

Stone Relation Algebras

Walter Guttmann

April 20, 2020

Abstract

We develop Stone relation algebras, which generalise relation algebras by replacing the underlying Boolean algebra structure with a Stone algebra. We show that finite matrices over bounded linear orders form an instance. As a consequence, relation-algebraic concepts and methods can be used for reasoning about weighted graphs. We also develop a fixpoint calculus and apply it to compare different definitions of reflexive-transitive closures in semirings.

Contents

1	Synopsis and Motivation	2
2	Fixpoints	3
3	Semirings	13
3.1	Idempotent Semirings	14
3.2	Bounded Idempotent Semirings	21
4	Relation Algebras	24
4.1	Single-Object Bounded Distributive Allegories	25
4.2	Single-Object Pseudocomplemented Distributive Allegories	39
4.3	Stone Relation Algebras	47
4.4	Relation Algebras	49
5	Subalgebras of Relation Algebras	52
6	Matrix Relation Algebras	56
6.1	Finite Suprema	57
6.2	Square Matrices	58
6.3	Stone Algebras	59
6.4	Semirings	61
6.5	Stone Relation Algebras	61
7	Matrices over Bounded Linear Orders	62

1 Synopsis and Motivation

This document describes the following six theory files:

- * `Fixpoints` develops a fixpoint calculus based on partial orders. We also consider least (pre)fixpoints and greatest (post)fixpoints. The derived rules include `unfold`, `square`, `rolling`, `fusion`, `exchange` and `diagonal` rules studied in [1]. Our results are based on the existence of fixpoints instead of completeness of the underlying structure.
- * `Semirings` contains a hierarchy of structures generalising idempotent semirings. In particular, several of these algebras do not assume that multiplication is associative in order to capture models such as multirelations. Even in such a weak setting we can derive several results comparing different definitions of reflexive-transitive closures based on fixpoints.
- * `Relation Algebras` introduces Stone relation algebras, which weaken the Boolean algebra structure of relation algebras to Stone algebras. This is motivated by the wish to represent weighted graphs (matrices over numbers) in addition to unweighted graphs (Boolean matrices) that form relations. Many results of relation algebras can be derived from the weaker axioms and therefore also apply to weighted graphs. Some results hold in Stone relation algebras after small modifications. This allows us to apply relational concepts and methods also to weighted graphs. In particular, we prove a number of properties that have been used to verify graph algorithms. Tarski's relation algebras [28] arise as a special case by imposing further axioms.
- * `Subalgebras of Relation Algebras` studies the structures of subsets of elements characterised by a given property. In particular we look at regular elements (which correspond to unweighted graphs), coreflexives (tests), vectors and covectors (which can be used to represent sets). The subsets are turned into Isabelle/HOL types, which are shown to form instances of various algebras.
- * `Matrix Relation Algebras` lifts the Stone algebra hierarchy, the semiring structure and, finally, Stone relation algebras to finite square matrices. These are mostly standard constructions similar to those in [3, 4] implemented so that they work for many algebraic structures. In particular, they can be instantiated to weighted graphs (see below) and extended to Kleene algebras (not part of this development).
- * `Matrices over Bounded Linear Orders` studies relational properties. In particular, we characterise univalent, injective, total, surjective, mapping, bijective, vector, covector, point, atom, reflexive, coreflexive,

irreflexive, symmetric, antisymmetric and asymmetric matrices. Definitions of these properties are taken from relation algebras and their meaning for matrices over bounded linear orders (weighted graphs) is explained by logical formulas in terms of matrix entries.

The development is based on a theory of Stone algebras [15] and forms the basis for an extension to Kleene algebras to capture further properties of graphs. We apply Stone relation algebras to verify Prim's minimum spanning tree algorithm in Isabelle/HOL in [14].

Related libraries for semirings and relation algebras in the Archive of Formal Proofs are [3, 4]. The theory `Kleene_Algebra/Dioid.thy` introduces a number of structures that generalise idempotent semirings, but does not cover most of the semiring structures in the present development. The theory `Relation_Algebra/Relation_Algebra.thy` covers Tarski's relation algebras and hence cannot be reused for the present development as most properties need to be derived from the weaker axioms of Stone relation algebras. The matrix constructions in theories `Kleene_Algebra/Inf_Matrix.thy` and `Relation_Algebra/Relation_Algebra_Models.thy` are similar, but have strong restrictions on the matrix entry types not appropriate for many algebraic structures in the present development. We also deviate from these hierarchies by basing idempotent semirings directly on the Isabelle/HOL semilattice structures instead of a separate structure; this results in a somewhat smoother integration with the lattice structure of relation algebras.

2 Fixpoints

This theory develops a fixpoint calculus based on partial orders. Besides fixpoints we consider least prefixpoints and greatest postfixpoints of functions on a partial order. We do not assume that the underlying structure is complete or that all functions are continuous or isotone. Assumptions about the existence of fixpoints and necessary properties of the involved functions will be stated explicitly in each theorem. This way, the results can be instantiated by various structures, such as complete lattices and Kleene algebras, which impose different kinds of restriction. See, for example, [1, 10] for fixpoint calculi in complete lattices. Our fixpoint calculus contains similar rules, in particular:

- * unfold rule,
- * fixpoint operators preserve isotonicity,
- * square rule,
- * rolling rule,
- * various fusion rules,

- * exchange rule and
- * diagonal rule.

All of our rules are based on existence rather than completeness of the underlying structure. We have applied results from this theory in [13] and subsequent papers for unifying and reasoning about the semantics of recursion in various relational and matrix-based computation models.

theory *Fixpoints*

imports *Stone-Algebras.Lattice-Basics*

begin

The whole calculus is based on partial orders only.

context *order*

begin

We first define when an element x is a least/greatest (pre/post)fixpoint of a given function f .

definition *is-fixpoint* $f x \equiv f x = x$ $:: ('a \Rightarrow 'a) \Rightarrow 'a \Rightarrow \text{bool}$ **where** *is-fixpoint*

definition *is-prefixpoint* $f x \equiv f x \leq x$ $:: ('a \Rightarrow 'a) \Rightarrow 'a \Rightarrow \text{bool}$ **where** *is-prefixpoint*

definition *is-postfixpoint* $f x \equiv f x \geq x$ $:: ('a \Rightarrow 'a) \Rightarrow 'a \Rightarrow \text{bool}$ **where** *is-postfixpoint*

definition *is-least-fixpoint* $f x \equiv f x = x \wedge (\forall y . f y = y \longrightarrow x \leq y)$ $:: ('a \Rightarrow 'a) \Rightarrow 'a \Rightarrow \text{bool}$ **where** *is-least-fixpoint*

definition *is-greatest-fixpoint* $f x \equiv f x = x \wedge (\forall y . f y = y \longrightarrow x \geq y)$ $:: ('a \Rightarrow 'a) \Rightarrow 'a \Rightarrow \text{bool}$ **where** *is-greatest-fixpoint*

definition *is-least-prefixpoint* $f x \equiv f x \leq x \wedge (\forall y . f y \leq y \longrightarrow x \leq y)$ $:: ('a \Rightarrow 'a) \Rightarrow 'a \Rightarrow \text{bool}$ **where** *is-least-prefixpoint*

definition *is-greatest-postfixpoint* $f x \equiv f x \geq x \wedge (\forall y . f y \geq y \longrightarrow x \geq y)$ $:: ('a \Rightarrow 'a) \Rightarrow 'a \Rightarrow \text{bool}$ **where** *is-greatest-postfixpoint*

Next follows the existence of the corresponding fixpoints for a given function f .

definition *has-fixpoint* $f \equiv \exists x . \text{is-fixpoint } f x$ $:: ('a \Rightarrow 'a) \Rightarrow \text{bool}$ **where** *has-fixpoint*

definition *has-prefixpoint* $f \equiv \exists x . \text{is-prefixpoint } f x$ $:: ('a \Rightarrow 'a) \Rightarrow \text{bool}$ **where** *has-prefixpoint*

definition *has-postfixpoint* $f \equiv \exists x . \text{is-postfixpoint } f x$ $:: ('a \Rightarrow 'a) \Rightarrow \text{bool}$ **where** *has-postfixpoint*

definition *has-least-fixpoint* $f \equiv \exists x . \text{is-least-fixpoint } f x$ $:: ('a \Rightarrow 'a) \Rightarrow \text{bool}$ **where** *has-least-fixpoint*

definition *has-greatest-fixpoint* $f \equiv \exists x . \text{is-greatest-fixpoint } f x$ $:: ('a \Rightarrow 'a) \Rightarrow \text{bool}$ **where** *has-greatest-fixpoint*

definition *has-least-prefixpoint* $f \equiv \exists x . \text{is-least-prefixpoint } f x$ $:: ('a \Rightarrow 'a) \Rightarrow \text{bool}$ **where** *has-least-prefixpoint*

definition *has-greatest-postfixpoint* :: ('a ⇒ 'a) ⇒ bool **where**
has-greatest-postfixpoint f ≡ ∃ x . *is-greatest-postfixpoint* f x

The actual least/greatest (pre/post)fixpoints of a given function *f* are extracted by the following operators.

definition *the-least-fixpoint* :: ('a ⇒ 'a) ⇒ 'a (μ - [201] 200) **where** μ f
= (THE x . *is-least-fixpoint* f x)

definition *the-greatest-fixpoint* :: ('a ⇒ 'a) ⇒ 'a (ν - [201] 200) **where** ν f
= (THE x . *is-greatest-fixpoint* f x)

definition *the-least-prefixpoint* :: ('a ⇒ 'a) ⇒ 'a (pμ - [201] 200) **where** pμ f
= (THE x . *is-least-prefixpoint* f x)

definition *the-greatest-postfixpoint* :: ('a ⇒ 'a) ⇒ 'a (pν - [201] 200) **where** pν f
= (THE x . *is-greatest-postfixpoint* f x)

We start with basic consequences of the above definitions.

lemma *least-fixpoint-unique*:
has-least-fixpoint f ⇒ ∃!x . *is-least-fixpoint* f x
⟨proof⟩

lemma *greatest-fixpoint-unique*:
has-greatest-fixpoint f ⇒ ∃!x . *is-greatest-fixpoint* f x
⟨proof⟩

lemma *least-prefixpoint-unique*:
has-least-prefixpoint f ⇒ ∃!x . *is-least-prefixpoint* f x
⟨proof⟩

lemma *greatest-postfixpoint-unique*:
has-greatest-postfixpoint f ⇒ ∃!x . *is-greatest-postfixpoint* f x
⟨proof⟩

lemma *least-fixpoint*:
has-least-fixpoint f ⇒ *is-least-fixpoint* f (μ f)
⟨proof⟩

lemma *greatest-fixpoint*:
has-greatest-fixpoint f ⇒ *is-greatest-fixpoint* f (ν f)
⟨proof⟩

lemma *least-prefixpoint*:
has-least-prefixpoint f ⇒ *is-least-prefixpoint* f (pμ f)
⟨proof⟩

lemma *greatest-postfixpoint*:
has-greatest-postfixpoint f ⇒ *is-greatest-postfixpoint* f (pν f)
⟨proof⟩

lemma *least-fixpoint-same*:
is-least-fixpoint f x ⇒ x = μ f

<proof>

lemma *greatest-fixpoint-same:*

is-greatest-fixpoint f x \implies x = ν f

<proof>

lemma *least-prefixpoint-same:*

is-least-prefixpoint f x \implies x = $p\mu$ f

<proof>

lemma *greatest-postfixpoint-same:*

is-greatest-postfixpoint f x \implies x = $p\nu$ f

<proof>

lemma *least-fixpoint-char:*

is-least-fixpoint f x \longleftrightarrow has-least-fixpoint f \wedge x = μ f

<proof>

lemma *least-prefixpoint-char:*

is-least-prefixpoint f x \longleftrightarrow has-least-prefixpoint f \wedge x = $p\mu$ f

<proof>

lemma *greatest-fixpoint-char:*

is-greatest-fixpoint f x \longleftrightarrow has-greatest-fixpoint f \wedge x = ν f

<proof>

lemma *greatest-postfixpoint-char:*

is-greatest-postfixpoint f x \longleftrightarrow has-greatest-postfixpoint f \wedge x = $p\nu$ f

<proof>

Next come the unfold rules for least/greatest (pre/post)fixpoints.

lemma *μ -unfold:*

has-least-fixpoint f \implies f (μ f) = μ f

<proof>

lemma *$p\mu$ -unfold:*

has-least-prefixpoint f \implies f ($p\mu$ f) \leq $p\mu$ f

<proof>

lemma *ν -unfold:*

has-greatest-fixpoint f \implies ν f = f (ν f)

<proof>

lemma *$p\nu$ -unfold:*

has-greatest-postfixpoint f \implies $p\nu$ f \leq f ($p\nu$ f)

<proof>

Pre-/postfixpoints of isotone functions are fixpoints.

lemma *least-prefixpoint-fixpoint:*

has-least-prefixpoint f \implies *isotone f* \implies *is-least-fixpoint f* ($p\mu f$)
(proof)

lemma *pmu-mu*:

has-least-prefixpoint f \implies *isotone f* $\implies p\mu f = \mu f$
(proof)

lemma *greatest-postfixpoint-fixpoint*:

has-greatest-postfixpoint f \implies *isotone f* \implies *is-greatest-fixpoint f* ($p\nu f$)
(proof)

lemma *pnu-nu*:

has-greatest-postfixpoint f \implies *isotone f* $\implies p\nu f = \nu f$
(proof)

The fixpoint operators preserve isotonicity.

lemma *pmu-isotone*:

has-least-prefixpoint f \implies *has-least-prefixpoint g* $\implies f \leq\leq g \implies p\mu f \leq p\mu g$
(proof)

lemma *mu-isotone*:

has-least-prefixpoint f \implies *has-least-prefixpoint g* \implies *isotone f* \implies *isotone g*
 $\implies f \leq\leq g \implies \mu f \leq \mu g$
(proof)

lemma *pnu-isotone*:

has-greatest-postfixpoint f \implies *has-greatest-postfixpoint g* $\implies f \leq\leq g \implies p\nu f$
 $\leq p\nu g$
(proof)

lemma *nu-isotone*:

has-greatest-postfixpoint f \implies *has-greatest-postfixpoint g* \implies *isotone f* \implies
isotone g $\implies f \leq\leq g \implies \nu f \leq \nu g$
(proof)

The square rule for fixpoints of a function applied twice.

lemma *mu-square*:

isotone f \implies *has-least-fixpoint f* \implies *has-least-fixpoint (f o f)* $\implies \mu f = \mu (f \circ f)$
(proof)

lemma *nu-square*:

isotone f \implies *has-greatest-fixpoint f* \implies *has-greatest-fixpoint (f o f)* $\implies \nu f =$
 $\nu (f \circ f)$
(proof)

The rolling rule for fixpoints of the composition of two functions.

lemma *mu-roll*:

assumes *isotone g*

and *has-least-fixpoint* $(f \circ g)$
and *has-least-fixpoint* $(g \circ f)$
shows $\mu (g \circ f) = g (\mu (f \circ g))$
 <proof>

lemma *nu-roll*:

assumes *isotone* g
and *has-greatest-fixpoint* $(f \circ g)$
and *has-greatest-fixpoint* $(g \circ f)$
shows $\nu (g \circ f) = g (\nu (f \circ g))$
 <proof>

Least (pre)fixpoints are below greatest (post)fixpoints.

lemma *mu-below-nu*:

has-least-fixpoint $f \implies \text{has-greatest-fixpoint } f \implies \mu f \leq \nu f$
 <proof>

lemma *pmu-below-pnu-fix*:

has-fixpoint $f \implies \text{has-least-prefixpoint } f \implies \text{has-greatest-postfixpoint } f \implies p\mu f \leq p\nu f$
 <proof>

lemma *pmu-below-pnu-iso*:

isotone $f \implies \text{has-least-prefixpoint } f \implies \text{has-greatest-postfixpoint } f \implies p\mu f \leq p\nu f$
 <proof>

Several variants of the fusion rule for fixpoints follow.

lemma *mu-fusion-1*:

assumes *galois* $l u$
and *isotone* h
and *has-least-prefixpoint* g
and *has-least-fixpoint* h
and $l (g (u (\mu h))) \leq h (l (u (\mu h)))$
shows $l (p\mu g) \leq \mu h$
 <proof>

lemma *mu-fusion-2*:

galois $l u \implies \text{isotone } h \implies \text{has-least-prefixpoint } g \implies \text{has-least-fixpoint } h \implies l \circ g \leq h \circ l \implies l (p\mu g) \leq \mu h$
 <proof>

lemma *mu-fusion-equal-1*:

galois $l u \implies \text{isotone } g \implies \text{isotone } h \implies \text{has-least-prefixpoint } g \implies \text{has-least-fixpoint } h \implies l (g (u (\mu h))) \leq h (l (u (\mu h))) \implies l (g (p\mu g)) = h (l (p\mu g)) \implies \mu h = l (p\mu g) \wedge \mu h = l (\mu g)$
 <proof>

lemma *mu-fusion-equal-2*:

$galois\ l\ u \implies isotone\ h \implies has\text{-}least\text{-}prefixpoint\ g \implies has\text{-}least\text{-}prefixpoint\ h$
 $\implies l\ (g\ (u\ (\mu\ h))) \leq h\ (l\ (u\ (\mu\ h))) \wedge l\ (g\ (p\mu\ g)) = h\ (l\ (p\mu\ g)) \implies p\mu\ h = l$
 $(p\mu\ g) \wedge \mu\ h = l\ (p\mu\ g)$
 ⟨proof⟩

lemma *mu-fusion-equal-3:*

assumes *galois* $l\ u$
 and *isotone* g
 and *isotone* h
 and *has-least-prefixpoint* g
 and *has-least-fixpoint* h
 and $l \circ g = h \circ l$
 shows $\mu\ h = l\ (p\mu\ g)$
 and $\mu\ h = l\ (\mu\ g)$
 ⟨proof⟩

lemma *mu-fusion-equal-4:*

assumes *galois* $l\ u$
 and *isotone* h
 and *has-least-prefixpoint* g
 and *has-least-prefixpoint* h
 and $l \circ g = h \circ l$
 shows $p\mu\ h = l\ (p\mu\ g)$
 and $\mu\ h = l\ (p\mu\ g)$
 ⟨proof⟩

lemma *nu-fusion-1:*

assumes *galois* $l\ u$
 and *isotone* h
 and *has-greatest-postfixpoint* g
 and *has-greatest-fixpoint* h
 and $h\ (u\ (l\ (\nu\ h))) \leq u\ (g\ (l\ (\nu\ h)))$
 shows $\nu\ h \leq u\ (p\nu\ g)$
 ⟨proof⟩

lemma *nu-fusion-2:*

$galois\ l\ u \implies isotone\ h \implies has\text{-}greatest\text{-}postfixpoint\ g \implies has\text{-}greatest\text{-}fixpoint\ h$
 $\implies h \circ u \leq u \circ g \implies \nu\ h \leq u\ (p\nu\ g)$
 ⟨proof⟩

lemma *nu-fusion-equal-1:*

$galois\ l\ u \implies isotone\ g \implies isotone\ h \implies has\text{-}greatest\text{-}postfixpoint\ g \implies$
 $has\text{-}greatest\text{-}fixpoint\ h \implies h\ (u\ (l\ (\nu\ h))) \leq u\ (g\ (l\ (\nu\ h))) \implies h\ (u\ (p\nu\ g)) = u$
 $(g\ (p\nu\ g)) \implies \nu\ h = u\ (p\nu\ g) \wedge \nu\ h = u\ (\nu\ g)$
 ⟨proof⟩

lemma *nu-fusion-equal-2:*

$galois\ l\ u \implies isotone\ h \implies has\text{-}greatest\text{-}postfixpoint\ g \implies$
 $has\text{-}greatest\text{-}postfixpoint\ h \implies h\ (u\ (l\ (\nu\ h))) \leq u\ (g\ (l\ (\nu\ h))) \wedge h\ (u\ (p\nu\ g)) =$

$u (g (p\nu g)) \implies p\nu h = u (p\nu g) \wedge \nu h = u (p\nu g)$
 ⟨proof⟩

lemma *nu-fusion-equal-3*:

assumes *galois* $l\ u$
 and *isotone* g
 and *isotone* h
 and *has-greatest-postfixpoint* g
 and *has-greatest-fixpoint* h
 and $h \circ u = u \circ g$
 shows $\nu h = u (p\nu g)$
 and $\nu h = u (\nu g)$
 ⟨proof⟩

lemma *nu-fusion-equal-4*:

assumes *galois* $l\ u$
 and *isotone* h
 and *has-greatest-postfixpoint* g
 and *has-greatest-postfixpoint* h
 and $h \circ u = u \circ g$
 shows $p\nu h = u (p\nu g)$
 and $\nu h = u (p\nu g)$
 ⟨proof⟩

Next come the exchange rules for replacing the first/second function in a composition.

lemma *mu-exchange-1*:

assumes *galois* $l\ u$
 and *isotone* g
 and *isotone* h
 and *has-least-prefixpoint* $(l \circ h)$
 and *has-least-prefixpoint* $(h \circ g)$
 and *has-least-fixpoint* $(g \circ h)$
 and $l \circ h \circ g \leq g \circ h \circ l$
 shows $\mu (l \circ h) \leq \mu (g \circ h)$
 ⟨proof⟩

lemma *mu-exchange-2*:

assumes *galois* $l\ u$
 and *isotone* g
 and *isotone* h
 and *has-least-prefixpoint* $(l \circ h)$
 and *has-least-prefixpoint* $(h \circ l)$
 and *has-least-prefixpoint* $(h \circ g)$
 and *has-least-fixpoint* $(g \circ h)$
 and *has-least-fixpoint* $(h \circ g)$
 and $l \circ h \circ g \leq g \circ h \circ l$
 shows $\mu (h \circ l) \leq \mu (h \circ g)$
 ⟨proof⟩

lemma *mu-exchange-equal*:

assumes *galois* $l\ u$
and *galois* $k\ t$
and *isotone* h
and *has-least-prefixpoint* $(l \circ h)$
and *has-least-prefixpoint* $(h \circ l)$
and *has-least-prefixpoint* $(k \circ h)$
and *has-least-prefixpoint* $(h \circ k)$
and $l \circ h \circ k = k \circ h \circ l$
shows $\mu(l \circ h) = \mu(k \circ h)$
and $\mu(h \circ l) = \mu(h \circ k)$

<proof>

lemma *nu-exchange-1*:

assumes *galois* $l\ u$
and *isotone* g
and *isotone* h
and *has-greatest-postfixpoint* $(u \circ h)$
and *has-greatest-postfixpoint* $(h \circ g)$
and *has-greatest-fixpoint* $(g \circ h)$
and $g \circ h \circ u \leq u \circ h \circ g$
shows $\nu(g \circ h) \leq \nu(u \circ h)$

<proof>

lemma *nu-exchange-2*:

assumes *galois* $l\ u$
and *isotone* g
and *isotone* h
and *has-greatest-postfixpoint* $(u \circ h)$
and *has-greatest-postfixpoint* $(h \circ u)$
and *has-greatest-postfixpoint* $(h \circ g)$
and *has-greatest-fixpoint* $(g \circ h)$
and *has-greatest-fixpoint* $(h \circ g)$
and $g \circ h \circ u \leq u \circ h \circ g$
shows $\nu(h \circ g) \leq \nu(h \circ u)$

<proof>

lemma *nu-exchange-equal*:

assumes *galois* $l\ u$
and *galois* $k\ t$
and *isotone* h
and *has-greatest-postfixpoint* $(u \circ h)$
and *has-greatest-postfixpoint* $(h \circ u)$
and *has-greatest-postfixpoint* $(t \circ h)$
and *has-greatest-postfixpoint* $(h \circ t)$
and $u \circ h \circ t = t \circ h \circ u$
shows $\nu(u \circ h) = \nu(t \circ h)$
and $\nu(h \circ u) = \nu(h \circ t)$

<proof>

The following results generalise parts of [10, Exercise 8.27] from continuous functions on complete partial orders to the present setting.

lemma *mu-commute-fixpoint-1:*

$isotone\ f \implies has\ least\ fixpoint\ (f \circ g) \implies f \circ g = g \circ f \implies is\ fixpoint\ f\ (\mu\ (f \circ g))$
<proof>

lemma *mu-commute-fixpoint-2:*

$isotone\ g \implies has\ least\ fixpoint\ (f \circ g) \implies f \circ g = g \circ f \implies is\ fixpoint\ g\ (\mu\ (f \circ g))$
<proof>

lemma *mu-commute-least-fixpoint:*

$isotone\ f \implies isotone\ g \implies has\ least\ fixpoint\ f \implies has\ least\ fixpoint\ g \implies has\ least\ fixpoint\ (f \circ g) \implies f \circ g = g \circ f \implies \mu\ (f \circ g) = \mu\ f \implies \mu\ g \leq \mu\ f$
<proof>

The converse of the preceding result is claimed for continuous f, g on a complete partial order; it is unknown whether it holds without these additional assumptions.

lemma *nu-commute-fixpoint-1:*

$isotone\ f \implies has\ greatest\ fixpoint\ (f \circ g) \implies f \circ g = g \circ f \implies is\ fixpoint\ f\ (\nu\ (f \circ g))$
<proof>

lemma *nu-commute-fixpoint-2:*

$isotone\ g \implies has\ greatest\ fixpoint\ (f \circ g) \implies f \circ g = g \circ f \implies is\ fixpoint\ g\ (\nu\ (f \circ g))$
<proof>

lemma *nu-commute-greatest-fixpoint:*

$isotone\ f \implies isotone\ g \implies has\ greatest\ fixpoint\ f \implies has\ greatest\ fixpoint\ g \implies has\ greatest\ fixpoint\ (f \circ g) \implies f \circ g = g \circ f \implies \nu\ (f \circ g) = \nu\ f \implies \nu\ f \leq \nu\ g$
<proof>

Finally, we show a number of versions of the diagonal rule for functions with two arguments.

lemma *mu-diagonal-1:*

assumes $isotone\ (\lambda x . \mu\ (\lambda y . f\ x\ y))$
and $\forall x . has\ least\ fixpoint\ (\lambda y . f\ x\ y)$
and $has\ least\ prefixpoint\ (\lambda x . \mu\ (\lambda y . f\ x\ y))$
shows $\mu\ (\lambda x . f\ x\ x) = \mu\ (\lambda x . \mu\ (\lambda y . f\ x\ y))$
<proof>

lemma *mu-diagonal-2:*

$\forall x . \text{isotone } (\lambda y . f x y) \wedge \text{isotone } (\lambda y . f y x) \wedge \text{has-least-prefixpoint } (\lambda y . f x y) \implies \text{has-least-prefixpoint } (\lambda x . \mu (\lambda y . f x y)) \implies \mu (\lambda x . f x x) = \mu (\lambda x . \mu (\lambda y . f x y))$
 <proof>

lemma *nu-diagonal-1*:

assumes *isotone* $(\lambda x . \nu (\lambda y . f x y))$
and $\forall x . \text{has-greatest-fixpoint } (\lambda y . f x y)$
and *has-greatest-postfixpoint* $(\lambda x . \nu (\lambda y . f x y))$
shows $\nu (\lambda x . f x x) = \nu (\lambda x . \nu (\lambda y . f x y))$
 <proof>

lemma *nu-diagonal-2*:

$\forall x . \text{isotone } (\lambda y . f x y) \wedge \text{isotone } (\lambda y . f y x) \wedge \text{has-greatest-postfixpoint } (\lambda y . f x y) \implies \text{has-greatest-postfixpoint } (\lambda x . \nu (\lambda y . f x y)) \implies \nu (\lambda x . f x x) = \nu (\lambda x . \nu (\lambda y . f x y))$
 <proof>

end

end

3 Semirings

This theory develops a hierarchy of idempotent semirings. All kinds of semiring considered here are bounded semilattices, but many lack additional properties typically assumed for semirings. In particular, we consider the variants of semirings, in which

- * multiplication is not required to be associative;
- * a right zero and unit of multiplication need not exist;
- * multiplication has a left residual;
- * multiplication from the left is not required to distribute over addition;
- * the semilattice order has a greatest element.

We have applied results from this theory a number of papers for unifying computation models. For example, see [13] for various relational and matrix-based computation models and [6] for multirelational models.

The main results in this theory relate different ways of defining reflexive-transitive closures as discussed in [6].

theory *Semirings*

imports *Fixpoints*

begin

3.1 Idempotent Semirings

The following definitions are standard for relations. Putting them into a general class that depends only on the signature facilitates reuse. Coreflexives are sometimes called partial identities, subidentities, monotypes or tests.

```

class times-one-ord = times + one + ord
begin

abbreviation reflexive :: 'a ⇒ bool where reflexive x ≡ 1 ≤ x
abbreviation coreflexive :: 'a ⇒ bool where coreflexive x ≡ x ≤ 1

abbreviation transitive :: 'a ⇒ bool where transitive x ≡ x * x ≤ x
abbreviation dense-rel :: 'a ⇒ bool where dense-rel x ≡ x ≤ x * x
abbreviation idempotent :: 'a ⇒ bool where idempotent x ≡ x * x = x

abbreviation preorder :: 'a ⇒ bool where preorder x ≡ reflexive x ∧
transitive x

abbreviation coreflexives ≡ { x . coreflexive x }

end

```

The first algebra is a very weak idempotent semiring, in which multiplication is not necessarily associative.

```

class non-associative-left-semiring = bounded-semilattice-sup-bot + times + one
+
  assumes mult-left-sub-dist-sup: x * y ⊔ x * z ≤ x * (y ⊔ z)
  assumes mult-right-dist-sup: (x ⊔ y) * z = x * z ⊔ y * z
  assumes mult-left-zero [simp]: bot * x = bot
  assumes mult-left-one [simp]: 1 * x = x
  assumes mult-sub-right-one: x ≤ x * 1
begin

subclass times-one-ord ⟨proof⟩

```

We first show basic isotonicity and subdistributivity properties of multiplication.

lemma *mult-left-isotone*:
 $x \leq y \implies x * z \leq y * z$
 ⟨proof⟩

lemma *mult-right-isotone*:
 $x \leq y \implies z * x \leq z * y$
 ⟨proof⟩

lemma *mult-isotone*:
 $w \leq y \implies x \leq z \implies w * x \leq y * z$
 ⟨proof⟩

lemma *affine-isotone*:

isotone $(\lambda x . y * x \sqcup z)$

<proof>

lemma *mult-left-sub-dist-sup-left*:

$x * y \leq x * (y \sqcup z)$

<proof>

lemma *mult-left-sub-dist-sup-right*:

$x * z \leq x * (y \sqcup z)$

<proof>

lemma *mult-right-sub-dist-sup-left*:

$x * z \leq (x \sqcup y) * z$

<proof>

lemma *mult-right-sub-dist-sup-right*:

$y * z \leq (x \sqcup y) * z$

<proof>

lemma *case-split-left*:

assumes $1 \leq w \sqcup z$

and $w * x \leq y$

and $z * x \leq y$

shows $x \leq y$

<proof>

lemma *case-split-left-equal*:

$w \sqcup z = 1 \implies w * x = w * y \implies z * x = z * y \implies x = y$

<proof>

Next we consider under which semiring operations the above properties are closed.

lemma *reflexive-one-closed*:

reflexive 1

<proof>

lemma *reflexive-sup-closed*:

reflexive $x \implies \text{reflexive } (x \sqcup y)$

<proof>

lemma *reflexive-mult-closed*:

reflexive $x \implies \text{reflexive } y \implies \text{reflexive } (x * y)$

<proof>

lemma *coreflexive-bot-closed*:

coreflexive bot

<proof>

lemma *coreflexive-one-closed:*

coreflexive 1

<proof>

lemma *coreflexive-sup-closed:*

coreflexive x \implies coreflexive y \implies coreflexive (x \sqcup y)

<proof>

lemma *coreflexive-mult-closed:*

*coreflexive x \implies coreflexive y \implies coreflexive (x * y)*

<proof>

lemma *transitive-bot-closed:*

transitive bot

<proof>

lemma *transitive-one-closed:*

transitive 1

<proof>

lemma *dense-bot-closed:*

dense-rel bot

<proof>

lemma *dense-one-closed:*

dense-rel 1

<proof>

lemma *dense-sup-closed:*

dense-rel x \implies dense-rel y \implies dense-rel (x \sqcup y)

<proof>

lemma *idempotent-bot-closed:*

idempotent bot

<proof>

lemma *idempotent-one-closed:*

idempotent 1

<proof>

lemma *preorder-one-closed:*

preorder 1

<proof>

lemma *coreflexive-transitive:*

coreflexive x \implies transitive x

<proof>

lemma *preorder-idempotent*:
preorder $x \implies$ *idempotent* x
 ⟨*proof*⟩

We study the following three ways of defining reflexive-transitive closures. Each of them is given as a least prefixpoint, but the underlying functions are different. They implement left recursion, right recursion and symmetric recursion, respectively.

abbreviation $Lf :: 'a \Rightarrow ('a \Rightarrow 'a)$ **where** $Lf\ y \equiv (\lambda x . 1 \sqcup x * y)$
abbreviation $Rf :: 'a \Rightarrow ('a \Rightarrow 'a)$ **where** $Rf\ y \equiv (\lambda x . 1 \sqcup y * x)$
abbreviation $Sf :: 'a \Rightarrow ('a \Rightarrow 'a)$ **where** $Sf\ y \equiv (\lambda x . 1 \sqcup y \sqcup x * x)$

abbreviation $lstar :: 'a \Rightarrow 'a$ **where** $lstar\ y \equiv p\mu\ (Lf\ y)$
abbreviation $rstar :: 'a \Rightarrow 'a$ **where** $rstar\ y \equiv p\mu\ (Rf\ y)$
abbreviation $sstar :: 'a \Rightarrow 'a$ **where** $sstar\ y \equiv p\mu\ (Sf\ y)$

All functions are isotone and, therefore, if the prefixpoints exist they are also fixpoints.

lemma *lstar-rec-isotone*:
isotone $(Lf\ y)$
 ⟨*proof*⟩

lemma *rstar-rec-isotone*:
isotone $(Rf\ y)$
 ⟨*proof*⟩

lemma *sstar-rec-isotone*:
isotone $(Sf\ y)$
 ⟨*proof*⟩

lemma *lstar-fixpoint*:
has-least-prefixpoint $(Lf\ y) \implies lstar\ y = \mu\ (Lf\ y)$
 ⟨*proof*⟩

lemma *rstar-fixpoint*:
has-least-prefixpoint $(Rf\ y) \implies rstar\ y = \mu\ (Rf\ y)$
 ⟨*proof*⟩

lemma *sstar-fixpoint*:
has-least-prefixpoint $(Sf\ y) \implies sstar\ y = \mu\ (Sf\ y)$
 ⟨*proof*⟩

lemma *sstar-increasing*:
has-least-prefixpoint $(Sf\ y) \implies y \leq sstar\ y$
 ⟨*proof*⟩

The fixpoint given by right recursion is always below the one given by symmetric recursion.

lemma *rstar-below-sstar*:

```

assumes has-least-prefixpoint (Rf y)
and has-least-prefixpoint (Sf y)
shows rstar y  $\leq$  sstar y
<proof>

```

end

Our next structure adds one half of the associativity property. This inequality holds, for example, for multirelations under the compositions defined by Parikh and Peleg [23, 25]. The converse inequality requires up-closed multirelations for Parikh's composition.

```

class pre-left-semiring = non-associative-left-semiring +
assumes mult-semi-associative:  $(x * y) * z \leq x * (y * z)$ 
begin

```

```

lemma mult-one-associative [simp]:
 $x * 1 * y = x * y$ 
<proof>

```

```

lemma mult-sup-associative-one:
 $(x * (y * 1)) * z \leq x * (y * z)$ 
<proof>

```

```

lemma rstar-increasing:
assumes has-least-prefixpoint (Rf y)
shows  $y \leq rstar y$ 
<proof>

```

end

For the next structure we add a left residual operation. Such a residual is available, for example, for multirelations.

The operator notation for binary division is introduced in a class that requires a unary inverse. This is appropriate for fields, but too strong in the present context of semirings. We therefore reintroduce it without requiring a unary inverse.

```

no-notation
inverse-divide (infixl  $'/$  70)

```

```

notation
divide (infixl  $'/$  70)

```

```

class residuated-pre-left-semiring = pre-left-semiring + divide +
assumes lres-galois:  $x * y \leq z \iff x \leq z / y$ 
begin

```

We first derive basic properties of left residuals from the Galois connection.

lemma *lres-left-isotone*:
 $x \leq y \implies x / z \leq y / z$
 ⟨proof⟩

lemma *lres-right-antitone*:
 $x \leq y \implies z / y \leq z / x$
 ⟨proof⟩

lemma *lres-inverse*:
 $(x / y) * y \leq x$
 ⟨proof⟩

lemma *lres-one*:
 $x / 1 \leq x$
 ⟨proof⟩

lemma *lres-mult-sub-lres-lres*:
 $x / (z * y) \leq (x / y) / z$
 ⟨proof⟩

lemma *mult-lres-sub-assoc*:
 $x * (y / z) \leq (x * y) / z$
 ⟨proof⟩

With the help of a left residual, it follows that left recursion is below right recursion.

lemma *lstar-below-rstar*:
assumes *has-least-prefixpoint* (*Lf* *y*)
and *has-least-prefixpoint* (*Rf* *y*)
shows *lstar* *y* \leq *rstar* *y*
 ⟨proof⟩

Moreover, right recursion gives the same result as symmetric recursion. The next proof follows an argument of [5, Satz 10.1.5].

lemma *rstar-sstar*:
assumes *has-least-prefixpoint* (*Rf* *y*)
and *has-least-prefixpoint* (*Sf* *y*)
shows *rstar* *y* = *sstar* *y*
 ⟨proof⟩

end

In the next structure we add full associativity of multiplication, as well as a right unit. Still, multiplication does not need to have a right zero and does not need to distribute over addition from the left.

class *idempotent-left-semiring* = *non-associative-left-semiring* + *monoid-mult*
begin

subclass *pre-left-semiring*

<proof>

lemma *zero-right-mult-decreasing*:

$x * bot \leq x$

<proof>

The following result shows that for dense coreflexives there are two equivalent ways to express that a property is preserved. In the setting of Kleene algebras, this is well known for tests, which form a Boolean subalgebra. The point here is that only very few properties of tests are needed to show the equivalence.

lemma *test-preserves-equation*:

assumes *dense-rel* p

and *coreflexive* p

shows $p * x \leq x * p \longleftrightarrow p * x = p * x * p$

<proof>

end

The next structure has both distributivity properties of multiplication. Only a right zero is missing from full semirings. This is important as many computation models do not have a right zero of sequential composition.

class *idempotent-left-zero-semiring* = *idempotent-left-semiring* +

assumes *mult-left-dist-sup*: $x * (y \sqcup z) = x * y \sqcup x * z$

begin

lemma *case-split-right*:

assumes $1 \leq w \sqcup z$

and $x * w \leq y$

and $x * z \leq y$

shows $x \leq y$

<proof>

lemma *case-split-right-equal*:

$w \sqcup z = 1 \implies x * w = y * w \implies x * z = y * z \implies x = y$

<proof>

This is the first structure we can connect to the semirings provided by Isabelle/HOL.

sublocale *semiring*: *ordered-semiring sup bot less-eq less times*

<proof>

sublocale *semiring*: *semiring-numeral 1 times sup* *<proof>*

end

Completing this part of the hierarchy, we obtain idempotent semirings by adding a right zero of multiplication.

```

class idempotent-semiring = idempotent-left-zero-semiring +
  assumes mult-right-zero [simp]:  $x * \text{bot} = \text{bot}$ 
begin

sublocale semiring: semiring-0 sup bot times
  ⟨proof⟩

end

```

3.2 Bounded Idempotent Semirings

All of the following semirings have a greatest element in the underlying semi-lattice order. With this element, we can express further standard properties of relations. We extend each class in the above hierarchy in turn.

```

class times-top = times + top
begin

abbreviation vector    :: 'a ⇒ bool where vector x    ≡ x * top = x
abbreviation covector :: 'a ⇒ bool where covector x    ≡ top * x = x
abbreviation total     :: 'a ⇒ bool where total x      ≡ x * top = top
abbreviation surjective :: 'a ⇒ bool where surjective x ≡ top * x = top

abbreviation vectors  ≡ { x . vector x }
abbreviation covectors ≡ { x . covector x }

end

```

```

class bounded-non-associative-left-semiring = non-associative-left-semiring + top
+
  assumes sup-right-top [simp]:  $x \sqcup \text{top} = \text{top}$ 
begin

subclass times-top ⟨proof⟩

```

We first give basic properties of the greatest element.

```

lemma sup-left-top [simp]:
  top  $\sqcup$  x = top
  ⟨proof⟩

```

```

lemma top-greatest [simp]:
   $x \leq \text{top}$ 
  ⟨proof⟩

```

```

lemma top-left-mult-increasing:
   $x \leq \text{top} * x$ 
  ⟨proof⟩

```

```

lemma top-right-mult-increasing:
   $x \leq x * \text{top}$ 

```

<proof>

lemma *top-mult-top* [*simp*]:

*top * top = top*

<proof>

Closure of the above properties under the semiring operations is considered next.

lemma *vector-bot-closed*:

vector bot

<proof>

lemma *vector-top-closed*:

vector top

<proof>

lemma *vector-sup-closed*:

vector x \implies vector y \implies vector (x \sqcup y)

<proof>

lemma *covector-top-closed*:

covector top

<proof>

lemma *total-one-closed*:

total 1

<proof>

lemma *total-top-closed*:

total top

<proof>

lemma *total-sup-closed*:

total x \implies total (x \sqcup y)

<proof>

lemma *surjective-one-closed*:

surjective 1

<proof>

lemma *surjective-top-closed*:

surjective top

<proof>

lemma *surjective-sup-closed*:

surjective x \implies surjective (x \sqcup y)

<proof>

lemma *reflexive-top-closed*:

reflexive top
<proof>

lemma *transitive-top-closed*:
transitive top
<proof>

lemma *dense-top-closed*:
dense-rel top
<proof>

lemma *idempotent-top-closed*:
idempotent top
<proof>

lemma *preorder-top-closed*:
preorder top
<proof>

end

Some closure properties require at least half of associativity.

class *bounded-pre-left-semiring* = *pre-left-semiring* +
bounded-non-associative-left-semiring
begin

lemma *vector-mult-closed*:
vector y \implies *vector (x * y)*
<proof>

lemma *surjective-mult-closed*:
surjective x \implies *surjective y* \implies *surjective (x * y)*
<proof>

end

We next consider residuals with the greatest element.

class *bounded-residuated-pre-left-semiring* = *residuated-pre-left-semiring* +
bounded-pre-left-semiring
begin

lemma *lres-top-decreasing*:
x / top \leq *x*
<proof>

lemma *top-lres-absorb* [*simp*]:
top / x = *top*
<proof>

lemma *covector-lres-closed*:
covector $x \implies \text{covector } (x / y)$
 ⟨*proof*⟩

end

Some closure properties require full associativity.

class *bounded-idempotent-left-semiring* = *bounded-pre-left-semiring* +
idempotent-left-semiring
begin

lemma *covector-mult-closed*:
covector $x \implies \text{covector } (x * y)$
 ⟨*proof*⟩

lemma *total-mult-closed*:
total $x \implies \text{total } y \implies \text{total } (x * y)$
 ⟨*proof*⟩

end

Some closure properties require distributivity from the left.

class *bounded-idempotent-left-zero-semiring* = *bounded-idempotent-left-semiring* +
idempotent-left-zero-semiring
begin

lemma *covector-sup-closed*:
covector $x \implies \text{covector } y \implies \text{covector } (x \sqcup y)$
 ⟨*proof*⟩

end

Our final structure is an idempotent semiring with a greatest element.

class *bounded-idempotent-semiring* = *bounded-idempotent-left-zero-semiring* +
idempotent-semiring
begin

lemma *covector-bot-closed*:
covector *bot*
 ⟨*proof*⟩

end

end

4 Relation Algebras

The main structures introduced in this theory are Stone relation algebras. They generalise Tarski's relation algebras [28] by weakening the Boolean

algebra lattice structure to a Stone algebra. Our motivation is to generalise relation-algebraic methods from unweighted graphs to weighted graphs. Unlike unweighted graphs, weighted graphs do not form a Boolean algebra because there is no complement operation on the edge weights. However, edge weights form a Stone algebra, and matrices over edge weights (that is, weighted graphs) form a Stone relation algebra.

The development in this theory is described in our papers [14, 16]. Our main application there is the verification of Prim’s minimum spanning tree algorithm. Related work about fuzzy relations [12, 29], Dedekind categories [18] and rough relations [9, 24] is also discussed in these papers. In particular, Stone relation algebras do not assume that the underlying lattice is complete or a Heyting algebra, and they do not assume that composition has residuals.

We proceed in two steps. First, we study the positive fragment in the form of single-object bounded distributive allegories [11]. Second, we extend these structures by a pseudocomplement operation with additional axioms to obtain Stone relation algebras.

Tarski’s relation algebras are then obtained by a simple extension that imposes a Boolean algebra. See, for example, [7, 17, 20, 21, 26, 27] for further details about relations and relation algebras, and [2, 8] for algebras of relations with a smaller signature.

theory *Relation-Algebras*

imports *Stone-Algebras.P-Algebras Semirings*

begin

4.1 Single-Object Bounded Distributive Allegories

We start with developing bounded distributive allegories. The following definitions concern properties of relations that require converse in addition to lattice and semiring operations.

class *conv* =

fixes *conv* :: 'a ⇒ 'a (-^T [100] 100)

class *bounded-distrib-allegory-signature* = *inf* + *sup* + *times* + *conv* + *bot* + *top* + *one* + *ord*

begin

subclass *times-one-ord* ⟨*proof*⟩

subclass *times-top* ⟨*proof*⟩

abbreviation *total-var* :: 'a ⇒ bool **where** *total-var* *x* ≡ 1 ≤ *x* * *x*^T

abbreviation *surjective-var* :: 'a ⇒ bool **where** *surjective-var* *x* ≡ 1 ≤ *x*^T * *x*

abbreviation *univalent* :: 'a ⇒ bool **where** *univalent* *x* ≡ *x*^T * *x* ≤ 1

abbreviation *injective* :: 'a ⇒ bool **where** *injective* *x* ≡ *x* * *x*^T ≤ 1

abbreviation *mapping* :: 'a ⇒ bool **where** *mapping* x ≡ *univalent* x
 ∧ *total* x
abbreviation *bijjective* :: 'a ⇒ bool **where** *bijjective* x ≡ *injective* x ∧
 surjective x

abbreviation *point* :: 'a ⇒ bool **where** *point* x ≡ *vector* x ∧
 bijjective x
abbreviation *arc* :: 'a ⇒ bool **where** *arc* x ≡ *bijjective* (x *
 top) ∧ *bijjective* (x^T * *top*)

abbreviation *symmetric* :: 'a ⇒ bool **where** *symmetric* x ≡ x^T = x
abbreviation *antisymmetric* :: 'a ⇒ bool **where** *antisymmetric* x ≡ x ⊓ x^T ≤
 1
abbreviation *asymmetric* :: 'a ⇒ bool **where** *asymmetric* x ≡ x ⊓ x^T =
 bot
abbreviation *linear* :: 'a ⇒ bool **where** *linear* x ≡ x ⊔ x^T = *top*

abbreviation *equivalence* :: 'a ⇒ bool **where** *equivalence* x ≡ *preorder* x ∧
 symmetric x
abbreviation *order* :: 'a ⇒ bool **where** *order* x ≡ *preorder* x ∧
 antisymmetric x
abbreviation *linear-order* :: 'a ⇒ bool **where** *linear-order* x ≡ *order* x ∧
 linear x

end

We reuse the relation algebra axioms given in [20] except for one – see lemma *conv-complement-sub* below – which we replace with the Dedekind rule (or modular law) *dedekind-1*. The Dedekind rule or variants of it are known from [7, 11, 19, 27]. We add *comp-left-zero*, which follows in relation algebras but not in the present setting. The main change is that only a bounded distributive lattice is required, not a Boolean algebra.

```

class bounded-distrib-allegory = bounded-distrib-lattice + times + one + conv +
  assumes comp-associative        : (x * y) * z = x * (y * z)
  assumes comp-right-dist-sup    : (x ⊔ y) * z = (x * z) ⊔ (y * z)
  assumes comp-left-zero    [simp]: bot * x = bot
  assumes comp-left-one        [simp]: 1 * x = x
  assumes conv-involutive       [simp]: xTT = x
  assumes conv-dist-sup        : (x ⊔ y)T = xT ⊔ yT
  assumes conv-dist-comp       : (x * y)T = yT * xT
  assumes dedekind-1           : x * y ⊓ z ≤ x * (y ⊓ (xT * z))
begin

```

```

subclass bounded-distrib-allegory-signature <proof>

```

Many properties of relation algebras already follow in bounded distributive allegories.

lemma *conv-isotone*:
 $x \leq y \implies x^T \leq y^T$

$\langle proof \rangle$

lemma *conv-order*:

$$x \leq y \iff x^T \leq y^T$$

$\langle proof \rangle$

lemma *conv-bot* [*simp*]:

$$bot^T = bot$$

$\langle proof \rangle$

lemma *conv-top* [*simp*]:

$$top^T = top$$

$\langle proof \rangle$

lemma *conv-dist-inf*:

$$(x \sqcap y)^T = x^T \sqcap y^T$$

$\langle proof \rangle$

lemma *conv-inf-bot-iff*:

$$bot = x^T \sqcap y \iff bot = x \sqcap y^T$$

$\langle proof \rangle$

lemma *conv-one* [*simp*]:

$$1^T = 1$$

$\langle proof \rangle$

lemma *comp-left-dist-sup*:

$$(x * y) \sqcup (x * z) = x * (y \sqcup z)$$

$\langle proof \rangle$

lemma *comp-right-isotone*:

$$x \leq y \implies z * x \leq z * y$$

$\langle proof \rangle$

lemma *comp-left-isotone*:

$$x \leq y \implies x * z \leq y * z$$

$\langle proof \rangle$

lemma *comp-isotone*:

$$x \leq y \implies w \leq z \implies x * w \leq y * z$$

$\langle proof \rangle$

lemma *comp-left-subdist-inf*:

$$(x \sqcap y) * z \leq x * z \sqcap y * z$$

$\langle proof \rangle$

lemma *comp-left-increasing-sup*:

$$x * y \leq (x \sqcup z) * y$$

$\langle proof \rangle$

lemma *comp-right-subdist-inf*:

$$x * (y \sqcap z) \leq x * y \sqcap x * z$$

<proof>

lemma *comp-right-increasing-sup*:

$$x * y \leq x * (y \sqcup z)$$

<proof>

lemma *comp-right-zero [simp]*:

$$x * \text{bot} = \text{bot}$$

<proof>

lemma *comp-right-one [simp]*:

$$x * 1 = x$$

<proof>

lemma *comp-left-conjugate*:

$$\text{conjugate } (\lambda y . x * y) (\lambda y . x^T * y)$$

<proof>

lemma *comp-right-conjugate*:

$$\text{conjugate } (\lambda y . y * x) (\lambda y . y * x^T)$$

<proof>

We still obtain a semiring structure.

subclass *bounded-idempotent-semiring*

<proof>

sublocale *inf: semiring-0 sup bot inf*

<proof>

lemma *schroeder-1*:

$$x * y \sqcap z = \text{bot} \iff x^T * z \sqcap y = \text{bot}$$

<proof>

lemma *schroeder-2*:

$$x * y \sqcap z = \text{bot} \iff z * y^T \sqcap x = \text{bot}$$

<proof>

lemma *comp-additive*:

$$\text{additive } (\lambda y . x * y) \wedge \text{additive } (\lambda y . x^T * y) \wedge \text{additive } (\lambda y . y * x) \wedge \text{additive } (\lambda y . y * x^T)$$

<proof>

lemma *dedekind-2*:

$$y * x \sqcap z \leq (y \sqcap (z * x^T)) * x$$

<proof>

The intersection with a vector can still be exported from the first argu-

ment of a composition, and many other properties of vectors and covectors continue to hold.

lemma *vector-inf-comp*:

$vector\ x \implies (x \sqcap y) * z = x \sqcap (y * z)$
<proof>

lemma *vector-inf-closed*:

$vector\ x \implies vector\ y \implies vector\ (x \sqcap y)$
<proof>

lemma *vector-inf-one-comp*:

$vector\ x \implies (x \sqcap 1) * y = x \sqcap y$
<proof>

lemma *covector-inf-comp-1*:

assumes *vector* x
shows $(y \sqcap x^T) * z = (y \sqcap x^T) * (x \sqcap z)$
<proof>

lemma *covector-inf-comp-2*:

assumes *vector* x
shows $y * (x \sqcap z) = (y \sqcap x^T) * (x \sqcap z)$
<proof>

lemma *covector-inf-comp-3*:

$vector\ x \implies (y \sqcap x^T) * z = y * (x \sqcap z)$
<proof>

lemma *covector-inf-closed*:

$covector\ x \implies covector\ y \implies covector\ (x \sqcap y)$
<proof>

lemma *vector-conv-covector*:

$vector\ v \longleftrightarrow covector\ (v^T)$
<proof>

lemma *covector-conv-vector*:

$covector\ v \longleftrightarrow vector\ (v^T)$
<proof>

lemma *covector-comp-inf*:

$covector\ z \implies x * (y \sqcap z) = x * y \sqcap z$
<proof>

lemma *vector-restrict-comp-conv*:

$vector\ x \implies x \sqcap y \leq x^T * y$
<proof>

lemma *covector-restrict-comp-conv*:

covector $x \implies y \sqcap x \leq y * x^T$
(proof)

lemma *covector-comp-inf-1*:
covector $x \implies (y \sqcap x) * z = y * (x^T \sqcap z)$
(proof)

We still have two ways to represent surjectivity and totality.

lemma *surjective-var*:
surjective $x \iff \text{surjective-var } x$
(proof)

lemma *total-var*:
total $x \iff \text{total-var } x$
(proof)

lemma *surjective-conv-total*:
surjective $x \iff \text{total } (x^T)$
(proof)

lemma *total-conv-surjective*:
total $x \iff \text{surjective } (x^T)$
(proof)

lemma *injective-conv-univalent*:
injective $x \iff \text{univalent } (x^T)$
(proof)

lemma *univalent-conv-injective*:
univalent $x \iff \text{injective } (x^T)$
(proof)

We continue with studying further closure properties.

lemma *univalent-bot-closed*:
univalent bot
(proof)

lemma *univalent-one-closed*:
univalent 1
(proof)

lemma *univalent-inf-closed*:
univalent $x \implies \text{univalent } (x \sqcap y)$
(proof)

lemma *univalent-mult-closed*:
assumes *univalent* x
and *univalent* y
shows *univalent* $(x * y)$

$\langle \text{proof} \rangle$

lemma *injective-bot-closed:*

injective bot

$\langle \text{proof} \rangle$

lemma *injective-one-closed:*

injective 1

$\langle \text{proof} \rangle$

lemma *injective-inf-closed:*

injective $x \implies \text{injective } (x \sqcap y)$

$\langle \text{proof} \rangle$

lemma *injective-mult-closed:*

*injective $x \implies \text{injective } y \implies \text{injective } (x * y)$*

$\langle \text{proof} \rangle$

lemma *mapping-one-closed:*

mapping 1

$\langle \text{proof} \rangle$

lemma *mapping-mult-closed:*

*mapping $x \implies \text{mapping } y \implies \text{mapping } (x * y)$*

$\langle \text{proof} \rangle$

lemma *bijjective-one-closed:*

bijjective 1

$\langle \text{proof} \rangle$

lemma *bijjective-mult-closed:*

*bijjective $x \implies \text{bijjective } y \implies \text{bijjective } (x * y)$*

$\langle \text{proof} \rangle$

lemma *bijjective-conv-mapping:*

bijjective $x \longleftrightarrow \text{mapping } (x^T)$

$\langle \text{proof} \rangle$

lemma *mapping-conv-bijjective:*

mapping $x \longleftrightarrow \text{bijjective } (x^T)$

$\langle \text{proof} \rangle$

lemma *reflexive-inf-closed:*

reflexive $x \implies \text{reflexive } y \implies \text{reflexive } (x \sqcap y)$

$\langle \text{proof} \rangle$

lemma *reflexive-conv-closed:*

reflexive $x \implies \text{reflexive } (x^T)$

$\langle \text{proof} \rangle$

lemma *coreflexive-inf-closed*:
coreflexive $x \implies \text{coreflexive } (x \sqcap y)$
(proof)

lemma *coreflexive-conv-closed*:
coreflexive $x \implies \text{coreflexive } (x^T)$
(proof)

lemma *coreflexive-symmetric*:
coreflexive $x \implies \text{symmetric } x$
(proof)

lemma *transitive-inf-closed*:
transitive $x \implies \text{transitive } y \implies \text{transitive } (x \sqcap y)$
(proof)

lemma *transitive-conv-closed*:
transitive $x \implies \text{transitive } (x^T)$
(proof)

lemma *dense-conv-closed*:
dense-rel $x \implies \text{dense-rel } (x^T)$
(proof)

lemma *idempotent-conv-closed*:
idempotent $x \implies \text{idempotent } (x^T)$
(proof)

lemma *preorder-inf-closed*:
preorder $x \implies \text{preorder } y \implies \text{preorder } (x \sqcap y)$
(proof)

lemma *preorder-conv-closed*:
preorder $x \implies \text{preorder } (x^T)$
(proof)

lemma *symmetric-bot-closed*:
symmetric bot
(proof)

lemma *symmetric-one-closed*:
symmetric 1
(proof)

lemma *symmetric-top-closed*:
symmetric top
(proof)

lemma *symmetric-inf-closed*:

symmetric $x \implies$ *symmetric* $y \implies$ *symmetric* $(x \sqcap y)$

<proof>

lemma *symmetric-sup-closed*:

symmetric $x \implies$ *symmetric* $y \implies$ *symmetric* $(x \sqcup y)$

<proof>

lemma *symmetric-conv-closed*:

symmetric $x \implies$ *symmetric* (x^T)

<proof>

lemma *one-inf-conv*:

$1 \sqcap x = 1 \sqcap x^T$

<proof>

lemma *antisymmetric-bot-closed*:

antisymmetric bot

<proof>

lemma *antisymmetric-one-closed*:

antisymmetric 1

<proof>

lemma *antisymmetric-inf-closed*:

antisymmetric $x \implies$ *antisymmetric* $(x \sqcap y)$

<proof>

lemma *antisymmetric-conv-closed*:

antisymmetric $x \implies$ *antisymmetric* (x^T)

<proof>

lemma *asymmetric-bot-closed*:

asymmetric bot

<proof>

lemma *asymmetric-inf-closed*:

asymmetric $x \implies$ *asymmetric* $(x \sqcap y)$

<proof>

lemma *asymmetric-conv-closed*:

asymmetric $x \implies$ *asymmetric* (x^T)

<proof>

lemma *linear-top-closed*:

linear top

<proof>

lemma *linear-sup-closed*:

linear $x \implies$ *linear* $(x \sqcup y)$
<proof>

lemma *linear-reflexive*:
linear $x \implies$ *reflexive* x
<proof>

lemma *linear-conv-closed*:
linear $x \implies$ *linear* (x^T)
<proof>

lemma *linear-comp-closed*:
assumes *linear* x
and *linear* y
shows *linear* $(x * y)$
<proof>

lemma *equivalence-one-closed*:
equivalence 1
<proof>

lemma *equivalence-top-closed*:
equivalence top
<proof>

lemma *equivalence-inf-closed*:
equivalence $x \implies$ *equivalence* $y \implies$ *equivalence* $(x \sqcap y)$
<proof>

lemma *equivalence-conv-closed*:
equivalence $x \implies$ *equivalence* (x^T)
<proof>

lemma *order-one-closed*:
order 1
<proof>

lemma *order-inf-closed*:
order $x \implies$ *order* $y \implies$ *order* $(x \sqcap y)$
<proof>

lemma *order-conv-closed*:
order $x \implies$ *order* (x^T)
<proof>

lemma *linear-order-conv-closed*:
linear-order $x \implies$ *linear-order* (x^T)
<proof>

We show a fact about equivalences.

lemma *equivalence-comp-dist-inf*:

$$\text{equivalence } x \implies x * y \sqcap x * z = x * (y \sqcap x * z)$$

<proof>

The following result generalises the fact that composition with a test amounts to intersection with the corresponding vector. Both tests and vectors can be used to represent sets as relations.

lemma *coreflexive-comp-top-inf*:

$$\text{coreflexive } x \implies x * \text{top} \sqcap y = x * y$$

<proof>

lemma *coreflexive-comp-top-inf-one*:

$$\text{coreflexive } x \implies x * \text{top} \sqcap 1 = x$$

<proof>

lemma *coreflexive-comp-inf*:

$$\text{coreflexive } x \implies \text{coreflexive } y \implies x * y = x \sqcap y$$

<proof>

lemma *coreflexive-comp-inf-comp*:

assumes *coreflexive* x

and *coreflexive* y

shows $(x * z) \sqcap (y * z) = (x \sqcap y) * z$

<proof>

lemma *coreflexive-idempotent*:

$$\text{coreflexive } x \implies \text{idempotent } x$$

<proof>

lemma *coreflexive-univalent*:

$$\text{coreflexive } x \implies \text{univalent } x$$

<proof>

lemma *coreflexive-injective*:

$$\text{coreflexive } x \implies \text{injective } x$$

<proof>

lemma *coreflexive-commutative*:

$$\text{coreflexive } x \implies \text{coreflexive } y \implies x * y = y * x$$

<proof>

lemma *coreflexive-dedekind*:

$$\text{coreflexive } x \implies \text{coreflexive } y \implies \text{coreflexive } z \implies x * y \sqcap z \leq x * (y \sqcap x * z)$$

<proof>

Also the equational version of the Dedekind rule continues to hold.

lemma *dedekind-eq*:

$$x * y \sqcap z = (x \sqcap (z * y^T)) * (y \sqcap (x^T * z)) \sqcap z$$

<proof>

lemma *dedekind*:

$$x * y \sqcap z \leq (x \sqcap (z * y^T)) * (y \sqcap (x^T * z))$$

<proof>

lemma *vector-export-comp*:

$$(x * top \sqcap y) * z = x * top \sqcap y * z$$

<proof>

lemma *vector-export-comp-unit*:

$$(x * top \sqcap 1) * y = x * top \sqcap y$$

<proof>

We solve a few exercises from [27].

lemma *ex231a* [*simp*]:

$$(1 \sqcap x * x^T) * x = x$$

<proof>

lemma *ex231b* [*simp*]:

$$x * (1 \sqcap x^T * x) = x$$

<proof>

lemma *ex231c*:

$$x \leq x * x^T * x$$

<proof>

lemma *ex231d*:

$$x \leq x * top * x$$

<proof>

lemma *ex231e* [*simp*]:

$$x * top * x * top = x * top$$

<proof>

lemma *arc-injective*:

$$arc\ x \implies injective\ x$$

<proof>

lemma *arc-conv-closed*:

$$arc\ x \implies arc\ (x^T)$$

<proof>

lemma *arc-univalent*:

$$arc\ x \implies univalent\ x$$

<proof>

The following result generalises [22, Exercise 2]. It is used to show that the while-loop preserves injectivity of the constructed tree.

lemma *injective-sup*:

assumes *injective t*
and $e * t^T \leq 1$
and *injective e*
shows *injective* $(t \sqcup e)$
 \langle *proof* \rangle

lemma *injective-inv*:
injective t $\implies e * t^T = \text{bot} \implies \text{arc } e \implies \text{injective } (t \sqcup e)$
 \langle *proof* \rangle

lemma *univalent-sup*:
univalent t $\implies e^T * t \leq 1 \implies \text{univalent } e \implies \text{univalent } (t \sqcup e)$
 \langle *proof* \rangle

lemma *point-injective*:
 $\text{arc } x \implies x^T * \text{top} * x \leq 1$
 \langle *proof* \rangle

lemma *vv-transitive*:
vector v $\implies (v * v^T) * (v * v^T) \leq v * v^T$
 \langle *proof* \rangle

lemma *epm-3*:
assumes $e \leq w$
and *injective w*
shows $e = w \sqcap \text{top} * e$
 \langle *proof* \rangle

lemma *comp-inf-vector*:
 $x * (y \sqcap z * \text{top}) = (x \sqcap \text{top} * z^T) * y$
 \langle *proof* \rangle

lemma *inf-vector-comp*:
 $(x \sqcap y * \text{top}) * z = y * \text{top} \sqcap x * z$
 \langle *proof* \rangle

lemma *comp-inf-covector*:
 $x * (y \sqcap \text{top} * z) = x * y \sqcap \text{top} * z$
 \langle *proof* \rangle

Well-known distributivity properties of univalent and injective relations over meet continue to hold.

lemma *univalent-comp-left-dist-inf*:
assumes *univalent x*
shows $x * (y \sqcap z) = x * y \sqcap x * z$
 \langle *proof* \rangle

lemma *injective-comp-right-dist-inf*:
injective z $\implies (x \sqcap y) * z = x * z \sqcap y * z$

<proof>

lemma *vector-covector*:

vector $v \implies$ *vector* $w \implies v \sqcap w^T = v * w^T$

<proof>

lemma *comp-inf-vector-1*:

$(x \sqcap \text{top} * y) * z = x * (z \sqcap (\text{top} * y)^T)$

<proof>

The shunting properties for bijective relations and mappings continue to hold.

lemma *shunt-bijective*:

assumes *bijective* z

shows $x \leq y * z \iff x * z^T \leq y$

<proof>

lemma *shunt-mapping*:

mapping $z \implies x \leq z * y \iff z^T * x \leq y$

<proof>

lemma *bijective-reverse*:

assumes *bijective* p

and *bijective* q

shows $p \leq r * q \iff q \leq r^T * p$

<proof>

lemma *arc-expanded*:

arc $x \iff x * \text{top} * x^T \leq 1 \wedge x^T * \text{top} * x \leq 1 \wedge \text{top} * x * \text{top} = \text{top}$

<proof>

lemma *arc-top-arc*:

assumes *arc* x

shows $x * \text{top} * x = x$

<proof>

lemma *arc-top-edge*:

assumes *arc* x

shows $x^T * \text{top} * x = x^T * x$

<proof>

lemma *arc-eq-1*:

assumes *arc* x

shows $x = x * x^T * x$

<proof>

lemma *arc-eq-2*:

assumes *arc* x

shows $x^T = x^T * x * x^T$

<proof>

end

4.2 Single-Object Pseudocomplemented Distributive Allegories

We extend single-object bounded distributive allegories by a pseudocomplement operation. The following definitions concern properties of relations that require a pseudocomplement.

class *relation-algebra-signature* = *bounded-distrib-allegory-signature* + *uminus*
begin

abbreviation *irreflexive* :: 'a ⇒ bool **where** *irreflexive* x ≡ x ≤ -1
abbreviation *strict-linear* :: 'a ⇒ bool **where** *strict-linear* x ≡ x ⊔ x^T = -1

abbreviation *strict-order* :: 'a ⇒ bool **where** *strict-order* x ≡
irreflexive x ∧ *transitive* x

abbreviation *linear-strict-order* :: 'a ⇒ bool **where** *linear-strict-order* x ≡
strict-order x ∧ *strict-linear* x

The following variants are useful for the graph model.

abbreviation *pp-mapping* :: 'a ⇒ bool **where** *pp-mapping* x ≡
univalent x ∧ *total* (¬¬x)

abbreviation *pp-bijective* :: 'a ⇒ bool **where** *pp-bijective* x ≡
injective x ∧ *surjective* (¬¬x)

abbreviation *pp-point* :: 'a ⇒ bool **where** *pp-point* x ≡ *vector*
x ∧ *pp-bijective* x

abbreviation *pp-arc* :: 'a ⇒ bool **where** *pp-arc* x ≡
pp-bijective (x * top) ∧ *pp-bijective* (x^T * top)

end

class *pd-allegory* = *bounded-distrib-allegory* + *p-algebra*
begin

subclass *relation-algebra-signature* *<proof>*

subclass *pd-algebra* *<proof>*

lemma *conv-complement-1*:
-(x^T) ⊔ (-x)^T = (-x)^T
<proof>

lemma *conv-complement*:
(-x)^T = -(x^T)
<proof>

lemma *conv-complement-sub-inf* [simp]:

$$x^T * -(x * y) \sqcap y = \text{bot}$$

<proof>

lemma *conv-complement-sub-leq*:

$$x^T * -(x * y) \leq -y$$

<proof>

lemma *conv-complement-sub* [simp]:

$$x^T * -(x * y) \sqcup -y = -y$$

<proof>

lemma *complement-conv-sub*:

$$-(y * x) * x^T \leq -y$$

<proof>

The following so-called Schröder equivalences, or De Morgan's Theorem K, hold only with a pseudocomplemented element on both right-hand sides.

lemma *schröder-3-p*:

$$x * y \leq -z \iff x^T * z \leq -y$$

<proof>

lemma *schröder-4-p*:

$$x * y \leq -z \iff z * y^T \leq -x$$

<proof>

lemma *comp-pp-semi-commute*:

$$x * --y \leq --(x * y)$$

<proof>

The following result looks similar to a property of (anti)domain.

lemma *p-comp-pp* [simp]:

$$-(x * --y) = -(x * y)$$

<proof>

lemma *pp-comp-semi-commute*:

$$--x * y \leq --(x * y)$$

<proof>

lemma *p-pp-comp* [simp]:

$$-(--x * y) = -(x * y)$$

<proof>

lemma *pp-comp-subdist*:

$$--x * --y \leq --(x * y)$$

<proof>

lemma *theorem24xxiii*:

$$x * y \sqcap -(x * z) = x * (y \sqcap -z) \sqcap -(x * z)$$

<proof>

Even in Stone relation algebras, we do not obtain the backward implication in the following result.

lemma *vector-complement-closed:*
vector $x \implies \text{vector } (-x)$
<proof>

lemma *covector-complement-closed:*
covector $x \implies \text{covector } (-x)$
<proof>

lemma *covector-vector-comp:*
vector $v \implies -v^T * v = \text{bot}$
<proof>

lemma *irreflexive-bot-closed:*
irreflexive bot
<proof>

lemma *irreflexive-inf-closed:*
irreflexive $x \implies \text{irreflexive } (x \sqcap y)$
<proof>

lemma *irreflexive-sup-closed:*
irreflexive $x \implies \text{irreflexive } y \implies \text{irreflexive } (x \sqcup y)$
<proof>

lemma *irreflexive-conv-closed:*
irreflexive $x \implies \text{irreflexive } (x^T)$
<proof>

lemma *reflexive-complement-irreflexive:*
reflexive $x \implies \text{irreflexive } (-x)$
<proof>

lemma *irreflexive-complement-reflexive:*
irreflexive $x \iff \text{reflexive } (-x)$
<proof>

lemma *symmetric-complement-closed:*
symmetric $x \implies \text{symmetric } (-x)$
<proof>

lemma *asymmetric-irreflexive:*
asymmetric $x \implies \text{irreflexive } x$
<proof>

lemma *linear-asymmetric*:

linear $x \implies$ *asymmetric* $(-x)$

\langle *proof* \rangle

lemma *strict-linear-sup-closed*:

strict-linear $x \implies$ *strict-linear* $y \implies$ *strict-linear* $(x \sqcup y)$

\langle *proof* \rangle

lemma *strict-linear-irreflexive*:

strict-linear $x \implies$ *irreflexive* x

\langle *proof* \rangle

lemma *strict-linear-conv-closed*:

strict-linear $x \implies$ *strict-linear* (x^T)

\langle *proof* \rangle

lemma *strict-order-var*:

strict-order $x \iff$ *asymmetric* $x \wedge$ *transitive* x

\langle *proof* \rangle

lemma *strict-order-bot-closed*:

strict-order *bot*

\langle *proof* \rangle

lemma *strict-order-inf-closed*:

strict-order $x \implies$ *strict-order* $y \implies$ *strict-order* $(x \sqcap y)$

\langle *proof* \rangle

lemma *strict-order-conv-closed*:

strict-order $x \implies$ *strict-order* (x^T)

\langle *proof* \rangle

lemma *order-strict-order*:

assumes *order* x

shows *strict-order* $(x \sqcap -1)$

\langle *proof* \rangle

lemma *strict-order-order*:

strict-order $x \implies$ *order* $(x \sqcup 1)$

\langle *proof* \rangle

lemma *linear-strict-order-conv-closed*:

linear-strict-order $x \implies$ *linear-strict-order* (x^T)

\langle *proof* \rangle

lemma *linear-order-strict-order*:

linear-order $x \implies$ *linear-strict-order* $(x \sqcap -1)$

\langle *proof* \rangle

lemma *regular-conv-closed*:
regular $x \implies \text{regular } (x^T)$
 ⟨proof⟩

We show a number of facts about equivalences.

lemma *equivalence-comp-left-complement*:
equivalence $x \implies x * -x = -x$
 ⟨proof⟩

lemma *equivalence-comp-right-complement*:
equivalence $x \implies -x * x = -x$
 ⟨proof⟩

The pseudocomplement of tests is given by the following operation.

abbreviation *coreflexive-complement* :: 'a \Rightarrow 'a (- '' [80] 80)
 where $x' \equiv -x \sqcap 1$

lemma *coreflexive-comp-top-coreflexive-complement*:
coreflexive $x \implies (x * \text{top})' = x'$
 ⟨proof⟩

lemma *coreflexive-comp-inf-complement*:
coreflexive $x \implies (x * y) \sqcap -z = (x * y) \sqcap -(x * z)$
 ⟨proof⟩

lemma *double-coreflexive-complement*:
 $x'' = (-x)'$
 ⟨proof⟩

lemma *coreflexive-pp-dist-comp*:
 assumes *coreflexive* x
 and *coreflexive* y
 shows $(x * y)'' = x'' * y''$
 ⟨proof⟩

lemma *coreflexive-pseudo-complement*:
coreflexive $x \implies x \sqcap y = \text{bot} \iff x \leq y'$
 ⟨proof⟩

lemma *pp-bijective-conv-mapping*:
pp-bijective $x \iff \text{pp-mapping } (x^T)$
 ⟨proof⟩

lemma *pp-arc-expanded*:
 $\text{pp-arc } x \iff x * \text{top} * x^T \leq 1 \wedge x^T * \text{top} * x \leq 1 \wedge \text{top} * --x * \text{top} = \text{top}$
 ⟨proof⟩

The following operation represents states with infinite executions of non-strict computations.

abbreviation $N :: 'a \Rightarrow 'a$
where $N x \equiv -(-x * top) \sqcap 1$

lemma *N-comp*:
 $N x * y = -(-x * top) \sqcap y$
 $\langle proof \rangle$

lemma *N-comp-top [simp]*:
 $N x * top = -(-x * top)$
 $\langle proof \rangle$

lemma *vector-N-pp*:
 $vector\ x \Longrightarrow N x = --x \sqcap 1$
 $\langle proof \rangle$

lemma *N-vector-pp [simp]*:
 $N (x * top) = --(x * top) \sqcap 1$
 $\langle proof \rangle$

lemma *N-vector-top-pp [simp]*:
 $N (x * top) * top = --(x * top)$
 $\langle proof \rangle$

lemma *N-below-inf-one-pp*:
 $N x \leq --x \sqcap 1$
 $\langle proof \rangle$

lemma *N-below-pp*:
 $N x \leq --x$
 $\langle proof \rangle$

lemma *N-comp-N*:
 $N x * N y = -(-x * top) \sqcap -(-y * top) \sqcap 1$
 $\langle proof \rangle$

lemma *N-bot [simp]*:
 $N bot = bot$
 $\langle proof \rangle$

lemma *N-top [simp]*:
 $N top = 1$
 $\langle proof \rangle$

lemma *n-split-omega-mult-pp*:
 $xs * --xo = xo \Longrightarrow vector\ xo \Longrightarrow N top * xo = xs * N xo * top$
 $\langle proof \rangle$

Many of the following results have been derived for verifying Prim's minimum spanning tree algorithm.

lemma *ee*:

assumes *vector v*
and $e \leq v * -v^T$
shows $e * e = \text{bot}$

<proof>

lemma *et*:

assumes *vector v*
and $e \leq v * -v^T$
and $t \leq v * v^T$
shows $e * t = \text{bot}$
and $e * t^T = \text{bot}$

<proof>

lemma *ve-dist*:

assumes $e \leq v * -v^T$
and *vector v*
and *arc e*
shows $(v \sqcup e^T * \text{top}) * (v \sqcup e^T * \text{top})^T = v * v^T \sqcup v * v^T * e \sqcup e^T * v * v^T \sqcup e^T * e$

<proof>

lemma *ev*:

vector v $\implies e \leq v * -v^T \implies e * v = \text{bot}$

<proof>

lemma *vTeT*:

vector v $\implies e \leq v * -v^T \implies v^T * e^T = \text{bot}$

<proof>

The following result is used to show that the while-loop of Prim's algorithm preserves that the constructed tree is a subgraph of g .

lemma *prim-subgraph-inv*:

assumes $e \leq v * -v^T \sqcap g$
and $t \leq v * v^T \sqcap g$
shows $t \sqcup e \leq ((v \sqcup e^T * \text{top}) * (v \sqcup e^T * \text{top})^T) \sqcap g$

<proof>

The following result shows how to apply the Schröder equivalence to the middle factor in a composition of three relations. Again the elements on the right-hand side need to be pseudocomplemented.

lemma *triple-schroeder-p*:

$x * y * z \leq -w \iff x^T * w * z^T \leq -y$

<proof>

The rotation versions of the Schröder equivalences continue to hold, again with pseudocomplemented elements on the right-hand side.

lemma *schroeder-5-p*:

$x * y \leq -z \iff y * z^T \leq -x^T$

<proof>

lemma *schroeder-6-p:*

$$x * y \leq -z \iff z^T * x \leq -y^T$$

<proof>

lemma *vector-conv-compl:*

$$\text{vector } v \implies \text{top} * -v^T = -v^T$$

<proof>

Composition commutes, relative to the diversity relation.

lemma *comp-commute-below-diversity:*

$$x * y \leq -1 \iff y * x \leq -1$$

<proof>

lemma *comp-injective-below-complement:*

$$\text{injective } y \implies -x * y \leq -(x * y)$$

<proof>

lemma *comp-univalent-below-complement:*

$$\text{univalent } x \implies x * -y \leq -(x * y)$$

<proof>

Bijjective relations and mappings can be exported from a pseudocomplement.

lemma *comp-bijjective-complement:*

$$\text{bijjective } y \implies -x * y = -(x * y)$$

<proof>

lemma *comp-mapping-complement:*

$$\text{mapping } x \implies x * -y = -(x * y)$$

<proof>

The following facts are used in the correctness proof of Kruskal's minimum spanning tree algorithm.

lemma *kruskal-injective-inv:*

assumes *injective* f

and *covector* q

$$\text{and } q * f^T \leq q$$

$$\text{and } e \leq q$$

$$\text{and } q * f^T \leq -e$$

and *injective* e

$$\text{and } q^T * q \sqcap f^T * f \leq 1$$

shows *injective* $((f \sqcap -q) \sqcup (f \sqcap q)^T \sqcup e)$

<proof>

lemma *kruskal-exchange-injective-inv-1:*

assumes *injective* f

and *covector* q

and $q * f^T \leq q$
and $q^T * q \sqcap f^T * f \leq 1$
shows *injective* $((f \sqcap -q) \sqcup (f \sqcap q)^T)$
 $\langle proof \rangle$

lemma *kruskal-exchange-acyclic-inv-3*:
assumes *injective* w
and $d \leq w$
shows $(w \sqcap -d) * d^T * top = bot$
 $\langle proof \rangle$

lemma *kruskal-subgraph-inv*:
assumes $f \leq --(-h \sqcap g)$
and $e \leq --g$
and *symmetric* h
and *symmetric* g
shows $(f \sqcap -q) \sqcup (f \sqcap q)^T \sqcup e \leq --(- (h \sqcap -e \sqcap -e^T) \sqcap g)$
 $\langle proof \rangle$

end

4.3 Stone Relation Algebras

We add *pp-dist-comp* and *pp-one*, which follow in relation algebras but not in the present setting. The main change is that only a Stone algebra is required, not a Boolean algebra.

class *stone-relation-algebra* = *pd-allegory* + *stone-algebra* +
assumes *pp-dist-comp* : $--(x * y) = --x * --y$
assumes *pp-one* [*simp*]: $--1 = 1$
begin

The following property is a simple consequence of the Stone axiom. We cannot hope to remove the double complement in it.

lemma *conv-complement-0-p* [*simp*]:
 $(-x)^T \sqcup (-x)^T = top$
 $\langle proof \rangle$

lemma *theorem24xxiv-pp*:
 $-(x * y) \sqcup --(x * z) = -(x * (y \sqcap -z)) \sqcup --(x * z)$
 $\langle proof \rangle$

lemma *asymmetric-linear*:
asymmetric $x \longleftrightarrow$ *linear* $(-x)$
 $\langle proof \rangle$

lemma *strict-linear-asymmetric*:
strict-linear $x \implies$ *antisymmetric* $(-x)$
 $\langle proof \rangle$

lemma *regular-complement-top*:
regular $x \implies x \sqcup -x = \text{top}$
 ⟨*proof*⟩

lemma *regular-mult-closed*:
regular $x \implies \text{regular } y \implies \text{regular } (x * y)$
 ⟨*proof*⟩

lemma *regular-one-closed*:
regular 1
 ⟨*proof*⟩

The following variants of total and surjective are useful for graphs.

lemma *pp-total*:
total $(--x) \iff -(x*\text{top}) = \text{bot}$
 ⟨*proof*⟩

lemma *pp-surjective*:
surjective $(--x) \iff -(top*x) = \text{bot}$
 ⟨*proof*⟩

Bijjective elements and mappings are necessarily regular, that is, invariant under double-complement. This implies that points are regular. Moreover, also arcs are regular.

lemma *bijjective-regular*:
bijjective $x \implies \text{regular } x$
 ⟨*proof*⟩

lemma *mapping-regular*:
mapping $x \implies \text{regular } x$
 ⟨*proof*⟩

lemma *arc-regular*:
assumes *arc* x
shows *regular* x
 ⟨*proof*⟩

end

Every Stone algebra can be expanded to a Stone relation algebra by identifying the semiring and lattice structures and taking identity as converse.

sublocale *stone-algebra* $<$ *comp-inf*: *stone-relation-algebra* **where** *one* = *top*
and *times* = *inf* **and** *conv* = *id*
 ⟨*proof*⟩

Every bounded linear order can be expanded to a Stone algebra, which can be expanded to a Stone relation algebra by reusing some of the opera-

tions. In particular, composition is meet, its identity is *top* and converse is the identity function.

```

class linorder-stone-relation-algebra-expansion = linorder-stone-algebra-expansion
+ times + conv + one +
  assumes times-def [simp]:  $x * y = \min x y$ 
  assumes conv-def [simp]:  $x^T = x$ 
  assumes one-def [simp]:  $1 = \text{top}$ 
begin

```

```

lemma times-inf [simp]:
 $x * y = x \sqcap y$ 
<proof>

```

```

subclass stone-relation-algebra
<proof>

```

```

lemma times-dense:
 $x \neq \text{bot} \implies y \neq \text{bot} \implies x * y \neq \text{bot}$ 
<proof>

```

```

end

```

4.4 Relation Algebras

For a relation algebra, we only require that the underlying lattice is a Boolean algebra. In fact, the only missing axiom is that double-complement is the identity.

```

class relation-algebra = boolean-algebra + stone-relation-algebra
begin

```

```

lemma conv-complement-0 [simp]:
 $x^T \sqcup (-x)^T = \text{top}$ 
<proof>

```

We now obtain the original formulations of the Schröder equivalences.

```

lemma schroeder-3:
 $x * y \leq z \iff x^T * -z \leq -y$ 
<proof>

```

```

lemma schroeder-4:
 $x * y \leq z \iff -z * y^T \leq -x$ 
<proof>

```

```

lemma theorem24xxiv:
 $-(x * y) \sqcup (x * z) = -(x * (y \sqcap -z)) \sqcup (x * z)$ 
<proof>

```

```

lemma vector-N:

```

vector $x \implies N(x) = x \sqcap 1$
<proof>

lemma *N-vector* [*simp*]:
 $N(x * top) = x * top \sqcap 1$
<proof>

lemma *N-vector-top* [*simp*]:
 $N(x * top) * top = x * top$
<proof>

lemma *N-below-inf-one*:
 $N(x) \leq x \sqcap 1$
<proof>

lemma *N-below*:
 $N(x) \leq x$
<proof>

lemma *n-split-omega-mult*:
 $xs * xo = xo \implies xo * top = xo \implies N(top) * xo = xs * N(xo) * top$
<proof>

lemma *complement-vector*:
vector $v \longleftrightarrow$ *vector* $(-v)$
<proof>

lemma *complement-covector*:
covector $v \longleftrightarrow$ *covector* $(-v)$
<proof>

lemma *triple-schroeder*:
 $x * y * z \leq w \longleftrightarrow x^T * -w * z^T \leq -y$
<proof>

lemma *schroeder-5*:
 $x * y \leq z \longleftrightarrow y * -z^T \leq -x^T$
<proof>

lemma *schroeder-6*:
 $x * y \leq z \longleftrightarrow -z^T * x \leq -y^T$
<proof>

end

We briefly look at the so-called Tarski rule. In some models of Stone relation algebras it only holds for regular elements, so we add this as an assumption.

class *stone-relation-algebra-tarski* = *stone-relation-algebra* +

assumes *tarski*: $\text{regular } x \implies x \neq \text{bot} \implies \text{top} * x * \text{top} = \text{top}$
begin

We can then show, for example, that every arc is contained in a pseudo-complemented relation or its pseudocomplement.

lemma *arc-in-partition*:
assumes *arc* *x*
shows $x \leq -y \vee x \leq --y$
 $\langle \text{proof} \rangle$

lemma *non-bot-arc-in-partition-xor*:
assumes *arc* *x*
and $x \neq \text{bot}$
shows $(x \leq -y \wedge \neg x \leq --y) \vee (\neg x \leq -y \wedge x \leq --y)$
 $\langle \text{proof} \rangle$

lemma *point-in-vector-or-pseudo-complement*:
assumes *point* *p*
and *vector* *v*
shows $p \leq --v \vee p \leq -v$
 $\langle \text{proof} \rangle$

lemma *point-in-vector-or-complement*:
assumes *point* *p*
and *vector* *v*
and *regular* *v*
shows $p \leq v \vee p \leq -v$
 $\langle \text{proof} \rangle$

lemma *point-in-vector-sup*:
assumes *point* *p*
and *vector* *v*
and *regular* *v*
and $p \leq v \sqcup w$
shows $p \leq v \vee p \leq w$
 $\langle \text{proof} \rangle$

lemma *arc-in-arc-or-complement*:
assumes *arc* *x*
and *arc* *y*
and $\neg x \leq y$
shows $x \leq -y$
 $\langle \text{proof} \rangle$

lemma *arc-in-sup-arc*:
assumes *arc* *x*
and *arc* *y*
and $x \leq z \sqcup y$
shows $x \leq z \vee x \leq y$

<proof>

lemma *different-arc-in-sup-arc:*

assumes *arc x*
and *arc y*
and $x \leq z \sqcup y$
and $x \neq y$
shows $x \leq z$

<proof>

end

class *relation-algebra-tarski* = *relation-algebra* + *stone-relation-algebra-tarski*

Finally, the above axioms of relation algebras do not imply that they contain at least two elements. This is necessary, for example, to show that arcs are not empty.

class *stone-relation-algebra-consistent* = *stone-relation-algebra* +

assumes *consistent: bot \neq top*

begin

lemma *arc-not-bot:*

arc x \implies x \neq bot

<proof>

end

class *relation-algebra-consistent* = *relation-algebra* +

stone-relation-algebra-consistent

class *stone-relation-algebra-tarski-consistent* = *stone-relation-algebra-tarski* +

stone-relation-algebra-consistent

begin

lemma *arc-in-partition-xor:*

arc x \implies (x \leq -y \wedge \neg x \leq --y) \vee (\neg x \leq -y \wedge x \leq --y)

<proof>

end

end

5 Subalgebras of Relation Algebras

In this theory we consider the algebraic structure of regular elements, coreflexives, vectors and covectors in Stone relation algebras. These elements form important subalgebras and substructures of relation algebras.

theory *Relation-Subalgebras*

imports *Stone-Algebras.Stone-Construction Relation-Algebras*

begin

The regular elements of a Stone relation algebra form a relation subalgebra.

instantiation *regular* :: (*stone-relation-algebra*) *relation-algebra*
begin

lift-definition *times-regular* :: 'a *regular* \Rightarrow 'a *regular* \Rightarrow 'a *regular* **is** *times*
<proof>

lift-definition *conv-regular* :: 'a *regular* \Rightarrow 'a *regular* **is** *conv*
<proof>

lift-definition *one-regular* :: 'a *regular* **is** *1*
<proof>

instance
<proof>

end

The coreflexives (tests) in an idempotent semiring form a bounded idempotent subsemiring.

typedef (**overloaded**) 'a *coreflexive* =
coreflexives::'a::non-associative-left-semiring set
<proof>

lemma *simp-coreflexive* [*simp*]:
 $\exists y . \text{Rep-coreflexive } x \leq 1$
<proof>

setup-lifting *type-definition-coreflexive*

instantiation *coreflexive* :: (*idempotent-semiring*) *bounded-idempotent-semiring*
begin

lift-definition *sup-coreflexive* :: 'a *coreflexive* \Rightarrow 'a *coreflexive* \Rightarrow 'a *coreflexive*
is *sup*
<proof>

lift-definition *times-coreflexive* :: 'a *coreflexive* \Rightarrow 'a *coreflexive* \Rightarrow 'a *coreflexive*
is *times*
<proof>

lift-definition *bot-coreflexive* :: 'a *coreflexive* **is** *bot*
<proof>

lift-definition *one-coreflexive* :: 'a coreflexive is 1
⟨proof⟩

lift-definition *top-coreflexive* :: 'a coreflexive is 1
⟨proof⟩

lift-definition *less-eq-coreflexive* :: 'a coreflexive ⇒ 'a coreflexive ⇒ bool is
less-eq ⟨proof⟩

lift-definition *less-coreflexive* :: 'a coreflexive ⇒ 'a coreflexive ⇒ bool is *less*
⟨proof⟩

instance
⟨proof⟩

end

The coreflexives (tests) in a Stone relation algebra form a Stone relation algebra where the pseudocomplement is taken relative to the identity relation and converse is the identity function.

instantiation *coreflexive* :: (stone-relation-algebra) stone-relation-algebra
begin

lift-definition *inf-coreflexive* :: 'a coreflexive ⇒ 'a coreflexive ⇒ 'a coreflexive is
inf
⟨proof⟩

lift-definition *minus-coreflexive* :: 'a coreflexive ⇒ 'a coreflexive ⇒ 'a
coreflexive is $\lambda x y . x \sqcap -y$
⟨proof⟩

lift-definition *uminus-coreflexive* :: 'a coreflexive ⇒ 'a coreflexive is $\lambda x . -x \sqcap 1$
⟨proof⟩

lift-definition *conv-coreflexive* :: 'a coreflexive ⇒ 'a coreflexive is *id*
⟨proof⟩

instance
⟨proof⟩

end

Vectors in a Stone relation algebra form a Stone subalgebra.

typedef (overloaded) 'a vector = vectors::'a::bounded-pre-left-semiring set
⟨proof⟩

lemma *simp-vector* [*simp*]:
 $\exists y . \text{Rep-vector } x * \text{top} = \text{Rep-vector } x$

```

    <proof>

setup-lifting type-definition-vector

instantiation vector :: (stone-relation-algebra) stone-algebra
begin

lift-definition sup-vector :: 'a vector  $\Rightarrow$  'a vector  $\Rightarrow$  'a vector is sup
    <proof>

lift-definition inf-vector :: 'a vector  $\Rightarrow$  'a vector  $\Rightarrow$  'a vector is inf
    <proof>

lift-definition uminus-vector :: 'a vector  $\Rightarrow$  'a vector is uminus
    <proof>

lift-definition bot-vector :: 'a vector is bot
    <proof>

lift-definition top-vector :: 'a vector is top
    <proof>

lift-definition less-eq-vector :: 'a vector  $\Rightarrow$  'a vector  $\Rightarrow$  bool is less-eq <proof>

lift-definition less-vector :: 'a vector  $\Rightarrow$  'a vector  $\Rightarrow$  bool is less <proof>

instance
    <proof>

end

    Covectors in a Stone relation algebra form a Stone subalgebra.

typedef (overloaded) 'a covector = covectors::'a::bounded-pre-left-semiring set
    <proof>

lemma simp-covector [simp]:
     $\exists y . \text{top} * \text{Rep-covector } x = \text{Rep-covector } x$ 
    <proof>

setup-lifting type-definition-covector

instantiation covector :: (stone-relation-algebra) stone-algebra
begin

lift-definition sup-covector :: 'a covector  $\Rightarrow$  'a covector  $\Rightarrow$  'a covector is sup
    <proof>

lift-definition inf-covector :: 'a covector  $\Rightarrow$  'a covector  $\Rightarrow$  'a covector is inf
    <proof>

```

lift-definition *uminus-covector* :: 'a covector \Rightarrow 'a covector **is** *uminus*
<proof>

lift-definition *bot-covector* :: 'a covector **is** *bot*
<proof>

lift-definition *top-covector* :: 'a covector **is** *top*
<proof>

lift-definition *less-eq-covector* :: 'a covector \Rightarrow 'a covector \Rightarrow bool **is** *less-eq*
<proof>

lift-definition *less-covector* :: 'a covector \Rightarrow 'a covector \Rightarrow bool **is** *less* *<proof>*

instance
<proof>

end

end

6 Matrix Relation Algebras

This theory gives matrix models of Stone relation algebras and more general structures. We consider only square matrices. The main result is that matrices over Stone relation algebras form a Stone relation algebra.

We use the monoid structure underlying semilattices to provide finite sums, which are necessary for defining the composition of two matrices. See [3, 4] for similar liftings to matrices for semirings and relation algebras. A technical difference is that those theories are mostly based on semirings whereas our hierarchy is mostly based on lattices (and our semirings directly inherit from semilattices).

Relation algebras have both a semiring and a lattice structure such that semiring addition and lattice join coincide. In particular, finite sums and finite suprema coincide. Isabelle/HOL has separate theories for semirings and lattices, based on separate addition and join operations and different operations for finite sums and finite suprema. Reusing results from both theories is beneficial for relation algebras, but not always easy to realise.

theory *Matrix-Relation-Algebras*

imports *Relation-Algebras*

begin

6.1 Finite Suprema

We consider finite suprema in idempotent semirings and Stone relation algebras. We mostly use the first of the following notations, which denotes the supremum of expressions $t(x)$ over all x from the type of x . For finite types, this is implemented in Isabelle/HOL as the repeated application of binary suprema.

syntax

$\text{-sum-sup-monoid} :: \text{idt} \Rightarrow 'a::\text{bounded-semilattice-sup-bot} \Rightarrow 'a \ ((\bigsqcup -) [0,10] 10)$

$\text{-sum-sup-monoid-bounded} :: \text{idt} \Rightarrow 'b \ \text{set} \Rightarrow 'a::\text{bounded-semilattice-sup-bot} \Rightarrow 'a \ ((\bigsqcup_{- \in -} -) [0,51,10] 10)$

translations

$\bigsqcup_x t \Rightarrow XCONST \ \text{sup-monoid.sum} \ (\lambda x . t) \ \{ x . CONST \ \text{True} \}$

$\bigsqcup_{x \in X} t \Rightarrow XCONST \ \text{sup-monoid.sum} \ (\lambda x . t) \ X$

context *idempotent-semiring*

begin

The following induction principles are useful for comparing two suprema. The first principle works because types are not empty.

lemma *one-sup-induct* [*case-names one sup*]:

fixes $f \ g :: 'b::\text{finite} \Rightarrow 'a$

assumes $\text{one}: \bigwedge i . P \ (f \ i) \ (g \ i)$

and $\text{sup}: \bigwedge j \ I . j \notin I \Longrightarrow P \ (\bigsqcup_{i \in I} f \ i) \ (\bigsqcup_{i \in I} g \ i) \Longrightarrow P \ (f \ j \sqcup (\bigsqcup_{i \in I} f \ i))$
 $(g \ j \sqcup (\bigsqcup_{i \in I} g \ i))$

shows $P \ (\bigsqcup_k f \ k) \ (\bigsqcup_k g \ k)$

<proof>

lemma *bot-sup-induct* [*case-names bot sup*]:

fixes $f \ g :: 'b::\text{finite} \Rightarrow 'a$

assumes $\text{bot}: P \ \text{bot} \ \text{bot}$

and $\text{sup}: \bigwedge j \ I . j \notin I \Longrightarrow P \ (\bigsqcup_{i \in I} f \ i) \ (\bigsqcup_{i \in I} g \ i) \Longrightarrow P \ (f \ j \sqcup (\bigsqcup_{i \in I} f \ i))$
 $(g \ j \sqcup (\bigsqcup_{i \in I} g \ i))$

shows $P \ (\bigsqcup_k f \ k) \ (\bigsqcup_k g \ k)$

<proof>

Now many properties of finite suprema follow by simple applications of the above induction rules. In particular, we show distributivity of composition, isotonicity and the upper-bound property.

lemma *comp-right-dist-sum*:

fixes $f :: 'b::\text{finite} \Rightarrow 'a$

shows $(\bigsqcup_k f \ k * x) = (\bigsqcup_k f \ k) * x$

<proof>

lemma *comp-left-dist-sum*:

fixes $f :: 'b::\text{finite} \Rightarrow 'a$

shows $(\bigsqcup_k x * f \ k) = x * (\bigsqcup_k f \ k)$

<proof>

lemma *leq-sum*:

fixes $f\ g :: 'b::\text{finite} \Rightarrow 'a$

shows $(\forall k . f\ k \leq g\ k) \implies (\bigsqcup_k f\ k) \leq (\bigsqcup_k g\ k)$

<proof>

lemma *ub-sum*:

fixes $f :: 'b::\text{finite} \Rightarrow 'a$

shows $f\ i \leq (\bigsqcup_k f\ k)$

<proof>

lemma *lub-sum*:

fixes $f :: 'b::\text{finite} \Rightarrow 'a$

assumes $\forall k . f\ k \leq x$

shows $(\bigsqcup_k f\ k) \leq x$

<proof>

lemma *lub-sum-iff*:

fixes $f :: 'b::\text{finite} \Rightarrow 'a$

shows $(\forall k . f\ k \leq x) \longleftrightarrow (\bigsqcup_k f\ k) \leq x$

<proof>

end

context *stone-relation-algebra*

begin

In Stone relation algebras, we can also show that converse, double complement and meet distribute over finite suprema.

lemma *conv-dist-sum*:

fixes $f :: 'b::\text{finite} \Rightarrow 'a$

shows $(\bigsqcup_k (f\ k)^T) = (\bigsqcup_k f\ k)^T$

<proof>

lemma *pp-dist-sum*:

fixes $f :: 'b::\text{finite} \Rightarrow 'a$

shows $(\bigsqcup_k \text{--}f\ k) = \text{--}(\bigsqcup_k f\ k)$

<proof>

lemma *inf-right-dist-sum*:

fixes $f :: 'b::\text{finite} \Rightarrow 'a$

shows $(\bigsqcup_k f\ k \sqcap x) = (\bigsqcup_k f\ k) \sqcap x$

<proof>

end

6.2 Square Matrices

Because our semiring and relation algebra type classes only work for homogeneous relations, we only look at square matrices.

type-synonym $(\text{'a}, \text{'b}) \text{ square} = \text{'a} \times \text{'a} \Rightarrow \text{'b}$

We use standard matrix operations. The Stone algebra structure is lifted componentwise. Composition is matrix multiplication using given composition and supremum operations. Its unit lifts given zero and one elements into an identity matrix. Converse is matrix transpose with an additional componentwise transpose.

definition *less-eq-matrix* $:: (\text{'a}, \text{'b}::\text{ord}) \text{ square} \Rightarrow (\text{'a}, \text{'b}) \text{ square} \Rightarrow \text{bool}$

(**infix** \preceq 50) **where** $f \preceq g = (\forall e . f e \leq g e)$

definition *less-matrix* $:: (\text{'a}, \text{'b}::\text{ord}) \text{ square} \Rightarrow (\text{'a}, \text{'b}) \text{ square} \Rightarrow \text{bool}$

(**infix** \prec 50) **where** $f \prec g = (f \preceq g \wedge \neg g \preceq f)$

definition *sup-matrix* $:: (\text{'a}, \text{'b}::\text{sup}) \text{ square} \Rightarrow (\text{'a}, \text{'b}) \text{ square} \Rightarrow (\text{'a}, \text{'b}) \text{ square}$

(**infixl** \oplus 65) **where** $f \oplus g = (\lambda e . f e \sqcup g e)$

definition *inf-matrix* $:: (\text{'a}, \text{'b}::\text{inf}) \text{ square} \Rightarrow (\text{'a}, \text{'b}) \text{ square} \Rightarrow (\text{'a}, \text{'b}) \text{ square}$

(**infixl** \otimes 67) **where** $f \otimes g = (\lambda e . f e \sqcap g e)$

definition *minus-matrix* $:: (\text{'a}, \text{'b}::\{\text{uminus}, \text{inf}\}) \text{ square} \Rightarrow (\text{'a}, \text{'b}) \text{ square} \Rightarrow$

$(\text{'a}, \text{'b}) \text{ square}$ (**infixl** \ominus 65) **where** $f \ominus g = (\lambda e . f e \sqcap -g e)$

definition *implies-matrix* $:: (\text{'a}, \text{'b}::\text{implies}) \text{ square} \Rightarrow (\text{'a}, \text{'b}) \text{ square} \Rightarrow (\text{'a}, \text{'b})$

square (**infixl** \odot 65) **where** $f \odot g = (\lambda e . f e \rightsquigarrow g e)$

definition *times-matrix* $:: (\text{'a}, \text{'b}::\{\text{times}, \text{bounded-semilattice-sup-bot}\}) \text{ square} \Rightarrow$

$(\text{'a}, \text{'b}) \text{ square} \Rightarrow (\text{'a}, \text{'b}) \text{ square}$ (**infixl** \odot 70) **where** $f \odot g = (\lambda(i,j) . \bigsqcup_k f(i,k) * g(k,j))$

definition *uminus-matrix* $:: (\text{'a}, \text{'b}::\text{uminus}) \text{ square} \Rightarrow (\text{'a}, \text{'b}) \text{ square}$

(**infix** \ominus 80) **where** $\ominus f = (\lambda e . -f e)$

definition *conv-matrix* $:: (\text{'a}, \text{'b}::\text{conv}) \text{ square} \Rightarrow (\text{'a}, \text{'b}) \text{ square}$

(**infix** t 100) **where** $f^t = (\lambda(i,j) . (f(j,i))^T)$

definition *bot-matrix* $:: (\text{'a}, \text{'b}::\text{bot}) \text{ square}$

(**infix** mbot) **where** $\text{mbot} = (\lambda e . \text{bot})$

definition *top-matrix* $:: (\text{'a}, \text{'b}::\text{top}) \text{ square}$

(**infix** mtop) **where** $\text{mtop} = (\lambda e . \text{top})$

definition *one-matrix* $:: (\text{'a}, \text{'b}::\{\text{one}, \text{bot}\}) \text{ square}$

(**infix** mone) **where** $\text{mone} = (\lambda(i,j) . \text{if } i = j \text{ then } 1 \text{ else } \text{bot})$

6.3 Stone Algebras

We first lift the Stone algebra structure. Because all operations are componentwise, this also works for infinite matrices.

interpretation *matrix-order*: *order* **where** $\text{less-eq} = \text{less-eq-matrix}$ **and** $\text{less} =$

$\text{less-matrix} :: (\text{'a}, \text{'b}::\text{order}) \text{ square} \Rightarrow (\text{'a}, \text{'b}) \text{ square} \Rightarrow \text{bool}$

<proof>

interpretation *matrix-semilattice-sup*: *semilattice-sup* **where** $\text{sup} = \text{sup-matrix}$

and $\text{less-eq} = \text{less-eq-matrix}$ **and** $\text{less} = \text{less-matrix} :: (\text{'a}, \text{'b}::\text{semilattice-sup})$

$\text{square} \Rightarrow (\text{'a}, \text{'b}) \text{ square} \Rightarrow \text{bool}$

<proof>

interpretation *matrix-semilattice-inf*: *semilattice-inf* **where** *inf* = *inf-matrix*
and *less-eq* = *less-eq-matrix* **and** *less* = *less-matrix* :: (*'a','b*::*semilattice-inf*)
square \Rightarrow (*'a','b*) *square* \Rightarrow *bool*
 ⟨*proof*⟩

interpretation *matrix-bounded-semilattice-sup-bot*: *bounded-semilattice-sup-bot*
where *sup* = *sup-matrix* **and** *less-eq* = *less-eq-matrix* **and** *less* = *less-matrix*
and *bot* = *bot-matrix* :: (*'a','b*::*bounded-semilattice-sup-bot*) *square*
 ⟨*proof*⟩

interpretation *matrix-bounded-semilattice-inf-top*: *bounded-semilattice-inf-top*
where *inf* = *inf-matrix* **and** *less-eq* = *less-eq-matrix* **and** *less* = *less-matrix*
and *top* = *top-matrix* :: (*'a','b*::*bounded-semilattice-inf-top*) *square*
 ⟨*proof*⟩

interpretation *matrix-lattice*: *lattice* **where** *sup* = *sup-matrix* **and** *inf* =
inf-matrix **and** *less-eq* = *less-eq-matrix* **and** *less* = *less-matrix* :: (*'a','b*::*lattice*)
square \Rightarrow (*'a','b*) *square* \Rightarrow *bool* ⟨*proof*⟩

interpretation *matrix-distrib-lattice*: *distrib-lattice* **where** *sup* = *sup-matrix*
and *inf* = *inf-matrix* **and** *less-eq* = *less-eq-matrix* **and** *less* = *less-matrix* ::
 (*'a','b*::*distrib-lattice*) *square* \Rightarrow (*'a','b*) *square* \Rightarrow *bool*
 ⟨*proof*⟩

interpretation *matrix-bounded-lattice*: *bounded-lattice* **where** *sup* = *sup-matrix*
and *inf* = *inf-matrix* **and** *less-eq* = *less-eq-matrix* **and** *less* = *less-matrix* **and**
bot = *bot-matrix* :: (*'a','b*::*bounded-lattice*) *square* **and** *top* = *top-matrix* ⟨*proof*⟩

interpretation *matrix-bounded-distrib-lattice*: *bounded-distrib-lattice* **where** *sup*
 = *sup-matrix* **and** *inf* = *inf-matrix* **and** *less-eq* = *less-eq-matrix* **and** *less* =
less-matrix **and** *bot* = *bot-matrix* :: (*'a','b*::*bounded-distrib-lattice*) *square* **and** *top*
 = *top-matrix* ⟨*proof*⟩

interpretation *matrix-p-algebra*: *p-algebra* **where** *sup* = *sup-matrix* **and** *inf* =
inf-matrix **and** *less-eq* = *less-eq-matrix* **and** *less* = *less-matrix* **and** *bot* =
bot-matrix :: (*'a','b*::*p-algebra*) *square* **and** *top* = *top-matrix* **and** *uminus* =
uminus-matrix
 ⟨*proof*⟩

interpretation *matrix-pd-algebra*: *pd-algebra* **where** *sup* = *sup-matrix* **and** *inf*
 = *inf-matrix* **and** *less-eq* = *less-eq-matrix* **and** *less* = *less-matrix* **and** *bot* =
bot-matrix :: (*'a','b*::*pd-algebra*) *square* **and** *top* = *top-matrix* **and** *uminus* =
uminus-matrix ⟨*proof*⟩

In particular, matrices over Stone algebras form a Stone algebra.

interpretation *matrix-stone-algebra*: *stone-algebra* **where** *sup* = *sup-matrix*
and *inf* = *inf-matrix* **and** *less-eq* = *less-eq-matrix* **and** *less* = *less-matrix* **and**
bot = *bot-matrix* :: (*'a','b*::*stone-algebra*) *square* **and** *top* = *top-matrix* **and**
uminus = *uminus-matrix*

<proof>

interpretation *matrix-heyting-stone-algebra: heyting-stone-algebra* **where** *sup = sup-matrix* **and** *inf = inf-matrix* **and** *less-eq = less-eq-matrix* **and** *less = less-matrix* **and** *bot = bot-matrix* **::** *('a,'b::heyting-stone-algebra)* **square** **and** *top = top-matrix* **and** *uminus = uminus-matrix* **and** *implies = implies-matrix*
<proof>

interpretation *matrix-boolean-algebra: boolean-algebra* **where** *sup = sup-matrix* **and** *inf = inf-matrix* **and** *less-eq = less-eq-matrix* **and** *less = less-matrix* **and** *bot = bot-matrix* **::** *('a,'b::boolean-algebra)* **square** **and** *top = top-matrix* **and** *uminus = uminus-matrix* **and** *minus = minus-matrix*
<proof>

6.4 Semirings

Next, we lift the semiring structure. Because of composition, this requires a restriction to finite matrices.

interpretation *matrix-monoid: monoid-mult* **where** *times = times-matrix* **and** *one = one-matrix* **::** *('a::finite,'b::idempotent-semiring)* **square**
<proof>

interpretation *matrix-idempotent-semiring: idempotent-semiring* **where** *sup = sup-matrix* **and** *less-eq = less-eq-matrix* **and** *less = less-matrix* **and** *bot = bot-matrix* **::** *('a::finite,'b::idempotent-semiring)* **square** **and** *one = one-matrix* **and** *times = times-matrix*
<proof>

interpretation *matrix-bounded-idempotent-semiring: bounded-idempotent-semiring* **where** *sup = sup-matrix* **and** *less-eq = less-eq-matrix* **and** *less = less-matrix* **and** *bot = bot-matrix* **::** *('a::finite,'b::bounded-idempotent-semiring)* **square** **and** *top = top-matrix* **and** *one = one-matrix* **and** *times = times-matrix*
<proof>

6.5 Stone Relation Algebras

Finally, we show that matrices over Stone relation algebras form a Stone relation algebra.

interpretation *matrix-stone-relation-algebra: stone-relation-algebra* **where** *sup = sup-matrix* **and** *inf = inf-matrix* **and** *less-eq = less-eq-matrix* **and** *less = less-matrix* **and** *bot = bot-matrix* **::** *('a::finite,'b::stone-relation-algebra)* **square** **and** *top = top-matrix* **and** *uminus = uminus-matrix* **and** *one = one-matrix* **and** *times = times-matrix* **and** *conv = conv-matrix*
<proof>

end

7 Matrices over Bounded Linear Orders

In this theory we characterise relation-algebraic properties of matrices over bounded linear orders (for example, extended real numbers) in terms of the entries in the matrices. We consider, in particular, the following properties: univalent, injective, total, surjective, mapping, bijective, vector, covec-
tor, point, arc, reflexive, coreflexive, irreflexive, symmetric, antisymmetric, asymmetric. We also consider the effect of composition with the matrix of greatest elements and with coreflexives (tests).

theory *Linear-Order-Matrices*

imports *Matrix-Relation-Algebras*

begin

class *non-trivial-linorder-stone-relation-algebra-expansion* =
linorder-stone-relation-algebra-expansion + *non-trivial*
begin

subclass *non-trivial-bounded-order* \langle *proof* \rangle

end

Before we look at matrices, we generalise selectivity to finite suprema.

lemma *linorder-finite-sup-selective*:

fixes $f :: 'a::finite \Rightarrow 'b::linorder-stone-relation-algebra-expansion$
shows $\exists i . (\bigsqcup_k f k) = f i$
 \langle *proof* \rangle

lemma *linorder-top-finite-sup*:

fixes $f :: 'a::finite \Rightarrow 'b::linorder-stone-relation-algebra-expansion$
assumes $\forall k . f k \neq top$
shows $(\bigsqcup_k f k) \neq top$
 \langle *proof* \rangle

The following results show the effect of composition with the *top* matrix from the left and from the right.

lemma *comp-top-linorder-matrix*:

fixes $f :: ('a::finite, 'b::linorder-stone-relation-algebra-expansion) square$
shows $(f \odot mtop) (i, j) = (\bigsqcup_k f (i, k))$
 \langle *proof* \rangle

lemma *top-comp-linorder-matrix*:

fixes $f :: ('a::finite, 'b::linorder-stone-relation-algebra-expansion) square$
shows $(mtop \odot f) (i, j) = (\bigsqcup_k f (k, j))$
 \langle *proof* \rangle

We characterise univalent matrices: in each row, at most one entry may be different from *bot*.

lemma *univalent-linorder-matrix-1*:
fixes $f :: ('a::\text{finite}, 'b::\text{linorder-stone-relation-algebra-expansion}) \text{square}$
assumes $\text{matrix-stone-relation-algebra.univalent } f$
and $f (i,j) \neq \text{bot}$
and $f (i,k) \neq \text{bot}$
shows $j = k$
 $\langle \text{proof} \rangle$

lemma *univalent-linorder-matrix-2*:
fixes $f :: ('a::\text{finite}, 'b::\text{linorder-stone-relation-algebra-expansion}) \text{square}$
assumes $\forall i j k . f (i,j) \neq \text{bot} \wedge f (i,k) \neq \text{bot} \longrightarrow j = k$
shows $\text{matrix-stone-relation-algebra.univalent } f$
 $\langle \text{proof} \rangle$

lemma *univalent-linorder-matrix*:
fixes $f :: ('a::\text{finite}, 'b::\text{linorder-stone-relation-algebra-expansion}) \text{square}$
shows $\text{matrix-stone-relation-algebra.univalent } f \longleftrightarrow (\forall i j k . f (i,j) \neq \text{bot} \wedge f (i,k) \neq \text{bot} \longrightarrow j = k)$
 $\langle \text{proof} \rangle$

Injective matrices can then be characterised by applying converse: in each column, at most one entry may be different from *bot*.

lemma *injective-linorder-matrix*:
fixes $f :: ('a::\text{finite}, 'b::\text{linorder-stone-relation-algebra-expansion}) \text{square}$
shows $\text{matrix-stone-relation-algebra.injective } f \longleftrightarrow (\forall i j k . f (j,i) \neq \text{bot} \wedge f (k,i) \neq \text{bot} \longrightarrow j = k)$
 $\langle \text{proof} \rangle$

Next come total matrices: each row has a *top* entry.

lemma *total-linorder-matrix-1*:
fixes $f :: ('a::\text{finite}, 'b::\text{linorder-stone-relation-algebra-expansion}) \text{square}$
assumes $\text{matrix-stone-relation-algebra.total-var } f$
shows $\exists j . f (i,j) = \text{top}$
 $\langle \text{proof} \rangle$

lemma *total-linorder-matrix-2*:
fixes $f :: ('a::\text{finite}, 'b::\text{linorder-stone-relation-algebra-expansion}) \text{square}$
assumes $\forall i . \exists j . f (i,j) = \text{top}$
shows $\text{matrix-stone-relation-algebra.total-var } f$
 $\langle \text{proof} \rangle$

lemma *total-linorder-matrix*:
fixes $f :: ('a::\text{finite}, 'b::\text{linorder-stone-relation-algebra-expansion}) \text{square}$
shows $\text{matrix-bounded-idempotent-semiring.total } f \longleftrightarrow (\forall i . \exists j . f (i,j) = \text{top})$
 $\langle \text{proof} \rangle$

Surjective matrices are again characterised by applying converse: each column has a *top* entry.

lemma *surjective-linorder-matrix*:

fixes $f :: ('a::finite, 'b::linorder-stone-relation-algebra-expansion)$ square
shows $matrix-bounded-idempotent-semiring.surjective\ f \longleftrightarrow (\forall j . \exists i . f\ (i,j) = top)$
 $\langle proof \rangle$

A mapping therefore means that each row has exactly one *top* entry and all others are *bot*.

lemma *mapping-linorder-matrix*:

fixes $f :: ('a::finite, 'b::linorder-stone-relation-algebra-expansion)$ square
shows $matrix-stone-relation-algebra.mapping\ f \longleftrightarrow (\forall i . \exists j . f\ (i,j) = top \wedge (\forall k . j \neq k \longrightarrow f\ (i,k) = bot))$
 $\langle proof \rangle$

lemma *mapping-linorder-matrix-unique*:

fixes $f :: ('a::finite, 'b::non-trivial-linorder-stone-relation-algebra-expansion)$ square
shows $matrix-stone-relation-algebra.mapping\ f \longleftrightarrow (\forall i . \exists! j . f\ (i,j) = top \wedge (\forall k . j \neq k \longrightarrow f\ (i,k) = bot))$
 $\langle proof \rangle$

Conversely, bijective means that each column has exactly one *top* entry and all others are *bot*.

lemma *bijective-linorder-matrix*:

fixes $f :: ('a::finite, 'b::linorder-stone-relation-algebra-expansion)$ square
shows $matrix-stone-relation-algebra.bijective\ f \longleftrightarrow (\forall j . \exists i . f\ (i,j) = top \wedge (\forall k . i \neq k \longrightarrow f\ (k,j) = bot))$
 $\langle proof \rangle$

lemma *bijective-linorder-matrix-unique*:

fixes $f :: ('a::finite, 'b::non-trivial-linorder-stone-relation-algebra-expansion)$ square
shows $matrix-stone-relation-algebra.bijective\ f \longleftrightarrow (\forall j . \exists! i . f\ (i,j) = top \wedge (\forall k . i \neq k \longrightarrow f\ (k,j) = bot))$
 $\langle proof \rangle$

We derive algebraic characterisations of matrices in which each row has an entry that is different from *bot*.

lemma *pp-total-linorder-matrix-1*:

fixes $f :: ('a::finite, 'b::non-trivial-linorder-stone-relation-algebra-expansion)$ square
assumes $\ominus(f \odot mtop) = mbot$
shows $\exists j . f\ (i,j) \neq bot$
 $\langle proof \rangle$

lemma *pp-total-linorder-matrix-2*:

fixes $f :: ('a::finite, 'b::linorder-stone-relation-algebra-expansion)$ square
assumes $\forall i . \exists j . f\ (i,j) \neq bot$
shows $\ominus(f \odot mtop) = mbot$
 $\langle proof \rangle$

lemma *pp-total-linorder-matrix-3:*

fixes $f :: ('a::\text{finite}, 'b::\text{non-trivial-linorder-stone-relation-algebra-expansion})$
square
shows $\ominus(f \odot \text{mtop}) = \text{mbot} \longleftrightarrow (\forall i . \exists j . f (i,j) \neq \text{bot})$
<proof>

lemma *pp-total-linorder-matrix:*

fixes $f :: ('a::\text{finite}, 'b::\text{non-trivial-linorder-stone-relation-algebra-expansion})$
square
shows *matrix-bounded-idempotent-semiring.total* $(\ominus \ominus f) \longleftrightarrow (\forall i . \exists j . f (i,j) \neq \text{bot})$
<proof>

lemma *pp-mapping-linorder-matrix:*

fixes $f :: ('a::\text{finite}, 'b::\text{non-trivial-linorder-stone-relation-algebra-expansion})$
square
shows *matrix-stone-relation-algebra.pp-mapping* $f \longleftrightarrow (\forall i . \exists j . f (i,j) \neq \text{bot} \wedge (\forall k . j \neq k \longrightarrow f (i,k) = \text{bot}))$
<proof>

lemma *pp-mapping-linorder-matrix-unique:*

fixes $f :: ('a::\text{finite}, 'b::\text{non-trivial-linorder-stone-relation-algebra-expansion})$
square
shows *matrix-stone-relation-algebra.pp-mapping* $f \longleftrightarrow (\forall i . \exists! j . f (i,j) \neq \text{bot} \wedge (\forall k . j \neq k \longrightarrow f (i,k) = \text{bot}))$
<proof>

Next follow matrices in which each column has an entry that is different from *bot*.

lemma *pp-surjective-linorder-matrix-1:*

fixes $f :: ('a::\text{finite}, 'b::\text{non-trivial-linorder-stone-relation-algebra-expansion})$
square
shows $\ominus(\text{mtop} \odot f) = \text{mbot} \longleftrightarrow (\forall j . \exists i . f (i,j) \neq \text{bot})$
<proof>

lemma *pp-surjective-linorder-matrix:*

fixes $f :: ('a::\text{finite}, 'b::\text{non-trivial-linorder-stone-relation-algebra-expansion})$
square
shows *matrix-bounded-idempotent-semiring.surjective* $(\ominus \ominus f) \longleftrightarrow (\forall j . \exists i . f (i,j) \neq \text{bot})$
<proof>

lemma *pp-bijective-linorder-matrix:*

fixes $f :: ('a::\text{finite}, 'b::\text{non-trivial-linorder-stone-relation-algebra-expansion})$
square
shows *matrix-stone-relation-algebra.pp-bijective* $f \longleftrightarrow (\forall j . \exists i . f (i,j) \neq \text{bot} \wedge (\forall k . i \neq k \longrightarrow f (k,j) = \text{bot}))$
<proof>

lemma *pp-bijective-linorder-matrix-unique:*

fixes $f :: ('a::\text{finite}, 'b::\text{non-trivial-linorder-stone-relation-algebra-expansion})$
square
shows $\text{matrix-stone-relation-algebra.pp-bijective } f \longleftrightarrow (\forall j . \exists ! i . f (i,j) \neq \text{bot})$
 $\wedge (\forall k . i \neq k \longrightarrow f (k,j) = \text{bot})$
(*proof*)

The regular matrices are those which contain only *bot* or *top* entries.

lemma *regular-linorder-matrix:*

fixes $f :: ('a::\text{finite}, 'b::\text{linorder-stone-relation-algebra-expansion})$ *square*
shows $\text{matrix-p-algebra.regular } f \longleftrightarrow (\forall e . f e = \text{bot} \vee f e = \text{top})$
(*proof*)

Vectors are precisely the row-constant matrices.

lemma *vector-linorder-matrix-0:*

fixes $f :: ('a::\text{finite}, 'b::\text{linorder-stone-relation-algebra-expansion})$ *square*
assumes $\text{matrix-bounded-idempotent-semiring.vector } f$
shows $f (i,j) = (\bigsqcup_k f (i,k))$
(*proof*)

lemma *vector-linorder-matrix-1:*

fixes $f :: ('a::\text{finite}, 'b::\text{linorder-stone-relation-algebra-expansion})$ *square*
assumes $\text{matrix-bounded-idempotent-semiring.vector } f$
shows $f (i,j) = f (i,k)$
(*proof*)

lemma *vector-linorder-matrix-2:*

fixes $f :: ('a::\text{finite}, 'b::\text{linorder-stone-relation-algebra-expansion})$ *square*
assumes $\forall i j k . f (i,j) = f (i,k)$
shows $\text{matrix-bounded-idempotent-semiring.vector } f$
(*proof*)

lemma *vector-linorder-matrix:*

fixes $f :: ('a::\text{finite}, 'b::\text{linorder-stone-relation-algebra-expansion})$ *square*
shows $\text{matrix-bounded-idempotent-semiring.vector } f \longleftrightarrow (\forall i j k . f (i,j) = f (i,k))$
(*proof*)

Hence covectors are precisely the column-constant matrices.

lemma *covector-linorder-matrix-0:*

fixes $f :: ('a::\text{finite}, 'b::\text{linorder-stone-relation-algebra-expansion})$ *square*
assumes $\text{matrix-bounded-idempotent-semiring.covector } f$
shows $f (i,j) = (\bigsqcup_k f (k,j))$
(*proof*)

lemma *covector-linorder-matrix:*

fixes $f :: ('a::\text{finite}, 'b::\text{linorder-stone-relation-algebra-expansion})$ *square*
shows $\text{matrix-bounded-idempotent-semiring.covector } f \longleftrightarrow (\forall i j k . f (i,j) = f (k,j))$

<proof>

A point is a matrix that has exactly one row, which is constant *top*, and all other rows are constant *bot*.

lemma *point-linorder-matrix*:

fixes $f :: ('a::\text{finite}, 'b::\text{linorder-stone-relation-algebra-expansion}) \text{ square}$

shows $\text{matrix-stone-relation-algebra.point } f \longleftrightarrow (\exists i . \forall j . f (i,j) = \text{top} \wedge (\forall k . i \neq k \longrightarrow f (k,j) = \text{bot}))$

<proof>

lemma *point-linorder-matrix-unique*:

fixes $f :: ('a::\text{finite}, 'b::\text{non-trivial-linorder-stone-relation-algebra-expansion}) \text{ square}$

square

shows $\text{matrix-stone-relation-algebra.point } f \longleftrightarrow (\exists !i . \forall j . f (i,j) = \text{top} \wedge (\forall k . i \neq k \longrightarrow f (k,j) = \text{bot}))$

<proof>

lemma *pp-point-linorder-matrix*:

fixes $f :: ('a::\text{finite}, 'b::\text{non-trivial-linorder-stone-relation-algebra-expansion}) \text{ square}$

square

shows $\text{matrix-stone-relation-algebra.pp-point } f \longleftrightarrow (\exists i . \forall j . f (i,j) \neq \text{bot} \wedge (\forall k . f (i,j) = f (i,k)) \wedge (\forall k . i \neq k \longrightarrow f (k,j) = \text{bot}))$

<proof>

lemma *pp-point-linorder-matrix-unique*:

fixes $f :: ('a::\text{finite}, 'b::\text{non-trivial-linorder-stone-relation-algebra-expansion}) \text{ square}$

square

shows $\text{matrix-stone-relation-algebra.pp-point } f \longleftrightarrow (\exists !i . \forall j . f (i,j) \neq \text{bot} \wedge (\forall k . f (i,j) = f (i,k)) \wedge (\forall k . i \neq k \longrightarrow f (k,j) = \text{bot}))$

<proof>

An arc is a matrix that has exactly one *top* entry and all other entries are *bot*.

lemma *arc-linorder-matrix-1*:

fixes $f :: ('a::\text{finite}, 'b::\text{non-trivial-linorder-stone-relation-algebra-expansion}) \text{ square}$

square

assumes $\text{matrix-stone-relation-algebra.arc } f$

shows $\exists e . f e = \text{top} \wedge (\forall d . e \neq d \longrightarrow f d = \text{bot})$

<proof>

lemma *pp-arc-linorder-matrix-2*:

fixes $f :: ('a::\text{finite}, 'b::\text{linorder-stone-relation-algebra-expansion}) \text{ square}$

assumes $\exists e . f e \neq \text{bot} \wedge (\forall d . e \neq d \longrightarrow f d = \text{bot})$

shows $\text{matrix-stone-relation-algebra.pp-arc } f$

<proof>

lemma *arc-linorder-matrix-2*:

fixes $f :: ('a::\text{finite}, 'b::\text{non-trivial-linorder-stone-relation-algebra-expansion}) \text{ square}$

square

assumes $\exists e . f e = top \wedge (\forall d . e \neq d \longrightarrow f d = bot)$
shows *matrix-stone-relation-algebra.arc f*
 <proof>

lemma *arc-linorder-matrix:*

fixes $f :: ('a::finite, 'b::non-trivial-linorder-stone-relation-algebra-expansion)$
square
shows *matrix-stone-relation-algebra.arc f* $\longleftrightarrow (\exists e . f e = top \wedge (\forall d . e \neq d \longrightarrow f d = bot))$
 <proof>

lemma *arc-linorder-matrix-unique:*

fixes $f :: ('a::finite, 'b::non-trivial-linorder-stone-relation-algebra-expansion)$
square
shows *matrix-stone-relation-algebra.arc f* $\longleftrightarrow (\exists !e . f e = top \wedge (\forall d . e \neq d \longrightarrow f d = bot))$
 <proof>

lemma *pp-arc-linorder-matrix-1:*

fixes $f :: ('a::finite, 'b::non-trivial-linorder-stone-relation-algebra-expansion)$
square
assumes *matrix-stone-relation-algebra.pp-arc f*
shows $\exists e . f e \neq bot \wedge (\forall d . e \neq d \longrightarrow f d = bot)$
 <proof>

lemma *pp-arc-linorder-matrix:*

fixes $f :: ('a::finite, 'b::non-trivial-linorder-stone-relation-algebra-expansion)$
square
shows *matrix-stone-relation-algebra.pp-arc f* $\longleftrightarrow (\exists e . f e \neq bot \wedge (\forall d . e \neq d \longrightarrow f d = bot))$
 <proof>

lemma *pp-arc-linorder-matrix-unique:*

fixes $f :: ('a::finite, 'b::non-trivial-linorder-stone-relation-algebra-expansion)$
square
shows *matrix-stone-relation-algebra.pp-arc f* $\longleftrightarrow (\exists !e . f e \neq bot \wedge (\forall d . e \neq d \longrightarrow f d = bot))$
 <proof>

Reflexive matrices are those with a constant *top* diagonal.

lemma *reflexive-linorder-matrix-1:*

fixes $f :: ('a::finite, 'b::linorder-stone-relation-algebra-expansion)$ *square*
assumes *matrix-idempotent-semiring.reflexive f*
shows $f (i, i) = top$
 <proof>

lemma *reflexive-linorder-matrix-2:*

fixes $f :: ('a::finite, 'b::linorder-stone-relation-algebra-expansion)$ *square*
assumes $\forall i . f (i, i) = top$

shows *matrix-idempotent-semiring.reflexive f*
<proof>

lemma *reflexive-linorder-matrix:*

fixes $f :: ('a::\text{finite}, 'b::\text{linorder-stone-relation-algebra-expansion}) \text{square}$
shows *matrix-idempotent-semiring.reflexive f* $\longleftrightarrow (\forall i . f (i,i) = \text{top})$
<proof>

Coreflexive matrices are those in which all non-diagonal entries are *bot*.

lemma *coreflexive-linorder-matrix-1:*

fixes $f :: ('a::\text{finite}, 'b::\text{linorder-stone-relation-algebra-expansion}) \text{square}$
assumes *matrix-idempotent-semiring.coreflexive f*
and $i \neq j$
shows $f (i,j) = \text{bot}$
<proof>

lemma *coreflexive-linorder-matrix-2:*

fixes $f :: ('a::\text{finite}, 'b::\text{linorder-stone-relation-algebra-expansion}) \text{square}$
assumes $\forall i j . i \neq j \longrightarrow f (i,j) = \text{bot}$
shows *matrix-idempotent-semiring.coreflexive f*
<proof>

lemma *coreflexive-linorder-matrix:*

fixes $f :: ('a::\text{finite}, 'b::\text{linorder-stone-relation-algebra-expansion}) \text{square}$
shows *matrix-idempotent-semiring.coreflexive f* $\longleftrightarrow (\forall i j . i \neq j \longrightarrow f (i,j) = \text{bot})$
<proof>

Irreflexive matrices are those with a constant *bot* diagonal.

lemma *irreflexive-linorder-matrix-1:*

fixes $f :: ('a::\text{finite}, 'b::\text{linorder-stone-relation-algebra-expansion}) \text{square}$
assumes *matrix-stone-relation-algebra.irreflexive f*
shows $f (i,i) = \text{bot}$
<proof>

lemma *irreflexive-linorder-matrix-2:*

fixes $f :: ('a::\text{finite}, 'b::\text{linorder-stone-relation-algebra-expansion}) \text{square}$
assumes $\forall i . f (i,i) = \text{bot}$
shows *matrix-stone-relation-algebra.irreflexive f*
<proof>

lemma *irreflexive-linorder-matrix:*

fixes $f :: ('a::\text{finite}, 'b::\text{linorder-stone-relation-algebra-expansion}) \text{square}$
shows *matrix-stone-relation-algebra.irreflexive f* $\longleftrightarrow (\forall i . f (i,i) = \text{bot})$
<proof>

As usual, symmetric matrices are those which do not change under transposition.

lemma *symmetric-linorder-matrix:*

fixes $f :: ('a::finite, 'b::linorder-stone-relation-algebra-expansion) \text{ square}$
shows $\text{matrix-stone-relation-algebra.symmetric } f \longleftrightarrow (\forall i j . f (i,j) = f (j,i))$
 $\langle \text{proof} \rangle$

Antisymmetric matrices are characterised as follows: each entry not on the diagonal or its mirror entry across the diagonal must be *bot*.

lemma *antisymmetric-linorder-matrix:*

fixes $f :: ('a::finite, 'b::linorder-stone-relation-algebra-expansion) \text{ square}$
shows $\text{matrix-stone-relation-algebra.antisymmetric } f \longleftrightarrow (\forall i j . i \neq j \longrightarrow f (i,j) = \text{bot} \vee f (j,i) = \text{bot})$
 $\langle \text{proof} \rangle$

For asymmetric matrices the diagonal is included: each entry or its mirror entry across the diagonal must be *bot*.

lemma *asymmetric-linorder-matrix:*

fixes $f :: ('a::finite, 'b::linorder-stone-relation-algebra-expansion) \text{ square}$
shows $\text{matrix-stone-relation-algebra.asymmetric } f \longleftrightarrow (\forall i j . f (i,j) = \text{bot} \vee f (j,i) = \text{bot})$
 $\langle \text{proof} \rangle$

In a transitive matrix, the weight of one of the edges on an indirect route must be below the weight of the direct edge.

lemma *transitive-linorder-matrix:*

fixes $f :: ('a::finite, 'b::linorder-stone-relation-algebra-expansion) \text{ square}$
shows $\text{matrix-idempotent-semiring.transitive } f \longleftrightarrow (\forall i j k . f (i,k) \leq f (i,j) \vee f (k,j) \leq f (i,j))$
 $\langle \text{proof} \rangle$

We finally show the effect of composing with a coreflexive (test) from the left and from the right. This amounts to a restriction of each row or column to the entry on the diagonal of the coreflexive. In this case, restrictions are formed by meets.

lemma *coreflexive-comp-linorder-matrix:*

fixes $f g :: ('a::finite, 'b::linorder-stone-relation-algebra-expansion) \text{ square}$
assumes $\text{matrix-idempotent-semiring.coreflexive } f$
shows $(f \odot g) (i,j) = f (i,i) \sqcap g (i,j)$
 $\langle \text{proof} \rangle$

lemma *comp-coreflexive-linorder-matrix:*

fixes $f g :: ('a::finite, 'b::linorder-stone-relation-algebra-expansion) \text{ square}$
assumes $\text{matrix-idempotent-semiring.coreflexive } g$
shows $(f \odot g) (i,j) = f (i,j) \sqcap g (j,j)$
 $\langle \text{proof} \rangle$

end

References

- [1] C. J. Aarts, R. C. Backhouse, E. A. Boiten, H. Doornbos, N. van Gasteren, R. van Geldrop, P. F. Hoogendijk, E. Voermans, and J. van der Woude. Fixed-point calculus. *Inf. Process. Lett.*, 53(3):131–136, 1995.
- [2] H. Andr eka and S. Mikul as. Axiomatizability of positive algebras of binary relations. *Algebra Universalis*, 66(1–2):7–34, 2011.
- [3] A. Armstrong, S. Foster, G. Struth, and T. Weber. Relation algebra. *Archive of Formal Proofs*, 2016, first version 2014.
- [4] A. Armstrong, V. B. F. Gomes, G. Struth, and T. Weber. Kleene algebra. *Archive of Formal Proofs*, 2016, first version 2013.
- [5] R. Berghammer. *Ordnungen, Verb ande und Relationen mit Anwendungen*. Springer, second edition, 2012.
- [6] R. Berghammer and W. Guttmann. Closure, properties and closure properties of multirelations. In W. Kahl, M. Winter, and J. N. Oliveira, editors, *Relational and Algebraic Methods in Computer Science*, volume 9348 of *Lecture Notes in Computer Science*, pages 67–83. Springer, 2015.
- [7] R. Bird and O. de Moor. *Algebra of Programming*. Prentice Hall, 1997.
- [8] D. A. Bredihin and B. M. Schein. Representations of ordered semi-groups and lattices by binary relations. *Colloquium Mathematicum*, 39(1):1–12, 1978.
- [9] S. D. Comer. On connections between information systems, rough sets and algebraic logic. In C. Rauszer, editor, *Algebraic Methods in Logic and in Computer Science*, volume 28 of *Banach Center Publications*, pages 117–124. Institute of Mathematics, Polish Academy of Sciences, 1993.
- [10] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, second edition, 2002.
- [11] P. J. Freyd and A.  cedrov. *Categories, Allegories*, volume 39 of *North-Holland Mathematical Library*. Elsevier Science Publishers, 1990.
- [12] J. A. Goguen. L-fuzzy sets. *Journal of Mathematical Analysis and Applications*, 18(1):145–174, 1967.
- [13] W. Guttmann. Algebras for iteration and infinite computations. *Acta Inf.*, 49(5):343–359, 2012.

- [14] W. Guttman. Relation-algebraic verification of Prim’s minimum spanning tree algorithm. In A. Sampaio and F. Wang, editors, *Theoretical Aspects of Computing – ICTAC 2016*, volume 9965 of *Lecture Notes in Computer Science*, pages 51–68. Springer, 2016.
- [15] W. Guttman. Stone algebras. *Archive of Formal Proofs*, 2016.
- [16] W. Guttman. Stone relation algebras. In P. Höfner, D. Pous, and G. Struth, editors, *Relational and Algebraic Methods in Computer Science*, volume 10226 of *Lecture Notes in Computer Science*, pages 127–143. Springer, 2017.
- [17] R. Hirsch and I. Hodkinson. *Relation Algebras by Games*. Elsevier Science B.V., 2002.
- [18] Y. Kawahara and H. Furusawa. Crispness in Dedekind categories. *Bulletin of Informatics and Cybernetics*, 33(1–2):1–18, 2001.
- [19] Y. Kawahara, H. Furusawa, and M. Mori. Categorical representation theorems of fuzzy relations. *Information Sciences*, 119(3–4):235–251, 1999.
- [20] R. D. Maddux. Relation-algebraic semantics. *Theoretical Comput. Sci.*, 160(1–2):1–85, 1996.
- [21] R. D. Maddux. *Relation Algebras*. Elsevier B.V., 2006.
- [22] J. N. Oliveira. Extended static checking by calculation using the point-free transform. In A. Bove, L. S. Barbosa, A. Pardo, and J. S. Pinto, editors, *Language Engineering and Rigorous Software Development*, volume 5520 of *Lecture Notes in Computer Science*, pages 195–251. Springer, 2009.
- [23] R. Parikh. Propositional logics of programs: new directions. In M. Karpinski, editor, *Foundations of Computation Theory*, volume 158 of *Lecture Notes in Computer Science*, pages 347–359. Springer, 1983.
- [24] Z. Pawlak. Rough sets, rough relations and rough functions. *Fundamenta Informaticae*, 27(2–3):103–108, 1996.
- [25] D. Peleg. Concurrent dynamic logic. *J. ACM*, 34(2):450–479, 1987.
- [26] G. Schmidt. *Relational Mathematics*. Cambridge University Press, 2011.
- [27] G. Schmidt and T. Ströhlein. *Relations and Graphs*. Springer, 1993.
- [28] A. Tarski. On the calculus of relations. *The Journal of Symbolic Logic*, 6(3):73–89, 1941.

- [29] M. Winter. A new algebraic approach to L-fuzzy relations convenient to study crispness. *Information Sciences*, 139(3-4):233–252, 2001.