum a) r → fst (Rep-OrdSum
b), 1::'b)) = Rep-OrdSum (Abs-OrdSum (1::'a, ba))))
apply (drule drop-assumption)
apply (simp add: second-def first-def Abs-OrdSum-inverse Rep-OrdSum-inverse)

```

```

apply (simp add: left-lesseq [THEN sym] right-lesseq [THEN sym])
by simp

class linear-pseudo-hoop-algebra = pseudo-hoop-algebra + linorder

instantiation OrdSum :: (linear-pseudo-hoop-algebra, linear-pseudo-hoop-algebra)
linear-pseudo-hoop-algebra
begin
instance
proof
  fix x y::('a, 'b) OrdSum show  $x \leq y \vee y \leq x$ 
    apply (simp add: order-OrdSum-def impl-OrdSum-def one-OrdSum-def sec-
ond-def first-def Abs-OrdSum-inverse Rep-OrdSum-inverse)
    apply (cut-tac x = fst (Rep-OrdSum x) and y = fst (Rep-OrdSum y) in linear)
    apply (cut-tac x = snd (Rep-OrdSum x) and y = snd (Rep-OrdSum y) in
linear)
    apply (simp add: left-lesseq)
    by auto [1]
qed
end

instantiation bool:: pseudo-hoop-algebra
begin
definition impl-bool-def:
   $a \rightarrow b = (a \longrightarrow b)$ 

definition impr-bool-def:
   $a \rightarrow b = (a \longrightarrow b)$ 

definition one-bool-def:
   $1 = True$ 

definition times-bool-def:
   $a * b = (a \wedge b)$ 

lemma inf-bool-def:  $(a :: bool) \sqcap b = (a \rightarrow b) * a$ 
by (auto simp add: times-bool-def impl-bool-def)

instance
  apply intro-classes
  apply (simp-all add: impl-bool-def impr-bool-def one-bool-def times-bool-def le-bool-def
less-bool-def inf-bool-def)
  by auto

end

context cancel-pseudo-hoop-algebra begin end

```

```

lemma  $\neg$  class.cancel-pseudo-hoop-algebra (*) ( $\sqcap$ ) ( $l \rightarrow$ ) ( $\leq$ ) ( $<$ ) (1:: bool) ( $r \rightarrow$ )
  apply (unfold class.cancel-pseudo-hoop-algebra-def)
  apply (unfold class.cancel-pseudo-hoop-algebra-axioms-def)
  apply safe
  apply (drule drop-assumption)
  apply (drule-tac x = False in spec)
  apply (drule drop-assumption)
  apply (drule-tac x = True in spec)
  apply (drule-tac x = False in spec)
  by (simp add: times-bool-def)

```

```

context pseudo-hoop-algebra begin
lemma classorder: class.order ( $\leq$ ) ( $<$ )
  proof qed
end

```

```

lemma impl-OrdSum-first: Abs-OrdSum (x, 1)  $l \rightarrow$  Abs-OrdSum (y, 1) = Abs-OrdSum
(x  $l \rightarrow$  y, 1)
  by (simp add: impl-OrdSum-def first-def second-def Abs-OrdSum-inverse Rep-OrdSum-inverse)

```

```

lemma impl-OrdSum-second: Abs-OrdSum (1, x)  $l \rightarrow$  Abs-OrdSum (1, y) = Abs-OrdSum
(1, x  $l \rightarrow$  y)
  by (simp add: impl-OrdSum-def first-def second-def Abs-OrdSum-inverse Rep-OrdSum-inverse)

```

```

lemma impl-OrdSum-first-second:  $x \neq 1 \implies$  Abs-OrdSum (x, 1)  $l \rightarrow$  Abs-OrdSum
(1, y) = 1
  by (simp add: one-OrdSum-def impl-OrdSum-def first-def second-def Abs-OrdSum-inverse
Rep-OrdSum-inverse)

```

```

lemma Abs-OrdSum-bijective:  $x \in$  OrdSum  $\implies$   $y \in$  OrdSum  $\implies$  (Abs-OrdSum x
= Abs-OrdSum y) = (x = y)
  apply safe
  apply (subgoal-tac Rep-OrdSum (Abs-OrdSum x) = Rep-OrdSum (Abs-OrdSum
y))
  apply (unfold Abs-OrdSum-inverse) [1]
  by simp-all

```

```

context pseudo-hoop-algebra begin end

```

```

context linear-pseudo-hoop-algebra begin end
context basic-pseudo-hoop-algebra begin end

```

```

lemma class.pseudo-hoop-algebra (*) ( $\sqcap$ ) ( $l \rightarrow$ ) ( $\leq$ ) ( $<$ ) (1::'a::pseudo-hoop-algebra)
( $r \rightarrow$ )
   $\implies$   $\neg$  (class.linear-pseudo-hoop-algebra ( $\leq$ ) ( $<$ ) (*) ( $\sqcap$ ) ( $l \rightarrow$ ) (1::'a) ( $r \rightarrow$ ))
   $\implies$   $\neg$  class.basic-pseudo-hoop-algebra (*) ( $\sqcap$ ) ( $l \rightarrow$ ) ( $\leq$ ) ( $<$ ) (1::('a, bool)
OrdSum) ( $r \rightarrow$ )
  apply (unfold class.linear-pseudo-hoop-algebra-def)

```

```

apply (unfold class.linorder-def)
apply (unfold class.linorder-axioms-def)
apply safe
apply (rule classorder)
apply (unfold class.basic-pseudo-hoop-algebra-def) [1]
apply simp
apply (unfold class.basic-pseudo-hoop-algebra-axioms-def) [1]
apply safe
apply (subgoal-tac (Abs-OrdSum (x, 1) l→ Abs-OrdSum (y, 1)) l→ Abs-OrdSum
(1, False) ≤
  ((Abs-OrdSum (y, 1) l→ Abs-OrdSum (x, 1)) l→ Abs-OrdSum (1, False))
l→ Abs-OrdSum (1, False))
apply (unfold impl-OrdSum-first) [1]
apply (case-tac x l→ y ≠ 1 ∧ y l→ x ≠ 1)
apply (simp add: impl-OrdSum-first-second)
apply (unfold order-OrdSum-def one-OrdSum-def one-bool-def impl-OrdSum-second
impl-bool-def ) [1]
apply simp
apply (cut-tac x = (1::'a, False) and y = (1::'a, True) in Abs-OrdSum-eq)
apply simp-all
apply (unfold left-lesseq)
by simp

end

```

## References

- [1] B. Bosbach. Komplementäre halbgruppen. axiomatik und arithmetik. *Fundamenta Mathematicae*, 64:257 – 287, 1969.
- [2] B. Bosbach. Komplementäre halbgruppen. kongruenzen and quotienten. *Fundamenta Mathematicae*, 69:1 – 14, 1970.
- [3] R. Ceterchi. Pseudo-Wajsberg algebras. *Mult.-Valued Log.*, 6(1-2):67–88, 2001. G. C. Moisil memorial issue.
- [4] G. Georgescu, L. Leuştean, and V. Preoteasa. Pseudo-hoops. *Journal of Multiple-Valued Logic and Soft Computing*, 11(1-2):153 – 184, 2005.