Kleene Algebra

Alasdair Armstrong, Victor B. F. Gomes, Georg Struth and Tjark Weber

May 21, 2019

Abstract

Variants of Dioids and Kleene algebras are formalised together with their most important models in Isabelle/HOL. The Kleene algebras presented include process algebras based on bisimulation equivalence (near Kleene algebras), simulation equivalence (pre-Kleene algebras) and language equivalence (Kleene algebras), as well as algebras with ambiguous finite or infinite iteration (Conway algebras), possibly infinite iteration (demonic refinement algebras), infinite iteration (omega algebras) and residuated variants (action algebras). Models implemented include binary relations, (regular) languages, sets of paths and traces, power series and matrices. Finally, min-plus and max-plus algebras as well as generalised Hoare logics for Kleene algebras and demonic refinement algebras are provided for applications.

Contents

1 Introductory Remarks

These theory files are intended as a reference formalisation of variants of Kleene algebras and as a basis for other variants, such as Kleene algebras with tests [2] and modal Kleene algebras [14], which are useful for program correctness and verification. To that end we have aimed at making proof accessible to readers at textbook granularity instead of fully automating them. In that sense, these files can be considered a machine-checked introduction to reasoning in Kleene algebra.

Beyond that, the theories are only sparsely commented. Additional information on the hierarchy of Kleene algebras and its formalisation in Isabelle/HOL can be found in a tutorial paper [13] or an overview article [17]. While these papers focus on the automation of algebraic reasoning, the present formalisation presents readable proofs whenever these are interesting and instructive.

Expansions of the hierarchy to modal Kleene algebras, Kleene algebras with tests and Hoare logics as well as infinitary and higher-order Kleene algebras [16, 3], and an alternative hierarchy of regular algebras and Kleene algebras [11]—orthogonal to the present one—have also been implemented in the Archive of Formal Proofs [12, 14, 2, 1].

2 Signatures

theory *Signatures*
imports *Main*
begin

Default notation in Isabelle/HOL is occasionally different from established notation in the relation/algebra community. We use the latter where possible.

notation
times (**infixl** · *70*)

Some classes in our algebraic hierarchy are most naturally defined as subclasses of two (or more) superclasses that impose different restrictions on the same parameter(s).

Alas, in Isabelle/HOL, a class cannot have multiple superclasses that independently declare the same parameter(s). One workaround, which motivated the following syntactic classes, is to shift the parameter declaration to a common superclass.

class *star-op* =
fixes *star* :: $'a \Rightarrow 'a$ ($\text{-}^{\star}$ [*101*] *100*)

class *omega-op* =
fixes *omega* :: $'a \Rightarrow 'a$ (-^{ω} [*101*] *100*)

We define a type class that combines addition and the definition of order in, e.g., semilattices. This class makes the definition of various other type classes more slick.

class *plus-ord* = *plus* + *ord* +
assumes *less-eq-def*: $x \leq y \longleftrightarrow x + y = y$
and *less-def*: $x < y \longleftrightarrow x \leq y \wedge x \neq y$

end

3 Dioids

theory *Dioid*
imports *Signatures*
begin

3.1 Join Semilattices

Join semilattices can be axiomatised order-theoretically or algebraically. A join semilattice (or upper semilattice) is either a poset in which every pair of elements has a join (or least upper bound), or a set endowed with an associative, commutative, idempotent binary operation. It is well known that the order-theoretic definition induces the algebraic one and vice versa. We start from the algebraic axiomatisation because it is easily expandable to dioids, using Isabelle's type class mechanism.

In Isabelle/HOL, a type class *semilattice-sup* is available. Alas, we cannot use this type class because we need the symbol + for the join operation in the dioid expansion and subclass proofs in Isabelle/HOL require the two type classes involved to have the same fixed signature.

Using *add_assoc* as a name for the first assumption in class *join_semilattice* would lead to name clashes: we will later define classes that inherit from *semigroup-add*, which provides its own assumption *add_assoc*, and prove that these are subclasses of *join_semilattice*. Hence the primed name.

class *join-semilattice* = *plus-ord* +
  assumes *add-assoc'* [*ac-simps*]: $(x + y) + z = x + (y + z)$
  and *add-comm* [*ac-simps*] : $x + y = y + x$
  and *add-idem* [*simp*]: $x + x = x$
begin

lemma *add-left-comm* [*ac-simps*]: $y + (x + z) = x + (y + z)$
  using *local.add-assoc' local.add-comm* **by** *auto*

lemma *add-left-idem* [*ac-simps*]: $x + (x + y) = x + y$
  unfolding *add-assoc'* [*symmetric*] **by** *simp*

The definition $(x \leq y) = (x + y = y)$ of the order is hidden in class *plus-ord*.

We show some simple order-based properties of semilattices. The first one states that every semilattice is a partial order.

subclass *order*
proof
  fix $x\ y\ z :: 'a$
  show $x < y \longleftrightarrow x \leq y \wedge \neg\ y \leq x$
    using *local.add-comm local.less-def local.less-eq-def* **by** *force*
  show $x \leq x$
    by (*simp add*: *local.less-eq-def*)
  show $x \leq y \Longrightarrow y \leq z \Longrightarrow x \leq z$
    by (*metis add-assoc' less-eq-def*)
  show $x \leq y \Longrightarrow y \leq x \Longrightarrow x = y$
    by (*simp add*: *local.add-comm local.less-eq-def*)
qed

Next we show that joins are least upper bounds.

sublocale *join*: *semilattice-sup* (+)
  by (*unfold-locales*; *simp add*: *ac-simps local.less-eq-def*)

Next we prove that joins are isotone (order preserving).

lemma *add-iso*: $x \leq y \Longrightarrow x + z \leq y + z$
  using *join.sup-mono* **by** *blast*

The next lemma links the definition of order as $(x \leq y) = (x + y = y)$ with a perhaps more conventional one known, e.g., from arithmetics.

lemma *order-prop*: $x \leq y \longleftrightarrow (\exists z.\ x + z = y)$
proof
  assume $x \leq y$
  hence $x + y = y$
    by (*simp add*: *less-eq-def*)
  thus $\exists z.\ x + z = y$
    by *auto*
next
  assume $\exists z.\ x + z = y$
  then obtain c **where** $x + c = y$
    by *auto*
  hence $x + c \leq y$
    by *simp*
  thus $x \leq y$
    by *simp*
qed

end

3.2 Join Semilattices with an Additive Unit

We now expand join semilattices by an additive unit 0. Is the least element with respect to the order, and therefore often denoted by ⊥. Semilattices with a least element are often called *bounded*.

class *join-semilattice-zero* = *join-semilattice* + *zero* +
  assumes *add-zero-l* [*simp*]: $0 + x = x$

begin

subclass *comm-monoid-add*
  by (*unfold-locales*, *simp-all add*: *add-assoc'*) (*simp add*: *add-comm*)

sublocale *join*: *bounded-semilattice-sup-bot* (+) (≤) (<) *0*
  by *unfold-locales* (*simp add*: *local.order-prop*)

lemma *no-trivial-inverse*: $x \neq 0 \Longrightarrow \neg(\exists y.\ x + y = 0)$
  by (*metis local.add-0-right local.join.sup-left-idem*)

end

3.3 Near Semirings

Near semirings (also called seminearrings) are generalisations of near rings to the semiring case. They have been studied, for instance, in G. Pilz's book [25] on near rings. According to his definition, a near semiring consists of an additive and a multiplicative semigroup that interact via a single distributivity law (left or right). The additive semigroup is not required to be commutative. The definition is influenced by partial transformation semigroups.

We only consider near semirings in which addition is commutative, and in which the right distributivity law holds. We call such near semirings *abelian*.

class *ab-near-semiring* = *ab-semigroup-add* + *semigroup-mult* +
assumes *distrib-right'* [*simp*]: $(x + y) \cdot z = x \cdot z + y \cdot z$

subclass (**in** *semiring*) *ab-near-semiring*
by (*unfold-locales*, *metis distrib-right*)

class *ab-pre-semiring* = *ab-near-semiring* +
assumes *subdistl-eq*: $z \cdot x + z \cdot (x + y) = z \cdot (x + y)$

3.4 Variants of Dioids

A *near dioid* is an abelian near semiring in which addition is idempotent. This generalises the notion of (additively) idempotent semirings by dropping one distributivity law. Near dioids are a starting point for process algebras.

By modelling variants of dioids as variants of semirings in which addition is idempotent we follow the tradition of Birkhoff [5], but deviate from the definitions in Gondran and Minoux's book [15].

class *near-dioid* = *ab-near-semiring* + *plus-ord* +
assumes *add-idem'* [*simp*]: $x + x = x$

begin

Since addition is idempotent, the additive (commutative) semigroup reduct of a near dioid is a semilattice. Near dioids are therefore ordered by the semilattice order.

subclass *join-semilattice*
by *unfold-locales* (*auto simp add*: *add.commute add.left-commute*)

It follows that multiplication is right-isotone (but not necessarily left-isotone).

lemma *mult-isor*: $x \le y \Longrightarrow x \cdot z \le y \cdot z$
proof −
assume $x \le y$
hence $x + y = y$
by (*simp add*: *less-eq-def*)
hence $(x + y) \cdot z = y \cdot z$

by *simp*
thus $x \cdot z \leq y \cdot z$
by (*simp add*: *less-eq-def*)
qed

lemma $x \leq y \Longrightarrow z \cdot x \leq z \cdot y$

oops

The next lemma states that, in every near dioid, left isotonicity and left subdistributivity are equivalent.

lemma *mult-isol-equiv-subdistl*:
$(\forall x\ y\ z.\ x \leq y \longrightarrow z \cdot x \leq z \cdot y) \longleftrightarrow (\forall x\ y\ z.\ z \cdot x \leq z \cdot (x + y))$
by (*metis local.join.sup-absorb2 local.join.sup-ge1*)

The following lemma is relevant to propositional Hoare logic.

lemma *phl-cons1*: $x \leq w \Longrightarrow w \cdot y \leq y \cdot z \Longrightarrow x \cdot y \leq y \cdot z$
using *dual-order.trans mult-isor* **by** *blast*

end

We now make multiplication in near dioids left isotone, which is equivalent to left subdistributivity, as we have seen. The corresponding structures form the basis of probabilistic Kleene algebras [24] and game algebras [29]. We are not aware that these structures have a special name, so we baptise them *pre-dioids*.

We do not explicitly define pre-semirings since we have no application for them.

class *pre-dioid* = *near-dioid* +
assumes *subdistl*: $z \cdot x \leq z \cdot (x + y)$

begin

Now, obviously, left isotonicity follows from left subdistributivity.

lemma *subdistl-var*: $z \cdot x + z \cdot y \leq z \cdot (x + y)$
using *local.mult-isol-equiv-subdistl local.subdistl* **by** *auto*

subclass *ab-pre-semiring*
apply *unfold-locales*
by (*simp add*: *local.join.sup-absorb2 local.subdistl*)

lemma *mult-isol*: $x \leq y \Longrightarrow z \cdot x \leq z \cdot y$
proof −
assume $x \leq y$
hence $x + y = y$
by (*simp add*: *less-eq-def*)
also have $z \cdot x + z \cdot y \leq z \cdot (x + y)$

using *subdistl-var* **by** *blast*
moreover have ... $= z \cdot y$
by (*simp add*: *calculation*)
ultimately show $z \cdot x \leq z \cdot y$
by *auto*
qed

lemma *mult-isol-var*: $u \leq x \Longrightarrow v \leq y \Longrightarrow u \cdot v \leq x \cdot y$
by (*meson local.dual-order.trans local.mult-isor mult-isol*)

lemma *mult-double-iso*: $x \leq y \Longrightarrow w \cdot x \cdot z \leq w \cdot y \cdot z$
by (*simp add*: *local.mult-isor mult-isol*)

The following lemmas are relevant to propositional Hoare logic.

lemma *phl-cons2*: $w \leq x \Longrightarrow z \cdot y \leq y \cdot w \Longrightarrow z \cdot y \leq y \cdot x$
using *local.order-trans mult-isol* **by** *blast*

lemma *phl-seq*:
assumes $p \cdot x \leq x \cdot r$
and $r \cdot y \leq y \cdot q$
shows $p \cdot (x \cdot y) \leq x \cdot y \cdot q$
proof −
have $p \cdot x \cdot y \leq x \cdot r \cdot y$
using *assms*(*1*) *mult-isor* **by** *blast*
thus *?thesis*
by (*metis assms*(*2*) *order-prop order-trans subdistl mult-assoc*)
qed

lemma *phl-cond*:
assumes $u \cdot v \leq v \cdot u \cdot v$ **and** $u \cdot w \leq w \cdot u \cdot w$
and $\bigwedge x\ y.\ u \cdot (x + y) \leq u \cdot x + u \cdot y$
and $u \cdot v \cdot x \leq x \cdot z$ **and** $u \cdot w \cdot y \leq y \cdot z$
shows $u \cdot (v \cdot x + w \cdot y) \leq (v \cdot x + w \cdot y) \cdot z$
proof −
have *a*: $u \cdot v \cdot x \leq v \cdot x \cdot z$ **and** *b*: $u \cdot w \cdot y \leq w \cdot y \cdot z$
by (*metis assms mult-assoc phl-seq*)+
have $u \cdot (v \cdot x + w \cdot y) \leq u \cdot v \cdot x + u \cdot w \cdot y$
using *assms*(*3*) *mult-assoc* **by** *auto*
also have ... $\leq v \cdot x \cdot z + w \cdot y \cdot z$
using *a b join.sup-mono* **by** *blast*
finally show *?thesis*
by *simp*
qed

lemma *phl-export1*:
assumes $x \cdot y \leq y \cdot x \cdot y$
and $(x \cdot y) \cdot z \leq z \cdot w$
shows $x \cdot (y \cdot z) \leq (y \cdot z) \cdot w$
proof −

have $x \cdot y \cdot z \leq y \cdot x \cdot y \cdot z$
by (*simp add*: *assms*(*1*) *mult-isor*)
thus *?thesis*
using *assms*(*1*) *assms*(*2*) *mult-assoc phl-seq* **by** *auto*
qed

lemma *phl-export2*:
assumes $z \cdot w \leq w \cdot z \cdot w$
and $x \cdot y \leq y \cdot z$
shows $x \cdot (y \cdot w) \leq y \cdot w \cdot (z \cdot w)$
proof −
have $x \cdot y \cdot w \leq y \cdot z \cdot w$
using *assms*(*2*) *mult-isor* **by** *blast*
thus *?thesis*
by (*metis assms*(*1*) *dual-order.trans order-prop subdistl mult-assoc*)
qed

end

By adding a full left distributivity law we obtain semirings (which are already available in Isabelle/HOL as *semiring*) from near semirings, and dioids from near dioids. Dioids are therefore idempotent semirings.

class *dioid* = *near-dioid* + *semiring*

subclass (**in** *dioid*) *pre-dioid*
by *unfold-locales* (*simp add*: *local.distrib-left*)

3.5 Families of Nearsemirings with a Multiplicative Unit

Multiplicative units are important, for instance, for defining an operation of finite iteration or Kleene star on dioids. We do not introduce left and right units separately since we have no application for this.

class *ab-near-semiring-one* = *ab-near-semiring* + *one* +
assumes *mult-onel* [*simp*]: $1 \cdot x = x$
and *mult-oner* [*simp*]: $x \cdot 1 = x$

begin

subclass *monoid-mult*
by (*unfold-locales*, *simp-all*)

end

class *ab-pre-semiring-one* = *ab-near-semiring-one* + *ab-pre-semiring*

class *near-dioid-one* = *near-dioid* + *ab-near-semiring-one*

begin

The following lemma is relevant to propositional Hoare logic.

lemma *phl-skip*: $x \cdot 1 \leq 1 \cdot x$
  by *simp*

end

For near dioids with one, it would be sufficient to require $1 + 1 = 1$. This implies $x + x = x$ for arbitray x (but that would lead to annoying redundant proof obligations in mutual subclasses of *near-dioid-one* and *near-dioid* later).

class *pre-dioid-one = pre-dioid + near-dioid-one*

class *dioid-one = dioid + near-dioid-one*

subclass (**in** *dioid-one*) *pre-dioid-one* **..**

3.6 Families of Nearsemirings with Additive Units

We now axiomatise an additive unit 0 for nearsemirings. The zero is usually required to satisfy annihilation properties with respect to multiplication. Due to applications we distinguish a zero which is only a left annihilator from one that is also a right annihilator. More briefly, we call zero either a left unit or a unit.

Semirings and dioids with a right zero only can be obtained from those with a left unit by duality.

class *ab-near-semiring-one-zerol = ab-near-semiring-one + zero +*
  assumes *add-zerol* [*simp*]: $0 + x = x$
  and *annil* [*simp*]: $0 \cdot x = 0$

begin

Note that we do not require $0 \neq 1$.

lemma *add-zeror* [*simp*]: $x + 0 = x$
  by (*subst add-commute*) *simp*

end

class *ab-pre-semiring-one-zerol = ab-near-semiring-one-zerol + ab-pre-semiring*

begin

The following lemma shows that there is no point defining pre-semirings separately from dioids.

lemma $1 + 1 = 1$
proof –
  have $1 + 1 = 1 \cdot 1 + 1 \cdot (1 + 0)$

by *simp*
also have ... = *1* · (*1* + *0*)
using *subdistl-eq* **by** *presburger*
finally show *?thesis*
by *simp*
qed

end

class *near-dioid-one-zerol* = *near-dioid-one* + *ab-near-semiring-one-zerol*

subclass (**in** *near-dioid-one-zerol*) *join-semilattice-zero*
by (*unfold-locales*, *simp*)

class *pre-dioid-one-zerol* = *pre-dioid-one* + *ab-near-semiring-one-zerol*

subclass (**in** *pre-dioid-one-zerol*) *near-dioid-one-zerol* **..**

class *semiring-one-zerol* = *semiring* + *ab-near-semiring-one-zerol*

class *dioid-one-zerol* = *dioid-one* + *ab-near-semiring-one-zerol*

subclass (**in** *dioid-one-zerol*) *pre-dioid-one-zerol* **..**

We now make zero also a right annihilator.

class *ab-near-semiring-one-zero* = *ab-near-semiring-one-zerol* +
assumes *annir* [*simp*]: *x* · *0* = *0*

class *semiring-one-zero* = *semiring* + *ab-near-semiring-one-zero*

class *near-dioid-one-zero* = *near-dioid-one-zerol* + *ab-near-semiring-one-zero*

class *pre-dioid-one-zero* = *pre-dioid-one-zerol* + *ab-near-semiring-one-zero*

subclass (**in** *pre-dioid-one-zero*) *near-dioid-one-zero* **..**

class *dioid-one-zero* = *dioid-one-zerol* + *ab-near-semiring-one-zero*

subclass (**in** *dioid-one-zero*) *pre-dioid-one-zero* **..**

subclass (**in** *dioid-one-zero*) *semiring-one-zero* **..**

3.7 Duality by Opposition

Swapping the order of multiplication in a semiring (or dioid) gives another semiring (or dioid), called its *dual* or *opposite*.

definition (**in** *times*) *opp-mult* (**infixl** ⊙ *70*)
where *x* ⊙ *y* ≡ *y* · *x*

lemma (**in** *semiring-1*) *dual-semiring-1*:
class.semiring-1 1 (⊙) (+) *0*
by *unfold-locales* (*auto simp add*: *opp-mult-def mult.assoc distrib-right distrib-left*)

lemma (**in** *dioid-one-zero*) *dual-dioid-one-zero*:
class.dioid-one-zero (+) (⊙) *1 0* (≤) (<)
by *unfold-locales* (*auto simp add*: *opp-mult-def mult.assoc distrib-right distrib-left*)

3.8 Selective Near Semirings

In this section we briefly sketch a generalisation of the notion of *dioid*. Some important models, e.g. max-plus and min-plus semirings, have that property.

class *selective-near-semiring* = *ab-near-semiring* + *plus-ord* +
assumes *select*: $x + y = x \vee x + y = y$

begin

lemma *select-alt*: $x + y \in \{x,y\}$
by (*simp add*: *local.select*)

It follows immediately that every selective near semiring is a near dioid.

subclass *near-dioid*
by (*unfold-locales*, *meson select*)

Moreover, the order in a selective near semiring is obviously linear.

subclass *linorder*
by (*unfold-locales*, *metis add.commute join.sup.orderI select*)

end

class *selective-semiring* = *selective-near-semiring* + *semiring-one-zero*

begin

subclass *dioid-one-zero* **..**

end

end

4 Models of Dioids

theory *Dioid-Models*
imports *Dioid HOL.Real*
begin

In this section we consider some well known models of dioids. These so far include the powerset dioid over a monoid, languages, binary relations, sets of traces, sets paths (in a graph), as well as the min-plus and the max-plus semirings. Most of these models are taken from an article about Kleene algebras with domain [9].

The advantage of formally linking these models with the abstract axiomatisations of dioids is that all abstract theorems are automatically available in all models. It therefore makes sense to establish models for the strongest possible axiomatisations (whereas theorems should be proved for the weakest ones).

4.1 The Powerset Dioid over a Monoid

We assume a multiplicative monoid and define the usual complex product on sets of elements. We formalise the well known result that this lifting induces a dioid.

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

  definition *one-set-def*:
    $1 = \{1\}$

  definition *c-prod-def*: — the complex product
    $A \cdot B = \{u * v \mid u\ v.\ u \in A \wedge v \in B\}$

  instance
  proof
    fix $X\ Y\ Z$ :: *'a set*
     show $X \cdot Y \cdot Z = X \cdot (Y \cdot Z)$
      by (*auto simp add*: *c-prod-def*) (*metis mult.assoc*)+
    show $1 \cdot X = X$
      by (*simp add*: *one-set-def c-prod-def*)
    show $X \cdot 1 = X$
      by (*simp add*: *one-set-def c-prod-def*)
  qed

end

instantiation *set* :: (*monoid-mult*) *dioid-one-zero*
begin

  definition *zero-set-def*:
    $0 = \{\}$

  definition *plus-set-def*:
    $A + B = A \cup B$

```
instance
proof
  fix X Y Z :: 'a set
  show X + Y + Z = X + (Y + Z)
    by (simp add: Un-assoc plus-set-def)
  show X + Y = Y + X
    by (simp add: Un-commute plus-set-def)
  show (X + Y) · Z = X · Z + Y · Z
    by (auto simp add: plus-set-def c-prod-def)
  show 1 · X = X
    by (simp add: one-set-def c-prod-def)
  show X · 1 = X
    by (simp add: one-set-def c-prod-def)
  show 0 + X = X
    by (simp add: plus-set-def zero-set-def)
  show 0 · X = 0
    by (simp add: c-prod-def zero-set-def)
  show X · 0 = 0
    by (simp add: c-prod-def zero-set-def)
  show X ⊆ Y ⟷ X + Y = Y
    by (simp add: plus-set-def subset-Un-eq)
  show X ⊂ Y ⟷ X ⊆ Y ∧ X ≠ Y
    by (fact psubset-eq)
  show X + X = X
    by (simp add: Un-absorb plus-set-def)
  show X · (Y + Z) = X · Y + X · Z
    by (auto simp add: plus-set-def c-prod-def)
qed

end
```

4.2 Language Dioids

Language dioids arise as special cases of the monoidal lifting because sets of words form free monoids. Moreover, monoids of words are isomorphic to monoids of lists under append.

To show that languages form dioids it therefore suffices to show that sets of lists closed under append and multiplication with the empty word form a (multiplicative) monoid. Isabelle then does the rest of the work automatically. Infix @ denotes word concatenation.

```
instantiation list :: (type) monoid-mult
begin

  definition times-list-def:
    xs * ys ≡ xs @ ys

  definition one-list-def:
    1 ≡ []
```

instance proof
  fix *xs ys zs* :: *'a list*
  show $xs * ys * zs = xs * (ys * zs)$
    by (*simp add*: *times-list-def*)
  show $1 * xs = xs$
    by (*simp add*: *one-list-def times-list-def*)
  show $xs * 1 = xs$
    by (*simp add*: *one-list-def times-list-def*)
qed

end

Languages as sets of lists have already been formalised in Isabelle in various places. We can now obtain much of their algebra for free.

type-synonym *'a lan* = *'a list set*

interpretation *lan-dioid*: *dioid-one-zero* (+) (·) *1*::*'a lan* *0* (⊆) (⊂) **..**

4.3 Relation Dioids

We now show that binary relations under union, relational composition, the identity relation, the empty relation and set inclusion form dioids. Due to the well developed relation library of Isabelle this is entirely trivial.

interpretation *rel-dioid*: *dioid-one-zero* (∪) (*O*) *Id* {} (⊆) (⊂)
  by (*unfold-locales*, *auto*)

interpretation *rel-monoid*: *monoid-mult* *Id* (*O*) **..**

4.4 Trace Dioids

Traces have been considered, for instance, by Kozen [22] in the context of Kleene algebras with tests. Intuitively, a trace is an execution sequence of a labelled transition system from some state to some other state, in which state labels and action labels alternate, and which begin and end with a state label.

Traces generalise words: words can be obtained from traces by forgetting state labels. Similarly, sets of traces generalise languages.

In this section we show that sets of traces under union, an appropriately defined notion of complex product, the set of all traces of length zero, the empty set of traces and set inclusion form a dioid.

We first define the notion of trace and the product of traces, which has been called *fusion product* by Kozen.

type-synonym (*'p*, *'a*) *trace* = *'p* × (*'a* × *'p*) *list*

```
definition first :: ('p, 'a) trace ⇒ 'p where
  first = fst

lemma first-conv [simp]: first (p, xs) = p
  by (unfold first-def, simp)

fun last :: ('p, 'a) trace ⇒ 'p where
  last (p, []) = p
| last (-, xs) = snd (List.last xs)

lemma last-append [simp]: last (p, xs @ ys) = last (last (p, xs), ys)
proof (cases xs)
  show xs = [] ⟹ last (p, xs @ ys) = last (last (p, xs), ys)
    by simp
  show ⋀a list. xs = a # list ⟹
    last (p, xs @ ys) = last (last (p, xs), ys)
  proof (cases ys)
    show ⋀a list. ⟦xs = a # list; ys = []⟧
      ⟹ last (p, xs @ ys) = last (last (p, xs), ys)
      by simp
    show ⋀a list aa lista. ⟦xs = a # list; ys = aa # lista⟧
      ⟹ last (p, xs @ ys) = last (last (p, xs), ys)
      by simp
  qed
qed
```

The fusion product is a partial operation. It is undefined if the last element of the first trace and the first element of the second trace are different. If these elements are the same, then the fusion product removes the first element from the second trace and appends the resulting object to the first trace.

```
definition t-fusion :: ('p, 'a) trace ⇒ ('p, 'a) trace ⇒ ('p, 'a) trace where
  t-fusion x y ≡ if last x = first y then (fst x, snd x @ snd y) else undefined
```

We now show that the first element and the last element of a trace are a left and right unit for that trace and prove some other auxiliary lemmas.

```
lemma t-fusion-leftneutral [simp]: t-fusion (first x, []) x = x
  by (cases x, simp add: t-fusion-def)

lemma fusion-rightneutral [simp]: t-fusion x (last x, []) = x
  by (simp add: t-fusion-def)

lemma first-t-fusion [simp]: last x = first y ⟹ first (t-fusion x y) = first x
  by (simp add: first-def t-fusion-def)

lemma last-t-fusion [simp]: last x = first y ⟹ last (t-fusion x y) = last y
  by (simp add: first-def t-fusion-def)
```

Next we show that fusion of traces is associative.

lemma *t-fusion-assoc* [*simp*]:
$[\![$ *last* x = *first* y; *last* y = *first* z $]\!] \Longrightarrow$ *t-fusion* x (*t-fusion* y z) = *t-fusion* (*t-fusion* x y) z
by (*cases* x, *cases* y, *cases* z, *simp add*: *t-fusion-def*)

4.5 Sets of Traces

We now lift the fusion product to a complex product on sets of traces. This operation is total.

no-notation
times (**infixl** $\cdot$ *70*)

definition *t-prod* :: ($'p$, $'a$) *trace set* $\Rightarrow$ ($'p$, $'a$) *trace set* $\Rightarrow$ ($'p$, $'a$) *trace set* (**infixl** $\cdot$ *70*)
where $X \cdot Y = \{$*t-fusion* u v| u v. $u \in X \wedge v \in Y \wedge$ *last* u = *first* $v\}$

Next we define the empty set of traces and the set of traces of length zero as the multiplicative unit of the trace dioid.

definition *t-zero* :: ($'p$, $'a$) *trace set* **where**
t-zero $\equiv \{\}$

definition *t-one* :: ($'p$, $'a$) *trace set* **where**
t-one $\equiv \bigcup p.\ \{(p, [])\}$

We now provide elimination rules for trace products.

lemma *t-prod-iff*:
$w \in X \cdot Y \longleftrightarrow (\exists\, u\ v.\ w =$ *t-fusion* u $v \wedge u \in X \wedge v \in Y \wedge$ *last* u = *first* v)
by (*unfold t-prod-def*) *auto*

lemma *t-prod-intro* [*simp*, *intro*]:
$[\![$ $u \in X$; $v \in Y$; *last* u = *first* v $]\!] \Longrightarrow$ *t-fusion* u $v \in X \cdot Y$
by (*meson t-prod-iff*)

lemma *t-prod-elim* [*elim*]:
$w \in X \cdot Y \Longrightarrow \exists\, u\ v.\ w =$ *t-fusion* u $v \wedge u \in X \wedge v \in Y \wedge$ *last* u = *first* v
by (*meson t-prod-iff*)

Finally we prove the interpretation statement that sets of traces under union and the complex product based on trace fusion together with the empty set of traces and the set of traces of length one forms a dioid.

interpretation *trace-dioid*: *dioid-one-zero* ($\cup$) *t-prod t-one t-zero* ($\subseteq$) ($\subset$)
apply *unfold-locales*
apply (*auto simp add*: *t-prod-def t-one-def t-zero-def t-fusion-def*)
apply (*metis last-append*)
apply (*metis last-append append-assoc*)
done

no-notation
t-prod (**infixl** · *70*)

4.6 The Path Diod

The next model we consider are sets of paths in a graph. We consider two variants, one that contains the empty path and one that doesn't. The former leads to more difficult proofs and a more involved specification of the complex product. We start with paths that include the empty path. In this setting, a path is a list of nodes.

4.7 Path Models with the Empty Path

type-synonym *'a path* = *'a list*

Path fusion is defined similarly to trace fusion. Mathematically it should be a partial operation. The fusion of two empty paths yields the empty path; the fusion between a non-empty path and an empty one is undefined; the fusion of two non-empty paths appends the tail of the second path to the first one.

We need to use a total alternative and make sure that undefined paths do not contribute to the complex product.

```
fun p-fusion :: 'a path ⇒ 'a path ⇒ 'a path where
  p-fusion [] - = []
| p-fusion - [] = []
| p-fusion ps (q # qs) = ps @ qs

lemma p-fusion-assoc:
  p-fusion ps (p-fusion qs rs) = p-fusion (p-fusion ps qs) rs
proof (induct rs)
  case Nil show ?case
    by (metis p-fusion.elims p-fusion.simps(2))
  case Cons show ?case
  proof (induct qs)
    case Nil show ?case
      by (metis neq-Nil-conv p-fusion.simps(1) p-fusion.simps(2))
    case Cons show ?case
      proof −
        have ∀ ps. ([] = ps ∨ hd ps # tl ps = ps) ∧ ((∀ q qs. q # qs ≠ ps) ∨ [] ≠
ps)
          using list.collapse by fastforce
        moreover hence ∀ ps q qs. p-fusion ps (q # qs) = ps @ qs ∨ [] = ps
          by (metis p-fusion.simps(3))
        ultimately show ?thesis
         by (metis (no-types) Cons-eq-appendI append-eq-appendI p-fusion.simps(1)
p-fusion.simps(3))
      qed
```

qed
qed

This lemma overapproximates the real situation, but it holds in all cases where path fusion should be defined.

lemma *p-fusion-last*:
assumes *List.last ps* = *hd qs*
and *ps* ≠ []
and *qs* ≠ []
shows *List.last* (*p-fusion ps qs*) = *List.last qs*
by (*metis* (*hide-lams*, *no-types*) *List.last.simps List.last-append append-Nil2 assms list.sel*(*1*) *neq-Nil-conv p-fusion.simps*(*3*))

lemma *p-fusion-hd*: ⟦*ps* ≠ []; *qs* ≠ []⟧ ⟹ *hd* (*p-fusion ps qs*) = *hd ps*
by (*metis list.exhaust p-fusion.simps*(*3*) *append-Cons list.sel*(*1*))

lemma *nonempty-p-fusion*: ⟦*ps* ≠ []; *qs* ≠ []⟧ ⟹ *p-fusion ps qs* ≠ []
by (*metis list.exhaust append-Cons p-fusion.simps*(*3*) *list.simps*(*2*))

We now define a condition that filters out undefined paths in the complex product.

abbreviation *p-filter* :: *'a path* ⇒ *'a path* ⇒ *bool* **where**
p-filter ps qs ≡ ((*ps* = [] ∧ *qs* = []) ∨ (*ps* ≠ [] ∧ *qs* ≠ [] ∧ (*List.last ps*) = *hd qs*))

no-notation
times (**infixl** · *70*)

definition *p-prod* :: *'a path set* ⇒ *'a path set* ⇒ *'a path set* (**infixl** · *70*)
where *X* · *Y* = {*rs* . ∃ *ps* ∈ *X*. ∃ *qs* ∈ *Y*. *rs* = *p-fusion ps qs* ∧ *p-filter ps qs*}

lemma *p-prod-iff*:
ps ∈ *X* · *Y* ⟷ (∃ *qs rs*. *ps* = *p-fusion qs rs* ∧ *qs* ∈ *X* ∧ *rs* ∈ *Y* ∧ *p-filter qs rs*)
by (*unfold p-prod-def*) *auto*

Due to the complexity of the filter condition, proving properties of complex products can be tedious.

lemma *p-prod-assoc*: (*X* · *Y*) · *Z* = *X* · (*Y* · *Z*)
proof (*rule set-eqI*)
fix *ps*
show *ps* ∈ (*X* · *Y*) · *Z* ⟷ *ps* ∈ *X* · (*Y* · *Z*)
proof (*cases ps*)
case *Nil* **thus** *?thesis*
by *auto* (*metis nonempty-p-fusion p-prod-iff*)+
next
case *Cons* **thus** *?thesis*
by (*auto simp add*: *p-prod-iff*) (*metis* (*hide-lams*, *mono-tags*) *nonempty-p-fusion p-fusion-assoc p-fusion-hd p-fusion-last*)+

qed
qed

We now define the multiplicative unit of the path dioid as the set of all paths of length one, including the empty path, and show the unit laws with respect to the path product.

definition *p-one* :: *'a path set* **where**
p-one ≡ {*p* . ∃ *q*::*'a*. *p* = [*q*]} ∪ {[]}

lemma *p-prod-onel* [*simp*]: *p-one* · *X* = *X*
proof (*rule set-eqI*)
fix *ps*
show *ps* ∈ *p-one* · *X* ⟷ *ps* ∈ *X*
proof (*cases ps*)
case *Nil* **thus** *?thesis*
by (*auto simp add*: *p-one-def p-prod-def*, *metis nonempty-p-fusion not-Cons-self*)
next
case *Cons* **thus** *?thesis*
by (*auto simp add*: *p-one-def p-prod-def*, *metis append-Cons append-Nil list.sel*(*1*) *neq-Nil-conv p-fusion.simps*(*3*), *metis Cons-eq-appendI list.sel*(*1*) *last-ConsL list.simps*(*3*) *p-fusion.simps*(*3*) *self-append-conv2*)
qed
qed

lemma *p-prod-oner* [*simp*]: *X* · *p-one* = *X*
proof (*rule set-eqI*)
fix *ps*
show *ps* ∈ *X* · *p-one* ⟷ *ps* ∈ *X*
proof (*cases ps*)
case *Nil* **thus** *?thesis*
by (*auto simp add*: *p-one-def p-prod-def*, *metis nonempty-p-fusion not-Cons-self2*, *metis p-fusion.simps*(*1*))
next
case *Cons* **thus** *?thesis*
by (*auto simp add*: *p-one-def p-prod-def*, *metis append-Nil2 neq-Nil-conv p-fusion.simps*(*3*), *metis list.sel*(*1*) *list.simps*(*2*) *p-fusion.simps*(*3*) *self-append-conv*)
qed
qed

Next we show distributivity laws at the powerset level.

lemma *p-prod-distl*: *X* · (*Y* ∪ *Z*) = *X* · *Y* ∪ *X* · *Z*
proof (*rule set-eqI*)
fix *ps*
show *ps* ∈ *X* · (*Y* ∪ *Z*) ⟷ *ps* ∈ *X* · *Y* ∪ *X* · *Z*
by (*cases ps*) (*auto simp add*: *p-prod-iff*)
qed

lemma *p-prod-distr*: (*X* ∪ *Y*) · *Z* = *X* · *Z* ∪ *Y* · *Z*

proof (*rule set-eqI*)
  fix *ps*
  show $ps \in (X \cup Y) \cdot Z \longleftrightarrow ps \in X \cdot Z \cup Y \cdot Z$
    by (*cases ps*) (*auto simp add*: *p-prod-iff*)
qed

Finally we show that sets of paths under union, the complex product, the unit set and the empty set form a dioid.

interpretation *path-dioid*: *dioid-one-zero* (∪) (·) *p-one* {} (⊆) (⊂)
proof
  fix *x y z* :: *'a path set*
  show $x \cup y \cup z = x \cup (y \cup z)$
    by *auto*
  show $x \cup y = y \cup x$
    by *auto*
  show $(x \cdot y) \cdot z = x \cdot (y \cdot z)$
    by (*fact p-prod-assoc*)
  show $(x \cup y) \cdot z = x \cdot z \cup y \cdot z$
    by (*fact p-prod-distr*)
  show *p-one* $\cdot x = x$
    by (*fact p-prod-onel*)
  show $x \cdot$ *p-one* $= x$
    by (*fact p-prod-oner*)
  show $\{\} \cup x = x$
    by *auto*
  show $\{\} \cdot x = \{\}$
    by (*metis all-not-in-conv p-prod-iff*)
  show $x \cdot \{\} = \{\}$
    by (*metis all-not-in-conv p-prod-iff*)
  show $(x \subseteq y) = (x \cup y = y)$
    by *auto*
  show $(x \subset y) = (x \subseteq y \wedge x \neq y)$
    by *auto*
  show $x \cup x = x$
    by *auto*
  show $x \cdot (y \cup z) = x \cdot y \cup x \cdot z$
    by (*fact p-prod-distl*)
qed

no-notation
  p-prod (**infixl** · *70*)

4.8 Path Models without the Empty Path

We now build a model of paths that does not include the empty path and therefore leads to a simpler complex product.

datatype *'a ppath* = *Node 'a* | *Cons 'a 'a ppath*

primrec *pp-first* :: *'a ppath* ⇒ *'a* **where**
pp-first (*Node x*) = *x*
| *pp-first* (*Cons x -*) = *x*

primrec *pp-last* :: *'a ppath* ⇒ *'a* **where**
pp-last (*Node x*) = *x*
| *pp-last* (*Cons - xs*) = *pp-last xs*

The path fusion product (although we define it as a total funcion) should only be applied when the last element of the first argument is equal to the first element of the second argument.

primrec *pp-fusion* :: *'a ppath* ⇒ *'a ppath* ⇒ *'a ppath* **where**
pp-fusion (*Node x*) *ys* = *ys*
| *pp-fusion* (*Cons x xs*) *ys* = *Cons x* (*pp-fusion xs ys*)

We now go through the same steps as for traces and paths before, showing that the first and last element of a trace a left or right unit for that trace and that the fusion product on traces is associative.

lemma *pp-fusion-leftneutral* [*simp*]: *pp-fusion* (*Node* (*pp-first x*)) *x* = *x*
by *simp*

lemma *pp-fusion-rightneutral* [*simp*]: *pp-fusion x* (*Node* (*pp-last x*)) = *x*
by (*induct x*) *simp-all*

lemma *pp-first-pp-fusion* [*simp*]:
pp-last x = *pp-first y* ⟹ *pp-first* (*pp-fusion x y*) = *pp-first x*
by (*induct x*) *simp-all*

lemma *pp-last-pp-fusion* [*simp*]:
pp-last x = *pp-first y* ⟹ *pp-last* (*pp-fusion x y*) = *pp-last y*
by (*induct x*) *simp-all*

lemma *pp-fusion-assoc* [*simp*]:
⟦ *pp-last x* = *pp-first y*; *pp-last y* = *pp-first z* ⟧ ⟹ *pp-fusion x* (*pp-fusion y z*) = *pp-fusion* (*pp-fusion x y*) *z*
by (*induct x*) *simp-all*

We now lift the path fusion product to a complex product on sets of paths. This operation is total.

definition *pp-prod* :: *'a ppath set* ⇒ *'a ppath set* ⇒ *'a ppath set* (**infixl** · *70*)
where $X \cdot Y$ = {*pp-fusion u v*| *u v*. $u \in X \wedge v \in Y \wedge$ *pp-last u* = *pp-first v*}

Next we define the set of paths of length one as the multiplicative unit of the path dioid.

definition *pp-one* :: *'a ppath set* **where**
pp-one ≡ *range Node*

We again provide an elimination rule.

lemma *pp-prod-iff*:
  $w \in X \cdot Y \longleftrightarrow (\exists\, u\ v.\ w = \textit{pp-fusion}\ u\ v \wedge u \in X \wedge v \in Y \wedge \textit{pp-last}\ u = \textit{pp-first}\ v)$
  by (*unfold pp-prod-def*) *auto*

interpretation *ppath-dioid*: *dioid-one-zero* (∪) (·) *pp-one* {} (⊆) (⊂)
proof
  fix $x\ y\ z$ :: *'a ppath set*
  show $x \cup y \cup z = x \cup (y \cup z)$
    by *auto*
  show $x \cup y = y \cup x$
    by *auto*
  show $x \cdot y \cdot z = x \cdot (y \cdot z)$
    by (*auto simp add*: *pp-prod-def*, *metis pp-first-pp-fusion pp-fusion-assoc*, *metis pp-last-pp-fusion*)
  show $(x \cup y) \cdot z = x \cdot z \cup y \cdot z$
    by (*auto simp add*: *pp-prod-def*)
  show $\textit{pp-one} \cdot x = x$
    by (*auto simp add*: *pp-one-def pp-prod-def*, *metis pp-fusion.simps*(*1*) *pp-last.simps*(*1*) *rangeI*)
  show $x \cdot \textit{pp-one} = x$
    by (*auto simp add*: *pp-one-def pp-prod-def*, *metis pp-first.simps*(*1*) *pp-fusion-rightneutral rangeI*)
  show $\{\} \cup x = x$
    by *auto*
  show $\{\} \cdot x = \{\}$
    by (*simp add*: *pp-prod-def*)
  show $x \cdot \{\} = \{\}$
    by (*simp add*: *pp-prod-def*)
  show $x \subseteq y \longleftrightarrow x \cup y = y$
    by *auto*
  show $x \subset y \longleftrightarrow x \subseteq y \wedge x \neq y$
    by *auto*
  show $x \cup x = x$
    by *auto*
  show $x \cdot (y \cup z) = x \cdot y \cup x \cdot z$
    by (*auto simp add*: *pp-prod-def*)
qed

no-notation
  pp-prod (**infixl** · *70*)

4.9 The Distributive Lattice Dioid

A bounded distributive lattice is a distributive lattice with a least and a greatest element. Using Isabelle's lattice theory file we define a bounded distributive lattice as an axiomatic type class and show, using a sublocale statement, that every bounded distributive lattice is a dioid with one and zero.

class *bounded-distributive-lattice* = *bounded-lattice* + *distrib-lattice*

sublocale *bounded-distributive-lattice* ⊆ *dioid-one-zero sup inf top bot less-eq*
proof
  fix *x y z*
  show *sup* (*sup x y*) *z* = *sup x* (*sup y z*)
    by (*fact sup-assoc*)
  show *sup x y* = *sup y x*
    by (*fact sup.commute*)
  show *inf* (*inf x y*) *z* = *inf x* (*inf y z*)
    by (*metis inf.commute inf.left-commute*)
  show *inf* (*sup x y*) *z* = *sup* (*inf x z*) (*inf y z*)
    by (*fact inf-sup-distrib2*)
  show *inf top x* = *x*
    by *simp*
  show *inf x top* = *x*
    by *simp*
  show *sup bot x* = *x*
    by *simp*
  show *inf bot x* = *bot*
    by *simp*
  show *inf x bot* = *bot*
    by *simp*
  show ($x \leq y$) = (*sup x y* = *y*)
    by (*fact le-iff-sup*)
  show ($x < y$) = ($x \leq y \wedge x \neq y$)
    by *auto*
  show *sup x x* = *x*
    by *simp*
  show *inf x* (*sup y z*) = *sup* (*inf x y*) (*inf x z*)
    by (*fact inf-sup-distrib1*)
qed

4.10 The Boolean Dioid

In this section we show that the booleans form a dioid, because the booleans form a bounded distributive lattice.

instantiation *bool* :: *bounded-distributive-lattice*
begin

  instance ..

end

interpretation *boolean-dioid*: *dioid-one-zero sup inf True False less-eq less*
  by (*unfold-locales*, *simp-all add*: *inf-bool-def sup-bool-def*)

4.11 The Max-Plus Dioid

The following dioids have important applications in combinatorial optimisations, control theory, algorithm design and computer networks.

A definition of reals extended with $+\infty$ *and* $-\infty$ may be found in *HOL/Library/Extended_Real.thy*. Alas, we require separate extensions with either $+\infty$ or $-\infty$.

The carrier set of the max-plus semiring is the set of real numbers extended by minus infinity. The operation of addition is maximum, the operation of multiplication is addition, the additive unit is minus infinity and the multiplicative unit is zero.

datatype *mreal* = *mreal real* | *MInfty* — minus infinity

fun *mreal-max* **where**
mreal-max (*mreal x*) (*mreal y*) = *mreal* (*max x y*)
| *mreal-max x MInfty* = *x*
| *mreal-max MInfty y* = *y*

lemma *mreal-max-simp-3* [*simp*]: *mreal-max MInfty y* = *y*
by (*cases y*, *simp-all*)

fun *mreal-plus* **where**
mreal-plus (*mreal x*) (*mreal y*) = *mreal* (*x* + *y*)
| *mreal-plus - -* = *MInfty*

We now show that the max plus-semiring satisfies the axioms of selective semirings, from which it follows that it satisfies the dioid axioms.

instantiation *mreal* :: *selective-semiring*
begin

definition *zero-mreal-def*:
0 ≡ *MInfty*

definition *one-mreal-def*:
1 ≡ *mreal 0*

definition *plus-mreal-def*:
x + *y* ≡ *mreal-max x y*

definition *times-mreal-def*:
x ∗ *y* ≡ *mreal-plus x y*

definition *less-eq-mreal-def*:
(*x*::*mreal*) ≤ *y* ≡ *x* + *y* = *y*

definition *less-mreal-def*:
(*x*::*mreal*) < *y* ≡ *x* ≤ *y* ∧ *x* ≠ *y*

```
  instance
  proof
    fix x y z :: mreal
    show x + y + z = x + (y + z)
      by (cases x, cases y, cases z, simp-all add: plus-mreal-def)
    show x + y = y + x
      by (cases x, cases y, simp-all add: plus-mreal-def)
    show x * y * z = x * (y * z)
      by (cases x, cases y, cases z, simp-all add: times-mreal-def)
    show (x + y) * z = x * z + y * z
      by (cases x, cases y, cases z, simp-all add: plus-mreal-def times-mreal-def)
    show 1 * x = x
      by (cases x, simp-all add: one-mreal-def times-mreal-def)
    show x * 1 = x
      by (cases x, simp-all add: one-mreal-def times-mreal-def)
    show 0 + x = x
      by (cases x, simp-all add: plus-mreal-def zero-mreal-def)
    show 0 * x = 0
      by (cases x, simp-all add: times-mreal-def zero-mreal-def)
    show x * 0 = 0
      by (cases x, simp-all add: times-mreal-def zero-mreal-def)
    show x ≤ y ⟷ x + y = y
      by (metis less-eq-mreal-def)
    show x < y ⟷ x ≤ y ∧ x ≠ y
      by (metis less-mreal-def)
    show x + y = x ∨ x + y = y
       by (cases x, cases y, simp-all add: plus-mreal-def, metis linorder-le-cases
max.absorb-iff2 max.absorb1)
    show x * (y + z) = x * y + x * z
      by (cases x, cases y, cases z, simp-all add: plus-mreal-def times-mreal-def)
qed

end
```

4.12 The Min-Plus Dioid

The min-plus dioid is also known as *tropical semiring*. Here we need to add a positive infinity to the real numbers. The procedere follows that of max-plus semirings.

```
datatype preal = preal real | PInfty  — plus infinity

fun preal-min where
  preal-min (preal x) (preal y) = preal (min x y)
| preal-min x PInfty = x
| preal-min PInfty y = y

lemma preal-min-simp-3 [simp]: preal-min PInfty y = y
  by (cases y, simp-all)
```

fun *preal-plus* **where**
preal-plus (*preal* x) (*preal* y) = *preal* (x + y)
| *preal-plus* - - = *PInfty*

instantiation *preal* :: *selective-semiring*
begin

definition *zero-preal-def*:
0 ≡ *PInfty*

definition *one-preal-def*:
1 ≡ *preal 0*

definition *plus-preal-def*:
$x + y$ ≡ *preal-min* x y

definition *times-preal-def*:
$x * y$ ≡ *preal-plus* x y

definition *less-eq-preal-def*:
(x::*preal*) $\leq y \equiv x + y = y$

definition *less-preal-def*:
(x::*preal*) $< y \equiv x \leq y \wedge x \neq y$

instance
proof
fix x y z :: *preal*
show $x + y + z = x + (y + z)$
by (*cases* x, *cases* y, *cases* z, *simp-all add*: *plus-preal-def*)
show $x + y = y + x$
by (*cases* x, *cases* y, *simp-all add*: *plus-preal-def*)
show $x * y * z = x * (y * z)$
by (*cases* x, *cases* y, *cases* z, *simp-all add*: *times-preal-def*)
show $(x + y) * z = x * z + y * z$
by (*cases* x, *cases* y, *cases* z, *simp-all add*: *plus-preal-def times-preal-def*)
show $1 * x = x$
by (*cases* x, *simp-all add*: *one-preal-def times-preal-def*)
show $x * 1 = x$
by (*cases* x, *simp-all add*: *one-preal-def times-preal-def*)
show $0 + x = x$
by (*cases* x, *simp-all add*: *plus-preal-def zero-preal-def*)
show $0 * x = 0$
by (*cases* x, *simp-all add*: *times-preal-def zero-preal-def*)
show $x * 0 = 0$
by (*cases* x, *simp-all add*: *times-preal-def zero-preal-def*)
show $x \leq y \longleftrightarrow x + y = y$
by (*metis less-eq-preal-def*)
show $x < y \longleftrightarrow x \leq y \wedge x \neq y$

by (*metis less-preal-def*)
show $x + y = x \vee x + y = y$
by (*cases x*, *cases y*, *simp-all add*: *plus-preal-def*, *metis linorder-le-cases min.absorb2 min.absorb-iff1*)
show $x * (y + z) = x * y + x * z$
by (*cases x*, *cases y*, *cases z*, *simp-all add*: *plus-preal-def times-preal-def*)
qed

end

Variants of min-plus and max-plus semirings can easily be obtained. Here we formalise the min-plus semiring over the natural numbers as an example.

datatype *pnat* = *pnat nat* | *PInfty* — plus infinity

fun *pnat-min* **where**
pnat-min (*pnat x*) (*pnat y*) = *pnat* (*min x y*)
| *pnat-min x PInfty* = *x*
| *pnat-min PInfty x* = *x*

lemma *pnat-min-simp-3* [*simp*]: *pnat-min PInfty y* = *y*
by (*cases y*, *simp-all*)

fun *pnat-plus* **where**
pnat-plus (*pnat x*) (*pnat y*) = *pnat* ($x + y$)
| *pnat-plus* - - = *PInfty*

instantiation *pnat* :: *selective-semiring*
begin

definition *zero-pnat-def*:
0 ≡ *PInfty*

definition *one-pnat-def*:
1 ≡ *pnat 0*

definition *plus-pnat-def*:
$x + y$ ≡ *pnat-min x y*

definition *times-pnat-def*:
$x * y$ ≡ *pnat-plus x y*

definition *less-eq-pnat-def*:
(*x*::*pnat*) ≤ *y* ≡ $x + y = y$

definition *less-pnat-def*:
(*x*::*pnat*) < *y* ≡ $x \le y \wedge x \neq y$

lemma *zero-pnat-top*: (*x*::*pnat*) ≤ *1*
by (*cases x*, *simp-all add*: *less-eq-pnat-def plus-pnat-def one-pnat-def*)

instance
proof
fix *x y z* :: *pnat*
show $x + y + z = x + (y + z)$
by (*cases x*, *cases y*, *cases z*, *simp-all add*: *plus-pnat-def*)
show $x + y = y + x$
by (*cases x*, *cases y*, *simp-all add*: *plus-pnat-def*)
show $x * y * z = x * (y * z)$
by (*cases x*, *cases y*, *cases z*, *simp-all add*: *times-pnat-def*)
show $(x + y) * z = x * z + y * z$
by (*cases x*, *cases y*, *cases z*, *simp-all add*: *plus-pnat-def times-pnat-def*)
show $1 * x = x$
by (*cases x*, *simp-all add*: *one-pnat-def times-pnat-def*)
show $x * 1 = x$
by (*cases x*, *simp-all add*: *one-pnat-def times-pnat-def*)
show $0 + x = x$
by (*cases x*, *simp-all add*: *plus-pnat-def zero-pnat-def*)
show $0 * x = 0$
by (*cases x*, *simp-all add*: *times-pnat-def zero-pnat-def*)
show $x * 0 = 0$
by (*cases x*, *simp-all add*: *times-pnat-def zero-pnat-def*)
show $x \leq y \longleftrightarrow x + y = y$
by (*metis less-eq-pnat-def*)
show $x < y \longleftrightarrow x \leq y \wedge x \neq y$
by (*metis less-pnat-def*)
show $x + y = x \vee x + y = y$
by (*cases x*, *cases y*, *simp-all add*: *plus-pnat-def*, *metis linorder-le-cases min.absorb2 min.absorb-iff1*)
show $x * (y + z) = x * y + x * z$
by (*cases x*, *cases y*, *cases z*, *simp-all add*: *plus-pnat-def times-pnat-def*)
qed

end

end

5 Matrices

theory *Matrix*
imports *HOL−Word.Word Dioid*
begin

In this section we formalise a perhaps more natural version of matrices of fixed dimension ($m \times n$-matrices). It is well known that such matrices over a Kleene algebra form a Kleene algebra [8].

5.1 Type Definition

typedef (**overloaded**) *'a atMost* = {..<*LENGTH*(*'a*::*len*)}
by *auto*

declare *Rep-atMost-inject* [*simp*]

lemma *UNIV-atMost*:
 (*UNIV*::*'a atMost set*) = *Abs-atMost* ' {..<*LENGTH*(*'a*::*len*)}
 apply *auto*
 apply (*rule Abs-atMost-induct*)
 apply *auto*
done

lemma *finite-UNIV-atMost* [*simp*]: *finite* (*UNIV*::(*'a*::*len*) *atMost set*)
 by (*simp add*: *UNIV-atMost*)

Our matrix type is similar to *'a^'n^'m* from *HOL/Multivariate_Analysis/Finite_Cartesian_Product.thy* but (i) we explicitly define a type constructor for matrices and square matrices, and (ii) in the definition of operations, e.g., matrix multiplication, we impose weaker sort requirements on the element type.

context notes [[*typedef-overloaded*]]
begin

datatype (*'a*,*'m*,*'n*) *matrix* = *Matrix 'm atMost* ⇒ *'n atMost* ⇒ *'a*

datatype (*'a*,*'m*) *sqmatrix* = *SqMatrix 'm atMost* ⇒ *'m atMost* ⇒ *'a*

end

fun *sqmatrix-of-matrix* **where**
 sqmatrix-of-matrix (*Matrix A*) = *SqMatrix A*

fun *matrix-of-sqmatrix* **where**
 matrix-of-sqmatrix (*SqMatrix A*) = *Matrix A*

5.2 0 and 1

instantiation *matrix* :: (*zero*,*type*,*type*) *zero*
begin
 definition *zero-matrix-def*: *0* ≡ *Matrix* (λ*i j*. *0*)
 instance ..
end

instantiation *sqmatrix* :: (*zero*,*type*) *zero*
begin
 definition *zero-sqmatrix-def*: *0* ≡ *SqMatrix* (λ*i j*. *0*)
 instance ..
end

Tricky sort issues: compare *one-matrix* with *one-sqmatrix* ...

instantiation *matrix* :: ({*zero*,*one*},*len*,*len*) *one*
begin
  definition *one-matrix-def*:
    1 ≡ *Matrix* (λ*i j*. *if Rep-atMost i* = *Rep-atMost j then 1 else 0*)
  instance ..
end

instantiation *sqmatrix* :: ({*zero*,*one*},*type*) *one*
begin
  definition *one-sqmatrix-def*:
    1 ≡ *SqMatrix* (λ*i j*. *if i* = *j then 1 else 0*)
  instance ..
end

5.3 Matrix Addition

fun *matrix-plus* **where**
  matrix-plus (*Matrix A*) (*Matrix B*) = *Matrix* (λ*i j*. *A i j* + *B i j*)

instantiation *matrix* :: (*plus*,*type*,*type*) *plus*
begin
  definition *plus-matrix-def*: *A* + *B* ≡ *matrix-plus A B*
  instance ..
end

lemma *plus-matrix-def'* [*simp*]:
  Matrix A + *Matrix B* = *Matrix* (λ*i j*. *A i j* + *B i j*)
  by (*simp add*: *plus-matrix-def*)

instantiation *sqmatrix* :: (*plus*,*type*) *plus*
begin
  definition *plus-sqmatrix-def*:
    A + *B* ≡ *sqmatrix-of-matrix* (*matrix-of-sqmatrix A* + *matrix-of-sqmatrix B*)
  instance ..
end

lemma *plus-sqmatrix-def'* [*simp*]:
  SqMatrix A + *SqMatrix B* = *SqMatrix* (λ*i j*. *A i j* + *B i j*)
  by (*simp add*: *plus-sqmatrix-def*)

lemma *matrix-add-0-right* [*simp*]:
  A + *0* = (*A*::(*'a*::*monoid-add*,*'m*,*'n*) *matrix*)
  by (*cases A*, *simp add*: *zero-matrix-def*)

lemma *matrix-add-0-left* [*simp*]:
  0 + *A* = (*A*::(*'a*::*monoid-add*,*'m*,*'n*) *matrix*)
  by (*cases A*, *simp add*: *zero-matrix-def*)

lemma *matrix-add-commute* [*simp*]:
  (*A*::(*'a*::*ab-semigroup-add*,*'m*,*'n*) *matrix*) + *B* = *B* + *A*
  by (*cases A*, *cases B*, *simp add*: *add.commute*)

lemma *matrix-add-assoc*:
  (*A*::(*'a*::*semigroup-add*,*'m*,*'n*) *matrix*) + *B* + *C* = *A* + (*B* + *C*)
  by (*cases A*, *cases B*, *cases C*, *simp add*: *add.assoc*)

lemma *matrix-add-left-commute* [*simp*]:
  (*A*::(*'a*::*ab-semigroup-add*,*'m*,*'n*) *matrix*) + (*B* + *C*) = *B* + (*A* + *C*)
  by (*metis matrix-add-assoc matrix-add-commute*)

lemma *sqmatrix-add-0-right* [*simp*]:
  A + *0* = (*A*::(*'a*::*monoid-add*,*'m*) *sqmatrix*)
  by (*cases A*, *simp add*: *zero-sqmatrix-def*)

lemma *sqmatrix-add-0-left* [*simp*]:
  0 + *A* = (*A*::(*'a*::*monoid-add*,*'m*) *sqmatrix*)
  by (*cases A*, *simp add*: *zero-sqmatrix-def*)

lemma *sqmatrix-add-commute* [*simp*]:
  (*A*::(*'a*::*ab-semigroup-add*,*'m*) *sqmatrix*) + *B* = *B* + *A*
  by (*cases A*, *cases B*, *simp add*: *add.commute*)

lemma *sqmatrix-add-assoc*:
  (*A*::(*'a*::*semigroup-add*,*'m*) *sqmatrix*) + *B* + *C* = *A* + (*B* + *C*)
  by (*cases A*, *cases B*, *cases C*, *simp add*: *add.assoc*)

lemma *sqmatrix-add-left-commute* [*simp*]:
  (*A*::(*'a*::*ab-semigroup-add*,*'m*) *sqmatrix*) + (*B* + *C*) = *B* + (*A* + *C*)
  by (*metis sqmatrix-add-commute sqmatrix-add-assoc*)

5.4 Order (via Addition)

instantiation *matrix* :: (*plus*,*type*,*type*) *plus-ord*
begin
  definition *less-eq-matrix-def*:
    (*A*::(*'a*, *'b*, *'c*) *matrix*) ≤ *B* ≡ *A* + *B* = *B*
  definition *less-matrix-def*:
    (*A*::(*'a*, *'b*, *'c*) *matrix*) < *B* ≡ *A* ≤ *B* ∧ *A* ≠ *B*

  instance
  proof
    fix *A B* :: (*'a*, *'b*, *'c*) *matrix*
    show *A* ≤ *B* ⟷ *A* + *B* = *B*
      by (*metis less-eq-matrix-def*)
    show *A* < *B* ⟷ *A* ≤ *B* ∧ *A* ≠ *B*
      by (*metis less-matrix-def*)
  qed

end

instantiation *sqmatrix* :: (*plus*,*type*) *plus-ord*
begin
  definition *less-eq-sqmatrix-def*:
    $(A::('a,\ 'b)\ sqmatrix) \leq B \equiv A + B = B$
  definition *less-sqmatrix-def*:
    $(A::('a,\ 'b)\ sqmatrix) < B \equiv A \leq B \wedge A \neq B$

  instance
  proof
    fix $A\ B :: ('a,\ 'b)\ sqmatrix$
    show $A \leq B \longleftrightarrow A + B = B$
      by (*metis less-eq-sqmatrix-def*)
    show $A < B \longleftrightarrow A \leq B \wedge A \neq B$
      by (*metis less-sqmatrix-def*)
  qed
end

5.5 Matrix Multiplication

fun *matrix-times* :: $('a::\{comm\text{-}monoid\text{-}add,times\},'m,'k)\ matrix \Rightarrow ('a,'k,'n)\ matrix \Rightarrow ('a,'m,'n)\ matrix$ **where**
  matrix-times (*Matrix A*) (*Matrix B*) = *Matrix* ($\lambda i\ j.\ sum\ (\lambda k.\ A\ i\ k * B\ k\ j)$ (*UNIV*::'*k atMost set*))

notation *matrix-times* (**infixl** $*_M$ *70*)

instantiation *sqmatrix* :: ({*comm-monoid-add*,*times*},*type*) *times*
begin
  definition *times-sqmatrix-def*:
    $A * B$ = *sqmatrix-of-matrix* (*matrix-of-sqmatrix A* $*_M$ *matrix-of-sqmatrix B*)
  instance ..
end

lemma *times-sqmatrix-def'* [*simp*]:
  SqMatrix A $*$ *SqMatrix B* = *SqMatrix* ($\lambda i\ j.\ sum\ (\lambda k.\ A\ i\ k * B\ k\ j)$ (*UNIV*::'*k atMost set*))
  by (*simp add*: *times-sqmatrix-def*)

lemma *matrix-mult-0-right* [*simp*]:
  $(A::('a::\{comm\text{-}monoid\text{-}add,mult\text{-}zero\},'m,'n)\ matrix) *_M 0 = 0$
  by (*cases A*, *simp add*: *zero-matrix-def*)

lemma *matrix-mult-0-left* [*simp*]:
  $0 *_M (A::('a::\{comm\text{-}monoid\text{-}add,mult\text{-}zero\},'m,'n)\ matrix) = 0$
  by (*cases A*, *simp add*: *zero-matrix-def*)

lemma *sum-delta-r-0* [*simp*]:

$[\![$ *finite S*; $j \notin S$ $]\!] \Longrightarrow (\sum k{\in}S.\ f\ k * (\textit{if}\ k = j\ \textit{then 1 else}\ (\textit{0}::'b::\{\textit{semiring-0},\textit{monoid-mult}\})))$ $= \textit{0}$
by (*induct S rule*: *finite-induct*, *auto*)

lemma *sum-delta-r-1* [*simp*]:
$[\![$ *finite S*; $j \in S$ $]\!] \Longrightarrow (\sum k{\in}S.\ f\ k * (\textit{if}\ k = j\ \textit{then 1 else}\ (\textit{0}::'b::\{\textit{semiring-0},\textit{monoid-mult}\})))$ $= f\ j$
by (*induct S rule*: *finite-induct*, *auto*)

lemma *matrix-mult-1-right* [*simp*]:
$(A::('a::\{\textit{semiring-0},\textit{monoid-mult}\},'m::\textit{len},'n::\textit{len})\ \textit{matrix}) *_M \textit{1} = A$
by (*cases A*, *simp add*: *one-matrix-def*)

lemma *sum-delta-l-0* [*simp*]:
$[\![$ *finite S*; $i \notin S$ $]\!] \Longrightarrow (\sum k{\in}S.\ (\textit{if}\ i = k\ \textit{then 1 else}\ (\textit{0}::'b::\{\textit{semiring-0},\textit{monoid-mult}\}))$ $* f\ k\ j) = \textit{0}$
by (*induct S rule*: *finite-induct*, *auto*)

lemma *sum-delta-l-1* [*simp*]:
$[\![$ *finite S*; $i \in S$ $]\!] \Longrightarrow (\sum k{\in}S.\ (\textit{if}\ i = k\ \textit{then 1 else}\ (\textit{0}::'b::\{\textit{semiring-0},\textit{monoid-mult}\}))$ $* f\ k\ j) = f\ i\ j$
by (*induct S rule*: *finite-induct*, *auto*)

lemma *matrix-mult-1-left* [*simp*]:
$\textit{1} *_M (A::('a::\{\textit{semiring-0},\textit{monoid-mult}\},'m::\textit{len},'n::\textit{len})\ \textit{matrix}) = A$
by (*cases A*, *simp add*: *one-matrix-def*)

lemma *matrix-mult-assoc*:
$(A::('a::\textit{semiring-0},'m,'n)\ \textit{matrix}) *_M B *_M C = A *_M (B *_M C)$
apply (*cases A*)
apply (*cases B*)
apply (*cases C*)
apply (*simp add*: *sum-distrib-right sum-distrib-left mult.assoc*)
apply (*subst sum.swap*)
apply (*rule refl*)
done

lemma *matrix-mult-distrib-left*:
$(A::('a::\{\textit{comm-monoid-add},\textit{semiring}\},'m,'n::\textit{len})\ \textit{matrix}) *_M (B + C) = A *_M$ $B + A *_M C$
by (*cases A*, *cases B*, *cases C*, *simp add*: *distrib-left sum.distrib*)

lemma *matrix-mult-distrib-right*:
$((A::('a::\{\textit{comm-monoid-add},\textit{semiring}\},'m,'n::\textit{len})\ \textit{matrix}) + B) *_M C = A *_M$ $C + B *_M C$
by (*cases A*, *cases B*, *cases C*, *simp add*: *distrib-right sum.distrib*)

lemma *sqmatrix-mult-0-right* [*simp*]:
$(A::('a::\{\textit{comm-monoid-add},\textit{mult-zero}\},'m)\ \textit{sqmatrix}) * \textit{0} = \textit{0}$

by (*cases A*, *simp add*: *zero-sqmatrix-def*)

lemma *sqmatrix-mult-0-left* [*simp*]:
0 ∗ (*A*::(*'a*::{*comm-monoid-add*,*mult-zero*},*'m*) *sqmatrix*) = *0*
by (*cases A*, *simp add*: *zero-sqmatrix-def*)

lemma *sqmatrix-mult-1-right* [*simp*]:
(*A*::(*'a*::{*semiring-0*,*monoid-mult*},*'m*::*len*) *sqmatrix*) ∗ *1* = *A*
by (*cases A*, *simp add*: *one-sqmatrix-def*)

lemma *sqmatrix-mult-1-left* [*simp*]:
1 ∗ (*A*::(*'a*::{*semiring-0*,*monoid-mult*},*'m*::*len*) *sqmatrix*) = *A*
by (*cases A*, *simp add*: *one-sqmatrix-def*)

lemma *sqmatrix-mult-assoc*:
(*A*::(*'a*::{*semiring-0*,*monoid-mult*},*'m*) *sqmatrix*) ∗ *B* ∗ *C* = *A* ∗ (*B* ∗ *C*)
apply (*cases A*)
apply (*cases B*)
apply (*cases C*)
apply (*simp add*: *sum-distrib-right sum-distrib-left mult.assoc*)
apply (*subst sum.swap*)
apply (*rule refl*)
done

lemma *sqmatrix-mult-distrib-left*:
(*A*::(*'a*::{*comm-monoid-add*,*semiring*},*'m*::*len*) *sqmatrix*) ∗ (*B* + *C*) = *A* ∗ *B* + *A* ∗ *C*
by (*cases A*, *cases B*, *cases C*, *simp add*: *distrib-left sum.distrib*)

lemma *sqmatrix-mult-distrib-right*:
((*A*::(*'a*::{*comm-monoid-add*,*semiring*},*'m*::*len*) *sqmatrix*) + *B*) ∗ *C* = *A* ∗ *C* + *B* ∗ *C*
by (*cases A*, *cases B*, *cases C*, *simp add*: *distrib-right sum.distrib*)

5.6 Square-Matrix Model of Dioids

The following subclass proofs are necessary to connect parts of our algebraic hierarchy to the hierarchy found in the Isabelle/HOL library.

subclass (**in** *ab-near-semiring-one-zerol*) *comm-monoid-add*
proof
fix *a* :: *'a*
show *0* + *a* = *a*
by (*fact add-zerol*)
qed

subclass (**in** *semiring-one-zero*) *semiring-0*
proof
fix *a* :: *'a*
show *0* ∗ *a* = *0*

```
    by (fact annil)
  show a ∗ 0 = 0
    by (fact annir)
qed

subclass (in ab-near-semiring-one) monoid-mult ..

instantiation sqmatrix :: (dioid-one-zero,len) dioid-one-zero
begin
  instance
  proof
    fix A B C :: ('a, 'b) sqmatrix
    show A + B + C = A + (B + C)
      by (fact sqmatrix-add-assoc)
    show A + B = B + A
      by (fact sqmatrix-add-commute)
    show A ∗ B ∗ C = A ∗ (B ∗ C)
      by (fact sqmatrix-mult-assoc)
    show (A + B) ∗ C = A ∗ C + B ∗ C
      by (fact sqmatrix-mult-distrib-right)
    show 1 ∗ A = A
      by (fact sqmatrix-mult-1-left)
    show A ∗ 1 = A
      by (fact sqmatrix-mult-1-right)
    show 0 + A = A
      by (fact sqmatrix-add-0-left)
    show 0 ∗ A = 0
      by (fact sqmatrix-mult-0-left)
    show A ∗ 0 = 0
      by (fact sqmatrix-mult-0-right)
    show A + A = A
      by (cases A, simp)
    show A ∗ (B + C) = A ∗ B + A ∗ C
      by (fact sqmatrix-mult-distrib-left)
  qed
end
```

5.7 Kleene Star for Matrices

We currently do not implement the Kleene star of matrices, since this is complicated.

end

6 Conway Algebras

```
theory Conway
  imports Dioid
begin
```

We define a weak regular algebra which can serve as a common basis for Kleene algebra and demonic reginement algebra. It is closely related to an axiomatisation given by Conway [8].

class *dagger-op* =
  fixes *dagger* :: $'a \Rightarrow 'a$ ($-^{\dagger}$ [*101*] *100*)

6.1 Near Conway Algebras

class *near-conway-base* = *near-dioid-one* + *dagger-op* +
  assumes *dagger-denest*: $(x + y)^{\dagger} = (x^{\dagger} \cdot y)^{\dagger} \cdot x^{\dagger}$
  and *dagger-prod-unfold* [*simp*]: $1 + x \cdot (y \cdot x)^{\dagger} \cdot y = (x \cdot y)^{\dagger}$

begin

lemma *dagger-unfoldl-eq* [*simp*]: $1 + x \cdot x^{\dagger} = x^{\dagger}$
  by (*metis dagger-prod-unfold mult-1-left mult-1-right*)

lemma *dagger-unfoldl*: $1 + x \cdot x^{\dagger} \leq x^{\dagger}$
  by *auto*

lemma *dagger-unfoldr-eq* [*simp*]: $1 + x^{\dagger} \cdot x = x^{\dagger}$
  by (*metis dagger-prod-unfold mult-1-right mult-1-left*)

lemma *dagger-unfoldr*: $1 + x^{\dagger} \cdot x \leq x^{\dagger}$
  by *auto*

lemma *dagger-unfoldl-distr* [*simp*]: $y + x \cdot x^{\dagger} \cdot y = x^{\dagger} \cdot y$
  by (*metis distrib-right′ mult-1-left dagger-unfoldl-eq*)

lemma *dagger-unfoldr-distr* [*simp*]: $y + x^{\dagger} \cdot x \cdot y = x^{\dagger} \cdot y$
  by (*metis dagger-unfoldr-eq distrib-right′ mult-1-left mult.assoc*)

lemma *dagger-refl*: $1 \leq x^{\dagger}$
  using *dagger-unfoldl local.join.sup.bounded-iff* **by** *blast*

lemma *dagger-plus-one* [*simp*]: $1 + x^{\dagger} = x^{\dagger}$
  by (*simp add*: *dagger-refl local.join.sup-absorb2*)

lemma *star-1l*: $x \cdot x^{\dagger} \leq x^{\dagger}$
  using *dagger-unfoldl local.join.sup.bounded-iff* **by** *blast*

lemma *star-1r*: $x^{\dagger} \cdot x \leq x^{\dagger}$
  using *dagger-unfoldr local.join.sup.bounded-iff* **by** *blast*

lemma *dagger-ext*: $x \leq x^{\dagger}$
  by (*metis dagger-unfoldl-distr local.join.sup.boundedE star-1r*)

lemma *dagger-trans-eq* [*simp*]: $x^{\dagger} \cdot x^{\dagger} = x^{\dagger}$
  by (*metis dagger-unfoldr-eq local.dagger-denest local.join.sup.idem mult-assoc*)

lemma *dagger-subdist*: $x^\dagger \le (x + y)^\dagger$
by (*metis dagger-unfoldr-distr local.dagger-denest local.order-prop*)

lemma *dagger-subdist-var*: $x^\dagger + y^\dagger \le (x + y)^\dagger$
using *dagger-subdist local.join.sup-commute* **by** *fastforce*

lemma *dagger-iso* [*intro*]: $x \le y \Longrightarrow x^\dagger \le y^\dagger$
by (*metis less-eq-def dagger-subdist*)

lemma *star-square*: $(x \cdot x)^\dagger \le x^\dagger$
by (*metis dagger-plus-one dagger-subdist dagger-trans-eq dagger-unfoldr-distr local.dagger-denest local.distrib-right′ local.eq-iff local.join.sup-commute local.less-eq-def local.mult-onel mult-assoc*)

lemma *dagger-rtc1-eq* [*simp*]: $1 + x + x^\dagger \cdot x^\dagger = x^\dagger$
by (*simp add*: *local.dagger-ext local.dagger-refl local.join.sup-absorb2*)

Nitpick refutes the next lemmas.

lemma $y + y \cdot x^\dagger \cdot x = y \cdot x^\dagger$

oops

lemma $y \cdot x^\dagger = y + y \cdot x \cdot x^\dagger$

oops

lemma $(x + y)^\dagger = x^\dagger \cdot (y \cdot x^\dagger)^\dagger$

oops

lemma $(x^\dagger)^\dagger = x^\dagger$

oops

lemma $(1 + x)^\star = x^\star$

oops

lemma $x^\dagger \cdot x = x \cdot x^\dagger$

oops

end

6.2 Pre-Conway Algebras

class *pre-conway-base* = *near-conway-base* + *pre-dioid-one*

begin

lemma *dagger-subdist-var-3*: $x^\dagger \cdot y^\dagger \leq (x + y)^\dagger$
using *local.dagger-subdist-var local.mult-isol-var* **by** *fastforce*

lemma *dagger-subdist-var-2*: $x \cdot y \leq (x + y)^\dagger$
by (*meson dagger-subdist-var-3 local.dagger-ext local.mult-isol-var local.order.trans*)

lemma *dagger-sum-unfold* [*simp*]: $x^\dagger + x^\dagger \cdot y \cdot (x + y)^\dagger = (x + y)^\dagger$
by (*metis local.dagger-denest local.dagger-unfoldl-distr mult-assoc*)

end

6.3 Conway Algebras

class *conway-base* = *pre-conway-base* + *dioid-one*

begin

lemma *troeger*: $(x + y)^\dagger \cdot z = x^\dagger \cdot (y \cdot (x + y)^\dagger \cdot z + z)$
proof −
have $(x + y)^\dagger \cdot z = x^\dagger \cdot z + x^\dagger \cdot y \cdot (x + y)^\dagger \cdot z$
by (*metis dagger-sum-unfold local.distrib-right′*)
thus *?thesis*
by (*metis add-commute local.distrib-left mult-assoc*)
qed

lemma *dagger-slide-var1*: $x^\dagger \cdot x \leq x \cdot x^\dagger$
by (*metis local.dagger-unfoldl-distr local.dagger-unfoldr-eq local.distrib-left local.eq-iff local.mult-1-right mult-assoc*)

lemma *dagger-slide-var1-eq*: $x^\dagger \cdot x = x \cdot x^\dagger$
by (*metis local.dagger-unfoldl-distr local.dagger-unfoldr-eq local.distrib-left local.mult-1-right mult-assoc*)

lemma *dagger-slide-eq*: $(x \cdot y)^\dagger \cdot x = x \cdot (y \cdot x)^\dagger$
proof −
have $(x \cdot y)^\dagger \cdot x = x + x \cdot (y \cdot x)^\dagger \cdot y \cdot x$
by (*metis local.dagger-prod-unfold local.distrib-right′ local.mult-onel*)
also have ... $= x \cdot (1 + (y \cdot x)^\dagger \cdot y \cdot x)$
using *local.distrib-left local.mult-1-right mult-assoc* **by** *presburger*
finally show *?thesis*
by (*simp add*: *mult-assoc*)
qed

end

6.4 Conway Algebras with Zero

class *near-conway-base-zerol* = *near-conway-base* + *near-dioid-one-zerol*

begin

lemma *dagger-annil* [*simp*]: $1 + x \cdot 0 = (x \cdot 0)^\dagger$
by (*metis annil dagger-unfoldl-eq mult.assoc*)

lemma *zero-dagger* [*simp*]: $0^\dagger = 1$
by (*metis add-0-right annil dagger-annil*)

end

class *pre-conway-base-zerol* = *near-conway-base-zerol* + *pre-dioid*

class *conway-base-zerol* = *pre-conway-base-zerol* + *dioid*

subclass (**in** *pre-conway-base-zerol*) *pre-conway-base* **..**

subclass (**in** *conway-base-zerol*) *conway-base* **..**

context *conway-base-zerol*
begin

lemma $z \cdot x \leq y \cdot z \Longrightarrow z \cdot x^\dagger \leq y^\dagger \cdot z$

oops

end

6.5 Conway Algebras with Simulation

class *near-conway* = *near-conway-base* +
assumes *dagger-simr*: $z \cdot x \leq y \cdot z \Longrightarrow z \cdot x^\dagger \leq y^\dagger \cdot z$

begin

lemma *dagger-slide-var*: $x \cdot (y \cdot x)^\dagger \leq (x \cdot y)^\dagger \cdot x$
by (*metis eq-refl dagger-simr mult.assoc*)

Nitpick refutes the next lemma.

lemma *dagger-slide*: $x \cdot (y \cdot x)^\dagger = (x \cdot y)^\dagger \cdot x$

oops

We say that y preserves x if $x \cdot y \cdot x = x \cdot y$ and $!x \cdot y \cdot !x = !x \cdot y$. This definition is taken from Solin [26]. It is useful for program transformation.

lemma *preservation1*: $x \cdot y \leq x \cdot y \cdot x \Longrightarrow x \cdot y^\dagger \leq (x \cdot y + z)^\dagger \cdot x$
proof −
assume $x \cdot y \leq x \cdot y \cdot x$
hence $x \cdot y \leq (x \cdot y + z) \cdot x$

by (*simp add*: *local.join.le-supI1*)
thus *?thesis*
by (*simp add*: *local.dagger-simr*)
qed

end

class *near-conway-zerol* = *near-conway* + *near-dioid-one-zerol*

class *pre-conway* = *near-conway* + *pre-dioid-one*

begin

subclass *pre-conway-base* **..**

lemma *dagger-slide*: $x \cdot (y \cdot x)^\dagger = (x \cdot y)^\dagger \cdot x$
by (*metis add.commute dagger-prod-unfold join.sup-least mult-1-right mult.assoc subdistl dagger-slide-var dagger-unfoldl-distr antisym*)

lemma *dagger-denest2*: $(x + y)^\dagger = x^\dagger \cdot (y \cdot x^\dagger)^\dagger$
by (*metis dagger-denest dagger-slide*)

lemma *preservation2*: $y \cdot x \leq y \Longrightarrow (x \cdot y)^\dagger \cdot x \leq x \cdot y^\dagger$
by (*metis dagger-slide local.dagger-iso local.mult-isol*)

lemma *preservation1-eq*: $x \cdot y \leq x \cdot y \cdot x \Longrightarrow y \cdot x \leq y \Longrightarrow (x \cdot y)^\dagger \cdot x = x \cdot y^\dagger$
by (*simp add*: *local.dagger-simr local.eq-iff preservation2*)

end

class *pre-conway-zerol* = *near-conway-zerol* + *pre-dioid-one-zerol*

begin

subclass *pre-conway* **..**

end

class *conway* = *pre-conway* + *dioid-one*

class *conway-zerol* = *pre-conway* + *dioid-one-zerol*

begin

subclass *conway-base* **..**

Nitpick refutes the next lemmas.

lemma $1 = 1^\dagger$

oops

lemma $(x^{\dagger})^{\dagger} = x^{\dagger}$

oops

lemma *dagger-denest-var* [*simp*]: $(x + y)^{\dagger} = (x^{\dagger} \cdot y^{\dagger})^{\dagger}$

oops

lemma *star2* [*simp*]: $(1 + x)^{\dagger} = x^{\dagger}$

oops

end

end

7 Kleene Algebras

theory *Kleene-Algebra*
imports *Conway*
begin

7.1 Left Near Kleene Algebras

Extending the hierarchy developed in *Kleene-Algebra.Dioid* we now add an operation of Kleene star, finite iteration, or reflexive transitive closure to variants of Dioids. Since a multiplicative unit is needed for defining the star we only consider variants with 1; 0 can be added separately. We consider the left star induction axiom and the right star induction axiom independently since in some applications, e.g., Salomaa's axioms, probabilistic Kleene algebras, or completeness proofs with respect to the equational theoy of regular expressions and regular languages, the right star induction axiom is not needed or not valid.

We start with near dioids, then consider pre-dioids and finally dioids. It turns out that many of the known laws of Kleene algebras hold already in these more general settings. In fact, all our equational theorems have been proved within left Kleene algebras, as expected.

Although most of the proofs in this file could be fully automated by Sledgehammer and Metis, we display step-wise proofs as they would appear in a text book. First, this file may then be useful as a reference manual on Kleene algebra. Second, it is better protected against changes in the underlying theories and supports easy translation of proofs into other settings.

class *left-near-kleene-algebra* = *near-dioid-one* + *star-op* +
  assumes *star-unfoldl*: $1 + x \cdot x^{\star} \leq x^{\star}$

and *star-inductl*: $z + x \cdot y \leq y \Longrightarrow x^\star \cdot z \leq y$

begin

First we prove two immediate consequences of the unfold axiom. The first one states that starred elements are reflexive.

lemma *star-ref* [*simp*]: $1 \leq x^\star$
using *star-unfoldl* **by** *auto*

Reflexivity of starred elements implies, by definition of the order, that 1 is an additive unit for starred elements.

lemma *star-plus-one* [*simp*]: $1 + x^\star = x^\star$
using *less-eq-def star-ref* **by** *blast*

lemma *star-1l* [*simp*]: $x \cdot x^\star \leq x^\star$
using *star-unfoldl* **by** *auto*

lemma $x^\star \cdot x \leq x^\star$

oops

lemma $x \cdot x^\star = x^\star$

oops

Next we show that starred elements are transitive.

lemma *star-trans-eq* [*simp*]: $x^\star \cdot x^\star = x^\star$
proof (*rule antisym*) — this splits an equation into two inequalities
have $x^\star + x \cdot x^\star \leq x^\star$
by *auto*
thus $x^\star \cdot x^\star \leq x^\star$
by (*simp add*: *star-inductl*)
next show $x^\star \leq x^\star \cdot x^\star$
using *mult-isor star-ref* **by** *fastforce*
qed

lemma *star-trans*: $x^\star \cdot x^\star \leq x^\star$
by *simp*

We now derive variants of the star induction axiom.

lemma *star-inductl-var*: $x \cdot y \leq y \Longrightarrow x^\star \cdot y \leq y$
proof −
assume $x \cdot y \leq y$
hence $y + x \cdot y \leq y$
by *simp*
thus $x^\star \cdot y \leq y$
by (*simp add*: *star-inductl*)
qed

lemma *star-inductl-var-equiv* [*simp*]: $x^\star \cdot y \leq y \longleftrightarrow x \cdot y \leq y$
proof
  assume $x \cdot y \leq y$
  thus $x^\star \cdot y \leq y$
    by (*simp add*: *star-inductl-var*)
next
  assume $x^\star \cdot y \leq y$
  hence $x^\star \cdot y = y$
    by (*metis eq-iff mult-1-left mult-isor star-ref*)
  moreover hence $x \cdot y = x \cdot x^\star \cdot y$
    by (*simp add*: *mult.assoc*)
  moreover have ... $\leq x^\star \cdot y$
    by (*metis mult-isor star-1l*)
  ultimately show $x \cdot y \leq y$
    by *auto*
qed

lemma *star-inductl-var-eq*: $x \cdot y = y \Longrightarrow x^\star \cdot y \leq y$
  by (*metis eq-iff star-inductl-var*)

lemma *star-inductl-var-eq2*: $y = x \cdot y \Longrightarrow y = x^\star \cdot y$
proof −
  assume *hyp*: $y = x \cdot y$
  hence $y \leq x^\star \cdot y$
    using *mult-isor star-ref* **by** *fastforce*
  thus $y = x^\star \cdot y$
    using *hyp eq-iff* **by** *auto*
qed

lemma $y = x \cdot y \longleftrightarrow y = x^\star \cdot y$

  oops

lemma $x^\star \cdot z \leq y \Longrightarrow z + x \cdot y \leq y$

  oops

lemma *star-inductl-one*: $1 + x \cdot y \leq y \Longrightarrow x^\star \leq y$
  using *star-inductl* **by** *force*

lemma *star-inductl-star*: $x \cdot y^\star \leq y^\star \Longrightarrow x^\star \leq y^\star$
  by (*simp add*: *star-inductl-one*)

lemma *star-inductl-eq*: $z + x \cdot y = y \Longrightarrow x^\star \cdot z \leq y$
  by (*simp add*: *star-inductl*)

We now prove two facts related to 1.

lemma *star-subid*: $x \leq 1 \Longrightarrow x^\star = 1$

proof −
  assume $x \leq 1$
  hence $1 + x \cdot 1 \leq 1$
    by *simp*
  hence $x^\star \leq 1$
    by (*metis mult-oner star-inductl*)
  thus $x^\star = 1$
    by (*simp add*: *order.antisym*)
qed

lemma *star-one* [*simp*]: $1^\star = 1$
  by (*simp add*: *star-subid*)

We now prove a subdistributivity property for the star (which is equivalent to isotonicity of star).

lemma *star-subdist*: $x^\star \leq (x + y)^\star$
proof −
  have $x \cdot (x + y)^\star \leq (x + y) \cdot (x + y)^\star$
    by *simp*
  also have ... $\leq (x + y)^\star$
    by (*metis star-1l*)
  thus *?thesis*
    using *calculation order-trans star-inductl-star* **by** *blast*
qed

lemma *star-subdist-var*: $x^\star + y^\star \leq (x + y)^\star$
  using *join.sup-commute star-subdist* **by** *force*

lemma *star-iso* [*intro*]: $x \leq y \Longrightarrow x^\star \leq y^\star$
  by (*metis less-eq-def star-subdist*)

We now prove some more simple properties.

lemma *star-invol* [*simp*]: $(x^\star)^\star = x^\star$
proof (*rule antisym*)
  have $x^\star \cdot x^\star = x^\star$
    by (*fact star-trans-eq*)
  thus $(x^\star)^\star \leq x^\star$
    by (*simp add*: *star-inductl-star*)
  have$(x^\star)^\star \cdot (x^\star)^\star \leq (x^\star)^\star$
    by (*fact star-trans*)
  hence $x \cdot (x^\star)^\star \leq (x^\star)^\star$
    by (*meson star-inductl-var-equiv*)
  thus $x^\star \leq (x^\star)^\star$
    by (*simp add*: *star-inductl-star*)
qed

lemma *star2* [*simp*]: $(1 + x)^\star = x^\star$
proof (*rule antisym*)
  show $x^\star \leq (1 + x)^\star$

by *auto*
have $x^\star + x \cdot x^\star \leq x^\star$
by *simp*
thus $(1 + x)^\star \leq x^\star$
by (*simp add*: *star-inductl-star*)
qed

lemma $1 + x^\star \cdot x \leq x^\star$

oops

lemma $x \leq x^\star$

oops

lemma $x^\star \cdot x \leq x^\star$

oops

lemma $1 + x \cdot x^\star = x^\star$

oops

lemma $x \cdot z \leq z \cdot y \Longrightarrow x^\star \cdot z \leq z \cdot y^\star$

oops

The following facts express inductive conditions that are used to show that $(x + y)^\star$ is the greatest term that can be built from x and y.

lemma *prod-star-closure*: $x \leq z^\star \Longrightarrow y \leq z^\star \Longrightarrow x \cdot y \leq z^\star$
proof −
assume *assm*: $x \leq z^\star$ $y \leq z^\star$
hence $y + z^\star \cdot z^\star \leq z^\star$
by *simp*
hence $z^\star \cdot y \leq z^\star$
by (*simp add*: *star-inductl*)
also have $x \cdot y \leq z^\star \cdot y$
by (*simp add*: *assm mult-isor*)
thus $x \cdot y \leq z^\star$
using *calculation order.trans* **by** *blast*
qed

lemma *star-star-closure*: $x^\star \leq z^\star \Longrightarrow (x^\star)^\star \leq z^\star$
by (*metis star-invol*)

lemma *star-closed-unfold*: $x^\star = x \Longrightarrow x = 1 + x \cdot x$
by (*metis star-plus-one star-trans-eq*)

lemma $x^\star = x \longleftrightarrow x = 1 + x \cdot x$

oops

end

7.2 Left Pre-Kleene Algebras

class *left-pre-kleene-algebra* = *left-near-kleene-algebra* + *pre-dioid-one*

begin

We first prove that the star operation is extensive.

lemma *star-ext* [*simp*]: $x \leq x^\star$
proof −
  have $x \leq x \cdot x^\star$
    by (*metis mult-oner mult-isol star-ref*)
  thus *?thesis*
    by (*metis order-trans star-1l*)
qed

We now prove a right star unfold law.

lemma *star-1r* [*simp*]: $x^\star \cdot x \leq x^\star$
proof −
  have $x + x \cdot x^\star \leq x^\star$
    by *simp*
  thus *?thesis*
    by (*fact star-inductl*)
qed

lemma *star-unfoldr*: $1 + x^\star \cdot x \leq x^\star$
  by *simp*

lemma $1 + x^\star \cdot x = x^\star$

  oops

Next we prove a simulation law for the star. It is instrumental in proving further properties.

lemma *star-sim1*: $x \cdot z \leq z \cdot y \Longrightarrow x^\star \cdot z \leq z \cdot y^\star$
proof −
  assume $x \cdot z \leq z \cdot y$
  hence $x \cdot z \cdot y^\star \leq z \cdot y \cdot y^\star$
    by (*simp add*: *mult-isor*)
  also have ... $\leq z \cdot y^\star$
    by (*simp add*: *mult-isol mult-assoc*)
  finally have $x \cdot z \cdot y^\star \leq z \cdot y^\star$
    by *simp*
  hence $z + x \cdot z \cdot y^\star \leq z \cdot y^\star$
    by (*metis join.sup-least mult-isol mult-oner star-ref*)

thus $x^\star \cdot z \leq z \cdot y^\star$
  by (*simp add*: *star-inductl mult-assoc*)
qed

The next lemma is used in omega algebras to prove, for instance, Bachmair and Dershowitz's separation of termination theorem [4]. The property at the left-hand side of the equivalence is known as *quasicommutation*.

lemma *quasicomm-var*: $y \cdot x \leq x \cdot (x + y)^\star \longleftrightarrow y^\star \cdot x \leq x \cdot (x + y)^\star$
proof
  assume $y \cdot x \leq x \cdot (x + y)^\star$
  thus $y^\star \cdot x \leq x \cdot (x + y)^\star$
    using *star-sim1* **by** *force*
next
  assume $y^\star \cdot x \leq x \cdot (x + y)^\star$
  thus $y \cdot x \leq x \cdot (x + y)^\star$
    by (*meson mult-isor order-trans star-ext*)
qed

lemma *star-slide1*: $(x \cdot y)^\star \cdot x \leq x \cdot (y \cdot x)^\star$
  by (*simp add*: *mult-assoc star-sim1*)

lemma $(x \cdot y)^\star \cdot x = x \cdot (y \cdot x)^\star$

  oops

lemma *star-slide-var1*: $x^\star \cdot x \leq x \cdot x^\star$
  by (*simp add*: *star-sim1*)

We now show that the (left) star unfold axiom can be strengthened to an equality.

lemma *star-unfoldl-eq* [*simp*]: $1 + x \cdot x^\star = x^\star$
proof (*rule antisym*)
  show $1 + x \cdot x^\star \leq x^\star$
    by (*fact star-unfoldl*)
  have $1 + x \cdot (1 + x \cdot x^\star) \leq 1 + x \cdot x^\star$
    by (*meson join.sup-mono eq-refl mult-isol star-unfoldl*)
  thus $x^\star \leq 1 + x \cdot x^\star$
    by (*simp add*: *star-inductl-one*)
qed

lemma $1 + x^\star \cdot x = x^\star$

  oops

Next we relate the star and the reflexive transitive closure operation.

lemma *star-rtc1-eq* [*simp*]: $1 + x + x^\star \cdot x^\star = x^\star$
  by (*simp add*: *join.sup.absorb2*)

lemma *star-rtc1*: $1 + x + x^\star \cdot x^\star \leq x^\star$
  by *simp*

lemma *star-rtc2*: $1 + x \cdot x \leq x \longleftrightarrow x = x^\star$
proof
  assume $1 + x \cdot x \leq x$
  thus $x = x^\star$
    by (*simp add*: *local.eq-iff local.star-inductl-one*)
next
  assume $x = x^\star$
  thus $1 + x \cdot x \leq x$
    using *local.star-closed-unfold* **by** *auto*
qed

lemma *star-rtc3*: $1 + x \cdot x = x \longleftrightarrow x = x^\star$
  by (*metis order-refl star-plus-one star-rtc2 star-trans-eq*)

lemma *star-rtc-least*: $1 + x + y \cdot y \leq y \Longrightarrow x^\star \leq y$
proof −
  assume $1 + x + y \cdot y \leq y$
  hence $1 + x \cdot y \leq y$
    by (*metis join.le-sup-iff mult-isol-var star-trans-eq star-rtc2*)
  thus $x^\star \leq y$
    by (*simp add*: *star-inductl-one*)
qed

lemma *star-rtc-least-eq*: $1 + x + y \cdot y = y \Longrightarrow x^\star \leq y$
  by (*simp add*: *star-rtc-least*)

lemma $1 + x + y \cdot y \leq y \longleftrightarrow x^\star \leq y$

  oops

The next lemmas are again related to closure conditions

lemma *star-subdist-var-1*: $x \leq (x + y)^\star$
  by (*meson join.sup.boundedE star-ext*)

lemma *star-subdist-var-2*: $x \cdot y \leq (x + y)^\star$
  by (*meson join.sup.boundedE prod-star-closure star-ext*)

lemma *star-subdist-var-3*: $x^\star \cdot y^\star \leq (x + y)^\star$
  by (*simp add*: *prod-star-closure star-iso*)

We now prove variants of sum-elimination laws under a star. These are also known a denesting laws or as sum-star laws.

lemma *star-denest* [*simp*]: $(x + y)^\star = (x^\star \cdot y^\star)^\star$
proof (*rule antisym*)
  have $x + y \leq x^\star \cdot y^\star$

by (*metis join.sup.bounded-iff mult-1-right mult-isol-var mult-onel star-ref star-ext*)
  thus $(x + y)^\star \le (x^\star \cdot y^\star)^\star$
    by (*fact star-iso*)
  have $x^\star \cdot y^\star \le (x + y)^\star$
    by (*fact star-subdist-var-3*)
  thus $(x^\star \cdot y^\star)^\star \le (x + y)^\star$
    by (*simp add*: *prod-star-closure star-inductl-star*)
qed

lemma *star-sum-var* [*simp*]: $(x^\star + y^\star)^\star = (x + y)^\star$
  by *simp*

lemma *star-denest-var* [*simp*]: $x^\star \cdot (y \cdot x^\star)^\star = (x + y)^\star$
proof (*rule antisym*)
  have $1 \le x^\star \cdot (y \cdot x^\star)^\star$
    by (*metis mult-isol-var mult-oner star-ref*)
  moreover have $x \cdot x^\star \cdot (y \cdot x^\star)^\star \le x^\star \cdot (y \cdot x^\star)^\star$
    by (*simp add*: *mult-isor*)
  moreover have $y \cdot x^\star \cdot (y \cdot x^\star)^\star \le x^\star \cdot (y \cdot x^\star)^\star$
    by (*metis mult-isol-var mult-onel star-1l star-ref*)
  ultimately have $1 + (x + y) \cdot x^\star \cdot (y \cdot x^\star)^\star \le x^\star \cdot (y \cdot x^\star)^\star$
    by *auto*
  thus $(x + y)^\star \le x^\star \cdot (y \cdot x^\star)^\star$
    by (*metis mult.assoc mult-oner star-inductl*)
  have $(y \cdot x^\star)^\star \le (y^\star \cdot x^\star)^\star$
    by (*simp add*: *mult-isol-var star-iso*)
  hence $(y \cdot x^\star)^\star \le (x + y)^\star$
    by (*metis add.commute star-denest*)
  moreover have $x^\star \le (x + y)^\star$
    by (*fact star-subdist*)
  ultimately show $x^\star \cdot (y \cdot x^\star)^\star \le (x + y)^\star$
    using *prod-star-closure* **by** *blast*
qed

lemma *star-denest-var-2* [*simp*]: $x^\star \cdot (y \cdot x^\star)^\star = (x^\star \cdot y^\star)^\star$
  by *simp*

lemma *star-denest-var-3* [*simp*]: $x^\star \cdot (y^\star \cdot x^\star)^\star = (x^\star \cdot y^\star)^\star$
  by *simp*

lemma *star-denest-var-4* [*ac-simps*]: $(y^\star \cdot x^\star)^\star = (x^\star \cdot y^\star)^\star$
  by (*metis add-comm star-denest*)

lemma *star-denest-var-5* [*ac-simps*]: $x^\star \cdot (y \cdot x^\star)^\star = y^\star \cdot (x \cdot y^\star)^\star$
  by (*simp add*: *star-denest-var-4*)

lemma $x^\star \cdot (y \cdot x^\star)^\star = (x^\star \cdot y)^\star \cdot x^\star$

oops

lemma *star-denest-var-6* [*simp*]: $x^\star \cdot y^\star \cdot (x + y)^\star = (x + y)^\star$
  using *mult-assoc* **by** *simp*

lemma *star-denest-var-7* [*simp*]: $(x + y)^\star \cdot x^\star \cdot y^\star = (x + y)^\star$
proof (*rule antisym*)
  have $(x + y)^\star \cdot x^\star \cdot y^\star \leq (x + y)^\star \cdot (x^\star \cdot y^\star)^\star$
    by (*simp add*: *mult-assoc*)
  thus $(x + y)^\star \cdot x^\star \cdot y^\star \leq (x + y)^\star$
    by *simp*
  have $1 \leq (x + y)^\star \cdot x^\star \cdot y^\star$
    by (*metis dual-order.trans mult-1-left mult-isor star-ref*)
  moreover have $(x + y) \cdot (x + y)^\star \cdot x^\star \cdot y^\star \leq (x + y)^\star \cdot x^\star \cdot y^\star$
    using *mult-isor star-1l* **by** *presburger*
  ultimately have $1 + (x + y) \cdot (x + y)^\star \cdot x^\star \cdot y^\star \leq (x + y)^\star \cdot x^\star \cdot y^\star$
    by *simp*
  thus $(x + y)^\star \leq (x + y)^\star \cdot x^\star \cdot y^\star$
    by (*metis mult.assoc star-inductl-one*)
qed

lemma *star-denest-var-8* [*simp*]: $x^\star \cdot y^\star \cdot (x^\star \cdot y^\star)^\star = (x^\star \cdot y^\star)^\star$
  by (*simp add*: *mult-assoc*)

lemma *star-denest-var-9* [*simp*]: $(x^\star \cdot y^\star)^\star \cdot x^\star \cdot y^\star = (x^\star \cdot y^\star)^\star$
  using *star-denest-var-7* **by** *simp*

The following statements are well known from term rewriting. They are all variants of the Church-Rosser theorem in Kleene algebra [27]. But first we prove a law relating two confluence properties.

lemma *confluence-var* [*iff*]: $y \cdot x^\star \leq x^\star \cdot y^\star \longleftrightarrow y^\star \cdot x^\star \leq x^\star \cdot y^\star$
proof
  assume $y \cdot x^\star \leq x^\star \cdot y^\star$
  thus $y^\star \cdot x^\star \leq x^\star \cdot y^\star$
    using *star-sim1* **by** *fastforce*
next
  assume $y^\star \cdot x^\star \leq x^\star \cdot y^\star$
  thus $y \cdot x^\star \leq x^\star \cdot y^\star$
    by (*meson mult-isor order-trans star-ext*)
qed

lemma *church-rosser* [*intro*]: $y^\star \cdot x^\star \leq x^\star \cdot y^\star \Longrightarrow (x + y)^\star = x^\star \cdot y^\star$
proof (*rule antisym*)
  assume $y^\star \cdot x^\star \leq x^\star \cdot y^\star$
  hence $x^\star \cdot y^\star \cdot (x^\star \cdot y^\star) \leq x^\star \cdot x^\star \cdot y^\star \cdot y^\star$
    by (*metis mult-isol mult-isor mult.assoc*)
  hence $x^\star \cdot y^\star \cdot (x^\star \cdot y^\star) \leq x^\star \cdot y^\star$
    by (*simp add*: *mult-assoc*)

moreover have $1 \leq x^\star \cdot y^\star$
by (*metis dual-order.trans mult-1-right mult-isol star-ref*)
ultimately have $1 + x^\star \cdot y^\star \cdot (x^\star \cdot y^\star) \leq x^\star \cdot y^\star$
by *simp*
hence $(x^\star \cdot y^\star)^\star \leq x^\star \cdot y^\star$
by (*simp add*: *star-inductl-one*)
thus $(x + y)^\star \leq x^\star \cdot y^\star$
by *simp*
thus $x^\star \cdot y^\star \leq (x + y)^\star$
by *simp*
qed

lemma *church-rosser-var*: $y \cdot x^\star \leq x^\star \cdot y^\star \Longrightarrow (x + y)^\star = x^\star \cdot y^\star$
by *fastforce*

lemma *church-rosser-to-confluence*: $(x + y)^\star \leq x^\star \cdot y^\star \Longrightarrow y^\star \cdot x^\star \leq x^\star \cdot y^\star$
by (*metis add-comm eq-iff star-subdist-var-3*)

lemma *church-rosser-equiv*: $y^\star \cdot x^\star \leq x^\star \cdot y^\star \longleftrightarrow (x + y)^\star = x^\star \cdot y^\star$
using *church-rosser-to-confluence eq-iff* **by** *blast*

lemma *confluence-to-local-confluence*: $y^\star \cdot x^\star \leq x^\star \cdot y^\star \Longrightarrow y \cdot x \leq x^\star \cdot y^\star$
by (*meson mult-isol-var order-trans star-ext*)

lemma $y \cdot x \leq x^\star \cdot y^\star \Longrightarrow y^\star \cdot x^\star \leq x^\star \cdot y^\star$

oops

lemma $y \cdot x \leq x^\star \cdot y^\star \Longrightarrow (x + y)^\star \leq x^\star \cdot y^\star$
oops

More variations could easily be proved. The last counterexample shows that Newman's lemma needs a wellfoundedness assumption. This is well known.

The next lemmas relate the reflexive transitive closure and the transitive closure.

lemma *sup-id-star1*: $1 \leq x \Longrightarrow x \cdot x^\star = x^\star$
proof −
assume $1 \leq x$
hence $x^\star \leq x \cdot x^\star$
using *mult-isor* **by** *fastforce*
thus $x \cdot x^\star = x^\star$
by (*simp add*: *eq-iff*)
qed

lemma *sup-id-star2*: $1 \leq x \Longrightarrow x^\star \cdot x = x^\star$
by (*metis order.antisym mult-isol mult-oner star-1r*)

lemma $1 + x^\star \cdot x = x^\star$

oops

lemma $(x \cdot y)^\star \cdot x = x \cdot (y \cdot x)^\star$

oops

lemma $x \cdot x = x \Longrightarrow x^\star = 1 + x$

oops

end

7.3 Left Kleene Algebras

class *left-kleene-algebra* = *left-pre-kleene-algebra* + *dioid-one*

begin

In left Kleene algebras the non-fact $z + y \cdot x \le y \Longrightarrow z \cdot x^\star \le y$ is a good challenge for counterexample generators. A model of left Kleene algebras in which the right star induction law does not hold has been given by Kozen [20].

We now show that the right unfold law becomes an equality.

lemma *star-unfoldr-eq* [*simp*]: $1 + x^\star \cdot x = x^\star$
proof (*rule antisym*)
  show $1 + x^\star \cdot x \le x^\star$
    by (*fact star-unfoldr*)
  have $1 + x \cdot (1 + x^\star \cdot x) = 1 + (1 + x \cdot x^\star) \cdot x$
    using *distrib-left distrib-right mult-1-left mult-1-right mult-assoc* **by** *presburger*
  also have ... $= 1 + x^\star \cdot x$
    by *simp*
  finally show $x^\star \le 1 + x^\star \cdot x$
    by (*simp add*: *star-inductl-one*)
qed

The following more complex unfold law has been used as an axiom, called prodstar, by Conway [8].

lemma *star-prod-unfold* [*simp*]: $1 + x \cdot (y \cdot x)^\star \cdot y = (x \cdot y)^\star$
proof (*rule antisym*)
  have $(x \cdot y)^\star = 1 + (x \cdot y)^\star \cdot x \cdot y$
    by (*simp add*: *mult-assoc*)
  thus $(x \cdot y)^\star \le 1 + x \cdot (y \cdot x)^\star \cdot y$
    by (*metis join.sup-mono mult-isor order-refl star-slide1*)
  have $1 + x \cdot (y \cdot x)^\star \cdot y \le 1 + x \cdot y \cdot (x \cdot y)^\star$
    by (*metis join.sup-mono eq-refl mult.assoc mult-isol star-slide1*)
  thus $1 + x \cdot (y \cdot x)^\star \cdot y \le (x \cdot y)^\star$

by *simp*
qed

The slide laws, which have previously been inequalities, now become equations.

lemma *star-slide* [*ac-simps*]: $(x \cdot y)^\star \cdot x = x \cdot (y \cdot x)^\star$
proof –
  have $x \cdot (y \cdot x)^\star = x \cdot (1 + y \cdot (x \cdot y)^\star \cdot x)$
    by *simp*
  also have ... $= (1 + x \cdot y \cdot (x \cdot y)^\star) \cdot x$
    by (*simp add*: *distrib-left mult-assoc*)
  finally show *?thesis*
    by *simp*
qed

lemma *star-slide-var* [*ac-simps*]: $x^\star \cdot x = x \cdot x^\star$
  by (*metis mult-onel mult-oner star-slide*)

lemma *star-sum-unfold-var* [*simp*]: $1 + x^\star \cdot (x + y)^\star \cdot y^\star = (x + y)^\star$
  by (*metis star-denest star-denest-var-3 star-denest-var-4 star-plus-one star-slide*)

The following law shows how starred sums can be unfolded.

lemma *star-sum-unfold* [*simp*]: $x^\star + x^\star \cdot y \cdot (x + y)^\star = (x + y)^\star$
proof –
  have $(x + y)^\star = x^\star \cdot (y \cdot x^\star)^\star$
    by *simp*
  also have ... $= x^\star \cdot (1 + y \cdot x^\star \cdot (y \cdot x^\star)^\star)$
    by *simp*
  also have ... $= x^\star \cdot (1 + y \cdot (x + y)^\star)$
    by (*simp add*: *mult.assoc*)
  finally show *?thesis*
    by (*simp add*: *distrib-left mult-assoc*)
qed

The following property appears in process algebra.

lemma *troeger*: $(x + y)^\star \cdot z = x^\star \cdot (y \cdot (x + y)^\star \cdot z + z)$
proof –
  have $(x + y)^\star \cdot z = x^\star \cdot z + x^\star \cdot y \cdot (x + y)^\star \cdot z$
    by (*metis* (*full-types*) *distrib-right star-sum-unfold*)
  thus *?thesis*
    by (*simp add*: *add-commute distrib-left mult-assoc*)
qed

The following properties are related to a property from propositional dynamic logic which has been attributed to Albert Meyer [18]. Here we prove it as a theorem of Kleene algebra.

lemma *star-square*: $(x \cdot x)^\star \leq x^\star$
proof –

have $x \cdot x \cdot x^\star \leq x^\star$
  by (*simp add*: *prod-star-closure*)
thus *?thesis*
  by (*simp add*: *star-inductl-star*)
qed

lemma *meyer-1* [*simp*]: $(1 + x) \cdot (x \cdot x)^\star = x^\star$
proof (*rule antisym*)
  have $x \cdot\ (1 + x) \cdot (x \cdot x)^\star = x \cdot (x \cdot x)^\star + x \cdot x \cdot (x \cdot x)^\star$
    by (*simp add*: *distrib-left*)
  also have ... $\leq x \cdot (x \cdot x)^\star + (x \cdot x)^\star$
    using *join.sup-mono star-1l* **by** *blast*
  finally have $x \cdot\ (1 + x) \cdot (x \cdot x)^\star \leq (1 + x) \cdot (x \cdot x)^\star$
    by (*simp add*: *join.sup-commute*)
  moreover have $1 \leq (1 + x) \cdot (x \cdot x)^\star$
    using *join.sup.coboundedI1* **by** *auto*
  ultimately have $1 + x \cdot (1 + x) \cdot (x \cdot x)^\star \leq (1 + x) \cdot (x \cdot x)^\star$
    by *auto*
  thus $x^\star \leq (1 + x) \cdot (x \cdot x)^\star$
    by (*simp add*: *star-inductl-one mult-assoc*)
  show $(1 + x) \cdot (x \cdot x)^\star \leq x^\star$
    by (*simp add*: *prod-star-closure star-square*)
qed

The following lemma says that transitive elements are equal to their transitive closure.

lemma *tc*: $x \cdot x \leq x \Longrightarrow x^\star \cdot x = x$
proof −
  assume $x \cdot x \leq x$
  hence $x + x \cdot x \leq x$
    by *simp*
  hence $x^\star \cdot x \leq x$
    by (*fact star-inductl*)
  thus $x^\star \cdot x = x$
    by (*metis mult-isol mult-oner star-ref star-slide-var eq-iff*)
qed

lemma *tc-eq*: $x \cdot x = x \Longrightarrow x^\star \cdot x = x$
  by (*auto intro*: *tc*)

The next fact has been used by Boffa [6] to axiomatise the equational theory of regular expressions.

lemma *boffa-var*: $x \cdot x \leq x \Longrightarrow x^\star = 1 + x$
proof −
  assume $x \cdot x \leq x$
  moreover have $x^\star = 1 + x^\star \cdot x$
    by *simp*
  ultimately show $x^\star = 1 + x$
    by (*simp add*: *tc*)

qed

lemma *boffa*: $x \cdot x = x \Longrightarrow x^\star = 1 + x$
  by (*auto intro*: *boffa-var*)

end

7.4 Left Kleene Algebras with Zero

There are applications where only a left zero is assumed, for instance in the context of total correctness and for demonic refinement algebras [31].

class *left-kleene-algebra-zerol* = *left-kleene-algebra* + *dioid-one-zerol*
begin

sublocale *conway*: *near-conway-base-zerol star*
  by *standard* (*simp-all add*: *local.star-slide*)

lemma *star-zero* [*simp*]: $0^\star = 1$
  by (*rule local.conway.zero-dagger*)

In principle, 1 could therefore be defined from 0 in this setting.

end

class *left-kleene-algebra-zero* = *left-kleene-algebra-zerol* + *dioid-one-zero*

7.5 Pre-Kleene Algebras

Pre-Kleene algebras are essentially probabilistic Kleene algebras [24]. They have a weaker right star unfold axiom. We are still looking for theorems that could be proved in this setting.

class *pre-kleene-algebra* = *left-pre-kleene-algebra* +
  assumes *weak-star-unfoldr*: $z + y \cdot (x + 1) \leq y \Longrightarrow z \cdot x^\star \leq y$

7.6 Kleene Algebras

class *kleene-algebra-zerol* = *left-kleene-algebra-zerol* +
  assumes *star-inductr*: $z + y \cdot x \leq y \Longrightarrow z \cdot x^\star \leq y$

begin

lemma *star-sim2*: $z \cdot x \leq y \cdot z \Longrightarrow z \cdot x^\star \leq y^\star \cdot z$
proof −
  assume $z \cdot x \leq y \cdot z$
  hence $y^\star \cdot z \cdot x \leq y^\star \cdot y \cdot z$
    using *mult-isol mult-assoc* **by** *auto*

also have ... $\leq y^\star \cdot z$
by (*simp add*: *mult-isor*)
finally have $y^\star \cdot z \cdot x \leq y^\star \cdot z$
by *simp*
moreover have $z \leq y^\star \cdot z$
using *mult-isor star-ref* **by** *fastforce*
ultimately have $z + y^\star \cdot z \cdot x \leq y^\star \cdot z$
by *simp*
thus $z \cdot x^\star \leq y^\star \cdot z$
by (*simp add*: *star-inductr*)
qed

sublocale *conway*: *pre-conway star*
by *standard* (*simp add*: *star-sim2*)

lemma *star-inductr-var*: $y \cdot x \leq y \Longrightarrow y \cdot x^\star \leq y$
by (*simp add*: *star-inductr*)

lemma *star-inductr-var-equiv*: $y \cdot x \leq y \longleftrightarrow y \cdot x^\star \leq y$
by (*meson order-trans mult-isol star-ext star-inductr-var*)

lemma *star-sim3*: $z \cdot x = y \cdot z \Longrightarrow z \cdot x^\star = y^\star \cdot z$
by (*simp add*: *eq-iff star-sim1 star-sim2*)

lemma *star-sim4*: $x \cdot y \leq y \cdot x \Longrightarrow x^\star \cdot y^\star \leq y^\star \cdot x^\star$
by (*auto intro*: *star-sim1 star-sim2*)

lemma *star-inductr-eq*: $z + y \cdot x = y \Longrightarrow z \cdot x^\star \leq y$
by (*auto intro*: *star-inductr*)

lemma *star-inductr-var-eq*: $y \cdot x = y \Longrightarrow y \cdot x^\star \leq y$
by (*auto intro*: *star-inductr-var*)

lemma *star-inductr-var-eq2*: $y \cdot x = y \Longrightarrow y \cdot x^\star = y$
by (*metis mult-onel star-one star-sim3*)

lemma *bubble-sort*: $y \cdot x \leq x \cdot y \Longrightarrow (x + y)^\star = x^\star \cdot y^\star$
by (*fastforce intro*: *star-sim4*)

lemma *independence1*: $x \cdot y = 0 \Longrightarrow x^\star \cdot y = y$
proof −
assume $x \cdot y = 0$
moreover have $x^\star \cdot y = y + x^\star \cdot x \cdot y$
by (*metis distrib-right mult-onel star-unfoldr-eq*)
ultimately show $x^\star \cdot y = y$
by (*metis add-0-left add.commute join.sup-ge1 eq-iff star-inductl-eq*)
qed

lemma *independence2*: $x \cdot y = 0 \Longrightarrow x \cdot y^\star = x$

by (*metis annil mult-onel star-sim3 star-zero*)

lemma *lazycomm-var*: $y \cdot x \leq x \cdot (x + y)^\star + y \longleftrightarrow y \cdot x^\star \leq x \cdot (x + y)^\star + y$
proof
let $?t = x \cdot (x + y)^\star$
assume *hyp*: $y \cdot x \leq ?t + y$
have $(?t + y) \cdot x = ?t \cdot x + y \cdot x$
by (*fact distrib-right*)
also have ... $\leq ?t \cdot x + ?t + y$
using *hyp join.sup.coboundedI2 join.sup-assoc* **by** *auto*
also have ... $\leq ?t + y$
using *eq-refl join.sup-least join.sup-mono mult-isol prod-star-closure star-subdist-var-1 mult-assoc* **by** *presburger*
finally have $y + (?t + y) \cdot x \leq ?t + y$
by *simp*
thus $y \cdot x^\star \leq x \cdot (x + y)^\star + y$
by (*fact star-inductr*)
next
assume $y \cdot x^\star \leq x \cdot (x + y)^\star + y$
thus $y \cdot x \leq x \cdot (x + y)^\star + y$
using *dual-order.trans mult-isol star-ext* **by** *blast*
qed

lemma *arden-var*: $(\forall y\ v.\ y \leq x \cdot y + v \longrightarrow y \leq x^\star \cdot v) \Longrightarrow z = x \cdot z + w \Longrightarrow z = x^\star \cdot w$
by (*auto simp*: *add-comm eq-iff star-inductl-eq*)

lemma $(\forall x\ y.\ y \leq x \cdot y \longrightarrow y = 0) \Longrightarrow y \leq x \cdot y + z \Longrightarrow y \leq x^\star \cdot z$
by (*metis eq-refl mult-onel*)

end

Finally, here come the Kleene algebras à la Kozen [21]. We only prove quasi-identities in this section. Since left Kleene algebras are complete with respect to the equational theory of regular expressions and regular languages, all identities hold already without the right star induction axiom.

class *kleene-algebra* = *left-kleene-algebra-zero* +
assumes *star-inductr'*: $z + y \cdot x \leq y \Longrightarrow z \cdot x^\star \leq y$
begin

subclass *kleene-algebra-zerol*
by *standard* (*simp add*: *star-inductr'*)

sublocale *conway-zerol*: *conway star* **..**

The next lemma shows that opposites of Kleene algebras (i.e., Kleene algebras with the order of multiplication swapped) are again Kleene algebras.

lemma *dual-kleene-algebra*:
class.kleene-algebra $(+)$ $(\odot)$ *1 0* $(\leq)$ $(<)$ *star*

proof
fix $x\ y\ z :: 'a$
show $(x \odot y) \odot z = x \odot (y \odot z)$
by (*metis mult.assoc opp-mult-def*)
show $(x + y) \odot z = x \odot z + y \odot z$
by (*metis opp-mult-def distrib-left*)
show $1 \odot x = x$
by (*metis mult-oner opp-mult-def*)
show $x \odot 1 = x$
by (*metis mult-onel opp-mult-def*)
show $0 + x = x$
by (*fact add-zerol*)
show $0 \odot x = 0$
by (*metis annir opp-mult-def*)
show $x \odot 0 = 0$
by (*metis annil opp-mult-def*)
show $x + x = x$
by (*fact add-idem*)
show $x \odot (y + z) = x \odot y + x \odot z$
by (*metis distrib-right opp-mult-def*)
show $z \odot x \leq z \odot (x + y)$
by (*metis mult-isor opp-mult-def order-prop*)
show $1 + x \odot x^\star \leq x^\star$
by (*metis opp-mult-def order-refl star-slide-var star-unfoldl-eq*)
show $z + x \odot y \leq y \Longrightarrow x^\star \odot z \leq y$
by (*metis opp-mult-def star-inductr*)
show $z + y \odot x \leq y \Longrightarrow z \odot x^\star \leq y$
by (*metis opp-mult-def star-inductl*)
qed

end

We finish with some properties on (multiplicatively) commutative Kleene algebras. A chapter in Conway's book [8] is devoted to this topic.

class *commutative-kleene-algebra* = *kleene-algebra* +
assumes *mult-comm* [*ac-simps*]: $x \cdot y = y \cdot x$

begin

lemma *conway-c3* [*simp*]: $(x + y)^\star = x^\star \cdot y^\star$
using *church-rosser mult-comm* **by** *auto*

lemma *conway-c4*: $(x^\star \cdot y)^\star = 1 + x^\star \cdot y^\star \cdot y$
by (*metis conway-c3 star-denest-var star-prod-unfold*)

lemma *cka-1*: $(x \cdot y)^\star \leq x^\star \cdot y^\star$
by (*metis conway-c3 star-invol star-iso star-subdist-var-2*)

lemma *cka-2* [*simp*]: $x^\star \cdot (x^\star \cdot y)^\star = x^\star \cdot y^\star$

by (*metis conway-c3 mult-comm star-denest-var*)

lemma *conway-c4-var* [*simp*]: $(x^\star \cdot y^\star)^\star = x^\star \cdot y^\star$
by (*metis conway-c3 star-invol*)

lemma *conway-c2-var*: $(x \cdot y)^\star \cdot x \cdot y \cdot y^\star \leq (x \cdot y)^\star \cdot y^\star$
by (*metis mult-isor star-1r mult-assoc*)

lemma *conway-c2* [*simp*]: $(x \cdot y)^\star \cdot (x^\star + y^\star) = x^\star \cdot y^\star$
proof (*rule antisym*)
show $(x \cdot y)^\star \cdot (x^\star + y^\star) \leq x^\star \cdot y^\star$
by (*metis cka-1 conway-c3 prod-star-closure star-ext star-sum-var*)
have $x \cdot (x \cdot y)^\star \cdot (x^\star + y^\star) = x \cdot (x \cdot y)^\star \cdot (x^\star + 1 + y \cdot y^\star)$
by (*simp add*: *add-assoc*)
also have ... $= x \cdot (x \cdot y)^\star \cdot (x^\star + y \cdot y^\star)$
by (*simp add*: *add-commute*)
also have ... $= (x \cdot y)^\star \cdot (x \cdot x^\star) + (x \cdot y)^\star \cdot x \cdot y \cdot y^\star$
using *distrib-left mult-comm mult-assoc* **by** *force*
also have ... $\leq (x \cdot y)^\star \cdot x^\star + (x \cdot y)^\star \cdot x \cdot y \cdot y^\star$
using *add-iso mult-isol* **by** *force*
also have ... $\leq (x \cdot y)^\star \cdot x^\star + (x \cdot y)^\star \cdot y^\star$
using *conway-c2-var join.sup-mono* **by** *blast*
also have ... $= (x \cdot y)^\star \cdot (x^\star + y^\star)$
by (*simp add*: *distrib-left*)
finally have $x \cdot (x \cdot y)^\star \cdot (x^\star + y^\star) \leq (x \cdot y)^\star \cdot (x^\star + y^\star)$ **.**
moreover have $y^\star \leq (x \cdot y)^\star \cdot (x^\star + y^\star)$
by (*metis dual-order.trans join.sup-ge2 mult-1-left mult-isor star-ref*)
ultimately have $y^\star + x \cdot (x \cdot y)^\star \cdot (x^\star + y^\star) \leq (x \cdot y)^\star \cdot (x^\star + y^\star)$
by *simp*
thus $x^\star \cdot y^\star \leq (x \cdot y)^\star \cdot (x^\star + y^\star)$
by (*simp add*: *mult.assoc star-inductl*)
qed

end

end

8 Models of Kleene Algebras

theory *Kleene-Algebra-Models*
imports *Kleene-Algebra Dioid-Models*
begin

We now show that most of the models considered for dioids are also Kleene algebras. Some of the dioid models cannot be expanded, for instance max-plus and min-plus semirings, but we do not formalise this fact. We also currently do not show that formal powerseries and matrices form Kleene algebras.

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

8.1 Preliminary Lemmas

We first prove two induction-style statements for dioids that are useful for establishing the full induction laws. In the future these will live in a theory file on finite sums for Kleene algebras.

context *dioid-one-zero*
begin

lemma *power-inductl*: $z + x \cdot y \leq y \Longrightarrow (x \,\hat{}\, n) \cdot z \leq y$
proof (*induct n*)
  case *0* **show** *?case*
    using *0.prems* **by** *auto*
  case *Suc* **thus** *?case*
    by (*auto*, *metis mult.assoc mult-isol order-trans*)
qed

lemma *power-inductr*: $z + y \cdot x \leq y \Longrightarrow z \cdot (x \,\hat{}\, n) \leq y$
proof (*induct n*)
  case *0* **show** *?case*
    using *0.prems* **by** *auto*
  case *Suc*
  {
    fix *n*
    assume $z + y \cdot x \leq y \Longrightarrow z \cdot x \,\hat{}\, n \leq y$
      and $z + y \cdot x \leq y$
    hence $z \cdot x \,\hat{}\, n \leq y$
      by *auto*
    also have $z \cdot x \,\hat{}\, Suc\ n = z \cdot x \cdot x \,\hat{}\, n$
      by (*metis mult.assoc power-Suc*)
    moreover have ... $= (z \cdot x \,\hat{}\, n) \cdot x$
      by (*metis mult.assoc power-commutes*)
    moreover have ... $\leq y \cdot x$
      by (*metis calculation(1) mult-isor*)
    moreover have ... $\leq y$
      using ‹$z + y \cdot x \leq y$› **by** *auto*
    ultimately have $z \cdot x \,\hat{}\, Suc\ n \leq y$ **by** *auto*
  }
  thus *?case*
    by (*metis Suc*)
qed

end

8.2 The Powerset Kleene Algebra over a Monoid

We now show that the powerset dioid forms a Kleene algebra. The Kleene star is defined as in language theory.

lemma *Un-0-Suc*: $(\bigcup n.\ f\ n) = f\ 0 \cup (\bigcup n.\ f\ (Suc\ n))$
by *auto* (*metis not0-implies-Suc*)

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

definition *star-def*: $X^\star = (\bigcup n.\ X\ \hat{}\ n)$

lemma *star-elim*: $x \in X^\star \longleftrightarrow (\exists k.\ x \in X\ \hat{}\ k)$
by (*simp add*: *star-def*)

lemma *star-contl*: $X \cdot Y^\star = (\bigcup n.\ X \cdot Y\ \hat{}\ n)$
by (*auto simp add*: *star-elim c-prod-def*)

lemma *star-contr*: $X^\star \cdot Y = (\bigcup n.\ X\ \hat{}\ n \cdot Y)$
by (*auto simp add*: *star-elim c-prod-def*)

instance
proof
fix *X Y Z* :: *'a set*
show $1 + X \cdot X^\star \subseteq X^\star$
proof −
have $1 + X \cdot X^\star = (X\ \hat{}\ 0) \cup (\bigcup n.\ X\ \hat{}\ (Suc\ n))$
by (*auto simp add*: *star-def c-prod-def plus-set-def one-set-def*)
also have ... $= (\bigcup n.\ X\ \hat{}\ n)$
by (*metis Un-0-Suc*)
also have ... $= X^\star$
by (*simp only*: *star-def*)
finally show *?thesis*
by (*metis subset-refl*)
qed
next
fix *X Y Z* :: *'a set*
assume *hyp*: $Z + X \cdot Y \subseteq Y$
show $X^\star \cdot Z \subseteq Y$
by (*simp add*: *star-contr SUP-le-iff*) (*meson hyp dioid-one-zero-class.power-inductl*)
next
fix *X Y Z* :: *'a set*
assume *hyp*: $Z + Y \cdot X \subseteq Y$
show $Z \cdot X^\star \subseteq Y$
by (*simp add*: *star-contl SUP-le-iff*) (*meson dioid-one-zero-class.power-inductr hyp*)
qed

end

8.3 Language Kleene Algebras

We now specialise this fact to languages.

interpretation *lan-kleene-algebra*: *kleene-algebra* (+) (·) *1*::′*a lan 0* (⊆) (⊂) *star*
..

8.4 Regular Languages

. . . and further to regular languages. For the sake of simplicity we just copy in the axiomatisation of regular expressions by Krauss and Nipkow [23].

datatype ′*a rexp* =
  Zero
| *One*
| *Atom* ′*a*
| *Plus* ′*a rexp* ′*a rexp*
| *Times* ′*a rexp* ′*a rexp*
| *Star* ′*a rexp*

The interpretation map that induces regular languages as the images of regular expressions in the set of languages has also been adapted from there.

fun *lang* :: ′*a rexp* ⇒ ′*a lan* **where**
  lang Zero = *0* —
| *lang One* = *1* — []
| *lang* (*Atom a*) = {[*a*]}
| *lang* (*Plus x y*) = *lang x* + *lang y*
| *lang* (*Times x y*) = *lang x* · *lang y*
| *lang* (*Star x*) = (*lang x*)*

typedef ′*a reg-lan* = *range lang* :: ′*a lan set*
  by *auto*

setup-lifting *type-definition-reg-lan*

instantiation *reg-lan* :: (*type*) *kleene-algebra*
begin

  lift-definition *star-reg-lan* :: ′*a reg-lan* ⇒ ′*a reg-lan*
    is *star*
    by (*metis* (*hide-lams*, *no-types*) *image-iff lang.simps*(*6*) *rangeI*)

  lift-definition *zero-reg-lan* :: ′*a reg-lan*
    is *0*
    by (*metis lang.simps*(*1*) *rangeI*)

  lift-definition *one-reg-lan* :: ′*a reg-lan*
    is *1*
    by (*metis lang.simps*(*2*) *rangeI*)

lift-definition *less-eq-reg-lan* :: *'a reg-lan* ⇒ *'a reg-lan* ⇒ *bool*
is *less-eq* **.**

lift-definition *less-reg-lan* :: *'a reg-lan* ⇒ *'a reg-lan* ⇒ *bool*
is *less* **.**

lift-definition *plus-reg-lan* :: *'a reg-lan* ⇒ *'a reg-lan* ⇒ *'a reg-lan*
is *plus*
by (*metis* (*hide-lams*, *no-types*) *image-iff lang.simps*(*4*) *rangeI*)

lift-definition *times-reg-lan* :: *'a reg-lan* ⇒ *'a reg-lan* ⇒ *'a reg-lan*
is *times*
by (*metis* (*hide-lams*, *no-types*) *image-iff lang.simps*(*5*) *rangeI*)

instance
proof
fix $x\ y\ z$ :: *'a reg-lan*
show $x + y + z = x + (y + z)$
by *transfer* (*metis join-semilattice-class.add-assoc'*)
show $x + y = y + x$
by *transfer* (*metis join-semilattice-class.add-comm*)
show $x \cdot y \cdot z = x \cdot (y \cdot z)$
by *transfer* (*metis semigroup-mult-class.mult.assoc*)
show $(x + y) \cdot z = x \cdot z + y \cdot z$
by *transfer* (*metis semiring-class.distrib-right*)
show $1 \cdot x = x$
by *transfer* (*metis monoid-mult-class.mult-1-left*)
show $x \cdot 1 = x$
by *transfer* (*metis monoid-mult-class.mult-1-right*)
show $0 + x = x$
by *transfer* (*metis join-semilattice-zero-class.add-zero-l*)
show $0 \cdot x = 0$
by *transfer* (*metis ab-near-semiring-one-zerol-class.annil*)
show $x \cdot 0 = 0$
by *transfer* (*metis ab-near-semiring-one-zero-class.annir*)
show $x \leq y \longleftrightarrow x + y = y$
by *transfer* (*metis plus-ord-class.less-eq-def*)
show $x < y \longleftrightarrow x \leq y \wedge x \neq y$
by *transfer* (*metis plus-ord-class.less-def*)
show $x + x = x$
by *transfer* (*metis join-semilattice-class.add-idem*)
show $x \cdot (y + z) = x \cdot y + x \cdot z$
by *transfer* (*metis semiring-class.distrib-left*)
show $z \cdot x \leq z \cdot (x + y)$
by *transfer* (*metis pre-dioid-class.subdistl*)
show $1 + x \cdot x^\star \leq x^\star$
by *transfer* (*metis star-unfoldl*)
show $z + x \cdot y \leq y \Longrightarrow x^\star \cdot z \leq y$
by *transfer* (*metis star-inductl*)

show $z + y \cdot x \leq y \Longrightarrow z \cdot x^\star \leq y$
by *transfer* (*metis star-inductr*)
qed

end

interpretation *reg-lan-kleene-algebra*: *kleene-algebra* (+) (·) *1*::'*a reg-lan* *0* (≤) (<) *star* **..**

8.5 Relation Kleene Algebras

We now show that binary relations form Kleene algebras. While we could have used the reflexive transitive closure operation as the Kleene star, we prefer the equivalent definition of the star as the sum of powers. This essentially allows us to copy previous proofs.

lemma *power-is-relpow*: *rel-dioid.power X n* = *X ^^ n*
proof (*induct n*)
case *0* **show** *?case*
by (*metis rel-dioid.power-0 relpow.simps*(*1*))
case *Suc* **thus** *?case*
by (*metis rel-dioid.power-Suc2 relpow.simps*(*2*))
qed

lemma *rel-star-def*: *X^∗* = ($\bigcup$*n. rel-dioid.power X n*)
by (*simp add*: *power-is-relpow rtrancl-is-UN-relpow*)

lemma *rel-star-contl*: *X O Y^∗* = ($\bigcup$*n. X O rel-dioid.power Y n*)
by (*metis rel-star-def relcomp-UNION-distrib*)

lemma *rel-star-contr*: *X^∗ O Y* = ($\bigcup$*n.* (*rel-dioid.power X n*) *O Y*)
by (*metis rel-star-def relcomp-UNION-distrib2*)

interpretation *rel-kleene-algebra*: *kleene-algebra* (∪) (*O*) *Id* {} (⊆) (⊂) *rtrancl*
proof
fix *x y z* :: '*a rel*
show *Id* ∪ *x O* x^* ⊆ x^*
by (*metis order-refl r-comp-rtrancl-eq rtrancl-unfold*)
next
fix *x y z* :: '*a rel*
assume *z* ∪ *x O y* ⊆ *y*
thus x^* *O z* ⊆ *y*
by (*simp only*: *rel-star-contr*, *metis* (*lifting*) *SUP-le-iff rel-dioid.power-inductl*)
next
fix *x y z* :: '*a rel*
assume *z* ∪ *y O x* ⊆ *y*
thus *z O* x^* ⊆ *y*
by (*simp only*: *rel-star-contl*, *metis* (*lifting*) *SUP-le-iff rel-dioid.power-inductr*)
qed

8.6 Trace Kleene Algebras

Again, the proof that sets of traces form Kleene algebras follows the same schema.

definition *t-star* :: ('p, 'a) *trace set* ⇒ ('p, 'a) *trace set* **where**
t-star X ≡ ⋃ *n. trace-dioid.power X n*

lemma *t-star-elim*: *x* ∈ *t-star X* ⟷ (∃ *n. x* ∈ *trace-dioid.power X n*)
by (*simp add*: *t-star-def*)

lemma *t-star-contl*: *t-prod X* (*t-star Y*) = (⋃ *n. t-prod X* (*trace-dioid.power Y n*))
by (*auto simp add*: *t-star-elim t-prod-def*)

lemma *t-star-contr*: *t-prod* (*t-star X*) *Y* = (⋃ *n. t-prod* (*trace-dioid.power X n*) *Y*)
by (*auto simp add*: *t-star-elim t-prod-def*)

interpretation *trace-kleene-algebra*: *kleene-algebra* (∪) *t-prod t-one t-zero* (⊆) (⊂) *t-star*
proof
fix *X Y Z* :: ('a, 'b) *trace set*
show *t-one* ∪ *t-prod X* (*t-star X*) ⊆ *t-star X*
proof −
have *t-one* ∪ *t-prod X* (*t-star X*) = (*trace-dioid.power X 0*) ∪ (⋃ *n. trace-dioid.power X* (*Suc n*))
by (*auto simp add*: *t-star-def t-prod-def*)
also have ... = (⋃ *n. trace-dioid.power X n*)
by (*metis Un-0-Suc*)
also have ... = *t-star X*
by (*metis t-star-def*)
finally show *?thesis*
by (*metis subset-refl*)
qed
show *Z* ∪ *t-prod X Y* ⊆ *Y* ⟹ *t-prod* (*t-star X*) *Z* ⊆ *Y*
by (*simp only*: *ball-UNIV t-star-contr SUP-le-iff*) (*metis trace-dioid.power-inductl*)
show *Z* ∪ *t-prod Y X* ⊆ *Y* ⟹ *t-prod Z* (*t-star X*) ⊆ *Y*
by (*simp only*: *ball-UNIV t-star-contl SUP-le-iff*) (*metis trace-dioid.power-inductr*)
qed

8.7 Path Kleene Algebras

We start with paths that include the empty path.

definition *p-star* :: '*a path set* ⇒ '*a path set* **where**
p-star X ≡ ⋃ *n. path-dioid.power X n*

lemma *p-star-elim*: *x* ∈ *p-star X* ⟷ (∃ *n. x* ∈ *path-dioid.power X n*)
by (*simp add*: *p-star-def*)

lemma *p-star-contl*: *p-prod X* (*p-star Y*) = (⋃*n*. *p-prod X* (*path-dioid.power Y n*))
apply (*auto simp add*: *p-prod-def p-star-elim*)
  apply (*metis p-fusion.simps*(*1*))
 apply *metis*
 apply (*metis p-fusion.simps*(*1*) *p-star-elim*)
apply (*metis p-star-elim*)
done

lemma *p-star-contr*: *p-prod* (*p-star X*) *Y* = (⋃*n*. *p-prod* (*path-dioid.power X n*) *Y*)
apply (*auto simp add*: *p-prod-def p-star-elim*)
  apply (*metis p-fusion.simps*(*1*))
 apply *metis*
 apply (*metis p-fusion.simps*(*1*) *p-star-elim*)
apply (*metis p-star-elim*)
done

interpretation *path-kleene-algebra*: *kleene-algebra* (∪) *p-prod p-one* {} (⊆) (⊂) *p-star*
proof
 fix *X Y Z* :: *'a path set*
 show *p-one* ∪ *p-prod X* (*p-star X*) ⊆ *p-star X*
  proof −
   have *p-one* ∪ *p-prod X* (*p-star X*) = (*path-dioid.power X 0*) ∪ (⋃*n*. *path-dioid.power X* (*Suc n*))
    by (*auto simp add*: *p-star-def p-prod-def*)
   also have ... = (⋃*n*. *path-dioid.power X n*)
    by (*metis Un-0-Suc*)
   also have ... = *p-star X*
    by (*metis p-star-def*)
   finally show *?thesis*
    by (*metis subset-refl*)
  qed
 show *Z* ∪ *p-prod X Y* ⊆ *Y* ⟹ *p-prod* (*p-star X*) *Z* ⊆ *Y*
  by (*simp only*: *ball-UNIV p-star-contr SUP-le-iff*) (*metis path-dioid.power-inductl*)
 show *Z* ∪ *p-prod Y X* ⊆ *Y* ⟹ *p-prod Z* (*p-star X*) ⊆ *Y*
  by (*simp only*: *ball-UNIV p-star-contl SUP-le-iff*) (*metis path-dioid.power-inductr*)
qed

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

definition *pp-star* :: *'a ppath set* ⇒ *'a ppath set* **where**
 pp-star X ≡ ⋃*n*. *ppath-dioid.power X n*

lemma *pp-star-elim*: *x* ∈ *pp-star X* ⟷ (∃*n*. *x* ∈ *ppath-dioid.power X n*)
by (*simp add*: *pp-star-def*)

lemma *pp-star-contl*: *pp-prod X* (*pp-star Y*) = (⋃*n*. *pp-prod X* (*ppath-dioid.power*

Y n))
by (*auto simp add*: *pp-prod-def pp-star-elim*)

lemma *pp-star-contr*: *pp-prod* (*pp-star X*) *Y* = (⋃ *n*. *pp-prod* (*ppath-dioid.power X n*) *Y*)
by (*auto simp add*: *pp-prod-def pp-star-elim*)

interpretation *ppath-kleene-algebra*: *kleene-algebra* (∪) *pp-prod pp-one* {} (⊆) (⊂) *pp-star*
proof
  fix *X Y Z* :: *'a ppath set*
  show *pp-one* ∪ *pp-prod X* (*pp-star X*) ⊆ *pp-star X*
    proof −
      have *pp-one* ∪ *pp-prod X* (*pp-star X*) = (*ppath-dioid.power X 0*) ∪ (⋃ *n*. *ppath-dioid.power X* (*Suc n*))
        by (*auto simp add*: *pp-star-def pp-prod-def*)
      also have ... = (⋃ *n*. *ppath-dioid.power X n*)
        by (*metis Un-0-Suc*)
      also have ... = *pp-star X*
        by (*metis pp-star-def*)
      finally show *?thesis*
        by (*metis subset-refl*)
    qed
  show *Z* ∪ *pp-prod X Y* ⊆ *Y* ⟹ *pp-prod* (*pp-star X*) *Z* ⊆ *Y*
   by (*simp only*: *ball-UNIV pp-star-contr SUP-le-iff*) (*metis ppath-dioid.power-inductl*)
  show *Z* ∪ *pp-prod Y X* ⊆ *Y* ⟹ *pp-prod Z* (*pp-star X*) ⊆ *Y*
   by (*simp only*: *ball-UNIV pp-star-contl SUP-le-iff*) (*metis ppath-dioid.power-inductr*)
qed

8.8 The Distributive Lattice Kleene Algebra

In the case of bounded distributive lattices, the star maps all elements to to the maximal element.

definition (**in** *bounded-distributive-lattice*) *bdl-star* :: *'a* ⇒ *'a* **where**
  bdl-star x = *top*

sublocale *bounded-distributive-lattice* ⊆ *kleene-algebra sup inf top bot less-eq less bdl-star*
proof
  fix *x y z* :: *'a*
  show *sup top* (*inf x* (*bdl-star x*)) ≤ *bdl-star x*
    by (*simp add*: *bdl-star-def*)
  show *sup z* (*inf x y*) ≤ *y* ⟹ *inf* (*bdl-star x*) *z* ≤ *y*
    by (*simp add*: *bdl-star-def*)
  show *sup z* (*inf y x*) ≤ *y* ⟹ *inf z* (*bdl-star x*) ≤ *y*
    by (*simp add*: *bdl-star-def*)
qed

8.9 The Min-Plus Kleene Algebra

One cannot define a Kleene star for max-plus and min-plus algebras that range over the real numbers. Here we define the star for a min-plus algebra restricted to natural numbers and $+\infty$. The resulting Kleene algebra is commutative. Similar variants can be obtained for max-plus algebras and other algebras ranging over the positive or negative integers.

instantiation *pnat* :: *commutative-kleene-algebra*
begin

definition *star-pnat* **where**
$x^\star \equiv (1{::}pnat)$

instance
proof
fix $x\ y\ z$ :: *pnat*
show $1 + x \cdot x^\star \leq x^\star$
by (*metis star-pnat-def zero-pnat-top*)
show $z + x \cdot y \leq y \Longrightarrow x^\star \cdot z \leq y$
by (*simp add*: *star-pnat-def*)
show $z + y \cdot x \leq y \Longrightarrow z \cdot x^\star \leq y$
by (*simp add*: *star-pnat-def*)
show $x \cdot y = y \cdot x$
unfolding *times-pnat-def* **by** (*cases x*, *cases y*, *simp-all*)
qed

end

end

9 Omega Algebras

theory *Omega-Algebra*
imports *Kleene-Algebra*
begin

Omega algebras [7] extend Kleene algebras by an ω-operation that axiomatizes infinite iteration (just like the Kleene star axiomatizes finite iteration).

9.1 Left Omega Algebras

In this section we consider *left omega algebras*, i.e., omega algebras based on left Kleene algebras. Surprisingly, we are still looking for statements mentioning ω that are true in omega algebras, but do not already hold in left omega algebras.

class *left-omega-algebra* = *left-kleene-algebra-zero* + *omega-op* +
assumes *omega-unfold*: $x^\omega \leq x \cdot x^\omega$

and *omega-coinduct*: $y \leq z + x \cdot y \Longrightarrow y \leq x^\omega + x^\star \cdot z$
begin

First we prove some variants of the coinduction axiom.

lemma *omega-coinduct-var1*: $y \leq 1 + x \cdot y \Longrightarrow y \leq x^\omega + x^\star$
using *local.omega-coinduct* **by** *fastforce*

lemma *omega-coinduct-var2*: $y \leq x \cdot y \Longrightarrow y \leq x^\omega$
by (*metis add.commute add-zero-l annir omega-coinduct*)

lemma *omega-coinduct-eq*: $y = z + x \cdot y \Longrightarrow y \leq x^\omega + x^\star \cdot z$
by (*simp add*: *local.omega-coinduct*)

lemma *omega-coinduct-eq-var1*: $y = 1 + x \cdot y \Longrightarrow y \leq x^\omega + x^\star$
by (*simp add*: *omega-coinduct-var1*)

lemma *omega-coinduct-eq-var2*: $y = x \cdot y \Longrightarrow y \leq x^\omega$
by (*simp add*: *omega-coinduct-var2*)

lemma $y = x \cdot y + z \Longrightarrow y = x^\star \cdot z + x^\omega$

oops

lemma $y = 1 + x \cdot y \Longrightarrow y = x^\omega + x^\star$

oops

lemma $y = x \cdot y \Longrightarrow y = x^\omega$

oops

Next we strengthen the unfold law to an equation.

lemma *omega-unfold-eq* [*simp*]: $x \cdot x^\omega = x^\omega$
proof (*rule antisym*)
have $x \cdot x^\omega \leq x \cdot x \cdot x^\omega$
by (*simp add*: *local.mult-isol local.omega-unfold mult-assoc*)
thus $x \cdot x^\omega \leq x^\omega$
by (*simp add*: *mult-assoc omega-coinduct-var2*)
show $x^\omega \leq x \cdot x^\omega$
by (*fact omega-unfold*)
qed

lemma *omega-unfold-var*: $z + x \cdot x^\omega \leq x^\omega + x^\star \cdot z$
by (*simp add*: *local.omega-coinduct*)

lemma $z + x \cdot x^\omega = x^\omega + x^\star \cdot z$

oops

We now prove subdistributivity and isotonicity of omega.

lemma *omega-subdist*: $x^\omega \leq (x + y)^\omega$
proof $-$
  have $x^\omega \leq (x + y) \cdot x^\omega$
    by *simp*
  thus *?thesis*
    by (*rule omega-coinduct-var2*)
qed

lemma *omega-iso*: $x \leq y \Longrightarrow x^\omega \leq y^\omega$
  by (*metis less-eq-def omega-subdist*)

lemma *omega-subdist-var*: $x^\omega + y^\omega \leq (x + y)^\omega$
  by (*simp add*: *omega-iso*)

lemma *zero-omega* [*simp*]: $0^\omega = 0$
  by (*metis annil omega-unfold-eq*)

The next lemma is another variant of omega unfold

lemma *star-omega-1* [*simp*]: $x^\star \cdot x^\omega = x^\omega$
proof (*rule antisym*)
  have $x \cdot x^\omega \leq x^\omega$
    by *simp*
  thus $x^\star \cdot x^\omega \leq x^\omega$
    by *simp*
  show $x^\omega \leq x^\star \cdot x^\omega$
    using *local.star-inductl-var-eq2* **by** *auto*
qed

The next lemma says that $(1::'a)^\omega$ is the maximal element of omega algebra. We therefore baptise it $\top$.

lemma *max-element*: $x \leq 1^\omega$
  by (*simp add*: *omega-coinduct-eq-var2*)

definition *top* ($\top$)
  where $\top = 1^\omega$

lemma *star-omega-3* [*simp*]: $(x^\star)^\omega = \top$
proof $-$
  have $1 \leq x^\star$
    by (*fact star-ref*)
  hence $\top \leq (x^\star)^\omega$
    by (*simp add*: *omega-iso top-def*)
  thus *?thesis*
    by (*simp add*: *local.order.antisym max-element top-def*)
qed

The following lemma is strange since it is counterintuitive that one should be able to append something after an infinite iteration.

lemma *omega-1*: $x^\omega \cdot y \leq x^\omega$

proof −
  have $x^\omega \cdot y \leq x \cdot x^\omega \cdot y$
    by *simp*
  thus *?thesis*
    by (*metis mult.assoc omega-coinduct-var2*)
qed

lemma $x^\omega \cdot y = x^\omega$

oops

lemma *omega-sup-id*: $1 \leq y \Longrightarrow x^\omega \cdot y = x^\omega$
  using *local.eq-iff local.mult-isol omega-1* **by** *fastforce*

lemma *omega-top* [*simp*]: $x^\omega \cdot \top = x^\omega$
  by (*simp add*: *max-element omega-sup-id top-def*)

lemma *supid-omega*: $1 \leq x \Longrightarrow x^\omega = \top$
  by (*simp add*: *local.order.antisym max-element omega-iso top-def*)

lemma $x^\omega = \top \Longrightarrow 1 \leq x$

oops

Next we prove a simulation law for the omega operation

lemma *omega-simulation*: $z \cdot x \leq y \cdot z \Longrightarrow z \cdot x^\omega \leq y^\omega$
proof −
  assume *hyp*: $z \cdot x \leq y \cdot z$
  have $z \cdot x^\omega = z \cdot x \cdot x^\omega$
    by (*simp add*: *mult-assoc*)
  also have ... $\leq y \cdot z \cdot x^\omega$
    by (*simp add*: *hyp local.mult-isor*)
  finally show $z \cdot x^\omega \leq y^\omega$
    by (*simp add*: *mult-assoc omega-coinduct-var2*)
qed

lemma $z \cdot x \leq y \cdot z \Longrightarrow z \cdot x^\omega \leq y^\omega \cdot z$

oops

lemma $y \cdot z \leq z \cdot x \Longrightarrow y^\omega \leq z \cdot x^\omega$

oops

lemma $y \cdot z \leq z \cdot x \Longrightarrow y^\omega \cdot z \leq x^\omega$

oops

Next we prove transitivity of omega elements.

lemma *omega-omega*: $(x^\omega)^\omega \leq x^\omega$
  by (*metis omega-1 omega-unfold-eq*)

The next lemmas are axioms of Wagner's complete axiomatisation for omega-regular languages [32], but in a slightly different setting.

lemma *wagner-1* [*simp*]: $(x \cdot x^\star)^\omega = x^\omega$
proof (*rule antisym*)
  have $(x \cdot x^\star)^\omega = x \cdot x^\star \cdot x \cdot x^\star \cdot (x \cdot x^\star)^\omega$
    by (*metis mult.assoc omega-unfold-eq*)
  also have ... $= x \cdot x \cdot x^\star \cdot x^\star \cdot (x \cdot x^\star)^\omega$
    by (*simp add*: *local.star-slide-var mult-assoc*)
  also have ... $= x \cdot x \cdot x^\star \cdot (x \cdot x^\star)^\omega$
    by (*simp add*: *mult-assoc*)
  also have ... $= x \cdot (x \cdot x^\star)^\omega$
    by (*simp add*: *mult-assoc*)
  thus $(x \cdot x^\star)^\omega \leq x^\omega$
    using *calculation omega-coinduct-eq-var2* **by** *auto*
   show $x^\omega \leq (x \cdot x^\star)^\omega$
    by (*simp add*: *mult-assoc omega-coinduct-eq-var2*)
qed

lemma *wagner-2-var*: $x \cdot (y \cdot x)^\omega \leq (x \cdot y)^\omega$
proof −
  have $x \cdot y \cdot x \leq x \cdot y \cdot x$
    by *auto*
  thus $x \cdot (y \cdot x)^\omega \leq (x \cdot y)^\omega$
    by (*simp add*: *mult-assoc omega-simulation*)
qed

lemma *wagner-2* [*simp*]: $x \cdot (y \cdot x)^\omega = (x \cdot y)^\omega$
proof (*rule antisym*)
  show $x \cdot (y \cdot x)^\omega \leq (x \cdot y)^\omega$
    by (*rule wagner-2-var*)
  have $(x \cdot y)^\omega = x \cdot y \cdot (x \cdot y)^\omega$
    by *simp*
  thus $(x \cdot y)^\omega \leq x \cdot (y \cdot x)^\omega$
    by (*metis mult.assoc mult-isol wagner-2-var*)
qed

This identity is called (A8) in Wagner's paper.

lemma *wagner-3*:
assumes $x \cdot (x + y)^\omega + z = (x + y)^\omega$
shows $(x + y)^\omega = x^\omega + x^\star \cdot z$
proof (*rule antisym*)
  show $(x + y)^\omega \leq x^\omega + x^\star \cdot z$
    using *assms local.join.sup-commute omega-coinduct-eq* **by** *auto*
  have $x^\star \cdot z \leq (x + y)^\omega$
    using *assms local.join.sup-commute local.star-inductl-eq* **by** *auto*
  thus $x^\omega + x^\star \cdot z \leq (x + y)^\omega$

  by (*simp add*: *omega-subdist*)
qed

This identity is called (R4) in Wagner's paper.

lemma *wagner-1-var* [*simp*]: $(x^\star \cdot x)^\omega = x^\omega$
 by (*simp add*: *local.star-slide-var*)

lemma *star-omega-4* [*simp*]: $(x^\omega)^\star = 1 + x^\omega$
proof (*rule antisym*)
 have $(x^\omega)^\star = 1 + x^\omega \cdot (x^\omega)^\star$
  by *simp*
 also have ... $\leq 1 + x^\omega \cdot \top$
  by (*simp add*: *omega-sup-id*)
 finally show $(x^\omega)^\star \leq 1 + x^\omega$
  by *simp*
 show $1 + x^\omega \leq (x^\omega)^\star$
  by *simp*
qed

lemma *star-omega-5* [*simp*]: $x^\omega \cdot (x^\omega)^\star = x^\omega$
proof (*rule antisym*)
 show $x^\omega \cdot (x^\omega)^\star \leq x^\omega$
  by (*rule omega-1*)
 show $x^\omega \leq x^\omega \cdot (x^\omega)^\star$
  by (*simp add*: *omega-sup-id*)
qed

The next law shows how omegas below a sum can be unfolded.

lemma *omega-sum-unfold*: $x^\omega + x^\star \cdot y \cdot (x + y)^\omega = (x + y)^\omega$
proof −
 have $(x + y)^\omega = x \cdot (x + y)^\omega + y \cdot (x+y)^\omega$
  by (*metis distrib-right omega-unfold-eq*)
 thus *?thesis*
  by (*metis mult.assoc wagner-3*)
qed

The next two lemmas apply induction and coinduction to this law.

lemma *omega-sum-unfold-coind*: $(x + y)^\omega \leq (x^\star \cdot y)^\omega + (x^\star \cdot y)^\star \cdot x^\omega$
 by (*simp add*: *omega-coinduct-eq omega-sum-unfold*)

lemma *omega-sum-unfold-ind*: $(x^\star \cdot y)^\star \cdot x^\omega \leq (x + y)^\omega$
 by (*simp add*: *local.star-inductl-eq omega-sum-unfold*)

lemma *wagner-1-gen*: $(x \cdot y^\star)^\omega \leq (x + y)^\omega$
proof −
 have $(x \cdot y^\star)^\omega \leq ((x + y) \cdot (x + y)^\star)^\omega$
  using *local.join.le-sup-iff local.join.sup.cobounded1 local.mult-isol-var local.star-subdist-var omega-iso* **by** *presburger*
 thus *?thesis*

by (*metis wagner-1*)
qed

lemma *wagner-1-var-gen*: $(x^\star \cdot y)^\omega \leq (x + y)^\omega$
proof −
have $(x^\star \cdot y)^\omega = x^\star \cdot (y \cdot x^\star)^\omega$
by *simp*
also have ... $\leq x^\star \cdot (x + y)^\omega$
by (*metis add.commute mult-isol wagner-1-gen*)
also have ... $\leq (x + y)^\star \cdot (x + y)^\omega$
using *local.mult-isor local.star-subdist* **by** *auto*
thus *?thesis*
by (*metis calculation order-trans star-omega-1*)
qed

The next lemma is a variant of the denest law for the star at the level of omega.

lemma *omega-denest* [*simp*]: $(x + y)^\omega = (x^\star \cdot y)^\omega + (x^\star \cdot y)^\star \cdot x^\omega$
proof (*rule antisym*)
show $(x + y)^\omega \leq (x^\star \cdot y)^\omega + (x^\star \cdot y)^\star \cdot x^\omega$
by (*rule omega-sum-unfold-coind*)
have $(x^\star \cdot y)^\omega \leq (x + y)^\omega$
by (*rule wagner-1-var-gen*)
hence $(x^\star \cdot y)^\star \cdot x^\omega \leq (x + y)^\omega$
by (*simp add*: *omega-sum-unfold-ind*)
thus $(x^\star \cdot y)^\omega + (x^\star \cdot y)^\star \cdot x^\omega \leq (x + y)^\omega$
by (*simp add*: *wagner-1-var-gen*)
qed

The next lemma yields a separation theorem for infinite iteration in the presence of a quasicommutation property. A nondeterministic loop over x and y can be refined into separate infinite loops over x and y.

lemma *omega-sum-refine*:
assumes $y \cdot x \leq x \cdot (x + y)^\star$
shows $(x + y)^\omega = x^\omega + x^\star \cdot y^\omega$
proof (*rule antisym*)
have *a*: $y^\star \cdot x \leq x \cdot (x + y)^\star$
using *assms local.quasicomm-var* **by** *blast*
have $(x + y)^\omega = y^\omega + y^\star \cdot x \cdot (x + y)^\omega$
by (*metis add.commute omega-sum-unfold*)
also have ... $\leq y^\omega + x \cdot (x + y)^\star \cdot (x + y)^\omega$
using *a local.join.sup-mono local.mult-isol-var* **by** *blast*
also have ... $\leq x \cdot (x + y)^\omega + y^\omega$
using *local.eq-refl local.join.sup-commute mult-assoc star-omega-1* **by** *presburger*
finally show $(x + y)^\omega \leq x^\omega + x^\star \cdot y^\omega$
by (*metis add-commute local.omega-coinduct*)
have $x^\omega + x^\star \cdot y^\omega \leq (x + y)^\omega + (x + y)^\star \cdot (x + y)^\omega$
using *local.join.sup.cobounded2 local.join.sup.mono local.mult-isol-var local.star-subdist omega-iso omega-subdist* **by** *presburger*

thus $x^\omega + x^\star \cdot y^\omega \leq (x + y)^\omega$
by (*metis local.join.sup-idem star-omega-1*)
qed

The following theorem by Bachmair and Dershowitz [4] is a corollary.

lemma *bachmair-dershowitz*:
assumes $y \cdot x \leq x \cdot (x + y)^\star$
shows $(x + y)^\omega = 0 \longleftrightarrow x^\omega + y^\omega = 0$
by (*metis add-commute assms local.annir local.join.le-bot local.no-trivial-inverse omega-subdist omega-sum-refine*)

The next lemmas consider an abstract variant of the empty word property from language theory and match it with the absence of infinite iteration [28].

definition (**in** *dioid-one-zero*) *ewp*
where $ewp\ x \equiv \neg(\forall y.\ y \leq x \cdot y \longrightarrow y = 0)$

lemma *ewp-super-id1*: $0 \neq 1 \Longrightarrow 1 \leq x \Longrightarrow ewp\ x$
by (*metis ewp-def mult-oner*)

lemma $0 \neq 1 \Longrightarrow 1 \leq x \longleftrightarrow ewp\ x$

oops

The next facts relate the absence of the empty word property with the absence of infinite iteration.

lemma *ewp-neg-and-omega*: $\neg\ ewp\ x \longleftrightarrow x^\omega = 0$
proof
assume $\neg\ ewp\ x$
hence $\forall\ y.\ y \leq x \cdot y \longrightarrow y = 0$
by (*meson ewp-def*)
thus $x^\omega = 0$
by *simp*
next
assume $x^\omega = 0$
hence $\forall\ y.\ y \leq x \cdot y \longrightarrow y = 0$
using *local.join.le-bot local.omega-coinduct-var2* **by** *blast*
thus $\neg\ ewp\ x$
by (*meson ewp-def*)
qed

lemma *ewp-alt1*: $(\forall z.\ x^\omega \leq x^\star \cdot z) \longleftrightarrow (\forall y\ z.\ y \leq x \cdot y + z \longrightarrow y \leq x^\star \cdot z)$
by (*metis add-comm less-eq-def omega-coinduct omega-unfold-eq order-prop*)

lemma *ewp-alt*: $x^\omega = 0 \longleftrightarrow (\forall y\ z.\ y \leq x \cdot y + z \longrightarrow y \leq x^\star \cdot z)$
by (*metis annir antisym ewp-alt1 join.bot-least*)

So we have obtained a condition for Arden's lemma in omega algebra.

lemma *omega-super-id1*: $0 \neq 1 \Longrightarrow 1 \leq x \Longrightarrow x^\omega \neq 0$

using *ewp-neg-and-omega ewp-super-id1* **by** *blast*

lemma *omega-super-id2*: $0 \neq 1 \Longrightarrow x^\omega = 0 \Longrightarrow \neg(1 \leq x)$
using *omega-super-id1* **by** *blast*

The next lemmas are abstract versions of Arden's lemma from language theory.

lemma *ardens-lemma-var*:
assumes $x^\omega = 0$
and $z + x \cdot y = y$
shows $x^\star \cdot z = y$
proof −
have $y \leq x^\omega + x^\star \cdot z$
by (*simp add*: *assms*(*2*) *local.omega-coinduct-eq*)
hence $y \leq x^\star \cdot z$
by (*simp add*: *assms*(*1*))
thus $x^\star \cdot z = y$
by (*simp add*: *assms*(*2*) *local.eq-iff local.star-inductl-eq*)
qed

lemma *ardens-lemma*: $\neg$ *ewp* $x \Longrightarrow z + x \cdot y = y \Longrightarrow x^\star \cdot z = y$
by (*simp add*: *ardens-lemma-var ewp-neg-and-omega*)

lemma *ardens-lemma-equiv*:
assumes $\neg$ *ewp* x
shows $z + x \cdot y = y \longleftrightarrow x^\star \cdot z = y$
by (*metis ardens-lemma-var assms ewp-neg-and-omega local.conway.dagger-unfoldl-distr mult-assoc*)

lemma *ardens-lemma-var-equiv*: $x^\omega = 0 \Longrightarrow (z + x \cdot y = y \longleftrightarrow x^\star \cdot z = y)$
by (*simp add*: *ardens-lemma-equiv ewp-neg-and-omega*)

lemma *arden-conv1*: $(\forall\, y\ z.\ z + x \cdot y = y \longrightarrow x^\star \cdot z = y) \Longrightarrow \neg$ *ewp* x
by (*metis add-zero-l annir ewp-neg-and-omega omega-unfold-eq*)

lemma *arden-conv2*: $(\forall\, y\ z.\ z + x \cdot y = y \longrightarrow x^\star \cdot z = y) \Longrightarrow x^\omega = 0$
using *arden-conv1 ewp-neg-and-omega* **by** *blast*

lemma *arden-var3*: $(\forall\, y\ z.\ z + x \cdot y = y \longrightarrow x^\star \cdot z = y) \longleftrightarrow x^\omega = 0$
using *arden-conv2 ardens-lemma-var* **by** *blast*

end

9.2 Omega Algebras

class *omega-algebra* = *kleene-algebra* + *left-omega-algebra*

end

10 Models of Omega Algebras

theory *Omega-Algebra-Models*
imports *Omega-Algebra Kleene-Algebra-Models*
begin

The trace, path and language model are not really interesting in this setting.

10.1 Relation Omega Algebras

In the relational model, the omega of a relation relates all those elements in the domain of the relation, from which an infinite chain starts, with all other elements; all other elements are not related to anything [19]. Thus, the omega of a relation is most naturally defined coinductively.

coinductive-set *omega* :: $('a \times 'a)$ *set* $\Rightarrow$ $('a \times 'a)$ *set* **for** R **where**
  $[\![(x, y) \in R; (y, z) \in omega\ R]\!] \Longrightarrow (x, z) \in omega\ R$

Isabelle automatically derives a case rule and a coinduction theorem for *Omega-Algebra-Models.omega*. We prove slightly more elegant variants.

lemma *omega-cases*: $(x, z) \in omega\ R \Longrightarrow$
  $(\bigwedge y.\ (x, y) \in R \Longrightarrow (y, z) \in omega\ R \Longrightarrow P) \Longrightarrow P$
by (*metis omega.cases*)

lemma *omega-coinduct*: $X\ x\ z \Longrightarrow$
  $(\bigwedge x\ z.\ X\ x\ z \Longrightarrow \exists\, y.\ (x, y) \in R \wedge (X\ y\ z \vee (y, z) \in omega\ R)) \Longrightarrow$
  $(x, z) \in omega\ R$
by (*metis* (*full-types*) *omega.coinduct*)

lemma *omega-weak-coinduct*: $X\ x\ z \Longrightarrow$
  $(\bigwedge x\ z.\ X\ x\ z \Longrightarrow \exists\, y.\ (x, y) \in R \wedge X\ y\ z) \Longrightarrow$
  $(x, z) \in omega\ R$
by (*metis omega.coinduct*)

interpretation *rel-omega-algebra*: *omega-algebra* $(\cup)$ (O) *Id* $\{\}$ $(\subseteq)$ $(\subset)$ *rtrancl* *omega*
proof
  fix x :: *'a rel*
  show $omega\ x \subseteq x\ O\ omega\ x$
    by (*auto elim*: *omega-cases*)
next
  fix $x\ y\ z$ :: *'a rel*
  assume $*$: $y \subseteq z \cup x\ O\ y$
  {
    fix $a\ b$
    assume *1*: $(a,b) \in y$ **and** *2*: $(a,b) \notin x^*\ O\ z$
    have $(a,b) \in omega\ x$
    proof (*rule omega-weak-coinduct*[**where** $X = \lambda a\ b.\ (a,b) \in x\ O\ y \wedge (a,b) \notin x^*\ O\ z$])

show $(a,b) \in x\ O\ y \wedge (a,b) \notin x^*\ O\ z$
  using $*$ *1 2* **by** *auto*
next
  fix $a\ c$
  assume *1*: $(a,c) \in x\ O\ y \wedge (a,c) \notin x^*\ O\ z$
  then obtain b **where** *2*: $(a,b) \in x$ **and** $(b,c) \in y$
    by *auto*
  then have $(b,c) \in x\ O\ y$
    using $*$ *1* **by** *blast*
  moreover have $(b,c) \notin x^*\ O\ z$
  using *1 2* **by** (*meson relcomp.cases relcomp.intros converse-rtrancl-into-rtrancl*)
  ultimately show $\exists\, b.\ (a,b) \in x \wedge (b,c) \in x\ O\ y \wedge (b,c) \notin x^*\ O\ z$
    using *2* **by** *blast*
qed
}
then show $y \subseteq$ *omega* $x \cup x^*\ O\ z$
  by *auto*
qed

end

11 Demonic Refinement Algebras

theory *DRA*
  imports *Kleene-Algebra*
begin

A demonic refinement algebra *DRA) [31] is a Kleene algebra without right annihilation plus an operation for possibly infinite iteration.

class *dra* = *kleene-algebra-zerol* +
  fixes *strong-iteration* :: $'a \Rightarrow 'a$ (-^{∞} [*101*] *100*)
  assumes *iteration-unfoldl* [*simp*] : $1 + x \cdot x^{\infty} = x^{\infty}$
  and *coinduction*: $y \leq z + x \cdot y \longrightarrow y \leq x^{\infty} \cdot z$
  and *isolation* [*simp*]: $x^{\star} + x^{\infty} \cdot 0 = x^{\infty}$
begin

$\top$ is an abort statement, defined as an infinite skip. It is the maximal element of any DRA.

abbreviation *top-elem* :: $'a$ ($\top$) **where** $\top \equiv 1^{\infty}$

Simple/basic lemmas about the iteration operator

lemma *iteration-refl*: $1 \leq x^{\infty}$
  using *local.iteration-unfoldl local.order-prop* **by** *blast*

lemma *iteration-1l*: $x \cdot x^{\infty} \leq x^{\infty}$
  by (*metis local.iteration-unfoldl local.join.sup.cobounded2*)

lemma *top-ref*: $x \leq \top$

proof −
  have $x \leq 1 + 1 \cdot x$
    by *simp*
  thus *?thesis*
    using *local.coinduction* **by** *fastforce*
qed

lemma *it-ext*: $x \leq x^\infty$
proof −
  have $x \leq x \cdot x^\infty$
    using *iteration-refl local.mult-isol* **by** *fastforce*
  thus *?thesis*
    by (*metis* (*full-types*) *local.isolation local.join.sup.coboundedI1 local.star-ext*)
qed

lemma *it-idem* [*simp*]: $(x^\infty)^\infty = x^\infty$

oops

lemma *top-mult-annil* [*simp*]: $\top \cdot x = \top$
  by (*simp add*: *local.coinduction local.order.antisym top-ref*)

lemma *top-add-annil* [*simp*]: $\top + x = \top$
  by (*simp add*: *local.join.sup.absorb1 top-ref*)

lemma *top-elim*: $x \cdot y \leq x \cdot \top$
  by (*simp add*: *local.mult-isol top-ref*)

lemma *iteration-unfoldl-distl* [*simp*]: $y + y \cdot x \cdot x^\infty = y \cdot x^\infty$
  by (*metis distrib-left mult.assoc mult-oner iteration-unfoldl*)

lemma *iteration-unfoldl-distr* [*simp*]: $y + x \cdot x^\infty \cdot y = x^\infty \cdot y$
  by (*metis distrib-right′ mult-1-left iteration-unfoldl*)

lemma *iteration-unfoldl′* [*simp*]: $z \cdot y + z \cdot x \cdot x^\infty \cdot y = z \cdot x^\infty \cdot y$
  by (*metis iteration-unfoldl-distl local.distrib-right*)

lemma *iteration-idem* [*simp*]: $x^\infty \cdot x^\infty = x^\infty$
proof (*rule antisym*)
  have $x^\infty \cdot x^\infty \leq 1 + x \cdot x^\infty \cdot x^\infty$
    by (*metis add-assoc iteration-unfoldl-distr local.eq-refl local.iteration-unfoldl local.subdistl-eq mult-assoc*)
  thus $x^\infty \cdot x^\infty \leq x^\infty$
    using *local.coinduction mult-assoc* **by** *fastforce*
  show $x^\infty \leq x^\infty \cdot x^\infty$
    using *local.coinduction* **by** *auto*
qed

lemma *iteration-induct*: $x \cdot x^\infty \leq x^\infty \cdot x$

proof –
  have $x + x \cdot (x \cdot x^\infty) = x \cdot x^\infty$
    by (*metis* (*no-types*) *local.distrib-left local.iteration-unfoldl local.mult-oner*)
  thus *?thesis*
    by (*simp add*: *local.coinduction*)
qed

lemma *iteration-ref-star*: $x^\star \leq x^\infty$
  by (*simp add*: *local.star-inductl-one*)

lemma *iteration-subdist*: $x^\infty \leq (x + y)^\infty$
  by (*metis add-assoc′ distrib-right′ mult-oner coinduction join.sup-ge1 iteration-unfoldl*)

lemma *iteration-iso*: $x \leq y \Longrightarrow x^\infty \leq y^\infty$
  using *iteration-subdist local.order-prop* **by** *auto*

lemma *iteration-unfoldr* [*simp*]: $1 + x^\infty \cdot x = x^\infty$
  by (*metis add-0-left annil eq-refl isolation mult.assoc iteration-idem iteration-unfoldl iteration-unfoldl-distr star-denest star-one star-prod-unfold star-slide tc*)

lemma *iteration-unfoldr-distl* [*simp*]: $y + y \cdot x^\infty \cdot x = y \cdot x^\infty$
  by (*metis distrib-left mult.assoc mult-oner iteration-unfoldr*)

lemma *iteration-unfoldr-distr* [*simp*]: $y + x^\infty \cdot x \cdot y = x^\infty \cdot y$
  by (*metis iteration-unfoldl-distr iteration-unfoldr-distl*)

lemma *iteration-unfold-eq*: $x^\infty \cdot x = x \cdot x^\infty$
  by (*metis iteration-unfoldl-distr iteration-unfoldr-distl*)

lemma *iteration-unfoldr′* [*simp*]: $z \cdot y + z \cdot x^\infty \cdot x \cdot y = z \cdot x^\infty \cdot y$
  by (*metis distrib-left mult.assoc iteration-unfoldr-distr*)

lemma *iteration-double* [*simp*]: $(x^\infty)^\infty = \top$
  by (*simp add*: *iteration-iso iteration-refl local.eq-iff top-ref*)

lemma *star-iteration* [*simp*]: $(x^\star)^\infty = \top$
  by (*simp add*: *iteration-iso local.eq-iff top-ref*)

lemma *iteration-star* [*simp*]: $(x^\infty)^\star = x^\infty$
  by (*metis* (*no-types*) *iteration-idem iteration-refl local.star-inductr-var-eq2 local.sup-id-star1*)

lemma *iteration-star2* [*simp*]: $x^\star \cdot x^\infty = x^\infty$
proof –
  have *f1*: $(x^\infty)^\star \cdot x^\star = x^\infty$
    by (*metis* (*no-types*) *it-ext iteration-induct iteration-star local.bubble-sort local.join.sup.absorb1*)
  have $x^\infty = x^\infty \cdot x^\infty$
    by *simp*

hence $x^\star \cdot x^\infty = x^\star \cdot (x^\infty)^\star \cdot (x^\star \cdot (x^\infty)^\star)^\star$
  using *f1* **by** (*metis* (*no-types*) *iteration-star local.star-denest-var-4 mult-assoc*)
thus *?thesis*
  using *f1* **by** (*metis* (*no-types*) *iteration-star local.star-denest-var-4 local.star-denest-var-8*)
qed

lemma *iteration-zero* [*simp*]: $0^\infty = 1$
  by (*metis add-zeror annil iteration-unfoldl*)

lemma *iteration-annil* [*simp*]: $(x \cdot 0)^\infty = 1 + x \cdot 0$
  by (*metis annil iteration-unfoldl mult.assoc*)

lemma *iteration-subdenest*: $x^\infty \cdot y^\infty \leq (x + y)^\infty$
  by (*metis add-commute iteration-idem iteration-subdist local.mult-isol-var*)

lemma *sup-id-top*: $1 \leq y \Longrightarrow y \cdot \top = \top$
  using *local.eq-iff local.mult-isol-var top-ref* **by** *fastforce*

lemma *iteration-top* [*simp*]: $x^\infty \cdot \top = \top$
  by (*simp add*: *iteration-refl sup-id-top*)

Next, we prove some simulation laws for data refinement.

lemma *iteration-sim*: $z \cdot y \leq x \cdot z \Longrightarrow z \cdot y^\infty \leq x^\infty \cdot z$
proof −
  assume *assms*: $z \cdot y \leq x \cdot z$
  have $z \cdot y^\infty = z + z \cdot y \cdot y^\infty$
    by *simp*
  also have ... $\leq z + x \cdot z \cdot y^\infty$
    by (*metis assms add.commute add-iso mult-isor*)
  finally show $z \cdot y^\infty \leq x^\infty \cdot z$
    by (*simp add*: *local.coinduction mult-assoc*)
qed

Nitpick gives a counterexample to the dual simulation law.

lemma $y \cdot z \leq z \cdot x \Longrightarrow y^\infty \cdot z \leq z \cdot x^\infty$

oops

Next, we prove some sliding laws.

lemma *iteration-slide-var*: $x \cdot (y \cdot x)^\infty \leq (x \cdot y)^\infty \cdot x$
  by (*simp add*: *iteration-sim mult-assoc*)

lemma *iteration-prod-unfold* [*simp*]: $1 + y \cdot (x \cdot y)^\infty \cdot x = (y \cdot x)^\infty$
proof (*rule antisym*)
  have $1 + y \cdot (x \cdot y)^\infty \cdot x \leq 1 + (y \cdot x)^\infty \cdot y \cdot x$
    using *iteration-slide-var local.join.sup-mono local.mult-isor* **by** *blast*
  thus $1 + y \cdot (x \cdot y)^\infty \cdot x \leq (y \cdot x)^\infty$
    by (*simp add*: *mult-assoc*)
  have $(y \cdot x)^\infty = 1 + y \cdot x \cdot (y \cdot x)^\infty$

by *simp*
thus $(y \cdot x)^\infty \leq 1 + y \cdot (x \cdot y)^\infty \cdot x$
by (*metis iteration-sim local.eq-refl local.join.sup.mono local.mult-isol mult-assoc*)
qed

lemma *iteration-slide*: $x \cdot (y \cdot x)^\infty = (x \cdot y)^\infty \cdot x$
by (*metis iteration-prod-unfold iteration-unfoldl-distr distrib-left mult-1-right mult.assoc*)

lemma *star-iteration-slide* [*simp*]: $y^\star \cdot (x^\star \cdot y)^\infty = (x^\star \cdot y)^\infty$
by (*metis iteration-star2 local.conway.dagger-unfoldl-distr local.join.sup.orderE local.mult-isor local.star-invol local.star-subdist local.star-trans-eq*)

The following laws are called denesting laws.

lemma *iteration-sub-denest*: $(x + y)^\infty \leq x^\infty \cdot (y \cdot x^\infty)^\infty$
proof −
have $(x + y)^\infty = x \cdot (x + y)^\infty + y \cdot (x + y)^\infty + 1$
by (*metis add.commute distrib-right′ iteration-unfoldl*)
hence $(x + y)^\infty \leq x^\infty \cdot (y \cdot (x + y)^\infty + 1)$
by (*metis add-assoc′ join.sup-least join.sup-ge1 join.sup-ge2 coinduction*)
moreover hence $x^\infty \cdot (y \cdot (x + y)^\infty + 1) \leq x^\infty \cdot (y \cdot x^\infty)^\infty$
by (*metis add-iso mult.assoc mult-isol add.commute coinduction mult-oner mult-isol*)
ultimately show *?thesis*
using *local.order-trans* **by** *blast*
qed

lemma *iteration-denest*: $(x + y)^\infty = x^\infty \cdot (y \cdot x^\infty)^\infty$
proof −
have $x^\infty \cdot (y \cdot x^\infty)^\infty \leq x \cdot x^\infty \cdot (y \cdot x^\infty)^\infty + y \cdot x^\infty \cdot (y \cdot x^\infty)^\infty + 1$
by (*metis add.commute iteration-unfoldl-distr add-assoc′ add.commute iteration-unfoldl order-refl*)
thus *?thesis*
by (*metis add.commute iteration-sub-denest order.antisym coinduction distrib-right′ iteration-sub-denest mult.assoc mult-oner order.antisym*)
qed

lemma *iteration-denest2* [*simp*]: $y^\star \cdot x \cdot (x + y)^\infty + y^\infty = (x + y)^\infty$
proof −
have $(x + y)^\infty = y^\infty \cdot x \cdot (y^\infty \cdot x)^\infty \cdot y^\infty + y^\infty$
by (*metis add.commute iteration-denest iteration-slide iteration-unfoldl-distr*)
also have ... $= y^\star \cdot x \cdot (y^\infty \cdot x)^\infty \cdot y^\infty + y^\infty \cdot 0 + y^\infty$
by (*metis isolation mult.assoc distrib-right′ annil mult.assoc*)
also have ... $= y^\star \cdot x \cdot (y^\infty \cdot x)^\infty \cdot y^\infty + y^\infty$
by (*metis add.assoc distrib-left mult-1-right add-0-left mult-1-right*)
finally show *?thesis*
by (*metis add.commute iteration-denest iteration-slide mult.assoc*)
qed

lemma *iteration-denest3*: $(y^\star \cdot x)^\infty \cdot y^\infty = (x + y)^\infty$

proof (*rule antisym*)
  have $(y^\star \cdot x)^\infty \cdot y^\infty \leq (y^\infty \cdot x)^\infty \cdot y^\infty$
    by (*simp add*: *iteration-iso iteration-ref-star local.mult-isor*)
  thus $(y^\star \cdot x)^\infty \cdot y^\infty \leq (x + y)^\infty$
    by (*metis iteration-denest iteration-slide local.join.sup-commute*)
  have $(x + y)^\infty = y^\infty + y^\star \cdot x \cdot (x + y)^\infty$
    by (*metis iteration-denest2 local.join.sup-commute*)
  thus $(x + y)^\infty \leq (y^\star \cdot x)^\infty \cdot y^\infty$
    by (*simp add*: *local.coinduction*)
qed

Now we prove separation laws for reasoning about distributed systems in the context of action systems.

lemma *iteration-sep*: $y \cdot x \leq x \cdot y \Longrightarrow (x + y)^\infty = x^\infty \cdot y^\infty$
proof −
  assume $y \cdot x \leq x \cdot y$
  hence $y^\star \cdot x \leq x \cdot (x + y)^\star$
    by (*metis star-sim1 add.commute mult-isol order-trans star-subdist*)
  hence $y^\star \cdot x \cdot (x + y)^\infty + y^\infty \leq x \cdot (x + y)^\infty + y^\infty$
    by (*metis mult-isor mult.assoc iteration-star2 join.sup.mono eq-refl*)
  thus *?thesis*
    by (*metis iteration-denest2 add.commute coinduction add.commute less-eq-def iteration-subdenest*)
qed

lemma *iteration-sim2*: $y \cdot x \leq x \cdot y \Longrightarrow y^\infty \cdot x^\infty \leq x^\infty \cdot y^\infty$
  by (*metis add.commute iteration-sep iteration-subdenest*)

lemma *iteration-sep2*: $y \cdot x \leq x \cdot y^\star \Longrightarrow (x + y)^\infty = x^\infty \cdot y^\infty$
proof −
  assume $y \cdot x \leq x \cdot y^\star$
  hence $y^\star \cdot (y^\star \cdot x)^\infty \cdot y^\infty \leq x^\infty \cdot y^\star \cdot y^\infty$
    by (*metis mult.assoc mult-isor iteration-sim star-denest-var-2 star-sim1 star-slide-var star-trans-eq tc-eq*)
  moreover have $x^\infty \cdot y^\star \cdot y^\infty \leq x^\infty \cdot y^\infty$
    by (*metis eq-refl mult.assoc iteration-star2*)
  moreover have $(y^\star \cdot x)^\infty \cdot y^\infty \leq y^\star \cdot (y^\star \cdot x)^\infty \cdot y^\infty$
    by (*metis mult-isor mult-onel star-ref*)
  ultimately show *?thesis*
    by (*metis antisym iteration-denest3 iteration-subdenest order-trans*)
qed

lemma *iteration-sep3*: $y \cdot x \leq x \cdot (x + y) \Longrightarrow (x + y)^\infty = x^\infty \cdot y^\infty$
proof −
  assume $y \cdot x \leq x \cdot (x + y)$
  hence $y^\star \cdot x \leq x \cdot (x + y)^\star$
    by (*metis star-sim1*)
  hence $y^\star \cdot x \cdot (x + y)^\infty + y^\infty \leq x \cdot (x + y)^\star \cdot (x + y)^\infty + y^\infty$
    by (*metis add-iso mult-isor*)

hence $(x + y)^{\infty} \leq x^{\infty} \cdot y^{\infty}$
by (*metis mult.assoc iteration-denest2 iteration-star2 add.commute coinduction*)
thus *?thesis*
by (*metis add.commute less-eq-def iteration-subdenest*)
qed

lemma *iteration-sep4*: $y \cdot 0 = 0 \Longrightarrow z \cdot x = 0 \Longrightarrow y \cdot x \leq (x + z) \cdot y^{\star} \Longrightarrow (x + y + z)^{\infty} = x^{\infty} \cdot (y + z)^{\infty}$
proof −
assume *assms*: $y \cdot 0 = 0$ $z \cdot x = 0$ $y \cdot x \leq (x + z) \cdot y^{\star}$
have $y \cdot y^{\star} \cdot z \leq y^{\star} \cdot z \cdot y^{\star}$
by (*metis mult-isor star-1l mult-oner order-trans star-plus-one subdistl*)
have $y^{\star} \cdot z \cdot x \leq x \cdot y^{\star} \cdot z$
by (*metis join.bot-least assms*(*1*) *assms*(*2*) *independence1 mult.assoc*)
have $y \cdot (x + y^{\star} \cdot z) \leq (x + z) \cdot y^{\star} + y \cdot y^{\star} \cdot z$
by (*metis assms*(*3*) *distrib-left mult.assoc add-iso*)
also have ... $\leq (x + y^{\star} \cdot z) \cdot y^{\star} + y \cdot y^{\star} \cdot z$
by (*metis star-ref join.sup.mono eq-refl mult-1-left mult-isor*)
also have ... $\leq (x + y^{\star} \cdot z) \cdot y^{\star} + y^{\star} \cdot z \cdot y^{\star}$ **using** ‹$y \cdot y^{\star} \cdot z \leq y^{\star} \cdot z \cdot y^{\star}$›
by (*metis add.commute add-iso*)
finally have $y \cdot (x + y^{\star} \cdot z) \leq (x + y^{\star} \cdot z) \cdot y^{\star}$
by (*metis add.commute add-idem′ add.left-commute distrib-right*)
moreover have $(x + y + z)^{\infty} \leq (x + y + y^{\star} \cdot z)^{\infty}$
by (*metis star-ref join.sup.mono eq-refl mult-1-left mult-isor iteration-iso*)
moreover have ... $= (x + y^{\star} \cdot z)^{\infty} \cdot y^{\infty}$
by (*metis add-commute calculation*(*1*) *iteration-sep2 local.add-left-comm*)
moreover have ... $= x^{\infty} \cdot (y^{\star} \cdot z)^{\infty} \cdot y^{\infty}$ **using** ‹$y^{\star} \cdot z \cdot x \leq x \cdot y^{\star} \cdot z$›
by (*metis iteration-sep mult.assoc*)
ultimately have $(x + y + z)^{\infty} \leq x^{\infty} \cdot (y + z)^{\infty}$
by (*metis add.commute mult.assoc iteration-denest3*)
thus *?thesis*
by (*metis add.commute add.left-commute less-eq-def iteration-subdenest*)
qed

Finally, we prove some blocking laws.

Nitpick refutes the next lemma.

lemma $x \cdot y = 0 \Longrightarrow x^{\infty} \cdot y = y$

oops

lemma *iteration-idep*: $x \cdot y = 0 \Longrightarrow x \cdot y^{\infty} = x$
by (*metis add-zeror annil iteration-unfoldl-distl*)

Nitpick refutes the next lemma.

lemma $y \cdot w \leq x \cdot y + z \Longrightarrow y \cdot w^{\infty} \leq x^{\infty} \cdot z$

oops

At the end of this file, we consider a data refinement example from von

Wright [30].

lemma *data-refinement*:
  assumes $s' \leq s \cdot z$ **and** $z \cdot e' \leq e$ **and** $z \cdot a' \leq a \cdot z$ **and** $z \cdot b \leq z$ **and** $b^\infty = b^\star$
  shows $s' \cdot (a' + b)^\infty \cdot e' \leq s \cdot a^\infty \cdot e$
proof $-$
  have $z \cdot b^\star \leq z$
    by (*metis assms*(*4*) *star-inductr-var*)
  have $(z \cdot a') \cdot b^\star \leq (a \cdot z) \cdot b^\star$
    by (*metis assms*(*3*) *mult.assoc mult-isor*)
  hence $z \cdot (a' \cdot b^\star)^\infty \leq a^\infty \cdot z$ **using** ‹$z \cdot b^\star \leq z$›
    by (*metis mult.assoc mult-isol order-trans iteration-sim mult.assoc*)
  have $s' \cdot (a' + b)^\infty \cdot e' \leq s' \cdot b^\star \cdot (a' \cdot b^\star)^\infty \cdot e'$
    by (*metis add.commute assms*(*5*) *eq-refl iteration-denest mult.assoc*)
  also have ... $\leq s \cdot z \cdot b^\star \cdot (a' \cdot b^\star)^\infty \cdot e'$
    by (*metis assms*(*1*) *mult-isor*)
  also have ... $\leq s \cdot z \cdot (a' \cdot b^\star)^\infty \cdot e'$ **using** ‹$z \cdot b^\star \leq z$›
    by (*metis mult.assoc mult-isol mult-isor*)
  also have ... $\leq s \cdot a^\infty \cdot z \cdot e'$ **using** ‹$z \cdot (a' \cdot b^\star)^\infty \leq a^\infty \cdot z$›
    by (*metis mult.assoc mult-isol mult-isor*)
  finally show *?thesis*
    by (*metis assms*(*2*) *mult.assoc mult-isol mult.assoc mult-isol order-trans*)
qed

end

end

12 Propositional Hoare Logic for Conway and Kleene Algebra

theory *PHL-KA*
  imports *Kleene-Algebra*

begin

This is a minimalist Hoare logic developed in the context of pre-dioids. In near-dioids, the sequencing rule would not be derivable. Iteration is modelled by a function that needs to satisfy a simulation law.

The main assumtions on pre-dioid elements needed to derive the Hoare rules are preservation properties; an additional distributivity propery is needed for the conditional rule.

This Hoare logic can be instantated in various ways. It covers notions of finite and possibly infinite iteration. In this theory, it it specialised to Conway and Kleene algebras.

class *it-pre-dioid* = *pre-dioid-one* +

fixes *it* :: $'a \Rightarrow 'a$
assumes *it-simr*: $y \cdot x \leq x \cdot y \Longrightarrow y \cdot it\ x \leq it\ x \cdot y$

begin

lemma *phl-while*:
assumes $p \cdot s \leq s \cdot p \cdot s$ **and** $p \cdot w \leq w \cdot p \cdot w$
and $(p \cdot s) \cdot x \leq x \cdot p$
shows $p \cdot (it\ (s \cdot x) \cdot w) \leq it\ (s \cdot x) \cdot w \cdot (p \cdot w)$
proof −
have $p \cdot s \cdot x \leq s \cdot x \cdot p$
by (*metis assms*(*1*) *assms*(*3*) *mult.assoc phl-export1*)
hence $p \cdot it\ (s \cdot x) \leq it\ (s \cdot x) \cdot p$
by (*simp add*: *it-simr mult.assoc*)
thus *?thesis*
using *assms*(*2*) *phl-export2* **by** *blast*
qed

end

Next we define a Hoare triple to make the format of the rules more explicit.

context *pre-dioid-one*
begin

abbreviation (**in** *near-dioid*) *ht* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$ (⦃-⦄-⦃-⦄) **where**
$⦃x⦄\ y\ ⦃z⦄ \equiv x \cdot y \leq y \cdot z$

lemma *ht-phl-skip*: $⦃x⦄\ 1\ ⦃x⦄$
by *simp*

lemma *ht-phl-cons1*: $x \leq w \Longrightarrow ⦃w⦄\ y\ ⦃z⦄ \Longrightarrow ⦃x⦄\ y\ ⦃z⦄$
by (*fact phl-cons1*)

lemma *ht-phl-cons2*: $w \leq x \Longrightarrow ⦃z⦄\ y\ ⦃w⦄ \Longrightarrow ⦃z⦄\ y\ ⦃x⦄$
by (*fact phl-cons2*)

lemma *ht-phl-seq*: $⦃p⦄\ x\ ⦃r⦄ \Longrightarrow ⦃r⦄\ y\ ⦃q⦄ \Longrightarrow ⦃p⦄\ x \cdot y\ ⦃q⦄$
by (*fact phl-seq*)

lemma *ht-phl-cond*:
assumes $u \cdot v \leq v \cdot u \cdot v$ **and** $u \cdot w \leq w \cdot u \cdot w$
and $\bigwedge x\ y.\ u \cdot (x + y) \leq u \cdot x + u \cdot y$
and $⦃u \cdot v⦄\ x\ ⦃z⦄$ **and** $⦃u \cdot w⦄\ y\ ⦃z⦄$
shows $⦃u⦄\ (v \cdot x + w \cdot y)\ ⦃z⦄$
using *assms* **by** (*fact phl-cond*)

lemma *ht-phl-export1*:
assumes $x \cdot y \leq y \cdot x \cdot y$
and $⦃x \cdot y⦄\ z\ ⦃w⦄$

shows $\{\!|x|\!\}\ y \cdot z\ \{\!|w|\!\}$
  using *assms* **by** (*fact phl-export1*)

lemma *ht-phl-export2*:
assumes $z \cdot w \leq w \cdot z \cdot w$
and $\{\!|x|\!\}\ y\ \{\!|z|\!\}$
shows $\{\!|x|\!\}\ y \cdot w\ \{\!|z \cdot w|\!\}$
  using *assms* **by** (*fact phl-export2*)

end

context *it-pre-dioid* **begin**

lemma *ht-phl-while*:
assumes $p \cdot s \leq s \cdot p \cdot s$ **and** $p \cdot w \leq w \cdot p \cdot w$
and $\{\!|p \cdot s|\!\}\ x\ \{\!|p|\!\}$
shows $\{\!|p|\!\}\ it\ (s \cdot x) \cdot w\ \{\!|p \cdot w|\!\}$
  using *assms* **by** (*fact phl-while*)

end

sublocale *pre-conway* < *phl*: *it-pre-dioid* **where** *it* = *dagger*
  by *standard* (*simp add*: *local.dagger-simr*)

sublocale *kleene-algebra* < *phl*: *it-pre-dioid* **where** *it* = *star* **..**

end

13 Propositional Hoare Logic for Demonic Refinement Algebra

In this section the generic iteration operator is instantiated to the strong iteration operator of demonic refinement algebra that models possibly infinite iteration.

theory *PHL-DRA*
  imports *DRA PHL-KA*
begin

sublocale *dra* < *total-phl*: *it-pre-dioid* **where** *it* = *strong-iteration*
  by *standard* (*simp add*: *local.iteration-sim*)

end

14 Finite Suprema

theory *Finite-Suprema*
imports *Dioid*

begin

This file contains an adaptation of Isabelle's library for finite sums to the case of (join) semilattices and dioids. In this setting, addition is idempotent; finite sums are finite suprema.

We add some basic properties of finite suprema for (join) semilattices and dioids.

14.1 Auxiliary Lemmas

lemma *fun-im*: $\{f\ a\ |a.\ a \in A\} = \{b.\ b \in f\ `\ A\}$
  by *auto*

lemma *fset-to-im*: $\{f\ x\ |x.\ x \in X\} = f\ `\ X$
  by *auto*

lemma *cart-flip-aux*: $\{f\ (snd\ p)\ (fst\ p)\ |p.\ p \in (B \times A)\} = \{f\ (fst\ p)\ (snd\ p)\ |p.\ p \in (A \times B)\}$
  by *auto*

lemma *cart-flip*: $(\lambda p.\ f\ (snd\ p)\ (fst\ p))\ `\ (B \times A) = (\lambda p.\ f\ (fst\ p)\ (snd\ p))\ `\ (A \times B)$
  by (*metis cart-flip-aux fset-to-im*)

lemma *fprod-aux*: $\{x \cdot y\ |x\ y.\ x \in (f\ `\ A) \wedge y \in (g\ `\ B)\} = \{f\ x \cdot g\ y\ |x\ y.\ x \in A \wedge y \in B\}$
  by *auto*

14.2 Finite Suprema in Semilattices

The first lemma shows that, in the context of semilattices, finite sums satisfy the defining property of finite suprema.

lemma *sum-sup*:
  assumes *finite* $(A :: 'a{::}join\text{-}semilattice\text{-}zero\ set)$
  shows $\sum A \leq z \longleftrightarrow (\forall a \in A.\ a \leq z)$
proof (*induct rule*: *finite-induct*[*OF assms*])
  fix $z :: 'a$
  show $(\sum \{\} \leq z) = (\forall a \in \{\}.\ a \leq z)$
    by *simp*
next
  fix $x\ z :: 'a$ **and** $F :: 'a\ set$
  assume *finF*: *finite F*
    and *xnF*: $x \notin F$
    and *indhyp*: $(\sum F \leq z) = (\forall a \in F.\ a \leq z)$
  show $(\sum (insert\ x\ F) \leq z) = (\forall a \in insert\ x\ F.\ a \leq z)$
  proof −
    have $\sum (insert\ x\ F) \leq z \longleftrightarrow (x + \sum F) \leq z$
      by (*metis finF sum.insert xnF*)

also have ... $\longleftrightarrow x \leq z \wedge \sum F \leq z$
by *simp*
also have ... $\longleftrightarrow x \leq z \wedge (\forall a \in F.\ a \leq z)$
by (*metis* (*lifting*) *indhyp*)
also have ... $\longleftrightarrow (\forall a \in insert\ x\ F.\ a \leq z)$
by (*metis insert-iff*)
ultimately show $(\sum (insert\ x\ F) \leq z) = (\forall a \in insert\ x\ F.\ a \leq z)$
by *blast*
qed
qed

This immediately implies some variants.

lemma *sum-less-eqI*:
$(\bigwedge x.\ x \in A \Longrightarrow f\ x \leq y) \Longrightarrow sum\ f\ A \leq (y::'a::join\text{-}semilattice\text{-}zero)$
apply (*atomize* (*full*))
apply (*case-tac finite A*)
apply (*erule finite-induct*)
apply *simp-all*
done

lemma *sum-less-eqE*:
$[\![\ sum\ f\ A \leq y;\ x \in A;\ finite\ A\ ]\!] \Longrightarrow f\ x \leq (y::'a::join\text{-}semilattice\text{-}zero)$
apply (*erule rev-mp*)
apply (*erule rev-mp*)
apply (*erule finite-induct*)
apply *auto*
done

lemma *sum-fun-image-sup*:
fixes $f :: 'a \Rightarrow 'b::join\text{-}semilattice\text{-}zero$
assumes *finite* $(A :: 'a\ set)$
shows $\sum (f\ `\ A) \leq z \longleftrightarrow (\forall a \in A.\ f\ a \leq z)$
by (*simp add*: *assms sum-sup*)

lemma *sum-fun-sup*:
fixes $f :: 'a \Rightarrow 'b::join\text{-}semilattice\text{-}zero$
assumes *finite* $(A ::'a\ set)$
shows $\sum \{f\ a \mid a.\ a \in A\} \leq z \longleftrightarrow (\forall a \in A.\ f\ a \leq z)$
by (*simp only*: *fset-to-im assms sum-fun-image-sup*)

lemma *sum-intro*:
assumes *finite* $(A :: 'a::join\text{-}semilattice\text{-}zero\ set)$ **and** *finite B*
shows $(\forall a \in A.\ \exists b \in B.\ a \leq b) \longrightarrow (\sum A \leq \sum B)$
by (*metis assms order-refl order-trans sum-sup*)

Next we prove an additivity property for suprema.

lemma *sum-union*:
assumes *finite* $(A :: 'a::join\text{-}semilattice\text{-}zero\ set)$
and *finite* $(B :: 'a::join\text{-}semilattice\text{-}zero\ set)$

shows $\sum (A \cup B) = \sum A + \sum B$
proof −
have $\forall z. \sum (A \cup B) \leq z \longleftrightarrow (\sum A + \sum B \leq z)$
by (*auto simp add*: *assms sum-sup*)
thus *?thesis*
by (*simp add*: *eq-iff*)
qed

It follows that the sum (supremum) of a two-element set is the join of its elements.

lemma *sum-bin*[*simp*]: $\sum \{(x :: 'a{::}join\text{-}semilattice\text{-}zero), y\} = x + y$
by (*subst insert-is-Un*, *subst sum-union*, *auto*)

Next we show that finite suprema are order preserving.

lemma *sum-iso*:
assumes *finite* (B :: $'a$::*join-semilattice-zero set*)
shows $A \subseteq B \longrightarrow \sum A \leq \sum B$
by (*metis assms finite-subset order-refl rev-subsetD sum-sup*)

The following lemmas state unfold properties for suprema and finite sets. They are subtly different from the non-idempotent case, where additional side conditions are required.

lemma *sum-insert* [*simp*]:
assumes *finite* (A :: $'a$::*join-semilattice-zero set*)
shows $\sum (insert\ x\ A) = x + \sum A$
proof −
have $\sum (insert\ x\ A) = \sum \{x\} + \sum A$
by (*metis insert-is-Un assms finite.emptyI finite.insertI sum-union*)
thus *?thesis*
by *auto*
qed

lemma *sum-fun-insert*:
fixes f :: $'a \Rightarrow 'b$::*join-semilattice-zero*
assumes *finite* (A :: $'a$ *set*)
shows $\sum (f \text{ ` } (insert\ x\ A)) = f\ x + \sum (f \text{ ` } A)$
by (*simp add*: *assms*)

Now we show that set comprehensions with nested suprema can be flattened.

lemma *flatten1-im*:
fixes f :: $'a \Rightarrow 'a \Rightarrow 'b$::*join-semilattice-zero*
assumes *finite* (A :: $'a$ *set*)
and *finite* (B :: $'a$ *set*)
shows $\sum ((\lambda x. \sum (f\ x \text{ ` } B)) \text{ ` } A) = \sum ((\lambda p. f\ (fst\ p)\ (snd\ p)) \text{ ` } (A \times B))$
proof −
have $\forall z. \sum ((\lambda x. \sum (f\ x \text{ ` } B)) \text{ ` } A) \leq z \longleftrightarrow \sum ((\lambda p. f\ (fst\ p)\ (snd\ p)) \text{ ` } (A \times B)) \leq z$
by (*simp add*: *assms finite-cartesian-product sum-fun-image-sup*)

thus *?thesis*
  by (*simp add*: *eq-iff*)
qed

lemma *flatten2-im*:
fixes f :: $'a \Rightarrow 'a \Rightarrow 'b$::*join-semilattice-zero*
assumes *finite* (A ::$'a$ *set*)
and *finite* (B ::$'a$ *set*)
shows $\sum((\lambda y.\ \sum\ ((\lambda x.\ f\ x\ y)\ `\ A))\ `\ B) = \sum((\lambda p.\ f\ (fst\ p)\ (snd\ p))\ `\ (A \times B))$
by (*simp only*: *flatten1-im assms cart-flip*)

lemma *sum-flatten1*:
fixes f :: $'a \Rightarrow 'a \Rightarrow 'b$::*join-semilattice-zero*
assumes *finite* (A :: $'a$ *set*)
and *finite* (B :: $'a$ *set*)
shows $\sum\{\sum\{f\ x\ y\ |y.\ y \in B\}\ |x.\ x \in A\} = \sum\{f\ x\ y\ |x\ y.\ x \in A \wedge y \in B\}$
apply (*simp add*: *fset-to-im assms flatten1-im*)
apply (*subst fset-to-im*[*symmetric*])
apply *simp*
done

lemma *sum-flatten2*:
fixes f :: $'a \Rightarrow 'a \Rightarrow 'b$::*join-semilattice-zero*
assumes *finite* A
and *finite* B
shows $\sum\{\sum\ \{f\ x\ y\ |x.\ x \in A\}\ |y.\ y \in B\} = \sum\{f\ x\ y\ |x\ y.\ x \in A \wedge y \in B\}$
apply (*simp add*: *fset-to-im assms flatten2-im*)
apply (*subst fset-to-im*[*symmetric*])
apply *simp*
done

Next we show another additivity property for suprema.

lemma *sum-fun-sum*:
fixes $f\ g$ :: $'a \Rightarrow 'b$::*join-semilattice-zero*
assumes *finite* (A :: $'a$ *set*)
shows $\sum((\lambda x.\ f\ x + g\ x)\ `\ A) = \sum(f\ `\ A) + \sum(g\ `\ A)$
proof −
{
  fix z:: $'b$
  have $\sum((\lambda x.\ f\ x + g\ x)\ `\ A) \le z \longleftrightarrow \sum(f\ `\ A) + \sum(g\ `\ A) \le z$
    by (*auto simp add*: *assms sum-fun-image-sup*)
}
thus *?thesis*
  by (*simp add*: *eq-iff*)
qed

The last lemma of this section prepares the distributivity laws that hold for dioids. It states that a strict additive function distributes over finite suprema, which is a continuity property in the finite.

lemma *sum-fun-add*:
  fixes f :: *'a::join-semilattice-zero* ⇒ *'b::join-semilattice-zero*
  assumes *finite* (X :: *'a set*)
  and *fstrict*: $f\ 0 = 0$
  and *fadd*: $\bigwedge x\ y.\ f\ (x + y) = f\ x + f\ y$
  shows $f\ (\sum X) = \sum (f\ `\ X)$
proof (*induct rule*: *finite-induct*[*OF assms*(*1*)])
  show $f\ (\sum \{\}) = \sum (f\ `\ \{\})$
    by (*metis fstrict image-empty sum.empty*)
  fix x :: *'a* **and** F :: *'a set*
  assume *finF*: *finite F*
    and *indhyp*: $f\ (\sum F) = \sum (f\ `\ F)$
  have $f\ (\sum (insert\ x\ F)) = f\ (x + \sum F)$
    by (*metis sum-insert finF*)
  also have ... $= f\ x + (f\ (\sum F))$
    by (*rule fadd*)
  also have ... $= f\ x + \sum (f\ `\ F)$
    by (*metis indhyp*)
  also have ... $= \sum (f\ `\ (insert\ x\ F))$
    by (*metis finF sum-fun-insert*)
  finally show $f\ (\sum (insert\ x\ F)) = \sum (f\ `\ insert\ x\ F)$ **.**
qed

14.3 Finite Suprema in Dioids

In this section we mainly prove variants of distributivity laws.

lemma *sum-distl*:
  assumes *finite Y*
  shows (x :: *'a::dioid-one-zero*) $\cdot (\sum Y) = \sum \{x \cdot y | y.\ y \in Y\}$
  by (*simp only*: *sum-fun-add assms annir distrib-left Collect-mem-eq fun-im*)

lemma *sum-distr*:
  assumes *finite X*
  shows $(\sum X) \cdot$ (y :: *'a::dioid-one-zero*) $= \sum \{x \cdot y | x.\ x \in X\}$
proof −
  have $(\sum X) \cdot y = \sum ((\lambda x.\ x \cdot y)\ `\ X)$
    by (*rule sum-fun-add*, *metis assms*, *rule annil*, *rule distrib-right*)
  thus *?thesis*
    by (*metis Collect-mem-eq fun-im*)
qed

lemma *sum-fun-distl*:
  fixes f :: *'a* ⇒ *'b::dioid-one-zero*
  assumes *finite* (Y :: *'a set*)
  shows $x \cdot \sum (f\ `\ Y) = \sum \{x \cdot f\ y\ | y.\ y \in Y\}$
  by (*simp add*: *assms fun-im image-image sum-distl*)

lemma *sum-fun-distr*:
  fixes f :: $'a \Rightarrow 'b$::*dioid-one-zero*
  assumes *finite* (X :: $'a$ *set*)
  shows $\sum (f \text{ ` } X) \cdot y = \sum \{f\ x \cdot y\ |x.\ x \in X\}$
  by (*simp add*: *assms fun-im image-image sum-distr*)

lemma *sum-distl-flat*:
  assumes *finite* (X ::$'a$::*dioid-one-zero set*)
  and *finite* Y
  shows $\sum \{x \cdot \sum Y\ |x.\ x \in X\} = \sum \{x \cdot y|x\ y.\ x \in X \wedge y \in Y\}$
  by (*simp only*: *assms sum-distl sum-flatten1*)

lemma *sum-distr-flat*:
  assumes *finite* X
  and *finite* (Y :: $'a$::*dioid-one-zero set*)
  shows $\sum \{(\sum X) \cdot y\ |y.\ y \in Y\} = \sum \{x \cdot y|x\ y.\ x \in X \wedge y \in Y\}$
  by (*simp only*: *assms sum-distr sum-flatten2*)

lemma *sum-sum-distl*:
  assumes *finite* (X :: $'a$::*dioid-one-zero set*)
  and *finite* Y
  shows $\sum ((\lambda x.\ x \cdot (\sum Y)) \text{ ` } X) = \sum \{x \cdot y\ |x\ y.\ x \in X \wedge y \in Y\}$
proof −
  have $\sum ((\lambda x.\ x \cdot (\sum Y)) \text{ ` } X) = \sum \{\sum \{x \cdot y\ |y.\ y \in Y\}\ |x.\ x \in X\}$
    by (*auto simp add*: *sum-distl assms fset-to-im*)
  thus *?thesis*
    by (*simp add*: *assms sum-flatten1*)
qed

lemma *sum-sum-distr*:
  assumes *finite* X
  and *finite* Y
  shows $\sum ((\lambda y.\ (\sum X) \cdot (y :: 'a\text{::dioid-one-zero})) \text{ ` } Y) = \sum \{x \cdot y|x\ y.\ x \in X \wedge$
$y \in Y\}$
proof −
  have $\sum ((\lambda y.\ (\sum X) \cdot y) \text{ ` } Y) = \sum \{\sum \{x \cdot y\ |x.\ x \in X\}\ |y.\ y \in Y\}$
    by (*auto simp add*: *sum-distr assms fset-to-im*)
  thus *?thesis*
    by (*simp add*: *assms sum-flatten2*)
qed

lemma *sum-sum-distl-fun*:
  fixes $f\ g$ :: $'a \Rightarrow 'b$::*dioid-one-zero*
  fixes h :: $'a \Rightarrow 'a$ *set*
  assumes $\bigwedge x.$ *finite* $(h\ x)$
  and *finite* X
  shows $\sum ((\lambda x.\ f\ x \cdot \sum (g \text{ ` } h\ x)) \text{ ` } X) = \sum \{\sum\ \{f\ x \cdot g\ y\ |y.\ y \in h\ x\}\ |x.\ x \in$
$X\}$
  by (*auto simp add*: *sum-fun-distl assms fset-to-im*)

lemma *sum-sum-distr-fun*:
  fixes *f g* :: $'a \Rightarrow 'b$::*dioid-one-zero*
  fixes *h* :: $'a \Rightarrow 'a$ *set*
  assumes *finite Y*
  and $\bigwedge y$. *finite* $(h\ y)$
  shows $\sum ((\lambda y. \sum (f \text{ ` } h\ y) \cdot g\ y) \text{ ` } Y) = \sum \{\sum \{f\ x \cdot g\ y\ | x.\ x \in (h\ y)\}\ | y.\ y \in Y\}$
  by (*auto simp add*: *sum-fun-distr assms fset-to-im*)

lemma *sum-dist*:
  assumes *finite* $(A :: 'a$::*dioid-one-zero set*$)$
  and *finite B*
  shows $(\sum A) \cdot (\sum B) = \sum \{x \cdot y\ | x\ y.\ x \in A \wedge y \in B\}$
proof −
  have $(\sum A) \cdot (\sum B) = \sum \{x \cdot \sum B\ | x.\ x \in A\}$
    by (*simp add*: *assms sum-distr*)
  also have ... $= \sum \{\sum \{x \cdot y\ | y.\ y \in B\}\ | x.\ x \in A\}$
    by (*simp add*: *assms sum-distl*)
  finally show *?thesis*
    by (*simp only*: *sum-flatten1 assms finite-cartesian-product*)
qed

lemma *dioid-sum-prod-var*:
  fixes *f g* :: $'a \Rightarrow 'b$::*dioid-one-zero*
  assumes *finite* $(A :: 'a$ *set*$)$
  shows $(\sum (f \text{ ` } A)) \cdot (\sum (g \text{ ` } A)) = \sum \{f\ x \cdot g\ y\ | x\ y.\ x \in A \wedge y \in A\}$
  by (*simp add*: *assms sum-dist fprod-aux*)

lemma *dioid-sum-prod*:
  fixes *f g* :: $'a \Rightarrow 'b$::*dioid-one-zero*
  assumes *finite* $(A :: 'a$ *set*$)$
  shows $(\sum \{f\ x\ | x.\ x \in A\}) \cdot (\sum \{g\ x\ | x.\ x \in A\}) = \sum \{f\ x \cdot g\ y\ | x\ y.\ x \in A \wedge y \in A\}$
  by (*simp add*: *assms dioid-sum-prod-var fset-to-im*)

lemma *sum-image*:
  fixes *f* :: $'a \Rightarrow 'b$::*join-semilattice-zero*
  assumes *finite X*
  shows *sum f X* $= \sum (f \text{ ` } X)$
using *assms*
proof (*induct rule*: *finite-induct*)
  case *empty* **thus** *?case* **by** *simp*
next
  case *insert* **thus** *?case*
    by (*metis sum.insert sum-fun-insert*)
qed

lemma *sum-interval-cong*:

$\llbracket \bigwedge i.\ \llbracket m \leq i; i \leq n \rrbracket \Longrightarrow P(i) = Q(i) \rrbracket \Longrightarrow (\sum i{=}m..n.\ P(i)) = (\sum i{=}m..n.\ Q(i))$
by (*auto intro*: *sum.cong*)

lemma *sum-interval-distl*:
fixes f :: *nat* ⇒ *'a*::*dioid-one-zero*
assumes $m \leq n$
shows $x \cdot (\sum i{=}m..n.\ f(i)) = (\sum i{=}m..n.\ (x \cdot f(i)))$
proof −
have $x \cdot (\sum i{=}m..n.\ f(i)) = x \cdot \sum (f \text{ ` } \{m..n\})$
by (*metis finite-atLeastAtMost sum-image*)
also have ... $= \sum \{x \cdot y \mid y.\ y \in f \text{ ` } \{m..n\}\}$
by (*metis finite-atLeastAtMost fset-to-im image-image sum-fun-distl*)
also have ... $= \sum ((\lambda i.\ x \cdot f\ i) \text{ ` } \{m..n\})$
by (*metis fset-to-im image-image*)
also have ... $= (\sum i{=}m..n.\ (x \cdot f(i)))$
by (*metis finite-atLeastAtMost sum-image*)
finally show *?thesis* **.**
qed

lemma *sum-interval-distr*:
fixes f :: *nat* ⇒ *'a*::*dioid-one-zero*
assumes $m \leq n$
shows $(\sum i{=}m..n.\ f(i)) \cdot y = (\sum i{=}m..n.\ (f(i) \cdot y))$
proof −
have $(\sum i{=}m..n.\ f(i)) \cdot y = \sum (f \text{ ` } \{m..n\}) \cdot y$
by (*metis finite-atLeastAtMost sum-image*)
also have ... $= \sum \{x \cdot y \mid x.\ x \in f \text{ ` } \{m..n\}\}$
by (*metis calculation finite-atLeastAtMost finite-imageI fset-to-im sum-distr*)
also have ... $= \sum ((\lambda i.\ f(i) \cdot y) \text{ ` } \{m..n\})$
by (*auto intro*: *sum.cong*)
also have ... $= (\sum i{=}m..n.\ (f(i) \cdot y))$
by (*metis finite-atLeastAtMost sum-image*)
finally show *?thesis* **.**
qed

There are interesting theorems for finite sums in Kleene algebras; we leave them for future consideration.

end

15 Formal Power Series

theory *Formal-Power-Series*
imports *Finite-Suprema Kleene-Algebra*
begin

15.1 The Type of Formal Power Series

Formal powerseries are functions from a free monoid into a dioid. They have applications in formal language theory, e.g., weighted automata. As usual, we represent elements of a free monoid by lists.

This theory generalises Amine Chaieb's development of formal power series as functions from natural numbers, which may be found in *HOL/Library/Formal_Power_Series.thy*.

typedef *('a, 'b) fps* = {*f*::*'a list* ⇒ *'b*. *True*}
  morphisms *fps-nth Abs-fps*
  by *simp*

It is often convenient to reason about functions, and transfer results to formal power series.

setup-lifting *type-definition-fps*

declare *fps-nth-inverse* [*simp*]

notation *fps-nth* (**infixl** $ *75*)

lemma *expand-fps-eq*: *p* = *q* ⟷ (∀ *n*. *p* $ *n* = *q* $ *n*)
by (*simp add*: *fps-nth-inject* [*symmetric*] *fun-eq-iff*)

lemma *fps-ext*: (⋀*n*. *p* $ *n* = *q* $ *n*) ⟹ *p* = *q*
by (*simp add*: *expand-fps-eq*)

lemma *fps-nth-Abs-fps* [*simp*]: *Abs-fps f* $ *n* = *f n*
by (*simp add*: *Abs-fps-inverse*)

15.2 Definition of the Basic Elements 0 and 1 and the Basic Operations of Addition and Multiplication

The zero formal power series maps all elements of the monoid (all lists) to zero.

instantiation *fps* :: (*type*,*zero*) *zero*
begin
  definition *zero-fps* **where**
    0 = *Abs-fps* (λ*n*. *0*)
  instance ..
end

lemma *fps-zero-nth* [*simp*]: *0* $ *n* = *0*
unfolding *zero-fps-def* **by** *simp*

The unit formal power series maps the monoidal unit (the empty list) to one and all other elements to zero.

instantiation *fps* :: (*type*,{*one*,*zero*}) *one*
begin

definition *one-fps* **where**
$1 = Abs\text{-}fps\ (\lambda n.\ if\ n = []\ then\ 1\ else\ 0)$
instance ..
end

lemma *fps-one-nth-Nil* [*simp*]: $1\ \$\ [] = 1$
unfolding *one-fps-def* **by** *simp*

lemma *fps-one-nth-Cons* [*simp*]: $1\ \$\ (x\ \#\ xs) = 0$
unfolding *one-fps-def* **by** *simp*

Addition of formal power series is the usual pointwise addition of functions.

instantiation *fps* :: (*type*,*plus*) *plus*
begin
definition *plus-fps* **where**
$f + g = Abs\text{-}fps\ (\lambda n.\ f\ \$\ n + g\ \$\ n)$
instance ..
end

lemma *fps-add-nth* [*simp*]: $(f + g)\ \$\ n = f\ \$\ n + g\ \$\ n$
unfolding *plus-fps-def* **by** *simp*

This directly shows that formal power series form a semilattice with zero.

lemma *fps-add-assoc*: $((f::('a,'b::semigroup\text{-}add)\ fps) + g) + h = f + (g + h)$
unfolding *plus-fps-def* **by** (*simp add*: *add.assoc*)

lemma *fps-add-comm* [*simp*]: $(f::('a,'b::ab\text{-}semigroup\text{-}add)\ fps) + g = g + f$
unfolding *plus-fps-def* **by** (*simp add*: *add.commute*)

lemma *fps-add-idem* [*simp*]: $(f::('a,'b::join\text{-}semilattice)\ fps) + f = f$
unfolding *plus-fps-def* **by** *simp*

lemma *fps-zerol* [*simp*]: $(f::('a,'b::monoid\text{-}add)\ fps) + 0 = f$
unfolding *plus-fps-def* **by** *simp*

lemma *fps-zeror* [*simp*]: $0 + (f::('a,'b::monoid\text{-}add)\ fps) = f$
unfolding *plus-fps-def* **by** *simp*

The product of formal power series is convolution. The product of two formal powerseries at a list is obtained by splitting the list into all possible prefix/suffix pairs, taking the product of the first series applied to the first coordinate and the second series applied to the second coordinate of each pair, and then adding the results.

instantiation *fps* :: (*type*,{*comm-monoid-add*,*times*}) *times*
begin
definition *times-fps* **where**
$f * g = Abs\text{-}fps\ (\lambda n.\ \sum\{f\ \$\ y * g\ \$\ z\ |y\ z.\ n = y\ @\ z\})$
instance ..

end

We call the set of all prefix/suffix splittings of a list *xs* the *splitset* of *xs*.

definition *splitset* **where**
splitset xs ≡ {(*p*, *q*). *xs* = *p* @ *q*}

Altenatively, splitsets can be defined recursively, which yields convenient simplification rules in Isabelle.

fun *splitset-fun* **where**
splitset-fun [] = {([], [])}
| *splitset-fun* (*x* # *xs*) = *insert* ([], *x* # *xs*) (*apfst* (*Cons x*) ' *splitset-fun xs*)

lemma *splitset-consl*:
splitset (*x* # *xs*) = *insert* ([], *x* # *xs*) (*apfst* (*Cons x*) ' *splitset xs*)
by (*auto simp add*: *image-def splitset-def*) (*metis append-eq-Cons-conv*)+

lemma *splitset-eq-splitset-fun*: *splitset xs* = *splitset-fun xs*
apply (*induct xs*)
apply (*simp add*: *splitset-def*)
apply (*simp add*: *splitset-consl*)
done

The definition of multiplication is now more precise.

lemma *fps-mult-var*:
$(f * g)\ \$\ n = \sum\{f\ \$\ (fst\ p) * g\ \$\ (snd\ p) \mid p.\ p \in splitset\ n\}$
by (*simp add*: *times-fps-def splitset-def*)

lemma *fps-mult-image*:
$(f * g)\ \$\ n = \sum((\lambda p.\ f\ \$\ (fst\ p) * g\ \$\ (snd\ p))\ `\ splitset\ n)$
by (*simp only*: *Collect-mem-eq fps-mult-var fun-im*)

Next we show that splitsets are finite and non-empty.

lemma *splitset-fun-finite* [*simp*]: *finite* (*splitset-fun xs*)
by (*induct xs*, *simp-all*)

lemma *splitset-finite* [*simp*]: *finite* (*splitset xs*)
by (*simp add*: *splitset-eq-splitset-fun*)

lemma *split-append-finite* [*simp*]: *finite* {(*p*, *q*). *xs* = *p* @ *q*}
by (*fold splitset-def*, *fact splitset-finite*)

lemma *splitset-fun-nonempty* [*simp*]: *splitset-fun xs* ≠ {}
by (*cases xs*, *simp-all*)

lemma *splitset-nonempty* [*simp*]: *splitset xs* ≠ {}
by (*simp add*: *splitset-eq-splitset-fun*)

We now proceed with proving algebraic properties of formal power series.

lemma *fps-annil* [*simp*]:
  0 ∗ (*f*::(′*a*::*type*,′*b*::{*comm-monoid-add*,*mult-zero*}) *fps*) = *0*
by (*rule fps-ext*) (*simp add*: *times-fps-def sum.neutral*)

lemma *fps-annir* [*simp*]:
  (*f*::(′*a*::*type*,′*b*::{*comm-monoid-add*,*mult-zero*}) *fps*) ∗ *0* = *0*
by (*simp add*: *fps-ext times-fps-def sum.neutral*)

lemma *fps-distl*:
  (*f*::(′*a*::*type*,′*b*::{*join-semilattice-zero*,*semiring*}) *fps*) ∗ (*g* + *h*) = (*f* ∗ *g*) + (*f* ∗ *h*)
by (*simp add*: *fps-ext fps-mult-image distrib-left sum-fun-sum*)

lemma *fps-distr*:
  ((*f*::(′*a*::*type*,′*b*::{*join-semilattice-zero*,*semiring*}) *fps*) + *g*) ∗ *h* = (*f* ∗ *h*) + (*g* ∗ *h*)
by (*simp add*: *fps-ext fps-mult-image distrib-right sum-fun-sum*)

The multiplicative unit laws are surprisingly tedious. For the proof of the left unit law we use the recursive definition, which we could as well have based on splitlists instead of splitsets.

However, a right unit law cannot simply be obtained along the lines of this proofs. The reason is that an alternative recursive definition that produces a unit with coordinates flipped would be needed. But this is difficult to obtain without snoc lists. We therefore prove the right unit law more directly by using properties of suprema.

lemma *fps-onel* [*simp*]:
  1 ∗ (*f*::(′*a*::*type*,′*b*::{*join-semilattice-zero*,*monoid-mult*,*mult-zero*}) *fps*) = *f*
proof (*rule fps-ext*)
  fix *n* :: ′*a list*
  show (*1* ∗ *f*) $ *n* = *f* $ *n*
  proof (*cases n*)
    case *Nil* **thus** *?thesis*
      by (*simp add*: *times-fps-def*)
  next
    case *Cons* **thus** *?thesis*
      by (*simp add*: *fps-mult-image splitset-eq-splitset-fun image-comp one-fps-def comp-def image-constant-conv*)
  qed
qed

lemma *fps-oner* [*simp*]:
  (*f*::(′*a*::*type*,′*b*::{*join-semilattice-zero*,*monoid-mult*,*mult-zero*}) *fps*) ∗ *1* = *f*
proof (*rule fps-ext*)
  fix *n* :: ′*a list*
  {
    fix *z* :: ′*b*
    have (*f* ∗ *1*) $ *n* ≤ *z* ⟷ (∀ *p* ∈ *splitset n*. *f* $ (*fst p*) ∗ *1* $ (*snd p*) ≤ *z*)

by (*simp add*: *fps-mult-image sum-fun-image-sup*)
also have ... ⟷ (∀ *a b*. *n* = *a* @ *b* ⟶ *f* $ *a* ∗ *1* $ *b* ≤ *z*)
unfolding *splitset-def* **by** *simp*
also have ... ⟷ (*f* $ *n* ∗ *1* $ [] ≤ *z*)
by (*simp add*: *one-fps-def*)
finally have (*f* ∗ *1*) $ *n* ≤ *z* ⟷ *f* $ *n* ≤ *z*
by *simp*
}
thus (*f* ∗ *1*) $ *n* = *f* $ *n*
by (*metis eq-iff*)
qed

Finally we prove associativity of convolution. This requires splitting lists into three parts and rearranging these parts in two different ways into splitsets. This rearrangement is captured by the following technical lemma.

lemma *splitset-rearrange*:
fixes *F* :: ′*a list* ⇒ ′*a list* ⇒ ′*a list* ⇒ ′*b*::*join-semilattice-zero*
shows ∑{∑{*F* (*fst p*) (*fst q*) (*snd q*) | *q*. *q* ∈ *splitset* (*snd p*)} | *p*. *p* ∈ *splitset* *x*} =
∑{∑{*F* (*fst q*) (*snd q*) (*snd p*) | *q*. *q* ∈ *splitset* (*fst p*)} | *p*. *p* ∈ *splitset* *x*}
(**is** *?lhs* = *?rhs*)
proof −
{
fix *z* :: ′*b*
have *?lhs* ≤ *z* ⟷ (∀ *p q r*. *x* = *p* @ *q* @ *r* ⟶ *F p q r* ≤ *z*)
by (*simp only*: *fset-to-im sum-fun-image-sup splitset-finite*)
(*auto simp add*: *splitset-def*)
hence *?lhs* ≤ *z* ⟷ *?rhs* ≤ *z*
by (*simp only*: *fset-to-im sum-fun-image-sup splitset-finite*)
(*auto simp add*: *splitset-def*)
}
thus *?thesis*
by (*simp add*: *eq-iff*)
qed

lemma *fps-mult-assoc*: (*f*::(′*a*::*type*,′*b*::*dioid-one-zero*) *fps*) ∗ (*g* ∗ *h*) = (*f* ∗ *g*) ∗ *h*
proof (*rule fps-ext*)
fix *n* :: ′*a list*
have (*f* ∗ (*g* ∗ *h*)) $ *n* = ∑{∑{*f* $ (*fst p*) ∗ *g* $ (*fst q*) ∗ *h* $ (*snd q*) | *q*. *q* ∈ *splitset* (*snd p*)} | *p*. *p* ∈ *splitset n*}
by (*simp add*: *fps-mult-image sum-sum-distl-fun mult.assoc*)
also have ... = ∑{∑{*f* $ (*fst q*) ∗ *g* $ (*snd q*) ∗ *h* $ (*snd p*) | *q*. *q* ∈ *splitset* (*fst p*)} | *p*. *p* ∈ *splitset n*}
by (*fact splitset-rearrange*)
finally show (*f* ∗ (*g* ∗ *h*)) $ *n* = ((*f* ∗ *g*) ∗ *h*) $ *n*
by (*simp add*: *fps-mult-image sum-sum-distr-fun mult.assoc*)
qed

15.3 The Dioid Model of Formal Power Series

We can now show that formal power series with suitably defined operations form a dioid. Many of the underlying properties already hold in weaker settings, where the target algebra is a semilattice or semiring. We currently ignore this fact.

subclass (**in** *dioid-one-zero*) *mult-zero*
proof
  fix x :: $'a$
  show $0 * x = 0$
    by (*fact annil*)
  show $x * 0 = 0$
    by (*fact annir*)
qed

instantiation *fps* :: (*type*,*dioid-one-zero*) *dioid-one-zero*
begin

  definition *less-eq-fps* **where**
    $(f::('a,'b)\ fps) \leq g \longleftrightarrow f + g = g$

  definition *less-fps* **where**
    $(f::('a,'b)\ fps) < g \longleftrightarrow f \leq g \wedge f \neq g$

  instance
  proof
    fix $f\ g\ h$ :: $('a,'b)$ *fps*
    show $f + g + h = f + (g + h)$
      by (*fact fps-add-assoc*)
    show $f + g = g + f$
      by (*fact fps-add-comm*)
    show $f * g * h = f * (g * h)$
      by (*metis fps-mult-assoc*)
    show $(f + g) * h = f * h + g * h$
      by (*fact fps-distr*)
    show $1 * f = f$
      by (*fact fps-onel*)
    show $f * 1 = f$
      by (*fact fps-oner*)
    show $0 + f = f$
      by (*fact fps-zeror*)
    show $0 * f = 0$
      by (*fact fps-annil*)
    show $f * 0 = 0$
      by (*fact fps-annir*)
    show $f \leq g \longleftrightarrow f + g = g$
      by (*fact less-eq-fps-def*)
    show $f < g \longleftrightarrow f \leq g \wedge f \neq g$
      by (*fact less-fps-def*)

show $f + f = f$
  by (*fact fps-add-idem*)
show $f * (g + h) = f \cdot g + f \cdot h$
  by (*fact fps-distl*)
qed

end

lemma *expand-fps-less-eq*: $(f::('a,'b::dioid\text{-}one\text{-}zero)\ fps) \leq g \longleftrightarrow (\forall n.\ f\ \$\ n \leq g\ \$\ n)$
by (*simp add*: *expand-fps-eq less-eq-def less-eq-fps-def*)

15.4 The Kleene Algebra Model of Formal Power Series

There are two approaches to define the Kleene star. The first one defines the star for a certain kind of (so-called proper) formal power series into a semiring or dioid. The second one, which is more interesting in the context of our algebraic hierarchy, shows that formal power series into a Kleene algebra form a Kleene algebra. We have only formalised the latter approach.

lemma *Sum-splitlist-nonempty*:
$\sum\{f\ ys\ zs\ |ys\ zs.\ xs = ys\ @\ zs\} = ((f\ []\ xs)::'a::join\text{-}semilattice\text{-}zero) + \sum\{f\ ys\ zs\ |ys\ zs.\ xs = ys\ @\ zs \wedge ys \neq []\}$
proof –
have $\{f\ ys\ zs\ |ys\ zs.\ xs = ys\ @\ zs\} = \{f\ ys\ zs\ |ys\ zs.\ xs = ys\ @\ zs \wedge ys = []\} \cup \{f\ ys\ zs\ |ys\ zs.\ xs = ys\ @\ zs \wedge ys \neq []\}$
  by *blast*
thus *?thesis* **using** [[*simproc add*: *finite-Collect*]]
  by (*simp add*: *sum.insert*)
qed

lemma (**in** *left-kleene-algebra*) *add-star-eq*:
$x + y \cdot y^\star \cdot x = y^\star \cdot x$
by (*metis add.commute mult-onel star2 star-one troeger*)

declare *rev-conj-cong*[*fundef-cong*]
— required for the function package to prove termination of *star-fps-rep*

fun *star-fps-rep* **where**
star-fps-rep-Nil: $star\text{-}fps\text{-}rep\ f\ [] = (f\ [])^\star$
| *star-fps-rep-Cons*: $star\text{-}fps\text{-}rep\ f\ n = (f\ [])^\star \cdot \sum\{f\ y \cdot star\text{-}fps\text{-}rep\ f\ z\ |y\ z.\ n = y\ @\ z \wedge y \neq []\}$

instantiation *fps* :: (*type*,*kleene-algebra*) *kleene-algebra*
begin

We first define the star on functions, where we can use Isabelle's package for recursive functions, before lifting the definition to the type of formal power series.

This definition of the star is from an unpublished manuscript by Esik and Kuich.

```
lift-definition star-fps :: ('a, 'b) fps ⇒ ('a, 'b) fps is star-fps-rep ..

lemma star-fps-Nil [simp]: f⋆ $ [] = (f $ [])⋆
by (simp add: star-fps-def)

lemma star-fps-Cons [simp]: f⋆ $ (x # xs) = (f $ [])⋆ · ∑{f $ y · f⋆ $ z |y z.
x # xs = y @ z ∧ y ≠ []}
by (simp add: star-fps-def)

instance
proof
  fix f g h :: ('a,'b) fps
  have 1 + f · f⋆ = f⋆
    apply (rule fps-ext)
    apply (case-tac n)
     apply (auto simp add: times-fps-def)
   apply (simp add: add-star-eq mult.assoc[THEN sym] Sum-splitlist-nonempty)
    apply (simp add: add-star-eq join.sup-commute)
  done
  thus 1 + f · f⋆ ≤ f⋆
    by (metis order-refl)
  have f · g ≤ g ⟶ f⋆ · g ≤ g
    proof
      assume f · g ≤ g
      hence 1: ⋀u v. f $ u · g $ v ≤ g $ (u @ v)
        using [[simproc add: finite-Collect]]
        apply (simp add: expand-fps-less-eq)
        apply (drule-tac x=u @ v in spec)
        apply (simp add: times-fps-def)
        apply (auto elim!: sum-less-eqE)
      done
      hence 2: ⋀v. (f $ []) ⋆ · g $ v ≤ g $ v
        apply (subgoal-tac f $ [] · g $ v ≤ g $ v)
         apply (metis star-inductl-var)
        apply (metis append-Nil)
      done
      show f⋆ · g ≤ g
        using [[simproc add: finite-Collect]]
       apply (auto intro!: sum-less-eqI simp add: expand-fps-less-eq times-fps-def)
        apply (induct-tac y rule: length-induct)
        apply (case-tac xs)
         apply (simp add: 2)
        using 2 apply (auto simp add: mult.assoc sum-distr)
        apply (rule-tac y=(f $ [])⋆ · g $ (a # list @ z) in order-trans)
         prefer 2
         apply (rule 2)
        apply (auto intro!: mult-isol[rule-format] sum-less-eqI)
```

apply (*drule-tac x=za* **in** *spec*)
apply (*drule mp*)
apply (*metis append-eq-Cons-conv length-append less-not-refl2 add.commute not-less-eq trans-less-add1*)
apply (*drule-tac z=f* \$ *y* **in** *mult-isol*[*rule-format*])
apply (*auto elim*!: *order-trans simp add*: *mult.assoc*)
apply (*metis 1 append-Cons append-assoc*)
done
qed
thus $h + f \cdot g \leq g \Longrightarrow f^\star \cdot h \leq g$
by (*metis* (*no-types*, *lifting*) *distrib-left join.sup.bounded-iff less-eq-def*)
have $g \cdot f \leq g \longrightarrow g \cdot f^\star \leq g$
— this property is dual to the previous one; the proof is slightly different
proof
assume $g \cdot f \leq g$
hence *1*: $\bigwedge u\ v.\ g\ \$\ u \cdot f\ \$\ v \leq g\ \$\ (u\ @\ v)$
using [[*simproc add*: *finite-Collect*]]
apply (*simp add*: *expand-fps-less-eq*)
apply (*drule-tac x=u* @ *v* **in** *spec*)
apply (*simp add*: *times-fps-def*)
apply (*auto elim*!: *sum-less-eqE*)
done
hence *2*: $\bigwedge u.\ g\ \$\ u \cdot (f\ \$\ [])^\star \leq g\ \$\ u$
apply (*subgoal-tac* $g\ \$\ u \cdot f\ \$\ [] \leq g\ \$\ u$)
apply (*metis star-inductr-var*)
apply (*metis append-Nil2*)
done
show $g \cdot f^\star \leq g$
using [[*simproc add*: *finite-Collect*]]
apply (*auto intro*!: *sum-less-eqI simp add*: *expand-fps-less-eq times-fps-def*)
apply (*rule-tac* $P=\lambda y.\ g\ \$\ y \cdot f^\star\ \$\ z \leq g\ \$\ (y\ @\ z)$ **and** *x=y* **in** *allE*)
prefer *2*
apply *assumption*
apply (*induct-tac z rule*: *length-induct*)
apply (*case-tac xs*)
apply (*simp add*: *2*)
apply (*auto intro*!: *sum-less-eqI simp add*: *sum-distl*)
apply (*rule-tac* $y=g\ \$\ x \cdot f\ \$\ yb \cdot f^\star\ \$\ z$ **in** *order-trans*)
apply (*simp add*: *2 mult.assoc*[*THEN sym*] *mult-isor*)
apply (*rule-tac* $y=g\ \$\ (x\ @\ yb) \cdot f^\star\ \$\ z$ **in** *order-trans*)
apply (*simp add*: *1 mult-isor*)
apply (*drule-tac x=z* **in** *spec*)
apply (*drule mp*)
apply (*metis append-eq-Cons-conv length-append less-not-refl2 add.commute not-less-eq trans-less-add1*)
apply (*metis append-assoc*)
done
qed
thus $h + g \cdot f \leq g \Longrightarrow h \cdot f^\star \leq g$

by (*metis* (*no-types*, *lifting*) *distrib-right′ join.sup.bounded-iff order-prop*)
qed

end

end

16 Infinite Matrices

theory *Inf-Matrix*
imports *Finite-Suprema*
begin

Matrices are functions from two index sets into some suitable algebra. We consider arbitrary index sets, not necessarily the positive natural numbers up to some bounds; our coefficient algebra is a dioid. Our only restriction is that summation in the product of matrices is over a finite index set. This follows essentially Droste and Kuich's introductory article in the Handbook of Weighted Automata [10].

Under these assumptions we show that dioids are closed under matrix formation. Our proofs are similar to those for formal power series, but simpler.

type-synonym ($'a$, $'b$, $'c$) *matrix* = $'a \Rightarrow 'b \Rightarrow 'c$

definition *mat-one* :: ($'a$, $'a$, $'c$::*dioid-one-zero*) *matrix* (ε) **where**
$\varepsilon\ i\ j \equiv$ (*if* ($i = j$) *then 1 else 0*)

definition *mat-zero* :: ($'a$, $'b$, $'c$::*dioid-one-zero*) *matrix* (δ) **where**
$\delta \equiv \lambda j\ i.\ 0$

definition *mat-add* :: ($'a$, $'b$, $'c$::*dioid-one-zero*) *matrix* $\Rightarrow$ ($'a$, $'b$, $'c$) *matrix* $\Rightarrow$ ($'a$, $'b$, $'c$) *matrix* (**infixl** $\oplus$ *70*) **where**
$(f \oplus g) \equiv \lambda i\ j.\ (f\ i\ j) + (g\ i\ j)$

lemma *mat-add-assoc*: $(f \oplus g) \oplus h = f \oplus (g \oplus h)$
by (*auto simp add*: *mat-add-def join.sup-assoc*)

lemma *mat-add-comm*: $f \oplus g = g \oplus f$
by (*auto simp add*: *mat-add-def join.sup-commute*)

lemma *mat-add-idem*[*simp*]: $f \oplus f = f$
by (*auto simp add*: *mat-add-def*)

lemma *mat-zerol*[*simp*]: $f \oplus \delta = f$
by (*auto simp add*: *mat-add-def mat-zero-def*)

lemma *mat-zeror*[*simp*]: $\delta \oplus f = f$
by (*auto simp add*: *mat-add-def mat-zero-def*)

definition *mat-mult* :: ($'a$, $'k$::*finite*, $'c$::*dioid-one-zero*) *matrix* ⇒ ($'k$, $'b$, $'c$) *matrix* ⇒ ($'a$, $'b$, $'c$) *matrix* (**infixl** ⊗ *60*) **where**
$(f \otimes g)\ i\ j \equiv \sum \{(f\ i\ k) \cdot (g\ k\ j) \mid k.\ k \in UNIV\}$

lemma *mat-annil*[*simp*]: $\delta \otimes f = \delta$
by (*rule ext*, *auto simp add*: *mat-mult-def mat-zero-def*)

lemma *mat-annir*[*simp*]: $f \otimes \delta = \delta$
by (*rule ext*, *auto simp add*: *mat-mult-def mat-zero-def*)

lemma *mat-distl*: $f \otimes (g \oplus h) = (f \otimes g) \oplus (f \otimes h)$
proof −
{
fix *i j*
have $(f \otimes (g \oplus h))\ i\ j = \sum\{f\ i\ k \cdot (g\ k\ j + h\ k\ j) \mid k.\ k \in UNIV\}$
by (*simp only*: *mat-mult-def mat-add-def*)
also have ... $= \sum\{f\ i\ k \cdot g\ k\ j + f\ i\ k \cdot h\ k\ j \mid k.\ k \in UNIV\}$
by (*simp only*: *distrib-left*)
also have ... $= \sum\{f\ i\ k \cdot g\ k\ j \mid k.\ k \in UNIV\} + \sum\{f\ i\ k \cdot h\ k\ j \mid k.\ k \in UNIV\}$
by (*simp only*: *fset-to-im sum-fun-sum finite-UNIV*)
finally have $(f \otimes (g \oplus h))\ i\ j = ((f \otimes g) \oplus (f \otimes h))\ i\ j$
by (*simp only*: *mat-mult-def mat-add-def*)
}
thus *?thesis*
by *auto*
qed

lemma *mat-distr*: $(f \oplus g) \otimes h = (f \otimes h) \oplus (g \otimes h)$
proof −
{
fix *i j*
have $((f \oplus g) \otimes h)\ i\ j = \sum\{(f\ i\ k + g\ i\ k) \cdot h\ k\ j \mid k.\ k \in UNIV\}$
by (*simp only*: *mat-mult-def mat-add-def*)
also have ... $= \sum\{f\ i\ k \cdot h\ k\ j + g\ i\ k \cdot h\ k\ j \mid k.\ k \in UNIV\}$
by (*simp only*: *distrib-right*)
also have ... $= \sum\{f\ i\ k \cdot h\ k\ j \mid k.\ k \in UNIV\} + \sum\{g\ i\ k \cdot h\ k\ j \mid k.\ k \in UNIV\}$
by (*simp only*: *fset-to-im sum-fun-sum finite-UNIV*)
finally have $((f \oplus g) \otimes h)\ i\ j = ((f \otimes h) \oplus (g \otimes h))\ i\ j$
by (*simp only*: *mat-mult-def mat-add-def*)
}
thus *?thesis*
by *auto*
qed

lemma *logic-aux1*: $(\exists k.\ (i = k \longrightarrow x = f\ i\ j) \wedge (i \neq k \longrightarrow x = 0)) \longleftrightarrow (\exists k.\ i = k \wedge x = f\ i\ j) \vee (\exists k.\ i \neq k \wedge x = 0)$
by *blast*

lemma *logic-aux2*: $(\exists k.\ (k = j \longrightarrow x = f\ i\ j) \wedge (k \neq j \longrightarrow x = 0)) \longleftrightarrow (\exists k.\ k = j \wedge x = f\ i\ j) \vee (\exists k.\ k \neq j \wedge x = 0)$
  by *blast*

lemma *mat-onel*[*simp*]: $\varepsilon \otimes f = f$
proof −
  {
    fix *i j*
    have $(\varepsilon \otimes f)\ i\ j = \sum \{x.\ (\exists k.\ (i = k \longrightarrow x = f\ i\ j) \wedge (i \neq k \longrightarrow x = 0))\}$
      by (*auto simp add*: *mat-mult-def mat-one-def*)
    also have ... $= \sum (\{x.\ \exists k.\ (i = k \wedge x = f\ i\ j)\} \cup \{x.\ \exists k.\ (i \neq k \wedge x = 0)\})$
      by (*simp only*: *Collect-disj-eq logic-aux1*)
    also have ... $= \sum \{x.\ \exists k.\ (i = k \wedge x = f\ i\ j)\} + \sum \{x.\ \exists k.\ (i \neq k \wedge x = 0)\}$
      by (*rule sum-union*, *auto*)
    finally have $(\varepsilon \otimes f)\ i\ j = f\ i\ j$
      by (*auto simp add*: *sum.neutral*)
  }
  thus *?thesis*
    by *auto*
qed

lemma *mat-oner*[*simp*]: $f \otimes \varepsilon = f$
proof −
  {
    fix *i j*
    have $(f \otimes \varepsilon)\ i\ j = \sum \{x.\ (\exists k.\ (k = j \longrightarrow x = f\ i\ j) \wedge (k \neq j \longrightarrow x = 0))\}$
      by (*auto simp add*: *mat-mult-def mat-one-def*)
    also have ... $= \sum (\{x.\ \exists k.\ (k = j \wedge x = f\ i\ j)\} \cup \{x.\ \exists k.\ (k \neq j \wedge x = 0)\})$
      by (*simp only*: *Collect-disj-eq logic-aux2*)
    also have ... $= \sum \{x.\ \exists k.\ (k = j \wedge x = f\ i\ j)\} + \sum \{x.\ \exists k.\ (k \neq j \wedge x = 0)\}$
      by (*rule sum-union*, *auto*)
    finally have $(f \otimes \varepsilon)\ i\ j = f\ i\ j$
      by (*auto simp add*: *sum.neutral*)
  }
  thus *?thesis*
    by *auto*
qed

lemma *mat-rearrange*:
  fixes $F :: 'a \Rightarrow 'k1 \Rightarrow 'k2 \Rightarrow 'b \Rightarrow 'c$::*dioid-one-zero*
  assumes *fUNk1*: *finite* (*UNIV*::*'k1 set*)
  assumes *fUNk2*: *finite* (*UNIV*::*'k2 set*)
  shows $\sum \{\sum \{F\ i\ k1\ k2\ j\ |k2.\ k2 \in (UNIV::'k2\ set)\}\ |k1.\ k1 \in (UNIV::'k1\ set)\} = \sum \{\sum \{F\ i\ k1\ k2\ j\ |k1.\ k1 \in UNIV\}\ |k2.\ k2 \in UNIV\}$
proof −
  {
    fix $z :: 'c$
    let $?lhs = \sum \{\sum \{F\ i\ k1\ k2\ j\ |k2.\ k2 \in UNIV\}\ |k1.\ k1 \in UNIV\}$

let *?rhs* = $\sum$ {$\sum$ {*F i k1 k2 j* |*k1*. *k1* ∈ *UNIV*} |*k2*. *k2* ∈ *UNIV*}
have *?lhs* ≤ *z* ⟷ (∀ *k1 k2*. *F i k1 k2 j* ≤ *z*)
by (*simp only*: *fset-to-im sum-fun-image-sup fUNk1 fUNk2*, *auto*)
hence *?lhs* ≤ *z* ⟷ *?rhs* ≤ *z*
by (*simp only*: *fset-to-im sum-fun-image-sup fUNk1 fUNk2*, *auto*)
}
thus *?thesis*
by (*simp add*: *eq-iff*)
qed

lemma *mat-mult-assoc*: *f* ⊗ (*g* ⊗ *h*) = (*f* ⊗ *g*) ⊗ *h*
proof −
{
fix *i j*
have (*f* ⊗ (*g* ⊗ *h*)) *i j* = $\sum$ {*f i k1* · $\sum$ {*g k1 k2* · *h k2 j* |*k2*. *k2* ∈ *UNIV*} |*k1*. *k1* ∈ *UNIV*}
by (*simp only*: *mat-mult-def*)
also have ... = $\sum$ {$\sum$ {*f i k1* · *g k1 k2* · *h k2 j* |*k2*. *k2* ∈ *UNIV*} |*k1*. *k1* ∈ *UNIV*}
by (*simp only*: *fset-to-im sum-fun-distl finite-UNIV mult.assoc*)
also have ... = $\sum$ {$\sum$ {(*f i k1* · *g k1 k2*) · *h k2 j*|*k1*. *k1* ∈ *UNIV*} |*k2*. *k2* ∈ *UNIV*}
by (*rule mat-rearrange*, *auto simp add*: *finite-UNIV*)
also have ... = $\sum$ {$\sum$ {*f i k1* · *g k1 k2* |*k1*. *k1* ∈ *UNIV*} · *h k2 j* |*k2*. *k2* ∈ *UNIV*}
by (*simp only*: *fset-to-im sum-fun-distr finite-UNIV*)
finally have (*f* ⊗ (*g* ⊗ *h*)) *i j* = ((*f* ⊗ *g*) ⊗ *h*) *i j*
by (*simp only*: *mat-mult-def*)
}
thus *?thesis*
by *auto*
qed

definition *mat-less-eq* :: ('*a*, '*b*, '*c*::*dioid-one-zero*) *matrix* ⇒ ('*a*, '*b*, '*c*) *matrix* ⇒ *bool* **where**
mat-less-eq f g = (*f* ⊕ *g* = *g*)

definition *mat-less* :: ('*a*, '*b*, '*c*::*dioid-one-zero*) *matrix* ⇒ ('*a*, '*b*, '*c*) *matrix* ⇒ *bool* **where**
mat-less f g = (*mat-less-eq f g* ∧ *f* ≠ *g*)

interpretation *matrix-dioid*: *dioid-one-zero mat-add mat-mult mat-one mat-zero mat-less-eq mat-less*
by (*unfold-locales*) (*metis mat-add-assoc mat-add-comm mat-mult-assoc*[*symmetric*] *mat-distr mat-onel mat-oner mat-zeror mat-annil mat-annir mat-less-eq-def mat-less-def mat-add-idem mat-distl*)+

As in the case of formal power series we currently do not implement the Kleene star of matrices, since this is complicated.

end

References

[1] A. Armstrong, S. Foster, W. Guttmann, G. Struth, and T. Weber. Relation algebra. *Archive of Formal Proofs*, 2014.

[2] A. Armstrong, V. B. F. Gomes, and G. Struth. Kleene algebra with tests. *Archive of Formal Proofs*, 2014.

[3] A. Armstrong and G. Struth. Automated reasoning in higher-order regular algebra. In W. Kahl and T. G. Griffin, editors, *RAMICS 2012*, volume 7560 of *Lecture Notes in Computer Science*, pages 66–81. Springer, 2012.

[4] L. Bachmair and N. Dershowitz. Commutation, transformation, and termination. In J. H. Siekmann, editor, *Conference on Automated Deduction*, volume 230 of *Lecture Notes in Computer Science*, pages 5–20. Springer, 1986.

[5] G. Birkhoff. *Lattice Theory*. American Mathematical Society Colloquium Publications, 1967.

[6] M. Boffa. Une remarque sur les systèmes complets d'identités rationnelles. *Informatique Theéorique et Applications*, 24(4):419–423, 1990.

[7] E. Cohen. Separation and reduction. In R. C. Backhouse and J. N. Oliveira, editors, *MPC*, volume 1837 of *Lecture Notes in Computer Science*, pages 45–59. Springer, 2000.

[8] J. H. Conway. *Regular Algebra and Finite Machines*. Chapman and Hall, 1971.

[9] J. Desharnais, B. Möller, and G. Struth. Kleene algebra with domain. *ACM Trans. Comput. Log.*, 7(4):798–833, 2006.

[10] M. Droste, W. Kuich, and H. Vogler, editors. *Handbook of Weighted Automata*. Springer, 2009.

[11] S. Foster and G. Struth. Automated analysis of regular algebra. In B. Gramlich, D. Miller, and U. Sattler, editors, *IJCAR 2012*, volume 7364 of *Lecture Notes in Computer Science*, pages 271–285. Springer, 2012.

[12] S. Foster and G. Struth. Regular algebras. *Archive of Formal Proofs*, 2014.

[13] S. Foster, G. Struth, and T. Weber. Automated engineering of relational and algebraic methods in Isabelle/HOL – (invited tutorial). In H. C. M. de Swart, editor, *RAMICS*, volume 6663 of *Lecture Notes in Computer Science*, pages 52–67. Springer, 2011.

[14] V. B. F. Gomes, W. Guttmann, P. Höfner, G. Struth, and T. Weber. Kleene algebra with domain. *Archive of Formal Proofs*, 2016.

[15] M. Gondran and M. Minoux. *Graphs, Dioids and Semirings: New Models and Algorithms*, volume 41 of *Operations Research/Computer Science Interfaces*. Springer, 2010.

[16] W. Guttmann, G. Struth, and T. Weber. Automating algebraic methods in Isabelle. In S. Qin and Z. Qiu, editors, *ICFEM 2011*, volume 6991 of *Lecture Notes in Computer Science*, pages 617–632. Springer, 2011.

[17] W. Guttmann, G. Struth, and T. Weber. A repository for Tarski-Kleene algebras. In P. Höfner, A. McIver, and G. Struth, editors, *ATE 2011*, volume 760 of *CEUR Workshop Proceedings*, pages 30–39. CEUR-WS.org, 2011.

[18] D. Haren, D. Kozen, and J. Tiuryn. *Dynamic Logic*. MIT Press, 2000.

[19] P. Höfner and G. Struth. Algebraic notions of nontermination: Omega and divergence in idempotent semirings. *J. Log. Algebr. Program.*, 79(8):794–811, 2010.

[20] D. Kozen. On Kleene algebras and closed semirings. In B. Rovan, editor, *Mathematical Foundations of Computer Science*, volume 452 of *Lecture Notes in Computer Science*, pages 26–47. Springer, 1990.

[21] D. Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Inf. Comput.*, 110(2):366–390, 1994.

[22] D. Kozen. On Hoare logic and Kleene algebra with tests. *ACM Trans. Comput. Log.*, 1(1):60–76, 2000.

[23] A. Krauss and T. Nipkow. Proof pearl: Regular expression equivalence and relation algebra. *J. Autom. Reasoning*, 49(1):95–106, 2012.

[24] A. McIver and T. Weber. Towards automated proof support for probabilistic distributed systems. In G. Sutcliffe and A. Voronkov, editors, *LPAR*, volume 3835 of *Lecture Notes in Computer Science*, pages 534–548, 2005.

[25] G. Pilz. *Near-rings*. North-Holland, Amsterdam, second edition, 1983.

[26] K. Solin. Normal forms in total correctness for while programs and action systems. *J. Logic and Algebraic Programming*, 80(6):362–375, 2011.

[27] G. Struth. Abstract abstract reduction. *J. Log. Algebr. Program.*, 66(2):239–270, 2006.

[28] G. Struth. Left omega algebras and regular equations. *J. Log. Algebr. Program.*, 81(6):705–717, 2012.

[29] Y. Venema. Representation of game algebras. *Studia Logica*, 75(2):239–256, 2003.

[30] J. von Wright. From Kleene algebra to refinement algebra. In E. A. Boiten and B. Möller, editors, *MPC*, volume 2386 of *LNCS*, pages 233–262. Springer, 2002.

[31] J. von Wright. Towards a refinement algebra. *Science of Computer Programming*, 51(1–2):23–45, 2004.

[32] K. W. Wagner. Eine topologische Charakterisierung einiger Klassen regulärer Folgenmengen. *Elektronische Informationsverarbeitung und Kybernetik*, 13(9):473–487, 1977.