Kleene Algebras with Domain

Victor B. F. Gomes, Walter Guttmann, Peter Höfner,
Georg Struth and Tjark Weber

April 12, 2016

Abstract

Kleene algebras with domain are Kleene algebras endowed with an operation that maps each element of the algebra to its domain of definition (or its complement) in abstract fashion. They form a simple algebraic basis for Hoare logics, dynamic logics or predicate transformer semantics. We formalise a modular hierarchy of algebras with domain and antidomain (domain complement) operations in Isabelle/HOL that ranges from domain and antidomain semigroups to modal Kleene algebras and divergence Kleene algebras. We link these algebras with models of binary relations and program traces. We include some examples from modal logics, termination and program analysis.

Contents

1 Introductory Remarks

These theory files are intended as a reference formalisation for variants of Kleene algebras with domain. The algebraic hierarchy is developed in a modular way from domain and antidomain semigroups to modal Kleene algebras in which forward and backward box and diamond operators interact via conjugations and Galois connections. Throughout the development we have aimed at readable proofs so that these theories can be seen as a machine-checked introduction to reasoning in this setting. Apart from that, the Isabelle code is only sparsely annotated, and we refer to a series of articles for further information.

Our formalisation follows the approaches of Desharnais, Jipsen and Struth to domain semigroups [3] and Desharnais and Struth to families of domain semirings and Kleene algebras with domain [7, 6]. The link with modal Kleene algebras, Hoare logics and predicate transformers has been elaborated by Möller and Struth [13]; a notion of divergence has been added by Desharnais, Möller and Struth [5]. A previous stage of this formalisation has been documented in a companion article [11].

The target model of these axiomatisations are binary relations, where the domain operation represents the set of those elements that are related to some other element. There is a vast amount of literature on axiomatising the domain of functions, especially in semigroup theory. The deterministic nature of functions, however, leads to different axiom sets. An integration of these approaches is left for future work.

Our Isabelle/HOL formalisation itself is based on a formalisation of variants of Kleene algebras [1]. An adaptation of Kleene algebras with domain to the setting of concurrent dynamic algebra [10] can also be found in the Archive of Formal Proofs [9]. A formalisation of the original two-sorted approach to Kleene algebra with domain [4] is left for future work as well.

2 Domain Semirings

```
theory Domain-Semiring
imports ../Kleene-Algebra/Kleene-Algebra

begin
```

2.1 Domain Semigroups and Domain Monoids

```
class domain-op =
  fixes domain-op :: 'a ⇒ 'a (d)
```

First we define the class of domain semigroups. Axioms are taken from [3].

```
class domain-semigroup = semigroup-mult + domain-op +
  assumes dsg1 [simp]: d x · x = x
  and dsg2 [simp]: d (x · d y) = d (x · y)
  and dsg3 [simp]: d (d x · y) = d x · d y
  and dsg4: d x · d y = d y · d x

begin

lemma domain-invol [simp]: d (d x) = d x
proof −
  have d (d x) = d (d (d x · x))
    by simp
  also have ... = d (d x · d x)
    using dsg3 by presburger
  also have ... = d (d x · x)
    by simp
  finally show ?thesis
    by simp
qed
```

The next lemmas show that domain elements form semilattices.

```
lemma dom-el-idem [simp]: d x · d x = d x
proof −
  have d x · d x = d (d x · x)
    using dsg3 by presburger
  thus ?thesis
    by simp
qed
```

lemma *dom-mult-closed* [*simp*]: $d\ (d\ x \cdot d\ y) = d\ x \cdot d\ y$
  by *simp*

lemma *dom-lc3* [*simp*]: $d\ x \cdot d\ (x \cdot y) = d\ (x \cdot y)$
proof −
  have $d\ x \cdot d\ (x \cdot y) = d\ (d\ x \cdot x \cdot y)$
    using *dsg3 mult-assoc* **by** *presburger*
  thus *?thesis*
    by *simp*
qed

lemma *d-fixpoint*: $(\exists\, y.\ x = d\ y) \longleftrightarrow x = d\ x$
  by *auto*

lemma *d-type*: $\forall\, P.\ (\forall\, x.\ x = d\ x \longrightarrow P\ x) \longleftrightarrow (\forall\, x.\ P\ (d\ x))$
  by (*metis domain-invol*)

We define the semilattice ordering on domain semigroups and explore the semilattice of domain elements from the order point of view.

definition *ds-ord* :: $'a \Rightarrow\ 'a \Rightarrow$ *bool* (**infix** $\sqsubseteq$ *50*) **where**
  $x \sqsubseteq y \longleftrightarrow x = d\ x \cdot y$

lemma *ds-ord-refl*: $x \sqsubseteq x$
  by (*simp add*: *ds-ord-def*)

lemma *ds-ord-trans*: $x \sqsubseteq y \Longrightarrow y \sqsubseteq z \Longrightarrow x \sqsubseteq z$
proof −
  assume $x \sqsubseteq y$ **and** *a*: $y \sqsubseteq z$
  hence *b*: $x = d\ x \cdot y$
    using *ds-ord-def* **by** *blast*
  hence $x = d\ x \cdot d\ y \cdot z$
    using *a ds-ord-def mult-assoc* **by** *force*
  also have ... $= d\ (d\ x \cdot y) \cdot z$
    by *simp*
  also have ... $= d\ x \cdot z$
    using *b* **by** *auto*
  finally show *?thesis*
    using *ds-ord-def* **by** *blast*
qed

lemma *ds-ord-antisym*: $x \sqsubseteq y \Longrightarrow y \sqsubseteq x \Longrightarrow x = y$
proof −
  assume *a*: $x \sqsubseteq y$ **and** $y \sqsubseteq x$
  hence *b*: $y = d\ y \cdot x$
    using *ds-ord-def* **by** *auto*
  have $x = d\ x \cdot d\ y \cdot x$
    using *a b ds-ord-def mult-assoc* **by** *force*
  also have ... $= d\ y \cdot x$
    by (*metis* (*full-types*) *b dsg3 dsg4*)

thus *?thesis*
  using *b calculation* **by** *presburger*
qed

This relation is indeed an order.

sublocale *ds*: *order op* $\sqsubseteq$ $\lambda x\ y.\ (x \sqsubseteq y \wedge x \neq y)$
proof
 show $\bigwedge x\ y.\ (x \sqsubseteq y \wedge x \neq y) = (x \sqsubseteq y \wedge \neg\ y \sqsubseteq x)$
  using *ds-ord-antisym* **by** *blast*
 show $\bigwedge x.\ x \sqsubseteq x$
  by (*rule ds-ord-refl*)
 show $\bigwedge x\ y\ z.\ x \sqsubseteq y \Longrightarrow y \sqsubseteq z \Longrightarrow x \sqsubseteq z$
  by (*rule ds-ord-trans*)
 show $\bigwedge x\ y.\ x \sqsubseteq y \Longrightarrow y \sqsubseteq x \Longrightarrow x = y$
  by (*rule ds-ord-antisym*)
qed

lemma *ds-ord-eq*: $x \sqsubseteq d\ x \longleftrightarrow x = d\ x$
 by (*simp add*: *ds-ord-def*)

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

oops

lemma *ds-ord-iso-right*: $x \sqsubseteq y \Longrightarrow x \cdot z \sqsubseteq y \cdot z$
proof −
 assume $x \sqsubseteq y$
 hence *a*: $x = d\ x \cdot y$
  by (*simp add*: *ds-ord-def*)
 hence $x \cdot z = d\ x \cdot y \cdot z$
  by *auto*
 also have ... $= d\ (d\ x \cdot y \cdot z) \cdot d\ x \cdot y \cdot z$
  using *dsg1 mult-assoc* **by** *presburger*
 also have ... $= d\ (x \cdot z) \cdot d\ x \cdot y \cdot z$
  using *a* **by** *presburger*
 finally show *?thesis*
  using *ds-ord-def dsg4 mult-assoc* **by** *auto*
qed

The order on domain elements could as well be defined based on multiplication/meet.

lemma *ds-ord-sl-ord*: $d\ x \sqsubseteq d\ y \longleftrightarrow d\ x \cdot d\ y = d\ x$
 using *ds-ord-def* **by** *auto*

lemma *ds-ord-1*: $d\ (x \cdot y) \sqsubseteq d\ x$
 by (*simp add*: *ds-ord-sl-ord dsg4*)

lemma *ds-subid-aux*: $d\ x \cdot y \sqsubseteq y$
 by (*simp add*: *ds-ord-def mult-assoc*)

lemma $y \cdot d\ x \sqsubseteq y$

oops

lemma *ds-dom-iso*: $x \sqsubseteq y \Longrightarrow d\ x \sqsubseteq d\ y$
proof −
  assume $x \sqsubseteq y$
  hence $x = d\ x \cdot y$
    by (*simp add*: *ds-ord-def*)
  hence $d\ x = d\ (d\ x \cdot y)$
    by *presburger*
  also have ... $= d\ x \cdot d\ y$
    by *simp*
  finally show *?thesis*
    using *ds-ord-sl-ord* **by** *auto*
qed

lemma *ds-dom-llp*: $x \sqsubseteq d\ y \cdot x \longleftrightarrow d\ x \sqsubseteq d\ y$
proof
  assume $x \sqsubseteq d\ y \cdot x$
  hence $x = d\ y \cdot x$
    by (*simp add*: *ds-subid-aux ds.order.antisym*)
  hence $d\ x = d\ (d\ y \cdot x)$
    by *presburger*
  thus $d\ x \sqsubseteq d\ y$
    using *ds-ord-sl-ord dsg4* **by** *force*
next
  assume $d\ x \sqsubseteq d\ y$
  thus $x \sqsubseteq d\ y \cdot x$
    by (*metis* (*no-types*) *ds-ord-iso-right dsg1*)
qed

lemma *ds-dom-llp-strong*: $x = d\ y \cdot x \longleftrightarrow d\ x \sqsubseteq d\ y$
  by (*simp add*: *ds-dom-llp ds.eq-iff ds-subid-aux*)

definition *refines* :: $'a \Rightarrow 'a \Rightarrow bool$
  where *refines* $x\ y \equiv d\ y \sqsubseteq d\ x \wedge (d\ y) \cdot x \sqsubseteq y$

lemma *refines-refl*: *refines* $x\ x$
  using *refines-def* **by** *simp*

lemma *refines-trans*: *refines* $x\ y \Longrightarrow$ *refines* $y\ z \Longrightarrow$ *refines* $x\ z$
  unfolding *refines-def*
  by (*metis domain-invol ds.dual-order.trans dsg1 dsg3 ds-ord-def*)

lemma *refines-antisym*: *refines* $x\ y \Longrightarrow$ *refines* $y\ x \Longrightarrow x = y$
  unfolding *refines-def*
  using *ds-dom-llp ds-ord-antisym* **by** *fastforce*

sublocale *ref*: *order refines* $\lambda x\ y.$ (*refines* $x\ y \wedge x \neq y$)
proof
  show $\bigwedge x\ y.$ (*refines* $x\ y \wedge x \neq y$) = (*refines* $x\ y \wedge \neg$ *refines* $y\ x$)
    using *refines-antisym* **by** *blast*
  show $\bigwedge x.$ *refines* $x\ x$
    by (*rule refines-refl*)
  show $\bigwedge x\ y\ z.$ *refines* $x\ y \Longrightarrow$ *refines* $y\ z \Longrightarrow$ *refines* $x\ z$
    by (*rule refines-trans*)
  show $\bigwedge x\ y.$ *refines* $x\ y \Longrightarrow$ *refines* $y\ x \Longrightarrow x = y$
    by (*rule refines-antisym*)
qed

end

We expand domain semigroups to domain monoids.

class *domain-monoid* = *monoid-mult* + *domain-semigroup*
begin

lemma *dom-one* [*simp*]: $d\ 1 = 1$
proof −
  have $1 = d\ 1 \cdot 1$
    using *dsg1* **by** *presburger*
  thus *?thesis*
    by *simp*
qed

lemma *ds-subid-eq*: $x \sqsubseteq 1 \longleftrightarrow x = d\ x$
  by (*simp add*: *ds-ord-def*)

end

2.2 Domain Near-Semirings

The axioms for domain near-semirings are taken from [6].

class *domain-near-semiring* = *ab-near-semiring* + *plus-ord* + *domain-op* +
  assumes *dns1* [*simp*]: $d\ x \cdot x = x$
  and *dns2* [*simp*]: $d\ (x \cdot d\ y) = d(x \cdot y)$
  and *dns3* [*simp*]: $d\ (x + y) = d\ x + d\ y$
  and *dns4*: $d\ x \cdot d\ y = d\ y \cdot d\ x$
  and *dns5* [*simp*]: $d\ x \cdot (d\ x + d\ y) = d\ x$

begin

Domain near-semirings are automatically dioids; addition is idempotent.

subclass *near-dioid*
proof
  show $\bigwedge x.\ x + x = x$

proof –
fix *x*
have *a*: $d\ x = d\ x \cdot d\ (x + x)$
using *dns3 dns5* **by** *presburger*
have $d\ (x + x) = d\ (x + x + (x + x)) \cdot d\ (x + x)$
by (*metis* (*no-types*) *dns3 dns4 dns5*)
hence $d\ (x + x) = d\ (x + x) + d\ (x + x)$
by *simp*
thus $x + x = x$
by (*metis a dns1 dns4 distrib-right′*)
qed
qed

Next we prepare to show that domain near-semirings are domain semigroups.

lemma *dom-iso*: $x \leq y \Longrightarrow d\ x \leq d\ y$
using *order-prop* **by** *auto*

lemma *dom-add-closed* [*simp*]: $d\ (d\ x + d\ y) = d\ x + d\ y$
proof –
have $d\ (d\ x + d\ y) = d\ (d\ x) + d\ (d\ y)$
by *simp*
thus *?thesis*
by (*metis dns1 dns2 dns3 dns4*)
qed

lemma *dom-absorp-2* [*simp*]: $d\ x + d\ x \cdot d\ y = d\ x$
proof –
have $d\ x + d\ x \cdot d\ y = d\ x \cdot d\ x + d\ x \cdot d\ y$
by (*metis add-idem′ dns5*)
also have ... $= (d\ x + d\ y) \cdot d\ x$
by (*simp add*: *dns4*)
also have ... $= d\ x \cdot (d\ x + d\ y)$
by (*metis dom-add-closed dns4*)
finally show *?thesis*
by *simp*
qed

lemma *dom-1*: $d\ (x \cdot y) \leq d\ x$
proof –
have $d\ (x \cdot y) = d\ (d\ x \cdot d\ (x \cdot y))$
by (*metis dns1 dns2 mult-assoc*)
also have ... $\leq d\ (d\ x) + d\ (d\ x \cdot d\ (x \cdot y))$
by *simp*
also have ... $= d\ (d\ x + d\ x \cdot d\ (x \cdot y))$
using *dns3* **by** *presburger*
also have ... $= d\ (d\ x)$
by *simp*
finally show *?thesis*
by (*metis dom-add-closed add-idem′*)

qed

lemma *dom-subid-aux2*: $d\ x \cdot y \leq y$
proof $-$
  have $d\ x \cdot y \leq d\ (x + d\ y) \cdot y$
    by (*simp add*: *mult-isor*)
  also have ... $= (d\ x + d\ (d\ y)) \cdot d\ y \cdot y$
    using *dns1 dns3 mult-assoc* **by** *presburger*
  also have ... $= (d\ y + d\ y \cdot d\ x) \cdot y$
    by (*simp add*: *dns4 add-commute*)
  finally show *?thesis*
    by *simp*
qed

lemma *dom-glb*: $d\ x \leq d\ y \Longrightarrow d\ x \leq d\ z \Longrightarrow d\ x \leq d\ y \cdot d\ z$
  by (*metis dns5 less-eq-def mult-isor*)

lemma *dom-glb-eq*: $d\ x \leq d\ y \cdot d\ z \longleftrightarrow d\ x \leq d\ y \wedge d\ x \leq d\ z$
proof $-$
  have $d\ x \leq d\ z \longrightarrow d\ x \leq d\ z$
    by *meson*
  then show *?thesis*
   by (*metis* (*no-types*) *dom-absorp-2 dom-glb dom-subid-aux2 local.dual-order.trans local.join.sup.coboundedI2*)
qed

lemma *dom-ord*: $d\ x \leq d\ y \longleftrightarrow d\ x \cdot d\ y = d\ x$
proof
  assume $d\ x \leq d\ y$
  hence $d\ x + d\ y = d\ y$
    by (*simp add*: *less-eq-def*)
  thus $d\ x \cdot d\ y = d\ x$
    by (*metis dns5*)
next
  assume $d\ x \cdot d\ y = d\ x$
  thus $d\ x \leq d\ y$
    by (*metis dom-subid-aux2*)
qed

lemma *dom-export* [*simp*]: $d\ (d\ x \cdot y) = d\ x \cdot d\ y$
proof (*rule antisym*)
  have $d\ (d\ x \cdot y) = d\ (d\ (d\ x \cdot y)) \cdot d\ (d\ x \cdot y)$
    using *dns1* **by** *presburger*
  also have ... $= d\ (d\ x \cdot d\ y) \cdot d\ (d\ x \cdot y)$
    by (*metis dns1 dns2 mult-assoc*)
  finally show *a*: $d\ (d\ x \cdot y) \leq d\ x \cdot d\ y$
    by (*metis* (*no-types*) *dom-add-closed dom-glb dom-1 add-idem′ dns2 dns4*)
  have $d\ (d\ x \cdot y) = d\ (d\ x \cdot y) \cdot d\ x$
    using *a dom-glb-eq dom-ord* **by** *force*

hence $d\ x \cdot d\ y = d\ (d\ x \cdot y) \cdot d\ y$
by (*metis dns1 dns2 mult-assoc*)
thus $d\ x \cdot d\ y \leq d\ (d\ x \cdot y)$
using *a dom-glb-eq dom-ord* **by** *auto*
qed

subclass *domain-semigroup*
by (*unfold-locales*, *auto simp*: *dns4*)

We compare the domain semigroup ordering with that of the dioid.

lemma *d-two-orders*: $d\ x \sqsubseteq d\ y \longleftrightarrow d\ x \leq d\ y$
by (*simp add*: *dom-ord ds-ord-sl-ord*)

lemma *two-orders*: $x \sqsubseteq y \Longrightarrow x \leq y$
by (*metis dom-subid-aux2 ds-ord-def*)

lemma $x \leq y \Longrightarrow x \sqsubseteq y$

oops

Next we prove additional properties.

lemma *dom-subdist*: $d\ x \leq d\ (x + y)$
by *simp*

lemma *dom-distrib*: $d\ x + d\ y \cdot d\ z = (d\ x + d\ y) \cdot (d\ x + d\ z)$
proof −
have $(d\ x + d\ y) \cdot (d\ x + d\ z) = d\ x \cdot (d\ x + d\ z) + d\ y \cdot (d\ x + d\ z)$
using *distrib-right′* **by** *blast*
also have ... $= d\ x + (d\ x + d\ z) \cdot d\ y$
by (*metis* (*no-types*) *dns3 dns5 dsg4*)
also have ... $= d\ x + d\ x \cdot d\ y + d\ z \cdot d\ y$
using *add-assoc′ distrib-right′* **by** *presburger*
finally show *?thesis*
by (*simp add*: *dsg4*)
qed

lemma *dom-llp1*: $x \leq d\ y \cdot x \Longrightarrow d\ x \leq d\ y$
proof −
assume $x \leq d\ y \cdot x$
hence $d\ x \leq d\ (d\ y \cdot x)$
using *dom-iso* **by** *blast*
also have ... $= d\ y \cdot d\ x$
by *simp*
finally show $d\ x \leq d\ y$
by (*simp add*: *dom-glb-eq*)
qed

lemma *dom-llp2*: $d\ x \leq d\ y \Longrightarrow x \leq d\ y \cdot x$
using *d-two-orders local.ds-dom-llp two-orders* **by** *blast*

```
lemma dom-llp: x ≤ d y · x ⟷ d x ≤ d y
  using dom-llp1 dom-llp2 by blast

end
```

We expand domain near-semirings by an additive unit, using slightly different axioms.

```
class domain-near-semiring-one = ab-near-semiring-one + plus-ord + domain-op
+
  assumes dnso1 [simp]: x + d x · x = d x · x
  and dnso2 [simp]: d (x · d y) = d (x · y)
  and dnso3 [simp]: d x + 1 = 1
  and dnso4 [simp]: d (x + y) = d x + d y
  and dnso5: d x · d y = d y · d x

begin
```

The previous axioms are derivable.

```
subclass domain-near-semiring
proof
  show a: ⋀x. d x · x = x
   by (metis add-commute local.dnso3 local.distrib-right' local.dnso1 local.mult-onel)
  show ⋀x y. d (x · d y) = d (x · y)
    by simp
  show ⋀x y. d (x + y) = d x + d y
    by simp
  show ⋀x y. d x · d y = d y · d x
    by (simp add: dnso5)
  show ⋀x y. d x · (d x + d y) = d x
  proof −
    fix x y
    have ⋀x. 1 + d x = 1
      using add-commute dnso3 by presburger
    thus d x · (d x + d y) = d x
      by (metis (no-types) a dnso2 dnso4 dnso5 distrib-right' mult-onel)
  qed
qed

subclass domain-monoid ..

lemma dom-subid: d x ≤ 1
  by (simp add: less-eq-def)

end
```

We add a left unit of multiplication.

```
class domain-near-semiring-one-zerol = ab-near-semiring-one-zerol + domain-near-semiring-one
+
```

assumes *dnso6* [*simp*]: $d\ 0 = 0$

begin

lemma *domain-very-strict*: $d\ x = 0 \longleftrightarrow x = 0$
by (*metis annil dns1 dnso6*)

lemma *dom-weakly-local*: $x \cdot y = 0 \longleftrightarrow x \cdot d\ y = 0$
proof −
have $x \cdot y = 0 \longleftrightarrow d\ (x \cdot y) = 0$
by (*simp add*: *domain-very-strict*)
also have ... $\longleftrightarrow d\ (x \cdot d\ y) = 0$
by *simp*
finally show *?thesis*
using *domain-very-strict* **by** *blast*
qed

end

2.3 Domain Pre-Dioids

Pre-semirings with one and a left zero are automatically dioids. Hence there is no point defining domain pre-semirings separately from domain dioids. The axioms are once again from [6].

class *domain-pre-dioid-one* = *pre-dioid-one* + *domain-op* +
assumes *dpd1* : $x \leq d\ x \cdot x$
and *dpd2* [*simp*]: $d\ (x \cdot d\ y) = d\ (x \cdot y)$
and *dpd3* [*simp*]: $d\ x \leq 1$
and *dpd4* [*simp*]: $d\ (x + y) = d\ x + d\ y$

begin

We prepare to show that every domain pre-dioid with one is a domain near-dioid with one.

lemma *dns1''* [*simp*]: $d\ x \cdot x = x$
proof (*rule antisym*)
show $d\ x \cdot x \leq x$
using *dpd3 mult-isor* **by** *fastforce*
show $x \leq d\ x \cdot x$
by (*simp add*: *dpd1*)
qed

lemma *d-iso*: $x \leq y \Longrightarrow d\ x \leq d\ y$
by (*metis dpd4 less-eq-def*)

lemma *domain-1''*: $d\ (x \cdot y) \leq d\ x$
proof −
have $d\ (x \cdot y) = d\ (x \cdot d\ y)$

by *simp*
also have ... $\leq$ *d* (*x* · *1*)
by (*meson d-iso dpd3 mult-isol*)
finally show *?thesis*
by *simp*
qed

lemma *domain-export''* [*simp*]: *d* (*d x* · *y*) = *d x* · *d y*
proof (*rule antisym*)
have *one*: *d* (*d x* · *y*) $\leq$ *d x*
by (*metis dpd2 domain-1'' mult-onel*)
have *two*: *d* (*d x* · *y*) $\leq$ *d y*
using *d-iso dpd3 mult-isor* **by** *fastforce*
have *d* (*d x* · *y*) = *d* (*d* (*d x* · *y*)) · *d* (*d x* · *y*)
by *simp*
also have ... = *d* (*d x* · *y*) · *d* (*d x* · *y*)
by (*metis dns1'' dpd2 mult-assoc*)
thus *d* (*d x* · *y*) $\leq$ *d x* · *d y*
using *mult-isol-var one two* **by** *force*
next
have *d x* · *d y* $\leq$ *1*
by (*metis dpd3 mult-1-right mult-isol order.trans*)
thus *d x* · *d y* $\leq$ *d* (*d x* · *y*)
by (*metis dns1'' dpd2 mult-isol mult-oner*)
qed

lemma *dom-subid-aux1''*: *d x* · *y* $\leq$ *y*
proof −
have *d x* · *y* $\leq$ *1* · *y*
using *dpd3 mult-isor* **by** *blast*
thus *?thesis*
by *simp*
qed

lemma *dom-subid-aux2''*: *x* · *d y* $\leq$ *x*
using *dpd3 mult-isol* **by** *fastforce*

lemma *d-comm*: *d x* · *d y* = *d y* · *d x*
proof (*rule antisym*)
have *d x* · *d y* = (*d x* · *d y*) · (*d x* · *d y*)
by (*metis dns1'' domain-export''*)
thus *d x* · *d y* $\leq$ *d y* · *d x*
by (*metis dom-subid-aux1'' dom-subid-aux2'' mult-isol-var*)
next
have *d y* · *d x* = (*d y* · *d x*) · (*d y* · *d x*)
by (*metis dns1'' domain-export''*)
thus *d y* · *d x* $\leq$ *d x* · *d y*
by (*metis dom-subid-aux1'' dom-subid-aux2'' mult-isol-var*)
qed

subclass *domain-near-semiring-one*
by (*unfold-locales*, *auto simp*: *d-comm local.join.sup.absorb2*)

lemma *domain-subid*: $x \leq 1 \Longrightarrow x \leq d\ x$
by (*metis dns1 mult-isol mult-oner*)

lemma *d-preserves-equation*: $d\ y \cdot x \leq x \cdot d\ z \longleftrightarrow d\ y \cdot x = d\ y \cdot x \cdot d\ z$
by (*metis dom-subid-aux2'' local.antisym local.dom-el-idem local.dom-subid-aux2 local.order-prop local.subdistl mult-assoc*)

lemma *d-restrict-iff*: $(x \leq y) \longleftrightarrow (x \leq d\ x \cdot y)$
by (*metis dom-subid-aux2 dsg1 less-eq-def order-trans subdistl*)

lemma *d-restrict-iff-1*: $(d\ x \cdot y \leq z) \longleftrightarrow (d\ x \cdot y \leq d\ x \cdot z)$
by (*metis dom-subid-aux2 domain-1'' domain-invol dsg1 mult-isol-var order-trans*)

end

We add once more a left unit of multiplication.

class *domain-pre-dioid-one-zerol* = *domain-pre-dioid-one* + *pre-dioid-one-zerol* +
assumes *dpd5* [*simp*]: $d\ 0 = 0$

begin

subclass *domain-near-semiring-one-zerol*
by (*unfold-locales*, *simp*)

end

2.4 Domain Semirings

We do not consider domain semirings without units separately at the moment. The axioms are taken from from [7]

class *domain-semiringl* = *semiring-one-zerol* + *plus-ord* + *domain-op* +
assumes *dsr1* [*simp*]: $x + d\ x \cdot x = d\ x \cdot x$
and *dsr2* [*simp*]: $d\ (x \cdot d\ y) = d\ (x \cdot y)$
and *dsr3* [*simp*]: $d\ x + 1 = 1$
and *dsr4* [*simp*]: $d\ 0 = 0$
and *dsr5* [*simp*]: $d\ (x + y) = d\ x + d\ y$

begin

Every domain semiring is automatically a domain pre-dioid with one and left zero.

subclass *dioid-one-zerol*
by (*standard*, *metis add-commute dsr1 dsr3 distrib-left mult-oner*)

subclass *domain-pre-dioid-one-zerol*
  by (*standard*, *auto simp*: *less-eq-def*)

end

class *domain-semiring* = *domain-semiringl* + *semiring-one-zero*

2.5 The Algebra of Domain Elements

We show that the domain elements of a domain semiring form a distributive lattice. Unfortunately we cannot prove this within the type class of domain semirings.

typedef (**overloaded**) *'a d-element* = $\{x :: 'a :: domain\text{-}semiring.\ x = d\ x\}$
  by (*rule-tac x* = *1* **in** *exI*, *simp add*: *domain-subid order-class.eq-iff*)

setup-lifting *type-definition-d-element*

instantiation *d-element* :: (*domain-semiring*) *bounded-lattice*

begin

lift-definition *less-eq-d-element* :: *'a d-element* ⇒ *'a d-element* ⇒ *bool* **is** *op* ≤ **.**

lift-definition *less-d-element* :: *'a d-element* ⇒ *'a d-element* ⇒ *bool* **is** *op* < **.**

lift-definition *bot-d-element* :: *'a d-element* **is** *0*
  by *simp*

lift-definition *top-d-element* :: *'a d-element* **is** *1*
  by *simp*

lift-definition *inf-d-element* :: *'a d-element* ⇒ *'a d-element* ⇒ *'a d-element* **is** *op* ·
  by (*metis dsg3*)

lift-definition *sup-d-element* :: *'a d-element* ⇒ *'a d-element* ⇒ *'a d-element* **is** *op* +
  by *simp*

instance
  apply (*standard*; *transfer*)
  apply (*simp add*: *less-le-not-le*)+
  apply (*metis dom-subid-aux2''*)
  apply (*metis dom-subid-aux2*)
  apply (*metis dom-glb*)
  apply *simp*+
  by (*metis dom-subid*)

end

instance *d-element* :: (*domain-semiring*) *distrib-lattice*
by (*standard*, *transfer*, *metis dom-distrib*)

2.6 Domain Semirings with a Greatest Element

If there is a greatest element in the semiring, then we have another equality.

class *domain-semiring-top* = *domain-semiring* + *order-top*

begin

notation *top* (⊤)

lemma *kat-equivalence-greatest*: $d\ x \leq d\ y \longleftrightarrow x \leq d\ y \cdot \top$
proof
assume $d\ x \leq d\ y$
thus $x \leq d\ y \cdot \top$
by (*metis dsg1 mult-isol-var top-greatest*)
next
assume $x \leq d\ y \cdot \top$
thus $d\ x \leq d\ y$
using *dom-glb-eq dom-iso* **by** *fastforce*
qed

end

2.7 Forward Diamond Operators

context *domain-semiringl*

begin

We define a forward diamond operator over a domain semiring. A more modular consideration is not given at the moment.

definition *fd* :: *'a* ⇒ *'a* ⇒ *'a* ((|-⟩ -) [*61*,*81*] *82*) **where**
$|x\rangle\ y = d\ (x \cdot y)$

lemma *fdia-d-simp* [*simp*]: $|x\rangle\ d\ y = |x\rangle\ y$
by (*simp add*: *fd-def*)

lemma *fdia-dom* [*simp*]: $|x\rangle\ 1 = d\ x$
by (*simp add*: *fd-def*)

lemma *fdia-add1*: $|x\rangle\ (y + z) = |x\rangle\ y + |x\rangle\ z$
by (*simp add*: *fd-def distrib-left*)

lemma *fdia-add2*: $|x + y\rangle\ z = |x\rangle\ z + |y\rangle\ z$
by (*simp add*: *fd-def distrib-right*)

lemma *fdia-mult*: $|x \cdot y\rangle\ z = |x\rangle\ |y\rangle\ z$
  by (*simp add*: *fd-def mult-assoc*)

lemma *fdia-one* [*simp*]: $|1\rangle\ x = d\ x$
  by (*simp add*: *fd-def*)

lemma *fdemodalisation1*: $d\ z \cdot |x\rangle\ y = 0 \longleftrightarrow d\ z \cdot x \cdot d\ y = 0$
proof −
  have $d\ z \cdot |x\rangle\ y = 0 \longleftrightarrow d\ z \cdot d\ (x \cdot y) = 0$
    by (*simp add*: *fd-def*)
  also have ... $\longleftrightarrow d\ z \cdot x \cdot y = 0$
    by (*metis annil dnso6 dsg1 dsg3 mult-assoc*)
  finally show *?thesis*
    using *dom-weakly-local* **by** *auto*
qed

lemma *fdemodalisation2*: $|x\rangle\ y \leq d\ z \longleftrightarrow x \cdot d\ y \leq d\ z \cdot x$
proof
  assume $|x\rangle\ y \leq d\ z$
  hence *a*: $d\ (x \cdot d\ y) \leq d\ z$
    by (*simp add*: *fd-def*)
  have $x \cdot d\ y = d\ (x \cdot d\ y) \cdot x \cdot d\ y$
    using *dsg1 mult-assoc* **by** *presburger*
  also have ... $\leq d\ z \cdot x \cdot d\ y$
    using *a calculation dom-llp2 mult-assoc* **by** *auto*
  finally show $x \cdot d\ y \leq d\ z \cdot x$
    using *dom-subid-aux2''* *order-trans* **by** *blast*
next
  assume $x \cdot d\ y \leq d\ z \cdot x$
  hence $d\ (x \cdot d\ y) \leq d\ (d\ z \cdot d\ x)$
    using *dom-iso* **by** *fastforce*
  also have ... $\leq d\ (d\ z)$
    using *domain-1''* **by** *blast*
  finally show $|x\rangle\ y \leq d\ z$
    by (*simp add*: *fd-def*)
qed

lemma *fd-iso1*: $d\ x \leq d\ y \Longrightarrow |z\rangle\ x \leq |z\rangle\ y$
  using *fd-def local.dom-iso local.mult-isol* **by** *fastforce*

lemma *fd-iso2*: $x \leq y \Longrightarrow |x\rangle\ z \leq |y\rangle\ z$
  by (*simp add*: *fd-def dom-iso mult-isor*)

lemma *fd-zero-var* [*simp*]: $|0\rangle\ x = 0$
  by (*simp add*: *fd-def*)

lemma *fd-subdist-1*: $|x\rangle\ y \leq |x\rangle\ (y + z)$
  by (*simp add*: *fd-iso1*)

```
lemma fd-subdist-2: |x⟩ (d y · d z) ≤ |x⟩ y
  by (simp add: fd-iso1 dom-subid-aux2'')

lemma fd-subdist: |x⟩ (d y · d z) ≤ |x⟩ y · |x⟩ z
  using fd-def fd-iso1 fd-subdist-2 dom-glb dom-subid-aux2 by auto

lemma fdia-export-1: d y · |x⟩ z = |d y · x⟩ z
  by (simp add: fd-def mult-assoc)

end

context domain-semiring

begin

lemma fdia-zero [simp]: |x⟩ 0 = 0
  by (simp add: fd-def)

end
```

2.8 Domain Kleene Algebras

We add the Kleene star to our considerations. Special domain axioms are not needed.

```
class domain-left-kleene-algebra = left-kleene-algebra-zerol + domain-semiringl

begin

lemma dom-star [simp]: d (x⋆) = 1
proof –
  have d (x⋆) = d (1 + x · x⋆)
    by simp
  also have ... = d 1 + d (x · x⋆)
    using dns3 by blast
  finally show ?thesis
    using add-commute local.dsr3 by auto
qed

lemma fdia-star-unfold [simp]: |1⟩ y + |x⟩ |x⋆⟩ y = |x⋆⟩ y
proof –
  have |1⟩ y + |x⟩ |x⋆⟩ y = |1 + x · x⋆⟩ y
    using local.fdia-add2 local.fdia-mult by presburger
  thus ?thesis
    by simp
qed

lemma fdia-star-unfoldr [simp]: |1⟩ y + |x⋆⟩ |x⟩ y = |x⋆⟩ y
proof –
  have |1⟩ y + |x⋆⟩ |x⟩ y = |1 + x⋆ · x⟩ y
```

using *fdia-add2 fdia-mult* **by** *presburger*
thus *?thesis*
by *simp*
qed

lemma *fdia-star-unfold-var* [*simp*]: $d\ y + |x\rangle\ |x^\star\rangle\ y = |x^\star\rangle\ y$
proof –
have $d\ y + |x\rangle\ |x^\star\rangle\ y = |1\rangle\ y + |x\rangle\ |x^\star\rangle\ y$
by *simp*
also have ... $= |1 + x \cdot x^\star\rangle\ y$
using *fdia-add2 fdia-mult* **by** *presburger*
finally show *?thesis*
by *simp*
qed

lemma *fdia-star-unfoldr-var* [*simp*]: $d\ y + |x^\star\rangle\ |x\rangle\ y = |x^\star\rangle\ y$
proof –
have $d\ y + |x^\star\rangle\ |x\rangle\ y = |1\rangle\ y + |x^\star\rangle\ |x\rangle\ y$
by *simp*
also have ... $= |1 + x^\star \cdot x\rangle\ y$
using *fdia-add2 fdia-mult* **by** *presburger*
finally show *?thesis*
by *simp*
qed

lemma *fdia-star-induct-var*: $|x\rangle\ y \leq d\ y \Longrightarrow |x^\star\rangle\ y \leq d\ y$
proof –
assume *a1*: $|x\rangle\ y \leq d\ y$
hence $x \cdot d\ y \leq d\ y \cdot x$
by (*simp add*: *fdemodalisation2*)
hence $x^\star \cdot d\ y \leq d\ y \cdot x^\star$
by (*simp add*: *star-sim1*)
thus *?thesis*
by (*simp add*: *fdemodalisation2*)
qed

lemma *fdia-star-induct*: $d\ z + |x\rangle\ y \leq d\ y \Longrightarrow |x^\star\rangle\ z \leq d\ y$
proof –
assume *a*: $d\ z + |x\rangle\ y \leq d\ y$
hence *b*: $d\ z \leq d\ y$ **and** *c*: $|x\rangle\ y \leq d\ y$
apply (*simp add*: *local.join.le-supE*)
using *a* **by** *auto*
hence *d*: $|x^\star\rangle\ z \leq |x^\star\rangle\ y$
using *fd-def fd-iso1* **by** *auto*
have $|x^\star\rangle\ y \leq d\ y$
using *c fdia-star-induct-var* **by** *blast*
thus *?thesis*
using *d* **by** *fastforce*
qed

lemma *fdia-star-induct-eq*: $d\ z + |x\rangle\ y = d\ y \Longrightarrow |x^\star\rangle\ z \leq d\ y$
  by (*simp add*: *fdia-star-induct*)

end

class *domain-kleene-algebra* = *kleene-algebra* + *domain-semiring*

begin

subclass *domain-left-kleene-algebra* ..

end

end

3 Antidomain Semirings

theory *Antidomain-Semiring*
imports *Domain-Semiring*
begin

3.1 Antidomain Monoids

We axiomatise antidomain monoids, using the axioms of [3].

class *antidomain-op* =
  fixes *antidomain-op* :: $'a \Rightarrow 'a$ (*ad*)

class *antidomain-left-monoid* = *monoid-mult* + *antidomain-op* +
  assumes *am1* [*simp*]: $ad\ x \cdot x = ad\ 1$
  and *am2*: $ad\ x \cdot ad\ y = ad\ y \cdot ad\ x$
  and *am3* [*simp*]: $ad\ (ad\ x) \cdot x = x$
  and *am4* [*simp*]: $ad\ (x \cdot y) \cdot ad\ (x \cdot ad\ y) = ad\ x$
  and *am5* [*simp*]: $ad\ (x \cdot y) \cdot x \cdot ad\ y = ad\ (x \cdot y) \cdot x$

begin

no-notation *domain-op* (*d*)
no-notation *zero-class.zero* (*0*)

We define a zero element and operations of domain and addition.

definition *a-zero* :: $'a$ (*0*) **where**
  $0 = ad\ 1$

definition *am-d* :: $'a \Rightarrow 'a$ (*d*) **where**
  $d\ x = ad\ (ad\ x)$

definition *am-add-op* :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixl** $\oplus$ *65*) **where**

x ⊕ *y* ≡ *ad* (*ad x* · *ad y*)

lemma *a-d-zero* [*simp*]: *ad x* · *d x* = *0*
by (*metis am1 am2 a-zero-def am-d-def*)

lemma *a-d-one* [*simp*]: *d x* ⊕ *ad x* = *1*
by (*metis am1 am3 mult-1-right am-d-def am-add-op-def*)

lemma *n-annil* [*simp*]: *0* · *x* = *0*
proof −
have *0* · *x* = *d x* · *ad x* · *x*
by (*simp add*: *a-zero-def am-d-def*)
also have ... = *d x* · *0*
by (*metis am1 mult-assoc a-zero-def*)
thus *?thesis*
by (*metis am1 am2 am3 mult-assoc a-zero-def*)
qed

lemma *a-mult-idem* [*simp*]: *ad x* · *ad x* = *ad x*
proof −
have *ad x* · *ad x* = *ad* (*1* · *x*) · *1* · *ad x*
by *simp*
also have ... = *ad* (*1* · *x*) · *1*
using *am5* **by** *blast*
finally show *?thesis*
by *simp*
qed

lemma *a-add-idem* [*simp*]: *ad x* ⊕ *ad x* = *ad x*
by (*metis am1 am3 am4 mult-1-right am-add-op-def*)

The next three axioms suffice to show that the domain elements form a Boolean algebra.

lemma *a-add-comm*: *x* ⊕ *y* = *y* ⊕ *x*
using *am2 am-add-op-def* **by** *auto*

lemma *a-add-assoc*: *x* ⊕ (*y* ⊕ *z*) = (*x* ⊕ *y*) ⊕ *z*
proof −
have ⋀*x y*. *ad x* · *ad* (*x* · *y*) = *ad x*
by (*metis a-mult-idem am2 am4 mult-assoc*)
thus *?thesis*
by (*metis a-add-comm am-add-op-def local.am3 local.am4 mult-assoc*)
qed

lemma *huntington* [*simp*]: *ad* (*x* ⊕ *y*) ⊕ *ad* (*x* ⊕ *ad y*) = *ad x*
using *a-add-idem am-add-op-def* **by** *auto*

lemma *a-absorb1* [*simp*]: (*ad x* ⊕ *ad y*) · *ad x* = *ad x*
by (*metis a-add-idem a-mult-idem am4 mult-assoc am-add-op-def*)

lemma *a-absorb2* [*simp*]: *ad x* ⊕ *ad x* · *ad y* = *ad x*
proof −
  have *ad* (*ad x*) · *ad* (*ad x* · *ad y*) = *ad* (*ad x*)
    by (*metis* (*no-types*) *a-mult-idem local.am4 local.mult.semigroup-axioms semigroup.assoc*)
  then show *?thesis*
    using *a-add-idem am-add-op-def* **by** *auto*
qed

The distributivity laws remain to be proved; our proofs follow those of Maddux [12].

lemma *prod-split* [*simp*]: *ad x* · *ad y* ⊕ *ad x* · *d y* = *ad x*
  using *a-add-idem am-d-def am-add-op-def* **by** *auto*

lemma *sum-split* [*simp*]: (*ad x* ⊕ *ad y*) · (*ad x* ⊕ *d y*) = *ad x*
  using *a-add-idem am-d-def am-add-op-def* **by** *fastforce*

lemma *a-comp-simp* [*simp*]: (*ad x* ⊕ *ad y*) · *d x* = *ad y* · *d x*
proof −
  have *f1*: (*ad x* ⊕ *ad y*) · *d x* = *ad* (*ad* (*ad x*) · *ad* (*ad y*)) · *ad* (*ad x*) · *ad* (*ad* (*ad y*))
    by (*simp add*: *am-add-op-def am-d-def*)
  have *f2*: *ad y* = *ad* (*ad* (*ad y*))
    using *a-add-idem am-add-op-def* **by** *auto*
  have *ad y* = *ad* (*ad* (*ad x*) · *ad* (*ad y*)) · *ad y*
    by (*metis* (*no-types*) *a-absorb1 a-add-comm am-add-op-def*)
  then show *?thesis*
    using *f2 f1* **by** (*simp add*: *am-d-def local.am2 local.mult.semigroup-axioms semigroup.assoc*)
qed

lemma *a-distrib1*: *ad x* · (*ad y* ⊕ *ad z*) = *ad x* · *ad y* ⊕ *ad x* · *ad z*
proof −
  have *f1*: ⋀*a*. *ad* (*ad* (*ad* (*a*::′*a*)) · *ad* (*ad a*)) = *ad a*
    using *a-add-idem am-add-op-def* **by** *auto*
  have *f2*: ⋀*a aa*. *ad* ((*a*::′*a*) · *aa*) · (*a* · *ad aa*) = *ad* (*a* · *aa*) · *a*
    using *local.am5 mult-assoc* **by** *auto*
  have *f3*: ⋀*a*. *ad* (*ad* (*ad* (*a*::′*a*))) = *ad a*
    using *f1* **by** *simp*
  have ⋀*a*. *ad* (*a*::′*a*) · *ad a* = *ad a*
    by *simp*
  then have ⋀*a aa*. *ad* (*ad* (*ad* (*a*::′*a*) · *ad aa*)) = *ad aa* · *ad a*
    using *f3 f2* **by** (*metis* (*no-types*) *local.am2 local.am4 mult-assoc*)
  then have *ad x* · (*ad y* ⊕ *ad z*) = *ad x* · (*ad y* ⊕ *ad z*) · *ad y* ⊕ *ad x* · (*ad y* ⊕ *ad z*) · *d y*
    using *am-add-op-def am-d-def local.am2 local.am4* **by** *presburger*
  also have ... = *ad x* · *ad y* ⊕ *ad x* · (*ad y* ⊕ *ad z*) · *d y*
    by (*simp add*: *mult-assoc*)

also have ... = $ad\ x \cdot ad\ y \oplus ad\ x \cdot ad\ z \cdot d\ y$
  by (*simp add*: *mult-assoc*)
also have ... = $ad\ x \cdot ad\ y \oplus ad\ x \cdot ad\ y \cdot ad\ z \oplus ad\ x \cdot ad\ z \cdot d\ y$
  by (*metis a-add-idem a-mult-idem local.am4 mult-assoc am-add-op-def*)
also have ... = $ad\ x \cdot ad\ y \oplus (ad\ x \cdot ad\ z \cdot ad\ y \oplus ad\ x \cdot ad\ z \cdot d\ y)$
  by (*metis am2 mult-assoc a-add-assoc*)
finally show *?thesis*
  by (*metis a-add-idem a-mult-idem am4 am-d-def am-add-op-def*)
qed

lemma *a-distrib2*: $ad\ x \oplus ad\ y \cdot ad\ z = (ad\ x \oplus ad\ y) \cdot (ad\ x \oplus ad\ z)$
proof –
have *f1*: $\bigwedge a\ aa\ ab.\ ad\ (ad\ (ad\ (a::'a) \cdot ad\ aa) \cdot ad\ (ad\ a \cdot ad\ ab)) = ad\ a \cdot ad\ (ad\ (ad\ aa) \cdot ad\ (ad\ ab))$
  using *a-distrib1 am-add-op-def* **by** *auto*
have $\bigwedge a.\ ad\ (ad\ (ad\ (a::'a))) = ad\ a$
  by (*metis a-absorb2 a-mult-idem am-add-op-def*)
then have $ad\ (ad\ (ad\ x) \cdot ad\ (ad\ y)) \cdot ad\ (ad\ (ad\ x) \cdot ad\ (ad\ z)) = ad\ (ad\ (ad\ x) \cdot ad\ (ad\ y \cdot ad\ z))$
  using *f1* **by** (*metis* (*full-types*))
then show *?thesis*
  by (*simp add*: *am-add-op-def*)
qed

lemma *aa-loc* [*simp*]: $d\ (x \cdot d\ y) = d\ (x \cdot y)$
proof –
have *f1*: $x \cdot d\ y \cdot y = x \cdot y$
  by (*metis am3 mult-assoc am-d-def*)
have *f2*: $\bigwedge w\ z.\ ad\ (w \cdot z) \cdot (w \cdot ad\ z) = ad\ (w \cdot z) \cdot w$
  by (*metis am5 mult-assoc*)
hence *f3*: $\bigwedge z.\ ad\ (x \cdot y) \cdot (x \cdot z) = ad\ (x \cdot y) \cdot (x \cdot (ad\ (ad\ (ad\ y) \cdot y) \cdot z))$
  using *f1* **by** (*metis* (*no-types*) *mult-assoc am-d-def*)
have $ad\ (x \cdot ad\ (ad\ y)) \cdot (x \cdot y) = 0$ **using** *f1*
  by (*metis am1 mult-assoc n-annil a-zero-def am-d-def*)
thus *?thesis*
  by (*metis a-d-zero am-d-def f3 local.am1 local.am2 local.am3 local.am4*)
qed

lemma *a-loc* [*simp*]: $ad\ (x \cdot d\ y) = ad\ (x \cdot y)$
proof –
have $\bigwedge a.\ ad\ (ad\ (ad\ (a::'a))) = ad\ a$
  using *am-add-op-def am-d-def prod-split* **by** *auto*
then show *?thesis*
  by (*metis* (*full-types*) *aa-loc am-d-def*)
qed

lemma *d-a-export* [*simp*]: $d\ (ad\ x \cdot y) = ad\ x \cdot d\ y$
proof –
have *f1*: $\bigwedge a\ aa.\ ad\ ((a::'a) \cdot ad\ (ad\ aa)) = ad\ (a \cdot aa)$

```
    using a-loc am-d-def by auto
  have ⋀a. ad (ad (a::'a) · a) = 1
    using a-d-one am-add-op-def am-d-def by auto
  then have ⋀a aa. ad (ad (ad (a::'a) · ad aa)) = ad a · ad aa
    using f1 by (metis a-distrib2 am-add-op-def local.mult-1-left)
  then show ?thesis
    using f1 by (metis (no-types) am-d-def)
qed
```

Every antidomain monoid is a domain monoid.

```
sublocale dm: domain-monoid am-d op · 1
  apply (unfold-locales)
  apply (simp add: am-d-def)
  apply simp
  using am-d-def d-a-export apply auto[1]
  by (simp add: am-d-def local.am2)

lemma ds-ord-iso1: x ⊑ y ⟹ z · x ⊑ z · y

oops

lemma a-very-costrict: ad x = 1 ⟷ x = 0
proof
  assume a: ad x = 1
  hence 0 = ad x · x
    using a-zero-def by force
  thus x = 0
    by (simp add: a)
next
  assume x = 0
  thus ad x = 1
    using a-zero-def am-d-def dm.dom-one by auto
qed

lemma a-weak-loc: x · y = 0 ⟷ x · d y = 0
proof −
  have x · y = 0 ⟷ ad (x · y) = 1
    by (simp add: a-very-costrict)
  also have ... ⟷ ad (x · d y) = 1
    by simp
  finally show ?thesis
    using a-very-costrict by blast
qed

lemma a-closure [simp]: d (ad x) = ad x
  using a-add-idem am-add-op-def am-d-def by auto

lemma a-d-mult-closure [simp]: d (ad x · ad y) = ad x · ad y
  by simp
```

lemma *kat-3′*: $d\ x \cdot y \cdot ad\ z = 0 \Longrightarrow d\ x \cdot y = d\ x \cdot y \cdot d\ z$
  by (*metis dm.dom-one local.am5 local.mult-1-left a-zero-def am-d-def*)

lemma *s4* [*simp*]: $ad\ x \cdot ad\ (ad\ x \cdot y) = ad\ x \cdot ad\ y$
proof −
  have $\bigwedge a\ aa.\ ad\ (a::'a) \cdot ad\ (ad\ aa) = ad\ (ad\ (ad\ a \cdot aa))$
    using *am-d-def d-a-export* **by** *presburger*
  then have $\bigwedge a\ aa.\ ad\ (ad\ (a::'a)) \cdot ad\ aa = ad\ (ad\ (ad\ aa \cdot a))$
    using *local.am2* **by** *presburger*
  then show *?thesis*
    by (*metis a-comp-simp a-d-mult-closure am-add-op-def am-d-def local.am2*)
qed

end

class *antidomain-monoid* = *antidomain-left-monoid* +
  assumes *am6* [*simp*]: $x \cdot ad\ 1 = ad\ 1$

begin

lemma *kat-3-equiv*: $d\ x \cdot y \cdot ad\ z = 0 \longleftrightarrow d\ x \cdot y = d\ x \cdot y \cdot d\ z$
  apply *standard*
  apply (*metis kat-3′*)
  by (*simp add*: *mult-assoc a-zero-def am-d-def*)

no-notation *a-zero* (*0*)
no-notation *am-d* (*d*)

end

3.2 Antidomain Near-Semirings

We define antidomain near-semirings. We do not consider units separately. The axioms are taken from [6].

notation *zero-class.zero* (*0*)

class *antidomain-near-semiring* = *ab-near-semiring-one-zerol* + *antidomain-op* + *plus-ord* +
  assumes *ans1* [*simp*]: $ad\ x \cdot x = 0$
  and *ans2* [*simp*]: $ad\ (x \cdot y) + ad\ (x \cdot ad\ (ad\ y)) = ad\ (x \cdot ad\ (ad\ y))$
  and *ans3* [*simp*]: $ad\ (ad\ x) + ad\ x = 1$
  and *ans4* [*simp*]: $ad\ (x + y) = ad\ x \cdot ad\ y$

begin

definition *ans-d* :: $'a \Rightarrow 'a$ (*d*) **where**
   $d\ x = ad\ (ad\ x)$

lemma *a-a-one* [*simp*]: *d 1* = *1*
proof −
  have *d 1* = *d 1* + *0*
    by *simp*
  also have ... = *d 1* + *ad 1*
    by (*metis ans1 mult-1-right*)
  finally show *?thesis*
    by (*simp add*: *ans-d-def*)
qed

lemma *a-very-costrict'*: *ad x* = *1* ⟷ *x* = *0*
proof
  assume *ad x* = *1*
  hence *x* = *ad x* · *x*
    by *simp*
  thus *x* = *0*
    by *auto*
next
  assume *x* = *0*
  hence *ad x* = *ad 0*
    by *blast*
  thus *ad x* = *1*
    by (*metis a-a-one ans-d-def local.ans1 local.mult-1-right*)
qed

lemma *one-idem* [*simp*]: *1* + *1* = *1*
proof −
  have *1* + *1* = *d 1* + *d 1*
    by *simp*
  also have ... = *ad* (*ad 1* · *1*) + *ad* (*ad 1* · *d 1*)
    using *a-a-one ans-d-def* **by** *auto*
  also have ... = *ad* (*ad 1* · *d 1*)
    using *ans-d-def local.ans2* **by** *presburger*
  also have ... = *ad* (*ad 1* · *1*)
    by *simp*
  also have ... = *d 1*
    by (*simp add*: *ans-d-def*)
  finally show *?thesis*
    by *simp*
qed

Every antidomain near-semiring is automatically a dioid, and therefore ordered.

subclass *near-dioid-one-zerol*
proof
  show ⋀*x*. *x* + *x* = *x*
  proof −
    fix *x*
    have *x* + *x* = *1* · *x* + *1* · *x*

```
      by simp
    also have ... = (1 + 1) · x
      using distrib-right′ by presburger
    finally show x + x = x
      by simp
  qed
qed

lemma d1-a [simp]: d x · x = x
proof −
  have x = (d x + ad x) · x
    by (simp add: ans-d-def)
  also have ... = d x · x + ad x · x
    using distrib-right′ by blast
  also have ... = d x · x + 0
    by simp
  finally show ?thesis
    by auto
qed

lemma a-comm: ad x · ad y = ad y · ad x
  using add-commute ans4 by fastforce

lemma a-subid: ad x ≤ 1
  using local.ans3 local.join.sup-ge2 by fastforce

lemma a-subid-aux1: ad x · y ≤ y
  using a-subid mult-isor by fastforce

lemma a-subdist: ad (x + y) ≤ ad x
  by (metis a-subid-aux1 ans4 add-comm)

lemma a-antitone: x ≤ y ⟹ ad y ≤ ad x
  using a-subdist local.order-prop by auto

lemma a-mul-d [simp]: ad x · d x = 0
  by (metis a-comm ans-d-def local.ans1)

lemma a-gla1: ad x · y = 0 ⟹ ad x ≤ ad y
proof −
  assume ad x · y = 0
  hence a: ad x · d y = 0
    by (metis a-subid a-very-costrict′ ans-d-def local.ans2 local.join.sup.order-iff)
  have ad x = (d y + ad y ) · ad x
    by (simp add: ans-d-def)
  also have ... = d y · ad x + ad y · ad x
    using distrib-right′ by blast
  also have ... = ad x · d y + ad x · ad y
    using a-comm ans-d-def by auto
```

also have ... = *ad x · ad y*
by (*simp add*: *a*)
finally show *ad x ≤ ad y*
by (*metis a-subid-aux1*)
qed

lemma *a-gla2*: *ad x ≤ ad y ⟹ ad x · y = 0*
proof −
assume *ad x ≤ ad y*
hence *ad x · y ≤ ad y · y*
using *mult-isor* **by** *blast*
thus *?thesis*
by (*simp add*: *join.le-bot*)
qed

lemma *a2-eq* [*simp*]: *ad* (*x · d y*) = *ad* (*x · y*)
proof (*rule antisym*)
show *ad* (*x · y*) ≤ *ad* (*x · d y*)
by (*simp add*: *ans-d-def local.less-eq-def*)
next
show *ad* (*x · d y*) ≤ *ad* (*x · y*)
by (*metis a-gla1 a-mul-d ans1 d1-a mult-assoc*)
qed

lemma *a-export′* [*simp*]: *ad* (*ad x · y*) = *d x* + *ad y*
proof (*rule antisym*)
have *ad* (*ad x · y*) · *ad x · d y* = *0*
by (*simp add*: *a-gla2 local.mult.semigroup-axioms semigroup.assoc*)
hence *a*: *ad* (*ad x · y*) · *d y* ≤ *ad* (*ad x*)
by (*metis a-comm a-gla1 ans4 mult-assoc ans-d-def*)
have *ad* (*ad x · y*) = *ad* (*ad x · y*) · *d y* + *ad* (*ad x · y*) · *ad y*
by (*metis* (*no-types*) *add-commute ans3 ans4 distrib-right′ mult-onel ans-d-def*)
thus *ad* (*ad x · y*) ≤ *d x* + *ad y*
by (*metis a-subid-aux1 a join.sup-mono ans-d-def*)
next
show *d x* + *ad y* ≤ *ad* (*ad x · y*)
by (*metis a2-eq a-antitone a-comm a-subid-aux1 join.sup-least ans-d-def*)
qed

Every antidomain near-semiring is a domain near-semiring.

sublocale *dnsz*: *domain-near-semiring-one-zerol op* + *op* · *1 0 ans-d op* ≤ *op* <
apply (*unfold-locales*)
apply *simp*
using *a2-eq ans-d-def* **apply** *auto*[*1*]
apply (*simp add*: *a-subid ans-d-def local.join.sup-absorb2*)
apply (*simp add*: *ans-d-def*)
apply (*simp add*: *a-comm ans-d-def*)
using *a-a-one a-very-costrict′ ans-d-def* **by** *force*

lemma *a-idem* [*simp*]: *ad* $x \cdot$ *ad* $x =$ *ad* x
proof −
  have *ad* $x = ($*d* $x +$ *ad* $x\ ) \cdot$ *ad* x
    by (*simp add*: *ans-d-def*)
  also have ... $=$ *d* $x \cdot$ *ad* $x +$ *ad* $x \cdot$ *ad* x
    using *distrib-right'* **by** *blast*
  finally show *?thesis*
    by (*simp add*: *ans-d-def*)
qed

lemma *a-3-var* [*simp*]: *ad* $x \cdot$ *ad* $y \cdot (x + y) = 0$
  by (*metis ans1 ans4*)

lemma *a-3* [*simp*]: *ad* $x \cdot$ *ad* $y \cdot$ *d* $(x + y) = 0$
  by (*metis a-mul-d ans4*)

lemma *a-closure'* [*simp*]: *d* (*ad* x) $=$ *ad* x
proof −
  have *d* (*ad* x) $=$ *ad* $(1 \cdot$ *d* $x)$
    by (*simp add*: *ans-d-def*)
  also have ... $=$ *ad* $(1 \cdot x)$
    using *a2-eq* **by** *blast*
  finally show *?thesis*
    by *simp*
qed

The following counterexamples show that some of the antidomain monoid axioms do not need to hold.

lemma $x \cdot$ *ad* $1 =$ *ad* 1

oops

lemma *ad* $(x \cdot y) \cdot$ *ad* $(x \cdot$ *ad* $y) =$ *ad* x

oops

lemma *ad* $(x \cdot y) \cdot$ *ad* $(x \cdot$ *ad* $y) =$ *ad* x

oops

lemma *phl-seq-inv*: *d* $v \cdot x \cdot y \cdot$ *ad* $w = 0 \Longrightarrow \exists z.$ *d* $v \cdot x \cdot$ *d* $z = 0 \wedge$ *ad* $z \cdot y \cdot$ *ad* $w = 0$
proof −
  assume *d* $v \cdot x \cdot y \cdot$ *ad* $w = 0$
  hence *d* $v \cdot x \cdot$ *d* $(y \cdot$ *ad* $w) = 0 \wedge$ *ad* $(y \cdot$ *ad* $w) \cdot y \cdot$ *ad* $w = 0$
    by (*metis dnsz.dom-weakly-local local.ans1 mult-assoc*)
  thus $\exists z.$ *d* $v \cdot x \cdot$ *d* $z = 0 \wedge$ *ad* $z \cdot y \cdot$ *ad* $w = 0$
    by *blast*
qed

lemma *a-fixpoint*: *ad x* = *x* ⟹ (∀ *y*. *y* = *0*)
proof −
  assume *a1*: *ad x* = *x*
  { fix *aa* :: *'a*
    have *aa* = *0*
      using *a1* **by** (*metis* (*no-types*) *a-mul-d ans-d-def local.annil local.ans3 local.join.sup.idem local.mult-1-left*)
  }
  then show *?thesis*
    by *blast*
qed

no-notation *ans-d* (*d*)

end

3.3 Antidomain Pre-Dioids

Antidomain pre-diods are based on a different set of axioms, which are again taken from [6].

class *antidomain-pre-dioid* = *pre-dioid-one-zerol* + *antidomain-op* +
  assumes *apd1* [*simp*]: *ad x* · *x* = *0*
  and *apd2* [*simp*]: *ad* (*x* · *y*) ≤ *ad* (*x* · *ad* (*ad y*))
  and *apd3* [*simp*]: *ad* (*ad x*) + *ad x* = *1*

begin

definition *apd-d* :: *'a* ⇒ *'a* (*d*) **where**
  d x = *ad* (*ad x*)

lemma *a-very-costrict''*: *ad x* = *1* ⟷ *x* = *0*
  by (*metis add-commute local.add-zerol local.antisym local.apd1 local.apd3 local.join.bot-least local.mult-1-right local.phl-skip*)

lemma *a-subid'*: *ad x* ≤ *1*
  using *local.apd3 local.join.sup-ge2* **by** *fastforce*

lemma *d1-a'* [*simp*]: *d x* · *x* = *x*
proof −
  have *x* = (*d x* + *ad x*) · *x*
    by (*simp add*: *apd-d-def*)
  also have ... = *d x* · *x* + *ad x* · *x*
    using *distrib-right'* **by** *blast*
  also have ... = *d x* · *x* + *0*
    by *simp*
  finally show *?thesis*
    by *auto*
qed

lemma *a-subid-aux1′*: *ad x · y ≤ y*
  using *a-subid′ mult-isor* **by** *fastforce*

lemma *a-mul-d′* [*simp*]: *ad x · d x = 0*
proof −
  have *1 = ad (ad x · x)*
    using *a-very-costrict′′* **by** *force*
  thus *?thesis*
    by (*metis a-subid′ a-very-costrict′′ apd-d-def local.antisym local.apd2*)
qed

lemma *a-d-closed* [*simp*]: *d (ad x) = ad x*
proof (*rule antisym*)
  have *d (ad x) = (d x + ad x) · d (ad x)*
    by (*simp add*: *apd-d-def*)
  also have ... = *ad (ad x) · ad (d x) + ad x · d (ad x)*
    using *apd-d-def local.distrib-right′* **by** *presburger*
  also have ... = *ad x · d (ad x)*
    using *a-mul-d′ apd-d-def* **by** *auto*
      finally show *d (ad x) ≤ ad x*
    by (*metis a-subid′ mult-1-right mult-isol apd-d-def*)
next
  have *ad x = ad (1 · x)*
    by *simp*
  also have ... ≤ *ad (1 · d x)*
    using *apd-d-def local.apd2* **by** *presburger*
  also have ... = *ad (d x)*
    by *simp*
  finally show *ad x ≤ d (ad x)*
   by (*simp add*: *apd-d-def*)
qed

lemma *meet-ord-def*: *ad x ≤ ad y ⟷ ad x · ad y = ad x*
  by (*metis a-d-closed a-subid-aux1′ d1-a′ eq-iff mult-1-right mult-isol*)

lemma *d-weak-loc*: *x · y = 0 ⟷ x · d y = 0*
proof −
  have *x · y = 0 ⟷ ad (x · y) = 1*
    by (*simp add*: *a-very-costrict′′*)
  also have ... ⟷ *ad (x · d y) = 1*
    by (*metis apd1 apd2 a-subid′ apd-d-def d1-a′ eq-iff mult-1-left mult-assoc*)
  finally show *?thesis*
    by (*simp add*: *a-very-costrict′′*)
qed

lemma *gla-1*: *ad x · y = 0 ⟹ ad x ≤ ad y*
proof −
  assume *ad x · y = 0*

hence *a*: *ad x · d y = 0*
  using *d-weak-loc* **by** *force*
hence *d y = ad x · d y + d y*
  by *simp*
also have ... = (*1 + ad x*) · *d y*
  using *join.sup-commute* **by** *auto*
also have ... = (*d x + ad x*) · *d y*
  using *apd-d-def calculation* **by** *auto*
also have ... = *d x · d y*
  by (*simp add*: *a join.sup-commute*)
finally have *d y ≤ d x*
  by (*metis apd-d-def a-subid′ mult-1-right mult-isol*)
hence *d y · ad x = 0*
  by (*metis apd-d-def a-d-closed a-mul-d′ distrib-right′ less-eq-def no-trivial-inverse*)
hence *ad x = ad y · ad x*
  by (*metis apd-d-def apd3 add-0-left distrib-right′ mult-1-left*)
thus *ad x ≤ ad y*
  by (*metis add-commute apd3 mult-oner subdistl*)
qed

lemma *a2-eq′* [*simp*]: *ad* (*x · d y*) = *ad* (*x · y*)
proof (*rule antisym*)
  show *ad* (*x · y*) ≤ *ad* (*x · d y*)
    by (*simp add*: *apd-d-def*)
next
  show *ad* (*x · d y*) ≤ *ad* (*x · y*)
    by (*metis gla-1 apd1 a-mul-d′ d1-a′ mult-assoc*)
qed

lemma *a-supdist-var*: *ad* (*x + y*) ≤ *ad x*
  by (*metis gla-1 apd1 join.le-bot subdistl*)

lemma *a-antitone′*: *x ≤ y ⟹ ad y ≤ ad x*
  using *a-supdist-var local.order-prop* **by** *auto*

lemma *a-comm-var*: *ad x · ad y ≤ ad y · ad x*
proof −
  have *ad x · ad y = d* (*ad x · ad y*) · *ad x · ad y*
    by (*simp add*: *mult-assoc*)
  also have ... ≤ *d* (*ad x · ad y*) · *ad x*
    using *a-subid′ mult-isol* **by** *fastforce*
  also have ... ≤ *d* (*ad y*) · *ad x*
    by (*simp add*: *a-antitone′ a-subid-aux1′ apd-d-def local.mult-isor*)
  finally show *?thesis*
    by *simp*
qed

lemma *a-comm′*: *ad x · ad y = ad y · ad x*
  by (*simp add*: *a-comm-var eq-iff*)

```
lemma a-closed [simp]: d (ad x · ad y) = ad x · ad y
proof –
  have f1: ⋀x y. ad x ≤ ad (ad y · x)
    by (simp add: a-antitone′ a-subid-aux1′)
  have ⋀x y. d (ad x · y) ≤ ad x
    by (metis a2-eq′ a-antitone′ a-comm′ a-d-closed apd-d-def f1)
  hence ⋀x y. d (ad x · y) · y = ad x · y
    by (metis d1-a′ meet-ord-def mult-assoc apd-d-def)
  thus ?thesis
    by (metis f1 a-comm′ apd-d-def meet-ord-def)
qed

lemma a-export′′ [simp]: ad (ad x · y) = d x + ad y
proof (rule antisym)
  have ad (ad x · y) · ad x · d y = 0
    using d-weak-loc mult-assoc by fastforce
  hence a: ad (ad x · y) · d y ≤ d x
    by (metis a-closed a-comm′ apd-d-def gla-1 mult-assoc)
  have ad (ad x · y) = ad (ad x · y) · d y + ad (ad x · y) · ad y
    by (metis apd3 a-comm′ d1-a′ distrib-right′ mult-1-right apd-d-def)
  thus ad (ad x · y) ≤ d x + ad y
    by (metis a-subid-aux1′ a join.sup-mono)
next
  have ad y ≤ ad (ad x · y)
    by (simp add: a-antitone′ a-subid-aux1′)
  thus d x + ad y ≤ ad (ad x · y)
    by (metis apd-d-def a-mul-d′ d1-a′ gla-1 apd1 join.sup-least mult-assoc)
qed

lemma d1-sum-var: x + y ≤ (d x + d y) · (x + y)
proof –
  have x + y = d x · x + d y · y
    by simp
  also have ... ≤ (d x + d y) · x + (d x + d y) · y
   using local.distrib-right′ local.join.sup-ge1 local.join.sup-ge2 local.join.sup-mono
by presburger
  finally show ?thesis
    using order-trans subdistl-var by blast
qed

lemma a4′: ad (x + y) = ad x · ad y
proof (rule antisym)
  show ad (x + y) ≤ ad x · ad y
    by (metis a-d-closed a-supdist-var add-commute d1-a′ local.mult-isol-var)
  hence ad x · ad y = ad x · ad y + ad (x + y)
    using less-eq-def add-commute by simp
  also have ... = ad (ad (ad x · ad y) · (x + y))
    by (metis a-closed a-export′′)
```

finally show *ad x* · *ad y* ≤ *ad* (*x* + *y*)
  using *a-antitone′ apd-d-def d1-sum-var* **by** *auto*
qed

Antidomain pre-dioids are domain pre-dioids and antidomain near-semirings, but still not antidomain monoids.

sublocale *dpdz*: *domain-pre-dioid-one-zerol op* + *op* · *1 0 op* ≤ *op* < λ*x*. *ad* (*ad x*)
  apply (*unfold-locales*)
  using *apd-d-def d1-a′* **apply** *auto*[*1*]
  using *a2-eq′ apd-d-def* **apply** *auto*[*1*]
  apply (*simp add*: *a-subid′*)
  apply (*simp add*: *a4′ apd-d-def*)
  by (*metis a-mul-d′ a-very-costrict″ apd-d-def local.mult-onel*)

subclass *antidomain-near-semiring*
  apply (*unfold-locales*)
  apply *simp*
  using *local.apd2 local.less-eq-def* **apply** *blast*
  apply *simp*
  by (*simp add*: *a4′*)

lemma *a-supdist*: *ad* (*x* + *y*) ≤ *ad x* + *ad y*
  using *a-supdist-var local.join.le-supI1* **by** *auto*

lemma *a-gla*: *ad x* · *y* = *0* ⟷ *ad x* ≤ *ad y*
  using *gla-1 a-gla2* **by** *blast*

lemma *a-subid-aux2*: *x* · *ad y* ≤ *x*
  using *a-subid′ mult-isol* **by** *fastforce*

lemma *a42-var*: *d x* · *d y* ≤ *ad* (*ad x* + *ad y*)
  by (*simp add*: *apd-d-def*)

lemma *d1-weak* [*simp*]: (*d x* + *d y*) · *x* = *x*
proof −
  have (*d x* + *d y*) · *x* = (*1* + *d y*) · *x*
    by *simp*
  thus *?thesis*
    by (*metis add-commute apd-d-def dpdz.dnso3 local.mult-1-left*)
qed

lemma *x* · *ad 1* = *ad 1*

oops

lemma *ad x* · (*y* + *z*) = *ad x* · *y* + *ad x* · *z*

oops

lemma *ad* $(x \cdot y) \cdot$ *ad* $(x \cdot$ *ad* $y) =$ *ad* x

oops

lemma *ad* $(x \cdot y) \cdot$ *ad* $(x \cdot$ *ad* $y) =$ *ad* x

oops

no-notation *apd-d* (d)

end

3.4 Antidomain Semirings

Antidomain semirings are direct expansions of antidomain pre-dioids, but do not require idempotency of addition. Hence we give a slightly different axiomatisation, following [7].

class *antidomain-semiringl* = *semiring-one-zerol* + *plus-ord* + *antidomain-op* +
  assumes *as1* [*simp*]: *ad* $x \cdot x = 0$
  and *as2* [*simp*]: *ad* $(x \cdot y) +$ *ad* $(x \cdot$ *ad* (*ad* $y)) =$ *ad* $(x \cdot$ *ad* (*ad* $y))$
  and *as3* [*simp*]: *ad* (*ad* x) + *ad* $x = 1$

begin

definition *ads-d* :: $'a \Rightarrow 'a$ (d) **where**
  $d\ x =$ *ad* (*ad* x)

lemma *one-idem'*: $1 + 1 = 1$
  by (*metis as1 as2 as3 add-zeror mult.right-neutral*)

Every antidomain semiring is a dioid and an antidomain pre-dioid.

subclass *dioid*
  by (*standard*, *metis distrib-left mult.right-neutral one-idem'*)

subclass *antidomain-pre-dioid*
  by (*unfold-locales*, *auto simp*: *local.less-eq-def*)

lemma *am5-lem* [*simp*]: *ad* $(x \cdot y) \cdot$ *ad* $(x \cdot$ *ad* $y) =$ *ad* x
proof −
  have *ad* $(x \cdot y\ ) \cdot$ *ad* $(x \cdot$ *ad* $y) =$ *ad* $(x \cdot d\ y) \cdot$ *ad* $(x \cdot$ *ad* $y)$
    using *ads-d-def local.a2-eq' local.apd-d-def* **by** *auto*
  also have ... = *ad* $(x \cdot d\ y + x \cdot$ *ad* $y)$
    using *ans4* **by** *presburger*
  also have ... = *ad* $(x \cdot (d\ y +$ *ad* $y))$
    using *distrib-left* **by** *presburger*
  finally show *?thesis*
    by (*simp add*: *ads-d-def*)

qed

lemma *am6-lem* [*simp*]: *ad* (*x* · *y*) · *x* · *ad y* = *ad* (*x* · *y*) · *x*
proof −
  fix *x y*
  have *ad* (*x* · *y*) · *x* · *ad y* = *ad* (*x* · *y*) · *x* · *ad y* + *0*
    by *simp*
  also have ... = *ad* (*x* · *y*) · *x* · *ad y* + *ad* (*x* · *d y*) · *x* · *d y*
    using *ans1 mult-assoc* **by** *presburger*
  also have ... = *ad* (*x* · *y*) · *x* · (*ad y* + *d y*)
    using *ads-d-def local.a2-eq′ local.apd-d-def local.distrib-left* **by** *auto*
  finally show *ad* (*x* · *y*) · *x* · *ad y* = *ad* (*x* · *y*) · *x*
    using *add-commute ads-d-def local.as3* **by** *auto*
qed

lemma *a-zero* [*simp*]: *ad 0* = *1*
  by (*simp add*: *local.a-very-costrict′′*)

lemma *a-one* [*simp*]: *ad 1* = *0*
  using *a-zero local.dpdz.dpd5* **by** *blast*

subclass *antidomain-left-monoid*
  by (*unfold-locales*, *auto simp*: *local.a-comm′*)

Every antidomain left semiring is a domain left semiring.

no-notation *domain-semiringl-class.fd* ((|-⟩ -) [*61*,*81*] *82*)

definition *fdia* :: ′*a* ⇒ ′*a* ⇒ ′*a* ((|-⟩ -) [*61*,*81*] *82*) **where**
  |*x*⟩ *y* = *ad* (*ad* (*x* · *y*))

sublocale *ds*: *domain-semiringl op* + *op* · *1 0* λ*x*. *ad* (*ad x*) *op* ≤ *op* <
  rewrites *ds.fd x y* ≡ *fdia x y*
proof −
  show *class.domain-semiringl op* + *op* · *1 0* (λ*x*. *ad* (*ad x*)) *op* ≤ *op* <
    by (*unfold-locales*, *auto simp*: *local.dpdz.dpd4 ans-d-def*)
  then interpret *ds*: *domain-semiringl op* + *op* · *1 0* λ*x*. *ad* (*ad x*) *op* ≤ *op* < **.**
  show *ds.fd x y* ≡ *fdia x y*
    by (*auto simp*: *fdia-def ds.fd-def*)
qed

lemma *fd-eq-fdia* [*simp*]: *domain-semiringl.fd* (*op* ·) *d x y* ≡ *fdia x y*
proof −
  have *class.domain-semiringl* (*op* +) (*op* ·) *1 0 d* (*op* ≤) (*op* <)
    by (*unfold-locales*, *auto simp*: *ads-d-def local.ans-d-def*)
  hence *domain-semiringl.fd* (*op* ·) *d x y* = *d* ((*op* ·) *x y*)
    by (*rule domain-semiringl.fd-def*)
  also have ... = *ds.fd x y*
    by (*simp add*: *ds.fd-def ads-d-def*)
  finally show *domain-semiringl.fd op* · *d x y* ≡ |*x*⟩ *y*

by *auto*
qed

end

class *antidomain-semiring* = *antidomain-semiringl* + *semiring-one-zero*

begin

Every antidomain semiring is an antidomain monoid.

subclass *antidomain-monoid*
by (*standard*, *metis ans1 mult-1-right annir*)

lemma *a-zero* = *0*
by (*simp add*: *local.a-zero-def*)

sublocale *ds*: *domain-semiring op* + *op* · *1 0* λx. *ad* (*ad x*) *op* ≤ *op* <
rewrites *ds.fd x y* ≡ *fdia x y*
by *unfold-locales*

end

3.5 The Boolean Algebra of Domain Elements

typedef (**overloaded**) ′*a a2-element* = {*x* :: ′*a* :: *antidomain-semiring*. *x* = *d x*}
by (*rule-tac x=1* **in** *exI*, *auto simp*: *ads-d-def*)

setup-lifting *type-definition-a2-element*

instantiation *a2-element* :: (*antidomain-semiring*) *boolean-algebra*

begin

lift-definition *less-eq-a2-element* :: ′*a a2-element* ⇒ ′*a a2-element* ⇒ *bool* **is** *op* ≤ **.**

lift-definition *less-a2-element* :: ′*a a2-element* ⇒ ′*a a2-element* ⇒ *bool* **is** *op* < **.**

lift-definition *bot-a2-element* :: ′*a a2-element* **is** *0*
by (*simp add*: *ads-d-def*)

lift-definition *top-a2-element* :: ′*a a2-element* **is** *1*
by (*simp add*: *ads-d-def*)

lift-definition *inf-a2-element* :: ′*a a2-element* ⇒ ′*a a2-element* ⇒ ′*a a2-element* **is** *op* ·
by (*metis* (*no-types*, *lifting*) *ads-d-def dpdz.dom-mult-closed*)

lift-definition *sup-a2-element* :: ′*a a2-element* ⇒ ′*a a2-element* ⇒ ′*a a2-element*

is *op* +
  by (*metis ads-d-def ds.dsr5*)

lift-definition *minus-a2-element* :: *'a a2-element* ⇒ *'a a2-element* ⇒ *'a a2-element*
is $\lambda x\ y.\ x \cdot ad\ y$
  by (*metis* (*no-types*, *lifting*) *ads-d-def dpdz.domain-export''*)

lift-definition *uminus-a2-element* :: *'a a2-element* ⇒ *'a a2-element* **is** *antidomain-op*
  by (*simp add*: *ads-d-def*)

instance
  apply (*standard*; *transfer*)
  apply (*simp add*: *less-le-not-le*)
  apply *simp*
  apply *auto*[*1*]
  apply *simp*
  apply (*metis a-subid-aux2 ads-d-def*)
  apply (*metis a-subid-aux1' ads-d-def*)
  apply (*metis* (*no-types*, *lifting*) *ads-d-def dpdz.dom-glb*)
  apply *simp*
  apply *simp*
  apply *simp*
  apply *simp*
  apply (*metis a-subid' ads-d-def*)
  apply (*metis* (*no-types*, *lifting*) *ads-d-def dpdz.dom-distrib*)
  apply (*metis ads-d-def ans1*)
  apply (*metis ads-d-def ans3*)
  by *simp*

end

3.6 Further Properties

context *antidomain-semiringl*

begin

lemma *a-2-var*: $ad\ x \cdot d\ y = 0 \longleftrightarrow ad\ x \leq ad\ y$
  using *local.a-gla local.ads-d-def local.dpdz.dom-weakly-local* **by** *auto*

The following two lemmas give the Galois connection of Heyting algebras.

lemma *da-shunt1*: $x \leq d\ y + z \Longrightarrow x \cdot ad\ y \leq z$
proof −
  assume $x \leq d\ y + z$
  hence $x \cdot ad\ y \leq (d\ y + z) \cdot ad\ y$
    using *mult-isor* **by** *blast*
  also have ... $= d\ y \cdot ad\ y + z \cdot ad\ y$
    by *simp*
  also have ... $\leq z$

    by (*simp add*: *a-subid-aux2 ads-d-def*)
  finally show $x \cdot ad\ y \leq z$
    by *simp*
qed

lemma *da-shunt2*: $x \leq ad\ y + z \Longrightarrow x \cdot d\ y \leq z$
  using *da-shunt1 local.a-add-idem local.ads-d-def am-add-op-def* **by** *auto*

lemma *d-a-galois1*: $d\ x \cdot ad\ y \leq d\ z \longleftrightarrow d\ x \leq d\ z + d\ y$
  by (*metis add-assoc local.a-gla local.ads-d-def local.am2 local.ans4 local.ans-d-def local.dnsz.dnso4*)

lemma *d-a-galois2*: $d\ x \cdot d\ y \leq d\ z \longleftrightarrow d\ x \leq d\ z + ad\ y$
proof −
  have $\bigwedge a\ aa.\ ad\ ((a::'a) \cdot ad\ (ad\ aa)) = ad\ (a \cdot aa)$
    using *local.a2-eq′ local.apd-d-def* **by** *force*
  then show *?thesis*
    by (*metis d-a-galois1 local.a-export′ local.ads-d-def local.ans-d-def*)
qed

lemma *d-cancellation-1*: $d\ x \leq d\ y + d\ x \cdot ad\ y$
proof −
  have *a*: $d\ (d\ x \cdot ad\ y) = ad\ y \cdot d\ x$
    using *local.a-closure′ local.ads-d-def local.am2 local.ans-d-def* **by** *auto*
  hence $d\ x \leq d\ (d\ x \cdot ad\ y) + d\ y$
    using *d-a-galois1 local.a-comm-var local.ads-d-def* **by** *fastforce*
  thus *?thesis*
    using *a add-commute local.ads-d-def local.am2* **by** *auto*
qed

lemma *d-cancellation-2*: $(d\ z + d\ y) \cdot ad\ y \leq d\ z$
  by (*simp add*: *da-shunt1*)

lemma *a-de-morgan*: $ad\ (ad\ x \cdot ad\ y) = d\ (x + y)$
  by (*simp add*: *local.ads-d-def*)

lemma *a-de-morgan-var-3*: $ad\ (d\ x + d\ y) = ad\ x \cdot ad\ y$
  using *local.a-add-idem local.ads-d-def am-add-op-def* **by** *auto*

lemma *a-de-morgan-var-4*: $ad\ (d\ x \cdot d\ y) = ad\ x + ad\ y$
  using *local.a-add-idem local.ads-d-def am-add-op-def* **by** *auto*

lemma *a-4*: $ad\ x \leq ad\ (x \cdot y)$
  using *local.a-add-idem local.a-antitone′ local.dpdz.domain-1′′ am-add-op-def* **by** *fastforce*

lemma *a-6*: $ad\ (d\ x \cdot y) = ad\ x + ad\ y$
  using *a-de-morgan-var-4 local.ads-d-def* **by** *auto*

lemma *a-7*: *d x* · *ad* (*d y* + *d z*) = *d x* · *ad y* · *ad z*
using *a-de-morgan-var-3 local.mult.semigroup-axioms semigroup.assoc* **by** *fastforce*

lemma *a-d-add-closure* [*simp*]: *d* (*ad x* + *ad y*) = *ad x* + *ad y*
using *local.a-add-idem local.ads-d-def am-add-op-def* **by** *auto*

lemma *d-6* [*simp*]: *d x* + *ad x* · *d y* = *d x* + *d y*
proof −
have *ad* (*ad x* · (*x* + *ad y*)) = *d* (*x* + *y*)
by (*simp add*: *distrib-left ads-d-def*)
thus *?thesis*
by (*simp add*: *local.ads-d-def local.ans-d-def*)
qed

lemma *d-7* [*simp*]: *ad x* + *d x* · *ad y* = *ad x* + *ad y*
by (*metis a-d-add-closure local.ads-d-def local.ans4 local.s4*)

lemma *a-mult-add*: *ad x* · (*y* + *x*) = *ad x* · *y*
by (*simp add*: *distrib-left*)

lemma *kat-2*: *y* · *ad z* ≤ *ad x* · *y* ⟹ *d x* · *y* · *ad z* = *0*
proof −
assume *a*: *y* · *ad z* ≤ *ad x* · *y*
hence *d x* · *y* · *ad z* ≤ *d x* · *ad x* · *y*
using *local.mult-isol mult-assoc* **by** *presburger*
thus *?thesis*
using *local.join.le-bot ads-d-def* **by** *auto*
qed

lemma *kat-3*: *d x* · *y* · *ad z* = *0* ⟹ *d x* · *y* = *d x* · *y* · *d z*
using *local.a-zero-def local.ads-d-def local.am-d-def local.kat-3'* **by** *auto*

lemma *kat-4*: *d x* · *y* = *d x* · *y* · *d z* ⟹ *d x* · *y* ≤ *y* · *d z*
using *a-subid-aux1 mult-assoc ads-d-def* **by** *auto*

lemma *kat-2-equiv*: *y* · *ad z* ≤ *ad x* · *y* ⟷ *d x* · *y* · *ad z* = *0*
proof
assume *y* · *ad z* ≤ *ad x* · *y*
thus *d x* · *y* · *ad z* = *0*
by (*simp add*: *kat-2*)
next
assume *1*: *d x* · *y* · *ad z* = *0*
have *y* · *ad z* = (*d x* + *ad x*) · *y* · *ad z*
by (*simp add*: *local.ads-d-def*)
also have ... = *d x* · *y* · *ad z* + *ad x* · *y* · *ad z*
using *local.distrib-right* **by** *presburger*
also have ... = *ad x* · *y* · *ad z*
using *1* **by** *auto*
also have ... ≤ *ad x* · *y*

by (*simp add*: *local.a-subid-aux2*)
finally show $y \cdot ad\ z \leq ad\ x \cdot y$ **.**
qed

lemma *kat-4-equiv*: $d\ x \cdot y = d\ x \cdot y \cdot d\ z \longleftrightarrow d\ x \cdot y \leq y \cdot d\ z$
using *local.ads-d-def local.dpdz.d-preserves-equation* **by** *auto*

lemma *kat-3-equiv-opp*: $ad\ z \cdot y \cdot d\ x = 0 \longleftrightarrow y \cdot d\ x = d\ z \cdot y \cdot d\ x$
proof −
have $ad\ z \cdot (y \cdot d\ x) = 0 \longrightarrow (ad\ z \cdot y \cdot d\ x = 0) = (y \cdot d\ x = d\ z \cdot y \cdot d\ x)$
by (*metis* (*no-types*, *hide-lams*) *add-commute local.add-zerol local.ads-d-def local.as3 local.distrib-right′ local.mult-1-left mult-assoc*)
thus *?thesis*
by (*metis a-4 local.a-add-idem local.a-gla2 local.ads-d-def mult-assoc am-add-op-def*)
qed

lemma *kat-4-equiv-opp*: $y \cdot d\ x = d\ z \cdot y \cdot d\ x \longleftrightarrow y \cdot d\ x \leq d\ z \cdot y$
using *kat-2-equiv kat-3-equiv-opp local.ads-d-def* **by** *auto*

3.7 Forward Box and Diamond Operators

lemma *fdemodalisation22*: $|x\rangle\ y \leq d\ z \longleftrightarrow ad\ z \cdot x \cdot d\ y = 0$
proof −
have $|x\rangle\ y \leq d\ z \longleftrightarrow d\ (x \cdot y) \leq d\ z$
by (*simp add*: *fdia-def ads-d-def*)
also have ... $\longleftrightarrow ad\ z \cdot d\ (x \cdot y) = 0$
by (*metis add-commute local.a-gla local.ads-d-def local.ans4*)
also have ... $\longleftrightarrow ad\ z \cdot x \cdot y = 0$
using *dpdz.dom-weakly-local mult-assoc ads-d-def* **by** *auto*
finally show *?thesis*
using *dpdz.dom-weakly-local ads-d-def* **by** *auto*
qed

lemma *dia-diff-var*: $|x\rangle\ y \leq |x\rangle\ (d\ y \cdot ad\ z) + |x\rangle\ z$
proof −
have *1*: $|x\rangle\ (d\ y \cdot d\ z) \leq |x\rangle\ (1 \cdot d\ z)$
using *dpdz.dom-glb-eq ds.fd-subdist fdia-def ads-d-def* **by** *force*
have $|x\rangle\ y = |x\rangle\ (d\ y \cdot (ad\ z + d\ z))$
by (*metis as3 add-comm ds.fdia-d-simp mult-1-right ads-d-def*)
also have ... $= |x\rangle\ (d\ y \cdot ad\ z) + |x\rangle\ (d\ y \cdot d\ z)$
by (*simp add*: *local.distrib-left local.ds.fdia-add1*)
also have ... $\leq |x\rangle\ (d\ y \cdot ad\ z) + |x\rangle\ (1 \cdot d\ z)$
using *1 local.join.sup.mono* **by** *blast*
finally show *?thesis*
by (*simp add*: *fdia-def ads-d-def*)
qed

lemma *dia-diff*: $|x\rangle\ y \cdot ad\ (\ |x\rangle\ z\ ) \leq |x\rangle\ (d\ y \cdot ad\ z)$
using *fdia-def dia-diff-var d-a-galois2 ads-d-def* **by** *metis*

lemma *fdia-export-2*: $ad\ y \cdot |x\rangle\ z = |ad\ y \cdot x\rangle\ z$
using *local.am-d-def local.d-a-export local.fdia-def mult-assoc* **by** *auto*

lemma *fdia-split*: $|x\rangle\ y = d\ z \cdot |x\rangle\ y + ad\ z \cdot |x\rangle\ y$
by (*metis mult-onel ans3 distrib-right ads-d-def*)

definition *fbox* :: $'a \Rightarrow 'a \Rightarrow 'a$ ((|-] -) [*61*,*81*] *82*) **where**
$|x]\ y = ad\ (x \cdot ad\ y)$

The next lemmas establish the De Morgan duality between boxes and diamonds.

lemma *fdia-fbox-de-morgan-2*: $ad\ (\ |x\rangle\ y) = |x]\ ad\ y$
using *fbox-def local.a-closure local.a-loc local.am-d-def local.fdia-def* **by** *auto*

lemma *fbox-simp*: $|x]\ y = |x]\ d\ y$
using *fbox-def local.a-add-idem local.ads-d-def am-add-op-def* **by** *auto*

lemma *fbox-dom* [*simp*]: $|x]\ 0 = ad\ x$
by (*simp add*: *fbox-def*)

lemma *fbox-add1*: $|x]\ (d\ y \cdot d\ z) = |x]\ y \cdot |x]\ z$
using *a-de-morgan-var-4 fbox-def local.distrib-left* **by** *auto*

lemma *fbox-add2*: $|x + y]\ z = |x]\ z \cdot |y]\ z$
by (*simp add*: *fbox-def*)

lemma *fbox-mult*: $|x \cdot y]\ z = |x]\ |y]\ z$
using *fbox-def local.a2-eq' local.apd-d-def mult-assoc* **by** *auto*

lemma *fbox-zero* [*simp*]: $|0]\ x = 1$
by (*simp add*: *fbox-def*)

lemma *fbox-one* [*simp*]: $|1]\ x = d\ x$
by (*simp add*: *fbox-def ads-d-def*)

lemma *fbox-iso*: $d\ x \le d\ y \Longrightarrow |z]\ x \le |z]\ y$
proof −
assume $d\ x \le d\ y$
hence $ad\ y \le ad\ x$
using *local.a-add-idem local.a-antitone' local.ads-d-def am-add-op-def* **by** *fastforce*
hence $z \cdot ad\ y \le z \cdot ad\ x$
by (*simp add*: *mult-isol*)
thus $|z]\ x \le |z]\ y$
by (*simp add*: *fbox-def a-antitone'*)
qed

lemma *fbox-antitone-var*: $x \le y \Longrightarrow |y]\ z \le |x]\ z$
by (*simp add*: *fbox-def a-antitone mult-isor*)

lemma *fbox-subdist-1*: $|x]$ $(d\ y \cdot d\ z) \leq |x]\ y$
using *a-de-morgan-var-4 fbox-def local.a-supdist-var local.distrib-left* **by** *force*

lemma *fbox-subdist-2*: $|x]\ y \leq|x]\ (d\ y + d\ z)$
by (*simp add*: *fbox-iso ads-d-def*)

lemma *fbox-subdist*: $|x]\ d\ y + |x]\ d\ z \leq |x]\ (d\ y + d\ z)$
by (*simp add*: *fbox-iso sup-least ads-d-def*)

lemma *fbox-diff-var*: $|x]\ (d\ y + ad\ z) \cdot |x]\ z \leq |x]\ y$
proof –
have $ad\ (ad\ y) \cdot ad\ (ad\ z) = ad\ (ad\ z + ad\ y)$
using *local.dpdz.dsg4* **by** *auto*
then have $d\ (d\ (d\ y + ad\ z) \cdot d\ z) \leq d\ y$
by (*simp add*: *local.a-subid-aux1′ local.ads-d-def*)
then show *?thesis*
by (*metis fbox-add1 fbox-iso*)
qed

lemma *fbox-diff*: $|x]\ (d\ y + ad\ z) \leq |x]\ y + ad\ (\ |x]\ z\ )$
proof –
have *f1*: $\bigwedge a.\ ad\ (ad\ (ad\ (a::'a))) = ad\ a$
using *local.a-closure′ local.ans-d-def* **by** *force*
have *f2*: $\bigwedge a\ aa.\ ad\ (ad\ (a::'a)) + ad\ aa = ad\ (ad\ a \cdot aa)$
using *local.ans-d-def* **by** *auto*
have *f3*: $\bigwedge a\ aa.\ ad\ ((a::'a) + aa) = ad\ (aa + a)$
by (*simp add*: *local.am2*)
then have *f4*: $\bigwedge a\ aa.\ ad\ (ad\ (ad\ (a::'a) \cdot aa)) = ad\ (ad\ aa + a)$
using *f2 f1* **by** (*metis* (*no-types*) *local.ans4*)
have *f5*: $\bigwedge a\ aa\ ab.\ ad\ ((a::'a) \cdot (aa + ab)) = ad\ (a \cdot (ab + aa))$
using *f3 local.distrib-left* **by** *presburger*
have *f6*: $\bigwedge a\ aa.\ ad\ (ad\ (ad\ (a::'a) + aa)) = ad\ (ad\ aa \cdot a)$
using *f3 f1* **by** *fastforce*
have $ad\ (x \cdot ad\ (y + ad\ z)) \leq ad\ (ad\ (x \cdot ad\ z) \cdot (x \cdot ad\ y))$
using *f5 f2 f1* **by** (*metis* (*no-types*) *a-mult-add fbox-def fbox-subdist-1 local.a-gla2 local.ads-d-def local.ans4 local.distrib-left local.gla-1 mult-assoc*)
then show *?thesis*
using *f6 f4 f3 f1* **by** (*simp add*: *fbox-def local.ads-d-def*)
qed

end

context *antidomain-semiring*

begin

lemma *kat-1*: $d\ x \cdot y \leq y \cdot d\ z \Longrightarrow y \cdot ad\ z \leq ad\ x \cdot y$
proof –

assume *a*: $d\ x \cdot y \le y \cdot d\ z$
have $y \cdot ad\ z = d\ x \cdot y \cdot ad\ z + ad\ x \cdot y \cdot ad\ z$
by (*metis local.ads-d-def local.as3 local.distrib-right local.mult-1-left*)
also have ... $\le y \cdot (d\ z \cdot ad\ z) + ad\ x \cdot y \cdot ad\ z$
by (*metis a add-iso mult-isor mult-assoc*)
also have ... $= ad\ x \cdot y \cdot ad\ z$
by (*simp add*: *ads-d-def*)
finally show $y \cdot ad\ z \le ad\ x \cdot y$
using *local.a-subid-aux2 local.dual-order.trans* **by** *blast*
qed

lemma *kat-1-equiv*: $d\ x \cdot y \le y \cdot d\ z \longleftrightarrow y \cdot ad\ z \le ad\ x \cdot y$
using *kat-1 kat-2 kat-3 kat-4* **by** *blast*

lemma *kat-3-equiv'*: $d\ x \cdot y \cdot ad\ z = 0 \longleftrightarrow d\ x \cdot y = d\ x \cdot y \cdot d\ z$
by (*simp add*: *kat-1-equiv local.kat-2-equiv local.kat-4-equiv*)

lemma *kat-1-equiv-opp*: $y \cdot d\ x \le d\ z \cdot y \longleftrightarrow ad\ z \cdot y \le y \cdot ad\ x$
by (*metis kat-1-equiv local.a-closure' local.ads-d-def local.ans-d-def*)

lemma *kat-2-equiv-opp*: $ad\ z \cdot y \le y \cdot ad\ x \longleftrightarrow ad\ z \cdot y \cdot d\ x = 0$
by (*simp add*: *kat-1-equiv-opp local.kat-3-equiv-opp local.kat-4-equiv-opp*)

lemma *fbox-one-1* [*simp*]: $|x]\ 1 = 1$
by (*simp add*: *fbox-def*)

lemma *fbox-demodalisation3*: $d\ y \le |x]\ d\ z \longleftrightarrow d\ y \cdot x \le x \cdot d\ z$
by (*simp add*: *fbox-def a-gla kat-2-equiv-opp mult-assoc ads-d-def*)

end

3.8 Antidomain Kleene Algebras

class *antidomain-left-kleene-algebra* = *antidomain-semiringl* + *left-kleene-algebra-zerol*

begin

sublocale *dka*: *domain-left-kleene-algebra op* + *op* $\cdot$ *1 0 d op* $\le$ *op* $<$ *star*
rewrites *domain-semiringl.fd op* $\cdot$ *d x y* $\equiv |x\rangle\ y$
by (*unfold-locales*, *auto simp add*: *local.ads-d-def ans-d-def*)

lemma *a-star* [*simp*]: $ad\ (x^\star) = 0$
using *dka.dom-star local.a-very-costrict'' local.ads-d-def* **by** *force*

lemma *fbox-star-unfold* [*simp*]: $|1]\ z \cdot |x]\ |x^\star]\ z = |x^\star]\ z$
proof −
have $ad\ (ad\ z + x \cdot (x^\star \cdot ad\ z)) = ad\ (x^\star \cdot ad\ z)$
using *local.conway.dagger-unfoldl-distr mult-assoc* **by** *auto*
then show *?thesis*

using *local.a-closure′ local.ans-d-def local.fbox-def local.fdia-def local.fdia-fbox-de-morgan-2*
by *fastforce*
qed

lemma *fbox-star-unfold-var* [*simp*]: $d\ z \cdot |x]\ |x^\star]\ z = |x^\star]\ z$
using *fbox-star-unfold* **by** *auto*

lemma *fbox-star-unfoldr* [*simp*]: $|1]\ z \cdot |x^\star]\ |x]\ z = |x^\star]\ z$
by (*metis fbox-star-unfold fbox-mult star-slide-var*)

lemma *fbox-star-unfoldr-var* [*simp*]: $d\ z \cdot |x^\star]\ |x]\ z = |x^\star]\ z$
using *fbox-star-unfoldr* **by** *auto*

lemma *fbox-star-induct-var*: $d\ y \leq |x]\ y \Longrightarrow d\ y \leq |x^\star]\ y$
proof −
assume *a1*: $d\ y \leq |x]\ y$
have $\bigwedge a.\ ad\ (ad\ (ad\ (a::'a))) = ad\ a$
using *local.a-closure′ local.ans-d-def* **by** *auto*
then have $ad\ (ad\ (x^\star \cdot ad\ y)) \leq ad\ y$
using *a1* **by** (*metis dka.fdia-star-induct local.a-export′ local.ads-d-def local.ans4 local.ans-d-def local.eq-refl local.fbox-def local.fdia-def local.meet-ord-def*)
then have $ad\ (ad\ y + ad\ (x^\star \cdot ad\ y)) = zero\text{-}class.zero$
by (*metis* (*no-types*) *add-commute local.a-2-var local.ads-d-def local.ans4*)
then show *?thesis*
using *local.a-2-var local.ads-d-def local.fbox-def* **by** *auto*
qed

lemma *fbox-star-induct*: $d\ y \leq d\ z \cdot |x]\ y \Longrightarrow d\ y \leq |x^\star]\ z$
proof −
assume *a1*: $d\ y \leq d\ z \cdot |x]\ y$
hence *a*: $d\ y \leq d\ z$ **and** $d\ y \leq |x]\ y$
apply (*metis local.a-subid-aux2 local.dual-order.trans local.fbox-def*)
using *a1 dka.dom-subid-aux2 local.dual-order.trans* **by** *blast*
hence $d\ y \leq |x^\star]\ y$
using *fbox-star-induct-var* **by** *blast*
thus *?thesis*
using *a local.fbox-iso local.order.trans* **by** *blast*
qed

lemma *fbox-star-induct-eq*: $d\ z \cdot |x]\ y = d\ y \Longrightarrow d\ y \leq |x^\star]\ z$
by (*simp add*: *fbox-star-induct*)

lemma *fbox-export-1*: $ad\ y + |x]\ y = |d\ y \cdot x]\ y$
by (*simp add*: *local.a-6 local.fbox-def mult-assoc*)

lemma *fbox-export-2*: $d\ y + |x]\ y = |ad\ y \cdot x]\ y$
by (*simp add*: *local.ads-d-def local.ans-d-def local.fbox-def mult-assoc*)

end

class *antidomain-kleene-algebra* = *antidomain-semiring* + *kleene-algebra*

begin

subclass *antidomain-left-kleene-algebra* **..**

lemma $d\ p \leq |(d\ t \cdot x)^{\star} \cdot ad\ t]\ (d\ q \cdot ad\ t) \Longrightarrow d\ p \leq |d\ t \cdot x]\ d\ q$

oops

end

end

4 Range and Antirange Semirings

theory *Range-Semiring*
imports *Antidomain-Semiring*
begin

4.1 Range Semirings

We set up the duality between domain and antidomain semirings on the one hand and range and antirange semirings on the other hand.

class *range-op* =
  fixes *range-op* :: $'a \Rightarrow 'a\ (r)$

class *range-semiring* = *semiring-one-zero* + *plus-ord* + *range-op* +
  assumes *rsr1* [*simp*]: $x + (x \cdot r\ x) = x \cdot r\ x$
  and *rsr2* [*simp*]: $r\ (r\ x \cdot y) = r(x \cdot y)$
  and *rsr3* [*simp*]: $r\ x + 1 = 1$
  and *rsr4* [*simp*]: $r\ 0 = 0$
  and *rsr5* [*simp*]: $r\ (x + y) = r\ x + r\ y$

begin

definition *bd* :: $'a \Rightarrow 'a \Rightarrow 'a$ (⟨-| - [*61*,*81*] *82*) **where**
  $\langle x|\ y = r\ (y \cdot x)$

no-notation *range-op* (*r*)

end

sublocale *range-semiring* ⊆ *rdual*: *domain-semiring op* + $\lambda x\ y.\ y \cdot x$ *1 0 range-op op* ≤ *op* <
  rewrites *rdual.fd* $x\ y = \langle x|\ y$
proof −

show *class.domain-semiring op* + ($\lambda x\ y.\ y \cdot x$) *1 0 range-op op* ≤ *op* <
by (*standard*, *auto simp*: *mult-assoc distrib-left*)
then interpret *rdual*: *domain-semiring op* + $\lambda x\ y.\ y \cdot x$ *1 0 range-op op* ≤ *op* < **.**
show *rdual.fd x y* = ⟨*x*| *y*
unfolding *rdual.fd-def bd-def* **by** *auto*
qed

sublocale *domain-semiring* ⊆ *ddual*: *range-semiring op* + $\lambda x\ y.\ y \cdot x$ *1 0 domain-op op* ≤ *op* <
rewrites *ddual.bd x y* = *domain-semiringl-class.fd x y*
proof −
show *class.range-semiring op* + ($\lambda x\ y.\ y \cdot x$) *1 0 domain-op op* ≤ *op* <
by (*standard*, *auto simp*: *mult-assoc distrib-left*)
then interpret *ddual*: *range-semiring op* + $\lambda x\ y.\ y \cdot x$ *1 0 domain-op op* ≤ *op* < **.**
show *ddual.bd x y* = *domain-semiringl-class.fd x y*
unfolding *ddual.bd-def fd-def* **by** *auto*
qed

4.2 Antirange Semirings

class *antirange-op* =
fixes *antirange-op* :: *'a* ⇒ *'a* (*ar* - [*999*] *1000*)

class *antirange-semiring* = *semiring-one-zero* + *plus-ord* + *antirange-op* +
assumes *ars1* [*simp*]: $x \cdot ar\ x = 0$
and *ars2* [*simp*]: $ar\ (x \cdot y) + ar\ (ar\ ar\ x \cdot y) = ar\ (ar\ ar\ x \cdot y)$
and *ars3* [*simp*]: $ar\ ar\ x + ar\ x = 1$

begin

no-notation *bd* (⟨-| - [*61*,*81*] *82*)

definition *ars-r* :: *'a* ⇒ *'a* (*r*) **where**
$r\ x = ar\ (ar\ x)$

definition *bdia* :: *'a* ⇒ *'a* ⇒ *'a* (⟨-| - [*61*,*81*] *82*) **where**
⟨*x*| $y = ar\ ar\ (y \cdot x)$

definition *bbox* :: *'a* ⇒ *'a* ⇒ *'a* ([-| - [*61*,*81*] *82*) **where**
[*x*| $y = ar\ (ar\ y \cdot x)$

end

sublocale *antirange-semiring* ⊆ *ardual*: *antidomain-semiring antirange-op op* + $\lambda x\ y.\ y \cdot x$ *1 0 op* ≤ *op* <
rewrites *ardual.ads-d x* = *r x*
and *ardual.fdia x y* = ⟨*x*| *y*

and *ardual.fbox x y* = [*x*| *y*
proof −
show *class.antidomain-semiring antirange-op op* + (λ*x y*. *y* · *x*) *1 0 op* ≤ *op* <
by (*standard*, *auto simp*: *mult-assoc distrib-left*)
then interpret *ardual*: *antidomain-semiring antirange-op op* + λ*x y*. *y* · *x 1 0 op* ≤ *op* < **.**
show *ardual.ads-d x* = *r x*
by (*simp add*: *ardual.ads-d-def local.ars-r-def*)
show *ardual.fdia x y* = ⟨*x*| *y*
unfolding *ardual.fdia-def bdia-def* **by** *auto*
show *ardual.fbox x y* = [*x*| *y*
unfolding *ardual.fbox-def bbox-def* **by** *auto*
qed

context *antirange-semiring*

begin

sublocale *rs*: *range-semiring op* + *op* · *1 0* λ*x*. *ar ar x op* ≤ *op* <
by *unfold-locales*

end

sublocale *antidomain-semiring* ⊆ *addual*: *antirange-semiring op* + λ*x y*. *y* · *x 1 0 antidomain-op op* ≤ *op* <
rewrites *addual.ars-r x* = *d x*
and *addual.bdia x y* = |*x*⟩ *y*
and *addual.bbox x y* = |*x*] *y*
proof −
show *class.antirange-semiring op* + (λ*x y*. *y* · *x*) *1 0 antidomain-op op* ≤ *op* <
by (*standard*, *auto simp*: *mult-assoc distrib-left*)
then interpret *addual*: *antirange-semiring op* + λ*x y*. *y* · *x 1 0 antidomain-op op* ≤ *op* < **.**
show *addual.ars-r x* = *d x*
by (*simp add*: *addual.ars-r-def local.ads-d-def*)
show *addual.bdia x y* = |*x*⟩ *y*
unfolding *addual.bdia-def fdia-def* **by** *auto*
show *addual.bbox x y* = |*x*] *y*
unfolding *addual.bbox-def fbox-def* **by** *auto*
qed

4.3 Antirange Kleene Algebras

class *antirange-kleene-algebra* = *antirange-semiring* + *kleene-algebra*

sublocale *antirange-kleene-algebra* ⊆ *dual*: *antidomain-kleene-algebra antirange-op op* + λ*x y*. *y* · *x 1 0 op* ≤ *op* < *star*
by (*standard*, *auto simp*: *local.star-inductr′ local.star-inductl*)

```
sublocale antidomain-kleene-algebra ⊆ dual: antirange-kleene-algebra op + λx y.
y · x 1 0 op ≤ op < star antidomain-op
  by (standard, simp-all add: star-inductr star-inductl)
```

Hence all range theorems have been derived by duality in a generic way.

end

5 Modal Kleene Algebras

This section studies laws that relate antidomain and antirange semirings and Kleene algebra, notably Galois connections and conjugations as those studied in [13, 7].

```
theory Modal-Kleene-Algebra
imports Range-Semiring
begin

class modal-semiring = antidomain-semiring + antirange-semiring +
  assumes domrange [simp]: d (r x) = r x
  and rangedom [simp]: r (d x) = d x

begin
```

These axioms force that the domain algebra and the range algebra coincide.

```
lemma domrangefix: d x = x ⟷ r x = x
  by (metis domrange rangedom)

lemma box-diamond-galois-1:
assumes d p = p and d q = q
shows ⟨x| p ≤ q ⟷ p ≤ |x] q
proof −
  have ⟨x| p ≤ q ⟷ p · x ≤ x · q
    by (metis assms domrangefix local.ardual.ds.fdemodalisation2 local.ars-r-def)
  thus ?thesis
    by (metis assms fbox-demodalisation3)
qed

lemma box-diamond-galois-2:
assumes d p = p and d q = q
shows |x⟩ p ≤ q ⟷ p ≤ [x| q
proof −
  have |x⟩ p ≤ q ⟷ x · p ≤ q · x
    by (metis assms local.ads-d-def local.ds.fdemodalisation2)
  thus ?thesis
    by (metis assms domrangefix local.ardual.fbox-demodalisation3)
qed
```

lemma *diamond-conjugation-var-1*:
assumes $d\ p = p$ **and** $d\ q = q$
shows $|x\rangle\ p \leq ad\ q \longleftrightarrow \langle x|\ q \leq ad\ p$
proof −
have $|x\rangle\ p \leq ad\ q \longleftrightarrow x \cdot p \leq ad\ q \cdot x$
by (*metis assms local.ads-d-def local.ds.fdemodalisation2*)
also have ... $\longleftrightarrow q \cdot x \leq x \cdot ad\ p$
by (*metis assms local.ads-d-def local.kat-1-equiv-opp*)
finally show *?thesis*
by (*metis assms box-diamond-galois-1 local.ads-d-def local.fbox-demodalisation3*)
qed

lemma *diamond-conjungation-aux*:
assumes $d\ p = p$ **and** $d\ q = q$
shows $\langle x|\ p \leq ad\ q \longleftrightarrow q \cdot \langle x|\ p = 0$
apply *standard*
apply (*metis assms(2) local.a-antitone′ local.a-gla local.ads-d-def*)
proof −
assume *a1*: $q \cdot \langle x|\ p =$ *zero-class.zero*
have $ad\ (ad\ q) = q$
using *assms(2) local.ads-d-def* **by** *fastforce*
then show $\langle x|\ p \leq ad\ q$
using *a1* **by** (*metis (no-types) domrangefix local.a-gla local.ads-d-def local.antisym local.ardual.a-gla2 local.ardual.gla-1 local.ars-r-def local.bdia-def local.eq-refl*)
qed

lemma *diamond-conjugation*:
assumes $d\ p = p$ **and** $d\ q = q$
shows $p \cdot |x\rangle\ q = 0 \longleftrightarrow q \cdot \langle x|\ p = 0$
proof −
have $p \cdot |x\rangle\ q = 0 \longleftrightarrow |x\rangle\ q \leq ad\ p$
by (*metis assms(1) local.a-gla local.ads-d-def local.am2 local.fdia-def*)
also have ... $\longleftrightarrow \langle x|\ p \leq ad\ q$
by (*simp add: assms diamond-conjugation-var-1*)
finally show *?thesis*
by (*simp add: assms diamond-conjungation-aux*)
qed

lemma *box-conjugation-var-1*:
assumes $d\ p = p$ **and** $d\ q = q$
shows $ad\ p \leq [x|\ q \longleftrightarrow ad\ q \leq |x]\ p$
by (*metis assms box-diamond-galois-1 box-diamond-galois-2 diamond-conjugation-var-1 local.ads-d-def*)

lemma *box-diamond-cancellation-1*: $d\ p = p \Longrightarrow p \leq |x]\ \langle x|\ p$
using *box-diamond-galois-1 domrangefix local.ars-r-def local.bdia-def* **by** *fastforce*

lemma *box-diamond-cancellation-2*: $d\ p = p \Longrightarrow p \leq [x|\ |x\rangle\ p$
by (*metis box-diamond-galois-2 local.ads-d-def local.dpdz.domain-invol local.eq-refl*

local.fdia-def)

lemma *box-diamond-cancellation-3*: $d\ p = p \Longrightarrow |x\rangle\ [x|\ p \leq p$
using *box-diamond-galois-2 domrangefix local.ardual.fdia-fbox-de-morgan-2 local.ars-r-def local.bbox-def local.bdia-def* **by** *auto*

lemma *box-diamond-cancellation-4*: $d\ p = p \Longrightarrow \langle x|\ |x]\ p \leq p$
using *box-diamond-galois-1 local.ads-d-def local.fbox-def local.fdia-def local.fdia-fbox-de-morgan-2* **by** *auto*

end

class *modal-kleene-algebra* = *modal-semiring* + *kleene-algebra*
begin

subclass *antidomain-kleene-algebra* **..**

subclass *antirange-kleene-algebra* **..**

end

end

6 Models of Modal Kleene Algebras

theory *Modal-Kleene-Algebra-Models*
imports *../Kleene-Algebra/Kleene-Algebra-Models*
Modal-Kleene-Algebra

begin

This section develops the relation model. We also briefly develop the trace model for antidomain Kleene algebras, but not for antirange or full modal Kleene algebras. The reason is that traces are implemented as lists; we therefore expect tedious inductive proofs in the presence of range. The language model is not particularly interesting.

definition *rel-ad* :: *'a rel* ⇒ *'a rel* **where**
rel-ad R = {(*x*,*x*) | *x*. ¬ (∃ *y*. (*x*,*y*) ∈ *R*)}

interpretation *rel-antidomain-kleene-algebra*: *antidomain-kleene-algebra rel-ad op* ∪ *op O Id* {} *op* ⊆ *op* ⊂ *rtrancl*
by *unfold-locales* (*auto simp*: *rel-ad-def*)

definition *trace-a* :: (*'p*, *'a*) *trace set* ⇒ (*'p*, *'a*) *trace set* **where**
trace-a X = {(*p*,[]) | *p*. ¬ (∃ *x*. *x* ∈ *X* ∧ *p* = *first x*)}

interpretation *trace-antidomain-kleene-algebra*: *antidomain-kleene-algebra trace-a op* ∪ *t-prod t-one t-zero op* ⊆ *op* ⊂ *t-star*

proof
 show ⋀*x*. *t-prod* (*trace-a x*) *x* = *t-zero*
  by (*auto simp*: *trace-a-def t-prod-def t-fusion-def t-zero-def*)
 show ⋀*x y*. *trace-a* (*t-prod x y*) ∪ *trace-a* (*t-prod x* (*trace-a* (*trace-a y*))) = *trace-a* (*t-prod x* (*trace-a* (*trace-a y*)))
  by (*auto simp*: *trace-a-def t-prod-def t-fusion-def*)
 show ⋀*x*. *trace-a* (*trace-a x*) ∪ *trace-a x* = *t-one*
  by (*auto simp*: *trace-a-def t-one-def*)
qed

The trace model should be extended to cover modal Kleene algebras in the future.

definition *rel-ar* :: *'a rel* ⇒ *'a rel* **where**
 rel-ar R = {(*y*,*y*) | *y*. ¬ (∃ *x*. (*x*,*y*) ∈ *R*)}

interpretation *rel-antirange-kleene-algebra*: *antirange-semiring op* ∪ *op O Id* {} *rel-ar op* ⊆ *op* ⊂
apply *unfold-locales*
apply (*simp-all add*: *rel-ar-def*)
by *auto*

interpretation *rel-modal-kleene-algebra*: *modal-kleene-algebra op* ∪ *op O Id* {} *op* ⊆ *op* ⊂ *rtrancl rel-ad rel-ar*
apply *standard*
apply (*metis* (*no-types*, *lifting*) *rel-antidomain-kleene-algebra.a-d-closed rel-antidomain-kleene-algebra.a-one rel-antidomain-kleene-algebra.addual.ars-r-def rel-antidomain-kleene-algebra.am5-lem rel-antidomain-kleene-algebra.am6-lem rel-antidomain-kleene-algebra.apd-d-def rel-antidomain-kleene-algebra.a rel-antidomain-kleene-algebra.dpdz.dom-one rel-antirange-kleene-algebra.ardual.a-comm' rel-antirange-kleene-algebra.ardual.a-d-closed rel-antirange-kleene-algebra.ardual.a-mul-d' rel-antirange-kleene-algebra.ardual.a-mult-idem rel-antirange-kleene-algebra.ardual.a-zero rel-antirange-kleene-algebra.ardual.ads-d-def rel-antirange-kleene-algebra.ardual.am6-lem rel-antirange-kleene-algebra.ardual.apd-d-def rel-antirange-kleene-algebra.ardual.s4*)
by (*metis rel-antidomain-kleene-algebra.a-zero rel-antidomain-kleene-algebra.addual.ars1 rel-antidomain-kleene-algebra.addual.ars-r-def rel-antidomain-kleene-algebra.am5-lem rel-antidomain-kleene-algebra.am6-lem rel-antidomain-kleene-algebra.ds.ddual.mult-oner rel-antidomain-kleene-algebra.s4 rel-antirange-kleene-algebra.ardual.ads-d-def rel-antirange-kleene-algebra.ardu rel-antirange-kleene-algebra.ardual.apd1 rel-antirange-kleene-algebra.ardual.dpdz.dns1''*)

end

7 Applications of Modal Kleene Algebras

theory *Modal-Kleene-Algebra-Applications*
imports *Antidomain-Semiring*
begin

This file collects some applications of the theories developed so far. These are described in [11].

context *antidomain-kleene-algebra*
begin

7.1 A Reachability Result

This example is taken from [4].

lemma *opti-iterate-var* [*simp*]: $|(ad\ y \cdot x)^\star\rangle\ y = |x^\star\rangle\ y$
proof (*rule antisym*)
  show $|(ad\ y \cdot x)^\star\rangle\ y \le |x^\star\rangle\ y$
    by (*simp add*: *a-subid-aux1′ ds.fd-iso2 star-iso*)
  have $d\ y + |x\rangle\ |(ad\ y \cdot x)^\star\rangle\ y = d\ y + ad\ y \cdot |x\rangle\ |(ad\ y \cdot x)^\star\rangle\ y$
    using *local.ads-d-def local.d-6 local.fdia-def* **by** *auto*
  also have ... $= d\ y + |ad\ y \cdot x\rangle\ |(ad\ y \cdot x)^\star\rangle\ y$
    by (*simp add*: *fdia-export-2*)
  finally have $d\ y + |x\rangle\ |(ad\ y \cdot x)^\star\rangle\ y = |(ad\ y \cdot x)^\star\rangle\ y$
    by *simp*
  thus $|x^\star\rangle\ y \le |(ad\ y \cdot x)^\star\rangle\ y$
    using *local.dka.fd-def local.dka.fdia-star-induct-eq* **by** *auto*
qed

lemma *opti-iterate* [*simp*]: $d\ y + |(x \cdot ad\ y)^\star\rangle\ |x\rangle\ y = |x^\star\rangle\ y$
proof −
  have $d\ y + |(x \cdot ad\ y)^\star\rangle\ |x\rangle\ y = d\ y + |x\rangle\ |(ad\ y \cdot x)^\star\rangle\ y$
    by (*metis local.conway.dagger-slide local.ds.fdia-mult*)
  also have ... $= d\ y + |x\rangle\ |x^\star\rangle\ y$
    by *simp*
  finally show *?thesis*
    by *force*
qed

lemma *opti-iterate-var-2* [*simp*]: $d\ y + |ad\ y \cdot x\rangle\ |x^\star\rangle\ y = |x^\star\rangle\ y$
  by (*metis local.dka.fdia-star-unfold-var opti-iterate-var*)

7.2 Derivation of Segerberg's Formula

This example is taken from [5].

definition *Alpha* :: $'a \Rightarrow 'a \Rightarrow 'a\ (A)$
  where $A\ x\ y = d\ (x \cdot y) \cdot ad\ y$

lemma *A-dom* [*simp*]: $d\ (A\ x\ y) = A\ x\ y$
  using *Alpha-def local.a-d-closed local.ads-d-def local.apd-d-def* **by** *auto*

lemma *A-fdia*: $A\ x\ y = |x\rangle\ y \cdot ad\ y$
  by (*simp add*: *Alpha-def local.dka.fd-def*)

lemma *A-fdia-var*: $A\ x\ y = |x\rangle\ d\ y \cdot ad\ y$
  by (*simp add*: *A-fdia*)

lemma *a-A*: $ad\ (A\ x\ (ad\ y)) = |x]\ y + ad\ y$
  using *Alpha-def local.a-6 local.a-d-closed local.apd-d-def local.fbox-def* **by** *force*

lemma *fsegerberg* [*simp*]: $d\ y + |x^\star\rangle\ A\ x\ y = |x^\star\rangle\ y$
proof (*rule antisym*)
  have $d\ y + |x\rangle\ (d\ y + |x^\star\rangle\ A\ x\ y) = d\ y + |x\rangle\ y + |x\rangle\ |x^\star\rangle\ (\ |x\rangle\ y \cdot ad\ y\ )$
    by (*simp add*: *A-fdia add-assoc local.ds.fdia-add1*)
  also have ... $= d\ y + |x\rangle\ y \cdot ad\ y + |x\rangle\ |x^\star\rangle\ (\ |x\rangle\ y \cdot ad\ y\ )$
    by (*metis local.am2 local.d-6 local.dka.fd-def local.fdia-def*)
  finally have $d\ y + |x\rangle\ (d\ y + |x^\star\rangle\ A\ x\ y) = d\ y + |x^\star\rangle\ A\ x\ y$
    by (*metis A-dom A-fdia add-assoc local.dka.fdia-star-unfold-var*)
  thus $|x^\star\rangle\ y \leq d\ y + |x^\star\rangle\ A\ x\ y$
    by (*metis local.a-d-add-closure local.ads-d-def local.dka.fdia-star-induct-eq local.fdia-def*)
  have $d\ y + |x^\star\rangle\ A\ x\ y = d\ y + |x^\star\rangle\ (\ |x\rangle\ y \cdot ad\ y\ )$
    by (*simp add*: *A-fdia*)
  also have ... $\leq d\ y + |x^\star\rangle\ |x\rangle\ y$
    using *local.dpdz.domain-1''* *local.ds.fd-iso1 local.join.sup-mono* **by** *blast*
  finally show $d\ y + |x^\star\rangle\ A\ x\ y \leq |x^\star\rangle\ y$
    by *simp*
qed

lemma *fbox-segerberg* [*simp*]: $d\ y \cdot |x^\star]\ (\ |x]\ y + ad\ y\ ) = |x^\star]\ y$
proof −
  have $|x^\star]\ (\ |x]\ y + ad\ y\ ) = d\ (\ |x^\star]\ (\ |x]\ y + ad\ y\ )\ )$
    using *local.a-d-closed local.ads-d-def local.apd-d-def local.fbox-def* **by** *auto*
  hence *f1*: $|x^\star]\ (\ |x]\ y + ad\ y\ ) = ad\ (\ |x^\star\rangle\ (A\ x\ (ad\ y))\ )$
    by (*simp add*: *a-A local.fdia-fbox-de-morgan-2*)
  have $ad\ y + |x^\star\rangle\ (A\ x\ (ad\ y)) = |x^\star\rangle\ ad\ y$
    by (*metis fsegerberg local.a-d-closed local.ads-d-def local.apd-d-def*)
  thus *?thesis*
    by (*metis f1 local.ads-d-def local.ans4 local.fbox-simp local.fdia-fbox-de-morgan-2*)
qed

7.3 Wellfoundedness and Loeb's Formula

This example is taken from [7].

definition *Omega* :: $'a \Rightarrow 'a \Rightarrow 'a$ (Ω)
  where $\Omega\ x\ y = d\ y \cdot ad\ (x \cdot y)$

If y is a set, then $\Omega(x, y)$ describes those elements in y from which no further x transitions are possible.

lemma *omega-fdia*: $\Omega\ x\ y = d\ y \cdot ad\ (\ |x\rangle\ y\ )$
  using *Omega-def local.a-d-closed local.ads-d-def local.apd-d-def local.dka.fd-def* **by** *auto*

lemma *omega-fbox*: $\Omega\ x\ y = d\ y \cdot |x]\ (ad\ y)$
  by (*simp add*: *fdia-fbox-de-morgan-2 omega-fdia*)

lemma *omega-absorb1* [*simp*]: $\Omega\ x\ y \cdot ad\ (\ |x\rangle\ y\ ) = \Omega\ x\ y$
  by (*simp add*: *mult-assoc omega-fdia*)

lemma *omega-absorb2* [*simp*]: $\Omega\ x\ y \cdot ad\ (x \cdot y) = \Omega\ x\ y$
  by (*simp add*: *Omega-def mult-assoc*)

lemma *omega-le-1*: $\Omega\ x\ y \leq d\ y$
  by (*simp add*: *Omega-def d-a-galois1*)

lemma *omega-subid*: $\Omega\ x\ (d\ y) \leq d\ y$
  by (*simp add*: *Omega-def local.a-subid-aux2*)

lemma *omega-le-2*: $\Omega\ x\ y \leq ad\ (\ |x\rangle\ y\ )$
  by (*simp add*: *local.dka.dom-subid-aux2 omega-fdia*)

lemma *omega-dom* [*simp*]: $d\ (\Omega\ x\ y) = \Omega\ x\ y$
  using *Omega-def local.a-d-closed local.ads-d-def local.apd-d-def* **by** *auto*

lemma *a-omega*: $ad\ (\Omega\ x\ y) = ad\ y + |x\rangle\ y$
  by (*simp add*: *Omega-def local.a-6 local.ds.fd-def*)

lemma *omega-fdia-3* [*simp*]: $d\ y \cdot ad\ (\Omega\ x\ y) = d\ y \cdot |x\rangle\ y$
  using *Omega-def local.ads-d-def local.fdia-def local.s4* **by** *presburger*

lemma *omega-zero-equiv-1*: $\Omega\ x\ y = 0 \longleftrightarrow d\ y \leq |x\rangle\ y$
  by (*simp add*: *Omega-def local.a-gla local.ads-d-def local.fdia-def*)

definition *Loebian* :: $'a \Rightarrow bool$
  where $Loebian\ x = (\forall y.\ |x\rangle\ y \leq |x\rangle\ \Omega\ x\ y)$

definition *PreLoebian* :: $'a \Rightarrow bool$
  where $PreLoebian\ x = (\forall y.\ d\ y \leq |x^\star\rangle\ \Omega\ x\ y)$

definition *Noetherian* :: $'a \Rightarrow bool$
  where $Noetherian\ x = (\forall y.\ \Omega\ x\ y = 0 \longrightarrow d\ y = 0)$

lemma *noetherian-alt*: $Noetherian\ x \longleftrightarrow (\forall y.\ d\ y \leq |x\rangle\ y \longrightarrow d\ y = 0)$
  by (*simp add*: *Noetherian-def omega-zero-equiv-1*)

lemma *Noetherian-iff-PreLoebian*: $Noetherian\ x \longleftrightarrow PreLoebian\ x$
proof
  assume *hyp*: *Noetherian x*
  {
    fix *y*
    have $d\ y \cdot ad\ (\ |x^\star\rangle\ \Omega\ x\ y\ ) = d\ y \cdot ad\ (\Omega\ x\ y + |x\rangle\ |x^\star\rangle\ \Omega\ x\ y)$
      by (*metis local.dka.fdia-star-unfold-var omega-dom*)
    also have ... $= d\ y \cdot ad\ (\Omega\ x\ y) \cdot ad\ (\ |x\rangle\ |x^\star\rangle\ \Omega\ x\ y\ )$
      using *ans4 mult-assoc* **by** *presburger*

also have ... $\leq |x\rangle\ d\ y \cdot ad\ (\ |x\rangle\ |x^\star\rangle\ \Omega\ x\ y\ )$
by (*simp add*: *local.dka.dom-subid-aux2 local.mult-isor*)
also have ... $\leq |x\rangle\ (d\ y \cdot ad\ (\ |x^\star\rangle\ \Omega\ x\ y\ ))$
by (*simp add*: *local.dia-diff*)
finally have $d\ y \cdot ad\ (\ |x^\star\rangle\ \Omega\ x\ y\ ) \leq |x\rangle\ (d\ y \cdot ad\ (\ |x^\star\rangle\ \Omega\ x\ y\ ))$
by *blast*
hence $d\ y \cdot ad\ (\ |x^\star\rangle\ \Omega\ x\ y\ ) = 0$
by (*metis hyp local.ads-d-def local.dpdz.dom-mult-closed local.fdia-def noetherian-alt*)
hence $d\ y \leq |x^\star\rangle\ \Omega\ x\ y$
by (*simp add*: *local.a-gla local.ads-d-def local.dka.fd-def*)
}
thus *PreLoebian x*
using *PreLoebian-def* **by** *blast*
next
assume *a*: *PreLoebian x*
{
fix *y*
assume *b*: $\Omega\ x\ y = 0$
hence $d\ y \leq |x\rangle\ d\ y$
using *omega-zero-equiv-1* **by** *auto*
hence $d\ y = 0$
by (*metis* (*no-types*) *PreLoebian-def a b a-one a-zero add-zeror annir fdia-def less-eq-def*)
}
thus *Noetherian x*
by (*simp add*: *Noetherian-def*)
qed

lemma *Loebian-imp-Noetherian*: *Loebian x* $\Longrightarrow$ *Noetherian x*
proof −
assume *a*: *Loebian x*
{
fix *y*
assume *b*: $\Omega\ x\ y = 0$
hence $d\ y \leq |x\rangle\ d\ y$
using *omega-zero-equiv-1* **by** *auto*
also have ... $\leq |x\rangle\ (\Omega\ x\ y)$
using *Loebian-def a* **by** *auto*
finally have $d\ y = 0$
by (*simp add*: *b local.antisym local.fdia-def*)
}
thus *Noetherian x*
by (*simp add*: *Noetherian-def*)
qed

lemma *d-transitive*: $(\forall\, y.\ |x\rangle\ |x\rangle\ y \leq |x\rangle\ y) \Longrightarrow (\forall\, y.\ |x\rangle\ y = |x^\star\rangle\ |x\rangle\ y)$
proof −
assume *a*: $\forall\, y.\ |x\rangle\ |x\rangle\ y \leq |x\rangle\ y$
{

fix y
have $|x\rangle\ y + |x\rangle\ |x\rangle\ y \leq |x\rangle\ y$
by (*simp add*: a)
hence $|x^{\star}\rangle\ |x\rangle\ y \leq |x\rangle\ y$
using *local.dka.fd-def local.dka.fdia-star-induct-var* **by** *auto*
have $|x\rangle\ y \leq |x^{\star}\rangle\ |x\rangle\ y$
using *local.dka.fd-def local.order-prop opti-iterate* **by** *force*
}
thus *?thesis*
using *a local.antisym local.dka.fd-def local.dka.fdia-star-induct-var* **by** *auto*
qed

lemma *d-transitive-var*: $(\forall\, y.\ |x\rangle\ |x\rangle\ y \leq |x\rangle\ y) \Longrightarrow (\forall\, y.\ |x\rangle\ y = |x\rangle\ |x^{\star}\rangle\ y)$
proof –
assume $\forall\, y.\ |x\rangle\ |x\rangle\ y \leq |x\rangle\ y$
then have $\forall\, a.\ |x\rangle\ |x^{\star}\rangle\ a = |x\rangle\ a$
by (*metis* (*full-types*) *d-transitive local.dka.fd-def local.dka.fdia-d-simp local.star-slide-var mult-assoc*)
then show *?thesis*
by *presburger*
qed

lemma *d-transitive-PreLoebian-imp-Loebian*: $(\forall\, y.\ |x\rangle\ |x\rangle\ y \leq |x\rangle\ y) \Longrightarrow$ *PreLoebian* $x \Longrightarrow$ *Loebian* x
proof –
assume *wt*: $(\forall\, y.\ |x\rangle\ |x\rangle\ y \leq |x\rangle\ y)$
and *PreLoebian* x
hence $\forall\, y.\ |x\rangle\ y \leq |x\rangle\ |x^{\star}\rangle\ \Omega\ x\ y$
using *PreLoebian-def local.ads-d-def local.dka.fd-def local.ds.fd-iso1* **by** *auto*
hence $\forall\, y.\ |x\rangle\ y \leq |x\rangle\ \Omega\ x\ y$
by (*metis wt d-transitive-var*)
then show *Loebian* x
using *Loebian-def fdia-def* **by** *auto*
qed

lemma *d-transitive-Noetherian-iff-Loebian*: $\forall\, y.\ |x\rangle\ |x\rangle\ y \leq |x\rangle\ y \Longrightarrow$ *Noetherian* $x \longleftrightarrow$ *Loebian* x
using *Loebian-imp-Noetherian Noetherian-iff-PreLoebian d-transitive-PreLoebian-imp-Loebian* **by** *blast*

lemma *Loeb-iff-box-Loeb*: *Loebian* $x \longleftrightarrow (\forall\, y.\ |x]\ (ad\ (\ |x]\ y\ ) + d\ y) \leq |x]\ y)$
proof –
have *Loebian* $x \longleftrightarrow (\forall\, y.\ |x\rangle\ y \leq |x\rangle\ (d\ y \cdot |x]\ (ad\ y)))$
using *Loebian-def omega-fbox* **by** *auto*
also have ... $\longleftrightarrow (\forall\, y.\ ad\ (\ |x\rangle\ (d\ y \cdot |x]\ (ad\ y))\ ) \leq ad\ (\ |x\rangle\ y\ ))$
using *a-antitone′ fdia-def* **by** *fastforce*
also have ... $\longleftrightarrow (\forall\, y.\ |x]\ ad\ (d\ y \cdot |x]\ (ad\ y)) \leq |x]\ (ad\ y))$
by (*simp add*: *fdia-fbox-de-morgan-2*)
also have ... $\longleftrightarrow (\forall\, y.\ |x]\ (d\ (ad\ y) + ad\ (\ |x]\ (ad\ y)\ )) \leq |x]\ (ad\ y))$

by (*simp add*: *local.ads-d-def local.fbox-def*)
finally show *?thesis*
by (*metis add-commute local.a-d-closed local.ads-d-def local.apd-d-def local.fbox-def*)
qed

end

7.4 Divergence Kleene Algebras and Separation of Termination

The notion of divergence has been added to modal Kleene algebras in [5]. More facts about divergence could be added in the future. Some could be adapted from omega algebras.

class *nabla-op* =
fixes *nabla* :: $'a \Rightarrow 'a$ (∇- [*999*] *1000*)

class *fdivergence-kleene-algebra* = *antidomain-kleene-algebra* + *nabla-op* +
assumes *nabla-closure* [*simp*]: $d\ \nabla\ x = \nabla\ x$
and *nabla-unfold*: $\nabla\ x \leq |x\rangle\ \nabla\ x$
and *nabla-coinduction*: $d\ y \leq |x\rangle\ y + d\ z \Longrightarrow d\ y \leq \nabla\ x + |x^\star\rangle\ z$

begin

lemma *nabla-coinduction-var*: $d\ y \leq |x\rangle\ y \Longrightarrow d\ y \leq \nabla\ x$
proof −
assume $d\ y \leq |x\rangle\ y$
hence $d\ y \leq |x\rangle\ y + d\ 0$
by *simp*
hence $d\ y \leq \nabla\ x + |x^\star\rangle\ 0$
using *nabla-coinduction* **by** *blast*
thus $d\ y \leq \nabla\ x$
by (*simp add*: *fdia-def*)
qed

lemma *nabla-unfold-eq* [*simp*]: $|x\rangle\ \nabla\ x = \nabla\ x$
proof −
have *f1*: $ad\ (ad\ (x \cdot \nabla x)) = ad\ (ad\ (x \cdot \nabla x)) + \nabla x$
using *local.ds.fd-def local.join.sup.order-iff local.nabla-unfold* **by** *presburger*
then have $ad\ (ad\ (x \cdot \nabla x)) \cdot ad\ (ad\ \nabla x) = \nabla x$
by (*metis local.ads-d-def local.dpdz.dns5 local.dpdz.dsg4 local.join.sup-commute local.nabla-closure*)
then show *?thesis*
using *f1* **by** (*metis local.ads-d-def local.ds.fd-def local.ds.fd-subdist-2 local.join.sup.order-iff local.join.sup-commute nabla-coinduction-var*)
qed

lemma *nabla-subdist*: $\nabla\ x \leq \nabla\ (x + y)$
proof −

have $d \nabla x \leq \nabla (x + y)$
by (*metis local.ds.fd-iso2 local.join.sup.cobounded1 local.nabla-closure nabla-coinduction-var nabla-unfold-eq*)
thus *?thesis*
by *simp*
qed

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

lemma *nabla-omega*: $\Omega\ x\ (d\ y) = 0 \Longrightarrow d\ y \leq \nabla x$
using *omega-zero-equiv-1 nabla-coinduction-var* **by** *fastforce*

lemma *nabla-noether*: $\nabla x = 0 \Longrightarrow$ *Noetherian x*
using *local.join.le-bot local.noetherian-alt nabla-coinduction-var* **by** *blast*

lemma *nabla-preloeb*: $\nabla x = 0 \Longrightarrow$ *PreLoebian x*
using *Noetherian-iff-PreLoebian nabla-noether* **by** *auto*

lemma *star-nabla-1* [*simp*]: $|x^\star\rangle \nabla x = \nabla x$
proof (*rule antisym*)
show $|x^\star\rangle \nabla x \leq \nabla x$
by (*metis local.dka.fdia-star-induct-var local.eq-iff local.nabla-closure nabla-unfold-eq*)
show $\nabla x \leq |x^\star\rangle \nabla x$
by (*metis local.ds.fd-iso2 local.star-ext nabla-unfold-eq*)
qed

lemma *nabla-sum-expand* [*simp*]: $|x\rangle \nabla (x + y) + |y\rangle \nabla (x + y) = \nabla (x + y)$
proof −
have $d\ ((x + y) \cdot \nabla(x + y)) = \nabla(x + y)$
using *local.dka.fd-def nabla-unfold-eq* **by** *presburger*
thus *?thesis*
by (*simp add*: *local.dka.fd-def*)
qed

lemma *wagner-3*:
assumes $d\ z + |x\rangle \nabla (x + y) = \nabla (x + y)$
shows $\nabla (x + y) = \nabla x + |x^\star\rangle z$
proof (*rule antisym*)
have $d\ \nabla(x + y) \leq d\ z + |x\rangle \nabla(x + y)$
by (*simp add*: *assms*)
thus $\nabla (x + y) \leq \nabla x + |x^\star\rangle z$
by (*metis add-commute nabla-closure nabla-coinduction*)
have $d\ z + |x\rangle \nabla (x + y) \leq \nabla (x + y)$
using *assms* **by** *auto*
hence $|x^\star\rangle z \leq \nabla (x + y)$
by (*metis local.dka.fdia-star-induct local.nabla-closure*)
thus $\nabla x + |x^\star\rangle z \leq \nabla (x + y)$
by (*simp add*: *sup-least nabla-subdist*)

qed

lemma *nabla-sum-unfold* [*simp*]: $\nabla\ x + |x^\star\rangle\ |y\rangle\ \nabla\ (x + y) = \nabla\ (x + y)$
proof –
  have $\nabla\ (x + y) = |x\rangle\ \nabla\ (x + y) + |y\rangle\ \nabla\ (x + y)$
    by *simp*
  thus *?thesis*
   by (*metis add-commute local.dka.fd-def local.ds.fd-def local.ds.fdia-d-simp wagner-3*)
qed

lemma *nabla-separation*: $y \cdot x \leq x \cdot (x + y)^\star \Longrightarrow (\nabla\ (x + y) = \nabla\ x + |x^\star\rangle\ \nabla\ y)$
proof (*rule antisym*)
  assume *quasi-comm*: $y \cdot x \leq x \cdot (x + y)^\star$
  hence *a*: $y^\star \cdot x \leq x \cdot (x + y)^\star$
    using *quasicomm-var* **by** *blast*
  have $\nabla\ (x + y) = \nabla\ y + |y^\star \cdot x\rangle\ \nabla\ (x + y)$
    by (*metis local.ds.fdia-mult local.join.sup-commute nabla-sum-unfold*)
  also have ... $\leq \nabla\ y + |x \cdot (x + y)^\star\rangle\ \nabla\ (x + y)$
    using *a local.ds.fd-iso2 local.join.sup.mono* **by** *blast*
  also have ... $= \nabla\ y + |x\rangle\ |(x + y)^\star\rangle\ \nabla\ (x + y)$
    by (*simp add*: *fdia-def mult-assoc*)
  finally have $\nabla\ (x + y) \leq \nabla\ y + |x\rangle\ \nabla\ (x + y)$
    by (*metis star-nabla-1*)
  thus $\nabla\ (x + y) \leq \nabla\ x + |x^\star\rangle\ \nabla\ y$
    by (*metis add-commute nabla-closure nabla-coinduction*)
next
  have $\nabla\ x + |x^\star\rangle\ \nabla\ y = \nabla\ x + |x^\star\rangle\ |y\rangle\ \nabla\ y$
    by *simp*
  also have ... $= \nabla\ x + |x^\star \cdot y\rangle\ \nabla\ y$
    by (*simp add*: *fdia-def mult-assoc*)
  also have ... $\leq \nabla\ x + |x^\star \cdot y\rangle\ \nabla\ (x + y)$
   using *dpdz.dom-iso eq-refl fdia-def join.sup-ge2 join.sup-mono mult-isol nabla-iso*
by *presburger*
  also have ... $= \nabla\ x + |x^\star\rangle\ |y\rangle\ \nabla\ (x + y)$
    by (*simp add*: *fdia-def mult-assoc*)
  finally show $\nabla\ x + |x^\star\rangle\ \nabla\ y \leq \nabla\ (x + y)$
    by (*metis nabla-sum-unfold*)
qed

The next lemma is a separation of termination theorem by Bachmair and Dershowitz [2].

lemma *bachmair-dershowitz*: $y \cdot x \leq x \cdot (x + y)^\star \Longrightarrow \nabla\ x + \nabla\ y = 0 \longleftrightarrow \nabla\ (x + y) = 0$
proof –
  assume *quasi-comm*: $y \cdot x \leq x \cdot (x + y)^\star$
  have $\forall x.\ |x\rangle\ 0 = 0$
    by (*simp add*: *fdia-def*)
  hence $\nabla y = 0 \Longrightarrow \nabla x + \nabla y = 0 \longleftrightarrow \nabla(x + y) = 0$

using *quasi-comm nabla-separation* **by** *presburger*
thus *?thesis*
using *add-commute local.join.le-bot nabla-subdist* **by** *fastforce*
qed

The next lemma is a more complex separation of termination theorem by Doornbos, Backhouse and van der Woude [8].

lemma *separation-of-termination*:
assumes $y \cdot x \leq x \cdot (x + y)^\star + y$
shows $\nabla\ x + \nabla\ y = 0 \longleftrightarrow \nabla\ (x + y) = 0$
proof
assume *xy-wf*: $\nabla\ x + \nabla\ y = 0$
hence *x-preloeb*: $\nabla\ (x + y) \leq |x^\star\rangle\ \Omega\ x\ (\nabla\ (x + y))$
by (*metis PreLoebian-def no-trivial-inverse nabla-closure nabla-preloeb*)
hence *y-div*: $\nabla\ y = 0$
using *add-commute no-trivial-inverse xy-wf* **by** *blast*
have $\nabla\ (x + y) \leq |y\rangle\ \nabla\ (x + y) + |x\rangle\ \nabla\ (x + y)$
by (*simp add*: *local.join.sup-commute*)
hence $\nabla\ (x + y) \cdot ad\ (\ |x\rangle\ \nabla\ (x + y)\ ) \leq |y\rangle\ \nabla(x + y)$
by (*simp add*: *local.da-shunt1 local.dka.fd-def local.join.sup-commute*)
hence $\Omega\ x\ \nabla(x + y) \leq |y\rangle\ \nabla\ (x + y)$
by (*simp add*: *fdia-def omega-fdia*)
also have ... $\leq |y\rangle\ |x^\star\rangle\ (\Omega\ x\ \nabla(x + y))$
using *local.dpdz.dom-iso local.ds.fd-iso1 x-preloeb* **by** *blast*
also have ... $= |y \cdot x^\star\rangle\ (\Omega\ x\ \nabla(x + y))$
by (*simp add*: *fdia-def mult-assoc*)
also have ... $\leq |x \cdot (x + y)^\star + y\rangle\ (\Omega\ x\ \nabla(x + y))$
using *assms local.ds.fd-iso2 local.lazycomm-var* **by** *blast*
also have ... $= |x \cdot (x + y)^\star\rangle\ (\Omega\ x\ \nabla(x + y)) + |y\rangle\ (\Omega\ x\ \nabla(x + y))$
by (*simp add*: *local.dka.fd-def*)
also have ... $\leq |(x \cdot (x + y)^\star)\rangle\ \nabla(x + y) + |y\rangle\ (\Omega\ x\ \nabla(x + y))$
using *local.add-iso local.dpdz.domain-1′′ local.ds.fd-iso1 local.omega-fdia* **by** *auto*
also have ... $\leq |x\rangle\ |(x + y)^\star\rangle\ \nabla(x + y) + |y\rangle\ (\Omega\ x\ \nabla(x + y))$
by (*simp add*: *fdia-def mult-assoc*)
finally have $\Omega\ x\ \nabla(x + y) \leq |x\rangle\ \nabla(x + y) + |y\rangle\ (\Omega\ x\ \nabla(x + y))$
by (*metis star-nabla-1*)
hence $\Omega\ x\ \nabla(x + y) \cdot ad\ (\ |x\rangle\ \nabla(x + y)\ ) \leq |y\rangle\ (\Omega\ x\ \nabla(x + y))$
by (*simp add*: *local.da-shunt1 local.dka.fd-def*)
hence $\Omega\ x\ \nabla(x + y) \leq |y\rangle\ (\Omega\ x\ \nabla(x + y))$
by (*simp add*: *omega-fdia mult-assoc*)
hence $(\Omega\ x\ \nabla(x + y)) = 0$
by (*metis noetherian-alt omega-dom nabla-noether y-div*)
thus $\nabla\ (x + y) = 0$
using *local.dka.fd-def local.join.le-bot x-preloeb* **by** *auto*
next
assume $\nabla\ (x + y) = 0$
thus $(\nabla\ x) + (\nabla\ y) = 0$
by (*metis local.join.le-bot local.join.sup.order-iff local.join.sup-commute nabla-subdist*)

qed

The final examples can be found in [11].

definition T :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ (- ⇝ - ⇝ - [*61*,*61*,*61*] *60*)
  where $p{\leadsto}x{\leadsto}q \equiv ad\ p + |x]\ d\ q$

lemma *T-d* [*simp*]: $d\ (p{\leadsto}x{\leadsto}q) = p{\leadsto}x{\leadsto}q$
  using *T-def local.a-d-add-closure local.fbox-def* **by** *auto*

lemma *T-p*: $d\ p \cdot (p{\leadsto}x{\leadsto}q) = d\ p \cdot |x]\ d\ q$
proof −
  have $d\ p \cdot (p{\leadsto}x{\leadsto}q) = ad\ (ad\ p + ad\ (p{\leadsto}x{\leadsto}q))$
    using *T-d local.ads-d-def* **by** *auto*
  thus *?thesis*
    using *T-def add-commute local.a-mult-add local.ads-d-def* **by** *auto*
qed

lemma *T-a* [*simp*]: $ad\ p \cdot (p{\leadsto}x{\leadsto}q) = ad\ p$
  by (*metis T-d T-def local.a-d-closed local.ads-d-def local.apd-d-def local.dpdz.dns5 local.join.sup.left-idem*)

lemma *T-seq*: $(p{\leadsto}x{\leadsto}q) \cdot |x](q{\leadsto}y{\leadsto}s) \leq p{\leadsto}x \cdot y{\leadsto}s$
proof −
  have *f1*: $|x]\ q = |x]\ d\ q$
    using *local.fbox-simp* **by** *auto*
  have $ad\ p \cdot ad\ (x \cdot ad\ (q{\leadsto}y{\leadsto}s)) + |x]\ d\ q \cdot |x]\ (ad\ q + |y]\ d\ s) \leq ad\ p + |x]\ d\ q \cdot |x]\ (ad\ q + |y]\ d\ s)$
    using *local.a-subid-aux2 local.add-iso* **by** *blast*
  hence $(p{\leadsto}x{\leadsto}q) \cdot |x](q{\leadsto}y{\leadsto}s) \leq ad\ p + |x](d\ q \cdot (q{\leadsto}y{\leadsto}s))$
    by (*metis T-d T-def f1 local.distrib-right′ local.fbox-add1 local.fbox-def*)
  also have ... $= ad\ p + |x](d\ q \cdot (ad\ q + |y]\ d\ s))$
    by (*simp add*: *T-def*)
  also have ... $= ad\ p + |x](d\ q \cdot |y]\ d\ s)$
    using *T-def T-p* **by** *auto*
  also have ... $\leq ad\ p + |x]\ |y]\ d\ s$
    by (*metis* (*no-types*, *lifting*) *local.dka.dom-subid-aux2 local.dka.dsg3 local.eq-iff local.fbox-iso local.join.sup.mono*)
  finally show *?thesis*
    by (*simp add*: *T-def fbox-mult*)
qed

lemma *T-square*: $(p{\leadsto}x{\leadsto}q) \cdot |x](q{\leadsto}x{\leadsto}p) \leq p{\leadsto}x{\cdot}x{\leadsto}p$
  by (*simp add*: *T-seq*)

lemma *T-segerberg* [*simp*]: $d\ p \cdot |x^\star](p{\leadsto}x{\leadsto}p) = |x^\star]\ d\ p$
  using *T-def add-commute local.fbox-segerberg local.fbox-simp* **by** *force*

lemma *lookahead* [*simp*]: $|x^\star](d\ p \cdot |x]\ d\ p) = |x^\star]\ d\ p$
  by (*metis* (*full-types*) *local.dka.dsg3 local.fbox-add1 local.fbox-mult local.fbox-simp*

local.fbox-star-unfold-var local.star-slide-var local.star-trans-eq)

lemma *alternation*: $d\ p \cdot |x^\star]((p{\leadsto}x{\leadsto}q) \cdot (q{\leadsto}x{\leadsto}p)) = |(x{\cdot}x)^\star](d\ p \cdot (q{\leadsto}x{\leadsto}p)) \cdot |x{\cdot}(x{\cdot}x)^\star](d\ q \cdot (p{\leadsto}x{\leadsto}q))$
proof −
have *fbox-simp-2*: $\bigwedge x\ p.\ |x]p = d(\ |x]\ p\ )$
using *local.a-d-closed local.ads-d-def local.apd-d-def local.fbox-def* **by** *fastforce*
have $|(x{\cdot}x)^\star]((p{\leadsto}x{\leadsto}q) \cdot |x](q{\leadsto}x{\leadsto}p) \cdot (q{\leadsto}x{\leadsto}p) \cdot |x](p{\leadsto}x{\leadsto}q)) \leq |(x{\cdot}x)^\star]((p{\leadsto}x{\leadsto}q) \cdot |x](q{\leadsto}x{\leadsto}p))$
using *local.dka.domain-1′′ local.fbox-iso local.order-trans* **by** *blast*
also have ... $\leq |(x{\cdot}x)^\star](p{\leadsto}x{\cdot}x{\leadsto}p)$
using *T-seq local.dka.dom-iso local.fbox-iso* **by** *blast*
finally have *1*: $|(x{\cdot}x)^\star]((p{\leadsto}x{\leadsto}q) \cdot |x](q{\leadsto}x{\leadsto}p) \cdot (q{\leadsto}x{\leadsto}p) \cdot |x](p{\leadsto}x{\leadsto}q)) \leq |(x{\cdot}x)^\star](p{\leadsto}x{\cdot}x{\leadsto}p)$.
have $d\ p \cdot |x^\star]((p{\leadsto}x{\leadsto}q) \cdot (q{\leadsto}x{\leadsto}p)) = d\ p \cdot |1{+}x]\ |(x{\cdot}x)^\star]((p{\leadsto}x{\leadsto}q) \cdot (q{\leadsto}x{\leadsto}p))$
by (*metis* (*full-types*) *fbox-mult meyer-1*)
also have ... $= d\ p \cdot |(x{\cdot}x)^\star]((p{\leadsto}x{\leadsto}q) \cdot (q{\leadsto}x{\leadsto}p)) \cdot |x{\cdot}(x{\cdot}x)^\star]((p{\leadsto}x{\leadsto}q) \cdot (q{\leadsto}x{\leadsto}p))$
using *fbox-simp-2 fbox-mult fbox-add2 mult-assoc* **by** *auto*
also have ... $= d\ p \cdot |(x{\cdot}x)^\star]((p{\leadsto}x{\leadsto}q) \cdot (q{\leadsto}x{\leadsto}p)) \cdot |(x{\cdot}x)^\star{\cdot}x]((p{\leadsto}x{\leadsto}q) \cdot (q{\leadsto}x{\leadsto}p))$
by (*simp add*: *star-slide*)
also have ... $= d\ p \cdot |(x{\cdot}x)^\star]((p{\leadsto}x{\leadsto}q) \cdot (q{\leadsto}x{\leadsto}p)) \cdot |(x{\cdot}x)^\star]\ |x]((p{\leadsto}x{\leadsto}q) \cdot (q{\leadsto}x{\leadsto}p))$
by (*simp add*: *fbox-mult*)
also have ... $= d\ p \cdot |(x{\cdot}x)^\star]((p{\leadsto}x{\leadsto}q) \cdot (q{\leadsto}x{\leadsto}p) \cdot |x]((p{\leadsto}x{\leadsto}q) \cdot (q{\leadsto}x{\leadsto}p)))$
by (*metis T-d fbox-simp-2 local.dka.dom-mult-closed local.fbox-add1 mult-assoc*)
also have ... $= d\ p \cdot |(x{\cdot}x)^\star]((p{\leadsto}x{\leadsto}q) \cdot |x](q{\leadsto}x{\leadsto}p) \cdot (q{\leadsto}x{\leadsto}p) \cdot |x](p{\leadsto}x{\leadsto}q))$
proof −
have *f1*: $d\ ((q \leadsto x \leadsto p) \cdot |x]\ (p \leadsto x \leadsto q)) = (q \leadsto x \leadsto p) \cdot |x]\ (p \leadsto x \leadsto q)$
by (*metis* (*full-types*) *T-d fbox-simp-2 local.dka.dsg3*)
then have $|(x \cdot x)^\star]\ (d\ (\ |x]\ (q \leadsto x \leadsto p)) \cdot ((q \leadsto x \leadsto p) \cdot |x]\ (p \leadsto x \leadsto q))) = |(x \cdot x)^\star]\ d\ (\ |x]\ (q \leadsto x \leadsto p)) \cdot |(x \cdot x)^\star]\ ((q \leadsto x \leadsto p) \cdot |x]\ (p \leadsto x \leadsto q))$
by (*metis* (*full-types*) *fbox-simp-2 local.fbox-add1*)
then have *f2*: $|(x \cdot x)^\star]\ (d\ (\ |x]\ (q \leadsto x \leadsto p)) \cdot ((q \leadsto x \leadsto p) \cdot |x]\ (p \leadsto x \leadsto q))) = ad\ ((x \cdot x)^\star \cdot ad\ ((q \leadsto x \leadsto p) \cdot |x]\ (p \leadsto x \leadsto q)) + (x \cdot x)^\star \cdot ad\ (d\ (\ |x]\ (q \leadsto x \leadsto p))))$
by (*simp add*: *add-commute local.fbox-def*)
have $d\ (\ |x]\ (p \leadsto x \leadsto q)) \cdot d\ (\ |x]\ (q \leadsto x \leadsto p)) = |x]\ ((p \leadsto x \leadsto q) \cdot (q \leadsto x \leadsto p))$
by (*metis* (*no-types*) *T-d fbox-simp-2 local.fbox-add1*)
then have $d\ ((q \leadsto x \leadsto p) \cdot |x]\ (p \leadsto x \leadsto q)) \cdot d\ (d\ (\ |x]\ (q \leadsto x \leadsto p))) = (q \leadsto x \leadsto p) \cdot |x]\ ((p \leadsto x \leadsto q) \cdot (q \leadsto x \leadsto p))$
using *f1 fbox-simp-2 mult-assoc* **by** *force*
then have $|(x \cdot x)^\star]\ (d\ (\ |x]\ (q \leadsto x \leadsto p)) \cdot ((q \leadsto x \leadsto p) \cdot |x]\ (p \leadsto x \leadsto q))) = |(x \cdot x)^\star]\ ((q \leadsto x \leadsto p) \cdot |x]\ ((p \leadsto x \leadsto q) \cdot (q \leadsto x \leadsto p)))$

using *f2* **by** (*metis* (*no-types*) *local.ans4 local.fbox-add1 local.fbox-def*)
then show *?thesis*
by (*metis* (*no-types*) *T-d fbox-simp-2 local.dka.dsg3 local.fbox-add1 mult-assoc*)
qed
also have ... $= d\ p \cdot |(x{\cdot}x)^\star](p{\leadsto}x{\cdot}x{\leadsto}p) \cdot |(x{\cdot}x)^\star]((p{\leadsto}x{\leadsto}q) \cdot |x](q{\leadsto}x{\leadsto}p) \cdot (q{\leadsto}x{\leadsto}p) \cdot |x](p{\leadsto}x{\leadsto}q))$ **using** *1*
by (*metis fbox-simp-2 local.dka.dns5 local.dka.dsg4 local.join.sup.absorb2 mult-assoc*)
also have ... $= |(x{\cdot}x)^\star](d\ p \cdot (p{\leadsto}x{\leadsto}q) \cdot |x](q{\leadsto}x{\leadsto}p) \cdot (q{\leadsto}x{\leadsto}p) \cdot |x](p{\leadsto}x{\leadsto}q))$
using *T-segerberg local.a-d-closed local.ads-d-def local.apd-d-def local.distrib-left local.fbox-def mult-assoc* **by** *auto*
also have ... $= |(x{\cdot}x)^\star](d\ p \cdot |x]\ d\ q \cdot |x](q{\leadsto}x{\leadsto}p) \cdot (q{\leadsto}x{\leadsto}p) \cdot |x](p{\leadsto}x{\leadsto}q))$
by (*simp add*: *T-p*)
also have ... $= |(x{\cdot}x)^\star](d\ p \cdot |x]\ d\ q \cdot |x]\ |x]\ d\ p \cdot (q{\leadsto}x{\leadsto}p) \cdot |x](p{\leadsto}x{\leadsto}q))$
by (*metis T-d T-p fbox-simp-2 fbox-add1 fbox-simp mult-assoc*)
also have ... $= |(x{\cdot}x)^\star](d\ p \cdot |x{\cdot}x]\ d\ p \cdot (q{\leadsto}x{\leadsto}p) \cdot |x]\ d\ q \cdot |x](p{\leadsto}x{\leadsto}q))$
proof −
have *f1*: $ad\ (x \cdot ad\ (\ |x]\ d\ p)) = |x \cdot x]\ d\ p$
using *local.fbox-def local.fbox-mult* **by** *presburger*
have *f2*: $ad\ (d\ q \cdot d\ (x \cdot ad\ (d\ p))) = q \leadsto x \leadsto p$
by (*simp add*: *T-def local.a-de-morgan-var-4 local.fbox-def*)
have $ad\ q + |x]\ d\ p = ad\ (d\ q \cdot d\ (x \cdot ad\ (d\ p)))$
by (*simp add*: *local.a-de-morgan-var-4 local.fbox-def*)
then have $ad\ (x \cdot ad\ (\ |x]\ d\ p)) \cdot ((q \leadsto x \leadsto p) \cdot |x]\ d\ q) = ad\ (x \cdot ad\ (\ |x]\ d\ p)) \cdot ad\ (x \cdot ad\ (d\ q)) \cdot (ad\ q + |x]\ d\ p)$
using *f2* **by** (*metis* (*no-types*) *local.am2 local.fbox-def mult-assoc*)
then show *?thesis*
using *f1* **by** (*simp add*: *T-def local.am2 local.fbox-def mult-assoc*)
qed
also have ... $= |(x{\cdot}x)^\star](d\ p \cdot |x{\cdot}x]\ d\ p \cdot (q{\leadsto}x{\leadsto}p) \cdot |x](d\ q \cdot (p{\leadsto}x{\leadsto}q)))$
using *local.a-d-closed local.ads-d-def local.apd-d-def local.distrib-left local.fbox-def mult-assoc* **by** *auto*
also have ... $= |(x{\cdot}x)^\star](d\ p \cdot |x{\cdot}x]\ d\ p) \cdot |(x{\cdot}x)^\star](q{\leadsto}x{\leadsto}p) \cdot |(x{\cdot}x)^\star]\ |x](d\ q \cdot (p{\leadsto}x{\leadsto}q))$
by (*metis T-d fbox-simp-2 local.dka.dom-mult-closed local.fbox-add1*)
also have ... $= |(x{\cdot}x)^\star](d\ p \cdot (q{\leadsto}x{\leadsto}p)) \cdot |(x{\cdot}x)^\star]\ |x]\ (d\ q \cdot (p{\leadsto}x{\leadsto}q))$
by (*metis T-d local.fbox-add1 local.fbox-simp lookahead*)
finally show *?thesis*
by (*metis fbox-mult star-slide*)
qed

lemma $|(x{\cdot}x)^\star]\ d\ p \cdot |x{\cdot}(x{\cdot}x)^\star]\ ad\ p = d\ p \cdot |x^\star]((p{\leadsto}x{\leadsto}ad\ p) \cdot (ad\ p{\leadsto}x{\leadsto}p))$
using *alternation local.a-d-closed local.ads-d-def local.apd-d-def* **by** *auto*

lemma $|x^\star]\ d\ p = d\ p \cdot |x^\star](p{\leadsto}x{\leadsto}p)$
by (*simp add*: *alternation*)

end

end

References

[1] A. Armstrong, V. B. F. Gomes, G. Struth, and T. Weber. Kleene algebra. *Archive of Formal Proofs*, 2013.

[2] 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.

[3] J. Desharnais, P. Jipsen, and G. Struth. Domain and antidomain semigroups. In R. Berghammer, A. Jaoua, and B. Möller, editors, *Relations and Kleene Algebra in Computer Science*, volume 5827 of *Lecture Notes in Computer Science*, pages 73–87. Springer, 2009.

[4] J. Desharnais, B. Möller, and G. Struth. Kleene algebra with domain. *ACM TOCL*, 7(4):798–833, 2006.

[5] J. Desharnais, B. Möller, and G. Struth. Algebraic notions of termination. *Logical Methods in Computer Science*, 7(1), 2011.

[6] J. Desharnais and G. Struth. Domain axioms for a family of near-semirings. In J. Meseguer and G. Rosu, editors, *AMAST 2008*, volume 5140 of *Lecture Notes in Computer Science*, pages 330–345. Springer, 2008.

[7] J. Desharnais and G. Struth. Internal axioms for domain semirings. *Science of Computer Programming*, 76(3):181–203, 2011.

[8] H. Doornbos, R. C. Backhouse, and J. van der Woude. A calculational approach to mathematical induction. *Theoretical Computer Science*, 179(1–2):103–135, 1997.

[9] H. Furusawa and G. Struth. Binary multirelations. *Archive of Formal Proofs*, 2015.

[10] H. Furusawa and G. Struth. Concurrent dynamic algebra. *ACM TOCL*, 16(4):30, 2015.

[11] 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.

[12] R. D. Maddux. *Relation Algebras*. Elsevier, 2006.

[13] B. Möller and G. Struth. Algebras of modal operators and partial correctness. *Theoretical Computer Science*, 351(2):221–239, 2006.