Residuated Lattices

Victor B. F. Gomes
Georg Struth
Department of Computer Science, University of Sheffield

March 17, 2025

Abstract

The theory of residuated lattices, first proposed by Ward and Dilworth [5], is formalised in Isabelle/HOL. This includes concepts of residuated functions; their adjoints and conjugates. It also contains necessary and sufficient conditions for the existence of these operations in an arbitrary lattice. The mathematical components for residuated lattices are linked to the AFP entry for relation algebra. In particular, we prove Jónsson and Tsinakis [2] conditions for a residuated boolean algebra to form a relation algebra.

Contents

1 Introduction

text * These theory files formalise algebraic residuated structures. They are briefly and sparsely commented. More information can be found in the books by Galatos and *al.* [1], or the originals papers by Ward and Dilworth [5], Jonsson and Tsinakis [2], and Maddux [3].

The mathematical components for residuated lattices are linked to the AFP entry for relation algebra. Residuated lattices are also important in the context of Pratt's action algebras, which are currently formalised whitin the AFP entry for Kleene algebra. We are planning to link Kleene algebras and action algebras with this entry in the future.

Isabelle/HOL default notation for lattices is used whenever possible. Nevertheless, we use $\cdot$ as the multiplicative symbol instead of $*$, which is the one used in Isabelle libraries.

theory *Residuated-Lattices*
imports *Kleene-Algebra.Signatures*
begin

notation
times (**infixl** ‹·› *70*)

unbundle *lattice-syntax*

2 Residuated Lattices

2.1 Residuated Functions on a Partial Order

We follow Galatos and *al.* to define residuated funtions on partial orders. Material from articles by Maddux, and Jósson and Tsinakis are also considered.

This development should in the future be expanded to functions or categories where the sources and targets have different type.

Let P be a partial order, or a poset. A map $f : P \to P$ is residuated if there exists a map $g : P \to P$ such that $f(x) \leq y \Leftrightarrow x \leq g(y)$ for all $x, y \in P$. That is, they are adjoints of a Galois connection.

context *order* **begin**

definition *residuated-pair* :: $('a \Rightarrow 'a) \Rightarrow ('a \Rightarrow 'a) \Rightarrow bool$ **where**
residuated-pair f g $\equiv \forall\, x\ y.\ f(x) \leq y \longleftrightarrow x \leq g(y)$

theorem *residuated-pairI* [*intro*]:
assumes $\forall\, x\ y.\ x \leq y \longrightarrow f\ x \leq f\ y$
and $\forall\, x\ y.\ x \leq y \longrightarrow g\ x \leq g\ y$
and $\forall\, x.\ x \leq (g\ o\ f)\ x$
and $\forall\, x.\ (f\ o\ g)\ x \leq x$
shows *residuated-pair f g*
by (*metis assms comp-apply local.order-trans residuated-pair-def antisym*)

definition *residuated* :: $('a \Rightarrow 'a) \Rightarrow bool$ **where**
residuated f $\equiv \exists\, g.$ *residuated-pair f g*

If a map f is residuated, then its residual g is unique.

lemma *residual-unique*: *residuated f* $\Longrightarrow \exists!g.$ *residuated-pair f g*
unfolding *residuated-def residuated-pair-def*
by (*metis ext eq-refl order.antisym*)

Since the residual of a map f is unique, it makes sense to define a residual operator.

definition *residual* :: $('a \Rightarrow 'a) \Rightarrow ('a \Rightarrow 'a)$ **where**
residual f $\equiv$ *THE g. residuated-pair f g*

lemma *residual-galois*: *residuated f* $\Longrightarrow f(x) \leq y \longleftrightarrow x \leq$ *residual f y*
apply (*clarsimp simp*: *residuated-def residuated-pair-def*)
apply (*subgoal-tac residual f* $= g$)
apply (*simp-all add*: *residual-def*)
apply (*rule the1-equality*)
apply (*metis residuated-def residuated-pair-def residual-unique*)
by (*simp add*: *residuated-pair-def*)

lemma *residual-galoisI1*: *residuated f* $\Longrightarrow f(x) \leq y \Longrightarrow x \leq$ *residual f y*
by (*metis residual-galois*)

lemma *residual-galoisI2*: *residuated f* $\Longrightarrow x \leq$ *residual f y* $\Longrightarrow f(x) \leq y$
by (*metis residual-galois*)

A closure operator on a poset is a map that is expansive, isotone and idempotent. The composition of the residual of a function f with f is a closure operator.

definition *closure* :: $('a \Rightarrow 'a) \Rightarrow bool$ **where**
closure f $\equiv (\forall\, x.\ x \leq f\ x) \wedge (\forall\, x\ y.\ x \leq y \longrightarrow f\ x \leq f\ y) \wedge (\forall\, x.\ f(f\ x) = f\ x)$

lemma *res-c1*: *residuated f* $\Longrightarrow x \leq$ *residual f* $(f\ x)$
by (*metis local.order.refl residual-galois*)

lemma *res-c2*: *residuated f* $\Longrightarrow$ $x \leq y \Longrightarrow$ *residual f* (*f x*) $\leq$ *residual f* (*f y*)
by (*metis local.order.refl local.order-trans residual-galois*)

lemma *res-c3*: *residuated f* $\Longrightarrow$ *residual f* (*f* (*residual f* (*f x*))) = *residual f* (*f x*)
by (*metis order.eq-iff local.order-trans res-c1 residual-galois*)

lemma *res-closure*: *residuated f* $\Longrightarrow$ *closure* (*residual f o f*)
by (*auto simp*: *closure-def intro*: *res-c1 res-c2 res-c3*)

Dually, an interior operator on a poset is a map that is contractive, isotone and idempotent. The composition of f with its residual is an interior operator.

definition *interior* :: ('*a* $\Rightarrow$ '*a*) $\Rightarrow$ *bool* **where**
interior f $\equiv$ ($\forall x.\ f\ x \leq x$) $\wedge$ ($\forall x\ y.\ x \leq y \longrightarrow f\ x \leq f\ y$) $\wedge$ ($\forall x.\ f(f\ x) = f\ x$)

lemma *res-i1*: *residuated f* $\Longrightarrow$ *f* (*residual f x*) $\leq$ *x*
by (*metis local.order.refl residual-galois*)

lemma *res-i2*: *residuated f* $\Longrightarrow$ $x \leq y \Longrightarrow$ *f* (*residual f x*) $\leq$ *f* (*residual f y*)
by (*metis local.order.refl local.order-trans residual-galois*)

lemma *res-i3*: *residuated f* $\Longrightarrow$ *f* (*residual f* (*f* (*residual f x*))) = *f* (*residual f x*)
by (*metis local.antisym-conv res-c1 res-c3 res-i1 residual-galois*)

lemma *res-interior*: *residuated f* $\Longrightarrow$ *interior* (*f o residual f*)
by (*auto simp*: *interior-def intro*: *res-i1 res-i2 res-i3*)

Next we show a few basic lemmas about residuated maps.

lemma *res-iso*: *residuated f* $\Longrightarrow$ $x \leq y \Longrightarrow$ *f x* $\leq$ *f y*
by (*metis local.order.trans res-c1 residual-galois*)

lemma *res-residual-iso*: *residuated f* $\Longrightarrow$ $x \leq y \Longrightarrow$ *residual f x* $\leq$ *residual f y*
by (*metis local.order.trans res-i1 residual-galois*)

lemma *res-comp1* [*simp*]: *residuated f* $\Longrightarrow$ *f o residual f o f* = *f*
proof −
assume *resf*: *residuated f*
{
fix *x*
have (*f o residual f o f*) *x* = *f x*
by (*metis resf comp-apply order.eq-iff res-c1 res-i1 res-iso*)
}
thus *?thesis*
by (*metis ext*)
qed

lemma *res-comp2* [*simp*]: *residuated f* $\Longrightarrow$ *residual f o f o residual f* = *residual f*
proof −
assume *resf*: *residuated f*

```
  {
    fix x
    have (residual f o f o residual f) x = residual f x
      by (metis resf comp-apply order.eq-iff res-c1 res-i1 res-residual-iso)
  }
  thus ?thesis
    by (metis ext)
qed

end
```

A residuated function f preserves all existing joins. Dually, its residual preserves all existing meets. We restrict our attention to semilattices, where binary joins or meets exist, and to complete lattices, where arbitrary joins and meets exist.

```
lemma (in semilattice-sup) residuated-sup: residuated f ⟹ f (x ⊔ y) = f x ⊔ f y
proof (rule order.antisym)
  assume assm: residuated f
  thus f (x ⊔ y) ≤ f x ⊔ f y
    by (metis local.residual-galoisI1 local.residual-galoisI2 local.sup.bounded-iff lo-
cal.sup-ge1 local.sup-ge2)
  show f x ⊔ f y ≤ f (x ⊔ y)
    by (metis assm local.res-iso local.sup.bounded-iff local.sup-ge1 local.sup-ge2)
qed

lemma (in semilattice-inf) residuated-inf: residuated f ⟹ residual f (x ⊓ y) =
residual f x ⊓ residual f y
proof (rule order.antisym)
  assume assm: residuated f
  thus residual f (x ⊓ y) ≤ residual f x ⊓ residual f y
   by (metis local.inf.boundedI local.inf.cobounded1 local.inf.cobounded2 local.res-residual-iso)
  show residual f x ⊓ residual f y ≤ residual f (x ⊓ y)
    by (metis assm local.inf.bounded-iff local.inf.cobounded1 local.inf.cobounded2
local.residual-galoisI1 local.residual-galoisI2)
qed

context bounded-lattice begin

lemma residuated-strict: residuated f ⟹ f ⊥ = ⊥
  by (metis local.bot-least local.bot-unique local.res-i1 local.res-iso)

lemma res-top: residuated f ⟹ residual f ⊤ = ⊤
  by (metis local.residual-galoisI1 local.top-greatest local.top-unique)

end

context complete-lattice begin
```

On a complete lattice, a function f is residuated if and only if it preserves

arbitrary (possibly infinite) joins. Dually, a function g is a residual of a residuated funtion f if and only if g preserver arbitrary meets.

lemma *residual-eq1*: *residuated f* $\Longrightarrow$ *residual f y* $= \bigsqcup \{x.\ f\ x \leq y\}$
proof (*rule order.antisym*)
  assume *assm*: *residuated f*
  thus *residual f y* $\leq \bigsqcup \{x.\ f\ x \leq y\}$
    by (*auto simp*: *res-i1 intro*!: *Sup-upper*)
  show $\bigsqcup \{x.\ f\ x \leq y\} \leq$ *residual f y*
    by (*auto simp*: *assm intro*!: *Sup-least residual-galoisI1*)
qed

lemma *residual-eq2*: *residuated f* $\Longrightarrow$ *f x* $= \bigsqcap \{y.\ x \leq$ *residual f y*$\}$
proof (*rule order.antisym*)
  assume *assm*: *residuated f*
  thus *f x* $\leq \bigsqcap \{y.\ x \leq$ *residual f y*$\}$
    by (*auto intro*: *Inf-greatest residual-galoisI2*)
  show $\bigsqcap \{y.\ x \leq$ *residual f y*$\} \leq$ *f x*
    using *assm* **by** (*auto simp*: *res-c1 intro*!: *Inf-lower*)
qed

lemma *residuated-Sup*: *residuated f* $\Longrightarrow$ *f*$(\bigsqcup\ X) = \bigsqcup \{f\ x \mid x.\ x \in X\}$
proof (*rule order.antisym*)
  assume *assm*: *residuated f*
  obtain *y* **where** *y-def*: $y = \bigsqcup \{f\ x \mid x.\ x \in X\}$
    by *auto*
  hence $\forall\, x \in X.\ f\ x \leq y$
    by (*auto simp*: *y-def intro*: *Sup-upper*)
  hence $\forall\, x \in X.\ x \leq$ *residual f y*
    by (*auto simp*: *assm intro*!: *residual-galoisI1*)
  hence $\bigsqcup X \leq$ *residual f y*
    by (*auto intro*: *Sup-least*)
  thus *f*$(\bigsqcup\ X) \leq \bigsqcup \{f\ x \mid x.\ x \in X\}$
    by (*metis y-def assm residual-galoisI2*)
qed (*clarsimp intro*!: *Sup-least res-iso Sup-upper*)

lemma *residuated-Inf*: *residuated f* $\Longrightarrow$ *residual f*$(\bigsqcap\ X) = \bigsqcap \{$*residual f x* $\mid x.\ x \in X\}$
proof (*rule order.antisym*, *clarsimp intro*!: *Inf-greatest res-residual-iso Inf-lower*)
  assume *assm*: *residuated f*
  obtain *y* **where** *y-def*: $y = \bigsqcap \{$*residual f x* $\mid x.\ x \in X\}$
    by *auto*
  hence $\forall\, x \in X.\ y \leq$ *residual f x*
    by (*auto simp*: *y-def intro*: *Inf-lower*)
  hence $\forall\, x \in X.\ f\ y \leq x$
    by (*metis assm residual-galoisI2*)
  hence *f y* $\leq \bigsqcap\ X$
    by (*auto intro*: *Inf-greatest*)
  thus $\bigsqcap \{$*residual f x* $\mid x.\ x \in X\} \leq$ *residual f* $(\bigsqcap X)$
    by (*auto simp*: *assm y-def intro*!: *residual-galoisI1*)

qed

lemma *Sup-sup*: $\forall X.\ f(\bigsqcup\ X) = \bigsqcup\{f\ x \mid x.\ x \in X\} \Longrightarrow f\ (x \sqcup y) = f\ x \sqcup f\ y$
apply (*erule-tac* x=$\{x,\ y\}$ **in** *allE*)
by (*force intro*: *Sup-eqI*)

lemma *Sup-residuatedI*: $\forall X.\ f(\bigsqcup\ X) = \bigsqcup\{f\ x \mid x.\ x \in X\} \Longrightarrow$ *residuated f*
proof (*unfold residuated-def residuated-pair-def*, *standard*+)
fix $x\ y$
assume $f\ x \leq y$
thus $x \leq \bigsqcup\{x.\ f\ x \leq y\}$
by (*clarsimp intro*!: *Sup-upper*)
next
fix $x\ y$
assume *assm*: $\forall X.\ f\ (\bigsqcup X) = \bigsqcup\{f\ x\ |x.\ x \in X\}$
hence *f-iso*: $\forall x\ y.\ x \leq y \longrightarrow f\ x \leq f\ y$
using *Sup-sup* **by** (*auto simp*: *le-iff-sup*)
assume $x \leq \bigsqcup\{x.\ f\ x \leq y\}$
hence $f\ x \leq f(\bigsqcup\{x.\ f\ x \leq y\})$
by (*metis f-iso*)
also have ... $= \bigsqcup\{f\ x \mid x\ .\ f\ x \leq y\}$
using *assm* **by** *auto*
finally show $f\ x \leq y$
apply (*rule order-trans*)
by (*auto intro*!: *Sup-least*)
qed

lemma *Inf-inf*: $\forall X.\ f(\bigsqcap\ X) = \bigsqcap\{f\ x \mid x.\ x \in X\} \Longrightarrow f\ (x \sqcap y) = f\ x \sqcap f\ y$
apply (*erule-tac* x=$\{x,\ y\}$ **in** *allE*)
by (*force intro*: *Inf-eqI*)

lemma *Inf-residuatedI*: $\forall X.\ \bigsqcap\{g\ x \mid x.\ x \in X\} = g\ (\bigsqcap\ X) \Longrightarrow \exists f.$ *residuated-pair f g*
proof (*unfold residuated-pair-def*, *standard*+)
fix $x\ y$
assume $x \leq g\ y$
thus $\bigsqcap\{y.\ x \leq g\ y\} \leq y$
by (*clarsimp intro*!: *Inf-lower*)
next
fix $x\ y$
assume *assm*: $\forall X.\ \bigsqcap\{g\ x \mid x.\ x \in X\} = g\ (\bigsqcap\ X)$
hence *g-iso*: $\forall x\ y.\ x \leq y \longrightarrow g\ x \leq g\ y$
using *Inf-inf* **by** (*auto simp*: *le-iff-inf*)
assume $\bigsqcap\{y.\ x \leq g\ y\} \leq y$
hence $g\ (\bigsqcap\{y.\ x \leq g\ y\}) \leq g\ y$
by (*metis g-iso*)
hence $(\bigsqcap\{g\ y \mid y.\ x \leq g\ y\}) \leq g\ y$
using *assm* **apply** (*erule-tac* x=$\{y.\ x \leq g\ y\}$ **in** *allE*)
by (*auto intro*!: *Inf-lower*)

thus $x \leq g\ y$
apply (*rule local.dual-order.trans*)
by (*auto intro*: *Inf-greatest*)
qed

end

2.2 Residuated Structures

In this section, we define residuated algebraic structures, starting from the simplest of all, a *residuated partial ordered groupoid*, to *residuated l-monoids*, which are residuated lattices where the multiplicative operator forms a monoid.

class *pogroupoid* = *order* + *times* +
assumes *mult-isor*: $x \leq y \Longrightarrow x \cdot z \leq y \cdot z$
and *mult-isol*: $x \leq y \Longrightarrow z \cdot x \leq z \cdot y$

A residuated partial ordered groupoid is simply a partial order and a multiplicative groupoid with two extra operators satisfying the residuation laws. It is straighforward to prove that multiplication is compatible with the order, that is, multiplication is isotone.

Most of the lemmas below come in pairs; they are related by opposition duality. Formalising this duality is left for future work.

class *residual-r-op* =
fixes *residual-r* :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixr** $\leftarrow$ *60*)

class *residual-l-op* =
fixes *residual-l* :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixl** $\rightarrow$ *60*)

class *residuated-pogroupoid* = *order* + *times* + *residual-l-op* + *residual-r-op* +
assumes *resl-galois*: $x \leq z \leftarrow y \longleftrightarrow x \cdot y \leq z$
and *resr-galois*: $x \cdot y \leq z \longleftrightarrow y \leq x \rightarrow z$
begin

lemma *reslI* [*intro*]: $x \cdot y \leq z \Longrightarrow x \leq z \leftarrow y$
by (*metis resl-galois*)

lemma *resrI* [*intro*]: $x \cdot y \leq z \Longrightarrow y \leq x \rightarrow z$
by (*metis resr-galois*)

lemma *residuated-pair-multl* [*simp*]: *residuated-pair* $(\lambda x.\ x \cdot y)$ $(\lambda x.\ x \leftarrow y)$
by (*auto simp*: *residuated-pair-def resl-galois*)

lemma *residuated-pair-multr* [*simp*]: *residuated-pair* $(\lambda y.\ x \cdot y)$ $(\lambda y.\ x \rightarrow y)$
by (*auto simp*: *residuated-pair-def resr-galois*)

Multiplication is then obviously residuated.

lemma *residuated-multl* [*simp*]: *residuated* $(\lambda x.\ x \cdot y)$
by (*metis residuated-def residuated-pair-multl*)

lemma *residuated-multr* [*simp*]: *residuated* $(\lambda y.\ x \cdot y)$
by (*metis residuated-def residuated-pair-multr*)

lemma *resl-eq* [*simp*]: *residual* $(\lambda x.\ x \cdot y) = (\lambda x.\ x \leftarrow y)$
unfolding *residual-def* **apply** (*rule the1-equality*)
by (*auto simp*: *intro*!: *residual-unique*)

lemma *resr-eq* [*simp*]: *residual* $(\lambda y.\ x \cdot y) = (\lambda y.\ x \rightarrow y)$
unfolding *residual-def* **apply** (*rule the1-equality*)
by (*auto simp*: *intro*!: *residual-unique*)

Next we prove a few lemmas, all of which are instantiation of more general facts about residuated functions.

lemma *res-lc1*: $x \leq x \cdot y \leftarrow y$
by *auto*

lemma *res-lc2*: $x \leq y \Longrightarrow x \cdot z \leftarrow z \leq y \cdot z \leftarrow z$
by (*metis local.res-c2 resl-eq residuated-multl*)

lemma *res-lc3* [*simp*]: $(x \cdot y \leftarrow y) \cdot y \leftarrow y = x \cdot y \leftarrow y$
by (*metis local.res-c3 resl-eq residuated-multl*)

lemma *res-rc1*: $x \leq y \rightarrow y \cdot x$
by *auto*

lemma *res-rc2*: $x \leq y \Longrightarrow z \rightarrow z \cdot x \leq z \rightarrow z \cdot y$
by (*metis local.res-c2 resr-eq residuated-multr*)

lemma *res-rc3* [*simp*]: $y \rightarrow y \cdot (y \rightarrow y \cdot x) = y \rightarrow y \cdot x$
by (*metis local.res-c3 resr-eq residuated-multr*)

lemma *res-li1*: $(x \leftarrow y) \cdot y \leq x$
by (*metis local.res-i1 resl-eq residuated-multl*)

lemma *res-li2*: $x \leq y \Longrightarrow (x \leftarrow z) \cdot z \leq (y \leftarrow z) \cdot z$
by (*metis local.res-i2 resl-eq residuated-multl*)

lemma *res-li3* [*simp*]: $((x \leftarrow y) \cdot y \leftarrow y) \cdot y = (x \leftarrow y) \cdot y$
by (*metis local.res-i3 resl-eq residuated-multl*)

lemma *res-ri1*: $y \cdot (y \rightarrow x) \leq x$
by (*metis local.res-i1 resr-eq residuated-multr*)

lemma *res-ri2*: $x \leq y \Longrightarrow z \cdot (z \rightarrow x) \leq z \cdot (z \rightarrow y)$
by (*metis local.res-i2 resr-eq residuated-multr*)

lemma *res-ri3* [*simp*]: $y \cdot (y \rightarrow y \cdot (y \rightarrow x)) = y \cdot (y \rightarrow x)$
by (*metis local.res-i3 resr-eq residuated-multr*)

subclass *pogroupoid*
proof
fix $x\ y\ z$
show $x \leq y \Longrightarrow x \cdot z \leq y \cdot z$
by (*metis local.res-iso residuated-multl*)
show $x \leq y \Longrightarrow z \cdot x \leq z \cdot y$
by (*metis local.res-iso residuated-multr*)
qed

lemma *resl-iso*: $x \leq y \Longrightarrow x \leftarrow z \leq y \leftarrow z$
by (*metis res-residual-iso resl-eq residuated-multl*)

lemma *resr-iso*: $x \leq y \Longrightarrow z \rightarrow x \leq z \rightarrow y$
by (*metis res-residual-iso resr-eq residuated-multr*)

lemma *resl-comp1* [*simp*]: $(x \cdot y \leftarrow y) \cdot y = x \cdot y$
by (*metis order.antisym local.mult-isor res-lc1 res-li1*)

lemma *resl-comp2* [*simp*]: $(x \leftarrow y) \cdot y \leftarrow y = x \leftarrow y$
by (*metis order.eq-iff res-lc1 res-li1 resl-iso*)

lemma *resr-comp1* [*simp*]: $y \cdot (y \rightarrow y \cdot x) = y \cdot x$
by (*metis order.antisym local.mult-isol res-rc1 res-ri1*)

lemma *resr-comp2* [*simp*]: $y \rightarrow y \cdot (y \rightarrow x) = y \rightarrow x$
by (*metis order.eq-iff res-rc1 res-ri1 resr-iso*)

lemma *resl-antitoner*: $x \leq y \longrightarrow z \leftarrow y \leq z \leftarrow x$
by (*metis local.dual-order.trans local.mult-isol res-li1 reslI*)

lemma *resr-antitonel*: $x \leq y \longrightarrow y \rightarrow z \leq x \rightarrow z$
by (*metis local.dual-order.trans local.resl-galois res-ri1 resrI*)

The following lemmas are taken from Galatos and *al.*

lemma *jipsen1l*: $x \leq y \leftarrow (x \rightarrow y)$
by (*metis res-ri1 reslI*)

lemma *jipsen1r*: $x \leq (y \leftarrow x) \rightarrow y$
by (*metis res-li1 resrI*)

lemma *jipsen2l*: $(y \leftarrow (x \rightarrow y)) \rightarrow y = x \rightarrow y$
by (*metis jipsen1l jipsen1r order.eq-iff local.resr-antitonel*)

lemma *jipsen2r*: $y \leftarrow ((y \leftarrow x) \rightarrow y) = y \leftarrow x$
by (*metis jipsen1l jipsen1r order.eq-iff local.resl-antitoner*)

end

In a residuated partial ordered semigroup, the multiplicative operator is associative.

class *residuated-posemigroup* = *semigroup-mult* + *residuated-pogroupoid*
begin

lemma *resl-trans*: $(x \leftarrow y) \cdot (y \leftarrow z) \leq x \leftarrow z$
by (*metis local.res-li1 local.resl-antitoner local.resl-galois mult-assoc*)

lemma *resr-trans*: $(x \rightarrow y) \cdot (y \rightarrow z) \leq x \rightarrow z$
by (*metis local.res-ri1 local.resr-antitonel local.resr-galois mult-assoc*)

lemma *resr1*: $(x \rightarrow y) \cdot z \leq (x \rightarrow y \cdot z)$
by (*metis local.mult-isor local.res-ri1 local.resrI mult-assoc*)

lemma *resr2*: $x \rightarrow y \leq z \cdot x \rightarrow z \cdot y$
by (*metis local.mult-isol local.res-ri1 local.resrI mult-assoc*)

lemma *resr3*: $x \cdot y \rightarrow z = y \rightarrow (x \rightarrow z)$
by (*metis order.eq-iff local.resr-galois mult-assoc*)

lemma *resl1*: $z \cdot (x \leftarrow y) \leq (z \cdot x \leftarrow y)$
by (*metis local.mult-isol local.res-li1 local.reslI mult-assoc*)

lemma *resl2*: $x \leftarrow y \leq x \cdot z \leftarrow y \cdot z$
by (*metis local.mult-isor local.res-li1 local.reslI mult-assoc*)

lemma *resl3*: $x \leftarrow y \cdot z = (x \leftarrow z) \leftarrow y$
by (*metis order.eq-iff local.resl-galois mult-assoc*)

lemma *residual-assoc*: $x \rightarrow (y \leftarrow z) = (x \rightarrow y) \leftarrow z$
proof (*rule order.antisym*)
show $x \rightarrow (y \leftarrow z) \leq (x \rightarrow y) \leftarrow z$
by (*metis local.res-ri1 local.resl-galois local.resr-galois mult-assoc*)
show $(x \rightarrow y) \leftarrow z \leq x \rightarrow (y \leftarrow z)$
by (*metis local.res-li1 local.resl-galois local.resr-galois mult-assoc*)
qed

end

In a residuated partially ordered monoid, the multiplicative operator has a unit 1; that is, its reduct forms a monoid.

class *residuated-pomonoid* = *monoid-mult* + *residuated-pogroupoid*
begin

subclass *residuated-posemigroup* **..**

lemma *resl-unit*: $x \leftarrow 1 = x$

by (*metis local.mult-1-right local.resl-comp1*)

lemma *resr-unit*: $1 \rightarrow x = x$
by (*metis local.mult-1-left local.resr-comp1*)

lemma *mult-one-resl*: $x \cdot (1 \leftarrow y) \leq x \leftarrow y$
by (*metis local.mult-1-right local.resl1*)

lemma *mult-one-resr*: $(x \rightarrow 1) \cdot y \leq x \rightarrow y$
by (*metis local.mult-1-left local.resr1*)

More lemmas from Galatos and *al.* follow.

lemma *jipsen3l*: $1 \leq x \leftarrow x$
by (*metis local.jipsen1l resr-unit*)

lemma *jipsen3r*: $1 \leq x \rightarrow x$
by (*metis local.jipsen1r resl-unit*)

lemma *jipsen4l* [*simp*]: $(x \leftarrow x) \cdot x = x$
by (*metis local.mult-1-left local.resl-comp1*)

lemma *jipsen4r* [*simp*]: $x \cdot (x \rightarrow x) = x$
by (*metis local.mult-1-right local.resr-comp1*)

lemma *jipsen5l* [*simp*]: $(x \leftarrow x) \cdot (x \leftarrow x) = x \leftarrow x$
by (*metis jipsen4l local.resl3*)

lemma *jipsen5r* [*simp*]: $(x \rightarrow x) \cdot (x \rightarrow x) = x \rightarrow x$
by (*metis jipsen4r local.resr3*)

lemma *jipsen6l*: $y \leftarrow x \leq (y \leftarrow z) \leftarrow (x \leftarrow z)$
by (*metis local.resl-galois local.resl-trans*)

lemma *jipsen6r*: $x \rightarrow y \leq (z \rightarrow x) \rightarrow (z \rightarrow y)$
by (*metis local.resr-galois local.resr-trans*)

lemma *jipsen7l*: $y \leftarrow x \leq (z \leftarrow y) \rightarrow (z \leftarrow x)$
by (*metis local.resr-galois local.resl-trans*)

lemma *jipsen7r*: $x \rightarrow y \leq (x \rightarrow z) \leftarrow (y \rightarrow z)$
by (*metis local.resl-galois local.resr-trans*)

end

Residuated partially ordered groupoids can be expanded residuated join semilattice ordered groupoids. They are used as a base for action algebras, which are expansions of Kleene algebras by operations of residuation. Action algebras can be found in the AFP entry for Kleene algebras.

class *residuated-sup-lgroupoid* = *semilattice-sup* + *residuated-pogroupoid*

begin

lemma *distl*: $x \cdot (y \sqcup z) = x \cdot y \sqcup x \cdot z$
by (*metis local.residuated-multr local.residuated-sup*)

lemma *distr*: $(x \sqcup y) \cdot z = x \cdot z \sqcup y \cdot z$
by (*metis local.residuated-multl local.residuated-sup*)

lemma *resl-subdist-var*: $x \leftarrow z \leq (x \sqcup y) \leftarrow z$
by (*metis local.resl-iso local.sup-ge1*)

lemma *resl-subdist*: $(x \leftarrow z) \sqcup (y \leftarrow z) \leq (x \sqcup y) \leftarrow z$
by (*metis local.le-sup-iff local.sup-commute resl-subdist-var*)

lemma *resr-subdist-var*: $(x \rightarrow y) \leq x \rightarrow (y \sqcup z)$
by (*metis local.resr-iso local.sup-ge1*)

lemma *resr-subdist*: $(x \rightarrow y) \sqcup (x \rightarrow z) \leq x \rightarrow (y \sqcup z)$
by (*metis sup-commute sup-least resr-subdist-var*)

lemma *resl-superdist-var*: $x \leftarrow (y \sqcup z) \leq x \leftarrow y$
by (*metis local.le-sup-iff local.res-li1 local.reslI local.resr-galois*)

lemma *resr-superdist-var*: $(x \sqcup y) \rightarrow z \leq x \rightarrow z$
by (*metis local.le-sup-iff local.res-ri1 local.resl-galois local.resrI*)

end

Similarly, semigroup and monoid variants can be defined.

class *residuated-sup-lsemigroup* = *semilattice-sup* + *residuated-posemigroup*
subclass (**in** *residuated-sup-lsemigroup*) *residuated-sup-lgroupoid* **..**

class *residuated-sup-lmonoid* = *semilattice-sup* + *residuated-posemigroup*
subclass (**in** *residuated-sup-lmonoid*) *residuated-sup-lsemigroup* **..**

By lattice duality, we define residuated meet semillatice ordered groupoid.

class *residuated-inf-lgroupoid* = *semilattice-inf* + *residuated-pogroupoid*
begin

lemma *resl-dist*: $(x \sqcap y) \leftarrow z = (x \leftarrow z) \sqcap (y \leftarrow z)$
by (*metis local.resl-eq local.residuated-inf local.residuated-multl*)

lemma *resr-dist*: $x \rightarrow (y \sqcap z) = (x \rightarrow y) \sqcap (x \rightarrow z)$
by (*metis local.resr-eq local.residuated-inf local.residuated-multr*)

lemma *resl-inf-superdist*: $x \leftarrow y \leq x \leftarrow (y \sqcap z)$
by (*metis local.inf-le1 local.resl-antitoner*)

lemma *resr-inf-superdist-var*: $x \rightarrow y \leq (x \sqcap z) \rightarrow y$

by (*metis local.inf-le1 local.resr-antitonel*)

end

class *residuated-inf-lsemigroup* = *semilattice-inf* + *residuated-posemigroup*
subclass (**in** *residuated-inf-lsemigroup*) *residuated-inf-lgroupoid* **..**

class *residuated-inf-lmonoid* = *semilattice-inf* + *residuated-posemigroup*
subclass (**in** *residuated-inf-lmonoid*) *residuated-inf-lsemigroup* **..**

Finally, we obtain residuated lattice groupoids.

class *residuated-lgroupoid* = *lattice* + *residuated-pogroupoid*
begin

subclass *residuated-sup-lgroupoid* **..**

lemma *resl-distr*: $z \leftarrow (x \sqcup y) = (z \leftarrow x) \sqcap (z \leftarrow y)$
proof (*rule order.antisym*)
show $z \leftarrow x \sqcup y \leq (z \leftarrow x) \sqcap (z \leftarrow y)$
by (*metis local.inf.bounded-iff local.resl-superdist-var local.sup-commute*)
show $(z \leftarrow x) \sqcap (z \leftarrow y) \leq z \leftarrow x \sqcup y$
by (*metis local.inf.cobounded1 local.inf.cobounded2 local.resl-galois local.resr-galois local.sup.bounded-iff*)
qed

lemma *resr-distl*: $(x \sqcup y) \rightarrow z = (x \rightarrow z) \sqcap (y \rightarrow z)$
proof (*rule order.antisym*)
show $x \sqcup y \rightarrow z \leq (x \rightarrow z) \sqcap (y \rightarrow z)$
by (*metis local.inf.bounded-iff local.resr-antitonel local.resr-superdist-var local.sup-ge2*)
show $(x \rightarrow z) \sqcap (y \rightarrow z) \leq x \sqcup y \rightarrow z$
by (*metis local.inf.cobounded1 local.inf.cobounded2 local.resl-galois local.resr-galois local.sup-least*)
qed

end

class *residuated-lsemigroup* = *lattice* + *residuated-sup-lsemigroup*
subclass (**in** *residuated-lsemigroup*) *residuated-lgroupoid* **..**

class *residuated-lmonoid* = *lattice* + *residuated-sup-lmonoid*
subclass (**in** *residuated-lmonoid*) *residuated-lsemigroup* **..**

class *residuated-blgroupoid* = *bounded-lattice* + *residuated-pogroupoid*
begin

lemma *multl-strict* [*simp*]: $x \cdot \bot = \bot$
by (*metis local.residuated-multr local.residuated-strict*)

lemma *multr-strict* [*simp*]: $\bot \cdot x = \bot$
by (*metis local.residuated-multl local.residuated-strict*)

lemma *resl-top* [*simp*]: $\top \leftarrow x = \top$
by (*metis local.res-top local.residuated-multl local.resl-eq*)

lemma *resl-impl* [*simp*]: $x \leftarrow \bot = \top$
by (*metis local.resl-comp2 multl-strict resl-top*)

lemma *resr-top* [*simp*]: $x \rightarrow \top = \top$
by (*metis local.resrI local.top-greatest local.top-unique*)

lemma *resr-impl* [*simp*]: $\bot \rightarrow x = \top$
by (*metis local.resr-comp2 multr-strict resr-top*)

end

class *residuated-blsemigroup* = *bounded-lattice* + *residuated-sup-lsemigroup*
subclass (**in** *residuated-blsemigroup*) *residuated-blgroupoid* **..**

class *residuated-blmonoid* = *bounded-lattice* + *residuated-sup-lmonoid*
subclass (**in** *residuated-blmonoid*) *residuated-blsemigroup* **..**

class *residuated-clgroupoid* = *complete-lattice* + *residuated-pogroupoid*
begin

lemma *resl-eq-def*: $y \leftarrow x = \bigsqcup \{z.\ z \cdot x \leq y\}$
by (*metis local.residual-eq1 local.residuated-multl local.resl-eq*)

lemma *resr-eq-def*: $x \rightarrow y = \bigsqcup \{z.\ x \cdot z \leq y\}$
by (*metis local.residual-eq1 local.residuated-multr local.resr-eq*)

lemma *multl-def*: $x \cdot y = \bigsqcap \{z.\ x \leq z \leftarrow y\}$
proof −
have $\bigsqcap \{ya.\ x \leq residual\ (\lambda a.\ a \cdot y)\ ya\} = \bigsqcap \{z.\ x \leq z \leftarrow y\}$
by *simp*
thus *?thesis*
by (*metis residuated-multl residual-eq2*)
qed

lemma *multr-def*: $x \cdot y = \bigsqcap \{z.\ y \leq x \rightarrow z\}$
proof −
have $\bigsqcap \{ya.\ y \leq residual\ ((\cdot)\ x)\ ya\} = \bigsqcap \{z.\ y \leq x \rightarrow z\}$
by *simp*
thus *?thesis*
by (*metis residuated-multr residual-eq2*)
qed

end

class *residuated-clsemigroup* = *complete-lattice* + *residuated-sup-lsemigroup*
subclass (**in** *residuated-clsemigroup*) *residuated-clgroupoid* **..**

class *residuated-clmonoid* = *complete-lattice* + *residuated-sup-lmonoid*
subclass (**in** *residuated-clmonoid*) *residuated-clsemigroup* **..**

end

3 Residuated Boolean Algebras

theory *Residuated-Boolean-Algebras*
imports *Residuated-Lattices*
begin

unbundle *lattice-syntax*

3.1 Conjugation on Boolean Algebras

Similarly, as in the previous section, we define the conjugation for arbitrary residuated functions on boolean algebras.

context *boolean-algebra*
begin

lemma *inf-bot-iff-le*: $x \sqcap y = \bot \longleftrightarrow x \leq -y$
by (*metis le-iff-inf inf-sup-distrib1 inf-top-right sup-bot.left-neutral sup-compl-top compl-inf-bot inf.assoc inf-bot-right*)

lemma *le-iff-inf-bot*: $x \leq y \longleftrightarrow x \sqcap -y = \bot$
by (*metis inf-bot-iff-le compl-le-compl-iff inf-commute*)

lemma *indirect-eq*: $(\bigwedge z.\ x \leq z \longleftrightarrow y \leq z) \Longrightarrow x = y$
by (*metis order.eq-iff*)

Let B be a boolean algebra. The maps f and g on B are a pair of conjugates if and only if for all $x, y \in B$, $f(x) \sqcap y = \bot \Leftrightarrow x \sqcap g(t) = \bot$.

definition *conjugation-pair* :: $('a \Rightarrow 'a) \Rightarrow ('a \Rightarrow 'a) \Rightarrow$ *bool* **where**
conjugation-pair f g $\equiv \forall x\ y.\ f(x) \sqcap y = \bot \longleftrightarrow x \sqcap g(y) = \bot$

lemma *conjugation-pair-commute*: *conjugation-pair f g* $\Longrightarrow$ *conjugation-pair g f*
by (*auto simp*: *conjugation-pair-def inf-commute*)

lemma *conjugate-iff-residuated*: *conjugation-pair f g* = *residuated-pair f* $(\lambda x.\ -g(-x))$
apply (*clarsimp simp*: *conjugation-pair-def residuated-pair-def inf-bot-iff-le*)
by (*metis double-compl*)

lemma *conjugate-residuated*: *conjugation-pair f g* $\Longrightarrow$ *residuated-pair f* $(\lambda x.\ -g(-x))$
by (*metis conjugate-iff-residuated*)

lemma *residuated-iff-conjugate*: *residuated-pair f g* = *conjugation-pair f* (λ*x*. −*g*(−*x*))
apply (*clarsimp simp*: *conjugation-pair-def residuated-pair-def inf-bot-iff-le*)
by (*metis double-compl*)

A map f has a conjugate pair if and only if it is residuated.

lemma *conj-residuatedI1*: ∃ *g*. *conjugation-pair f g* ⟹ *residuated f*
by (*metis conjugate-iff-residuated residuated-def*)

lemma *conj-residuatedI2*: ∃ *g*. *conjugation-pair g f* ⟹ *residuated f*
by (*metis conj-residuatedI1 conjugation-pair-commute*)

lemma *exist-conjugateI*[*intro*]: *residuated f* ⟹ ∃ *g*. *conjugation-pair f g*
by (*metis residuated-def residuated-iff-conjugate*)

lemma *exist-conjugateI2*[*intro*]: *residuated f* ⟹ ∃ *g*. *conjugation-pair g f*
by (*metis exist-conjugateI conjugation-pair-commute*)

The conjugate of a residuated function f is unique.

lemma *unique-conjugate*[*intro*]: *residuated f* ⟹ ∃!*g*. *conjugation-pair f g*
proof −
{
fix *g h x* **assume** *conjugation-pair f g* **and** *conjugation-pair f h*
hence *g* = *h*
apply (*unfold conjugation-pair-def*)
apply (*rule ext*)
apply (*rule order.antisym*)
by (*metis le-iff-inf-bot inf-commute inf-compl-bot*)+
}
moreover assume *residuated f*
ultimately show *?thesis* **by** *force*
qed

lemma *unique-conjugate2*[*intro*]: *residuated f* ⟹ ∃!*g*. *conjugation-pair g f*
by (*metis unique-conjugate conjugation-pair-commute*)

Since the conjugate of a residuated map is unique, we define a conjugate operation.

definition *conjugate* :: (′*a* ⇒ ′*a*) ⇒ (′*a* ⇒ ′*a*) **where**
conjugate f ≡ *THE g*. *conjugation-pair g f*

lemma *conjugate-iff-def*: *residuated f* ⟹ *f*(*x*) ⊓ *y* = ⊥ ⟷ *x* ⊓ *conjugate f y* = ⊥
apply (*clarsimp simp*: *conjugate-def dest*!: *unique-conjugate*)
apply (*subgoal-tac* (*THE g*. *conjugation-pair g f*) = *g*)
apply (*clarsimp simp add*: *conjugation-pair-def*)
apply (*rule the1-equality*)
by (*auto intro*: *conjugation-pair-commute*)

lemma *conjugateI1*: *residuated f* $\Longrightarrow$ $f(x) \sqcap y = \bot \Longrightarrow x \sqcap$ *conjugate f y* $= \bot$
by (*metis conjugate-iff-def*)

lemma *conjugateI2*: *residuated f* $\Longrightarrow x \sqcap$ *conjugate f y* $= \bot \Longrightarrow f(x) \sqcap y = \bot$
by (*metis conjugate-iff-def*)

Few more lemmas about conjugation follow.

lemma *residuated-conj1*: *residuated f* $\Longrightarrow$ *conjugation-pair f* (*conjugate f*)
using *conjugateI1 conjugateI2 conjugation-pair-def* **by** *auto*

lemma *residuated-conj2*: *residuated f* $\Longrightarrow$ *conjugation-pair* (*conjugate f*) *f*
using *conjugateI1 conjugateI2 conjugation-pair-def inf-commute* **by** *auto*

lemma *conj-residuated*: *residuated f* $\Longrightarrow$ *residuated* (*conjugate f*)
by (*force dest*!: *residuated-conj2 intro*: *conj-residuatedI1*)

lemma *conj-involution*: *residuated f* $\Longrightarrow$ *conjugate* (*conjugate f*) $= f$
by (*metis conj-residuated residuated-conj1 residuated-conj2 unique-conjugate*)

lemma *residual-conj-eq*: *residuated f* $\Longrightarrow$ *residual* (*conjugate f*) $= (\lambda x.\ -f(-x))$
apply (*unfold residual-def*)
apply (*rule the1-equality*)
apply (*rule residual-unique*)
apply (*auto intro*: *conj-residuated conjugate-residuated residuated-conj2*)
done

lemma *residual-conj-eq-ext*: *residuated f* $\Longrightarrow$ *residual* (*conjugate f*) $x = -f(-x)$
by (*metis residual-conj-eq*)

lemma *conj-iso*: *residuated f* $\Longrightarrow x \leq y \Longrightarrow$ *conjugate f x* $\leq$ *conjugate f y*
by (*metis conj-residuated res-iso*)

lemma *conjugate-strict*: *residuated f* $\Longrightarrow$ *conjugate f* $\bot = \bot$
by (*metis conj-residuated residuated-strict*)

lemma *conjugate-sup*: *residuated f* $\Longrightarrow$ *conjugate f* $(x \sqcup y) =$ *conjugate f x* $\sqcup$ *conjugate f y*
by (*metis conj-residuated residuated-sup*)

lemma *conjugate-subinf*: *residuated f* $\Longrightarrow$ *conjugate f* $(x \sqcap y) \leq$ *conjugate f x* $\sqcap$ *conjugate f y*
by (*auto simp*: *conj-iso*)

Next we prove some lemmas from Maddux's article. Similar lemmas have been proved in AFP entry for relation algebras. They should be consolidated in the future.

lemma *maddux1*: *residuated f* $\Longrightarrow f(x \sqcap -$ *conjugate* $f(y)) \leq f(x) \sqcap -y$
proof −
assume *assm*: *residuated f*

hence $f(x \sqcap - \ conjugate\ f(y)) \leq f\ x$
by (*metis inf-le1 res-iso*)
moreover have $f(x \sqcap - \ conjugate\ f\ (y)) \sqcap y = \bot$
by (*metis assm conjugateI2 inf-bot-iff-le inf-le2*)
ultimately show *?thesis*
by (*metis inf-bot-iff-le le-inf-iff*)
qed

lemma *maddux1'*: *residuated f* $\Longrightarrow$ *conjugate* $f(x \sqcap -f(y)) \leq$ *conjugate* $f(x) \sqcap -y$
by (*metis conj-involution conj-residuated maddux1*)

lemma *maddux2*: *residuated f* $\Longrightarrow f(x) \sqcap y \leq f(x \sqcap$ *conjugate f y*$)$
proof −
assume *resf*: *residuated f*
obtain *z* **where** *z-def*: $z = f(x \sqcap$ *conjugate f y*$)$ **by** *auto*
hence $f(x \sqcap$ *conjugate f y*$) \sqcap -z = \bot$
by (*metis inf-compl-bot*)
hence $x \sqcap$ *conjugate f y* $\sqcap$ *conjugate f* $(-z) = \bot$
by (*metis conjugate-iff-def resf*)
hence $x \sqcap$ *conjugate f* $(y \sqcap -z) = \bot$
apply (*subgoal-tac conjugate f* $(y \sqcap -z) \leq$ *conjugate f y* $\sqcap$ *conjugate f* $(-z)$)
apply (*metis* (*no-types*, *opaque-lifting*) *dual-order.trans inf.commute inf-bot-iff-le inf-left-commute*)
by (*metis conj-iso inf-le2 inf-top.left-neutral le-inf-iff resf*)
hence $f(x) \sqcap y \sqcap -z = \bot$
by (*metis conjugateI2 inf.assoc resf*)
thus *?thesis*
by (*metis double-compl inf-bot-iff-le z-def*)
qed

lemma *maddux2'*: *residuated f* $\Longrightarrow$ *conjugate* $f(x) \sqcap y \leq$ *conjugate* $f(x \sqcap f\ y)$
by (*metis conj-involution conj-residuated maddux2*)

lemma *residuated-conjugate-ineq*: *residuated f* $\Longrightarrow$ *conjugate f x* $\leq y \longleftrightarrow x \leq -f(-y)$
by (*metis conj-residuated residual-galois residual-conj-eq*)

lemma *residuated-comp-closed*: *residuated f* $\Longrightarrow$ *residuated g* $\Longrightarrow$ *residuated* $(f\ o\ g)$
by (*auto simp add*: *residuated-def residuated-pair-def*)

lemma *conjugate-comp*: *residuated f* $\Longrightarrow$ *residuated g* $\Longrightarrow$ *conjugate* $(f\ o\ g) =$ *conjugate g o conjugate f*
proof (*rule ext*, *rule indirect-eq*)
fix *x y*
assume *assms*: *residuated f residuated g*
have *conjugate* $(f\ o\ g)\ x \leq y \longleftrightarrow x \leq -f(g(-y))$
apply (*subst residuated-conjugate-ineq*)

using *assms* **by** (*auto intro*!: *residuated-comp-closed*)
also have ... $\longleftrightarrow$ *conjugate g* (*conjugate f x*) $\leq y$
using *assms* **by** (*simp add*: *residuated-conjugate-ineq*)
finally show (*conjugate* (*f* $\circ$ *g*) *x* $\leq$ *y*) = ((*conjugate g* $\circ$ *conjugate f*) *x* $\leq$ *y*)
by *auto*
qed

lemma *conjugate-comp-ext*: *residuated f* $\Longrightarrow$ *residuated g* $\Longrightarrow$ *conjugate* (λ*x*. *f* (*g x*)) *x* = *conjugate g* (*conjugate f x*)
using *conjugate-comp* **by** (*simp add*: *comp-def*)

end

context *complete-boolean-algebra* **begin**

On a complete boolean algebra, it is possible to give an explicit definition of conjugation.

lemma *conjugate-eq*: *residuated f* $\Longrightarrow$ *conjugate f y* = $\bigsqcap\{x.\ y \leq -f(-x)\}$
proof −
assume *assm*: *residuated f* **obtain** *g* **where** *g-def*: *g* = *conjugate f* **by** *auto*
have *g y* = $\bigsqcap\{x.\ x \geq g\ y\}$
by (*auto intro*!: *order.antisym Inf-lower Inf-greatest*)
also have ... = $\bigsqcap\{x.\ -x \sqcap g\ y = \bot\}$
by (*simp add*: *inf-bot-iff-le*)
also have ... = $\bigsqcap\{x.\ f(-x) \sqcap y = \bot\}$
by (*metis conjugate-iff-def assm g-def*)
finally show *?thesis*
by (*simp add*: *g-def le-iff-inf-bot inf-commute*)
qed

end

3.2 Residuated Boolean Structures

In this section, we present various residuated structures based on boolean algebras. The left and right conjugation of the multiplicative operation is defined, and a number of facts is derived.

class *residuated-boolean-algebra* = *boolean-algebra* + *residuated-pogroupoid*
begin

subclass *residuated-lgroupoid* **..**

definition *conjugate-l* :: *'a* $\Rightarrow$ *'a* $\Rightarrow$ *'a* (**infixl** ‹$\triangleleft$› *60*) **where**
x $\triangleleft$ *y* $\equiv$ −(−*x* $\leftarrow$ *y*)

definition *conjugate-r* :: *'a* $\Rightarrow$ *'a* $\Rightarrow$ *'a* (**infixl** ‹$\triangleright$› *60*) **where**
x $\triangleright$ *y* $\equiv$ −(*x* $\rightarrow$ −*y*)

lemma *residual-conjugate-r*: $x \rightarrow y = -(x \rhd -y)$
by (*metis conjugate-r-def double-compl*)

lemma *residual-conjugate-l*: $x \leftarrow y = -(-x \lhd y)$
by (*metis conjugate-l-def double-compl*)

lemma *conjugation-multl*: $x{\cdot}y \sqcap z = \bot \longleftrightarrow x \sqcap (z \lhd y) = \bot$
by (*metis conjugate-l-def double-compl le-iff-inf-bot resl-galois*)

lemma *conjugation-multr*: $x{\cdot}y \sqcap z = \bot \longleftrightarrow y \sqcap (x \rhd z) = \bot$
by (*metis conjugate-r-def inf-bot-iff-le le-iff-inf-bot resr-galois*)

lemma *conjugation-conj*: $(x \lhd y) \sqcap z = \bot \longleftrightarrow y \sqcap (z \rhd x) = \bot$
by (*metis inf-commute conjugation-multr conjugation-multl*)

lemma *conjugation-pair-multl* [*simp*]: *conjugation-pair* $(\lambda x.\ x{\cdot}y)$ $(\lambda x.\ x \lhd y)$
by (*simp add*: *conjugation-pair-def conjugation-multl*)

lemma *conjugation-pair-multr* [*simp*]: *conjugation-pair* $(\lambda x.\ y{\cdot}x)$ $(\lambda x.\ y \rhd x)$
by (*simp add*: *conjugation-pair-def conjugation-multr*)

lemma *conjugation-pair-conj* [*simp*]: *conjugation-pair* $(\lambda x.\ y \lhd x)$ $(\lambda x.\ x \rhd y)$
by (*simp add*: *conjugation-pair-def conjugation-conj*)

lemma *residuated-conjl1* [*simp*]: *residuated* $(\lambda x.\ x \lhd y)$
by (*metis conj-residuatedI2 conjugation-pair-multl*)

lemma *residuated-conjl2* [*simp*]: *residuated* $(\lambda x.\ y \lhd x)$
by (*metis conj-residuatedI1 conjugation-pair-conj*)

lemma *residuated-conjr1* [*simp*]: *residuated* $(\lambda x.\ y \rhd x)$
by (*metis conj-residuatedI2 conjugation-pair-multr*)

lemma *residuated-conjr2* [*simp*]: *residuated* $(\lambda x.\ x \rhd y)$
by (*metis conj-residuatedI2 conjugation-pair-conj*)

lemma *conjugate-multr* [*simp*]: *conjugate* $(\lambda x.\ y{\cdot}x) = (\lambda x.\ y \rhd x)$
by (*metis conjugation-pair-multr residuated-conj1 residuated-multr unique-conjugate*)

lemma *conjugate-conjr1* [*simp*]: *conjugate* $(\lambda x.\ y \rhd x) = (\lambda x.\ y{\cdot}x)$
by (*metis conjugate-multr conj-involution residuated-multr*)

lemma *conjugate-multl* [*simp*]: *conjugate* $(\lambda x.\ x{\cdot}y) = (\lambda x.\ x \lhd y)$
by (*metis conjugation-pair-multl residuated-conj1 residuated-multl unique-conjugate*)

lemma *conjugate-conjl1* [*simp*]: *conjugate* $(\lambda x.\ x \lhd y) = (\lambda x.\ x{\cdot}y)$
proof −
have *conjugate* (*conjugate* $(\lambda x.\ x{\cdot}y)$) = *conjugate* $(\lambda x.\ x \lhd y)$ **by** *simp*
thus *?thesis*

by (*metis conj-involution*[*OF residuated-multl*])
qed

lemma *conjugate-conjl2*[*simp*]: *conjugate* $(\lambda x.\ y \lhd x) = (\lambda x.\ x \rhd y)$
by (*metis conjugation-pair-conj unique-conjugate residuated-conj1 residuated-conjl2*)

lemma *conjugate-conjr2*[*simp*]: *conjugate* $(\lambda x.\ x \rhd y) = (\lambda x.\ y \lhd x)$
proof −
have *conjugate* (*conjugate* $(\lambda x.\ y \lhd x)$) = *conjugate* $(\lambda x.\ x \rhd y)$ **by** *simp*
thus *?thesis*
by (*metis conj-involution*[*OF residuated-conjl2*])
qed

lemma *conjl1-iso*: $x \leq y \Longrightarrow x \lhd z \leq y \lhd z$
by (*metis conjugate-l-def compl-mono resl-iso*)

lemma *conjl2-iso*: $x \leq y \Longrightarrow z \lhd x \leq z \lhd y$
by (*metis res-iso residuated-conjl2*)

lemma *conjr1-iso*: $x \leq y \Longrightarrow z \rhd x \leq z \rhd y$
by (*metis res-iso residuated-conjr1*)

lemma *conjr2-iso*: $x \leq y \Longrightarrow x \rhd z \leq y \rhd z$
by (*metis conjugate-r-def compl-mono resr-antitonel*)

lemma *conjl1-sup*: $z \lhd (x \sqcup y) = (z \lhd x) \sqcup (z \lhd y)$
by (*metis conjugate-l-def compl-inf resl-distr*)

lemma *conjl2-sup*: $(x \sqcup y) \lhd z = (x \lhd z) \sqcup (y \lhd z)$
by (*metis* (*poly-guards-query*) *residuated-sup residuated-conjl1*)

lemma *conjr1-sup*: $z \rhd (x \sqcup y) = (z \rhd x) \sqcup (z \rhd y)$
by (*metis residuated-sup residuated-conjr1*)

lemma *conjr2-sup*: $(x \sqcup y) \rhd z = (x \rhd z) \sqcup (y \rhd z)$
by (*metis conjugate-r-def compl-inf resr-distl*)

lemma *conjl1-strict*: $\bot \lhd x = \bot$
by (*metis residuated-strict residuated-conjl1*)

lemma *conjl2-strict*: $x \lhd \bot = \bot$
by (*metis residuated-strict residuated-conjl2*)

lemma *conjr1-strict*: $\bot \rhd x = \bot$
by (*metis residuated-strict residuated-conjr2*)

lemma *conjr2-strict*: $x \rhd \bot = \bot$
by (*metis residuated-strict residuated-conjr1*)

lemma *conjl1-iff*: $x \lhd y \leq z \longleftrightarrow x \leq -(-z \cdot y)$
by (*metis conjugate-l-def compl-le-swap1 compl-le-swap2 resl-galois*)

lemma *conjl2-iff*: $x \lhd y \leq z \longleftrightarrow y \leq -(-z \rhd x)$
by (*metis conjl1-iff conjugate-r-def compl-le-swap2 double-compl resr-galois*)

lemma *conjr1-iff*: $x \rhd y \leq z \longleftrightarrow y \leq -(x \cdot -z)$
by (*metis conjugate-r-def compl-le-swap1 double-compl resr-galois*)

lemma *conjr2-iff*: $x \rhd y \leq z \longleftrightarrow x \leq -(y \lhd -z)$
by (*metis conjugation-conj double-compl inf.commute le-iff-inf-bot*)

We apply Maddux's lemmas regarding conjugation of an arbitrary residuated function for each of the 6 functions.

lemma *maddux1a*: $a \cdot (x \sqcap -(a \rhd y)) \leq a \cdot x$
by (*insert maddux1* [*of* $\lambda x.\ a \cdot x$]) *simp*

lemma *maddux1a'*: $a \cdot (x \sqcap -(a \rhd y)) \leq -y$
by (*insert maddux1* [*of* $\lambda x.\ a \cdot x$]) *simp*

lemma *maddux1b*: $(x \sqcap -(y \lhd a)) \cdot a \leq x \cdot a$
by (*insert maddux1* [*of* $\lambda x.\ x \cdot a$]) *simp*

lemma *maddux1b'*: $(x \sqcap -(y \lhd a)) \cdot a \leq -y$
by (*insert maddux1* [*of* $\lambda x.\ x \cdot a$]) *simp*

lemma *maddux1c*: $a \lhd x \sqcap -(y \rhd a) \leq a \lhd x$
by (*insert maddux1* [*of* $\lambda x.\ a \lhd x$]) *simp*

lemma *maddux1c'*: $a \lhd x \sqcap -(y \rhd a) \leq -y$
by (*insert maddux1* [*of* $\lambda x.\ a \lhd x$]) *simp*

lemma *maddux1d*: $a \rhd x \sqcap -(a \cdot y) \leq a \rhd x$
by (*insert maddux1* [*of* $\lambda x.\ a \rhd x$]) *simp*

lemma *maddux1d'*: $a \rhd x \sqcap -(a \cdot y) \leq -y$
by (*insert maddux1* [*of* $\lambda x.\ a \rhd x$]) *simp*

lemma *maddux1e*: $x \sqcap -(y \cdot a) \lhd a \leq x \lhd a$
by (*insert maddux1* [*of* $\lambda x.\ x \lhd a$]) *simp*

lemma *maddux1e'*: $x \sqcap -(y \cdot a) \lhd a \leq -y$
by (*insert maddux1* [*of* $\lambda x.\ x \lhd a$]) *simp*

lemma *maddux1f*: $x \sqcap -(a \lhd y) \rhd a \leq x \rhd a$
by (*insert maddux1* [*of* $\lambda x.\ x \rhd a$]) *simp*

lemma *maddux1f'*: $x \sqcap -(a \lhd y) \rhd a \leq -y$
by (*insert maddux1* [*of* $\lambda x.\ x \rhd a$]) *simp*

lemma *maddux2a*: $a{\cdot}x \sqcap y \leq a{\cdot}(x \sqcap (a \rhd y))$
by (*insert maddux2* [*of* $\lambda x.\ a{\cdot}x$]) *simp*

lemma *maddux2b*: $x{\cdot}a \sqcap y \leq (x \sqcap (y \lhd a)){\cdot}a$
by (*insert maddux2* [*of* $\lambda x.\ x{\cdot}a$]) *simp*

lemma *maddux2c*: $(a \lhd x) \sqcap y \leq a \lhd (x \sqcap (y \rhd a))$
by (*insert maddux2* [*of* $\lambda x.\ a \lhd x$]) *simp*

lemma *maddux2d*: $(a \rhd x) \sqcap y \leq a \rhd (x \sqcap (a{\cdot}y))$
by (*insert maddux2* [*of* $\lambda x.\ a \rhd x$]) *simp*

lemma *maddux2e*: $(x \lhd a) \sqcap y \leq (x \sqcap (y{\cdot}a)) \lhd a$
by (*insert maddux2* [*of* $\lambda x.\ x \lhd a$]) *simp*

lemma *maddux2f*: $(x \rhd a) \sqcap y \leq (x \sqcap (a \lhd y)) \rhd a$
by (*insert maddux2* [*of* $\lambda x.\ x \rhd a$]) *simp*

The multiplicative operation $\cdot$ on a residuated boolean algebra is generally not associative. We prove some equivalences related to associativity.

lemma *res-assoc-iff1*: $(\forall\, x\ y\ z.\ x{\cdot}(y{\cdot}z) = (x{\cdot}y){\cdot}z) \longleftrightarrow (\forall\, x\ y\ z.\ x \rhd (y \rhd z) = y{\cdot}x \rhd z)$
proof *safe*
fix $x\ y\ z$ **assume** $\forall\, x\ y\ z.\ x{\cdot}(y{\cdot}z) = (x{\cdot}y){\cdot}z$
thus $x \rhd (y \rhd z) = y \cdot x \rhd z$
using *conjugate-comp-ext*[*of* $\lambda z.\ y{\cdot}z$ $\lambda z.\ x{\cdot}z$] **by** *auto*
next
fix $x\ y\ z$ **assume** $\forall\, x\ y\ z.\ x \rhd (y \rhd z) = y{\cdot}x \rhd z$
thus $x{\cdot}(y{\cdot}z) = (x{\cdot}y){\cdot}z$
using *conjugate-comp-ext*[*of* $\lambda z.\ y \rhd z$ $\lambda z.\ x \rhd z$] **by** *auto*
qed

lemma *res-assoc-iff2*: $(\forall\, x\ y\ z.\ x{\cdot}(y{\cdot}z) = (x{\cdot}y){\cdot}z) \longleftrightarrow (\forall\, x\ y\ z.\ x \lhd (y \cdot z) = (x \lhd z) \lhd y)$
proof *safe*
fix $x\ y\ z$ **assume** $\forall\, x\ y\ z.\ x{\cdot}(y{\cdot}z) = (x{\cdot}y){\cdot}z$
hence $\forall\, x\ y\ z.\ (x{\cdot}y){\cdot}z = x{\cdot}(y{\cdot}z)$ **by** *simp*
thus $x \lhd (y \cdot z) = (x \lhd z) \lhd y$
using *conjugate-comp-ext*[*of* $\lambda x.\ x{\cdot}z$ $\lambda x.\ x{\cdot}y$] **by** *auto*
next
fix $x\ y\ z$ **assume** $\forall\, x\ y\ z.\ x \lhd (y \cdot z) = (x \lhd z) \lhd y$
hence $\forall\, x\ y\ z.\ (x \lhd z) \lhd y = x \lhd (y \cdot z)$ **by** *simp*
thus $x{\cdot}(y{\cdot}z) = (x{\cdot}y){\cdot}z$
using *conjugate-comp-ext*[*of* $\lambda z.\ z \lhd y$ $\lambda x.\ x \lhd z$] **by** *auto*
qed

lemma *res-assoc-iff3*: $(\forall\, x\ y\ z.\ x{\cdot}(y{\cdot}z) = (x{\cdot}y){\cdot}z) \longleftrightarrow (\forall\, x\ y\ z.\ (x \rhd y) \lhd z = x \rhd (y \lhd z))$

proof *safe*
fix $x\ y\ z$ **assume** $\forall\, x\ y\ z.\ x{\cdot}(y{\cdot}z) = (x{\cdot}y){\cdot}z$
thus $(x \rhd y) \lhd z = x \rhd (y \lhd z)$
using *conjugate-comp-ext*[*of* $\lambda u.\ x{\cdot}u$ $\lambda u.\ u{\cdot}z$] **and**
conjugate-comp-ext[*of* $\lambda u.\ u{\cdot}z$ $\lambda u.\ x{\cdot}u$, *symmetric*]
by *auto*
next
fix $x\ y\ z$ **assume** $\forall\, x\ y\ z.\ (x \rhd y) \lhd z = x \rhd (y \lhd z)$
thus $x{\cdot}(y{\cdot}z) = (x{\cdot}y){\cdot}z$
using *conjugate-comp-ext*[*of* $\lambda u.\ x \rhd u$ $\lambda u.\ u \lhd z$] **and**
conjugate-comp-ext[*of* $\lambda u.\ u \lhd z$ $\lambda u.\ x \rhd u$, *symmetric*]
by *auto*
qed

end

class *unital-residuated-boolean* = *residuated-boolean-algebra* + *one* +
assumes *mult-onel* [*simp*]: $1{\cdot}x = x$
and *mult-oner* [*simp*]: $x{\cdot}1 = x$
begin

The following equivalences are taken from Jósson and Tsinakis.

lemma *jonsson1a*: $(\exists f.\ \forall\, x\ y.\ x \rhd y = f(x){\cdot}y) \longleftrightarrow (\forall\, x\ y.\ x \rhd y = (x \rhd 1){\cdot}y)$
apply *standard*
apply *force*
apply (*rule-tac* $x{=}\lambda x.\ x \rhd 1$ **in** *exI*)
apply *force*
done

lemma *jonsson1b*: $(\forall\, x\ y.\ x \rhd y = (x \rhd 1){\cdot}y) \longleftrightarrow (\forall\, x\ y.\ x{\cdot}y = (x \rhd 1) \rhd y)$
proof *safe*
fix $x\ y$
assume $\forall\, x\ y.\ x \rhd y = (x \rhd 1){\cdot}y$
hence *conjugate* $(\lambda y.\ x \rhd y)$ = *conjugate* $(\lambda y.\ (x \rhd 1){\cdot}y)$ **by** *metis*
thus $x{\cdot}y = (x \rhd 1) \rhd y$ **by** *simp*
next
fix $x\ y$
assume $\forall\, x\ y.\ x \cdot y = x \rhd 1 \rhd y$
thus $x \rhd y = (x \rhd 1) \cdot y$
by (*metis mult-onel*)
qed

lemma *jonsson1c*: $(\forall\, x\ y.\ x \rhd y = (x \rhd 1){\cdot}y) \longleftrightarrow (\forall\, x\ y.\ y \lhd x = 1 \lhd (x \lhd y))$
proof *safe*
fix $x\ y$
assume $\forall\, x\ y.\ x \rhd y = (x \rhd 1){\cdot}y$
hence $(\lambda x.\ x \rhd y) = (\lambda x.\ (x \rhd 1){\cdot}y)$ **by** *metis*
hence $(\lambda x.\ x \rhd y) = (\lambda x.\ x{\cdot}y)\ o\ (\lambda x.\ x \rhd 1)$ **by** *force*
hence *conjugate* $(\lambda x.\ y \lhd x) = (\lambda x.\ x{\cdot}y)\ o$ *conjugate* $(\lambda x.\ 1 \lhd x)$ **by** *simp*

hence *conjugate* (*conjugate* (λx. y ◁ x)) = *conjugate* ((λx. x·y) *o conjugate* (λx. *1* ◁ x)) **by** *simp*
hence (λx. y ◁ x) = *conjugate* ((λx. x·y) *o conjugate* (λx. *1* ◁ x)) **by** *simp*
also have ... = *conjugate* (*conjugate* (λx. *1* ◁ x)) *o conjugate* (λx. x·y)
by (*subst conjugate-comp*[*symmetric*]) *simp-all*
finally show y ◁ x = *1* ◁ (x ◁ y) **by** *simp*
next
fix x y
assume ∀ x y. y ◁ x = *1* ◁ (x ◁ y)
hence (λx. y ◁ x) = (λx. *1* ◁ (x ◁ y)) **by** *metis*
hence (λx. y ◁ x) = (λx. *1* ◁ x) *o conjugate* (λx. x·y) **by** *force*
hence *conjugate* (λx. y ◁ x) = *conjugate* ((λx. *1* ◁ x) *o conjugate* (λx. x·y)) **by** *metis*
also have ... = *conjugate* (*conjugate* (λx. x·y)) *o conjugate* (λx. *1* ◁ x)
by (*subst conjugate-comp*[*symmetric*]) *simp-all*
finally have (λx. x ▷ y) = (λx. x·y) *o* (λx. x ▷ *1*) **by** *simp*
hence (λx. x ▷ y) = (λx. (x ▷ *1*) · y) **by** (*simp add*: *comp-def*)
thus x ▷ y = (x ▷ *1*) · y **by** *metis*
qed

lemma *jonsson2a*: (∃ g. ∀ x y. x ◁ y = x·g(y)) ⟷ (∀ x y. x ◁ y = x·(*1* ◁ y))
apply *standard*
apply *force*
apply (*rule-tac* x=λx. *1* ◁ x **in** *exI*)
apply *force*
done

lemma *jonsson2b*: (∀ x y. x ◁ y = x·(*1* ◁ y)) ⟷ (∀ x y. x·y = x ◁ (*1* ◁ y))
proof *safe*
fix x y
assume ∀ x y. x ◁ y = x·(*1* ◁ y)
hence *conjugate* (λx. x ◁ y) = *conjugate* (λx. x·(*1* ◁ y)) **by** *metis*
thus x·y = x ◁ (*1* ◁ y) **by** *simp metis*
next
fix x y
assume ∀ x y. x·y = x ◁ (*1* ◁ y)
hence (λx. x·y) = (λx. x ◁ (*1* ◁ y)) **by** *metis*
hence *conjugate* (λx. x·y) = *conjugate* (λx. x ◁ (*1* ◁ y)) **by** *metis*
thus x ◁ y = x · (*1* ◁ y) **by** *simp metis*
qed

lemma *jonsson2c*: (∀ x y. x ◁ y = x·(*1* ◁ y)) ⟷ (∀ x y. y ▷ x = (x ▷ y) ▷ *1*)
proof *safe*
fix x y
assume ∀ x y. x ◁ y = x·(*1* ◁ y)
hence (λy. x ◁ y) = (λy. x·(*1* ◁ y)) **by** *metis*
hence (λy. x ◁ y) = (λy. x·y) *o* (λy. *1* ◁ y) **by** *force*
hence *conjugate* (λy. y ▷ x) = (λy. x·y) *o conjugate* (λy. y ▷ *1*) **by** *force*
hence *conjugate* (*conjugate* (λy. y ▷ x)) = *conjugate* ((λy. x·y) *o conjugate* (λy.

$y \rhd 1$)) **by** *metis*
hence $(\lambda y.\ y \rhd x) = conjugate\ ((\lambda y.\ x{\cdot}y)\ o\ conjugate\ (\lambda y.\ y \rhd 1))$ **by** *simp*
also have ... $= conjugate\ (conjugate\ (\lambda y.\ y \rhd 1))\ o\ conjugate\ (\lambda y.\ x{\cdot}y)$
by (*subst conjugate-comp*[*symmetric*]) *simp-all*
finally have $(\lambda y.\ y \rhd x) = (\lambda y.\ x \rhd y \rhd 1)$ **by** (*simp add*: *comp-def*)
thus $y \rhd x = x \rhd y \rhd 1$ **by** *metis*
next
fix $x\ y$
assume $\forall x\ y.\ y \rhd x = x \rhd y \rhd 1$
hence $(\lambda y.\ y \rhd x) = (\lambda y.\ x \rhd y \rhd 1)$ **by** *force*
hence $(\lambda y.\ y \rhd x) = (\lambda y.\ y \rhd 1)\ o\ conjugate\ (\lambda y.\ x{\cdot}y)$ **by** *force*
hence $conjugate\ (\lambda y.\ y \rhd x) = conjugate\ ((\lambda y.\ y \rhd 1)\ o\ conjugate\ (\lambda y.\ x{\cdot}y))$
by *metis*
also have ... $= conjugate\ (conjugate\ (\lambda y.\ x{\cdot}y))\ o\ conjugate\ (\lambda y.\ y \rhd 1)$
by (*subst conjugate-comp*[*symmetric*]) *simp-all*
finally have $(\lambda y.\ x \lhd y) = (\lambda y.\ x{\cdot}y)\ o\ (\lambda y.\ 1 \lhd y)$
by (*metis conjugate-conjr1 conjugate-conjr2 conjugate-multr*)
thus $x \lhd y = x \cdot (1 \lhd y)$ **by** (*simp add*: *comp-def*)
qed

lemma *jonsson3a*: $(\forall x.\ (x \rhd 1) \rhd 1 = x) \longleftrightarrow (\forall x.\ 1 \lhd (1 \lhd x) = x)$
proof *safe*
fix x **assume** $\forall x.\ x \rhd 1 \rhd 1 = x$
thus $1 \lhd (1 \lhd x) = x$
by (*metis compl-le-swap1 compl-le-swap2 conjr2-iff order.eq-iff*)
next
fix x **assume** $\forall x.\ 1 \lhd (1 \lhd x) = x$
thus $x \rhd 1 \rhd 1 = x$
by (*metis conjugate-l-def conjugate-r-def double-compl jipsen2r*)
qed

lemma *jonsson3b*: $(\forall x.\ (x \rhd 1) \rhd 1 = x) \Longrightarrow (x \sqcap y) \rhd 1 = (x \rhd 1) \sqcap (y \rhd 1)$
proof (*rule order.antisym*, *auto simp*: *conjr2-iso*)
assume *assm*: $\forall x.\ (x \rhd 1) \rhd 1 = x$
hence $(x \rhd 1) \sqcap (y \rhd 1) \rhd 1 = x \sqcap (((x \rhd 1) \sqcap (y \rhd 1) \rhd 1) \sqcap y)$
by (*metis* (*no-types*) *conjr2-iso inf.cobounded2 inf.commute inf.orderE*)
hence $(x \rhd 1) \sqcap (y \rhd 1) \rhd 1 \leq x \sqcap y$
using *inf.orderI inf-left-commute* **by** *presburger*
thus $(x \rhd 1) \sqcap (y \rhd 1) \leq x \sqcap y \rhd 1$
using *assm* **by** (*metis* (*no-types*) *conjr2-iso*)
qed

lemma *jonsson3c*: $\forall x.\ (x \rhd 1) \rhd 1 = x \Longrightarrow x \rhd 1 = 1 \lhd x$
proof (*rule indirect-eq*)
fix z
assume *assms*: $\forall x.\ (x \rhd 1) \rhd 1 = x$
hence $(x \rhd 1) \sqcap -z = \bot \longleftrightarrow ((x \rhd 1) \sqcap -z) \rhd 1 = \bot$
by (*metis compl-sup conjugation-conj double-compl inf-bot-right sup-bot.left-neutral*)
also have ... $\longleftrightarrow -z{\cdot}x \sqcap 1 = \bot$

by (*metis assms jonsson3b conjugation-multr*)
finally have $(x \rhd 1) \sqcap -z = \bot \longleftrightarrow (1 \lhd x) \sqcap -z = \bot$
by (*metis conjugation-multl inf.commute*)
thus $(x \rhd 1 \leq z) \longleftrightarrow (1 \lhd x \leq z)$
by (*metis le-iff-inf-bot*)
qed

end

class *residuated-boolean-semigroup* = *residuated-boolean-algebra* + *semigroup-mult*
begin

subclass *residuated-boolean-algebra* **..**

The following lemmas hold trivially, since they are equivalent to associativity.

lemma *res-assoc1*: $x \rhd (y \rhd z) = y{\cdot}x \rhd z$
by (*metis res-assoc-iff1 mult-assoc*)

lemma *res-assoc2*: $x \lhd (y \cdot z) = (x \lhd z) \lhd y$
by (*metis res-assoc-iff2 mult-assoc*)

lemma *res-assoc3*: $(x \rhd y) \lhd z = x \rhd (y \lhd z)$
by (*metis res-assoc-iff3 mult-assoc*)

end

class *residuated-boolean-monoid* = *residuated-boolean-algebra* + *monoid-mult*
begin

subclass *unital-residuated-boolean*
by *standard auto*

subclass *residuated-lmonoid* **..**

lemma *jonsson4*: $(\forall x\ y.\ x \lhd y = x{\cdot}(1 \lhd y)) \longleftrightarrow (\forall x\ y.\ x \rhd y = (x \rhd 1){\cdot}y)$
proof *safe*
fix $x\ y$ **assume** *assms*: $\forall x\ y.\ x \lhd y = x{\cdot}(1 \lhd y)$
have $x \rhd y = (y \rhd x) \rhd 1$
by (*metis assms jonsson2c*)
also have $\ldots = (y \rhd ((x \rhd 1) \rhd 1)) \rhd 1$
by (*metis assms jonsson2b jonsson3a mult-oner*)
also have $\ldots = (((x \rhd 1){\cdot}y) \rhd 1) \rhd 1$
by (*metis conjugate-r-def double-compl resr3*)
also have $\ldots = (x \rhd 1){\cdot}y$
by (*metis assms jonsson2b jonsson3a mult-oner*)
finally show $x \rhd y = (x \rhd 1){\cdot}y$ **.**
next
fix $x\ y$ **assume** *assms*: $\forall x\ y.\ x \rhd y = (x \rhd 1){\cdot}y$

have $y \lhd x = 1 \lhd (x \lhd y)$
by (*metis assms jonsson1c*)
also have ... $= 1 \lhd ((1 \lhd (1 \lhd x)) \lhd y)$
by (*metis assms conjugate-l-def double-compl jonsson1c mult-1-right resl3*)
also have ... $= 1 \lhd (1 \lhd (y \cdot (1 \lhd x)))$
by (*metis conjugate-l-def double-compl resl3*)
also have ... $= y \cdot (1 \lhd x)$
by (*metis assms jonsson1b jonsson1c jonsson3c mult-onel*)
finally show $y \lhd x = y \cdot (1 \lhd x)$ **.**
qed

end

end

4 Involutive Residuated Structures

theory *Involutive-Residuated*
imports *Residuated-Lattices*
begin

class *uminus'* =
fixes *uminus'* :: *'a* ⇒ *'a* (‹−'' -› [*81*] *80*)

Involutive posets is a structure where the double negation property holds for the negation operations, and a Galois connection for negations exists.

class *involutive-order* = *order* + *uminus* + *uminus'* +
assumes *gn*: $x \leq -'y \longleftrightarrow y \leq -x$
and *dn1*[*simp*]: $-'(-x) = x$
and *dn2*[*simp*]: $-(-'x) = x$

class *involutive-pogroupoid* = *order* + *times* + *involutive-order* +
assumes *ipg1*: $x \cdot y \leq z \longleftrightarrow (-z) \cdot x \leq -y$
and *ipg2*: $x \cdot y \leq z \longleftrightarrow y \cdot (-'z) \leq -'x$
begin

lemma *neg-antitone*: $x \leq y \Longrightarrow -y \leq -x$
by (*metis local.dn1 local.gn*)

lemma *neg'-antitone*: $x \leq y \Longrightarrow -'y \leq -'x$
by (*metis local.dn2 local.gn*)

subclass *pogroupoid*
proof
fix *x y z* **assume** *assm*: $x \leq y$
show $x \cdot z \leq y \cdot z$
by (*metis assm local.ipg2 local.order-refl local.order-trans neg'-antitone*)
show $z \cdot x \leq z \cdot y$
by (*metis assm local.dual-order.trans local.ipg1 local.order-refl neg-antitone*)

qed

abbreviation *inv-resl* :: $'a \Rightarrow 'a \Rightarrow 'a$ **where**
inv-resl $y\ x \equiv -(x \cdot (-' y))$

abbreviation *inv-resr* :: $'a \Rightarrow 'a \Rightarrow 'a$ **where**
inv-resr $x\ y \equiv -'((-y) \cdot x)$

sublocale *residuated-pogroupoid* - - - *inv-resl inv-resr*
proof
fix $x\ y\ z$
show $(x \leq -\ (y \cdot -'\ z)) = (x \cdot y \leq z)$
by (*metis local.gn local.ipg2*)
show $(x \cdot y \leq z) = (y \leq -'\ (-\ z \cdot x))$
by (*metis local.gn local.ipg1*)
qed

end

class *division-order* = *order* + *residual-l-op* + *residual-r-op* +
assumes *div-galois*: $x \leq z \leftarrow y \longleftrightarrow y \leq x \rightarrow z$

class *involutive-division-order* = *division-order* + *involutive-order* +
assumes *contraposition*: $y \rightarrow -x = -'y \leftarrow x$

context *involutive-pogroupoid* **begin**

sublocale *involutive-division-order* - - *inv-resl inv-resr*
proof
fix $x\ y\ z$
show $(x \leq -\ (y \cdot -'\ z)) = (y \leq -'\ (-\ z \cdot x))$
by (*metis local.gn local.ipg1 local.ipg2*)
show $-'\ (-\ (-\ x) \cdot y) = -\ (x \cdot -'\ (-'\ y))$
by (*metis local.dn1 local.dn2 order.eq-iff local.gn local.jipsen1l local.jipsen1r local.resl-galois local.resr-galois*)
qed

lemma *inv-resr-neg* [*simp*]: *inv-resr* $(-x)\ (-y) =$ *inv-resl* $x\ y$
by (*metis local.contraposition local.dn1*)

lemma *inv-resl-neg'* [*simp*]: *inv-resl* $(-'x)\ (-'y) =$ *inv-resr* $x\ y$
by (*metis local.contraposition local.dn2*)

lemma *neg'-mult-resl*: $-'((-y) \cdot (-x)) =$ *inv-resl* $x\ (-'y)$
by (*metis inv-resr-neg local.dn2*)

lemma *neg-mult-resr*: $-((-'y) \cdot (-'x)) =$ *inv-resr* $(-x)\ y$
by (*metis neg'-mult-resl*)

lemma *resr-de-morgan1*: $-'(\mathit{inv\text{-}resr}\ (-y)\ (-x)) = -'(\mathit{inv\text{-}resl}\ y\ x)$
by (*metis local.dn1 neg-mult-resr*)

lemma *resr-de-morgan2*: $-(\mathit{inv\text{-}resl}\ (-'x)\ (-'y)) = -(\mathit{inv\text{-}resr}\ x\ y)$
by (*metis inv-resl-neg'*)

end

We prove that an involutive division poset is equivalent to an involutive po-groupoid by a lemma to avoid cyclic definitions

lemma (**in** *involutive-division-order*) *inv-pogroupoid*:
class.involutive-pogroupoid $(\lambda x\ y.\ -(y \rightarrow -'x))$ *uminus uminus'* $(\leq)$ $(<)$
proof
fix $x\ y\ z$
have $(-\ (y \rightarrow -'\ x) \leq z) = (-z \leq -y \leftarrow x)$
by (*metis local.contraposition local.dn1 local.dn2 local.gn local.div-galois*)
thus $(-\ (y \rightarrow -'\ x) \leq z) = (-\ (x \rightarrow -'\ (-\ z)) \leq -\ y)$
by (*metis local.contraposition local.div-galois local.dn1 local.dn2 local.gn*)
moreover have $(-\ (x \rightarrow -'\ (-\ z)) \leq -\ y) = (-\ (-'\ z \rightarrow -'\ y) \leq -'\ x)$
apply (*auto*, *metis local.contraposition local.div-galois local.dn1 local.dn2 local.gn*)
by (*metis local.contraposition local.div-galois local.dn1 local.dn2 local.gn*)
ultimately show $(-\ (y \rightarrow -'\ x) \leq z) = (-\ (-'\ z \rightarrow -'\ y) \leq -'\ x)$
by *metis*
qed

context *involutive-pogroupoid* **begin**

definition *negation-constant* :: $'a \Rightarrow \mathit{bool}$ **where**
$\mathit{negation\text{-}constant}\ a \equiv \forall\, x.\ -'x = \mathit{inv\text{-}resr}\ x\ a \wedge -x = \mathit{inv\text{-}resl}\ a\ x$

definition *division-unit* :: $'a \Rightarrow \mathit{bool}$ **where**
$\mathit{division\text{-}unit}\ a \equiv \forall\, x.\ x = \mathit{inv\text{-}resr}\ a\ x \wedge x = \mathit{inv\text{-}resl}\ x\ a$

lemma *neg-iff-div-unit*: $(\exists\, a.\ \mathit{negation\text{-}constant}\ a) \longleftrightarrow (\exists\, b.\ \mathit{division\text{-}unit}\ b)$
unfolding *negation-constant-def division-unit-def*
apply *safe*
apply (*rule-tac* $x{=}-a$ **in** *exI*, *auto*)
apply (*metis local.dn1 local.dn2*)
apply (*metis local.dn2*)
apply (*rule-tac* $x{=}-b$ **in** *exI*, *auto*)
apply (*metis local.contraposition*)
apply (*metis local.dn2*)
done

end

end

5 Action Algebras

theory *Action-Algebra*
imports *../Residuated-Lattices/Residuated-Lattices Kleene-Algebra.Kleene-Algebra*
begin

Action algebras have been defined and discussed in Pratt's paper on *Action Logic and Pure Induction* [4]. They are expansions of Kleene algebras by operations of left and right residuation. They are interesting, first because most models of Kleene algebras, e.g. relations, traces, paths and languages, possess the residuated structure, and second because, in this setting, the Kleene star can be defined equationally.

Action algebras can be based on residuated semilattices. Many important properties of action algebras already arise at this level.

We can define an action algebra as a residuated join semilattice that is also a dioid. Following Pratt, we also add a star operation that is axiomatised as a reflexive transitive closure operation.

class *action-algebra* = *residuated-sup-lgroupoid* + *dioid-one-zero* + *star-op* +
  assumes *star-rtc1*: $1 + x^\star \cdot x^\star + x \leq x^\star$
  and *star-rtc2*: $1 + y \cdot y + x \leq y \Longrightarrow x^\star \leq y$
begin

lemma *plus-sup*: $(+) = (\sqcup)$
  by (*rule ext*)+ (*simp add*: *local.join.sup-unique*)

We first prove a reflexivity property for residuals.

lemma *residual-r-refl*: $1 \leq x \rightarrow x$
  by (*simp add*: *local.resrI*)

lemma *residual-l-refl*: $1 \leq x \leftarrow x$
  by (*simp add*: *local.reslI*)

We now derive pure induction laws for residuals.

lemma *residual-l-pure-induction*: $(x \leftarrow x)^\star \leq x \leftarrow x$
proof −
  have $1 + (x \leftarrow x) \cdot (x \leftarrow x) + (x \leftarrow x) \leq (x \leftarrow x)$
    using *local.resl-antitoner local.resl-galois mult-assoc* **by** *auto*
  thus *?thesis*
    by (*fact star-rtc2*)
qed

lemma *residual-r-pure-induction*: $(x \rightarrow x)^\star \leq x \rightarrow x$
proof −
  have $1 + (x \rightarrow x) \cdot (x \rightarrow x) + (x \rightarrow x) \leq (x \rightarrow x)$
    using *local.resr-antitonel local.resr-galois mult-assoc* **apply** *clarsimp*
    by (*metis local.mult-oner residual-r-refl*)
  thus *?thesis*

by (*fact star-rtc2*)
qed

Next we show that every action algebra is a Kleene algebra. First, we derive the star unfold law and the star induction laws in action algebra. Then we prove a subclass statement.

lemma *star-unfoldl*: $1 + x \cdot x^\star \leq x^\star$
proof −
have $x \cdot x^\star \leq x^\star$
by (*meson local.dual-order.trans local.join.le-sup-iff local.mult-isor local.star-rtc1*)
thus *?thesis*
using *local.star-rtc1* **by** *auto*
qed

lemma *star-mon* [*intro*]: $x \leq y \Longrightarrow x^\star \leq y^\star$
proof −
assume $x \leq y$
hence $x \leq y^\star$
by (*meson local.dual-order.trans local.join.le-supE local.star-rtc1*)
hence $1 + x + y^\star \cdot y^\star \leq y^\star$
using *local.star-rtc1* **by** *auto*
thus $x^\star \leq y^\star$
by (*simp add*: *local.star-rtc2*)
qed

lemma *star-subdist'*: $x^\star \leq (x + y)^\star$
by *force*

lemma *star-inductl*: $z + x \cdot y \leq y \Longrightarrow x^\star \cdot z \leq y$
proof −
assume *hyp*: $z + x \cdot y \leq y$
hence $x \leq y \leftarrow y$
by (*simp add*: *local.resl-galois*)
hence $x^\star \leq (y \leftarrow y)^\star$
by (*fact star-mon*)
hence $x^\star \leq y \leftarrow y$
using *local.order-trans residual-l-pure-induction* **by** *blast*
hence $x^\star \cdot y \leq y$
by (*simp add*: *local.resl-galois*)
thus $x^\star \cdot z \leq y$
by (*meson hyp local.dual-order.trans local.join.le-supE local.mult-isol*)
qed

lemma *star-inductr*: $z + y \cdot x \leq y \Longrightarrow z \cdot x^\star \leq y$
proof −
assume *hyp*: $z + y \cdot x \leq y$
hence $x \leq y \rightarrow y$
by (*simp add*: *resr-galois*)
hence $x^\star \leq (y \rightarrow y)^\star$

by (*fact star-mon*)
hence $x^\star \leq y \rightarrow y$
by (*metis order-trans residual-r-pure-induction*)
hence $y \cdot x^\star \leq y$
by (*simp add*: *local.resr-galois*)
thus $z \cdot x^\star \leq y$
by (*meson hyp local.join.le-supE local.order-trans local.resl-galois*)
qed

subclass *kleene-algebra*
proof
fix $x\ y\ z$
show $1 + x \cdot x^\star \leq x^\star$
using *local.star-unfoldl* **by** *blast*
show $z + x \cdot y \leq y \Longrightarrow x^\star \cdot z \leq y$
by (*simp add*: *local.star-inductl*)
show $z + y \cdot x \leq y \Longrightarrow z \cdot x^\star \leq y$
by (*simp add*: *star-inductr*)
qed

end

5.1 Equational Action Algebras

The induction axioms of Kleene algebras are universal Horn formulas. This is unavoidable, because due to a well known result of Redko, there is no finite equational axiomatisation for the equational theory of regular expressions.

Action algebras, in contrast, admit a finite equational axiomatization, as Pratt has shown. We now formalise this result. Consequently, the equational action algebra axioms, which imply those based on Galois connections, which in turn imply those of Kleene algebras, are complete with respect to the equational theory of regular expressions. However, this completeness result does not account for residuation.

class *equational-action-algebra* = *residuated-sup-lgroupoid* + *dioid-one-zero* + *star-op* +
assumes *star-ax*: $1 + x^\star \cdot x^\star + x \leq x^\star$
and *star-subdist*: $x^\star \leq (x + y)^\star$
and *right-pure-induction*: $(x \rightarrow x)^\star \leq x \rightarrow x$
begin

We now show that the equational axioms of action algebra satisfy those based on the Galois connections. Since we can use our correspondence between the two variants of residuated semilattice, it remains to derive the second reflexive transitive closure axiom for the star, essentially copying Pratt's proof step by step. We then prove a subclass statement.

lemma *star-rtc-2*: $1 + y \cdot y + x \leq y \Longrightarrow x^\star \leq y$
proof −

assume $1 + y \cdot y + x \leq y$
hence $1 \leq y$ **and** $x \leq y$ **and** $y \cdot y \leq y$
by *auto*
hence $y \leq y \rightarrow y$ **and** $x \leq y \rightarrow y$
using *local.order-trans* **by** *blast+*
hence $x^\star \leq (y \rightarrow y)^\star$
by (*metis local.join.sup.absorb2 local.star-subdist*)
hence $x^\star \leq y \rightarrow y$
by (*metis order-trans right-pure-induction*)
hence $y \cdot x^\star \leq y$
by (*simp add*: *local.resr-galois*)
thus $x^\star \leq y$
by (*metis ‹1 ≤ y› local.dual-order.trans local.mult-onel local.resl-galois*)
qed

subclass *action-algebra*
by (*unfold-locales*, *metis star-ax*, *metis star-rtc-2*)

end

Conversely, every action algebra satisfies the equational axioms of equational action algebras.

Because the subclass relation must be acyclic in Isabelle, we can only establish this for the corresponding locales. Again this proof is based on the residuated semilattice result.

sublocale *action-algebra* $\subseteq$ *equational-action-algebra*
by (*unfold-locales*, *metis star-rtc1*, *metis star-subdist*, *metis residual-r-pure-induction*)

5.2 Another Variant

Finally we show that Pratt and Kozen's star axioms generate precisely the same theory.

class *action-algebra-var* = *residuated-sup-lgroupoid* + *dioid-one-zero* + *star-op* +
assumes *star-unfold′*: $1 + x \cdot x^\star \leq x^\star$
and *star-inductl′*: $z + x \cdot y \leq y \Longrightarrow x^\star \cdot z \leq y$
and *star-inductr′*: $z + y \cdot x \leq y \Longrightarrow z \cdot x^\star \leq y$
begin

subclass *kleene-algebra*
by (*unfold-locales*, *metis star-unfold′*, *metis star-inductl′*, *metis star-inductr′*)

subclass *action-algebra*
by (*unfold-locales*, *metis add.commute less-eq-def order-refl star-ext star-plus-one star-trans-eq*, *metis add.assoc add.commute star-rtc-least*)

end

sublocale *action-algebra* $\subseteq$ *action-algebra-var*

by (*unfold-locales*, *metis star-unfoldl*, *metis star-inductl*, *metis star-inductr*)

end

6 Models of Action Algebras

theory *Action-Algebra-Models*
imports *Action-Algebra Kleene-Algebra.Kleene-Algebra-Models*
begin

6.1 The Powerset Action Algebra over a Monoid

Here we show that various models of Kleene algebras are also residuated; hence they form action algebras. In each case the main work is to establish the residuated lattice structure.

The interpretation proofs for some of the following models are quite similar. One could, perhaps, abstract out common reasoning.

6.2 The Powerset Action Algebra over a Monoid

instantiation *set* :: (*monoid-mult*) *residuated-sup-lgroupoid*
begin

definition *residual-r-set* **where**
$X \rightarrow Z = \bigcup \{Y.\ X \cdot Y \subseteq Z\}$

definition *residual-l-set* **where**
$Z \leftarrow Y = \bigcup \{X.\ X \cdot Y \subseteq Z\}$

instance
proof
fix $X\ Y\ Z :: 'a\ set$
show $X \subseteq (Z \leftarrow Y) \longleftrightarrow X \cdot Y \subseteq Z$
proof
assume $X \subseteq Z \leftarrow Y$
hence $X \cdot Y \subseteq (Z \leftarrow Y) \cdot Y$
by (*metis near-dioid-class.mult-isor*)
also have $\ldots \subseteq \bigcup \{X.\ X \cdot Y \subseteq Z\} \cdot Y$
by (*simp add*: *residual-l-set-def*)
also have $\ldots = \bigcup \{X \cdot Y \mid X.\ X \cdot Y \subseteq Z\}$
by (*auto simp only*: *c-prod-def*)
finally show $X \cdot Y \subseteq Z$
by *auto*
next
assume $X \cdot Y \subseteq Z$
hence $X \subseteq \bigcup \{X.\ X \cdot Y \subseteq Z\}$
by (*metis Sup-upper mem-Collect-eq*)

thus $X \subseteq Z \leftarrow Y$
by (*simp add*: *residual-l-set-def*)
qed
show $X \cdot Y \subseteq Z \longleftrightarrow Y \subseteq (X \rightarrow Z)$
proof
assume $Y \subseteq X \rightarrow Z$
hence $X \cdot Y \subseteq X \cdot (X \rightarrow Z)$
by (*metis pre-dioid-class.mult-isol*)
also have $\ldots \subseteq X \cdot \bigcup \{Y.\ X \cdot Y \subseteq Z\}$
by (*simp add*: *residual-r-set-def*)
also have $\ldots = \bigcup \{X \cdot Y \mid Y.\ X \cdot Y \subseteq Z\}$
by (*auto simp only*: *c-prod-def*)
finally show $X \cdot Y \subseteq Z$
by *auto*
next
assume $X \cdot Y \subseteq Z$
hence $Y \subseteq \bigcup \{Y.\ X \cdot Y \subseteq Z\}$
by (*metis Sup-upper mem-Collect-eq*)
thus $Y \subseteq X \rightarrow Z$
by (*simp add*: *residual-r-set-def*)
qed
qed

end

instantiation *set* :: (*monoid-mult*) *action-algebra*
begin

instance
proof
fix $x\ y$ :: *'a set*
show $1 + x^\star \cdot x^\star + x \subseteq x^\star$
by *simp*
show $1 + y \cdot y + x \subseteq y \Longrightarrow x^\star \subseteq y$
by (*simp add*: *left-pre-kleene-algebra-class.star-rtc-least*)
qed

end

6.3 Language Action Algebras

definition *limp-lan* :: *'a lan* ⇒ *'a lan* ⇒ *'a lan* **where**
limp-lan Z Y = $\{x.\ \forall y \in Y.\ x \mathbin{@} y \in Z\}$

definition *rimp-lan* :: *'a lan* ⇒ *'a lan* ⇒ *'a lan* **where**
rimp-lan X Z = $\{y.\ \forall x \in X.\ x \mathbin{@} y \in Z\}$

interpretation *lan-residuated-join-semilattice*: *residuated-sup-lgroupoid* (+) (⊆) (⊂) (·) *limp-lan rimp-lan*

proof
fix $x\ y\ z :: 'a\ lan$
show $x \subseteq limp\text{-}lan\ z\ y \longleftrightarrow x \cdot y \subseteq z$
by (*auto simp add*: *c-prod-def limp-lan-def times-list-def*)
show $x \cdot y \subseteq z \longleftrightarrow y \subseteq rimp\text{-}lan\ x\ z$
by (*auto simp add*: *c-prod-def rimp-lan-def times-list-def*)
qed

interpretation *lan-action-algebra*: *action-algebra* $(+)$ $(\cdot)$ *1 0* $(\subseteq)$ $(\subset)$ $(+)$ *limp-lan rimp-lan star*
proof
fix $x\ y :: 'a\ lan$
show $1 + x^\star \cdot x^\star + x \subseteq x^\star$
by *simp*
show $1 + y \cdot y + x \subseteq y \Longrightarrow x^\star \subseteq y$
by (*simp add*: *action-algebra-class.star-rtc-2*)
qed

6.4 Relation Action Algebras

definition *limp-rel* :: $'a\ rel \Rightarrow 'a\ rel \Rightarrow 'a\ rel$ **where**
$limp\text{-}rel\ T\ S = \{(y,x) \mid y\ x.\ \forall z.\ (x,z) \in S \longrightarrow (y,z) \in T\}$

definition *rimp-rel* :: $'a\ rel \Rightarrow 'a\ rel \Rightarrow 'a\ rel$ **where**
$rimp\text{-}rel\ R\ T = \{(y,z) \mid y\ z.\ \forall x.\ (x,y) \in R \longrightarrow (x,z) \in T\}$

interpretation *rel-residuated-join-semilattice*: *residuated-sup-lgroupoid* $(\cup)$ $(\subseteq)$ $(\subset)$ (O) *limp-rel rimp-rel*
proof
fix $x\ y\ z :: 'a\ rel$
show $x \subseteq limp\text{-}rel\ z\ y \longleftrightarrow x\ O\ y \subseteq z$
by (*auto simp add*: *limp-rel-def*)
show $x\ O\ y \subseteq z \longleftrightarrow y \subseteq rimp\text{-}rel\ x\ z$
by (*auto simp add*: *rimp-rel-def*)
qed

interpretation *rel-action-algebra*: *action-algebra* $(\cup)$ (O) *Id* $\{\}$ $(\subseteq)$ $(\subset)$ $(\cup)$ *limp-rel rimp-rel rtrancl*
proof
fix $x\ y :: 'a\ rel$
show $Id \cup x^*\ O\ x^* \cup x \subseteq x^*$
by *auto*
show $Id \cup y\ O\ y \cup x \subseteq y \Longrightarrow x^* \subseteq y$
by (*simp add*: *rel-kleene-algebra.star-rtc-least*)
qed

6.5 Trace Action Algebras

definition *limp-trace* :: $('p, 'a)\ trace\ set \Rightarrow ('p, 'a)\ trace\ set \Rightarrow ('p, 'a)\ trace\ set$ **where**

limp-trace Z Y = $\bigcup$ {*X*. *t-prod X Y* ⊆ *Z*}

definition *rimp-trace* :: ('*p*, '*a*) *trace set* ⇒ ('*p*, '*a*) *trace set* ⇒ ('*p*, '*a*) *trace set*
where
rimp-trace X Z = $\bigcup$ {*Y*. *t-prod X Y* ⊆ *Z*}

interpretation *trace-residuated-join-semilattice*: *residuated-sup-lgroupoid* (∪) (⊆) (⊂) *t-prod limp-trace rimp-trace*
proof
fix *X Y Z* :: ('*a*,'*b*) *trace set*
show *X* ⊆ *limp-trace Z Y* ⟷ *t-prod X Y* ⊆ *Z*
proof
assume *X* ⊆ *limp-trace Z Y*
hence *t-prod X Y* ⊆ *t-prod* (*limp-trace Z Y*) *Y*
by (*metis trace-dioid.mult-isor*)
also have ... ⊆ *t-prod* ($\bigcup$ {*X*. *t-prod X Y* ⊆ *Z*}) *Y*
by (*simp add*: *limp-trace-def*)
also have ... = $\bigcup$ {*t-prod X Y* | *X*. *t-prod X Y* ⊆ *Z*}
by (*auto simp only*: *t-prod-def*)
finally show *t-prod X Y* ⊆ *Z*
by *auto*
next
assume *t-prod X Y* ⊆ *Z*
hence *X* ⊆ $\bigcup$ {*X*. *t-prod X Y* ⊆ *Z*}
by (*metis Sup-upper mem-Collect-eq*)
thus *X* ⊆ *limp-trace Z Y*
by (*simp add*: *limp-trace-def*)
qed
show *t-prod X Y* ⊆ *Z* ⟷ *Y* ⊆ *rimp-trace X Z*
proof
assume *t-prod X Y* ⊆ *Z*
hence *Y* ⊆ $\bigcup$ {*Y*. *t-prod X Y* ⊆ *Z*}
by (*metis Sup-upper mem-Collect-eq*)
thus *Y* ⊆ *rimp-trace X Z*
by (*simp add*: *rimp-trace-def*)
next
assume *Y* ⊆ *rimp-trace X Z*
hence *t-prod X Y* ⊆ *t-prod X* (*rimp-trace X Z*)
by (*metis trace-dioid.mult-isol*)
also have ... ⊆ *t-prod X* ($\bigcup$ {*Y*. *t-prod X Y* ⊆ *Z*})
by (*simp add*: *rimp-trace-def*)
also have ... = $\bigcup$ {*t-prod X Y* | *Y*. *t-prod X Y* ⊆ *Z*}
by (*auto simp only*: *t-prod-def*)
finally show *t-prod X Y* ⊆ *Z*
by *auto*
qed
qed

interpretation *trace-action-algebra*: *action-algebra* (∪) *t-prod t-one t-zero* (⊆) (⊂)

(∪) *limp-trace rimp-trace t-star*
proof
fix *X Y* :: (*'a*,*'b*) *trace set*
show *t-one* ∪ *t-prod* (*t-star X*) (*t-star X*) ∪ *X* ⊆ *t-star X*
by *auto*
show *t-one* ∪ *t-prod Y Y* ∪ *X* ⊆ *Y* ⟹ *t-star X* ⊆ *Y*
by (*simp add*: *trace-kleene-algebra.star-rtc-least*)
qed

6.6 Path Action Algebras

We start with paths that include the empty path.

definition *limp-path* :: *'a path set* ⇒ *'a path set* ⇒ *'a path set* **where**
limp-path Z Y = ⋃ {*X*. *p-prod X Y* ⊆ *Z*}

definition *rimp-path* :: *'a path set* ⇒ *'a path set* ⇒ *'a path set* **where**
rimp-path X Z = ⋃ {*Y*. *p-prod X Y* ⊆ *Z*}

interpretation *path-residuated-join-semilattice*: *residuated-sup-lgroupoid* (∪) (⊆) (⊂) *p-prod limp-path rimp-path*
proof
fix *X Y Z* :: *'a path set*
show *X* ⊆ *limp-path Z Y* ⟷ *p-prod X Y* ⊆ *Z*
proof
assume *X* ⊆ *limp-path Z Y*
hence *p-prod X Y* ⊆ *p-prod* (*limp-path Z Y*) *Y*
by (*metis path-dioid.mult-isor*)
also have ... ⊆ *p-prod* (⋃ {*X*. *p-prod X Y* ⊆ *Z*}) *Y*
by (*simp add*: *limp-path-def*)
also have ... = ⋃ {*p-prod X Y* | *X*. *p-prod X Y* ⊆ *Z*}
by (*auto simp only*: *p-prod-def*)
finally show *p-prod X Y* ⊆ *Z*
by *auto*
next
assume *p-prod X Y* ⊆ *Z*
hence *X* ⊆ ⋃ {*X*. *p-prod X Y* ⊆ *Z*}
by (*metis Sup-upper mem-Collect-eq*)
thus *X* ⊆ *limp-path Z Y*
by (*simp add*: *limp-path-def*)
qed
show *p-prod X Y* ⊆ *Z* ⟷ *Y* ⊆ *rimp-path X Z*
proof
assume *p-prod X Y* ⊆ *Z*
hence *Y* ⊆ ⋃ {*Y*. *p-prod X Y* ⊆ *Z*}
by (*metis Sup-upper mem-Collect-eq*)
thus *Y* ⊆ *rimp-path X Z*
by (*simp add*: *rimp-path-def*)
next
assume *Y* ⊆ *rimp-path X Z*

hence *p-prod X Y* ⊆ *p-prod X* (*rimp-path X Z*)
by (*metis path-dioid.mult-isol*)
also have ... ⊆ *p-prod X* (⋃ {*Y*. *p-prod X Y* ⊆ *Z*})
by (*simp add*: *rimp-path-def*)
also have ... = ⋃ {*p-prod X Y* | *Y*. *p-prod X Y* ⊆ *Z*}
by (*auto simp only*: *p-prod-def*)
finally show *p-prod X Y* ⊆ *Z*
by *auto*
qed
qed

interpretation *path-action-algebra*: *action-algebra* (∪) *p-prod p-one* {} (⊆) (⊂) (∪) *limp-path rimp-path p-star*
proof
fix *X Y* :: *'a path set*
show *p-one* ∪ *p-prod* (*p-star X*) (*p-star X*) ∪ *X* ⊆ *p-star X*
by *auto*
show *p-one* ∪ *p-prod Y Y* ∪ *X* ⊆ *Y* ⟹ *p-star X* ⊆ *Y*
by (*simp add*: *path-kleene-algebra.star-rtc-least*)
qed

We now consider a notion of paths that does not include the empty path.

definition *limp-ppath* :: *'a ppath set* ⇒ *'a ppath set* ⇒ *'a ppath set* **where**
limp-ppath Z Y = ⋃ {*X*. *pp-prod X Y* ⊆ *Z*}

definition *rimp-ppath* :: *'a ppath set* ⇒ *'a ppath set* ⇒ *'a ppath set* **where**
rimp-ppath X Z = ⋃ {*Y*. *pp-prod X Y* ⊆ *Z*}

interpretation *ppath-residuated-join-semilattice*: *residuated-sup-lgroupoid* (∪) (⊆) (⊂) *pp-prod limp-ppath rimp-ppath*
proof
fix *X Y Z* :: *'a ppath set*
show *X* ⊆ *limp-ppath Z Y* ⟷ *pp-prod X Y* ⊆ *Z*
proof
assume *X* ⊆ *limp-ppath Z Y*
hence *pp-prod X Y* ⊆ *pp-prod* (*limp-ppath Z Y*) *Y*
by (*metis ppath-dioid.mult-isor*)
also have ... ⊆ *pp-prod* (⋃ {*X*. *pp-prod X Y* ⊆ *Z*}) *Y*
by (*simp add*: *limp-ppath-def*)
also have ... = ⋃ {*pp-prod X Y* | *X*. *pp-prod X Y* ⊆ *Z*}
by (*auto simp only*: *pp-prod-def*)
finally show *pp-prod X Y* ⊆ *Z*
by *auto*
next
assume *pp-prod X Y* ⊆ *Z*
hence *X* ⊆ ⋃ {*X*. *pp-prod X Y* ⊆ *Z*}
by (*metis Sup-upper mem-Collect-eq*)
thus *X* ⊆ *limp-ppath Z Y*
by (*simp add*: *limp-ppath-def*)

```
    qed
  show pp-prod X Y ⊆ Z ⟷ Y ⊆ rimp-ppath X Z
    proof
      assume pp-prod X Y ⊆ Z
      hence Y ⊆ ⋃ {Y. pp-prod X Y ⊆ Z}
        by (metis Sup-upper mem-Collect-eq)
      thus Y ⊆ rimp-ppath X Z
        by (simp add: rimp-ppath-def)
    next
      assume Y ⊆ rimp-ppath X Z
      hence pp-prod X Y ⊆ pp-prod X (rimp-ppath X Z)
        by (metis ppath-dioid.mult-isol)
      also have ... ⊆ pp-prod X (⋃ {Y. pp-prod X Y ⊆ Z})
        by (simp add: rimp-ppath-def)
      also have ... = ⋃ {pp-prod X Y | Y. pp-prod X Y ⊆ Z}
        by (auto simp only: pp-prod-def)
      finally show pp-prod X Y ⊆ Z
        by auto
    qed
qed

interpretation ppath-action-algebra: action-algebra (∪) pp-prod pp-one {} (⊆) (⊂)
(∪) limp-ppath rimp-ppath pp-star
proof
  fix X Y :: 'a ppath set
  show pp-one ∪ pp-prod (pp-star X) (pp-star X) ∪ X ⊆ pp-star X
    by auto
  show pp-one ∪ pp-prod Y Y ∪ X ⊆ Y ⟹ pp-star X ⊆ Y
    by (simp add: ppath-kleene-algebra.star-rtc-least)
qed
```

6.7 The Min-Plus Action Algebra

```
instantiation pnat :: minus
begin

  fun minus-pnat where
    (pnat x) − (pnat y) = pnat (x − y)
  | x − PInfty = 1
  | PInfty − (pnat x) = 0

  instance ..

end

instantiation pnat :: semilattice-sup
begin
  definition sup-pnat: sup-pnat x y ≡ pnat-min x y
  instance
```

proof
fix $x\ y\ z$ :: *pnat*
show $x \leq x \sqcup y$
by (*metis* (*no-types*) *sup-pnat join-semilattice-class.order-prop plus-pnat-def*)
show $y \leq x \sqcup y$
by (*metis add.left-commute less-eq-pnat-def linear plus-pnat-def sup-pnat*)
show $y \leq x \Longrightarrow z \leq x \Longrightarrow y \sqcup z \leq x$
by (*metis add.commute add.left-commute less-eq-pnat-def plus-pnat-def sup-pnat*)

qed

end

instantiation *pnat* :: *residuated-sup-lgroupoid*
begin

definition *residual-r-pnat* **where**
$(x$::*pnat*$) \rightarrow y \equiv y - x$

definition *residual-l-pnat* **where**
$(y$::*pnat*$) \leftarrow x \equiv y - x$

instance
proof
fix $x\ y\ z$ :: *pnat*
show $x \leq (z \leftarrow y) \longleftrightarrow x \cdot y \leq z$
by (*cases x*, *cases y*, *cases z*) (*auto simp add*: *plus-pnat-def times-pnat-def residual-l-pnat-def less-eq-pnat-def zero-pnat-def one-pnat-def*)
show $x \cdot y \leq z \longleftrightarrow y \leq (x \rightarrow z)$
by (*cases y*, *cases x*, *cases z*) (*auto simp add*: *plus-pnat-def times-pnat-def residual-r-pnat-def less-eq-pnat-def zero-pnat-def one-pnat-def*)
qed

end

instantiation *pnat* :: *action-algebra*
begin

The Kleene star for type *pnat* has already been defined in theory *Kleene-Algebra.Kleene-Algebra-Model*

instance
proof
fix $x\ y$ :: *pnat*
show $1 + x^\star \cdot x^\star + x \leq x^\star$
by *auto*
show $1 + y \cdot y + x \leq y \Longrightarrow x^\star \leq y$
by (*simp add*: *star-pnat-def*)
qed

end

end

7 Residuated Relation Algebras

theory *Residuated-Relation-Algebra*
  imports *Residuated-Boolean-Algebras Relation-Algebra.Relation-Algebra*
begin

context *boolean-algebra* **begin**

The notation used in the relation algebra AFP entry differs a little from ours.

notation
  times (**infixl** ‹·› *70*)
  and *plus* (**infixl** ‹+› *65*)
  and *Groups.zero-class.zero* (‹*0*›)
  and *Groups.one-class.one* (‹*1*›)

no-notation
  inf (**infixl** ‹·› *70*)
  and *sup* (**infixl** ‹+› *65*)
  and *bot* (‹*0*›)
  and *top* (‹*1*›)

end

We prove that a unital residuated boolean algebra enriched with two simple equalities form a non-associative relation algebra, that is, a relation algebra where the associativity law does not hold.

class *nra* = *unital-residuated-boolean* +
  assumes *conv1*: $x \rhd y = (x \rhd 1)\cdot y$
  and *conv2*: $x \lhd y = x\cdot(1 \lhd y)$
begin

When the converse operation is set to be $\lambda x.\ x \rhd 1$, a unital residuated boolean algebra forms a non associative relation algebra.

lemma *conv-invol*: $x \rhd 1 \rhd 1 = x$
  by (*metis local.conv1 local.jonsson1b local.mult-onel*)

lemma *conv-add*: $x \sqcup y \rhd 1 = (x \rhd 1) \sqcup (y \rhd 1)$
  by (*metis local.conjr2-sup*)

lemma *conv-contrav*: $x\cdot y \rhd 1 = (y \rhd 1)\cdot(x \rhd 1)$
  by (*metis local.conv1 local.conv2 local.jonsson1b local.jonsson2c*)

lemma *conv-res*: $(x \rhd 1)\cdot - (x\cdot y) \leq - y$

by (*metis local.comp-anti local.conjugate-r-def local.conv1 local.double-compl local.res-rc1*)

Similarly, for $x^\smile = 1 \lhd x$, since $x \rhd 1 = 1 \lhd x$ when $x \rhd 1 \rhd 1 = x$ holds.

lemma *conv-invol'*: $1 \lhd (1 \lhd x) = x$
by (*metis local.conv-invol local.jonsson3c*)

lemma *conv-add'*: $1 \lhd (x \sqcup y) = (1 \lhd x) \sqcup (1 \lhd y)$
by (*metis local.conjl1-sup*)

lemma *conv-contrav'*: $1 \lhd x{\cdot}y = (1 \lhd y){\cdot}(1 \lhd x)$
by (*metis local.conv-contrav local.conv-invol local.jonsson3c*)

lemma *conv-res'*: $(1 \lhd x){\cdot}{-}\,(x{\cdot}y) \leq -\,y$
by (*metis conv-res local.conv-invol local.jonsson3c*)

end

Since the previous axioms are equivalent when multiplication is associative in a residuated boolean monoid, one of them are sufficient to derive a relation algebra.

class *residuated-ra* = *residuated-boolean-monoid* +
assumes *conv*: $x \rhd y = (x \rhd 1){\cdot}y$
begin

subclass *nra*
proof (*standard*, *fact conv*)
fix $x\ y$ **show** $x \lhd y = x{\cdot}(1 \lhd y)$
by (*metis conv jonsson4*)
qed

sublocale *relation-algebra* **where**
composition = $(\cdot)$ **and** *unit* = *1* **and**
converse = $\lambda x.\ x \rhd 1$
proof
fix $x\ y\ z$
show $x{\cdot}y{\cdot}z = x{\cdot}(y{\cdot}z)$
by (*metis local.mult-assoc*)
show $x{\cdot}1 = x$
by (*metis local.mult-onel*)
show $(x \sqcup y){\cdot}z = x{\cdot}z \sqcup y{\cdot}z$
by (*metis local.distr*)
show $x \rhd 1 \rhd 1 = x$
by (*metis local.conv-invol*)
show $x \sqcup y \rhd 1 = (x \rhd 1) \sqcup (y \rhd 1)$
by (*metis local.conv-add*)
show $x{\cdot}y \rhd 1 = (y \rhd 1){\cdot}(x \rhd 1)$
by (*metis local.conv-contrav*)
show $(x \rhd 1){\cdot}{-}\,(x{\cdot}y) \leq -\,y$

by (*metis local.conv-res*)
qed

end

end

References

[1] N. Galatos, P. Jipsen, T. Kowalski, and H. Ono. *Residuated Lattices: An Algebraic Glimpse at Substructural Logics: An Algebraic Glimpse at Substructural Logics*, volume 151. Elsevier, 2007.

[2] B. Jónsson and C. Tsinakis. Relation algebras as residuated boolean algebras. *Algebra Universalis*, 30(4):469–478, 1993.

[3] R. D. Maddux. Relation-algebraic semantics. *Theoretical Computer Sciences*, 160(1&2):1–85, 1996.

[4] V. R. Pratt. Action logic and pure induction. In J. van Eijck, editor, *Logics in AI, European Workshop, JELIA '90*, volume 478 of *Lecture Notes in Computer Science*, pages 97–120. Springer, 1991.

[5] M. Ward and R. P. Dilworth. Residuated lattices. *Transactions of the American Mathematical Society*, 45(3):335–354, 1939.