Stone Relation Algebras

Walter Guttmann

August 7, 2022

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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*
using *has-least-fixpoint-def is-least-fixpoint-def order.antisym* **by** *auto*

lemma *greatest-fixpoint-unique*:
has-greatest-fixpoint f ⟹ ∃!*x* . *is-greatest-fixpoint f x*
using *has-greatest-fixpoint-def is-greatest-fixpoint-def order.antisym* **by** *auto*

lemma *least-prefixpoint-unique*:
has-least-prefixpoint f ⟹ ∃!*x* . *is-least-prefixpoint f x*
using *has-least-prefixpoint-def is-least-prefixpoint-def order.antisym* **by** *auto*

lemma *greatest-postfixpoint-unique*:
has-greatest-postfixpoint f ⟹ ∃!*x* . *is-greatest-postfixpoint f x*
using *has-greatest-postfixpoint-def is-greatest-postfixpoint-def order.antisym* **by** *auto*

lemma *least-fixpoint*:
has-least-fixpoint f ⟹ *is-least-fixpoint f* (*μ f*)
by (*simp add*: *least-fixpoint-unique theI' the-least-fixpoint-def*)

lemma *greatest-fixpoint*:
has-greatest-fixpoint f ⟹ *is-greatest-fixpoint f* (*ν f*)
by (*simp add*: *greatest-fixpoint-unique theI' the-greatest-fixpoint-def*)

lemma *least-prefixpoint*:
has-least-prefixpoint f ⟹ *is-least-prefixpoint f* (*pμ f*)
by (*simp add*: *least-prefixpoint-unique theI' the-least-prefixpoint-def*)

lemma *greatest-postfixpoint*:
has-greatest-postfixpoint f ⟹ *is-greatest-postfixpoint f* (*pν f*)
by (*simp add*: *greatest-postfixpoint-unique theI' the-greatest-postfixpoint-def*)

lemma *least-fixpoint-same*:

is-least-fixpoint f x $\Longrightarrow$ *x* = μ *f*
by (*simp add*: *is-least-fixpoint-def order.antisym the-equality the-least-fixpoint-def*)

lemma *greatest-fixpoint-same*:
is-greatest-fixpoint f x $\Longrightarrow$ *x* = ν *f*
using *greatest-fixpoint greatest-fixpoint-unique has-greatest-fixpoint-def* **by** *auto*

lemma *least-prefixpoint-same*:
is-least-prefixpoint f x $\Longrightarrow$ *x* = *p*μ *f*
using *has-least-prefixpoint-def least-prefixpoint least-prefixpoint-unique* **by** *blast*

lemma *greatest-postfixpoint-same*:
is-greatest-postfixpoint f x $\Longrightarrow$ *x* = *p*ν *f*
using *greatest-postfixpoint greatest-postfixpoint-unique has-greatest-postfixpoint-def* **by** *auto*

lemma *least-fixpoint-char*:
is-least-fixpoint f x $\longleftrightarrow$ *has-least-fixpoint f* $\wedge$ *x* = μ *f*
using *has-least-fixpoint-def least-fixpoint-same* **by** *auto*

lemma *least-prefixpoint-char*:
is-least-prefixpoint f x $\longleftrightarrow$ *has-least-prefixpoint f* $\wedge$ *x* = *p*μ *f*
using *has-least-prefixpoint-def least-prefixpoint-same* **by** *auto*

lemma *greatest-fixpoint-char*:
is-greatest-fixpoint f x $\longleftrightarrow$ *has-greatest-fixpoint f* $\wedge$ *x* = ν *f*
using *greatest-fixpoint-same has-greatest-fixpoint-def* **by** *auto*

lemma *greatest-postfixpoint-char*:
is-greatest-postfixpoint f x $\longleftrightarrow$ *has-greatest-postfixpoint f* $\wedge$ *x* = *p*ν *f*
using *greatest-postfixpoint-same has-greatest-postfixpoint-def* **by** *auto*

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

lemma *mu-unfold*:
has-least-fixpoint f $\Longrightarrow$ *f* (μ *f*) = μ *f*
using *is-least-fixpoint-def least-fixpoint* **by** *auto*

lemma *pmu-unfold*:
has-least-prefixpoint f $\Longrightarrow$ *f* (*p*μ *f*) $\le$ *p*μ *f*
using *is-least-prefixpoint-def least-prefixpoint* **by** *blast*

lemma *nu-unfold*:
has-greatest-fixpoint f $\Longrightarrow$ ν *f* = *f* (ν *f*)
by (*metis is-greatest-fixpoint-def greatest-fixpoint*)

lemma *pnu-unfold*:
has-greatest-postfixpoint f $\Longrightarrow$ *p*ν *f* $\le$ *f* (*p*ν *f*)
using *greatest-postfixpoint is-greatest-postfixpoint-def* **by** *auto*

Pre-/postfixpoints of isotone functions are fixpoints.

lemma *least-prefixpoint-fixpoint*:
has-least-prefixpoint f $\Longrightarrow$ *isotone f* $\Longrightarrow$ *is-least-fixpoint f* $(p\mu\ f)$
using *is-least-fixpoint-def is-least-prefixpoint-def least-prefixpoint order.antisym isotone-def* **by** *auto*

lemma *pmu-mu*:
has-least-prefixpoint f $\Longrightarrow$ *isotone f* $\Longrightarrow$ $p\mu\ f = \mu\ f$
by (*simp add*: *least-fixpoint-same least-prefixpoint-fixpoint*)

lemma *greatest-postfixpoint-fixpoint*:
has-greatest-postfixpoint f $\Longrightarrow$ *isotone f* $\Longrightarrow$ *is-greatest-fixpoint f* $(p\nu\ f)$
using *greatest-postfixpoint is-greatest-fixpoint-def is-greatest-postfixpoint-def order.antisym isotone-def* **by** *auto*

lemma *pnu-nu*:
has-greatest-postfixpoint f $\Longrightarrow$ *isotone f* $\Longrightarrow$ $p\nu\ f = \nu\ f$
by (*simp add*: *greatest-fixpoint-same greatest-postfixpoint-fixpoint*)

The fixpoint operators preserve isotonicity.

lemma *pmu-isotone*:
has-least-prefixpoint f $\Longrightarrow$ *has-least-prefixpoint g* $\Longrightarrow$ $f \le\le g \Longrightarrow p\mu\ f \le p\mu\ g$
by (*metis is-least-prefixpoint-def least-prefixpoint order-trans lifted-less-eq-def*)

lemma *mu-isotone*:
has-least-prefixpoint f $\Longrightarrow$ *has-least-prefixpoint g* $\Longrightarrow$ *isotone f* $\Longrightarrow$ *isotone g* $\Longrightarrow f \le\le g \Longrightarrow \mu\ f \le \mu\ g$
using *pmu-isotone pmu-mu* **by** *fastforce*

lemma *pnu-isotone*:
has-greatest-postfixpoint f $\Longrightarrow$ *has-greatest-postfixpoint g* $\Longrightarrow f \le\le g \Longrightarrow p\nu\ f \le p\nu\ g$
by (*metis greatest-postfixpoint is-greatest-postfixpoint-def order-trans lifted-less-eq-def*)

lemma *nu-isotone*:
has-greatest-postfixpoint f $\Longrightarrow$ *has-greatest-postfixpoint g* $\Longrightarrow$ *isotone f* $\Longrightarrow$ *isotone g* $\Longrightarrow f \le\le g \Longrightarrow \nu\ f \le \nu\ g$
using *pnu-isotone pnu-nu* **by** *fastforce*

The square rule for fixpoints of a function applied twice.

lemma *mu-square*:
isotone f $\Longrightarrow$ *has-least-fixpoint f* $\Longrightarrow$ *has-least-fixpoint* $(f \circ f) \Longrightarrow \mu\ f = \mu\ (f \circ f)$
by (*metis* (*no-types*, *opaque-lifting*) *order.antisym is-least-fixpoint-def isotone-def least-fixpoint-char least-fixpoint-unique o-apply*)

lemma *nu-square*:

isotone $f \Longrightarrow$ *has-greatest-fixpoint* $f \Longrightarrow$ *has-greatest-fixpoint* $(f \circ f) \Longrightarrow \nu\ f = \nu\ (f \circ f)$
by (*metis* (*no-types*, *opaque-lifting*) *order.antisym is-greatest-fixpoint-def isotone-def greatest-fixpoint-char greatest-fixpoint-unique o-apply*)

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 (*rule order.antisym*)
show $\mu\ (g \circ f) \le g\ (\mu\ (f \circ g))$
by (*metis assms*(*2*−*3*) *comp-apply is-least-fixpoint-def least-fixpoint*)
next
have *is-least-fixpoint* $(g \circ f)\ (\mu\ (g \circ f))$
by (*simp add*: *assms*(*3*) *least-fixpoint*)
thus $g\ (\mu\ (f \circ g)) \le \mu\ (g \circ f)$
by (*metis* (*no-types*) *assms*(*1*−*2*) *comp-def is-least-fixpoint-def least-fixpoint isotone-def*)
qed

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 (*rule order.antisym*)
have *1*: *is-greatest-fixpoint* $(f \circ g)\ (\nu\ (f \circ g))$
by (*simp add*: *assms*(*2*) *greatest-fixpoint*)
have *is-greatest-fixpoint* $(g \circ f)\ (\nu\ (g \circ f))$
by (*simp add*: *assms*(*3*) *greatest-fixpoint*)
thus $\nu\ (g \circ f) \le g\ (\nu\ (f \circ g))$
using *1* **by** (*metis* (*no-types*) *assms*(*1*) *comp-def is-greatest-fixpoint-def isotone-def*)
next
show $g\ (\nu\ (f \circ g)) \le \nu\ (g \circ f)$
by (*metis assms*(*2*−*3*) *comp-apply greatest-fixpoint is-greatest-fixpoint-def*)
qed

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

lemma *mu-below-nu*:
has-least-fixpoint $f \Longrightarrow$ *has-greatest-fixpoint* $f \Longrightarrow \mu\ f \le \nu\ f$
using *greatest-fixpoint is-greatest-fixpoint-def mu-unfold* **by** *auto*

lemma *pmu-below-pnu-fix*:
has-fixpoint $f \Longrightarrow$ *has-least-prefixpoint* $f \Longrightarrow$ *has-greatest-postfixpoint* $f \Longrightarrow p\mu\ f \le p\nu\ f$
by (*metis greatest-postfixpoint has-fixpoint-def is-fixpoint-def*

is-greatest-postfixpoint-def is-least-prefixpoint-def least-prefixpoint order-refl order-trans)

lemma *pmu-below-pnu-iso*:
  isotone $f \Longrightarrow$ *has-least-prefixpoint* $f \Longrightarrow$ *has-greatest-postfixpoint* $f \Longrightarrow p\mu\ f \leq p\nu\ f$
  using *greatest-postfixpoint-fixpoint is-greatest-fixpoint-def is-least-fixpoint-def least-prefixpoint-fixpoint* **by** *auto*

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 −
  have $l\ (g\ (u\ (\mu\ h))) \leq \mu\ h$
    by (*metis assms*(*1*,*2*,*4*,*5*) *galois-char isotone-def order-lesseq-imp mu-unfold*)
  thus $l\ (p\mu\ g) \leq \mu\ h$
  using *assms*(*1*,*3*) *is-least-prefixpoint-def least-prefixpoint galois-def* **by** *auto*
qed

lemma *mu-fusion-2*:
  galois $l\ u \Longrightarrow$ *isotone* $h \Longrightarrow$ *has-least-prefixpoint* $g \Longrightarrow$ *has-least-fixpoint* $h \Longrightarrow l \circ g \leq\leq h \circ l \Longrightarrow l\ (p\mu\ g) \leq \mu\ h$
  by (*simp add*: *mu-fusion-1 lifted-less-eq-def*)

lemma *mu-fusion-equal-1*:
  galois $l\ u \Longrightarrow$ *isotone* $g \Longrightarrow$ *isotone* $h \Longrightarrow$ *has-least-prefixpoint* $g \Longrightarrow$ *has-least-fixpoint* $h \Longrightarrow l\ (g\ (u\ (\mu\ h))) \leq h(l(u(\mu\ h))) \Longrightarrow l\ (g\ (p\mu\ g)) = h\ (l\ (p\mu\ g)) \Longrightarrow \mu\ h = l\ (p\mu\ g) \wedge \mu\ h = l\ (\mu\ g)$
  by (*metis order.antisym least-fixpoint least-prefixpoint-fixpoint is-least-fixpoint-def mu-fusion-1 pmu-mu*)

lemma *mu-fusion-equal-2*:
  galois $l\ u \Longrightarrow$ *isotone* $h \Longrightarrow$ *has-least-prefixpoint* $g \Longrightarrow$ *has-least-prefixpoint* $h \Longrightarrow l\ (g\ (u\ (\mu\ h))) \leq h\ (l\ (u\ (\mu\ h))) \wedge l\ (g\ (p\mu\ g)) = h\ (l\ (p\mu\ g)) \longrightarrow p\mu\ h = l\ (p\mu\ g) \wedge \mu\ h = l\ (p\mu\ g)$
  by (*metis is-least-prefixpoint-def least-fixpoint-char least-prefixpoint least-prefixpoint-fixpoint order.antisym galois-char isotone-def mu-fusion-1*)

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 ∘ g = h ∘ l
    shows μ h = l (pμ g)
      and μ h = l (μ g)
proof -
  have ∀ x . l (g x) = h (l x)
    using assms(6) comp-eq-elim by blast
  thus μ h = l (pμ g)
    using assms(1-5) mu-fusion-equal-1 by auto
  thus μ h = l (μ g)
    by (simp add: assms(2,4) pmu-mu)
qed

lemma mu-fusion-equal-4:
  assumes galois l u
      and isotone h
      and has-least-prefixpoint g
      and has-least-prefixpoint h
      and l ∘ g = h ∘ l
    shows pμ h = l (pμ g)
      and μ h = l (pμ g)
proof -
  have ∀ x . l (g x) = h (l x)
    using assms(5) comp-eq-elim by blast
  thus pμ h = l (pμ g)
    using assms(1-4) mu-fusion-equal-2 by auto
  thus μ h = l (pμ g)
    by (simp add: assms(2,4) pmu-mu)
qed

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 (ν h))) ≤ u (g (l (ν h)))
    shows ν h ≤ u (pν g)
proof -
  have ν h ≤ u (g (l (ν h)))
    by (metis assms(1,2,4,5) order-trans galois-char isotone-def nu-unfold)
  thus ν h ≤ u (pν g)
    by (metis assms(1,3) greatest-postfixpoint is-greatest-postfixpoint-def
ord.galois-def)
qed

lemma nu-fusion-2:
  galois l u ⟹ isotone h ⟹ has-greatest-postfixpoint g ⟹ has-greatest-fixpoint
h ⟹ h ∘ u ≤≤ u ∘ g ⟹ ν h ≤ u (pν g)
  by (simp add: nu-fusion-1 lifted-less-eq-def)
```

```
lemma nu-fusion-equal-1:
  galois l u ⟹ isotone g ⟹ isotone h ⟹ has-greatest-postfixpoint g ⟹
has-greatest-fixpoint h ⟹ h (u (l (ν h))) ≤ u (g (l (ν h))) ⟹ h (u (pν g)) = u
(g (pν g)) ⟹ ν h = u (pν g) ∧ ν h = u (ν g)
  by (metis greatest-fixpoint-char greatest-postfixpoint-fixpoint
is-greatest-fixpoint-def order.antisym nu-fusion-1)

lemma nu-fusion-equal-2:
  galois l u ⟹ isotone h ⟹ has-greatest-postfixpoint g ⟹
has-greatest-postfixpoint h ⟹ h (u (l (ν h))) ≤ u (g (l (ν h))) ∧ h (u (pν g)) =
u (g (pν g)) ⟹ pν h = u (pν g) ∧ ν h = u (pν g)
  by (metis greatest-fixpoint-char greatest-postfixpoint greatest-postfixpoint-fixpoint
is-greatest-postfixpoint-def order.antisym galois-char nu-fusion-1 isotone-def)

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 ∘ u = u ∘ g
    shows ν h = u (pν g)
      and ν h = u (ν g)
proof −
  have ∀ x . u (g x) = h (u x)
    using assms(6) comp-eq-dest by fastforce
  thus ν h = u (pν g)
    using assms(1−5) nu-fusion-equal-1 by auto
  thus ν h = u (ν g)
    by (simp add: assms(2−4) pnu-nu)
qed

lemma nu-fusion-equal-4:
  assumes galois l u
      and isotone h
      and has-greatest-postfixpoint g
      and has-greatest-postfixpoint h
      and h ∘ u = u ∘ g
    shows pν h = u (pν g)
      and ν h = u (pν g)
proof −
  have ∀ x . u (g x) = h (u x)
    using assms(5) comp-eq-dest by fastforce
  thus pν h = u (pν g)
    using assms(1−4) nu-fusion-equal-2 by auto
  thus ν h = u (pν g)
    by (simp add: assms(2,4) pnu-nu)
qed
```

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 ∘ h)
      and has-least-prefixpoint (h ∘ g)
      and has-least-fixpoint (g ∘ h)
      and l ∘ h ∘ g ≤≤ g ∘ h ∘ l
    shows μ (l ∘ h) ≤ μ (g ∘ h)
proof −
  have 1: l ∘ (h ∘ g) ≤≤ (g ∘ h) ∘ l
    by (simp add: assms(7) rewriteL-comp-comp)
  have (l ∘ h) (μ (g ∘ h)) = l (μ (h ∘ g))
    by (metis assms(2,3,5,6) comp-apply least-fixpoint-char
least-prefixpoint-fixpoint isotone-def mu-roll)
  also have ... ≤ μ (g ∘ h)
    using 1 by (metis assms(1−3,5,6) comp-apply least-fixpoint-char
least-prefixpoint-fixpoint isotone-def mu-fusion-2)
  finally have pμ (l ∘ h) ≤ μ (g ∘ h)
    using assms(4) is-least-prefixpoint-def least-prefixpoint by blast
  thus μ (l ∘ h) ≤ μ (g ∘ h)
    by (metis assms(1,3,4) galois-char isotone-def least-fixpoint-char
least-prefixpoint-fixpoint o-apply)
qed

lemma mu-exchange-2:
  assumes galois l u
      and isotone g
      and isotone h
      and has-least-prefixpoint (l ∘ h)
      and has-least-prefixpoint (h ∘ l)
      and has-least-prefixpoint (h ∘ g)
      and has-least-fixpoint (g ∘ h)
      and has-least-fixpoint (h ∘ g)
      and l ∘ h ∘ g ≤≤ g ∘ h ∘ l
    shows μ (h ∘ l) ≤ μ (h ∘ g)
proof −
  have μ (h ∘ l) = h (μ (l ∘ h))
    by (metis (no-types, lifting) assms(1,3−5) galois-char isotone-def
least-fixpoint-char least-prefixpoint-fixpoint mu-roll o-apply)
  also have ... ≤ h (μ (g ∘ h))
    using assms(1−4,6,7,9) isotone-def mu-exchange-1 by blast
  also have ... = μ (h ∘ g)
    by (simp add: assms(3,7,8) mu-roll)
  finally show ?thesis
    .
qed
```

```
lemma mu-exchange-equal:
  assumes galois l u
      and galois k t
      and isotone h
      and has-least-prefixpoint (l ∘ h)
      and has-least-prefixpoint (h ∘ l)
      and has-least-prefixpoint (k ∘ h)
      and has-least-prefixpoint (h ∘ k)
      and l ∘ h ∘ k = k ∘ h ∘ l
    shows μ (l ∘ h) = μ (k ∘ h)
      and μ (h ∘ l) = μ (h ∘ k)
proof −
  have 1: has-least-fixpoint (k ∘ h)
    using assms(2,3,6) least-fixpoint-char least-prefixpoint-fixpoint galois-char
isotone-def by auto
  have 2: has-least-fixpoint (h ∘ k)
    using assms(2,3,7) least-fixpoint-char least-prefixpoint-fixpoint galois-char
isotone-def by auto
  have 3: has-least-fixpoint (l ∘ h)
    using assms(1,3,4) least-fixpoint-char least-prefixpoint-fixpoint galois-char
isotone-def by auto
  have 4: has-least-fixpoint (h ∘ l)
    using assms(1,3,5) least-fixpoint-char least-prefixpoint-fixpoint galois-char
isotone-def by auto
  show μ (h ∘ l) = μ (h ∘ k)
    using 1 2 3 4 assms order.antisym galois-char lifted-reflexive mu-exchange-2
by auto
  show μ (l ∘ h) = μ (k ∘ h)
    using 1 2 3 4 assms order.antisym galois-char lifted-reflexive mu-exchange-1
by auto
qed

lemma nu-exchange-1:
  assumes galois l u
      and isotone g
      and isotone h
      and has-greatest-postfixpoint (u ∘ h)
      and has-greatest-postfixpoint (h ∘ g)
      and has-greatest-fixpoint (g ∘ h)
      and g ∘ h ∘ u ≤≤ u ∘ h ∘ g
    shows ν (g ∘ h) ≤ ν (u ∘ h)
proof −
  have (g ∘ h) ∘ u ≤≤ u ∘ (h ∘ g)
    by (simp add: assms(7) o-assoc)
  hence ν (g ∘ h) ≤ u (ν (h ∘ g))
    by (metis assms(1−3,5,6) greatest-fixpoint-char greatest-postfixpoint-fixpoint
isotone-def nu-fusion-2 o-apply)
  also have ... = (u ∘ h) (ν (g ∘ h))
    by (metis assms(2,3,5,6) greatest-fixpoint-char greatest-postfixpoint-fixpoint
```

```
isotone-def nu-roll o-apply)
  finally have ν (g ∘ h) ≤ pν (u ∘ h)
    using assms(4) greatest-postfixpoint is-greatest-postfixpoint-def by blast
  thus ν (g ∘ h) ≤ ν (u ∘ h)
    using assms(1,3,4) galois-char greatest-fixpoint-char
greatest-postfixpoint-fixpoint isotone-def by auto
qed

lemma nu-exchange-2:
  assumes galois l u
      and isotone g
      and isotone h
      and has-greatest-postfixpoint (u ∘ h)
      and has-greatest-postfixpoint (h ∘ u)
      and has-greatest-postfixpoint (h ∘ g)
      and has-greatest-fixpoint (g ∘ h)
      and has-greatest-fixpoint (h ∘ g)
      and g ∘ h ∘ u ≤≤ u ∘ h ∘ g
    shows ν (h ∘ g) ≤ ν (h ∘ u)
proof –
  have ν (h ∘ g) = h (ν (g ∘ h))
    by (simp add: assms(3,7,8) nu-roll)
  also have ... ≤ h (ν (u ∘ h))
    using assms(1–4,6,7,9) isotone-def nu-exchange-1 by blast
  also have ... = ν (h ∘ u)
    by (metis (no-types, lifting) assms(1,3–5) galois-char greatest-fixpoint-char
greatest-postfixpoint-fixpoint isotone-def nu-roll o-apply)
  finally show ν (h ∘ g) ≤ ν (h ∘ u)
    .
qed

lemma nu-exchange-equal:
  assumes galois l u
      and galois k t
      and isotone h
      and has-greatest-postfixpoint (u ∘ h)
      and has-greatest-postfixpoint (h ∘ u)
      and has-greatest-postfixpoint (t ∘ h)
      and has-greatest-postfixpoint (h ∘ t)
      and u ∘ h ∘ t = t ∘ h ∘ u
    shows ν (u ∘ h) = ν (t ∘ h)
      and ν (h ∘ u) = ν (h ∘ t)
proof –
  have 1: has-greatest-fixpoint (u ∘ h)
    using assms(1,3,4) greatest-fixpoint-char greatest-postfixpoint-fixpoint
galois-char isotone-def by auto
  have 2: has-greatest-fixpoint (h ∘ u)
    using assms(1,3,5) greatest-fixpoint-char greatest-postfixpoint-fixpoint
galois-char isotone-def by auto
```

have *3*: *has-greatest-fixpoint* $(t \circ h)$
using *assms*(*2*,*3*,*6*) *greatest-fixpoint-char greatest-postfixpoint-fixpoint galois-char isotone-def* **by** *auto*
have *4*: *has-greatest-fixpoint* $(h \circ t)$
using *assms*(*2*,*3*,*7*) *greatest-fixpoint-char greatest-postfixpoint-fixpoint galois-char isotone-def* **by** *auto*
show $\nu\ (u \circ h) = \nu\ (t \circ h)$
using *1 2 3 4 assms order.antisym galois-char lifted-reflexive nu-exchange-1* **by** *auto*
show $\nu\ (h \circ u) = \nu\ (h \circ t)$
using *1 2 3 4 assms order.antisym galois-char lifted-reflexive nu-exchange-2* **by** *auto*
qed

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 $f \Longrightarrow$ *has-least-fixpoint* $(f \circ g) \Longrightarrow f \circ g = g \circ f \Longrightarrow$ *is-fixpoint* $f\ (\mu\ (f \circ g))$
by (*metis is-fixpoint-def mu-roll*)

lemma *mu-commute-fixpoint-2*:
isotone $g \Longrightarrow$ *has-least-fixpoint* $(f \circ g) \Longrightarrow f \circ g = g \circ f \Longrightarrow$ *is-fixpoint* $g\ (\mu\ (f \circ g))$
by (*simp add*: *mu-commute-fixpoint-1*)

lemma *mu-commute-least-fixpoint*:
isotone $f \Longrightarrow$ *isotone* $g \Longrightarrow$ *has-least-fixpoint* $f \Longrightarrow$ *has-least-fixpoint* $g \Longrightarrow$ *has-least-fixpoint* $(f \circ g) \Longrightarrow f \circ g = g \circ f \Longrightarrow \mu\ (f \circ g) = \mu\ f \Longrightarrow \mu\ g \le \mu\ f$
by (*metis is-least-fixpoint-def least-fixpoint mu-roll*)

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 $f \Longrightarrow$ *has-greatest-fixpoint* $(f \circ g) \Longrightarrow f \circ g = g \circ f \Longrightarrow$ *is-fixpoint* $f\ (\nu(f \circ g))$
by (*metis is-fixpoint-def nu-roll*)

lemma *nu-commute-fixpoint-2*:
isotone $g \Longrightarrow$ *has-greatest-fixpoint* $(f \circ g) \Longrightarrow f \circ g = g \circ f \Longrightarrow$ *is-fixpoint* $g\ (\nu(f \circ g))$
by (*simp add*: *nu-commute-fixpoint-1*)

lemma *nu-commute-greatest-fixpoint*:
isotone $f \Longrightarrow$ *isotone* $g \Longrightarrow$ *has-greatest-fixpoint* $f \Longrightarrow$ *has-greatest-fixpoint* $g \Longrightarrow$ *has-greatest-fixpoint* $(f \circ g) \Longrightarrow f \circ g = g \circ f \Longrightarrow \nu\ (f \circ g) = \nu\ f \Longrightarrow \nu\ f \le \nu\ g$
by (*metis greatest-fixpoint is-greatest-fixpoint-def nu-roll*)

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

```
lemma mu-diagonal-1:
  assumes isotone (λx . μ (λy . f x y))
      and ∀ x . has-least-fixpoint (λy . f x y)
      and has-least-prefixpoint (λx . μ (λy . f x y))
    shows μ (λx . f x x) = μ (λx . μ (λy . f x y))
proof –
  let ?g = λx . μ (λy . f x y)
  have 1: is-least-prefixpoint ?g (μ ?g)
    using assms(1,3) least-prefixpoint pmu-mu by fastforce
  have f (μ ?g) (μ ?g) = μ ?g
    by (metis (no-types, lifting) assms is-least-fixpoint-def least-fixpoint-char least-prefixpoint-fixpoint)
  hence is-least-fixpoint (λx . f x x) (μ ?g)
    using 1 assms(2) is-least-fixpoint-def is-least-prefixpoint-def least-fixpoint by auto
  thus ?thesis
    using least-fixpoint-same by simp
qed

lemma mu-diagonal-2:
  ∀ x . isotone (λy . f x y) ∧ isotone (λy . f y x) ∧ has-least-prefixpoint (λy . f x y) ⟹ has-least-prefixpoint (λx . μ (λy . f x y)) ⟹ μ (λx . f x x) = μ (λx . μ (λy . f x y))
  apply (rule mu-diagonal-1)
  using isotone-def lifted-less-eq-def mu-isotone apply simp
  using has-least-fixpoint-def least-prefixpoint-fixpoint apply blast
  by simp

lemma nu-diagonal-1:
  assumes isotone (λx . ν (λy . f x y))
      and ∀ x . has-greatest-fixpoint (λy . f x y)
      and has-greatest-postfixpoint (λx . ν (λy . f x y))
    shows ν (λx . f x x) = ν (λx . ν (λy . f x y))
proof –
  let ?g = λx . ν (λy . f x y)
  have 1: is-greatest-postfixpoint ?g (ν ?g)
    using assms(1,3) greatest-postfixpoint pnu-nu by fastforce
  have f (ν ?g) (ν ?g) = ν ?g
    by (metis (no-types, lifting) assms is-greatest-fixpoint-def greatest-fixpoint-char greatest-postfixpoint-fixpoint)
  hence is-greatest-fixpoint (λx . f x x) (ν ?g)
    using 1 assms(2) is-greatest-fixpoint-def is-greatest-postfixpoint-def greatest-fixpoint by auto
  thus ?thesis
    using greatest-fixpoint-same by simp
qed
```

lemma *nu-diagonal-2*:
∀ *x* . *isotone* (λ*y* . *f x y*) ∧ *isotone* (λ*y* . *f y x*) ∧ *has-greatest-postfixpoint* (λ*y* . *f x y*) ⟹ *has-greatest-postfixpoint* (λ*x* . ν (λ*y* . *f x y*)) ⟹ ν (λ*x* . *f x x*) = ν (λ*x* . ν (λ*y* . *f x y*))
apply (*rule nu-diagonal-1*)
using *isotone-def lifted-less-eq-def nu-isotone* **apply** *simp*
using *has-greatest-fixpoint-def greatest-postfixpoint-fixpoint* **apply** *blast*
by *simp*

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* :: $'a \Rightarrow bool$ **where** *reflexive* $x \equiv 1 \leq x$
abbreviation *coreflexive* :: $'a \Rightarrow bool$ **where** *coreflexive* $x \equiv x \leq 1$

abbreviation *transitive* :: $'a \Rightarrow bool$ **where** *transitive* $x \equiv x * x \leq x$
abbreviation *dense-rel* :: $'a \Rightarrow bool$ **where** *dense-rel* $x \equiv x \leq x * x$
abbreviation *idempotent* :: $'a \Rightarrow bool$ **where** *idempotent* $x \equiv x * x = x$

abbreviation *preorder* :: $'a \Rightarrow bool$ **where** *preorder* $x \equiv$ *reflexive* $x \wedge$ *transitive* x

abbreviation *coreflexives* $\equiv \{ x$. *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*: $x * y \sqcup x * z \leq x * (y \sqcup z)$
  assumes *mult-right-dist-sup*: $(x \sqcup y) * z = x * z \sqcup y * z$
  assumes *mult-left-zero* [*simp*]: $bot * x = bot$
  assumes *mult-left-one* [*simp*]: $1 * x = x$
  assumes *mult-sub-right-one*: $x \leq x * 1$
begin

subclass *times-one-ord* **.**

We first show basic isotonicity and subdistributivity properties of multiplication.

lemma *mult-left-isotone*:
  $x \leq y \Longrightarrow x * z \leq y * z$
  using *mult-right-dist-sup sup-right-divisibility* **by** *auto*

lemma *mult-right-isotone*:
  $x \leq y \Longrightarrow z * x \leq z * y$
  using *mult-left-sub-dist-sup sup.bounded-iff sup-right-divisibility* **by** *auto*

lemma *mult-isotone*:
  $w \leq y \Longrightarrow x \leq z \Longrightarrow w * x \leq y * z$
  using *order-trans mult-left-isotone mult-right-isotone* **by** *blast*

lemma *affine-isotone*:
  isotone $(\lambda x \,.\, y * x \sqcup z)$
  using *isotone-def mult-right-isotone sup-left-isotone* **by** *auto*

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

```
  x ∗ y ≤ x ∗ (y ⊔ z)
  by (simp add: mult-right-isotone)

lemma mult-left-sub-dist-sup-right:
  x ∗ z ≤ x ∗ (y ⊔ z)
  by (simp add: mult-right-isotone)

lemma mult-right-sub-dist-sup-left:
  x ∗ z ≤ (x ⊔ y) ∗ z
  by (simp add: mult-left-isotone)

lemma mult-right-sub-dist-sup-right:
  y ∗ z ≤ (x ⊔ y) ∗ z
  by (simp add: mult-left-isotone)

lemma case-split-left:
  assumes 1 ≤ w ⊔ z
      and w ∗ x ≤ y
      and z ∗ x ≤ y
    shows x ≤ y
proof −
  have (w ⊔ z) ∗ x ≤ y
    by (simp add: assms(2−3) mult-right-dist-sup)
  thus ?thesis
    by (metis assms(1) dual-order.trans mult-left-one mult-left-isotone)
qed

lemma case-split-left-equal:
  w ⊔ z = 1 ⟹ w ∗ x = w ∗ y ⟹ z ∗ x = z ∗ y ⟹ x = y
  by (metis mult-left-one mult-right-dist-sup)
```

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

```
lemma reflexive-one-closed:
  reflexive 1
  by simp

lemma reflexive-sup-closed:
  reflexive x ⟹ reflexive (x ⊔ y)
  by (simp add: le-supI1)

lemma reflexive-mult-closed:
  reflexive x ⟹ reflexive y ⟹ reflexive (x ∗ y)
  using mult-isotone by fastforce

lemma coreflexive-bot-closed:
  coreflexive bot
  by simp
```

lemma *coreflexive-one-closed*:
coreflexive 1
by *simp*

lemma *coreflexive-sup-closed*:
coreflexive $x \Longrightarrow$ *coreflexive* $y \Longrightarrow$ *coreflexive* $(x \sqcup y)$
by *simp*

lemma *coreflexive-mult-closed*:
coreflexive $x \Longrightarrow$ *coreflexive* $y \Longrightarrow$ *coreflexive* $(x * y)$
using *mult-isotone* **by** *fastforce*

lemma *transitive-bot-closed*:
transitive bot
by *simp*

lemma *transitive-one-closed*:
transitive 1
by *simp*

lemma *dense-bot-closed*:
dense-rel bot
by *simp*

lemma *dense-one-closed*:
dense-rel 1
by *simp*

lemma *dense-sup-closed*:
dense-rel $x \Longrightarrow$ *dense-rel* $y \Longrightarrow$ *dense-rel* $(x \sqcup y)$
by (*metis mult-right-dist-sup order-lesseq-imp sup.mono mult-left-sub-dist-sup-left mult-left-sub-dist-sup-right*)

lemma *idempotent-bot-closed*:
idempotent bot
by *simp*

lemma *idempotent-one-closed*:
idempotent 1
by *simp*

lemma *preorder-one-closed*:
preorder 1
by *simp*

lemma *coreflexive-transitive*:
coreflexive $x \Longrightarrow$ *transitive* x
using *mult-left-isotone* **by** *fastforce*

lemma *preorder-idempotent*:
preorder x $\Longrightarrow$ *idempotent x*
using *order.antisym mult-isotone* **by** *fastforce*

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

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

lemma *lstar-rec-isotone*:
isotone (*Lf y*)
using *isotone-def sup-right-divisibility sup-right-isotone mult-right-sub-dist-sup-right* **by** *auto*

lemma *rstar-rec-isotone*:
isotone (*Rf y*)
using *isotone-def sup-right-divisibility sup-right-isotone mult-left-sub-dist-sup-right* **by** *auto*

lemma *sstar-rec-isotone*:
isotone (*Sf y*)
using *isotone-def sup-right-isotone mult-isotone* **by** *auto*

lemma *lstar-fixpoint*:
has-least-prefixpoint (*Lf y*) $\Longrightarrow$ *lstar y* $=$ *μ* (*Lf y*)
by (*simp add*: *pmu-mu lstar-rec-isotone*)

lemma *rstar-fixpoint*:
has-least-prefixpoint (*Rf y*) $\Longrightarrow$ *rstar y* $=$ *μ* (*Rf y*)
by (*simp add*: *pmu-mu rstar-rec-isotone*)

lemma *sstar-fixpoint*:
has-least-prefixpoint (*Sf y*) $\Longrightarrow$ *sstar y* $=$ *μ* (*Sf y*)
by (*simp add*: *pmu-mu sstar-rec-isotone*)

lemma *sstar-increasing*:
has-least-prefixpoint (*Sf y*) $\Longrightarrow y \le$ *sstar y*
using *order-trans pmu-unfold sup-ge1 sup-ge2* **by** *blast*

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 ≤ sstar y
proof –
  have y ≤ sstar y
    using assms(2) pmu-unfold by force
  hence Rf y (sstar y) ≤ Sf y (sstar y)
    by (meson sup.cobounded1 sup.mono mult-left-isotone)
  also have ... ≤ sstar y
    using assms(2) pmu-unfold by blast
  finally show ?thesis
    using assms(1) is-least-prefixpoint-def least-prefixpoint by auto
qed

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 ≤ x ∗ (y ∗ z)
begin

lemma mult-one-associative [simp]:
  x ∗ 1 ∗ y = x ∗ y
  by (metis dual-order.antisym mult-left-isotone mult-left-one
mult-semi-associative mult-sub-right-one)

lemma mult-sup-associative-one:
  (x ∗ (y ∗ 1)) ∗ z ≤ x ∗ (y ∗ z)
  by (metis mult-semi-associative mult-one-associative)

lemma rstar-increasing:
  assumes has-least-prefixpoint (Rf y)
    shows y ≤ rstar y
proof –
  have Rf y (rstar y) ≤ rstar y
    using assms pmu-unfold by blast
  thus ?thesis
    by (metis le-supE mult-right-isotone mult-sub-right-one sup.absorb-iff2)
qed

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*: $x * y \le z \longleftrightarrow x \le z \;/\; y$
begin

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

lemma *lres-left-isotone*:
  $x \le y \Longrightarrow x \;/\; z \le y \;/\; z$
  using *dual-order.trans lres-galois* **by** *blast*

lemma *lres-right-antitone*:
  $x \le y \Longrightarrow z \;/\; y \le z \;/\; x$
  using *dual-order.trans lres-galois mult-right-isotone* **by** *blast*

lemma *lres-inverse*:
  $(x \;/\; y) * y \le x$
  by (*simp add*: *lres-galois*)

lemma *lres-one*:
  $x \;/\; 1 \le x$
  using *mult-sub-right-one order-trans lres-inverse* **by** *blast*

lemma *lres-mult-sub-lres-lres*:
  $x \;/\; (z * y) \le (x \;/\; y) \;/\; z$
  using *lres-galois mult-semi-associative order.trans* **by** *blast*

lemma *mult-lres-sub-assoc*:
  $x * (y \;/\; z) \le (x * y) \;/\; z$
  by (*meson dual-order.trans lres-galois mult-right-isotone lres-inverse lres-mult-sub-lres-lres*)

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* $\le$ *rstar y*
proof –
  have $y * (\mathit{rstar}\ y \;/\; y) * y \le y * \mathit{rstar}\ y$

```
    using lres-galois mult-lres-sub-assoc by auto
  also have ... ≤ rstar y
    using assms(2) le-supE pmu-unfold by blast
  finally have y ∗ (rstar y / y) ≤ rstar y / y
    by (simp add: lres-galois)
  hence Rf y (rstar y / y) ≤ rstar y / y
    using assms(2) lres-galois rstar-increasing by fastforce
  hence rstar y ≤ rstar y / y
    using assms(2) is-least-prefixpoint-def least-prefixpoint by auto
  hence Lf y (rstar y) ≤ rstar y
    using assms(2) lres-galois pmu-unfold by fastforce
  thus ?thesis
    using assms(1) is-least-prefixpoint-def least-prefixpoint by auto
qed
```

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 −
  have Rf y (rstar y / rstar y) ∗ rstar y ≤ rstar y ⊔ y ∗ ((rstar y / rstar y) ∗
rstar y)
    using mult-right-dist-sup mult-semi-associative sup-right-isotone by auto
  also have ... ≤ rstar y ⊔ y ∗ rstar y
    using mult-right-isotone sup-right-isotone lres-inverse by blast
  also have ... ≤ rstar y
    using assms(1) pmu-unfold by fastforce
  finally have Rf y (rstar y / rstar y) ≤ rstar y / rstar y
    by (simp add: lres-galois)
  hence rstar y ∗ rstar y ≤ rstar y
    using assms(1) is-least-prefixpoint-def least-prefixpoint lres-galois by auto
  hence y ⊔ rstar y ∗ rstar y ≤ rstar y
    by (simp add: assms(1) rstar-increasing)
  hence Sf y (rstar y) ≤ rstar y
    using assms(1) pmu-unfold by force
  hence sstar y ≤ rstar y
    using assms(2) is-least-prefixpoint-def least-prefixpoint by auto
  thus ?thesis
    by (simp add: assms order.antisym rstar-below-sstar)
qed

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
  by unfold-locales (simp add: mult-assoc)

lemma zero-right-mult-decreasing:
  x ∗ bot ≤ x
  by (metis bot-least mult-1-right mult-right-isotone)
```

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 ≤ x ∗ p ⟷ p ∗ x = p ∗ x ∗ p
proof
  assume 1: p ∗ x ≤ x ∗ p
  have p ∗ x ≤ p ∗ p ∗ x
    by (simp add: assms(1) mult-left-isotone)
  also have ... ≤ p ∗ x ∗ p
    using 1 by (simp add: mult-right-isotone mult-assoc)
  finally show p ∗ x = p ∗ x ∗ p
    using assms(2) order.antisym mult-right-isotone by fastforce
next
  assume p ∗ x = p ∗ x ∗ p
  thus p ∗ x ≤ x ∗ p
    by (metis assms(2) mult-left-isotone mult-left-one)
qed

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 ⊔ z) = x ∗ y ⊔ x ∗ z
begin

lemma case-split-right:
  assumes 1 ≤ w ⊔ z
      and x ∗ w ≤ y
      and x ∗ z ≤ y
    shows x ≤ y
proof −
  have x ∗ (w ⊔ z) ≤ y
    by (simp add: assms(2−3) mult-left-dist-sup)
```

```
  thus ?thesis
    by (metis assms(1) dual-order.trans mult-1-right mult-right-isotone)
qed

lemma case-split-right-equal:
  w ⊔ z = 1 ⟹ x ∗ w = y ∗ w ⟹ x ∗ z = y ∗ z ⟹ x = y
  by (metis mult-1-right mult-left-dist-sup)
```

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
  apply unfold-locales
  using sup-right-isotone apply blast
  apply (simp add: mult-right-dist-sup)
  apply (simp add: mult-left-dist-sup)
  apply (simp add: mult-right-isotone)
  by (simp add: mult-left-isotone)

sublocale semiring: semiring-numeral 1 times sup ..

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 ∗ bot = bot
begin

sublocale semiring: semiring-0 sup bot times
  by unfold-locales simp-all

end
```

3.2 Bounded Idempotent Semirings

All of the following semirings have a greatest element in the underlying semilattice 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*]: $x \sqcup top = top$
begin

subclass *times-top* **.**

We first give basic properties of the greatest element.

lemma *sup-left-top* [*simp*]:
  $top \sqcup x = top$
  using *sup-right-top sup.commute* **by** *fastforce*

lemma *top-greatest* [*simp*]:
  $x \leq top$
  by (*simp add*: *le-iff-sup*)

lemma *top-left-mult-increasing*:
  $x \leq top * x$
  by (*metis mult-left-isotone mult-left-one top-greatest*)

lemma *top-right-mult-increasing*:
  $x \leq x * top$
  using *mult-right-isotone mult-sub-right-one order-trans top-greatest* **by** *blast*

lemma *top-mult-top* [*simp*]:
  $top * top = top$
  by (*simp add*: *order.antisym top-left-mult-increasing*)

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

lemma *vector-bot-closed*:
  vector bot
  by *simp*

lemma *vector-top-closed*:
  vector top
  by *simp*

lemma *vector-sup-closed*:
  vector $x \Longrightarrow$ *vector* $y \Longrightarrow$ *vector* $(x \sqcup y)$
  by (*simp add*: *mult-right-dist-sup*)

lemma *covector-top-closed*:
  covector top
  by *simp*

lemma *total-one-closed*:
total 1
by *simp*

lemma *total-top-closed*:
total top
by *simp*

lemma *total-sup-closed*:
total $x \Longrightarrow$ *total* $(x \sqcup y)$
by (*simp add*: *mult-right-dist-sup*)

lemma *surjective-one-closed*:
surjective 1
by (*simp add*: *order.antisym mult-sub-right-one*)

lemma *surjective-top-closed*:
surjective top
by *simp*

lemma *surjective-sup-closed*:
surjective $x \Longrightarrow$ *surjective* $(x \sqcup y)$
by (*metis le-iff-sup mult-left-sub-dist-sup-left sup-left-top*)

lemma *reflexive-top-closed*:
reflexive top
by *simp*

lemma *transitive-top-closed*:
transitive top
by *simp*

lemma *dense-top-closed*:
dense-rel top
by *simp*

lemma *idempotent-top-closed*:
idempotent top
by *simp*

lemma *preorder-top-closed*:
preorder top
by *simp*

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*:
$\mathit{vector}\ y \Longrightarrow \mathit{vector}\ (x * y)$
by (*metis order.antisym mult-semi-associative top-right-mult-increasing*)

lemma *surjective-mult-closed*:
$\mathit{surjective}\ x \Longrightarrow \mathit{surjective}\ y \Longrightarrow \mathit{surjective}\ (x * y)$
by (*metis order.antisym mult-semi-associative top-greatest*)

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*:
$x \ / \ \mathit{top} \le x$
using *lres-inverse order.trans top-right-mult-increasing* **by** *blast*

lemma *top-lres-absorb* [*simp*]:
$\mathit{top} \ / \ x = \mathit{top}$
using *order.antisym lres-galois top-greatest* **by** *blast*

lemma *covector-lres-closed*:
$\mathit{covector}\ x \Longrightarrow \mathit{covector}\ (x \ / \ y)$
by (*metis order.antisym mult-lres-sub-assoc top-left-mult-increasing*)

end

Some closure properties require full associativity.

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

lemma *covector-mult-closed*:
$\mathit{covector}\ x \Longrightarrow \mathit{covector}\ (x * y)$
by (*metis mult-assoc*)

lemma *total-mult-closed*:
$\mathit{total}\ x \Longrightarrow \mathit{total}\ y \Longrightarrow \mathit{total}\ (x * y)$
by (*simp add*: *mult-assoc*)

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 $x \Longrightarrow$ *covector* $y \Longrightarrow$ *covector* $(x \sqcup y)$
by (*simp add*: *mult-left-dist-sup*)

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
by *simp*

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* **.**
subclass *times-top* **.**

abbreviation *total-var* :: *'a* ⇒ *bool* **where** *total-var* x ≡ $1 \le x * x^T$
abbreviation *surjective-var* :: *'a* ⇒ *bool* **where** *surjective-var* x ≡ $1 \le x^T * x$
abbreviation *univalent* :: *'a* ⇒ *bool* **where** *univalent* x ≡ $x^T * x \le 1$
abbreviation *injective* :: *'a* ⇒ *bool* **where** *injective* x ≡ $x * x^T \le 1$

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

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

abbreviation *symmetric* :: *'a* ⇒ *bool* **where** *symmetric* x ≡ $x^T = x$
abbreviation *antisymmetric* :: *'a* ⇒ *bool* **where** *antisymmetric* x ≡ $x \sqcap x^T \le 1$
abbreviation *asymmetric* :: *'a* ⇒ *bool* **where** *asymmetric* x ≡ $x \sqcap x^T = bot$
abbreviation *linear* :: *'a* ⇒ *bool* **where** *linear* x ≡ $x \sqcup x^T = top$

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* : $(x \sqcup y) * z = (x * z) \sqcup (y * z)$
  assumes *comp-left-zero* [*simp*]: $bot * x = bot$
  assumes *comp-left-one* [*simp*]: $1 * x = x$
  assumes *conv-involutive* [*simp*]: $x^{T T} = x$
  assumes *conv-dist-sup* : $(x \sqcup y)^T = x^T \sqcup y^T$
  assumes *conv-dist-comp* : $(x * y)^T = y^T * x^T$
  assumes *dedekind-1* : $x * y \sqcap z \leq x * (y \sqcap (x^T * z))$
begin

subclass *bounded-distrib-allegory-signature* **.**

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

lemma *conv-isotone*:
  $x \leq y \Longrightarrow x^T \leq y^T$
  by (*metis conv-dist-sup le-iff-sup*)

lemma *conv-order*:
  $x \leq y \longleftrightarrow x^T \leq y^T$
  using *conv-isotone* **by** *fastforce*

lemma *conv-bot* [*simp*]:
  $bot^T = bot$
  using *conv-order bot-unique* **by** *force*

lemma *conv-top* [*simp*]:
  $top^T = top$
  by (*metis conv-involutive conv-order order.eq-iff top-greatest*)

lemma *conv-dist-inf*:
  $(x \sqcap y)^T = x^T \sqcap y^T$
  apply (*rule order.antisym*)
  using *conv-order* **apply** *simp*
  by (*metis conv-order conv-involutive inf.boundedI inf.cobounded1 inf.cobounded2*)

lemma *conv-inf-bot-iff*:
  $bot = x^T \sqcap y \longleftrightarrow bot = x \sqcap y^T$
  using *conv-dist-inf conv-bot* **by** *fastforce*

lemma *conv-one* [*simp*]:
$\mathit{1}^T = \mathit{1}$
by (*metis comp-left-one conv-dist-comp conv-involutive*)

lemma *comp-left-dist-sup*:
$(x * y) \sqcup (x * z) = x * (y \sqcup z)$
by (*metis comp-right-dist-sup conv-involutive conv-dist-sup conv-dist-comp*)

lemma *comp-right-isotone*:
$x \leq y \Longrightarrow z * x \leq z * y$
by (*simp add*: *comp-left-dist-sup sup.absorb-iff1*)

lemma *comp-left-isotone*:
$x \leq y \Longrightarrow x * z \leq y * z$
by (*metis comp-right-dist-sup le-iff-sup*)

lemma *comp-isotone*:
$x \leq y \Longrightarrow w \leq z \Longrightarrow x * w \leq y * z$
using *comp-left-isotone comp-right-isotone order.trans* **by** *blast*

lemma *comp-left-subdist-inf*:
$(x \sqcap y) * z \leq x * z \sqcap y * z$
by (*simp add*: *comp-left-isotone*)

lemma *comp-left-increasing-sup*:
$x * y \leq (x \sqcup z) * y$
by (*simp add*: *comp-left-isotone*)

lemma *comp-right-subdist-inf*:
$x * (y \sqcap z) \leq x * y \sqcap x * z$
by (*simp add*: *comp-right-isotone*)

lemma *comp-right-increasing-sup*:
$x * y \leq x * (y \sqcup z)$
by (*simp add*: *comp-right-isotone*)

lemma *comp-right-zero* [*simp*]:
$x * \mathit{bot} = \mathit{bot}$
by (*metis comp-left-zero conv-dist-comp conv-involutive*)

lemma *comp-right-one* [*simp*]:
$x * \mathit{1} = x$
by (*metis comp-left-one conv-dist-comp conv-involutive*)

lemma *comp-left-conjugate*:
conjugate $(\lambda y\ .\ x * y)\ (\lambda y\ .\ x^T * y)$
apply (*unfold conjugate-def*, *intro allI*)
by (*metis comp-right-zero bot.extremum-unique conv-involutive dedekind-1*

inf.commute)

lemma *comp-right-conjugate*:
conjugate $(\lambda y\ .\ y * x)\ (\lambda y\ .\ y * x^T)$
apply (*unfold conjugate-def*, *intro allI*)
by (*metis comp-left-conjugate*[*unfolded conjugate-def*] *conv-inf-bot-iff conv-dist-comp conv-involutive*)

We still obtain a semiring structure.

subclass *bounded-idempotent-semiring*
by (*unfold-locales*)
(*auto simp*: *comp-right-isotone comp-right-dist-sup comp-associative comp-left-dist-sup*)

sublocale *inf*: *semiring-0 sup bot inf*
by (*unfold-locales*, *auto simp*: *inf-sup-distrib2 inf-sup-distrib1 inf-assoc*)

lemma *schroeder-1*:
$x * y \sqcap z = \mathit{bot} \longleftrightarrow x^T * z \sqcap y = \mathit{bot}$
using *abel-semigroup.commute comp-left-conjugate conjugate-def inf.abel-semigroup-axioms* **by** *fastforce*

lemma *schroeder-2*:
$x * y \sqcap z = \mathit{bot} \longleftrightarrow z * y^T \sqcap x = \mathit{bot}$
by (*metis comp-right-conjugate conjugate-def inf-commute*)

lemma *comp-additive*:
additive $(\lambda y\ .\ x * y) \wedge$ *additive* $(\lambda y\ .\ x^T * y) \wedge$ *additive* $(\lambda y\ .\ y * x) \wedge$ *additive* $(\lambda y\ .\ y * x^T)$
by (*simp add*: *comp-left-dist-sup additive-def comp-right-dist-sup*)

lemma *dedekind-2*:
$y * x \sqcap z \le (y \sqcap (z * x^T)) * x$
by (*metis conv-dist-inf conv-order conv-dist-comp dedekind-1*)

The intersection with a vector can still be exported from the first argument of a composition, and many other properties of vectors and covectors continue to hold.

lemma *vector-inf-comp*:
vector $x \Longrightarrow (x \sqcap y) * z = x \sqcap (y * z)$
apply (*rule order.antisym*)
apply (*metis comp-left-subdist-inf comp-right-isotone inf.sup-left-isotone order-lesseq-imp top-greatest*)
by (*metis comp-left-isotone comp-right-isotone dedekind-2 inf-commute inf-mono order-refl order-trans top-greatest*)

lemma *vector-inf-closed*:
vector $x \Longrightarrow$ *vector* $y \Longrightarrow$ *vector* $(x \sqcap y)$
by (*simp add*: *vector-inf-comp*)

lemma *vector-inf-one-comp*:
$\mathit{vector}\ x \Longrightarrow (x \sqcap \mathit{1}) * y = x \sqcap y$
by (*simp add*: *vector-inf-comp*)

lemma *covector-inf-comp-1*:
assumes *vector x*
shows $(y \sqcap x^T) * z = (y \sqcap x^T) * (x \sqcap z)$
proof –
have $(y \sqcap x^T) * z \leq (y \sqcap x^T) * (z \sqcap ((y^T \sqcap x) * \mathit{top}))$
by (*metis inf-top-right dedekind-1 conv-dist-inf conv-involutive*)
also have ... $\leq (y \sqcap x^T) * (x \sqcap z)$
by (*metis assms comp-left-isotone comp-right-isotone inf-le2 inf-mono order-refl inf-commute*)
finally show *?thesis*
by (*simp add*: *comp-right-isotone order.antisym*)
qed

lemma *covector-inf-comp-2*:
assumes *vector x*
shows $y * (x \sqcap z) = (y \sqcap x^T) * (x \sqcap z)$
proof –
have $y * (x \sqcap z) \leq (y \sqcap (\mathit{top} * (x \sqcap z)^T)) * (x \sqcap z)$
by (*metis dedekind-2 inf-top-right*)
also have ... $\leq (y \sqcap x^T) * (x \sqcap z)$
by (*metis assms comp-left-isotone conv-dist-comp conv-order conv-top eq-refl inf-le1 inf-mono*)
finally show *?thesis*
using *comp-left-subdist-inf order.antisym* **by** *auto*
qed

lemma *covector-inf-comp-3*:
$\mathit{vector}\ x \Longrightarrow (y \sqcap x^T) * z = y * (x \sqcap z)$
by (*metis covector-inf-comp-1 covector-inf-comp-2*)

lemma *covector-inf-closed*:
$\mathit{covector}\ x \Longrightarrow \mathit{covector}\ y \Longrightarrow \mathit{covector}\ (x \sqcap y)$
by (*metis comp-right-subdist-inf order.antisym top-left-mult-increasing*)

lemma *vector-conv-covector*:
$\mathit{vector}\ v \longleftrightarrow \mathit{covector}\ (v^T)$
by (*metis conv-dist-comp conv-involutive conv-top*)

lemma *covector-conv-vector*:
$\mathit{covector}\ v \longleftrightarrow \mathit{vector}\ (v^T)$
by (*simp add*: *vector-conv-covector*)

lemma *covector-comp-inf*:
$\mathit{covector}\ z \Longrightarrow x * (y \sqcap z) = x * y \sqcap z$

apply (*rule order.antisym*)
apply (*metis comp-isotone comp-right-subdist-inf inf.boundedE inf.boundedI inf.cobounded2 top.extremum*)
by (*metis comp-left-isotone comp-right-isotone dedekind-1 inf-commute inf-mono order-refl order-trans top-greatest*)

lemma *vector-restrict-comp-conv*:
vector $x \Longrightarrow x \sqcap y \leq x^T * y$
by (*metis covector-inf-comp-3 eq-refl inf.sup-monoid.add-commute inf-top-right le-supE sup.orderE top-left-mult-increasing*)

lemma *covector-restrict-comp-conv*:
covector $x \Longrightarrow y \sqcap x \leq y * x^T$
by (*metis conv-dist-comp conv-dist-inf conv-order conv-top inf.sup-monoid.add-commute vector-restrict-comp-conv*)

lemma *covector-comp-inf-1*:
covector $x \Longrightarrow (y \sqcap x) * z = y * (x^T \sqcap z)$
using *covector-conv-vector covector-inf-comp-3* **by** *fastforce*

We still have two ways to represent surjectivity and totality.

lemma *surjective-var*:
surjective $x \longleftrightarrow$ *surjective-var* x
proof
assume *surjective* x
thus *surjective-var* x
by (*metis dedekind-2 comp-left-one inf-absorb2 top-greatest*)
next
assume *surjective-var* x
hence $x^T * (x * top) = top$
by (*metis comp-left-isotone comp-associative comp-left-one top-le*)
thus *surjective* x
by (*metis comp-right-isotone conv-top conv-dist-comp conv-involutive top-greatest top-le*)
qed

lemma *total-var*:
total $x \longleftrightarrow$ *total-var* x
by (*metis conv-top conv-dist-comp conv-involutive surjective-var*)

lemma *surjective-conv-total*:
surjective $x \longleftrightarrow$ *total* (x^T)
by (*metis conv-top conv-dist-comp conv-involutive*)

lemma *total-conv-surjective*:
total $x \longleftrightarrow$ *surjective* (x^T)
by (*simp add*: *surjective-conv-total*)

lemma *injective-conv-univalent*:

injective x $\longleftrightarrow$ *univalent* (x^T)
by *simp*

lemma *univalent-conv-injective*:
univalent x $\longleftrightarrow$ *injective* (x^T)
by *simp*

We continue with studying further closure properties.

lemma *univalent-bot-closed*:
univalent bot
by *simp*

lemma *univalent-one-closed*:
univalent 1
by *simp*

lemma *univalent-inf-closed*:
univalent x $\Longrightarrow$ *univalent* $(x \sqcap y)$
by (*metis comp-left-subdist-inf comp-right-subdist-inf conv-dist-inf inf.cobounded1 order-lesseq-imp*)

lemma *univalent-mult-closed*:
assumes *univalent x*
and *univalent y*
shows *univalent* $(x * y)$
proof –
have $(x * y)^T * x \leq y^T$
by (*metis assms*(*1*) *comp-left-isotone comp-right-one conv-one conv-order comp-associative conv-dist-comp conv-involutive*)
thus *?thesis*
by (*metis assms*(*2*) *comp-left-isotone comp-associative dual-order.trans*)
qed

lemma *injective-bot-closed*:
injective bot
by *simp*

lemma *injective-one-closed*:
injective 1
by *simp*

lemma *injective-inf-closed*:
injective x $\Longrightarrow$ *injective* $(x \sqcap y)$
by (*metis conv-dist-inf injective-conv-univalent univalent-inf-closed*)

lemma *injective-mult-closed*:
injective x $\Longrightarrow$ *injective y* $\Longrightarrow$ *injective* $(x * y)$
by (*metis injective-conv-univalent conv-dist-comp univalent-mult-closed*)

lemma *mapping-one-closed*:
mapping 1
by *simp*

lemma *mapping-mult-closed*:
mapping x ⟹ *mapping y* ⟹ *mapping* (*x* ∗ *y*)
by (*simp add*: *comp-associative univalent-mult-closed*)

lemma *bijective-one-closed*:
bijective 1
by *simp*

lemma *bijective-mult-closed*:
bijective x ⟹ *bijective y* ⟹ *bijective* (*x* ∗ *y*)
by (*metis injective-mult-closed comp-associative*)

lemma *bijective-conv-mapping*:
bijective x ⟷ *mapping* (x^T)
by (*simp add*: *surjective-conv-total*)

lemma *mapping-conv-bijective*:
mapping x ⟷ *bijective* (x^T)
by (*simp add*: *total-conv-surjective*)

lemma *reflexive-inf-closed*:
reflexive x ⟹ *reflexive y* ⟹ *reflexive* (*x* ⊓ *y*)
by *simp*

lemma *reflexive-conv-closed*:
reflexive x ⟹ *reflexive* (x^T)
using *conv-isotone* **by** *force*

lemma *coreflexive-inf-closed*:
coreflexive x ⟹ *coreflexive* (*x* ⊓ *y*)
by (*simp add*: *le-infI1*)

lemma *coreflexive-conv-closed*:
coreflexive x ⟹ *coreflexive* (x^T)
using *conv-order* **by** *force*

lemma *coreflexive-symmetric*:
coreflexive x ⟹ *symmetric x*
by (*metis comp-right-one comp-right-subdist-inf conv-dist-inf conv-dist-comp conv-involutive dedekind-1 inf.absorb1 inf-absorb2*)

lemma *transitive-inf-closed*:
transitive x ⟹ *transitive y* ⟹ *transitive* (*x* ⊓ *y*)
by (*meson comp-left-subdist-inf inf.cobounded1 inf.sup-mono inf-le2 mult-right-isotone order.trans*)

lemma *transitive-conv-closed*:
transitive $x \Longrightarrow$ *transitive* (x^T)
using *conv-order conv-dist-comp* **by** *fastforce*

lemma *dense-conv-closed*:
dense-rel $x \Longrightarrow$ *dense-rel* (x^T)
using *conv-order conv-dist-comp* **by** *fastforce*

lemma *idempotent-conv-closed*:
idempotent $x \Longrightarrow$ *idempotent* (x^T)
by (*metis conv-dist-comp*)

lemma *preorder-inf-closed*:
preorder $x \Longrightarrow$ *preorder* $y \Longrightarrow$ *preorder* $(x \sqcap y)$
using *transitive-inf-closed* **by** *auto*

lemma *preorder-conv-closed*:
preorder $x \Longrightarrow$ *preorder* (x^T)
by (*simp add*: *reflexive-conv-closed transitive-conv-closed*)

lemma *symmetric-bot-closed*:
symmetric bot
by *simp*

lemma *symmetric-one-closed*:
symmetric 1
by *simp*

lemma *symmetric-top-closed*:
symmetric top
by *simp*

lemma *symmetric-inf-closed*:
symmetric $x \Longrightarrow$ *symmetric* $y \Longrightarrow$ *symmetric* $(x \sqcap y)$
by (*simp add*: *conv-dist-inf*)

lemma *symmetric-sup-closed*:
symmetric $x \Longrightarrow$ *symmetric* $y \Longrightarrow$ *symmetric* $(x \sqcup y)$
by (*simp add*: *conv-dist-sup*)

lemma *symmetric-conv-closed*:
symmetric $x \Longrightarrow$ *symmetric* (x^T)
by *simp*

lemma *one-inf-conv*:
$1 \sqcap x = 1 \sqcap x^T$
by (*metis conv-dist-inf coreflexive-symmetric inf.cobounded1 symmetric-one-closed*)

lemma *antisymmetric-bot-closed*:
  antisymmetric bot
  by *simp*

lemma *antisymmetric-one-closed*:
  antisymmetric 1
  by *simp*

lemma *antisymmetric-inf-closed*:
  antisymmetric x $\Longrightarrow$ *antisymmetric* $(x \sqcap y)$
  by (*rule order-trans*[**where** $y{=}x \sqcap x^T$]) (*simp-all add*: *conv-isotone inf.coboundedI2 inf.sup-assoc*)

lemma *antisymmetric-conv-closed*:
  antisymmetric x $\Longrightarrow$ *antisymmetric* (x^T)
  by (*simp add*: *inf-commute*)

lemma *asymmetric-bot-closed*:
  asymmetric bot
  by *simp*

lemma *asymmetric-inf-closed*:
  asymmetric x $\Longrightarrow$ *asymmetric* $(x \sqcap y)$
  by (*metis conv-dist-inf inf.mult-zero-left inf.left-commute inf-assoc*)

lemma *asymmetric-conv-closed*:
  asymmetric x $\Longrightarrow$ *asymmetric* (x^T)
  by (*simp add*: *inf-commute*)

lemma *linear-top-closed*:
  linear top
  by *simp*

lemma *linear-sup-closed*:
  linear x $\Longrightarrow$ *linear* $(x \sqcup y)$
  by (*metis conv-dist-sup sup-assoc sup-commute sup-top-right*)

lemma *linear-reflexive*:
  linear x $\Longrightarrow$ *reflexive x*
  by (*metis one-inf-conv inf.distrib-left inf.cobounded2 inf.orderE reflexive-top-closed sup.idem*)

lemma *linear-conv-closed*:
  linear x $\Longrightarrow$ *linear* (x^T)
  by (*simp add*: *sup-commute*)

lemma *linear-comp-closed*:
  assumes *linear x*

and *linear* y
shows *linear* $(x * y)$
proof −
have *reflexive* y
by (*simp add*: *assms*(*2*) *linear-reflexive*)
hence $x \sqcup x^T \leq x * y \sqcup y^T * x^T$
by (*metis case-split-left case-split-right le-supI sup.coboundled1 sup.coboundled2 sup.idem reflexive-conv-closed*)
thus *?thesis*
by (*simp add*: *assms*(*1*) *conv-dist-comp top-le*)
qed

lemma *equivalence-one-closed*:
equivalence 1
by *simp*

lemma *equivalence-top-closed*:
equivalence top
by *simp*

lemma *equivalence-inf-closed*:
equivalence $x \Longrightarrow$ *equivalence* $y \Longrightarrow$ *equivalence* $(x \sqcap y)$
using *conv-dist-inf preorder-inf-closed* **by** *auto*

lemma *equivalence-conv-closed*:
equivalence $x \Longrightarrow$ *equivalence* (x^T)
by *simp*

lemma *order-one-closed*:
order 1
by *simp*

lemma *order-inf-closed*:
order $x \Longrightarrow$ *order* $y \Longrightarrow$ *order* $(x \sqcap y)$
using *antisymmetric-inf-closed transitive-inf-closed* **by** *auto*

lemma *order-conv-closed*:
order $x \Longrightarrow$ *order* (x^T)
by (*simp add*: *inf-commute reflexive-conv-closed transitive-conv-closed*)

lemma *linear-order-conv-closed*:
linear-order $x \Longrightarrow$ *linear-order* (x^T)
using *equivalence-top-closed conv-dist-sup inf-commute reflexive-conv-closed transitive-conv-closed* **by** *force*

We show a fact about equivalences.

lemma *equivalence-comp-dist-inf*:
equivalence $x \Longrightarrow x * y \sqcap x * z = x * (y \sqcap x * z)$
by (*metis order.antisym comp-right-subdist-inf dedekind-1 order.eq-iff*

inf.absorb1 inf.absorb2 mult-1-right mult-assoc)

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*:
  $\mathit{coreflexive}\ x \Longrightarrow x * \mathit{top} \sqcap y = x * y$
  apply (*rule order.antisym*)
  apply (*metis comp-left-isotone comp-left-one coreflexive-symmetric dedekind-1 inf-top-left order-trans*)
  using *comp-left-isotone comp-right-isotone* **by** *fastforce*

lemma *coreflexive-comp-top-inf-one*:
  $\mathit{coreflexive}\ x \Longrightarrow x * \mathit{top} \sqcap 1 = x$
  by (*simp add*: *coreflexive-comp-top-inf*)

lemma *coreflexive-comp-inf*:
  $\mathit{coreflexive}\ x \Longrightarrow \mathit{coreflexive}\ y \Longrightarrow x * y = x \sqcap y$
  by (*metis* (*full-types*) *coreflexive-comp-top-inf coreflexive-comp-top-inf-one inf.mult-assoc inf.absorb2*)

lemma *coreflexive-comp-inf-comp*:
  assumes *coreflexive* x
    and *coreflexive* y
   shows $(x * z) \sqcap (y * z) = (x \sqcap y) * z$
proof –
  have $(x * z) \sqcap (y * z) = x * \mathit{top} \sqcap z \sqcap y * \mathit{top} \sqcap z$
   using *assms coreflexive-comp-top-inf inf-assoc* **by** *auto*
  also have ... $= x * \mathit{top} \sqcap y * \mathit{top} \sqcap z$
   by (*simp add*: *inf.commute inf.left-commute*)
  also have ... $= (x \sqcap y) * \mathit{top} \sqcap z$
   by (*metis assms coreflexive-comp-inf coreflexive-comp-top-inf mult-assoc*)
  also have ... $= (x \sqcap y) * z$
   by (*simp add*: *assms*(*1*) *coreflexive-comp-top-inf coreflexive-inf-closed*)
  finally show *?thesis*
   .
qed

lemma *test-comp-test-inf*:
  $(x \sqcap 1) * y * (z \sqcap 1) = (x \sqcap 1) * y \sqcap y * (z \sqcap 1)$
  by (*smt comp-right-one comp-right-subdist-inf coreflexive-comp-top-inf inf.left-commute inf.orderE inf-le2 mult-assoc*)

lemma *test-comp-test-top*:
  $y \sqcap (x \sqcap 1) * \mathit{top} * (z \sqcap 1) = (x \sqcap 1) * y * (z \sqcap 1)$
proof –
  have $\forall\, u\ v\ .\ (v \sqcap u^T)^T = v^T \sqcap u$
   using *conv-dist-inf* **by** *auto*
  thus *?thesis*

by (*smt conv-dist-comp conv-involutive coreflexive-comp-top-inf inf.cobounded2 inf.left-commute inf.sup-monoid.add-commute symmetric-one-closed mult-assoc symmetric-top-closed*)
qed

lemma *coreflexive-idempotent*:
coreflexive $x \Longrightarrow$ *idempotent* x
by (*simp add*: *coreflexive-comp-inf*)

lemma *coreflexive-univalent*:
coreflexive $x \Longrightarrow$ *univalent* x
by (*simp add*: *coreflexive-idempotent coreflexive-symmetric*)

lemma *coreflexive-injective*:
coreflexive $x \Longrightarrow$ *injective* x
by (*simp add*: *coreflexive-idempotent coreflexive-symmetric*)

lemma *coreflexive-commutative*:
coreflexive $x \Longrightarrow$ *coreflexive* $y \Longrightarrow x * y = y * x$
by (*simp add*: *coreflexive-comp-inf inf.commute*)

lemma *coreflexive-dedekind*:
coreflexive $x \Longrightarrow$ *coreflexive* $y \Longrightarrow$ *coreflexive* $z \Longrightarrow x * y \sqcap z \le x * (y \sqcap x * z)$
by (*simp add*: *coreflexive-comp-inf inf.coboundedI1 inf.left-commute*)

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 (*rule order.antisym*)
have $x * y \sqcap z \le x * (y \sqcap (x^T * z)) \sqcap z$
by (*simp add*: *dedekind-1*)
also have ... $\le (x \sqcap (z * (y \sqcap (x^T * z))^T)) * (y \sqcap (x^T * z)) \sqcap z$
by (*simp add*: *dedekind-2*)
also have ... $\le (x \sqcap (z * y^T)) * (y \sqcap (x^T * z)) \sqcap z$
by (*metis comp-left-isotone comp-right-isotone inf-mono conv-order inf.cobounded1 order-refl*)
finally show $x * y \sqcap z \le (x \sqcap (z * y^T)) * (y \sqcap (x^T * z)) \sqcap z$
.
next
show $(x \sqcap (z * y^T)) * (y \sqcap (x^T * z)) \sqcap z \le x * y \sqcap z$
using *comp-isotone inf.sup-left-isotone* **by** *auto*
qed

lemma *dedekind*:
$x * y \sqcap z \le (x \sqcap (z * y^T)) * (y \sqcap (x^T * z))$
by (*metis dedekind-eq inf.cobounded1*)

lemma *vector-export-comp*:
$(x * top \sqcap y) * z = x * top \sqcap y * z$

proof −
have *vector* $(x * top)$
by (*simp add*: *comp-associative*)
thus *?thesis*
by (*simp add*: *vector-inf-comp*)
qed

lemma *vector-export-comp-unit*:
$(x * top \sqcap 1) * y = x * top \sqcap y$
by (*simp add*: *vector-export-comp*)

We solve a few exercises from [27].

lemma *ex231a* [*simp*]:
$(1 \sqcap x * x^T) * x = x$
by (*metis inf.coboundedl inf.idem inf-right-idem comp-left-one conv-one coreflexive-comp-top-inf dedekind-eq*)

lemma *ex231b* [*simp*]:
$x * (1 \sqcap x^T * x) = x$
by (*metis conv-dist-comp conv-dist-inf conv-involutive conv-one ex231a*)

lemma *ex231c*:
$x \leq x * x^T * x$
by (*metis comp-left-isotone ex231a inf-le2*)

lemma *ex231d*:
$x \leq x * top * x$
by (*metis comp-left-isotone comp-right-isotone top-greatest order-trans ex231c*)

lemma *ex231e* [*simp*]:
$x * top * x * top = x * top$
by (*metis ex231d order.antisym comp-associative mult-right-isotone top.extremum*)

lemma *arc-injective*:
arc $x \Longrightarrow$ *injective* x
by (*metis conv-dist-inf conv-involutive inf.absorb2 top-right-mult-increasing univalent-inf-closed*)

lemma *arc-conv-closed*:
arc $x \Longrightarrow$ *arc* (x^T)
by *simp*

lemma *arc-univalent*:
arc $x \Longrightarrow$ *univalent* x
using *arc-conv-closed arc-injective univalent-conv-injective* **by** *blast*

lemma *injective-codomain*:
assumes *injective* x

shows $x * (x \sqcap 1) = x \sqcap 1$
proof (*rule order.antisym*)
show $x * (x \sqcap 1) \leq x \sqcap 1$
by (*metis assms comp-right-one dual-order.trans inf.boundedI inf.cobounded1 inf.sup-monoid.add-commute mult-right-isotone one-inf-conv*)
next
show $x \sqcap 1 \leq x * (x \sqcap 1)$
by (*metis coreflexive-idempotent inf.cobounded1 inf.cobounded2 mult-left-isotone*)
qed

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* $(t \sqcup e)$
proof −
have $(t \sqcup e) * (t \sqcup e)^T = t * t^T \sqcup t * e^T \sqcup e * t^T \sqcup e * e^T$
by (*simp add*: *comp-left-dist-sup conv-dist-sup semiring.distrib-right sup.assoc*)
thus *?thesis*
using *assms coreflexive-symmetric conv-dist-comp* **by** *fastforce*
qed

lemma *injective-inv*:
injective $t \Longrightarrow e * t^T = bot \Longrightarrow$ *arc* $e \Longrightarrow$ *injective* $(t \sqcup e)$
using *arc-injective injective-sup bot-least* **by** *blast*

lemma *univalent-sup*:
univalent $t \Longrightarrow e^T * t \leq 1 \Longrightarrow$ *univalent* $e \Longrightarrow$ *univalent* $(t \sqcup e)$
by (*metis injective-sup conv-dist-sup conv-involutive*)

lemma *point-injective*:
arc $x \Longrightarrow x^T * top * x \leq 1$
by (*metis conv-top comp-associative conv-dist-comp conv-involutive vector-top-closed*)

lemma *vv-transitive*:
vector $v \Longrightarrow (v * v^T) * (v * v^T) \leq v * v^T$
by (*metis comp-associative comp-left-isotone comp-right-isotone top-greatest*)

lemma *epm-3*:
assumes $e \leq w$
and *injective* w
shows $e = w \sqcap top * e$
proof −
have $w \sqcap top * e \leq w * e^T * e$
by (*metis* (*no-types*, *lifting*) *inf.absorb2 top.extremum dedekind-2*

inf.commute)
also have ... $\leq w * w^T * e$
by (*simp add*: *assms*(*1*) *conv-isotone mult-left-isotone mult-right-isotone*)
also have ... $\leq e$
using *assms*(*2*) *coreflexive-comp-top-inf inf.sup-right-divisibility* **by** *blast*
finally show *?thesis*
by (*simp add*: *assms*(*1*) *top-left-mult-increasing order.antisym*)
qed

lemma *comp-inf-vector*:
$x * (y \sqcap z * top) = (x \sqcap top * z^T) * y$
by (*metis conv-top covector-inf-comp-3 comp-associative conv-dist-comp inf.commute vector-top-closed*)

lemma *inf-vector-comp*:
$(x \sqcap y * top) * z = y * top \sqcap x * z$
using *inf.commute vector-export-comp* **by** *auto*

lemma *comp-inf-covector*:
$x * (y \sqcap top * z) = x * y \sqcap top * z$
by (*simp add*: *covector-comp-inf covector-mult-closed*)

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

lemma *univalent-comp-left-dist-inf*:
assumes *univalent* x
shows $x * (y \sqcap z) = x * y \sqcap x * z$
proof (*rule order.antisym*)
show $x * (y \sqcap z) \leq x * y \sqcap x * z$
by (*simp add*: *comp-right-isotone*)
next
have $x * y \sqcap x * z \leq (x \sqcap x * z * y^T) * (y \sqcap x^T * x * z)$
by (*metis comp-associative dedekind*)
also have ... $\leq x * (y \sqcap x^T * x * z)$
by (*simp add*: *comp-left-isotone*)
also have ... $\leq x * (y \sqcap 1 * z)$
using *assms comp-left-isotone comp-right-isotone inf.sup-right-isotone* **by** *blast*
finally show $x * y \sqcap x * z \leq x * (y \sqcap z)$
by *simp*
qed

lemma *injective-comp-right-dist-inf*:
injective $z \Longrightarrow (x \sqcap y) * z = x * z \sqcap y * z$
by (*metis univalent-comp-left-dist-inf conv-dist-comp conv-involutive conv-dist-inf*)

lemma *vector-covector*:
vector $v \Longrightarrow$ *vector* $w \Longrightarrow v \sqcap w^T = v * w^T$

```
  by (metis covector-comp-inf inf-top-left vector-conv-covector)

lemma comp-inf-vector-1:
  (x ⊓ top ∗ y) ∗ z = x ∗ (z ⊓ (top ∗ y)^T)
  by (simp add: comp-inf-vector conv-dist-comp)
```

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

```
lemma shunt-bijective:
  assumes bijective z
    shows x ≤ y ∗ z ⟷ x ∗ z^T ≤ y
proof
  assume x ≤ y ∗ z
  hence x ∗ z^T ≤ y ∗ z ∗ z^T
    by (simp add: mult-left-isotone)
  also have ... ≤ y
    using assms comp-associative mult-right-isotone by fastforce
  finally show x ∗ z^T ≤ y
    .
next
  assume 1: x ∗ z^T ≤ y
  have x = x ⊓ top ∗ z
    by (simp add: assms)
  also have ... ≤ x ∗ z^T ∗ z
    by (metis dedekind-2 inf-commute inf-top.right-neutral)
  also have ... ≤ y ∗ z
    using 1 by (simp add: mult-left-isotone)
  finally show x ≤ y ∗ z
    .
qed

lemma shunt-mapping:
  mapping z ⟹ x ≤ z ∗ y ⟷ z^T ∗ x ≤ y
  by (metis shunt-bijective mapping-conv-bijective conv-order conv-dist-comp
conv-involutive)

lemma bijective-reverse:
  assumes bijective p
      and bijective q
    shows p ≤ r ∗ q ⟷ q ≤ r^T ∗ p
proof –
  have p ≤ r ∗ q ⟷ p ∗ q^T ≤ r
    by (simp add: assms(2) shunt-bijective)
  also have ... ⟷ q^T ≤ p^T ∗ r
    by (metis assms(1) conv-dist-comp conv-involutive conv-order shunt-bijective)
  also have ... ⟷ q ≤ r^T ∗ p
    using conv-dist-comp conv-isotone by fastforce
  finally show ?thesis
    by simp
```

qed

lemma *arc-expanded*:
$arc\ x \longleftrightarrow x * top * x^T \leq 1 \wedge x^T * top * x \leq 1 \wedge top * x * top = top$
by (*metis conv-top comp-associative conv-dist-comp conv-involutive vector-top-closed*)

lemma *arc-top-arc*:
assumes *arc x*
shows $x * top * x = x$
by (*metis assms epm-3 top-right-mult-increasing vector-inf-comp vector-mult-closed vector-top-closed*)

lemma *arc-top-edge*:
assumes *arc x*
shows $x^T * top * x = x^T * x$
proof –
have $x^T = x^T * top \sqcap top * x^T$
using *assms epm-3 top-right-mult-increasing* **by** *simp*
thus *?thesis*
by (*metis comp-inf-vector-1 conv-dist-comp conv-involutive conv-top inf.absorb1 top-right-mult-increasing*)
qed

Lemmas *arc-eq-1* and *arc-eq-2* were contributed by Nicolas Robinson-O'Brien.

lemma *arc-eq-1*:
assumes *arc x*
shows $x = x * x^T * x$
proof –
have $x * x^T * x \leq x * top * x$
by (*simp add*: *mult-left-isotone mult-right-isotone*)
also have ... $\leq x$
by (*simp add*: *assms arc-top-arc*)
finally have $x * x^T * x \leq x$
by *simp*
thus *?thesis*
by (*simp add*: *order.antisym ex231c*)
qed

lemma *arc-eq-2*:
assumes *arc x*
shows $x^T = x^T * x * x^T$
using *arc-eq-1 assms conv-involutive* **by** *fastforce*

lemma *points-arc*:
$point\ x \Longrightarrow point\ y \Longrightarrow arc\ (x * y^T)$
by (*metis comp-associative conv-dist-comp conv-involutive equivalence-top-closed*)

lemma *point-arc*:
point x $\Longrightarrow$ *arc* $(x * x^T)$
by (*simp add*: *points-arc*)

lemma *arc-expanded-1*:
arc e $\Longrightarrow$ $e * x * e^T \leq 1$
by (*meson arc-expanded order-trans top-greatest mult-left-isotone mult-right-isotone*)

lemma *arc-expanded-2*:
arc e $\Longrightarrow$ $e^T * x * e \leq 1$
by (*meson arc-expanded order-trans top-greatest mult-left-isotone mult-right-isotone*)

lemma *point-conv-comp*:
point x $\Longrightarrow$ $x^T * x = top$
using *order-eq-iff shunt-bijective top-greatest vector-conv-covector* **by** *blast*

lemma *point-antisymmetric*:
point x $\Longrightarrow$ *antisymmetric x*
by (*simp add*: *vector-covector*)

lemma *mapping-inf-point-arc*:
assumes *mapping x*
and *point y*
shows *arc* $(x \sqcap y)$
proof (*unfold arc-expanded*, *intro conjI*)
show $(x \sqcap y) * top * (x \sqcap y)^T \leq 1$
by (*metis assms conv-dist-comp covector-conv-vector inf.orderE inf.sup-monoid.add-commute surjective-conv-total top.extremum top-right-mult-increasing vector-export-comp*)
have $(x \sqcap y)^T * top * (x \sqcap y) = x^T * y * (x \sqcap y)$
by (*simp add*: *assms*(*2*) *conv-dist-inf covector-inf-comp-3*)
also have ... $= x^T * (y \sqcap y^T) * x$
by (*simp add*: *assms*(*2*) *comp-associative covector-inf-comp-3 inf.sup-monoid.add-commute*)
also have ... $\leq x^T * x$
by (*metis assms*(*2*) *comp-right-one mult-left-isotone mult-right-isotone vector-covector*)
also have ... ≤ 1
by (*simp add*: *assms*(*1*))
finally show $(x \sqcap y)^T * top * (x \sqcap y) \leq 1$
.
show $top * (x \sqcap y) * top = top$
by (*metis assms inf-top-right inf-vector-comp mult-assoc*)
qed

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* :: $'a \Rightarrow$ *bool* **where** *irreflexive x* $\equiv x \leq -1$
abbreviation *strict-linear* :: $'a \Rightarrow$ *bool* **where** *strict-linear x* $\equiv x \sqcup x^T = -1$

abbreviation *strict-order* :: $'a \Rightarrow$ *bool* **where** *strict-order x* $\equiv$ *irreflexive x* $\wedge$ *transitive x*
abbreviation *linear-strict-order* :: $'a \Rightarrow$ *bool* **where** *linear-strict-order x* $\equiv$ *strict-order x* $\wedge$ *strict-linear x*

The following variants are useful for the graph model.

abbreviation *pp-mapping* :: $'a \Rightarrow$ *bool* **where** *pp-mapping x* $\equiv$ *univalent x* $\wedge$ *total* $(--x)$
abbreviation *pp-bijective* :: $'a \Rightarrow$ *bool* **where** *pp-bijective x* $\equiv$ *injective x* $\wedge$ *surjective* $(--x)$

abbreviation *pp-point* :: $'a \Rightarrow$ *bool* **where** *pp-point x* $\equiv$ *vector x* $\wedge$ *pp-bijective x*
abbreviation *pp-arc* :: $'a \Rightarrow$ *bool* **where** *pp-arc x* $\equiv$ *pp-bijective* $(x * top)$ $\wedge$ *pp-bijective* $(x^T * top)$

end

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

subclass *relation-algebra-signature* **.**

subclass *pd-algebra* **..**

lemma *conv-complement-1*:
$-(x^T) \sqcup (-x)^T = (-x)^T$
by (*metis conv-dist-inf conv-order bot-least conv-involutive pseudo-complement sup.absorb2 sup.cobounded2*)

lemma *conv-complement*:
$(-x)^T = -(x^T)$
by (*metis conv-complement-1 conv-dist-sup conv-involutive sup-commute*)

lemma *conv-complement-sub-inf* [*simp*]:
$x^T * -(x * y) \sqcap y = bot$

by (*metis comp-left-zero conv-dist-comp conv-involutive dedekind-1 inf-import-p inf-p inf-right-idem ppp pseudo-complement regular-closed-bot*)

lemma *conv-complement-sub-leq*:
$x^T * -(x * y) \leq -y$
using *pseudo-complement conv-complement-sub-inf* **by** *blast*

lemma *conv-complement-sub* [*simp*]:
$x^T * -(x * y) \sqcup -y = -y$
by (*simp add*: *conv-complement-sub-leq sup.absorb2*)

lemma *complement-conv-sub*:
$-(y * x) * x^T \leq -y$
by (*metis conv-complement conv-complement-sub-leq conv-order conv-dist-comp*)

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

lemma *schroeder-3-p*:
$x * y \leq -z \longleftrightarrow x^T * z \leq -y$
using *pseudo-complement schroeder-1* **by** *auto*

lemma *schroeder-4-p*:
$x * y \leq -z \longleftrightarrow z * y^T \leq -x$
using *pseudo-complement schroeder-2* **by** *auto*

lemma *comp-pp-semi-commute*:
$x * --y \leq --(x * y)$
using *conv-complement-sub-leq schroeder-3-p* **by** *fastforce*

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

lemma *p-comp-pp* [*simp*]:
$-(x * --y) = -(x * y)$
using *comp-pp-semi-commute comp-right-isotone order.eq-iff p-antitone pp-increasing* **by** *fastforce*

lemma *pp-comp-semi-commute*:
$--x * y \leq --(x * y)$
using *complement-conv-sub schroeder-4-p* **by** *fastforce*

lemma *p-pp-comp* [*simp*]:
$-(--x * y) = -(x * y)$
using *pp-comp-semi-commute comp-left-isotone order.eq-iff p-antitone pp-increasing* **by** *fastforce*

lemma *pp-comp-subdist*:
$--x * --y \leq --(x * y)$
by (*simp add*: *p-antitone-iff*)

lemma *theorem24xxiii*:

$x * y \sqcap -(x * z) = x * (y \sqcap -z) \sqcap -(x * z)$
proof −
have $x * y \sqcap -(x * z) \leq x * (y \sqcap (x^T * -(x * z)))$
by (*simp add*: *dedekind-1*)
also have ... $\leq x * (y \sqcap -z)$
using *comp-right-isotone conv-complement-sub-leq inf.sup-right-isotone* **by** *auto*
finally show *?thesis*
using *comp-right-subdist-inf order.antisym inf.coboundedI2 inf.commute* **by** *auto*
qed

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

lemma *vector-complement-closed*:
vector $x \Longrightarrow$ *vector* $(-x)$
by (*metis complement-conv-sub conv-top order.eq-iff top-right-mult-increasing*)

lemma *covector-complement-closed*:
covector $x \Longrightarrow$ *covector* $(-x)$
by (*metis conv-complement-sub-leq conv-top order.eq-iff top-left-mult-increasing*)

lemma *covector-vector-comp*:
vector $v \Longrightarrow -v^T * v =$ *bot*
by (*metis conv-bot conv-complement conv-complement-sub-inf conv-dist-comp conv-involutive inf-top.right-neutral*)

lemma *irreflexive-bot-closed*:
irreflexive bot
by *simp*

lemma *irreflexive-inf-closed*:
irreflexive $x \Longrightarrow$ *irreflexive* $(x \sqcap y)$
by (*simp add*: *le-infI1*)

lemma *irreflexive-sup-closed*:
irreflexive $x \Longrightarrow$ *irreflexive* $y \Longrightarrow$ *irreflexive* $(x \sqcup y)$
by *simp*

lemma *irreflexive-conv-closed*:
irreflexive $x \Longrightarrow$ *irreflexive* (x^T)
using *conv-complement conv-isotone* **by** *fastforce*

lemma *reflexive-complement-irreflexive*:
reflexive $x \Longrightarrow$ *irreflexive* $(-x)$
by (*simp add*: *p-antitone*)

lemma *irreflexive-complement-reflexive*:
irreflexive $x \longleftrightarrow$ *reflexive* $(-x)$

by (*simp add*: *p-antitone-iff*)

lemma *symmetric-complement-closed*:
symmetric x $\Longrightarrow$ *symmetric* $(-x)$
by (*simp add*: *conv-complement*)

lemma *asymmetric-irreflexive*:
asymmetric x $\Longrightarrow$ *irreflexive x*
by (*metis inf.mult-not-zero inf.left-commute inf.right-idem inf.sup-monoid.add-commute pseudo-complement one-inf-conv*)

lemma *linear-asymmetric*:
linear x $\Longrightarrow$ *asymmetric* $(-x)$
using *conv-complement p-top* **by** *force*

lemma *strict-linear-sup-closed*:
strict-linear x $\Longrightarrow$ *strict-linear y* $\Longrightarrow$ *strict-linear* $(x \sqcup y)$
by (*metis* (*mono-tags*, *opaque-lifting*) *conv-dist-sup sup.right-idem sup-assoc sup-commute*)

lemma *strict-linear-irreflexive*:
strict-linear x $\Longrightarrow$ *irreflexive x*
using *sup-left-divisibility* **by** *blast*

lemma *strict-linear-conv-closed*:
strict-linear x $\Longrightarrow$ *strict-linear* (x^T)
by (*simp add*: *sup-commute*)

lemma *strict-order-var*:
strict-order x $\longleftrightarrow$ *asymmetric x* $\wedge$ *transitive x*
by (*metis asymmetric-irreflexive comp-right-one irreflexive-conv-closed conv-dist-comp dual-order.trans pseudo-complement schroeder-3-p*)

lemma *strict-order-bot-closed*:
strict-order bot
by *simp*

lemma *strict-order-inf-closed*:
strict-order x $\Longrightarrow$ *strict-order y* $\Longrightarrow$ *strict-order* $(x \sqcap y)$
using *inf.coboundedI1 transitive-inf-closed* **by** *auto*

lemma *strict-order-conv-closed*:
strict-order x $\Longrightarrow$ *strict-order* (x^T)
using *irreflexive-conv-closed transitive-conv-closed* **by** *blast*

lemma *order-strict-order*:
assumes *order x*
shows *strict-order* $(x \sqcap -1)$
proof (*rule conjI*)

```
  show 1: irreflexive (x ⊓ −1)
    by simp
  have antisymmetric (x ⊓ −1)
    using antisymmetric-inf-closed assms by blast
  hence (x ⊓ −1) ∗ (x ⊓ −1) ⊓ 1 ≤ (x ⊓ −1 ⊓ (x ⊓ −1)^T) ∗ (x ⊓ −1 ⊓ (x ⊓
−1)^T)
    using 1 by (metis (no-types) coreflexive-symmetric irreflexive-inf-closed
coreflexive-transitive dedekind-1 inf-idem mult-1-right semiring.mult-not-zero
strict-order-var)
  also have ... = (x ⊓ x^T ⊓ −1) ∗ (x ⊓ x^T ⊓ −1)
    by (simp add: conv-complement conv-dist-inf inf.absorb2
inf.sup-monoid.add-assoc)
  also have ... = bot
    using assms order.antisym reflexive-conv-closed by fastforce
  finally have (x ⊓ −1) ∗ (x ⊓ −1) ≤ −1
    using le-bot pseudo-complement by blast
  thus transitive (x ⊓ −1)
    by (meson assms comp-isotone inf.boundedI inf.cobounded1
inf.order-lesseq-imp)
qed

lemma strict-order-order:
  strict-order x ⟹ order (x ⊔ 1)
  apply (unfold strict-order-var, intro conjI)
  apply simp
  apply (simp add: mult-left-dist-sup mult-right-dist-sup sup.absorb2)
  using conv-dist-sup coreflexive-bot-closed sup.absorb2 sup-inf-distrib2 by
fastforce

lemma linear-strict-order-conv-closed:
  linear-strict-order x ⟹ linear-strict-order (x^T)
  by (simp add: irreflexive-conv-closed sup-monoid.add-commute
transitive-conv-closed)

lemma linear-order-strict-order:
  linear-order x ⟹ linear-strict-order (x ⊓ −1)
  apply (rule conjI)
  using order-strict-order apply simp
  by (metis conv-complement conv-dist-inf coreflexive-symmetric order.eq-iff
inf.absorb2 inf.distrib-left inf.sup-monoid.add-commute top.extremum)

lemma regular-conv-closed:
  regular x ⟹ regular (x^T)
  by (metis conv-complement)
```

We show a number of facts about equivalences.

```
lemma equivalence-comp-left-complement:
  equivalence x ⟹ x ∗ −x = −x
  apply (rule order.antisym)
```

apply (*metis conv-complement-sub-leq preorder-idempotent*)
using *mult-left-isotone* **by** *fastforce*

lemma *equivalence-comp-right-complement*:
equivalence $x \Longrightarrow -x * x = -x$
by (*metis equivalence-comp-left-complement conv-complement conv-dist-comp*)

The pseudocomplement of tests is given by the following operation.

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

lemma *coreflexive-comp-top-coreflexive-complement*:
coreflexive $x \Longrightarrow (x * top)' = x\ '$
by (*metis coreflexive-comp-top-inf-one inf.commute inf-import-p*)

lemma *coreflexive-comp-inf-complement*:
coreflexive $x \Longrightarrow (x * y) \sqcap -z = (x * y) \sqcap -(x * z)$
by (*metis coreflexive-comp-top-inf inf.sup-relative-same-increasing inf-import-p inf-le1*)

lemma *double-coreflexive-complement*:
$x\ '' = (-x)'$
using *inf.sup-monoid.add-commute inf-import-p* **by** *auto*

lemma *coreflexive-pp-dist-comp*:
assumes *coreflexive* x
and *coreflexive* y
shows $(x * y)'' = x\ '' * y\ ''$
proof –
have $(x * y)'' = --(x * y) \sqcap 1$
by (*simp add*: *double-coreflexive-complement*)
also have ... $= --x \sqcap --y \sqcap 1$
by (*simp add*: *assms coreflexive-comp-inf*)
also have ... $= (--x \sqcap 1) * (--y \sqcap 1)$
by (*simp add*: *coreflexive-comp-inf inf.left-commute inf.sup-monoid.add-assoc*)
also have ... $= x\ '' * y\ ''$
by (*simp add*: *double-coreflexive-complement*)
finally show *?thesis*
.
qed

lemma *coreflexive-pseudo-complement*:
coreflexive $x \Longrightarrow x \sqcap y = bot \longleftrightarrow x \leq y\ '$
by (*simp add*: *pseudo-complement*)

lemma *pp-bijective-conv-mapping*:
pp-bijective $x \longleftrightarrow$ *pp-mapping* (x^T)
by (*simp add*: *conv-complement surjective-conv-total*)

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

proof

assume *1*: $pp\text{-}arc\ x$

have *2*: $x * top * x^T \leq 1$

using *1* **by** (*metis comp-associative conv-dist-comp equivalence-top-closed vector-top-closed*)

have *3*: $x^T * top * x \leq 1$

using *1* **by** (*metis conv-dist-comp conv-involutive equivalence-top-closed vector-top-closed mult-assoc*)

have *4*: $x^T \leq x^T * x * x^T$

by (*metis conv-involutive ex231c*)

have $top = --(top * x) * top$

using *1* **by** (*metis conv-complement conv-dist-comp conv-involutive equivalence-top-closed*)

also have ... $\leq --(top * x^T * top * x) * top$

using *1* **by** (*metis eq-refl mult-assoc p-comp-pp p-pp-comp*)

also have ... $= (top * --(x * top) \sqcap --(top * x^T * top * x)) * top$

using *1* **by** *simp*

also have ... $= top * (--(x * top) \sqcap --(top * x^T * top * x)) * top$

by (*simp add*: *covector-complement-closed covector-comp-inf covector-mult-closed*)

also have ... $= top * --(x * top \sqcap top * x^T * top * x) * top$

by *simp*

also have ... $= top * --(x * top * x^T * top * x) * top$

by (*metis comp-associative comp-inf-covector inf-top.left-neutral*)

also have ... $\leq top * --(x * top * x^T * x * x^T * top * x) * top$

using *4* **by** (*metis comp-associative comp-left-isotone comp-right-isotone pp-isotone*)

also have ... $\leq top * --(x * x^T * top * x) * top$

using *2* **by** (*metis comp-associative comp-left-isotone comp-right-isotone pp-isotone comp-left-one*)

also have ... $\leq top * --x * top$

using *3* **by** (*metis comp-associative comp-left-isotone comp-right-isotone pp-isotone comp-right-one*)

finally show $x * top * x^T \leq 1 \wedge x^T * top * x \leq 1 \wedge top * --x * top = top$

using *2 3 top-le* **by** *blast*

next

assume $x * top * x^T \leq 1 \wedge x^T * top * x \leq 1 \wedge top * --x * top = top$

thus $pp\text{-}arc\ x$

apply (*intro conjI*)

apply (*metis comp-associative conv-dist-comp equivalence-top-closed vector-top-closed*)

apply (*metis comp-associative mult-right-isotone top-le pp-comp-semi-commute*)

apply (*metis conv-dist-comp coreflexive-symmetric vector-conv-covector vector-top-closed mult-assoc*)

by (*metis conv-complement conv-dist-comp equivalence-top-closed inf.orderE inf-top.left-neutral mult-right-isotone pp-comp-semi-commute*)

qed

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*:
  $N\ x * y = -(-x * top) \sqcap y$
  by (*simp add*: *vector-mult-closed vector-complement-closed vector-inf-one-comp*)

lemma *N-comp-top* [*simp*]:
  $N\ x * top = -(-x * top)$
  by (*simp add*: *N-comp*)

lemma *vector-N-pp*:
  $vector\ x \Longrightarrow N\ x = --x \sqcap 1$
  by (*simp add*: *vector-complement-closed*)

lemma *N-vector-pp* [*simp*]:
  $N\ (x * top) = --(x * top) \sqcap 1$
  by (*simp add*: *comp-associative vector-complement-closed*)

lemma *N-vector-top-pp* [*simp*]:
  $N\ (x * top) * top = --(x * top)$
  by (*metis N-comp-top comp-associative vector-top-closed vector-complement-closed*)

lemma *N-below-inf-one-pp*:
  $N\ x \leq --x \sqcap 1$
  using *inf.sup-left-isotone p-antitone top-right-mult-increasing* **by** *auto*

lemma *N-below-pp*:
  $N\ x \leq --x$
  using *N-below-inf-one-pp* **by** *auto*

lemma *N-comp-N*:
  $N\ x * N\ y = -(-x * top) \sqcap -(-y * top) \sqcap 1$
  by (*simp add*: *N-comp inf.mult-assoc*)

lemma *N-bot* [*simp*]:
  $N\ bot = bot$
  by *simp*

lemma *N-top* [*simp*]:
  $N\ top = 1$
  by *simp*

lemma *n-split-omega-mult-pp*:

$xs * --xo = xo \Longrightarrow vector\ xo \Longrightarrow N\ top * xo = xs * N\ xo * top$
by (*metis N-top N-vector-top-pp comp-associative comp-left-one*)

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 $e * e = bot$
proof –
have $e * v \leq bot$
by (*metis assms covector-vector-comp comp-associative mult-left-isotone mult-right-zero*)
thus *?thesis*
by (*metis assms*(*2*) *bot-unique comp-associative mult-right-isotone semiring.mult-not-zero*)
qed

lemma *et*:
assumes $vector\ v$
and $e \leq v * -v^T$
and $t \leq v * v^T$
shows $e * t = bot$
and $e * t^T = bot$
proof –
have $e * t \leq v * -v^T * v * v^T$
using *assms*(*2–3*) *comp-isotone mult-assoc* **by** *fastforce*
thus $e * t = bot$
by (*simp add*: *assms*(*1*) *covector-vector-comp le-bot mult-assoc*)
next
have $t^T \leq v * v^T$
using *assms*(*3*) *conv-order conv-dist-comp* **by** *fastforce*
hence $e * t^T \leq v * -v^T * v * v^T$
by (*metis assms*(*2*) *comp-associative comp-isotone*)
thus $e * t^T = bot$
by (*simp add*: *assms*(*1*) *covector-vector-comp le-bot mult-assoc*)
qed

lemma *ve-dist*:
assumes $e \leq v * -v^T$
and $vector\ v$
and $arc\ e$
shows $(v \sqcup e^T * top) * (v \sqcup e^T * top)^T = v * v^T \sqcup v * v^T * e \sqcup e^T * v * v^T \sqcup e^T * e$
proof –
have $e \leq v * top$
using *assms*(*1*) *comp-right-isotone dual-order.trans top-greatest* **by** *blast*
hence $v * top * e = v * top * (v * top \sqcap e)$
by (*simp add*: *inf.absorb2*)

also have ... $= (v * top \sqcap top * v^T) * e$
using *assms(2) covector-inf-comp-3 vector-conv-covector* **by** *force*
also have ... $= v * top * v^T * e$
by (*metis assms(2) inf-top-right vector-inf-comp*)
also have ... $= v * v^T * e$
by (*simp add*: *assms(2)*)
finally have *1*: $v * top * e = v * v^T * e$
.
have $e^T * top * e \leq e^T * top * e * e^T * e$
using *ex231c comp-associative mult-right-isotone* **by** *auto*
also have ... $\leq e^T * e$
by (*metis assms(3) coreflexive-comp-top-inf le-infE mult-semi-associative point-injective*)
finally have *2*: $e^T * top * e = e^T * e$
by (*simp add*: *order.antisym mult-left-isotone top-right-mult-increasing*)
have $(v \sqcup e^T * top) * (v \sqcup e^T * top)^T = (v \sqcup e^T * top) * (v^T \sqcup top * e)$
by (*simp add*: *conv-dist-comp conv-dist-sup*)
also have ... $= v * v^T \sqcup v * top * e \sqcup e^T * top * v^T \sqcup e^T * top * top * e$
by (*metis semiring.distrib-left semiring.distrib-right sup-assoc mult-assoc*)
also have ... $= v * v^T \sqcup v * top * e \sqcup (v * top * e)^T \sqcup e^T * top * e$
by (*simp add*: *comp-associative conv-dist-comp*)
also have ... $= v * v^T \sqcup v * v^T * e \sqcup (v * v^T * e)^T \sqcup e^T * e$
using *1 2* **by** *simp*
finally show *?thesis*
by (*simp add*: *comp-associative conv-dist-comp*)
qed

lemma *ev*:
$vector\ v \Longrightarrow e \leq v * -v^T \Longrightarrow e * v = bot$
by (*metis covector-vector-comp order.antisym bot-least comp-associative mult-left-isotone mult-right-zero*)

lemma *vTeT*:
$vector\ v \Longrightarrow e \leq v * -v^T \Longrightarrow v^T * e^T = bot$
using *conv-bot ev conv-dist-comp* **by** *fastforce*

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 $t \sqcup e \leq ((v \sqcup e^T * top) * (v \sqcup e^T * top)^T) \sqcap g$
proof (*rule sup-least*)
have $t \leq ((v \sqcup e^T * top) * v^T) \sqcap g$
using *assms(2) le-supI1 mult-right-dist-sup* **by** *auto*
also have ... $\leq ((v \sqcup e^T * top) * (v \sqcup e^T * top)^T) \sqcap g$
using *comp-right-isotone conv-dist-sup inf.sup-left-isotone* **by** *auto*
finally show $t \leq ((v \sqcup e^T * top) * (v \sqcup e^T * top)^T) \sqcap g$
.

next
  have $e \le v * \mathit{top}$
    by (*meson assms*(*1*) *inf.boundedE mult-right-isotone order.trans top.extremum*)
  hence $e \le v * \mathit{top} \sqcap \mathit{top} * e$
    by (*simp add*: *top-left-mult-increasing*)
  also have ... $= v * \mathit{top} * e$
    by (*metis inf-top-right vector-export-comp*)
  finally have $e \le v * \mathit{top} * e \sqcap g$
    using *assms*(*1*) **by** *auto*
  also have ... $= v * (e^T * \mathit{top})^T \sqcap g$
    by (*simp add*: *comp-associative conv-dist-comp*)
  also have ... $\le v * (v \sqcup e^T * \mathit{top})^T \sqcap g$
    by (*simp add*: *conv-dist-sup mult-left-dist-sup sup.assoc sup.orderI*)
  also have ... $\le (v \sqcup e^T * \mathit{top}) * (v \sqcup e^T * \mathit{top})^T \sqcap g$
    using *inf.sup-left-isotone mult-right-sub-dist-sup-left* **by** *auto*
  finally show $e \le ((v \sqcup e^T * \mathit{top}) * (v \sqcup e^T * \mathit{top})^T) \sqcap g$
    .
qed

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*:
  $x * y * z \le -w \longleftrightarrow x^T * w * z^T \le -y$
  using *mult-assoc p-antitone-iff schroeder-3-p schroeder-4-p* **by** *auto*

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

lemma *schroeder-5-p*:
  $x * y \le -z \longleftrightarrow y * z^T \le -x^T$
  using *schroeder-3-p schroeder-4-p* **by** *auto*

lemma *schroeder-6-p*:
  $x * y \le -z \longleftrightarrow z^T * x \le -y^T$
  using *schroeder-3-p schroeder-4-p* **by** *auto*

lemma *vector-conv-compl*:
  $\mathit{vector}\ v \Longrightarrow \mathit{top} * -v^T = -v^T$
  by (*simp add*: *covector-complement-closed vector-conv-covector*)

Composition commutes, relative to the diversity relation.

lemma *comp-commute-below-diversity*:
  $x * y \le -1 \longleftrightarrow y * x \le -1$
  by (*metis comp-right-one conv-dist-comp conv-one schroeder-3-p schroeder-4-p*)

lemma *comp-injective-below-complement*:
  $\mathit{injective}\ y \Longrightarrow -x * y \le -(x * y)$

by (*metis p-antitone-iff comp-associative comp-right-isotone comp-right-one schroeder-4-p*)

lemma *comp-univalent-below-complement*:
univalent $x \Longrightarrow x * -y \leq -(x * y)$
by (*metis p-inf pseudo-complement semiring.mult-zero-right univalent-comp-left-dist-inf*)

Bijective relations and mappings can be exported from a pseudocomplement.

lemma *comp-bijective-complement*:
bijective $y \Longrightarrow -x * y = -(x * y)$
using *comp-injective-below-complement complement-conv-sub order.antisym shunt-bijective* **by** *blast*

lemma *comp-mapping-complement*:
mapping $x \Longrightarrow x * -y = -(x * y)$
by (*metis* (*full-types*) *comp-bijective-complement conv-complement conv-dist-comp conv-involutive total-conv-surjective*)

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
and $q * f^T \leq q$
and $e \leq q$
and $q * f^T \leq -e$
and *injective* e
and $q^T * q \sqcap f^T * f \leq 1$
shows *injective* $((f \sqcap -q) \sqcup (f \sqcap q)^T \sqcup e)$
proof –
have *1*: $(f \sqcap -q) * (f \sqcap -q)^T \leq 1$
by (*simp add*: *assms*(*1*) *injective-inf-closed*)
have *2*: $(f \sqcap -q) * (f \sqcap q) \leq 1$
proof –
have *21*: $bot = q * f^T \sqcap -\ q$
by (*metis assms*(*3*) *inf.sup-monoid.add-assoc inf.sup-right-divisibility inf-import-p inf-p*)
have $(f \sqcap -q) * (f \sqcap q) \leq -q * f \sqcap q$
by (*metis assms*(*2*) *comp-inf-covector comp-isotone inf.cobounded2 inf.left-idem*)
also have ... = *bot*
using *21 schroeder-2* **by** *auto*
finally show *?thesis*
by (*simp add*: *bot-unique*)
qed
have *3*: $(f \sqcap -q) * e^T \leq 1$
proof –

have $(f \sqcap -q) * e^T \le -q * e^T$
by (*simp add*: *mult-left-isotone*)
also have ... $=$ *bot*
by (*metis assms*(*2*,*4*) *bot-unique conv-bot conv-complement covector-complement-closed p-antitone p-bot regular-closed-bot schroeder-5-p*)
finally show *?thesis*
by (*simp add*: *bot-unique*)
qed
have *4*: $(f \sqcap q)^T * (f \sqcap -q)^T \le 1$
using *2 conv-dist-comp conv-isotone* **by** *force*
have *5*: $(f \sqcap q)^T * (f \sqcap q) \le 1$
proof −
have $(f \sqcap q)^T * (f \sqcap q) \le q^T * q \sqcap f^T * f$
by (*simp add*: *conv-isotone mult-isotone*)
also have ... ≤ 1
by (*simp add*: *assms*(*7*))
finally show *?thesis*
by *simp*
qed
have *6*: $(f \sqcap q)^T * e^T \le 1$
proof −
have $f^T * e^T \le -q^T$
using *assms*(*5*) *schroeder-5-p* **by** *simp*
hence $(f \sqcap q)^T * e^T =$ *bot*
by (*metis assms*(*2*,*5*) *conv-bot conv-dist-comp covector-comp-inf inf.absorb1 inf.cobounded2 inf.sup-monoid.add-commute inf-left-commute inf-p schroeder-4-p*)
thus *?thesis*
by (*simp add*: *bot-unique*)
qed
have *7*: $e * (f \sqcap -q)^T \le 1$
using *3 conv-dist-comp coreflexive-symmetric* **by** *fastforce*
have *8*: $e * (f \sqcap q) \le 1$
using *6 conv-dist-comp coreflexive-symmetric* **by** *fastforce*
have *9*: $e * e^T \le 1$
by (*simp add*: *assms*(*6*))
have $((f \sqcap -q) \sqcup (f \sqcap q)^T \sqcup e) * ((f \sqcap -q) \sqcup (f \sqcap q)^T \sqcup e)^T = (f \sqcap -q) * (f \sqcap -q)^T \sqcup (f \sqcap -q) * (f \sqcap q) \sqcup (f \sqcap -q) * e^T \sqcup (f \sqcap q)^T * (f \sqcap -q)^T \sqcup (f \sqcap q)^T * (f \sqcap q) \sqcup (f \sqcap q)^T * e^T \sqcup e * (f \sqcap -q)^T \sqcup e * (f \sqcap q) \sqcup e * e^T$
using *comp-left-dist-sup comp-right-dist-sup conv-dist-sup sup.assoc* **by** *simp*
also have ... ≤ 1
using *1 2 3 4 5 6 7 8 9* **by** *simp*
finally show *?thesis*
by *simp*
qed

lemma *kruskal-exchange-injective-inv-1*:
assumes *injective f*
and *covector q*
and $q * f^T \le q$

```
    and q^T ∗ q ⊓ f^T ∗ f ≤ 1
  shows injective ((f ⊓ −q) ⊔ (f ⊓ q)^T)
 using kruskal-injective-inv[where e=bot] by (simp add: assms)

lemma kruskal-exchange-acyclic-inv-3:
 assumes injective w
    and d ≤ w
  shows (w ⊓ −d) ∗ d^T ∗ top = bot
proof −
 have (w ⊓ −d) ∗ d^T ∗ top = (w ⊓ −d ⊓ (d^T ∗ top)^T) ∗ top
  by (simp add: comp-associative comp-inf-vector-1 conv-dist-comp)
 also have ... = (w ⊓ top ∗ d ⊓ −d) ∗ top
  by (simp add: conv-dist-comp inf-commute inf-left-commute)
 finally show ?thesis
  using assms epm-3 by simp
qed

lemma kruskal-subgraph-inv:
 assumes f ≤ −−(−h ⊓ g)
    and e ≤ −−g
    and symmetric h
    and symmetric g
  shows (f ⊓ −q) ⊔ (f ⊓ q)^T ⊔ e ≤ −−(−(h ⊓ −e ⊓ −e^T) ⊓ g)
proof −
 let ?f = (f ⊓ −q) ⊔ (f ⊓ q)^T ⊔ e
 let ?h = h ⊓ −e ⊓ −e^T
 have 1: f ⊓ −q ≤ −h ⊓ −−g
  using assms(1) inf.coboundedI1 by simp
 have (f ⊓ q)^T ≤ (−h ⊓ −−g)^T
  using assms(1) inf.coboundedI1 conv-isotone by simp
 also have ... = −h ⊓ −−g
  using assms(3,4) conv-complement conv-dist-inf by simp
 finally have ?f ≤ (−h ⊓ −−g) ⊔ (e ⊓ −−g)
  using 1 assms(2) inf.absorb1 semiring.add-right-mono by simp
 also have ... ≤ (−h ⊔ −−e) ⊓ −−g
  by (simp add: inf.coboundedI1 le-supI2 pp-increasing)
 also have ... ≤ −?h ⊓ −−g
  using inf.sup-left-isotone order-trans p-antitone-inf p-supdist-inf by blast
 finally show ?f ≤ −−(−?h ⊓ g)
  using inf-pp-semi-commute order-lesseq-imp by blast
qed

lemma antisymmetric-inf-diversity:
 antisymmetric x ⟹ x ⊓ −1 = x ⊓ −x^T
 by (smt (verit, del-insts) inf.orderE inf.sup-monoid.add-assoc
inf.sup-monoid.add-commute inf-import-p one-inf-conv)

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*]:
  $(-x)^T \sqcup (--x)^T = top$
  by (*metis conv-top conv-dist-sup stone*)

lemma *theorem24xxiv-pp*:
  $-(x * y) \sqcup --(x * z) = -(x * (y \sqcap -z)) \sqcup --(x * z)$
  by (*metis p-dist-inf theorem24xxiii*)

lemma *asymmetric-linear*:
  asymmetric $x \longleftrightarrow$ *linear* $(-x)$
  by (*metis conv-complement inf.distrib-left inf-p maddux-3-11-pp p-bot p-dist-inf*)

lemma *strict-linear-asymmetric*:
  strict-linear $x \Longrightarrow$ *antisymmetric* $(-x)$
  by (*metis conv-complement eq-refl p-dist-sup pp-one*)

lemma *regular-complement-top*:
  regular $x \Longrightarrow x \sqcup -x = top$
  by (*metis stone*)

lemma *regular-mult-closed*:
  regular $x \Longrightarrow$ *regular* $y \Longrightarrow$ *regular* $(x * y)$
  by (*simp add*: *pp-dist-comp*)

lemma *regular-one-closed*:
  regular 1
  by *simp*

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

lemma *pp-total*:
  total $(--x) \longleftrightarrow -(x{*}top) = bot$
  by (*simp add*: *dense-pp pp-dist-comp*)

lemma *pp-surjective*:
  surjective $(--x) \longleftrightarrow -(top{*}x) = bot$
  by (*metis p-bot p-comp-pp p-top pp-dist-comp*)

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

```
lemma bijective-regular:
  bijective x ⟹ regular x
  by (metis comp-bijective-complement mult-left-one regular-one-closed)

lemma mapping-regular:
  mapping x ⟹ regular x
  by (metis bijective-regular conv-complement conv-involutive
total-conv-surjective)

lemma arc-regular:
  assumes arc x
    shows regular x
proof –
  have −−x ≤ −−(x ∗ top ⊓ top ∗ x)
    by (simp add: pp-isotone top-left-mult-increasing top-right-mult-increasing)
  also have ... = −−(x ∗ top) ⊓ −−(top ∗ x)
    by simp
  also have ... = x ∗ top ⊓ top ∗ x
    by (metis assms bijective-regular conv-top conv-dist-comp conv-involutive
mapping-regular)
  also have ... ≤ x ∗ x^T ∗ top ∗ x
    by (metis comp-associative dedekind-1 inf.commute inf-top.right-neutral)
  also have ... ≤ x
    by (metis assms comp-right-one conv-top comp-associative conv-dist-comp
conv-involutive mult-right-isotone vector-top-closed)
  finally show ?thesis
    by (simp add: order.antisym pp-increasing)
qed



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 (unfold-locales, goal-cases)
  case 7
  show ?case by (simp add: inf-commute)
qed (auto simp: inf.assoc inf-sup-distrib2 inf-left-commute)
```

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 operations. 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*]: $1 = top$
begin

lemma *times-inf* [*simp*]:
$x * y = x \sqcap y$
by *simp*

subclass *stone-relation-algebra*
apply *unfold-locales*
using *comp-inf.mult-right-dist-sup inf-commute inf-assoc inf-left-commute pp-dist-inf min-def* **by** *simp-all*

lemma *times-dense*:
$x \neq bot \Longrightarrow y \neq bot \Longrightarrow x * y \neq bot$
using *inf-dense min-inf times-def* **by** *presburger*

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*]:
$x^T \sqcup (-x)^T = top$
by (*simp add*: *conv-complement*)

We now obtain the original formulations of the Schröder equivalences.

lemma *schroeder-3*:
$x * y \leq z \longleftrightarrow x^T * -z \leq -y$
by (*simp add*: *schroeder-3-p*)

lemma *schroeder-4*:
$x * y \leq z \longleftrightarrow -z * y^T \leq -x$
by (*simp add*: *schroeder-4-p*)

lemma *theorem24xxiv*:
$-(x * y) \sqcup (x * z) = -(x * (y \sqcap -z)) \sqcup (x * z)$
using *theorem24xxiv-pp* **by** *auto*

lemma *vector-N*:
$vector\ x \Longrightarrow N(x) = x \sqcap 1$

by (*simp add*: *vector-N-pp*)

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

lemma *N-vector-top* [*simp*]:
$N(x * top) * top = x * top$
using *N-vector-top-pp* **by** *simp*

lemma *N-below-inf-one*:
$N(x) \leq x \sqcap 1$
using *N-below-inf-one-pp* **by** *simp*

lemma *N-below*:
$N(x) \leq x$
using *N-below-pp* **by** *simp*

lemma *n-split-omega-mult*:
$xs * xo = xo \Longrightarrow xo * top = xo \Longrightarrow N(top) * xo = xs * N(xo) * top$
using *n-split-omega-mult-pp* **by** *simp*

lemma *complement-vector*:
$vector\ v \longleftrightarrow vector\ (-v)$
using *vector-complement-closed* **by** *fastforce*

lemma *complement-covector*:
$covector\ v \longleftrightarrow covector\ (-v)$
using *covector-complement-closed* **by** *force*

lemma *triple-schroeder*:
$x * y * z \leq w \longleftrightarrow x^T * -w * z^T \leq -y$
by (*simp add*: *triple-schroeder-p*)

lemma *schroeder-5*:
$x * y \leq z \longleftrightarrow y * -z^T \leq -x^T$
by (*simp add*: *conv-complement schroeder-5-p*)

lemma *schroeder-6*:
$x * y \leq z \longleftrightarrow -z^T * x \leq -y^T$
by (*simp add*: *conv-complement schroeder-5-p*)

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*: $regular\ x \Longrightarrow x \neq bot \Longrightarrow top * x * top = top$

```
begin
```

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

```
lemma arc-in-partition:
 assumes arc x
  shows x ≤ −y ∨ x ≤ −−y
proof −
 have 1: x ∗ top ∗ x^T ≤ 1 ∧ x^T ∗ top ∗ x ≤ 1
  using assms arc-expanded by auto
 have ¬ x ≤ −−y ⟶ x ≤ −y
 proof
  assume ¬ x ≤ −−y
  hence x ⊓ −y ≠ bot
   using pseudo-complement by simp
  hence top ∗ (x ⊓ −y) ∗ top = top
   using assms arc-regular tarski by auto
  hence x = x ⊓ top ∗ (x ⊓ −y) ∗ top
   by simp
  also have ... ≤ x ⊓ x ∗ ((x ⊓ −y) ∗ top)^T ∗ (x ⊓ −y) ∗ top
   by (metis dedekind-2 inf.coboundedI inf.boundedI inf-commute mult-assoc
inf.absorb2 top.extremum)
  also have ... = x ⊓ x ∗ top ∗ (x^T ⊓ −y^T) ∗ (x ⊓ −y) ∗ top
   by (simp add: comp-associative conv-complement conv-dist-comp
conv-dist-inf)
  also have ... ≤ x ⊓ x ∗ top ∗ x^T ∗ (x ⊓ −y) ∗ top
   using inf.sup-right-isotone mult-left-isotone mult-right-isotone by auto
  also have ... ≤ x ⊓ 1 ∗ (x ⊓ −y) ∗ top
   using 1 by (metis comp-associative comp-isotone inf.sup-right-isotone
mult-1-left mult-semi-associative)
  also have ... = x ⊓ (x ⊓ −y) ∗ top
   by simp
  also have ... ≤ (x ⊓ −y) ∗ ((x ⊓ −y)^T ∗ x)
   by (metis dedekind-1 inf-commute inf-top-right)
  also have ... ≤ (x ⊓ −y) ∗ (x^T ∗ x)
   by (simp add: conv-dist-inf mult-left-isotone mult-right-isotone)
  also have ... ≤ (x ⊓ −y) ∗ (x^T ∗ top ∗ x)
   by (simp add: mult-assoc mult-right-isotone top-left-mult-increasing)
  also have ... ≤ x ⊓ −y
   using 1 by (metis mult-right-isotone mult-1-right)
  finally show x ≤ −y
   by simp
 qed
 thus ?thesis
  by auto
qed

lemma non-bot-arc-in-partition-xor:
 assumes arc x
```

and $x \neq \mathit{bot}$
shows $(x \leq -y \wedge \neg\ x \leq --y) \vee (\neg\ x \leq -y \wedge x \leq --y)$
proof $-$
have $x \leq -y \wedge x \leq --y \longrightarrow \mathit{False}$
by (*simp add*: *assms*(*2*) *inf-absorb1 shunting-1-pp*)
thus *?thesis*
using *assms*(*1*) *arc-in-partition* **by** *auto*
qed

lemma *point-in-vector-or-pseudo-complement*:
assumes *point p*
and *vector v*
shows $p \leq --v \vee p \leq -v$
proof (*rule disjCI*)
assume $\neg(p \leq -v)$
hence $\mathit{top} * (p \sqcap --v) = \mathit{top}$
by (*smt assms bijective-regular regular-closed-inf regular-closed-p shunting-1-pp tarski vector-complement-closed vector-inf-closed vector-mult-closed*)
thus $p \leq --v$
by (*metis assms*(*1*) *epm-3 inf.absorb-iff1 inf.coboundedl inf-top.right-neutral*)
qed

lemma *distinct-points*:
assumes *point x*
and *point y*
and $x \neq y$
shows $x \sqcap y = \mathit{bot}$
by (*metis assms order.antisym comp-bijective-complement inf.sup-monoid.add-commute mult-left-one pseudo-complement regular-one-closed point-in-vector-or-pseudo-complement*)

lemma *point-in-vector-or-complement*:
assumes *point p*
and *vector v*
and *regular v*
shows $p \leq v \vee p \leq -v$
using *assms point-in-vector-or-pseudo-complement* **by** *fastforce*

lemma *point-in-vector-sup*:
assumes *point p*
and *vector v*
and *regular v*
and $p \leq v \sqcup w$
shows $p \leq v \vee p \leq w$
by (*metis assms inf.absorb1 shunting-var-p sup-commute point-in-vector-or-complement*)

lemma *point-atomic-vector*:
assumes *point x*

```
    and vector y
    and regular y
    and y ≤ x
  shows y = x ∨ y = bot
proof (cases x ≤ −y)
  case True
  thus ?thesis
    using assms(4) inf.absorb2 pseudo-complement by force
next
  case False
  thus ?thesis
    using assms point-in-vector-or-pseudo-complement by fastforce
qed

lemma point-in-vector-or-complement-2:
  assumes point x
    and vector y
    and regular y
    and ¬ y ≤ −x
  shows x ≤ y
  using assms point-in-vector-or-pseudo-complement p-antitone-iff by fastforce
```

The next three lemmas *arc-in-arc-or-complement*, *arc-in-sup-arc* and *different-arc-in-sup-arc* were contributed by Nicolas Robinson-O'Brien.

```
lemma arc-in-arc-or-complement:
  assumes arc x
      and arc y
      and ¬ x ≤ y
    shows x ≤ −y
  using assms arc-in-partition arc-regular by force

lemma arc-in-sup-arc:
  assumes arc x
      and arc y
      and x ≤ z ⊔ y
    shows x ≤ z ∨ x ≤ y
proof (cases x ≤ y)
  case True
  thus ?thesis
    by simp
next
  case False
  hence x ≤ −y
    using assms(1,2) arc-in-arc-or-complement by blast
  hence x ≤ −y ⊓ (z ⊔ y)
    using assms(3) by simp
  hence x ≤ z
    by (metis inf.boundedE inf.sup-monoid.add-commute maddux-3-13
sup-commute)
```

```
  thus ?thesis
    by simp
qed

lemma different-arc-in-sup-arc:
  assumes arc x
      and arc y
      and x ≤ z ⊔ y
      and x ≠ y
    shows x ≤ z
proof −
  have x ≤ −y
    using arc-in-arc-or-complement assms(1,2,4) order.eq-iff p-antitone-iff by
blast
  hence x ≤ −y ⊓ (z ⊔ y)
    using assms arc-in-sup-arc by simp
  thus ?thesis
    by (metis order-lesseq-imp p-inf-sup-below sup-commute)
qed

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 ≠ top
begin

lemma arc-not-bot:
  arc x ⟹ x ≠ bot
  using consistent mult-right-zero by auto

lemma point-not-bot:
  point p ⟹ p ≠ bot
  using consistent by force

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 $x \Longrightarrow (x \le -y \wedge \neg\, x \le --y) \vee (\neg\, x \le -y \wedge x \le --y)$
by (*simp add*: *non-bot-arc-in-partition-xor arc-not-bot*)

end

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

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* ⇒ *'a regular* ⇒ *'a regular* **is** *times*
using *regular-mult-closed regular-closed-p* **by** *blast*

lift-definition *conv-regular* :: *'a regular* ⇒ *'a regular* **is** *conv*
using *conv-complement* **by** *blast*

lift-definition *one-regular* :: *'a regular* **is** *1*
using *regular-one-closed* **by** *blast*

instance
apply *intro-classes*
apply (*metis* (*mono-tags*, *lifting*) *times-regular.rep-eq Rep-regular-inject*
comp-associative)
apply (*metis* (*mono-tags*, *lifting*) *times-regular.rep-eq Rep-regular-inject*
mult-right-dist-sup sup-regular.rep-eq)
apply (*metis* (*mono-tags*, *lifting*) *times-regular.rep-eq Rep-regular-inject*
bot-regular.rep-eq semiring.mult-zero-left)
apply (*simp add*: *one-regular.rep-eq times-regular.rep-eq*
Rep-regular-inject[*THEN sym*])
using *Rep-regular-inject conv-regular.rep-eq* **apply** *force*
apply (*metis* (*mono-tags*, *lifting*) *Rep-regular-inject conv-dist-sup*
conv-regular.rep-eq sup-regular.rep-eq)

```
  apply (metis (mono-tags, lifting) conv-regular.rep-eq times-regular.rep-eq
Rep-regular-inject conv-dist-comp)
  by (auto simp add: conv-regular.rep-eq dedekind-1 inf-regular.rep-eq
less-eq-regular.rep-eq times-regular.rep-eq)

end
```

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

```
typedef (overloaded) 'a coreflexive =
coreflexives::'a::non-associative-left-semiring set
  by auto

lemma simp-coreflexive [simp]:
  ∃ y . Rep-coreflexive x ≤ 1
  using Rep-coreflexive by simp

setup-lifting type-definition-coreflexive

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

lift-definition sup-coreflexive :: 'a coreflexive ⇒ 'a coreflexive ⇒ 'a coreflexive is
sup
  by simp

lift-definition times-coreflexive :: 'a coreflexive ⇒ 'a coreflexive ⇒ 'a coreflexive
is times
  by (simp add: coreflexive-mult-closed)

lift-definition bot-coreflexive :: 'a coreflexive is bot
  by simp

lift-definition one-coreflexive :: 'a coreflexive is 1
  by simp

lift-definition top-coreflexive :: 'a coreflexive is 1
  by simp

lift-definition less-eq-coreflexive :: 'a coreflexive ⇒ 'a coreflexive ⇒ bool is
less-eq .

lift-definition less-coreflexive :: 'a coreflexive ⇒ 'a coreflexive ⇒ bool is less .

instance
  apply intro-classes
  apply (simp-all add: less-coreflexive.rep-eq less-eq-coreflexive.rep-eq
less-le-not-le)[2]
  apply (meson less-eq-coreflexive.rep-eq order-trans)
```

apply (*simp-all add*: *Rep-coreflexive-inject bot-coreflexive.rep-eq less-eq-coreflexive.rep-eq sup-coreflexive.rep-eq*)[*5*]
apply (*simp add*: *semiring.distrib-left less-eq-coreflexive.rep-eq sup-coreflexive.rep-eq times-coreflexive.rep-eq*)
apply (*metis* (*mono-tags*, *lifting*) *sup-coreflexive.rep-eq times-coreflexive.rep-eq Rep-coreflexive-inject mult-right-dist-sup*)
apply (*simp add*: *times-coreflexive.rep-eq bot-coreflexive.rep-eq Rep-coreflexive-inject*[*THEN sym*])
apply (*simp add*: *one-coreflexive.rep-eq times-coreflexive.rep-eq Rep-coreflexive-inject*[*THEN sym*])
apply (*simp add*: *one-coreflexive.rep-eq less-eq-coreflexive.rep-eq times-coreflexive.rep-eq*)
apply (*simp only*: *sup-coreflexive.rep-eq top-coreflexive.rep-eq Rep-coreflexive-inject*[*THEN sym*], *metis Abs-coreflexive-cases Abs-coreflexive-inverse mem-Collect-eq sup.absorb2*)
apply (*simp add*: *less-eq-coreflexive.rep-eq mult.assoc times-coreflexive.rep-eq*)
apply (*metis* (*mono-tags*, *lifting*) *times-coreflexive.rep-eq Rep-coreflexive-inject mult.assoc*)
using *Rep-coreflexive-inject one-coreflexive.rep-eq times-coreflexive.rep-eq*
apply *fastforce*
apply (*metis* (*mono-tags*, *lifting*) *sup-coreflexive.rep-eq times-coreflexive.rep-eq Rep-coreflexive-inject mult-left-dist-sup*)
by (*simp add*: *times-coreflexive.rep-eq bot-coreflexive.rep-eq Rep-coreflexive-inject*[*THEN sym*])

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*
by (*simp add*: *le-infI1*)

lift-definition *minus-coreflexive* :: *'a coreflexive* ⇒ *'a coreflexive* ⇒ *'a coreflexive* **is** $\lambda x\ y\ .\ x \sqcap -y$
by (*simp add*: *le-infI1*)

lift-definition *uminus-coreflexive* :: *'a coreflexive* ⇒ *'a coreflexive* **is** $\lambda x\ .\ -x \sqcap 1$
by *simp*

lift-definition *conv-coreflexive* :: *'a coreflexive* ⇒ *'a coreflexive* **is** *id*
by *simp*

instance
apply *intro-classes*

apply (*auto simp*: *inf-coreflexive.rep-eq less-eq-coreflexive.rep-eq*)[*3*]
apply *simp*
apply (*metis* (*mono-tags*, *lifting*) *Rep-coreflexive-inject inf-coreflexive.rep-eq sup-coreflexive.rep-eq sup-inf-distrib1*)
apply (*metis* (*mono-tags*, *lifting*) *Rep-coreflexive-inject bot-coreflexive.rep-eq top-greatest coreflexive-pseudo-complement inf-coreflexive.rep-eq less-eq-coreflexive.rep-eq one-coreflexive.rep-eq one-coreflexive-def top-coreflexive-def uminus-coreflexive.rep-eq*)
apply (*metis* (*mono-tags*, *lifting*) *Rep-coreflexive-inject maddux-3-21-pp one-coreflexive.rep-eq one-coreflexive-def pp-dist-inf pp-one regular-closed-p sup-coreflexive.rep-eq sup-right-top top-coreflexive-def uminus-coreflexive.rep-eq*)
apply (*auto simp*: *mult.assoc mult-right-dist-sup*)[*4*]
using *Rep-coreflexive-inject conv-coreflexive.rep-eq* **apply** *fastforce*
apply (*metis* (*mono-tags*) *Rep-coreflexive-inject conv-coreflexive.rep-eq*)
apply (*metis* (*mono-tags*, *lifting*) *Rep-coreflexive-inject top-greatest conv-coreflexive.rep-eq coreflexive-commutative less-eq-coreflexive.rep-eq one-coreflexive.rep-eq one-coreflexive-def times-coreflexive.rep-eq top-coreflexive-def*)
apply (*simp only*: *conv-coreflexive.rep-eq less-eq-coreflexive.rep-eq one-coreflexive.rep-eq times-coreflexive.rep-eq inf-coreflexive.rep-eq Rep-coreflexive-inject*[*THEN sym*], *metis coreflexive-dedekind Rep-coreflexive mem-Collect-eq*)
apply (*metis* (*mono-tags*, *lifting*) *Rep-coreflexive Rep-coreflexive-inject coreflexive-pp-dist-comp mem-Collect-eq times-coreflexive.rep-eq uminus-coreflexive.rep-eq*)
by (*metis* (*mono-tags*, *opaque-lifting*) *Rep-coreflexive-inverse inf.commute inf.idem inf-import-p one-coreflexive.rep-eq pp-one uminus-coreflexive.rep-eq*)

end

Vectors in a Stone relation algebra form a Stone subalgebra.

typedef (**overloaded**) *'a vector* = *vectors*::*'a*::*bounded-pre-left-semiring set*
using *surjective-top-closed* **by** *blast*

lemma *simp-vector* [*simp*]:
∃ *y* . *Rep-vector x* ∗ *top* = *Rep-vector x*
using *Rep-vector* **by** *simp*

setup-lifting *type-definition-vector*

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

lift-definition *sup-vector* :: *'a vector* ⇒ *'a vector* ⇒ *'a vector* **is** *sup*
by (*simp add*: *vector-sup-closed*)

lift-definition *inf-vector* :: *'a vector* ⇒ *'a vector* ⇒ *'a vector* **is** *inf*
by (*simp add*: *vector-inf-closed*)

```
lift-definition uminus-vector :: 'a vector ⇒ 'a vector is uminus
  by (simp add: vector-complement-closed)

lift-definition bot-vector :: 'a vector is bot
  by simp

lift-definition top-vector :: 'a vector is top
  by simp

lift-definition less-eq-vector :: 'a vector ⇒ 'a vector ⇒ bool is less-eq .

lift-definition less-vector :: 'a vector ⇒ 'a vector ⇒ bool is less .

instance
  apply intro-classes
  apply (auto simp: Rep-vector-inject top-vector.rep-eq bot-vector.rep-eq
less-le-not-le inf-vector.rep-eq sup-vector.rep-eq less-eq-vector.rep-eq
less-vector.rep-eq)[12]
  apply (metis (mono-tags, lifting) Rep-vector-inject inf-vector.rep-eq
sup-inf-distrib1 sup-vector.rep-eq)
  apply (metis (mono-tags, lifting) Rep-vector-inject bot-vector-def
bot-vector.rep-eq pseudo-complement inf-vector.rep-eq less-eq-vector.rep-eq
uminus-vector.rep-eq)
  by (metis (mono-tags, lifting) sup-vector.rep-eq uminus-vector.rep-eq
Rep-vector-inverse stone top-vector.abs-eq)

end
```

Covectors in a Stone relation algebra form a Stone subalgebra.

```
typedef (overloaded) 'a covector = covectors::'a::bounded-pre-left-semiring set
  using surjective-top-closed by blast

lemma simp-covector [simp]:
  ∃ y . top ∗ Rep-covector x = Rep-covector x
  using Rep-covector by simp

setup-lifting type-definition-covector

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

lift-definition sup-covector :: 'a covector ⇒ 'a covector ⇒ 'a covector is sup
  by (simp add: covector-sup-closed)

lift-definition inf-covector :: 'a covector ⇒ 'a covector ⇒ 'a covector is inf
  by (simp add: covector-inf-closed)

lift-definition uminus-covector :: 'a covector ⇒ 'a covector is uminus
  by (simp add: covector-complement-closed)
```

```
lift-definition bot-covector :: 'a covector is bot
  by simp

lift-definition top-covector :: 'a covector is top
  by simp

lift-definition less-eq-covector :: 'a covector ⇒ 'a covector ⇒ bool is less-eq .

lift-definition less-covector :: 'a covector ⇒ 'a covector ⇒ bool is less .

instance
  apply intro-classes
  apply (auto simp: Rep-covector-inject less-eq-covector.rep-eq inf-covector.rep-eq
bot-covector.rep-eq top-covector.rep-eq sup-covector.rep-eq less-le-not-le
less-covector.rep-eq)[12]
  apply (metis (mono-tags, lifting) Rep-covector-inject inf-covector.rep-eq
sup-inf-distrib1 sup-covector.rep-eq)
  apply (metis (mono-tags, lifting) Rep-covector-inject bot-covector-def
bot-covector.rep-eq pseudo-complement inf-covector.rep-eq less-eq-covector.rep-eq
uminus-covector.rep-eq)
  by (metis (mono-tags, lifting) sup-covector.rep-eq uminus-covector.rep-eq
Rep-covector-inverse stone top-covector.abs-eq)

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
  -sum-sup-monoid :: idt ⇒ 'a::bounded-semilattice-sup-bot ⇒ 'a ((⨆_{-} -) [0,10]
10)
  -sum-sup-monoid-bounded :: idt ⇒ 'b set ⇒ 'a::bounded-semilattice-sup-bot ⇒
'a ((⨆_{-∈-} -) [0,51,10] 10)
translations
  ⨆_{x} t => XCONST sup-monoid.sum (λx . t) { x . CONST True }
  ⨆_{x∈X} t => XCONST sup-monoid.sum (λx . 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::finite ⇒ 'a
  assumes one: ⋀i . P (f i) (g i)
      and sup: ⋀j I . j ∉ I ⟹ P (⨆_{i∈I} f i) (⨆_{i∈I} g i) ⟹ P (f j ⊔ (⨆_{i∈I} f i))
(g j ⊔ (⨆_{i∈I} g i))
    shows P (⨆_{k} f k) (⨆_{k} g k)
proof −
  let ?X = { k::'b . True }
  have finite ?X and ?X ≠ {}
    by auto
  thus ?thesis
  proof (induct rule: finite-ne-induct)
    case (singleton i) thus ?case
      using one by simp
  next
    case (insert j I) thus ?case
      using sup by simp
  qed
qed

lemma bot-sup-induct [case-names bot sup]:
  fixes f g :: 'b::finite ⇒ 'a
```

```
  assumes bot: P bot bot
    and sup: ⋀j I . j ∉ I ⟹ P (⨆ i∈I f i) (⨆ i∈I g i) ⟹ P (f j ⊔ (⨆ i∈I f i))
(g j ⊔ (⨆ i∈I g i))
    shows P (⨆ k f k) (⨆ k g k)
  apply (induct rule: one-sup-induct)
  using bot sup apply fastforce
  using sup by blast
```

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::finite ⇒ 'a
  shows (⨆ k f k ∗ x) = (⨆ k f k) ∗ x
proof (induct rule: one-sup-induct)
  case one show ?case
    by simp
next
  case (sup j I) thus ?case
    using mult-right-dist-sup by auto
qed

lemma comp-left-dist-sum:
  fixes f :: 'b::finite ⇒ 'a
  shows (⨆ k x ∗ f k) = x ∗ (⨆ k f k)
proof (induct rule: one-sup-induct)
  case one show ?case
    by simp
next
  case (sup j I) thus ?case
    by (simp add: mult-left-dist-sup)
qed

lemma leq-sum:
  fixes f g :: 'b::finite ⇒ 'a
  shows (∀ k . f k ≤ g k) ⟹ (⨆ k f k) ≤ (⨆ k g k)
proof (induct rule: one-sup-induct)
  case one thus ?case
    by simp
next
  case (sup j I) thus ?case
    using sup-mono by blast
qed

lemma ub-sum:
  fixes f :: 'b::finite ⇒ 'a
  shows f i ≤ (⨆ k f k)
proof −
  have i ∈ { k . True }
```

```
    by simp
  thus f i ≤ (⨆_k f (k::'b))
    by (metis finite-code sup-monoid.sum.insert sup-ge1 mk-disjoint-insert)
qed

lemma lub-sum:
  fixes f :: 'b::finite ⇒ 'a
  assumes ∀ k . f k ≤ x
    shows (⨆_k f k) ≤ x
proof (induct rule: one-sup-induct)
  case one show ?case
    by (simp add: assms)
next
  case (sup j I) thus ?case
    using assms le-supI by blast
qed

lemma lub-sum-iff:
  fixes f :: 'b::finite ⇒ 'a
  shows (∀ k . f k ≤ x) ⟷ (⨆_k f k) ≤ x
  using order.trans ub-sum lub-sum by blast

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::finite ⇒ 'a
  shows (⨆_k (f k)^T) = (⨆_k f k)^T
proof (induct rule: one-sup-induct)
  case one show ?case
    by simp
next
  case (sup j I) thus ?case
    by (simp add: conv-dist-sup)
qed

lemma pp-dist-sum:
  fixes f :: 'b::finite ⇒ 'a
  shows (⨆_k −−f k) = −−(⨆_k f k)
proof (induct rule: one-sup-induct)
  case one show ?case
    by simp
next
  case (sup j I) thus ?case
    by simp
```

qed

lemma *inf-right-dist-sum*:
fixes $f :: {'b}{::}finite \Rightarrow {'a}$
shows $(\bigsqcup_k f\ k \sqcap x) = (\bigsqcup_k f\ k) \sqcap x$
by (*rule comp-inf.comp-right-dist-sum*)

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 $({'a},{'b})\ square = {'a} \times {'a} \Rightarrow {'b}$

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* $:: ({'a},{'b}{::}ord)\ square \Rightarrow ({'a},{'b})\ square \Rightarrow bool$
(**infix** $\preceq$ *50*) **where** $f \preceq g = (\forall\, e\ .\ f\ e \le g\ e)$
definition *less-matrix* $:: ({'a},{'b}{::}ord)\ square \Rightarrow ({'a},{'b})\ square \Rightarrow bool$
(**infix** $\prec$ *50*) **where** $f \prec g = (f \preceq g \wedge \neg\, g \preceq f)$
definition *sup-matrix* $:: ({'a},{'b}{::}sup)\ square \Rightarrow ({'a},{'b})\ square \Rightarrow ({'a},{'b})\ square$
(**infixl** $\oplus$ *65*) **where** $f \oplus g = (\lambda e\ .\ f\ e \sqcup g\ e)$
definition *inf-matrix* $:: ({'a},{'b}{::}inf)\ square \Rightarrow ({'a},{'b})\ square \Rightarrow ({'a},{'b})\ square$
(**infixl** $\otimes$ *67*) **where** $f \otimes g = (\lambda e\ .\ f\ e \sqcap g\ e)$
definition *minus-matrix* $:: ({'a},{'b}{::}\{uminus,inf\})\ square \Rightarrow ({'a},{'b})\ square \Rightarrow ({'a},{'b})\ square$ (**infixl** $\ominus$ *65*) **where** $f \ominus g = (\lambda e\ .\ f\ e \sqcap -g\ e)$
definition *implies-matrix* $:: ({'a},{'b}{::}implies)\ square \Rightarrow ({'a},{'b})\ square \Rightarrow ({'a},{'b})\ square$ (**infixl** $\oslash$ *65*) **where** $f \oslash g = (\lambda e\ .\ f\ e \leadsto g\ e)$
definition *times-matrix* $:: ({'a},{'b}{::}\{times,bounded\text{-}semilattice\text{-}sup\text{-}bot\})\ square \Rightarrow ({'a},{'b})\ square \Rightarrow ({'a},{'b})\ square$ (**infixl** $\odot$ *70*) **where** $f \odot g = (\lambda(i,j)\ .\ \bigsqcup_k f\ (i,k) * g\ (k,j))$
definition *uminus-matrix* $:: ({'a},{'b}{::}uminus)\ square \Rightarrow ({'a},{'b})\ square$
($\ominus$ - [*80*] *80*) **where** $\ominus f = (\lambda e\ .\ -f\ e)$
definition *conv-matrix* $:: ({'a},{'b}{::}conv)\ square \Rightarrow ({'a},{'b})\ square$
(-^t [*100*] *100*) **where** $f^t = (\lambda(i,j)\ .\ (f\ (j,i))^T)$
definition *bot-matrix* $:: ({'a},{'b}{::}bot)\ square$
(*mbot*) **where** $mbot = (\lambda e\ .\ bot)$
definition *top-matrix* $:: ({'a},{'b}{::}top)\ square$
(*mtop*) **where** $mtop = (\lambda e\ .\ top)$
definition *one-matrix* $:: ({'a},{'b}{::}\{one,bot\})\ square$
(*mone*) **where** $mone = (\lambda(i,j)\ .\ if\ i = j\ then\ 1\ else\ bot)$

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** *less-eq* = *less-eq-matrix* **and** *less* = *less-matrix* :: (*'a*,*'b*::*order*) *square* ⇒ (*'a*,*'b*) *square* ⇒ *bool*
  apply *unfold-locales*
  apply (*simp add*: *less-matrix-def*)
  apply (*simp add*: *less-eq-matrix-def*)
  apply (*meson less-eq-matrix-def order-trans*)
  by (*meson less-eq-matrix-def antisym ext*)

interpretation *matrix-semilattice-sup*: *semilattice-sup* **where** *sup* = *sup-matrix* **and** *less-eq* = *less-eq-matrix* **and** *less* = *less-matrix* :: (*'a*,*'b*::*semilattice-sup*) *square* ⇒ (*'a*,*'b*) *square* ⇒ *bool*
  apply *unfold-locales*
  apply (*simp add*: *sup-matrix-def less-eq-matrix-def*)
  apply (*simp add*: *sup-matrix-def less-eq-matrix-def*)
  by (*simp add*: *sup-matrix-def less-eq-matrix-def*)

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* ⇒ (*'a*,*'b*) *square* ⇒ *bool*
  apply *unfold-locales*
  apply (*simp add*: *inf-matrix-def less-eq-matrix-def*)
  apply (*simp add*: *inf-matrix-def less-eq-matrix-def*)
  by (*simp add*: *inf-matrix-def less-eq-matrix-def*)

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*
  apply *unfold-locales*
  by (*simp add*: *bot-matrix-def less-eq-matrix-def*)

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*
  apply *unfold-locales*
  by (*simp add*: *less-eq-matrix-def top-matrix-def*)

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* ⇒ (*'a*,*'b*) *square* ⇒ *bool* **..**

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* ⇒ (*'a*,*'b*) *square* ⇒ *bool*
  apply *unfold-locales*
  by (*simp add*: *sup-inf-distrib1 sup-matrix-def inf-matrix-def*)

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* **..**

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* **..**

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*
apply *unfold-locales*
apply (*unfold inf-matrix-def bot-matrix-def less-eq-matrix-def uminus-matrix-def*)
by (*meson pseudo-complement*)

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* **..**

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*
by *unfold-locales* (*simp add*: *sup-matrix-def uminus-matrix-def top-matrix-def*)

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*
apply *unfold-locales*
apply (*unfold inf-matrix-def sup-matrix-def bot-matrix-def top-matrix-def less-eq-matrix-def uminus-matrix-def implies-matrix-def*)
apply (*simp add*: *implies-galois*)
apply (*simp add*: *uminus-eq*)
by *simp*

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*
apply *unfold-locales*
apply *simp*
apply (*simp add*: *sup-matrix-def uminus-matrix-def top-matrix-def*)
by (*simp add*: *inf-matrix-def uminus-matrix-def minus-matrix-def*)

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
  fix f g h :: ('a,'b) square
  show (f ⊙ g) ⊙ h = f ⊙ (g ⊙ h)
  proof (rule ext, rule prod-cases)
    fix i j
    have ((f ⊙ g) ⊙ h) (i,j) = (⨆_l (f ⊙ g) (i,l) * h (l,j))
      by (simp add: times-matrix-def)
    also have ... = (⨆_l (⨆_k f (i,k) * g (k,l)) * h (l,j))
      by (simp add: times-matrix-def)
    also have ... = (⨆_l ⨆_k (f (i,k) * g (k,l)) * h (l,j))
      by (metis (no-types) comp-right-dist-sum)
    also have ... = (⨆_l ⨆_k f (i,k) * (g (k,l) * h (l,j)))
      by (simp add: mult.assoc)
    also have ... = (⨆_k ⨆_l f (i,k) * (g (k,l) * h (l,j)))
      using sup-monoid.sum.swap by auto
    also have ... = (⨆_k f (i,k) * (⨆_l g (k,l) * h (l,j)))
      by (metis (no-types) comp-left-dist-sum)
    also have ... = (⨆_k f (i,k) * (g ⊙ h) (k,j))
      by (simp add: times-matrix-def)
    also have ... = (f ⊙ (g ⊙ h)) (i,j)
      by (simp add: times-matrix-def)
    finally show ((f ⊙ g) ⊙ h) (i,j) = (f ⊙ (g ⊙ h)) (i,j)
      .
  qed
next
  fix f :: ('a,'b) square
  show mone ⊙ f = f
  proof (rule ext, rule prod-cases)
    fix i j
    have (mone ⊙ f) (i,j) = (⨆_k mone (i,k) * f (k,j))
      by (simp add: times-matrix-def)
    also have ... = (⨆_k (if i = k then 1 else bot) * f (k,j))
      by (simp add: one-matrix-def)
    also have ... = (⨆_k if i = k then 1 * f (k,j) else bot * f (k,j))
      by (metis (full-types, opaque-lifting))
    also have ... = (⨆_k if i = k then f (k,j) else bot)
      by (meson mult-left-one mult-left-zero)
    also have ... = f (i,j)
      by simp
    finally show (mone ⊙ f) (i,j) = f (i,j)
      .
  qed
next
```

```
  fix f :: ('a,'b) square
  show f ⊙ mone = f
  proof (rule ext, rule prod-cases)
    fix i j
    have (f ⊙ mone) (i,j) = (⨆_k f (i,k) ∗ mone (k,j))
      by (simp add: times-matrix-def)
    also have ... = (⨆_k f (i,k) ∗ (if k = j then 1 else bot))
      by (simp add: one-matrix-def)
    also have ... = (⨆_k if k = j then f (i,k) ∗ 1 else f (i,k) ∗ bot)
      by (metis (full-types, opaque-lifting))
    also have ... = (⨆_k if k = j then f (i,k) else bot)
      by (meson mult.right-neutral semiring.mult-zero-right)
    also have ... = f (i,j)
      by simp
    finally show (f ⊙ mone) (i,j) = f (i,j)
      .
  qed
qed

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
  fix f g h :: ('a,'b) square
  show f ⊙ g ⊕ f ⊙ h ≼ f ⊙ (g ⊕ h)
  proof (unfold less-eq-matrix-def, rule allI, rule prod-cases)
    fix i j
    have (f ⊙ g ⊕ f ⊙ h) (i,j) = (f ⊙ g) (i,j) ⊔ (f ⊙ h) (i,j)
      by (simp add: sup-matrix-def)
    also have ... = (⨆_k f (i,k) ∗ g (k,j)) ⊔ (⨆_k f (i,k) ∗ h (k,j))
      by (simp add: times-matrix-def)
    also have ... = (⨆_k f (i,k) ∗ g (k,j) ⊔ f (i,k) ∗ h (k,j))
      by (simp add: sup-monoid.sum.distrib)
    also have ... = (⨆_k f (i,k) ∗ (g (k,j) ⊔ h (k,j)))
      by (simp add: mult-left-dist-sup)
    also have ... = (⨆_k f (i,k) ∗ (g ⊕ h) (k,j))
      by (simp add: sup-matrix-def)
    also have ... = (f ⊙ (g ⊕ h)) (i,j)
      by (simp add: times-matrix-def)
    finally show (f ⊙ g ⊕ f ⊙ h) (i,j) ≤ (f ⊙ (g ⊕ h)) (i,j)
      by simp
  qed
next
  fix f g h :: ('a,'b) square
  show (f ⊕ g) ⊙ h = f ⊙ h ⊕ g ⊙ h
  proof (rule ext, rule prod-cases)
    fix i j
    have ((f ⊕ g) ⊙ h) (i,j) = (⨆_k (f ⊕ g) (i,k) ∗ h (k,j))
```

```
      by (simp add: times-matrix-def)
    also have ... = (⨆k (f (i,k) ⊔ g (i,k)) ∗ h (k,j))
      by (simp add: sup-matrix-def)
    also have ... = (⨆k f (i,k) ∗ h (k,j) ⊔ g (i,k) ∗ h (k,j))
      by (meson mult-right-dist-sup)
    also have ... = (⨆k f (i,k) ∗ h (k,j)) ⊔ (⨆k g (i,k) ∗ h (k,j))
      by (simp add: sup-monoid.sum.distrib)
    also have ... = (f ⊙ h) (i,j) ⊔ (g ⊙ h) (i,j)
      by (simp add: times-matrix-def)
    also have ... = (f ⊙ h ⊕ g ⊙ h) (i,j)
      by (simp add: sup-matrix-def)
    finally show ((f ⊕ g) ⊙ h) (i,j) = (f ⊙ h ⊕ g ⊙ h) (i,j)
      .
  qed
next
  fix f :: ('a,'b) square
  show mbot ⊙ f = mbot
  proof (rule ext, rule prod-cases)
    fix i j
    have (mbot ⊙ f) (i,j) = (⨆k mbot (i,k) ∗ f (k,j))
      by (simp add: times-matrix-def)
    also have ... = (⨆k bot ∗ f (k,j))
      by (simp add: bot-matrix-def)
    also have ... = bot
      by simp
    also have ... = mbot (i,j)
      by (simp add: bot-matrix-def)
    finally show (mbot ⊙ f) (i,j) = mbot (i,j)
      .
  qed
next
  fix f :: ('a,'b) square
  show mone ⊙ f = f
    by simp
next
  fix f :: ('a,'b) square
  show f ≼ f ⊙ mone
    by simp
next
  fix f g h :: ('a,'b) square
  show f ⊙ (g ⊕ h) = f ⊙ g ⊕ f ⊙ h
  proof (rule ext, rule prod-cases)
    fix i j
    have (f ⊙ (g ⊕ h)) (i,j) = (⨆k f (i,k) ∗ (g ⊕ h) (k,j))
      by (simp add: times-matrix-def)
    also have ... = (⨆k f (i,k) ∗ (g (k,j) ⊔ h (k,j)))
      by (simp add: sup-matrix-def)
    also have ... = (⨆k f (i,k) ∗ g (k,j) ⊔ f (i,k) ∗ h (k,j))
      by (meson mult-left-dist-sup)
```

```
    also have ... = (⨆k f (i,k) ∗ g (k,j)) ⊔ (⨆k f (i,k) ∗ h (k,j))
      by (simp add: sup-monoid.sum.distrib)
    also have ... = (f ⊙ g) (i,j) ⊔ (f ⊙ h) (i,j)
      by (simp add: times-matrix-def)
    also have ... = (f ⊙ g ⊕ f ⊙ h) (i,j)
      by (simp add: sup-matrix-def)
    finally show (f ⊙ (g ⊕ h)) (i,j) = (f ⊙ g ⊕ f ⊙ h) (i,j)
      .
  qed
next
  fix f :: ('a,'b) square
  show f ⊙ mbot = mbot
  proof (rule ext, rule prod-cases)
    fix i j
    have (f ⊙ mbot) (i,j) = (⨆k f (i,k) ∗ mbot (k,j))
      by (simp add: times-matrix-def)
    also have ... = (⨆k f (i,k) ∗ bot)
      by (simp add: bot-matrix-def)
    also have ... = bot
      by simp
    also have ... = mbot (i,j)
      by (simp add: bot-matrix-def)
    finally show (f ⊙ mbot) (i,j) = mbot (i,j)
      .
  qed
qed

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
  fix f :: ('a,'b) square
  show f ⊕ mtop = mtop
  proof
    fix e
    have (f ⊕ mtop) e = f e ⊔ mtop e
      by (simp add: sup-matrix-def)
    also have ... = f e ⊔ top
      by (simp add: top-matrix-def)
    also have ... = top
      by simp
    also have ... = mtop e
      by (simp add: top-matrix-def)
    finally show (f ⊕ mtop) e = mtop e
      .
  qed
qed
```

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
  fix f g h :: ('a,'b) square
  show (f ⊙ g) ⊙ h = f ⊙ (g ⊙ h)
    by (simp add: matrix-monoid.mult-assoc)
next
  fix f g h :: ('a,'b) square
  show (f ⊕ g) ⊙ h = f ⊙ h ⊕ g ⊙ h
    by (simp add: matrix-idempotent-semiring.mult-right-dist-sup)
next
  fix f :: ('a,'b) square
  show mbot ⊙ f = mbot
    by simp
next
  fix f :: ('a,'b) square
  show mone ⊙ f = f
    by simp
next
  fix f :: ('a,'b) square
  show f^tt = f
  proof (rule ext, rule prod-cases)
    fix i j
    have (f^tt) (i,j) = ((f^t) (j,i))^T
      by (simp add: conv-matrix-def)
    also have ... = f (i,j)
      by (simp add: conv-matrix-def)
    finally show (f^tt) (i,j) = f (i,j)
      .
  qed
next
  fix f g :: ('a,'b) square
  show (f ⊕ g)^t = f^t ⊕ g^t
  proof (rule ext, rule prod-cases)
    fix i j
    have ((f ⊕ g)^t) (i,j) = ((f ⊕ g) (j,i))^T
      by (simp add: conv-matrix-def)
    also have ... = (f (j,i) ⊔ g (j,i))^T
      by (simp add: sup-matrix-def)
    also have ... = (f^t) (i,j) ⊔ (g^t) (i,j)
      by (simp add: conv-matrix-def conv-dist-sup)
    also have ... = (f^t ⊕ g^t) (i,j)
```

by (*simp add*: *sup-matrix-def*)
finally show $((f \oplus g)^t)\ (i,j) = (f^t \oplus g^t)\ (i,j)$
.
qed
next
fix $f\ g$ :: $('a,'b)$ *square*
show $(f \odot g)^t = g^t \odot f^t$
proof (*rule ext*, *rule prod-cases*)
fix $i\ j$
have $((f \odot g)^t)\ (i,j) = ((f \odot g)\ (j,i))^T$
by (*simp add*: *conv-matrix-def*)
also have ... $= (\bigsqcup_k f\ (j,k) * g\ (k,i))^T$
by (*simp add*: *times-matrix-def*)
also have ... $= (\bigsqcup_k (f\ (j,k) * g\ (k,i))^T)$
by (*metis* (*no-types*) *conv-dist-sum*)
also have ... $= (\bigsqcup_k (g\ (k,i))^T * (f\ (j,k))^T)$
by (*simp add*: *conv-dist-comp*)
also have ... $= (\bigsqcup_k (g^t)\ (i,k) * (f^t)\ (k,j))$
by (*simp add*: *conv-matrix-def*)
also have ... $= (g^t \odot f^t)\ (i,j)$
by (*simp add*: *times-matrix-def*)
finally show $((f \odot g)^t)\ (i,j) = (g^t \odot f^t)\ (i,j)$
.
qed
next
fix $f\ g\ h$ :: $('a,'b)$ *square*
show $(f \odot g) \otimes h \preceq f \odot (g \otimes (f^t \odot h))$
proof (*unfold less-eq-matrix-def*, *rule allI*, *rule prod-cases*)
fix $i\ j$
have $((f \odot g) \otimes h)\ (i,j) = (f \odot g)\ (i,j) \sqcap h\ (i,j)$
by (*simp add*: *inf-matrix-def*)
also have ... $= (\bigsqcup_k f\ (i,k) * g\ (k,j)) \sqcap h\ (i,j)$
by (*simp add*: *times-matrix-def*)
also have ... $= (\bigsqcup_k f\ (i,k) * g\ (k,j) \sqcap h\ (i,j))$
by (*metis* (*no-types*) *inf-right-dist-sum*)
also have ... $\leq (\bigsqcup_k f\ (i,k) * (g\ (k,j) \sqcap (f\ (i,k))^T * h\ (i,j)))$
by (*rule leq-sum*, *meson dedekind-1*)
also have ... $= (\bigsqcup_k f\ (i,k) * (g\ (k,j) \sqcap (f^t)\ (k,i) * h\ (i,j)))$
by (*simp add*: *conv-matrix-def*)
also have ... $\leq (\bigsqcup_k f\ (i,k) * (g\ (k,j) \sqcap (\bigsqcup_l (f^t)\ (k,l) * h\ (l,j))))$
by (*rule leq-sum*, *rule allI*, *rule comp-right-isotone*, *rule inf.sup-right-isotone*, *rule ub-sum*)
also have ... $= (\bigsqcup_k f\ (i,k) * (g\ (k,j) \sqcap (f^t \odot h)\ (k,j)))$
by (*simp add*: *times-matrix-def*)
also have ... $= (\bigsqcup_k f\ (i,k) * (g \otimes (f^t \odot h))\ (k,j))$
by (*simp add*: *inf-matrix-def*)
also have ... $= (f \odot (g \otimes (f^t \odot h)))\ (i,j)$
by (*simp add*: *times-matrix-def*)
finally show $((f \odot g) \otimes h)\ (i,j) \leq (f \odot (g \otimes (f^t \odot h)))\ (i,j)$

```
      .
  qed
next
  fix f g :: ('a,'b) square
  show ⊖⊖(f ⊙ g) = ⊖⊖f ⊙ ⊖⊖g
  proof (rule ext, rule prod-cases)
    fix i j
    have (⊖⊖(f ⊙ g)) (i,j) = −−((f ⊙ g) (i,j))
      by (simp add: uminus-matrix-def)
    also have ... = −−(⨆k f (i,k) ∗ g (k,j))
      by (simp add: times-matrix-def)
    also have ... = (⨆k −−(f (i,k) ∗ g (k,j)))
      by (metis (no-types) pp-dist-sum)
    also have ... = (⨆k −−(f (i,k)) ∗ −−(g (k,j)))
      by (meson pp-dist-comp)
    also have ... = (⨆k (⊖⊖f) (i,k) ∗ (⊖⊖g) (k,j))
      by (simp add: uminus-matrix-def)
    also have ... = (⊖⊖f ⊙ ⊖⊖g) (i,j)
      by (simp add: times-matrix-def)
    finally show (⊖⊖(f ⊙ g)) (i,j) = (⊖⊖f ⊙ ⊖⊖g) (i,j)
      .
  qed
next
  let ?o = mone :: ('a,'b) square
  show ⊖⊖?o = ?o
  proof (rule ext, rule prod-cases)
    fix i j
    have (⊖⊖?o) (i,j) = −−(?o (i,j))
      by (simp add: uminus-matrix-def)
    also have ... = −−(if i = j then 1 else bot)
      by (simp add: one-matrix-def)
    also have ... = (if i = j then −−1 else −−bot)
      by simp
    also have ... = (if i = j then 1 else bot)
      by auto
    also have ... = ?o (i,j)
      by (simp add: one-matrix-def)
    finally show (⊖⊖?o) (i,j) = ?o (i,j)
      .
  qed
qed

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, covector, 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* **..**

end

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

lemma *linorder-finite-sup-selective*:
  fixes *f* :: *'a::finite* ⇒ *'b::linorder-stone-algebra-expansion*
  shows $\exists\, i\ .\ (\bigsqcup_k f\ k) = f\ i$
  apply (*induct rule*: *comp-inf.one-sup-induct*)
  apply *blast*
  using *sup-selective* **by** *fastforce*

lemma *linorder-top-finite-sup*:
  fixes *f* :: *'a::finite* ⇒ *'b::linorder-stone-algebra-expansion*
  assumes $\forall\, k\ .\ f\ k \neq$ *top*
    shows $(\bigsqcup_k f\ k) \neq$ *top*
  by (*metis assms linorder-finite-sup-selective*)

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* ⊙ *mtop*) (*i*,*j*) = $(\bigsqcup_k f\ (i,k))$
  apply (*unfold times-matrix-def top-matrix-def*)
  by (*metis* (*no-types*, *lifting*) *case-prod-conv comp-right-one one-def*
sup-monoid.sum.cong)

lemma *top-comp-linorder-matrix*:
  fixes *f* :: (*'a::finite,'b::linorder-stone-relation-algebra-expansion*) *square*
  shows (*mtop* ⊙ *f*) (*i*,*j*) = $(\bigsqcup_k f\ (k,j))$
  apply (*unfold times-matrix-def top-matrix-def*)
  by (*metis* (*no-types*, *lifting*) *case-prod-conv comp-left-one one-def*
sup-monoid.sum.cong)

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

```
lemma univalent-linorder-matrix-1:
  fixes f :: ('a::finite,'b::linorder-stone-relation-algebra-expansion) square
  assumes matrix-stone-relation-algebra.univalent f
      and f (i,j) ≠ bot
      and f (i,k) ≠ bot
    shows j = k
proof −
  have (f^t ⊙ f) (j,k) = (⨆_l (f^t) (j,l) ∗ f (l,k))
    by (simp add: times-matrix-def)
  also have ... = (⨆_l (f (l,j))^T ∗ f (l,k))
    by (simp add: conv-matrix-def)
  also have ... = (⨆_l f (l,j) ∗ f (l,k))
    by simp
  also have ... ≥ f (i,j) ∗ f (i,k)
    using comp-inf.ub-sum by fastforce
  finally have (f^t ⊙ f) (j,k) ≠ bot
    using assms(2,3) bot.extremum-uniqueI times-dense by fastforce
  hence mone (j,k) ≠ (bot::'b)
    by (metis assms(1) bot.extremum-uniqueI less-eq-matrix-def)
  thus ?thesis
    by (metis (mono-tags, lifting) case-prod-conv one-matrix-def)
qed

lemma univalent-linorder-matrix-2:
  fixes f :: ('a::finite,'b::linorder-stone-relation-algebra-expansion) square
  assumes ∀ i j k . f (i,j) ≠ bot ∧ f (i,k) ≠ bot ⟶ j = k
    shows matrix-stone-relation-algebra.univalent f
proof −
  show f^t ⊙ f ⪯ mone
  proof (unfold less-eq-matrix-def, rule allI, rule prod-cases)
    fix j k
    show (f^t ⊙ f) (j,k) ≤ mone (j,k)
    proof (cases j = k)
      assume j = k
      thus ?thesis
        by (simp add: one-matrix-def)
    next
      assume j ≠ k
      hence (⨆_i f (i,j) ∗ f (i,k)) = bot
        by (metis (no-types, lifting) assms semiring.mult-not-zero
sup-monoid.sum.neutral)
      thus ?thesis
        by (simp add: times-matrix-def conv-matrix-def)
    qed
  qed
qed
```

```
lemma univalent-linorder-matrix:
  fixes f :: ('a::finite,'b::linorder-stone-relation-algebra-expansion) square
  shows matrix-stone-relation-algebra.univalent f ⟷ (∀ i j k . f (i,j) ≠ bot ∧ f
(i,k) ≠ bot ⟶ j = k)
  using univalent-linorder-matrix-1 univalent-linorder-matrix-2 by auto
```

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::finite,'b::linorder-stone-relation-algebra-expansion) square
  shows matrix-stone-relation-algebra.injective f ⟷ (∀ i j k . f (j,i) ≠ bot ∧ f
(k,i) ≠ bot ⟶ j = k)
  by (unfold matrix-stone-relation-algebra.injective-conv-univalent
univalent-linorder-matrix) (simp add: conv-matrix-def)
```

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

```
lemma total-linorder-matrix-1:
  fixes f :: ('a::finite,'b::linorder-stone-relation-algebra-expansion) square
  assumes matrix-stone-relation-algebra.total-var f
    shows ∃ j . f (i,j) = top
proof −
  have mone (i,i) ≤ (f ⊙ f^t) (i,i)
    using assms less-eq-matrix-def by blast
  hence top = (f ⊙ f^t) (i,i)
    by (simp add: one-matrix-def top.extremum-unique)
  also have ... = (⨆_j f (i,j) ∗ (f^t) (j,i))
    by (simp add: times-matrix-def)
  also have ... = (⨆_j f (i,j) ∗ f (i,j))
    by (simp add: conv-matrix-def)
  also have ... = (⨆_j f (i,j))
    by simp
  finally show ?thesis
    by (metis linorder-top-finite-sup)
qed

lemma total-linorder-matrix-2:
  fixes f :: ('a::finite,'b::linorder-stone-relation-algebra-expansion) square
  assumes ∀ i . ∃ j . f (i,j) = top
    shows matrix-stone-relation-algebra.total-var f
proof (unfold less-eq-matrix-def, rule allI, rule prod-cases)
  fix j k
  show mone (j,k) ≤ (f ⊙ f^t) (j,k)
  proof (cases j = k)
    assume j = k
    hence (⨆_i f (j,i) ∗ (f^t) (i,k)) = (⨆_i f (j,i))
      by (simp add: conv-matrix-def)
    also have ... = top
      by (metis (no-types) assms comp-inf.ub-sum sup.absorb2 sup-top-left)
    finally show ?thesis
```

```
      by (simp add: times-matrix-def)
  next
    assume j ≠ k
    thus ?thesis
      by (simp add: one-matrix-def)
  qed
qed

lemma total-linorder-matrix:
  fixes f :: ('a::finite,'b::linorder-stone-relation-algebra-expansion) square
  shows matrix-bounded-idempotent-semiring.total f ⟷ (∀ i . ∃ j . f (i,j) = top)
  using total-linorder-matrix-1 total-linorder-matrix-2
matrix-stone-relation-algebra.total-var by auto
```

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 ⟷ (∀ j . ∃ i . f (i,j) =
top)
  by (unfold matrix-stone-relation-algebra.surjective-conv-total
total-linorder-matrix) (simp add: conv-matrix-def)
```

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 ⟷ (∀ i . ∃ j . f (i,j) = top ∧
(∀ k . j ≠ k ⟶ f (i,k) = bot))
  by (unfold total-linorder-matrix univalent-linorder-matrix) (metis (mono-tags,
opaque-lifting) comp-inf.mult-1-right comp-inf.mult-right-zero)

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 ⟷ (∀ i . ∃!j . f (i,j) = top ∧
(∀ k . j ≠ k ⟶ f (i,k) = bot))
  apply (unfold mapping-linorder-matrix)
  using bot-not-top by auto
```

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 ⟷ (∀ j . ∃ i . f (i,j) = top ∧
(∀ k . i ≠ k ⟶ f (k,j) = bot))
  by (unfold matrix-stone-relation-algebra.bijective-conv-mapping
mapping-linorder-matrix) (simp add: conv-matrix-def)
```

```
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 ⟷ (∀ j . ∃!i . f (i,j) = top ∧
(∀ k . i ≠ k ⟶ f (k,j) = bot))
  by (unfold matrix-stone-relation-algebra.bijective-conv-mapping
mapping-linorder-matrix-unique) (simp add: conv-matrix-def)
```

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 ⊖(f ⊙ mtop) = mbot
    shows ∃ j . f (i,j) ≠ bot
proof –
  have ¬(∃ j . f (i,j) ≠ bot) ⟹ ⊖(f ⊙ mtop) ≠ mbot
  proof –
    assume ¬(∃ j . f (i,j) ≠ bot)
    hence top = −(f ⊙ mtop) (i,i)
      by (simp add: comp-top-linorder-matrix linorder-finite-sup-selective)
    also have ... = (⊖(f ⊙ mtop)) (i,i)
      by (simp add: uminus-matrix-def)
    finally show ⊖(f ⊙ mtop) ≠ mbot
      by (metis bot-matrix-def bot-not-top)
  qed
  thus ?thesis
    using assms by blast
qed

lemma pp-total-linorder-matrix-2:
  fixes f :: ('a::finite,'b::linorder-stone-relation-algebra-expansion) square
  assumes ∀ i . ∃ j . f (i,j) ≠ bot
    shows ⊖(f ⊙ mtop) = mbot
proof (rule ext, rule prod-cases)
  fix i j
  have (⊖(f ⊙ mtop)) (i,j) = −(⨆k f (i,k))
    by (simp add: comp-top-linorder-matrix uminus-matrix-def)
  also have ... = bot
    by (metis antisym assms bot.extremum comp-inf.ub-sum uminus-def)
  finally show (⊖(f ⊙ mtop)) (i,j) = mbot (i,j)
    by (simp add: bot-matrix-def)
qed

lemma pp-total-linorder-matrix-3:
  fixes f :: ('a::finite,'b::non-trivial-linorder-stone-relation-algebra-expansion)
square
  shows ⊖(f ⊙ mtop) = mbot ⟷ (∀ i . ∃ j . f (i,j) ≠ bot)
  using pp-total-linorder-matrix-1 pp-total-linorder-matrix-2 by auto
```

lemma *pp-total-linorder-matrix*:
fixes *f* :: (*'a::finite,'b::non-trivial-linorder-stone-relation-algebra-expansion*) *square*
shows *matrix-bounded-idempotent-semiring.total* (⊖⊖*f*) ⟷ (∀ *i* . ∃ *j* . *f* (*i*,*j*) ≠ *bot*)
using *matrix-stone-relation-algebra.pp-total pp-total-linorder-matrix-1 pp-total-linorder-matrix-2* **by** *auto*

lemma *pp-mapping-linorder-matrix*:
fixes *f* :: (*'a::finite,'b::non-trivial-linorder-stone-relation-algebra-expansion*) *square*
shows *matrix-stone-relation-algebra.pp-mapping f* ⟷ (∀ *i* . ∃ *j* . *f* (*i*,*j*) ≠ *bot* ∧ (∀ *k* . *j* ≠ *k* ⟶ *f* (*i*,*k*) = *bot*))
by (*metis* (*mono-tags*, *opaque-lifting*) *pp-total-linorder-matrix univalent-linorder-matrix-1 univalent-linorder-matrix-2*)

lemma *pp-mapping-linorder-matrix-unique*:
fixes *f* :: (*'a::finite,'b::non-trivial-linorder-stone-relation-algebra-expansion*) *square*
shows *matrix-stone-relation-algebra.pp-mapping f* ⟷ (∀ *i* . ∃!*j* . *f* (*i*,*j*) ≠ *bot* ∧ (∀ *k* . *j* ≠ *k* ⟶ *f* (*i*,*k*) = *bot*))
apply (*rule iffI*)
using *pp-mapping-linorder-matrix* **apply** *blast*
by (*metis pp-total-linorder-matrix univalent-linorder-matrix*)

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

lemma *pp-surjective-linorder-matrix-1*:
fixes *f* :: (*'a::finite,'b::non-trivial-linorder-stone-relation-algebra-expansion*) *square*
shows ⊖(*mtop* ⊙ *f*) = *mbot* ⟷ (∀ *j* . ∃ *i* . *f* (*i*,*j*) ≠ *bot*)
proof –
have ⊖(*mtop* ⊙ *f*) = *mbot* ⟷ (⊖(*mtop* ⊙ *f*))^t = *mbot*^{t}
by (*metis matrix-stone-relation-algebra.conv-involutive*)
also have ... ⟷ ⊖(*f*^{t} ⊙ *mtop*) = *mbot*
by (*simp add*: *matrix-stone-relation-algebra.conv-complement matrix-stone-relation-algebra.conv-dist-comp*)
also have ... ⟷ (∀ *i* . ∃ *j* . (*f*^{t}) (*i*,*j*) ≠ *bot*)
using *pp-total-linorder-matrix-3* **by** *auto*
also have ... ⟷ (∀ *j* . ∃ *i* . *f* (*i*,*j*) ≠ *bot*)
by (*simp add*: *conv-matrix-def*)
finally show *?thesis*
.
qed

lemma *pp-surjective-linorder-matrix*:
fixes *f* :: (*'a::finite,'b::non-trivial-linorder-stone-relation-algebra-expansion*) *square*

shows *matrix-bounded-idempotent-semiring.surjective* (⊖⊖*f*) ⟷ (∀ *j* . ∃ *i* . *f*
(*i*,*j*) ≠ *bot*)
using *matrix-stone-relation-algebra.pp-surjective pp-surjective-linorder-matrix-1*
by *auto*

lemma *pp-bijective-linorder-matrix*:
fixes *f* :: (*'a*::*finite*,*'b*::*non-trivial-linorder-stone-relation-algebra-expansion*)
square
shows *matrix-stone-relation-algebra.pp-bijective f* ⟷ (∀ *j* . ∃ *i* . *f* (*i*,*j*) ≠ *bot* ∧
(∀ *k* . *i* ≠ *k* ⟶ *f* (*k*,*j*) = *bot*))
by (*unfold matrix-stone-relation-algebra.pp-bijective-conv-mapping*
pp-mapping-linorder-matrix) (*simp add*: *conv-matrix-def*)

lemma *pp-bijective-linorder-matrix-unique*:
fixes *f* :: (*'a*::*finite*,*'b*::*non-trivial-linorder-stone-relation-algebra-expansion*)
square
shows *matrix-stone-relation-algebra.pp-bijective f* ⟷ (∀ *j* . ∃!*i* . *f* (*i*,*j*) ≠ *bot*
∧ (∀ *k* . *i* ≠ *k* ⟶ *f* (*k*,*j*) = *bot*))
by (*unfold matrix-stone-relation-algebra.pp-bijective-conv-mapping*
pp-mapping-linorder-matrix-unique) (*simp add*: *conv-matrix-def*)

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

lemma *regular-linorder-matrix*:
fixes *f* :: (*'a*::*finite*,*'b*::*linorder-stone-relation-algebra-expansion*) *square*
shows *matrix-p-algebra.regular f* ⟷ (∀ *e* . *f e* = *bot* ∨ *f e* = *top*)
proof −
have *matrix-p-algebra.regular f* ⟷ (⊖⊖*f* = *f*)
by *auto*
also have ... ⟷ (∀ *e* . −−*f e* = *f e*)
by (*metis uminus-matrix-def ext*)
also have ... ⟷ (∀ *e* . *f e* = *bot* ∨ *f e* = *top*)
by *force*
finally show *?thesis*
.
qed

Vectors are precisely the row-constant matrices.

lemma *vector-linorder-matrix-0*:
fixes *f* :: (*'a*::*finite*,*'b*::*linorder-stone-relation-algebra-expansion*) *square*
assumes *matrix-bounded-idempotent-semiring.vector f*
shows *f* (*i*,*j*) = ($\bigsqcup_k$ *f* (*i*,*k*))
by (*metis assms comp-top-linorder-matrix*)

lemma *vector-linorder-matrix-1*:
fixes *f* :: (*'a*::*finite*,*'b*::*linorder-stone-relation-algebra-expansion*) *square*
assumes *matrix-bounded-idempotent-semiring.vector f*
shows *f* (*i*,*j*) = *f* (*i*,*k*)
by (*metis assms vector-linorder-matrix-0*)

lemma *vector-linorder-matrix-2*:
  fixes *f* :: (*'a*::*finite*,*'b*::*linorder-stone-relation-algebra-expansion*) *square*
  assumes ∀ *i j k* . *f* (*i*,*j*) = *f* (*i*,*k*)
    shows *matrix-bounded-idempotent-semiring.vector f*
proof (*rule ext*, *rule prod-cases*)
  fix *i j*
  have (*f* ⊙ *mtop*) (*i*,*j*) = ($\bigsqcup_k$ *f* (*i*,*k*))
    by (*simp add*: *comp-top-linorder-matrix*)
  also have ... = *f* (*i*,*j*)
    by (*metis assms linorder-finite-sup-selective*)
  finally show (*f* ⊙ *mtop*) (*i*,*j*) = *f* (*i*,*j*)
    .
qed

lemma *vector-linorder-matrix*:
  fixes *f* :: (*'a*::*finite*,*'b*::*linorder-stone-relation-algebra-expansion*) *square*
  shows *matrix-bounded-idempotent-semiring.vector f* ⟷ (∀ *i j k* . *f* (*i*,*j*) = *f* (*i*,*k*))
  using *vector-linorder-matrix-1 vector-linorder-matrix-2* **by** *auto*

Hence covectors are precisely the column-constant matrices.

lemma *covector-linorder-matrix-0*:
  fixes *f* :: (*'a*::*finite*,*'b*::*linorder-stone-relation-algebra-expansion*) *square*
  assumes *matrix-bounded-idempotent-semiring.covector f*
    shows *f* (*i*,*j*) = ($\bigsqcup_k$ *f* (*k*,*j*))
  by (*metis assms top-comp-linorder-matrix*)

lemma *covector-linorder-matrix*:
  fixes *f* :: (*'a*::*finite*,*'b*::*linorder-stone-relation-algebra-expansion*) *square*
  shows *matrix-bounded-idempotent-semiring.covector f* ⟷ (∀ *i j k* . *f* (*i*,*j*) = *f* (*k*,*j*))
  by (*unfold matrix-stone-relation-algebra.covector-conv-vector vector-linorder-matrix*) (*metis* (*no-types*, *lifting*) *case-prod-conv conv-matrix-def conv-def*)

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*::*finite*,*'b*::*linorder-stone-relation-algebra-expansion*) *square*
  shows *matrix-stone-relation-algebra.point f* ⟷ (∃ *i* . ∀ *j* . *f* (*i*,*j*) = *top* ∧ (∀ *k* . *i* ≠ *k* ⟶ *f* (*k*,*j*) = *bot*))
  apply (*unfold vector-linorder-matrix bijective-linorder-matrix*)
  apply (*rule iffI*)
  apply *metis*
  by *metis*

lemma *point-linorder-matrix-unique*:
  fixes *f* :: (*'a*::*finite*,*'b*::*non-trivial-linorder-stone-relation-algebra-expansion*) *square*

```
  shows matrix-stone-relation-algebra.point f ⟷ (∃!i . ∀ j . f (i,j) = top ∧ (∀ k . i ≠ k ⟶ f (k,j) = bot))
  apply (unfold vector-linorder-matrix bijective-linorder-matrix)
  apply (rule iffI)
  apply (metis bot-not-top)
  by metis

lemma pp-point-linorder-matrix:
  fixes f :: ('a::finite,'b::non-trivial-linorder-stone-relation-algebra-expansion) square
  shows matrix-stone-relation-algebra.pp-point f ⟷ (∃ i . ∀ j . f (i,j) ≠ bot ∧ (∀ k . f (i,j) = f (i,k)) ∧ (∀ k . i ≠ k ⟶ f (k,j) = bot))
  apply (unfold vector-linorder-matrix pp-bijective-linorder-matrix)
  apply (rule iffI)
  apply metis
  by metis

lemma pp-point-linorder-matrix-unique:
  fixes f :: ('a::finite,'b::non-trivial-linorder-stone-relation-algebra-expansion) square
  shows matrix-stone-relation-algebra.pp-point f ⟷ (∃!i . ∀ j . f (i,j) ≠ bot ∧ (∀ k . f (i,j) = f (i,k)) ∧ (∀ k . i ≠ k ⟶ f (k,j) = bot))
  apply (unfold vector-linorder-matrix pp-bijective-linorder-matrix)
  apply (rule iffI)
  apply metis
  by metis
```

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::finite,'b::non-trivial-linorder-stone-relation-algebra-expansion) square
  assumes matrix-stone-relation-algebra.arc f
    shows ∃ e . f e = top ∧ (∀ d . e ≠ d ⟶ f d = bot)
proof -
  have matrix-stone-relation-algebra.point (f ⊙ mtop)
    by (simp add: assms matrix-bounded-idempotent-semiring.vector-mult-closed)
  from this obtain i where 1: ∀ j . (f ⊙ mtop) (i,j) = top ∧ (∀ k . i ≠ k ⟶ (f ⊙ mtop) (k,j) = bot)
    using point-linorder-matrix by blast
  have matrix-stone-relation-algebra.point (f^t ⊙ mtop)
    by (simp add: assms matrix-bounded-idempotent-semiring.vector-mult-closed)
  from this obtain j where ∀ i . (f^t ⊙ mtop) (j,i) = top ∧ (∀ k . j ≠ k ⟶ (f^t ⊙ mtop) (k,i) = bot)
    using point-linorder-matrix by blast
  hence 2: ∀ i . (mtop ⊙ f) (i,j) = top ∧ (∀ k . j ≠ k ⟶ (mtop ⊙ f) (i,k) = bot)
    by (metis (no-types) old.prod.case conv-matrix-def conv-def matrix-stone-relation-algebra.conv-dist-comp matrix-stone-relation-algebra.conv-top)
```

```
  have 3: ∀ i k . j ≠ k ⟶ f (i,k) = bot
  proof (intro allI, rule impI)
    fix i k
    assume j ≠ k
    hence (⨆_l f (l,k)) = bot
      using 2 by (simp add: top-comp-linorder-matrix)
    thus f (i,k) = bot
      by (metis bot.extremum-uniqueI comp-inf.ub-sum)
  qed
  have (⨆_k f (i,k)) = top
    using 1 by (simp add: comp-top-linorder-matrix)
  hence 4: f (i,j) = top
    using 3 by (metis bot-not-top linorder-finite-sup-selective)
  have ∀ k l . k ≠ i ∨ l ≠ j ⟶ f (k,l) = bot
  proof (intro allI, unfold imp-disjL, rule conjI)
    fix k l
    show k ≠ i ⟶ f (k,l) = bot
    proof
      assume k ≠ i
      hence (⨆_m f (k,m)) = bot
        using 1 by (simp add: comp-top-linorder-matrix)
      thus f (k,l) = bot
        by (metis bot.extremum-uniqueI comp-inf.ub-sum)
    qed
    show l ≠ j ⟶ f (k,l) = bot
      using 3 by simp
  qed
  thus ?thesis using 4
    by (metis old.prod.exhaust)
qed

lemma pp-arc-linorder-matrix-2:
  fixes f :: ('a::finite,'b::linorder-stone-relation-algebra-expansion) square
  assumes ∃ e . f e ≠ bot ∧ (∀ d . e ≠ d ⟶ f d = bot)
    shows matrix-stone-relation-algebra.pp-arc f
proof (unfold matrix-stone-relation-algebra.pp-arc-expanded, intro conjI)
  show f ⊙ mtop ⊙ f^t ≼ mone
  proof (unfold less-eq-matrix-def, rule allI, rule prod-cases)
    fix i j
    show (f ⊙ mtop ⊙ f^t) (i,j) ≤ mone (i,j)
    proof (cases i = j)
      assume i = j
      thus ?thesis
        by (simp add: one-matrix-def)
    next
      assume i ≠ j
      hence 1: ∀ k l . f (i,k) * f (j,l) = bot
        by (metis assms Pair-inject semiring.mult-not-zero)
      have (f ⊙ mtop ⊙ f^t) (i,j) = (⨆_l (f ⊙ mtop) (i,l) * (f^t) (l,j))
```

```
        by (simp add: times-matrix-def)
      also have ... = (⨆_l (f ⊙ mtop) (i,l) ∗ f (j,l))
        by (simp add: conv-matrix-def)
      also have ... = (⨆_l (⨆_k f (i,k)) ∗ f (j,l))
        by (simp add: comp-top-linorder-matrix)
      also have ... = (⨆_l ⨆_k f (i,k) ∗ f (j,l))
        by (metis comp-right-dist-sum)
      also have ... = bot
        using 1 linorder-finite-sup-selective by simp
      finally show ?thesis
        by simp
    qed
  qed
next
  show f^t ⊙ mtop ⊙ f ≼ mone
  proof (unfold less-eq-matrix-def, rule allI, rule prod-cases)
    fix i j
    show (f^t ⊙ mtop ⊙ f) (i,j) ≤ mone (i,j)
    proof (cases i = j)
      assume i = j
      thus ?thesis
        by (simp add: one-matrix-def)
    next
      assume i ≠ j
      hence 2: ∀ k l . f (k,i) ∗ f (l,j) = bot
        by (metis assms Pair-inject semiring.mult-not-zero)
      have (f^t ⊙ mtop ⊙ f) (i,j) = (⨆_l (f^t ⊙ mtop) (i,l) ∗ f (l,j))
        by (simp add: times-matrix-def)
      also have ... = (⨆_l (⨆_k (f^t) (i,k)) ∗ f (l,j))
        by (simp add: comp-top-linorder-matrix)
      also have ... = (⨆_l (⨆_k f (k,i)) ∗ f (l,j))
        by (simp add: conv-matrix-def)
      also have ... = (⨆_l ⨆_k f (k,i) ∗ f (l,j))
        by (metis comp-right-dist-sum)
      also have ... = bot
        using 2 linorder-finite-sup-selective by simp
      finally show ?thesis
        by simp
    qed
  qed
next
  show mtop ⊙ ⊖⊖f ⊙ mtop = mtop
  proof (rule ext, rule prod-cases)
    fix i j
    from assms obtain k l where f (k,l) ≠ bot
      using prod.collapse by auto
    hence top = −−f (k,l)
      by simp
    also have ... ≤ (⨆_k −−f (k,l))
```

```
      using comp-inf.ub-sum by metis
    also have ... ≤ (⨆_l ⨆_k −−f (k,l))
      using comp-inf.ub-sum by simp
    finally have 3: top ≤ (⨆_l ⨆_k −−f (k,l))
      by simp
    have (mtop ⊙ ⊖⊖f ⊙ mtop) (i,j) = (⨆_l (⨆_k top ∗ −−f (k,l)) ∗ top)
      by (simp add: times-matrix-def top-matrix-def uminus-matrix-def)
    also have ... = (⨆_l ⨆_k −−f (k,l))
      by (metis (no-types, lifting) sup-monoid.sum.cong comp-inf.mult-1-left
times-inf comp-inf.mult-1-right)
    also have ... = top
      using 3 top.extremum-unique by blast
    finally show (mtop ⊙ ⊖⊖f ⊙ mtop) (i,j) = mtop (i,j)
      by (simp add: top-matrix-def)
  qed
qed

lemma arc-linorder-matrix-2:
  fixes f :: ('a::finite,'b::non-trivial-linorder-stone-relation-algebra-expansion)
square
  assumes ∃ e . f e = top ∧ (∀ d . e ≠ d ⟶ f d = bot)
    shows matrix-stone-relation-algebra.arc f
proof (unfold matrix-stone-relation-algebra.arc-expanded, intro conjI)
  show f ⊙ mtop ⊙ f^t ≼ mone
    by (metis (no-types, lifting) assms bot-not-top
matrix-stone-relation-algebra.pp-arc-expanded pp-arc-linorder-matrix-2)
next
  show f^t ⊙ mtop ⊙ f ≼ mone
    by (metis (no-types, lifting) assms bot-not-top
matrix-stone-relation-algebra.pp-arc-expanded pp-arc-linorder-matrix-2)
next
  show mtop ⊙ f ⊙ mtop = mtop
  proof (rule ext, rule prod-cases)
    fix i j
    from assms obtain k l where f (k,l) = top
      using prod.collapse by auto
    hence (⨆_k f (k,l)) = top
      by (metis (mono-tags) comp-inf.ub-sum top-unique)
    hence 3: top ≤ (⨆_l ⨆_k f (k,l))
      by (metis (no-types) comp-inf.ub-sum)
    have (mtop ⊙ f ⊙ mtop) (i,j) = (⨆_l (⨆_k top ∗ f (k,l)) ∗ top)
      by (simp add: times-matrix-def top-matrix-def)
    also have ... = (⨆_l ⨆_k f (k,l))
      by (metis (no-types, lifting) sup-monoid.sum.cong comp-inf.mult-1-left
times-inf comp-inf.mult-1-right)
    also have ... = top
      using 3 top.extremum-unique by blast
    finally show (mtop ⊙ f ⊙ mtop) (i,j) = mtop (i,j)
      by (simp add: top-matrix-def)
```

```
  qed
qed

lemma arc-linorder-matrix:
  fixes f :: ('a::finite,'b::non-trivial-linorder-stone-relation-algebra-expansion)
square
  shows matrix-stone-relation-algebra.arc f ⟷ (∃ e . f e = top ∧ (∀ d . e ≠ d
⟶ f d = bot))
  using arc-linorder-matrix-1 arc-linorder-matrix-2 by blast

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 ⟷ (∃!e . f e = top ∧ (∀ d . e ≠ d
⟶ f d = bot))
  apply (rule iffI)
  apply (metis (no-types, opaque-lifting) arc-linorder-matrix bot-not-top)
  using arc-linorder-matrix by blast

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 ∃ e . f e ≠ bot ∧ (∀ d . e ≠ d ⟶ f d = bot)
proof –
  have matrix-stone-relation-algebra.pp-point (f ⊙ mtop)
    by (simp add: assms matrix-bounded-idempotent-semiring.vector-mult-closed)
  from this obtain i where 1: ∀ j . (f ⊙ mtop) (i,j) ≠ bot ∧ (∀ k . (f ⊙ mtop)
(i,j) = (f ⊙ mtop) (i,k)) ∧ (∀ k . i ≠ k ⟶ (f ⊙ mtop) (k,j) = bot)
    by (metis pp-point-linorder-matrix)
  have matrix-stone-relation-algebra.pp-point (f^t ⊙ mtop)
    by (simp add: assms matrix-bounded-idempotent-semiring.vector-mult-closed)
  from this obtain j where ∀ i . (f^t ⊙ mtop) (j,i) ≠ bot ∧ (∀ k . (f^t ⊙ mtop)
(j,i) = (f^t ⊙ mtop) (j,k)) ∧ (∀ k . j ≠ k ⟶ (f^t ⊙ mtop) (k,i) = bot)
    by (metis pp-point-linorder-matrix)
  hence 2: ∀ i . (mtop ⊙ f) (i,j) ≠ bot ∧ (∀ k . (mtop ⊙ f) (i,j) = (mtop ⊙ f)
(k,j)) ∧ (∀ k . j ≠ k ⟶ (mtop ⊙ f) (i,k) = bot)
    by (metis (no-types) old.prod.case conv-matrix-def conv-def
matrix-stone-relation-algebra.conv-dist-comp
matrix-stone-relation-algebra.conv-top)
  have 3: ∀ i k . j ≠ k ⟶ f (i,k) = bot
  proof (intro allI, rule impI)
    fix i k
    assume j ≠ k
    hence (⨆_l f (l,k)) = bot
      using 2 by (simp add: top-comp-linorder-matrix)
    thus f (i,k) = bot
      by (metis bot.extremum-uniqueI comp-inf.ub-sum)
  qed
```

```
  have (⨆k f (i,k)) ≠ bot
    using 1 by (simp add: comp-top-linorder-matrix)
  hence 4: f (i,j) ≠ bot
    using 3 by (metis linorder-finite-sup-selective)
  have ∀ k l . k ≠ i ∨ l ≠ j ⟶ f (k,l) = bot
  proof (intro allI, unfold imp-disjL, rule conjI)
    fix k l
    show k ≠ i ⟶ f (k,l) = bot
    proof
      assume k ≠ i
      hence (⨆m f (k,m)) = bot
        using 1 by (simp add: comp-top-linorder-matrix)
      thus f (k,l) = bot
        by (metis bot.extremum-uniqueI comp-inf.ub-sum)
    qed
    show l ≠ j ⟶ f (k,l) = bot
      using 3 by simp
  qed
  thus ?thesis using 4
    by (metis old.prod.exhaust)
qed

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 ⟷ (∃ e . f e ≠ bot ∧ (∀ d . e ≠ d
⟶ f d = bot))
  using pp-arc-linorder-matrix-1 pp-arc-linorder-matrix-2 by blast

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 ⟷ (∃!e . f e ≠ bot ∧ (∀ d . e ≠
d ⟶ f d = bot))
  apply (rule iffI)
  apply (metis (no-types, opaque-lifting) pp-arc-linorder-matrix)
  using pp-arc-linorder-matrix by blast
```

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 –
  have (top::'b) = mone (i,i)
    by (simp add: one-matrix-def)
  also have ... ≤ f (i,i)
    using assms less-eq-matrix-def by blast
  finally show ?thesis
```

```
    by (simp add: top.extremum-unique)
qed

lemma reflexive-linorder-matrix-2:
  fixes f :: ('a::finite,'b::linorder-stone-relation-algebra-expansion) square
  assumes ∀ i . f (i,i) = top
    shows matrix-idempotent-semiring.reflexive f
proof (unfold less-eq-matrix-def, rule allI, rule prod-cases)
  fix i j
  show mone (i,j) ≤ f (i,j)
  proof (cases i = j)
    assume i = j
    thus ?thesis
      by (simp add: assms)
  next
    assume i ≠ j
    hence (bot::'b) = mone (i,j)
      by (simp add: one-matrix-def)
    thus ?thesis
      by simp
  qed
qed

lemma reflexive-linorder-matrix:
  fixes f :: ('a::finite,'b::linorder-stone-relation-algebra-expansion) square
  shows matrix-idempotent-semiring.reflexive f ⟷ (∀ i . f (i,i) = top)
  using reflexive-linorder-matrix-1 reflexive-linorder-matrix-2 by auto
```

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

```
lemma coreflexive-linorder-matrix-1:
  fixes f :: ('a::finite,'b::linorder-stone-relation-algebra-expansion) square
  assumes matrix-idempotent-semiring.coreflexive f
      and i ≠ j
    shows f (i,j) = bot
proof –
  have f (i,j) ≤ mone (i,j)
    using assms less-eq-matrix-def by blast
  also have ... = bot
    by (simp add: assms one-matrix-def)
  finally show ?thesis
    by (simp add: bot.extremum-unique)
qed

lemma coreflexive-linorder-matrix-2:
  fixes f :: ('a::finite,'b::linorder-stone-relation-algebra-expansion) square
  assumes ∀ i j . i ≠ j ⟶ f (i,j) = bot
    shows matrix-idempotent-semiring.coreflexive f
proof (unfold less-eq-matrix-def, rule allI, rule prod-cases)
  fix i j
```

```
  show f (i,j) ≤ mone (i,j)
  proof (cases i = j)
    assume i = j
    hence (top::'b) = mone (i,j)
      by (simp add: one-matrix-def)
    thus ?thesis
      by simp
  next
    assume i ≠ j
    thus ?thesis
      by (simp add: assms)
  qed
qed

lemma coreflexive-linorder-matrix:
  fixes f :: ('a::finite,'b::linorder-stone-relation-algebra-expansion) square
  shows matrix-idempotent-semiring.coreflexive f ⟷ (∀ i j . i ≠ j ⟶ f (i,j) =
bot)
  using coreflexive-linorder-matrix-1 coreflexive-linorder-matrix-2 by auto
```

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

```
lemma irreflexive-linorder-matrix-1:
  fixes f :: ('a::finite,'b::linorder-stone-relation-algebra-expansion) square
  assumes matrix-stone-relation-algebra.irreflexive f
    shows f (i,i) = bot
proof –
  have (top::'b) = mone (i,i)
    by (simp add: one-matrix-def)
  hence (bot::'b) = (⊖mone) (i,i)
    by (simp add: uminus-matrix-def)
  hence f (i,i) ≤ bot
    by (metis assms less-eq-matrix-def)
  thus ?thesis
    by (simp add: bot.extremum-unique)
qed

lemma irreflexive-linorder-matrix-2:
  fixes f :: ('a::finite,'b::linorder-stone-relation-algebra-expansion) square
  assumes ∀ i . f (i,i) = bot
    shows matrix-stone-relation-algebra.irreflexive f
proof (unfold less-eq-matrix-def, rule allI, rule prod-cases)
  fix i j
  show f (i,j) ≤ (⊖mone) (i,j)
  proof (cases i = j)
    assume i = j
    thus ?thesis
      by (simp add: assms)
  next
    assume i ≠ j
```

```
    hence (bot::'b) = mone (i,j)
      by (simp add: one-matrix-def)
    hence (top::'b) = (⊖mone) (i,j)
      by (simp add: uminus-matrix-def)
    thus ?thesis
      by simp
  qed
qed
```

```
lemma irreflexive-linorder-matrix:
  fixes f :: ('a::finite,'b::linorder-stone-relation-algebra-expansion) square
  shows matrix-stone-relation-algebra.irreflexive f ⟷ (∀ i . f (i,i) = bot)
  using irreflexive-linorder-matrix-1 irreflexive-linorder-matrix-2 by auto
```

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) square
  shows matrix-stone-relation-algebra.symmetric f ⟷ (∀ i j . f (i,j) = f (j,i))
  by (metis (mono-tags, lifting) case-prod-conv cond-case-prod-eta conv-matrix-def
conv-def)
```

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) square
  shows matrix-stone-relation-algebra.antisymmetric f ⟷ (∀ i j . i ≠ j ⟶ f
(i,j) = bot ∨ f (j,i) = bot)
proof –
  have matrix-stone-relation-algebra.antisymmetric f ⟷ (∀ i j . i ≠ j ⟶ f (i,j)
⊓ f (j,i) ≤ bot)
    by (simp add: conv-matrix-def inf-matrix-def less-eq-matrix-def
one-matrix-def)
  thus ?thesis
    by (metis (no-types, opaque-lifting) inf.absorb-iff1 inf.cobounded1 inf-bot-right
inf-dense)
qed
```

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) square
  shows matrix-stone-relation-algebra.asymmetric f ⟷ (∀ i j . f (i,j) = bot ∨ f
(j,i) = bot)
proof –
  have matrix-stone-relation-algebra.asymmetric f ⟷ (∀ i j . f (i,j) ⊓ f (j,i) ≤
bot)
    apply (unfold conv-matrix-def inf-matrix-def conv-def id-def bot-matrix-def)
```

by (*metis* (*mono-tags*, *lifting*) *bot.extremum bot.extremum-uniqueI case-prod-conv old.prod.exhaust*)
thus *?thesis*
by (*metis* (*no-types*, *opaque-lifting*) *inf.absorb-iff1 inf.cobounded1 inf-bot-right inf-dense*)
qed

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*) *square*
shows *matrix-idempotent-semiring.transitive* $f \longleftrightarrow (\forall\, i\ j\ k\ .\ f\ (i,k) \le f\ (i,j) \lor f\ (k,j) \le f\ (i,j))$
proof –
have *matrix-idempotent-semiring.transitive* $f \longleftrightarrow (\forall\, i\ j\ .\ (\bigsqcup_k f\ (i,k) * f\ (k,j)) \le f\ (i,j))$
by (*simp add*: *times-matrix-def less-eq-matrix-def*)
also have ... $\longleftrightarrow (\forall\, i\ j\ k\ .\ f\ (i,k) * f\ (k,j) \le f\ (i,j))$
by (*simp add*: *lub-sum-iff*)
also have ... $\longleftrightarrow (\forall\, i\ j\ k\ .\ f\ (i,k) \le f\ (i,j) \lor f\ (k,j) \le f\ (i,j))$
using *inf-less-eq* **by** *fastforce*
finally show *?thesis*
.
qed

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*) *square*
assumes *matrix-idempotent-semiring.coreflexive* f
shows $(f \odot g)\ (i,j) = f\ (i,i) \sqcap g\ (i,j)$
proof –
have *1*: $\forall\, k\ .\ i \ne k \longrightarrow f\ (i,k) = \mathit{bot}$
using *assms coreflexive-linorder-matrix* **by** *auto*
have $(\bigsqcup_k f\ (i,k)) = f\ (i,i) \sqcup (\bigsqcup_{k \in \mathit{UNIV}-\{i\}} f\ (i,k))$
by (*metis* (*no-types*) *UNIV-def brouwer.inf-bot-right finite-UNIV insert-def sup-monoid.sum.insert-remove*)
hence *2*: $(\bigsqcup_k f\ (i,k)) = f\ (i,i)$
using *1* **by** (*metis* (*no-types*) *linorder-finite-sup-selective sup-not-bot*)
have $(f \odot g)\ (i,j) = (f \odot \mathit{mtop} \otimes g)\ (i,j)$
by (*metis assms matrix-stone-relation-algebra.coreflexive-comp-top-inf*)
also have ... $= (\bigsqcup_k f\ (i,k)) \sqcap g\ (i,j)$
by (*metis inf-matrix-def comp-top-linorder-matrix*)
finally show *?thesis*
using *2* **by** *simp*
qed

lemma *comp-coreflexive-linorder-matrix*:
fixes *f g* :: (*'a*::*finite*,*'b*::*linorder-stone-relation-algebra-expansion*) *square*
assumes *matrix-idempotent-semiring.coreflexive g*
shows $(f \odot g)\ (i,j) = f\ (i,j) \sqcap g\ (j,j)$
proof −
have $(f \odot g)\ (i,j) = ((f \odot g)^t)\ (j,i)$
by (*simp add*: *conv-matrix-def*)
also have ... $= (g \odot f^t)\ (j,i)$
by (*simp add*: *assms matrix-stone-relation-algebra.conv-dist-comp matrix-stone-relation-algebra.coreflexive-symmetric*)
also have ... $= g\ (j,j) \sqcap (f^t)\ (j,i)$
by (*simp add*: *assms coreflexive-comp-linorder-matrix*)
also have ... $= f\ (i,j) \sqcap g\ (j,j)$
by (*metis* (*no-types*, *lifting*) *conv-def old.prod.case conv-matrix-def inf-commute*)
finally show *?thesis*
.
qed

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. Andréka and S. Mikulás. 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, Verbände 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 semigroups 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. Ščedrov. *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. Guttmann. 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. Guttmann. Stone algebras. *Archive of Formal Proofs*, 2016.

[16] W. Guttmann. 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.