Abstract
We develop Stone-Kleene relation algebras, which expand Stone relation algebras with a Kleene star operation to describe reachability in weighted graphs. Many properties of the Kleene star arise as a special case of a more general theory of iteration based on Conway semirings extended by simulation axioms. This includes several theorems representing complex program transformations. We formally prove the correctness of Conway’s automata-based construction of the Kleene star of a matrix. We prove numerous results useful for reasoning about weighted graphs.

Contents
1 Synopsis and Motivation ........................................ 2
2 Iterings .............................................................. 3
2.1 Conway Semirings ................................................. 3
2.2 Iterings ............................................................ 10
3 Kleene Algebras .................................................... 16
4 Kleene Relation Algebras .......................................... 25
  4.1 Prim’s Algorithm ................................................. 29
    4.1.1 Preservation of Invariant ................................... 30
    4.1.2 Exchange gives Spanning Trees ............................. 31
    4.1.3 Exchange gives Minimum Spanning Trees ................. 33
    4.1.4 Invariant implies Postcondition ............................ 36
  4.2 Kruskal’s Algorithm ............................................. 37
    4.2.1 Preservation of Invariant ................................... 37
    4.2.2 Exchange gives Spanning Trees ............................. 38
    4.2.3 Exchange gives Minimum Spanning Trees ................. 39
  4.3 Related Structures .............................................. 40
5 Subalgebras of Kleene Relation Algebras ....................... 40
1 Synopsis and Motivation

This document describes the following five theory files:

* Iterings describes a general iteration operation that works for many different computation models. We first consider equational axioms based on variants of Conway semirings. We expand these structures by generalised simulation axioms, which hold in total and general correctness models, not just in partial correctness models like the induction axioms. Simulation axioms are still powerful enough to prove separation theorems and Back’s atomicity refinement theorem [4].

* Kleene Algebras form a particular instance of iterings in which the iteration is implemented as a least fixpoint. We implement them based on Kozen’s axioms [13], but most results are inherited from Conway semirings and iterings.

* Kleene Relation Algebras introduces Stone-Kleene relation algebras, which combine Stone relation algebras and Kleene algebras. This is similar to relation algebras with transitive closure [16] but allows us to talk about reachability in weighted graphs. Many results in this theory are useful for verifying the correctness of Prim’s and Kruskal’s minimum spanning tree algorithms.

* Subalgebras of Kleene Relation Algebras studies the regular elements of a Stone-Kleene relation algebra and shows that they form a Kleene relation subalgebra.

* Matrix Kleene Algebras lifts the Kleene star to finite square matrices using Conway’s automata-based construction. This involves an operation to restrict matrices to specific indices and a calculus for such restrictions. An implementation for the Kleene star of matrices was given in [3] without proof; this is the first formally verified correctness proof.

The development is based on a theory of Stone relation algebras [11, 12]. We apply Stone-Kleene relation algebras to verify Prim’s minimum spanning tree algorithm in Isabelle/HOL in [10].

Related libraries for Kleene algebras, regular algebras and relation algebras in the Archive of Formal Proofs are [1, 2, 8]. Kleene algebras are covered in the theory Kleene_Algebra/Kleene_Algebra.thy, but unlike the
present development it is not based on general algebras using simulation axioms, which are useful to describe various computation models. The theory `Regular_Algebras/Regular_Algebras.thy` compares different axiomatisations of regular algebras. The theory `Kleene_Algebra/Matrix.thy` covers matrices over dioids, but does not implement the Kleene star of matrices. The theory `Relation_Algebra/Relation_Algebra_RTC.thy` combines Kleene algebras and relation algebras, but is very limited in scope and not applicable as we need the weaker axioms of Stone relation algebras.

## 2 Iterings

This theory introduces algebraic structures with an operation that describes iteration in various relational computation models. An iteration describes the repeated sequential execution of a computation. This is typically modelled by fixpoints, but different computation models use different fixpoints in the refinement order. We therefore look at equational and simulation axioms rather than induction axioms. Our development is based on [9] and the proposed algebras generalise Kleene algebras.

We first consider a variant of Conway semirings [5] based on idempotent left semirings. Conway semirings expand semirings by an iteration operation satisfying Conway’s sumstar and productstar axioms [7]. Many properties of iteration follow already from these equational axioms.

Next we introduce iterings, which use generalised versions of simulation axioms in addition to sumstar and productstar. Unlike the induction axioms of the Kleene star, which hold only in partial-correctness models, the simulation axioms are also valid in total and general correctness models. They are still powerful enough to prove the correctness of complex results such as separation theorems of [6] and Back’s atomicity refinement theorem [4, 17].

theory Iterings

imports Stone-Relation-Algebras.Semirings

begin

### 2.1 Conway Semirings

In this section, we consider equational axioms for iteration. The algebraic structures are based on idempotent left semirings, which are expanded by a unary iteration operation. We start with an unfold property, one inequality of the sliding rule and distributivity over joins, which is similar to Conway’s sumstar.

class circ =
fixes circ :: 'a ⇒ 'a (infixr 100)

theory Iterings

imports Stone-Relation-Algebras.Semirings

begin
class left-conway-semiring = idempotent-left-semiring + circ +
assumes circ-left-unfold: 1 ⊔ x * x^o = x^o
assumes circ-left-slide: (x * y)^o * x ≤ x * (y * x)^o
assumes circ-sup-1: (x ⊔ y)^o = x^o * (y * x^o)^o

begin

We obtain one inequality of Conway’s productstar, as well as of the other unfold rule.

lemma circ-mult-sub:
1 ⊔ x * (y * x)^o * y ≤ (x * y)^o
⟨proof⟩

lemma circ-right-unfold-sub:
1 ⊔ x^o * x ≤ x^o
⟨proof⟩

lemma circ-zero:
bot^o = 1
⟨proof⟩

lemma circ-increasing:
x ≤ x^o
⟨proof⟩

lemma circ-reflexive:
1 ≤ x^o
⟨proof⟩

lemma circ-mult-increasing:
x ≤ x * x^o
⟨proof⟩

lemma circ-mult-increasing-2:
x ≤ x^o * x
⟨proof⟩

lemma circ-transitive-equal:
x^o * x^o = x^o
⟨proof⟩

While iteration is not idempotent, a fixpoint is reached after applying this operation twice. Iteration is idempotent for the unit.

lemma circ-circ-circ:
x^{o o o} = x^{o o}
⟨proof⟩

lemma circ-one:
1^o = 1^{o o}
⟨proof⟩
lemma circ-sup-sub:
\((x^o * y)^o * x^o \leq (x \sqcup y)^o\)
(proof)

lemma circ-plus-one:
\(x^o = 1 \sqcup x^o\)
(proof)

Iteration satisfies a characteristic property of reflexive transitive closures.

lemma circ-rtc-2:
\(1 \sqcup x \sqcup x^o * x^o = x^o\)
(proof)

lemma mult-zero-circ:
\((x * \text{bot})^o = 1 \sqcup x * \text{bot}\)
(proof)

lemma mult-zero-sup-circ:
\((x \sqcup y * \text{bot})^o = x^o * (y * \text{bot})^o\)
(proof)

lemma circ-plus-sub:
\(x^o * x \leq x * x^o\)
(proof)

lemma circ-loop-fixpoint:
\(y * (y^o * z) \sqcup z = y^o * z\)
(proof)

lemma left-plus-below-circ:
\(x * x^o \leq x^o\)
(proof)

lemma right-plus-below-circ:
\(x^o * x \leq x^o\)
(proof)

lemma circ-sup-upper-bound:
\(x \leq z^o \implies y \leq z^o \implies x \sqcup y \leq z^o\)
(proof)

lemma circ-mult-upper-bound:
\(x \leq z^o \implies y \leq z^o \implies x * y \leq z^o\)
(proof)

lemma circ-sub-dist:
\(x^o \leq (x \sqcup y)^o\)
(proof)
lemma circ-sub-dist-1:
\[ x \leq (x \sqcup y)\circ \]
(proof)

lemma circ-sub-dist-2:
\[ x \ast y \leq (x \sqcup y)\circ \]
(proof)

lemma circ-sub-dist-3:
\[ x\circ \ast y \leq (x \sqcup y)\circ \]
(proof)

lemma circ-isotone:
\[ x \leq y \implies x\circ \leq y\circ \]
(proof)

lemma circ-sup-2:
\[ (x \sqcup y)\circ \leq (x\circ \ast y)\circ \]
(proof)

lemma circ-sup-one-left-unfold:
\[ 1 \leq x \implies x \ast x = x\circ \]
(proof)

lemma circ-sup-one-right-unfold:
\[ 1 \leq x \implies x\circ \ast x = x\circ \]
(proof)

lemma circ-decompose-4:
\[ (x\circ \ast y)\circ \circ = x\circ \circ \ast (y\circ \circ \ast x)\circ \]
(proof)

lemma circ-decompose-5:
\[ (x\circ \ast y)\circ = (y\circ \ast x)\circ \]
(proof)

lemma circ-decompose-6:
\[ x\circ \circ \ast (y \circ x)\circ = y\circ \circ \ast (x \circ y)\circ \]
(proof)

lemma circ-decompose-7:
\[ (x \sqcup y)\circ = x\circ \circ \ast y\circ \circ \ast (x \sqcup y)\circ \]
(proof)

lemma circ-decompose-8:
\[ (x \sqcup y)\circ = (x \sqcup y)\circ \ast x\circ \circ \ast y\circ \]
(proof)
**Lemma** circ-decompose-9:
\[(x \circ y)^\circ = x^\circ \circ y^\circ \circ (x^\circ \circ y^\circ)^\circ\]
\langle proof \rangle

**Lemma** circ-decompose-10:
\[(x \circ y)^\circ = (x^\circ \circ y^\circ)^\circ \circ x^\circ \circ y^\circ\]
\langle proof \rangle

**Lemma** circ-back-loop-prefixpoint:
\[(z \circ y) \circ y \sqcup z \leq z \circ y^\circ\]
\langle proof \rangle

We obtain the fixpoint and prefixpoint properties of iteration, but not least or greatest fixpoint properties.

**Lemma** circ-loop-is-fixpoint:
\[\text{is-fixpoint} (\lambda x \cdot y \circ x \sqcup z) (y^\circ \circ z)\]
\langle proof \rangle

**Lemma** circ-back-loop-is-prefixpoint:
\[\text{is-prefixpoint} (\lambda x \cdot x \circ y \sqcup z) (z \circ y^\circ)\]
\langle proof \rangle

**Lemma** circ-circ-sup:
\[(1 \sqcup x)^\circ = x^{\circ\circ}\]
\langle proof \rangle

**Lemma** circ-circ-mult-sub:
\[x^\circ \circ 1 \circ \leq x^\circ\]
\langle proof \rangle

**Lemma** left-plus-circ:
\[(x \circ x^\circ)^\circ = x^\circ\]
\langle proof \rangle

**Lemma** right-plus-circ:
\[(x^\circ \circ x)^\circ = x^\circ\]
\langle proof \rangle

**Lemma** circ-square:
\[(x \circ x)^\circ \leq x^9\]
\langle proof \rangle

**Lemma** circ-mult-sub-sup:
\[(x \circ y)^\circ \leq (x \sqcup y)^\circ\]
\langle proof \rangle

**Lemma** circ-sup-mult-zero:
\[x^\circ \circ y = (x \sqcup y \circ \bot)^\circ \circ y\]
\langle proof \rangle
lemma troeger-1:
\((x \sqcup y)^o = x^o \ast (I \sqcup y \ast (x \sqcup y)^o)\)
\(\langle proof\rangle\)

lemma troeger-2:
\((x \sqcup y)^o \ast z = x^o \ast (y \ast (x \sqcup y)^o \ast z \sqcup z)\)
\(\langle proof\rangle\)

lemma troeger-3:
\((x \sqcup y \ast \bot)^o = x^o \ast (I \sqcup y \ast \bot)\)
\(\langle proof\rangle\)

lemma circ-sup-sub-sup-one-1:
x \sqcup y \leq x^o \ast (I \sqcup y)
\(\langle proof\rangle\)

lemma circ-sup-sub-sup-one-2:
x^o \ast (x \sqcup y) \leq x^o \ast (I \sqcup y)
\(\langle proof\rangle\)

lemma circ-sup-sub-sup-one:
x \ast x^o \ast (x \sqcup y) \leq x \ast x^o \ast (I \sqcup y)
\(\langle proof\rangle\)

lemma circ-square-2:
\((x \ast x)^o \ast (x \sqcup 1) \leq x^o\)
\(\langle proof\rangle\)

lemma circ-extra-circ:
\((y \ast x^o)^o = (y \ast y^o \ast x^o)^o\)
\(\langle proof\rangle\)

lemma circ-circ-sub-mult:
\(1^o \ast x^o \leq x^{o^o}\)
\(\langle proof\rangle\)

lemma circ-decompose-11:
\((x^o \ast y^o)^o = (x^o \ast y^o)^o \ast x^o\)
\(\langle proof\rangle\)

lemma circ-mult-below-circ-circ:
\((x \ast y)^o \leq (x^o \ast y)^o \ast x^o\)
\(\langle proof\rangle\)

end

The next class considers the interaction of iteration with a greatest ele-
ment.

class bounded-left-conway-semiring = bounded-idempotent-left-semiring + left-conway-semiring
begin

lemma circ-top:
  top° = top
⟨proof⟩

lemma circ-right-top:
  x° * top = top
⟨proof⟩

lemma circ-left-top:
  top * x° = top
⟨proof⟩

lemma mult-top-circ:
  (x * top)° = 1 ∪ x * top
⟨proof⟩
end

class left-zero-conway-semiring = idempotent-left-zero-semiring + left-conway-semiring
begin

lemma mult-zero-sup-circ-2:
  (x ∪ y * bot)° = x° ∪ x° * y * bot
⟨proof⟩

lemma circ-unfold-sum:
  (x ∪ y)° = x° ∪ x° * y * (x ∪ y)°
⟨proof⟩
end

The next class assumes the full sliding equation.

class left-conway-semiring-1 = left-conway-semiring + assumes circ-right-slide: x * (y * x)° ≤ (x * y)° * x
begin

lemma circ-slide-1:
  x * (y * x)° = (x * y)° * x
⟨proof⟩

This implies the full unfold rules and Conway's productstar.

lemma circ-right-unfold-1:
  1 ∪ x° * x = x°
lemma circ-mult-1:
\[(x \cdot y)^0 = 1 \sqcup x \cdot (y \cdot x)^0 \cdot y\]
(proof)

lemma circ-sup-9:
\[(x \sqcup y)^0 = (x^0 \cdot y)^0 \cdot x^0\]
(proof)

lemma circ-plus-same:
\[x^0 \cdot x = x \cdot x^0\]
(proof)

lemma circ-decompose-12:
\[x^0 \cdot y^0 \leq (x^0 \cdot y)^0 \cdot x^0\]
(proof)

end

class left-zero-conway-semiring-1 = left-zero-conway-semiring +
left-conway-semiring-1
begin

lemma circ-back-loop-fixpoint:
\[(z \cdot y^0) \cdot y \sqcup z = z \cdot y^0\]
(proof)

lemma circ-back-loop-is-fixpoint:
\[\text{is-fixpoint} (\lambda x . x \cdot y \sqcup z) (z \cdot y^0)\]
(proof)

lemma circ-elimination:
\[x \cdot y = \text{bot} \implies x \cdot y^0 \leq x\]
(proof)

end

2.2 Iterings

This section adds simulation axioms to Conway semirings. We consider
several classes with increasingly general simulation axioms.

class itering-1 = left-conway-semiring-1 +
assumes circ-simulate: \[z \cdot x \leq y \cdot z \implies z \cdot x^0 \leq y^0 \cdot z\]
begin

lemma circ-circ-mult:
\[1^0 \cdot x^0 = x^{0^0}\]
(proof)
lemma sub-mult-one-circ:
\[ x \circ 1 = 1 \circ x \]
\langle proof \rangle

The left simulation axioms is enough to prove a basic import property of tests.

lemma circ-import:
\begin{align*}
& \text{assumes } p \leq p \circ p \\
& \quad \text{and } p \leq 1 \\
& \quad \text{and } p \circ x \leq x \circ p \\
& \text{shows } p \circ x^\circ = p \circ (p \circ x) \circ
\end{align*}
\langle proof \rangle

end

Including generalisations of both simulation axioms allows us to prove separation rules.

class itering-2 = left-conway-semiring-1 +
\begin{align*}
& \text{assumes } \text{circ-simulate-right: } z \circ x \leq y \circ z \sqcup w \rightarrow z \circ x^\circ \leq y^\circ \circ (z \sqcup w \circ x^\circ) \\
& \text{assumes } \text{circ-simulate-left: } x \circ z \leq z \circ y \sqcup w \rightarrow x^\circ \circ z \leq (z \sqcup x^\circ \circ w) \circ y^\circ
\end{align*}
begin

subclass itering-1
\langle proof \rangle

lemma circ-simulate-left-1:
\begin{align*}
x \circ z \leq z \circ y \Rightarrow x^\circ \circ z \leq z \circ y^\circ \sqcup x^\circ \circ \bot
\end{align*}
\langle proof \rangle

lemma circ-separate-1:
\begin{align*}
& \text{assumes } y \circ x \leq x \circ y \\
& \text{shows } (x \sqcup y)^\circ = x^\circ \circ y^\circ
\end{align*}
\langle proof \rangle

lemma circ-circ-mult-1:
\begin{align*}
x^\circ \circ 1^\circ = x^\circ \circ
\end{align*}
\langle proof \rangle

end

With distributivity, we also get Back’s atomicity refinement theorem.

class itering-3 = itering-2 + left-zero-conway-semiring-1
begin

lemma circ-simulate-1:
\begin{align*}
& \text{assumes } y \circ x \leq x \circ y \\
& \text{shows } y^\circ \circ x^\circ \leq x^\circ \circ y^\circ
\end{align*}
\langle proof \rangle

11
lemma atomicity-refinement:
assumes $s = s \ast q$
and $x = q \ast x$
and $q \ast b = \text{bot}$
and $r \ast b \leq b \ast r$
and $r \ast l \leq l \ast r$
and $x \ast l \leq l \ast x$
and $b \ast l \leq l \ast b$
and $q \ast l \leq l \ast q$
and $r^0 \ast q \leq q \ast r^0$
and $q \leq 1$
shows $s \ast (x \sqcup b \sqcup r \sqcup l)^\circ \ast q \leq s \ast (x \ast b^\circ \ast q \sqcup r \sqcup l)^\circ$
⟨proof⟩
end

The following class contains the most general simulation axioms we consider. They allow us to prove further separation properties.

class itering = idempotent-left-zero-semiring + circ +
assumes circ-sup: $(x \sqcup y)^\circ = (x^\circ \ast y)^\circ \ast x^\circ$
assumes circ-mult: $(x \ast y)^\circ = 1 \sqcup x \ast (y \ast x)^\circ \ast y$
assumes circ-simulate-right-plus: $z \ast x \leq y \ast y^0 \ast z \sqcup w \Longrightarrow z \ast x^\circ \leq y^0 \ast (z \sqcup w \ast x^\circ)$
assumes circ-simulate-left-plus: $x \ast z \leq z \ast y^0 \sqcup w \Longrightarrow x^\circ \ast z \leq (z \sqcup x^\circ \ast w) \ast y^0$
begin

lemma circ-right-unfold:
$1 \sqcup x^\circ \ast x = x^\circ$
⟨proof⟩

lemma circ-slide:
$x \ast (y \ast x)^\circ = (x \ast y)^\circ \ast x$
⟨proof⟩

subclass itering-3
⟨proof⟩

lemma circ-simulate-right-plus-1:
$z \ast x \leq y \ast y^0 \ast z \Longrightarrow z \ast x^\circ \leq y^0 \ast z$
⟨proof⟩

lemma circ-simulate-left-plus-1:
$x \ast z \leq z \ast y^0 \Longrightarrow x^\circ \ast z \leq z \ast y^0 \sqcup x^\circ \ast \text{bot}$
⟨proof⟩

lemma circ-simulate-2:
$y \ast x^\circ \leq x^\circ \ast y^0 \Longleftrightarrow y^0 \ast x^\circ \leq x^\circ \ast y^0$
lemma circ-simulate-absorb:
\[ y \leq x \Rightarrow y \leq x \sqcup y \]
\[ \]
lemma circ-separate-6:
y * x ≤ x * (x ⊔ y) → (x ⊔ y)₀ = x₀ * y₀
⟨proof⟩
end

class bounded-itering = bounded-idempotent-left-zero-semiring + itering
begin
subclass bounded-left-conway-semiring ⟨proof⟩
end

We finally expand Conway semirings and iterings by an element that

corresponds to the endless loop.

class L =
    fixes L :: 'a

class left-conway-semiring-L = left-conway-semiring + L +
    assumes one-circ-mult-split: 1₀ * x = L ⊔ x
    assumes L-split-sup: x * (y ⊔ L) ≤ x * y ⊔ L
begin
lemma L-def:
    L = 1₀ * bot
⟨proof⟩
lemma one-circ-split:
    1₀ = L ⊔ 1
⟨proof⟩
lemma one-circ-circ-split:
    1₀₀ = L ⊔ 1
⟨proof⟩
lemma sub-mult-one-circ:
    x * 1₀ ≤ 1₀ * x
⟨proof⟩
lemma one-circ-mult-split-2:
    1₀ * x = x * 1₀ ⊔ L
⟨proof⟩
lemma sub-mult-one-circ-split:
    x * 1₀ ≤ x ⊔ L
⟨proof⟩
lemma sub-mult-one-circ-split-2:
\[ x \ast 1^\lor \leq x \sqcup 1^\lor \]
⟨proof⟩

lemma L-split:
\[ x \ast L \leq x \ast \bot \sqcup L \]
⟨proof⟩

lemma L-left-zero:
\[ L \ast x = L \]
⟨proof⟩

lemma one-circ-L:
\[ 1^\lor \ast L = L \]
⟨proof⟩

lemma mult-L-circ:
\[ (x \ast L)^\lor = 1 \sqcup x \ast L \]
⟨proof⟩

lemma mult-L-circ-mult:
\[ (x \ast L)^\lor \ast y = y \sqcup x \ast L \]
⟨proof⟩

lemma circ-L:
\[ L^\lor = L \sqcup 1 \]
⟨proof⟩

lemma L-below-one-circ:
\[ L \leq 1^\lor \]
⟨proof⟩

lemma circ-circ-mult-1:
\[ x^\lor \ast 1^\lor = x^{\lor \lor} \]
⟨proof⟩

lemma circ-circ-mult:
\[ 1^\lor \ast x^\lor = x^{\lor \lor} \]
⟨proof⟩

lemma circ-circ-split:
\[ x^{\lor \lor} = L \sqcup x^\lor \]
⟨proof⟩

lemma circ-sup-6:
\[ L \sqcup (x \sqcup y)^\lor = (x^\lor \ast y^\lor)^\lor \]
⟨proof⟩

end
class iterating-L = iterating + L +
assumes L-def: L = 1° * bot
begin

lemma one-circ-split:
1° = L ⊔ 1
⟨proof⟩

lemma one-circ-mult-split:
1° * x = L ⊔ x
⟨proof⟩

lemma sub-mult-one-circ-split:
x * 1° ≤ x ⊔ L
⟨proof⟩

lemma sub-mult-one-circ-split-2:
x * 1° ≤ x ⊔ 1°
⟨proof⟩

lemma L-split:
x * L ≤ x * bot ⊔ L
⟨proof⟩

subclass left-conway-semiring-L
⟨proof⟩

lemma circ-left-induct-mult-L:
L ≤ x ⇒ x * y ≤ x ⇒ x * y° ≤ x
⟨proof⟩

lemma circ-left-induct-mult-iff-L:
L ≤ x ⇒ x * y ≤ x ←→ x * y° ≤ x
⟨proof⟩

lemma circ-left-induct-L:
L ≤ x ⇒ x * y ⊔ z ≤ x ⇒ z * y° ≤ x
⟨proof⟩

end

end

3 Kleene Algebras

Kleene algebras have been axiomatised by Kozen to describe the equational theory of regular languages [13]. Binary relations are another important
model. This theory implements variants of Kleene algebras based on idempotent left semirings [15]. The weakening of some semiring axioms allows the treatment of further computation models. The presented algebras are special cases of iterings, so many results can be inherited.

theory Kleene-Algebras

imports Iterings

begin

We start with left Kleene algebras, which use the left unfold and left induction axioms of Kleene algebras.

class star =
  fixes star :: 'a ⇒ 'a (** [100] 100)

class left-kleene-algebra = idempotent-left-semiring + star +
  assumes star-left-unfold : I ⊔ y * y* ≤ y*
  assumes star-left-induct : z ⊔ y * x ≤ x −→ y* * z ≤ x
begin

no-notation
  trancl ((-†) [1000] 999)

abbreviation tc (-† [100] 100) where tc x ≡ x * x*

lemma star-left-unfold-equal:
  I ⊔ x * x* = x*
  ⟨proof⟩

  This means that for some properties of Kleene algebras, only one inequality can be derived, as exemplified by the following sliding rule.

lemma star-left-slide:
  (x * y)* * x ≤ x * (y * x)*
  ⟨proof⟩

lemma star-isotone:
  x ≤ y −→ x* ≤ y*
  ⟨proof⟩

lemma star-sup-1:
  (x ⊔ y)* = x* * (y * x*)*
  ⟨proof⟩

end

We now show that left Kleene algebras form iterings. A sublocale is used instead of a subclass, because iterings use a different iteration operation.

sublocale left-kleene-algebra < star: left-conway-semiring where circ = star
context left-kleene-algebra begin

A number of lemmas in this class are taken from Georg Struth’s Kleene algebra theory [2].

lemma star-sub-one: 
\[ x \leq 1 \implies x^* = 1 \]  
(proof)

lemma star-one: 
\[ 1^* = 1 \]  
(proof)

lemma star-left-induct-mult: 
\[ x \cdot y \leq y \implies x^* \cdot y \leq y \]  
(proof)

lemma star-left-induct-mult-iff: 
\[ x \cdot y \leq y \iff x^* \cdot y \leq y \]  
(proof)

lemma star-involutive: 
\[ x^* = x^{**} \]  
(proof)

lemma star-sup-one: 
\[ (1 \sqcup x)^* = x^* \]  
(proof)

lemma star-left-induct-equal: 
\[ z \sqcup x \cdot y = y \implies x^* \cdot z \leq y \]  
(proof)

lemma star-left-induct-mult-equal: 
\[ x \cdot y = y \implies x^* \cdot y \leq y \]  
(proof)

lemma star-star-upper-bound: 
\[ x^* \leq z^* \implies x^{**} \leq z^* \]  
(proof)

lemma star-simulation-left: 
assumes \[ x \cdot z \leq z \cdot y \]
shows \[ x^* \cdot z \leq z \cdot y^* \]  
(proof)

lemma quasicomm-1:
\[ y \ast x \leq x \ast (x \uplus y)^* \iff y^* \ast x \leq x \ast (x \uplus y)^* \]

\textbf{lemma} \textit{star-rtc-3}:
\[ 1 \uplus x \uplus y \ast y = y \implies x^* \leq y \]
\textit{proof}

\textbf{lemma} \textit{star-decompose-1}:
\[ (x \uplus y)^* = (x^* \ast y^*)^* \]
\textit{proof}

\textbf{lemma} \textit{star-sum}:
\[ (x \uplus y)^* = (x^* \uplus y^*)^* \]
\textit{proof}

\textbf{lemma} \textit{star-decompose-3}:
\[ (x^* \ast y^*)^* = x^* \ast (y \ast x^*)^* \]
\textit{proof}

In contrast to iterings, we now obtain that the iteration operation results in least fixpoints.

\textbf{lemma} \textit{star-loop-least-fixpoint}:
\[ y \ast x \uplus z = x \implies y^* \ast z \leq x \]
\textit{proof}

\textbf{lemma} \textit{star-loop-is-least-fixpoint}:
\[ \text{is-least-fixpoint} (\lambda x \ . \ y \ast x \uplus z) (y^* \ast z) \]
\textit{proof}

\textbf{lemma} \textit{star-loop-mu}:
\[ \mu (\lambda x \ . \ y \ast x \uplus z) = y^* \ast z \]
\textit{proof}

\textbf{lemma} \textit{affine-has-least-fixpoint}:
\[ \text{has-least-fixpoint} (\lambda x \ . \ y \ast x \uplus z) \]
\textit{proof}

\textbf{lemma} \textit{star-outer-increasing}:
\[ x \leq y^* \ast x \ast y^* \]
\textit{proof}

end

We next add the right induction rule, which allows us to strengthen many inequalities of left Kleene algebras to equalities.

\textbf{class} \textit{strong-left-kleene-algebra} = \textit{left-kleene-algebra} +
\textbf{assumes} \textit{star-right-induct}:
\[ z \uplus x \ast y \leq x \implies z \ast y^* \leq x \]
begin

lemma star-plus:
\[ y^* \ast y = y \ast y^* \]
⟨proof⟩

lemma star-slide:
\[(x \ast y)^* \ast x = x \ast (y \ast x)^* \]
⟨proof⟩

lemma star-simulation-right:
assumes \[ z \ast x \leq y \ast z \]
shows \[ z \ast x^* \leq y^* \ast z \]
⟨proof⟩

end

Again we inherit results from the iterating hierarchy.

sublocale strong-left-kleene-algebra < star: iterating-1 where circ = star
⟨proof⟩

context strong-left-kleene-algebra
begin

lemma star-right-induct-mult:
\[ y \ast x \leq y \implies y \ast x^* \leq y \]
⟨proof⟩

lemma star-right-induct-mult-iff:
\[ y \ast x \leq y \iff y \ast x^* \leq y \]
⟨proof⟩

lemma star-simulation-right-equal:
\[ z \ast x = y \ast z \implies z \ast x^* = y^* \ast z \]
⟨proof⟩

lemma star-simulation-star:
\[ x \ast y \leq y \ast x \implies x^* \ast y^* \leq y^* \ast x^* \]
⟨proof⟩

lemma star-right-induct-equal:
\[ z \uplus y \ast x = y \implies z \ast x^* \leq y \]
⟨proof⟩

lemma star-right-induct-mult-equal:
\[ y \ast x = y \implies y \ast x^* \leq y \]
⟨proof⟩

lemma star-back-loop-least-fixpoint:
\[ x \star y \sqcup z = x \implies z \star y^* \leq x \]
\(\langle \text{proof} \rangle\)

**Lemma star-back-loop-is-least-fixpoint:**
\[ \text{is-least-fixpoint } (\lambda x . x \star y \sqcup z) (z \star y^*) \]
\(\langle \text{proof} \rangle\)

**Lemma star-back-loop-mu:**
\[ \mu (\lambda x . x \star y \sqcup z) = z \star y^* \]
\(\langle \text{proof} \rangle\)

**Lemma star-square:**
\[ x^* = (I \sqcup x) \star (x \star x)^* \]
\(\langle \text{proof} \rangle\)

**Lemma star-square-2:**
\[ x^* = (x \star x)^* \star (x \sqcup 1) \]
\(\langle \text{proof} \rangle\)

**Lemma star-circ-simulate-right-plus:**
\[ \text{assumes } z \star x \leq y \star y^* \star z \sqcup w \]
\[ \text{shows } z \star x^* \leq y^* \star (z \sqcup w \star x^*) \]
\(\langle \text{proof} \rangle\)

**Lemma transitive-star:**
\[ x \star x \leq x \implies x^* = I \sqcup x \]
\(\langle \text{proof} \rangle\)

**end**

The following class contains a generalisation of Kleene algebras, which lacks the right zero axiom.

**Class left-zero-kleene-algebra = idempotent-left-zero-semiring + strong-left-kleene-algebra**
\(\langle \text{proof} \rangle\)

**Lemma star-star-absorb:**
\[ y^* \star (y^* \star x)^* \star y^* = (y^* \star x)^* \star y^* \]
\(\langle \text{proof} \rangle\)

**Lemma star-circ-simulate-left-plus:**
\[ \text{assumes } x \star z \leq z \star y^* \sqcup w \]
\[ \text{shows } x^* \star z \leq (z \sqcup x^* \star w) \star y^* \]
\(\langle \text{proof} \rangle\)

**Lemma star-one-sup-below:**
\[ x \star y^* \star (I \sqcup z) \leq x \star (y \sqcup z)^* \]
proof

The following theorem is similar to the puzzle where persons insert themselves always in the middle between two groups of people in a line. Here, however, items in the middle annihilate each other, leaving just one group of items behind.

lemma cancel-separate:
assumes \( x \ast y \leq 1 \)
shows \( x^* \ast y^* \leq x^* \sqcup y^* \)

(proof)

lemma star-separate:
assumes \( x \ast y = \text{bot} \)
and \( y \ast y = \text{bot} \)
shows \( (x \sqcup y)^* = x^* \sqcup y \ast x^* \)

(proof)

end

We can now inherit from the strongest variant of iterings.

sublocale left-zero-kleene-algebra < star:itering where circ = star

(proof)

context left-zero-kleene-algebra
begin

lemma star-absorb:
\( x \ast y = \text{bot} \implies x \ast y^* = x \)

(proof)

lemma star-separate-2:
assumes \( x \ast z^+ \ast y = \text{bot} \)
and \( y \ast z^+ \ast y = \text{bot} \)
and \( z \ast x = \text{bot} \)
shows \( (x^* \sqcup y \ast x^*) \ast (z \ast (1 \sqcup y \ast x^*))^* = z^* \ast (x^* \sqcup y \ast x^*) \ast z^* \)

(proof)

lemma cancel-separate-eq:
\( x \ast y \leq 1 \implies x^* \ast y^* = x^* \sqcup y^* \)

(proof)

lemma cancel-separate-1:
assumes \( x \ast y \leq 1 \)
shows \( (x \sqcup y)^* = y^* \ast x^* \)

(proof)

lemma plus-sup:
\( (x \sqcup y)^+ = (x^* \ast y)^* \ast x^+ \sqcup (x^* \ast y)^+ \)

(proof)
lemma plus-half-bot:
\[ x \cdot y \cdot x = \bot \implies (x \cdot y)^+ = x \cdot y \]
(proof)

lemma cancel-separate-1-sup:
assumes \[x \cdot y \leq 1\]
and \[y \cdot x \leq 1\]
shows \[(x \sqcup y)^* = x^* \sqcup y^*\]
(proof)

end

A Kleene algebra is obtained by requiring an idempotent semiring.

class kleene-algebra = left-zero-kleene-algebra + idempotent-semiring

The following classes are variants of Kleene algebras expanded by an additional iteration operation. This is useful to study the Kleene star in computation models that do not use least fixpoints in the refinement order as the semantics of recursion.

class left-kleene-conway-semiring = left-kleene-algebra + left-conway-semiring
begin

lemma star-below-circ:
\[ x^* \leq x^0 \]
(proof)

lemma star-zero-below-circ-mult:
\[ x^* \cdot \bot \leq x^0 \cdot y \]
(proof)

lemma star-mult-circ:
\[ x^* \cdot x^0 = x^0 \]
(proof)

lemma circ-mult-star:
\[ x^0 \cdot x^* = x^0 \]
(proof)

lemma circ-star:
\[ x^{o*} = x^o \]
(proof)

lemma star-circ:
\[ x^{o*} = x^{oo} \]
(proof)

lemma circ-sup-3:
\[ (x^0 \cdot y^0)^* \leq (x \sqcup y)^o \]
\begin{proof}
\end{proof}

class left-zero-kleene-conway-semiring = left-zero-kleene-algebra + itering
begin
subclass left-kleene-conway-semiring (proof)

lemma circ-isolate:
\[ x^\circ = x^\circ \uplus x^\star \]
\begin{proof}
\end{proof}

lemma circ-isolate-mult:
\[ x^\circ \cdot y = x^\circ \uplus x^\star \cdot y \]
\begin{proof}
\end{proof}

lemma circ-isolate-mult-sub:
\[ x^\circ \cdot y \leq x^\circ \uplus x^\star \cdot y \]
\begin{proof}
\end{proof}

lemma circ-sub-decompose:
\[ (x^\circ \cdot y)^\circ \leq (x^\star \cdot y)^\circ \cdot x^\circ \]
\begin{proof}
\end{proof}

lemma circ-sup-4:
\[ (x \uplus y)^\circ = (x^\star \cdot y)^\circ \cdot x^\circ \]
\begin{proof}
\end{proof}

lemma circ-sup-5:
\[ (x^\circ \cdot y)^\circ \cdot x^\circ = (x^\star \cdot y)^\circ \cdot x^\circ \]
\begin{proof}
\end{proof}

lemma plus-circ:
\[ (x^\star \cdot x)^\circ = x^\circ \]
\begin{proof}
\end{proof}

end

The following classes add a greatest element.

class bounded-left-kleene-algebra = bounded-idempotent-left-semiring + left-kleene-algebra

sublocale bounded-left-kleene-algebra < star: bounded-left-conway-semiring
where circ = star (proof)

class bounded-left-zero-kleene-algebra = bounded-idempotent-left-semiring +
left-zero-kleene-algebra

sublocale bounded-left-zero-kleene-algebra < star: bounded-itering where circ = star ⟨proof⟩

class bounded-kleene-algebra = bounded-idempotent-semiring + kleene-algebra

sublocale bounded-kleene-algebra < star: bounded-itering where circ = star ⟨proof⟩

We conclude with an alternative axiomatisation of Kleene algebras.

class kleene-algebra-var = idempotent-semiring + star +
  assumes star-left-unfold-var : 1 ⊔ y * y* ≤ y*
  assumes star-left-induct-var : y * x ≤ x → y* * x ≤ x
  assumes star-right-induct-var : x * y ≤ x → x * y* ≤ x
begin

subclass kleene-algebra ⟨proof⟩

end

end

4 Kleene Relation Algebras

This theory combines Kleene algebras with Stone relation algebras. Relation algebras with transitive closure have been studied by [16]. The weakening to Stone relation algebras allows us to talk about reachability in weighted graphs, for example.

Many results in this theory are used in the correctness proof of Prim’s minimum spanning tree algorithm. In particular, they are concerned with the exchange property, preservation of parts of the invariant and with establishing parts of the postcondition.

theory Kleene-Relation-Algebras


begin

We first note that bounded distributive lattices can be expanded to Kleene algebras by reusing some of the operations.

sublocale bounded-distrib-lattice < comp-inf: bounded-kleene-algebra where star = λx . top and one = top and times = inf ⟨proof⟩
We add the Kleene star operation to each of bounded distributive allegories, pseudocomplemented distributive allegories and Stone relation algebras. We start with single-object bounded distributive allegories.

```plaintext
class bounded-distrib-kleene-allegory = bounded-distrib-allegory + kleene-algebra
begin
subclass bounded-kleene-algebra ⟨proof⟩

lemma conv-star-conv:
  x^* ≤ x^{T*T}
⟨proof⟩

It follows that star and converse commute.

lemma conv-star-commute:
  x^{T*} = x^T*
⟨proof⟩

lemma conv-plus-commute:
  x^{+T} = x^{T+}
⟨proof⟩

The following results are variants of a separation lemma of Kleene algebras.

lemma cancel-separate-2:
  assumes x * y ≤ 1
  shows ((w ∩ x) ∪ (z ∩ y))^* = (z ∩ y)^* * (w ∩ x)^*
⟨proof⟩

lemma cancel-separate-3:
  assumes x * y ≤ 1
  shows (w ∩ x)^* * (z ∩ y)^* = (w ∩ x)^* ∪ (z ∩ y)^*
⟨proof⟩

lemma cancel-separate-4:
  assumes z * y ≤ 1
  and w ≤ y ∪ z
  and x ≤ y ∪ z
  shows w^* * x^* = (w ∩ y)^* * ((w ∩ z)^* ∪ (x ∩ y)^*) * (x ∩ z)^*
⟨proof⟩

lemma cancel-separate-5:
  assumes w * z^T ≤ 1
  shows w ∩ x * (y ∩ z) ≤ y
⟨proof⟩

lemma cancel-separate-6:
  assumes z * y ≤ 1
  and w ≤ y ∪ z
  and x ≤ y ∪ z
```

26
and \( v \ast z^T \leq 1 \)
and \( v \sqcap y^* = \text{bot} \)
shows \( v \sqcap w^* \ast x^* \leq x \sqcup w \)
⟨proof⟩

We show several results about the interaction of vectors and the Kleene star.

**lemma** vector-star-1:

assumes vector \( x \)
shows \( x^T \ast (x \ast x^T)^* \leq x^T \)
⟨proof⟩

**lemma** vector-star-2:

vector \( x \implies x^T \ast (x \ast x^T)^* \leq x^T \ast \text{bot}^* \)
⟨proof⟩

**lemma** vector-vector-star:

vector \( v \implies (v \ast v^T)^* = I \sqcup v \ast v^T \)
⟨proof⟩

The following equivalence relation characterises the component trees of a forest. This is a special case of undirected reachability in a directed graph.

**abbreviation** forest-components \( f \equiv f^{T^*} \ast f^* \)

**lemma** forest-components-equivalence:

injective \( x \implies \text{equivalence} (\text{forest-components } x) \)
⟨proof⟩

**lemma** forest-components-increasing:

\( x \leq \text{forest-components } x \)
⟨proof⟩

**lemma** forest-components-isotone:

\( x \leq y \implies \text{forest-components } x \leq \text{forest-components } y \)
⟨proof⟩

**lemma** forest-components-idempotent:

injective \( x \implies \text{forest-components} (\text{forest-components } x) = \text{forest-components } x \)
⟨proof⟩

**lemma** forest-components-star:

injective \( x \implies (\text{forest-components } x)^* = \text{forest-components } x \)
⟨proof⟩

The following lemma shows that the nodes reachable in the graph can be reached by only using edges between reachable nodes.

**lemma** reachable-restrict:

assumes vector \( r \)
shows \( r^T \ast g^* = r^T \ast ((r^T \ast g^*)^T \ast (r^T \ast g^*) \sqcap g)^* \)
proof

lemma kruskal-acyclic-inv-1:
  assumes injective f
  and e * forest-components f * e = bot
  shows (f ∩ top * e * f^T) * f * e = bot
  ⟨proof⟩

lemma kruskal-forest-components-inf-1:
  assumes f ≤ w⊔w^T
  and injective w
  and f ≤ forest-components g
  shows f * forest-components (forest-components g ∩ w) ≤ forest-components (forest-components g ∩ w)
  ⟨proof⟩

lemma kruskal-forest-components-inf:
  assumes f ≤ w⊔w^T
  and injective w
  shows forest-components f ≤ forest-components (forest-components f ⊓ w)
  ⟨proof⟩

end

We next add the Kleene star to single-object pseudocomplemented distributive allegories.

class pd-kleene-allegory = pd-allegory + bounded-distrib-kleene-allegory
begin

  The following definitions and results concern acyclic graphs and forests.

abbreviation acyclic :: 'a ⇒ bool where acyclic x ≡ x^+ ≤ −1

abbreviation forest :: 'a ⇒ bool where forest x ≡ injective x ∧ acyclic x

lemma forest-bot:
  forest bot
  ⟨proof⟩

lemma acyclic-star-below-complement:
  acyclic w ⟷ w^T ≤ −w
  ⟨proof⟩

lemma acyclic-star-below-complement-1:
  acyclic w ⟷ w^* ∩ w^T = bot
  ⟨proof⟩

lemma acyclic-star-inf-conv:
  assumes acyclic w
  shows w^* ∩ w^T = 1

28
**Proof**

**Lemma acyclic-asymmetric:**
acyclic \( w \Rightarrow \) asymmetric \( w \)

**Proof**

**Lemma forest-separate:**
assumes forest \( x \)
sows \( x^* \star x^T \star \cap x^T \star x \leq 1 \)

**Proof**

The following definition captures the components of undirected weighted graphs.

**Abbreviation** components \( g \equiv (-g)^* \)

**Lemma components-equivalence:**
symmetric \( x \Rightarrow \) equivalence (components \( x \))

**Proof**

**Lemma components-increasing:**
ex \( \leq \) components \( x \)

**Proof**

**Lemma components-isotone:**
ex \( \leq y \Rightarrow \) components \( x \leq \) components \( y \)

**Proof**

**Lemma cut-reachable:**
assumes \( v^T = r^T \star t^* \)
and \( t \leq g \)
sows \( v \star -v^T \cap g \leq (r^T \star g^*)^T \star (r^T \star g^*) \)

**Proof**

The following lemma shows that the predecessors of visited nodes in the minimum spanning tree extending the current tree have all been visited.

**Lemma predecessors-reachable:**
assumes vector \( r \)
and injective \( r \)
and \( v^T = r^T \star t^* \)
and forest \( w \)
and \( t \leq w \)
and \( w \leq (r^T \star g^*)^T \star (r^T \star g^*) \cap g \)
and \( r^T \star g^* \leq r^T \star w^* \)
sows \( w \star v \leq v \)

**Proof**

4.1 Prim’s Algorithm
The following results are used for proving the correctness of Prim's minimum spanning tree algorithm.

4.1.1 Preservation of Invariant

We first treat the preservation of the invariant. The following lemma shows that the while-loop preserves that $v$ represents the nodes of the constructed tree. The remaining lemmas in this section show that $t$ is a spanning tree. The exchange property is treated in the following two sections.

**lemma reachable-inv:**

**assumes** vector $v$

and $e \leq v \ast -v^T$

and $e \ast t = \bot$

and $v^T = r^T \ast t^*$

**shows** $(v \sqcup e^T \ast top)^T = r^T \ast (t \sqcup e)^*$

\(\langle proof \rangle\)

The next result is used to show that the while-loop preserves acyclicity of the constructed tree.

**lemma acyclic-inv:**

**assumes** acyclic $t$

and vector $v$

and $e \leq v \ast -v^T$

and $t \leq v \ast v^T$

**shows** acyclic $(t \sqcup e)$

\(\langle proof \rangle\)

The following lemma shows that the extended tree is in the component reachable from the root.

**lemma mst-subgraph-inv-2:**

**assumes** regular $(v \ast v^T)$

and $t \leq v \ast v^T \sqcap --g$

and $v^T = r^T \ast t^*$

and $e \leq v \ast -v^T \sqcap --g$

and vector $v$

and regular $(v \sqcup e^T \ast top) \ast (v \sqcup e^T \ast top)^T$

**shows** $t \sqcup e \leq (r^T \ast (---((v \sqcup e^T \ast top) \ast (v \sqcup e^T \ast top)^T \sqcap g))^T \ast (r^T \ast$$(---((v \sqcup e^T \ast top) \ast (v \sqcup e^T \ast top)^T \sqcap g))^*)\ast$

\(\langle proof \rangle\)

**lemma span-inv:**

**assumes** $e \leq v \ast -v^T$

and vector $v$

and arc $e$

and $t \leq (v \ast v^T) \sqcap g$

and $g^T = g$

and $v^T = r^T \ast t^*$

and injective $r$
\[ r^T \leq v^T \]
\[ r^T \ast ((v \ast v^T) \cap g)^* \leq r^T \ast t^* \]
shows \[ r^T \ast ((v \cup e^T \ast \text{top}) \ast (v \cup e^T \ast \text{top})^T) \cap g)^* \leq r^T \ast (t \cup e)^* \]

The lemmas in this section are used to show that the relation after exchange represents a spanning tree. The results in the next section are used to show that it is a minimum spanning tree.

**4.1.2 Exchange gives Spanning Trees**

The following abbreviations are used in the spanning tree application using Prim’s algorithm to construct the new tree for the exchange property. It is obtained by replacing an edge with one that has minimal weight and reversing the path connecting these edges. Here, \( w \) represents a weighted graph, \( v \) represents a set of nodes and \( e \) represents an edge.

**abbreviation** \( \text{prim-E} :: 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a \text{ where prim-E } w \ v \ e \equiv w \ominus v^T \ominus \text{top}^* e \ominus w e^T \)\[\]

**abbreviation** \( \text{prim-P} :: 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a \text{ where prim-P } w \ v \ e \equiv w \ominus v^T \ominus \text{top}^* e \ominus w e^T \)\[\]

**abbreviation** \( \text{prim-EP} :: 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a \text{ where prim-EP } w \ v \ e \equiv w \ominus v^T \ominus \text{top}^* e \ominus w e^T \)\[\]

**abbreviation** \( \text{prim-W} :: 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a \text{ where prim-W } w \ v \ e \equiv (w \ominus -(\text{prim-EP } w \ v \ e)) \cup (\text{prim-P } w \ v \ e)^T \cup e \)\[\]

The graphs after exchanging are injective.

**lemma** \( \text{exchange-injective-3:} \)

**assumes** \( e \leq v \ast -v^T \)

**and** vector \( v \)

**shows** \( (w \ominus -(\text{prim-EP } w \ v \ e))^T \ast e^T = \text{bot} \)

**proof**

**lemma** \( \text{exchange-injective-6:} \)

**assumes** arc \( e \)

**and** forest \( w \)

**shows** \( (\text{prim-P } w \ v \ e)^T \ast e^T = \text{bot} \)

**proof**

The graph after exchanging is injective.

**lemma** \( \text{exchange-injective:} \)

**assumes** arc \( e \)

**and** \( e \leq v \ast -v^T \)

**and** forest \( w \)

**and** vector \( v \)

**shows** injective \( (\text{prim-W } w \ v \ e) \)

**proof**

**lemma** \( \text{pv:} \)

**assumes** vector \( v \)

**shows** \( (\text{prim-P } w \ v \ e)^T \ast v = \text{bot} \)
lemma vector-pred-inv:
assumes arc e
and \( e \leq v \ast -v^T \)
and forest \( w \)
and vector \( v \)
and \( w \ast v \leq v \)
shows \((\text{prim-W } w \ast v \ast e) \ast (v \sqcup e^T \ast \text{top}) \leq v \sqcup e^T \ast \text{top}\)

(proof)

The graph after exchanging is acyclic.

lemma exchange-acyclic:
assumes vector \( v \)
and \( e \leq v \ast -v^T \)
and \( w \ast v \leq v \)
and acyclic \( w \)
shows acyclic \((\text{prim-W } w \ast v \ast e)\)

(proof)

The following lemma shows that an edge across the cut between visited nodes and unvisited nodes does not leave the component of visited nodes.

lemma mst-subgraph-inv:
assumes \( e \leq v \ast -v^T \sqcap g \)
and \( t \leq g \)
and \( v^T = r^T \ast t^* \)
shows \( e \leq (r^T \ast g^*)^T \ast (r^T \ast g^*) \sqcap g \)

(proof)

The following lemmas show that the tree after exchanging contains the currently constructed and tree and its extension by the chosen edge.

lemma mst-extends-old-tree:
assumes \( t \leq w \)
and \( t \leq v \ast v^T \)
and vector \( v \)
shows \( t \leq \text{prim-W } w \ast v \ast e \)

(proof)

lemma mst-extends-new-tree:
\( t \leq w \Rightarrow t \leq v \ast v^T \Rightarrow \text{vector } v \Rightarrow t \sqcup e \leq \text{prim-W } w \ast v \ast e \)

(proof)

end

We finally add the Kleene star to Stone relation algebras. Kleene star and the relational operations are reasonably independent. The only additional axiom we need in the generalisation to Stone-Kleene relation algebras is that star distributes over double complement.
\textbf{class} stone-kleene-relation-algebra = stone-relation-algebra + pd-kleene-allegory

\begin{verbatim}
  \textbf{assumes} pp-dist-star: \(\neg\neg(x^*) = (\neg\neg x)^*\)
\end{verbatim}

\textbf{begin}

\textbf{lemma} regular-closed-star:
\[
\text{regular } x \implies \text{regular } (x^*)
\]

\textbf{lemma} components-idempotent:
\[
\text{components (components } x) = \text{components } x
\]

\textbf{proof}\ 

The following lemma shows that the nodes reachable in the tree after exchange contain the nodes reachable in the tree before exchange.

\textbf{lemma} mst-reachable-inv:
\begin{verbatim}
  \textbf{assumes} regular (prim-EP \(w v e\))
  and \textbf{vector } r
  and \textbf{vector } v
  and \textbf{vector } u
  and \textbf{vector } t
  and \textbf{vector } w
  and \textbf{vector } v
  and \textbf{vector } u
  \textbf{and } t \leq w
  \textbf{and } t \leq u
  \textbf{and } w * v \leq u
  \textbf{shows } r^T * w^* \leq r^T * (prim-W \(w v e\))^*
\end{verbatim}

\textbf{proof}\ 

Some of the following lemmas already hold in pseudocomplemented distributive Kleene allegories.

\subsection{Exchange gives Minimum Spanning Trees}

The lemmas in this section are used to show that the after exchange we obtain a minimum spanning tree. The following lemmas show various interactions between the three constituents of the tree after exchange.

\textbf{lemma} epm-1:
\[
\text{vector } v \implies \text{prim-E } w v e \sqcup \text{prim-P } w v e = \text{prim-EP } w v e
\]

\textbf{lemma} epm-2:
\begin{verbatim}
  \textbf{assumes} regular (prim-EP \(w v e\))
  \textbf{and } vector v
  \textbf{shows } (w \sqcap \neg(prim-EP \(w v e\))) \sqcup prim-P \(w v e\) = w
\end{verbatim}

\textbf{lemma} epm-4:
\begin{verbatim}
  \textbf{assumes} e \leq w
  \textbf{and } injective w
\end{verbatim}
\(\text{and } w \ast v \leq v\)
\(\text{and } e \leq v \ast -v^T\)
\(\text{shows } \top \ast e \ast w^T \leq \top \ast v^T\)

\langle proof \rangle

lemma epm-5:
\text{assumes } e \leq w
\text{and } \text{injective } w
\text{and } w \ast v \leq v
\text{and } e \leq v \ast -v^T
\text{and } \text{vector } v
\text{shows } \text{prim-P } w \ast v \ast e = \text{bot}

\langle proof \rangle

lemma epm-6:
\text{assumes } e \leq w
\text{and } \text{injective } w
\text{and } w \ast v \leq v
\text{and } e \leq v \ast -v^T
\text{and } \text{vector } v
\text{shows } \text{prim-E } w \ast v \ast e = e

\langle proof \rangle

lemma epm-7:
\text{regular (prim-EP } w \ast v \ast e) \Rightarrow e \leq w \Rightarrow \text{injective } w \Rightarrow w \ast v \leq v \Rightarrow e \leq v \ast -v^T \Rightarrow \text{vector } v \Rightarrow \text{prim-W } w \ast v \ast e = w

\langle proof \rangle

lemma epm-8:
\text{assumes acyclic } w
\text{shows } (w \cap (\neg (\text{prim-EP } w \ast v \ast e))) \cap (\text{prim-P } w \ast v \ast e)^T = \text{bot}

\langle proof \rangle

lemma epm-9:
\text{assumes } e \leq v \ast -v^T
\text{and } \text{vector } v
\text{shows } (w \cap (\neg (\text{prim-EP } w \ast v \ast e))) \cap e = \text{bot}

\langle proof \rangle

lemma epm-10:
\text{assumes } e \leq v \ast -v^T
\text{and } \text{vector } v
\text{shows } (\text{prim-P } w \ast v \ast e)^T \cap e = \text{bot}

\langle proof \rangle

lemma epm-11:
\text{assumes } \text{vector } v
\text{shows } (w \cap (\neg (\text{prim-EP } w \ast v \ast e))) \cap \text{prim-P } w \ast v \ast e = \text{bot}

\langle proof \rangle

34
lemma \textit{epm-12}:
assumes vector v
shows \((w \cap -(\text{prim-EP } w v e)) \cap \text{prim-E } w v e = \bot)$$
\langle proof \rangle

lemma \textit{epm-13}:
assumes vector v
shows \(\text{prim-P } w v e \cap \text{prim-E } w v e = \bot)$$
\langle proof \rangle

The following lemmas show that the relation characterising the edge across the cut is an arc.

lemma \textit{arc-edge-1}:
assumes \(e \leq v \ast -v^T \cap g)
and vector v
and \(v^T = r^T \ast t^\ast\)
and \(t \leq g\)
and \(r^T \ast g^\ast \leq r^T \ast w^\ast\)
shows \(\text{top } \ast e \leq v^T \ast w^\ast\)
\langle proof \rangle

lemma \textit{arc-edge-2}:
assumes \(e \leq v \ast -v^T \cap g)
and vector v
and \(v^T = r^T \ast t^\ast\)
and \(t \leq g\)
and \(r^T \ast g^\ast \leq r^T \ast w^\ast\)
and \(w \ast v \leq v\)
and injective w
shows \(\text{top } \ast e \ast w^{T \ast} \leq v^T \ast w^\ast\)
\langle proof \rangle

lemma \textit{arc-edge-3}:
assumes \(e \leq v \ast -v^T \cap g)
and vector v
and \(v^T = r^T \ast t^\ast\)
and \(t \leq g\)
and \(r^T \ast g^\ast \leq r^T \ast w^\ast\)
and \(w \ast v \leq v\)
and injective w
and \(\text{prim-E } w v e = \bot\)
shows \(e = \bot\)
\langle proof \rangle

lemma \textit{arc-edge-4}:
assumes \(e \leq v \ast -v^T \cap g)
and vector v
and \(v^T = r^T \ast t^\ast\)
and \( t \leq g \)
and \( r^T \ast g^* \leq r^T \ast w^* \)
and arc e
shows \( \top \ast \text{prim-E } w \ e \ast \top = \top \)
\begin{proof}
\end{proof}

**lemma arc-edge-5:**

**assumes** vector v
and \( w \ast v \leq v \)
and injective w
and arc e
shows \( (\text{prim-E } w \ v \ e)^T \ast \top \ast \text{prim-E } w \ v \ e \leq 1 \)
\begin{proof}
\end{proof}

**lemma arc-edge-6:**

**assumes** vector v
and \( w \ast v \leq v \)
and injective w
and arc e
shows \( \text{prim-E } w \ v \ e \ast \top \ast (\text{prim-E } w \ v \ e)^T \leq 1 \)
\begin{proof}
\end{proof}

**lemma arc-edge:**

**assumes** \( e \leq v \ast -v^T \cap g \)
and vector v
and \( v^T = r^T \ast t^* \)
and \( t \leq g \)
and \( r^T \ast g^* \leq r^T \ast w^* \)
and \( w \ast v \leq v \)
and injective w
and arc e
shows \( \text{arc } (\text{prim-E } w \ v \ e) \)
\begin{proof}
\end{proof}

4.1.4 Invariant implies Postcondition

The lemmas in this section are used to show that the invariant implies the postcondition at the end of the algorithm. The following lemma shows that the nodes reachable in the graph are the same as those reachable in the constructed tree.

**lemma span-post:**

**assumes** regular v
and vector v
and \( v^T = r^T \ast t^* \)
and \( v \ast -v^T \cap g = \text{bot} \)
and \( t \leq v \ast v^T \cap g \)
and \( r^T \ast (v \ast v^T \cap g)^* \leq r^T \ast t^* \)
shows \( v^T = r^T \ast g^* \)
\begin{proof}
\end{proof}
The following lemma shows that the minimum spanning tree extending a tree is the same as the tree at the end of the algorithm.

**Lemma mst-post:**
- ** Assumes vector r 
  - and injective r 
  - and $v^T = r^T * t^r$ 
  - and forest w 
  - and $t \leq w$ 
  - and $w \leq v * v^T$
- ** Shows $w = t$**

(Proof)

### 4.2 Kruskal’s Algorithm

The following results are used for proving the correctness of Kruskal’s minimum spanning tree algorithm.

#### 4.2.1 Preservation of Invariant

We first treat the preservation of the invariant. The following lemmas show conditions necessary for preserving that $f$ is a forest.

**Lemma kruskal-injective-inv-2:**
- ** Assumes arc e 
  - and acyclic f 
- ** Shows $top * e * f^{T*} * f^T \leq -e$**

(Proof)

**Lemma kruskal-injective-inv-3:**
- ** Assumes arc e 
  - and forest f 
- ** Shows $(top * e * f^{T*})^T * (top * e * f^{T*}) \cap f^T * f \leq 1$**

(Proof)

**Lemma kruskal-acyclic-inv:**
- ** Assumes acyclic f 
  - and covector q 
  - and $(f \cap q)^T * f^* * e = bot$ 
  - and $e * f^* * e = bot$ 
  - and $f^{T*} * f^* \leq -e$ 
- ** Shows acyclic $(f \cap q) \cup (f \cap q)^T \cup e$**

(Proof)

**Lemma kruskal-exchange-acyclic-inv-1:**
- ** Assumes acyclic f 
  - and covector q 
- ** Shows acyclic $(f \cap q) \cup (f \cap q)^T$**

(Proof)
lemma kruskal-exchange-acyclic-inv-2:
assumes acyclic \( w \)
and injective \( w \)
and \( d \leq w \)
and bijective \( (d^T \ast \top) \)
and bijective \( (e \ast \top) \)
and \( d \leq \top \ast e^T \ast w^T \)
and \( w \ast e^T \ast \top = \bot \)
shows acyclic \( ((w \cap -d) \sqcup e) \)
\langle proof \rangle

4.2.2 Exchange gives Spanning Trees
The lemmas in this section are used to show that the relation after exchange represents a spanning tree.

lemma inf-star-import:
assumes \( x \leq z \)
and univalent \( z \)
and reflexive \( y \)
and regular \( z \)
shows \( x \ast y \sqcap z \ast \leq x \ast (y \sqcap z) \)
\langle proof \rangle

lemma kruskal-exchange-forest-components-inv:
assumes injective \( ((w \cap -d) \sqcup e) \)
and regular \( d \)
and \( e \ast \top \ast e = e \)
and \( d \leq \top \ast e^T \ast w^T \)
and \( w \ast e^T \ast \top = \bot \)
and injective \( w \)
and \( d \leq w \)
and \( d \leq (w \cap -d)^T \ast e^T \ast \top \)
sshows forest-components \( w \leq \) forest-components \( ((w \cap -d) \sqcup e) \)
\langle proof \rangle

lemma kruskal-spanning-inv:
assumes injective \( ((f \cap -q) \sqcup (f \cap q)^T \sqcup e) \)
and regular \( q \)
and regular \( e \)
and \((-h \cap -g)^* \leq \) forest-components \( f \)
sshows components \((h \cap -e \cap -e^T) \cap q) \leq \) forest-components \((f \cap -q) \sqcup (f \cap q)^T \sqcup e) \)
\langle proof \rangle

lemma kruskal-exchange-spanning-inv-1:
assumes injective \((w \cap -q) \sqcup (w \cap q)^T\)
and regular \((w \cap q)\)
and components \( q \leq \) forest-components \( w \)
sshows components \( g \leq \) forest-components \((w \cap -q) \sqcup (w \cap q)^T)\)
\textit{proof}

\textbf{lemma} \textit{kruskal-exchange-spanning-inv-2}:
\begin{itemize}
\item \textbf{assumes} injective $w$
\item and $w^* \cdot e^T = e^T$
\item and $f \sqcup f^T \leq (w \cap -d \cap -d^T) \sqcup (w^T \cap -d \cap -d^T)$
\item and $d \leq \text{forest-components} f \cdot e^T \cdot \text{top}$
\item \textbf{shows} $d \leq (w \cap -d)^T \cdot e^T \cdot \text{top}$
\end{itemize}
\textit{proof}

\textbf{lemma} \textit{kruskal-spanning-inv-1}:
\begin{itemize}
\item \textbf{assumes} $e \leq F$
\item and regular $e$
\item and components $(-h \cap g) \leq F$
\item and equivalence $F$
\item \textbf{shows} components $(-(h \cap -e \cap -e^T) \cap g) \leq F$
\end{itemize}
\textit{proof}

\textbf{lemma} \textit{kruskal-reroot-edge}:
\begin{itemize}
\item \textbf{assumes} injective $(e^T \cdot \text{top})$
\item and acyclic $w$
\item \textbf{shows} $(w \cap -(top \cdot e \cdot w^{T*})) \sqcup (w \cap \text{top} \cdot e \cdot w^{T*})^T \cdot e^T = \text{bot}$
\end{itemize}
\textit{proof}

\textbf{4.2.3 Exchange gives Minimum Spanning Trees}

The lemmas in this section are used to show that the after exchange we obtain a minimum spanning tree. The following lemmas show that the relation characterising the edge across the cut is an arc.

\textbf{lemma} \textit{kruskal-edge-arc}:
\begin{itemize}
\item \textbf{assumes} equivalence $F$
\item and forest $w$
\item and arc $e$
\item and regular $F$
\item and $F \leq \text{forest-components} (F \cap w)$
\item and regular $w$
\item and $w \cdot e^T = \text{bot}$
\item and $e \cdot F \cdot e = \text{bot}$
\item and $e^T \leq w^*$
\item \textbf{shows} arc $((w \cap top \cdot e^T \cdot w^{T*}) \cap (w \cap \text{top} \cdot e \cdot w^{T*})^T) \cdot e^T = \text{bot}$
\end{itemize}
\textit{proof}

\textbf{lemma} \textit{kruskal-edge-arc-1}:
\begin{itemize}
\item \textbf{assumes} $e \leq -h$
\item and $h \leq g$
\item and symmetric $h$
\item and components $g \leq \text{forest-components} w$
\item and $w \cdot e^T = \text{bot}$
\item \textbf{shows} $e^T \leq w^*$
\end{itemize}
\textit{proof}
proof

lemma kruskal-edge-between-components-1:
assumes equivalence F
and mapping (top * e)
shows F \leq -(w \cap top * e^T * w^{T*} \cap F * e^T * top \cap top * e * -F)
(proof)

lemma kruskal-edge-between-components-2:
assumes forest-components f \leq -d
and injective f
and f \sqcup f^T \leq w \sqcup w^T
shows f \sqcup f^T \leq (w \cap -d \cap -d^T) \sqcup (w^T \cap -d \cap -d^T)
(proof)

4.3 Related Structures

Stone algebras can be expanded to Stone-Kleene relation algebras by reusing some operations.

sublocale stone-algebra < comp-inf: stone-kleene-relation-algebra where star = \lambda x. top and one = top and times = inf and conv = id
(proof)

Every bounded linear order can be expanded to a Stone algebra, which can be expanded to a Stone relation algebra, which can be expanded to a Stone-Kleene relation algebra.

class linorder-stone-kleene-relation-algebra-expansion =
linorder-stone-relation-algebra-expansion + star +
assumes star-def [simp]: x^* = top
begin

subclass kleene-algebra
(proof)

subclass stone-kleene-relation-algebra
(proof)

end

A Kleene relation algebra is based on a relation algebra.

class kleene-relation-algebra = relation-algebra + stone-kleene-relation-algebra
end

5 Subalgebras of Kleene Relation Algebras
In this theory we show that the regular elements of a Stone-Kleene relation algebra form a Kleene relation subalgebra.

theory Kleene-Relation-Subalgebras

imports Stone-Relation-Algebras Relation-Subalgebras Kleene-Relation-Algebras

begin

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

lift-definition star-regular :: 'a regular ⇒ 'a regular is star ⟨proof⟩

instance ⟨proof⟩

end

end

6 Matrix Kleene Algebras

This theory gives a matrix model of Stone-Kleene relation algebras. The main result is that matrices over Kleene algebras form Kleene algebras. The automata-based construction is due to Conway [7]. An implementation of the construction in Isabelle/HOL that extends [2] was given in [3] without a correctness proof.

For specifying the size of matrices, Isabelle/HOL’s type system requires the use of types, not sets. This creates two issues when trying to implement Conway’s recursive construction directly. First, the matrix size changes for recursive calls, which requires dependent types. Second, some submatrices used in the construction are not square, which requires typed Kleene algebras [14], that is, categories of Kleene algebras.

Because these instruments are not available in Isabelle/HOL, we use square matrices with a constant size given by the argument of the Kleene star operation. Smaller, possibly rectangular submatrices are identified by two lists of indices: one for the rows to include and one for the columns to include. Lists are used to make recursive calls deterministic; otherwise sets would be sufficient.

theory Matrix-Kleene-Algebras


begin
6.1 Matrix Restrictions

In this section we develop a calculus of matrix restrictions. The restriction of a matrix to specific row and column indices is implemented by the following function, which keeps the size of the matrix and sets all unused entries to \texttt{bot}.

**definition** \texttt{restrict-matrix} :: \texttt{\textquotesingle a list} \Rightarrow \texttt{\textquotesingle a,\textquotesingle b::bot} square \Rightarrow \texttt{\textquotesingle a list} \Rightarrow \texttt{\textquotesingle a,\textquotesingle b} square (- (-) - [90,41,90] 91)

where \texttt{restrict-matrix as f bs} = (\lambda (i,j). if List.member as i \land List.member bs j then f (i,j) else bot)

The following function captures Conway’s automata-based construction of the Kleene star of a matrix. An index \( k \) is chosen and \( s \) contains all other indices. The matrix is split into four submatrices \( a, b, c, d \) including/not including row/column \( k \). Four matrices are computed containing the entries given by Conway’s construction. These four matrices are added to obtain the result. All matrices involved in the function have the same size, but matrix restriction is used to set irrelevant entries to \texttt{bot}.

**primrec** \texttt{star-matrix} :: \texttt{\textquotesingle a list} \Rightarrow \texttt{\textquotesingle a,\textquotesingle b::\{star,times,bounded-semilattice-sup-bot\}} square \Rightarrow \texttt{\textquotesingle a,\textquotesingle b} square where

\texttt{star-matrix Nil g} = \texttt{mbot} | \texttt{star-matrix \textquotesingle k#s} g = ( let \( r = [k] \) in let \( a = r(g)r \) in let \( b = r(g)s \) in let \( c = s(g)r \) in let \( d = s(g)s \) in let \( as = r\star a\star r \) in let \( ds = star-matrix\textquotesingle s d \) in let \( e = a \ominus b \ominus ds \ominus c \) in let \( es = r\star e\star r \) in let \( f = d \ominus c \ominus as \ominus b \) in let \( fs = star-matrix\textquotesingle s f \) in \( es \oplus as \ominus b \ominus fs \oplus ds \ominus c \ominus es \ominus fs \))

The Kleene star of the whole matrix is obtained by taking as indices all elements of the underlying type \texttt{\textquotesingle a}. This is conveniently supplied by the \texttt{enum} class.

**fun** \texttt{star-matrix} :: \texttt{\textquotesingle a::enum,\textquotesingle b::\{star,times,bounded-semilattice-sup-bot\}} square \Rightarrow \texttt{\textquotesingle a,\textquotesingle b} square \texttt{\{\ominus [100] 100\}} where \texttt{star-matrix f = star-matrix\textquotesingle\textquotesingle (enum-class enum::\textquotesingle a list) f}

The following lemmas deconstruct matrices with non-empty restrictions.

**lemma** \texttt{restrict-empty-left:} \texttt{\{f\}ls = mbot} \texttt{\langle proof\rangle}
lemma restrict-empty-right:
ks(f)[] = mbot

lemma restrict-nonempty-left:
fixes f :: ('a,'b::bounded-semilattice-sup-bot) square
shows (k#ks)(f)ls = [k](f)ls ⊕ ks(f)ls

lemma restrict-nonempty-right:
fixes f :: ('a,'b::bounded-semilattice-sup-bot) square
shows ks(f)(l#ls) = ks(f)[l] ⊕ ks(f)ls

The following predicate captures that two index sets are disjoint. This has consequences for composition and the unit matrix.

abbreviation disjoint ks ls ≡ ¬(∃x . List.member ks x ∧ List.member ls x)

lemma times-disjoint:
fixes f g :: ('a,'b::idempotent-semiring) square
assumes disjoint ls ms
shows ks(f)ls ⊙ ms(g)ns = mbot

lemma one-disjoint:
assumes disjoint ks ls
shows ks((none::('a,'b::idempotent-semiring) square))ls = mbot

The following predicate captures that an index set is a subset of another index set. This has consequences for repeated restrictions.

abbreviation is-sublist ks ls ≡ ∀x . List.member ks x → List.member ls x

lemma restrict-sublist:
assumes is-sublist ls ks
and is-sublist ms ns
shows ls(k)(f)(ns)ms = ls(f)ms

lemma restrict-superlist:
assumes is-sublist ls ks
and is-sublist ms ns
shows ks(ls(f)(ms)ns)ms = ls(f)ms

43
The following lemmas give the sizes of the results of some matrix operations.

**Lemma restrict-sup:**

```plaintext
fixes f g :: ('a,'b::bounded-semilattice-sup-bot) square
shows ks(f ⊕ g)ls = ks(f)ls ⊕ ks(g)ls
(proof)
```

**Lemma restrict-times:**

```plaintext
fixes f g :: ('a,'b::idempotent-semiring) square
shows ks(ks(f)ls ⊙ ls(g)ms)ms = ks(f)ls ⊙ ls(g)ms
(proof)
```

**Lemma restrict-star:**

```plaintext
fixes g :: ('a,'b::kleene-algebra) square
shows t(star-matrix') t g = star-matrix' t g
(proof)
```

**Lemma restrict-one:**

```plaintext
assumes ¬ List.member ks k
shows (k # ks)⟨ mone ::('a::finite,'b::idempotent-semiring) square ⟩ (k # ks) = [k](mone)[k] ⊕ ks(mone)ks
(proof)
```

**Lemma restrict-one-left-unit:**

```plaintext
ks((mone::('a::finite,'b::idempotent-semiring) square))ks ⊙ ks(f)ls = ks(f)ls
(proof)
```

The following lemmas consider restrictions to singleton index sets.

**Lemma restrict-singleton:**

```plaintext
([k]⟨ f ⟩[l]) (i,j) = (if i = k ∧ j = l then f (i,j) else bot)
(proof)
```

**Lemma restrict-singleton-list:**

```plaintext
([k]⟨ f ⟩ls) (i,j) = (if i = k ∧ List.member ls j then f (i,j) else bot)
(proof)
```

**Lemma restrict-list-singleton:**

```plaintext
(ks⟨ f ⟩[l]) (i,j) = (if List.member ks i ∧ j = l then f (i,j) else bot)
(proof)
```

**Lemma restrict-singleton-product:**

```plaintext
fixes f g :: ('a::finite,'b::kleene-algebra) square
shows ([k]⟨ f ⟩[l] ⊙ [m]⟨ g ⟩[n]) (i,j) = (if i = k ∧ l = m ∧ j = n then f (i,l) * g (m,j) else bot)
(proof)
```

The Kleene star unfold law holds for matrices with a single entry on the diagonal.

**Lemma restrict-star-unfold:**

```plaintext
```
\[ ([l](\text{monoe}::\langle a::\text{finite}, b::\text{kleene-algebra} \rangle \text{ square}))[[l] \oplus [l](f)[l] \circ [l](\text{star} \circ f)[l] = [l](\text{star} \circ f)[l] \]

\langle \text{proof} \rangle

\textbf{lemma restrict-all:}
\textit{enum-class enum(f) enum-class enum} = f
\langle \text{proof} \rangle

The following shows the various components of a matrix product. It is essentially a recursive implementation of the product.

\textbf{lemma restrict-nonempty-product:}
\textit{fixes f g :: \langle a::\text{finite}, b::\text{idempotent-semiring} \rangle \text{ square}
assumes \sim \text{ List.member} \text{ ls l}
shows \langle k\#ks \rangle(f)(l\#ls) \circ (l\#ls)(g)(m\#ms) = ([k]f)[l] \circ [l](g)[m] \oplus \langle k \rangle(k)f ls \circ \langle l \rangle(g)ms \oplus \langle k \rangle(g)ls \circ \langle l \rangle(g)ms \oplus \langle k\#ks \rangle l\#ls \circ \langle l \rangle(g)ms
\langle \text{proof} \rangle

Equality of matrices is componentwise.

\textbf{lemma restrict-nonempty-eq:}
\langle k\#ks \rangle(f)(l\#ls) = \langle k\#ks \rangle(g)(l\#ls) \iff \langle [k\#ks]f \rangle(l\#ls) = \langle [k\#ks]g \rangle(l\#ls) \iff \langle [k\#ks]f \rangle(l\#ls) \ls \langle [k\#ks]g \rangle(l\#ls)
\langle \text{proof} \rangle

Inequality of matrices is componentwise.

\textbf{lemma restrict-nonempty-less-eq:}
\textit{fixes f g :: \langle a::\text{finite}, b::\text{idempotent-semiring} \rangle \text{ square}
shows \langle k\#ks \rangle(f)(l\#ls) \ls \langle k\#ks \rangle(g)(l\#ls) \iff \langle [k\#ks]f \rangle(l\#ls) \leq \langle [k\#ks]g \rangle(l\#ls) \iff \langle [k\#ks]f \rangle(l\#ls) \leq \langle [k\#ks]g \rangle(l\#ls)
\langle \text{proof} \rangle

The following lemmas treat repeated restrictions to disjoint index sets.

\textbf{lemma restrict-disjoint-left:}
\textit{assumes disjoint ks ms
shows ms(\langle k\#ks \rangle l\#ls) ns = mbot}
\langle \text{proof} \rangle

\textbf{lemma restrict-disjoint-right:}
\textit{assumes disjoint ls ns
shows ms(\langle k\#ks \rangle l\#ls) ns = mbot}
\langle \text{proof} \rangle

The following lemma expresses the equality of a matrix and a product of two matrices componentwise.

\textbf{lemma restrict-nonempty-product-eq:}
\textit{fixes f g h :: \langle a::\text{finite}, b::\text{idempotent-semiring} \rangle \text{ square}
assumes \sim \text{ List.member} \text{ ks k}
and \sim \text{ List.member} \text{ ls l}
and \sim \text{ List.member} \text{ ms m}
\langle \text{proof} \rangle

45
shows \((k\#ks)[f](l\#ls)(g\#ms) = (k\#ks)[h](m\#ms) \iff \)
\[\small{k}[f][l] \odot [l][g][m] \odot [k][f][ls] \odot ls(g)[m] = [k][h][m] \land [k][f][l] \odot [l][g][ms] \land [k][f][ls] \odot ls(g)[m] =
ks(h)[m] \land ks(f)[l] \odot [l][g][ms] \odot ks(f)[ls] \odot ls(g)[ms] = ks(h)[ms]\]
\(\langle \text{proof} \rangle\)

The following lemma gives a componentwise characterisation of the inequality of a matrix and a product of two matrices.

**Lemma** restrict-nonempty-product-less-eq:
fixes \(f\ g\ h\) :: \(\langle 'a\::\text{finite}, 'b\::\text{idempotent-semiring} \rangle\) square
assumes \(\neg \text{List.member}\ ks\ k\)
and \(\neg \text{List.member}\ ls\ l\)
and \(\neg \text{List.member}\ ms\ m\)
shows \((k\#ks)[f](l\#ls)(g\#ms) \leq (k\#ks)[h](m\#ms) \iff \)
\[\small{k}[f][l] \odot [l][g][ms] \leq [k][f][l] \odot [l][g][ms] \leq ks(h)[ms] \land ks(f)[l] \odot [l][g][ms] \leq ks(h)[ms] \land ks(f)[ls] \odot ls(g)[ms] \leq ks(h)[ms]\]
\(\langle \text{proof} \rangle\)

The Kleene star induction laws hold for matrices with a single entry on the diagonal. The matrix \(g\) can actually contain a whole row/column at the appropriate index.

**Lemma** restrict-star-left-induct:
fixes \(f\ g\) :: \(\langle 'a\::\text{finite}, 'b\::\text{kleene-algebra} \rangle\) square
shows distinct \(ms\) \implies \([l][f][l] \odot [l][g][ms] \leq [l][g][ms] \implies [l][\text{star} o f][l] \odot [l][g][ms] = [l][g][ms]\)
\(\langle \text{proof} \rangle\)

**Lemma** restrict-star-right-induct:
fixes \(f\ g\) :: \(\langle 'a\::\text{finite}, 'b\::\text{kleene-algebra} \rangle\) square
shows distinct \(ms\) \implies \(ms(g)[l] \odot [l][f][l] \leq ms(g)[l] \implies ms(g)[l] \odot [l][\text{star} o f][l] \leq ms(g)[l]\)
\(\langle \text{proof} \rangle\)

**Lemma** restrict-pp:
fixes \(f\) :: \(\langle 'a, 'b\::\text{p-algebra} \rangle\) square
shows \(ks(\bullet o f)[ls] = \bullet o (ks(f))[ls]\)
\(\langle \text{proof} \rangle\)

**Lemma** pp-star-commute:
fixes \(f\) :: \(\langle 'a, 'b\::\text{stone-kleene-relation-algebra} \rangle\) square
shows \(\bullet o (\text{star} o f) = \text{star} o \bullet o f\)
\(\langle \text{proof} \rangle\)

### 6.2 Matrices form a Kleene Algebra

Matrices over Kleene algebras form a Kleene algebra using Conway’s construction. It remains to prove one unfold and two induction axioms of the
Kleene star. Each proof is by induction over the size of the matrix represented by an index list.

interpretation matrix-kleene-algebra: kleene-algebra-var where sup = sup-matrix and less-eq = less-eq-matrix and less = less-matrix and bot = bot-matrix:('a::enum,'b::kleene-algebra) square and one = one-matrix and times = times-matrix and star = star-matrix

⟨proof⟩

6.3 Matrices form a Stone-Kleene Relation Algebra

Matrices over Stone-Kleene relation algebras form a Stone-Kleene relation algebra. It remains to prove the axiom about the interaction of Kleene star and double complement.

interpretation matrix-stone-kleene-relation-algebra: stone-kleene-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::enum,'b::stone-kleene-relation-algebra) square and top = top-matrix and uminus = uminus-matrix and one = one-matrix and times = times-matrix and conv = conv-matrix and star = star-matrix

⟨proof⟩

References


