

Kleene Algebra with Tests and Demonic Refinement Algebras

Alasdair Armstrong Victor B. F. Gomes Georg Struth

December 17, 2016

Abstract

We formalise Kleene algebra with tests (KAT) and demonic refinement algebra (DRA) with tests in Isabelle/HOL. KAT is relevant for program verification and correctness proofs in the partial correctness setting. 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.

Contents

1	Test Dioids	2
1.1	Test Monoids	2
1.2	Test Near-Semirings	7
1.3	Test Near Semirings with Distributive Tests	16
1.4	Test Pre-dioids	18
1.5	Test Semirings	20
2	Pre-Conway Algebra with Tests	21
3	Kleene Algebra with Tests	22
4	Demonic Refinement Algebra with Tests	25
5	Models for Demonic Refinement Algebra with Tests	30
6	Models for Kleene Algebra with Tests	33
7	Transformation Theorem for while Loops	34
8	Propositional Hoare Logic	38
9	Propositional Hoare Logic	42

10 Two sorted Kleene Algebra with Tests	43
11 Two sorted Demonic Refinement Algebras	48

1 Test Dioids

```

theory Test-Dioid
  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 complementation operator. This allows us to use tests in weak settings, such as Probabilistic Kleene Algebra and Demonic Refinement Algebra.

1.1 Test Monoids

```

class n-op =
  fixes n-op :: 'a  $\Rightarrow$  'a (n - [90] 91)

class test-monoid = monoid-mult + n-op +
  assumes tm1 [simp]: n n 1 = 1
  and tm2 [simp]: n x  $\cdot$  n n x = n 1
  and tm3: n x  $\cdot$  n (n n z  $\cdot$  n n y) = n (n (n x  $\cdot$  n y)  $\cdot$  n (n x  $\cdot$  n z))

begin

```

```

definition a-zero :: 'a (o) where
  o  $\equiv$  n 1

```

```

abbreviation t x  $\equiv$  n n x

```

```

definition n-add-op :: 'a  $\Rightarrow$  'a  $\Rightarrow$  'a (infixl  $\oplus$  65) where
  x  $\oplus$  y  $\equiv$  n (n x  $\cdot$  n y)

```

```

lemma n 1  $\cdot$  x = n 1
oops

```

```

lemma x  $\cdot$  n 1 = n 1
oops

```

```

lemma n 1  $\cdot$  x = n 1  $\implies$  n x  $\cdot$  y  $\cdot$  t z = n 1  $\implies$  n x  $\cdot$  y = n x  $\cdot$  y  $\cdot$  n z
oops

```

```

lemma n-t-closed [simp]: t (n x) = n x
proof -

```

have $\bigwedge x y. n x \cdot n (t (n x) \cdot t y) = t (n x \cdot n y)$
by (*simp add: local.tm3*)
thus *?thesis*
by (*metis (no-types) local.mult-1-right local.tm1 local.tm2 local.tm3 mult-assoc*)
qed

lemma *mult-t-closed* [*simp*]: $t (n x \cdot n y) = n x \cdot n y$
by (*metis local.mult-1-right local.tm1 local.tm2 local.tm3 n-t-closed*)

lemma *n-comm-var*: $n (n x \cdot n y) = n (n y \cdot n x)$
by (*metis local.mult-1-left local.tm1 local.tm3 n-t-closed*)

lemma *n-comm*: $n x \cdot n y = n y \cdot n x$
using *mult-t-closed n-comm-var* **by** *fastforce*

lemma *huntington1* [*simp*]: $n (n (n n x \cdot n y) \cdot n (n n x \cdot n n y)) = n n x$
by (*metis local.mult-1-right local.tm1 local.tm2 local.tm3*)

lemma *huntington2* [*simp*]: $n (n x \oplus n n y) \oplus n (n x \oplus n y) = n n x$
by (*simp add: n-add-op-def*)

lemma *add-assoc*: $n x \oplus (n y \oplus n z) = (n x \oplus n y) \oplus n z$
by (*simp add: mult-assoc n-add-op-def*)

lemma *t-mult-closure*: $t x = x \implies t y = y \implies t (x \cdot y) = x \cdot y$
by (*metis mult-t-closed*)

lemma *n-t-compl* [*simp*]: $n x \oplus t x = 1$
using *n-add-op-def local.tm1 local.tm2* **by** *presburger*

lemma *zero-least1* [*simp*]: $o \oplus n x = n x$
by (*simp add: a-zero-def n-add-op-def*)

lemma *zero-least2* [*simp*]: $o \cdot n x = o$
by (*metis a-zero-def local.tm2 local.tm3 mult-assoc mult-t-closed zero-least1*)

lemma *zero-least3* [*simp*]: $n x \cdot o = o$
using *a-zero-def n-comm zero-least2* **by** *fastforce*

lemma *one-greatest1* [*simp*]: $1 \oplus n x = 1$
by (*metis (no-types) a-zero-def local.tm1 n-add-op-def n-comm-var zero-least3*)

lemma *one-greatest2* [*simp*]: $n x \oplus 1 = 1$
by (*metis n-add-op-def n-comm-var one-greatest1*)

lemma *n-add-idem* [*simp*]: $n x \oplus n x = n x$
by (*metis huntington1 local.mult-1-right n-t-closed n-t-compl n-add-op-def*)

lemma *n-mult-idem* [*simp*]: $n x \cdot n x = n x$

by (*metis mult-t-closed n-add-idem n-add-op-def*)

lemma *n-preserve* [*simp*]: $n x \cdot n y \cdot n x = n y \cdot n x$
by (*metis mult-assoc n-comm n-mult-idem*)

lemma *n-preserve2* [*simp*]: $n x \cdot n y \cdot t x = o$
by (*metis a-zero-def local.tm2 mult-assoc n-comm zero-least3*)

lemma *de-morgan1* [*simp*]: $n (n x \cdot n y) = t x \oplus t y$
by (*simp add: n-add-op-def*)

lemma *de-morgan4* [*simp*]: $n (t x \oplus t y) = n x \cdot n y$
using *n-add-op-def mult-t-closed n-t-closed* **by** *presburger*

lemma *n-absorb1* [*simp*]: $n x \oplus n x \cdot n y = n x$
by (*metis local.mult-1-right local.tm1 local.tm3 one-greatest2 n-add-op-def*)

lemma *n-absorb2* [*simp*]: $n x \cdot (n x \oplus n y) = n x$
by (*metis mult-t-closed n-absorb1 n-add-op-def*)

lemma *n-distrib1*: $n x \cdot (n y \oplus n z) = (n x \cdot n y) \oplus (n x \cdot n z)$
using *local.tm3 n-comm-var n-add-op-def* **by** *presburger*

lemma *n-distrib1-opp*: $(n x \oplus n y) \cdot n z = (n x \cdot n z) \oplus (n y \cdot n z)$
using *n-add-op-def n-comm n-distrib1* **by** *presburger*

lemma *n-distrib2*: $n x \oplus n y \cdot n z = (n x \oplus n y) \cdot (n x \oplus n z)$
by (*metis mult-t-closed n-distrib1 n-mult-idem n-add-op-def*)

lemma *n-distrib2-opp*: $n x \cdot n y \oplus n z = (n x \oplus n z) \cdot (n y \oplus n z)$
by (*metis de-morgan1 mult-t-closed n-distrib1-opp n-add-op-def*)

definition *ts-ord* :: $'a \Rightarrow 'a \Rightarrow \text{bool}$ (**infix** \sqsubseteq 50) **where**
 $x \sqsubseteq y = (n x \cdot n y = n y)$

lemma *ts-ord-alt*: $n x \sqsubseteq n y \iff n x \oplus n y = n y$
by (*metis mult-t-closed n-t-closed ts-ord-def n-add-op-def*)

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

lemma *ts-ord-trans*: $n x \sqsubseteq n y \implies n y \sqsubseteq n z \implies n x \sqsubseteq n z$
by (*metis mult-assoc ts-ord-def*)

lemma *ts-ord-antisym*: $n x \sqsubseteq n y \implies n y \sqsubseteq n x \implies n x = n y$
by (*metis n-add-idem n-comm ts-ord-def n-add-op-def*)

lemma *ts-ord-mult-isol*: $n x \sqsubseteq n y \implies n z \cdot n x \sqsubseteq n z \cdot n y$
proof –

assume $n x \sqsubseteq n y$
hence $n (n (n z \cdot n x) \cdot n (n z \cdot n y)) = n z \cdot n y$
by (*metis mult-t-closed n-add-idem n-distrib1 ts-ord-def n-add-op-def*)
thus *?thesis*
by (*metis mult-t-closed ts-ord-def*)
qed

lemma *ts-ord-mult-isor*: $n x \sqsubseteq n y \implies n x \cdot n z \sqsubseteq n y \cdot n z$
using *n-comm ts-ord-mult-isol* **by** *auto*

lemma *ts-ord-add-isol*: $n x \sqsubseteq n y \implies n z \oplus n x \sqsubseteq n z \oplus n y$
by (*metis mult-assoc mult-t-closed n-mult-idem ts-ord-def n-add-op-def*)

lemma *ts-ord-add-isor*: $n x \sqsubseteq n y \implies n x \oplus n z \sqsubseteq n y \oplus n z$
using *n-add-op-def n-comm ts-ord-add-isol* **by** *presburger*

lemma *ts-ord-anti*: $n x \sqsubseteq n y \implies t y \sqsubseteq t x$
by (*metis n-absorb2 n-add-idem n-comm ts-ord-def n-add-op-def*)

lemma *ts-ord-anti-iff*: $n x \sqsubseteq n y \iff t y \sqsubseteq t x$
using *ts-ord-anti* **by** *force*

lemma *zero-ts-ord*: $o \sqsubseteq n x$
by (*simp add: a-zero-def ts-ord-def*)

lemma *n-subid*: $n x \sqsubseteq 1$
by (*simp add: a-zero-def[symmetric] ts-ord-def*)

lemma *n-mult-lb1*: $n x \cdot n y \sqsubseteq n x$
by (*metis (no-types) local.mult-1-right local.tm1 n-comm n-subid ts-ord-mult-isor*)

lemma *n-mult-lb2*: $n x \cdot n y \sqsubseteq n y$
by (*metis n-comm n-mult-lb1*)

lemma *n-mult-glbI*: $n z \sqsubseteq n x \implies n z \sqsubseteq n y \implies n z \sqsubseteq n x \cdot n y$
by (*metis n-t-closed ts-ord-anti-iff ts-ord-def ts-ord-mult-isol*)

lemma *n-mult-glb*: $n z \sqsubseteq n x \wedge n z \sqsubseteq n y \iff n z \sqsubseteq n x \cdot n y$
by (*metis mult-t-closed n-mult-glbI n-mult-lb1 n-mult-lb2 ts-ord-trans*)

lemma *n-add-ub1*: $n x \sqsubseteq n x \oplus n y$
by (*metis (no-types) n-absorb2 n-mult-lb2 n-add-op-def*)

lemma *n-add-ub2*: $n y \sqsubseteq n x \oplus n y$
by (*metis n-add-ub1 n-comm-var n-add-op-def*)

lemma *n-add-lubI*: $n x \sqsubseteq n z \implies n y \sqsubseteq n z \implies n x \oplus n y \sqsubseteq n z$
by (*metis ts-ord-add-isol ts-ord-alt*)

lemma *n-add-lub*: $n x \sqsubseteq n z \wedge n y \sqsubseteq n z \longleftrightarrow n x \oplus n y \sqsubseteq n z$
by (*metis n-add-lubI n-add-op-def n-add-ub1 n-add-ub2 ts-ord-trans*)

lemma *n-galois1*: $n x \sqsubseteq n y \oplus n z \longleftrightarrow n x \cdot t y \sqsubseteq n z$

proof

assume $n x \sqsubseteq n y \oplus n z$
hence $n x \cdot t y \sqsubseteq (n y \oplus n z) \cdot t y$
by (*metis n-add-op-def ts-ord-mult-isor*)
also have $\dots = n y \cdot t y \oplus n z \cdot t y$
using *n-add-op-def local.tm3 n-comm* **by** *presburger*
also have $\dots = n z \cdot t y$
using *a-zero-def local.tm2 n-distrib2 zero-least1* **by** *presburger*
finally show $n x \cdot t y \sqsubseteq n z$
by (*metis mult-t-closed n-mult-glb*)

next

assume $a: n x \cdot t y \sqsubseteq n z$
have $n x = t (n x \cdot (n y \oplus t y))$
using *local.mult-1-right n-t-closed n-t-compl* **by** *presburger*
also have $\dots = t (n x \cdot n y \oplus n x \cdot t y)$
using *n-distrib1* **by** *presburger*
also have $\dots \sqsubseteq t (n y \oplus n x \cdot t y)$
by (*metis calculation local.mult-1-right mult-t-closed n-add-op-def n-add-ub2 n-distrib2 n-t-compl*)
finally show $n x \sqsubseteq n y \oplus n z$
by (*metis (no-types, hide-lams) a mult-t-closed ts-ord-add-isol ts-ord-trans n-add-op-def*)

qed

lemma *n-galois2*: $n x \sqsubseteq t y \oplus n z \longleftrightarrow n x \cdot n y \sqsubseteq n z$
by (*metis n-galois1 n-t-closed*)

lemma *n-distrib-alt*: $n x \cdot n z = n y \cdot n z \implies n x \oplus n z = n y \oplus n z \implies n x = n y$

proof –

assume $a: n x \cdot n z = n y \cdot n z$ **and** $b: n x \oplus n z = n y \oplus n z$
have $n x = n x \oplus n x \cdot n z$
using *n-absorb1* **by** *presburger*
also have $\dots = n x \oplus n y \cdot n z$
using *a* **by** *presburger*
also have $\dots = (n x \oplus n y) \cdot (n x \oplus n z)$
using *n-distrib2* **by** *blast*
also have $\dots = (n y \oplus n x) \cdot (n y \oplus n z)$
using *n-add-op-def b n-comm* **by** *presburger*
also have $\dots = n y \cdot (n x \oplus n z)$
by (*metis a b n-distrib1 n-distrib2 n-mult-idem*)
also have $\dots = n y \cdot (n y \oplus n z)$
using *b* **by** *presburger*
finally show $n x = n y$
using *n-absorb2* **by** *presburger*

qed

lemma *n-dist-var1*: $(n\ x \oplus n\ y) \cdot (t\ x \oplus n\ z) = t\ x \cdot n\ y \oplus n\ x \cdot n\ z$

proof –

have $(n\ x \oplus n\ y) \cdot (t\ x \oplus n\ z) = n\ x \cdot t\ x \oplus n\ y \cdot t\ x \oplus (n\ x \cdot n\ z \oplus n\ y \cdot n\ z)$

using *n-add-op-def n-distrib1 n-distrib1-opp* **by** *presburger*

also have $\dots = t\ x \cdot n\ y \oplus (n\ x \cdot n\ z \oplus (n\ x \oplus t\ x) \cdot t\ (n\ y \cdot n\ z))$

using *n-add-op-def local.mult-1-left local.tm1 local.tm2 n-comm n-t-closed* **by** *presburger*

also have $\dots = t\ x \cdot n\ y \oplus (n\ x \cdot n\ z \oplus (n\ x \cdot n\ y \cdot n\ z \oplus t\ x \cdot n\ y \cdot n\ z))$

by (*metis mult-assoc mult-t-closed n-distrib1-opp*)

also have $\dots = (t\ x \cdot n\ y \oplus t\ x \cdot n\ y \cdot n\ z) \oplus (n\ x \cdot n\ z \oplus n\ x \cdot n\ y \cdot n\ z)$

proof –

have $f1: \bigwedge a\ aa\ ab. n\ n\ (n\ (a::'a) \cdot (n\ aa \cdot n\ ab)) = n\ a \cdot (n\ aa \cdot n\ ab)$

by (*metis (full-types) mult-t-closed*)

have $\bigwedge a\ aa\ ab. n\ (a::'a) \cdot (n\ aa \cdot n\ ab) = n\ aa \cdot (n\ ab \cdot n\ a)$

by (*metis mult-assoc n-comm*)

hence $n\ (n\ (n\ y \cdot n\ n\ x) \cdot n\ n\ (n\ (n\ x \cdot n\ z) \cdot (n\ (n\ x \cdot n\ y \cdot n\ z) \cdot n\ (n\ y \cdot n\ n\ x \cdot n\ z)))) = n\ (n\ (n\ y \cdot n\ n\ x) \cdot n\ (n\ y \cdot n\ n\ x \cdot n\ z) \cdot (n\ (n\ x \cdot n\ z) \cdot n\ (n\ x \cdot n\ y \cdot n\ z)))$

using *f1 mult-assoc* **by** *presburger*

thus *?thesis*

using *mult-t-closed n-add-op-def n-comm* **by** *presburger*

qed

also have $\dots = (t\ x \cdot n\ y \oplus t\ (t\ x \cdot n\ y) \cdot n\ z) \oplus (n\ x \cdot n\ z \oplus t\ (n\ x \cdot n\ z) \cdot n\ y)$

using *mult-assoc mult-t-closed n-comm* **by** *presburger*

also have $\dots = (t\ x \cdot n\ y \cdot (1 \oplus n\ z)) \oplus (n\ x \cdot n\ z \cdot (1 \oplus n\ y))$

by (*metis a-zero-def n-add-op-def local.mult-1-right local.tm1 mult-t-closed n-absorb1 zero-least2*)

finally show *?thesis*

by *simp*

qed

lemma *n-dist-var2*: $n\ (n\ x \cdot n\ y \oplus t\ x \cdot n\ z) = n\ x \cdot t\ y \oplus t\ x \cdot t\ z$

by (*metis (no-types) n-add-op-def n-dist-var1 n-t-closed*)

end

1.2 Test Near-Semirings

class *test-near-semiring-zero1* = *ab-near-semiring-one-zero1* + *n-op* + *plus-ord* +

assumes *test-one* [*simp*]: $n\ n\ 1 = 1$

and *test-mult* [*simp*]: $n\ n\ (n\ x \cdot n\ y) = n\ x \cdot n\ y$

and *test-mult-comp* [*simp*]: $n\ x \cdot n\ n\ x = 0$

and *test-de-morgan* [*simp*]: $n\ (n\ n\ x \cdot n\ n\ y) = n\ x + n\ y$

begin

```

lemma n-zero [simp]:  $n\ 0 = 1$ 
proof -
  have  $n\ 0 = n\ (n\ 1 \cdot n\ n\ 1)$ 
    using local.test-mult-comp by presburger
  also have  $\dots = n\ (n\ 1 \cdot 1)$ 
    by simp
  finally show ?thesis
    by simp
qed

```

```

lemma n-one [simp]:  $n\ 1 = 0$ 
proof -
  have  $n\ 1 = n\ 1 \cdot 1$ 
    by simp
  also have  $\dots = n\ 1 \cdot n\ n\ 1$ 
    by simp
  finally show ?thesis
    using test-mult-comp by metis
qed

```

```

lemma one-idem [simp]:  $1 + 1 = 1$ 
proof -
  have  $1 + 1 = n\ n\ 1 + n\ n\ 1$ 
    by simp
  also have  $\dots = n\ (n\ n\ n\ 1 \cdot n\ n\ n\ 1)$ 
    using local.test-de-morgan by presburger
  also have  $\dots = n\ (0 \cdot n\ n\ n\ 1)$ 
    by simp
  also have  $\dots = n\ 0$ 
    by simp
  finally show ?thesis
    by simp
qed

```

```

subclass near-dioid-one-zero
proof
  fix  $x$ 
  have  $x + x = (1 + 1) \cdot x$ 
    using local.distrib-right' local.mult-onel by presburger
  also have  $\dots = 1 \cdot x$ 
    by simp
  finally show  $x + x = x$ 
    by simp
qed

```

```

lemma t-n-closed [simp]:  $n\ n\ (n\ x) = n\ x$ 
proof -
  have  $n\ n\ (n\ x) = n\ (n\ n\ x \cdot 1)$ 
    by simp

```


also have $\dots = n (n n x \cdot n n 1)$
 by *simp*
also have $\dots = n x + n 1$
 using *local.test-de-morgan* by *presburger*
also have $\dots = n x + 0$
 by *simp*
finally show *?thesis*
 by *simp*
qed

lemma *t-de-morgan-var1* [*simp*]: $n (n x \cdot n y) = n n x + n n y$
 by (*metis local.test-de-morgan t-n-closed*)

lemma *n-mult-comm*: $n x \cdot n y = n y \cdot n x$
proof –
have $n x \cdot n y = n n (n x \cdot n y)$
 by (*metis local.test-mult*)
also have $\dots = n (n n x + n n y)$
 by *simp*
also have $\dots = n (n n y + n n x)$
 by (*metis add-commute*)
also have $\dots = n n (n y \cdot n x)$
 by *simp*
finally show *?thesis*
 by (*metis local.test-mult*)
qed

lemma *tm3'*: $n x \cdot n (n n z \cdot n n y) = n (n (n x \cdot n y) \cdot n (n x \cdot n z))$
proof –
have $n x \cdot n (n n z \cdot n n y) = n (n n y \cdot n n z) \cdot n x$
 using *n-mult-comm* by *presburger*
also have $\dots = (n y + n z) \cdot n x$
 by *simp*
also have $\dots = n y \cdot n x + n z \cdot n x$
 by *simp*
also have $\dots = n n (n x \cdot n y) + n n (n x \cdot n z)$
 by (*metis local.test-mult n-mult-comm*)
finally show *?thesis*
 by (*metis t-de-morgan-var1*)
qed

subclass *test-monoid*

proof
show $n n 1 = 1$
 by *simp*
show $\bigwedge x. n x \cdot n n x = n 1$
 by *simp*
show $\bigwedge x z y. n x \cdot n (n n z \cdot n n y) = n (n (n x \cdot n y) \cdot n (n x \cdot n z))$
 using *tm3'* by *blast*

qed

lemma *ord-transl* [*simp*]: $n x \leq n y \longleftrightarrow n x \sqsubseteq n y$
by (*simp add: local.join.sup.absorb-iff2 local.n-add-op-def local.ts-ord-alt*)

lemma *add-transl* [*simp*]: $n x + n y = n x \oplus n y$
by (*simp add: local.n-add-op-def*)

lemma *zero-trans*: $0 = o$
by (*metis local.a-zero-def n-one*)

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

notation *n-op* (!- [101] 100)

lemma *test-prop*: $(\forall x. \text{test } x \longrightarrow P x) \longleftrightarrow (\forall x. P (t x))$
by (*metis test-def t-n-closed*)

lemma *test-propI*: $\text{test } x \Longrightarrow P x \Longrightarrow P (t x)$
by (*simp add: test-def*)

lemma *test-propE* [*elim!*]: $\text{test } x \Longrightarrow P (t x) \Longrightarrow P x$
by (*simp add: test-def*)

lemma *test-comp-closed* [*simp*]: $\text{test } p \Longrightarrow \text{test } (!p)$
by (*simp add: test-def*)

lemma *test-double-comp-var*: $\text{test } p \Longrightarrow p = !(!p)$
by *auto*

lemma *test-mult-closed*: $\text{test } p \Longrightarrow \text{test } q \Longrightarrow \text{test } (p \cdot q)$
by (*metis local.test-mult test-def*)

lemma *t-add-closed* [*simp*]: $t (n x + n y) = n x + n y$
by (*metis local.test-de-morgan t-n-closed*)

lemma *test-add-closed*: $\text{test } p \Longrightarrow \text{test } q \Longrightarrow \text{test } (p + q)$
by (*metis t-add-closed test-def*)

lemma *test-mult-comm-var*: $\text{test } p \Longrightarrow \text{test } q \Longrightarrow p \cdot q = q \cdot p$
using *n-mult-comm* **by** *auto*

lemma *t-zero* [*simp*]: $t 0 = 0$
by *simp*

lemma *test-zero-var*: $\text{test } 0$
by (*simp add: test-def*)

lemma *test-one-var*: $test\ 1$
by (*simp add: test-def*)

lemma *test-preserve*: $test\ p \implies test\ q \implies p \cdot q \cdot p = q \cdot p$
by *auto*

lemma *test-preserve2*: $test\ p \implies test\ q \implies p \cdot q \cdot !p = 0$
by (*metis local.a-zero-def local.n-preserve2 n-one test-double-comp-var*)

lemma *n-subid'*: $n\ x \leq 1$
using *local.n-subid n-zero ord-transl* **by** *blast*

lemma *test-subid*: $test\ p \implies p \leq 1$
using *n-subid'* **by** *auto*

lemma *test-mult-idem-var* [*simp*]: $test\ p \implies p \cdot p = p$
by *auto*

lemma *n-add-comp* [*simp*]: $n\ x + t\ x = 1$
by *simp*

lemma *n-add-comp-var* [*simp*]: $t\ x + n\ x = 1$
by (*simp add: add-commute*)

lemma *test-add-comp* [*simp*]: $test\ p \implies p + !p = 1$
using *n-add-comp* **by** *fastforce*

lemma *test-comp-mult1* [*simp*]: $test\ p \implies !p \cdot p = 0$
by *auto*

lemma *test-comp-mult2* [*simp*]: $test\ p \implies p \cdot !p = 0$
using *local.test-mult-comp* **by** *fastforce*

lemma *test-comp*: $test\ p \implies \exists q. test\ q \wedge p + q = 1 \wedge p \cdot q = 0$
using *test-add-comp test-comp-closed test-comp-mult2* **by** *blast*

lemma *n-absorb1'* [*simp*]: $n\ x + n\ x \cdot n\ y = n\ x$
by (*metis add-transl local.de-morgan4 local.n-absorb1*)

lemma *test-absorb1* [*simp*]: $test\ p \implies test\ q \implies p + p \cdot q = p$
by *auto*

lemma *n-absorb2'* [*simp*]: $n\ x \cdot (n\ x + n\ y) = n\ x$
by *simp*

lemma *test-absorb2*: $test\ p \implies test\ q \implies p \cdot (p + q) = p$
by *auto*

lemma *n-distrib-left*: $n x \cdot (n y + n z) = (n x \cdot n y) + (n x \cdot n z)$
by (*metis* (*no-types*) *add-transl local.de-morgan4 local.n-distrib1*)

lemma *test-distrib-left*: $test p \implies test q \implies test r \implies p \cdot (q + r) = p \cdot q + p \cdot r$
using *n-distrib-left* **by** *auto*

lemma *de-morgan1'*: $test p \implies test q \implies !p + !q = !(p \cdot q)$
by *auto*

lemma *n-de-morgan-var2* [*simp*]: $n (n x + n y) = t x \cdot t y$
by (*metis local.test-de-morgan local.test-mult*)

lemma *n-de-morgan-var3* [*simp*]: $n (t x + t y) = n x \cdot n y$
by *simp*

lemma *de-morgan2*: $test p \implies test q \implies !p \cdot !q = !(p + q)$
by *auto*

lemma *de-morgan3*: $test p \implies test q \implies !(!p + !q) = p \cdot q$
using *local.de-morgan1' local.t-mult-closure test-def* **by** *auto*

lemma *de-morgan4'*: $test p \implies test q \implies !(!p \cdot !q) = p + q$
by *auto*

lemma *n-add-distr*: $n x + (n y \cdot n z) = (n x + n y) \cdot (n x + n z)$
by (*metis add-transl local.n-distrib2 n-de-morgan-var3*)

lemma *test-add-distr*: $test p \implies test q \implies test r \implies p + q \cdot r = (p + q) \cdot (p + r)$
using *n-add-distr* **by** *fastforce*

lemma *n-add-distl*: $(n x \cdot n y) + n z = (n x + n z) \cdot (n y + n z)$
by (*simp add: add-commute n-add-distr*)

lemma *test-add-distl-var*: $test p \implies test q \implies test r \implies p \cdot q + r = (p + r) \cdot (q + r)$
by (*simp add: add-commute test-add-distr*)

lemma *n-ord-def-alt*: $n x \leq n y \iff n x \cdot n y = n x$
by (*metis* (*no-types*) *local.join.sup.absorb-iff2 local.n-absorb2' local.n-mult-lb2 ord-transl*)

lemma *test-leq-mult-def-var*: $test p \implies test q \implies p \leq q \iff p \cdot q = p$
using *n-ord-def-alt* **by** *auto*

lemma *n-anti*: $n x \leq n y \implies t y \leq t x$
using *local.ts-ord-anti ord-transl* **by** *blast*

lemma *n-anti-iff*: $n x \leq n y \longleftrightarrow t y \leq t x$
using *n-anti* **by** *fastforce*

lemma *test-comp-anti-iff*: $test p \implies test q \implies p \leq q \longleftrightarrow !q \leq !p$
using *n-anti-iff* **by** *auto*

lemma *n-restrictl*: $n x \cdot y \leq y$
using *local.mult-isor n-subid'* **by** *fastforce*

lemma *test-restrictl*: $test p \implies p \cdot x \leq x$
by (*auto simp: n-restrictl*)

lemma *n-mult-lb1'*: $n x \cdot n y \leq n x$
by (*simp add: local.join.sup.orderI*)

lemma *test-mult-lb1*: $test p \implies test q \implies p \cdot q \leq p$
by (*auto simp: n-mult-lb1'*)

lemma *n-mult-lb2'*: $n x \cdot n y \leq n y$
by (*fact local.n-restrictl*)

lemma *test-mult-lb2*: $test p \implies test q \implies p \cdot q \leq q$
by (*rule test-restrictl*)

lemma *n-mult-glbI'*: $n z \leq n x \implies n z \leq n y \implies n z \leq n x \cdot n y$
by (*metis mult-isor n-ord-def-alt*)

lemma *test-mult-glbI*: $test p \implies test q \implies test r \implies p \leq q \implies p \leq r \implies p \leq q \cdot r$
by (*metis (no-types) local.mult-isor test-leq-mult-def-var*)

lemma *n-mult-glb'*: $n z \leq n x \wedge n z \leq n y \longleftrightarrow n z \leq n x \cdot n y$
using *local.order-trans n-mult-glbI' n-mult-lb1' n-mult-lb2'* **by** *blast*

lemma *test-mult-glb*: $test p \implies test q \implies test r \implies p \leq q \wedge p \leq r \longleftrightarrow p \leq q \cdot r$
using *local.n-mult-glb'* **by** *force*

lemma *n-galois1'*: $n x \leq n y + n z \longleftrightarrow n x \cdot t y \leq n z$
proof –
have $n x \leq n y + n z \longleftrightarrow n x \sqsubseteq n y \oplus n z$
by (*metis local.test-de-morgan ord-transl n-add-op-def*)
also have $\dots \longleftrightarrow n x \cdot t y \sqsubseteq n z$
using *local.n-galois1* **by** *auto*
also have $\dots \longleftrightarrow n x \cdot t y \leq n z$
by (*metis local.test-mult ord-transl*)
finally show *?thesis*
by *simp*
qed

lemma *test-galois1*: $test\ p \implies test\ q \implies test\ r \implies p \leq q + r \iff p \cdot !q \leq r$
using *n-galois1'* **by** *auto*

lemma *n-galois2'*: $n\ x \leq t\ y + n\ z \iff n\ x \cdot n\ y \leq n\ z$
using *local.n-galois1'* **by** *auto*

lemma *test-galois2*: $test\ p \implies test\ q \implies test\ r \implies p \leq !q + r \iff p \cdot q \leq r$
using *test-galois1* **by** *auto*

lemma *n-huntington2*: $n\ (n\ x + t\ y) + n\ (n\ x + n\ y) = n\ n\ x$
by *simp*

lemma *test-huntington2*: $test\ p \implies test\ q \implies !(p + q) + !(p + !q) = !p$
proof –

assume *a1*: *test p*

assume *a2*: *test q*

have $p = !(!p)$

using *a1 test-double-comp-var* **by** *blast*

thus *?thesis*

using *a2 by (metis (full-types) n-huntington2 test-double-comp-var)*

qed

lemma *n-kat-1-opp*: $n\ x \cdot y \cdot n\ z = y \cdot n\ z \iff t\ x \cdot y \cdot n\ z = 0$
proof

assume $n\ x \cdot y \cdot n\ z = y \cdot n\ z$

hence $t\ x \cdot y \cdot n\ z = t\ x \cdot n\ x \cdot y \cdot n\ z$

by (*simp add: mult-assoc*)

thus $t\ x \cdot y \cdot n\ z = 0$

by (*simp add: n-mult-comm*)

next

assume $n\ n\ x \cdot y \cdot n\ z = 0$

hence $n\ x \cdot y \cdot n\ z = 1 \cdot y \cdot n\ z$

by (*metis local.add-zero local.distrib-right' n-add-comp t-n-closed*)

thus $n\ x \cdot y \cdot n\ z = y \cdot n\ z$

by *auto*

qed

lemma *test-eq4*: $test\ p \implies test\ q \implies !p \cdot x \cdot !q = x \cdot !q \iff p \cdot x \cdot !q = 0$
using *n-kat-1-opp* **by** *auto*

lemma *n-kat-1-var*: $t\ x \cdot y \cdot t\ z = y \cdot t\ z \iff n\ x \cdot y \cdot t\ z = 0$
by (*simp add: n-kat-1-opp*)

lemma *test-kat-1*: $test\ p \implies test\ q \implies p \cdot x \cdot q = x \cdot q \iff !p \cdot x \cdot q = 0$
using *n-kat-1-var* **by** *auto*

lemma *n-kat-21-opp*: $y \cdot n\ z \leq n\ x \cdot y \implies n\ x \cdot y \cdot n\ z = y \cdot n\ z$

proof –

assume $y \cdot n z \leq n x \cdot y$
hence $y \cdot n z + n x \cdot y = n x \cdot y$
by (*meson local.join.sup-absorb2*)
hence $n x \cdot y \cdot n z = y \cdot n z + (n x \cdot (y \cdot n z) + 0)$
by (*metis local.add-zero local.distrib-right' local.n-mult-idem mult-assoc*)
thus *?thesis*
by (*metis local.add-zero local.distrib-right' local.join.sup-absorb2 local.join.sup-left-commute local.mult-one local.n-subid'*)
qed

lemma *test-kat-21-opp*: $test\ p \implies test\ q \implies x \cdot q \leq p \cdot x \longrightarrow p \cdot x \cdot q = x \cdot q$
using *n-kat-21-opp* **by** *auto*

lemma $n x \cdot y \cdot n z = y \cdot n z \implies y \cdot n z \leq n x \cdot y$
oops

lemma *n-distrib-alt'*: $n x \cdot n z = n y \cdot n z \implies n x + n z = n y + n z \implies n x = n y$
using *local.n-distrib-alt* **by** *auto*

lemma *test-distrib-alt*: $test\ p \implies test\ q \implies test\ r \implies p \cdot r = q \cdot r \wedge p + r = q + r \longrightarrow p = q$
using *n-distrib-alt* **by** *auto*

lemma *n-eq1*: $n x \cdot y \leq z \wedge t x \cdot y \leq z \longleftrightarrow y \leq z$
proof –
have $n x \cdot y \leq z \wedge t x \cdot y \leq z \longleftrightarrow n x \cdot y + t x \cdot y \leq z$
by *simp*
also have $\dots \longleftrightarrow (n x + t x) \cdot y \leq z$
using *local.distrib-right'* **by** *presburger*
finally show *?thesis*
by *auto*
qed

lemma *test-eq1*: $test\ p \implies y \leq x \longleftrightarrow p \cdot y \leq x \wedge !p \cdot y \leq x$
using *n-eq1* **by** *auto*

lemma *n-dist-var1'*: $(n x + n y) \cdot (t x + n z) = t x \cdot n y + n x \cdot n z$
by (*metis add-transl local.n-dist-var1 local.test-mult*)

lemma *test-dist-var1*: $test\ p \implies test\ q \implies test\ r \implies (p + q) \cdot (!p + r) = !p \cdot q + p \cdot r$
using *n-dist-var1'* **by** *fastforce*

lemma *n-dist-var2'*: $n (n x \cdot n y + t x \cdot n z) = n x \cdot t y + t x \cdot t z$
proof –
have $f1: !x \cdot !y = !(!(x \cdot !y))$
using *n-de-morgan-var3 t-de-morgan-var1* **by** *presburger*
have $!(!(x)) = !x$

```

    by simp
  thus ?thesis
    using f1 by (metis (no-types) n-de-morgan-var3 n-dist-var1' t-de-morgan-var1)
qed

```

```

lemma test-dist-var2: test p  $\implies$  test q  $\implies$  test r  $\implies$  !(p · q + !p · r) = (p · !q + !p · !r)
  using n-dist-var2' by auto

```

```

lemma test-restrictr: test p  $\implies$  x · p  $\leq$  x
  oops

```

```

lemma test-eq2: test p  $\implies$  z  $\leq$  p · x + !p · y  $\iff$  p · z  $\leq$  p · x  $\wedge$  !p · z  $\leq$  !p · y
  oops

```

```

lemma test-eq3:  $\llbracket$ test p; test q $\rrbracket \implies$  p · x = p · x · q  $\iff$  p · x  $\leq$  x · q
  oops

```

```

lemma test1:  $\llbracket$ test p; test q; p · x · !q = 0 $\rrbracket \implies$  p · x = p · x · q
  oops

```

```

lemma  $\llbracket$ test p; test q; x · !q = !p · x · !q $\rrbracket \implies$  p · x = p · x · q
  oops

```

```

lemma comm-add:  $\llbracket$ test p; p · x = x · p; p · y = y · p $\rrbracket \implies$  p · (x + y) = (x + y) · p
  oops

```

```

lemma comm-add-var:  $\llbracket$ test p; test q; test r; p · x = x · p; p · y = y · p $\rrbracket \implies$  p · (q · x + r · y) = (q · x + r · y) · p
  oops

```

```

lemma test p  $\implies$  p · x = x · p  $\implies$  p · x = p · x · p  $\wedge$  !p · x = !p · x · !p
  oops

```

```

lemma test-distrib:  $\llbracket$ test p; test q $\rrbracket \implies$  (p + q) · (q · y + !q · x) = q · y + !q · p · x
  oops

```

end

1.3 Test Near Semirings with Distributive Tests

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 test-near-semiring-zero-distrib = test-near-semiring-zero +
  assumes n-left-distrib: n x · (y + z) = n x · y + n x · z

```

begin

lemma *n-left-distrib-var*: $test\ p \implies p \cdot (x + y) = p \cdot x + p \cdot y$
using *n-left-distrib* **by** *auto*

lemma *n-mult-left-iso*: $x \leq y \implies n\ z \cdot x \leq n\ z \cdot y$
by (*metis local.join.sup.absorb-iff1 local.n-left-distrib*)

lemma *test-mult-isol*: $test\ p \implies x \leq y \implies p \cdot x \leq p \cdot y$
using *n-mult-left-iso* **by** *auto*

lemma *test p* $\implies x \cdot p \leq x$
oops

lemma $\llbracket test\ p; test\ q \rrbracket \implies p \cdot x = p \cdot x \cdot q \longleftrightarrow p \cdot x \leq x \cdot q$
oops

lemma $\llbracket test\ p; test\ q; p \cdot x \cdot !q = 0 \rrbracket \implies p \cdot x = p \cdot x \cdot q$
oops

lemma $\llbracket test\ p; test\ q; x \cdot !q = !p \cdot x \cdot !q \rrbracket \implies p \cdot x = p \cdot x \cdot q$
oops

Next, we study tests with commutativity conditions.

lemma *comm-add*: $test\ p \implies p \cdot x = x \cdot p \implies p \cdot y = y \cdot p \implies p \cdot (x + y) = (x + y) \cdot p$
by (*simp add: n-left-distrib-var*)

lemma *comm-add-var*: $test\ p \implies test\ q \implies test\ r \implies p \cdot x = x \cdot p \implies p \cdot y = y \cdot p \implies p \cdot (q \cdot x + r \cdot y) = (q \cdot x + r \cdot y) \cdot p$

proof –

assume *a1*: *test p*

assume *a2*: *test q*

assume *a3*: $p \cdot x = x \cdot p$

assume *a4*: *test r*

assume *a5*: $p \cdot y = y \cdot p$

have *f6*: $p \cdot (q \cdot x) = q \cdot p \cdot x$

using *a2 a1 local.test-mult-comm-var mult-assoc* **by** *presburger*

have $p \cdot (r \cdot y) = r \cdot p \cdot y$

using *a4 a1* **by** (*simp add: local.test-mult-comm-var mult-assoc*)

thus *?thesis*

using *f6 a5 a3 a1* **by** (*simp add: mult-assoc n-left-distrib-var*)

qed

lemma *test-distrib*: $test\ p \implies test\ q \implies (p + q) \cdot (q \cdot y + !q \cdot x) = q \cdot y + !q \cdot p \cdot x$

proof –

assume *a*: *test p* **and** *b*: *test q*

hence $(p + q) \cdot (q \cdot y + !q \cdot x) = p \cdot q \cdot y + p \cdot !q \cdot x + q \cdot q \cdot y + q \cdot !q \cdot x$

using *local.add.assoc local.distrib-right' mult-assoc n-left-distrib-var* **by** *presburger*

also have $\dots = p \cdot q \cdot y + !q \cdot p \cdot x + q \cdot y$

```

    by (simp add: a b local.test-mult-comm-var)
  also have ... = (p + 1) · q · y + !q · p · x
    using add-commute local.add.assoc by force
  also have ... = q · y + !q · p · x
    by (simp add: a local.join.sup.absorb2 local.test-subid)
  finally show ?thesis
    by simp
qed

end

```

1.4 Test Predioids

The following class is relevant for probabilistic Kleene algebras.

```
class test-pre-dioid-zero1 = test-near-semiring-zero1-distrib + pre-dioid
```

```
begin
```

```
lemma n-restrictr: x · n y ≤ x
  using local.mult-isol local.n-subid' by fastforce
```

```
lemma test-restrictr: test p ⇒ x · p ≤ x
  using n-restrictr by fastforce
```

```
lemma n-kat-2: n x · y = n x · y · n z ⇔ n x · y ≤ y · n z
```

```
proof
```

```
  assume n x · y = n x · y · n z
```

```
  thus n x · y ≤ y · n z
```

```
    by (metis mult.assoc n-restrictl)
```

```
next
```

```
  assume n x · y ≤ y · n z
```

```
  hence n x · y ≤ n x · y · n z
```

```
    by (metis local.mult-isol local.n-mult-idem mult-assoc)
```

```
  thus n x · y = n x · y · n z
```

```
    by (simp add: local.order.antisym n-restrictr)
```

```
qed
```

```
lemma test-kat-2: test p ⇒ test q ⇒ p · x = p · x · q ⇔ p · x ≤ x · q
  using n-kat-2 by auto
```

```
lemma n-kat-2-opp: y · n z = n x · y · n z ⇔ y · n z ≤ n x · y
  by (metis local.n-kat-21-opp n-restrictr)
```

```
lemma test-kat-2-opp: test p ⇒ test q ⇒ x · q = p · x · q ⇔ x · q ≤ p · x
  by (metis local.test-kat-21-opp test-restrictr)
```

```
lemma [[test p; test q; p · x · !q = 0]] ⇒ p · x = p · x · q
  oops
```

lemma $\llbracket \text{test } p; \text{test } q; x \cdot !q = !p \cdot x \cdot !q \rrbracket \implies p \cdot x = p \cdot x \cdot q$
oops

lemma $\llbracket \text{test } p; \text{test } q \rrbracket \implies x \cdot (p + q) \leq x \cdot p + x \cdot q$
oops

end

The following class is relevant for Demonic Refinement Algebras.

class *test-semiring-zero1* = *test-near-semiring-zero1* + *semiring-one-zero1*

begin

subclass *dioid-one-zero1* ..

subclass *test-pre-dioid-zero1*

proof

show $\bigwedge x y z. !x \cdot (y + z) = !x \cdot y + !x \cdot z$

by (*simp add: local.distrib-left*)

qed

lemma *n-kat-31*: $n x \cdot y \cdot t z = 0 \implies n x \cdot y \cdot n z = n x \cdot y$

proof –

assume *a*: $n x \cdot y \cdot t z = 0$

have $n x \cdot y = n x \cdot y \cdot (n z + t z)$

by *simp*

also have ... = $n x \cdot y \cdot n z + n x \cdot y \cdot t z$

using *local.distrib-left* **by** *blast*

finally show $n x \cdot y \cdot n z = n x \cdot y$

using *a* **by** *auto*

qed

lemma *test-kat-31*: $\text{test } p \implies \text{test } q \implies p \cdot x \cdot !q = 0 \implies p \cdot x = p \cdot x \cdot q$

by (*metis local.test-double-comp-var n-kat-31*)

lemma *n-kat-var*: $t x \cdot y \cdot t z = y \cdot t z \implies n x \cdot y \cdot n z = n x \cdot y$

using *local.n-kat-1-var n-kat-31* **by** *blast*

lemma *test1-var*: $\text{test } p \implies \text{test } q \implies x \cdot !q = !p \cdot x \cdot !q \implies p \cdot x = p \cdot x \cdot q$

by (*metis local.test-eq4 test-kat-31*)

lemma $\llbracket \text{test } p; \text{test } q; p \cdot x \cdot !q = 0 \rrbracket \implies !p \cdot x \cdot q = 0$

oops

lemma $\llbracket \text{test } p; \text{test } q; p \cdot x = p \cdot x \cdot q \rrbracket \implies x \cdot !q = !p \cdot x \cdot !q$

oops

lemma $\llbracket \text{test } p; \text{test } q; p \cdot x = p \cdot x \cdot q \rrbracket \implies p \cdot x \cdot !q = 0$

oops

```

lemma  $\llbracket \text{test } p; \text{test } q; p \cdot x = p \cdot x \cdot q \rrbracket \implies !p \cdot x \cdot q = 0$ 
  oops

lemma  $\llbracket \text{test } p; \text{test } q; x \cdot !q = !p \cdot x \cdot !q \rrbracket \implies !p \cdot x \cdot q = 0$ 
  oops

lemma  $\llbracket \text{test } p; \text{test } q; !p \cdot x \cdot q = 0 \rrbracket \implies p \cdot x = p \cdot x \cdot q$ 
  oops

lemma  $\llbracket \text{test } p; \text{test } q; !p \cdot x \cdot q = 0 \rrbracket \implies x \cdot !q = !p \cdot x \cdot !q$ 
  oops

lemma  $\llbracket \text{test } p; \text{test } q; !p \cdot x \cdot q = 0 \rrbracket \implies p \cdot x \cdot !q = 0$ 
  oops

lemma  $\text{test } p \implies p \cdot x = p \cdot x \cdot p \wedge !p \cdot x = !p \cdot x \cdot !p \implies p \cdot x = x \cdot p$ 
  oops

end

```

1.5 Test Semirings

The following class is relevant for Kleene Algebra with Tests.

```

class test-semiring = test-semiring-zero + semiring-one-zero

```

```

begin

```

```

lemma n-kat-1:  $n x \cdot y \cdot t z = 0 \iff n x \cdot y \cdot n z = n x \cdot y$ 
  by (metis local.annir local.n-kat-31 local.test-mult-comp mult-assoc)

```

```

lemma test-kat-1':  $\text{test } p \implies \text{test } q \implies p \cdot x \cdot !q = 0 \iff p \cdot x = p \cdot x \cdot q$ 
  by (metis local.test-double-comp-var n-kat-1)

```

```

lemma n-kat-3:  $n x \cdot y \cdot t z = 0 \iff n x \cdot y \leq y \cdot n z$ 
  using local.n-kat-2 n-kat-1 by force

```

```

lemma test-kat-3:  $\text{test } p \implies \text{test } q \implies p \cdot x \cdot !q = 0 \iff p \cdot x \leq x \cdot q$ 
  using n-kat-3 by auto

```

```

lemma n-kat-prop:  $n x \cdot y \cdot n z = n x \cdot y \iff t x \cdot y \cdot t z = y \cdot t z$ 
  by (metis local.annir local.n-kat-1-opp local.n-kat-var local.t-n-closed local.test-mult-comp mult-assoc)

```

```

lemma test-kat-prop:  $\text{test } p \implies \text{test } q \implies p \cdot x = p \cdot x \cdot q \iff x \cdot !q = !p \cdot x \cdot !q$ 
  by (metis local.annir local.test1-var local.test-comp-mult2 local.test-eq4 mult-assoc)

```

```

lemma n-kat-3-opp:  $t x \cdot y \cdot n z = 0 \iff y \cdot n z \leq n x \cdot y$ 

```

by (*metis local.n-kat-1-var local.n-kat-2-opp local.t-n-closed*)

lemma *kat-1-var*: $n x \cdot y \cdot n z = y \cdot n z \longleftrightarrow y \cdot n z \leq n x \cdot y$
using *local.n-kat-2-opp* by *force*

lemma $\llbracket \text{test } p; \text{test } q \rrbracket \Longrightarrow (p \cdot x \cdot !q = 0) \Longrightarrow (!p \cdot x \cdot q = 0)$
oops

lemma $n x \cdot y + t x \cdot z = n x \cdot y \vee n x \cdot y + t x \cdot z = t x \cdot z$

oops

end

end

2 Pre-Conway Algebra with Tests

theory *Conway-Tests*
imports *../Kleene-Algebra/Conway Test-Dioid*

begin

class *near-conway-zero1-tests* = *near-conway-zero1* + *test-near-semiring-zero1-distrib*

begin

lemma *n-preserve1-var*: $n x \cdot y \leq n x \cdot y \cdot n x \Longrightarrow n x \cdot (n x \cdot y + t x \cdot z)^\dagger \leq (n x \cdot y)^\dagger \cdot n x$
proof –
assume *a*: $n x \cdot y \leq n x \cdot y \cdot n x$
have $n x \cdot (n x \cdot y + t x \cdot z) = n x \cdot y$
by (*metis (no-types) local.add-zero1 local.annil local.n-left-distrib local.n-mult-idem local.test-mult-comp mult-assoc*)
hence $n x \cdot (n x \cdot y + t x \cdot z) \leq n x \cdot y \cdot n x$
by (*simp add: a*)
thus $n x \cdot (n x \cdot y + t x \cdot z)^\dagger \leq (n x \cdot y)^\dagger \cdot n x$
by (*simp add: local.dagger-simr*)
qed

lemma *test-preserve1-var*: $\text{test } p \Longrightarrow p \cdot x \leq p \cdot x \cdot p \Longrightarrow p \cdot (p \cdot x + !p \cdot y)^\dagger \leq (p \cdot x)^\dagger \cdot p$
by (*metis local.test-double-comp-var n-preserve1-var*)

end

class *test-pre-conway* = *pre-conway* + *test-pre-dioid-zero1*

begin

```

subclass near-conway-zerol-tests
  by (unfold-locales)

lemma test-preserve: test p  $\implies p \cdot x \leq p \cdot x \cdot p \implies p \cdot x^\dagger = (p \cdot x)^\dagger \cdot p$ 
  using local.preservation1-eq local.test-restrictr by auto

lemma test-preserve1: test p  $\implies p \cdot x \leq p \cdot x \cdot p \implies p \cdot (p \cdot x + !p \cdot y)^\dagger = (p \cdot x)^\dagger \cdot p$ 
proof (rule antisym)
  assume a: test p
  and b:  $p \cdot x \leq p \cdot x \cdot p$ 
  hence  $p \cdot (p \cdot x + !p \cdot y) \leq (p \cdot x) \cdot p$ 
  by (metis local.add-0-right local.annil local.n-left-distrib-var local.test-comp-mult2
local.test-mult-idem-var mult-assoc)
  thus  $p \cdot (p \cdot x + !p \cdot y)^\dagger \leq (p \cdot x)^\dagger \cdot p$ 
  using local.dagger-simr by blast
next
  assume a: test p
  and b:  $p \cdot x \leq p \cdot x \cdot p$ 
  hence  $(p \cdot x)^\dagger \cdot p = p \cdot (p \cdot x \cdot p)^\dagger$ 
  by (metis dagger-slide local.test-mult-idem-var mult-assoc)
  also have  $\dots = p \cdot (p \cdot x)^\dagger$ 
  by (metis a b local.order.antisym local.test-restrictr)
  finally show  $(p \cdot x)^\dagger \cdot p \leq p \cdot (p \cdot x + !p \cdot y)^\dagger$ 
  by (simp add: a local.dagger-iso local.test-mult-isol)
qed

lemma test-preserve2: test p  $\implies p \cdot x \cdot p = p \cdot x \implies p \cdot (p \cdot x + !p \cdot y)^\dagger \leq x^\dagger$ 
  by (metis (no-types) local.eq-refl local.test-restrictl test-preserve test-preserve1)

end

end

```

3 Kleene Algebra with Tests

```

theory KAT
  imports ../Kleene-Algebra/Kleene-Algebra Conway-Tests
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 + test-semiring-zerol
begin

```

```

lemma star-n-export1:  $(n \ x \cdot y)^* \cdot n \ x \leq n \ x \cdot y^*$ 
  by (simp add: local.n-restrictr local.star-sim1)

```

lemma *star-test-export1*: $test\ p \implies (p \cdot x)^* \cdot p \leq p \cdot x^*$
using *star-n-export1* **by** *auto*

lemma *star-n-export2*: $(n\ x \cdot y)^* \cdot n\ x \leq y^* \cdot n\ x$
by (*simp add: local.mult-isor local.n-restrictl local.star-iso*)

lemma *star-test-export2*: $test\ p \implies (p \cdot x)^* \cdot p \leq x^* \cdot p$
using *star-n-export2* **by** *auto*

lemma *star-n-export-left*: $x \cdot n\ y \leq n\ y \cdot x \implies x^* \cdot n\ y = n\ y \cdot (x \cdot n\ y)^*$
proof (*rule antisym*)

assume *a1*: $x \cdot n\ y \leq n\ y \cdot x$

hence $x \cdot n\ y = n\ y \cdot x \cdot n\ y$

by (*simp add: local.n-kat-2-opp*)

thus $x^* \cdot n\ y \leq n\ y \cdot (x \cdot n\ y)^*$

by (*simp add: local.star-sim1 mult-assoc*)

next

assume *a1*: $x \cdot n\ y \leq n\ y \cdot x$

thus $n\ y \cdot (x \cdot n\ y)^* \leq x^* \cdot n\ y$

using *local.star-slide star-n-export2* **by** *force*

qed

lemma *star-test-export-left*: $test\ p \implies x \cdot p \leq p \cdot x \longrightarrow x^* \cdot p = p \cdot (x \cdot p)^*$
using *star-n-export-left* **by** *auto*

lemma *star-n-export-right*: $x \cdot n\ y \leq n\ y \cdot x \implies x^* \cdot n\ y = (n\ y \cdot x)^* \cdot n\ y$
by (*simp add: local.star-slide star-n-export-left*)

lemma *star-test-export-right*: $test\ p \implies x \cdot p \leq p \cdot x \longrightarrow x^* \cdot p = (p \cdot x)^* \cdot p$
using *star-n-export-right* **by** *auto*

lemma *star-n-folk*: $n\ z \cdot x = x \cdot n\ z \implies n\ z \cdot y = y \cdot n\ z \implies (n\ z \cdot x + t\ z \cdot y)^* \cdot n\ z = n\ z \cdot (n\ z \cdot x)^*$

proof –

assume *a*: $n\ z \cdot x = x \cdot n\ z$ **and** *b*: $n\ z \cdot y = y \cdot n\ z$

hence $n\ z \cdot (n\ z \cdot x + t\ z \cdot y) = (n\ z \cdot x + t\ z \cdot y) \cdot n\ z$

using *local.comm-add-var local.t-n-closed local.test-def* **by** *blast*

hence $(n\ z \cdot x + t\ z \cdot y)^* \cdot n\ z = n\ z \cdot ((n\ z \cdot x + t\ z \cdot y) \cdot n\ z)^*$

using *local.order-refl star-n-export-left* **by** *presburger*

also have $\dots = n\ z \cdot (n\ z \cdot x \cdot n\ z + t\ z \cdot y \cdot n\ z)^*$

by *simp*

also have $\dots = n\ z \cdot (n\ z \cdot n\ z \cdot x + t\ z \cdot n\ z \cdot y)^*$

by (*simp add: a b mult-assoc*)

also have $\dots = n\ z \cdot (n\ z \cdot x + 0 \cdot y)^*$

by (*simp add: local.n-mult-comm*)

finally show $(n\ z \cdot x + t\ z \cdot y)^* \cdot n\ z = n\ z \cdot (n\ z \cdot x)^*$

by *simp*

qed

```

lemma star-test-folk: test  $p \implies p \cdot x = x \cdot p \longrightarrow p \cdot y = y \cdot p \longrightarrow (p \cdot x + !p \cdot y)^* \cdot p = p \cdot (p \cdot x)^*$ 
  using star-n-folk by auto

end

class kat-zero1 = kleene-algebra-zero1 + test-semiring-zero1
begin

sublocale conway: near-conway-zero1-tests star ..

lemma n-star-sim-right:  $n y \cdot x = x \cdot n y \implies n y \cdot x^* = (n y \cdot x)^* \cdot n y$ 
  by (metis local.n-mult-idem local.star-sim3 mult-assoc)

lemma star-sim-right: test  $p \implies p \cdot x = x \cdot p \longrightarrow p \cdot x^* = (p \cdot x)^* \cdot p$ 
  using n-star-sim-right by auto

lemma n-star-sim-left:  $n y \cdot x = x \cdot n y \implies n y \cdot x^* = n y \cdot (x \cdot n y)^*$ 
  by (metis local.star-slide n-star-sim-right)

lemma star-sim-left: test  $p \implies p \cdot x = x \cdot p \longrightarrow p \cdot x^* = p \cdot (x \cdot p)^*$ 
  using n-star-sim-left by auto

lemma n-comm-star:  $n z \cdot x = x \cdot n z \implies n z \cdot y = y \cdot n z \implies n z \cdot x \cdot (n z \cdot y)^* = n z \cdot x \cdot y^*$ 
  using mult-assoc n-star-sim-left by presburger

lemma comm-star: test  $p \implies p \cdot x = x \cdot p \longrightarrow p \cdot y = y \cdot p \longrightarrow p \cdot x \cdot (p \cdot y)^* = p \cdot x \cdot y^*$ 
  using n-comm-star by auto

lemma n-star-sim-right-var:  $n y \cdot x = x \cdot n y \implies x^* \cdot n y = n y \cdot (x \cdot n y)^*$ 
  using local.star-sim3 n-star-sim-left by force

lemma star-sim-right-var: test  $p \implies p \cdot x = x \cdot p \longrightarrow x^* \cdot p = p \cdot (x \cdot p)^*$ 
  using n-star-sim-right-var by auto

```

end

Finally, we define Kleene algebra with tests.

```

class kat = kleene-algebra + test-semiring

begin

sublocale conway: test-pre-conway star ..

end

end

```


4 Demonic Refinement Algebra with Tests

```

theory DRAT
  imports KAT ../Kleene-Algebra/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 drat = dra + test-semiring-zero
begin

```

```

subclass kat-zero ..

```

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* :: $'a \Rightarrow 'a$ ($-^\circ$ [101] 100) **where**
 $test\ p \Longrightarrow p^\circ = !p \cdot \top + 1$

lemma *asg*: $\llbracket test\ p; test\ q \rrbracket \Longrightarrow q \leq 1 \wedge 1 \leq p^\circ$
by (*simp add: assertion-def local.test-subid*)

lemma *assertion-isol*: $test\ p \Longrightarrow y \leq p^\circ \cdot x \longleftrightarrow p \cdot y \leq x$
proof

```

  assume assms: test p y ≤ p°.x
  hence p·y ≤ p·!p·⊤·x + p·x
    by (metis add-commute assertion-def local.distrib-left local.iteration-prod-unfold
local.iteration-unfoldl-distr local.mult-isol local.top-mult-annil mult-assoc)
  also have ... ≤ x
    by (simp add: assms(1) local.test-restrictl)
  finally show p·y ≤ x
    by metis

```

next

```

  assume assms: test p p·y ≤ x
  hence p°.p·y = !p·⊤·p·y + p·y
    by (metis assertion-def distrib-right' mult-1-left mult.assoc)
  also have ... = !p·⊤ + p·y
    by (metis mult.assoc top-mult-annil)
  moreover have p°.p·y ≤ p°.x
    by (metis assms(2) mult.assoc mult-isol)
  moreover have !p·y + p·y ≤ !p·⊤ + p·y
    using local.add-iso local.top-elim by blast
  ultimately show y ≤ p°.x
    by (metis add.commute assms(1) distrib-right' mult-1-left order-trans test-add-comp)
qed

```

lemma *assertion-isor*: $test\ p \Longrightarrow y \leq x \cdot p \longleftrightarrow y \cdot p^\circ \leq x$
proof

```

  assume assms: test p y ≤ x·p

```

hence $y \cdot p^\circ \leq x \cdot p \cdot !p \cdot \top + x \cdot p$
by (*metis mult-isor assertion-def assms(1) distrib-left mult-1-right mult.assoc*)
also have $\dots \leq x$
by (*metis assms(1) local.iteration-idep local.join.sup.absorb-iff1 local.join.sup-commute local.join.sup-ge2 local.mult-1-right local.mult-isol-var local.mult-isor local.mult-one1 local.test-add-comp local.test-comp-mult2 mult-assoc*)
finally show $y \cdot p^\circ \leq x$
by *metis*
next
assume *assms: test p y · p^o ≤ x*
have $y \leq y \cdot (!p \cdot \top + p)$
by (*metis join.sup-mono mult-isol order-refl order-refl top-elim add.commute assms(1) mult-1-right test-add-comp*)
also have $\dots = y \cdot p^\circ \cdot p$
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*: $\llbracket \text{test } p; \text{test } q \rrbracket \implies x \cdot q^\circ \leq p^\circ \cdot x \iff p \cdot x \leq x \cdot q$
by (*metis assertion-isol assertion-isor mult.assoc*)

lemma *total-correctness*: $\llbracket \text{test } p; \text{test } q \rrbracket \implies p \cdot x \cdot !q = 0 \iff x \cdot !q \leq !p \cdot \top$
apply *standard*
apply (*metis local.test-eq4 local.top-elim mult-assoc*)
by (*metis annil antisym test-comp-mult2 join.bot-least mult-assoc mult-isol*)

lemma *test-iteration-sim*: $\llbracket \text{test } p; p \cdot x \leq x \cdot p \rrbracket \implies p \cdot x^\infty \leq x^\infty \cdot p$
by (*metis iteration-sim*)

lemma *test-iteration-annil*: $\text{test } p \implies !p \cdot (p \cdot x)^\infty = !p$
by (*metis annil test-comp-mult1 iteration-idep mult.assoc*)

Next we give an example of a program transformation from von Wright [5].

lemma *loop-refinement*: $\llbracket \text{test } p; \text{test } q \rrbracket \implies (p \cdot x)^\infty \cdot !p \leq (p \cdot q \cdot x)^\infty \cdot !(p \cdot q) \cdot (p \cdot x)^\infty \cdot !p$
proof –

assume *assms: test p test q*
hence $(p \cdot x)^\infty \cdot !p = ((p \cdot q) + !(p \cdot q)) \cdot (p \cdot x)^\infty \cdot !p$
by (*simp add: local.test-mult-closed*)
also have $\dots = (p \cdot q) \cdot (p \cdot x)^\infty \cdot !p + !(p \cdot q) \cdot (p \cdot x)^\infty \cdot !p$
by (*metis distrib-right'*)
also have $\dots = (p \cdot q) \cdot !p + (p \cdot q) \cdot (p \cdot x) \cdot (p \cdot x)^\infty \cdot !p + !(p \cdot q) \cdot (p \cdot x)^\infty \cdot !p$
by (*metis iteration-unfoldr-distr mult.assoc iteration-unfold-eq distrib-left mult.assoc*)
also have $\dots = (p \cdot q) \cdot (p \cdot x) \cdot (p \cdot x)^\infty \cdot !p + !(p \cdot q) \cdot (p \cdot x)^\infty \cdot !p$
by (*metis assms less-eq-def test-preserve2 join.bot-least*)
finally have $(p \cdot x)^\infty \cdot !p \leq p \cdot q \cdot x \cdot (p \cdot x)^\infty \cdot !p + !(p \cdot q) \cdot (p \cdot x)^\infty \cdot !p$
by (*metis assms(1) assms(2) local.eq-iff local.test-mult-comm-var local.test-preserve mult-assoc*)
thus *?thesis*

by (*metis coinduction add.commute mult.assoc*)
qed

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

lemma *atom-step1*: $r \cdot b \leq b \cdot r \implies (a + b + r)^\infty = b^\infty \cdot r^\infty \cdot (a \cdot b^\infty \cdot r^\infty)^\infty$
apply (*subgoal-tac* $(a + b + r)^\infty = (b + r)^\infty \cdot (a \cdot (b + r)^\infty)^\infty$)
apply (*metis iteration-sep mult.assoc*)
by (*metis add-assoc' add.commute iteration-denest*)

lemma *atom-step2*:

assumes $s = s \cdot q \cdot q \cdot b = 0 \cdot r \cdot q \leq q \cdot r \cdot q \cdot l \leq l \cdot q \cdot r^\infty = r^* \cdot \text{test } q$
shows $s \cdot l^\infty \cdot b^\infty \cdot r^\infty \cdot q \cdot (a \cdot b^\infty \cdot r^\infty \cdot q)^\infty \leq s \cdot l^\infty \cdot r^\infty \cdot (a \cdot b^\infty \cdot q \cdot r^\infty)^\infty$
proof –
have $s \cdot l^\infty \cdot b^\infty \cdot r^\infty \cdot q \cdot (a \cdot b^\infty \cdot r^\infty \cdot q)^\infty \leq s \cdot l^\infty \cdot b^\infty \cdot r^\infty \cdot q \cdot (a \cdot b^\infty \cdot q \cdot r^\infty)^\infty$
by (*metis assms(3) assms(5) star-sim1 mult.assoc mult-isol iteration-iso*)
also have $\dots \leq s \cdot q \cdot l^\infty \cdot b^\infty \cdot r^\infty \cdot (a \cdot b^\infty \cdot q \cdot r^\infty)^\infty$
using *assms(1) assms(6) local.mult-isol local.test-restrictr* **by** *auto*
also have $\dots \leq s \cdot l^\infty \cdot q \cdot b^\infty \cdot r^\infty \cdot (a \cdot b^\infty \cdot q \cdot r^\infty)^\infty$
by (*metis assms(4) iteration-sim mult.assoc mult-double-iso mult-double-iso*)
also have $\dots \leq s \cdot l^\infty \cdot r^\infty \cdot q \cdot r^\infty \cdot (a \cdot b^\infty \cdot q \cdot r^\infty)^\infty$
by (*metis assms(2) join.bot-least iteration-sim mult.assoc mult-double-iso*)
also have $\dots \leq s \cdot l^\infty \cdot r^\infty \cdot (a \cdot b^\infty \cdot q \cdot r^\infty)^\infty$
by (*metis assms(6) mult.assoc mult-isol test-restrictl iteration-idem mult.assoc*)
finally show $s \cdot l^\infty \cdot b^\infty \cdot r^\infty \cdot q \cdot (a \cdot b^\infty \cdot r^\infty \cdot q)^\infty \leq s \cdot l^\infty \cdot r^\infty \cdot (a \cdot b^\infty \cdot q \cdot r^\infty)^\infty$
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 join.sup-mono assms(1,4) mult.assoc mult-isol*)
also have $\dots \leq a \cdot l \cdot b^\infty \cdot q + l \cdot r$
by (*metis assms(3) assms(5) star-sim1 add-iso mult.assoc mult-double-iso*)
also have $\dots \leq l \cdot (a \cdot b^\infty \cdot q + r)$
by (*metis add-iso assms(2) mult-isol 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 [5].

theorem *atom-ref-back*:

assumes $s = s \cdot q$ $a = q \cdot a$ $q \cdot b = 0$
 $r \cdot b \leq b \cdot r$ $r \cdot l \leq l \cdot r$ $r \cdot q \leq q \cdot r$
 $a \cdot l \leq l \cdot a$ $b \cdot l \leq l \cdot b$ $q \cdot l \leq l \cdot q$
 $r^\infty = r^*$ $b^\infty = b^*$ *test* 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 join.sup-mono 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 $\dots \leq s \cdot l^\infty \cdot b^\infty \cdot r^\infty \cdot q \cdot (a \cdot b^\infty \cdot r^\infty \cdot q)^\infty$
by (*metis assms(2,4,10,11) atom-step1 iteration-slide eq-refl mult.assoc*)
also have $\dots \leq s \cdot l^\infty \cdot r^\infty \cdot (a \cdot b^\infty \cdot q \cdot r^\infty)^\infty$
by (*metis assms(1) assms(10) assms(12) assms(3) assms(6) assms(9) atom-step2*)
also have $\dots \leq s \cdot (a \cdot b^\infty \cdot q + l + r)^\infty$
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 [2].

theorem *atom-ref-back-struth*:

assumes $s \leq s \cdot q$ $a \leq q \cdot a$ $q \cdot b = 0$
 $r \cdot b \leq b \cdot r$ $r \cdot q \leq q \cdot r$
 $(a + r + b) \cdot l \leq l \cdot (a + r + b)$ $q \cdot l \leq l \cdot q$
 $r^\infty = r^*$ $q \leq 1$
shows $s \cdot (a + b + r + l)^\infty \cdot q \leq s \cdot (a \cdot b^\infty \cdot q + r + l)^\infty$

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

also have $\dots = s.l^\infty.q.r^\infty.q.(a.b^\infty.q.r^\infty)^\infty$
by (*metis assms(8)*)
also have $\dots \leq s.l^\infty.r^\infty.q.(a.b^\infty.q.r^\infty)^\infty$
by (*metis assms(9) mult-1-right mult-double-iso mult-isor*)
also have $\dots \leq s.l^\infty.r^\infty.(a.b^\infty.q.r^\infty)^\infty$
by (*metis assms(9) mult-1-right mult-double-iso*)
also have $\dots = s.l^\infty.(a.b^\infty.q + r)^\infty$
by (*metis add.commute mult.assoc iteration-denest*)
also have $\dots \leq s.(a.b^\infty.q + r + l)^\infty$
by (*metis add.commute iteration-subdenest mult.assoc mult-isol*)
finally show *?thesis* .
qed

Finally, we prove Cohen's [1] variation of the atomicity refinement theorem.

lemma *atom-ref-cohen*:

assumes $r.p.y \leq y.r \ y.p.l \leq l.y \ r.p.l \leq l.r$
 $p.r.!p = 0 \ p.l.!p = 0 \ !p.l.p = 0$
 $y.0 = 0 \ r.0 = 0 \ \text{test } p$
shows $(y + r + l)^\infty = (p.l)^\infty.(y + !p.l + r.!p)^\infty.(r.p)^\infty$
proof –
have $(y + r).p.l \leq l.y + l.r$
by (*metis distrib-right' join.sup-mono assms(2) assms(3)*)
hence *stepA*: $(y + r).p.l \leq (p.l + !p.l).(y + r)^*$
by (*metis assms(9) distrib-left distrib-right' mult-1-left mult-isol order-trans star-ext test-add-comp*)
have *subStepB*: $(!p.l + y + p.r + !p.r)^\infty = (!p.l + y + r.p + r.!p)^\infty$
by (*metis add-assoc' annil assms(8) assms(9) distrib-left distrib-right' star-slide star-subid test-add-comp join.bot-least*)
have $r.p.(y + r.!p + !p.l) \leq y.(r.p + r.!p)$
by (*metis assms(1,4,9) distrib-left add-0-left add.commute annil mult.assoc test-comp-mult2 distrib-left mult-oner test-add-comp*)
also have $\dots \leq (y + r.!p + !p.l).(r.p + (y + r.!p + !p.l))$
by (*meson local.eq-refl local.join.sup-ge1 local.join.sup-ge2 local.join.sup-mono local.mult-isol-var local.order-trans*)
finally have $r.p.(y + r.!p + !p.l) \leq (y + r.!p + !p.l).(y + r.!p + !p.l + r.p)$
by (*metis add.commute*)
hence *stepB*: $(!p.l + y + p.r + !p.r)^\infty = (y + !p.l + r.!p)^\infty.(r.p)^\infty$
by (*metis subStepB iteration-sep3[of r.p y + r.!p + !p.l] add-assoc' add.commute*)
have $(y + r + l)^\infty = (p.l + !p.l + y + r)^\infty$
by (*metis add-comm add.left-commute assms(9) distrib-right' mult-1-onel test-add-comp*)
also have $\dots = (p.l)^\infty.(!p.l + y + r)^\infty$ **using** *stepA*
by (*metis assms(6-8) annil add.assoc add-0-left distrib-right' add.commute mult.assoc iteration-sep4[of y+r !p.l p.l]*)
also have $\dots = (p.l)^\infty.(!p.l + y + p.r + !p.r)^\infty$
by (*metis add.commute assms(9) combine-common-factor mult-1-left test-add-comp*)
finally show *?thesis* **using** *stepB*
by (*metis mult.assoc*)
qed

end

end

5 Models for Demonic Refinement Algebra with Tests

theory *DRA-Models*

imports *DRAT*

begin

We formalise the predicate transformer model of demonic refinement algebra. Predicate transformers are formalised as strict and additive functions over a field of sets, or alternatively as costrict and multiplicative functions. In the future, this should be merged with Preoteasa's more abstract formalisation [4].

no-notation

plus (**infixl** + 65) **and**

less-eq ((-/ ≤ -) [51, 51] 50)

notation *comp* (**infixl** · 55)

type-synonym 'a *bfun* = 'a *set* ⇒ 'a *set*

Definitions of signature:

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* ≡ λp. *f* p ∩ *g* p

definition *fun-union* :: 'a *bfun* ⇒ 'a *bfun* ⇒ 'a *bfun* (**infix** + 52) **where**

f + *g* ≡ λp. *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 *oder-def*: $f \sqcap g = f \implies f \leq g$
by (*metis fun-inter-def fun-order-def le-iff-inf*)

lemma *order-def-var*: $f \leq g \implies f \sqcap g = f$
by (*auto simp: fun-inter-def fun-order-def*)

lemma *adjoint-idem* [*simp*]: $\text{adjoint} (\text{adjoint } f) = f$
by (*auto simp: adjoint-def*)

lemma *adjoint-prop1* [*simp*]: $(f \circ \text{top}) \sqcap (\text{adjoint } f \circ \text{bot}) = \text{bot}$
by (*auto simp: fun-inter-def adjoint-def bot-def top-def*)

lemma *adjoint-prop2* [*simp*]: $(f \circ \text{top}) + (\text{adjoint } f \circ \text{bot}) = \text{top}$
by (*auto simp: fun-union-def adjoint-def bot-def top-def*)

lemma *adjoint-mult*: $\text{adjoint} (f \circ g) = \text{adjoint } f \circ \text{adjoint } g$
by (*auto simp: adjoint-def*)

lemma *adjoint-top* [*simp*]: $\text{adjoint } \text{top} = \text{bot}$
by (*auto simp: adjoint-def bot-def top-def*)

lemma *N-comp1*: $(N (N f)) + N f = \text{id}$
by (*auto simp: fun-union-def N-def fun-inter-def adjoint-def bot-def*)

lemma *N-comp2*: $(N (N f)) \circ N f = \text{bot}$
by (*auto simp: N-def fun-inter-def adjoint-def bot-def*)

lemma *N-comp3*: $N f \circ (N (N f)) = \text{bot}$
by (*auto simp: N-def fun-inter-def adjoint-def bot-def*)

lemma *N-de-morgan*: $N (N f) \circ N (N g) = N (N f) \sqcap N (N g)$
by (*auto simp: fun-union-def N-def fun-inter-def adjoint-def bot-def*)

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

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

typedef *'a bool-op* = $\{f::'a \text{ bfun. } (\forall g h. \text{mono } f \wedge f \circ g + f \circ h = f \circ (g + h) \wedge \text{bot} \circ f = \text{bot})\}$
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*

Conjunctive predicate transformers form a dioid with tests without right

annihilator.

instantiation *bool-op* :: (type) *dioid-one-zero*

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 mono-def*)

apply (*metis set-mp*)

apply (*metis set-mp*)

apply (*rule ext*)

by (*metis (no-types, lifting) semilattice-sup-class.sup.assoc semilattice-sup-class.sup.left-commute*)

instance

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

end

instantiation *bool-op* :: (type) *test-semiring-zero*

begin

lift-definition *n-op-bool-op* :: 'a *bool-op* \Rightarrow 'a *bool-op* **is** *N*

by (*auto simp: N-def fun-inter-def adjoint-def bot-def fun-union-def mono-def*)

instance

apply *standard*

apply (*transfer, clarsimp simp add: N-def adjoint-def bot-def id-def comp-def fun-inter-def*)

apply (*transfer, clarsimp simp add: N-def adjoint-def bot-def id-def comp-def fun-inter-def fun-union-def mono-def, blast*)

apply (*transfer, clarsimp simp add: N-def adjoint-def bot-def comp-def mono-def fun-union-def fun-inter-def*)

by (*transfer, clarsimp simp add: N-def adjoint-def bot-def comp-def mono-def fun-union-def fun-inter-def, blast*)

end

definition *fun-star* :: 'a *bfun* \Rightarrow 'a *bfun* **where**

fun-star *f* = *lfp* ($\lambda r. f \circ r + id$)

definition *fun-iteration* :: 'a bfun \Rightarrow 'a bfun **where**
fun-iteration $f = \text{gfp } (\lambda g. f \circ g + \text{id})$

Verifying the iteration laws is left for future work. This could be obtained by integrating Preoteasa's approach [4].

end

6 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 a Kleene algebra with tests.

interpretation *rel-dioid-tests*: *test-semiring* $op \cup op \ O \ Id \ \{\}$ $op \subseteq op \subset \lambda x. Id \cap (- x)$
by (*standard*, *auto*)

interpretation *rel-kat*: *kat* $op \cup op \ O \ Id \ \{\}$ $op \subseteq op \subset rtrancl \ \lambda x. Id \cap (- x)$
by (*unfold-locales*)

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

setup-lifting *type-definition-relation*

instantiation *relation* :: (*type*) *kat*

begin

lift-definition *n-op-relation* :: 'a *relation* \Rightarrow 'a *relation* **is** $\lambda x. Id \cap (- x)$ **done**

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

lift-definition *star-relation* :: 'a *relation* \Rightarrow 'a *relation* **is** *rtrancl* **done**

lift-definition *less-eq-relation* :: 'a *relation* \Rightarrow 'a *relation* \Rightarrow *bool* **is** $op \subseteq$ **done**

lift-definition *less-relation* :: 'a *relation* \Rightarrow 'a *relation* \Rightarrow *bool* **is** $op \subset$ **done**

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

lift-definition *times-relation* :: 'a *relation* \Rightarrow 'a *relation* \Rightarrow 'a *relation* **is** $op \ O$ **done**

lift-definition *plus-relation* :: 'a *relation* \Rightarrow 'a *relation* \Rightarrow 'a *relation* **is** $op \cup$ **done**

instance

apply *standard*

apply (*transfer*, *simp add*: *Un-assoc*)

apply (*transfer*, *simp add*: *Un-commute*)

apply (*transfer*, *simp add*: *rel-dioid.mult-assoc*)

apply (*transfer*, *simp*)

apply (*transfer*, *simp*)

```

apply (transfer, simp)
apply (transfer, simp)
apply (transfer, simp)
apply (transfer, simp)
apply (transfer, simp add: rel-diod.less-eq-def)
apply (transfer, simp add: rel-diod.less-def)
apply (transfer, simp)
apply (transfer, simp)
apply (transfer, simp)
apply (transfer, simp)
apply (transfer, simp add: rel-kleene-algebra.star-inductl)
apply (transfer, simp add: rel-kleene-algebra.star-inductr)
apply (transfer, simp)
apply (transfer, blast)
apply (transfer, blast)
by (transfer, blast)

```

end

end

7 Transformation Theorem for while Loops

theory *FolkTheorem*

imports *Conway-Tests KAT DRAT*

begin

We prove Kozen's transformation theorem for while loops [3] in a weak setting that unifies previous proofs in Kleene algebra with tests, demonic refinement algebras and a variant of probabilistic Kleene algebra.

context *test-pre-conway*

begin

abbreviation *pres* :: '*a* ⇒ '*a* ⇒ *bool* **where**

pres *x y* ≡ *y* · *x* = *y* · *x* · *y*

lemma *pres-comp*: *pres y z* ⇒ *pres x z* ⇒ *pres (x · y) z*

by (*metis mult-assoc*)

lemma *test-pres1*: *test p* ⇒ *test q* ⇒ *pres p q*

by (*simp add: local.test-mult-comm-var mult-assoc*)

lemma *test-pres2*: *test p* ⇒ *test q* ⇒ *pres x q* ⇒ *pres (p · x) q*

using *pres-comp test-pres1* **by** *blast*

lemma *cond-helper1*:

assumes *test p* **and** *pres x p*

shows $p \cdot (p \cdot x + !p \cdot y)^\dagger \cdot (p \cdot z + !p \cdot w) = p \cdot x^\dagger \cdot z$

proof –
have $p \cdot (p \cdot z + !p \cdot w) = p \cdot z$
by (*metis* *assms(1)* *local.add-zero1* *local.annil* *local.join.sup-commute* *local.n-left-distrib-var* *local.test-comp-mult2* *local.test-mult-idem-var* *mult-assoc*)
hence $p \cdot (p \cdot x + !p \cdot y)^\dagger \cdot (p \cdot z + !p \cdot w) = (p \cdot x)^\dagger \cdot p \cdot z$
using *assms(1)* *assms(2)* *local.test-preserve1* *mult-assoc* **by** *auto*
thus *?thesis*
using *assms(1)* *assms(2)* *local.test-preserve* *mult-assoc* **by** *auto*
qed

lemma *cond-helper2*:
assumes *test p* **and** *pres y* $(!p)$
shows $!p \cdot (p \cdot x + !p \cdot y)^\dagger \cdot (p \cdot z + !p \cdot w) = !p \cdot y^\dagger \cdot w$
proof –
have $!p \cdot (!p \cdot y + !(!p) \cdot x)^\dagger \cdot (!p \cdot w + !(!p) \cdot z) = !p \cdot y^\dagger \cdot w$
using *assms(1)* *local.test-comp-closed* *assms(2)* *cond-helper1* **by** *blast*
thus *?thesis*
using *add-commute* *assms(1)* **by** *auto*
qed

lemma *cond-distr-var*:
assumes *test p* **and** *test q* **and** *test r*
shows $(q \cdot p + r \cdot !p) \cdot (p \cdot x + !p \cdot y) = q \cdot p \cdot x + r \cdot !p \cdot y$
proof –
have $(q \cdot p + r \cdot !p) \cdot (p \cdot x + !p \cdot y) = q \cdot p \cdot p \cdot x + q \cdot p \cdot !p \cdot y + r \cdot !p \cdot p \cdot x + r \cdot !p \cdot !p \cdot y$
using *assms(1)* *assms(2)* *assms(3)* *local.distrib-right'* *local.join.sup-assoc* *local.n-left-distrib-var* *local.test-comp-closed* *mult-assoc* **by** *presburger*
also have $\dots = q \cdot p \cdot x + q \cdot 0 \cdot y + r \cdot 0 \cdot x + r \cdot !p \cdot y$
by (*simp* *add: assms(1)* *mult-assoc*)
finally show *?thesis*
by (*metis* *assms(2)* *assms(3)* *local.add-zero1* *local.annil* *local.join.sup-commute* *local.test-mult-comm-var* *local.test-zero-var*)
qed

lemma *cond-distr*:
assumes *test p* **and** *test q* **and** *test r*
shows $(p \cdot q + !p \cdot r) \cdot (p \cdot x + !p \cdot y) = p \cdot q \cdot x + !p \cdot r \cdot y$
using *assms(1)* *assms(2)* *assms(3)* *local.test-mult-comm-var* *assms(1)* *assms(2)* *assms(3)* *cond-distr-var* *local.test-mult-comm-var* **by** *auto*

theorem *conditional*:
assumes *test p* **and** *test q* **and** *test r1* **and** *test r2*
and *pres x1* *q* **and** *pres y1* *q* **and** *pres x2* $(!q)$ **and** *pres y2* $(!q)$
shows $(p \cdot q + !p \cdot !q) \cdot (q \cdot x1 + !q \cdot x2) \cdot ((q \cdot r1 + !q \cdot r2) \cdot (q \cdot y1 + !q \cdot y2))^\dagger \cdot !(q \cdot r1 + !q \cdot r2) =$
 $(p \cdot q + !p \cdot !q) \cdot (p \cdot x1 \cdot (r1 \cdot y1)^\dagger \cdot !r1 + !p \cdot x2 \cdot (r2 \cdot y2)^\dagger \cdot !r2)$
proof –
have $a: p \cdot q \cdot x1 \cdot q \cdot (q \cdot r1 \cdot y1 + !q \cdot r2 \cdot y2)^\dagger \cdot (q \cdot !r1 + !q \cdot !r2) = p \cdot q$

$\cdot x1 \cdot q \cdot (r1 \cdot y1)^\dagger \cdot !r1$
by (*metis* *assms*(2) *assms*(3) *assms*(6) *cond-helper1* *mult-assoc* *test-pres2*)
have $b: !q \cdot (q \cdot r1 \cdot y1 + !q \cdot r2 \cdot y2)^\dagger \cdot (q \cdot !r1 + !q \cdot !r2) = !q \cdot (r2 \cdot y2)^\dagger \cdot !r2$
by (*metis* *assms*(2) *assms*(4) *assms*(8) *local.test-comp-closed* *cond-helper2* *mult-assoc* *test-pres2*)
have $(p \cdot q + !p \cdot !q) \cdot (q \cdot x1 + !q \cdot x2) \cdot ((q \cdot r1 + !q \cdot r2) \cdot (q \cdot y1 + !q \cdot y2))^\dagger \cdot !(q \cdot r1 + !q \cdot r2) =$
 $p \cdot q \cdot x1 \cdot q \cdot (q \cdot r1 \cdot y1 + !q \cdot r2 \cdot y2)^\dagger \cdot (q \cdot !r1 + !q \cdot !r2) + !p \cdot !q \cdot$
 $x2 \cdot !q \cdot (q \cdot r1 \cdot y1 + !q \cdot r2 \cdot y2)^\dagger \cdot (q \cdot !r1 + !q \cdot !r2)$
using *assms*(1) *assms*(2) *assms*(3) *assms*(4) *assms*(5) *assms*(7) *cond-distr* *cond-distr-var* *local.test-dist-var2* *mult-assoc* **by** *auto*
also have $\dots = p \cdot q \cdot x1 \cdot (r1 \cdot y1)^\dagger \cdot !r1 + !p \cdot !q \cdot x2 \cdot (r2 \cdot y2)^\dagger \cdot !r2$
by (*metis* *a* *assms*(5) *assms*(7) *b* *mult-assoc*)
finally show *?thesis*
using *assms*(1) *assms*(2) *cond-distr* *mult-assoc* **by** *auto*
qed

theorem *nested-loops*:

assumes *test p* **and** *test q*
shows $p \cdot x \cdot ((p + q) \cdot (q \cdot y + !q \cdot x))^\dagger \cdot !(p + q) + !p = (p \cdot x \cdot (q \cdot y)^\dagger \cdot !q)^\dagger \cdot !p$
proof –
have $p \cdot x \cdot ((p + q) \cdot (q \cdot y + !q \cdot x))^\dagger \cdot !(p + q) + !p = p \cdot x \cdot (q \cdot y)^\dagger \cdot (!q$
 $\cdot p \cdot x \cdot (q \cdot y)^\dagger)^\dagger \cdot !q \cdot !p + !p$
using *assms*(1) *assms*(2) *local.dagger-denest2* *local.test-distrib* *mult-assoc* *test-mult-comm-var*
by *auto*
also have $\dots = p \cdot x \cdot (q \cdot y)^\dagger \cdot !q \cdot (p \cdot x \cdot (q \cdot y)^\dagger \cdot !q)^\dagger \cdot !p + !p$
by (*metis* *local.dagger-slide* *mult-assoc*)
finally show *?thesis*
using *add-commute* **by** *force*
qed

lemma *postcomputation*:

assumes *test p* **and** *pres y* (*!p*)
shows $!p \cdot y + p \cdot (p \cdot x \cdot (!p \cdot y + p))^\dagger \cdot !p = (p \cdot x)^\dagger \cdot !p \cdot y$
proof –
have $p \cdot (p \cdot x \cdot (!p \cdot y + p))^\dagger \cdot !p = p \cdot (1 + p \cdot x \cdot (!p \cdot y + p) \cdot p \cdot x)^\dagger \cdot$
 $(!p \cdot y + p) \cdot !p$
by (*metis* *dagger-prod-unfold* *mult.assoc*)
also have $\dots = (p + p \cdot p \cdot x \cdot (!p \cdot y + p) \cdot p \cdot x)^\dagger \cdot (!p \cdot y + p) \cdot !p$
using *assms*(1) *local.mult-oner* *local.n-left-distrib-var* *mult-assoc* **by** *presburger*

also have $\dots = p \cdot x \cdot (!p \cdot y \cdot p \cdot x + p \cdot x)^\dagger \cdot !p \cdot y \cdot !p$
by (*simp* *add: assms*(1) *mult-assoc*)
also have $\dots = p \cdot x \cdot (!p \cdot y \cdot 0 + p \cdot x)^\dagger \cdot !p \cdot y$
by (*metis* *assms*(1) *assms*(2) *local.annil* *local.test-comp-mult1* *mult-assoc*)
also have $\dots = p \cdot x \cdot (p \cdot x)^\dagger \cdot (!p \cdot y \cdot 0 \cdot (p \cdot x)^\dagger)^\dagger \cdot !p \cdot y$
by (*metis* *mult.assoc* *add.commute* *dagger-denest2*)

finally have $p \cdot (p \cdot x \cdot (!p \cdot y + p))^\dagger \cdot !p = p \cdot x \cdot (p \cdot x)^\dagger \cdot !p \cdot y$
by (*metis local.add-zero local.annil local.dagger-prod-unfold local.dagger-slide*
local.mult-oner mult-assoc)
thus *?thesis*
by (*metis dagger-unfoldl-distr mult.assoc*)
qed

lemma *composition-helper*:
assumes *test p and test q*
and *pres x p*
shows $p \cdot (q \cdot x)^\dagger \cdot !q \cdot p = p \cdot (q \cdot x)^\dagger \cdot !q$
proof (*rule antisym*)
show $p \cdot (q \cdot x)^\dagger \cdot !q \cdot p \leq p \cdot (q \cdot x)^\dagger \cdot !q$
by (*simp add: assms(1) local.test-restrict*)
next
have $p \cdot q \cdot x \leq q \cdot x \cdot p$
by (*metis assms(1) assms(2) assms(3) local.test-kat-2 mult-assoc test-pres2*)
hence $p \cdot (q \cdot x)^\dagger \leq (q \cdot x)^\dagger \cdot p$
by (*simp add: local.dagger-simr mult-assoc*)
thus $p \cdot (q \cdot x)^\dagger \cdot !q \leq p \cdot (q \cdot x)^\dagger \cdot !q \cdot p$
by (*metis assms(1) assms(2) local.eq-iff local.test-comp-closed local.test-kat-2*
local.test-mult-comm-var mult-assoc)
qed

theorem *composition*:
assumes *test p and test q*
and *pres y p and pres y (!p)*
shows $(p \cdot x)^\dagger \cdot !p \cdot (q \cdot y)^\dagger \cdot !q = !p \cdot (q \cdot y)^\dagger \cdot !q + p \cdot (p \cdot x \cdot (!p \cdot (q \cdot y)^\dagger \cdot !q + p))^\dagger \cdot !p$
by (*metis assms(1) assms(2) assms(4) composition-helper local.test-comp-closed*
mult-assoc postcomputation)
end

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

sublocale *kat* \subseteq *pre-conway star ..*

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

sublocale *drat* \subseteq *pre-conway strong-iteration*
apply *standard*
apply (*simp add: local.iteration-denest local.iteration-slide*)
apply (*metis local.iteration-prod-unfold*)
by (*simp add: local.iteration-sim*)

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

end

8 Propositional Hoare Logic

```
theory PHL-KAT
  imports KAT ../Kleene-Algebra/PHL-KA
begin
```

We define a class of pre-dioids with notions of assertions, tests and iteration. The above rules of PHL are derivable in that class.

```
class at-pre-dioid = pre-dioid-one +
  fixes alpha :: 'a ⇒ 'a (α)
  and tau :: 'a ⇒ 'a (τ)
  assumes at-pres: α x · τ y ≤ τ y · α x · τ y
  and as-left-supdistl: α x · (y + z) ≤ α x · y + α x · z
```

```
begin
```

Only the conditional and the iteration rule need to be considered (in addition to the export laws. In this context, they no longer depend on external assumptions. The other ones do not depend on any assumptions anyway.

```
lemma at-phl-cond:
```

```
assumes α u · τ v · x ≤ x · z and α u · τ w · y ≤ y · z
shows α u · (τ v · x + τ w · y) ≤ (τ v · x + τ w · y) · z
  using assms as-left-supdistl at-pres phl-cond by blast
```

```
lemma ht-at-phl-cond:
```

```
assumes {α u · τ v} x {z} and {α u · τ w} y {z}
shows {α u} (τ v · x + τ w · y) {z}
  using assms by (fact at-phl-cond)
```

```
lemma ht-at-phl-export1:
```

```
assumes {α x · τ y} z {w}
shows {α x} τ y · z {w}
  by (simp add: assms at-pres ht-phl-export1)
```

```
lemma ht-at-phl-export2:
```

```
assumes {x} y {α z}
shows {x} y · τ w {α z · τ w}
  using assms at-pres ht-phl-export2 by auto
```

```
end
```

```
class at-it-pre-dioid = at-pre-dioid + it-pre-dioid
begin
```

```
lemma at-phl-while:
```

```
assumes α p · τ s · x ≤ x · α p
shows α p · (it (τ s · x) · τ w) ≤ it (τ s · x) · τ w · (α p · τ w)
  by (simp add: assms at-pres it-simr phl-while)
```

lemma *ht-at-phl-while*:
assumes $\{\alpha\ p \cdot \tau\ s\} \ x \ \{\alpha\ p\}$
shows $\{\alpha\ p\} \ \text{it} \ (\tau\ s \cdot x) \cdot \tau\ w \ \{\alpha\ p \cdot \tau\ w\}$
using *assms* **by** (*fact at-phl-while*)

end

The following statements show that pre-Conway algebras, Kleene algebras with tests and demonic refinement algebras form pre-dioids with test and assertions. This automatically generates propositional Hoare logics for these structures.

sublocale *test-pre-dioid-zero* < *phl*: *at-pre-dioid* **where** *alpha* = *t* **and** *tau* = *t*
proof

show $\bigwedge x\ y. \ t\ x \cdot t\ y \leq t\ y \cdot t\ x \cdot t\ y$
by (*simp add: n-mult-comm mult-assoc*)
show $\bigwedge x\ y\ z. \ t\ x \cdot (y + z) \leq t\ x \cdot y + t\ x \cdot z$
by (*simp add: n-left-distrib*)

qed

sublocale *test-pre-conway* < *phl*: *at-it-pre-dioid* **where** *alpha* = *t* **and** *tau* = *t*
and *it* = *dagger* ..

sublocale *kat-zero* < *phl*: *at-it-pre-dioid* **where** *alpha* = *t* **and** *tau* = *t* **and** *it*
= *star* ..

context *test-pre-dioid-zero* **begin**

abbreviation *if-then-else* :: 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a (*if - then - else - fi* [64,64,64]
63) **where**

if p then x else y fi $\equiv (p \cdot x + !p \cdot y)$

lemma *phl-n-cond*:

$\{\!n\ v \cdot n\ w\} \ x \ \{\!z\} \Longrightarrow \{\!n\ v \cdot t\ w\} \ y \ \{\!z\} \Longrightarrow \{\!n\ v\} \ n\ w \cdot x + t\ w \cdot y \ \{\!z\}$
by (*metis phl.ht-at-phl-cond t-n-closed*)

lemma *phl-test-cond*:

assumes *test p* **and** *test b*
and $\{\!p \cdot b\} \ x \ \{\!q\}$ **and** $\{\!p \cdot !b\} \ y \ \{\!q\}$
shows $\{\!p\} \ b \cdot x + !b \cdot y \ \{\!q\}$
by (*metis assms local.test-double-comp-var phl-n-cond*)

lemma *phl-cond-syntax*:

assumes *test p* **and** *test b*
and $\{\!p \cdot b\} \ x \ \{\!q\}$ **and** $\{\!p \cdot !b\} \ y \ \{\!q\}$
shows $\{\!p\} \ \text{if } b \ \text{then } x \ \text{else } y \ \text{fi} \ \{\!q\}$
using *assms* **by** (*fact phl-test-cond*)

lemma *phl-cond-syntax-iff*:

```

assumes test p and test b
shows  $\{p \cdot b\} x \{q\} \wedge \{p \cdot !b\} y \{q\} \longleftrightarrow \{p\} \text{ if } b \text{ then } x \text{ else } y \text{ fi } \{q\}$ 
proof
  assume a:  $\{p \cdot b\} x \{q\} \wedge \{p \cdot !b\} y \{q\}$ 
  show  $\{p\} \text{ if } b \text{ then } x \text{ else } y \text{ fi } \{q\}$ 
    using a assms(1) assms(2) phl-test-cond by auto
next
  assume a:  $\{p\} \text{ if } b \text{ then } x \text{ else } y \text{ fi } \{q\}$ 
  have  $p \cdot b \cdot x \leq b \cdot p \cdot (b \cdot x + !b \cdot y)$ 
    by (metis assms(1) assms(2) local.mult.assoc local.subdistl local.test-preserve)
  also have  $\dots \leq b \cdot (b \cdot x + !b \cdot y) \cdot q$ 
    using a local.mult-isol mult-assoc by auto
  also have  $\dots = (b \cdot b \cdot x + b \cdot !b \cdot y) \cdot q$ 
    using assms(2) local.n-left-distrib-var mult-assoc by presburger
  also have  $\dots = b \cdot x \cdot q$ 
    by (simp add: assms(2))
  finally have b:  $p \cdot b \cdot x \leq x \cdot q$ 
    by (metis assms(2) local.order-trans local.test-restrictl mult-assoc)
  have  $p \cdot !b \cdot y = !b \cdot p \cdot !b \cdot y$ 
    by (simp add: assms(1) assms(2) local.test-preserve)
  also have  $\dots \leq !b \cdot p \cdot (b \cdot x + !b \cdot y)$ 
    by (simp add: local.mult-isol mult-assoc)
  also have  $\dots \leq !b \cdot (b \cdot x + !b \cdot y) \cdot q$ 
    using a local.mult-isol mult-assoc by auto
  also have  $\dots = (!b \cdot b \cdot x + !b \cdot !b \cdot y) \cdot q$ 
    using local.n-left-distrib mult-assoc by presburger
  also have  $\dots = !b \cdot y \cdot q$ 
    by (simp add: assms(2))
  finally have c:  $p \cdot !b \cdot y \leq y \cdot q$ 
    by (metis local.dual-order.trans local.n-restrictl mult-assoc)
  show  $\{p \cdot b\} x \{q\} \wedge \{p \cdot !b\} y \{q\}$ 
    using b c by auto
qed

end

context test-pre-conway
begin

lemma phl-n-while:
  assumes  $\{n x \cdot n y\} z \{n x\}$ 
  shows  $\{n x\} (n y \cdot z)^\dagger \cdot t y \{n x \cdot t y\}$ 
  by (metis assms phl.ht-at-phl-while t-n-closed)

end

context kat-zero1
begin

```


abbreviation $\text{while} :: 'a \Rightarrow 'a \Rightarrow 'a$ (*while - do - od [64,64] 63*) **where**
 $\text{while } b \text{ do } x \text{ od} \equiv (b \cdot x)^* \cdot !b$

lemma *phl-n-while*:
assumes $\{n\ x \cdot n\ y\} z \{n\ x\}$
shows $\{n\ x\} (n\ y \cdot z)^* \cdot t\ y \{n\ x \cdot t\ y\}$
by (*metis assms phl.ht-at-phl-while t-n-closed*)

lemma *phl-test-while*:
assumes *test p and test b*
and $\{p \cdot b\} x \{p\}$
shows $\{p\} (b \cdot x)^* \cdot !b \{p \cdot !b\}$
by (*metis assms phl-n-while test-double-comp-var*)

lemma *phl-while-syntax*:
assumes *test p and test b and* $\{p \cdot b\} x \{p\}$
shows $\{p\} \text{while } b \text{ do } x \text{ od} \{p \cdot !b\}$
by (*metis assms phl-test-while*)

end

lemma (*in kat*) $\text{test } p \Longrightarrow p \cdot x^* \leq x^* \cdot p \Longrightarrow p \cdot x \leq x \cdot p$
oops

lemma (*in kat*) $\text{test } p \Longrightarrow \text{test } b \Longrightarrow p \cdot (b \cdot x)^* \cdot !b \leq (b \cdot x)^* \cdot !b \cdot p \cdot !b \Longrightarrow p \cdot b \cdot x \leq x \cdot p$
oops

lemma (*in kat*) $\text{test } p \Longrightarrow \text{test } q \Longrightarrow p \cdot x \cdot y \leq x \cdot y \cdot q \Longrightarrow (\exists r. \text{test } r \wedge p \cdot x \leq x \cdot r \wedge r \cdot y \leq y \cdot q)$
oops

lemma (*in kat*) $\text{test } p \Longrightarrow \text{test } q \Longrightarrow p \cdot x \cdot y \cdot !q = 0 \Longrightarrow (\exists r. \text{test } r \wedge p \cdot x \cdot !r = 0 \wedge r \cdot y \cdot !q = 0)$
oops

The following facts should be moved. They show that the rules of Hoare logic based on Tarlecki triples are invertible.

abbreviation (*in near-dioid*) $\text{tt} :: 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow \text{bool}$ ($(|-)|-|)$ **where**
 $(|x|) y (|z|) \equiv x \cdot y \leq z$

lemma (*in near-dioid-one*) *tt-skip*: $(|p|) 1 (|p|)$
by *simp*

lemma (*in near-dioid*) *tt-cons1*: $(\exists q'. (|p|) x (|q'|) \wedge q' \leq q) \longleftrightarrow (|p|) x (|q|)$
using *local.order-trans* **by** *blast*

lemma (*in near-dioid*) *tt-cons2*: $(\exists p'. (|p'|) x (|q|) \wedge p \leq p') \longleftrightarrow (|p|) x (|q|)$
using *local.dual-order.trans local.mult-isor* **by** *blast*

lemma (in *near-diod*) *tt-seq*: $(\exists r. \langle p \rangle x \langle r \rangle \wedge \langle r \rangle y \langle q \rangle) \longleftrightarrow \langle p \rangle x \cdot y \langle q \rangle$
by (*metis local.tt-cons2 mult-assoc*)

lemma (in *diod*) *tt-cond*: $\langle p \cdot v \rangle x \langle q \rangle \wedge \langle p \cdot w \rangle y \langle q \rangle \longleftrightarrow \langle p \rangle (v \cdot x + w \cdot y) \langle q \rangle$
by (*simp add: local.distrib-left mult-assoc*)

lemma (in *kleene-algebra*) *tt-while*: $w \leq 1 \implies \langle p \cdot v \rangle x \langle p \rangle \implies \langle p \rangle (v \cdot x)^* \cdot w \langle p \rangle$
by (*metis local.star-inductr-var-equiv local.star-subid local.tt-seq local.tt-skip mult-assoc*)

The converse implication can be refuted. The situation is similar to the ht case.

lemma (in *kat*) *tt-while*: $\text{test } v \implies \langle p \rangle (v \cdot x)^* \cdot !v \langle p \rangle \implies \langle p \cdot v \rangle x \langle p \rangle$
oops

lemma (in *kat*) *tt-while*: $\text{test } v \implies \langle p \rangle (v \cdot x)^* \langle p \rangle \implies \langle p \cdot v \rangle x \langle p \rangle$
using *local.star-inductr-var-equiv mult-assoc* **by** *auto*

Perhaps this holds with possibly infinite loops in DRA...

wlp in KAT

lemma (in *kat*) *test y*: $\text{test } y \implies (\exists z. \text{test } z \wedge z \cdot x \cdot !y = 0)$
by (*metis local.annil local.test-zero-var*)

end

9 Propositional Hoare Logic

theory *PHL-DRAT*
imports *DRAT ../Kleene-Algebra/PHL-DRA PHL-KAT*
begin

sublocale *drat* < *phl*: *at-it-pre-diod* **where** *alpha* = *t* **and** *tau* = *t* **and** *it* = *strong-iteration ..*

context *drat*
begin

no-notation *while* (*while - do - od* [64,64] 63)

abbreviation *while* :: '*a* \Rightarrow '*a* \Rightarrow '*a* (*while - do - od* [64,64] 63) **where**
while *b do x od* $\equiv (b \cdot x)^\infty \cdot !b$

lemma *phl-n-while*:
assumes $\langle n x \cdot n y \rangle z \langle n x \rangle$
shows $\langle n x \rangle (n y \cdot z)^\infty \cdot t y \langle n x \cdot t y \rangle$
by (*metis assms phl.ht-at-phl-while t-n-closed*)

```

lemma phl-test-while:
  assumes test p and test b
  and  $\{p \cdot b\} x \{p\}$ 
  shows  $\{p\} (b \cdot x)^\infty \cdot !b \{p \cdot !b\}$ 
  by (metis assms phl-n-while test-double-comp-var)

```

```

lemma phl-while-syntax:
  assumes test p and test b and  $\{p \cdot b\} x \{p\}$ 
  shows  $\{p\} \text{ while } b \text{ do } x \text{ od } \{p \cdot !b\}$ 
  by (metis assms phl-test-while)

```

end

end

10 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  $\Rightarrow$  'a ('-')

```

```

ML  $\langle\langle$ 
  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)

```

```

structure KATHomRules = Named-Thms
  (val name = @{binding kat-hom}
   val description = KAT test homomorphism rules)

```

```

fun kat-hom-tac ctxt n =
  let
    val rev-rules = map (fn thm => thm RS @{thm sym}) (KATHomRules.get ctxt)

```

```

    in
      asm-full-simp-tac (put-simpset HOL-basic-ss ctxt addsimps rev-rules) n
    end
  >>

setup << KATHomRules.setup >>

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
  >>

ML <<
  structure VCGRules = Named-Thms
    (val name = @{binding vcg}
     val description = verification condition generator rules)

  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' (VCGRules.get ctxt))
       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 = <<
  Scan.succeed (fn ctxt => SIMPLE-METHOD (CHANGED (vcg-tac ctxt 1)))
  >>

setup << VCGRules.setup >>

locale dioid-tests =
  fixes test :: 'a::boolean-algebra => 'b::dioid-one-zero
  and not :: 'b::dioid-one-zero => 'b::dioid-one-zero (-)
  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]:  $p \leq q \longleftrightarrow 'p \leq q'$ 
begin

notation test ( $\iota$ )

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

ML-val  $\ll \text{map } (fn \text{ thm } \Rightarrow \text{ thm } RS \ @\{ \text{ thm } \text{ sym} \}) (KATHomRules.get \ @\{ \text{ context} \})$ 
\gg

lemma test-iso:  $p \leq q \Longrightarrow 'p \leq q'$ 
  by (simp add: test-iso-eq)

lemma test-meet-comm:  $'p \cdot q = q \cdot p'$ 
  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]:  $'(-p) \cdot p = 0'$ 
  by (metis inf.commute inf-compl-bot test-bot test-inf test-not)

lemma [simp]:  $'p \cdot (-p) = 0'$ 
  by (metis inf-compl-bot test-bot test-inf test-not)

end

locale kat =
  fixes test ::  $'a::boolean-algebra \Rightarrow 'b::kleene-algebra$ 
  and not ::  $'b::kleene-algebra \Rightarrow 'b::kleene-algebra (!)$ 
  assumes is-diooid-tests: diooid-tests test not

sublocale kat  $\subseteq$  diooid-tests using is-diooid-tests .

context kat
begin

notation test ( $\iota$ )

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

```

ML-val $\ll \text{map } (fn \text{ thm} \Rightarrow \text{thm } RS \ @\{ \text{thm } sym \}) (KATHomRules.get \ @\{ \text{context} \})$
 \gg

lemma *test-iso*: $p \leq q \implies 'p \leq q'$
by (*simp add: test-iso-eq*)

lemma *test-meet-comm*: $'p \cdot q = q \cdot p'$
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*]: $'!p \cdot p = 0'$
by (*metis inf.commute inf-compl-bot test-bot test-inf test-not*)

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

definition *hoare-triple* :: $'b \Rightarrow 'b \Rightarrow 'b \Rightarrow bool$ ($\{\!-\!\}$ - $\{\!-\!\}$) **where**
 $\{\!p\!\} c \{\!q\!\} \equiv p \cdot c \leq c \cdot q$

declare *hoare-triple-def*[*iff*]

lemma *hoare-triple-def-var*: $'p \cdot c \leq c \cdot q \iff p \cdot c \cdot !q = 0'$
apply (*intro iffI antisym*)
apply (*rule order-trans*[*of - 'c \cdot q \cdot !q'*])
apply (*rule mult-isor*[*rule-format*])
apply (*simp add: mult.assoc*)
apply (*simp add: mult.assoc*[*symmetric*])
apply (*rule order-trans*[*of - 'p \cdot c \cdot (!q + q)'*])
apply *simp*
apply (*simp only: distrib-left add-zero*)
apply (*rule order-trans*[*of - '1 \cdot c \cdot q'*])
apply (*simp only: mult.assoc*)
apply (*rule mult-isor*[*rule-format*])
by *simp-all*

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

lemma hoare-weakening: $p \leq p' \implies q' \leq q \implies \{p'\} c \{q'\} \implies \{p\} c \{q\}$
by auto (metis mult-isol mult-isor order-trans test-iso)

lemma hoare-star: $\{p\} c \{p\} \implies \{p\} c^* \{p\}$
by auto

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

lemma hoare-test [vcg]: $\{p \cdot t \leq q\} \implies \{p\} t \{q\}$
by auto (metis inf-le2 le-inf-iff test-inf test-iso-eq)

lemma hoare-mult [vcg]: $\{p\} x \{r\} \implies \{r\} y \{q\} \implies \{p\} x \cdot y \{q\}$
proof auto

assume [simp]: $p \cdot x \leq x \cdot r$ **and** [simp]: $r \cdot y \leq y \cdot q$

have $p \cdot (x \cdot y) \leq x \cdot r \cdot y$

by (auto simp add: mult.assoc[symmetric] intro!: mult-isor[rule-format])

also have $\dots \leq x \cdot y \cdot q$

by (auto simp add: mult.assoc intro!: mult-isol[rule-format])

finally show $p \cdot (x \cdot y) \leq x \cdot y \cdot q$.

qed

lemma [simp]: $\{!p \cdot !p = !p\}$
by (metis inf.idem test-inf test-not)

lemma hoare-plus [vcg]: $\{p\} x \{q\} \implies \{p\} y \{q\} \implies \{p\} x + y \{q\}$

proof –

assume a1: $\{! \iota p\} x \{! \iota q\}$

assume $\{! \iota p\} y \{! \iota q\}$

hence $\iota p \cdot (x + y) \leq x \cdot \iota q + y \cdot \iota q$

using a1 **by** (metis (no-types) distrib-left hoare-triple-def join.sup.mono)

thus ?thesis

by force

qed

definition While :: $'b \Rightarrow 'b \Rightarrow 'b$ (While - Do - End [50,50] 51) **where**
 While t Do c End = (t.c)*.!t

lemma hoare-while: $\{p \cdot t\} c \{p\} \implies \{p\} \text{While } t \text{ Do } c \text{ End } \{!t \cdot p\}$
unfolding While-def **by** vcg (metis inf-commute order-refl)

lemma [vcg]: $\{p \cdot t\} c \{p\} \implies \{!t \cdot p \leq q\} \implies \{p\} \text{While } t \text{ Do } c \text{ End } \{q\}$
by (metis hoare-weakening hoare-while order-refl test-inf test-iso-eq test-not)

definition If :: $'b \Rightarrow 'b \Rightarrow 'b \Rightarrow 'b$ (If - Then - Else - [50,50,50] 51) **where**
 If p Then c1 Else c2 $\equiv p \cdot c1 + !p \cdot c2$

lemma *hoare-if* [vcg]: ‘ $\{p \cdot t\} c1 \{q\}$ ’ \implies ‘ $\{p \cdot !t\} c2 \{q\}$ ’ \implies ‘ $\{p\}$ If *t* Then *c1* Else *c2* $\{q\}$ ’

unfolding *If-def* by *vcg assumption*

end

end

11 Two sorted Demonic Refinement Algebras

theory *DRAT2*

imports *../Kleene-Algebra/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* (‘-‘)

ML $\langle\langle$

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 map-ast-variables xs)

structure DRAHomRules = *Named-Thms*

(val name = $\text{@}\{\textit{binding kat-hom}\}$

val description = *KAT test homomorphism rules*)

fun dra-hom-tac ctxt n =

let

val rev-rules = *map (fn thm => thm RS $\text{@}\{\textit{thm sym}\}$) (DRAHomRules.get*

ctxt)

in

asm-full-simp-tac (put-simpset HOL-basic-ss ctxt addsimps rev-rules) n

end

$\rangle\rangle$

setup $\langle\langle$ *DRAHomRules.setup* $\rangle\rangle$


```

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

```

```

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

```

```

ML ⟨⟨
  structure VCGRules = Named-Thms
    (val name = @{binding vcg}
     val description = verification condition generator rules)

```

```

  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' (VCGRules.get ctxt))
       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)))
  ⟩⟩

```

```

setup ⟨⟨ VCGRules.setup ⟩⟩

```

```

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*]: $p = q \iff 'p = q'$
by (*metis eq-iff test-iso-eq*)

ML-val $\ll \text{map } (fn \text{ thm} \Rightarrow \text{thm } RS \ @\{ \text{thm } \text{sym}\}) (DRAHomRules.get \ @\{ \text{context}\}) \gg$

lemma *test-iso*: $p \leq q \implies 'p \leq q'$
by (*simp add: test-iso-eq*)

lemma *test-meet-comm*: $'p \cdot q = q \cdot p'$
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*]: $'!p \cdot p = 0'$
by (*metis inf.commute inf-compl-bot test-bot test-inf test-not*)

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

lemma *test-eq1*: $'y \leq x' \iff 'p \cdot y \leq x' \wedge '!p \cdot y \leq x'$
apply *standard*
apply (*metis mult-isol-var mult-onel test-not test-one-top*)
apply (*subgoal-tac '(p + !p) \cdot y \leq x'*)
apply (*metis mult-onel sup-compl-top test-not test-sup test-top*)
by (*metis distrib-right' join.sup.bounded-iff*)

lemma $'p \cdot x = p \cdot x \cdot q' \implies 'p \cdot x \cdot !q = 0'$
nitpick **oops**

lemma *test1*: $'p \cdot x \cdot !q = 0' \implies 'p \cdot x = p \cdot x \cdot q'$
by (*metis add-0-left distrib-left mult-oner test-comp-add1*)

lemma *test2*: $'p \cdot q \cdot p = p \cdot q'$
by (*metis inf.commute inf.left-idem test-inf*)

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

lemma *test4*: ‘ $!p \cdot q \cdot p = 0$ ’
by (*metis double-compl test3 test-not*)

lemma *total-correctness*: ‘ $p \cdot x \cdot !q = 0 \iff x \cdot !q \leq !p \cdot \top$ ’
apply *standard*
apply (*metis join.bot.extremum mult.assoc test-eq1 top-elim*)
by (*metis (no-types, hide-lams) add-zero annil less-eq-def mult.assoc mult-isol test-comp-mult2*)

lemma *test-iteration-sim*: ‘ $p \cdot x \leq x \cdot p \implies p \cdot x^\infty \leq x^\infty \cdot p$ ’
by (*metis iteration-sim*)

lemma *test-iteration-annil*: ‘ $!p \cdot (p \cdot x)^\infty = !p$ ’
by (*metis annil iteration-idep mult.assoc test-comp-mult1*)

end

end

References

- [1] E. Cohen. Separation and reduction. In R. C. Backhouse and J. N. Oliveira, editors, *MPC*, volume 1837 of *LNCS*, pages 45–59. Springer, 2000.
- [2] P. Höfner, G. Struth, and G. Sutcliffe. Automated verification of refinement laws. *Ann. Mathematics and Artificial Intelligence*, 55(1-2):35–62, 2009.
- [3] D. Kozen. Kleene algebra with tests. *ACM TOPLAS*, 19(3):427–443, 1997.
- [4] V. Preteasa. Algebra of monotonic boolean transformers. In A. S. Simão and C. Morgan, editors, *SBMF*, volume 7021 of *LNCS*, pages 140–155. Springer, 2011.
- [5] J. von Wright. From Kleene algebra to refinement algebra. In E. A. Boiten and B. Möller, editors, *MPC*, volume 2386 of *LNCS*, pages 233–262. Springer, 2002.