Kleene Algebra with Tests and Demonic Refinement Algebras

Alasdair Armstrong    Victor B. F. Gomes    Georg Struth

September 19, 2015

Abstract

We formalise Kleene algebra with tests (KAT) and demonic refinement algebra (DRA) in Isabelle/HOL. KAT is relevant for program verification and correctness proofs in the partial correctness setting. While DRA targets similar applications in the context of total correctness. Our formalisation contains the two most important models of these algebras: binary relations in the case of KAT and predicate transformers in the case of DRA. In addition, we derive the inference rules for Hoare logic in KAT and its relational model and present a simple formally verified program verification tool prototype based on the algebraic approach.

Contents

1 Basis for Demonic Refinement Algebras 2
2 Demonic Refinement Algebras 4
3 Test Dioids 10
4 Kleene Algebra with Tests 21
5 Demonic Refinement Algebra with Tests 23
6 Models for Demonic Refinement Algebra with Tests 28
7 Models for Kleene Algebra with Tests 31
8 Pre-Conway Algebra 32
9 Transformation Theorem for while Loops 35
10 Hoare Logic 38
Demonic refinement algebra is based on a Kleene algebra without the right annihilation law $x \cdot 0 = 0$. In the Archive [1], only left Kleene algebras without the right annihilation law exist. So we need to define an expansion.

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

These lemmas were copied from AFP (Kleene Algebra). They are also valid without right annihilation.

```

```isar
lemma star-inductr-var: $y \cdot x \leq y \Rightarrow y \cdot x^* \leq y$
by (metis add-lub order-refl star-inductr)

```

```isar
lemma star-inductr-var-equiv: $y \cdot x \leq y \iff y \cdot x^* \leq y$
by (metis order-trans mult-isol star-ext star-inductr-var)

```

```isar
lemma star-sim2: $z \cdot x \leq y \cdot z \Rightarrow z \cdot x^* \leq y^* \cdot z$
proof
  assume $z \cdot x \leq y \cdot z$
  hence $y^* \cdot z \cdot x \leq y^* \cdot y \cdot z$
    by (metis distrib-left less-eq-def mult.assoc)
  also have $\ldots \leq y^* \cdot z$
    by (metis (full-types) mult-isol star-ll star-slide-var)
  hence $z + y^* \cdot z \cdot x \leq y^* \cdot z$
    by (metis add-lub-var calculation mult-isol mult-onel order-trans star-ref)
  thus $z \cdot x^* \leq y^* \cdot z$
    by (metis mult.assoc star-inductr)
qed

```

```isar
lemma star-inductr-eq: $z + y \cdot x \leq y \Rightarrow z \cdot x^* \leq y$
by (metis eq-iff star-inductr)

```
lemma star-inductr-var-eq: \( y \cdot x = y \implies y \cdot x^* \leq y \)
by (metis eq_iff star-inductr-var)

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

lemma bubble-sort: \( y \cdot x \leq x \cdot y = \implies (x + y)^* = x^* \cdot y^* \)
by (metis church-rosser star-sim4)

lemma independence1: \( x \cdot y = 0 \implies x \cdot y^* = y \)
proof
  assume \( x \cdot y = 0 \)
also have \( x^* \cdot y = y + x^* \cdot x \cdot y \)
    by (metis distrib-right mult-oneal star-unfoldr)
thus \( x^* \cdot y = y \)
    by (metis add-0-left add.commute add-ub1 calculation eq_iff star-inductl)
qed

lemma independence2: \( x \cdot y = 0 \implies x \cdot y^* = x \)
by (metis annil mult-oneal star-sim3 star-zero)

lemma lazycomm-var: \( y \cdot x \leq x \cdot (x + y)^* + y \leftrightarrow y \cdot x^* \leq x \cdot (x + y)^* + y \)
proof
  let \(?t = x \cdot (x + y)^* \)
  assume \( y \cdot x \leq ?t + y \)
also have \(?t + y) \cdot x = ?t \cdot x + y \cdot x \)
    by (metis distrib-right)
moreover have \( \ldots \leq ?t \cdot x + ?t + y \)
    by (metis add-iso-var calculation le-less add.assoc)
moreover have \( \ldots \leq ?t + y \)
    by (metis add-iso-var add-lub-var mult.assoc mult-isol order-refl prod-star-closure
star-subdist-var-1)
  hence \( y + (?t + y) \cdot x \leq ?t + y \)
    by (metis add.commute add-lub-var add-ub1 calculation less-eq_def mult.assoc
distrib-left star-subdist-var-1 star-trans)
  thus \( y \cdot x^* \leq x \cdot (x + y)^* + y \)
    by (metis star-inductr)
next
  assume \( y \cdot x^* \leq x \cdot (x + y)^* + y \)
thus \( y \cdot x \leq x \cdot (x + y)^* + y \)
    by (metis mult-isol order-trans star-ext)
qed

lemma arden-var: \((\forall y v. y \leq x \cdot y + v \implies y \leq x^* \cdot v) \implies z = x \cdot z + w \implies 
\)
\( z = x^* \cdot w \)
by (metis add-comm eq_iff star-inductl)

lemma \((\forall x y. y \leq x \cdot y \implies y = 0) \implies y \leq x \cdot y + z \implies y \leq x^* \cdot z \)
by (metis eq-refl mult-oneI)
end

end

2 Demonic Refinement Algebras

theory DRA
  imports DRA-Base
begin

A demonic refinement algebra [8] is a Kleene algebra without right annihilation plus an operation for possibly infinite iteration.

class dra = kleene-algebra-zerol +
  fixes strong-iteration :: 'a ⇒ 'a (∞ [101] 100)
  assumes iteration-unfoldl [simp]: I + x · x∞ = x∞
  and coinduction: y ≤ z + x · y → y ≤ x∞ · z
  and isolation [simp]: x* + x∞.0 = x∞
begin

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

abbreviation top-elem :: 'a (⊤) where ⊤≡ 1∞

Simple/basic lemmas about iteration operator

lemma iteration-refl: 1 ≤ x∞
  by (metis add-ub1 iteration-unfoldl)

lemma top-ref: x ≤ ⊤
  by (metis add-idem’ add-lub add-ub1 mult-1-left mult-1-right coinduction)

lemma top-mult-annil[simp]: ⊤·x = ⊤
  by (metis add-ub2 eq-iff mult-1-left mult-1-right coinduction top-ref)

lemma top-add-annil[simp]: ⊤ + x = ⊤
  by (metis add.commute less-eq-def top-ref)

lemma top-elim: x·y ≤ x·⊤
  by (metis mult-isol top-ref)

lemma iteration-unfoldl-distl: y·x∞ = y + y·x∞
  by (metis distrib-left mult.assoc mult-oneI iteration-unfoldl)

lemma iteration-unfoldl-distr: x∞·y = y + x·x∞·y
  by (metis distrib-right’ mult-1-left iteration-unfoldl)

lemma iteration-unfoldl’: z·x∞·y = z·y + z·x·x∞·y
  by (metis distrib-left mult.assoc iteration-unfoldl-distr)
lemma iteration-idem[simp]: $x^\infty \cdot x^\infty = x^\infty$
  by (metis add-zerol annil isolation mult.assoc iteration-refl iteration-unfoldl-distr star-inductl-var-eq2 star-invol star-sum-unfold sup-id-star1)

lemma iteration-induct: $x \cdot x^\infty \leq x^\infty \cdot x$
  by (metis eq-iff mult.assoc coinduction iteration-unfoldl-distl)

lemma iteration-ref-star: $x^* \leq x^\infty$
  by (metis eq-refl iteration-unfoldl star-inductl-one)

lemma iteration-subdist: $x^\infty \leq (x + y)^\infty$
  by (metis add-assoc' distrib-right' mult-oner coinduction add-ub1 iteration-unfoldl)

lemma iteration-iso: $x \leq y \Rightarrow x^\infty \leq y^\infty$
  by (metis iteration-subdist order-prop)

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

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

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

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

lemma iteration-unfoldr-dist: $z \cdot x^\infty \cdot y = z \cdot y + z \cdot x^\infty \cdot x \cdot y$
  by (metis distrib-left mult.assoc iteration-unfoldr-dist)

lemma iteration-double[simp]: $(x^\infty)^\infty = \top$
  by (metis less-eq-def iteration-iso iteration-refl top-add-annil)

lemma star-iteration[simp]: $(x^*)^\infty = \top$
  by (metis less-eq-def iteration-iso star-ref top-add-annil)

lemma iteration-star[simp]: $(x^\infty)^* = x^\infty$
  by (metis boffa less-eq-def iteration-idem iteration-refl)

lemma iteration-star2[simp]: $x^* \cdot x^\infty = x^\infty$
  by (metis add.assoc boffa isolation mult-1-right iteration-idem iteration-unfoldl star-denest star-denest-var-2 star-invol star-slide star-zero)

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

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

lemma sup-id-top: \(I \leq y \implies y \cdot \top = \top\)
by (metis antisym-conv iteration-unfold-eq mult-isol-var top-mult-annil top-ref)

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

Next, we prove some simulation laws for data refinement.

lemma iteration-sim: \(z \cdot y \leq x \cdot z \implies z \cdot y^\infty \leq x^\infty \cdot z\)
proof –
  assume \(assms: z \cdot y \leq x \cdot z\)
  have \(z \cdot y^\infty = z + z \cdot y \cdot y^\infty\)
  by (metis distrib-left mult.assoc mult-oner iteration-unfoldl)
  also have \(...) \leq z + x \cdot z \cdot y^\infty\)
  by (metis assms add.commute add-iso mult-isor)
  finally show \(z \cdot y^\infty \leq x^\infty \cdot z\)
  by (metis mult.assoc coinduction)
qed

Nitpick gives a counterexample to the dual simulation law.

lemma \(y \cdot z \leq z \cdot x \implies y^\infty \cdot z \leq z \cdot x^\infty\)
oops

Next, we prove some sliding laws.

lemma iteration-slide-var: \(x \cdot (y \cdot x)^\infty \leq (x \cdot y)^\infty \cdot x\)
by (metis eq-refl iteration-sim mult.assoc)

lemma iteration-prod-unfold: \((y \cdot x)^\infty = 1 + y \cdot (x \cdot y)^\infty \cdot x\)
apply (rule antisym, metis iteration-unfoldl add-iso-var eq-refl iteration-slide-var
mult.assoc mult-isol)
by (metis add-iso-var iteration-refl iteration-slide-var iteration-unfoldl iteration-zero
mult.assoc mult-isol-var mult-isol-var mult-oner)

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

Nitpick refutes the next lemma.

lemma \((x \cdot y \cdot *)^\infty = (x \cdot y)^\infty\)
oops

lemma star-iteration-slide: \((x \cdot y)^\infty = y \cdot (x \cdot y)^\infty\)
proof –
  have \(y \cdot (x \cdot y)^\infty \leq 1 + (x \cdot y \cdot (x \cdot y)^\infty + x \cdot y \cdot y \cdot (x \cdot y)^\infty\)
  by (metis star-unfoldl-eq distrib-right eq-refl iteration-unfoldl star-ref mult.1-left
mult-isol add-iso-var)
hence $y^*(x^*y)^\infty \leq 1 + x^*y^*(x^*y)^\infty$

by (metis less-eq-def add.assoc distrib-left distrib-right mult-1-left mult.assoc star-ref)

thus ?thesis

by (metis mult-1-right mult.assoc coinduction star-ref mult-1-left mult-isor add.commute less-eq-def)

qed

The following laws are called denesting laws.

**Lemma iteration-sub-denest:** $(x + y)^\infty \leq x^\infty \cdot (y^\infty x^\infty)^\infty$

**Proof**

- have $(x + y)^\infty = x \cdot (x + y)^\infty + y \cdot (x + y)^\infty + 1$

by (metis add.commute distrib-right' iteration-unfoldl)

- hence $(x + y)^\infty \leq x^\infty \cdot (y^\infty (x + y)^\infty + 1)$

by (metis add-assoc' add-lub-var add-ub1 add-ub2 coinduction)

- moreover hence $x^\infty \cdot (y^\infty (x + y)^\infty + 1) \leq x^\infty \cdot (y^\infty x^\infty)^\infty$

by (metis add-isol mult.assoc mult-isol add.commute coinduction mult-oner mult-isol)

ultimately show ?thesis

by (metis dual-order.trans)

qed

**Lemma iteration-denest:** $(x + y)^\infty = x^\infty \cdot (y^\infty x^\infty)^\infty$

**Proof**

- have $x^\infty \cdot (y^\infty x^\infty)^\infty \leq x^\infty \cdot (y^\infty \cdot x^\infty)^\infty + y^\infty \cdot (y^\infty \cdot x^\infty)^\infty + 1$

by (metis add.commute iteration-unfoldl-distr add-assoc' add.commute iteration-unfoldl order-refl)

thus ?thesis

by (metis add.commute iteration-sub-denest order.antisym coinduction distrib-right' iteration-sub-denest mult.assoc mult-oner order.antisym)

qed

**Lemma iteration-denest2:** $(x + y)^\infty = y^* \cdot x \cdot (x + y)^\infty + y^\infty$

**Proof**

- have $(x + y)^\infty = y^\infty \cdot x \cdot (y^\infty \cdot x)^\infty \cdot y^\infty + y^\infty$

by (metis add.commute iteration-denest iteration-slide iteration-unfoldl-distr)

also have ... = $y^* \cdot x \cdot (y^\infty \cdot x)^\infty \cdot y^\infty + y^\infty \cdot 0 + y^\infty$

by (metis isolation mult.assoc distrib-right' annil mult.assoc)

also have ... = $y^* \cdot x \cdot (y^\infty \cdot x)^\infty \cdot y^\infty + y^\infty$

by (metis add.assoc distrib-left mult-1-right add-0-left mult-1-right)

finally show ?thesis

by (metis add.commute iteration-denest iteration-slide mult.assoc)

qed

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

**Apply** (rule antisym, metis add.commute iteration-denest2 eq-refl coinduction)

by (metis add.commute iteration-denest iteration-slide mult-isor iteration-iso iteration-ref-star)

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

**Lemma** iteration-sep: \( y \cdot x \leq x \cdot y \Longrightarrow (x + y)^\infty = x^\infty \cdot y^\infty \)

**Proof** –
- **Assume** \( y \cdot x \leq x \cdot y \)
  - **Hence** \( y^* \cdot x \leq x \cdot (x + y)^* \)
    - By (metis star-sim1 add.commute mult-isol order-trans star-subdist)
  - **Hence** \( y^* \cdot x \cdot (x + y)^\infty + y^\infty \leq x \cdot (x + y)^\infty + y^\infty \)
    - By (metis mult-isol mult.assoc iteration-star2 add-iso-var eq-refl)
  - **Thus** \(?thesis\)
    - By (metis iteration-denest2 add.commute coinduction add.commute less-eq-def iteration-subdenest)

**QED**

**Lemma** iteration-sim2: \( y \cdot x \leq x \cdot y \Longrightarrow y^\infty \cdot x^\infty \leq x^\infty \cdot y^\infty \)

**Proof** –
- **Assume** \( y \cdot x \leq x \cdot y^* \)
  - **Hence** \( y^* \cdot x \cdot (x + y)^\infty \cdot y^\infty \leq x^\infty \cdot y^* \cdot y^\infty \)
    - By (metis mult.assoc mult-isol iteration-sim star-denest-star var-2 star-sim1 star-slide-var star-trans-eq tc-eq)
  - **Moreover have** \( x^\infty \cdot y^* \cdot y^\infty \leq x^\infty \cdot y^\infty \)
    - By (metis eq-refl mult.assoc iteration-star2)
  - **Moreover have** \( (y^* \cdot x)^\infty \cdot y^\infty \leq y^* \cdot (y^* \cdot x)^\infty \cdot y^\infty \)
    - By (metis mult-iso mult-one start-ref)
  - **Ultimately show** \(?thesis\)
    - By (metis antisym iteration-denest3 iteration-subdenest order-trans)

**QED**

**Lemma** iteration-sep3: \( y \cdot x \leq x \cdot (x + y) \Longrightarrow (x + y)^\infty = x^\infty \cdot y^\infty \)

**Proof** –
- **Assume** \( y \cdot x \leq x \cdot (x + y) \)
  - **Hence** \( y^* \cdot x \leq x \cdot (x + y)^* \)
    - By (metis star-sim1)
  - **Hence** \( y^* \cdot x \cdot (x + y)^\infty + y^\infty \leq x \cdot (x + y)^* \cdot (x + y)^\infty + y^\infty \)
    - By (metis add-iso mult.assoc)
  - **Hence** \( (x + y)^\infty \leq x^\infty \cdot y^\infty \)
    - By (metis mult.assoc iteration-denest2 iteration-star2 add.commute coinduction)
  - **Thus** \(?thesis\)
    - By (metis add.commute less-eq-def iteration-subdenest)

**QED**

**Lemma** iteration-sep4: \([y \cdot 0 = 0; z \cdot x = 0; y \cdot x \leq (x + z) \cdot y^*] \Longrightarrow (x + y + z)^\infty = x^\infty \cdot (y + z)^\infty\)

**Proof** –
- **Assume** \( y \cdot 0 = 0; z \cdot x = 0; y \cdot x \leq (x + z) \cdot y^* \)
  - **Have** \( y \cdot y^* \cdot z \leq y^* \cdot z \cdot y^* \)
    - By (metis mult-isol star-11 mult-onel order-trans star-plus-one distl)
have \( y^* \cdot z \cdot x \leq x \cdot y^* \cdot z \)
  by (metis zero-least assms(1) assms(2) independence1 mult.assoc)
have \( y \cdot (x + y^* \cdot z) \leq (x + z) \cdot y^* + y \cdot y^* \cdot z \)
  by (metis assms(3) distrib-left mult.assoc add-iso)
also have \( \ldots \leq (x + y^* \cdot z) \cdot y^* + y^* \cdot z \cdot y^* \)
  by (metis star-ref add-iso-var eq-refl mult-I-left mult-isor)
also have \( \ldots \leq (x + y^* \cdot z) \cdot y^* + y^* \cdot z \cdot y^* \)
  using \( y \cdot y^* \cdot z \leq y^* \cdot z \cdot y^* \)
  by (metis add.commute add-iso)
finally have \( y \cdot (x + y^* \cdot z) \leq (x + y^* \cdot z) \cdot y^* \)
  by (metis add.commute add-idem' add.left-commute distrib-right)
moreover have \( (x + y + z)^\infty \leq (x + y + y^* \cdot z)^\infty \)
  by (metis star-ref add-iso-var eq-refl mult-I-left mult-isor iteration-iso)
moreover have \( \ldots = (x + y^* \cdot z)^\infty \cdot y^\infty \)
  by (metis add.assoc add.commute calculation iteration-sep2)
moreover have \( \ldots = x^\infty \cdot (y^* \cdot z)^\infty \cdot y^\infty \)
  using \( (y^* \cdot z) \cdot x \leq x \cdot y^* \cdot z \)
  by (metis iteration-sep mult.assoc)
ultimately have \( (x + y + z)^\infty \leq x^\infty \cdot (y + z)^\infty \)
  by (metis add.commute mult.assoc iteration-denest3)
thus \( \text{thesis} \)
  by (metis add.commute add.left-commute less-eq_def iteration-subdenest)
qed

Finally, we prove some blocking laws.

Nitpick refutes the next lemma.
lemma \( x \cdot y = 0 \implies x^\infty \cdot y = y \)
  oops

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

Nitpick refutes the next lemma.
lemma \( y \cdot w \leq x \cdot y + z \implies y \cdot w^\infty \leq x^\infty \cdot z \)
  oops

At the end of this file, we consider a data refinement example from von Wright [8].

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

end

end

3 Test Dioids

theory Test-Dioids
  imports ../Kleene-Algebra/Dioid
begin

  Tests are embedded in a weak dioid, a dioid without the right annihilation
  and left distributivity axioms, using an operator t defined by a complemen-
  tation operator. This allows us to use tests in weak settings, such as
  Probabilistic Kleene Algebra and Demonic Refinement Algebra.

class near-dioid-tests-zerol = ab-near-semiring-one-zerol + plus-ord +
  fixes comp-op :: 'a ⇒ 'a (n·[90] 91)
  assumes test-one: n n 1 = 1
  and test-mult: n n (n n x · n n y) = n n y · n n x
  and test-mult-comp: n x · n n x = 0
  and test-de-morgan: n x + n y = n (n n x · n n y)
begin

  lemma add-idem'[simp]; x + x = x
    by (metis annil distrib-right' mult-1-left test-de-morgan test-mult-comp test-one)

  subclass near-dioid-one-zerol
    by (unfold-locales, simp)

  lemma x · (y + z) = x · y + x · z
    oops

  lemma n n x · (y + z) = n n x · y + n n x · z
    oops

  A test operator t can then be defined as an abbreviation of applying n
  twice. The elements of the image, t(K), form a Boolean algebra, but we
do not express this in Isabelle. Instead, we prove all the obvious laws of
Boolean algebra.
abbreviation test-operator :: 'a ⇒ 'a (t- [100] 101) where
  t x ≡ n (n x)

lemma test-zero[simp]: t 0 = 0
  by (metis mult-1-left test-mult-comp test-one)

lemma test-distrib-left: t x · (t y + t z) = (t x · t y) + (t x · t z)
  by (metis add.commute distrib-right' test-de-morgan test-mult)

lemma test-distrib-right: (t x + t y) · t z = (t x · t z) + (t y · t z)
  by (metis distrib-right')

lemma test-mult-idem[simp]: t x · t x = t x
  by (metis add-0-right test-distrib-left mult-1-right test-de-morgan test-mult-comp test-one)

lemma test-idem[simp]: t t x = t x
  by (metis add-idem' test-de-morgan test-mult)

lemma test-mult-comm: t x · t y = t y · t x
  by (metis test-mult)

lemma test-add-comm: t x + t y = t y + t x
  by (metis add-comm)

lemma test-mult-assoc: t x · (t y · t z) = (t x · t y) · t z
  by (metis mult.assoc)

lemma test-add-assoc: t x + (t y + t z) = (t x + t y) + t z
  by (metis add.assoc)

lemma test-mult-lb1: t x · t y ≤ t x
  by (metis add.commute add-ub1 mult-1-left mult-isor test-add-comp test-de-morgan test-mult)

lemma test-mult-lb2: t x · t y ≤ t y
  by (metis test-mult-comm test-mult-lb1)

lemma test-add-lb: t x · (t x + t y) = t x
  by (metis add.commute less-eq-def test-distrib-left test-mult-idem test-mult-lb1)
lemma test-leq-mult-def: \((t \leq t y) = (t \cdot t y = t x)\)
by (metis less-eq-def test-add-lb test-mult-comm test-mult-lb1)

lemma test-mult-glbI: \([t z \leq t x; \ t z \leq t y] \implies t z \leq t x \cdot t y\)
by (metis (full-types) order-trans test-mult-glb test-mult-lb1 test-mult-lb2)

lemma test-mult-glb: \(t z \leq t x \land t z \leq t y \iff t z \leq t x \cdot t y\)
by (metis (full-types) order-trans test-mult-glbI test-mult-lb1 test-mult-lb2)

lemma test-add-distl: \((t x \cdot t y) + t z = (t x + t z) \cdot (t y + t z)\)
proof (rule antisym)
  have \(t x \cdot t y \leq (t x + t z) \cdot (t y + t z)\)
  by (metis add-lub mult-isor order-prop test-distrib-left)
  thus \(t x \cdot t y + t z \leq (t x + t z) \cdot (t y + t z)\)
  by (metis add-lub-var add-ub2 distrib-right test-add-comm test-add-lb)
next
  show \((t x + t z) \cdot (t y + t z) \leq t x \cdot t y + t z\)
  by (metis add.commute test-add-lb test-de-morgan test-distrib-left test-mult test-mult-lb2)
qed

lemma test-add-distr: \(t x + (t y \cdot t z) = (t x + t y) \cdot (t x + t z)\)
by (metis add-comm test-add-distl)

lemma test-add-zerol: \(0 + t x = t x\)
by (metis add-zerol)

lemma test-add-zeror: \(t x + 0 = t x\)
by (metis add-zeror)

lemma test-mult-onel: \(1 \cdot t x = t x\)
by (metis mult-onel)

lemma test-mult-oner: \(t x \cdot 1 = t x\)
by (metis mult-oner)

lemma test-lb: \(t x \geq 0\)
by (metis zero-least)

lemma test-ub: \(t x \leq 1\)
by (metis add-ub1 test-add-comp)

  A test is an element \(p\) where \(t p = p\).

definition test :: 'a \Rightarrow bool where
  test \(p \equiv t p = p\)

notation comp-op \((\_ \cdot [101] \_00)\)
lemma test-add-closed-var: \([\text{test } p; \text{test } q] \Rightarrow \text{test } (p + q)\)
  by (metis test-add-closed test-def)

lemma test-mult-closed: \([\text{test } p; \text{test } q] \Rightarrow \text{test } (p \cdot q)\)
  by (metis test-def test-mult test-mult-comm)

lemma test-comp-closed: \(\text{test } p \Rightarrow \text{test } (!p)\)
  by (metis test-def)

lemma test-ub-var: \(\text{test } p \Rightarrow p \leq 1\)
  by (metis test-def test-ub)

lemma test-lb-var: \(\text{test } p \Rightarrow p \geq 0\)
  by (metis zero-least)

lemma test-zero-var: \(\text{test } 0\)
  by (metis test-def test-zero)

lemma test-one-var: \(\text{test } 1\)
  by (metis test-def test-one)

lemma test-not-one: \(!1 = 0\)
  by (metis mult-oner test-mult-comp test-one)

lemma test-add-idem: \(\text{test } p \Rightarrow p + p = p\)
  by (metis add-idem')

lemma test-mult-idem-var [simp]: \(\text{test } p \Rightarrow p \cdot p = p\)
  by (metis test-def test-mult-idem)

lemma test-add-comm-var: \([\text{test } p; \text{test } q] \Rightarrow p + q = q + p\)
  by (metis add.commute)

lemma test-mult-comm-var: \([\text{test } p; \text{test } q] \Rightarrow p \cdot q = q \cdot p\)
  by (metis test-def test-mult-comm)

lemma test-distrib-left-var: \([\text{test } p; \text{test } q; \text{test } r] \Rightarrow p \cdot (q + r) = p \cdot q + p \cdot r\)
  by (metis distrib-right' test-add-closed-var test-mult-comm-var)

lemma test-distrib-right-var: \([\text{test } p; \text{test } q; \text{test } r] \Rightarrow (p + q) \cdot r = p \cdot r + q \cdot r\)
  by (metis distrib-right)

lemma test-add-distl-var: \([\text{test } p; \text{test } q; \text{test } r] \Rightarrow p \cdot q + r = (p + r) \cdot (q + r)\)
  using test-add-distl[of p q r] by (simp add: test-def)

lemma test-add-distr-var: \([\text{test } p; \text{test } q; \text{test } r] \Rightarrow p + q \cdot r = (p + q) \cdot (p + r)\)
  by (metis test-add-comm test-add-distl-var)

lemma test-absorb1: \([\text{test } p; \text{test } q] \Rightarrow p + p \cdot q = p\)
by (metis test-add-distr-var test-add-idem test-add-lb test-def)

lemma test-absorb2: \([\text{test } p; \text{test } q] \Rightarrow p \cdot (p + q) = p\)
by (metis test-distrib-left-var test-mult-idem-var test-absorb1)

lemma test-absorb3: \([\text{test } p; \text{test } q] \Rightarrow (p + q) \cdot q = q\)
by (metis add.commute test-absorb2 test-add-closed-var test-mult-comm-var)

lemma test-leq-mult-def-var: \([\text{test } p; \text{test } q] \Rightarrow (p \leq q) \equiv (p \cdot q = p)\)
by (safe, metis test-comp, metis test-dist-var)

lemma test-comp-mult: \(\text{test } p \Rightarrow \exists q. \text{test } q \land p + q = 1 \land p \cdot q = 0\)
by (metis test-comp-mult2 simp test-comp-add simp test-comp-closed-var)

lemma test-comp-mult2 [simp]: \(\text{test } p \Rightarrow !p \cdot p = 0\)
by (metis test-comp-mult simp)

lemma test-comp-add [simp]: \(\text{test } p \Rightarrow p + !p = 1\)
by (metis test-comp-add simp)

lemma test-comp-closed-var: \(\text{test } p \Rightarrow p \equiv !(!p)\)
by (metis test-comp-mult2 simp)

lemma de-morgan1: \([\text{test } p; \text{test } q] \Rightarrow !p + !q = !(p \cdot q)\)
by (metis test-de-morgan test-def)

lemma de-morgan2: \([\text{test } p; \text{test } q] \Rightarrow !p \cdot !q = !(p + q)\)
by (metis add.commute add-idem' opp-mult-def test-de-morgan test-double-comp-var test-mult)

lemma de-morgan3: \([\text{test } p; \text{test } q] \Rightarrow !(p + !q) = p \cdot q\)
by (metis de-morgan test-double-comp-var test-mult-closed)

lemma de-morgan4: \([\text{test } p; \text{test } q] \Rightarrow !(p \cdot !q) = p + q\)
by (metis de-morgan2 test-add-closed-var test-def)
lemma test-comp-anti: \([\text{test } p; \text{test } q] \Rightarrow (p \leq q) = (!q \leq !p)\)
by (metis add.commute de-morgan1 test-double-comp-var test-mult-closed less-eq-def test-leq-mult-def)

lemma ba1: \([\text{test } p; \text{test } q; \text{test } r] \Rightarrow p + q + !q = r + !r\)
by (metis ba1 add.assoc mult-onel test-absorb1 test-add-comp test-def test-one-var)

lemma ba2: \([\text{test } p; \text{test } q; \text{test } r] \Rightarrow p + p + !q = r + !r\)
by (metis ba2 add.commute mult-onel test-absorb1 test-add-comp test-def test-one-var)

lemma ba3: \([\text{test } p; \text{test } q] \Rightarrow (p + q) = (p + !q)\)
by (metis ba3 add-idem add-zeror ba1 test-absorb1 test-add-comp test-def test-one-var)

lemma ba4: \([\text{test } p; \text{test } q] \Rightarrow p + !q = p + (p + !q)\)
by (metis ba4 mult-onel test-comp-mult test-def)

lemma ba5: \([\text{test } p; \text{test } q] \Rightarrow !q = !q + !q\)
by (metis ba5 test-comp-uniq)

lemma ba6: \(\text{test } p =!p \cdot p = 0\)
by (metis test-def test-mult-comp)

lemma ba7: \([\text{test } p; \text{test } q] \Rightarrow !p = !(p + q) + !(p + !q)\)
by (metis ba7 add-mult ba3 add-comp-closed test-double-comp-var)

lemma test-restrictl: \(\text{test } p \Rightarrow p \cdot x \leq x\)
by (metis distrib-right' test-comp-mult add-zerol)

lemma test-restrictr: \(\text{test } p \Rightarrow x \cdot p \leq x\)
oops

lemma \([\text{test } p; \text{test } q; \text{test } r] \Rightarrow p \cdot q \leq r \iff p \leq !q + r\)
proof auto
assume \(\text{assms: test } p \text{ test } q \text{ test } r \text{ p.q} \leq r\)
  hence \(p \leq r + p \cdot !q\)
  by (metis add-iso ba3 distrib-left distrib-left)
thus \(p \leq r + !q\)
  by (metis add-iso-var assms(1) dual-order.trans order-refl test-restrictl)
next
assume \(\text{test } p \text{ test } q \text{ test } r \text{ p.q} \leq r + !q\)
thus \(p \cdot q \leq r\)
  by (metis add.commute mult-isor distrib-right' add-zeror test-comp-mult2 order-trans test-mult-comm-var test-restrictl)
qed

Next, we prove lemmas mixing the embedded tests and any element of the carrier set.

lemma test-eq1: \(\text{test } p \Rightarrow y \leq x \iff p \cdot y \leq x \land !p \cdot y \leq x\)
apply standard
apply (metis order-trans test-comp-closed-var test-restrictl)
apply (metis add-iso-var add-idem' distrib-right' mult-onel test-comp-add)
done

Nitpick refutes the next four lemmas.

lemma test-eq2: test p \implies z \leq p \cdot x + !p \cdot y \iff p \cdot z \leq p \cdot x \land !p \cdot z \leq !p \cdot y
  oops

lemma test-eq3: [test p; test q] \implies p \cdot x = p \cdot x \cdot q \iff p \cdot x \leq x \cdot q
  oops

lemma test1: [test p; test q; p \cdot x!q = 0] \implies p \cdot x = p \cdot x \cdot q
  oops

lemma test-eq4: [test p; test q] \implies x!q = !p \cdot x!q \iff p \cdot x!q = 0
  apply standard
  apply (metis annil mult.assoc test-comp-mult)
  apply (metis add-zerol distrib-right' mult-onel test-comp-add)
done

lemma test2: [test p; test q] \implies p \cdot q \cdot p = p \cdot q
  by (metis mult.assoc test-mult-comm-var test-mult-idem-var)

lemma test3: [test p; test q] \implies p \cdot q!p = 0
  by (metis ba5 test-absorb1 test-comp-mult test-mult-closed)

lemma test4: [test p; test q] \implies !p \cdot q \cdot p = 0
  by (metis annil ba6 mult.assoc test-mult-comm-var)

Nitpick refutes the next two lemmas.

lemma comm-add: [test p; p \cdot x = x \cdot p; p \cdot y = y \cdot p] \implies p \cdot (x + y) = (x + y) \cdot p
  oops

lemma comm-add-var: [test p; test q; test r; p \cdot x = x \cdot p; p \cdot y = y \cdot p] \implies p \cdot (q \cdot x + r \cdot y) = (q \cdot x + r \cdot y) \cdot p
  oops

lemma comm-mult: [test p; test q; q \cdot x = x \cdot q] \implies p \cdot q \cdot x = q \cdot p \cdot x
  by (metis mult.assoc test-mult-comm-var)

lemma de-morgan-var1: [test p; test q; test r] \implies (!p + q) \cdot (p + r) = p \cdot q + !p \cdot r
  proof –
  assume tests: test p test q test r
  hence (!p + q) \cdot (p + r) = !p \cdot p + !p \cdot r + q \cdot p + r \cdot q
  by (metis add-assoc' distrib-right' test-comp-closed-var test-distrib-left-var test-mult-comm-var)
also have ... = !p·r + q·p + (p + !p)·r·q
  by (metis add-zeroI mult-oneI test-comp-add test-comp-mult2 tests(1))
also have ... = !p·r + q·p + p·r·q + !p·r·q
  by (metis add-assoc' test-comp-closed-var test-distrib-right-var test-mult-closed tests)
also have ... = !p·r + p·q + p·q·r + !p·r·q
  by (metis mult.assoc test-mult-comm-var tests)
also have ... = !p·r·1 + !p·r·q + p·q·1 + p·q·r
  by (metis add-commute add.left-commute mult-oneI)
also have ... = !p·r·(1 + q) + p·q·(1 + r)
  by (metis add-assoc' test-comp-closed-var test-distrib-left-var test-mult-closed test-one-var tests)
finally show ?thesis
  by (metis add.commute less-eqI mult-oneI test-ub-var tests(2−3))
qed

lemma de-morgan-var2: [test p; test q; test r] =⇒ !(p·q + !p·r) = (p·!q + !p·!r)
  by (metis de-morgan1 de-morgan2 de-morgan-var1 test-def)

Nitpick refutes the next two lemmas.

lemma test p =⇒ p·x = x·p =⇒ p·x = p·x·p ∧ !p·x = !p·x·!p
  oops

lemma test-distrib: [test p; test q] =⇒ (p + q)·(q·y + !q·x) = q·y + !q·p·x
  oops

end

We now make the assumption that tests distribute over finite sums of arbitrary elements from the left. This can be justified in models such as multirelations and probabilistic predicate transformers.

class near-dioid-test-zeroI-dist = near-dioid-tests-zeroI +
  assumes weak-distrib-left: t x · (y + z) = t x · y + t x · z
begin

lemma weak-distrib-left-var: test p =⇒ p·(x + y) = p·x + p·y
  by (metis weak-distrib-left test-double-comp-var)

lemma weak-subdistl: test p =⇒ p·x ≤ p·(x + y)
  by (metis order-prop weak-distrib-left-var)

lemma weak-subdistl-var: test p =⇒ p·x + p·y ≤ p·(x + y)
  by (metis add.commute add-lub weak-subdistl)

lemma weak-mult-isol: test p =⇒ x ≤ y =⇒ p·x ≤ p·y
  by (metis less-eqI weak-subdistl)

lemma weak-mult-isol-var: [test p; test q] =⇒ p ≤ x ∧ q ≤ y =⇒ p·q ≤ x·y
  by (metis weak-mult-isol mult-isor order-trans)
lemma \textit{weak-mult-double-iso}: \( \text{test } p \iff x \leq y \rightarrow p \cdot x \cdot z \leq p \cdot y \cdot z \)

by (metis weak-mult-isol mult-isor)

Nitpick refutes the next lemma.

lemma \textit{test-restrictr}: \( \text{test } p \iff x \cdot p \leq x \)

oops

lemma \textit{test-eq2}: \( \text{test } p \iff z \leq p \cdot x + !p \cdot y \leftrightarrow p \cdot x \leq p \cdot y \)

proof auto

assume \textit{assms}: \( \text{test } p \leq z \leq \text{p} \cdot x + !p \cdot y \)

hence \( p \cdot (p \cdot x + !p \cdot y) \leq p \cdot x \)

by (metis add-zero annil mult.assoc weak-distrib-left-var test-comp-mult test-restrictl)

thus \( p \cdot z \leq p \cdot x \)

by (metis \textit{assms} weak-mult-isol order-trans)

next

assume \textit{assms}: \( \text{test } p \leq z \leq p \cdot x + !p \cdot y \)

hence \( !p \cdot (p \cdot x + !p \cdot y) \leq !p \cdot y \)

by (metis \textit{assms} test-comp-closed weak-mult-isol order-trans)

thus \( !p \cdot z \leq !p \cdot y \)

by (metis \textit{assms} test-comp-closed weak-mult-isol order-trans)

next

assume \textit{assms}: \( \text{test } p \leq z \leq p \cdot x + !p \cdot y \)

thus \( z \leq p \cdot x + !p \cdot y \)

by (metis mult.assoc test-comp-add)

qed

Nitpick refutes the next three lemmas.

lemma \textit{test-eq3}: \( [\text{test } p; \text{test } q] \iff p \cdot x = p \cdot x \cdot q \leftrightarrow p \cdot x \leq x \cdot q \)

oops

lemma \textit{test1}: \( [\text{test } p; \text{test } q; p \cdot x \cdot !q = 0] \iff p \cdot x = p \cdot x \cdot q \)

oops

lemma \( [\text{test } p; \text{test } q; x \cdot !q = !p \cdot x \cdot !q] \iff p \cdot x = p \cdot x \cdot q \)

oops

Next, we study tests with commutativity conditions.

lemma \textit{comm-add}: \( [\text{test } p; p \cdot x = x \cdot p; p \cdot y = y \cdot p] \Rightarrow p \cdot (x + y) = (x + y) \cdot p \)

by (metis distrib-right weak-distrib-left-var)

lemma \textit{comm-add-var}: \( [\text{test } p; \text{test } q; \text{test } r; p \cdot x = x \cdot p; p \cdot y = y \cdot p] \Rightarrow p \cdot (q \cdot x + r \cdot y) = (q \cdot x + r \cdot y) \cdot p \)

by (metis \textit{comm-add} weak-distrib-right)

lemma \textit{test-distrib}: \( [\text{test } p; \text{test } q] \Rightarrow (p + q) \cdot (q \cdot y + !q \cdot x) = q \cdot y + !q \cdot p \cdot x \)

proof

assume \textit{tests}: \( \text{test } p \text{ test } q \)
hence \((p + q)\cdot(q\cdot y + !q\cdot x) = p\cdot q\cdot y + p\cdot !q\cdot x + q\cdot q\cdot y + q\cdot !q\cdot x\)
by (metis add-assoc' distrib-right' mult.assoc weak-distrib-left-var)
also have \(... = p\cdot q\cdot y + p\cdot !q\cdot x + q\cdot y\)
by (metis add.commute add zeroal annil test-comp-mult test-mult-idem-var tests(2))
also have \(... = (p + 1)\cdot q\cdot y + p\cdot !q\cdot x\)
by (metis add.commute add. left-commute distrib-right' mult-oner test-mult-comm-var
    test-one-var tests(2))
finally show \(?thesis
by (metis mult-oner test-absorb3 test-comp-closed-var test-mult-comm-var test-one-var
    tests(1) tests(2))
qed

end

The following class is relevant for probabilistic Kleene algebras.

class pre-dioid-test-zerol = near-dioid-test-zerol-dist + pre-dioid
begin

subclass pre-dioid-one-zerol
by (unfold-locales)

lemma test-restrictr: test p \Rightarrow x \cdot p \leq x
by (metis mult-oner subdistl test-comp-uniq)

lemma test-eq3: \([test p; test q] \Rightarrow p\cdot x = p\cdot x\cdot q \leftrightarrow p\cdot x = x\cdot q\)
apply standard
apply (metis mult. assoc test-restrictl)
apply (metis eq-iff mult. assoc mult-isol test-mult-idem-var test-restrictr)
done

lemma test1: \([test p; test q; p\cdot x\cdot !q = 0] \Rightarrow p\cdot x = p\cdot x\cdot q\)
oops

lemma \([test p; test q; x\cdot !q = !p\cdot x\cdot !q] \Rightarrow p\cdot x = p\cdot x\cdot q\)
oops

lemma \([test p; test q] \Rightarrow x \cdot (p + q) \leq x \cdot p + x \cdot q\)
oops

end

The following class is relevant for Demonic Refinement Algebras.

class dioid-tests-zerol = dioid-one-zrol + pre-dioid-test-zerol
begin

lemma test1: \([test p; test q; p\cdot x\cdot !q = 0] \Rightarrow p\cdot x = p\cdot x\cdot q\)
by (metis add. 0-left add. commute distrib-left mult. oner test-comp-add)

Nitpick refutes the following five lemmas.
lemma \([\text{test } p; \text{test } q; p \cdot x \cdot q = 0] \implies p \cdot x \cdot q = 0\)

oops

lemma \([\text{test } p; \text{test } q; p \cdot x = p \cdot x \cdot q] \implies x \cdot q = p \cdot x \cdot q\)

oops

lemma \([\text{test } p; \text{test } q; p \cdot x = p \cdot x \cdot q] \implies p \cdot x \cdot q = 0\)

oops

lemma \([\text{test } p; \text{test } q; p \cdot x = p \cdot x \cdot q] \implies p \cdot x \cdot q = 0\)

oops

lemma \([\text{test } p; \text{test } q; x \cdot q = !p \cdot x \cdot q] \implies p \cdot x \cdot q = 0\)

oops

lemma \([\text{test } p; \text{test } q; x \cdot q = !p \cdot x \cdot q] \implies p \cdot x \cdot q = 0\)

oops

lemma \([\text{test } p; \text{test } q; p \cdot x = p \cdot x \cdot q] \implies p \cdot x = p \cdot x \cdot q\)

by (metis annil mult.assoc test1 test-comp-mult)

Nitpick refutes the following four lemmas.

lemma \([\text{test } p; \text{test } q; !p \cdot x \cdot q = 0] \implies p \cdot x = p \cdot x \cdot q\)

oops

lemma \([\text{test } p; \text{test } q; !p \cdot x \cdot q = 0] \implies x \cdot q = !p \cdot x \cdot q\)

oops

lemma \([\text{test } p; \text{test } q; !p \cdot x \cdot q = 0] \implies p \cdot x \cdot !q = 0\)

oops

lemma \([\text{test } p; \text{test } q; p \cdot x = p \cdot x \cdot q] \implies p \cdot x = x \cdot p\)

oops

lemma assumes \(\text{test } p\) and \(p \cdot x = x \cdot p\)

shows \(p \cdot x = p \cdot x \cdot p \land !p \cdot x = !p \cdot x \cdot !p \implies p \cdot x = x \cdot p\)

proof

show \(p \cdot x = p \cdot x \cdot p \land !p \cdot x = !p \cdot x \cdot !p\)

by (metis assms eq-refl test-eq3)

next

have \(!p \cdot x = !p \cdot x \cdot (p + \lnot p)\)

by (metis assms(1) mult-oner test-comp-add)

thus \(!p \cdot x = !p \cdot x \cdot !p\)

by (metis assms distrib-left mult.assoc add-zerol annil test-comp-mult2)

qed

end

The following class is relevant for Kleene Algebra with Tests.

class dioid-tests = dioid-tests-zero + dioid-one-zero
begin
lemma kat-eq1: \([test p; test q] \implies (p \cdot x \cdot !q = 0) = (p \cdot x = p \cdot x \cdot q)\)
by (metis annir mult.assoc test1 test-comp-mult)

lemma kat-eq2: \([test p; test q] \implies (p \cdot x \cdot !q = 0) = (p \cdot x \leq x \cdot q)\)
by (metis kat-eq1 test-eq3)

lemma kat-eq3: \([test p; test q] \implies (p \cdot x = p \cdot x \cdot q) = (x \cdot !q = !p \cdot x \cdot !q)\)
by (metis kat-eq1 test-eq4)

Nitpick refutes the next lemma.

lemma \([test p; test q] \implies (p \cdot x \cdot !q = 0) \implies (!p \cdot x = 0)\)
oops

lemma comm-eq1: \(test b \implies (p \cdot b = b \cdot p) = (b \cdot p \cdot !b + !b \cdot p \cdot b = 0)\)
apply standard
apply (metis add-0-left annil annir test-double-comp-var test-mult-comp mult.assoc)
apply (metis add0-left ba6 de-morgan1 distrib-right! test-double-comp-var kat-eq1
test-one mult.assoc mult-onel no-trivial-inverse test-comp-closed-var test-not-one)
done

lemma comm-eq2: \(test b \implies (p \cdot !b = !b \cdot p) = (b \cdot p \cdot !b + !b \cdot p \cdot b = 0)\)
by (metis add-comm comm-eq1 test-comp-closed-var test-double-comp-var)

lemma comm-eq3: \(test b \implies (p \cdot b = b \cdot p) = (p \cdot !b = !b \cdot p)\)
by (metis comm-eq1 comm-eq2)

lemma comm-pres: \(test p \implies p \cdot x = p \cdot x \cdot p \land !p \cdot x = !p \cdot x \cdot !p \iff p \cdot x = x \cdot p\)
apply standard
apply (metis comm-eq3 kat-eq3)
apply (metis annil ba6 comm-eq3 mult.assoc test-eq4 test-mult-idem-var)
done

end

end

4 Kleene Algebra with Tests

theory KAT
  imports ../DRA-Base Test-Dioids
begin
  First, we study left Kleene algebras with tests which also have only a left
zero. These structures can be expanded to demonic refinement algebras.

class left-kat-zerol = left-kleene-algebra-zerol + dioid-tests-zerol
begin

lemma star-test-export1: \(test p \implies (p \cdot x)^* \cdot p \leq p \cdot x^*\)
by (metis mult-isol mult-oner star-iso star-slide test-eq3 test-one-var)
lemma star-test-export2: test p \implies (p \cdot x)^* \cdot p \leq x^* \cdot p
by (metis mult-isor star2 star-denest star-invol star-iso star-slide star-subdist-var-2 star-subid test-ub-var)

lemma star-test-export-left: \[ [\text{test } p; x \cdot p \leq p \cdot x] \implies x^* \cdot p = (p \cdot x)^* \]
apply (rule antisym)
apply (metis mult.assoc mult-isol-var star-sim1 test-double-comp-var test-mult-idem-var test-mult-lb1)
by (metis star-slide star-test-export2)

lemma star-test-export-right: \[ [\text{test } p; p \cdot x \leq x \cdot p] \implies x^* \cdot p = p \cdot (p \cdot x)^* \]
apply (rule antisym)
apply (metis mult.assoc mult-isol-var star-sim1 test-double-comp-var test-mult-idem-var test-mult-lb1)
by (metis star-slide star-test-export-left)

lemma star-test-export2-left: \[ [\text{test } p; p \cdot x = x \cdot p] \implies x^* \cdot p = p \cdot (x \cdot p)^* \]
by (metis order-refl star-test-export-left)

lemma star-test-export2-right: \[ [\text{test } p; p \cdot x = x \cdot p] \implies x^* \cdot p = (x \cdot p)^* \cdot p \]
by (metis order-refl star-test-export-left)

lemma star-test-folk: \[ [\text{test } p; p \cdot x = x \cdot p; p \cdot y = y \cdot p] \implies (p \cdot x + !p \cdot y)^* \cdot p = (p \cdot x)^* \]
proof
assume assms: test p p \cdot x = x \cdot p p \cdot y = y \cdot p
hence (p \cdot x + !p \cdot y)^* \cdot p = p \cdot (p \cdot x + p \cdot !p \cdot y)^*
  by (metis mult.assoc star-test-export2-left distrib-left)
thus ?thesis
  by (metis assms(1) test-double-comp-var test-mult-comp test-mult-idem-var add-zeror annil)
qed

end

class kat-zerol = kleene-algebra-zerol + dioid-tests-zerol
begin

subclass left-kat-zerol
by (unfold-locales)

lemma star-sim-right: \[ [\text{test } p; p \cdot x = x \cdot p] \implies p \cdot x^* = (p \cdot x)^* \cdot p \]
by (metis mult.assoc star-sim3 test-mult-idem-var)

lemma star-sim-left: \[ [\text{test } p; p \cdot x = x \cdot p] \implies p \cdot x^* = p \cdot (x \cdot p)^* \]
by (metis star-sim-right star-slide)

lemma comm-star: \[ [\text{test } p; p \cdot x = x \cdot p; p \cdot y = y \cdot p] \implies p \cdot x \cdot p \cdot y)^* \cdot p = p \cdot x \cdot y^* \]
by (metis star-sim-right mult.assoc star-slide)

lemma star-sim-right-var: \[ [\text{test } p; p \cdot x = x \cdot p] \implies x^* \cdot p = p \cdot (x \cdot p)^* \]

22
by (metis mult.assoc star-sim3 test-mult-idem-var)

lemma star-folk-var[simp]: [test p; p·x = x·p; p·y = y·p] \(\implies\) (p·x + !p·y)·p = p·x²
by (metis star-test-folk comm-star mult-onel mult-oner)

lemma star-folk-var2[simp]: [test p; !p·x = x·!p; !p·y = y·!p] \(\implies\) (p·x + !p·y)·!p = !p·y²
by (metis star-folk-var add.commute test-def)

end

Finally, we define Kleene algebra with tests.

class kat = kleene-algebra + dioid-tests
begin
subclass kat-zerol
apply (unfold-locales)
by (metis star-inductr)
end

end

5 Demonic Refinement Algebra with Tests

theory DRAT
  imports Test-Dioids ../DRA
begin

  In this section, we define demonic refinement algebras with tests and
prove the most important theorems from the literature. In this context,
tests are also known as guards.

class dra-tests = dra + dioid-tests-zerol
begin

  An assertion is a mapping from a guard to a subset similar to tests, but
it aborts if the predicate does not hold.

definition assertion :: 'a ⇒ 'a (cob [101] 100) where
  test p \(\implies\) p° = !p·⊤ + 1

lemma asg: [test p; test q] \(\implies\) q ≤ 1 ∧ 1 ≤ p°
by (metis add.commute add-ub1 assertion-def test-ub-var)

lemma assertion-isol: test p \(\implies\) y ≤ p°·x \iff p·y ≤ x
proof
  assume assms: test p y ≤ p°·x
  hence p·y ≤ !p·⊤·x + p·x
  by (metis mult.assoc mult-isol assertion-def assms(1) distrib-left distrib-right' mult-1-left mult.assoc)
also have ... \leq x
  by (metis assms(1) distrib-right' mult.assoc add-zerol annil test-comp-mult
eq-refl test-eq1)
finally show \( p \cdot y \leq x \)
  by metis

next
assume assms: test p \( p \cdot y \leq x \)
hence \( p^o \cdot p \cdot y = !p \cdot \top \cdot p \cdot y + p \cdot y \)
  by (metis assertion-def distrib-right' mult-1-left mult.assoc)
also have ... = \( !p \cdot \top + p \cdot y \)
  by (metis mult.assoc top-mult-annil)
moreover have \( p^o \cdot p \cdot y \leq p^o \cdot x \)
  by (metis assms(2) mult.assoc mult-isol)
moreover have \( !p \cdot y + p \cdot y \leq !p \cdot \top + p \cdot y \)
  by (metis add.commute assms(1) order-refl test-eq2 top-elim)
ultimately show \( y \leq p^o \cdot x \)
  by (metis add.commute assms(1) distrib-right' mult-1-left order-trans test-comp-add)
qed

lemma assertion-isor: test p \Rightarrow y \leq x \cdot p \leftrightarrow y \cdot p^o \leq x
proof
assume assms: test p y \leq x \cdot p
hence \( y \cdot p^o \leq x \cdot p \cdot !p \cdot \top + x \cdot p \)
  by (metis mult-isor assertion-def assms(1) distrib-left mult-1-right mult.assoc)
also have ... \leq x
  by (metis assms(1) add-zerol annil distrib-left mult.assoc test-comp-mult distrib-left
mult-1-right order-prop test-comp)
finally show \( y \cdot p^o \leq x \)
  by metis

next
assume assms: test p \( y \cdot p^o \leq x \)
have \( y \leq y \cdot ( !p \cdot \top + p ) \)
  by (metis add-iso-var mult-isol order-refl order-refl top-elim add.commute
assms(1) mult-1-right test-comp-add)
also have ... = \( y \cdot p^o \cdot p \)
  by (metis assertion-def assms(1) distrib-right' mult-1-left mult.assoc top-mult-annil)
finally show \( y \leq x \cdot p \)
  by (metis assms(2) mult-isor order-trans)
qed

lemma assertion-iso: \([ \text{test} p ; \text{test} q] \Rightarrow x \cdot q^o \leq p^o \cdot x \leftrightarrow p \cdot x \leq x \cdot q \)
by (metis assertion-isol assertion-mult.assoc)

lemma total-correctness: \([ \text{test} p ; \text{test} q] \Rightarrow p \cdot x \cdot !q = 0 \leftrightarrow x \cdot !q \leq !p \cdot \top \)
apply standard
apply (metis mult.assoc test-eq1 top-elim zero-least)
apply (metis annil test-comp-mult zero-unique mult.assoc mult-isol)
done
lemma test-iteration-sim: ![test p; p·x ≤ x·p] ⇒ p·x ≤ x·p
by (metis iteration-sim)

lemma test-iteration-annir: ![test p] ⇒ !(p·x) = !p
by (metis annid bab iteration-idem mult.assoc)

Next we give an example of a program transformation from von Wright [8].

lemma loop-refinement: ![test p; test q] ⇒ (p·x) ≤ (p·q·x) ≤ !(p·q) · (p·x)!
p

proof –
  assume asms: ![test p test q]
  hence ![p·x] = ((p·q) + !(p·q)) · !(p·x)!
  by (metis de-morgan3 mult-oneel test-add-comp)
  also have ![p·x] = (p·q) · !(p·x)!
  by (metis distrib-right)
  also have ![p·x] = (p·q) · !(p·x)!
  by (metis iteration-unfoldr-distr mult.assoc iteration-unfoldr-distr-left mult.assoc)
  thus ![p·x] = (p·q) · !(p·x)!
  by (metis assms less-def test3 zero-least)
  finally have ![p·x] ≤ (p·q·x) · !(p·x)!
  by (metis assms mult.assoc test2 eq iff)
  qed

Finally, we prove different versions of Back’s atomicity refinement theorem for action systems.

lemma atom-step1: ![r·b ≤ b·r] ⇒ ![a + b + r) = b·r · (a · b·r)∞]
apply (subgoal-tac ![a + b + r) = b·r · (a · b·r)∞])
apply (metis iteration-sep mult.assoc)
by (metis add-assoc' add.commute iteration-denest)

lemma atom-step2:
  assumes ![s = sq q b = 0; r·q ≤ q·r q·l ≤ l·q r ≤ r·test q]
  shows ![s·l·b·r·r·q·(a · b·r·q·r·∞)] ≤ ![s·l·b·r·r·q·(a · b·r·q·r·∞)]

proof –
  have ![s·l·b·r·r·q·(a · b·r·q·r·∞)] ≤ ![s·l·b·r·r·q·(a · b·r·q·r·∞)]
  by (metis assms(3) assms(5) star-sim1 mult.assoc mult-isol iteration-iso)
  also have ![s·l·b·r·r·q·(a · b·r·q·r·∞)] ≤ ![s·l·b·r·r·q·(a · b·r·q·r·∞)]
  by (metis assms(1,6) test-ub-var mult-double-iso mult-oner)
  also have ![s·l·b·r·r·q·(a · b·r·q·r·∞)] ≤ ![s·l·b·r·r·q·(a · b·r·q·r·∞)]
  by (metis assms(4) iteration-sim mult.assoc mult-double-iso mult-double-iso)
  also have ![s·l·b·r·r·q·(a · b·r·q·r·∞)] ≤ ![s·l·b·r·r·q·(a · b·r·q·r·∞)]
  by (metis assms(2) zero-least iteration-sim mult.assoc mult-double-iso)
  also have ![s·l·b·r·r·q·(a · b·r·q·r·∞)] ≤ ![s·l·b·r·r·q·(a · b·r·q·r·∞)]
  by (metis assms(6) mult.assoc mult-isol test-restrict iteration-idem mult.assoc)
  finally show ![s·l·b·r·r·q·(a · b·r·q·r·∞)] ≤ ![s·l·b·r·r·q·(a · b·r·q·r·∞)]
  by metis
qed
lemma atom-step3:
  assumes $r \cdot l \leq l \cdot r \cdot a \cdot l \leq l \cdot a \cdot b \cdot l \leq l \cdot b \cdot q \cdot l \leq l \cdot q \cdot b^\infty = b^*$
  shows $l^\infty \cdot r^\infty \cdot (a \cdot b^\infty \cdot q \cdot r^\infty)^\infty = (a \cdot b^\infty \cdot q + l + r)^\infty$

proof –
  have $(a \cdot b^\infty \cdot q + r) \cdot l \leq a \cdot b^\infty \cdot l \cdot q + l \cdot r$
    by (metis distrib-right add-iso-var assms(1,4) mult.assoc mult-isol)
  also have \ldots \leq a \cdot l \cdot b^\infty \cdot q + l \cdot r
    by (metis assms(3) assms(5) star-sim1 add-iso_var mult.assoc mult-double-iso)
  also have \ldots \leq l \cdot (a \cdot b^\infty \cdot q + r)
    by (metis add-iso assms(2) mult-isor distrib-left mult.assoc)
  finally have $(a \cdot b^\infty \cdot q + r) \cdot l \leq l \cdot (a \cdot b^\infty \cdot q + r)$
    by metis
  moreover have $l^\infty \cdot r^\infty \cdot (a \cdot b^\infty \cdot q \cdot r^\infty)^\infty = l^\infty \cdot (a \cdot b^\infty \cdot q + r)^\infty$
    by (metis add.commute mult.assoc iteration-denest)
  ultimately show \?thesis
    by (metis add.commute add.left-commute iteration-sep)
qed

This is Back’s atomicity refinement theorem, as specified by von Wright [8].

theorem atom-ref-back:
  assumes $s = s \cdot q \cdot a = q \cdot a \cdot q \cdot b = 0$
  \ldots $r \cdot b \leq b \cdot r \cdot l \leq l \cdot r \cdot r \cdot q \leq q \cdot r$
  \ldots $a \cdot l \leq l \cdot a \cdot b \cdot l \leq l \cdot b \cdot q \cdot l \leq l \cdot q$
  $r^\infty = r^* \cdot b^\infty = b^* \cdot test \cdot q$
  shows $s \cdot (a + b + r + l)^\infty \cdot q \leq s \cdot (a \cdot b^\infty \cdot q + r + l)^\infty$

proof –
  have $(a + b + r) \cdot l \leq l \cdot (a + b + r)$
    by (metis add-iso-var distrib-right! assms(5) assms(7) assms(8) distrib-left)
  hence $s \cdot (l + a + b + r)^\infty \cdot q = s \cdot l^\infty \cdot (a + b + r)^\infty \cdot q$
    by (metis add.commute add.left-commute mult.assoc iteration-sep)
  also have \ldots \leq s \cdot r^\infty \cdot b^\infty \cdot r^\infty \cdot q \cdot (a \cdot b^\infty \cdot r^\infty \cdot q)^\infty
    by (metis assms(2,4,10,11) atom-step1 iteration-slide eq-refl mult.assoc)
  also have \ldots \leq s \cdot r^\infty \cdot r^\infty \cdot (a \cdot b^\infty \cdot q \cdot r^\infty)^\infty
    by (metis assms(1) assms(10) assms(12) assms(3) assms(6) assms(9) atom-step2)
  also have \ldots \leq s \cdot (a \cdot b^\infty \cdot q + l + r)^\infty
    by (metis assms(11) assms(5) assms(7) assms(8) assms(9) atom-step3 eq-refl mult.assoc)
  finally show \?thesis
    by (metis add.commute add.left-commute)
qed

This variant is due to Höfner, Struth and Sutcliffe [4].
proof –

have \(s(a + b + r + l)\infty = s(\infty(a + b + r))\infty\)
  by (metis add.commute add.left-commute assms(6) iteration-sep mult.assoc)
also have \(\ldots = s(\infty(a + b + r))\infty\cdot(a(\infty)\infty)\infty\cdot q\)
  by (metis add-assoc' add.commute iteration-denest add.commute mult.assoc)
also have \(\ldots = s(\infty(\infty))\infty\cdot(a(\infty)\infty)\infty\cdot q\)
  by (metis assms(4) iteration-sep mult.assoc)
also have \(\ldots \leq s(\infty(\infty)\infty\cdot(q(\infty)\infty)\infty\cdot q\)
  by (metis assms(2) iteration-iso mult-isol-var eq-refl order-refl)
also have \(\ldots = s(\infty(\infty)\infty\cdot(q(\infty)\infty)\infty\cdot q)\infty\)
  by (metis iteration-slide mult.assoc)
also have \(\ldots \leq s(\infty(q(q)\infty)\infty\cdot(q(\infty)\infty)\infty\cdot q)\infty\)
  by (metis assms(1) mult-isor)
also have \(\ldots \leq s(\infty(q(q)\infty)\infty\cdot(q(\infty)\infty)\infty\cdot q)\infty\)
  by (metis assms(7) iteration-sim mult.assoc mult-double-iso)
also have \(\ldots \leq s(\infty(q(q)\infty)\infty\cdot(q(\infty)\infty)\infty\cdot q)\infty\)
  by (metis assms(3) iteration-idep mult.assoc order-refl)
also have \(\ldots \leq s(\infty(q(q)\infty)\infty\cdot(q(\infty)\infty)\infty\cdot q)\infty\)
  by (metis assms(8) eq-refl)
also have \(\ldots \leq s(\infty(q(q)\infty)\infty\cdot(q(\infty)\infty)\infty\cdot q)\infty\)
  by (metis assms(5) iteration-iso mult.assoc mult-isol star-sim1)
also have \(\ldots = s(\infty(q(q)\infty)\infty\cdot(q(\infty)\infty)\infty\cdot q)\infty\)
  by (metis assms(8))
also have \(\ldots \leq s(\infty(q(q)\infty)\infty\cdot(q(\infty)\infty)\infty\cdot q)\infty\)
  by (metis assms(9) mult-1-right mult-double-iso mult-isor)
also have \(\ldots \leq s(\infty(q(q)\infty)\infty\cdot(q(\infty)\infty)\infty\cdot q)\infty\)
  by (metis assms(9) mult-1-right mult-double-iso)
also have \(\ldots = s(\infty(a(\infty)\infty)\infty\cdot q + r)\infty\)
  by (metis add.commute mult.assoc iteration-denest)
also have \(\ldots \leq s(\infty(a(\infty)\infty)\infty\cdot q + r+ l)\infty\)
  by (metis add.commute iteration-subdenest mult.assoc mult-isol)

finally show ?thesis.

qed

Finally, we prove Cohen’s [2] variation of the atomicity refinement theorem.

lemma atom-ref-cohen:
  assumes \(r \cdot p \cdot y \leq y \cdot r \cdot p \cdot l \leq l \cdot r\)
  \(p \cdot r \cdot p \cdot l = 0 \cdot p \cdot l \cdot p \cdot l = 0 \cdot p \cdot l \cdot p = 0\)
  \(y \cdot 0 = 0 \cdot r \cdot 0 = 0 \cdot test \cdot p\)
  shows \((y + r + l)\infty = (p \cdot l)\infty \cdot (p \cdot l + r \cdot p)\infty \cdot (r \cdot p)\infty\)

proof –

have \((y + r) \cdot p \cdot l \leq l \cdot y + l \cdot r\)
  by (metis distrib-right' add-iso-var assms(2) assms(3))
  hence stepA: \((y + r) \cdot p \cdot l \leq (p \cdot l + !p \cdot l) \cdot (y + r)\)
    by (metis assms(9) distrib-left distrib-right' mult-1-left mult-isol order-trans
    star-ext test-comp-add)

have subStepB: \((l \cdot p + y + p \cdot r + !p \cdot r)\infty = (l \cdot p + y + r \cdot p + r \cdot p)\infty\)
  by (metis add-assoc annil assms(8) assms(9) distrib-left distrib-right' star-slide

27
6 Models for Demonic Refinement Algebra with Tests

definition top :: 'a bfun where top ≡ λx. UNIV

definition bot :: 'a bfun where bot ≡ λx. { }

definition adjoint :: 'a bfun ⇒ 'a bfun where adjoint f ≡ (λp. - f (-p))
definition fun-inter :: 'a bfun ⇒ 'a bfun ⇒ 'a bfun (infix ⊓ 51) where
  f ⊓ g ≡ λ p. f p ⊓ g p

definition fun-union :: 'a bfun ⇒ 'a bfun ⇒ 'a bfun (infix + 52) where
  f + g ≡ λ p. f p ∪ g p

definition fun-order :: 'a bfun ⇒ 'a bfun ⇒ bool (infix ≤ 50) where
  f ≤ g ≡ ∀ p. f p ⊆ g p

definition fun-strict-order :: 'a bfun ⇒ 'a bfun ⇒ bool (infix <. 50) where
  f <. g ≡ f ≤ g ∧ f ≠ g

definition N :: 'a bfun ⇒ 'a bfun where
  N f ≡ (adjoint f o bot) ⊓ id

lemma top-max: f ≤ top
  by (auto simp: top-def fun-order-def)

lemma bot-min: bot ≤ f
  by (auto simp: bot-def fun-order-def)

lemma order-def: f ⊓ g = f ⇒ f ≤ g
  by (metis fun-inter-def fun-order-def le-iff-inf)

lemma order-def-var: f ≤ g ⇒ f ⊓ g = f
  by (auto simp: fun-inter-def fun-order-def)

lemma adjoint-idem [simp]: adjoint (adjoint f) = f
  by (auto simp: adjoint-def)

lemma adjoint-prop1[simp]: (f o top) ⊓ (adjoint f o bot) = bot
  by (auto simp: fun-inter-def adjoint-def bot-def top-def)

lemma adjoint-prop2[simp]: (f o top) + (adjoint f o bot) = top
  by (auto simp: fun-union-def adjoint-def bot-def top-def)

lemma adjoint-mult: adjoint (f o g) = adjoint f o adjoint g
  by (auto simp: adjoint-def)

lemma adjoint-top[simp]: adjoint top = bot
  by (auto simp: adjoint-def bot-def top-def)

lemma N-comp1: (N (N f)) + N f = id
  by (auto simp: fun-union-def N-def fun-inter-def adjoint-def bot-def)

lemma N-comp2: (N (N f)) o N f = bot
  by (auto simp: N-def fun-inter-def adjoint-def bot-def)
lemma N-comp3: \( N f o (N (N f)) = \bot \)
  by (auto simp: N-def fun-inter-def adjoint-def bot-def)

lemma N-de-morgan: \( N (N f) o N (N g) = N (N f) \cap N (N g) \)
  by (auto simp: fun-union-def N-def fun-inter-def adjoint-def bot-def)

lemma conj-pred-aux [simp]: \( (\lambda p. x p \cup y p) = y \Longrightarrow \forall p. x p \subseteq y p \)
  by (metis Un-upper1)

Next, we define a type for conjuctive or multiplicative predicate transformers.

typedef 'a bool-op = {
  f::'a bfun. (\forall g h. mono f \land f \circ g + f \circ h = f \circ (g + h) 
  \land bot o f = bot)
}

apply (rule-tac x=\lambda x. x in exI)
apply auto
apply (metis monoI)
by (auto simp: fun-order-def fun-union-def)

setup-lifting type-definition-bool-op

Conjuctive predicate transformers form a dioid with tests without right annihilator.

instantiation bool-op :: (type) dioid-one-zerol
begin
  lift-definition less-eq-bool-op :: 'a bool-op \Rightarrow 'a bool-op \Rightarrow bool is fun-order .

  lift-definition less-bool-op :: 'a bool-op \Rightarrow 'a bool-op \Rightarrow bool is op < .

  lift-definition zero-bool-op :: 'a bool-op is bot
    by (auto simp: bot-def fun-union-def fun-order-def mono-def)

  lift-definition one-bool-op :: 'a bool-op is id
    by (auto simp: fun-union-def fun-order-def mono-def)

  lift-definition times-bool-op :: 'a bool-op \Rightarrow 'a bool-op \Rightarrow 'a bool-op is op o
    by (auto simp: o-def fun-union-def fun-order-def bot-def mono-def) metis

  lift-definition plus-bool-op :: 'a bool-op \Rightarrow 'a bool-op \Rightarrow 'a bool-op is op +
    apply (auto simp: o-def fun-union-def fun-order-def bot-def bot-def mono-def)
    apply (metis set-mp)
    apply (metis set-mp)
    by (metis (hide-lams, no-types) Un-left-commute sup-assoc)

  instance
    by standard (transfer, auto simp: fun-order-def fun-strict-order-def fun-union-def
      bot-def) +

end
instantiation bool-op :: (type) dioid-tests-zerol
begin
lift-definition comp-op-bool-op :: 'a bool-op ⇒ 'a bool-op is N
  by (auto simp: N-def fun-inter-def adjoint-def bot-def fun-union-def mono-def)

instance
  by standard (transfer, auto simp: N-def fun-inter-def adjoint-def bot-def fun-union-def)+
end

definition fun-star :: 'a bfun ⇒ 'a bfun where
  fun-star f = lfp (λr. f o r + id)

definition fun-iteration :: 'a bfun ⇒ 'a bfun where
  fun-iteration f = gfp (λg. f o g + id)

  Verifying the iteration laws is left for future work. This could be obtained
  by integrating Preoteasa’s approach [6].

end

7 Models for Kleene Algebra with Tests

theory KAT-Models
  imports ../../Kleene-Algebra/Kleene-Algebra-Models KAT
begin
  We show that binary relations under the obvious definitions form Kleene
  algebra with tests.

interpretation rel-dioid-tests: dioid-tests op ∪ op O Id {} op ⊆ op ⊂ λx. Id ∩ (− x)
  by (unfold-locales, auto)

interpretation rel-kat: kat op ∪ op O Id {} op ⊆ op ⊂ rtrancl λx. Id ∩ (− x)
  by (unfold-locales)

typedef 'a relation = UNIV::'a rel set by auto

setup-lifting type-definition-relation

instantiation relation :: (type) kat
begin

  lift-definition comp-op-relation :: 'a relation ⇒ 'a relation is λx. Id ∩ (− x)
  done

  lift-definition zero-relation :: 'a relation is {} done

  lift-definition star-relation :: 'a relation ⇒ 'a relation is rtrancl done

  lift-definition less-eq-relation :: 'a relation ⇒ 'a relation ⇒ bool is op ⊆ done

  lift-definition less-relation :: 'a relation ⇒ 'a relation ⇒ bool is op ⊂ done

  lift-definition one-relation :: 'a relation is Id done

31
lift-definition times-relation :: 'a relation ⇒ 'a relation ⇒ 'a relation is op \ O
done

lift-definition plus-relation :: 'a relation ⇒ 'a relation ⇒ 'a relation is op \ U
done

instance
by standard (transfer, auto simp: o-def rel-kleene-algebra.star-inductl rel-kleene-algebra.star-inductr)+
end
end

8 Pre-Conway Algebra

theory Conway
imports ../DRA-Base Test-Dioids
begin

We define a weak regular algebra with serves as a common basis for
Kleene algebra and demonic refinement algebra. It is closely related to an
axiomatisation given by Conway [3].

class dagger-op =
  fixes dagger :: 'a ⇒ 'a \ [101] \ 100

class near-regalg-base-zerol = near-dioid-one-zerol + dagger-op +
  assumes dagger-denest: \ (x \ y)\ = (x \ x)\ \ y
and dagger-prod-unfold: \ (x \ y)\ = 1 + x \ y
begin

lemma dagger-unfoldl: 1 \ x \ x\ = x\ by (metis dagger-prod-unfold mult-1-left mult-1-right)
  Nitpick refutes the next lemma.
lemma dagger-unfoldl-distl: y \ x\ = y + y \ x \ x\ oops

lemma dagger-unfoldl-distr: x\ \ y = y + x \ x\ \ y
  by (metis distrib-right' mult-1-left dagger-unfoldl)

lemma dagger-unfoldr: 1 \ x\ \ x = x\ by (metis dagger-prod-unfold mult-1-right mult-1-left)
  Nitpick refutes the next lemma.
lemma dagger-unfoldr-distl: y \ x\ = y + y \ x\ \ x
oops

lemma dagger-unfoldr-distr: x\ \ y = y + x \ x\ \ y
by (metis dagger-unfoldr distrib-right mult-1-left mult.assoc)

\textbf{lemma} \emph{dagger-annil[simp]}: \((x\cdot 0)^\dagger = 1 + x\cdot 0\)
by (metis annil dagger-unfoldl mult.assoc)

\textbf{lemma} \emph{zero-dagger[simp]}: \(0^\dagger = 1\)
by (metis add-0-right annil dagger-annil)

Nitpick refutes the next lemma.
\textbf{lemma} \emph{dagger-denest2}: \((x + y)^\dagger = x^\dagger \cdot (y\cdot x)^\dagger\)
oops{}

NITPICK refutes the next lemma.
\textbf{lemma} \emph{dagger-isol}: \(x \leq y \Rightarrow x^\dagger \leq y^\dagger\)
by (metis dagger-simr mult-1-left mult-1-right)

\textbf{lemma} \emph{dagger-slide-var}: \(x \cdot (y \cdot x)^\dagger \leq (x \cdot y)^\dagger \cdot x\)
by (metis eq-refl dagger-simr mult.assoc)

\textbf{lemma} \emph{dagger-slide}: \(x \cdot (y \cdot x)^\dagger = (x \cdot y)^\dagger \cdot x\)
oops{}

We say that \(y\) preserves \(x\) if \(x \cdot y \cdot x = x \cdot y\) and \(!x \cdot y \cdot !x = !x \cdot y\). This
definition is taken from Solin [7]. It is useful for program transformation.
\textbf{lemma} \emph{preservation}: \(x \cdot y \cdot x = x \cdot y \Rightarrow x \cdot y^\dagger \leq (x \cdot y + z)^\dagger \cdot x\)
by (metis add-ab1 dagger-simr mult.isor)

end

class \emph{near-conway-sim-right-zerol-tests} = near-conway-sim-right-zerol + near-dioid-test-zerol-dist
begin

\textbf{lemma} \emph{test-preserv1-var}: \([\text{test } p; \ p \cdot x \cdot p = p \cdot x] \rightarrow p \cdot (p \cdot x + !p \cdot y)^\dagger \leq (p \cdot x)^\dagger \cdot p\)
by (metis eq-refl test-eq2 dagger-simr)

The next lemma could neither be proved nor refuted by nitpick.
\textbf{lemma} \emph{test-preserv2-var}: \([\text{test } p; \ p \cdot x \cdot p = p \cdot x] \rightarrow p \cdot (p \cdot x + !p \cdot y)^\dagger \leq x^\dagger\)
oops{}

end

class \emph{pre-conway} = near-conway-sim-right-zerol + pre-dioid-test-zerol
begin
subclass near-conway-sim-right-zerol-tests
  by (unfold-locales)

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

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

lemma test-preserve: \([\text{test p}; p \cdot x \cdot p = p \cdot x] \implies p \cdot x \dagger = (p \cdot x) \cdot p\)
proof (rule antisym, metis add-0-right dagger-slide mult.assoc preservation)
  assume test p p \cdot x \cdot p = p \cdot x
  hence \(x \cdot p \leq x\)
  hence \(p \cdot (x \cdot p) \dagger \leq p \cdot x \dagger\)
  thus \((p \cdot x) \cdot p \leq p \cdot x \dagger\)
  by (metis dagger-iso mult-isol)
qed

lemma test-dumb-var: \([\text{test p}; p \cdot x \cdot p = p \cdot x] \implies p \cdot (p \cdot x + !p \cdot y) \dagger = (p \cdot x) \cdot p\)
proof
  assume assms: test p p \cdot x \cdot p = p \cdot x
  have \(p \cdot (p \cdot x + !p \cdot y) \dagger = (p \cdot (p \cdot x + !p \cdot y)) \cdot p\)
    by (metis assms(1) assms(2) test-dumb-var test-preserve)
  thus \(?thesis\)
    by metis
qed

lemma test-preserve1: \([\text{test p}; p \cdot x \cdot p = p \cdot x] \implies p \cdot (p \cdot x + !p \cdot y) \dagger = (p \cdot x) \cdot p\)
proof
  assume assms: test p p \cdot x \cdot p = p \cdot x
  have \(p \cdot (p \cdot x + !p \cdot y) \dagger = (p \cdot (p \cdot x + !p \cdot y)) \cdot p\)
    by (metis assms(1) assms(2) test-dumb-var test-preserve)
  thus \(?thesis\)
    by (metis assms mult.assoc weak-distrib-left-var test-mult-idem-var annil test-comp-mult
      add-zeror)
qed

lemma test-preserve2: \([\text{test p}; p \cdot x \cdot p = p \cdot x] \implies p \cdot (p \cdot x + !p \cdot y) \dagger \leq x \dagger\)
by (metis test-preserve test-preserve1 test-restrict1)
end
9 Transformation Theorem for while Loops

theory FolkTheorem
imports Conway KAT DRAT
begin
We prove Kozen’s transformation theorem for while loops [5] in a weak setting that unifies previous proofs in Kleene algebra with tests, demonic refinement algebras and a variant of probabilistic Kleene algebra.

context pre-conway
begin

abbreviation preservation :: "'a ⇒ 'a ⇒ bool (infix preserves 60) where
x preserves p ≡ test p ∧ p·x·p = p·x ∧ !p·x = !p·x

lemma preserves-test-closed: [ [test p; x preserves q] ] ⇒ p·x preserves q
apply (auto, metis mult.assoc test-mult-comm-var)
by (metis mult.assoc test-comp-closed-var test-mult-comm-var)

lemma conditional-helper1:
assumes test r1 x1 preserves q y1 preserves q x2 preserves q y2 preserves q
shows p·q·x1 · (q·r1·y1 + !q·r2·y2)† · (q·!r1 + !q·!r2) = p·q·x1 · (r1·y1)† · !r1
proof
−
let ?B = q·!r1 + !q·!r2
have pres: q·(r1·y1) = q · (r1·y1) · q
by (metis assms preserves-test-closed)

hence q·(q·r1·y1 + !q·r2·y2)† = (q·r1·y1)† · q
by (metis assms (2−) test-preserve1 dagger-slide mult.assoc)

hence p·q·x1 · (q·r1·y1 + !q·r2·y2)† · ?B = p·q·x1 · (q·r1·y1)† · q·?B
by (metis assms (2) mult.assoc)

also have ... = p·q·x1 · (q·r1·y1)† · !r1
by (metis assms (5) mult.assoc weak-distrib-left-var test-comp-mult annil add-zeror
test-mult-idem-var)

also have ... = p·q·x1 · (r1·y1)† · !r1
by (metis assms (2) mult.assoc test-preserve)

finally show ?thesis .
qed

lemma conditional-helper2:
assumes test r2 x1 preserves q y1 preserves q x2 preserves q y2 preserves q
shows p·!q·x2 · (q·r1·y1 + !q·r2·y2)† · (q·!r1 + !q·!r2) = p·!q·x2 · (r2·y2)† · !r2
proof −

35
\[
\text{let } \exists B = q \cdot r_1 + !q \cdot r_2 \\
\text{have pres: } !q \cdot (r_2 \cdot y_2) = !q \cdot (r_1 \cdot y_1) \\
\text{by (metis assms preserves-test-closed)} \\
\text{hence } !q \cdot (r_1 \cdot y_1 + !q \cdot r_2 \cdot y_2)^\dagger = (q \cdot r_1 \cdot y_1 + !q \cdot r_2 \cdot y_2)^\dagger \cdot !q \\
\text{by (metis assms(2) test-preserve1 [of !q \cdot r_2 \cdot y_2 \cdot r_1 \cdot y_1] add.commute mult.assoc test-comp-closed-var test-double-comp-var)} \\
\text{hence } p \cdot !q \cdot x_2 \cdot (q \cdot r_1 \cdot y_1 + !q \cdot r_2 \cdot y_2)^\dagger \cdot !q \cdot B = p \cdot !q \cdot x_2 \cdot (q \cdot r_1 \cdot y_1 + !q \cdot r_2 \cdot y_2)^\dagger \cdot !q \cdot ?B \\
\text{by (metis assms(4) mult.assoc)} \\
\text{also have } ... = p \cdot !q \cdot x_2 \cdot (q \cdot r_2 \cdot y_2)^\dagger \cdot !q \cdot r_2 \\
\text{by (metis assms(5) mult.assoc test-comp-closed-var weak-distrib-left-var test-comp-mult2 test-mult-idem-var add-zero var annihil)} \\
\text{also have } ... = p \cdot !q \cdot x_2 \cdot (r_2 \cdot y_2)^\dagger \cdot !r_2 \\
\text{by (metis assms(4) mult.assoc pres test-comp-closed-var test-preserve)} \\
\text{finally show } ?thesis . \\
\text{qed}
\]

\text{lemma cond-distr:} \\
\text{assumes test p test q test r} \\
\text{shows } (p \cdot q + !p \cdot r) \cdot (p \cdot x + !p \cdot y) = p \cdot q \cdot x + !p \cdot r \cdot y \\
\text{proof} - \\
\text{have } (p \cdot q + !p \cdot r) \cdot (p \cdot x + !p \cdot y) = p \cdot q \cdot x + !p \cdot r \cdot y + !p \cdot r \cdot p \cdot x + !p \cdot r \cdot !p \cdot y \\
\text{by (metis assms distrib-right\' mult.assoc weak-distrib-left-var add.assoc test-comp-closed-var)} \\
\text{thus } ?thesis \\
\text{by (metis assms mult.assoc test2 test3 test4 annihil add-zero var test-comp-closed-var)} \\
\text{qed}

\text{theorem conditional:} \\
\text{assumes test p test r test r2} \\
\text{x1 preserves q y1 preserves q} \\
\text{x2 preserves q y2 preserves q} \\
\text{shows } (p \cdot q + !p \cdot q) \cdot (p \cdot x_1 \cdot (r_1 \cdot y_1)^\dagger \cdot !r_1 + !p \cdot x_2 \cdot (r_2 \cdot y_2)^\dagger \cdot !r_2) = \\
(p \cdot q + !p \cdot q) \cdot (p \cdot x_1 + !p \cdot x_2) \cdot ((q \cdot r_1 + !q \cdot r_2) \cdot (q \cdot y_1 + !q \cdot y_2)^\dagger \cdot !q \cdot r_1 + !q \cdot r_2) \\
\text{proof} - \\
\text{have } p \cdot q \cdot x_1 \cdot (r_1 \cdot y_1)^\dagger \cdot !r_1 = p \cdot q \cdot x_1 \cdot (q \cdot r_1 \cdot y_1 + !q \cdot r_2 \cdot y_2)^\dagger \cdot (q \cdot r_1 + !q \cdot r_2) \text{ and} \\
!p \cdot q \cdot x_2 \cdot (r_2 \cdot y_2)^\dagger \cdot !r_2 = !p \cdot q \cdot x_2 \cdot (q \cdot r_1 \cdot y_1 + !q \cdot r_2 \cdot y_2)^\dagger \cdot (q \cdot r_1 + !q \cdot r_2) \\
\text{apply (metis assms(2,4) \text{ conditional-helper1 [of r1 q x1 y1 x2 y2 p] mult.assoc})} \\
\text{by (metis assms(3,4) \text{ conditional-helper2 [of r2 q x1 y1 x2 y2 p] mult.assoc})} \\
\text{moreover have } (p \cdot q + !p \cdot q) \cdot (p \cdot x_1 \cdot (r_1 \cdot y_1)^\dagger \cdot !r_1 + !p \cdot x_2 \cdot (r_2 \cdot y_2)^\dagger \cdot !r_2) = p \cdot q \cdot (x_1 \cdot (r_1 \cdot y_1)^\dagger \cdot !r_1) \\
+ !p \cdot q \cdot (x_2 \cdot (r_2 \cdot y_2)^\dagger \cdot !r_2) \\
\text{by (metis assms(1,4) \text{ cond-distr mult.assoc test-def})} \\
\text{moreover have } ... = (p \cdot q \cdot x_1 + !p \cdot q \cdot x_2) \cdot (q \cdot r_1 \cdot y_1 + !q \cdot r_2 \cdot y_2)^\dagger \cdot (q \cdot r_1 + !q \cdot r_2) \\
\text{by (metis calculation(1) calculation(2) distrib-right\')} \\
\text{moreover have } ... = (q \cdot p \cdot x_1 + !q \cdot p \cdot x_2) \cdot (q \cdot r_1 \cdot y_1 + !q \cdot r_2 \cdot y_2)^\dagger \cdot (q \cdot r_1 + !q \cdot r_2) \\
\text{by (metis assms(1) assms(5) test-comp-closed-var test-comp-mult-comm-var)} \\
\text{moreover have } ... = (q \cdot p + !q \cdot p) \cdot (q \cdot x_1 + !q \cdot x_2) \cdot ((q \cdot r_1 + !q \cdot r_2) \cdot (q \cdot y_1 + !q \cdot y_2)^\dagger \cdot (q \cdot r_1 + !q \cdot r_2) \\
\text{by (metis assms(1\text{,3,5}) cond-distr de-morgan-var2 test-comp-closed-var)} \\
\text{ultimately show } ?thesis \\
\text{by (metis assms(1,5) test-comp-closed-var test-mult-comm-var)}
qed

theorem nested-loops:
  assumes test p test q
  shows \((p \cdot x \cdot (q \cdot y))\!\!\cdot!\!p = p \cdot x \cdot ((p + q) \cdot (q \cdot y + !q \cdot x))\!\!\cdot!\!((p + q) + !p)\nproof
  have \((p \cdot x \cdot ((p + q) \cdot (q \cdot y + !q \cdot x))\!\!\cdot!\!((p + q) + !p) = p \cdot x \cdot ((q \cdot y)\!\!\cdot!\!(p \cdot x))\!\!\cdot!\!(q \cdot x))\!\!\cdot!\!p \cdot q + !p + !p\n    by (metis assms test-distrib mult.assoc de-morgan2 dagger-denest2)
  thus \?thesis
    by (metis assms mult.assoc test-comp-closed-var test-mult-comm-var add.commute dagger-slide dagger-anfoldl-dist)
qed

lemma postcomputation:
  assumes \(y\) preserves \(p\)
  shows \((p \cdot x)\!\!\cdot!\!(p \cdot y) = p \cdot (p \cdot (p \cdot y + p))\!\!\cdot!\!p\)
proof
  have \((p \cdot x \cdot ((p \cdot y + p))\!\!\cdot!\!p = p \cdot (p \cdot x \cdot ((p \cdot y + p))\!\!\cdot!\!(p \cdot y + p))\!\!\cdot!\!(p \cdot y + p))\!\!\cdot!\!p\n    by (metis dagger-prod-unfold mult.assoc)
  also have \(\ldots = (p + p \cdot x \cdot ((p \cdot y + p))\!\!\cdot!\!(p \cdot y + p))\!\!\cdot!\!(p \cdot y + p))\!\!\cdot!\!p\n    by (metis assms weak-distrib-left-var distrib-right' mult-1-left)
  also have \(\ldots = p \cdot p \cdot x \cdot ((p \cdot y + p))\!\!\cdot!\!(p \cdot y + p))\!\!\cdot!\!(p \cdot y + p))\!\!\cdot!\!(p \cdot y + p))\!\!\cdot!\!p\n    by (metis assms distrib-right' test-mult-idem-var)
  also have \(\ldots = p \cdot p \cdot x \cdot ((p \cdot y + p))\!\!\cdot!\!(p \cdot y + p))\!\!\cdot!\!(p \cdot y + p))\!\!\cdot!\!(p \cdot y + p))\!\!\cdot!\!p\n    by (metis distrib-right' mult.assoc)
  also have \(\ldots = p \cdot x \cdot ((p \cdot y + p) \cdot p \cdot x)\!\!\cdot!\!(p \cdot y)\)
    by (metis assms mult.assoc test-double-comp-var test-mult-comp annil)
  also have \(\ldots = p \cdot x \cdot ((p \cdot y + p) \cdot (p \cdot x))\!\!\cdot!\!(p \cdot y + p))\!\!\cdot!\!(p \cdot y + p))\!\!\cdot!\!p\n    by (metis distri mult.assoc add-commute dagger-denest2)
  moreover have \(\ldots = p \cdot x \cdot ((p \cdot y + p) \cdot (p \cdot x))\!\!\cdot!\!(p \cdot y + p))\!\!\cdot!\!(p \cdot y + p))\!\!\cdot!\!p\n    by (metis assms dagger-slide mult.assoc)
  ultimately have \(p \cdot (p \cdot x \cdot ((p \cdot y + p))\!\!\cdot!\!(p \cdot x))\!\!\cdot!\!(p \cdot y + p))\!\!\cdot!\!(p \cdot y + p))\!\!\cdot!\!p\n    by (metis assms dagger-denest mult.assoc)
qed

lemma composition-helper:
  assumes test \(g\) test \(h\) \(g \cdot y = y \cdot g\)
  shows \((h \cdot y)\!\!\cdot!\!h \cdot g = g \cdot (h \cdot y)\!\!\cdot!\!h\)
  apply (subgoal_tac \((h \cdot y)\!\!\cdot!\!h \cdot g \leq (h \cdot y)\!\!\cdot!\!h \cdot g\))
  apply (metis assms(1) test-eq3 mult.assoc)
  by (metis assms mult.assoc test-mult-comm-var order-refl dagger-simr mult-isor test-comp-closed-var)

theorem composition:
assumes test g test h g·y = y·g !g·y = y·!g
shows (g·x)†!g·(h·y)†h = !g·(h·y)†!h + g·(g·x·(!g·(h·y)†!h + g))†!g
apply (subgoal-tac (h·y)†!h preserves g)
by (metis postcomputation mult.assoc, metis assms composition-helper test-comp-closed-var mult.assoc)

end

Kleene algebras with tests form pre-Conway algebras, therefore the transformation theorem is valid for KAT as well.

sublocale kat ⊆ pre-conway star
apply standard
apply (simp-all only: star-prod-unfold star-sim2)
apply (metis star-denest-var star-slide)
done

Demonic refinement algebras form pre-Conway algebras, therefore the transformation theorem is valid for DRA as well.

sublocale dra-tests ⊆ pre-conway strong-iteration
apply standard
apply (metis iteration-denest-var iteration-slide)
apply (metis iteration-prod-unfold, metis iteration-sim)
done

We do not currently consider an expansion of probabilistic Kleene algebra.

end

10 Hoare Logic

theory HoareLogic
  imports KAT
begin
context kat
begin

We encode validity of Hoare triples and derive the inference rules of propositional Hoare logic, that is, Hoare logic without the assignment rule, in Kleene algebra with tests.

definition hoare-triple :: ’a ⇒ ’a ⇒ ’a ⇒ bool (⦃⦃•⦄⦄) where
⦃⦃p⦄⦄ x ⦃⦃q⦄⦄ ≡ p·x = p·x·q ∧ test p ∧ test q

lemma hoare-triple-def-var: ⦃⦃p⦄⦄ x ⦃⦃q⦄⦄ = (p·x ≤ x·q ∧ test p ∧ test q)
by (metis hoare-triple-def kat-eq1 kat-eq2)

lemma skip-rule: test p ⇒ ⦃⦃p⦄⦄ 1⦃⦃p⦄⦄
by (simp add: hoare-triple-def)
lemma sequence-rule: \( \{p\} x \{q'\} \implies \{p\} x x' \{q'\} \)
  by (simp add: hoare-triple-def, metis mult.assoc)

lemma conditional-rule: \[
\begin{align*}
&\{p b\} x \{q\} ; \{p!b\} x' \{q'\} ; \text{test } p ; \text{test } b \implies \{p\} b x + \nonumber
&\{p\} x x' \{q\} = \nonumber
\end{align*}
\]
  by (simp add: hoare-triple-def, metis mult.assoc distrib-left distrib-right)

lemma consequence-rule: \[
\begin{align*}
&\text{test } p ; \nonumber
&\{p\} x \{q\} ; \nonumber
&\{p\} x \{q\} ; \nonumber
&\text{test } q \implies \{p\} x \{q\} \nonumber
\end{align*}
\]
  by (unfold hoare-triple-def, metis (full-types) mult.assoc test-leq-mult-def-var)

lemma while-rule-var: \[
\begin{align*}
&\{p\} x \{p\} = \nonumber
\end{align*}
\]
  by (metis hoare-triple-def-var star-sim2)

proof (unfold hoare-triple-def-var, auto)
  assume assms: \text{test } p \text{ test } q \text{ p } q \cdot x \leq x \cdot p
  hence \text{z p q x } x \leq q \cdot x \cdot p
  by (metis mult.assoc mult-isol)
  thus \text{p ((q \cdot x) \cdot !q \cdot p \cdot !q)} \leq \text{z (q \cdot x) \cdot !q \cdot (p \cdot !q)}
  by (metis assms(1,2) test-mult-comm-var star-sim2 mult-isor kat-eq3 mult.assoc test2)
qed (metis test-comp-closed-var test-mult-closed)

definition (in kat) while-inv :: \text{'a } \Rightarrow \text{'a } \Rightarrow \text{'a }
  (\text{while } \text{ inv } \text{ do } \text{ [64,64,64] 63}) \text{ where}
  \text{while } b \text{ inv } i \text{ do } p = (b \cdot p) \cdot !b

lemma hoare-while-inv:
  assumes tb: \text{test } b \text{ and } tp: \text{test } p \text{ and } tq: \text{test } q
  and pi: \text{p } \leq \text{i } \text{i } \cdot \text{!b } \leq \text{q}
  and inv-loop: \text{\{i } \cdot \text{b } c \text{ \{i\}}
  shows \text{\{p\} while } b \text{ inv } i \text{ do } c \text{ \{q\}}
  by (metis assms while-inv-def white-rule consequence-rule)

end
end

11 Verification Tool Prototype

theory Verification-Tool-Proto
  imports KAT-Models HoareLogic
begin

We show how a simple verification tool prototype can be obtained from
our formalisation of Kleene algebra with tests and its relational model. So
far, only natural numbers are admitted as datatypes. A more versatile
implementation is under construction.

First, we implement states and stores of a program in the relational
model.

**type-synonym** state = string ⇒ nat

**notation** times (infixr : 64)

Next, we define assignement.

**definition** lift-fn :: (state ⇒ state) ⇒ state relation where
lift-fn f = Abs-relation \{(x, f x) \mid x. True\}

**definition** assign-fn :: string ⇒ (state ⇒ nat) ⇒ state ⇒ state where
assign-fn x f σ = (λy. if x = y then f σ else σ y)

**definition** assign :: string ⇒ (state ⇒ nat) ⇒ state relation (infix := 99) where
x := e = lift-fn (assign-fn x e)

**lemma** lift-fn-compose [simp]: lift-fn f • lift-fn g = lift-fn (g ◦ f)
apply (simp add: lift-fn-def)
apply transfer
by auto

Then, we lift tests to the relational model.

**definition** assert :: 'a set ⇒ 'a relation where
assert X ≡ Abs-relation (Id-on X)

**lemma** test-assert [intro!, simp]: test (assert X)
apply (simp add: assert-def test-def comp-op-relation-def)
apply transfer
by auto

**lemma** test-not-assert [intro!, simp]: test (!(assert X))
apply (simp add: assert-def test-def comp-op-relation-def)
apply transfer
by auto

**lemma** [iff]: assert X ≤ assert Y \iff X ≤ Y
apply (auto simp add: assert-def)
apply transfer
apply auto
apply transfer
by auto

**lemma** [simp]: t (assert X) = assert X
by (metis test-assert test-double-comp-var)

**lemma** [simp]: assert X ; assert Y = assert (X ∩ Y)
apply (simp add: assert-def)
apply transfer
by auto

**lemma** [simp]: assert X ; !assert Y = assert (X − Y)

40
apply (simp add: assert-def)
apply transfer
by auto

Now, we implement Hoare's assignment axiom.

abbreviation assigns-notation :: state set ⇒ string ⇒ (state ⇒ nat) ⇒ state set
(\[ | ] [100,100,100] 101) where
P[x|m] ≡ (λstate. assign-fn x m state) ∘ P

lemma hoare-assignment': P[x|m] ⊆ Q =⇒ assert P ∙ x := m ≤ x := m ∙ assert Q
apply (simp add: assert-def assign-def image-def lift-fn-def)
apply transfer
by auto

lemma hoare-assignment: P[x|m] ⊆ Q =⇒ {assert P} x := m {Q}
by (metis (full-types) hoare-assignment' hoare-triple-def-var test-assert)

This a variant of the sequential rule where tests are assertions.

lemma hoare-seq: {assert p} x {assert q} =⇒ {assert q} x' {assert q}
by (metis sequence-rule)

We write a simple tactic for verification condition generation.

named-theorems hoare-simp simplification rules for the hoare tactic
named-theorems hoare-rule extra Hoare Rules

ML ⟨⟨
fun hoare-step-tac ctxt n = 
  resolve-tac ctxt ◁{thms hoare-assignment} n THEN TRY (resolve-tac ctxt ◁{thms subset-refl} n)
  ORELSE (resolve-tac ctxt ◁{thms hoare-while-inv} n THEN asm-full-simp-tac ctxt 1)
  ORELSE (FIRST'
    (map (fn thm =⇒ resolve-tac ctxt [thm]) (rev (Named-Theorems.get ctxt ◁{named-theorems hoare-rule})))) n)
val hoare-tac = Subgoal.FOCUS (fn {context = ctxt, ...} =⇒
  REPEAT (hoare-step-tac ctxt 1)
  THEN auto-tac ((fn ss =⇒ ss addsimps Named-Theorems.get ctxt ◁{named-theorems hoare-simp}) ctxt))
⟩⟩

declare hoare-seq [hoare-rule]

method-setup hoare-auto = ⟨⟨
Scan.succeed (fn ctxt =⇒ SIMPLE-METHOD (REPEAT (CHANGED (hoare-tac ctxt 1))))
⟩⟩
We add some syntactic sugar.

**abbreviation** assign-sugar :: string ⇒ string ⇒ state relation (infix := 99) where
\[ x := y \equiv x := (\lambda \sigma. \sigma y) \]

**abbreviation** hoare-sugar :: 'a set ⇒ 'a relation ⇒ 'a set ⇒ bool (\{|-\}|-\}) where
\( \{p\} \ c \ \{q\} \equiv \{assert p\} \ c \ \{assert q\} \)

**abbreviation** mod-sugar :: string ⇒ string ⇒ state ⇒ nat (infix mod 100) where
\[ x \mod y \equiv \lambda \sigma. (\sigma x) \mod (\sigma y) \]

**abbreviation** while-inv-sugar :: 'a set ⇒ 'a relation ⇒ 'a relation ⇒ 'a relation (while - inv - do - [64,64,64] 63) where
while \( b \ inv \ i \ do \ p \equiv \) while (assert b) inv (assert i) do \( p \)

As a complex example, we verify the partial correctness of Euclid’s algorithm.

**lemma** euclids-algorithm:
\( \{\{\sigma. \sigma "x" = x \land \sigma "y" = y\}\} \) while \( \{\sigma. \sigma "y" \neq 0\}\) inv \( \{\sigma. \gcd (\sigma "x") (\sigma "y") = \gcd x y\}\) do (\( "z" := "y"\); \( "y" := "x" \mod "y"\); \( "x" := "z"\)) \( \}\{\{\sigma. \sigma "x" = \gcd x y\}\}\) apply hoare-auto by (metis gcd-red-nat)

**12 Two sorted Kleene Algebra with Tests**

**theory** KAT2 imports ../../../Kleene-Algebra/Kleene-Algebra begin

As an alternative to the one-sorted implementation of tests, we provide a two-sorted, more conventional one. In this setting, Isabelle’s Boolean algebra theory can be used. This alternative can be developed further along the lines of the one-sorted implementation.

**syntax** -kat :: 'a ⇒ 'a (\{\})
named-theorems kat-hom KAT test homomorphism rules

ML ⟨⟨
val kat-test-vars = [p,q,r,s,t,p',q',r',s',t',p''',q''',r''',s''',t'']

fun map-ast-variables ast =
  case ast of
    (Ast.Variable v) =>
      if exists (fn tv => tv = v) kat-test-vars
      then Ast.Appl [Ast.Variable test, Ast.Variable v]
      else Ast.Variable v
  | (Ast.Constant c) => Ast.Constant c
  | (Ast.Appl []) => Ast.Appl []
  | (Ast.Appl (f :: xs)) => Ast.Appl (f :: map map-ast-variables xs)

fun kat-hom-tac ctxt n =
  let
    val rev-rules =
      map (fn thm => thm RS @{thm sym}) (Named-Theorems.get ctxt @{named-theorems kat-hom})
    in
      asm-full-simp-tac (put-simpset HOL-basic-ss ctxt addsimps rev-rules) n
    end
  end
⟩⟩

method-setup kat-hom = ⟨⟨
  Scan.succeed (fn ctxt => SIMPLE-METHOD (CHANGED (kat-hom-tac ctxt 1)))
⟩⟩

parse-ast-translation ⟨⟨
  let
    fun kat-tr ctxt [t] = map-ast-variables t
  in
    ([@{syntax-const -kat}, kat-tr])
  end
⟩⟩

named-theorems vcg verification condition generator rules

ML ⟨⟨
fun vcg-tac ctxt n =
  let
    fun vcg’ [] = no-tac
    | vcg’ (r :: rs) = resolve-tac ctxt [r] n ORELSE vcg’ rs;
  in
    REPEAT (CHANGED (kat-hom-tac ctxt n)
      THEN REPEAT (vcg’ (rev (Named-Theorems.get ctxt @{named-theorems vcg}))))
    THEN kat-hom-tac ctxt n
    THEN TRY (resolve-tac ctxt @{thms order-refl} n ORELSE asm-full-simp-tac (put-simpset HOL-basic-ss ctxt) n))
  end
⟩⟩
method-setup vcg = \[ \langle \langle Scan.succeed (fn ctxt => SIMPLE-METHOD (CHANGED (vcg-tac ctxt 1))) \rangle \rangle \\
locale dioid-tests = 
  fixes test :: 'a::boolean-algebra ⇒ 'b::dioid-one-zerol 
  and not :: 'b::dioid-one-zerol ⇒ 'b::dioid-one-zerol (−) 
  assumes test-sup [simp,kat-hom]: test (sup p q) = 'p + 'q 
  and test-inf [simp,kat-hom]: test (inf p q) = 'p · 'q 
  and test-top [simp,kat-hom]: test top = 1 
  and test-bot [simp,kat-hom]: test bot = 0 
  and test-not [simp,kat-hom]: test (− p) = '−'p 
  and test-iso-eq [kat-hom]: p ≤ q ←→ 'p ≤ 'q 
begin
notation test (ι)

lemma test-eq [kat-hom]: p = q ←→ 'p = 'q 
  by (metis eq-iff test-iso-eq)

lemma test-iso: p ≤ q ⇒ 'p ≤ 'q 
  by (simp add: test-iso-eq)

lemma test-meet-comm: 'p · q = q · 'p 
  by kat-hom (fact inf-commute)

lemmas test-one-top[simp] = test-iso[OF top-greatest, simplified]

lemma [simp]: '−'p + p = 1' 
  by kat-hom (metis compl-sup-top)

lemma [simp]: 'p + (−p) = 1' 
  by kat-hom (metis sup-compl-top)

lemma [simp]: (−'p) · p = 0' 
  by (metis inf.commute inf-compl-bot test-bot test-inf test-not)

lemma [simp]: 'p · (−p) = 0' 
  by (metis inf-compl-bot test-bot test-inf test-not)
end
locale kat = 
  fixes test :: 'a::boolean-algebra ⇒ 'b::kleene-algebra 
  and not :: 'b::kleene-algebra ⇒ 'b::kleene-algebra (!)
assumes is-dioid-tests: dioid-tests test not

sublocale kat ⊆ dioid-tests using is-dioid-tests.

calendar kat

begin

notation test (ι)

lemma test-eq [kat-hom]: p = q ↔ 'p = q'
  by (metis eq-iff test-iso-eq)

lemma test-iso: p ≤ q ⟹ 'p ≤ q'
  by (simp add: test-iso-eq)

lemma test-meet-comm: 'p · q = q · p'
  by kat-hom (fact inf-commute)

lemmas test-one-top[simp] = test-iso[OF top-greatest, simplified]

lemma test-star [simp]: 'p* = 1'
  by (metis star-subid test-iso test-top top-greatest)

lemmas [kat-hom] = test-star[symmetric]

lemma [simp]: '!p + p = 1'
  by kat-hom (metis compl-sup-top)

lemma [simp]: 'p + !p = 1'
  by kat-hom (metis sup-compl-top)

lemma [simp]: '!p · p = 0'
  by (metis inf-compl-bot test-bot test-inf test-not)

lemma [simp]: 'p · !p = 0'
  by (metis inf-compl-bot test-bot test-inf test-not)

definition hoare-triple :: 'b ⇒ 'b ⇒ 'b ⇒ bool (⌜⌜ - ⌝⌝) where
  \{p\} c \{q\} ≡ p·c ≤ c·q

declare hoare-triple-def[iff]

lemma hoare-triple-def-var: 'p·c ≤ c·q ℜ p·c!q = 0'
  apply (intro iffI antisym)
  apply (rule order-trans[of - 'c · q · !q'])
  apply (rule mult-isor[rule-format])
  apply (simp add: mult.assoc)
apply (simp add: mult.assoc[symmetric])
apply (rule order-trans[of - ‘p@c-!(q + q) ’])
apply simp
apply (simp only: distrib-left add-zerol)
apply (rule order-trans[of - ‘1 · c · q’])
apply (simp only: mult.assoc)
apply (rule mult-isol[rule-format])
by simp-all

lemmas [intro!] = star-sim2[rule-format]

lemma hoare-weakening: p ≤ p′ ⇒ q ≤ q ⇒ ’p’ c ’q’ ⇒ ’q’ ⇒ ’c’ ’p’ c ’q’
by auto (metis mult-isol mult-order-trans test-iso)

lemma hoare-star: ’p’ c ’p’ ⇒ ’p’ c* ’p’
by auto

lemmas [vcg] = hoare-weakening[of order-refl - hoare-star]

lemma hoare-test [vcg]: ’p’ t ≤ q ⇒ ’p’ t ’q’
by auto (metis inf-le2 le-inf-iff test-inf test-iso-eq)

lemma hoare-mult [vcg]: ’p’ x ’r’ ⇒ ’p’ y ’q’ ⇒ ’p’ x · y ’q’
proof auto
  assume [simp]: ’p’ x ≤ x · r’ and [simp]: ’r’ y ≤ y · q’
  have ’p’ x · y ≤ x · r · y’
    by (auto simp add: mult.assoc[symmetric] intro!: mult-isol[rule-format])
  also have ’... ≤ x · y · q’
    by (auto simp add: mult.assoc intro!: mult-isol[rule-format])
  finally show ’p’ (x · y) ≤ x · y · q’.
qed

lemma [simp]: ’!p · !p = !p’
by (metis inf.idem test-inf test-not)

lemma hoare-plus [vcg]: ’p’ x ’q’ ⇒ ’p’ y ’q’ ⇒ ’p’ x + y ’q’
by (auto simp add: distrib-left distrib-right add-iso-var)

definition While :: ’b ⇒ ’b ⇒ ’b (While - Do - End [50,50] 51) where
While t Do c End = (t-c)* !t

lemma hoare-while: ’p’ t ’p’ ⇒ ’p’ While t Do c End ’q’ ⇒ ’p’ ’q’ While t Do c End ’p’
unfolding While-def by vcg (metis inf-commute order-refl)

lemma [vcg]: ’p’ t ’p’ ⇒ ’!t · p ≤ q’ ⇒ ’p’ ’q’ While t Do c End ’p’
by (metis hoare-weakening hoare-while order-refl test-inf test-iso-eq test-not)

definition If :: ’b ⇒ ’b ⇒ ’b (If - Then - Else - [50,50,50] 51) where
If p Then c1 Else c2 ≡ p·c1 + !p·c2
lemma hoare-if \[ \{ \langle p \cdot \mathit{t} \rangle \} c1 \{ \langle q \rangle \} \implies \{ \langle p \cdot \mathit{!t} \rangle \} c2 \{ \langle q \rangle \} \implies \{ \langle p \rangle \} \] If \( \mathit{t} \) Then \( c1 \) Else \( c2 \) \{ \langle q \rangle \}

unfolding If-def by vcg assumption

end
end

13 Two sorted Demonic Refinement Algebras

thephy DRA2
    imports ../DRA
begin

As an alternative to the one-sorted implementation of demonic refinement algebra with tests, we provide a two-sorted, more conventional one. This alternative can be developed further along the lines of the one-sorted implementation.

syntax -dra :: 'a \Rightarrow 'a ('\rightarrow')

named-theorems kat-hom KAT test homomorphism rules

ML ⟨⟨
val dra-test-vars = [p,q,r,s,t,p',q',r',s',t',p'',q'',r'',s'',t'']

fun map-ast-variables ast =
  case ast of
    (Ast.Variable v) =>
      if exists (fn tv => tv = v) dra-test-vars
        then Ast.Appl [Ast.Variable test, Ast.Variable v]
        else Ast.Variable v
    | (Ast.Constant c) => Ast.Constant c
    | (Ast.Appl []) => Ast.Appl []
    | (Ast.Appl (f :: xs)) => Ast.Appl (f :: map-ast-variables xs)

fun dra-hom-tac ctxt n =
  let
    val rev-rules =
      map (fn thm => thm RS \{ thm sym \}) (Named-Theorems.get ctxt \{ named-theorems kat-hom \})
  in
    asm-full-simp-tac (put-simpset HOL-basic-ss ctxt addsimps rev-rules) n
  end
⟩⟩

method-setup kat-hom = ⟨⟨
  Scan.succeed (fn ctxt => SIMPLE-METHOD (CHANGED (dra-hom-tac ctxt 1)))
⟩⟩
let
  fun dra-tr ctxt [t] = map-ast-variables t
in [[[@{syntax-const -dra}, dra-tr]] end

named-theorems vcg verification condition generator rules

ML

fun vcg-tac ctxt n =
let
  fun vcg' [] = no-tac
  | vcg' (r :: rs) = resolve-tac ctxt [r] n ORELSE vcg' rs;
  in REPEAT (CHANGED
               (dra-hom-tac ctxt n
                THEN REPEAT (vcg' (rev (Named-Theorems.get ctxt @[named-theorems vcg])))
                THEN dra-hom-tac ctxt n
                THEN TRY (resolve-tac ctxt @[thms order-refl] n ORELSE asm-full-simp-tac
                           (put-simpset HOL-basic-ss ctxt) n)))
  end

method-setup vcg = Scan.succeed (fn ctxt => SIMPLE-METHOD (CHANGED (vcg-tac ctxt 1)))

locale drat =
  fixes test :: 'a::boolean-algebra ⇒ 'b::dra
  and not :: 'b::dra ⇒ 'b::dra (!)
  assumes test-sup [simp,kat-hom]: test (sup p q) = 'p + q'
  and test-inf [simp,kat-hom]: test (inf p q) = 'p · q'
  and test-top [simp,kat-hom]: test top = 1
  and test-bot [simp,kat-hom]: test bot = 0
  and test-not [simp,kat-hom]: test (¬ p) = '!'p'
  and test-iso-eq [kat-hom]: p ≤ q ⇔ 'p ≤ q'

begin

notation test (ι)

lemma test-eq [kat-hom]: p = q ⇔ 'p = q'
  by (metis eq_iff test-iso-eq)

lemma test-iso: p ≤ q ⇒ 'p ≤ q'
  by (simp add: test-iso-eq)
lemma test-meet-comm: 'p · q = q · p'
  by kat-hom (fact inf-commute)

lemmas test-one-top[simp] = test-iso[OF top-greatest, simplified]

lemma test-star [simp]: 'p* = 1'
  by (metis star-subid test-iso test-top top-greatest)

lemmas [kat-hom] = test-star[symmetric]

lemma test-comp-add1 [simp]: '!p + p = 1'
  by kat-hom (metis compl-sup-top)

lemma test-comp-add2 [simp]: 'p + !p = 1'
  by kat-hom (metis sup-compl-top)

lemma test-comp-mult1 [simp]: '!p · p = 0'
  by (metis inf.commute inf-compl-bot test-bot test-inf test-not)

lemma test-comp-mult2 [simp]: 'p · !p = 0'
  by (metis inf-compl-bot test-bot test-inf test-not)

lemma test-eq1:
  'y ≤ x' ←→ 'p · y ≤ x' ∧ '!p · y ≤ x'
  apply standard
  apply (metis mult-isol-var mult-onel test-not test-one-top)
  apply (subgoal-tac '(p + !p) · y ≤ x')
  apply (metis mult-onel sup-compl-top test-not test-sup test-top)
  apply (metis add-lub distrib-right)
  done

lemma 'p · x = p · x · q' ⇒ 'p · x · !q = 0'
  nitpick oops

lemma test-eq1: 'p · x · !q = 0' ⇒ 'p · x = p · x · q'
  by (metis add-0-left distrib-left mult-onel test-comp-add1)

lemma test2: 'p · q · p = p · q'
  by (metis inf.commute inf.left-idem test-inf)

lemma test3: 'p · q · !p = 0'
  by (metis inf.assoc inf.idem inf.left-commute inf-compl-bot test-bot test-inf test-not)

lemma test4: '!p · q · p = 0'
  by (metis double-compl test3 test-not)

lemma total-correctness:
  'p · x · !q = 0' ←→ 'x · !q ≤ !p · ⊤'
  apply standard
  apply (metis mult.assoc test-eq1 top-elim zero-least)
  apply (metis annil test-comp-mult2 zero-unique mult.assoc mult-isol)
done

lemma test-iteration-sim: ‘p·x ≤ x·p’ ⟹ ‘p·x^∞ ≤ x^∞·p’
   by (metis iteration-sim)

lemma test-iteration-annir: ‘!(p·(p·x)^∞) = !(p·x)^∞’
   by (metis (no-types) monoid-add-class.add.left-neutral double-compl iteration-idep
    monoid-mult-class.mult.right-neutral test-comp-add2 test-inf test-not top-elim total-correctness)

end

end

References


