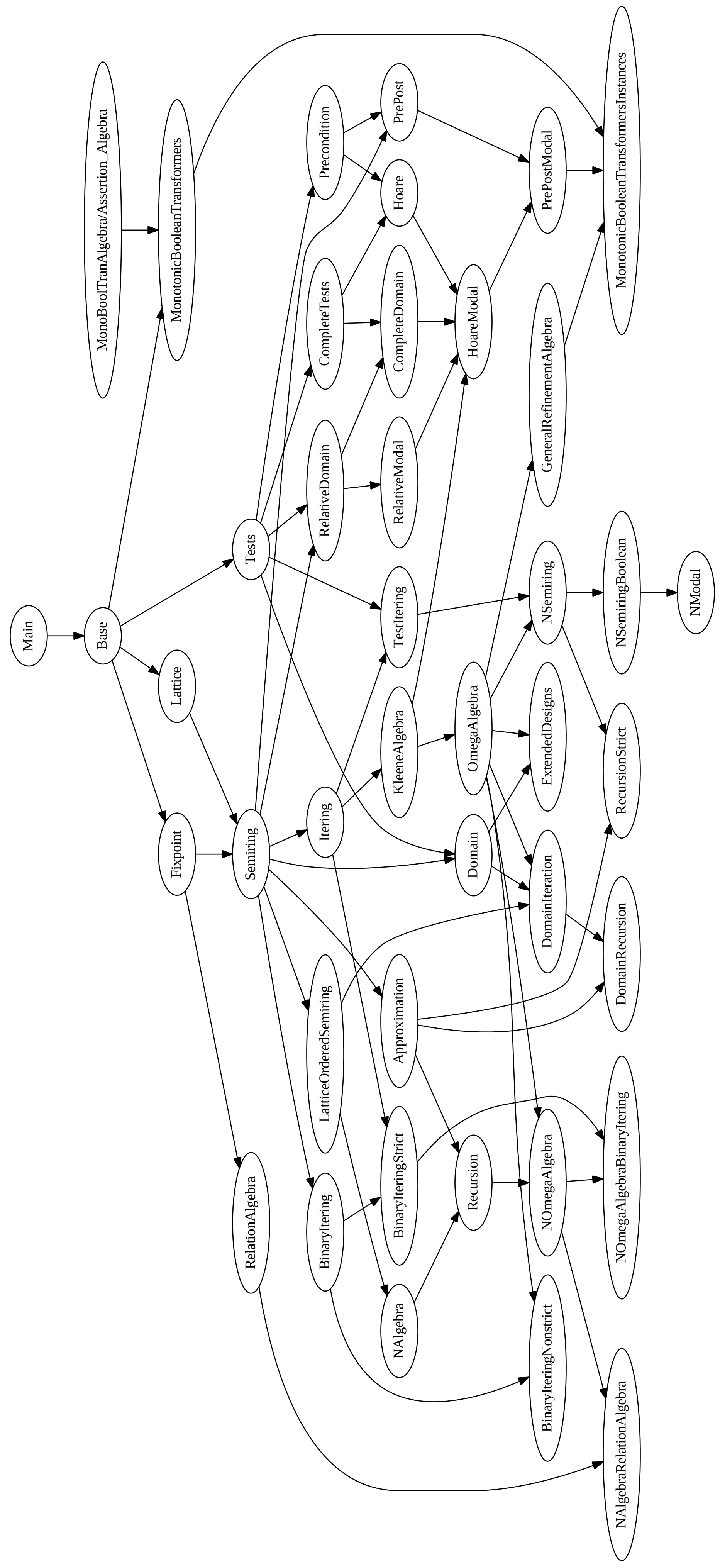
Isabelle/HOL Theories of Algebras for Iteration, Infinite Executions and Correctness of Sequential Computations

Walter Guttmann

Technical Report TR-COSC 02/15
Department of Computer Science and Software Engineering
University of Canterbury
2015

This technical report is a reference for a separate document that describes its results. The following theories have been developed with Isabelle2014. The dependences of the theory files are shown on the following page. Not reproduced are the Isabelle/HOL Main theory and Viorel Preoteasa's theories LatticeProperties and MonoBoolTranAlgebra, which are available from the Archive of Formal Proofs and required by MonotonicBooleanTransformers.

Isabelle/HOL Theories



Main
Base
MonoBoolTranAlgebra/Assertion_Algebra
MonotonicBooleanTransformers
Lattice
Fixpoint
Tests
Semiring
Precondition
PrePost
Hoare
CompleteTests
CompleteDomain
HoareModal
PrePostModal
MonotonicBooleanTransformersInstances
RelativeDomain
RelativeModal
GeneralRefinementAlgebra
TestItering
NSemiring
NSemiringBoolean
NModal
Itering
KleeneAlgebra
OmegaAlgebra
ExtendedDesigns
RecursionStrict
Domain
DomainIteration
DomainRecursion
LatticeOrderedSemiring
Approximation
BinaryIteringStrict
Recursion
NOmegaAlgebra
NOmegaAlgebraBinaryItering
RelationAlgebra
BinaryItering
NAlgebra
BinaryIteringNonstrict
NAlgebraRelationAlgebra

1 Base

theory *Base*

imports *Main*

begin

— This theory and the subsequent theories have been developed with Isabelle2014.

nitpick-params [*timeout* = *600*]

declare [[*smt-timeout* = *600*]]

class *mult* = *times*

begin

notation
  times (**infixl** · *70*) **and**
  times (**infixl** ; *70*)

end

class *neg* = *uminus*

begin

no-notation
  uminus (− - [*81*] *80*)

notation
  uminus (− - [*80*] *80*)

end

class *while* =
  fixes *while* :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixr** ⋆ *59*)

class *L* =
  fixes *L* :: $'a$

class *n* =
  fixes *n* :: $'a \Rightarrow 'a$

class *d* =
  fixes *d* :: $'a \Rightarrow 'a$

class *diamond* =
  fixes *diamond* :: $'a \Rightarrow 'a \Rightarrow 'a$ (| - > - [*50*,*90*] *95*)

class *box* =
  fixes *box* :: $'a \Rightarrow 'a \Rightarrow 'a$ (| -] - [*50*,*90*] *95*)

context *ord*

begin

definition *isotone* :: $('a \Rightarrow 'a) \Rightarrow bool$
  where *isotone f* $\longleftrightarrow (\forall x\ y\ .\ x \le y \longrightarrow f(x) \le f(y))$

definition *lifted-less-eq* :: $('a \Rightarrow 'a) \Rightarrow ('a \Rightarrow 'a) \Rightarrow bool$ ((- ≤≤ -) [*51*, *51*] *50*)
  where $f \le\le g \longleftrightarrow (\forall x\ .\ f(x) \le g(x))$

definition *galois* :: $('a \Rightarrow 'a) \Rightarrow ('a \Rightarrow 'a) \Rightarrow bool$
  where *galois l u* $\longleftrightarrow (\forall x\ y\ .\ l(x) \le y \longleftrightarrow x \le u(y))$

definition *ascending-chain* :: $(nat \Rightarrow 'a) \Rightarrow bool$
  where *ascending-chain f* $\longleftrightarrow (\forall n\ .\ f\ n \le f\ (Suc\ n))$

definition *descending-chain* :: (*nat* ⇒ ′*a*) ⇒ *bool*
  where *descending-chain f* ⟷ (∀ *n* . *f* (*Suc n*) ≤ *f n*)

definition *directed* :: ′*a set* ⇒ *bool*
  where *directed X* ⟷ *X* ≠ {} ∧ (∀ *x*∈*X* . ∀ *y*∈*X* . ∃ *z*∈*X* . *x* ≤ *z* ∧ *y* ≤ *z*)

definition *codirected* :: ′*a set* ⇒ *bool*
  where *codirected X* ⟷ *X* ≠ {} ∧ (∀ *x*∈*X* . ∀ *y*∈*X* . ∃ *z*∈*X* . *z* ≤ *x* ∧ *z* ≤ *y*)

definition *chain* :: ′*a set* ⇒ *bool*
  where *chain X* ⟷ (∀ *x*∈*X* . ∀ *y*∈*X* . *x* ≤ *y* ∨ *y* ≤ *x*)

end

context *order*

begin

lemma *lifted-reflexive*: *f* = *g* ⟶ *f* ≤≤ *g*
  by (*metis lifted-less-eq-def order-refl*)

lemma *lifted-transitive*: *f* ≤≤ *g* ∧ *g* ≤≤ *h* ⟶ *f* ≤≤ *h*
  by (*smt lifted-less-eq-def order-trans*)

lemma *lifted-antisymmetric*: *f* ≤≤ *g* ∧ *g* ≤≤ *f* ⟶ *f* = *g*
  by (*metis* (*full-types*) *antisym ext lifted-less-eq-def*)

lemma *galois-char*: *galois l u* ⟷ (∀ *x* . *x* ≤ *u*(*l*(*x*))) ∧ (∀ *x* . *l*(*u*(*x*)) ≤ *x*) ∧ *isotone l* ∧ *isotone u*
  apply (*rule iffI*)
  apply (*metis* (*full-types*) *galois-def isotone-def order-refl order-trans*)
  apply (*metis galois-def isotone-def order-trans*)
  done

lemma *galois-closure*: *galois l u* ⟶ *l*(*x*) = *l*(*u*(*l*(*x*))) ∧ *u*(*x*) = *u*(*l*(*u*(*x*)))
  by (*smt antisym galois-char isotone-def*)

lemma *ascending-chain-k*: *ascending-chain f* ⟶ *f m* ≤ *f* (*m* + *k*)
  apply (*induct k*)
  apply *simp*
  apply (*metis add-Suc-right ascending-chain-def order-trans*)
  done

lemma *ascending-chain-isotone*: *ascending-chain f* ∧ *m* ≤ *k* ⟶ *f m* ≤ *f k*
  by (*metis ascending-chain-k le-iff-add*)

lemma *ascending-chain-comparable*: *ascending-chain f* ⟶ *f k* ≤ *f m* ∨ *f m* ≤ *f k*
  by (*metis nat-le-linear ascending-chain-isotone*)

lemma *ascending-chain-chain*: *ascending-chain f* ⟶ *chain* (*range f*)
  by (*smt ascending-chain-comparable chain-def image-iff*)

lemma *chain-directed*: *X* ≠ {} ∧ *chain X* ⟶ *directed X*
  by (*metis chain-def directed-def*)

lemma *ascending-chain-directed*: *ascending-chain f* ⟶ *directed* (*range f*)
  by (*metis UNIV-not-empty ascending-chain-chain chain-directed empty-is-image*)

lemma *descending-chain-k*: *descending-chain f* ⟶ *f* (*m* + *k*) ≤ *f m*
  apply (*induct k*)
  apply *simp*
  apply (*metis add-Suc-right descending-chain-def order-trans*)
  done

lemma *descending-chain-antitone*: *descending-chain f* ∧ *m* ≤ *k* ⟶ *f k* ≤ *f m*
  by (*metis descending-chain-k le-iff-add*)

lemma *descending-chain-comparable*: *descending-chain f* ⟶ *f k* ≤ *f m* ∨ *f m* ≤ *f k*
  by (*metis nat-le-linear descending-chain-antitone*)

```
lemma descending-chain-chain: descending-chain f ⟶ chain (range f)
  unfolding chain-def
  apply simp
  apply (smt descending-chain-comparable)
  done

lemma chain-codirected: X ≠ {} ∧ chain X ⟶ codirected X
  by (metis chain-def codirected-def)

lemma descending-chain-codirected: descending-chain f ⟶ codirected (range f)
  by (metis UNIV-not-empty descending-chain-chain chain-codirected empty-is-image)

end

context complete-lattice

begin

lemma sup-Sup: assumes nonempty: A ≠ {}
   shows sup x (Sup A) = Sup ((sup x) ‘ A)
  apply (rule antisym)
  apply (metis Sup-mono Sup-upper2 assms ex-in-conv imageI le-supI sup-ge1 sup-ge2)
  apply (smt Sup-least Sup-upper image-iff le-iff-sup sup.commute sup-ge1 sup-left-commute)
  done

lemma sup-SUP: Y ≠ {} ⟶ sup x (SUP y:Y . f y) = (SUP y:Y. sup x (f y))
  unfolding SUP-def
  apply rule
  apply (subst sup-Sup)
  apply (smt empty-is-image)
  apply (metis image-image)
  done

lemma inf-Inf: assumes nonempty: A ≠ {}
   shows inf x (Inf A) = Inf ((inf x) ‘ A)
  apply (rule antisym)
  apply (smt Inf-greatest Inf-lower image-iff le-iff-inf inf.commute inf-le1 inf-left-commute)
  apply (metis Inf-mono Inf-lower2 assms ex-in-conv imageI le-infI inf-le1 inf-le2)
  done

lemma inf-INF: Y ≠ {} ⟶ inf x (INF y:Y . f y) = (INF y:Y. inf x (f y))
  unfolding INF-def
  apply rule
  apply (subst inf-Inf)
  apply (smt empty-is-image)
  apply (metis image-image)
  done

lemma SUP-image-id[simp]: (SUP x:f‘A . x) = (SUP x:A . f x)
  by simp

lemma INF-image-id[simp]: (INF x:f‘A . x) = (INF x:A . f x)
  by simp

end

lemma image-Collect-2: f ‘ { g x | x . P x } = { f (g x) | x . P x }
  by auto
```

— The following instantiation and four lemmas are from Jose Divason Mallagaray.

```
instantiation fun :: (type, type) power

begin

definition one-fun :: 'a ⇒ 'a
  where one-fun-def: one-fun ≡ id

definition times-fun :: ('a ⇒ 'a) ⇒ ('a ⇒ 'a) ⇒ ('a ⇒ 'a)
```

where *times-fun-def*: *times-fun* $\equiv$ *comp*

instance
by *intro-classes*

end

lemma *id-power*: *id^m = id*
apply (*induct m*)
apply (*metis one-fun-def power-0*)
apply (*simp add*: *times-fun-def*)
done

lemma *power-zero-id*: *f^0 = id*
by (*metis one-fun-def power-0*)

lemma *power-succ-unfold*: *f^Suc m = f* $\circ$ *f^m*
by (*metis power-Suc times-fun-def*)

lemma *power-succ-unfold-ext*: (*f^Suc m*) *x = f* ((*f^m*) *x*)
by (*metis o-apply power-succ-unfold*)

end

2 Fixpoint

theory *Fixpoint*

imports *Base*

begin

context *order*

begin

definition *is-fixpoint* :: $('a \Rightarrow 'a) \Rightarrow 'a \Rightarrow bool$ **where** *is-fixpoint* $f\,x \longleftrightarrow f(x) = x$
definition *is-prefixpoint* :: $('a \Rightarrow 'a) \Rightarrow 'a \Rightarrow bool$ **where** *is-prefixpoint* $f\,x \longleftrightarrow f(x) \leq x$
definition *is-postfixpoint* :: $('a \Rightarrow 'a) \Rightarrow 'a \Rightarrow bool$ **where** *is-postfixpoint* $f\,x \longleftrightarrow f(x) \geq x$
definition *is-least-fixpoint* :: $('a \Rightarrow 'a) \Rightarrow 'a \Rightarrow bool$ **where** *is-least-fixpoint* $f\,x \longleftrightarrow f(x) = x \wedge (\forall y\ .\ f(y) = y \longrightarrow x \leq y)$
definition *is-greatest-fixpoint* :: $('a \Rightarrow 'a) \Rightarrow 'a \Rightarrow bool$ **where** *is-greatest-fixpoint* $f\,x \longleftrightarrow f(x) = x \wedge (\forall y\ .\ f(y) = y \longrightarrow x \geq y)$
definition *is-least-prefixpoint* :: $('a \Rightarrow 'a) \Rightarrow 'a \Rightarrow bool$ **where** *is-least-prefixpoint* $f\,x \longleftrightarrow f(x) \leq x \wedge (\forall y\ .\ f(y) \leq y \longrightarrow x \leq y)$
definition *is-greatest-postfixpoint* :: $('a \Rightarrow 'a) \Rightarrow 'a \Rightarrow bool$ **where** *is-greatest-postfixpoint* $f\,x \longleftrightarrow f(x) \geq x \wedge (\forall y\ .\ f(y) \geq y \longrightarrow x \geq y)$
definition *has-fixpoint* :: $('a \Rightarrow 'a) \Rightarrow bool$ **where** *has-fixpoint* $f \longleftrightarrow (\exists x\ .$ *is-fixpoint* $f\,x)$
definition *has-prefixpoint* :: $('a \Rightarrow 'a) \Rightarrow bool$ **where** *has-prefixpoint* $f \longleftrightarrow (\exists x\ .$ *is-prefixpoint* $f\,x)$
definition *has-postfixpoint* :: $('a \Rightarrow 'a) \Rightarrow bool$ **where** *has-postfixpoint* $f \longleftrightarrow (\exists x\ .$ *is-postfixpoint* $f\,x)$
definition *has-least-fixpoint* :: $('a \Rightarrow 'a) \Rightarrow bool$ **where** *has-least-fixpoint* $f \longleftrightarrow (\exists x\ .$ *is-least-fixpoint* $f\,x)$
definition *has-greatest-fixpoint* :: $('a \Rightarrow 'a) \Rightarrow bool$ **where** *has-greatest-fixpoint* $f \longleftrightarrow (\exists x\ .$ *is-greatest-fixpoint* $f\,x)$
definition *has-least-prefixpoint* :: $('a \Rightarrow 'a) \Rightarrow bool$ **where** *has-least-prefixpoint* $f \longleftrightarrow (\exists x\ .$ *is-least-prefixpoint* $f\,x)$
definition *has-greatest-postfixpoint* :: $('a \Rightarrow 'a) \Rightarrow bool$ **where** *has-greatest-postfixpoint* $f \longleftrightarrow (\exists x\ .$ *is-greatest-postfixpoint* $f\,x)$
definition *the-least-fixpoint* :: $('a \Rightarrow 'a) \Rightarrow 'a$ $(\mu$ - [201] 200) **where** $\mu\ f = (THE\ x\ .$ *is-least-fixpoint* $f\,x)$
definition *the-greatest-fixpoint* :: $('a \Rightarrow 'a) \Rightarrow 'a$ $(\nu$ - [201] 200) **where** $\nu\ f = (THE\ x\ .$ *is-greatest-fixpoint* $f\,x)$
definition *the-least-prefixpoint* :: $('a \Rightarrow 'a) \Rightarrow 'a$ $(p\mu$ - [201] 200) **where** $p\mu\ f = (THE\ x\ .$ *is-least-prefixpoint* $f\,x)$
definition *the-greatest-postfixpoint* :: $('a \Rightarrow 'a) \Rightarrow 'a$ $(p\nu$ - [201] 200) **where** $p\nu\ f = (THE\ x\ .$ *is-greatest-postfixpoint* $f\,x)$

lemma *least-fixpoint-unique*: *has-least-fixpoint* $f \longrightarrow (\exists!x\ .$ *is-least-fixpoint* $f\,x)$
  by (*smt antisym has-least-fixpoint-def is-least-fixpoint-def*)

lemma *greatest-fixpoint-unique*: *has-greatest-fixpoint* $f \longrightarrow (\exists!x\ .$ *is-greatest-fixpoint* $f\,x)$
  by (*smt antisym has-greatest-fixpoint-def is-greatest-fixpoint-def*)

lemma *least-prefixpoint-unique*: *has-least-prefixpoint* $f \longrightarrow (\exists!x\ .$ *is-least-prefixpoint* $f\,x)$
  by (*smt antisym has-least-prefixpoint-def is-least-prefixpoint-def*)

lemma *greatest-postfixpoint-unique*: *has-greatest-postfixpoint* $f \longrightarrow (\exists!x\ .$ *is-greatest-postfixpoint* $f\,x)$
  by (*smt antisym has-greatest-postfixpoint-def is-greatest-postfixpoint-def*)

lemma *least-fixpoint*: *has-least-fixpoint* $f \longrightarrow$ *is-least-fixpoint* $f\ (\mu\ f)$
proof
  assume *has-least-fixpoint f*
  hence *is-least-fixpoint* $f\ (THE\ x\ .$ *is-least-fixpoint* $f\,x)$
    by (*smt least-fixpoint-unique theI'*)
  thus *is-least-fixpoint* $f\ (\mu\ f)$
    by (*simp add*: *is-least-fixpoint-def the-least-fixpoint-def*)
qed

lemma *greatest-fixpoint*: *has-greatest-fixpoint* $f \longrightarrow$ *is-greatest-fixpoint* $f\ (\nu\ f)$
proof
  assume *has-greatest-fixpoint f*
  hence *is-greatest-fixpoint* $f\ (THE\ x\ .$ *is-greatest-fixpoint* $f\,x)$
    by (*smt greatest-fixpoint-unique theI'*)
  thus *is-greatest-fixpoint* $f\ (\nu\ f)$
    by (*simp add*: *is-greatest-fixpoint-def the-greatest-fixpoint-def*)
qed

lemma *least-prefixpoint*: *has-least-prefixpoint* $f \longrightarrow$ *is-least-prefixpoint* $f\ (p\mu\ f)$
proof
  assume *has-least-prefixpoint f*

  hence *is-least-prefixpoint f* $(THE\ x\ .\ $*is-least-prefixpoint f x*$)$
    by (*smt least-prefixpoint-unique theI'*)
  thus *is-least-prefixpoint f* $(p\mu\ f)$
    by (*simp add*: *is-least-prefixpoint-def the-least-prefixpoint-def*)
qed

lemma *greatest-postfixpoint*: *has-greatest-postfixpoint f* $\longrightarrow$ *is-greatest-postfixpoint f* $(p\nu\ f)$
proof
  assume *has-greatest-postfixpoint f*
  hence *is-greatest-postfixpoint f* $(THE\ x\ .\ $*is-greatest-postfixpoint f x*$)$
    by (*smt greatest-postfixpoint-unique theI'*)
  thus *is-greatest-postfixpoint f* $(p\nu\ f)$
    by (*simp add*: *is-greatest-postfixpoint-def the-greatest-postfixpoint-def*)
qed

lemma *least-fixpoint-same*: *is-least-fixpoint f x* $\longrightarrow x = \mu\ f$
  by (*metis least-fixpoint least-fixpoint-unique has-least-fixpoint-def*)

lemma *greatest-fixpoint-same*: *is-greatest-fixpoint f x* $\longrightarrow x = \nu\ f$
  by (*metis greatest-fixpoint greatest-fixpoint-unique has-greatest-fixpoint-def*)

lemma *least-prefixpoint-same*: *is-least-prefixpoint f x* $\longrightarrow x = p\mu\ f$
  by (*metis least-prefixpoint least-prefixpoint-unique has-least-prefixpoint-def*)

lemma *greatest-postfixpoint-same*: *is-greatest-postfixpoint f x* $\longrightarrow x = p\nu\ f$
  by (*metis greatest-postfixpoint greatest-postfixpoint-unique has-greatest-postfixpoint-def*)

lemma *least-fixpoint-char*: *is-least-fixpoint f x* $\longleftrightarrow$ *has-least-fixpoint f* $\wedge\ x = \mu\ f$
  by (*metis least-fixpoint-same has-least-fixpoint-def*)

lemma *least-prefixpoint-char*: *is-least-prefixpoint f x* $\longleftrightarrow$ *has-least-prefixpoint f* $\wedge\ x = p\mu\ f$
  by (*metis least-prefixpoint-same has-least-prefixpoint-def*)

lemma *greatest-fixpoint-char*: *is-greatest-fixpoint f x* $\longleftrightarrow$ *has-greatest-fixpoint f* $\wedge\ x = \nu\ f$
  by (*metis greatest-fixpoint-same has-greatest-fixpoint-def*)

lemma *greatest-postfixpoint-char*: *is-greatest-postfixpoint f x* $\longleftrightarrow$ *has-greatest-postfixpoint f* $\wedge\ x = p\nu\ f$
  by (*metis greatest-postfixpoint-same has-greatest-postfixpoint-def*)

lemma *mu-unfold*: *has-least-fixpoint f* $\longrightarrow f\ (\mu\ f) = \mu\ f$
  by (*metis is-least-fixpoint-def least-fixpoint*)

lemma *pmu-unfold*: *has-least-prefixpoint f* $\longrightarrow f\ (p\mu\ f) \le p\mu\ f$
  by (*metis is-least-prefixpoint-def least-prefixpoint*)

lemma *nu-unfold*: *has-greatest-fixpoint f* $\longrightarrow \nu\ f = f\ (\nu\ f)$
  by (*metis is-greatest-fixpoint-def greatest-fixpoint*)

lemma *pnu-unfold*: *has-greatest-postfixpoint f* $\longrightarrow p\nu\ f \le f\ (p\nu\ f)$
  by (*metis is-greatest-postfixpoint-def greatest-postfixpoint*)

lemma *least-prefixpoint-fixpoint*: *has-least-prefixpoint f* $\wedge$ *isotone f* $\longrightarrow$ *is-least-fixpoint f* $(p\mu\ f)$
  by (*smt eq-iff is-least-fixpoint-def is-least-prefixpoint-def isotone-def least-prefixpoint*)

lemma *pmu-mu*: *has-least-prefixpoint f* $\wedge$ *isotone f* $\longrightarrow p\mu\ f = \mu\ f$
  by (*smt has-least-fixpoint-def is-least-fixpoint-def least-fixpoint-unique least-prefixpoint-fixpoint least-fixpoint*)

lemma *greatest-postfixpoint-fixpoint*: *has-greatest-postfixpoint f* $\wedge$ *isotone f* $\longrightarrow$ *is-greatest-fixpoint f* $(p\nu\ f)$
  by (*smt eq-iff is-greatest-fixpoint-def is-greatest-postfixpoint-def isotone-def greatest-postfixpoint*)

lemma *pnu-nu*: *has-greatest-postfixpoint f* $\wedge$ *isotone f* $\longrightarrow p\nu\ f = \nu\ f$
  by (*smt has-greatest-fixpoint-def is-greatest-fixpoint-def greatest-fixpoint-unique greatest-postfixpoint-fixpoint greatest-fixpoint*)

lemma *pmu-isotone*: *has-least-prefixpoint f* $\wedge$ *has-least-prefixpoint g* $\wedge\ f \le\le g \longrightarrow p\mu\ f \le p\mu\ g$
  by (*smt is-least-prefixpoint-def least-prefixpoint lifted-less-eq-def order-trans*)

lemma *mu-isotone*: *has-least-prefixpoint f* $\wedge$ *has-least-prefixpoint g* $\wedge$ *isotone f* $\wedge$ *isotone g* $\wedge\ f \le\le g \longrightarrow \mu\ f \le \mu\ g$
  by (*metis pmu-isotone pmu-mu*)

lemma *pnu-isotone*: *has-greatest-postfixpoint* $f \wedge$ *has-greatest-postfixpoint* $g \wedge f \le\le g \longrightarrow p\nu\ f \le p\nu\ g$
  by (*smt is-greatest-postfixpoint-def lifted-less-eq-def order-trans greatest-postfixpoint*)

lemma *nu-isotone*: *has-greatest-postfixpoint* $f \wedge$ *has-greatest-postfixpoint* $g \wedge$ *isotone* $f \wedge$ *isotone* $g \wedge f \le\le g \longrightarrow \nu\ f \le \nu\ g$
  by (*metis pnu-isotone pnu-nu*)

lemma *mu-square*: *isotone* $f \wedge$ *has-least-fixpoint* $f \wedge$ *has-least-fixpoint* $(f \circ f) \longrightarrow \mu\ f = \mu\ (f \circ f)$
  by (*metis* (*no-types*, *hide-lams*) *antisym is-least-fixpoint-def isotone-def least-fixpoint-char least-fixpoint-unique o-apply*)

lemma *nu-square*: *isotone* $f \wedge$ *has-greatest-fixpoint* $f \wedge$ *has-greatest-fixpoint* $(f \circ f) \longrightarrow \nu\ f = \nu\ (f \circ f)$
  by (*metis* (*no-types*, *hide-lams*) *antisym is-greatest-fixpoint-def isotone-def greatest-fixpoint-char greatest-fixpoint-unique o-apply*)

lemma *mu-roll*: *isotone* $g \wedge$ *has-least-fixpoint* $(f \circ g) \wedge$ *has-least-fixpoint* $(g \circ f) \longrightarrow \mu\ (g \circ f) = g(\mu\ (f \circ g))$
  apply (*rule impI*)
  apply (*rule antisym*)
  apply (*smt is-least-fixpoint-def least-fixpoint o-apply*)
  apply (*smt is-least-fixpoint-def isotone-def least-fixpoint o-apply*)
  done

lemma *nu-roll*: *isotone* $g \wedge$ *has-greatest-fixpoint* $(f \circ g) \wedge$ *has-greatest-fixpoint* $(g \circ f) \longrightarrow \nu\ (g \circ f) = g(\nu\ (f \circ g))$
  apply (*rule impI*)
  apply (*rule antisym*)
  apply (*smt is-greatest-fixpoint-def greatest-fixpoint isotone-def o-apply*)
  apply (*smt is-greatest-fixpoint-def greatest-fixpoint o-apply*)
  done

lemma *mu-below-nu*: *has-least-fixpoint* $f \wedge$ *has-greatest-fixpoint* $f \longrightarrow \mu\ f \le \nu\ f$
  by (*metis is-greatest-fixpoint-def is-least-fixpoint-def least-fixpoint greatest-fixpoint*)

lemma *pmu-below-pnu-fix*: *has-fixpoint* $f \wedge$ *has-least-prefixpoint* $f \wedge$ *has-greatest-postfixpoint* $f \longrightarrow p\mu\ f \le p\nu\ f$
  by (*smt has-fixpoint-def is-fixpoint-def is-greatest-postfixpoint-def is-least-prefixpoint-def le-less order-trans least-prefixpoint greatest-postfixpoint*)

lemma *pmu-below-pnu-iso*: *isotone* $f \wedge$ *has-least-prefixpoint* $f \wedge$ *has-greatest-postfixpoint* $f \longrightarrow p\mu\ f \le p\nu\ f$
  by (*metis has-fixpoint-def is-fixpoint-def is-least-fixpoint-def least-prefixpoint-fixpoint pmu-below-pnu-fix*)

lemma *mu-fusion-1*: *galois* $l\ u \wedge$ *isotone* $h \wedge$ *has-least-prefixpoint* $g \wedge$ *has-least-fixpoint* $h \wedge l(g(u(\mu\ h))) \le h(l(u(\mu\ h))) \longrightarrow l(p\mu\ g) \le \mu\ h$
proof
  assume *1*: *galois* $l\ u \wedge$ *isotone* $h \wedge$ *has-least-prefixpoint* $g \wedge$ *has-least-fixpoint* $h \wedge l(g(u(\mu\ h))) \le h(l(u(\mu\ h)))$
  hence $l(g(u(\mu\ h))) \le \mu\ h$
    by (*metis galois-char least-fixpoint-same least-fixpoint-unique is-least-fixpoint-def isotone-def order-trans*)
  thus $l(p\mu\ g) \le \mu\ h$ **using** *1*
    by (*metis galois-def least-prefixpoint is-least-prefixpoint-def least-fixpoint-same least-fixpoint-unique*)
qed

lemma *mu-fusion-2*: *galois* $l\ u \wedge$ *isotone* $h \wedge$ *has-least-prefixpoint* $g \wedge$ *has-least-fixpoint* $h \wedge l \circ g \le\le h \circ l \longrightarrow l(p\mu\ g) \le \mu\ h$
  by (*metis lifted-less-eq-def mu-fusion-1 o-apply*)

lemma *mu-fusion-equal-1*: *galois* $l\ u \wedge$ *isotone* $g \wedge$ *isotone* $h \wedge$ *has-least-prefixpoint* $g \wedge$ *has-least-fixpoint* $h \wedge l(g(u(\mu\ h))) \le h(l(u(\mu\ h))) \wedge l(g(p\mu\ g)) = h(l(p\mu\ g)) \longrightarrow \mu\ h = l(p\mu\ g) \wedge \mu\ h = l(\mu\ g)$
  by (*metis antisym least-fixpoint least-prefixpoint-fixpoint is-least-fixpoint-def mu-fusion-1 pmu-mu*)

lemma *mu-fusion-equal-2*: *galois* $l\ u \wedge$ *isotone* $h \wedge$ *has-least-prefixpoint* $g \wedge$ *has-least-prefixpoint* $h \wedge l(g(u(\mu\ h))) \le h(l(u(\mu\ h))) \wedge l(g(p\mu\ g)) = h(l(p\mu\ g)) \longrightarrow p\mu\ h = l(p\mu\ g) \wedge \mu\ h = l(p\mu\ g)$
  by (*smt antisym galois-char least-fixpoint-char least-prefixpoint least-prefixpoint-fixpoint is-least-prefixpoint-def isotone-def mu-fusion-1*)

lemma *mu-fusion-equal-3*: *galois* $l\ u \wedge$ *isotone* $g \wedge$ *isotone* $h \wedge$ *has-least-prefixpoint* $g \wedge$ *has-least-fixpoint* $h \wedge l \circ g = h \circ l \longrightarrow \mu\ h = l(p\mu\ g) \wedge \mu\ h = l(\mu\ g)$
  by (*metis mu-fusion-equal-1 o-apply order-refl*)

lemma *mu-fusion-equal-4*: *galois* $l\ u \wedge$ *isotone* $h \wedge$ *has-least-prefixpoint* $g \wedge$ *has-least-prefixpoint* $h \wedge l \circ g = h \circ l \longrightarrow p\mu\ h = l(p\mu\ g) \wedge \mu\ h = l(p\mu\ g)$
  by (*metis mu-fusion-equal-2 o-apply order-refl*)

lemma *nu-fusion-1*: *galois* $l\ u \wedge$ *isotone* $h \wedge$ *has-greatest-postfixpoint* $g \wedge$ *has-greatest-fixpoint* $h \wedge h(u(l(\nu\ h))) \le u(g(l(\nu\ h))) \longrightarrow \nu\ h \le u(p\nu\ g)$

proof
assume *1*: *galois l u* $\wedge$ *isotone h* $\wedge$ *has-greatest-postfixpoint g* $\wedge$ *has-greatest-fixpoint h* $\wedge$ $h(u(l(\nu\ h))) \leq u(g(l(\nu\ h)))$
hence $\nu\ h \leq u(g(l(\nu\ h)))$ **using** *1*
by (*metis galois-char greatest-fixpoint-same greatest-fixpoint-unique is-greatest-fixpoint-def isotone-def order-trans*)
thus $\nu\ h \leq u(p\nu\ g)$ **using** *1*
by (*smt galois-def greatest-postfixpoint is-greatest-postfixpoint-def greatest-fixpoint-same greatest-fixpoint-unique*)
qed

lemma *nu-fusion-2*: *galois l u* $\wedge$ *isotone h* $\wedge$ *has-greatest-postfixpoint g* $\wedge$ *has-greatest-fixpoint h* $\wedge$ $h \circ u \leq\leq u \circ g \longrightarrow \nu\ h \leq u(p\nu\ g)$
by (*metis lifted-less-eq-def nu-fusion-1 o-apply*)

lemma *nu-fusion-equal-1*: *galois l u* $\wedge$ *isotone g* $\wedge$ *isotone h* $\wedge$ *has-greatest-postfixpoint g* $\wedge$ *has-greatest-fixpoint h* $\wedge$ $h(u(l(\nu\ h))) \leq u(g(l(\nu\ h))) \wedge h(u(p\nu\ g)) = u(g(p\nu\ g)) \longrightarrow \nu\ h = u(p\nu\ g) \wedge \nu\ h = u(\nu\ g)$
by (*metis antisym greatest-fixpoint greatest-postfixpoint-fixpoint is-greatest-fixpoint-def nu-fusion-1 pnu-nu*)

lemma *nu-fusion-equal-2*: *galois l u* $\wedge$ *isotone h* $\wedge$ *has-greatest-postfixpoint g* $\wedge$ *has-greatest-postfixpoint h* $\wedge$ $h(u(l(\nu\ h))) \leq u(g(l(\nu\ h))) \wedge h(u(p\nu\ g)) = u(g(p\nu\ g)) \longrightarrow p\nu\ h = u(p\nu\ g) \wedge \nu\ h = u(p\nu\ g)$
by (*smt antisym galois-char greatest-fixpoint-char greatest-postfixpoint greatest-postfixpoint-fixpoint is-greatest-postfixpoint-def isotone-def nu-fusion-1*)

lemma *nu-fusion-equal-3*: *galois l u* $\wedge$ *isotone g* $\wedge$ *isotone h* $\wedge$ *has-greatest-postfixpoint g* $\wedge$ *has-greatest-fixpoint h* $\wedge$ $h \circ u = u \circ g \longrightarrow \nu\ h = u(p\nu\ g) \wedge \nu\ h = u(\nu\ g)$
by (*metis nu-fusion-equal-1 o-apply order-refl*)

lemma *nu-fusion-equal-4*: *galois l u* $\wedge$ *isotone h* $\wedge$ *has-greatest-postfixpoint g* $\wedge$ *has-greatest-postfixpoint h* $\wedge$ $h \circ u = u \circ g \longrightarrow p\nu\ h = u(p\nu\ g) \wedge \nu\ h = u(p\nu\ g)$
by (*metis nu-fusion-equal-2 o-apply order-refl*)

lemma *mu-exchange-1*: *galois l u* $\wedge$ *isotone g* $\wedge$ *isotone h* $\wedge$ *has-least-prefixpoint* $(l \circ h)$ $\wedge$ *has-least-prefixpoint* $(h \circ g)$ $\wedge$ *has-least-fixpoint* $(g \circ h) \wedge l \circ h \circ g \leq\leq g \circ h \circ l \longrightarrow \mu(l \circ h) \leq \mu(g \circ h)$
proof
assume *1*: *galois l u* $\wedge$ *isotone g* $\wedge$ *isotone h* $\wedge$ *has-least-prefixpoint* $(l \circ h)$ $\wedge$ *has-least-prefixpoint* $(h \circ g)$ $\wedge$ *has-least-fixpoint* $(g \circ h) \wedge l \circ h \circ g \leq\leq g \circ h \circ l$
hence $l \circ (h \circ g) \leq\leq (g \circ h) \circ l$
by (*metis o-assoc*)
thus $\mu(l \circ h) \leq \mu(g \circ h)$ **using** *1*
by (*smt galois-char is-least-prefixpoint-def isotone-def least-fixpoint-char least-prefixpoint least-prefixpoint-fixpoint mu-fusion-2 mu-roll o-apply*)
qed

lemma *mu-exchange-2*: *galois l u* $\wedge$ *isotone g* $\wedge$ *isotone h* $\wedge$ *has-least-prefixpoint* $(l \circ h)$ $\wedge$ *has-least-prefixpoint* $(h \circ l)$ $\wedge$ *has-least-prefixpoint* $(h \circ g)$ $\wedge$ *has-least-fixpoint* $(g \circ h)$ $\wedge$ *has-least-fixpoint* $(h \circ g) \wedge l \circ h \circ g \leq\leq g \circ h \circ l \longrightarrow \mu(h \circ l) \leq \mu(h \circ g)$
by (*smt galois-char isotone-def least-fixpoint-char least-prefixpoint-fixpoint mu-exchange-1 mu-roll o-apply*)

lemma *mu-exchange-equal*: *galois l u* $\wedge$ *galois k t* $\wedge$ *isotone h* $\wedge$ *has-least-prefixpoint* $(l \circ h)$ $\wedge$ *has-least-prefixpoint* $(h \circ l)$ $\wedge$ *has-least-prefixpoint* $(k \circ h)$ $\wedge$ *has-least-prefixpoint* $(h \circ k) \wedge l \circ h \circ k = k \circ h \circ l \longrightarrow \mu(l \circ h) = \mu(k \circ h) \wedge \mu(h \circ l) = \mu(h \circ k)$
by (*smt antisym galois-char isotone-def least-fixpoint-char least-prefixpoint-fixpoint lifted-reflexive mu-exchange-1 mu-exchange-2 o-apply*)

lemma *nu-exchange-1*: *galois l u* $\wedge$ *isotone g* $\wedge$ *isotone h* $\wedge$ *has-greatest-postfixpoint* $(u \circ h)$ $\wedge$ *has-greatest-postfixpoint* $(h \circ g)$ $\wedge$ *has-greatest-fixpoint* $(g \circ h) \wedge g \circ h \circ u \leq\leq u \circ h \circ g \longrightarrow \nu(g \circ h) \leq \nu(u \circ h)$
proof
assume *1*: *galois l u* $\wedge$ *isotone g* $\wedge$ *isotone h* $\wedge$ *has-greatest-postfixpoint* $(u \circ h)$ $\wedge$ *has-greatest-postfixpoint* $(h \circ g)$ $\wedge$ *has-greatest-fixpoint* $(g \circ h) \wedge g \circ h \circ u \leq\leq u \circ h \circ g$
hence $(g \circ h) \circ u \leq\leq u \circ (h \circ g)$
by (*metis o-assoc*)
thus $\nu(g \circ h) \leq \nu(u \circ h)$ **using** *1*
by (*smt galois-char is-greatest-postfixpoint-def isotone-def greatest-fixpoint-char greatest-postfixpoint greatest-postfixpoint-fixpoint nu-fusion-2 nu-roll o-apply*)
qed

lemma *nu-exchange-2*: *galois l u* $\wedge$ *isotone g* $\wedge$ *isotone h* $\wedge$ *has-greatest-postfixpoint* $(u \circ h)$ $\wedge$ *has-greatest-postfixpoint* $(h \circ u)$ $\wedge$ *has-greatest-postfixpoint* $(h \circ g)$ $\wedge$ *has-greatest-fixpoint* $(g \circ h)$ $\wedge$ *has-greatest-fixpoint* $(h \circ g) \wedge g \circ h \circ u \leq\leq u \circ h \circ g \longrightarrow \nu(h \circ g) \leq \nu(h \circ u)$
by (*smt galois-char isotone-def greatest-fixpoint-char greatest-postfixpoint-fixpoint nu-exchange-1 nu-roll o-apply*)

lemma *nu-exchange-equal*: *galois l u* $\wedge$ *galois k t* $\wedge$ *isotone h* $\wedge$ *has-greatest-postfixpoint* $(u \circ h)$ $\wedge$ *has-greatest-postfixpoint* $(h \circ u)$ $\wedge$ *has-greatest-postfixpoint* $(t \circ h)$ $\wedge$ *has-greatest-postfixpoint* $(h \circ t)$ $\wedge$ $u \circ h \circ t = t \circ h \circ u \longrightarrow \nu(u \circ h) = \nu(t \circ h) \wedge \nu(h \circ u) = \nu(h \circ t)$
by (*smt antisym galois-char isotone-def greatest-fixpoint-char greatest-postfixpoint-fixpoint lifted-reflexive nu-exchange-1 nu-exchange-2 o-apply*)

lemma *mu-commute-fixpoint-1*: *isotone f* $\wedge$ *has-least-fixpoint* $(f \circ g) \wedge f \circ g = g \circ f \longrightarrow$ *is-fixpoint f* $(\mu(f \circ g))$
by (*metis is-fixpoint-def mu-roll*)

lemma *mu-commute-fixpoint-2*: *isotone g* $\wedge$ *has-least-fixpoint* $(f \circ g) \wedge f \circ g = g \circ f \longrightarrow$ *is-fixpoint g* $(\mu(f \circ g))$
by (*metis is-fixpoint-def mu-roll*)

lemma *mu-commute-least-fixpoint*: *isotone f* $\wedge$ *isotone g* $\wedge$ *has-least-fixpoint f* $\wedge$ *has-least-fixpoint g* $\wedge$ *has-least-fixpoint* $(f \circ g) \wedge f \circ g = g \circ f \longrightarrow (\mu(f \circ g) = \mu\ f \longrightarrow \mu\ g \leq \mu\ f)$
by (*metis is-least-fixpoint-def least-fixpoint-same least-fixpoint-unique mu-roll*)

lemma *nu-commute-fixpoint-1*: *isotone f* $\wedge$ *has-greatest-fixpoint* $(f \circ g) \wedge f \circ g = g \circ f \longrightarrow$ *is-fixpoint f* $(\nu(f \circ g))$
by (*metis is-fixpoint-def nu-roll*)

lemma *nu-commute-fixpoint-2*: *isotone g* $\wedge$ *has-greatest-fixpoint* $(f \circ g) \wedge f \circ g = g \circ f \longrightarrow$ *is-fixpoint g* $(\nu(f \circ g))$
by (*metis is-fixpoint-def nu-roll*)

lemma *nu-commute-greatest-fixpoint*: *isotone f* $\wedge$ *isotone g* $\wedge$ *has-greatest-fixpoint f* $\wedge$ *has-greatest-fixpoint g* $\wedge$ *has-greatest-fixpoint* $(f \circ g) \wedge f \circ g = g \circ f \longrightarrow (\nu(f \circ g) = \nu\ f \longrightarrow \nu\ f \leq \nu\ g)$
by (*smt is-greatest-fixpoint-def greatest-fixpoint-same greatest-fixpoint-unique nu-roll*)

lemma *mu-diagonal-1*: *isotone* $(\lambda x\ .\ f\ x\ x) \wedge (\forall x\ .$ *isotone* $(\lambda y\ .\ f\ x\ y)) \wedge$ *isotone* $(\lambda x\ .\ \mu(\lambda y\ .\ f\ x\ y)) \wedge (\forall x\ .$ *has-least-fixpoint* $(\lambda y\ .\ f\ x\ y)) \wedge$ *has-least-prefixpoint* $(\lambda x\ .\ \mu(\lambda y\ .\ f\ x\ y)) \longrightarrow \mu(\lambda x\ .\ f\ x\ x) = \mu(\lambda x\ .\ \mu(\lambda y\ .\ f\ x\ y))$
by (*smt is-least-fixpoint-def is-least-prefixpoint-def least-fixpoint-same least-fixpoint-unique least-prefixpoint least-prefixpoint-fixpoint*)

lemma *mu-diagonal-2*: $(\forall x\ .$ *isotone* $(\lambda y\ .\ f\ x\ y) \wedge$ *isotone* $(\lambda y\ .\ f\ y\ x) \wedge$ *has-least-prefixpoint* $(\lambda y\ .\ f\ x\ y)) \wedge$ *has-least-prefixpoint* $(\lambda x\ .\ \mu(\lambda y\ .\ f\ x\ y)) \longrightarrow \mu(\lambda x\ .\ f\ x\ x) = \mu(\lambda x\ .\ \mu(\lambda y\ .\ f\ x\ y))$
proof
assume *1*: $(\forall x\ .$ *isotone* $(\lambda y\ .\ f\ x\ y) \wedge$ *isotone* $(\lambda y\ .\ f\ y\ x) \wedge$ *has-least-prefixpoint* $(\lambda y\ .\ f\ x\ y)) \wedge$ *has-least-prefixpoint* $(\lambda x\ .\ \mu(\lambda y\ .\ f\ x\ y))$
hence *isotone* $(\lambda x\ .\ \mu(\lambda y\ .\ f\ x\ y))$
by (*smt isotone-def lifted-less-eq-def mu-isotone*)
thus $\mu(\lambda x\ .\ f\ x\ x) = \mu(\lambda x\ .\ \mu(\lambda y\ .\ f\ x\ y))$ **using** *1*
by (*smt is-least-fixpoint-def is-least-prefixpoint-def least-fixpoint-same least-prefixpoint least-prefixpoint-fixpoint*)
qed

lemma *nu-diagonal-1*: *isotone* $(\lambda x\ .\ f\ x\ x) \wedge (\forall x\ .$ *isotone* $(\lambda y\ .\ f\ x\ y)) \wedge$ *isotone* $(\lambda x\ .\ \nu(\lambda y\ .\ f\ x\ y)) \wedge (\forall x\ .$ *has-greatest-fixpoint* $(\lambda y\ .\ f\ x\ y)) \wedge$ *has-greatest-postfixpoint* $(\lambda x\ .\ \nu(\lambda y\ .\ f\ x\ y)) \longrightarrow \nu(\lambda x\ .\ f\ x\ x) = \nu(\lambda x\ .\ \nu(\lambda y\ .\ f\ x\ y))$
by (*smt is-greatest-fixpoint-def is-greatest-postfixpoint-def greatest-fixpoint-same greatest-fixpoint-unique greatest-postfixpoint greatest-postfixpoint-fixpoint*)

lemma *nu-diagonal-2*: $(\forall x\ .$ *isotone* $(\lambda y\ .\ f\ x\ y) \wedge$ *isotone* $(\lambda y\ .\ f\ y\ x) \wedge$ *has-greatest-postfixpoint* $(\lambda y\ .\ f\ x\ y)) \wedge$ *has-greatest-postfixpoint* $(\lambda x\ .\ \nu(\lambda y\ .\ f\ x\ y)) \longrightarrow \nu(\lambda x\ .\ f\ x\ x) = \nu(\lambda x\ .\ \nu(\lambda y\ .\ f\ x\ y))$
proof
assume *1*: $(\forall x\ .$ *isotone* $(\lambda y\ .\ f\ x\ y) \wedge$ *isotone* $(\lambda y\ .\ f\ y\ x) \wedge$ *has-greatest-postfixpoint* $(\lambda y\ .\ f\ x\ y)) \wedge$ *has-greatest-postfixpoint* $(\lambda x\ .\ \nu(\lambda y\ .\ f\ x\ y))$
hence *isotone* $(\lambda x\ .\ \nu(\lambda y\ .\ f\ x\ y))$
by (*smt isotone-def lifted-less-eq-def nu-isotone*)
thus $\nu(\lambda x\ .\ f\ x\ x) = \nu(\lambda x\ .\ \nu(\lambda y\ .\ f\ x\ y))$ **using** *1*
by (*smt greatest-fixpoint-same greatest-postfixpoint greatest-postfixpoint-fixpoint is-greatest-fixpoint-def is-greatest-postfixpoint-def*)
qed

end

end

3 Lattice

theory *Lattice*

imports *Base*

begin

class *join-semilattice* = *plus* + *ord* +
assumes *add-associative*: $(x + y) + z = x + (y + z)$
assumes *add-commutative*: $x + y = y + x$
assumes *add-idempotent*: $x + x = x$
assumes *less-eq-def*: $x \le y \longleftrightarrow x + y = y$
assumes *less-def*: $x < y \longleftrightarrow x \le y \wedge \neg (y \le x)$

begin

subclass *order*
apply *unfold-locales*
apply (*metis less-def*)
apply (*metis add-idempotent less-eq-def*)
apply (*metis add-associative less-eq-def*)
apply (*metis add-commutative less-eq-def*)
done

lemma *add-left-isotone*: $x \le y \longrightarrow x + z \le y + z$
by (*smt add-associative add-commutative add-idempotent less-eq-def*)

lemma *add-right-isotone*: $x \le y \longrightarrow z + x \le z + y$
by (*metis add-commutative add-left-isotone*)

lemma *add-isotone*: $w \le y \wedge x \le z \longrightarrow w + x \le y + z$
by (*smt add-associative add-commutative less-eq-def*)

lemma *add-left-upper-bound*: $x \le x + y$
by (*metis add-associative add-idempotent less-eq-def*)

lemma *add-right-upper-bound*: $y \le x + y$
by (*metis add-commutative add-left-upper-bound*)

lemma *add-least-upper-bound*: $x \le z \wedge y \le z \longleftrightarrow x + y \le z$
by (*smt add-associative add-commutative add-left-upper-bound less-eq-def*)

lemma *add-left-divisibility*: $x \le y \longleftrightarrow (\exists z \,.\, x + z = y)$
by (*metis add-left-upper-bound less-eq-def*)

lemma *add-right-divisibility*: $x \le y \longleftrightarrow (\exists z \,.\, z + x = y)$
by (*metis add-commutative add-left-divisibility*)

lemma *add-same-context*:$x \le y + z \wedge y \le x + z \longrightarrow x + z = y + z$
by (*smt add-associative add-commutative less-eq-def*)

lemma *add-relative-same-increasing*: $x \le y \wedge x + z = x + w \longrightarrow y + z = y + w$
by (*smt add-associative add-right-divisibility*)

lemma *ascending-chain-left-add*: *ascending-chain f* $\longrightarrow$ *ascending-chain* $(\lambda n \,.\, x + f\, n)$
by (*metis ascending-chain-def add-right-isotone*)

lemma *ascending-chain-right-add*: *ascending-chain f* $\longrightarrow$ *ascending-chain* $(\lambda n \,.\, f\, n + x)$
by (*metis ascending-chain-def add-left-isotone*)

lemma *descending-chain-left-add*: *descending-chain f* $\longrightarrow$ *descending-chain* $(\lambda n \,.\, x + f\, n)$
by (*metis descending-chain-def add-right-isotone*)

lemma *descending-chain-right-add*: *descending-chain f* $\longrightarrow$ *descending-chain* $(\lambda n \,.\, f\, n + x)$
by (*metis descending-chain-def add-left-isotone*)

primrec *pSum0* :: $(nat \Rightarrow {'a}) \Rightarrow nat \Rightarrow {'a}$
where *pSum0 f 0* = *f 0*

```
    | pSum0 f (Suc m) = pSum0 f m + f m

lemma pSum0-below: (∀ i . f i ≤ x) ⟶ pSum0 f m ≤ x
  apply (induct m)
  apply (metis pSum0.simps(1))
  by (metis add-least-upper-bound pSum0.simps(2))

end

class bounded-join-semilattice = join-semilattice + zero +
  assumes add-left-zero: 0 + x = x

begin

lemma add-right-zero: x + 0 = x
  by (metis add-commutative add-left-zero)

lemma zero-least: 0 ≤ x
  by (metis add-left-upper-bound add-left-zero)

end

class meet =
  fixes meet :: 'a ⇒ 'a ⇒ 'a (infixl ⌢ 65)

class meet-semilattice = meet + ord +
  assumes meet-associative: (x ⌢ y) ⌢ z = x ⌢ (y ⌢ z)
  assumes meet-commutative: x ⌢ y = y ⌢ x
  assumes meet-idempotent: x ⌢ x = x
  assumes meet-less-eq-def: x ≤ y ⟷ x ⌢ y = x
  assumes meet-less-def: x < y ⟷ x ≤ y ∧ ¬ (y ≤ x)

sublocale meet-semilattice < meet!: join-semilattice where plus = meet and less-eq = (λx y . y ≤ x) and less = (λx y . y
< x)
  apply unfold-locales
  apply (rule meet-associative)
  apply (rule meet-commutative)
  apply (rule meet-idempotent)
  apply (metis meet-commutative meet-less-eq-def)
  apply (metis meet-less-def)
  done

class T =
  fixes T :: 'a (⊤)

class bounded-meet-semilattice = meet-semilattice + T +
  assumes meet-left-top: T ⌢ x = x

sublocale bounded-meet-semilattice < meet!: bounded-join-semilattice where plus = meet and less-eq = (λx y . y ≤ x) and
less = (λx y . y < x) and zero = T
  apply unfold-locales
  apply (rule meet-left-top)
  done

class bounded-distributive-lattice = bounded-join-semilattice + bounded-meet-semilattice +
  assumes meet-left-dist-add: x ⌢ (y + z) = (x ⌢ y) + (x ⌢ z)
  assumes add-left-dist-meet: x + (y ⌢ z) = (x + y) ⌢ (x + z)
  assumes meet-absorb: x ⌢ (x + y) = x
  assumes add-absorb: x + (x ⌢ y) = x

begin

lemma meet-left-zero: 0 ⌢ x = 0
  by (metis add-absorb add-left-zero)

lemma meet-right-zero: x ⌢ 0 = 0
  by (metis meet-commutative meet-left-zero)

lemma add-left-top-1: T + x = T
```

by (*metis add-absorb meet-left-top*)

lemma *add-right-top-1*: $x + T = T$
by (*metis add-commutative add-left-top-1*)

lemma *meet-same-context*:$x \leq y \frown z \wedge y \leq x \frown z \longrightarrow x \frown z = y \frown z$
by (*metis eq-iff meet.add-least-upper-bound*)

lemma *relative-equality*: $x + z = y + z \wedge x \frown z = y \frown z \longrightarrow x = y$
by (*metis add-absorb add-commutative add-left-dist-meet*)

end

end

4 Semiring

theory *Semiring*

imports *Fixpoint Lattice*

begin

class *monoid* = *mult* + *one* +
  assumes *mult-associative*: $(x ; y) ; z = x ; (y ; z)$
  assumes *mult-left-one-1*: $1 ; x = x$
  assumes *mult-right-one*: $x ; 1 = x$

class *non-associative-left-semiring* = *bounded-join-semilattice* + *mult* + *one* +
  assumes *mult-left-sub-dist-add*: $x ; y + x ; z \leq x ; (y + z)$
  assumes *mult-right-dist-add*: $(x + y) ; z = x ; z + y ; z$
  assumes *mult-left-zero*: $0 ; x = 0$
  assumes *mult-left-one*: $1 ; x = x$
  assumes *mult-sub-right-one*: $x \leq x ; 1$

begin

lemma *mult-left-isotone*: $x \leq y \longrightarrow x ; z \leq y ; z$
  by (*metis less-eq-def mult-right-dist-add*)

lemma *mult-right-isotone*: $x \leq y \longrightarrow z ; x \leq z ; y$
  by (*metis add-least-upper-bound less-eq-def mult-left-sub-dist-add*)

lemma *mult-isotone*: $w \leq y \land x \leq z \longrightarrow w ; x \leq y ; z$
  by (*smt mult-left-isotone mult-right-isotone order-trans*)

lemma *affine-isotone*: *isotone* $(\lambda x . y ; x + z)$
  by (*smt add-commutative add-right-isotone isotone-def mult-right-isotone*)

lemma *mult-left-sub-dist-add-left*: $x ; y \leq x ; (y + z)$
  by (*metis add-left-upper-bound mult-right-isotone*)

lemma *mult-left-sub-dist-add-right*: $x ; z \leq x ; (y + z)$
  by (*metis add-right-upper-bound mult-right-isotone*)

lemma *mult-right-sub-dist-add-left*: $x ; z \leq (x + y) ; z$
  by (*metis add-left-upper-bound mult-right-dist-add*)

lemma *mult-right-sub-dist-add-right*: $y ; z \leq (x + y) ; z$
  by (*metis add-right-upper-bound mult-right-dist-add*)

lemma *case-split-left*: $1 \leq w + z \land w ; x \leq y \land z ; x \leq y \longrightarrow x \leq y$
  by (*smt add-associative add-commutative less-eq-def mult-left-one mult-right-dist-add order-refl*)

lemma *case-split-left-equal*: $w + z = 1 \land w ; x = w ; y \land z ; x = z ; y \longrightarrow x = y$
  by (*metis mult-left-one mult-right-dist-add*)

lemma *ascending-chain-left-mult*: *ascending-chain* $f \longrightarrow$ *ascending-chain* $(\lambda n . x ; f\ n)$
  by (*metis ascending-chain-def mult-right-isotone*)

lemma *ascending-chain-right-mult*: *ascending-chain* $f \longrightarrow$ *ascending-chain* $(\lambda n . f\ n ; x)$
  by (*metis ascending-chain-def mult-left-isotone*)

lemma *descending-chain-left-mult*: *descending-chain* $f \longrightarrow$ *descending-chain* $(\lambda n . x ; f\ n)$
  by (*metis descending-chain-def mult-right-isotone*)

lemma *descending-chain-right-mult*: *descending-chain* $f \longrightarrow$ *descending-chain* $(\lambda n . f\ n ; x)$
  by (*metis descending-chain-def mult-left-isotone*)

— Some results about transitive closures in this class and the next two classes are taken from a joint paper with Rudolf Berghammer.

abbreviation *Lf* :: $'a \Rightarrow ('a \Rightarrow 'a)$ **where** *Lf* $y \equiv (\lambda x . 1 + x ; y)$
abbreviation *Rf* :: $'a \Rightarrow ('a \Rightarrow 'a)$ **where** *Rf* $y \equiv (\lambda x . 1 + y ; x)$

abbreviation *Sf* :: *'a* ⇒ (*'a* ⇒ *'a*) **where** *Sf y* ≡ (λ *x* . *1* + *y* + *x* ; *x*)

abbreviation *lstar* :: *'a* ⇒ *'a* **where** *lstar y* ≡ *pμ* (*Lf y*)
abbreviation *rstar* :: *'a* ⇒ *'a* **where** *rstar y* ≡ *pμ* (*Rf y*)
abbreviation *sstar* :: *'a* ⇒ *'a* **where** *sstar y* ≡ *pμ* (*Sf y*)

lemma *lstar-rec-isotone*: *isotone* (*Lf y*)
  by (*smt2 add-isotone add-right-divisibility isotone-def mult-right-sub-dist-add-right order.refl*)

lemma *rstar-rec-isotone*: *isotone* (*Rf y*)
  by (*metis add-isotone isotone-def less-eq-def mult-left-sub-dist-add-left order-refl*)

lemma *sstar-rec-isotone*: *isotone* (*Sf y*)
  by (*simp add*: *add-right-isotone isotone-def mult-isotone*)

lemma *lstar-fixpoint*: *has-least-prefixpoint* (*Lf y*) ⟶ *lstar y* = *μ* (*Lf y*)
  by (*metis pmu-mu lstar-rec-isotone*)

lemma *rstar-fixpoint*: *has-least-prefixpoint* (*Rf y*) ⟶ *rstar y* = *μ* (*Rf y*)
  by (*metis pmu-mu rstar-rec-isotone*)

lemma *sstar-fixpoint*: *has-least-prefixpoint* (*Sf y*) ⟶ *sstar y* = *μ* (*Sf y*)
  by (*metis pmu-mu sstar-rec-isotone*)

lemma *sstar-increasing*: *has-least-prefixpoint* (*Sf y*) ⟶ *y* ≤ *sstar y*
  by (*metis add-least-upper-bound add-left-upper-bound order.trans pmu-unfold*)

lemma *rstar-below-sstar*: *has-least-prefixpoint* (*Rf y*) ∧ *has-least-prefixpoint* (*Sf y*) ⟶ *rstar y* ≤ *sstar y*
proof
  assume *1*: *has-least-prefixpoint* (*Rf y*) ∧ *has-least-prefixpoint* (*Sf y*)
  hence *Rf y* (*sstar y*) ≤ *Sf y* (*sstar y*)
    by (*smt2 add-isotone add-left-upper-bound add-right-upper-bound dual-order.trans mult-left-isotone pmu-unfold*)
  also have ... ≤ *sstar y* **using** *1*
    by (*metis* (*erased*, *lifting*) *pmu-unfold*)
  finally show *rstar y* ≤ *sstar y* **using** *1*
    by (*metis* (*erased*, *lifting*) *is-least-prefixpoint-def least-prefixpoint*)
qed

end

class *pre-left-semiring* = *non-associative-left-semiring* +
  assumes *mult-semi-associative*: (*x* ; *y*) ; *z* ≤ *x* ; (*y* ; *z*)

begin

lemma *mult-one-associative*: *x* ; *1* ; *y* = *x* ; *y*
  by (*metis dual-order.antisym mult-left-isotone mult-left-one mult-semi-associative mult-sub-right-one*)

lemma *mult-sup-associative-one*: (*x* ; (*y* ; *1*)) ; *z* ≤ *x* ; (*y* ; *z*)
  by (*metis mult-semi-associative mult-one-associative*)

lemma *rstar-increasing*: *has-least-prefixpoint* (*Rf y*) ⟶ *y* ≤ *rstar y*
proof
  assume *has-least-prefixpoint* (*Rf y*)
  hence *Rf y* (*rstar y*) ≤ *rstar y*
    by (*metis pmu-unfold*)
  thus *y* ≤ *rstar y*
    by (*metis add-least-upper-bound mult-right-isotone mult-sub-right-one order.trans*)
qed

end

class *residuated-pre-left-semiring* = *pre-left-semiring* + *inverse* +
  assumes *lres-galois*: *x* ; *y* ≤ *z* ⟷ *x* ≤ *z* / *y*

begin

— Theorem 32.2

```
lemma lres-left-isotone: x ≤ y ⟶ x / z ≤ y / z
  by (metis lres-galois order.refl order.trans)
```

— Theorem 32.2

```
lemma lres-right-antitone: x ≤ y ⟶ z / y ≤ z / x
  by (metis lres-galois mult-right-isotone order.refl order-trans)

lemma lres-inverse: (x / y) ; y ≤ x
  by (metis lres-galois order-refl)

lemma lres-one: x / 1 ≤ x
  by (metis dual-order.trans mult-sub-right-one lres-inverse)

lemma lres-mult-sub-lres-lres: x / (z ; y) ≤ (x / y) / z
  by (metis lres-galois lres-inverse mult-semi-associative order.trans)
```

— Theorem 32.4

```
lemma mult-lres-sub-assoc: x ; (y / z) ≤ (x ; y) / z
  by (metis (full-types) lres-galois lres-inverse mult-right-isotone mult-semi-associative order-trans)

lemma lstar-below-rstar: has-least-prefixpoint (Lf y) ∧ has-least-prefixpoint (Rf y) ⟶ lstar y ≤ rstar y
proof
  assume 1: has-least-prefixpoint (Lf y) ∧ has-least-prefixpoint (Rf y)
  have y ; (rstar y / y) ; y ≤ y ; rstar y
    by (metis mult-right-isotone mult-semi-associative order-trans lres-inverse)
  also have ... ≤ rstar y using 1
    by (metis add-least-upper-bound pmu-unfold)
  finally have y ; (rstar y / y) ≤ rstar y / y
    by (metis lres-galois)
  hence Rf y (rstar y / y) ≤ rstar y / y using 1
    by (metis add-least-upper-bound lres-galois mult-left-one rstar-increasing)
  hence rstar y ≤ rstar y / y using 1
    by (metis is-least-prefixpoint-def least-prefixpoint)
  hence Lf y (rstar y) ≤ rstar y using 1
    by (metis add-least-upper-bound lres-galois pmu-unfold)
  thus lstar y ≤ rstar y using 1
    by (metis (erased, lifting) is-least-prefixpoint-def least-prefixpoint)
qed
```

— The next proof follows an argument of Rudolf Berghammer; see Satz 10.1.5 in his 2012 book.

```
lemma rstar-sstar: has-least-prefixpoint (Rf y) ∧ has-least-prefixpoint (Sf y) ⟶ rstar y = sstar y
proof
  assume 1: has-least-prefixpoint (Rf y) ∧ has-least-prefixpoint (Sf y)
  have Rf y (rstar y / rstar y) ; rstar y ≤ rstar y + y ; ((rstar y / rstar y) ; rstar y)
    by (metis add-isotone mult-left-one mult-right-dist-add mult-semi-associative)
  also have ... ≤ rstar y + y ; rstar y
    by (metis add-right-isotone mult-right-isotone lres-inverse)
  also have ... ≤ rstar y using 1
    by (metis (full-types) add-least-upper-bound order-refl pmu-unfold)
  finally have Rf y (rstar y / rstar y) ≤ rstar y / rstar y
    by (metis lres-galois)
  hence rstar y ; rstar y ≤ rstar y using 1
    by (metis (erased, lifting) is-least-prefixpoint-def least-prefixpoint lres-galois)
  hence y + rstar y ; rstar y ≤ rstar y using 1
    by (metis add-least-upper-bound rstar-increasing)
  hence Sf y (rstar y) ≤ rstar y using 1
    by (metis (full-types) add-least-upper-bound pmu-unfold)
  hence sstar y ≤ rstar y using 1
    by (metis (erased, lifting) is-least-prefixpoint-def least-prefixpoint)
  thus rstar y = sstar y using 1
    by (metis antisym rstar-below-sstar)
qed

end

class idempotent-left-semiring = non-associative-left-semiring + monoid
```

begin

subclass *pre-left-semiring*
  apply *unfold-locales*
  apply (*metis mult-associative order-refl*)
  done

lemma *zero-right-mult-decreasing*: $x\ ;\ 0 \le x$
  by (*metis add-right-zero mult-left-sub-dist-add-right mult-right-one*)

lemma *test-preserves-equation*: $p \le p\ ;\ p \wedge p \le 1 \longrightarrow (p\ ;\ x \le x\ ;\ p \longleftrightarrow p\ ;\ x = p\ ;\ x\ ;\ p)$
  apply *rule*
  apply *rule*
  apply (*rule antisym*)
  apply (*smt antisym mult-associative mult-right-isotone mult-right-one*)
  apply (*metis mult-right-isotone mult-right-one*)
  apply (*metis mult-left-isotone mult-left-one*)
  done

end

class *idempotent-left-zero-semiring* = *idempotent-left-semiring* +
  assumes *mult-left-dist-add*: $x\ ;\ (y + z) = x\ ;\ y + x\ ;\ z$

begin

lemma *case-split-right*: $1 \le w + z \wedge x\ ;\ w \le y \wedge x\ ;\ z \le y \longrightarrow x \le y$
  by (*smt add-associative add-commutative less-eq-def mult-left-dist-add mult-right-one*)

lemma *case-split-right-equal*: $w + z = 1 \wedge x\ ;\ w = y\ ;\ w \wedge x\ ;\ z = y\ ;\ z \longrightarrow x = y$
  by (*metis mult-left-dist-add mult-right-one*)

end

class *idempotent-semiring* = *idempotent-left-zero-semiring* +
  assumes *mult-right-zero*: $x\ ;\ 0 = 0$

class *bounded-pre-left-semiring* = *pre-left-semiring* + T +
  assumes *add-right-top*: $x + T = T$

begin

lemma *add-left-top*: $T + x = T$
  by (*metis add-commutative add-right-top*)

lemma *top-greatest*: $x \le T$
  by (*metis add-left-top add-right-upper-bound*)

lemma *top-left-mult-increasing*: $x \le T\ ;\ x$
  by (*metis mult-left-isotone mult-left-one top-greatest*)

lemma *top-right-mult-increasing*: $x \le x\ ;\ T$
  by (*metis order-trans mult-right-isotone mult-sub-right-one top-greatest*)

lemma *top-mult-top*: $T\ ;\ T = T$
  by (*metis add-right-divisibility add-right-top top-right-mult-increasing*)

definition *vector* :: $'a \Rightarrow bool$
  where *vector* $x \longleftrightarrow x = x\ ;\ T$

lemma *vector-zero*: *vector* 0
  by (*metis mult-left-zero vector-def*)

lemma *vector-top*: *vector* T
  by (*metis top-mult-top vector-def*)

lemma *vector-add-closed*: *vector* $x \wedge$ *vector* $y \longrightarrow$ *vector* $(x + y)$
  by (*metis mult-right-dist-add vector-def*)

lemma *vector-left-mult-closed*: *vector* $y \longrightarrow$ *vector* $(x\ ;\ y)$
  by (*metis antisym mult-semi-associative top-right-mult-increasing vector-def*)

end

class *bounded-residuated-pre-left-semiring* = *residuated-pre-left-semiring* + *bounded-pre-left-semiring*

begin

— Theorem 32.8

lemma *lres-top-decreasing*: $x\ /\ T \le x$
  by (*metis lres-one lres-right-antitone order-trans top-greatest*)

— Theorem 32.9

lemma *top-lres-absorb*: $T\ /\ x = T$
  by (*metis eq-iff lres-galois top-greatest*)

end

class *bounded-idempotent-left-semiring* = *bounded-pre-left-semiring* + *idempotent-left-semiring*

class *bounded-idempotent-left-zero-semiring* = *bounded-idempotent-left-semiring* + *idempotent-left-zero-semiring*

class *bounded-idempotent-semiring* = *bounded-idempotent-left-zero-semiring* + *idempotent-semiring*

end

5 Itering

theory *Itering*

imports *Semiring*

begin

class *circ* =
  fixes *circ* :: *'a* ⇒ *'a* (-$^\circ$ [*100*] *100*)

class *left-conway-semiring* = *idempotent-left-semiring* + *circ* +
  assumes *circ-left-unfold*: $1 + x ; x^\circ = x^\circ$
  assumes *circ-left-slide*: $(x ; y)^\circ ; x \le x ; (y ; x)^\circ$
  assumes *circ-add-1*: $(x + y)^\circ = x^\circ ; (y ; x^\circ)^\circ$

begin

— Theorem 14.19

lemma *circ-mult-sub*: $1 + x ; (y ; x)^\circ ; y \le (x ; y)^\circ$
  by (*metis add-right-isotone circ-left-slide circ-left-unfold mult-associative mult-right-isotone*)

— Theorem 14.8

lemma *circ-right-unfold-sub*: $1 + x^\circ ; x \le x^\circ$
  by (*metis circ-mult-sub mult-left-one mult-right-one*)

— Theorem 1.1 and Theorem 14.1

lemma *circ-zero*: $0^\circ = 1$
  by (*metis add-right-zero circ-left-unfold mult-left-zero*)

— Theorem 1.4 and Theorem 1.13 and Theorem 14.4 and Theorem 14.13

lemma *circ-increasing*: $x \le x^\circ$
  by (*metis add-right-upper-bound circ-left-unfold circ-right-unfold-sub mult-left-one mult-right-sub-dist-add-left order-trans*)

— Theorem 1.3 and Theorem 14.3

lemma *circ-reflexive*: $1 \le x^\circ$
  by (*metis add-left-divisibility circ-left-unfold*)

— Theorem 1.5

lemma *circ-mult-increasing*: $x \le x ; x^\circ$
  by (*metis circ-reflexive mult-right-isotone mult-right-one*)

— Theorem 14.5

lemma *circ-mult-increasing-2*: $x \le x^\circ ; x$
  by (*metis circ-reflexive mult-left-isotone mult-left-one-1*)

— Theorem 1.11 and Theorem 14.11

lemma *circ-transitive-equal*: $x^\circ ; x^\circ = x^\circ$
  by (*metis add-idempotent circ-add-1 circ-left-unfold mult-associative*)

— Theorem 1.17 and Theorem 14.17

lemma *circ-circ-circ*: $x^{\circ\circ\circ} = x^{\circ\circ}$
  by (*metis add-idempotent circ-add-1 circ-increasing circ-transitive-equal less-eq-def*)

— Theorem 1.18 and Theorem 14.18

lemma *circ-one*: $1^\circ = 1^{\circ\circ}$
  by (*metis circ-circ-circ circ-zero*)

— Theorem 14.21

lemma *circ-add-sub*: $(x^\circ \ ; \ y)^\circ \ ; \ x^\circ \leq (x + y)^\circ$
 by (*metis circ-add-1 circ-left-slide*)

lemma *circ-plus-one*: $x^\circ = \mathit{1} + x^\circ$
 by (*metis less-eq-def circ-reflexive*)

— Theorem 1.12 and Theorem 14.12

lemma *circ-rtc-2*: $\mathit{1} + x + x^\circ \ ; \ x^\circ = x^\circ$
 by (*metis add-associative circ-increasing circ-plus-one circ-transitive-equal less-eq-def*)

— Theorem 1.2 and Theorem 14.2

lemma *mult-zero-circ*: $(x \ ; \ \mathit{0})^\circ = \mathit{1} + x \ ; \ \mathit{0}$
 by (*metis circ-left-unfold mult-associative mult-left-zero*)

lemma *mult-zero-add-circ*: $(x + y \ ; \ \mathit{0})^\circ = x^\circ \ ; \ (y \ ; \ \mathit{0})^\circ$
 by (*metis circ-add-1 mult-associative mult-left-zero*)

— Theorem 14.6

lemma *circ-plus-sub*: $x^\circ \ ; \ x \leq x \ ; \ x^\circ$
 by (*metis circ-left-slide mult-left-one mult-right-one*)

lemma *circ-loop-fixpoint*: $y \ ; \ (y^\circ \ ; \ z) + z = y^\circ \ ; \ z$
 by (*metis add-commutative circ-left-unfold mult-associative mult-left-one mult-right-dist-add*)

— Theorem 1.6 and Theorem 14.7

lemma *left-plus-below-circ*: $x \ ; \ x^\circ \leq x^\circ$
 by (*metis add-right-upper-bound circ-left-unfold*)

lemma *right-plus-below-circ*: $x^\circ \ ; \ x \leq x^\circ$
 by (*metis add-least-upper-bound circ-right-unfold-sub*)

lemma *circ-add-upper-bound*: $x \leq z^\circ \wedge y \leq z^\circ \longrightarrow x + y \leq z^\circ$
 by (*metis add-least-upper-bound*)

lemma *circ-mult-upper-bound*: $x \leq z^\circ \wedge y \leq z^\circ \longrightarrow x \ ; \ y \leq z^\circ$
 by (*metis mult-isotone circ-transitive-equal*)

lemma *circ-sub-dist*: $x^\circ \leq (x + y)^\circ$
 by (*metis circ-add-sub circ-plus-one mult-left-one mult-right-sub-dist-add-left order-trans*)

lemma *circ-sub-dist-1*: $x \leq (x + y)^\circ$
 by (*metis add-least-upper-bound circ-increasing*)

lemma *circ-sub-dist-2*: $x \ ; \ y \leq (x + y)^\circ$
 by (*metis add-commutative circ-mult-upper-bound circ-sub-dist-1*)

— Theorem 1.20 and Theorem 14.23

lemma *circ-sub-dist-3*: $x^\circ \ ; \ y^\circ \leq (x + y)^\circ$
 by (*metis add-commutative circ-mult-upper-bound circ-sub-dist*)

— Theorem 1 and Theorem 14

lemma *circ-isotone*: $x \leq y \longrightarrow x^\circ \leq y^\circ$
 by (*metis circ-sub-dist less-eq-def*)

— Theorem 1.21 and Theorem 14.24

lemma *circ-add-2*: $(x + y)^\circ \leq (x^\circ \ ; \ y^\circ)^\circ$
 by (*metis add-least-upper-bound circ-increasing circ-isotone circ-reflexive mult-isotone mult-left-one mult-right-one*)

lemma *circ-sup-one-left-unfold*: $\mathit{1} \leq x \longrightarrow x \ ; \ x^\circ = x^\circ$
 by (*metis antisym less-eq-def mult-left-one mult-right-sub-dist-add-left left-plus-below-circ*)

lemma *circ-sup-one-right-unfold*: $\mathit{1} \leq x \longrightarrow x^\circ \ ; \ x = x^\circ$

by (*metis antisym less-eq-def mult-left-sub-dist-add-left mult-right-one right-plus-below-circ*)

— Theorem 1.23 and Theorem 14.26

lemma *circ-decompose-4*: $(x^\circ ; y^\circ)^\circ = x^\circ ; (y^\circ ; x^\circ)^\circ$
by (*metis add-associative add-commutative circ-add-1 circ-loop-fixpoint circ-plus-one circ-rtc-2 circ-transitive-equal mult-associative*)

— Theorem 1.22 and Theorem 14.25

lemma *circ-decompose-5*: $(x^\circ ; y^\circ)^\circ = (y^\circ ; x^\circ)^\circ$
by (*smt add-associative add-commutative add-left-zero circ-add-1 circ-decompose-4 mult-left-zero mult-right-one*)

lemma *circ-decompose-6*: $x^\circ ; (y ; x^\circ)^\circ = y^\circ ; (x ; y^\circ)^\circ$
by (*metis add-commutative circ-add-1*)

lemma *circ-decompose-7*: $(x + y)^\circ = x^\circ ; y^\circ ; (x + y)^\circ$
by (*metis circ-add-1 circ-decompose-6 circ-transitive-equal mult-associative*)

lemma *circ-decompose-8*: $(x + y)^\circ = (x + y)^\circ ; x^\circ ; y^\circ$
by (*metis antisym eq-refl mult-associative mult-isotone mult-right-one circ-mult-upper-bound circ-reflexive circ-sub-dist-3*)

lemma *circ-decompose-9*: $(x^\circ ; y^\circ)^\circ = x^\circ ; y^\circ ; (x^\circ ; y^\circ)^\circ$
by (*metis circ-decompose-4 mult-associative*)

lemma *circ-decompose-10*: $(x^\circ ; y^\circ)^\circ = (x^\circ ; y^\circ)^\circ ; x^\circ ; y^\circ$
by (*metis add-right-upper-bound circ-loop-fixpoint circ-reflexive circ-sup-one-right-unfold mult-associative order-trans*)

lemma *circ-back-loop-prefixpoint*: $(z ; y^\circ) ; y + z \leq z ; y^\circ$
by (*metis add-least-upper-bound circ-left-unfold mult-associative mult-left-sub-dist-add-left mult-right-isotone mult-right-one right-plus-below-circ*)

— Theorem 1 and Theorem 14

lemma *circ-loop-is-fixpoint*: *is-fixpoint* $(\lambda x . y ; x + z)$ $(y^\circ ; z)$
by (*metis circ-loop-fixpoint is-fixpoint-def*)

— Theorem 14

lemma *circ-back-loop-is-prefixpoint*: *is-prefixpoint* $(\lambda x . x ; y + z)$ $(z ; y^\circ)$
by (*metis circ-back-loop-prefixpoint is-prefixpoint-def*)

— Theorem 1.16 and Theorem 14.16

lemma *circ-circ-add*: $(1 + x)^\circ = x^{\circ\circ}$
by (*metis add-commutative circ-add-1 circ-decompose-4 circ-zero mult-right-one*)

— Theorem 14.14

lemma *circ-circ-mult-sub*: $x^\circ ; 1^\circ \leq x^{\circ\circ}$
by (*metis circ-increasing circ-isotone circ-mult-upper-bound circ-reflexive*)

— Theorem 14.9

lemma *left-plus-circ*: $(x ; x^\circ)^\circ = x^\circ$
by (*smt add-idempotent circ-add-1 circ-loop-fixpoint circ-transitive-equal mult-right-dist-add*)

— Theorem 1.10 and Theorem 14.10

lemma *right-plus-circ*: $(x^\circ ; x)^\circ = x^\circ$
by (*metis add-commutative circ-isotone circ-loop-fixpoint circ-plus-sub circ-sub-dist eq-iff left-plus-circ*)

lemma *circ-square*: $(x ; x)^\circ \leq x^\circ$
by (*metis circ-increasing circ-isotone left-plus-circ mult-right-isotone*)

— Theorem 1.19

lemma *circ-mult-sub-add*: $(x ; y)^\circ \leq (x + y)^\circ$
by (*metis add-left-upper-bound add-right-upper-bound circ-isotone circ-square mult-isotone order-trans*)

lemma *circ-add-mult-zero*: $x^\circ ; y = (x + y ; 0)^\circ ; y$
proof –
  have $(x + y ; 0)^\circ ; y = x^\circ ; (1 + y ; 0) ; y$
    by (*metis mult-zero-add-circ mult-zero-circ*)
  also have ... $= x^\circ ; (y + y ; 0)$
    by (*metis mult-associative mult-left-one mult-left-zero mult-right-dist-add*)
  also have ... $= x^\circ ; y$
    by (*metis add-commutative less-eq-def zero-right-mult-decreasing*)
  finally show *?thesis*
    by *metis*
qed

— Theorem 14.22

lemma *troeger-1*: $(x + y)^\circ = x^\circ ; (1 + y ; (x + y)^\circ)$
  by (*metis circ-add-1 circ-left-unfold mult-associative*)

lemma *troeger-2*: $(x + y)^\circ ; z = x^\circ ; (y ; (x + y)^\circ ; z + z)$
  by (*metis circ-add-1 circ-loop-fixpoint mult-associative*)

lemma *troeger-3*: $(x + y ; 0)^\circ = x^\circ ; (1 + y ; 0)$
  by (*metis mult-zero-add-circ mult-zero-circ*)

lemma *circ-add-sub-add-one-1*: $x + y \leq x^\circ ; (1 + y)$
  by (*smt add-associative add-commutative add-idempotent circ-increasing circ-loop-fixpoint less-eq-def mult-left-sub-dist-add-left mult-right-one*)

lemma *circ-add-sub-add-one-2*: $x^\circ ; (x + y) \leq x^\circ ; (1 + y)$
  by (*metis circ-add-sub-add-one-1 circ-transitive-equal mult-associative mult-right-isotone*)

lemma *circ-add-sub-add-one*: $x ; x^\circ ; (x + y) \leq x ; x^\circ ; (1 + y)$
  by (*metis circ-add-sub-add-one-2 mult-associative mult-right-isotone*)

lemma *circ-square-2*: $(x ; x)^\circ ; (x + 1) \leq x^\circ$
  by (*metis add-least-upper-bound circ-increasing circ-mult-upper-bound circ-reflexive circ-square*)

— Theorem 1.25 and Theorem 14.28

lemma *circ-extra-circ*: $(y ; x^\circ)^\circ = (y ; y^\circ ; x^\circ)^\circ$
  by (*smt add-commutative add-idempotent circ-add-1 circ-left-unfold mult-associative*)

— Theorem 14.15

lemma *circ-circ-sub-mult*: $1^\circ ; x^\circ \leq x^{\circ\circ}$
  by (*metis circ-increasing circ-isotone circ-mult-upper-bound circ-reflexive*)

— Theorem 14.27

lemma *circ-decompose-11*: $(x^\circ ; y^\circ)^\circ = (x^\circ ; y^\circ)^\circ ; x^\circ$
  by (*smt circ-add-1 circ-circ-add circ-decompose-4 circ-decompose-8 circ-rtc-2 circ-transitive-equal mult-associative*)

— Theorem 14.20

lemma *circ-mult-below-circ-circ*: $(x ; y)^\circ \leq (x^\circ ; y)^\circ ; x^\circ$
  by (*metis circ-increasing circ-isotone circ-reflexive dual-order.trans mult-left-isotone mult-right-isotone mult-right-one*)

— Theorem 14 counterexamples

lemma *circ-right-unfold*: $1 + x^\circ ; x = x^\circ$ **nitpick** [*expect=genuine*] **oops**
lemma *circ-mult*: $1 + x ; (y ; x)^\circ ; y = (x ; y)^\circ$ **nitpick** [*expect=genuine*] **oops**
lemma *circ-slide*: $(x ; y)^\circ ; x = x ; (y ; x)^\circ$ **nitpick** [*expect=genuine*] **oops**
lemma *circ-plus-same*: $x^\circ ; x = x ; x^\circ$ **nitpick** [*expect=genuine*] **oops**
lemma $1^\circ ; x^\circ \leq x^\circ ; 1^\circ$ **nitpick** [*expect=genuine,card=7*] **oops**
lemma *circ-circ-mult-1*: $x^\circ ; 1^\circ = x^{\circ\circ}$ **nitpick** [*expect=genuine,card=7*] **oops**
lemma $x^\circ ; 1^\circ \leq 1^\circ ; x^\circ$ **nitpick** [*expect=genuine,card=7*] **oops**
lemma *circ-circ-mult*: $1^\circ ; x^\circ = x^{\circ\circ}$ **nitpick** [*expect=genuine,card=7*] **oops**
lemma *circ-add*: $(x^\circ ; y)^\circ ; x^\circ = (x + y)^\circ$ **nitpick** [*expect=genuine,card=8*] **oops**
lemma *circ-unfold-sum*: $(x + y)^\circ = x^\circ + x^\circ ; y ; (x + y)^\circ$ **nitpick** [*expect=genuine,card=7*] **oops**

lemma *mult-zero-add-circ-2*: $(x + y\ ;\ 0)^\circ = x^\circ + x^\circ\ ;\ y\ ;\ 0$ **nitpick** [*expect=genuine,card=7*] **oops**
lemma *sub-mult-one-circ*: $x\ ;\ 1^\circ \le 1^\circ\ ;\ x$ **nitpick** [*expect=genuine*] **oops**
lemma *circ-back-loop-fixpoint*: $(z\ ;\ y^\circ)\ ;\ y + z = z\ ;\ y^\circ$ **nitpick** [*expect=genuine*] **oops**
lemma *circ-back-loop-is-fixpoint*: *is-fixpoint* $(\lambda x\ .\ x\ ;\ y + z)\ (z\ ;\ y^\circ)$ **nitpick** [*expect=genuine*] **oops**
lemma $x^\circ\ ;\ y^\circ \le (x^\circ\ ;\ y)^\circ\ ;\ x^\circ$ **nitpick** [*expect=genuine,card=7*] **oops**

end

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

begin

lemma *circ-top-1*: $T^\circ = T$
  by (*metis add-right-top antisym circ-left-unfold mult-left-sub-dist-add-left mult-right-one top-greatest*)

lemma *circ-right-top-1*: $x^\circ\ ;\ T = T$
  by (*metis add-right-top circ-loop-fixpoint*)

lemma *circ-left-top-1*: $T\ ;\ x^\circ = T$
  by (*metis antisym circ-plus-one mult-left-sub-dist-add-left mult-right-one top-greatest*)

lemma *mult-top-circ-1*: $(x\ ;\ T)^\circ = 1 + x\ ;\ T$
  by (*metis circ-left-top-1 circ-left-unfold mult-associative*)

end

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

begin

lemma *mult-zero-add-circ-2*: $(x + y\ ;\ 0)^\circ = x^\circ + x^\circ\ ;\ y\ ;\ 0$
  by (*metis mult-associative mult-left-dist-add mult-right-one troeger-3*)

lemma *circ-unfold-sum*: $(x + y)^\circ = x^\circ + x^\circ\ ;\ y\ ;\ (x + y)^\circ$
  by (*metis mult-associative mult-left-dist-add mult-right-one troeger-1*)

end

class *left-conway-semiring-1* = *left-conway-semiring* +
  assumes *circ-right-slide*: $x\ ;\ (y\ ;\ x)^\circ \le (x\ ;\ y)^\circ\ ;\ x$

begin

— Theorem 1.26

lemma *circ-slide-1*: $x\ ;\ (y\ ;\ x)^\circ = (x\ ;\ y)^\circ\ ;\ x$
  by (*metis antisym circ-left-slide circ-right-slide*)

— Theorem 1.9

lemma *circ-right-unfold-1*: $1 + x^\circ\ ;\ x = x^\circ$
  by (*metis circ-left-unfold circ-slide-1 mult-left-one mult-right-one*)

lemma *circ-mult-1*: $(x\ ;\ y)^\circ = 1 + x\ ;\ (y\ ;\ x)^\circ\ ;\ y$
  by (*metis circ-left-unfold circ-slide-1 mult-associative*)

lemma *circ-add-9*: $(x + y)^\circ = (x^\circ\ ;\ y)^\circ\ ;\ x^\circ$
  by (*metis circ-add-1 circ-slide-1*)

— Theorem 1.7

lemma *circ-plus-same*: $x^\circ\ ;\ x = x\ ;\ x^\circ$
  by (*metis circ-slide-1 mult-left-one mult-right-one*)

lemma *circ-decompose-12*: $x^\circ\ ;\ y^\circ \le (x^\circ\ ;\ y)^\circ\ ;\ x^\circ$
  by (*metis circ-add-9 circ-sub-dist-3*)

end

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

begin

lemma *circ-back-loop-fixpoint*: $(z ; y^\circ) ; y + z = z ; y^\circ$
 by (*metis add-commutative circ-left-unfold circ-plus-same mult-associative mult-left-dist-add mult-right-one*)

— Theorem 1

lemma *circ-back-loop-is-fixpoint*: *is-fixpoint* $(\lambda x . x ; y + z) (z ; y^\circ)$
 by (*metis circ-back-loop-fixpoint is-fixpoint-def*)

lemma *circ-elimination*: $x ; y = 0 \longrightarrow x ; y^\circ \le x$
 by (*metis add-left-zero circ-back-loop-fixpoint circ-plus-same mult-associative mult-left-zero order-refl*)

end

class *itering-1* = *left-conway-semiring-1* +
 assumes *circ-simulate*: $z ; x \le y ; z \longrightarrow z ; x^\circ \le y^\circ ; z$

begin

— Theorem 1.15

lemma *circ-circ-mult*: $1^\circ ; x^\circ = x^{\circ\circ}$
 by (*metis antisym circ-circ-add circ-reflexive circ-simulate circ-sub-dist-3 circ-sup-one-left-unfold circ-transitive-equal mult-left-one order-refl*)

lemma *sub-mult-one-circ*: $x ; 1^\circ \le 1^\circ ; x$
 by (*metis circ-simulate mult-left-one mult-right-one order-refl*)

lemma *circ-import*: $p \le p ; p \wedge p \le 1 \wedge p ; x \le x ; p \longrightarrow p ; x^\circ = p ; (p ; x)^\circ$
 apply *rule*
 apply (*rule antisym*)
 apply (*smt antisym circ-simulate circ-slide-1 mult-associative mult-right-isotone mult-right-one order-refl test-preserves-equation*)
 apply (*metis circ-isotone mult-left-isotone mult-left-one mult-right-isotone*)
 done

end

class *itering-2* = *left-conway-semiring-1* +
 assumes *circ-simulate-right*: $z ; x \le y ; z + w \longrightarrow z ; x^\circ \le y^\circ ; (z + w ; x^\circ)$
 assumes *circ-simulate-left*: $x ; z \le z ; y + w \longrightarrow x^\circ ; z \le (z + x^\circ ; w) ; y^\circ$

begin

subclass *itering-1*
 apply *unfold-locales*
 apply (*metis add-right-zero circ-simulate-right mult-left-zero*)
 done

lemma *circ-simulate-left-1*: $x ; z \le z ; y \longrightarrow x^\circ ; z \le z ; y^\circ + x^\circ ; 0$
 by (*metis add-right-zero circ-simulate-left mult-associative mult-left-zero mult-right-dist-add*)

lemma *circ-separate-1*: $y ; x \le x ; y \longrightarrow (x + y)^\circ = x^\circ ; y^\circ$
proof −
 have $y ; x \le x ; y \longrightarrow y^\circ ; x ; y^\circ \le x ; y^\circ + y^\circ ; 0$
  by (*smt circ-simulate-left-1 circ-transitive-equal mult-associative mult-left-isotone mult-left-zero mult-right-dist-add*)
 thus *?thesis*
  by (*smt add-commutative circ-add-1 circ-simulate-right circ-sub-dist-3 less-eq-def mult-associative mult-left-zero zero-right-mult-decreasing*)
qed

— Theorem 1.14

lemma *circ-circ-mult-1*: $x^\circ ; 1^\circ = x^{\circ\circ}$
 by (*metis add-commutative circ-circ-add circ-separate-1 mult-left-one mult-right-one order-refl*)

end

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

begin

lemma *circ-simulate-1*: $y ; x \leq x ; y \longrightarrow y^\circ ; x^\circ \leq x^\circ ; y^\circ$
by (*smt add-associative add-right-zero circ-loop-fixpoint circ-simulate circ-simulate-left-1 mult-associative mult-left-zero mult-zero-add-circ-2*)

— Theorem 4

lemma *atomicity-refinement*: $s = s ; q \wedge x = q ; x \wedge q ; b = 0 \wedge r ; b \leq b ; r \wedge r ; l \leq l ; r \wedge x ; l \leq l ; x \wedge b ; l \leq l ; b \wedge q ; l \leq l ; q \wedge r^\circ ; q \leq q ; r^\circ \wedge q \leq 1 \longrightarrow s ; (x + b + r + l)^\circ ; q \leq s ; (x ; b^\circ ; q + r + l)^\circ$
proof
assume *1*: $s = s ; q \wedge x = q ; x \wedge q ; b = 0 \wedge r ; b \leq b ; r \wedge r ; l \leq l ; r \wedge x ; l \leq l ; x \wedge b ; l \leq l ; b \wedge q ; l \leq l ; q \wedge r^\circ ; q \leq q ; r^\circ \wedge q \leq 1$
hence $s ; (x + b + r + l)^\circ ; q = s ; l^\circ ; (x + b + r)^\circ ; q$
by (*smt add-commutative add-least-upper-bound circ-separate-1 mult-associative mult-left-sub-dist-add-right mult-right-dist-add order-trans*)
also have ... $= s ; l^\circ ; b^\circ ; r^\circ ; q ; (x ; b^\circ ; r^\circ ; q)^\circ$ **using** *1*
by (*smt add-associative add-commutative circ-add-1 circ-separate-1 circ-slide-1 mult-associative*)
also have ... $\leq s ; l^\circ ; b^\circ ; r^\circ ; q ; (x ; b^\circ ; q ; r^\circ)^\circ$ **using** *1*
by (*metis circ-isotone mult-associative mult-right-isotone*)
also have ... $\leq s ; q ; l^\circ ; b^\circ ; r^\circ ; (x ; b^\circ ; q ; r^\circ)^\circ$ **using** *1*
by (*metis mult-left-isotone mult-right-isotone mult-right-one*)
also have ... $\leq s ; l^\circ ; q ; b^\circ ; r^\circ ; (x ; b^\circ ; q ; r^\circ)^\circ$ **using** *1*
by (*metis circ-simulate mult-associative mult-left-isotone mult-right-isotone*)
also have ... $\leq s ; l^\circ ; r^\circ ; (x ; b^\circ ; q ; r^\circ)^\circ$ **using** *1*
by (*metis add-left-zero circ-back-loop-fixpoint circ-plus-same mult-associative mult-left-zero mult-left-isotone mult-right-isotone mult-right-one*)
also have ... $\leq s ; (x ; b^\circ ; q + r + l)^\circ$ **using** *1*
by (*metis add-commutative circ-add-1 circ-sub-dist-3 mult-associative mult-right-isotone*)
finally show $s ; (x + b + r + l)^\circ ; q \leq s ; (x ; b^\circ ; q + r + l)^\circ$ **.**
qed

end

class *itering* = *idempotent-left-zero-semiring* + *circ* +
assumes *circ-add*: $(x + y)^\circ = (x^\circ ; y)^\circ ; x^\circ$
assumes *circ-mult*: $(x ; y)^\circ = 1 + x ; (y ; x)^\circ ; y$
assumes *circ-simulate-right-plus*: $z ; x \leq y ; y^\circ ; z + w \longrightarrow z ; x^\circ \leq y^\circ ; (z + w ; x^\circ)$
assumes *circ-simulate-left-plus*: $x ; z \leq z ; y^\circ + w \longrightarrow x^\circ ; z \leq (z + x^\circ ; w) ; y^\circ$

begin

lemma *circ-right-unfold*: $1 + x^\circ ; x = x^\circ$
by (*metis circ-mult mult-left-one mult-right-one*)

lemma *circ-slide*: $x ; (y ; x)^\circ = (x ; y)^\circ ; x$
by (*smt2 circ-mult mult-associative mult-left-dist-add mult-left-one mult-right-dist-add mult-right-one order-refl*)

— Theorem 50.6

subclass *itering-3*
apply *unfold-locales*
apply (*metis circ-mult mult-left-one mult-right-one*) — Theorem 1.8
apply (*metis circ-slide order-refl*)
apply (*metis circ-add circ-slide*)
apply (*metis circ-slide order-refl*)
apply (*metis add-left-isotone circ-right-unfold mult-left-isotone mult-left-sub-dist-add-left mult-right-one order-trans circ-simulate-right-plus*)
apply (*metis add-commutative add-left-upper-bound add-right-isotone circ-mult mult-right-isotone mult-right-one order-trans circ-simulate-left-plus*)
done

lemma *circ-simulate-right-plus-1*: $z ; x \leq y ; y^\circ ; z \longrightarrow z ; x^\circ \leq y^\circ ; z$
by (*metis add-right-zero circ-simulate-right-plus mult-left-zero*)

lemma *circ-simulate-left-plus-1*: $x\ ;\ z \le z\ ;\ y^\circ \longrightarrow x^\circ\ ;\ z \le z\ ;\ y^\circ + x^\circ\ ;\ 0$
by (*smt add-right-zero circ-simulate-left-plus mult-associative mult-left-zero mult-right-dist-add*)

lemma *circ-simulate-2*: $y\ ;\ x^\circ \le x^\circ\ ;\ y^\circ \longleftrightarrow y^\circ\ ;\ x^\circ \le x^\circ\ ;\ y^\circ$
apply (*rule iffI*)
apply (*smt add-associative add-right-zero circ-loop-fixpoint circ-simulate-left-plus-1 mult-associative mult-left-zero mult-zero-add-circ-2*)
apply (*metis circ-increasing mult-left-isotone order-trans*)
done

lemma *circ-simulate-absorb*: $y\ ;\ x \le x \longrightarrow y^\circ\ ;\ x \le x + y^\circ\ ;\ 0$
by (*metis circ-simulate-left-plus-1 circ-zero mult-right-one*)

lemma *circ-simulate-3*: $y\ ;\ x^\circ \le x^\circ \longrightarrow y^\circ\ ;\ x^\circ \le x^\circ\ ;\ y^\circ$
by (*metis add-least-upper-bound circ-reflexive circ-simulate-2 less-eq-def mult-right-isotone mult-right-one*)

lemma *circ-separate-mult-1*: $y\ ;\ x \le x\ ;\ y \longrightarrow (x\ ;\ y)^\circ \le x^\circ\ ;\ y^\circ$
by (*metis circ-mult-sub-add circ-separate-1*)

— Theorem 1.24

lemma *circ-separate-unfold*: $(y\ ;\ x^\circ)^\circ = y^\circ + y^\circ\ ;\ y\ ;\ x\ ;\ x^\circ\ ;\ (y\ ;\ x^\circ)^\circ$
by (*smt add-commutative circ-add circ-left-unfold circ-loop-fixpoint mult-associative mult-left-dist-add mult-right-one*)

— Theorem 3

lemma *separation*: $y\ ;\ x \le x\ ;\ y^\circ \longrightarrow (x + y)^\circ = x^\circ\ ;\ y^\circ$
proof –
have $y\ ;\ x \le x\ ;\ y^\circ \longrightarrow y^\circ\ ;\ x\ ;\ y^\circ \le x\ ;\ y^\circ + y^\circ\ ;\ 0$
by (*smt circ-simulate-left-plus-1 circ-transitive-equal mult-associative mult-left-isotone mult-left-zero mult-right-dist-add*)
thus *?thesis*
by (*smt add-commutative circ-add-1 circ-simulate-right circ-sub-dist-3 less-eq-def mult-associative mult-left-zero zero-right-mult-decreasing*)
qed

— Theorem 3

lemma *simulation*: $y\ ;\ x \le x\ ;\ y^\circ \longrightarrow y^\circ\ ;\ x^\circ \le x^\circ\ ;\ y^\circ$
by (*metis add-right-upper-bound circ-isotone circ-mult-upper-bound circ-sub-dist separation*)

— Theorem 3

lemma *circ-simulate-4*: $y\ ;\ x \le x\ ;\ x^\circ\ ;\ (1 + y) \longrightarrow y^\circ\ ;\ x^\circ \le x^\circ\ ;\ y^\circ$
proof
assume $y\ ;\ x \le x\ ;\ x^\circ\ ;\ (1 + y)$
hence $(1 + y)\ ;\ x \le x\ ;\ x^\circ\ ;\ (1 + y)$
by (*smt add-associative add-commutative add-left-upper-bound circ-back-loop-fixpoint less-eq-def mult-left-dist-add mult-left-one mult-right-dist-add mult-right-one*)
hence $y\ ;\ x^\circ \le x^\circ\ ;\ y^\circ$
by (*metis circ-add-upper-bound circ-increasing circ-reflexive circ-simulate-right-plus-1 mult-right-isotone mult-right-sub-dist-add-right order-trans*)
thus $y^\circ\ ;\ x^\circ \le x^\circ\ ;\ y^\circ$
by (*metis circ-simulate-2*)
qed

lemma *circ-simulate-5*: $y\ ;\ x \le x\ ;\ x^\circ\ ;\ (x + y) \longrightarrow y^\circ\ ;\ x^\circ \le x^\circ\ ;\ y^\circ$
by (*metis circ-add-sub-add-one circ-simulate-4 order-trans*)

lemma *circ-simulate-6*: $y\ ;\ x \le x\ ;\ (x + y) \longrightarrow y^\circ\ ;\ x^\circ \le x^\circ\ ;\ y^\circ$
by (*metis add-commutative circ-back-loop-fixpoint circ-simulate-5 mult-right-sub-dist-add-left order-trans*)

— Theorem 3

lemma *circ-separate-4*: $y\ ;\ x \le x\ ;\ x^\circ\ ;\ (1 + y) \longrightarrow (x + y)^\circ = x^\circ\ ;\ y^\circ$
proof
assume *1*: $y\ ;\ x \le x\ ;\ x^\circ\ ;\ (1 + y)$
hence $y\ ;\ x\ ;\ x^\circ \le x\ ;\ x^\circ + x\ ;\ x^\circ\ ;\ y\ ;\ x^\circ$
by (*smt circ-transitive-equal less-eq-def mult-associative mult-left-dist-add mult-right-dist-add mult-right-one*)
also have ... $\le x\ ;\ x^\circ + x\ ;\ x^\circ\ ;\ x^\circ\ ;\ y^\circ$ **using** *1*

by (*metis add-right-isotone circ-simulate-2 circ-simulate-4 mult-associative mult-right-isotone*)
finally have $y ; x ; x^\circ \le x ; x^\circ ; y^\circ$
by (*metis circ-reflexive circ-transitive-equal less-eq-def mult-associative mult-right-isotone mult-right-one*)
hence $y^\circ ; (y^\circ ; x)^\circ \le x^\circ ; (y^\circ + y^\circ ; 0 ; (y^\circ ; x)^\circ)$
by (*smt add-right-upper-bound circ-back-loop-fixpoint circ-simulate-left-plus-1 circ-simulate-right-plus circ-transitive-equal mult-associative order-trans*)
thus $(x + y)^\circ = x^\circ ; y^\circ$
by (*smt add-commutative antisym circ-add-1 circ-slide circ-sub-dist-3 circ-transitive-equal less-eq-def mult-associative mult-left-zero mult-right-sub-dist-add-right zero-right-mult-decreasing*)
qed

lemma *circ-separate-5*: $y ; x \le x ; x^\circ ; (x + y) \longrightarrow (x + y)^\circ = x^\circ ; y^\circ$
by (*metis circ-add-sub-add-one circ-separate-4 order-trans*)

lemma *circ-separate-6*: $y ; x \le x ; (x + y) \longrightarrow (x + y)^\circ = x^\circ ; y^\circ$
by (*metis add-commutative circ-back-loop-fixpoint circ-separate-5 mult-right-sub-dist-add-left order-trans*)

end

class *bounded-itering* = *bounded-idempotent-left-zero-semiring* + *itering*

begin

— Theorem 1

lemma *circ-top*: $T^\circ = T$
by (*metis add-right-top antisym circ-left-unfold mult-left-sub-dist-add-left mult-right-one top-greatest*)

lemma *circ-right-top*: $x^\circ ; T = T$
by (*metis add-right-top circ-loop-fixpoint*)

lemma *circ-left-top*: $T ; x^\circ = T$
by (*metis add-right-top circ-add circ-right-top circ-top*)

lemma *mult-top-circ*: $(x ; T)^\circ = 1 + x ; T$
by (*metis circ-left-top circ-mult mult-associative*)

— Theorem 1 counterexamples

lemma $1 = x^\circ$ **nitpick** [*expect=genuine*] **oops**
lemma $x = x^\circ$ **nitpick** [*expect=genuine*] **oops**
lemma $x = x ; x^\circ$ **nitpick** [*expect=genuine*] **oops**
lemma $x ; x^\circ = x^\circ$ **nitpick** [*expect=genuine*] **oops**
lemma $x^\circ = x^{\circ\circ}$ **nitpick** [*expect=genuine*] **oops**
lemma $(x ; y)^\circ = (x + y)^\circ$ **nitpick** [*expect=genuine*] **oops**
lemma $x^\circ ; y^\circ = (x + y)^\circ$ **nitpick** [*expect=genuine,card=6*] **oops**
lemma $(x + y)^\circ = (x^\circ ; y^\circ)^\circ$ **nitpick** [*expect=genuine*] **oops**
lemma $1 = 1^\circ$ **nitpick** [*expect=genuine*] **oops**

lemma $1 = (x ; 0)^\circ$ **nitpick** [*expect=genuine*] **oops**
lemma $1 + x ; 0 = x^\circ$ **nitpick** [*expect=genuine*] **oops**
lemma $x^\circ = x^\circ ; 1^\circ$ **nitpick** [*expect=genuine*] **oops**
lemma $z + y ; x = x \longrightarrow y^\circ ; z \le x$ **nitpick** [*expect=genuine*] **oops**
lemma $y ; x = x \longrightarrow y^\circ ; x \le x$ **nitpick** [*expect=genuine*] **oops**
lemma $z + x ; y = x \longrightarrow z ; y^\circ \le x$ **nitpick** [*expect=genuine*] **oops**
lemma $x ; y = x \longrightarrow x ; y^\circ \le x$ **nitpick** [*expect=genuine*] **oops**
lemma $x = z + y ; x \longrightarrow x \le y^\circ ; z$ **nitpick** [*expect=genuine*] **oops**
lemma $x = y ; x \longrightarrow x \le y^\circ$ **nitpick** [*expect=genuine*] **oops**
lemma $x ; z = z ; y \longrightarrow x^\circ ; z \le z ; y^\circ$ **nitpick** [*expect=genuine*] **oops**

lemma $x^\circ = (x ; x)^\circ ; (x + 1)$ **oops**
lemma $y^\circ ; x^\circ \le x^\circ ; y^\circ \longrightarrow (x + y)^\circ = x^\circ ; y^\circ$ **oops**
lemma $y ; x \le (1 + x) ; y^\circ \longrightarrow (x + y)^\circ = x^\circ ; y^\circ$ **oops**
lemma $y ; x \le x \longrightarrow y^\circ ; x \le 1^\circ ; x$ **oops**

end

class *left-conway-semiring-L* = *left-conway-semiring* + *L* +
assumes *one-circ-mult-split*: $1^\circ ; x = L + x$

assumes *L-split-add*: $x ; (y + L) \leq x ; y + L$

begin

lemma *L-def*: $L = 1^\circ ; 0$
by (*metis add-right-zero one-circ-mult-split*)

lemma *one-circ-split*: $1^\circ = L + 1$
by (*metis mult-right-one one-circ-mult-split*)

lemma *one-circ-circ-split*: $1^{\circ\circ} = L + 1$
by (*metis circ-one one-circ-split*)

lemma *sub-mult-one-circ*: $x ; 1^\circ \leq 1^\circ ; x$
by (*metis L-split-add add-commutative mult-right-one one-circ-mult-split*)

lemma *one-circ-mult-split-2*: $1^\circ ; x = x ; 1^\circ + L$
by (*smt add-associative add-commutative add-least-upper-bound circ-back-loop-prefixpoint less-eq-def one-circ-mult-split sub-mult-one-circ*)

lemma *sub-mult-one-circ-split*: $x ; 1^\circ \leq x + L$
by (*metis add-commutative one-circ-mult-split sub-mult-one-circ*)

lemma *sub-mult-one-circ-split-2*: $x ; 1^\circ \leq x + 1^\circ$
by (*metis L-def add-right-isotone order-trans sub-mult-one-circ-split zero-right-mult-decreasing*)

lemma *L-split*: $x ; L \leq x ; 0 + L$
by (*metis L-split-add add-left-zero*)

lemma *L-left-zero*: $L ; x = L$
by (*metis L-def mult-associative mult-left-zero*)

lemma *one-circ-L*: $1^\circ ; L = L$
by (*metis L-def circ-transitive-equal mult-associative*)

lemma *mult-L-circ*: $(x ; L)^\circ = 1 + x ; L$
by (*metis L-left-zero circ-left-unfold mult-associative*)

lemma *mult-L-circ-mult*: $(x ; L)^\circ ; y = y + x ; L$
by (*metis L-left-zero mult-L-circ mult-associative mult-left-one mult-right-dist-add*)

lemma *circ-L*: $L^\circ = L + 1$
by (*metis L-left-zero add-commutative circ-left-unfold*)

lemma *L-below-one-circ*: $L \leq 1^\circ$
by (*metis L-def zero-right-mult-decreasing*)

lemma *circ-circ-mult-1*: $x^\circ ; 1^\circ = x^{\circ\circ}$
by (*metis L-left-zero add-commutative circ-add-1 circ-circ-add mult-zero-circ one-circ-split*)

lemma *circ-circ-mult*: $1^\circ ; x^\circ = x^{\circ\circ}$
by (*metis antisym circ-circ-mult-1 circ-circ-sub-mult sub-mult-one-circ*)

lemma *circ-circ-split*: $x^{\circ\circ} = L + x^\circ$
by (*metis circ-circ-mult one-circ-mult-split*)

lemma *circ-add-6*: $L + (x + y)^\circ = (x^\circ ; y^\circ)^\circ$
by (*metis add-associative add-commutative circ-add-1 circ-circ-add circ-circ-split circ-decompose-4*)

end

class *itering-L* = *itering* + *L* +
assumes *L-def*: $L = 1^\circ ; 0$

begin

lemma *one-circ-split*: $1^\circ = L + 1$
by (*metis L-def add-commutative antisym circ-add-upper-bound circ-reflexive circ-simulate-absorb mult-right-one order-refl zero-right-mult-decreasing*)

lemma *one-circ-mult-split*: $1^\circ \,;\, x = L + x$
  by (*metis L-def add-commutative circ-loop-fixpoint mult-associative mult-left-zero mult-zero-circ one-circ-split*)

lemma *sub-mult-one-circ-split*: $x \,;\, 1^\circ \leq x + L$
  by (*metis add-commutative one-circ-mult-split sub-mult-one-circ*)

lemma *sub-mult-one-circ-split-2*: $x \,;\, 1^\circ \leq x + 1^\circ$
  by (*metis L-def add-right-isotone order-trans sub-mult-one-circ-split zero-right-mult-decreasing*)

lemma *L-split*: $x \,;\, L \leq x \,;\, 0 + L$
  by (*smt L-def mult-associative mult-left-isotone mult-right-dist-add sub-mult-one-circ-split-2*)

subclass *left-conway-semiring-L*
  apply *unfold-locales*
  apply (*metis L-def add-commutative circ-loop-fixpoint mult-associative mult-left-zero mult-zero-circ one-circ-split*)
  apply (*metis add-commutative mult-associative mult-left-isotone one-circ-mult-split sub-mult-one-circ*)
  done

lemma *circ-left-induct-mult-L*: $L \leq x \longrightarrow x \,;\, y \leq x \longrightarrow x \,;\, y^\circ \leq x$
  by (*metis circ-one circ-simulate less-eq-def one-circ-mult-split*)

lemma *circ-left-induct-mult-iff-L*: $L \leq x \longrightarrow x \,;\, y \leq x \longleftrightarrow x \,;\, y^\circ \leq x$
  by (*smt add-least-upper-bound circ-back-loop-fixpoint circ-left-induct-mult-L less-eq-def*)

lemma *circ-left-induct-L*: $L \leq x \longrightarrow x \,;\, y + z \leq x \longrightarrow z \,;\, y^\circ \leq x$
  by (*metis add-least-upper-bound circ-left-induct-mult-L less-eq-def mult-right-dist-add*)

end

end

6 KleeneAlgebra

theory *KleeneAlgebra*

imports *Itering*

begin

class *star* =
  fixes *star* :: $'a \Rightarrow 'a$ (-* [*100*] *100*)

class *left-kleene-algebra* = *idempotent-left-semiring* + *star* +
  assumes *star-left-unfold* : $1 + y ; y^\star \le y^\star$
  assumes *star-left-induct* : $z + y ; x \le x \longrightarrow y^\star ; z \le x$

begin

lemma *star-left-unfold-equal*: $1 + x ; x^\star = x^\star$
  by (*smt add-right-isotone antisym mult-right-isotone mult-right-one star-left-induct star-left-unfold*)

lemma *star-left-slide*: $(x ; y)^\star ; x \le x ; (y ; x)^\star$
  by (*metis mult-associative mult-left-sub-dist-add mult-right-one star-left-induct star-left-unfold-equal*)

lemma *star-isotone*: $x \le y \longrightarrow x^\star \le y^\star$
  by (*metis add-right-isotone mult-left-isotone order-trans star-left-unfold mult-right-one star-left-induct*)

lemma *star-add-1-sub*: $x^\star ; (y ; x^\star)^\star \le (x + y)^\star$
proof −
  have $x^\star ; (y ; x^\star)^\star \le x^\star ; (y ; (x + y)^\star)^\star$
    by (*smt add-left-upper-bound mult-right-isotone star-isotone*)
  also have ... $\le x^\star ; ((x + y) ; (x + y)^\star)^\star$
    by (*smt add-right-upper-bound mult-left-isotone mult-right-isotone star-isotone*)
  also have ... $\le x^\star ; (x + y)^{\star\star}$
    by (*smt add-least-upper-bound mult-right-isotone star-isotone star-left-unfold*)
  also have ... $\le (x + y)^\star ; (x + y)^{\star\star}$
    by (*smt add-left-upper-bound mult-left-isotone star-isotone*)
  also have ... $\le (x + y)^\star$
    by (*smt add-least-upper-bound mult-right-one star-left-induct star-left-unfold*)
  finally show $x^\star ; (y ; x^\star)^\star \le (x + y)^\star$
    by *smt*
qed

lemma *star-add-1*: $(x + y)^\star = x^\star ; (y ; x^\star)^\star$
  apply (*rule antisym*)
    apply (*smt add-least-upper-bound add-left-upper-bound add-right-upper-bound mult-associative mult-left-one mult-right-dist-add mult-right-one star-left-induct star-left-unfold-equal*)
  apply (*smt star-add-1-sub*)
  done

end

— Theorem 50.1

sublocale *left-kleene-algebra* < *star*!: *left-conway-semiring* **where** *circ* = *star*
  apply *unfold-locales*
  apply (*metis star-left-unfold-equal*)
  apply (*metis star-left-slide*)
  apply (*metis star-add-1*)
  done

context *left-kleene-algebra*

begin

— Many lemmas in this class are taken from Georg Struth's Isabelle theories.

lemma *star-sub-one*: $x \le 1 \longrightarrow x^\star = 1$
  by (*metis add-right-isotone eq-iff less-eq-def mult-right-one star.circ-plus-one star-left-induct*)

lemma *star-one*: $1^\star = 1$
by (*metis eq-iff star-sub-one*)

lemma *star-left-induct-mult*: $x ; y \le y \longrightarrow x^\star ; y \le y$
by (*metis add-commutative less-eq-def order-refl star-left-induct*)

lemma *star-left-induct-mult-iff*: $x ; y \le y \longleftrightarrow x^\star ; y \le y$
by (*metis mult-associative mult-left-isotone mult-left-one mult-right-isotone order-trans star-left-induct-mult star.circ-reflexive star.left-plus-below-circ*)

lemma *star-involutive*: $x^\star = x^{\star\star}$
by (*smt antisym less-eq-def mult-left-sub-dist-add-left mult-right-one star-left-induct star.circ-plus-one star.left-plus-below-circ star.circ-transitive-equal*)

lemma *star-sup-one*: $(1 + x)^\star = x^\star$
by (*metis star.circ-circ-add star-involutive*)

lemma *star-left-induct-equal*: $z + x ; y = y \longrightarrow x^\star ; z \le y$
by (*metis order-refl star-left-induct*)

lemma *star-left-induct-mult-equal*: $x ; y = y \longrightarrow x^\star ; y \le y$
by (*metis order-refl star-left-induct-mult*)

lemma *star-star-upper-bound*: $x^\star \le z^\star \longrightarrow x^{\star\star} \le z^\star$
by (*metis star-involutive*)

lemma *star-simulation-left*: $x ; z \le z ; y \longrightarrow x^\star ; z \le z ; y^\star$
by (*smt add-commutative add-least-upper-bound mult-right-dist-add less-eq-def mult-associative mult-right-one star.left-plus-below-circ star.circ-increasing star-left-induct star-involutive star.circ-isotone star.circ-reflexive mult-left-sub-dist-add-left*)

lemma *quasicomm-1*: $y ; x \le x ; (x + y)^\star \longleftrightarrow y^\star ; x \le x ; (x + y)^\star$
by (*smt mult-isotone order-refl order-trans star.circ-increasing star-involutive star-simulation-left*)

lemma *star-rtc-3*: $1 + x + y ; y = y \longrightarrow x^\star \le y$
by (*metis add-least-upper-bound less-eq-def mult-left-sub-dist-add-left mult-right-one star-left-induct-mult-iff star.circ-sub-dist*)

lemma *star-decompose-1*: $(x + y)^\star = (x^\star ; y^\star)^\star$
apply (*rule antisym*)
apply (*smt add-least-upper-bound mult-isotone mult-left-one mult-right-one star.circ-increasing star.circ-isotone star.circ-reflexive*)
apply (*smt star.circ-isotone star.circ-sub-dist-3 star-involutive*)
done

lemma *star-sum*: $(x + y)^\star = (x^\star + y^\star)^\star$
by (*metis star-decompose-1 star-involutive*)

lemma *star-decompose-3*: $(x^\star ; y^\star)^\star = x^\star ; (y ; x^\star)^\star$
by (*metis star-decompose-1 star.circ-add-1*)

lemma *star-loop-least-fixpoint*: $y ; x + z = x \longrightarrow y^\star ; z \le x$
by (*metis add-commutative star-left-induct-equal*)

lemma *star-loop-is-least-fixpoint*: *is-least-fixpoint* $(\lambda x . y ; x + z)$ $(y^\star ; z)$
by (*smt is-least-fixpoint-def star.circ-loop-fixpoint star-loop-least-fixpoint*)

lemma *star-loop-mu*: μ $(\lambda x . y ; x + z) = y^\star ; z$
by (*metis least-fixpoint-same star-loop-is-least-fixpoint*)

lemma *affine-has-least-fixpoint*: *has-least-fixpoint* $(\lambda x . y ; x + z)$
by (*metis has-least-fixpoint-def star-loop-is-least-fixpoint*)

lemma *circ-add*: $(x^\star ; y)^\star ; x^\star = (x + y)^\star$ **nitpick** [*expect=genuine,card=7*] **oops**
lemma *circ-mult*: $1 + x ; (y ; x)^\star ; y = (x ; y)^\star$ **nitpick** [*expect=genuine*] **oops**
lemma *circ-plus-same*: $x^\star ; x = x ; x^\star$ **nitpick** [*expect=genuine*] **oops**
lemma *circ-unfold-sum*: $(x + y)^\star = x^\star + x^\star ; y ; (x + y)^\star$ **nitpick** [*expect=genuine,card=8*] **oops**
lemma *mult-zero-add-circ-2*: $(x + y ; 0)^\star = x^\star + x^\star ; y ; 0$ **nitpick** [*expect=genuine,card=7*] **oops**
lemma *circ-simulate-left*: $x ; z \le z ; y + w \longrightarrow x^\star ; z \le (z + x^\star ; w) ; y^\star$ **nitpick** [*expect=genuine*] **oops**
lemma *circ-simulate-1*: $y ; x \le x ; y \longrightarrow y^\star ; x^\star \le x^\star ; y^\star$ **nitpick** [*expect=genuine,card=7*] **oops**

lemma *circ-separate-1*: $y ; x \le x ; y \longrightarrow (x + y)^\star = x^\star ; y^\star$ **nitpick** [*expect=genuine,card=7*] **oops**
lemma *atomicity-refinement*: $s = s ; q \wedge x = q ; x \wedge q ; b = 0 \wedge r ; b \le b ; r \wedge r ; l \le l ; r \wedge x ; l \le l ; x \wedge b ; l \le l ; b \wedge q ; l \le l ; q \wedge r^\star ; q \le q ; r^\star \wedge q \le 1 \longrightarrow s ; (x + b + r + l)^\star ; q \le s ; (x ; b^\star ; q + r + l)^\star$ **nitpick** [*expect=genuine*] **oops**
lemma *circ-simulate-left-plus*: $x ; z \le z ; y^\star + w \longrightarrow x^\star ; z \le (z + x^\star ; w) ; y^\star$ **nitpick** [*expect=genuine*] **oops**
lemma *circ-separate-unfold*: $(y ; x^\star)^\star = y^\star + y^\star ; y ; x ; x^\star ; (y ; x^\star)^\star$ **nitpick** [*expect=genuine*] **oops**
lemma *separation*: $y ; x \le x ; y^\star \longrightarrow (x + y)^\star = x^\star ; y^\star$ **nitpick** [*expect=genuine,card=7*] **oops**
lemma *circ-simulate-4*: $y ; x \le x ; x^\star ; (1 + y) \longrightarrow y^\star ; x^\star \le x^\star ; y^\star$ **nitpick** [*expect=genuine,card=7*] **oops**
lemma *circ-simulate-5*: $y ; x \le x ; x^\star ; (x + y) \longrightarrow y^\star ; x^\star \le x^\star ; y^\star$ **nitpick** [*expect=genuine,card=7*] **oops**
lemma *circ-simulate-6*: $y ; x \le x ; (x + y) \longrightarrow y^\star ; x^\star \le x^\star ; y^\star$ **nitpick** [*expect=genuine,card=7*] **oops**
lemma *circ-separate-4*: $y ; x \le x ; x^\star ; (1 + y) \longrightarrow (x + y)^\star = x^\star ; y^\star$ **nitpick** [*expect=genuine,card=7*] **oops**
lemma *circ-separate-5*: $y ; x \le x ; x^\star ; (x + y) \longrightarrow (x + y)^\star = x^\star ; y^\star$ **nitpick** [*expect=genuine,card=7*] **oops**
lemma *circ-separate-6*: $y ; x \le x ; (x + y) \longrightarrow (x + y)^\star = x^\star ; y^\star$ **nitpick** [*expect=genuine,card=7*] **oops**

end

class *strong-left-kleene-algebra* = *left-kleene-algebra* +
 assumes *star-right-induct*: $z + x ; y \le x \longrightarrow z ; y^\star \le x$

begin

lemma *star-plus*: $y^\star ; y = y ; y^\star$
 by (*smt add-least-upper-bound antisym less-eq-def mult-left-one mult-right-dist-add star.circ-plus-sub star-left-unfold star-right-induct*)

lemma *star-slide*: $(x ; y)^\star ; x = x ; (y ; x)^\star$
 by (*smt add-least-upper-bound antisym mult-associative mult-left-isotone mult-left-one mult-right-one order-trans star-left-slide star-left-unfold star-right-induct*)

lemma *star-simulation-right*: $z ; x \le y ; z \longrightarrow z ; x^\star \le y^\star ; z$
 by (*smt add-commutative add-least-upper-bound add-left-upper-bound mult-associative order-trans star.circ-loop-fixpoint star-left-induct star-plus star-right-induct*)

end

sublocale *strong-left-kleene-algebra* < *star*!: *itering-1* **where** *circ* = *star*
 apply *unfold-locales*
 apply (*metis star-slide order-refl*)
 apply (*metis star-simulation-right*)
 done

context *strong-left-kleene-algebra*

begin

lemma *star-right-induct-mult*: $y ; x \le y \longrightarrow y ; x^\star \le y$
 by (*metis add-least-upper-bound eq-refl star-right-induct*)

lemma *star-right-induct-mult-iff*: $y ; x \le y \longleftrightarrow y ; x^\star \le y$
 by (*metis mult-right-isotone order-trans star.circ-increasing star-right-induct-mult*)

lemma *star-simulation-right-equal*: $z ; x = y ; z \longrightarrow z ; x^\star = y^\star ; z$
 by (*metis eq-iff star-simulation-left star-simulation-right*)

lemma *star-simulation-star*: $x ; y \le y ; x \longrightarrow x^\star ; y^\star \le y^\star ; x^\star$
 by (*metis star-simulation-left star-simulation-right*)

lemma *star-right-induct-equal*: $z + y ; x = y \longrightarrow z ; x^\star \le y$
 by (*metis order-refl star-right-induct*)

lemma *star-right-induct-mult-equal*: $y ; x = y \longrightarrow y ; x^\star \le y$
 by (*metis order-refl star-right-induct-mult*)

lemma *star-back-loop-least-fixpoint*: $x ; y + z = x \longrightarrow z ; y^\star \le x$
 by (*metis add-commutative star-right-induct-equal*)

lemma *star-back-loop-is-least-fixpoint*: *is-least-fixpoint* $(\lambda x . x ; y + z)$ $(z ; y^\star)$
 by (*smt add-commutative add-right-isotone antisym is-least-fixpoint-def mult-left-isotone star.circ-back-loop-prefixpoint star-back-loop-least-fixpoint star-right-induct*)

lemma *star-back-loop-mu*: μ (λx . x ; y + z) = z ; $y^\star$
  by (*metis least-fixpoint-same star-back-loop-is-least-fixpoint*)

lemma *star-square*: $x^\star$ = (*1* + x) ; (x ; x)*
proof −
  let *?f* = λy . y ; x + *1*
  have *1*: *isotone ?f*
    by (*smt add-left-isotone isotone-def mult-left-isotone*)
  have *2*: *?f* ∘ *?f* = (λy . y ; (x ; x) + (*1* + x))
    by (*simp add*: *add-associative add-commutative mult-associative mult-left-one mult-right-dist-add o-def*)
  thus *?thesis* **using** *1*
    by (*metis mu-square mult-left-one star-back-loop-mu has-least-fixpoint-def star-back-loop-is-least-fixpoint*)
qed

lemma *star-square-2*: $x^\star$ = (x ; x)* ; (x + *1*)
  by (*smt add-commutative antisym mult-left-one mult-left-sub-dist-add mult-right-dist-add mult-right-one star.circ-square-2 star-slide star-square*)

lemma *star-circ-simulate-right-plus*: z ; $x \leq y$; $y^\star$; z + $w \longrightarrow z$; $x^\star \leq y^\star$; (z + w ; $x^\star$)
proof
  assume *1*: z ; $x \leq y$; $y^\star$; z + w
  have (z + w ; $x^\star$) ; $x \leq z$; x + w ; $x^\star$
    by (*metis add-right-isotone mult-associative mult-right-dist-add mult-right-isotone star.circ-increasing star.circ-transitive-equal*)
  also have ... $\leq y$; $y^\star$; z + w + w ; $x^\star$ **using** *1*
    by (*metis add-left-isotone*)
  also have ... $\leq y$; $y^\star$; z + w ; $x^\star$
    by (*metis add-least-upper-bound add-right-isotone add-right-upper-bound star.circ-back-loop-prefixpoint*)
  also have ... $\leq y^\star$; (z + w ; $x^\star$)
    by (*metis add-least-upper-bound mult-isotone mult-left-isotone mult-left-one mult-left-sub-dist-add-left star.circ-reflexive star.left-plus-below-circ*)
  finally have $y^\star$; (z + w ; $x^\star$) ; $x \leq y^\star$; (z + w ; $x^\star$)
    by (*metis mult-associative mult-right-isotone star.circ-transitive-equal*)
  thus z ; $x^\star \leq y^\star$; (z + w ; $x^\star$)
    by (*metis add-least-upper-bound star-right-induct mult-left-sub-dist-add-left star.circ-loop-fixpoint*)
qed

lemma *star-circ-simulate-left-plus*: x ; $z \leq z$; $y^\star$ + $w \longrightarrow x^\star$; $z \leq$ (z + $x^\star$; w) ; $y^\star$ **nitpick** [*expect=genuine,card=7*] **oops**

end

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

begin

lemma *star-star-absorb*: $y^\star$; ($y^\star$; x)* ; $y^\star$ = ($y^\star$; x)* ; $y^\star$
  by (*metis add-commutative mult-associative star.circ-decompose-4 star.circ-slide-1 star-decompose-1 star-decompose-3*)

lemma *star-circ-simulate-left-plus*: x ; $z \leq z$; $y^\star$ + $w \longrightarrow x^\star$; $z \leq$ (z + $x^\star$; w) ; $y^\star$
proof
  assume *1*: x ; $z \leq z$; $y^\star$ + w
  have x ; ((z + $x^\star$; w) ; $y^\star$) $\leq x$; z ; $y^\star$ + $x^\star$; w ; $y^\star$
    by (*smt add-right-isotone mult-associative mult-left-dist-add mult-right-dist-add mult-right-sub-dist-add-left star.circ-loop-fixpoint*)
  also have ... $\leq$ (z + w + $x^\star$; w) ; $y^\star$ **using** *1*
    by (*smt add-left-divisibility add-left-isotone mult-associative mult-right-dist-add star.circ-transitive-equal*)
  also have ... = (z + $x^\star$; w) ; $y^\star$
    by (*metis add-associative add-right-upper-bound less-eq-def star.circ-loop-fixpoint*)
  finally show $x^\star$; $z \leq$ (z + $x^\star$; w) ; $y^\star$
    by (*metis add-least-upper-bound mult-left-sub-dist-add-left mult-right-one star.circ-right-unfold-1 star-left-induct*)
qed

end

— Theorem 2.1

sublocale *left-zero-kleene-algebra* < *star*!: *itering* **where** *circ* = *star*
  apply *unfold-locales*
  apply (*metis star.circ-add-9*)

apply (*metis star.circ-mult-1*)
apply (*rule star-circ-simulate-right-plus*)
apply (*rule star-circ-simulate-left-plus*)
done

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

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

begin

lemma *star-below-circ*: $x^\star \le x^\circ$
by (*metis circ-left-unfold mult-right-one order-refl star-left-induct*)

lemma *star-zero-below-circ-mult*: $x^\star ; 0 \le x^\circ ; y$
by (*metis mult-isotone star-below-circ zero-least*)

lemma *star-mult-circ*: $x^\star ; x^\circ = x^\circ$
by (*metis add-right-divisibility antisym circ-left-unfold star-left-induct-mult star.circ-loop-fixpoint*)

lemma *circ-mult-star*: $x^\circ ; x^\star = x^\circ$
by (*metis add-associative add-least-upper-bound circ-left-unfold circ-rtc-2 eq-iff left-plus-circ star.circ-add-sub star.circ-back-loop-prefixpoint star.circ-increasing star-below-circ star-mult-circ star-sup-one*)

lemma *circ-star*: $x^{\circ\star} = x^\circ$
by (*metis circ-left-unfold left-plus-circ less-def less-le star.circ-increasing star-below-circ star-sup-one*)

lemma *star-circ*: $x^{\star\circ} = x^{\circ\circ}$
by (*metis antisym circ-circ-add circ-sub-dist less-eq-def star.circ-rtc-2 star-below-circ*)

lemma *circ-add-3*: $(x^\circ ; y^\circ)^\star \le (x + y)^\circ$
by (*metis circ-add-1 circ-isotone circ-left-unfold circ-star mult-left-sub-dist-add-left mult-right-isotone mult-right-one star.circ-isotone*)

end

class *left-zero-kleene-conway-semiring* = *left-zero-kleene-algebra* + *itering*

begin

subclass *left-kleene-conway-semiring* **..**

lemma *circ-isolate*: $x^\circ = x^\circ ; 0 + x^\star$
by (*metis add-commutative antisym circ-add-upper-bound circ-mult-star circ-simulate-absorb star.left-plus-below-circ star-below-circ zero-right-mult-decreasing*)

lemma *circ-isolate-mult*: $x^\circ ; y = x^\circ ; 0 + x^\star ; y$
by (*metis circ-isolate mult-associative mult-left-zero mult-right-dist-add*)

lemma *circ-isolate-mult-sub*: $x^\circ ; y \le x^\circ + x^\star ; y$
by (*metis add-left-isotone circ-isolate-mult zero-right-mult-decreasing*)

lemma *circ-sub-decompose*: $(x^\circ ; y)^\circ \le (x^\star ; y)^\circ ; x^\circ$
by (*smt add-commutative add-least-upper-bound add-right-upper-bound circ-back-loop-fixpoint circ-isolate-mult mult-zero-add-circ-2 zero-right-mult-decreasing*)

lemma *circ-add-4*: $(x + y)^\circ = (x^\star ; y)^\circ ; x^\circ$
apply (*rule antisym*)
apply (*smt circ-add circ-sub-decompose circ-transitive-equal mult-associative mult-left-isotone*)
apply (*smt circ-add circ-isotone mult-left-isotone star-below-circ*)
done

lemma *circ-add-5*: $(x^\circ ; y)^\circ ; x^\circ = (x^\star ; y)^\circ ; x^\circ$
by (*metis circ-add circ-add-4*)

lemma *plus-circ*: $(x^\star ; x)^\circ = x^\circ$
by (*smt add-idempotent circ-add-4 circ-decompose-7 circ-star star.circ-decompose-5 star.right-plus-circ*)

lemma $(x^\star ; y ; x^\star)^\circ = (x^\star ; y)^\circ$ **nitpick** [*expect=genuine*] **oops**

end

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

sublocale *bounded-left-kleene-algebra* < *star*!: *bounded-left-conway-semiring* **where** *circc* = *star* **..**

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

sublocale *bounded-left-zero-kleene-algebra* < *star*!: *bounded-itering* **where** *circ* = *star* **..**

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

sublocale *bounded-kleene-algebra* < *star*!: *bounded-itering* **where** *circ* = *star* **..**

end

7 OmegaAlgebra

theory *OmegaAlgebra*

imports *KleeneAlgebra*

begin

class *omega* =
  fixes *omega* :: $'a \Rightarrow 'a$ (-$^\omega$ [*100*] *100*)

class *left-omega-algebra* = *left-kleene-algebra* + *omega* +
  assumes *omega-unfold*: $y^\omega = y \mathbin{;} y^\omega$
  assumes *omega-induct*: $x \le z + y \mathbin{;} x \longrightarrow x \le y^\omega + y^\star \mathbin{;} z$

begin

— Many lemmas in this class are taken from Georg Struth's Isabelle theories.

lemma *star-zero-below-omega*: $x^\star \mathbin{;} 0 \le x^\omega$
  by (*metis add-left-zero omega-unfold star-left-induct-equal*)

lemma *star-zero-below-omega-zero*: $x^\star \mathbin{;} 0 \le x^\omega \mathbin{;} 0$
  by (*metis add-left-zero mult-associative omega-unfold star-left-induct-equal*)

lemma *omega-induct-mult*: $y \le x \mathbin{;} y \longrightarrow y \le x^\omega$
  by (*metis add-commutative add-left-zero less-eq-def omega-induct star-zero-below-omega*)

lemma *omega-sub-dist*: $x^\omega \le (x+y)^\omega$
  by (*metis mult-right-sub-dist-add-left omega-induct-mult omega-unfold*)

lemma *omega-isotone*: $x \le y \longrightarrow x^\omega \le y^\omega$
  by (*metis less-eq-def omega-sub-dist*)

lemma *omega-induct-equal*: $y = z + x \mathbin{;} y \longrightarrow y \le x^\omega + x^\star \mathbin{;} z$
  by (*metis omega-induct order-refl*)

lemma *omega-zero*: $0^\omega = 0$
  by (*metis mult-left-zero omega-unfold*)

lemma *omega-one-greatest*: $x \le 1^\omega$
  by (*metis mult-left-one omega-induct-mult order-refl*)

lemma *star-mult-omega*: $x^\star \mathbin{;} x^\omega = x^\omega$
  by (*metis antisym-conv mult-isotone omega-unfold star.circ-increasing star-left-induct-mult-equal star-left-induct-mult-iff*)

lemma *omega-sub-vector*: $x^\omega \mathbin{;} y \le x^\omega$
  by (*metis mult-associative omega-induct-mult omega-unfold order-refl*)

lemma *omega-simulation*: $z \mathbin{;} x \le y \mathbin{;} z \longrightarrow z \mathbin{;} x^\omega \le y^\omega$
  by (*smt less-eq-def mult-associative mult-right-sub-dist-add-left omega-induct-mult omega-unfold*)

lemma *omega-omega*: $x^{\omega\omega} \le x^\omega$
  by (*metis omega-sub-vector omega-unfold*)

lemma *left-plus-omega*: $(x \mathbin{;} x^\star)^\omega = x^\omega$
  by (*metis antisym mult-associative omega-induct-mult omega-unfold order-refl star.left-plus-circ star-mult-omega*)

lemma *omega-slide*: $x \mathbin{;} (y \mathbin{;} x)^\omega = (x \mathbin{;} y)^\omega$
  by (*metis antisym mult-associative mult-right-isotone omega-simulation omega-unfold order-refl*)

lemma *omega-simulation-2*: $y \mathbin{;} x \le x \mathbin{;} y \longrightarrow (x \mathbin{;} y)^\omega \le x^\omega$
  by (*metis less-eq-def mult-right-isotone omega-induct-mult omega-slide omega-sub-dist*)

lemma *wagner*: $(x + y)^\omega = x \mathbin{;} (x + y)^\omega + z \longrightarrow (x + y)^\omega = x^\omega + x^\star \mathbin{;} z$
  by (*metis add-commutative add-least-upper-bound eq-iff omega-induct omega-sub-dist star-left-induct*)

lemma *right-plus-omega*: $(x^\star \mathbin{;} x)^\omega = x^\omega$
  by (*metis left-plus-omega omega-slide star-mult-omega*)

lemma *omega-sub-dist-1*: $(x ; y^\star)^\omega \leq (x + y)^\omega$
by (*metis add-least-upper-bound left-plus-omega mult-isotone mult-left-one mult-right-dist-add omega-isotone order-refl star-decompose-1 star.circ-increasing star.circ-plus-one*)

lemma *omega-sub-dist-2*: $(x^\star ; y)^\omega \leq (x + y)^\omega$
by (*metis add-commutative mult-isotone omega-slide omega-sub-dist-1 star-mult-omega star.circ-sub-dist*)

lemma *omega-star*: $(x^\omega)^\star = 1 + x^\omega$
by (*metis eq-iff mult-left-sub-dist-add-left mult-right-one omega-sub-vector star.circ-left-unfold*)

lemma *omega-mult-omega-star*: $x^\omega ; x^{\omega\star} = x^\omega$
by (*metis add-least-upper-bound antisym omega-sub-vector star.circ-back-loop-prefixpoint*)

lemma *omega-sum-unfold-1*: $(x + y)^\omega = x^\omega + x^\star ; y ; (x + y)^\omega$
by (*metis mult-associative mult-right-dist-add omega-unfold wagner*)

lemma *omega-sum-unfold-2*: $(x + y)^\omega \leq (x^\star ; y)^\omega + (x^\star ; y)^\star ; x^\omega$
by (*metis omega-induct-equal omega-sum-unfold-1*)

lemma *omega-sum-unfold-3*: $(x^\star ; y)^\star ; x^\omega \leq (x + y)^\omega$
by (*metis omega-sum-unfold-1 star-left-induct-equal*)

lemma *omega-decompose*: $(x + y)^\omega = (x^\star ; y)^\omega + (x^\star ; y)^\star ; x^\omega$
by (*metis add-least-upper-bound antisym omega-sub-dist-2 omega-sum-unfold-2 omega-sum-unfold-3*)

lemma *omega-loop-fixpoint*: $y ; (y^\omega + y^\star ; z) + z = y^\omega + y^\star ; z$
apply (*rule antisym*)
apply (*smt add-commutative add-least-upper-bound add-right-isotone add-right-upper-bound mult-left-sub-dist-add-left mult-right-isotone omega-induct omega-unfold order-trans star.circ-loop-fixpoint*)
apply (*smt add-associative add-left-isotone mult-left-sub-dist-add omega-unfold star.circ-loop-fixpoint*)
done

lemma *omega-loop-greatest-fixpoint*: $y ; x + z = x \longrightarrow x \leq y^\omega + y^\star ; z$
by (*metis add-commutative omega-induct-equal*)

lemma *omega-square*: $x^\omega = (x ; x)^\omega$
by (*metis antisym mult-associative omega-induct-mult omega-mult-omega-star omega-slide omega-sub-vector omega-unfold*)

lemma *mult-zero-omega*: $(x ; 0)^\omega = x ; 0$
by (*metis mult-left-zero omega-slide*)

lemma *mult-zero-add-omega*: $(x + y ; 0)^\omega = x^\omega + x^\star ; y ; 0$
by (*smt add-associative add-commutative add-idempotent mult-associative mult-left-one mult-left-zero mult-right-dist-add mult-zero-omega star.mult-zero-circ omega-decompose*)

lemma *omega-mult-star*: $x^\omega ; x^\star = x^\omega$
by (*metis antisym mult-left-sub-dist-add-left mult-right-one omega-sub-vector star.circ-plus-one*)

lemma *omega-loop-is-greatest-fixpoint*: *is-greatest-fixpoint* $(\lambda x . y ; x + z) (y^\omega + y^\star ; z)$
by (*smt is-greatest-fixpoint-def omega-loop-fixpoint omega-loop-greatest-fixpoint*)

lemma *omega-loop-nu*: $\nu (\lambda x . y ; x + z) = y^\omega + y^\star ; z$
by (*metis greatest-fixpoint-same omega-loop-is-greatest-fixpoint*)

lemma *omega-loop-zero-is-greatest-fixpoint*: *is-greatest-fixpoint* $(\lambda x . y ; x) (y^\omega)$
by (*metis is-greatest-fixpoint-def omega-induct-mult omega-unfold order-refl*)

lemma *omega-loop-zero-nu*: $\nu (\lambda x . y ; x) = y^\omega$
by (*metis greatest-fixpoint-same omega-loop-zero-is-greatest-fixpoint*)

lemma *affine-has-greatest-fixpoint*: *has-greatest-fixpoint* $(\lambda x . y ; x + z)$
by (*metis has-greatest-fixpoint-def omega-loop-is-greatest-fixpoint*)

lemma *omega-separate-unfold*: $(x^\star ; y)^\omega = y^\omega + y^\star ; x ; (x^\star ; y)^\omega$
by (*metis add-commutative mult-associative omega-slide omega-sum-unfold-1 star.circ-loop-fixpoint*)

lemma *omega-zero-left-slide*: $(x ; y)^\star ; ((x ; y)^\omega ; 0 + 1) ; x \leq x ; (y ; x)^\star ; ((y ; x)^\omega ; 0 + 1)$
proof −
have $x + x ; (y ; x) ; (y ; x)^\star ; ((y ; x)^\omega ; 0 + 1) \leq x ; (y ; x)^\star ; ((y ; x)^\omega ; 0 + 1)$

by (*smt add-commutative add-least-upper-bound mult-associative mult-left-isotone mult-right-isotone star.circ-back-loop-prefixpoint star.left-plus-below-circ star.mult-zero-add-circ star.mult-zero-circ*)
hence $((x ; y)^\omega ; 0 + 1) ; x + x ; y ; (x ; (y ; x)^\star ; ((y ; x)^\omega ; 0 + 1)) \le x ; (y ; x)^\star ; ((y ; x)^\omega ; 0 + 1)$
by (*smt add-associative less-eq-def mult-associative mult-left-one mult-left-sub-dist-add-left mult-left-zero mult-right-dist-add omega-slide star-mult-omega*)
thus *?thesis*
by (*metis mult-associative star-left-induct*)
qed

lemma *omega-zero-add-1*: $(x + y)^\star ; ((x + y)^\omega ; 0 + 1) = x^\star ; (x^\omega ; 0 + 1) ; (y ; x^\star ; (x^\omega ; 0 + 1))^\star ; ((y ; x^\star ; (x^\omega ; 0 + 1))^\omega ; 0 + 1)$
proof (*rule antisym*)
have *1*: $(x + y) ; x^\star ; (x^\omega ; 0 + 1) ; (y ; x^\star ; (x^\omega ; 0 + 1))^\star ; ((y ; x^\star ; (x^\omega ; 0 + 1))^\omega ; 0 + 1) \le x^\star ; (x^\omega ; 0 + 1) ; (y ; x^\star ; (x^\omega ; 0 + 1))^\star ; ((y ; x^\star ; (x^\omega ; 0 + 1))^\omega ; 0 + 1)$
by (*smt add-associative add-commutative less-eq-def mult-associative mult-left-isotone mult-right-dist-add star.circ-add-1 star.left-plus-below-circ star.mult-zero-add-circ star.mult-zero-circ star-decompose-1*)
have *2*: $1 \le x^\star ; (x^\omega ; 0 + 1) ; (y ; x^\star ; (x^\omega ; 0 + 1))^\star ; ((y ; x^\star ; (x^\omega ; 0 + 1))^\omega ; 0 + 1)$
by (*smt add-commutative mult-associative star.circ-add-1 star.circ-reflexive star.mult-zero-add-circ star.mult-zero-circ*)
have $(y ; x^\star)^\omega ; 0 \le (y ; x^\star ; (x^\omega ; 0 + 1))^\omega ; 0$
by (*smt mult-left-isotone mult-left-sub-dist-add-right mult-right-one omega-isotone*)
also have *3*: ... $\le (x^\omega ; 0 + 1) ; (y ; x^\star ; (x^\omega ; 0 + 1))^\star ; ((y ; x^\star ; (x^\omega ; 0 + 1))^\omega ; 0 + 1)$
by (*smt add-commutative mult-associative mult-left-one mult-right-sub-dist-add-left order-trans star.circ-sub-dist-1 star.mult-zero-add-circ star.mult-zero-circ*)
finally have *4*: $(x^\star ; y)^\omega ; 0 \le x^\star ; (x^\omega ; 0 + 1) ; (y ; x^\star ; (x^\omega ; 0 + 1))^\star ; ((y ; x^\star ; (x^\omega ; 0 + 1))^\omega ; 0 + 1)$
by (*smt mult-associative mult-right-isotone omega-slide*)
have $y ; (x^\star ; y)^\star ; x^\omega ; 0 \le y ; (x^\star ; (x^\omega ; 0 + y))^\star ; x^\star ; x^\omega ; 0 ; (y ; x^\star ; (x^\omega ; 0 + 1))^\omega ; 0$
by (*metis mult-left-isotone mult-left-sub-dist-add-right mult-right-isotone star.circ-isotone mult-associative mult-left-zero star-mult-omega*)
also have ... $\le y ; (x^\star ; (x^\omega ; 0 + y))^\star ; (x^\star ; (x^\omega ; 0 + 1) ; y)^\omega ; 0$
by (*smt mult-associative mult-left-isotone mult-left-sub-dist-add-left omega-slide*)
also have ... $= y ; (x^\star ; (x^\omega ; 0 + 1) ; y)^\omega ; 0$
by (*smt mult-associative mult-left-one mult-left-zero mult-right-dist-add star-mult-omega*)
finally have $x^\star ; y ; (x^\star ; y)^\star ; x^\omega ; 0 \le x^\star ; (x^\omega ; 0 + 1) ; (y ; x^\star ; (x^\omega ; 0 + 1))^\star ; ((y ; x^\star ; (x^\omega ; 0 + 1))^\omega ; 0 + 1)$ **using** *3*
by (*smt mult-associative mult-right-isotone omega-slide order-trans*)
hence $(x^\star ; y)^\star ; x^\omega ; 0 \le x^\star ; (x^\omega ; 0 + 1) ; (y ; x^\star ; (x^\omega ; 0 + 1))^\star ; ((y ; x^\star ; (x^\omega ; 0 + 1))^\omega ; 0 + 1)$
by (*smt add-associative add-commutative less-eq-def mult-associative mult-isotone mult-left-one mult-right-one mult-right-sub-dist-add-left order-trans star.circ-loop-fixpoint star.circ-reflexive star.mult-zero-circ*)
hence $(x + y)^\omega ; 0 \le x^\star ; (x^\omega ; 0 + 1) ; (y ; x^\star ; (x^\omega ; 0 + 1))^\star ; ((y ; x^\star ; (x^\omega ; 0 + 1))^\omega ; 0 + 1)$ **using** *4*
by (*metis add-least-upper-bound mult-right-dist-add omega-decompose*)
thus $(x + y)^\star ; ((x + y)^\omega ; 0 + 1) \le x^\star ; (x^\omega ; 0 + 1) ; (y ; x^\star ; (x^\omega ; 0 + 1))^\star ; ((y ; x^\star ; (x^\omega ; 0 + 1))^\omega ; 0 + 1)$ **using** *1 2*
by (*smt add-least-upper-bound mult-associative star-left-induct*)
next
have *5*: $x^\omega ; 0 \le (x + y)^\star ; ((x + y)^\omega ; 0 + 1)$
by (*metis add-commutative add-left-zero mult-associative mult-left-isotone mult-left-one mult-right-dist-add omega-sub-dist order-trans star-mult-omega zero-right-mult-decreasing*)
have *6*: $(y ; x^\star)^\omega ; 0 \le (x + y)^\star ; ((x + y)^\omega ; 0 + 1)$
by (*metis add-commutative mult-left-isotone omega-sub-dist-1 mult-associative mult-left-sub-dist-add-left order-trans star-mult-omega*)
have *7*: $(y ; x^\star)^\star \le (x + y)^\star$
by (*metis mult-left-one mult-right-sub-dist-add-left star.circ-add-1 star.circ-plus-one*)
hence $(y ; x^\star)^\star ; x^\omega ; 0 \le (x + y)^\star ; ((x + y)^\omega ; 0 + 1)$
by (*smt add-associative less-eq-def mult-associative mult-isotone mult-right-dist-add omega-sub-dist*)
hence $(x^\omega ; 0 + y ; x^\star)^\omega ; 0 \le (x + y)^\star ; ((x + y)^\omega ; 0 + 1)$ **using** *6*
by (*smt add-commutative add-least-upper-bound mult-associative mult-right-dist-add mult-zero-add-omega omega-unfold omega-zero*)
hence $(y ; x^\star ; (x^\omega ; 0 + 1))^\omega ; 0 \le y ; x^\star ; (x + y)^\star ; ((x + y)^\omega ; 0 + 1)$
by (*smt mult-associative mult-left-one mult-left-zero mult-right-dist-add mult-right-isotone omega-slide*)
also have ... $\le (x + y)^\star ; ((x + y)^\omega ; 0 + 1)$ **using** *7*
by (*metis mult-left-isotone order-refl star.circ-mult-upper-bound star-left-induct-mult-iff*)
finally have $(y ; x^\star ; (x^\omega ; 0 + 1))^\star ; ((y ; x^\star ; (x^\omega ; 0 + 1))^\omega ; 0 + 1) \le (x + y)^\star ; ((x + y)^\omega ; 0 + 1)$ **using** *5*
by (*smt add-commutative add-least-upper-bound mult-associative order-refl star.circ-mult-upper-bound star.circ-reflexive star.circ-sub-dist-1 star.mult-zero-add-circ star.mult-zero-circ star-left-induct*)
hence $(x^\omega ; 0 + 1) ; (y ; x^\star ; (x^\omega ; 0 + 1))^\star ; ((y ; x^\star ; (x^\omega ; 0 + 1))^\omega ; 0 + 1) \le (x + y)^\star ; ((x + y)^\omega ; 0 + 1)$ **using** *5*
by (*metis add-commutative mult-associative star.circ-isotone star.circ-mult-upper-bound star.mult-zero-add-circ star.mult-zero-circ star-involutive*)
thus $x^\star ; (x^\omega ; 0 + 1) ; (y ; x^\star ; (x^\omega ; 0 + 1))^\star ; ((y ; x^\star ; (x^\omega ; 0 + 1))^\omega ; 0 + 1) \le (x + y)^\star ; ((x + y)^\omega ; 0 + 1)$
by (*smt add-associative add-commutative mult-associative star.circ-mult-upper-bound star.circ-sub-dist*

star.mult-zero-add-circ star.mult-zero-circ)
qed

lemma *star-omega-greatest*: $x^{\star\omega} = 1^\omega$
  by (*metis add-commutative less-eq-def omega-one-greatest omega-sub-dist star.circ-plus-one*)

lemma *omega-vector-greatest*: $x^\omega \ ; \ 1^\omega = x^\omega$
  by (*metis antisym mult-isotone omega-mult-omega-star omega-one-greatest omega-sub-vector*)

lemma *mult-greatest-omega*: $(x \ ; \ 1^\omega)^\omega \leq x \ ; \ 1^\omega$
  by (*metis mult-right-isotone omega-slide omega-sub-vector*)

lemma *omega-mult-star-2*: $x^\omega \ ; \ y^\star = x^\omega$
  by (*metis mult-associative omega-mult-star omega-vector-greatest star-involutive star-omega-greatest*)

lemma *omega-import*: $p \leq p \ ; \ p \wedge p \ ; \ x \leq x \ ; \ p \longrightarrow p \ ; \ x^\omega = p \ ; \ (p \ ; \ x)^\omega$
proof
  assume *1*: $p \leq p \ ; \ p \wedge p \ ; \ x \leq x \ ; \ p$
  hence $p \ ; \ x^\omega \leq p \ ; \ (p \ ; \ x) \ ; \ x^\omega$
    by (*metis mult-associative mult-left-isotone omega-unfold*)
  also have ... $\leq p \ ; \ x \ ; \ p \ ; \ x^\omega$ **using** *1*
    by (*metis mult-associative mult-left-isotone mult-right-isotone*)
  finally have $p \ ; \ x^\omega \leq (p \ ; \ x)^\omega$
    by (*metis mult-associative omega-induct-mult*)
  hence $p \ ; \ x^\omega \leq p \ ; \ (p \ ; \ x)^\omega$ **using** *1*
    by (*metis mult-associative mult-left-isotone mult-right-isotone order-trans*)
  thus $p \ ; \ x^\omega = p \ ; \ (p \ ; \ x)^\omega$ **using** *1*
    by (*metis add-left-divisibility antisym mult-right-isotone omega-induct-mult omega-slide omega-sub-dist*)
qed

lemma *omega-circ-simulate-right-plus*: $z \ ; \ x \leq y \ ; \ (y^\omega \ ; \ 0 + y^\star) \ ; \ z + w \longrightarrow z \ ; \ (x^\omega \ ; \ 0 + x^\star) \leq (y^\omega \ ; \ 0 + y^\star) \ ; \ (z + w \ ; \ (x^\omega \ ; \ 0 + x^\star))$ **nitpick** [*expect=genuine*] **oops**
lemma *omega-circ-simulate-left-plus*: $x \ ; \ z \leq z \ ; \ (y^\omega \ ; \ 0 + y^\star) + w \longrightarrow (x^\omega \ ; \ 0 + x^\star) \ ; \ z \leq (z + (x^\omega \ ; \ 0 + x^\star) \ ; \ w) \ ; \ (y^\omega \ ; \ 0 + y^\star)$ **nitpick** [*expect=genuine*] **oops**

end

— Theorem 50.2

sublocale *left-omega-algebra* < *comb0*!: *left-conway-semiring* **where** *circ* = $(\lambda x \ . \ x^\star \ ; \ (x^\omega \ ; \ 0 + 1))$
  apply *unfold-locales*
    apply (*smt add-associative add-commutative less-eq-def mult-associative mult-left-sub-dist-add-left omega-unfold star.circ-loop-fixpoint star-mult-omega*)
  apply (*smt mult-associative omega-zero-left-slide*)
  apply (*smt mult-associative omega-zero-add-1*)
  done

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

begin

lemma *star-omega-absorb*: $y^\star \ ; \ (y^\star \ ; \ x)^\star \ ; \ y^\omega = (y^\star \ ; \ x)^\star \ ; \ y^\omega$
proof −
  have $y^\star \ ; \ (y^\star \ ; \ x)^\star \ ; \ y^\omega = y^\star \ ; \ y^\star \ ; \ x \ ; \ (y^\star \ ; \ x)^\star \ ; \ y^\omega + y^\star \ ; \ y^\omega$
    by (*metis add-commutative mult-associative mult-right-dist-add star.circ-back-loop-fixpoint star.circ-plus-same*)
  thus *?thesis*
    by (*metis mult-associative star.circ-loop-fixpoint star.circ-transitive-equal star-mult-omega*)
qed

lemma *omega-circ-simulate-right-plus*: $z \ ; \ x \leq y \ ; \ (y^\omega \ ; \ 0 + y^\star) \ ; \ z + w \longrightarrow z \ ; \ (x^\omega \ ; \ 0 + x^\star) \leq (y^\omega \ ; \ 0 + y^\star) \ ; \ (z + w \ ; \ (x^\omega \ ; \ 0 + x^\star))$
proof
  assume $z \ ; \ x \leq y \ ; \ (y^\omega \ ; \ 0 + y^\star) \ ; \ z + w$
  hence *1*: $z \ ; \ x \leq y^\omega \ ; \ 0 + y \ ; \ y^\star \ ; \ z + w$
    by (*metis mult-associative mult-left-dist-add mult-left-zero mult-right-dist-add omega-unfold*)
  hence $(y^\omega \ ; \ 0 + y^\star \ ; \ z + y^\star \ ; \ w \ ; \ x^\omega \ ; \ 0 + y^\star \ ; \ w \ ; \ x^\star) \ ; \ x \leq y^\omega \ ; \ 0 + y^\star \ ; \ (y^\omega \ ; \ 0 + y \ ; \ y^\star \ ; \ z + w) + y^\star \ ; \ w \ ; \ x^\omega \ ; \ 0 + y^\star \ ; \ w \ ; \ x^\star$
      by (*smt add-associative add-left-upper-bound add-right-upper-bound less-eq-def mult-associative mult-left-dist-add mult-left-zero mult-right-dist-add star.circ-back-loop-fixpoint*)

also have ... $= y^\omega \mathbin{;} 0 + y^\star \mathbin{;} y \mathbin{;} y^\star \mathbin{;} z + y^\star \mathbin{;} w \mathbin{;} x^\omega \mathbin{;} 0 + y^\star \mathbin{;} w \mathbin{;} x^\star$
by (*smt add-associative add-right-upper-bound less-eq-def mult-associative mult-left-dist-add star.circ-back-loop-fixpoint star-mult-omega*)
also have ... $\le y^\omega \mathbin{;} 0 + y^\star \mathbin{;} z + y^\star \mathbin{;} w \mathbin{;} x^\omega \mathbin{;} 0 + y^\star \mathbin{;} w \mathbin{;} x^\star$
by (*smt add-commutative add-left-isotone mult-left-isotone star.circ-increasing star.circ-plus-same star.circ-transitive-equal*)
finally have $z + (y^\omega \mathbin{;} 0 + y^\star \mathbin{;} z + y^\star \mathbin{;} w \mathbin{;} x^\omega \mathbin{;} 0 + y^\star \mathbin{;} w \mathbin{;} x^\star) \mathbin{;} x \le y^\omega \mathbin{;} 0 + y^\star \mathbin{;} z + y^\star \mathbin{;} w \mathbin{;} x^\omega \mathbin{;} 0 + y^\star \mathbin{;} w \mathbin{;} x^\star$
by (*smt add-least-upper-bound add-left-upper-bound star.circ-loop-fixpoint*)
hence *2*: $z \mathbin{;} x^\star \le y^\omega \mathbin{;} 0 + y^\star \mathbin{;} z + y^\star \mathbin{;} w \mathbin{;} x^\omega \mathbin{;} 0 + y^\star \mathbin{;} w \mathbin{;} x^\star$
by (*metis star-right-induct*)
have $z \mathbin{;} x^\omega \mathbin{;} 0 \le (y^\omega \mathbin{;} 0 + y \mathbin{;} y^\star \mathbin{;} z + w) \mathbin{;} x^\omega \mathbin{;} 0$ **using** *1*
by (*smt add-left-divisibility mult-associative mult-right-sub-dist-add-left omega-unfold*)
hence $z \mathbin{;} x^\omega \mathbin{;} 0 \le y^\omega + y^\star \mathbin{;} (y^\omega \mathbin{;} 0 + w \mathbin{;} x^\omega \mathbin{;} 0)$
by (*smt add-associative add-commutative left-plus-omega mult-associative mult-left-zero mult-right-dist-add omega-induct star.left-plus-circ*)
thus $z \mathbin{;} (x^\omega \mathbin{;} 0 + x^\star) \le (y^\omega \mathbin{;} 0 + y^\star) \mathbin{;} (z + w \mathbin{;} (x^\omega \mathbin{;} 0 + x^\star))$ **using** *2*
by (*smt add-associative add-commutative less-eq-def mult-associative mult-left-dist-add mult-left-zero mult-right-dist-add omega-unfold omega-zero star-mult-omega zero-right-mult-decreasing*)
qed

lemma *omega-circ-simulate-left-plus*: $x \mathbin{;} z \le z \mathbin{;} (y^\omega \mathbin{;} 0 + y^\star) + w \longrightarrow (x^\omega \mathbin{;} 0 + x^\star) \mathbin{;} z \le (z + (x^\omega \mathbin{;} 0 + x^\star) \mathbin{;} w) \mathbin{;} (y^\omega \mathbin{;} 0 + y^\star)$
proof
assume *1*: $x \mathbin{;} z \le z \mathbin{;} (y^\omega \mathbin{;} 0 + y^\star) + w$
have $x \mathbin{;} (z \mathbin{;} y^\omega \mathbin{;} 0 + z \mathbin{;} y^\star + x^\omega \mathbin{;} 0 + x^\star \mathbin{;} w \mathbin{;} y^\omega \mathbin{;} 0 + x^\star \mathbin{;} w \mathbin{;} y^\star) = x \mathbin{;} z \mathbin{;} y^\omega \mathbin{;} 0 + x \mathbin{;} z \mathbin{;} y^\star + x^\omega \mathbin{;} 0 + x \mathbin{;} x^\star \mathbin{;} w \mathbin{;} y^\omega \mathbin{;} 0 + x \mathbin{;} x^\star \mathbin{;} w \mathbin{;} y^\star$
by (*smt mult-associative mult-left-dist-add omega-unfold*)
also have ... $\le x \mathbin{;} z \mathbin{;} y^\omega \mathbin{;} 0 + x \mathbin{;} z \mathbin{;} y^\star + x^\omega \mathbin{;} 0 + x^\star \mathbin{;} w \mathbin{;} y^\omega \mathbin{;} 0 + x^\star \mathbin{;} w \mathbin{;} y^\star$
by (*metis add-isotone add-right-isotone mult-left-isotone star.left-plus-below-circ*)
also have ... $\le (z \mathbin{;} y^\omega \mathbin{;} 0 + z \mathbin{;} y^\star + w) \mathbin{;} y^\omega \mathbin{;} 0 + (z \mathbin{;} y^\omega \mathbin{;} 0 + z \mathbin{;} y^\star + w) \mathbin{;} y^\star + x^\omega \mathbin{;} 0 + x^\star \mathbin{;} w \mathbin{;} y^\omega \mathbin{;} 0 + x^\star \mathbin{;} w \mathbin{;} y^\star$ **using** *1*
by (*metis add-left-isotone mult-associative mult-left-dist-add mult-left-isotone*)
also have ... $= z \mathbin{;} y^\omega \mathbin{;} 0 + z \mathbin{;} y^\star \mathbin{;} y^\omega \mathbin{;} 0 + w \mathbin{;} y^\omega \mathbin{;} 0 + z \mathbin{;} y^\omega \mathbin{;} 0 + z \mathbin{;} y^\star \mathbin{;} y^\star + w \mathbin{;} y^\star + x^\omega \mathbin{;} 0 + x^\star \mathbin{;} w \mathbin{;} y^\omega \mathbin{;} 0 + x^\star \mathbin{;} w \mathbin{;} y^\star$
by (*smt add-associative mult-associative mult-left-zero mult-right-dist-add*)
also have ... $= z \mathbin{;} y^\omega \mathbin{;} 0 + z \mathbin{;} y^\star + x^\omega \mathbin{;} 0 + x^\star \mathbin{;} w \mathbin{;} y^\omega \mathbin{;} 0 + x^\star \mathbin{;} w \mathbin{;} y^\star$
by (*smt add-associative add-commutative add-idempotent mult-associative mult-right-dist-add star.circ-loop-fixpoint star.circ-transitive-equal star-mult-omega*)
finally have $(x^\omega \mathbin{;} 0 + x^\star) \mathbin{;} z \le z \mathbin{;} y^\omega \mathbin{;} 0 + z \mathbin{;} y^\star + x^\omega \mathbin{;} 0 + x^\star \mathbin{;} w \mathbin{;} y^\omega \mathbin{;} 0 + x^\star \mathbin{;} w \mathbin{;} y^\star$
by (*smt add-least-upper-bound add-left-upper-bound mult-associative mult-left-zero mult-right-dist-add star.circ-back-loop-fixpoint star-left-induct*)
thus $(x^\omega \mathbin{;} 0 + x^\star) \mathbin{;} z \le (z + (x^\omega \mathbin{;} 0 + x^\star) \mathbin{;} w) \mathbin{;} (y^\omega \mathbin{;} 0 + y^\star)$
by (*smt add-associative mult-associative mult-left-dist-add mult-left-zero mult-right-dist-add*)
qed

lemma *omega-translate*: $x^\star \mathbin{;} (x^\omega \mathbin{;} 0 + 1) = x^\omega \mathbin{;} 0 + x^\star$
by (*metis mult-associative mult-left-dist-add mult-right-one star-mult-omega*)

lemma *omega-circ-simulate-right*: $z \mathbin{;} x \le y \mathbin{;} z + w \longrightarrow z \mathbin{;} (x^\omega \mathbin{;} 0 + x^\star) \le (y^\omega \mathbin{;} 0 + y^\star) \mathbin{;} (z + w \mathbin{;} (x^\omega \mathbin{;} 0 + x^\star))$
proof
assume $z \mathbin{;} x \le y \mathbin{;} z + w$
also have ... $\le y \mathbin{;} (y^\omega \mathbin{;} 0 + y^\star) \mathbin{;} z + w$
by (*metis add-left-isotone comb0.circ-reflexive mult-left-isotone mult-right-isotone mult-right-one omega-translate*)
finally show $z \mathbin{;} (x^\omega \mathbin{;} 0 + x^\star) \le (y^\omega \mathbin{;} 0 + y^\star) \mathbin{;} (z + w \mathbin{;} (x^\omega \mathbin{;} 0 + x^\star))$
by (*metis omega-circ-simulate-right-plus*)
qed

end

sublocale *left-zero-omega-algebra* < *comb1*!: *left-conway-semiring-1* **where** $circ = (\lambda x \,.\, x^\star \mathbin{;} (x^\omega \mathbin{;} 0 + 1))$
apply *unfold-locales*
apply (*smt eq-iff mult-associative mult-left-dist-add mult-left-zero mult-right-dist-add mult-right-one omega-slide star-slide*)
done

sublocale *left-zero-omega-algebra* < *comb0*!: *itering* **where** $circ = (\lambda x \,.\, x^\star \mathbin{;} (x^\omega \mathbin{;} 0 + 1))$
apply *unfold-locales*
apply (*metis comb1.circ-add-9*)
apply (*metis comb1.circ-mult-1*)
apply (*metis omega-circ-simulate-right-plus omega-translate*)
apply (*metis omega-circ-simulate-left-plus omega-translate*)

done

— Theorem 2.2

sublocale *left-zero-omega-algebra* < *comb2*!: *itering* **where** *circ* = $(\lambda x\ .\ x^{\omega}\ ;\ 0 + x^{\star})$
apply *unfold-locales*
apply (*metis comb1.circ-add-9 omega-translate*)
apply (*metis comb1.circ-mult-1 omega-translate*)
apply (*metis omega-circ-simulate-right-plus*)
apply (*metis omega-circ-simulate-left-plus*)
done

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

class *left-omega-conway-semiring* = *left-omega-algebra* + *left-conway-semiring*

begin

subclass *left-kleene-conway-semiring* **..**

lemma *circ-below-omega-star*: $x^{\circ} \le x^{\omega} + x^{\star}$
by (*metis circ-left-unfold mult-right-one omega-induct order-refl*)

lemma *omega-mult-circ*: $x^{\omega}\ ;\ x^{\circ} = x^{\omega}$
by (*metis circ-star mult-associative omega-mult-star omega-vector-greatest star-omega-greatest*)

lemma *circ-mult-omega*: $x^{\circ}\ ;\ x^{\omega} = x^{\omega}$
by (*metis antisym add-right-divisibility circ-loop-fixpoint circ-plus-sub omega-simulation*)

lemma *circ-omega-greatest*: $x^{\circ\omega} = 1^{\omega}$
by (*metis circ-star star-omega-greatest*)

lemma *omega-circ*: $x^{\omega\circ} = 1 + x^{\omega}$
by (*metis antisym circ-left-unfold mult-left-sub-dist-add-left mult-right-one omega-sub-vector*)

end

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

begin

lemma *omega-one*: $1^{\omega} = T$
by (*smt add-left-top less-eq-def omega-one-greatest*)

lemma *star-omega-top*: $x^{\star\omega} = T$
by (*metis add-left-top less-eq-def omega-one omega-sub-dist star.circ-plus-one*)

lemma *omega-vector*: $x^{\omega}\ ;\ T = x^{\omega}$
by (*metis add-commutative less-eq-def omega-sub-vector top-right-mult-increasing*)

lemma *mult-top-omega*: $(x\ ;\ T)^{\omega} \le x\ ;\ T$
by (*metis mult-right-isotone omega-slide top-greatest*)

end

sublocale *bounded-left-omega-algebra* < *comb0*!: *bounded-left-conway-semiring* **where** *circ* = $(\lambda x\ .\ x^{\star}\ ;\ (x^{\omega}\ ;\ 0 + 1))$ **..**

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

begin

subclass *bounded-left-omega-algebra* **..**

end

sublocale *bounded-left-zero-omega-algebra* < *comb0*!: *bounded-itering* **where** *circ* = $(\lambda x\ .\ x^{\star}\ ;\ (x^{\omega}\ ;\ 0 + 1))$ **..**

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

begin

subclass *bounded-left-zero-omega-algebra* **..**

end

class *bounded-left-omega-conway-semiring* = *bounded-left-omega-algebra* + *left-omega-conway-semiring*

begin

subclass *left-kleene-conway-semiring* **..**

subclass *bounded-left-conway-semiring* **..**

lemma *circ-omega*: $x^{\circ\omega} = T$
  by (*metis circ-star star-omega-top*)

end

class *top-left-omega-algebra* = *bounded-left-omega-algebra* +
  assumes *top-left-zero*: $T \ ; x = T$

begin

lemma *omega-translate-3*: $x^\star \ ; (x^\omega \ ; 0 + 1) = x^\star \ ; (x^\omega + 1)$
  by (*metis mult-associative omega-mult-star-2 star.circ-top-1 top-left-zero*)

end

— Theorem 50.2

sublocale *top-left-omega-algebra* < *comb4*!: *left-conway-semiring* **where** *circ* = $(\lambda x \ . \ x^\star \ ; (x^\omega + 1))$
  apply *unfold-locales*
  apply (*metis comb0.circ-left-unfold omega-translate-3*)
  apply (*metis comb0.circ-left-slide omega-translate-3*)
  apply (*metis comb0.circ-add-1 omega-translate-3*)
  done

class *top-left-zero-omega-algebra* = *bounded-left-zero-omega-algebra* +
  assumes *top-left-zero*: $T \ ; x = T$

begin

lemma *omega-translate-2*: $x^\omega \ ; 0 + x^\star = x^\omega + x^\star$
  by (*metis mult-associative omega-mult-star-2 star.circ-top top-left-zero*)

end

— Theorem 2.3

sublocale *top-left-zero-omega-algebra* < *comb3*!: *itering* **where** *circ* = $(\lambda x \ . \ x^\omega + x^\star)$
  apply *unfold-locales*
  apply (*metis comb2.circ-add-9 omega-translate-2*)
  apply (*metis comb2.circ-mult-1 omega-translate-2*)
  apply (*metis omega-circ-simulate-right-plus omega-translate-2*)
  apply (*metis omega-circ-simulate-left-plus omega-translate-2*)
  done

class *Omega* =
  fixes *Omega* :: $'a \Rightarrow 'a$ (-^{Ω} [*100*] *100*)

end

8 GeneralRefinementAlgebra

theory *GeneralRefinementAlgebra*

imports *OmegaAlgebra*

begin

class *general-refinement-algebra* = *left-kleene-algebra* + *Omega* +
  assumes *Omega-unfold* : $y^\Omega \leq 1 + y ; y^\Omega$
  assumes *Omega-induct* : $x \leq z + y ; x \longrightarrow x \leq y^\Omega ; z$

begin

lemma *Omega-unfold-equal*: $y^\Omega = 1 + y ; y^\Omega$
  by (*smt Omega-induct Omega-unfold add-right-isotone antisym mult-right-isotone mult-right-one*)

lemma *Omega-add-1*: $(x + y)^\Omega = x^\Omega ; (y ; x^\Omega)^\Omega$
  apply (*rule antisym*)
  apply (*smt Omega-induct Omega-unfold-equal add-associative add-commutative add-right-isotone mult-associative mult-right-dist-add mult-right-isotone mult-right-one order-refl*)
  apply (*smt Omega-induct Omega-unfold-equal add-associative add-commutative mult-associative mult-left-one mult-right-dist-add mult-right-one order-refl*)
  done

lemma *Omega-left-slide*: $(x ; y)^\Omega ; x \leq x ; (y ; x)^\Omega$
proof –
  have $1 + y ; (x ; y)^\Omega ; x \leq 1 + y ; x ; (1 + (y ; (x ; y)^\Omega) ; x)$
  by (*smt Omega-unfold-equal add-right-isotone mult-associative mult-left-one mult-left-sub-dist-add mult-right-dist-add mult-right-isotone mult-right-one*)
  thus *?thesis*
  by (*smt Omega-induct Omega-unfold-equal add-least-upper-bound mult-associative mult-left-one mult-right-dist-add mult-right-isotone mult-right-one*)
qed

end

— Theorem 50.3

sublocale *general-refinement-algebra* < *Omega*!: *left-conway-semiring* **where** *circ* = *Omega*
  apply *unfold-locales*
  apply (*metis Omega-unfold-equal*)
  apply (*metis Omega-left-slide*)
  apply (*metis Omega-add-1*)
  done

context *general-refinement-algebra*

begin

lemma *star-below-Omega*: $x^\star \leq x^\Omega$
  by (*metis Omega-induct mult-right-one order-refl star.circ-left-unfold*)

lemma *star-mult-Omega*: $x^\Omega = x^\star ; x^\Omega$
  by (*metis Omega.left-plus-below-circ add-commutative add-left-upper-bound eq-iff star.circ-loop-fixpoint star-left-induct-mult-iff*)

lemma *Omega-one-greatest*: $x \leq 1^\Omega$
  by (*metis Omega-induct add-left-zero mult-left-one order-refl order-trans zero-right-mult-decreasing*)

lemma *greatest-left-zero*: $1^\Omega ; x = 1^\Omega$
  by (*metis antisym Omega-one-greatest Omega-induct add-right-upper-bound mult-left-one*)

lemma *circ-right-unfold*: $1 + x^\Omega ; x = x^\Omega$ **nitpick** [*expect=genuine,card=8*] **oops**
lemma *circ-slide*: $(x ; y)^\Omega ; x = x ; (y ; x)^\Omega$ **nitpick** [*expect=genuine,card=6*] **oops**
lemma *circ-simulate*: $z ; x \leq y ; z \longrightarrow z ; x^\Omega \leq y^\Omega ; z$ **nitpick** [*expect=genuine,card=6*] **oops**
lemma *circ-simulate-right*: $z ; x \leq y ; z + w \longrightarrow z ; x^\Omega \leq y^\Omega ; (z + w ; x^\Omega)$ **nitpick** [*expect=genuine,card=6*] **oops**
lemma *circ-simulate-right-1*: $z ; x \leq y ; z \longrightarrow z ; x^\Omega \leq y^\Omega ; z$ **nitpick** [*expect=genuine,card=6*] **oops**
lemma *circ-simulate-right-plus*: $z ; x \leq y ; y^\Omega ; z + w \longrightarrow z ; x^\Omega \leq y^\Omega ; (z + w ; x^\Omega)$ **nitpick** [*expect=genuine,card=6*]

oops
lemma *circ-simulate-right-plus-1*: $z ; x \le y ; y^\Omega ; z \longrightarrow z ; x^\Omega \le y^\Omega ; z$ **nitpick** [*expect=genuine,card=6*] **oops**
lemma *circ-simulate-left-1*: $x ; z \le z ; y \longrightarrow x^\Omega ; z \le z ; y^\Omega + x^\Omega ; 0$ **oops**
lemma *circ-simulate-left-plus-1*: $x ; z \le z ; y^\Omega \longrightarrow x^\Omega ; z \le z ; y^\Omega + x^\Omega ; 0$ **nitpick** [*expect=genuine,card=8*] **oops**
lemma *circ-simulate-absorb*: $y ; x \le x \longrightarrow y^\Omega ; x \le x + y^\Omega ; 0$ **nitpick** [*expect=genuine,card=8*] **oops**

end

class *bounded-general-refinement-algebra* = *general-refinement-algebra* + *bounded-left-kleene-algebra*

begin

lemma *Omega-one*: $1^\Omega = T$
by (*metis Omega.circ-transitive-equal Omega-induct add-left-top add-right-upper-bound less-eq-def mult-left-one*)

lemma *top-left-zero*: $T ; x = T$
by (*metis Omega-induct Omega-one add-left-top add-right-upper-bound less-eq-def mult-left-one*)

end

sublocale *bounded-general-refinement-algebra* < *Omega*!: *bounded-left-conway-semiring* **where** *circ* = *Omega* **..**

class *left-demonic-refinement-algebra* = *general-refinement-algebra* +
assumes *Omega-isolate*: $y^\Omega \le y^\Omega ; 0 + y^\star$

begin

lemma *Omega-isolate-equal*: $y^\Omega = y^\Omega ; 0 + y^\star$
by (*metis Omega-isolate add-commutative add-same-context less-eq-def star-below-Omega zero-right-mult-decreasing*)

lemma *Omega-sum-unfold-1*: $(x + y)^\Omega = y^\Omega + y^\star ; x ; (x + y)^\Omega$ **oops**
lemma *Omega-add-3*: $(x + y)^\Omega = (x^\star ; y)^\Omega ; x^\Omega$ **oops**

end

class *bounded-left-demonic-refinement-algebra* = *left-demonic-refinement-algebra* + *bounded-left-kleene-algebra*

begin

lemma *Omega-mult*: $(x ; y)^\Omega = 1 + x ; (y ; x)^\Omega ; y$ **oops**
lemma *Omega-add*: $(x + y)^\Omega = (x^\Omega ; y)^\Omega ; x^\Omega$ **oops**
lemma *Omega-simulate*: $z ; x \le y ; z \longrightarrow z ; x^\Omega \le y^\Omega ; z$ **nitpick** [*expect=genuine,card=6*] **oops**
lemma *Omega-separate-2*: $y ; x \le x ; (x + y) \longrightarrow (x + y)^\Omega = x^\Omega ; y^\Omega$ **oops**
lemma *Omega-circ-simulate-right-plus*: $z ; x \le y ; y^\Omega ; z + w \longrightarrow z ; x^\Omega \le y^\Omega ; (z + w ; x^\Omega)$ **nitpick** [*expect=genuine,card=6*]
oops
lemma *Omega-circ-simulate-left-plus*: $x ; z \le z ; y^\Omega + w \longrightarrow x^\Omega ; z \le (z + x^\Omega ; w) ; y^\Omega$ **oops**

end

sublocale *bounded-left-demonic-refinement-algebra* < *Omega*!: *bounded-left-conway-semiring* **where** *circ* = *Omega* **..**

class *demonic-refinement-algebra* = *left-zero-kleene-algebra* + *left-demonic-refinement-algebra*

begin

lemma *Omega-mult*: $(x ; y)^\Omega = 1 + x ; (y ; x)^\Omega ; y$
by (*smt Omega.circ-left-slide Omega-induct Omega-unfold-equal eq-iff mult-associative mult-left-dist-add mult-right-one*)

lemma *Omega-add*: $(x + y)^\Omega = (x^\Omega ; y)^\Omega ; x^\Omega$
by (*smt Omega-add-1 Omega-mult mult-associative mult-left-dist-add mult-left-one mult-right-dist-add mult-right-one*)

lemma *Omega-simulate*: $z ; x \le y ; z \longrightarrow z ; x^\Omega \le y^\Omega ; z$
by (*smt Omega-induct Omega-unfold-equal add-right-isotone mult-associative mult-left-dist-add mult-left-isotone mult-right-one*)

end

— Theorem 2.4

sublocale *demonic-refinement-algebra* < *Omega1*!: *itering-1* **where** *circ* = *Omega*
  apply *unfold-locales*
  apply (*metis Omega-simulate mult-associative order-refl*)
  apply (*metis Omega-simulate*)
  done

sublocale *demonic-refinement-algebra* < *Omega1*!: *left-zero-conway-semiring-1* **where** *circ* = *Omega* **..**

context *demonic-refinement-algebra*

begin

lemma *Omega-sum-unfold-1*: $(x + y)^\Omega = y^\Omega + y^\star ; x ; (x + y)^\Omega$
  by (*smt Omega1.circ-add-9 Omega.circ-loop-fixpoint Omega-isolate-equal add-associative add-commutative mult-associative mult-left-zero mult-right-dist-add*)

lemma *Omega-add-3*: $(x + y)^\Omega = (x^\star ; y)^\Omega ; x^\Omega$
  by (*smt Omega1.circ-add-9 Omega.circ-isotone Omega-induct Omega-sum-unfold-1 add-commutative antisym mult-left-isotone order-refl star-below-Omega*)

lemma *Omega-separate-2*: $y ; x \leq x ; (x + y) \longrightarrow (x + y)^\Omega = x^\Omega ; y^\Omega$
  by (*smt Omega.circ-sub-dist-3 Omega-induct Omega-sum-unfold-1 add-right-isotone antisym mult-associative mult-left-isotone star-mult-Omega star-simulation-left*)

lemma *Omega-circ-simulate-right-plus*: $z ; x \leq y ; y^\Omega ; z + w \longrightarrow z ; x^\Omega \leq y^\Omega ; (z + w ; x^\Omega)$
proof
  assume *1*: $z ; x \leq y ; y^\Omega ; z + w$
  have $z ; x^\Omega = z + z ; x ; x^\Omega$
    by (*metis Omega1.circ-back-loop-fixpoint Omega1.circ-plus-same add-commutative mult-associative*)
  also have ... $\leq y ; y^\Omega ; z ; x^\Omega + z + w ; x^\Omega$ **using** *1*
    by (*smt add-associative add-commutative add-right-isotone less-eq-def mult-right-dist-add*)
  finally have $z ; x^\Omega \leq (y ; y^\Omega)^\Omega ; (z + w ; x^\Omega)$
    by (*smt Omega-induct add-associative add-commutative mult-associative*)
  thus $z ; x^\Omega \leq y^\Omega ; (z + w ; x^\Omega)$
    by (*metis Omega.left-plus-circ*)
qed

lemma *Omega-circ-simulate-left-plus*: $x ; z \leq z ; y^\Omega + w \longrightarrow x^\Omega ; z \leq (z + x^\Omega ; w) ; y^\Omega$
proof
  assume $x ; z \leq z ; y^\Omega + w$
  hence $x ; ((z + x^\Omega ; w) ; y^\Omega) \leq (z ; y^\Omega + w + x ; x^\Omega ; w) ; y^\Omega$
    by (*smt mult-associative mult-left-dist-add add-left-isotone mult-left-isotone*)
  also have ... $\leq z ; y^\Omega ; y^\Omega + w ; y^\Omega + x^\Omega ; w ; y^\Omega$
    by (*smt Omega.left-plus-below-circ add-right-isotone mult-left-isotone mult-right-dist-add*)
  finally have $x ; ((z + x^\Omega ; w) ; y^\Omega) \leq (z + x^\Omega ; w) ; y^\Omega$
    by (*metis Omega.circ-transitive-equal mult-associative Omega.circ-reflexive add-associative less-eq-def mult-left-one mult-right-dist-add*)
  thus $x^\Omega ; z \leq (z + x^\Omega ; w) ; y^\Omega$
    by (*smt Omega1.circ-back-loop-fixpoint Omega-isolate-equal add-least-upper-bound mult-associative mult-left-zero mult-right-dist-add mult-right-sub-dist-add-left mult-right-sub-dist-add-right star-left-induct*)
qed

lemma *Omega-circ-simulate-right*: $z ; x \leq y ; z + w \longrightarrow z ; x^\Omega \leq y^\Omega ; (z + w ; x^\Omega)$
proof
  assume $z ; x \leq y ; z + w$
  also have ... $\leq y ; y^\Omega ; z + w$
    by (*smt Omega.circ-loop-fixpoint add-associative add-commutative add-left-upper-bound mult-associative mult-left-dist-add*)
  finally show $z ; x^\Omega \leq y^\Omega ; (z + w ; x^\Omega)$
    by (*metis Omega-circ-simulate-right-plus*)
qed

end

sublocale *demonic-refinement-algebra* < *Omega*!: *itering* **where** *circ* = *Omega*
  apply *unfold-locales*
  apply (*metis Omega-add*)
  apply (*metis Omega-mult*)
  apply (*metis Omega-circ-simulate-right-plus*)
  apply (*metis Omega-circ-simulate-left-plus*)

done

class *bounded-demonic-refinement-algebra* = *demonic-refinement-algebra* + *bounded-left-zero-kleene-algebra*

begin

lemma *Omega-one*: $1^{\Omega} = T$
by (*metis Omega.circ-transitive-equal Omega-induct add-left-top add-right-upper-bound less-eq-def mult-left-one*)

lemma *top-left-zero*: $T ; x = T$
by (*metis Omega-induct Omega-one add-left-top add-right-upper-bound less-eq-def mult-left-one*)

end

sublocale *bounded-demonic-refinement-algebra* < *Omega*!: *bounded-itering* **where** *circ* = *Omega* **..**

class *general-refinement-algebra-omega* = *left-omega-algebra* + *Omega* +
assumes *omega-left-zero*: $x^{\omega} \leq x^{\omega} ; y$
assumes *Omega-def*: $x^{\Omega} = x^{\omega} + x^{\star}$

begin

lemma *omega-left-zero-equal*: $x^{\omega} ; y = x^{\omega}$
by (*metis antisym omega-left-zero omega-sub-vector*)

subclass *left-demonic-refinement-algebra*
apply *unfold-locales*
apply (*metis Omega-def add-commutative eq-refl mult-right-one omega-loop-fixpoint*)
apply (*metis Omega-def mult-right-dist-add omega-induct omega-left-zero-equal*)
apply (*smt Omega-def add-least-upper-bound antisym mult-right-dist-add mult-right-sub-dist-add-left omega-left-zero-equal order-refl star-zero-below-omega*)
done

end

class *left-demonic-refinement-algebra-omega* = *bounded-left-omega-algebra* + *Omega* +
assumes *top-left-zero*: $T ; x = T$
assumes *Omega-def*: $x^{\Omega} = x^{\omega} + x^{\star}$

begin

subclass *general-refinement-algebra-omega*
apply *unfold-locales*
apply (*metis mult-associative omega-vector order-refl top-left-zero*)
apply (*rule Omega-def*)
done

end

class *demonic-refinement-algebra-omega* = *left-demonic-refinement-algebra-omega* + *bounded-left-zero-omega-algebra*

begin

lemma *Omega-mult*: $(x ; y)^{\Omega} = 1 + x ; (y ; x)^{\Omega} ; y$
by (*metis Omega-def comb1.circ-mult-1 omega-left-zero-equal omega-translate*)

lemma *Omega-add*: $(x + y)^{\Omega} = (x^{\Omega} ; y)^{\Omega} ; x^{\Omega}$
proof −
have $(x^{\Omega} ; y)^{\Omega} ; x^{\Omega} = (x^{\star} ; y)^{\star} ; x^{\omega} + (x^{\star} ; y)^{\omega} + (x^{\star} ; y)^{\star} ; x^{\omega\star} ; x^{\Omega}$
by (*metis add-commutative Omega-def mult-associative mult-right-dist-add mult-zero-add-omega omega-left-zero-equal star.circ-add-1*)
thus *?thesis*
by (*smt add-associative add-commutative Omega-def mult-associative mult-left-dist-add omega-decompose omega-left-zero-equal star.circ-add-1 star.circ-loop-fixpoint star.circ-slide*)
qed

lemma *Omega-simulate*: $z ; x \leq y ; z \longrightarrow z ; x^{\Omega} \leq y^{\Omega} ; z$
by (*smt add-isotone Omega-def mult-left-dist-add mult-right-dist-add omega-left-zero-equal omega-simulation star-simulation-right*)

subclass *demonic-refinement-algebra* **..**

end

end

9 Tests

theory *Tests*

imports *Base*

begin

class *tests* = *mult* + *neg* + *one* + *ord* + *plus* + *zero* +
  assumes *sub-assoc*: $-x ; (-y ; -z) = (-x ; -y) ; -z$
  assumes *sub-comm*: $-x ; -y = -y ; -x$
  assumes *sub-compl*: $-x = -(--x ; -y) ; -(--x ; --y)$
  assumes *sub-mult-closed*: $-x ; -y = --(-x ; -y)$
  assumes *the-zero-def*: $0 = (\mathit{THE}\ x\ .\ (\forall y\ .\ x = -y ; --y))$ — define without imposing uniqueness
  assumes *one-def*: $1 = -\ 0$
  assumes *plus-def*: $-x + -y = -(--x ; --y)$
  assumes *leq-def*: $-x \leq -y \longleftrightarrow -x ; -y = -x$
  assumes *strict-leq-def*: $-x < -y \longleftrightarrow -x \leq -y \wedge \neg (-y \leq -x)$

begin

— uniqueness of 0, resulting in the lemma *zero-def* to replace the assumption *the-zero-def*

lemma *unique-zero*: $-x ; --x = -y ; --y$
  by (*metis sub-assoc sub-comm sub-compl*)

definition *is-zero* :: $'a \Rightarrow bool$
  where *is-zero-def*: $\mathit{is\text{-}zero}(x) \equiv (\forall y\ .\ x = -y ; --y)$

lemma *the-zero-def-p*: $0 = (\mathit{THE}\ x\ .\ \mathit{is\text{-}zero}(x))$
  by (*simp only*: *the-zero-def is-zero-def*)

lemma *zero-def*: $0 = -x ; --x$
  by (*metis unique-zero the-zero-def-p is-zero-def theI'*)

— consequences for meet and complement

lemma *double-negation*: $-x = ---x$
  by (*metis sub-mult-closed sub-compl*)

lemma *compl-1*: $--x = -(-x ; -y) ; -(-x ; --y)$
  by (*metis double-negation sub-compl*)

lemma *right-zero*: $-x ; (-y ; --y) = -z ; --z$
  by (*metis compl-1 sub-assoc sub-mult-closed zero-def*)

lemma *right-one*: $-x ; -x = -x ; -(-y ; --y)$
  by (*metis compl-1 right-zero sub-mult-closed zero-def*)

lemma *mult-idempotent*: $-x ; -x = -x$
  by (*metis compl-1 double-negation sub-assoc sub-mult-closed zero-def*)

lemma *compl-2*: $-x = -(-(-x ; -y) ; -(-x ; --y))$
  by (*metis compl-1 double-negation*)

— consequences for join

lemma *plus-closed*: $-x + -y = --(-x + -y)$
  by (*metis plus-def double-negation*)

lemma *plus-assoc*: $-x + (-y + -z) = (-x + -y) + -z$
  by (*metis plus-def sub-assoc sub-mult-closed*)

lemma *plus-comm*: $-x + -y = -y + -x$
  by (*metis plus-def sub-comm*)

lemma *plus-idempotent*: $-x + -x = -x$
  by (*metis double-negation mult-idempotent plus-def*)

lemma *plus-absorb*: $-x ; -y + -x = -x$
  by (*smt compl-1 mult-idempotent plus-def sub-assoc sub-mult-closed*)

lemma *mult-absorb*: $-x ; (-x + -y) = -x$
  by (*smt plus-absorb plus-def sub-mult-closed sub-comm*)

lemma *plus-deMorgan*: $-(-x + -y) = --x ; --y$
  by (*metis plus-def sub-mult-closed*)

lemma *mult-deMorgan*: $-(-x ; -y) = --x + --y$
  by (*metis double-negation plus-def*)

lemma *mult-cases*: $-x = (-x + -y) ; (-x + --y)$
  by (*metis compl-1 double-negation plus-def*)

lemma *plus-cases*: $-x = -x ; -y + -x ; --y$
  by (*smt mult-deMorgan double-negation mult-cases sub-mult-closed*)

lemma *plus-compl-intro*: $(-x ; -y) + --x = -y + --x$
  by (*smt compl-1 mult-deMorgan plus-absorb plus-cases sub-assoc sub-comm sub-mult-closed*)

lemma *mult-compl-intro*: $-x ; -y = -x ; (--x + -y)$
  by (*metis sub-mult-closed mult-cases plus-absorb plus-compl-intro plus-comm*)

lemma *mult-distr-plus-left*: $-x ; (-y + -z) = (-x ; -y) + (-x ; -z)$
  by (*smt mult-cases plus-absorb plus-assoc plus-comm plus-compl-intro plus-deMorgan plus-def sub-assoc sub-mult-closed*)

lemma *plus-distr-mult-left*: $-x + (-y ; -z) = (-x + -y) ; (-x + -z)$
  by (*smt mult-deMorgan mult-distr-plus-left plus-def sub-mult-closed*)

lemma *mult-distr-plus-right*: $(-y + -z) ; -x = (-y ; -x) + (-z ; -x)$
  by (*metis mult-distr-plus-left plus-def sub-comm*)

lemma *plus-distr-mult-right*: $(-y ; -z) + -x = (-y + -x) ; (-z + -x)$
  by (*metis plus-distr-mult-left sub-mult-closed plus-comm*)

lemma *case-duality*: $(--x + -y) ; (-x + -z) = -x ; -y + --x ; -z$
proof −
  have $(--x + -y) ; (-x + -z) = --(-y ; -z ; -x) + --(-y ; -x) + (--(-y ; -z ; --x) + --(--x ; -z))$
    by (*smt mult-distr-plus-left plus-closed mult-compl-intro sub-comm plus-assoc plus-cases sub-mult-closed plus-comm*)
  thus *?thesis*
    by (*smt sub-comm sub-assoc sub-mult-closed plus-absorb*)
qed

lemma *case-duality-2*: $(-x + -y) ; (--x + -z) = -x ; -z + --x ; -y$
  by (*metis case-duality double-negation plus-comm sub-mult-closed*)

lemma *compl-cases*: $(-v + -w) ; (--v + -x) + -((-v + -y) ; (--v + -z)) = (-v + -w + --y) ; (--v + -x + --z)$
  by (*smt mult-deMorgan plus-deMorgan sub-mult-closed plus-closed double-negation case-duality sub-comm plus-assoc plus-comm mult-distr-plus-left case-duality-2*)

lemma *plus-cases-2*: $--x = -(-x + -y) + -(-x + --y)$
  by (*metis mult-deMorgan plus-deMorgan double-negation mult-cases sub-mult-closed plus-closed*)

— consequences for 0 and 1

lemma *mult-compl*: $-x ; --x = 0$
  by (*metis zero-def*)

lemma *plus-compl*: $-x + --x = 1$
  by (*metis one-def plus-def zero-def*)

lemma *one-compl*: $-\ 1 = 0$
  by (*metis mult-compl one-def sub-mult-closed*)

lemma *bs-mult-right-zero*: $-x\ ;\ 0 = 0$
  by (*metis right-zero zero-def*)

lemma *bs-mult-left-zero*: $0\ ;\ -x = 0$
  by (*metis bs-mult-right-zero one-compl sub-comm*)

lemma *plus-right-one*: $-x + 1 = 1$
  by (*metis one-compl one-def mult-deMorgan double-negation bs-mult-right-zero*)

lemma *plus-left-one*: $1 + -x = 1$
  by (*metis plus-right-one one-def plus-comm*)

lemma *bs-mult-right-one*: $-x\ ;\ 1 = -x$
  by (*metis mult-compl one-def mult-idempotent right-one*)

lemma *bs-mult-left-one*: $1\ ;\ -x = -x$
  by (*metis one-def bs-mult-right-one sub-comm*)

lemma *plus-right-zero*: $-x + 0 = -x$
  by (*metis mult-compl mult-cases plus-distr-mult-left*)

lemma *plus-left-zero*: $0 + -x = -x$
  by (*metis plus-right-zero one-compl plus-comm*)

lemma *one-double-compl*: $--\ 1 = 1$
  by (*metis one-compl one-def*)

lemma *zero-double-compl*: $--\ 0 = 0$
  by (*metis one-compl one-def*)

— consequences for the order

lemma *reflexive*: $-x \leq -x$
  by (*metis leq-def mult-idempotent*)

lemma *transitive*: $-x \leq -y \wedge -y \leq -z \longrightarrow -x \leq -z$
  by (*metis leq-def sub-assoc*)

lemma *antisymmetric*: $-x \leq -y \wedge -y \leq -x \longrightarrow -x = -y$
  by (*metis leq-def sub-comm*)

lemma *zero-least-test*: $0 \leq -x$
  by (*metis one-compl leq-def bs-mult-right-zero sub-comm*)

lemma *one-greatest*: $-x \leq 1$
  by (*metis leq-def one-def bs-mult-right-one*)

lemma *lower-bound-left*: $-x\ ;\ -y \leq -x$
  by (*metis leq-def mult-idempotent sub-assoc sub-mult-closed sub-comm*)

lemma *lower-bound-right*: $-x\ ;\ -y \leq -y$
  by (*metis leq-def mult-idempotent sub-assoc sub-mult-closed*)

lemma *mult-iso-left*: $-x \leq -y \longrightarrow -x\ ;\ -z \leq -y\ ;\ -z$
  by (*metis leq-def lower-bound-left sub-assoc sub-comm sub-mult-closed*)

lemma *mult-iso-right*: $-x \leq -y \longrightarrow -z\ ;\ -x \leq -z\ ;\ -y$
  by (*metis mult-iso-left sub-comm*)

lemma *mult-iso*: $-p \leq -q \wedge -r \leq -s \longrightarrow -p\ ;\ -r \leq -q\ ;\ -s$
  by (*smt transitive mult-iso-left mult-iso-right sub-mult-closed*)

lemma *compl-anti*: $-x \leq -y \longrightarrow --y \leq --x$
  by (*smt one-compl plus-compl plus-deMorgan double-negation leq-def plus-comm plus-compl-intro plus-right-zero*)

lemma *leq-plus*: $-x \leq -y \longleftrightarrow -x + -y = -y$
  by (*metis double-negation leq-def mult-absorb plus-def sub-comm*)

lemma *plus-compl-iso*: $-x \leq -y \longrightarrow -(-y + -z) \leq -(-x + -z)$

by (*metis plus-deMorgan leq-plus lower-bound-left mult-iso-left*)

lemma *plus-iso-left*: $-x \le -y \longrightarrow -x + -z \le -y + -z$
by (*metis plus-compl-iso compl-anti double-negation plus-def*)

lemma *plus-iso-right*: $-x \le -y \longrightarrow -z + -x \le -z + -y$
by (*metis plus-iso-left plus-comm*)

lemma *plus-iso*: $-p \le -q \wedge -r \le -s \longrightarrow -p + -r \le -q + -s$
by (*smt transitive plus-iso-left plus-iso-right plus-closed*)

lemma *greatest-lower-bound*: $-x \le -y \wedge -x \le -z \longleftrightarrow -x \le -y ; -z$
by (*metis leq-def plus-absorb plus-comm sub-assoc sub-mult-closed*)

lemma *upper-bound-left*: $-x \le -x + -y$
by (*metis one-compl plus-iso-right plus-right-zero zero-least-test*)

lemma *upper-bound-right*: $-y \le -x + -y$
by (*metis upper-bound-left plus-comm*)

lemma *least-upper-bound*: $-x \le -z \wedge -y \le -z \longleftrightarrow -x + -y \le -z$
by (*metis leq-plus plus-assoc plus-def upper-bound-right*)

lemma *leq-mult-zero*: $-x \le -y \longleftrightarrow -x ; --y = 0$
proof –
have $-x \le -y \longrightarrow -x ; --y = 0$
by (*metis leq-def sub-assoc mult-compl bs-mult-right-zero*)
also have $-x ; --y = 0 \longrightarrow -x \le -y$
by (*metis compl-1 one-def leq-def bs-mult-right-one sub-mult-closed*)
ultimately show *?thesis* **by** *metis*
qed

lemma *leq-plus-right-one*: $-x \le -y \longleftrightarrow --x + -y = 1$
by (*metis one-compl one-def mult-deMorgan plus-deMorgan double-negation leq-mult-zero*)

lemma *shunting*: $-x ; -y \le -z \longleftrightarrow -y \le --x + -z$
by (*smt leq-mult-zero sub-assoc sub-mult-closed sub-comm plus-deMorgan double-negation mult-deMorgan*)

lemma *shunting-right*: $-x ; -y \le -z \longleftrightarrow -x \le -z + --y$
by (*metis plus-comm shunting sub-comm*)

lemma *leq-cases*: $-x ; -y \le -z \wedge --x ; -y \le -z \longrightarrow -y \le -z$
by (*smt least-upper-bound sub-mult-closed mult-distr-plus-left sub-comm plus-compl bs-mult-right-one*)

lemma *leq-cases-2*: $-x ; -y \le -x ; -z \wedge --x ; -y \le --x ; -z \longrightarrow -y \le -z$
by (*metis greatest-lower-bound leq-cases sub-mult-closed*)

lemma *leq-cases-3*: $-y ; -x \le -z ; -x \wedge -y ; --x \le -z ; --x \longrightarrow -y \le -z$
by (*metis leq-cases-2 sub-comm*)

lemma *eq-cases*: $-x ; -y = -x ; -z \wedge --x ; -y = --x ; -z \longrightarrow -y = -z$
by (*metis plus-cases sub-comm*)

lemma *eq-cases-2*: $-y ; -x = -z ; -x \wedge -y ; --x = -z ; --x \longrightarrow -y = -z$
by (*metis eq-cases sub-comm*)

lemma *wnf-lemma-1*: $(-x ; -y + --x ; -z) ; -x = -x ; -y$
by (*smt mult-compl mult-distr-plus-right mult-idempotent plus-right-zero sub-assoc sub-comm sub-mult-closed*)

lemma *wnf-lemma-2*: $(-x ; -y + -z ; --y) ; -y = -x ; -y$
by (*metis sub-comm wnf-lemma-1*)

lemma *wnf-lemma-3*: $(-x ; -z + --x ; -y) ; --x = --x ; -y$
by (*smt mult-compl mult-distr-plus-right mult-idempotent plus-comm plus-right-zero sub-assoc sub-comm sub-mult-closed*)

lemma *wnf-lemma-4*: $(-z ; -y + -x ; --y) ; --y = -x ; --y$
by (*metis sub-comm wnf-lemma-3*)

— sets and sequences of tests

```
definition test-set :: 'a set ⇒ bool
  where test-set A ⟷ (∀ x∈A . x = −−x)

lemma mult-left-dist-test-set: test-set A ⟶ test-set { −p ; x | x . x ∈ A }
  by (smt mem-Collect-eq sub-mult-closed test-set-def)

lemma mult-right-dist-test-set: test-set A ⟶ test-set { x ; −p | x . x ∈ A }
  by (smt mem-Collect-eq sub-mult-closed test-set-def)

lemma plus-left-dist-test-set: test-set A ⟶ test-set { −p + x | x . x ∈ A }
  by (smt mem-Collect-eq plus-closed test-set-def)

lemma plus-right-dist-test-set: test-set A ⟶ test-set { x + −p | x . x ∈ A }
  by (smt mem-Collect-eq plus-closed test-set-def)

lemma test-set-closed: A ⊆ B ∧ test-set B ⟶ test-set A
  by (smt set-rev-mp test-set-def)

definition test-seq :: (nat ⇒ 'a) ⇒ bool
  where test-seq t ⟷ (∀ n . t n = −−t n)

lemma test-seq-test-set: test-seq t ⟶ test-set { t n | n::nat . True }
  by (smt mem-Collect-eq test-seq-def test-set-def)

definition nat-test :: (nat ⇒ 'a) ⇒ 'a ⇒ bool
  where nat-test t s ⟷ (∀ n . t n = −−t n) ∧ s = −−s ∧ (∀ n . t n ≤ s) ∧ (∀ x y . (∀ n . t n ; −x ≤ −y) ⟶ s ; −x ≤ −y)

lemma nat-test-seq: nat-test t s ⟶ test-seq t
  by (metis nat-test-def test-seq-def)

primrec pSum :: (nat ⇒ 'a) ⇒ nat ⇒ 'a
  where pSum f 0 = 0
      | pSum f (Suc m) = pSum f m + f m

lemma pSum-test: test-seq t ⟶ pSum t m = −−(pSum t m)
  apply (induct m)
  apply (metis pSum.simps(1) one-compl one-def)
  apply (smt pSum.simps(2) plus-closed test-seq-def)
  done

lemma pSum-test-nat: nat-test t s ⟶ pSum t m = −−(pSum t m)
  by (metis nat-test-seq pSum-test)

lemma pSum-upper: test-seq t ∧ i<m ⟶ t i ≤ pSum t m
proof (induct m)
  show test-seq t ∧ i<0 ⟶ t i ≤ pSum t 0
    by (smt less-zeroE)
next
  fix n
  assume test-seq t ∧ i<n ⟶ t i ≤ pSum t n
  hence test-seq t ∧ i<n ⟶ t i ≤ pSum t (Suc n)
    by (smt pSum.simps(2) pSum-test test-seq-def transitive upper-bound-left)
  thus test-seq t ∧ i<Suc n ⟶ t i ≤ pSum t (Suc n)
    by (metis pSum.simps(2) pSum-test test-seq-def upper-bound-right less-Suc-eq)
qed

lemma pSum-below: test-seq t ∧ (∀ m<k . t m ; −p ≤ −q) ⟶ pSum t k ; −p ≤ −q
  apply (induct k)
  apply (metis bs-mult-left-zero pSum.simps(1) zero-least-test)
  apply (smt least-upper-bound mult-distr-plus-right pSum.simps(2) pSum-test test-seq-def sub-mult-closed)
  done

lemma pSum-below-nat: nat-test t s ∧ (∀ m<k . t m ; −p ≤ −q) ⟶ pSum t k ; −p ≤ −q
  by (metis nat-test-seq pSum-below)

lemma pSum-below-sum: nat-test t s ⟶ pSum t x ≤ s
  by (smt bs-mult-right-one nat-test-def one-def pSum-below-nat pSum-test-nat)

lemma ascending-chain-plus-left: ascending-chain t ∧ test-seq t ⟶ ascending-chain (λn . −p + t n) ∧ test-seq (λn . −p + t
```

n)
  by (*smt ascending-chain-def plus-closed plus-iso-right test-seq-def*)

lemma *ascending-chain-plus-right*: *ascending-chain* $t \wedge$ *test-seq* $t \longrightarrow$ *ascending-chain* $(\lambda n\ .\ t\ n + -p) \wedge$ *test-seq* $(\lambda n\ .\ t\ n + -p)$
  by (*smt ascending-chain-def plus-closed plus-iso-left test-seq-def*)

lemma *ascending-chain-mult-left*: *ascending-chain* $t \wedge$ *test-seq* $t \longrightarrow$ *ascending-chain* $(\lambda n\ .\ -p\ ;\ t\ n) \wedge$ *test-seq* $(\lambda n\ .\ -p\ ;\ t\ n)$
  by (*smt ascending-chain-def sub-mult-closed mult-iso-right test-seq-def*)

lemma *ascending-chain-mult-right*: *ascending-chain* $t \wedge$ *test-seq* $t \longrightarrow$ *ascending-chain* $(\lambda n\ .\ t\ n\ ;\ -p) \wedge$ *test-seq* $(\lambda n\ .\ t\ n\ ;\ -p)$
  by (*smt ascending-chain-def sub-mult-closed mult-iso-left test-seq-def*)

lemma *descending-chain-plus-left*: *descending-chain* $t \wedge$ *test-seq* $t \longrightarrow$ *descending-chain* $(\lambda n\ .\ -p + t\ n) \wedge$ *test-seq* $(\lambda n\ .\ -p + t\ n)$
  by (*smt descending-chain-def plus-closed plus-iso-right test-seq-def*)

lemma *descending-chain-plus-right*: *descending-chain* $t \wedge$ *test-seq* $t \longrightarrow$ *descending-chain* $(\lambda n\ .\ t\ n + -p) \wedge$ *test-seq* $(\lambda n\ .\ t\ n + -p)$
  by (*smt descending-chain-def plus-closed plus-iso-left test-seq-def*)

lemma *descending-chain-mult-left*: *descending-chain* $t \wedge$ *test-seq* $t \longrightarrow$ *descending-chain* $(\lambda n\ .\ -p\ ;\ t\ n) \wedge$ *test-seq* $(\lambda n\ .\ -p\ ;\ t\ n)$
  by (*smt descending-chain-def sub-mult-closed mult-iso-right test-seq-def*)

lemma *descending-chain-mult-right*: *descending-chain* $t \wedge$ *test-seq* $t \longrightarrow$ *descending-chain* $(\lambda n\ .\ t\ n\ ;\ -p) \wedge$ *test-seq* $(\lambda n\ .\ t\ n\ ;\ -p)$
  by (*smt descending-chain-def sub-mult-closed mult-iso-left test-seq-def*)

end

typedef *'a negImage* = { x::*'a*::*tests* . ($\exists\, y$::*'a* . $x = -y$) }
  by *auto*

lemma *simp-negImage* [*simp*]: $\exists\, y$. *Rep-negImage* $x = -y$
  using *Rep-negImage*
  by *simp*

setup-lifting *type-definition-negImage*

instantiation *negImage* :: (*tests*) *boolean-algebra*

begin

lift-definition *sup-negImage* :: *'a negImage* $\Rightarrow$ *'a negImage* $\Rightarrow$ *'a negImage* **is** *plus*
  by (*metis plus-closed*)

lift-definition *inf-negImage* :: *'a negImage* $\Rightarrow$ *'a negImage* $\Rightarrow$ *'a negImage* **is** *times*
  by (*metis sub-mult-closed*)

lift-definition *minus-negImage* :: *'a negImage* $\Rightarrow$ *'a negImage* $\Rightarrow$ *'a negImage* **is** $\lambda x\ y\ .\ x\ ;\ -y$
  by (*metis sub-mult-closed*)

lift-definition *uminus-negImage* :: *'a negImage* $\Rightarrow$ *'a negImage* **is** *uminus*
  by *metis*

lift-definition *bot-negImage* :: *'a negImage* **is** *0*
  by (*metis one-compl*)

lift-definition *top-negImage* :: *'a negImage* **is** *1*
  by (*metis one-def*)

lift-definition *less-eq-negImage* :: *'a negImage* $\Rightarrow$ *'a negImage* $\Rightarrow$ *bool* **is** *less-eq* **.**

lift-definition *less-negImage* :: *'a negImage* $\Rightarrow$ *'a negImage* $\Rightarrow$ *bool* **is** *less* **.**

instance

```
  apply intro-classes
  apply (metis (mono-tags) less-eq-negImage.rep-eq less-negImage.rep-eq strict-leq-def simp-negImage)
  apply (metis less-eq-negImage.rep-eq simp-negImage reflexive)
  apply (metis (mono-tags) less-eq-negImage.rep-eq simp-negImage transitive)
  apply (metis Rep-negImage-inject antisymmetric less-eq-negImage.rep-eq simp-negImage)
  apply (metis (mono-tags) inf-negImage.rep-eq less-eq-negImage.rep-eq lower-bound-left simp-negImage)
  apply (metis (mono-tags) inf-negImage.rep-eq less-eq-negImage.rep-eq lower-bound-right simp-negImage)
  apply (smt2 inf-negImage.rep-eq leq-def less-eq-negImage.rep-eq simp-negImage sub-assoc)
  apply (metis (mono-tags) less-eq-negImage.rep-eq simp-negImage sup-negImage.rep-eq upper-bound-left)
  apply (metis (mono-tags) less-eq-negImage.rep-eq simp-negImage sup-negImage.rep-eq upper-bound-right)
  apply (smt2 leq-plus less-eq-negImage.rep-eq plus-assoc simp-negImage sup-negImage.rep-eq)
  apply (smt2 bot-negImage.rep-eq less-eq-negImage.rep-eq simp-negImage zero-least-test)
  apply (smt2 less-eq-negImage.rep-eq one-greatest simp-negImage top-negImage.rep-eq)
  apply (metis (mono-tags, hide-lams) Rep-negImage-inject inf-negImage.rep-eq plus-distr-mult-left sup-negImage.rep-eq simp-negImage)
  apply (smt2 Rep-negImage-inject inf-negImage.rep-eq bot-negImage.rep-eq uminus-negImage.rep-eq zero-def simp-negImage)
  apply (smt2 Rep-negImage-inject sup-negImage.rep-eq top-negImage.rep-eq plus-compl uminus-negImage.rep-eq simp-negImage)
  apply (metis (mono-tags) Rep-negImage-inject inf-negImage.rep-eq minus-negImage.rep-eq uminus-negImage.rep-eq)
  done

end

end
```

10 TestItering

theory *TestItering*

imports *Itering Tests*

begin

class *test-itering* = *itering* + *tests* + *while* +
  assumes *while-def*: $p \star y = (p \ ; \ y)^\circ \ ; \ -p$

begin

lemma *wnf-lemma-5*: $(-p + -q) \ ; \ (-q \ ; \ x + --q \ ; \ y) = -q \ ; \ x + --q \ ; \ -p \ ; \ y$
  by (*smt mult-absorb mult-associative mult-compl-intro mult-idempotent mult-left-dist-add plus-def sub-comm*)

lemma *test-case-split-left-equal*: $-z \ ; \ x = -z \ ; \ y \wedge --z \ ; \ x = --z \ ; \ y \longrightarrow x = y$
  by (*metis case-split-left-equal plus-compl*)

lemma *preserves-equation*: $-y \ ; \ x \leq x \ ; \ -y \longleftrightarrow -y \ ; \ x = -y \ ; \ x \ ; \ -y$
  apply (*rule iffI*)
  apply (*metis eq-refl mult-idempotent one-greatest test-preserves-equation*)
  apply (*metis mult-left-isotone mult-left-one-1 one-greatest*)
  done

— Theorem 5

lemma *preserve-test*: $-y \ ; \ x \leq x \ ; \ -y \longrightarrow -y \ ; \ x^\circ = -y \ ; \ x^\circ \ ; \ -y$
  by (*metis circ-simulate preserves-equation*)

— Theorem 5

lemma *import-test*: $-y \ ; \ x \leq x \ ; \ -y \longrightarrow -y \ ; \ x^\circ = -y \ ; \ (-y \ ; \ x)^\circ$
  apply *rule*
  apply (*rule antisym*)
  apply (*metis circ-simulate circ-slide mult-associative mult-idempotent preserves-equation*)
  apply (*metis circ-isotone mult-left-isotone mult-left-one mult-right-isotone one-greatest*)
  done

definition *ite* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ (- ◁ - ▷ - [*58*,*58*,*58*] *57*)
  where $x \lhd p \rhd y = p \ ; \ x + -p \ ; \ y$

definition *it* :: $'a \Rightarrow 'a \Rightarrow 'a$ (- ▷ - [*58*,*58*] *57*)
  where $p \rhd x = p \ ; \ x + -p$

definition *assigns* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$
  where *assigns x p q* $\longleftrightarrow x = x \ ; \ (p \ ; \ q + -p \ ; \ -q)$

definition *preserves* :: $'a \Rightarrow 'a \Rightarrow bool$
  where *preserves x p* $\longleftrightarrow p \ ; \ x \leq x \ ; \ p \wedge -p \ ; \ x \leq x \ ; \ -p$

lemma *ite-neg*: $x \lhd -p \rhd y = y \lhd --p \rhd x$
  by (*metis add-commutative double-negation ite-def*)

lemma *ite-import-true*: $x \lhd -p \rhd y = -p \ ; \ x \lhd -p \rhd y$
  by (*metis ite-def mult-associative mult-idempotent*)

lemma *ite-import-false*: $x \lhd -p \rhd y = x \lhd -p \rhd --p \ ; \ y$
  by (*metis ite-def mult-associative mult-idempotent*)

lemma *ite-import-true-false*: $x \lhd -p \rhd y = -p \ ; \ x \lhd -p \rhd --p \ ; \ y$
  by (*metis ite-import-false ite-import-true*)

lemma *ite-context-true*: $-p \ ; \ (x \lhd -p \rhd y) = -p \ ; \ x$
  by (*metis add-right-zero ite-def mult-associative mult-compl mult-idempotent mult-left-dist-add mult-left-zero*)

lemma *ite-context-false*: $--p \ ; \ (x \lhd -p \rhd y) = --p \ ; \ y$

by (*metis ite-neg ite-context-true*)

lemma *ite-context-import*: $-p ; (x \lhd -q \rhd y) = -p ; (x \lhd -p ; -q \rhd y)$
by (*smt ite-def mult-associative mult-compl-intro mult-deMorgan mult-idempotent mult-left-dist-add*)

lemma *ite-conjunction*: $(x \lhd -q \rhd y) \lhd -p \rhd y = x \lhd -p ; -q \rhd y$
by (*smt add-associative add-commutative ite-def mult-associative mult-deMorgan mult-left-dist-add mult-right-dist-add plus-compl-intro*)

lemma *ite-disjunction*: $x \lhd -p \rhd (x \lhd -q \rhd y) = x \lhd -p + -q \rhd y$
by (*smt add-associative double-negation ite-def mult-associative mult-compl-intro mult-deMorgan mult-left-dist-add mult-right-dist-add plus-deMorgan*)

lemma *wnf-lemma-6*: $(-p + -q) ; (x \lhd --p ; -q \rhd y) = (-p + -q) ; (y \lhd -p \rhd x)$
by (*smt add-commutative double-negation ite-def mult-associative mult-compl mult-deMorgan mult-idempotent mult-left-dist-add plus-compl-intro sub-comm*)

lemma *it-ite*: $-p \rhd x = x \lhd -p \rhd 1$
by (*metis it-def ite-def mult-right-one*)

lemma *it-neg*: $--p \rhd x = 1 \lhd -p \rhd x$
by (*metis it-ite ite-neg*)

lemma *it-import-true*: $-p \rhd x = -p \rhd -p ; x$
by (*metis it-ite ite-import-true*)

lemma *it-context-true*: $-p ; (-p \rhd x) = -p ; x$
by (*metis it-ite ite-context-true*)

lemma *it-context-false*: $--p ; (-p \rhd x) = --p$
by (*metis it-ite ite-context-false mult-right-one*)

lemma *while-unfold-it*: $-p \star x = -p \rhd x ; (-p \star x)$
by (*metis circ-loop-fixpoint it-def mult-associative while-def*)

lemma *while-context-false*: $--p ; (-p \star x) = --p$
by (*metis it-context-false while-unfold-it*)

lemma *while-context-true*: $-p ; (-p \star x) = -p ; x ; (-p \star x)$
by (*metis it-context-true mult-associative while-unfold-it*)

lemma *while-zero*: $0 \star x = 1$
by (*metis circ-zero mult-left-one mult-left-zero one-def while-def*)

lemma *wnf-lemma-7*: $1 ; (0 \star 1) = 1$
by (*metis mult-left-one while-zero*)

lemma *while-import-condition*: $-p \star x = -p \star -p ; x$
by (*metis mult-associative mult-idempotent while-def*)

lemma *while-import-condition-2*: $-p ; -q \star x = -p ; -q \star -p ; x$
by (*metis mult-associative mult-idempotent sub-comm while-def*)

lemma *wnf-lemma-8*: $-r ; (-p + --p ; -q) \star (x \lhd --p ; -q \rhd y) = -r ; (-p + -q) \star (y \lhd -p \rhd x)$
by (*metis add-commutative double-negation mult-associative plus-compl-intro while-def wnf-lemma-6*)

— Theorem 6 - see Theorem 31 on page 329 of Back and von Wright, Acta Informatica 36:295-334, 1999

lemma *split-merge-loops*: $--p ; y \leq y ; --p \longrightarrow (-p + -q) \star (x \lhd -p \rhd y) = (-p \star x) ; (-q \star y)$
proof –
have $-p + -q \star (x \lhd -p \rhd y) = (-p ; x + --p ; -q ; y)^\circ ; --p ; --q$
by (*smt ite-def mult-associative plus-comm plus-deMorgan while-def wnf-lemma-5*)
thus *?thesis*
by (*smt circ-add-1 circ-slide import-test mult-associative preserves-equation sub-comm while-context-false while-def*)
qed

lemma *assigns-same*: *assigns* x $(-p)$ $(-p)$
by (*metis assigns-def mult-idempotent mult-right-one plus-compl*)

lemma *preserves-equation-test*: $preserves\ x\ (-p) \longleftrightarrow -p\ ;\ x = -p\ ;\ x\ ;\ -p \wedge --p\ ;\ x = --p\ ;\ x\ ;\ --p$
  by (*metis preserves-equation preserves-def*)

lemma *preserves-test*: $preserves\ (-q)\ (-p)$
  by (*metis order-refl preserves-def sub-comm*)

lemma *preserves-zero*: $preserves\ 0\ (-p)$
  by (*metis one-compl preserves-test*)

lemma *preserves-one*: $preserves\ 1\ (-p)$
  by (*metis one-def preserves-test*)

lemma *preserves-add*: $preserves\ x\ (-p) \wedge preserves\ y\ (-p) \longrightarrow preserves\ (x + y)\ (-p)$
  by (*smt mult-left-dist-add mult-right-dist-add preserves-equation-test*)

lemma *preserves-mult*: $preserves\ x\ (-p) \wedge preserves\ y\ (-p) \longrightarrow preserves\ (x\ ;\ y)\ (-p)$
  by (*smt mult-associative preserves-equation-test*)

lemma *preserves-ite*: $preserves\ x\ (-p) \wedge preserves\ y\ (-p) \longrightarrow preserves\ (x \lhd -q \rhd y)\ (-p)$
  by (*metis ite-def preserves-add preserves-mult preserves-test*)

lemma *preserves-it*: $preserves\ x\ (-p) \longrightarrow preserves\ (-q \rhd x)\ (-p)$
  by (*metis it-def preserves-add preserves-mult preserves-test*)

lemma *preserves-circ*: $preserves\ x\ (-p) \longrightarrow preserves\ (x^\circ)\ (-p)$
  by (*metis circ-simulate preserves-def*)

lemma *preserves-while*: $preserves\ x\ (-p) \longrightarrow preserves\ (-q \star x)\ (-p)$
  by (*metis preserves-circ preserves-mult preserves-test while-def*)

lemma *preserves-test-neg*: $preserves\ x\ (-p) \longrightarrow preserves\ x\ (--p)$
  by (*metis double-negation preserves-def*)

lemma *preserves-import-circ*: $preserves\ x\ (-p) \longrightarrow -p\ ;\ x^\circ = -p\ ;\ (-p\ ;\ x)^\circ$
  by (*metis import-test preserves-def*)

lemma *preserves-simulate*: $preserves\ x\ (-p) \longrightarrow -p\ ;\ x^\circ = -p\ ;\ x^\circ\ ;\ -p$
  by (*metis preserves-circ preserves-equation-test*)

lemma *preserves-import-ite*: $preserves\ z\ (-p) \longrightarrow z\ ;\ (x \lhd -p \rhd y) = z\ ;\ x \lhd -p \rhd z\ ;\ y$
proof −
  have *1*: $preserves\ z\ (-p) \longrightarrow -p\ ;\ z\ ;\ (x \lhd -p \rhd y) = -p\ ;\ (z\ ;\ x \lhd -p \rhd z\ ;\ y)$
    by (*metis add-right-zero ite-def mult-associative mult-compl mult-idempotent mult-left-dist-add mult-left-zero preserves-equation-test*)
  have $preserves\ z\ (-p) \longrightarrow --p\ ;\ z\ ;\ (x \lhd -p \rhd y) = --p\ ;\ (z\ ;\ x \lhd -p \rhd z\ ;\ y)$
    by (*smt add-left-zero ite-def mult-associative mult-compl mult-idempotent mult-left-dist-add mult-left-zero preserves-equation-test sub-comm*)
  thus *?thesis* **using** *1*
    by (*metis test-case-split-left-equal mult-associative*)
qed

lemma *preserves-while-context*: $preserves\ x\ (-p) \longrightarrow -p\ ;\ (-q \star x) = -p\ ;\ (-p\ ;\ -q \star x)$
  by (*smt mult-associative mult-compl-intro mult-deMorgan preserves-import-circ preserves-mult preserves-simulate preserves-test while-def*)

lemma *while-ite-context-false*: $preserves\ y\ (-p) \longrightarrow --p\ ;\ (-p + -q \star (x \lhd -p \rhd y)) = --p\ ;\ (-q \star y)$
proof −
  have $preserves\ y\ (-p) \longrightarrow --p\ ;\ (-p + -q \star (x \lhd -p \rhd y)) = --p\ ;\ (--p\ ;\ -q\ ;\ y)^\circ\ ;\ -(-p + -q)$
    by (*smt import-test double-negation ite-context-false mult-associative mult-compl-intro plus-def preserves-equation preserves-equation-test sub-comm while-def*)
  thus *?thesis*
    by (*smt import-test circ-simulate mult-associative plus-deMorgan preserves-def preserves-equation preserves-test while-def*)
qed

— Theorem 7.1

lemma *while-ite-norm*: $assigns\ z\ (-p)\ (-q) \wedge preserves\ x1\ (-q) \wedge preserves\ x2\ (-q) \wedge preserves\ y1\ (-q) \wedge preserves\ y2\ (-q) \longrightarrow$
  $z\ ;\ (x1\ ;\ (-r1 \star y1) \lhd -p \rhd x2\ ;\ (-r2 \star y2)) = z\ ;\ (x1 \lhd -q \rhd x2)\ ;\ ((-q\ ;\ -r1 + --q\ ;\ -r2) \star (y1 \lhd$

$-q \rhd y2))$
proof
assume *1*: *assigns* $z\ (-p)\ (-q) \wedge$ *preserves* $x1\ (-q) \wedge$ *preserves* $x2\ (-q) \wedge$ *preserves* $y1\ (-q) \wedge$ *preserves* $y2\ (-q)$
have *2*: $(-p\ ;\ -q + --p\ ;\ --q)\ ;\ (x1 \lhd -q \rhd x2) = -p\ ;\ -q\ ;\ x1 + --p\ ;\ --q\ ;\ x2$
by (*smt ite-def mult-associative mult-left-dist-add wnf-lemma-2 wnf-lemma-4*)
have *3*: $(-q\ ;\ -r1 + --q\ ;\ -r2)\ ;\ (y1 \lhd -q \rhd y2) = -q\ ;\ -r1\ ;\ y1 + --q\ ;\ -r2\ ;\ y2$
by (*smt ite-def mult-associative mult-idempotent mult-left-dist-add wnf-lemma-1 wnf-lemma-3*)
have *4*: $-(-q\ ;\ -r1 + --q\ ;\ -r2) = -q\ ;\ --r1 + --q\ ;\ --r2$
by (*smt mult-absorb mult-idempotent mult-right-dist-add plus-compl-intro plus-deMorgan plus-def sub-comm*)
have $-p\ ;\ -q\ ;\ x1\ ;\ (-q\ ;\ -r1\ ;\ y1 + --q\ ;\ -r2\ ;\ y2)^\circ\ ;\ (-q\ ;\ --r1 + --q\ ;\ --r2) =$
$-p\ ;\ -q\ ;\ x1\ ;\ -q\ ;\ (-q\ ;\ (-q\ ;\ -r1\ ;\ y1 + --q\ ;\ -r2\ ;\ y2))^\circ\ ;\ (-q\ ;\ --r1 + --q\ ;\ --r2)$ **using** *1*
by (*smt mult-associative preserves-add preserves-equation-test preserves-import-circ preserves-mult preserves-test*)
also have ... $= -p\ ;\ -q\ ;\ x1\ ;\ -q\ ;\ (-q\ ;\ -r1\ ;\ y1)^\circ\ ;\ (-q\ ;\ --r1 + --q\ ;\ --r2)$
by (*smt add-commutative add-left-zero mult-associative mult-compl mult-idempotent mult-left-dist-add mult-left-zero*)
finally have *5*: $-p\ ;\ -q\ ;\ x1\ ;\ (-q\ ;\ -r1\ ;\ y1 + --q\ ;\ -r2\ ;\ y2)^\circ\ ;\ (-q\ ;\ --r1 + --q\ ;\ --r2) =$
$-p\ ;\ -q\ ;\ x1\ ;\ (-r1\ ;\ y1)^\circ\ ;\ --r1$ **using** *1*
by (*smt ite-context-true ite-def mult-associative preserves-equation-test preserves-import-circ preserves-mult preserves-simulate preserves-test*)
have $--p\ ;\ --q\ ;\ x2\ ;\ (-q\ ;\ -r1\ ;\ y1 + --q\ ;\ -r2\ ;\ y2)^\circ\ ;\ (-q\ ;\ --r1 + --q\ ;\ --r2) =$
$--p\ ;\ --q\ ;\ x2\ ;\ --q\ ;\ (--q\ ;\ (-q\ ;\ -r1\ ;\ y1 + --q\ ;\ -r2\ ;\ y2))^\circ\ ;\ (-q\ ;\ --r1 + --q\ ;\ --r2)$ **using** *1*
by (*smt mult-associative preserves-add preserves-equation-test preserves-import-circ preserves-mult preserves-test preserves-test-neg*)
also have ... $= --p\ ;\ --q\ ;\ x2\ ;\ --q\ ;\ (--q\ ;\ -r2\ ;\ y2)^\circ\ ;\ (-q\ ;\ --r1 + --q\ ;\ --r2)$
by (*smt add-commutative add-right-zero mult-associative mult-compl mult-idempotent mult-left-dist-add mult-left-zero sub-comm*)
finally have $--p\ ;\ --q\ ;\ x2\ ;\ (-q\ ;\ -r1\ ;\ y1 + --q\ ;\ -r2\ ;\ y2)^\circ\ ;\ (-q\ ;\ --r1 + --q\ ;\ --r2) =$
$--p\ ;\ --q\ ;\ x2\ ;\ (-r2\ ;\ y2)^\circ\ ;\ --r2$ **using** *1*
by (*smt ite-context-false ite-def mult-associative preserves-equation-test preserves-import-circ preserves-mult preserves-simulate preserves-test preserves-test-neg*)
thus $z\ ;\ (x1\ ;\ (-r1 \star y1) \lhd -p \rhd x2\ ;\ (-r2 \star y2)) = z\ ;\ (x1 \lhd -q \rhd x2)\ ;\ ((-q\ ;\ -r1 + --q\ ;\ -r2) \star (y1 \lhd -q \rhd y2))$
using *1 2 3 4 5*
by (*smt assigns-def ite-def mult-associative mult-left-dist-add mult-right-dist-add wnf-lemma-1 wnf-lemma-3 while-def*)
qed

lemma *while-it-norm*: *assigns* $z\ (-p)\ (-q) \wedge$ *preserves* $x\ (-q) \wedge$ *preserves* $y\ (-q) \longrightarrow z\ ;\ (-p \rhd x\ ;\ (-r \star y)) = z\ ;\ (-q \rhd x)\ ;\ (-q\ ;\ -r \star y)$
by (*metis add-right-zero bs-mult-right-zero it-context-true it-ite one-compl preserves-one while-import-condition-2 while-ite-norm wnf-lemma-7*)

lemma *while-else-norm*: *assigns* $z\ (-p)\ (-q) \wedge$ *preserves* $x\ (-q) \wedge$ *preserves* $y\ (-q) \longrightarrow z\ ;\ (1 \lhd -p \rhd x\ ;\ (-r \star y)) = z\ ;\ (1 \lhd -q \rhd x)\ ;\ (--q\ ;\ -r \star y)$
by (*metis add-left-zero bs-mult-right-zero ite-context-false one-compl preserves-one while-import-condition-2 while-ite-norm wnf-lemma-7*)

lemma *while-while-pre-norm*: $-p \star x\ ;\ (-q \star y) = -p \rhd x\ ;\ (-p + -q \star (y \lhd -q \rhd x))$
by (*smt add-commutative circ-add-1 circ-left-unfold circ-slide it-def ite-def mult-associative mult-left-one mult-right-dist-add plus-deMorgan while-def wnf-lemma-5*)

— Theorem 7.2

lemma *while-while-norm*: *assigns* $z\ (-p)\ (-r) \wedge$ *preserves* $x\ (-r) \wedge$ *preserves* $y\ (-r) \longrightarrow$
$z\ ;\ (-p \star x\ ;\ (-q \star y)) = z\ ;\ (-r \rhd x)\ ;\ (-r\ ;\ (-p + -q) \star (y \lhd -q \rhd x))$
by (*smt double-negation mult-deMorgan plus-deMorgan preserves-ite while-it-norm while-while-pre-norm*)

lemma *while-seq-replace*: *assigns* $z\ (-p)\ (-q) \longrightarrow z\ ;\ (-p \star x\ ;\ z)\ ;\ y = z\ ;\ (-q \star x\ ;\ z)\ ;\ y$
by (*smt assigns-def circ-slide mult-associative wnf-lemma-1 wnf-lemma-2 wnf-lemma-3 wnf-lemma-4 while-def*)

lemma *while-ite-replace*: *assigns* $z\ (-p)\ (-q) \longrightarrow z\ ;\ (x \lhd -p \rhd y) = z\ ;\ (x \lhd -q \rhd y)$
by (*smt assigns-def ite-def mult-associative mult-left-dist-add sub-comm wnf-lemma-1 wnf-lemma-3*)

lemma *while-post-norm-an*: *preserves* $y\ (-p) \longrightarrow (-p \star x)\ ;\ y = y \lhd --p \rhd (-p \star x\ ;\ (--p \rhd y))$
proof
assume *1*: *preserves* $y\ (-p)$
have $-p\ ;\ (-p\ ;\ x\ ;\ (--p\ ;\ y + -p))^\circ\ ;\ --p = -p\ ;\ x\ ;\ ((--p\ ;\ y + -p)\ ;\ -p\ ;\ x)^\circ\ ;\ (--p\ ;\ y + -p)\ ;\ --p$
by (*smt add-left-zero circ-left-unfold circ-slide mult-associative mult-compl mult-idempotent mult-left-dist-add mult-right-dist-add mult-right-one*)
also have ... $= -p\ ;\ x\ ;\ (--p\ ;\ y\ ;\ 0 + -p\ ;\ x)^\circ\ ;\ --p\ ;\ y$ **using** *1*
by (*smt add-right-zero mult-associative mult-compl mult-idempotent mult-left-zero mult-right-dist-add preserves-equation-test sub-comm*)

finally have $-p ; (-p ; x ; (--p ; y + -p))^{\circ} ; --p = -p ; x ; (-p ; x)^{\circ} ; --p ; y$
by (*smt add-commutative circ-slide circ-zero mult-associative mult-left-zero mult-right-one mult-zero-add-circ*)
thus $(-p \star x) ; y = y \lhd --p \rhd (-p \star x ; (--p \rhd y))$
by (*smt circ-left-unfold double-negation it-def ite-def mult-associative mult-left-one mult-right-dist-add while-def*)
qed

lemma *while-post-norm*: $preserves\ y\ (-p) \longrightarrow (-p \star x) ; y = -p \star x ; (1 \lhd -p \rhd y) \lhd -p \rhd y$
by (*metis it-neg ite-neg while-post-norm-an*)

lemma *wnf-lemma-9*: $assigns\ z\ (-p)\ (-q) \land preserves\ x1\ (-q) \land preserves\ y1\ (-q) \land preserves\ x2\ (-q) \land preserves\ y2\ (-q) \land preserves\ x2\ (-p) \land preserves\ y2\ (-p) \longrightarrow$
$z ; (x1 \lhd -q \rhd x2) ; (-q ; -p + -r \star (y1 \lhd -q ; -p \rhd y2)) = z ; (x1 \lhd -p \rhd x2) ; (-p + -r \star (y1 \lhd -p \rhd y2))$
proof
assume *1*: $assigns\ z\ (-p)\ (-q) \land preserves\ x1\ (-q) \land preserves\ y1\ (-q) \land preserves\ x2\ (-q) \land preserves\ y2\ (-q) \land preserves\ x2\ (-p) \land preserves\ y2\ (-p)$
hence $z ; --p ; --q ; (x1 \lhd -q \rhd x2) ; (-q ; -p + -r \star (y1 \lhd -q ; -p \rhd y2)) =$
$z ; --p ; --q ; x2 ; --q ; (--q ; (-q ; -p + -r) \star (y1 \lhd -q ; -p \rhd y2))$
by (*smt double-negation ite-context-false mult-associative mult-deMorgan plus-deMorgan preserves-equation-test preserves-ite preserves-while-context*)
also have ... $= z ; --p ; --q ; x2 ; --q ; (--q ; -r \star --q ; y2)$
by (*smt add-left-zero double-negation ite-conjunction ite-context-false mult-associative mult-compl mult-left-dist-add mult-left-zero while-import-condition-2*)
also have ... $= z ; --p ; --q ; x2 ; (-r \star y2)$ **using** *1*
by (*smt mult-associative preserves-equation-test preserves-test-neg preserves-while-context while-import-condition-2*)
finally have *2*: $z ; --p ; --q ; (x1 \lhd -q \rhd x2) ; (-q ; -p + -r \star (y1 \lhd -q ; -p \rhd y2)) =$
$z ; --p ; --q ; (x1 \lhd -q \rhd x2) ; (-p + -r \star (y1 \lhd -p \rhd y2))$ **using** *1*
by (*smt ite-context-false mult-associative preserves-equation-test sub-comm while-ite-context-false*)
have $z ; -p ; -q ; (x1 \lhd -q \rhd x2) ; (-q ; -p + -r \star (y1 \lhd -q ; -p \rhd y2)) =$
$z ; -p ; -q ; (x1 \lhd -q \rhd x2) ; -q ; (-q ; (-p + -r) \star -q ; (y1 \lhd -p \rhd y2))$ **using** *1*
by (*smt double-negation ite-context-import mult-associative mult-deMorgan mult-idempotent mult-left-dist-add plus-deMorgan preserves-equation-test preserves-ite preserves-while-context while-import-condition-2*)
hence $z ; -p ; -q ; (x1 \lhd -q \rhd x2) ; (-q ; -p + -r \star (y1 \lhd -q ; -p \rhd y2)) =$
$z ; -p ; -q ; (x1 \lhd -q \rhd x2) ; (-p + -r \star (y1 \lhd -p \rhd y2))$ **using** *1*
by (*smt double-negation mult-associative mult-deMorgan mult-idempotent preserves-equation-test preserves-ite preserves-while-context while-import-condition-2*)
thus $z ; (x1 \lhd -q \rhd x2) ; (-q ; -p + -r \star (y1 \lhd -q ; -p \rhd y2)) = z ; (x1 \lhd -p \rhd x2) ; (-p + -r \star (y1 \lhd -p \rhd y2))$ **using** *1 2*
by (*smt assigns-def mult-associative mult-left-dist-add mult-right-dist-add while-ite-replace*)
qed

— Theorem 7.3

lemma *while-seq-norm*: $assigns\ z1\ (-r1)\ (-q) \land preserves\ x2\ (-q) \land preserves\ y2\ (-q) \land preserves\ z2\ (-q) \land z1 ; z2 = z2 ; z1 \land$
$assigns\ z2\ (-q)\ (-r) \land preserves\ y1\ (-r) \land preserves\ z1\ (-r) \land preserves\ x2\ (-r) \land preserves\ y2\ (-r) \longrightarrow$
$x1 ; z1 ; z2 ; (-r1 \star y1 ; z1) ; x2 ; (-r2 \star y2) =$
$x1 ; z1 ; z2 ; (y1 ; z1 ; (1 \lhd -q \rhd x2) \lhd -q \rhd x2) ; (-q + -r2 \star (y1 ; z1 ; (1 \lhd -q \rhd x2) \lhd -q \rhd y2))$
proof
assume *1*: $assigns\ z1\ (-r1)\ (-q) \land preserves\ x2\ (-q) \land preserves\ y2\ (-q) \land preserves\ z2\ (-q) \land z1 ; z2 = z2 ; z1 \land$
$assigns\ z2\ (-q)\ (-r) \land preserves\ y1\ (-r) \land preserves\ z1\ (-r) \land preserves\ x2\ (-r) \land preserves\ y2\ (-r)$
have $x1 ; z1 ; z2 ; (-r1 \star y1 ; z1) ; x2 ; (-r2 \star y2) = x1 ; z1 ; z2 ; (-q \star y1 ; z1) ; x2 ; (-r2 \star y2)$ **using** *1*
by (*smt mult-associative while-seq-replace*)
also have ... $= x1 ; z1 ; z2 ; (-q \star y1 ; z1 ; (1 \lhd -q \rhd x2 ; (-r2 \star y2)) \lhd -q \rhd x2 ; (-r2 \star y2))$ **using** *1*
by (*smt mult-associative preserves-mult preserves-while while-post-norm*)
also have ... $= x1 ; z1 ; (z2 ; (-q \star y1 ; z1 ; (1 \lhd -q \rhd x2) ; (--q ; -r2 \star y2)) \lhd -q \rhd z2 ; x2 ; (-r2 \star y2))$ **using** *1*
by (*smt assigns-same mult-associative preserves-import-ite while-else-norm*)
also have ... $= x1 ; z1 ; (z2 ; (-r \rhd y1 ; z1 ; (1 \lhd -q \rhd x2)) ; (-r ; (-q + -r2) \star (y1 ; z1 ; (1 \lhd -q \rhd x2) \lhd -q \rhd y2)) \lhd -q \rhd z2 ; x2 ; (-r2 \star y2))$ **using** *1*
by (*smt double-negation mult-deMorgan plus-deMorgan preserves-ite preserves-mult preserves-one while-while-norm wnf-lemma-8*)
also have ... $= x1 ; z1 ; z2 ; ((-r \rhd y1 ; z1 ; (1 \lhd -q \rhd x2)) ; (-r ; (-q + -r2) \star (y1 ; z1 ; (1 \lhd -q \rhd x2) \lhd -q \rhd y2)) \lhd -r \rhd x2 ; (-r2 \star y2))$ **using** *1*
by (*smt mult-associative preserves-import-ite while-ite-replace*)
also have ... $= x1 ; z1 ; z2 ; (-r ; y1 ; z1 ; (1 \lhd -q \rhd x2) ; -r ; (-q + -r2 \star (y1 ; z1 ; (1 \lhd -q \rhd x2) \lhd -q \rhd y2)) \lhd -r \rhd x2 ; (-r2 \star y2))$ **using** *1*
by (*smt double-negation it-context-true ite-import-true mult-associative mult-deMorgan mult-idempotent plus-deMorgan preserves-equation-test preserves-ite preserves-mult preserves-one preserves-while-context*)
also have ... $= x1 ; z1 ; z2 ; (y1 ; z1 ; (1 \lhd -q \rhd x2) ; (-q + -r2 \star (y1 ; z1 ; (1 \lhd -q \rhd x2) \lhd -q \rhd y2)) \lhd -q \rhd x2$

; $(-r2 \star y2)$) **using** *1*
  by (*smt double-negation ite-import-true mult-associative mult-idempotent preserves-equation-test preserves-ite preserves-one while-ite-replace*)
 also have ... $= x1 \; ; z1 \; ; z2 \; ; (y1 \; ; z1 \; ; (1 \lhd -q \rhd x2) \lhd -r \rhd x2) \; ; ((-r \; ; (-q + -r2) + --r \; ; -r2) \star ((y1 \; ; z1 \; ; (1 \lhd -q \rhd x2) \lhd -q \rhd y2) \lhd -r \rhd y2))$ **using** *1*
  by (*smt double-negation mult-associative mult-deMorgan plus-deMorgan preserves-ite preserves-mult preserves-one while-ite-norm*)
 also have ... $= x1 \; ; z1 \; ; z2 \; ; (y1 \; ; z1 \; ; (1 \lhd -q \rhd x2) \lhd -q \rhd x2) \; ; (-q + -r2 \star (y1 \; ; z1 \; ; (1 \lhd -q \rhd x2) \lhd -q \rhd y2))$ **using** *1*
  by (*smt add-associative ite-conjunction mult-associative mult-left-dist-add mult-left-one mult-right-dist-add plus-compl preserves-ite preserves-mult preserves-one wnf-lemma-9*)
 finally show $x1 \; ; z1 \; ; z2 \; ; (-r1 \star y1 \; ; z1) \; ; x2 \; ; (-r2 \star y2) =$
  $x1 \; ; z1 \; ; z2 \; ; (y1 \; ; z1 \; ; (1 \lhd -q \rhd x2) \lhd -q \rhd x2) \; ; (-q + -r2 \star (y1 \; ; z1 \; ; (1 \lhd -q \rhd x2) \lhd -q \rhd y2))$ **.**
qed

end

end

11 BinaryItering

theory *BinaryItering*

imports *Semiring*

begin

class *binary-itering* = *idempotent-left-zero-semiring* + *while* +
  assumes *while-productstar*: $(x ; y) \star z = z + x ; ((y ; x) \star (y ; z))$
  assumes *while-sumstar*: $(x + y) \star z = (x \star y) \star (x \star z)$
  assumes *while-left-dist-add*: $x \star (y + z) = (x \star y) + (x \star z)$
  assumes *while-sub-associative*: $(x \star y) ; z \le x \star (y ; z)$
  assumes *while-simulate-left-plus*: $x ; z \le z ; (y \star 1) + w \longrightarrow x \star (z ; v) \le z ; (y \star v) + (x \star (w ; (y \star v)))$
  assumes *while-simulate-right-plus*: $z ; x \le y ; (y \star z) + w \longrightarrow z ; (x \star v) \le y \star (z ; v + w ; (x \star v))$

begin

— Theorem 9.1

lemma *while-zero*: $0 \star x = x$
  by (*metis add-right-zero mult-left-zero while-productstar*)

— Theorem 9.4

lemma *while-mult-increasing*: $x ; y \le x \star y$
  by (*metis add-least-upper-bound mult-left-one order-refl while-productstar*)

— Theorem 9.2

lemma *while-one-increasing*: $x \le x \star 1$
  by (*metis mult-right-one while-mult-increasing*)

— Theorem 9.3

lemma *while-increasing*: $y \le x \star y$
  by (*metis add-left-divisibility mult-left-one while-productstar*)

— Theorem 9.42

lemma *while-right-isotone*: $y \le z \longrightarrow x \star y \le x \star z$
  by (*metis less-eq-def while-left-dist-add*)

— Theorem 9.41

lemma *while-left-isotone*: $x \le y \longrightarrow x \star z \le y \star z$
  by (*metis less-eq-def while-increasing while-sumstar*)

lemma *while-isotone*: $w \le x \wedge y \le z \longrightarrow w \star y \le x \star z$
  by (*smt order-trans while-left-isotone while-right-isotone*)

— Theorem 9.17

lemma *while-left-unfold*: $x \star y = y + x ; (x \star y)$
  by (*metis mult-left-one mult-right-one while-productstar*)

lemma *while-simulate-left-plus-1*: $x ; z \le z ; (y \star 1) \longrightarrow x \star (z ; w) \le z ; (y \star w) + (x \star 0)$
  by (*metis add-right-zero mult-left-zero while-simulate-left-plus*)

— Theorem 11.1

lemma *while-simulate-absorb*: $y ; x \le x \longrightarrow y \star x \le x + (y \star 0)$
  by (*metis while-simulate-left-plus-1 while-zero mult-right-one*)

— Theorem 9.10

lemma *while-transitive*: $x \star (x \star y) = x \star y$
  by (*metis add-right-upper-bound add-right-zero antisym while-increasing while-left-dist-add while-left-unfold while-simulate-absorb*)

— Theorem 9.25

lemma *while-slide*: $(x\ ;\ y) \star (x\ ;\ z) = x\ ;\ ((y\ ;\ x) \star z)$
  by (*metis mult-associative mult-left-dist-add while-left-unfold while-productstar*)

— Theorem 9.21

lemma *while-zero-2*: $(x\ ;\ 0) \star y = x\ ;\ 0 + y$
  by (*metis add-commutative mult-associative mult-left-zero while-left-unfold*)

— Theorem 9.5

lemma *while-mult-star-exchange*: $x\ ;\ (x \star y) = x \star (x\ ;\ y)$
  by (*metis mult-left-one while-slide*)

— Theorem 9.18

lemma *while-right-unfold*: $x \star y = y + (x \star (x\ ;\ y))$
  by (*metis while-left-unfold while-mult-star-exchange*)

— Theorem 9.7

lemma *while-one-mult-below*: $(x \star 1)\ ;\ y \le x \star y$
  by (*metis mult-left-one while-sub-associative*)

lemma *while-plus-one*: $x \star y = y + (x \star y)$
  by (*metis less-eq-def while-increasing*)

— Theorem 9.19

lemma *while-rtc-2*: $y + x\ ;\ y + (x \star (x \star y)) = x \star y$
  by (*metis add-associative less-eq-def while-mult-increasing while-plus-one while-transitive*)

— Theorem 9.6

lemma *while-left-plus-below*: $x\ ;\ (x \star y) \le x \star y$
  by (*metis add-right-divisibility while-left-unfold*)

lemma *while-right-plus-below*: $x \star (x\ ;\ y) \le x \star y$
  by (*metis while-left-plus-below while-mult-star-exchange*)

lemma *while-right-plus-below-2*: $(x \star x)\ ;\ y \le x \star y$
  by (*smt order-trans while-right-plus-below while-sub-associative*)

— Theorem 9.47

lemma *while-mult-transitive*: $x \le z \star y \wedge y \le z \star w \longrightarrow x \le z \star w$
  by (*smt order-trans while-right-isotone while-transitive*)

— Theorem 9.48

lemma *while-mult-upper-bound*: $x \le z \star 1 \wedge y \le z \star w \longrightarrow x\ ;\ y \le z \star w$
  by (*metis less-eq-def mult-right-sub-dist-add-left order-trans while-mult-transitive while-one-mult-below*)

lemma *while-one-mult-while-below*: $(y \star 1)\ ;\ (y \star v) \le y \star v$
  by (*metis order-refl while-mult-upper-bound*)

— Theorem 9.34

lemma *while-sub-dist*: $x \star z \le (x + y) \star z$
  by (*metis add-left-upper-bound while-left-isotone*)

lemma *while-sub-dist-1*: $x\ ;\ z \le (x + y) \star z$
  by (*metis order-trans while-mult-increasing while-sub-dist*)

lemma *while-sub-dist-2*: $x\ ;\ y\ ;\ z \le (x + y) \star z$
  by (*metis add-commutative mult-associative while-mult-transitive while-sub-dist-1*)

— Theorem 9.36

lemma *while-sub-dist-3*: $x \star (y \star z) \le (x + y) \star z$
by (*metis add-right-upper-bound while-left-isotone while-mult-transitive while-sub-dist*)

— Theorem 9.44

lemma *while-absorb-2*: $x \le y \longrightarrow y \star (x \star z) = y \star z$
by (*metis add-commutative less-eq-def while-left-dist-add while-plus-one while-sub-dist-3*)

lemma *while-simulate-right-plus-1*: $z \mathbin{;} x \le y \mathbin{;} (y \star z) \longrightarrow z \mathbin{;} (x \star w) \le y \star (z \mathbin{;} w)$
by (*metis add-right-zero mult-left-zero while-simulate-right-plus*)

— Theorem 9.39

lemma *while-sumstar-1-below*: $x \star ((y \mathbin{;} (x \star 1)) \star z) \le ((x \star 1) \mathbin{;} y) \star (x \star z)$
proof –
have *1*: $x \mathbin{;} (((x \star 1) \mathbin{;} y) \star (x \star z)) \le ((x \star 1) \mathbin{;} y) \star (x \star z)$
by (*smt add-isotone add-right-upper-bound mult-associative mult-left-dist-add mult-right-sub-dist-add-right while-left-unfold*)
have $x \star ((y \mathbin{;} (x \star 1)) \star z) \le (x \star z) + (x \star (y \mathbin{;} (((x \star 1) \mathbin{;} y) \star ((x \star 1) \mathbin{;} z))))$
by (*metis eq-refl while-left-dist-add while-productstar*)
also have ... $\le (x \star z) + (x \star ((x \star 1) \mathbin{;} y \mathbin{;} (((x \star 1) \mathbin{;} y) \star ((x \star 1) \mathbin{;} z))))$
by (*metis add-right-isotone mult-associative mult-left-one mult-right-sub-dist-add-left while-left-unfold while-right-isotone*)
also have ... $\le (x \star z) + (x \star (((x \star 1) \mathbin{;} y) \star ((x \star 1) \mathbin{;} z)))$
by (*metis add-right-isotone add-right-upper-bound while-left-unfold while-right-isotone*)
also have ... $\le x \star (((x \star 1) \mathbin{;} y) \star (x \star z))$
by (*smt add-associative add-left-upper-bound less-eq-def mult-left-one while-left-dist-add while-left-unfold while-sub-associative*)
also have ... $\le (((x \star 1) \mathbin{;} y) \star (x \star z)) + (x \star 0)$ **using** *1*
by (*metis while-simulate-absorb*)
also have ... $= ((x \star 1) \mathbin{;} y) \star (x \star z)$
by (*smt add-associative add-commutative add-left-zero while-left-dist-add while-left-unfold*)
finally show *?thesis*
.
qed

lemma *while-sumstar-2-below*: $((x \star 1) \mathbin{;} y) \star (x \star z) \le (x \star y) \star (x \star z)$
by (*metis mult-left-one while-left-isotone while-sub-associative*)

— Theorem 9.38

lemma *while-add-1-below*: $x \star ((y \mathbin{;} (x \star 1)) \star z) \le (x + y) \star z$
proof –
have $((x \star 1) \mathbin{;} y) \star ((x \star 1) \mathbin{;} z) \le (x + y) \star z$
by (*metis while-isotone while-one-mult-below while-sumstar*)
hence $(y \mathbin{;} (x \star 1)) \star z \le z + y \mathbin{;} ((x + y) \star z)$
by (*metis add-right-isotone mult-right-isotone while-productstar*)
also have ... $\le (x + y) \star z$
by (*metis add-right-isotone add-right-upper-bound mult-left-isotone while-left-unfold*)
finally show *?thesis*
by (*metis add-commutative add-right-upper-bound while-isotone while-transitive*)
qed

— Theorem 9.16

lemma *while-while-while*: $((x \star 1) \star 1) \star y = (x \star 1) \star y$
by (*smt add-commutative less-eq-def mult-right-one while-left-plus-below while-mult-star-exchange while-plus-one while-sumstar while-transitive*)

lemma *while-one*: $(1 \star 1) \star y = 1 \star y$
by (*metis while-while-while while-zero*)

— Theorem 9.22

lemma *while-add-below*: $x + y \le x \star (y \star 1)$
by (*smt add-commutative add-isotone case-split-right order-trans while-increasing while-left-plus-below while-mult-increasing while-plus-one*)

— Theorem 9.32

lemma *while-add-2*: $(x + y) \star z \le (x \star (y \star 1)) \star z$

by (*metis while-add-below while-left-isotone*)

— Theorem 9.45

lemma *while-sup-one-left-unfold*: $1 \le x \longrightarrow x \; ; \; (x \star y) = x \star y$
by (*metis less-eq-def mult-left-one mult-right-dist-add while-mult-star-exchange while-right-unfold while-transitive*)

lemma *while-sup-one-right-unfold*: $1 \le x \longrightarrow x \star (x \; ; \; y) = x \star y$
by (*metis while-mult-star-exchange while-sup-one-left-unfold*)

— Theorem 9.30

lemma *while-decompose-7*: $(x + y) \star z = x \star (y \star ((x + y) \star z))$
by (*metis eq-iff order-trans while-increasing while-sub-dist-3 while-transitive*)

— Theorem 9.31

lemma *while-decompose-8*: $(x + y) \star z = (x + y) \star (x \star (y \star z))$
by (*metis add-commutative while-sumstar while-transitive*)

— Theorem 9.27

lemma *while-decompose-9*: $(x \star (y \star 1)) \star z = x \star (y \star ((x \star (y \star 1)) \star z))$
by (*smt add-commutative less-eq-def order-trans while-add-below while-increasing while-sub-dist-3*)

lemma *while-decompose-10*: $(x \star (y \star 1)) \star z = (x \star (y \star 1)) \star (x \star (y \star z))$
proof −
have *1*: $(x \star (y \star 1)) \star z \le (x \star (y \star 1)) \star (x \star (y \star z))$
by (*metis add-associative less-eq-def while-left-dist-add while-plus-one*)
have $x + (y \star 1) \le x \star (y \star 1)$
by (*metis add-least-upper-bound while-add-below while-increasing*)
hence $(x \star (y \star 1)) \star (x \star (y \star z)) \le (x \star (y \star 1)) \star z$
by (*smt add-least-upper-bound eq-refl order-trans while-absorb-2 while-one-increasing*)
thus *?thesis* **using** *1*
by (*metis antisym*)
qed

lemma *while-back-loop-fixpoint*: $z \; ; \; (y \star (y \; ; \; x)) + z \; ; \; x = z \; ; \; (y \star x)$
by (*metis add-commutative mult-left-dist-add while-right-unfold*)

lemma *while-back-loop-prefixpoint*: $z \; ; \; (y \star 1) \; ; \; y + z \le z \; ; \; (y \star 1)$
by (*metis add-least-upper-bound mult-associative mult-right-isotone mult-right-one order-refl while-increasing while-mult-upper-bound while-one-increasing*)

— Theorem 9

lemma *while-loop-is-fixpoint*: *is-fixpoint* $(\lambda x \, . \, y \; ; \; x + z) \; (y \star z)$
by (*smt add-commutative is-fixpoint-def while-left-unfold*)

— Theorem 9

lemma *while-back-loop-is-prefixpoint*: *is-prefixpoint* $(\lambda x \, . \, x \; ; \; y + z) \; (z \; ; \; (y \star 1))$
by (*metis is-prefixpoint-def while-back-loop-prefixpoint*)

— Theorem 9.20

lemma *while-while-add*: $(1 + x) \star y = (x \star 1) \star y$
by (*metis add-commutative while-decompose-10 while-sumstar while-zero*)

lemma *while-while-mult-sub*: $x \star (1 \star y) \le (x \star 1) \star y$
by (*metis add-commutative while-sub-dist-3 while-while-add*)

— Theorem 9.11

lemma *while-right-plus*: $(x \star x) \star y = x \star y$
by (*metis add-idempotent while-plus-one while-sumstar while-transitive*)

— Theorem 9.12

lemma *while-left-plus*: $(x ; (x \star 1)) \star y = x \star y$
by (*metis mult-right-one while-mult-star-exchange while-right-plus*)

— Theorem 9.9

lemma *while-below-while-one*: $x \star x \leq x \star 1$
by (*metis while-one-increasing while-right-plus*)

lemma *while-below-while-one-mult*: $x ; (x \star x) \leq x ; (x \star 1)$
by (*metis mult-right-isotone while-below-while-one*)

— Theorem 9.23

lemma *while-add-sub-add-one*: $x \star (x + y) \leq x \star (1 + y)$
by (*metis add-left-isotone while-below-while-one while-left-dist-add*)

lemma *while-add-sub-add-one-mult*: $x ; (x \star (x + y)) \leq x ; (x \star (1 + y))$
by (*metis mult-right-isotone while-add-sub-add-one*)

lemma *while-elimination*: $x ; y = 0 \longrightarrow x ; (y \star z) = x ; z$
by (*metis add-right-zero mult-associative mult-left-dist-add mult-left-zero while-left-unfold*)

— Theorem 9.8

lemma *while-square*: $(x ; x) \star y \leq x \star y$
by (*metis while-left-isotone while-mult-increasing while-right-plus*)

— Theorem 9.35

lemma *while-mult-sub-add*: $(x ; y) \star z \leq (x + y) \star z$
by (*metis while-increasing while-isotone while-mult-increasing while-sumstar*)

— Theorem 9.43

lemma *while-absorb-1*: $x \leq y \longrightarrow x \star (y \star z) = y \star z$
by (*metis antisym less-eq-def while-increasing while-sub-dist-3*)

lemma *while-absorb-3*: $x \leq y \longrightarrow x \star (y \star z) = y \star (x \star z)$
by (*metis while-absorb-1 while-absorb-2*)

— Theorem 9.24

lemma *while-square-2*: $(x ; x) \star ((x + 1) ; y) \leq x \star y$
by (*metis add-least-upper-bound while-increasing while-mult-transitive while-mult-upper-bound while-one-increasing while-square*)

lemma *while-separate-unfold-below*: $(y ; (x \star 1)) \star z \leq (y \star z) + (y \star (y ; x ; (x \star ((y ; (x \star 1)) \star z))))$
proof –
have $(y ; (x \star 1)) \star z = (y \star (y ; x ; (x \star 1))) \star (y \star z)$
by (*metis mult-associative mult-left-dist-add mult-right-one while-left-unfold while-sumstar*)
hence $(y ; (x \star 1)) \star z = (y \star z) + (y \star (y ; x ; (x \star 1))) ; ((y ; (x \star 1)) \star z)$
by (*metis while-left-unfold*)
also have ... $\leq (y \star z) + (y \star (y ; x ; (x \star 1)) ; ((y ; (x \star 1)) \star z))$
by (*metis add-right-isotone while-sub-associative*)
also have ... $\leq (y \star z) + (y \star (y ; x ; (x \star ((y ; (x \star 1)) \star z))))$
by (*smt add-right-isotone mult-associative mult-right-isotone while-one-mult-below while-right-isotone*)
finally show *?thesis*
.
qed

— Theorem 9.33

lemma *while-mult-zero-add*: $(x + y ; 0) \star z = x \star ((y ; 0) \star z)$
proof –
have $(x + y ; 0) \star z = (x \star (y ; 0)) \star (x \star z)$
by (*metis while-sumstar*)
also have ... $= (x \star z) + (x \star (y ; 0)) ; ((x \star (y ; 0)) \star (x \star z))$
by (*metis while-left-unfold*)
also have ... $\leq (x \star z) + (x \star (y ; 0))$

by (*metis add-right-isotone mult-associative mult-left-zero while-sub-associative*)
also have ... $= x \star ((y \ ; \ 0) \star z)$
by (*metis add-commutative while-left-dist-add while-zero-2*)
finally show *?thesis*
by (*metis le-neq-trans less-def while-sub-dist-3*)
qed

lemma *while-add-mult-zero*: $(x + y \ ; \ 0) \star y = x \star y$
by (*metis less-eq-def while-mult-zero-add while-zero-2 zero-right-mult-decreasing*)

lemma *while-mult-zero-add-2*: $(x + y \ ; \ 0) \star z = (x \star z) + (x \star (y \ ; \ 0))$
by (*metis add-commutative while-left-dist-add while-mult-zero-add while-zero-2*)

lemma *while-add-zero-star*: $(x + y \ ; \ 0) \star z = x \star (y \ ; \ 0 + z)$
by (*metis while-mult-zero-add while-zero-2*)

lemma *while-unfold-sum*: $(x + y) \star z = (x \star z) + (x \star (y \ ; \ ((x + y) \star z)))$
apply (*rule antisym*)
apply (*smt add-associative less-eq-def while-absorb-1 while-increasing while-mult-star-exchange while-right-unfold while-sub-associative while-sumstar*)
apply (*metis add-least-upper-bound while-decompose-7 while-mult-increasing while-right-isotone while-sub-dist*)
done

lemma *while-simulate-left*: $x \ ; \ z \leq z \ ; \ y + w \longrightarrow x \star (z \ ; \ v) \leq z \ ; \ (y \star v) + (x \star (w \ ; \ (y \star v)))$
by (*metis add-left-isotone mult-right-isotone order-trans while-one-increasing while-simulate-left-plus*)

lemma *while-simulate-right*: $z \ ; \ x \leq y \ ; \ z + w \longrightarrow z \ ; \ (x \star v) \leq y \star (z \ ; \ v + w \ ; \ (x \star v))$
proof –
have $y \ ; \ z + w \leq y \ ; \ (y \star z) + w$
by (*metis add-left-isotone mult-right-isotone while-increasing*)
thus *?thesis*
by (*smt order-trans while-simulate-right-plus*)
qed

lemma *while-simulate*: $z \ ; \ x \leq y \ ; \ z \longrightarrow z \ ; \ (x \star v) \leq y \star (z \ ; \ v)$
by (*metis add-right-zero mult-left-zero while-simulate-right*)

— Theorem 9.14

lemma *while-while-mult*: $1 \star (x \star y) = (x \star 1) \star y$
proof –
have $(x \star 1) \star y \leq (x \star 1) \ ; \ ((x \star 1) \star y)$
by (*metis order-refl while-increasing while-sup-one-left-unfold*)
also have ... $\leq 1 \star ((x \star 1) \ ; \ y)$
by (*metis mult-left-one order-refl while-mult-upper-bound while-simulate*)
also have ... $\leq 1 \star (x \star y)$
by (*metis while-one-mult-below while-right-isotone*)
finally show *?thesis*
by (*metis antisym while-sub-dist-3 while-while-add*)
qed

lemma *while-simulate-left-1*: $x \ ; \ z \leq z \ ; \ y \longrightarrow x \star (z \ ; \ v) \leq z \ ; \ (y \star v) + (x \star 0)$
by (*metis add-right-zero mult-left-zero while-simulate-left*)

— Theorem 9.46

lemma *while-associative-1*: $1 \leq z \longrightarrow x \star (y \ ; \ z) = (x \star y) \ ; \ z$
proof
assume *1*: $1 \leq z$
have $x \star (y \ ; \ z) \leq x \star ((x \star y) \ ; \ z)$
by (*metis less-eq-def mult-right-dist-add while-plus-one while-right-isotone*)
also have ... $\leq (x \star y) \ ; \ (0 \star z) + (x \star 0)$
by (*metis mult-associative mult-right-sub-dist-add-right while-left-unfold while-simulate-absorb while-zero*)
also have ... $\leq (x \star y) \ ; \ z + (x \star 0) \ ; \ z$ **using** *1*
by (*metis add-least-upper-bound add-left-upper-bound add-right-upper-bound case-split-right while-plus-one while-zero*)
also have ... $= (x \star y) \ ; \ z$
by (*metis add-right-zero mult-right-dist-add while-left-dist-add*)
finally show $x \star (y \ ; \ z) = (x \star y) \ ; \ z$
by (*metis antisym while-sub-associative*)

qed

— Theorem 9.29

lemma *while-associative-while-1*: $x \star (y ; (z \star 1)) = (x \star y) ; (z \star 1)$
  by (*metis while-associative-1 while-increasing*)

— Theorem 9.13

lemma *while-one-while*: $(x \star 1) ; (y \star 1) = x \star (y \star 1)$
  by (*metis mult-left-one while-associative-while-1*)

lemma *while-decompose-5-below*: $(x \star (y \star 1)) \star z \leq (y \star (x \star 1)) \star z$
  by (*smt add-commutative mult-left-dist-add mult-right-one while-increasing while-left-unfold while-mult-star-exchange while-one-while while-plus-one while-sumstar*)

— Theorem 9.26

lemma *while-decompose-5*: $(x \star (y \star 1)) \star z = (y \star (x \star 1)) \star z$
  by (*metis antisym while-decompose-5-below*)

lemma *while-decompose-4*: $(x \star (y \star 1)) \star z = x \star ((y \star (x \star 1)) \star z)$
  by (*metis while-decompose-5 while-decompose-9 while-transitive*)

— Theorem 11.7

lemma *while-simulate-2*: $y ; (x \star 1) \leq x \star (y \star 1) \longleftrightarrow y \star (x \star 1) \leq x \star (y \star 1)$
proof (*rule iffI*)
  assume $y ; (x \star 1) \leq x \star (y \star 1)$
  hence $y ; (x \star 1) \leq (x \star 1) ; (y \star 1)$
    by (*metis while-one-while*)
  hence $y \star ((x \star 1) ; 1) \leq (x \star 1) ; (y \star 1) + (y \star 0)$
    by (*metis while-simulate-left-plus-1*)
  hence $y \star (x \star 1) \leq (x \star (y \star 1)) + (y \star 0)$
    by (*metis mult-right-one while-one-while*)
  also have ... $= x \star (y \star 1)$
    by (*metis add-commutative less-eq-def order-trans while-increasing while-right-isotone zero-least*)
  finally show $y \star (x \star 1) \leq x \star (y \star 1)$
    .
next
  assume $y \star (x \star 1) \leq x \star (y \star 1)$
  thus $y ; (x \star 1) \leq x \star (y \star 1)$
    by (*metis order-trans while-mult-increasing*)
qed

lemma *while-simulate-1*: $y ; x \leq x ; y \longrightarrow y \star (x \star 1) \leq x \star (y \star 1)$
  by (*metis order-trans while-mult-increasing while-right-isotone while-simulate while-simulate-2*)

lemma *while-simulate-3*: $y ; (x \star 1) \leq x \star 1 \longrightarrow y \star (x \star 1) \leq x \star (y \star 1)$
  by (*metis add-idempotent case-split-right while-increasing while-mult-upper-bound while-simulate-2*)

— Theorem 9.28

lemma *while-extra-while*: $(y ; (x \star 1)) \star z = (y ; (y \star (x \star 1))) \star z$
proof −
  have $y ; (y \star (x \star 1)) \leq y ; (x \star 1) ; (y ; (x \star 1) \star 1)$
    by (*smt add-commutative add-left-upper-bound mult-right-one order-trans while-back-loop-prefixpoint while-left-isotone while-mult-star-exchange*)
  hence *1*: $(y ; (y \star (x \star 1))) \star z \leq (y ; (x \star 1)) \star z$
    by (*metis while-simulate-right-plus-1 mult-left-one*)
  have $(y ; (x \star 1)) \star z \leq (y ; (y \star (x \star 1))) \star z$
    by (*metis while-increasing while-left-isotone while-mult-star-exchange*)
  thus *?thesis* **using** *1*
    by (*metis antisym*)
qed

— Theorem 11.6

lemma *while-separate-4*: $y ; x \leq x ; (x \star (1 + y)) \longrightarrow (x + y) \star z = x \star (y \star z)$

proof
assume *1*: $y ; x \leq x ; (x \star (1 + y))$
hence $(1 + y) ; x \leq x ; (x \star (1 + y))$
by (*smt add-associative add-least-upper-bound mult-left-one mult-left-sub-dist-add-left mult-right-dist-add mult-right-one while-left-unfold*)
hence *2*: $(1 + y) ; (x \star 1) \leq x \star (1 + y)$
by (*metis mult-right-one while-simulate-right-plus-1*)
have $y ; x ; (x \star 1) \leq x ; (x \star ((1 + y) ; (x \star 1)))$ **using** *1*
by (*smt less-eq-def mult-associative mult-right-dist-add while-associative-1 while-increasing*)
also have ... $\leq x ; (x \star (1 + y))$ **using** *2*
by (*metis mult-right-isotone order-refl while-mult-transitive*)
also have ... $\leq x ; (x \star 1) ; (y \star 1)$
by (*metis add-least-upper-bound mult-associative mult-right-isotone while-increasing while-one-increasing while-one-while while-right-isotone*)
finally have $y \star (x ; (x \star 1)) \leq x ; (x \star 1) ; (y \star 1) + (y \star 0)$
by (*metis mult-associative mult-right-one while-simulate-left-plus-1*)
hence $(y \star 1) ; (y \star x) \leq x ; (x \star y \star 1) + (y \star 0)$
by (*smt less-eq-def mult-associative mult-right-one order-refl order-trans while-absorb-2 while-left-dist-add while-mult-star-exchange while-one-mult-below while-one-while while-plus-one*)
hence $(y \star 1) ; ((y \star x) \star (y \star z)) \leq x \star ((y \star 1) ; (y \star z) + (y \star 0) ; ((y \star x) \star (y \star z)))$
by (*metis while-simulate-right-plus*)
also have ... $\leq x \star ((y \star z) + (y \star 0))$
by (*metis add-isotone mult-left-zero order-refl while-absorb-2 while-one-mult-below while-right-isotone while-sub-associative*)
also have ... $= x \star y \star z$
by (*metis add-right-zero while-left-dist-add*)
finally show $(x + y) \star z = x \star (y \star z)$
by (*smt add-commutative less-eq-def mult-left-one mult-right-dist-add while-plus-one while-sub-associative while-sumstar*)
qed

lemma *while-separate-5*: $y ; x \leq x ; (x \star (x + y)) \longrightarrow (x + y) \star z = x \star (y \star z)$
by (*smt order-trans while-add-sub-add-one-mult while-separate-4*)

lemma *while-separate-6*: $y ; x \leq x ; (x + y) \longrightarrow (x + y) \star z = x \star (y \star z)$
by (*smt order-trans while-increasing while-mult-star-exchange while-separate-5*)

— Theorem 11.4

lemma *while-separate-1*: $y ; x \leq x ; y \longrightarrow (x + y) \star z = x \star (y \star z)$
by (*metis add-least-upper-bound less-eq-def mult-left-sub-dist-add-right while-separate-6*)

— Theorem 11.2

lemma *while-separate-mult-1*: $y ; x \leq x ; y \longrightarrow (x ; y) \star z \leq x \star (y \star z)$
by (*metis while-mult-sub-add while-separate-1*)

— Theorem 11.5

lemma *separation*: $y ; x \leq x ; (y \star 1) \longrightarrow (x + y) \star z = x \star (y \star z)$
proof
assume $y ; x \leq x ; (y \star 1)$
hence $y \star x \leq x ; (y \star 1) + (y \star 0)$
by (*metis mult-right-one while-simulate-left-plus-1*)
also have ... $\leq x ; (x \star y \star 1) + (y \star 0)$
by (*metis add-left-isotone while-increasing while-mult-star-exchange*)
finally have $(y \star 1) ; (y \star x) \leq x ; (x \star y \star 1) + (y \star 0)$
by (*metis order-refl order-trans while-absorb-2 while-one-mult-below*)
hence $(y \star 1) ; ((y \star x) \star (y \star z)) \leq x \star ((y \star 1) ; (y \star z) + (y \star 0) ; ((y \star x) \star (y \star z)))$
by (*metis while-simulate-right-plus*)
also have ... $\leq x \star ((y \star z) + (y \star 0))$
by (*metis add-isotone mult-left-zero order-refl while-absorb-2 while-one-mult-below while-right-isotone while-sub-associative*)
also have ... $= x \star y \star z$
by (*metis add-right-zero while-left-dist-add*)
finally show $(x + y) \star z = x \star (y \star z)$
by (*smt add-commutative less-eq-def mult-left-one mult-right-dist-add while-plus-one while-sub-associative while-sumstar*)
qed

— Theorem 11.5

lemma *while-separate-left*: $y ; x \leq x ; (y \star 1) \longrightarrow y \star (x \star z) \leq x \star (y \star z)$

by (*metis add-commutative separation while-sub-dist-3*)

— Theorem 11.6

lemma *while-simulate-4*: $y ; x \leq x ; (x \star (1 + y)) \longrightarrow y \star (x \star z) \leq x \star (y \star z)$
by (*metis add-commutative while-separate-4 while-sub-dist-3*)

lemma *while-simulate-5*: $y ; x \leq x ; (x \star (x + y)) \longrightarrow y \star (x \star z) \leq x \star (y \star z)$
by (*smt order-trans while-add-sub-add-one-mult while-simulate-4*)

lemma *while-simulate-6*: $y ; x \leq x ; (x + y) \longrightarrow y \star (x \star z) \leq x \star (y \star z)$
by (*smt order-trans while-increasing while-mult-star-exchange while-simulate-5*)

— Theorem 11.3

lemma *while-simulate-7*: $y ; x \leq x ; y \longrightarrow y \star (x \star z) \leq x \star (y \star z)$
by (*metis add-commutative mult-left-sub-dist-add-left order-trans while-simulate-6*)

lemma *while-while-mult-1*: $x \star (1 \star y) = 1 \star (x \star y)$
by (*metis add-commutative mult-left-one mult-right-one order-refl while-separate-1*)

— Theorem 9.15

lemma *while-while-mult-2*: $x \star (1 \star y) = (x \star 1) \star y$
by (*metis while-while-mult while-while-mult-1*)

— Theorem 11.8

lemma *while-import*: $p \leq p ; p \wedge p \leq 1 \wedge p ; x \leq x ; p \longrightarrow p ; (x \star y) = p ; ((p ; x) \star y)$
proof
  assume *1*: $p \leq p ; p \wedge p \leq 1 \wedge p ; x \leq x ; p$
  hence $p ; (x \star y) \leq (p ; x) \star (p ; y)$
    by (*smt add-commutative less-eq-def mult-associative mult-left-dist-add mult-right-one while-simulate*)
  also have ... $\leq (p ; x) \star y$ **using** *1*
    by (*metis less-eq-def mult-left-one mult-right-dist-add while-right-isotone*)
  finally have *2*: $p ; (x \star y) \leq p ; ((p ; x) \star y)$ **using** *1*
    by (*smt add-commutative less-eq-def mult-associative mult-left-dist-add mult-right-one*)
  have $p ; ((p ; x) \star y) \leq p ; (x \star y)$ **using** *1*
    by (*metis mult-left-isotone mult-left-one mult-right-isotone while-left-isotone*)
  thus $p ; (x \star y) = p ; ((p ; x) \star y)$ **using** *2*
    by (*metis antisym*)
qed

— Theorem 11.8

lemma *while-preserve*: $p \leq p ; p \wedge p \leq 1 \wedge p ; x \leq x ; p \longrightarrow p ; (x \star y) = p ; (x \star (p ; y))$
  apply *rule*
  apply (*rule antisym*)
  apply (*metis mult-associative mult-left-isotone mult-right-isotone order-trans while-simulate*)
  apply (*metis mult-left-isotone mult-left-one mult-right-isotone while-right-isotone*)
  done

lemma *while-plus-below-while*: $(x \star 1) ; x \leq x \star 1$
by (*metis order-refl while-mult-upper-bound while-one-increasing*)

— Theorem 9.40

lemma *while-01*: $(w ; (x \star 1)) \star (y ; z) \leq (x \star w) \star ((x \star y) ; z)$
proof –
  have $(w ; (x \star 1)) \star (y ; z) = y ; z + w ; (((x \star 1) ; w) \star ((x \star 1) ; y ; z))$
    by (*metis mult-associative while-productstar*)
  also have ... $\leq y ; z + w ; ((x \star w) \star ((x \star y) ; z))$
    by (*metis add-right-isotone mult-left-isotone mult-right-isotone while-isotone while-one-mult-below*)
  also have ... $\leq (x \star y) ; z + (x \star w) ; ((x \star w) \star ((x \star y) ; z))$
    by (*metis add-isotone mult-right-sub-dist-add-left while-left-unfold*)
  finally show *?thesis*
    by (*metis while-left-unfold*)
qed

— Theorem 9.37

lemma *while-while-sub-associative*: $x \star (y \star z) \leq ((x \star y) \star z) + (x \star z)$
proof –
  have *1*: $x ; (x \star 1) \leq (x \star 1) ; ((x \star y) \star 1)$
    by (*metis add-least-upper-bound order-trans while-back-loop-prefixpoint while-left-plus-below*)
  have $x \star (y \star z) \leq x \star ((x \star 1) ; (y \star z))$
    by (*metis mult-left-isotone mult-left-one while-increasing while-right-isotone*)
  also have ... $\leq (x \star 1) ; ((x \star y) \star (y \star z)) + (x \star 0)$ **using** *1*
    by (*metis while-simulate-left-plus-1*)
  also have ... $\leq (x \star 1) ; ((x \star y) \star z) + (x \star z)$
    by (*metis add-isotone order-refl while-absorb-2 while-increasing while-right-isotone zero-least*)
  also have ... $= (x \star 1) ; z + (x \star 1) ; (x \star y) ; ((x \star y) \star z) + (x \star z)$
    by (*metis mult-associative mult-left-dist-add while-left-unfold*)
  also have ... $= (x \star y) ; ((x \star y) \star z) + (x \star z)$
    by (*smt add-associative add-commutative less-eq-def mult-left-one mult-right-dist-add order-refl while-absorb-1 while-plus-one while-sub-associative*)
  also have ... $\leq ((x \star y) \star z) + (x \star z)$
    by (*metis add-left-isotone while-left-plus-below*)
  finally show *?thesis*
    .
qed

lemma *while-induct*: $x ; z \leq z \wedge y \leq z \wedge x \star 1 \leq z \longrightarrow x \star y \leq z$
  by (*metis add-commutative add-least-upper-bound add-left-zero less-eq-def while-right-isotone while-simulate-absorb*)

lemma *while-sumstar-4-below*: $(x \star y) \star ((x \star 1) ; z) \leq x \star ((y ; (x \star 1)) \star z)$ **oops**
lemma *while-sumstar-2*: $(x + y) \star z = x \star ((y ; (x \star 1)) \star z)$ **oops**
lemma *while-sumstar-3*: $(x + y) \star z = ((x \star 1) ; y) \star (x \star z)$ **oops**
lemma *while-decompose-6*: $x \star ((y ; (x \star 1)) \star z) = y \star ((x ; (y \star 1)) \star z)$ **oops**
lemma *while-finite-associative*: $x \star 0 = 0 \longrightarrow (x \star y) ; z = x \star (y ; z)$ **oops**
lemma *atomicity-refinement*: $s = s ; q \wedge x = q ; x \wedge q ; b = 0 \wedge r ; b \leq b ; r \wedge r ; l \leq l ; r \wedge x ; l \leq l ; x \wedge b ; l \leq l ; b \wedge q ; l \leq l ; q \wedge r \star q \leq q ; (r \star 1) \wedge q \leq 1 \longrightarrow s ; ((x + b + r + l) \star (q ; z)) \leq s ; ((x ; (b \star q) + r + l) \star z)$ **oops**

lemma *while-separate-right-plus*: $y ; x \leq x ; (x \star (1 + y)) + 1 \longrightarrow y \star (x \star z) \leq x \star (y \star z)$ **oops**
lemma *while-square-1*: $x \star 1 = (x ; x) \star (x + 1)$ **oops**
lemma *while-absorb-below-one*: $y ; x \leq x \longrightarrow y \star x \leq 1 \star x$ **oops**
lemma $y \star (x \star 1) \leq x \star (y \star 1) \longrightarrow (x + y) \star 1 = x \star (y \star 1)$ **oops**
lemma $y ; x \leq (1 + x) ; (y \star 1) \longrightarrow (x + y) \star 1 = x \star (y \star 1)$ **oops**

end

class *bounded-binary-itering* = *bounded-idempotent-left-zero-semiring* + *binary-itering*

begin

— Theorem 9

lemma *while-right-top*: $x \star T = T$
  by (*metis add-left-top while-left-unfold*)

— Theorem 9

lemma *while-left-top*: $T ; (x \star 1) = T$
  by (*metis add-right-top antisym top-greatest while-back-loop-prefixpoint*)

— Theorem 10.10 counterexamples

lemma *while-sum-below-one*: $y ; ((x + y) \star z) \leq (y ; (x \star 1)) \star z$ **nitpick** [*expect=genuine*] **oops**
lemma *while-denest-1*: $w ; (x \star (y ; z)) \leq (w ; (x \star y)) \star z$ **nitpick** [*expect=genuine*] **oops**
lemma *while-denest-2*: $w ; ((x \star (y ; w)) \star z) = w ; (((x \star y) ; w) \star z)$ **nitpick** [*expect=genuine*] **oops**
lemma *while-denest-4*: $(x \star w) \star (x \star (y ; z)) = (x \star w) \star ((x \star y) ; z)$ **nitpick** [*expect=genuine*] **oops**
lemma *while-denest-6*: $(w ; (x \star y)) \star z = z + w ; ((x + y ; w) \star (y ; z))$ **nitpick** [*expect=genuine*] **oops**
lemma *while-decompose-6*: $x \star ((y ; (x \star 1)) \star z) = y \star ((x ; (y \star 1)) \star z)$ **oops**
lemma *while-finite-associative*: $x \star 0 = 0 \longrightarrow (x \star y) ; z = x \star (y ; z)$ **oops**
lemma *while-sumstar-2*: $(x + y) \star z = x \star ((y ; (x \star 1)) \star z)$ **oops**
lemma *while-sumstar-3*: $(x + y) \star z = ((x \star 1) ; y) \star (x \star z)$ **oops**
lemma *while-sumstar-1*: $(x + y) \star z = (x \star y) \star ((x \star 1) ; z)$ **nitpick** [*expect=genuine*] **oops**
lemma *while-mult-zero-zero*: $(x ; (y \star 0)) \star 0 = x ; (y \star 0)$ **nitpick** [*expect=genuine*] **oops**

```
lemma while-denest-3: (x ⋆ w) ⋆ (x ⋆ 0) = (x ⋆ w) ⋆ 0 nitpick [expect=genuine] oops
lemma while-denest-5: w ; ((x ⋆ (y ; w)) ⋆ (x ⋆ (y ; z))) = w ; (((x ⋆ y) ; w) ⋆ ((x ⋆ y) ; z)) nitpick [expect=genuine] oops
lemma while-separate-unfold: (y ; (x ⋆ 1)) ⋆ z = (y ⋆ z) + (y ⋆ (y ; x ; (x ⋆ ((y ; (x ⋆ 1)) ⋆ z)))) nitpick [expect=genuine]
oops
lemma while-02: x ⋆ ((x ⋆ w) ⋆ ((x ⋆ y) ; z)) = (x ⋆ w) ⋆ ((x ⋆ y) ; z) nitpick [expect=genuine] oops

lemma while-denest-0: w ; (x ⋆ (y ; z)) ≤ (w ; (x ⋆ y)) ⋆ (w ; (x ⋆ y) ; z) nitpick [expect=genuine] oops
lemma while-mult-sub-while-while: x ⋆ (y ; z) ≤ (x ⋆ y) ⋆ z nitpick [expect=genuine] oops
lemma while-zero-zero: (x ⋆ 0) ⋆ 0 = x ⋆ 0 nitpick [expect=genuine] oops
lemma while-sumstar-3-below: (x ⋆ y) ⋆ (x ⋆ z) ≤ (x ⋆ y) ⋆ ((x ⋆ 1) ; z) nitpick [expect=genuine] oops

end

class extended-binary-itering = binary-itering +
  assumes while-denest-0: w ; (x ⋆ (y ; z)) ≤ (w ; (x ⋆ y)) ⋆ (w ; (x ⋆ y) ; z)

begin

— Theorem 10.2

lemma while-denest-1: w ; (x ⋆ (y ; z)) ≤ (w ; (x ⋆ y)) ⋆ z
  by (metis order-trans while-denest-0 while-right-plus-below)

lemma while-mult-sub-while-while: x ⋆ (y ; z) ≤ (x ⋆ y) ⋆ z
  by (metis mult-left-one while-denest-1)

lemma while-zero-zero: (x ⋆ 0) ⋆ 0 = x ⋆ 0
  by (smt less-eq-def mult-left-zero while-left-dist-add while-mult-star-exchange while-mult-sub-while-while while-mult-zero-add-2
while-plus-one while-sumstar)

— Theorem 10.11

lemma while-mult-zero-zero: (x ; (y ⋆ 0)) ⋆ 0 = x ; (y ⋆ 0)
  apply (rule antisym)
    apply (metis add-least-upper-bound add-right-zero mult-left-zero mult-right-isotone while-left-dist-add while-slide
while-sub-associative)
  apply (metis mult-left-zero while-denest-1)
  done

— Theorem 10.3

lemma while-denest-2: w ; ((x ⋆ (y ; w)) ⋆ z) = w ; (((x ⋆ y) ; w) ⋆ z)
  apply (rule antisym)
  apply (metis mult-associative while-denest-0 while-simulate-right-plus-1 while-slide)
  apply (metis mult-right-isotone while-left-isotone while-sub-associative)
  done

— Theorem 10.12

lemma while-denest-3: (x ⋆ w) ⋆ (x ⋆ 0) = (x ⋆ w) ⋆ 0
  by (metis while-absorb-2 while-right-isotone while-zero-zero zero-least)

— Theorem 10.15

lemma while-02: x ⋆ ((x ⋆ w) ⋆ ((x ⋆ y) ; z)) = (x ⋆ w) ⋆ ((x ⋆ y) ; z)
proof -
  have x ; ((x ⋆ w) ⋆ ((x ⋆ y) ; z)) = x ; (x ⋆ y) ; z + x ; (x ⋆ w) ; ((x ⋆ w) ⋆ ((x ⋆ y) ; z))
    by (metis mult-associative mult-left-dist-add while-left-unfold)
  also have ... ≤ (x ⋆ w) ⋆ ((x ⋆ y) ; z)
    by (metis add-isotone mult-right-sub-dist-add-right while-left-unfold)
  finally have x ⋆ ((x ⋆ w) ⋆ ((x ⋆ y) ; z)) ≤ ((x ⋆ w) ⋆ ((x ⋆ y) ; z)) + (x ⋆ 0)
    by (metis while-simulate-absorb)
  also have ... = (x ⋆ w) ⋆ ((x ⋆ y) ; z)
    by (metis add-commutative less-eq-def order-trans while-mult-sub-while-while while-right-isotone zero-least)
  finally show ?thesis
    by (metis antisym while-increasing)
qed

lemma while-sumstar-3-below: (x ⋆ y) ⋆ (x ⋆ z) ≤ (x ⋆ y) ⋆ ((x ⋆ 1) ; z)
```

proof –
have $(x \star y) \star (x \star z) = (x \star z) + ((x \star y) \star ((x \star y) ; (x \star z)))$
by (*metis while-right-unfold*)
also have ... $\leq (x \star z) + ((x \star y) \star (x \star (y ; (x \star z))))$
by (*metis add-right-isotone while-right-isotone while-sub-associative*)
also have ... $\leq (x \star z) + ((x \star y) \star (x \star ((x \star y) \star (x \star z))))$
by (*smt add-right-isotone order-trans while-increasing while-mult-upper-bound while-one-increasing while-right-isotone*)
also have ... $\leq (x \star z) + ((x \star y) \star (x \star ((x \star y) \star ((x \star 1) ; z))))$
by (*metis add-right-isotone mult-left-isotone mult-left-one order-trans while-increasing while-right-isotone while-sumstar while-transitive*)
also have ... $= (x \star z) + ((x \star y) \star ((x \star 1) ; z))$
by (*metis while-02 while-transitive*)
also have ... $= (x \star y) \star ((x \star 1) ; z)$
by (*smt add-associative mult-left-one mult-right-dist-add while-02 while-left-dist-add while-plus-one*)
finally show *?thesis*
.
qed

lemma *while-sumstar-4-below*: $(x \star y) \star ((x \star 1) ; z) \leq x \star ((y ; (x \star 1)) \star z)$
proof –
have $(x \star y) \star ((x \star 1) ; z) = (x \star 1) ; z + (x \star y) ; ((x \star y) \star ((x \star 1) ; z))$
by (*metis while-left-unfold*)
also have ... $\leq (x \star z) + (x \star (y ; ((x \star y) \star ((x \star 1) ; z))))$
by (*metis add-isotone while-one-mult-below while-sub-associative*)
also have ... $= (x \star z) + (x \star (y ; (((x \star 1) ; y) \star ((x \star 1) ; z))))$
by (*metis mult-left-one while-denest-2*)
also have ... $= x \star ((y ; (x \star 1)) \star z)$
by (*metis while-left-dist-add while-productstar*)
finally show *?thesis*
.
qed

— Theorem 10.10

lemma *while-sumstar-1*: $(x + y) \star z = (x \star y) \star ((x \star 1) ; z)$
by (*smt eq-iff order-trans while-add-1-below while-sumstar while-sumstar-3-below while-sumstar-4-below*)

— Theorem 10.8

lemma *while-sumstar-2*: $(x + y) \star z = x \star ((y ; (x \star 1)) \star z)$
by (*metis eq-iff while-add-1-below while-sumstar-1 while-sumstar-4-below*)

— Theorem 10.9

lemma *while-sumstar-3*: $(x + y) \star z = ((x \star 1) ; y) \star (x \star z)$
by (*metis eq-iff while-sumstar while-sumstar-1-below while-sumstar-2 while-sumstar-2-below*)

— Theorem 10.6

lemma *while-decompose-6*: $x \star ((y ; (x \star 1)) \star z) = y \star ((x ; (y \star 1)) \star z)$
by (*metis add-commutative while-sumstar-2*)

— Theorem 10.4

lemma *while-denest-4*: $(x \star w) \star (x \star (y ; z)) = (x \star w) \star ((x \star y) ; z)$
proof –
have $(x \star w) \star (x \star (y ; z)) = x \star ((w ; (x \star 1)) \star (y ; z))$
by (*metis while-sumstar while-sumstar-2*)
also have ... $\leq (x \star w) \star ((x \star y) ; z)$
by (*smt antisym while-01 while-02 while-increasing while-right-isotone*)
finally show *?thesis*
by (*metis antisym while-right-isotone while-sub-associative*)
qed

— Theorem 10.13

lemma *while-denest-5*: $w ; ((x \star (y ; w)) \star (x \star (y ; z))) = w ; (((x \star y) ; w) \star ((x \star y) ; z))$
by (*metis while-denest-2 while-denest-4*)

— Theorem 10.5

lemma *while-denest-6*: $(w ; (x \star y)) \star z = z + w ; ((x + y ; w) \star (y ; z))$
  by (*metis while-denest-5 while-productstar while-sumstar*)

— Theorem 10.1

lemma *while-sum-below-one*: $y ; ((x + y) \star z) \leq (y ; (x \star 1)) \star z$
  by (*metis add-right-divisibility mult-left-one while-denest-6*)

— Theorem 10.14

lemma *while-separate-unfold*: $(y ; (x \star 1)) \star z = (y \star z) + (y \star (y ; x ; (x \star ((y ; (x \star 1)) \star z))))$
proof –
  have $y \star (y ; x ; (x \star ((y ; (x \star 1)) \star z))) \leq y \star (y ; ((x + y) \star z))$
    by (*metis mult-associative mult-right-isotone while-sumstar-2 while-left-plus-below while-right-isotone*)
  also have ... $\leq (y ; (x \star 1)) \star z$
    by (*metis add-commutative add-left-upper-bound while-absorb-1 while-mult-star-exchange while-sum-below-one*)
  finally have $(y \star z) + (y \star (y ; x ; (x \star ((y ; (x \star 1)) \star z)))) \leq (y ; (x \star 1)) \star z$
    by (*metis add-least-upper-bound mult-left-sub-dist-add-left mult-right-one while-left-isotone while-left-unfold*)
  thus *?thesis*
    by (*metis antisym while-separate-unfold-below*)
qed

— Theorem 10.7

lemma *while-finite-associative*: $x \star 0 = 0 \longrightarrow (x \star y) ; z = x \star (y ; z)$
  by (*metis while-denest-4 while-zero*)

— Theorem 12

lemma *atomicity-refinement*: $s = s ; q \wedge x = q ; x \wedge q ; b = 0 \wedge r ; b \leq b ; r \wedge r ; l \leq l ; r \wedge x ; l \leq l ; x \wedge b ; l \leq l ; b \wedge q ; l \leq l ; q \wedge r \star q \leq q ; (r \star 1) \wedge q \leq 1 \longrightarrow s ; ((x + b + r + l) \star (q ; z)) \leq s ; ((x ; (b \star q) + r + l) \star z)$
proof
  assume *1*: $s = s ; q \wedge x = q ; x \wedge q ; b = 0 \wedge r ; b \leq b ; r \wedge r ; l \leq l ; r \wedge x ; l \leq l ; x \wedge b ; l \leq l ; b \wedge q ; l \leq l ; q \wedge r \star q \leq q ; (r \star 1) \wedge q \leq 1$
  hence *2*: $(x + b + r) ; l \leq l ; (x + b + r)$
    by (*smt add-commutative add-least-upper-bound mult-left-sub-dist-add-right mult-right-dist-add order-trans*)
  have $q ; ((x ; (b \star r \star 1) ; q) \star z) \leq (x ; (b \star r \star 1) ; q) \star z$ **using** *1*
    by (*smt eq-refl order-trans while-increasing while-mult-upper-bound*)
  also have ... $\leq (x ; (b \star ((r \star 1) ; q))) \star z$
    by (*metis mult-associative mult-right-isotone while-left-isotone while-sub-associative*)
  also have ... $\leq (x ; (b \star r \star q)) \star z$
    by (*metis mult-right-isotone while-left-isotone while-one-mult-below while-right-isotone*)
  also have ... $\leq (x ; (b \star (q ; (r \star 1)))) \star z$ **using** *1*
    by (*metis mult-right-isotone while-left-isotone while-right-isotone*)
  finally have *3*: $q ; ((x ; (b \star r \star 1) ; q) \star z) \leq (x ; (b \star q) ; (r \star 1)) \star z$
    by (*metis mult-associative while-associative-while-1*)
  have $s ; ((x + b + r + l) \star (q ; z)) = s ; (l \star (x + b + r) \star (q ; z))$ **using** *2*
    by (*metis add-commutative while-separate-1*)
  also have ... $= s ; q ; (l \star b \star r \star (q ; x ; (b \star r \star 1)) \star (q ; z))$ **using** *1*
    by (*smt add-associative add-commutative while-sumstar-2 while-separate-1*)
  also have ... $= s ; q ; (l \star b \star r \star (q ; ((x ; (b \star r \star 1) ; q) \star z)))$
    by (*smt mult-associative while-slide*)
  also have ... $\leq s ; q ; (l \star b \star r \star (x ; (b \star q) ; (r \star 1)) \star z)$ **using** *3*
    by (*metis mult-right-isotone while-right-isotone*)
  also have ... $\leq s ; (l \star q ; (b \star r \star (x ; (b \star q) ; (r \star 1)) \star z))$ **using** *1*
    by (*smt mult-associative mult-right-isotone while-simulate*)
  also have ... $= s ; (l \star q ; (r \star (x ; (b \star q) ; (r \star 1)) \star z))$ **using** *1*
    by (*metis while-elimination*)
  also have ... $\leq s ; (l \star r \star (x ; (b \star q) ; (r \star 1)) \star z)$ **using** *1*
    by (*metis add-left-divisibility mult-left-one mult-right-dist-add mult-right-isotone while-right-isotone*)
  also have ... $= s ; (l \star (r + x ; (b \star q)) \star z)$
    by (*metis while-sumstar-2*)
  also have ... $\leq s ; ((x ; (b \star q) + r + l) \star z)$
    by (*metis add-commutative mult-right-isotone while-sub-dist-3*)
  finally show $s ; ((x + b + r + l) \star (q ; z)) \leq s ; ((x ; (b \star q) + r + l) \star z)$
    .
qed

end

class *bounded-extended-binary-itering* = *bounded-binary-itering* + *extended-binary-itering*

begin

lemma *while-unfold-below*: $x = z + y ; x \longrightarrow x \leq y \star z$ **nitpick** [*expect=genuine*] **oops**
lemma *while-unfold-below-1*: $x = y ; x \longrightarrow x \leq y \star 1$ **nitpick** [*expect=genuine*] **oops**
lemma *while-loop-is-greatest-postfixpoint*: *is-greatest-postfixpoint* $(\lambda x . y ; x + z) (y \star z)$ **nitpick** [*expect=genuine*] **oops**
lemma *while-loop-is-greatest-fixpoint*: *is-greatest-fixpoint* $(\lambda x . y ; x + z) (y \star z)$ **nitpick** [*expect=genuine*] **oops**
lemma *while-sub-mult-one*: $x ; (1 \star y) \leq 1 \star x$ **nitpick** [*expect=genuine*] **oops**
lemma *while-sub-while-zero*: $x \star z \leq (x \star y) \star z$ **nitpick** [*expect=genuine*] **oops**
lemma *while-while-sub-associative*: $x \star (y \star z) \leq (x \star y) \star z$ **nitpick** [*expect=genuine*] **oops**
lemma *while-top*: $T \star x = T$ **nitpick** [*expect=genuine*] **oops**
lemma *while-one-top*: $1 \star x = T$ **nitpick** [*expect=genuine*] **oops**
lemma *while-mult-top*: $(x ; T) \star z = z + x ; T$ **nitpick** [*expect=genuine*] **oops**
lemma *tarski*: $x \leq x ; T ; x ; T$ **nitpick** [*expect=genuine*] **oops**
lemma *tarski-mult-top-idempotent*: $x ; T = x ; T ; x ; T$ **nitpick** [*expect=genuine*] **oops**
lemma *tarski-top-omega-below*: $x ; T \leq (x ; T) \star 0$ **nitpick** [*expect=genuine*] **oops**
lemma *tarski-top-omega*: $x ; T = (x ; T) \star 0$ **nitpick** [*expect=genuine*] **oops**
lemma *tarski-below-top-omega*: $x \leq (x ; T) \star 0$ **nitpick** [*expect=genuine*] **oops**
lemma *tarski*: $x = 0 \vee T ; x ; T = T$ **nitpick** [*expect=genuine*] **oops**
lemma $(x + y) \star z = ((x \star 1) ; y) \star ((x \star 1) ; z)$ **nitpick** [*expect=genuine*] **oops**
lemma $1 = (x ; 0) \star 1$ **nitpick** [*expect=genuine*] **oops**
lemma *while-top-2*: $T \star z = T ; z$ **nitpick** [*expect=genuine*] **oops**
lemma *while-mult-top-2*: $(x ; T) \star z = z + x ; T ; z$ **nitpick** [*expect=genuine*] **oops**
lemma *while-one-mult*: $(x \star 1) ; x = x \star x$ **nitpick** [*expect=genuine*] **oops**
lemma $(x \star 1) ; y = x \star y$ **nitpick** [*expect=genuine*] **oops**
lemma *while-associative*: $(x \star y) ; z = x \star (y ; z)$ **nitpick** [*expect=genuine*] **oops**
lemma *while-back-loop-is-fixpoint*: *is-fixpoint* $(\lambda x . x ; y + z) (z ; (y \star 1))$ **nitpick** [*expect=genuine*] **oops**
lemma $1 + x ; 0 = x \star 1$ **nitpick** [*expect=genuine*] **oops**
lemma $x = x ; (x \star 1)$ **nitpick** [*expect=genuine*] **oops**
lemma $x ; (x \star 1) = x \star 1$ **nitpick** [*expect=genuine*] **oops**
lemma $x \star 1 = x \star (1 \star 1)$ **nitpick** [*expect=genuine*] **oops**
lemma $(x + y) \star 1 = (x \star (y \star 1)) \star 1$ **nitpick** [*expect=genuine*] **oops**
lemma $z + y ; x = x \longrightarrow y \star z \leq x$ **nitpick** [*expect=genuine*] **oops**
lemma $y ; x = x \longrightarrow y \star x \leq x$ **nitpick** [*expect=genuine*] **oops**
lemma $z + x ; y = x \longrightarrow z ; (y \star 1) \leq x$ **nitpick** [*expect=genuine*] **oops**
lemma $x ; y = x \longrightarrow x ; (y \star 1) \leq x$ **nitpick** [*expect=genuine*] **oops**
lemma $x ; z = z ; y \longrightarrow x \star z \leq z ; (y \star 1)$ **nitpick** [*expect=genuine*] **oops**
lemma $x ; ((y ; x) \star y) \leq x ; ((y ; x) \star 1) ; y$ **nitpick** [*expect=genuine*] **oops**

end

end

12 BinaryIteringStrict

theory *BinaryIteringStrict*

imports *BinaryItering Itering*

begin

class *strict-itering* = *itering* + *while* +
  assumes *while-def*: $x \star y = x^{\circ}$; y

begin

— Theorem 8.1

subclass *extended-binary-itering*
  apply *unfold-locales*
  apply (*metis add-commutative circ-loop-fixpoint circ-slide mult-associative while-def*)
  apply (*metis circ-add mult-associative while-def*)
  apply (*metis mult-left-dist-add while-def*)
  apply (*metis mult-associative order-refl while-def*)
  apply (*metis circ-simulate-left-plus mult-associative mult-left-isotone mult-right-dist-add mult-right-one while-def*)
  apply (*metis circ-simulate-right-plus mult-associative mult-left-isotone mult-right-dist-add while-def*)
  apply (*metis add-right-divisibility circ-loop-fixpoint mult-associative while-def*)
  done

— Theorem 13.1

lemma *while-associative*: $(x \star y)$; $z = x \star (y$; $z)$
  by (*metis mult-associative while-def*)

— Theorem 13.3

lemma *while-one-mult*: $(x \star 1)$; $x = x \star x$
  by (*metis mult-right-one while-def*)

lemma *while-back-loop-is-fixpoint*: *is-fixpoint* $(\lambda x\ .\ x$; $y + z)$ $(z$; $(y \star 1))$
  by (*metis circ-back-loop-is-fixpoint mult-right-one while-def*)

— Theorem 13.4

lemma $(x + y) \star z = ((x \star 1)$; $y) \star ((x \star 1)$; $z)$
  by (*metis mult-right-one while-def while-sumstar*)

— Theorem 13.2

lemma $(x \star 1)$; $y = x \star y$
  by (*metis mult-left-one while-associative*)

lemma $y \star (x \star 1) \leq x \star (y \star 1) \longrightarrow (x + y) \star 1 = x \star (y \star 1)$ **oops**
lemma y ; $x \leq (1 + x)$; $(y \star 1) \longrightarrow (x + y) \star 1 = x \star (y \star 1)$ **oops**
lemma *while-square-1*: $x \star 1 = (x$; $x) \star (x + 1)$ **oops**
lemma *while-absorb-below-one*: y ; $x \leq x \longrightarrow y \star x \leq 1 \star x$ **oops**

end

class *bounded-strict-itering* = *bounded-itering* + *strict-itering*

begin

subclass *bounded-extended-binary-itering* **..**

— Theorem 13.6

lemma *while-top-2*: $T \star z = T$; z
  by (*metis circ-top while-def*)

— Theorem 13.5

lemma *while-mult-top-2*: $(x \ ; \ T) \star z = z + x \ ; \ T \ ; \ z$
  by (*metis circ-left-top mult-associative while-def while-left-unfold*)

— Theorem 13 counterexamples

lemma *while-one-top*: $1 \star x = T$ **nitpick** [*expect=genuine*] **oops**
lemma *while-top*: $T \star x = T$ **nitpick** [*expect=genuine*] **oops**
lemma *while-sub-mult-one*: $x \ ; \ (1 \star y) \leq 1 \star x$ **nitpick** [*expect=genuine*] **oops**
lemma *while-unfold-below-1*: $x = y \ ; \ x \longrightarrow x \leq y \star 1$ **nitpick** [*expect=genuine*] **oops**
lemma *while-unfold-below*: $x = z + y \ ; \ x \longrightarrow x \leq y \star z$ **nitpick** [*expect=genuine*] **oops**
lemma *while-unfold-below*: $x \leq z + y \ ; \ x \longrightarrow x \leq y \star z$ **nitpick** [*expect=genuine*] **oops**
lemma *while-mult-top*: $(x \ ; \ T) \star z = z + x \ ; \ T$ **nitpick** [*expect=genuine*] **oops**
lemma *tarski-mult-top-idempotent*: $x \ ; \ T = x \ ; \ T \ ; \ x \ ; \ T$ **nitpick** [*expect=genuine*] **oops**

lemma *while-loop-is-greatest-postfixpoint*: *is-greatest-postfixpoint* $(\lambda x \ . \ y \ ; \ x + z) \ (y \star z)$ **nitpick** [*expect=genuine*] **oops**
lemma *while-loop-is-greatest-fixpoint*: *is-greatest-fixpoint* $(\lambda x \ . \ y \ ; \ x + z) \ (y \star z)$ **nitpick** [*expect=genuine*] **oops**
lemma *while-sub-while-zero*: $x \star z \leq (x \star y) \star z$ **nitpick** [*expect=genuine*] **oops**
lemma *while-while-sub-associative*: $x \star (y \star z) \leq (x \star y) \star z$ **nitpick** [*expect=genuine*] **oops**
lemma *tarski*: $x \leq x \ ; \ T \ ; \ x \ ; \ T$ **nitpick** [*expect=genuine*] **oops**
lemma *tarski-top-omega-below*: $x \ ; \ T \leq (x \ ; \ T) \star 0$ **nitpick** [*expect=genuine*] **oops**
lemma *tarski-top-omega*: $x \ ; \ T = (x \ ; \ T) \star 0$ **nitpick** [*expect=genuine*] **oops**
lemma *tarski-below-top-omega*: $x \leq (x \ ; \ T) \star 0$ **nitpick** [*expect=genuine*] **oops**
lemma *tarski*: $x = 0 \vee T \ ; \ x \ ; \ T = T$ **nitpick** [*expect=genuine*] **oops**
lemma $1 = (x \ ; \ 0) \star 1$ **nitpick** [*expect=genuine*] **oops**
lemma $1 + x \ ; \ 0 = x \star 1$ **nitpick** [*expect=genuine*] **oops**
lemma $x = x \ ; \ (x \star 1)$ **nitpick** [*expect=genuine*] **oops**
lemma $x \ ; \ (x \star 1) = x \star 1$ **nitpick** [*expect=genuine*] **oops**
lemma $x \star 1 = x \star (1 \star 1)$ **nitpick** [*expect=genuine*] **oops**
lemma $(x + y) \star 1 = (x \star (y \star 1)) \star 1$ **nitpick** [*expect=genuine*] **oops**
lemma $z + y \ ; \ x = x \longrightarrow y \star z \leq x$ **nitpick** [*expect=genuine*] **oops**
lemma $y \ ; \ x = x \longrightarrow y \star x \leq x$ **nitpick** [*expect=genuine*] **oops**
lemma $z + x \ ; \ y = x \longrightarrow z \ ; \ (y \star 1) \leq x$ **nitpick** [*expect=genuine*] **oops**
lemma $x \ ; \ y = x \longrightarrow x \ ; \ (y \star 1) \leq x$ **nitpick** [*expect=genuine*] **oops**
lemma $x \ ; \ z = z \ ; \ y \longrightarrow x \star z \leq z \ ; \ (y \star 1)$ **nitpick** [*expect=genuine*] **oops**

end

class *binary-itering-unary* = *extended-binary-itering* + *circ* +
  assumes *circ-def*: $x^\circ = x \star 1$

begin

— Theorem 50.7

subclass *left-conway-semiring*
  apply *unfold-locales*
  apply (*metis circ-def while-left-unfold*)
  apply (*metis circ-def mult-right-one while-one-mult-below while-slide*)
  apply (*metis circ-def while-one-while while-sumstar-2*)
  done

end

class *strict-binary-itering* = *binary-itering* + *circ* +
  assumes *while-associative*: $(x \star y) \ ; \ z = x \star (y \ ; \ z)$
  assumes *circ-def*: $x^\circ = x \star 1$

begin

— Theorem 2.8

subclass *itering*
  apply *unfold-locales*
  apply (*metis circ-def mult-left-one-1 while-associative while-sumstar*)
  apply (*metis circ-def mult-right-one while-associative while-productstar while-slide*)
  apply (*metis circ-def mult-right-one while-associative mult-left-one-1 while-slide while-simulate-right-plus*)
  apply (*metis circ-def mult-right-one while-associative mult-left-one-1 while-simulate-left-plus mult-right-dist-add*)
  done

— Theorem 8.5

subclass *extended-binary-itering*
  apply *unfold-locales*
  apply (*metis mult-associative while-associative while-increasing*)
  done

end

end

13 BinaryIteringNonstrict

theory *BinaryIteringNonstrict*

imports *BinaryItering OmegaAlgebra*

begin

class *nonstrict-itering* = *bounded-left-zero-omega-algebra* + *while* +
  assumes *while-def*: $x \star y = x^\omega + x^\star \; ; y$

begin

— Theorem 8.2

subclass *bounded-binary-itering*
proof (*unfold-locales*)
  fix $x\ y\ z$
  show $(x \; ; y) \star z = z + x \; ; ((y \; ; x) \star (y \; ; z))$
    by (*metis add-commutative mult-associative mult-left-dist-add omega-loop-fixpoint omega-slide star.circ-slide while-def*)
next
  fix $x\ y\ z$
  show $(x + y) \star z = (x \star y) \star (x \star z)$
  proof −
    have *1*: $(x + y) \star z = (x^\star \; ; y)^\omega + (x^\star \; ; y)^\star \; ; (x^\omega + x^\star \; ; z)$
      by (*smt add-associative mult-associative mult-left-dist-add omega-decompose star.circ-add while-def*)
    hence *2*: $(x + y) \star z \leq (x \star y) \star (x \star z)$
      by (*smt add-isotone add-right-upper-bound less-eq-def mult-left-isotone omega-sub-dist star.circ-sub-dist while-def*)
    let *?rhs* = $x^\star \; ; y \; ; ((x^\omega + x^\star \; ; y)^\omega + (x^\omega + x^\star \; ; y)^\star \; ; (x^\omega + x^\star \; ; z)) + (x^\omega + x^\star \; ; z)$
    have $x^\omega \; ; (x^\omega + x^\star \; ; y)^\omega \leq x^\omega$
      by (*metis omega-sub-vector*)
    hence $x^\omega \; ; (x^\omega + x^\star \; ; y)^\omega + x^\star \; ; y \; ; (x^\omega + x^\star \; ; y)^\omega \leq$ *?rhs*
      by (*smt add-commutative add-isotone add-left-upper-bound mult-left-dist-add order-trans*)
    hence *3*: $(x^\omega + x^\star \; ; y)^\omega \leq$ *?rhs*
      by (*metis mult-right-dist-add omega-unfold*)
    have $x^\omega \; ; (x^\omega + x^\star \; ; y)^\star \; ; (x^\omega + x^\star \; ; z) \leq x^\omega$
      by (*metis mult-associative omega-sub-vector*)
    hence $x^\omega \; ; (x^\omega + x^\star \; ; y)^\star \; ; (x^\omega + x^\star \; ; z) + x^\star \; ; y \; ; (x^\omega + x^\star \; ; y)^\star \; ; (x^\omega + x^\star \; ; z) \leq$ *?rhs*
      by (*smt add-commutative add-isotone add-right-upper-bound mult-associative mult-left-dist-add order-trans*)
    hence $(x^\omega + x^\star \; ; y)^\star \; ; (x^\omega + x^\star \; ; z) \leq$ *?rhs*
      by (*smt add-associative add-right-upper-bound less-eq-def mult-associative mult-right-dist-add star.circ-loop-fixpoint*)
    hence $(x^\omega + x^\star \; ; y)^\omega + (x^\omega + x^\star \; ; y)^\star \; ; (x^\omega + x^\star \; ; z) \leq$ *?rhs* **using** *3*
      by (*metis add-least-upper-bound*)
    hence $(x^\omega + x^\star \; ; y)^\omega + (x^\omega + x^\star \; ; y)^\star \; ; (x^\omega + x^\star \; ; z) \leq (x^\star \; ; y)^\omega + (x^\star \; ; y)^\star \; ; (x^\omega + x^\star \; ; z)$
      by (*metis add-commutative omega-induct*)
    thus *?thesis* **using** *1 2*
      by (*smt antisym while-def*)
  qed
next
  fix $x\ y\ z$
  show $x \star (y + z) = (x \star y) + (x \star z)$
    by (*smt add-associative add-commutative add-left-upper-bound less-eq-def mult-left-dist-add while-def*)
next
  fix $x\ y\ z$
  show $(x \star y) \; ; z \leq x \star (y \; ; z)$
    by (*metis mult-associative mult-right-dist-add omega-loop-fixpoint omega-loop-greatest-fixpoint while-def*)
next
  fix $v\ w\ x\ y\ z$
  show $x \; ; z \leq z \; ; (y \star 1) + w \longrightarrow x \star (z \; ; v) \leq z \; ; (y \star v) + (x \star (w \; ; (y \star v)))$
  proof
    assume $x \; ; z \leq z \; ; (y \star 1) + w$
    hence *1*: $x \; ; z \leq z \; ; y^\omega + z \; ; y^\star + w$
      by (*metis mult-left-dist-add mult-right-one while-def*)
    let *?rhs* = $z \; ; (y^\omega + y^\star \; ; v) + x^\omega + x^\star \; ; w \; ; (y^\omega + y^\star \; ; v)$
    have *2*: $z \; ; v \leq$ *?rhs*
      by (*metis add-least-upper-bound add-left-upper-bound mult-left-dist-add omega-loop-fixpoint*)
    have $x \; ; z \; ; (y^\omega + y^\star \; ; v) \leq$ *?rhs*
    proof −
      have $x \; ; z \; ; (y^\omega + y^\star \; ; v) \leq (z \; ; y^\omega + z \; ; y^\star + w) \; ; (y^\omega + y^\star \; ; v)$ **using** *1*

by (*metis mult-left-isotone*)
also have ... $= z ; (y^\omega ; (y^\omega + y^\star ; v) + y^\star ; (y^\omega + y^\star ; v)) + w ; (y^\omega + y^\star ; v)$
by (*smt mult-associative mult-left-dist-add mult-right-dist-add*)
also have ... $= z ; (y^\omega ; (y^\omega + y^\star ; v) + y^\omega + y^\star ; v) + w ; (y^\omega + y^\star ; v)$
by (*smt add-associative mult-associative mult-left-dist-add star.circ-transitive-equal star-mult-omega*)
also have ... $\leq z ; (y^\omega + y^\star ; v) + x^\star ; w ; (y^\omega + y^\star ; v)$
by (*smt add-commutative add-isotone add-left-top mult-left-dist-add mult-left-one mult-right-dist-add mult-right-sub-dist-add-left omega-vector order-refl star.circ-plus-one*)
finally show *?thesis*
by (*smt add-associative add-commutative less-eq-def*)
qed
hence $x ; ?rhs \leq ?rhs$
by (*smt add-associative add-commutative add-left-upper-bound less-eq-def mult-associative mult-left-dist-add mult-right-dist-add omega-unfold star.circ-increasing star.circ-transitive-equal*)
hence $z ; v + x ; ?rhs \leq ?rhs$ **using** *2*
by (*metis add-least-upper-bound*)
hence $x^\star ; z ; v \leq ?rhs$
by (*metis mult-associative star-left-induct*)
hence $x^\omega + x^\star ; z ; v \leq ?rhs$
by (*metis add-least-upper-bound add-left-upper-bound*)
thus $x \star (z ; v) \leq z ; (y \star v) + (x \star (w ; (y \star v)))$
by (*smt add-associative mult-associative mult-left-dist-add while-def*)
qed
next
fix *v w x y z*
show $z ; x \leq y ; (y \star z) + w \longrightarrow z ; (x \star v) \leq y \star (z ; v + w ; (x \star v))$
proof
assume $z ; x \leq y ; (y \star z) + w$
hence $z ; x \leq y ; (y^\omega + y^\star ; z) + w$
by (*metis while-def*)
hence *1*: $z ; x \leq y^\omega + y ; y^\star ; z + w$
by (*metis mult-associative mult-left-dist-add omega-unfold*)
let $?rhs = y^\omega + y^\star ; z ; v + y^\star ; w ; (x^\omega + x^\star ; v)$
have *2*: $z ; x^\omega \leq ?rhs$
proof −
have $z ; x^\omega \leq y ; y^\star ; z ; x^\omega + y^\omega ; x^\omega + w ; x^\omega$ **using** *1*
by (*smt add-commutative less-eq-def mult-associative mult-right-dist-add omega-unfold*)
also have ... $\leq y ; y^\star ; z ; x^\omega + y^\omega + w ; x^\omega$
by (*metis add-left-isotone add-right-isotone omega-sub-vector*)
also have ... $= y ; y^\star ; (z ; x^\omega) + (y^\omega + w ; x^\omega)$
by (*metis add-associative mult-associative*)
finally have $z ; x^\omega \leq (y ; y^\star)^\omega + (y ; y^\star)^\star ; (y^\omega + w ; x^\omega)$
by (*metis add-commutative omega-induct*)
also have ... $= y^\omega + y^\star ; w ; x^\omega$
by (*metis left-plus-omega less-eq-def mult-associative mult-left-dist-add mult-left-sub-dist-add-left star.left-plus-circ star-mult-omega*)
also have ... $\leq ?rhs$
by (*metis add-isotone add-left-upper-bound mult-left-sub-dist-add-left*)
finally show *?thesis*
by *metis*
qed
let $?rhs2 = y^\omega + y^\star ; z + y^\star ; w ; (x^\omega + x^\star)$
have $?rhs2 ; x \leq ?rhs2$
proof −
have *3*: $y^\omega ; x \leq ?rhs2$
by (*metis add-associative less-eq-def omega-sub-vector*)
have $y^\star ; z ; x \leq y^\star ; (y^\omega + y ; y^\star ; z + w)$ **using** *1*
by (*metis mult-associative mult-right-isotone*)
also have ... $= y^\omega + y^\star ; y ; y^\star ; z + y^\star ; w$
by (*metis mult-associative mult-left-dist-add star-mult-omega*)
also have ... $= y^\omega + y ; y^\star ; z + y^\star ; w$
by (*metis mult-associative star.circ-transitive-equal star-simulation-right-equal*)
also have ... $\leq y^\omega + y^\star ; z + y^\star ; w$
by (*metis add-left-isotone add-right-isotone mult-left-isotone star.left-plus-below-circ*)
also have ... $\leq y^\omega + y^\star ; z + y^\star ; w ; x^\star$
by (*metis add-right-isotone add-right-upper-bound star.circ-back-loop-fixpoint*)
finally have *4*: $y^\star ; z ; x \leq ?rhs2$
by (*smt add-associative add-commutative less-eq-def mult-left-dist-add*)
have $(x^\omega + x^\star) ; x \leq x^\omega + x^\star$

by (*metis add-isotone mult-right-dist-add omega-sub-vector star.circ-plus-same star.left-plus-below-circ*)
hence $y^\star \; ; \; w \; ; \; (x^\omega + x^\star) \; ; \; x \leq$ *?rhs2*
by (*smt add-associative add-commutative less-eq-def mult-associative mult-left-dist-add*)
thus *?thesis* **using** *3 4*
by (*smt add-associative less-eq-def mult-right-dist-add*)
qed
hence $z +$ *?rhs2* $; \; x \leq$ *?rhs2*
by (*smt add-commutative add-least-upper-bound add-right-divisibility while-def omega-loop-fixpoint*)
hence *5*: $z \; ; \; x^\star \leq$ *?rhs2*
by (*metis star-right-induct*)
have $z \; ; \; x^\star \; ; \; v \leq$ *?rhs*
proof −
have $z \; ; \; x^\star \; ; \; v \leq$ *?rhs2* $; \; v$ **using** *5*
by (*metis mult-left-isotone*)
also have ... $= y^\omega \; ; \; v + y^\star \; ; \; z \; ; \; v + y^\star \; ; \; w \; ; \; (x^\omega \; ; \; v + x^\star \; ; \; v)$
by (*metis mult-associative mult-right-dist-add*)
also have ... $\leq y^\omega + y^\star \; ; \; z \; ; \; v + y^\star \; ; \; w \; ; \; (x^\omega \; ; \; v + x^\star \; ; \; v)$
by (*metis add-left-isotone omega-sub-vector*)
also have ... $\leq$ *?rhs*
by (*metis add-left-isotone add-right-isotone mult-right-isotone omega-sub-vector*)
finally show *?thesis*
by *metis*
qed
hence $z \; ; \; (x^\omega + x^\star \; ; \; v) \leq$ *?rhs* **using** *2*
by (*smt add-associative less-eq-def mult-associative mult-left-dist-add*)
thus $z \; ; \; (x \star v) \leq y \star (z \; ; \; v + w \; ; \; (x \star v))$
by (*metis add-associative mult-associative mult-left-dist-add while-def*)
qed
qed

— Theorem 13.8

lemma *while-top*: $T \star x = T$
by (*metis add-left-top star.circ-top star-omega-top while-def*)

— Theorem 13.7

lemma *while-one-top*: $1 \star x = T$
by (*metis add-left-top omega-one while-def*)

lemma *while-finite-associative*: $x^\omega = 0 \longrightarrow (x \star y) \; ; \; z = x \star (y \; ; \; z)$
by (*metis add-left-zero mult-associative while-def*)

lemma *star-below-while*: $x^\star \; ; \; y \leq x \star y$
by (*metis add-right-upper-bound while-def*)

— Theorem 13.9

lemma *while-sub-mult-one*: $x \; ; \; (1 \star y) \leq 1 \star x$
by (*metis top-greatest while-one-top*)

lemma *while-while-one*: $y \star (x \star 1) = y^\omega + y^\star \; ; \; x^\omega + y^\star \; ; \; x^\star$
by (*metis add-associative mult-left-dist-add mult-right-one while-def*)

lemma *while-simulate-4-plus*: $y \; ; \; x \leq x \; ; \; (x \star (1 + y)) \longrightarrow y \; ; \; x \; ; \; x^\star \leq x \; ; \; (x \star (1 + y))$
proof
have *1*: $x \; ; \; (x \star (1 + y)) = x^\omega + x \; ; \; x^\star + x \; ; \; x^\star \; ; \; y$
by (*metis add-associative mult-associative mult-left-dist-add mult-right-one omega-unfold while-def*)
assume $y \; ; \; x \leq x \; ; \; (x \star (1 + y))$
hence $y \; ; \; x \; ; \; x^\star \leq (x^\omega + x \; ; \; x^\star + x \; ; \; x^\star \; ; \; y) \; ; \; x^\star$ **using** *1*
by (*metis mult-left-isotone*)
also have ... $= x^\omega \; ; \; x^\star + x \; ; \; x^\star \; ; \; x^\star + x \; ; \; x^\star \; ; \; y \; ; \; x^\star$
by (*metis mult-right-dist-add*)
also have ... $= x \; ; \; x^\star \; ; \; (y \; ; \; x \; ; \; x^\star) + x^\omega + x \; ; \; x^\star + x \; ; \; x^\star \; ; \; y$
by (*smt add-associative add-commutative mult-associative omega-mult-star-2 star.circ-back-loop-fixpoint star.circ-plus-same star.circ-transitive-equal*)
finally have $y \; ; \; x \; ; \; x^\star \leq x \; ; \; x^\star \; ; \; (y \; ; \; x \; ; \; x^\star) + (x^\omega + x \; ; \; x^\star + x \; ; \; x^\star \; ; \; y)$
by (*metis add-associative*)
hence $y \; ; \; x \; ; \; x^\star \leq (x \; ; \; x^\star)^\omega + (x \; ; \; x^\star)^\star \; ; \; (x^\omega + x \; ; \; x^\star + x \; ; \; x^\star \; ; \; y)$

  by (*metis add-commutative omega-induct*)
 also have ... $= x^\omega + x^\star ; (x^\omega + x ; x^\star + x ; x^\star ; y)$
  by (*metis left-plus-omega star.left-plus-circ*)
 finally show $y ; x ; x^\star \leq x ; (x \star (1 + y))$ **using** *1*
  by (*metis while-def while-mult-star-exchange while-transitive*)
qed

lemma *while-simulate-4-omega*: $y ; x \leq x ; (x \star (1 + y)) \longrightarrow y ; x^\omega \leq x^\omega$
proof
 have *1*: $x ; (x \star (1 + y)) = x^\omega + x ; x^\star + x ; x^\star ; y$
  by (*metis add-associative mult-associative mult-left-dist-add mult-right-one omega-unfold while-def*)
 assume $y ; x \leq x ; (x \star (1 + y))$
 hence $y ; x^\omega \leq (x^\omega + x ; x^\star + x ; x^\star ; y) ; x^\omega$ **using** *1*
  by (*smt less-eq-def mult-associative mult-right-dist-add omega-unfold*)
 also have ... $= x^\omega ; x^\omega + x ; x^\star ; x^\omega + x ; x^\star ; y ; x^\omega$
  by (*metis mult-right-dist-add*)
 also have ... $= x ; x^\star ; (y ; x^\omega) + x^\omega$
  by (*metis add-commutative less-eq-def mult-associative omega-sub-vector omega-unfold star-mult-omega*)
 finally have $y ; x^\omega \leq x ; x^\star ; (y ; x^\omega) + x^\omega$
  by *metis*
 hence $y ; x^\omega \leq (x ; x^\star)^\omega + (x ; x^\star)^\star ; x^\omega$
  by (*metis add-commutative omega-induct*)
 thus $y ; x^\omega \leq x^\omega$
  by (*metis add-idempotent left-plus-omega star-mult-omega*)
qed

— Theorem 13.11

lemma *while-unfold-below*: $x = z + y ; x \longrightarrow x \leq y \star z$
 by (*metis omega-induct-equal while-def*)

— Theorem 13.12

lemma $x \leq z + y ; x \longrightarrow x \leq y \star z$
 by (*metis omega-induct while-def*)

— Theorem 13.10

lemma *while-unfold-below-1*: $x = y ; x \longrightarrow x \leq y \star 1$
 by (*metis add-right-upper-bound omega-induct while-def*)

lemma *while-square-1*: $x \star 1 = (x ; x) \star (x + 1)$
 by (*metis mult-right-one omega-square star-square-2 while-def*)

lemma *while-absorb-below-one*: $y ; x \leq x \longrightarrow y \star x \leq 1 \star x$
 by (*metis top-greatest while-one-top*)

lemma *while-loop-is-greatest-postfixpoint*: *is-greatest-postfixpoint* $(\lambda x . y ; x + z) (y \star z)$
proof −
 have $(y \star z) \leq (\lambda x . y ; x + z) (y \star z)$
  by (*metis is-fixpoint-def order-refl while-loop-is-fixpoint*)
 thus *?thesis*
  by (*smt add-commutative is-greatest-postfixpoint-def omega-induct while-def*)
qed

lemma *while-loop-is-greatest-fixpoint*: *is-greatest-fixpoint* $(\lambda x . y ; x + z) (y \star z)$
 by (*metis omega-loop-is-greatest-fixpoint while-def*)

lemma *while-sumstar-4-below*: $(x \star y) \star ((x \star 1) ; z) \leq x \star ((y ; (x \star 1)) \star z)$ **nitpick** [*expect=genuine,card=6*] **oops**
lemma *while-sumstar-2*: $(x + y) \star z = x \star ((y ; (x \star 1)) \star z)$ **nitpick** [*expect=genuine,card=6*] **oops**
lemma *while-sumstar-3*: $(x + y) \star z = ((x \star 1) ; y) \star (x \star z)$ **oops**
lemma *while-decompose-6*: $x \star ((y ; (x \star 1)) \star z) = y \star ((x ; (y \star 1)) \star z)$ **nitpick** [*expect=genuine,card=6*] **oops**
lemma *while-finite-associative*: $x \star 0 = 0 \longrightarrow (x \star y) ; z = x \star (y ; z)$ **oops**
lemma *atomicity-refinement*: $s = s ; q \wedge x = q ; x \wedge q ; b = 0 \wedge r ; b \leq b ; r \wedge r ; l \leq l ; r \wedge x ; l \leq l ; x \wedge b ; l \leq l ; b \wedge q ; l \leq l ; q \wedge r \star q \leq q ; (r \star 1) \wedge q \leq 1 \longrightarrow s ; ((x + b + r + l) \star (q ; z)) \leq s ; ((x ; (b \star q) + r + l) \star z)$ **oops**

lemma *while-separate-right-plus*: $y ; x \leq x ; (x \star (1 + y)) + 1 \longrightarrow y \star (x \star z) \leq x \star (y \star z)$ **oops**
lemma $y \star (x \star 1) \leq x \star (y \star 1) \longrightarrow (x + y) \star 1 = x \star (y \star 1)$ **oops**
lemma $y ; x \leq (1 + x) ; (y \star 1) \longrightarrow (x + y) \star 1 = x \star (y \star 1)$ **oops**

lemma *while-mult-sub-while-while*: $x \star (y ; z) \leq (x \star y) \star z$ **oops**
lemma *while-zero-zero*: $(x \star 0) \star 0 = x \star 0$ **oops**
lemma *while-denest-3*: $(x \star w) \star (x \star 0) = (x \star w) \star 0$ **oops**
lemma *while-02*: $x \star ((x \star w) \star ((x \star y) ; z)) = (x \star w) \star ((x \star y) ; z)$ **oops**
lemma *while-sumstar-3-below*: $(x \star y) \star (x \star z) \leq (x \star y) \star ((x \star 1) ; z)$ **oops**
lemma *while-sumstar-1*: $(x + y) \star z = (x \star y) \star ((x \star 1) ; z)$ **oops**
lemma *while-denest-4*: $(x \star w) \star (x \star (y ; z)) = (x \star w) \star ((x \star y) ; z)$ **oops**

end

class *nonstrict-itering-zero* = *nonstrict-itering* +
  assumes *mult-right-zero*: $x ; 0 = 0$

begin

lemma *while-finite-associative-2*: $x \star 0 = 0 \longrightarrow (x \star y) ; z = x \star (y ; z)$
  by (*metis add-left-zero add-right-zero mult-associative mult-right-zero while-def*)

— Theorem 13 counterexamples

lemma *while-mult-top*: $(x ; T) \star z = z + x ; T$ **nitpick** [*expect=genuine*] **oops**
lemma *tarski-mult-top-idempotent*: $x ; T = x ; T ; x ; T$ **nitpick** [*expect=genuine*] **oops**

lemma *while-denest-0*: $w ; (x \star (y ; z)) \leq (w ; (x \star y)) \star (w ; (x \star y) ; z)$ **nitpick** [*expect=genuine*] **oops**
lemma *while-denest-1*: $w ; (x \star (y ; z)) \leq (w ; (x \star y)) \star z$ **nitpick** [*expect=genuine*] **oops**
lemma *while-mult-zero-zero*: $(x ; (y \star 0)) \star 0 = x ; (y \star 0)$ **nitpick** [*expect=genuine*] **oops**
lemma *while-denest-2*: $w ; ((x \star (y ; w)) \star z) = w ; (((x \star y) ; w) \star z)$ **nitpick** [*expect=genuine*] **oops**
lemma *while-denest-5*: $w ; ((x \star (y ; w)) \star (x \star (y ; z))) = w ; (((x \star y) ; w) \star ((x \star y) ; z))$ **nitpick** [*expect=genuine*] **oops**
lemma *while-denest-6*: $(w ; (x \star y)) \star z = z + w ; ((x + y ; w) \star (y ; z))$ **nitpick** [*expect=genuine*] **oops**
lemma *while-sum-below-one*: $y ; ((x + y) \star z) \leq (y ; (x \star 1)) \star z$ **nitpick** [*expect=genuine*] **oops**
lemma *while-separate-unfold*: $(y ; (x \star 1)) \star z = (y \star z) + (y \star (y ; x ; (x \star ((y ; (x \star 1)) \star z))))$ **nitpick** [*expect=genuine*]
oops

lemma *while-sub-while-zero*: $x \star z \leq (x \star y) \star z$ **nitpick** [*expect=genuine*] **oops**
lemma *while-while-sub-associative*: $x \star (y \star z) \leq (x \star y) \star z$ **nitpick** [*expect=genuine*] **oops**
lemma *tarski*: $x \leq x ; T ; x ; T$ **nitpick** [*expect=genuine*] **oops**
lemma *tarski-top-omega-below*: $x ; T \leq (x ; T)^\omega$ **nitpick** [*expect=genuine*] **oops**
lemma *tarski-top-omega*: $x ; T = (x ; T)^\omega$ **nitpick** [*expect=genuine*] **oops**
lemma *tarski-below-top-omega*: $x \leq (x ; T)^\omega$ **nitpick** [*expect=genuine*] **oops**
lemma *tarski-mult-omega-omega*: $(x ; y^\omega)^\omega = x ; y^\omega$ **nitpick** [*expect=genuine*] **oops**
lemma *tarski-mult-omega-omega*: $(\forall z . z^{\omega\omega} = z^\omega) \longrightarrow (x ; y^\omega)^\omega = x ; y^\omega$ **nitpick** [*expect=genuine*] **oops**
lemma *tarski*: $x = 0 \vee T ; x ; T = T$ **nitpick** [*expect=genuine*] **oops**

end

class *nonstrict-itering-tarski* = *nonstrict-itering* +
  assumes *tarski*: $x \leq x ; T ; x ; T$

begin

— Theorem 13.14

lemma *tarski-mult-top-idempotent*: $x ; T = x ; T ; x ; T$
  by (*metis add-commutative less-eq-def mult-associative star.circ-back-loop-fixpoint star.circ-left-top tarski top-mult-top*)

lemma *tarski-top-omega-below*: $x ; T \leq (x ; T)^\omega$
  by (*metis mult-associative omega-induct-mult order-refl tarski-mult-top-idempotent*)

lemma *tarski-top-omega*: $x ; T = (x ; T)^\omega$
  by (*metis antisym mult-top-omega tarski-top-omega-below*)

lemma *tarski-below-top-omega*: $x \leq (x ; T)^\omega$
  by (*metis tarski-top-omega top-right-mult-increasing*)

lemma *tarski-mult-omega-omega*: $(x ; y^\omega)^\omega = x ; y^\omega$
  by (*metis mult-associative omega-vector tarski-top-omega*)

lemma *tarski-omega-idempotent*: $x^{\omega\omega} = x^\omega$
  by (*metis omega-vector tarski-top-omega*)

lemma *while-denest-2a*: $w ; ((x \star (y ; w)) \star z) = w ; (((x \star y) ; w) \star z)$
proof –
have $(x^\omega + x^\star ; y ; w)^\omega = (x^\star ; y ; w)^\star ; x^\omega ; (((x^\star ; y ; w)^\star ; x^\omega)^\omega + ((x^\star ; y ; w)^\star ; x^\omega)^\star ; (x^\star ; y ; w)^\omega) + (x^\star ; y ; w)^\omega$
by (*metis add-commutative omega-decompose omega-loop-fixpoint*)
also have ... $\le (x^\star ; y ; w)^\star ; x^\omega + (x^\star ; y ; w)^\omega$
by (*metis add-left-isotone mult-associative mult-right-isotone omega-sub-vector*)
finally have *1*: $w ; (x^\omega + x^\star ; y ; w)^\omega \le (w ; x^\star ; y)^\star ; w ; x^\omega + (w ; x^\star ; y)^\omega$
by (*smt add-commutative less-eq-def mult-associative mult-left-dist-add while-def while-slide*)
have $(x^\omega + x^\star ; y ; w)^\star ; z = (x^\star ; y ; w)^\star ; x^\omega ; ((x^\star ; y ; w)^\star ; x^\omega)^\star ; (x^\star ; y ; w)^\star ; z + (x^\star ; y ; w)^\star ; z$
by (*smt add-commutative mult-associative star.circ-add star.circ-loop-fixpoint*)
also have ... $\le (x^\star ; y ; w)^\star ; x^\omega + (x^\star ; y ; w)^\star ; z$
by (*smt add-commutative add-right-isotone mult-associative mult-right-isotone omega-sub-vector*)
finally have $w ; (x^\omega + x^\star ; y ; w)^\star ; z \le (w ; x^\star ; y)^\star ; w ; x^\omega + (w ; x^\star ; y)^\star ; w ; z$
by (*metis mult-associative mult-left-dist-add mult-right-isotone star.circ-slide*)
hence $w ; (x^\omega + x^\star ; y ; w)^\omega + w ; (x^\omega + x^\star ; y ; w)^\star ; z \le (w ; x^\star ; y)^\star ; (w ; x^\omega)^\omega + (w ; x^\star ; y)^\omega + (w ; x^\star ; y)^\star ; w ; z$ **using** *1*
by (*smt add-associative add-commutative less-eq-def mult-associative tarski-mult-omega-omega*)
also have ... $\le (w ; x^\omega + w ; x^\star ; y)^\star ; (w ; x^\omega + w ; x^\star ; y)^\omega + (w ; x^\omega + w ; x^\star ; y)^\omega + (w ; x^\omega + w ; x^\star ; y)^\star ; w ; z$
by (*metis add-isotone add-left-upper-bound add-right-upper-bound mult-isotone mult-left-isotone omega-isotone star.circ-isotone*)
also have ... $= (w ; x^\omega + w ; x^\star ; y)^\omega + (w ; x^\omega + w ; x^\star ; y)^\star ; w ; z$
by (*metis add-idempotent star-mult-omega*)
finally have $w ; ((x^\omega + x^\star ; y ; w)^\omega + (x^\omega + x^\star ; y ; w)^\star ; z) \le w ; ((x^\omega + x^\star ; y) ; w)^\omega + w ; ((x^\omega + x^\star ; y) ; w)^\star ; z$
by (*smt mult-associative mult-left-dist-add omega-slide star.circ-slide*)
hence *2*: $w ; ((x \star (y ; w)) \star z) \le w ; (((x \star y) ; w) \star z)$
by (*smt mult-associative mult-left-dist-add while-def while-slide*)
have $w ; (((x \star y) ; w) \star z) \le w ; ((x \star (y ; w)) \star z)$
by (*metis mult-right-isotone while-left-isotone while-sub-associative*)
thus *?thesis* **using** *2*
by (*metis antisym*)
qed

lemma *while-denest-3*: $(x \star w) \star x^\omega = (x \star w)^\omega$
proof –
have *1*: $(x \star w) \star x^\omega = (x \star w)^\omega + (x \star w)^\star ; x^{\omega\omega}$
by (*metis tarski-omega-idempotent while-def*)
also have ... $\le (x \star w)^\omega + (x \star w)^\star ; (x^\omega + x^\star ; w)^\omega$
by (*metis add-left-upper-bound add-right-isotone mult-right-isotone omega-isotone*)
also have ... $= (x \star w)^\omega$
by (*metis add-idempotent star-mult-omega while-def*)
finally show *?thesis* **using** *1*
by (*metis add-left-upper-bound antisym-conv*)
qed

lemma *while-denest-4a*: $(x \star w) \star (x \star (y ; z)) = (x \star w) \star ((x \star y) ; z)$
proof –
have $(x \star w) \star (x \star (y ; z)) = (x \star w)^\omega + ((x \star w) \star (x^\star ; y ; z))$
by (*smt mult-associative while-denest-3 while-def while-left-dist-add*)
also have ... $\le (x \star w)^\omega + ((x \star w) \star ((x \star y) ; z))$
by (*metis add-right-isotone mult-left-isotone star-below-while while-right-isotone*)
finally have *1*: $(x \star w) \star (x \star (y ; z)) \le (x \star w) \star ((x \star y) ; z)$
by (*smt add-left-upper-bound less-eq-def while-def*)
have $(x \star w) \star ((x \star y) ; z) \le (x \star w) \star (x \star (y ; z))$
by (*metis while-right-isotone while-sub-associative*)
thus *?thesis* **using** *1*
by (*metis antisym*)
qed

— Theorem 8.3

subclass *bounded-extended-binary-itering*
apply *unfold-locales*
apply (*smt mult-associative while-denest-2a while-denest-4a while-increasing while-slide*)
done

— Theorem 13.13

lemma *while-mult-top*: $(x ; T) \star z = z + x ; T$
proof –

have *1*: $z + x ; T \leq (x ; T) \star z$
by (*metis add-least-upper-bound while-denest-1 while-increasing while-one-top*)
have $(x ; T) \star z = z + x ; T ; ((x ; T) \star z)$
by (*metis while-left-unfold*)
also have ... $\leq z + x ; T$
by (*metis add-right-isotone mult-associative mult-right-isotone top-greatest*)
finally show *?thesis* **using** *1*
by (*metis antisym*)
qed

lemma *tarski-top-omega-below-2*: $x ; T \leq (x ; T) \star 0$
by (*metis add-right-divisibility while-mult-top*)

lemma *tarski-top-omega-2*: $x ; T = (x ; T) \star 0$
by (*metis add-left-zero while-mult-top*)

lemma *tarski-below-top-omega-2*: $x \leq (x ; T) \star 0$
by (*metis tarski-top-omega-2 top-right-mult-increasing*)

lemma $1 = (x ; 0) \star 1$ **nitpick** [*expect=genuine*] **oops**

end

class *nonstrict-itering-tarski-zero* = *nonstrict-itering-tarski* + *nonstrict-itering-zero*

begin

lemma $1 = (x ; 0) \star 1$
by (*metis mult-right-zero while-zero*)

— Theorem 13 counterexamples

lemma *while-associative*: $(x \star y) ; z = x \star (y ; z)$ **nitpick** [*expect=genuine*] **oops**
lemma $(x \star 1) ; y = x \star y$ **nitpick** [*expect=genuine*] **oops**
lemma *while-one-mult*: $(x \star 1) ; x = x \star x$ **nitpick** [*expect=genuine*] **oops**
lemma $(x + y) \star z = ((x \star 1) ; y) \star ((x \star 1) ; z)$ **nitpick** [*expect=genuine*] **oops**
lemma *while-mult-top-2*: $(x ; T) \star z = z + x ; T ; z$ **nitpick** [*expect=genuine*] **oops**
lemma *while-top-2*: $T \star z = T ; z$ **nitpick** [*expect=genuine*] **oops**

lemma *tarski*: $x = 0 \vee T ; x ; T = T$ **nitpick** [*expect=genuine*] **oops**
lemma *while-back-loop-is-fixpoint*: *is-fixpoint* $(\lambda x . x ; y + z) (z ; (y \star 1))$ **nitpick** [*expect=genuine*] **oops**
lemma $1 + x ; 0 = x \star 1$ **nitpick** [*expect=genuine*] **oops**
lemma $x = x ; (x \star 1)$ **nitpick** [*expect=genuine*] **oops**
lemma $x ; (x \star 1) = x \star 1$ **nitpick** [*expect=genuine*] **oops**
lemma $x \star 1 = x \star (1 \star 1)$ **nitpick** [*expect=genuine*] **oops**
lemma $(x + y) \star 1 = (x \star (y \star 1)) \star 1$ **nitpick** [*expect=genuine*] **oops**
lemma $z + y ; x = x \longrightarrow y \star z \leq x$ **nitpick** [*expect=genuine*] **oops**
lemma $y ; x = x \longrightarrow y \star x \leq x$ **nitpick** [*expect=genuine*] **oops**
lemma $z + x ; y = x \longrightarrow z ; (y \star 1) \leq x$ **nitpick** [*expect=genuine*] **oops**
lemma $x ; y = x \longrightarrow x ; (y \star 1) \leq x$ **nitpick** [*expect=genuine*] **oops**
lemma $x ; z = z ; y \longrightarrow x \star z \leq z ; (y \star 1)$ **nitpick** [*expect=genuine*] **oops**

lemma *tarski*: $x = 0 \vee T ; x ; T = T$ **nitpick** [*expect=genuine*] **oops**
lemma *tarski-case*: **assumes** *t1*: $x = 0 \longrightarrow P\ x$ **and** *t2*: $T ; x ; T = T \longrightarrow P\ x$ **shows** $P\ x$ **nitpick** [*expect=genuine*] **oops**

end

end

14 NSemiring

theory *NSemiring*

imports *TestItering OmegaAlgebra*

begin

class *n-semiring* = *bounded-idempotent-left-zero-semiring* + *n* + *L* +
  assumes *n-zero* : $n(0) = 0$
  assumes *n-top* : $n(T) = 1$
  assumes *n-dist-add* : $n(x + y) = n(x) + n(y)$
  assumes *n-export* : $n(n(x) ; y) = n(x) ; n(y)$
  assumes *n-sub-mult-zero*: $n(x) = n(x ; 0) ; n(x)$
  assumes *n-L-split* : $x ; n(y) ; L = x ; 0 + n(x ; y) ; L$
  assumes *n-split* : $x \leq x ; 0 + n(x ; L) ; T$

begin

lemma *n-sub-one*: $n(x) \leq 1$
  by (*metis add-left-top add-right-upper-bound n-dist-add n-top*)

— Theorem 15

lemma *n-isotone*: $x \leq y \longrightarrow n(x) \leq n(y)$
  by (*metis less-eq-def n-dist-add*)

lemma *n-mult-idempotent*: $n(x) ; n(x) = n(x)$
  by (*metis mult-associative mult-right-one n-export n-sub-mult-zero n-top*)

— Theorem 15.3

lemma *n-mult-zero*: $n(x) = n(x ; 0)$
  by (*metis add-commutative add-left-top add-right-zero mult-left-dist-add mult-right-one n-dist-add n-sub-mult-zero n-top*)

lemma *n-mult-left-upper-bound*: $n(x) \leq n(x ; y)$
  by (*metis mult-right-isotone n-isotone n-mult-zero zero-least*)

lemma *n-mult-right-zero*: $n(x) ; 0 = 0$
  by (*metis add-left-top add-left-zero mult-left-one mult-right-one n-export n-dist-add n-sub-mult-zero n-top n-zero*)

— Theorem 15.9

lemma *n-mult-n*: $n(x ; n(y)) = n(x)$
  by (*metis mult-associative n-mult-right-zero n-mult-zero*)

lemma *n-mult-left-absorb-add*: $n(x) ; (n(x) + n(y)) = n(x)$
  by (*metis add-left-top mult-left-dist-add mult-right-one n-dist-add n-mult-idempotent n-top*)

lemma *n-mult-right-absorb-add*: $(n(x) + n(y)) ; n(y) = n(y)$
  by (*metis add-commutative add-left-top mult-left-one mult-right-dist-add n-dist-add n-mult-idempotent n-top*)

lemma *n-add-left-absorb-mult*: $n(x) + n(x) ; n(y) = n(x)$
  by (*metis add-left-top mult-left-dist-add mult-right-one n-dist-add n-top*)

lemma *n-add-right-absorb-mult*: $n(x) ; n(y) + n(y) = n(y)$
  by (*metis less-eq-def mult-left-one mult-right-dist-add n-sub-one*)

lemma *n-mult-commutative*: $n(x) ; n(y) = n(y) ; n(x)$
  by (*smt add-commutative mult-left-dist-add mult-right-dist-add n-add-left-absorb-mult n-add-right-absorb-mult n-export n-mult-idempotent*)

lemma *n-add-left-dist-mult*: $n(x) + n(y) ; n(z) = (n(x) + n(y)) ; (n(x) + n(z))$
  by (*metis add-associative mult-left-dist-add mult-right-dist-add n-add-right-absorb-mult n-mult-commutative n-mult-left-absorb-add*)

lemma *n-add-right-dist-mult*: $n(x) ; n(y) + n(z) = (n(x) + n(z)) ; (n(y) + n(z))$
  by (*metis add-commutative n-add-left-dist-mult*)

lemma *n-order*: $n(x) \leq n(y) \longleftrightarrow n(x) ; n(y) = n(x)$
 by (*metis less-eq-def n-add-right-absorb-mult n-mult-left-absorb-add*)

lemma *n-mult-left-lower-bound*: $n(x) ; n(y) \leq n(x)$
 by (*metis add-right-upper-bound n-add-left-absorb-mult*)

lemma *n-mult-right-lower-bound*: $n(x) ; n(y) \leq n(y)$
 by (*metis n-mult-commutative n-mult-left-lower-bound*)

lemma *n-mult-least-upper-bound*: $n(x) \leq n(y) \wedge n(x) \leq n(z) \longleftrightarrow n(x) \leq n(y) ; n(z)$
 by (*smt mult-associative n-export n-mult-left-lower-bound n-order*)

lemma *n-mult-left-divisibility*: $n(x) \leq n(y) \longleftrightarrow (\exists z . n(x) = n(y) ; n(z))$
 by (*metis n-mult-commutative n-mult-left-lower-bound n-order*)

lemma *n-mult-right-divisibility*: $n(x) \leq n(y) \longleftrightarrow (\exists z . n(x) = n(z) ; n(y))$
 by (*metis n-mult-commutative n-mult-left-divisibility*)

— Theorem 15.1

lemma *n-one*: $n(1) = 0$
 by (*metis mult-left-one n-mult-zero n-zero*)

lemma *n-split-equal*: $x + n(x ; L) ; T = x ; 0 + n(x ; L) ; T$
 by (*smt add-associative add-commutative less-eq-def n-split zero-right-mult-decreasing*)

lemma *n-split-top*: $x ; T \leq x ; 0 + n(x ; L) ; T$
 by (*smt mult-associative mult-left-isotone mult-left-zero mult-right-dist-add n-split top-mult-top*)

— Theorem 15.2

lemma *n-L*: $n(L) = 1$
 by (*metis add-left-zero antisym mult-left-one n-export n-isotone n-mult-commutative n-split-top n-sub-one n-top*)

— Theorem 15.5

lemma *n-split-L*: $x ; L = x ; 0 + n(x ; L) ; L$
 by (*metis mult-right-one n-L n-L-split*)

lemma *n-n-L*: $n(n(x) ; L) = n(x)$
 by (*metis mult-right-one n-export n-L*)

lemma *n-L-decreasing*: $n(x) ; L \leq x$
 by (*metis add-left-isotone add-left-zero less-eq-def mult-associative mult-left-zero mult-right-isotone mult-right-one n-mult-zero n-split-L order-trans*)

— Theorem 15.10

lemma *n-galois*: $n(x) \leq n(y) \longleftrightarrow n(x) ; L \leq y$
 by (*metis less-eq-def mult-left-one mult-right-sub-dist-add-left n-export n-isotone n-L n-L-decreasing n-mult-commutative order-trans*)

— Theorem 15.6

lemma *split-L*: $x ; L \leq x ; 0 + L$
 by (*metis add-commutative add-left-isotone n-galois n-L n-split-L n-sub-one*)

— Theorem 15.7

lemma *L-left-zero*: $L ; x = L$
 by (*metis add-right-zero less-eq-def mult-associative mult-left-one mult-left-sub-dist-add-right mult-left-zero mult-right-one n-L n-mult-left-upper-bound n-order n-split-L*)

— Theorem 15.8

lemma *n-mult*: $n(x ; n(y) ; L) = n(x ; y)$
 by (*metis less-eq-def n-dist-add n-mult-left-upper-bound n-mult-zero n-n-L n-L-split*)

lemma *n-mult-top*: $n(x ; n(y) ; T) = n(x ; y)$

by (*metis mult-right-one n-mult n-top*)

— Theorem 15.4

lemma *n-top-L*: $n(x ; T) = n(x ; L)$
by (*metis mult-right-one n-L n-mult-top*)

lemma *n-top-split*: $x ; n(y) ; T \leq x ; 0 + n(x ; y) ; T$
by (*metis mult-associative n-mult n-mult-right-zero n-split-top*)

lemma *n-mult-right-upper-bound*: $n(x ; y) \leq n(z) \longleftrightarrow n(x) \leq n(z) \wedge x ; n(y) ; L \leq x ; 0 + n(z) ; L$
apply (*rule iffI*)
apply (*metis add-right-isotone eq-iff mult-isotone n-L-split n-mult-left-upper-bound order-trans*)
apply (*smt add-least-upper-bound less-eq-def mult-left-one n-export n-dist-add n-galois n-L n-L-split n-mult-commutative n-mult-zero*)
done

lemma *n-preserves-equation*: $n(y) ; x \leq x ; n(y) \longleftrightarrow n(y) ; x = n(y) ; x ; n(y)$
by (*metis eq-refl test-preserves-equation n-mult-idempotent n-sub-one*)

definition *ni* :: $'a \Rightarrow 'a$
where $ni\ x = n(x) ; L$

lemma *ni-zero*: $ni(0) = 0$
by (*metis mult-left-zero n-zero ni-def*)

lemma *ni-one*: $ni(1) = 0$
by (*metis mult-left-zero n-one ni-def*)

lemma *ni-L*: $ni(L) = L$
by (*metis mult-left-one n-L ni-def*)

lemma *ni-top*: $ni(T) = L$
by (*metis mult-left-one n-top ni-def*)

lemma *ni-dist-add*: $ni(x + y) = ni(x) + ni(y)$
by (*metis mult-right-dist-add n-dist-add ni-def*)

lemma *ni-mult-zero*: $ni(x) = ni(x ; 0)$
by (*metis n-mult-zero ni-def*)

lemma *ni-split*: $x ; ni(y) = x ; 0 + ni(x ; y)$
by (*metis mult-associative n-L-split ni-def*)

lemma *ni-decreasing*: $ni(x) \leq x$
by (*metis n-L-decreasing ni-def*)

lemma *ni-isotone*: $x \leq y \longrightarrow ni(x) \leq ni(y)$
by (*metis less-eq-def ni-dist-add*)

lemma *ni-mult-left-upper-bound*: $ni(x) \leq ni(x ; y)$
by (*metis n-galois n-mult-left-upper-bound n-n-L ni-def*)

lemma *ni-idempotent*: $ni(ni(x)) = ni(x)$
by (*metis n-n-L ni-def*)

lemma *ni-below-L*: $ni(x) \leq L$
by (*metis n-L n-galois n-sub-one ni-def*)

lemma *ni-left-zero*: $ni(x) ; y = ni(x)$
by (*metis L-left-zero mult-associative ni-def*)

lemma *ni-split-L*: $x ; L = x ; 0 + ni(x ; L)$
by (*metis n-split-L ni-def*)

lemma *ni-top-L*: $ni(x ; T) = ni(x ; L)$
by (*metis n-top-L ni-def*)

lemma *ni-galois*: $ni(x) \leq ni(y) \longleftrightarrow ni(x) \leq y$

by (*metis n-galois n-n-L ni-def*)

lemma *ni-mult*: $ni(x ; ni(y)) = ni(x ; y)$
by (*metis mult-associative n-mult ni-def*)

lemma *ni-n-order*: $ni(x) \leq ni(y) \longleftrightarrow n(x) \leq n(y)$
by (*metis n-galois n-n-L ni-def*)

lemma *ni-n-equal*: $ni(x) = ni(y) \longleftrightarrow n(x) = n(y)$
by (*metis n-n-L ni-def*)

lemma *ni-mult-right-upper-bound*: $ni(x ; y) \leq ni(z) \longleftrightarrow ni(x) \leq ni(z) \wedge x ; ni(y) \leq x ; 0 + ni(z)$
by (*smt mult-associative n-mult-right-upper-bound ni-def ni-n-order*)

lemma *n-ni*: $n(ni(x)) = n(x)$
by (*metis n-n-L ni-def*)

lemma *ni-n*: $ni(n(x)) = 0$
by (*metis n-mult-right-zero ni-mult-zero ni-zero*)

lemma *ni-n-galois*: $n(x) \leq n(y) \longleftrightarrow ni(x) \leq y$
by (*metis n-galois ni-def*)

lemma *n-mult-ni*: $n(x ; ni(y)) = n(x ; y)$
by (*metis mult-associative n-mult ni-def*)

lemma *ni-mult-n*: $ni(x ; n(y)) = ni(x)$
by (*metis n-mult-n ni-def*)

lemma *ni-export*: $ni(n(x) ; y) = n(x) ; ni(y)$
by (*metis mult-associative n-export ni-def*)

lemma *ni-mult-top*: $ni(x ; n(y) ; T) = ni(x ; y)$
by (*metis n-mult-top ni-def*)

lemma *ni-n-zero*: $ni(x) = 0 \longleftrightarrow n(x) = 0$
by (*metis mult-left-zero n-ni n-zero ni-def*)

lemma *ni-n-L*: $ni(x) = L \longleftrightarrow n(x) = 1$
by (*metis mult-left-one n-L n-n-L ni-def*)

end

typedef *'a nImage* = $\{ x::'a::n\text{-}semiring . (\exists y::'a . x = n(y)) \}$
by *auto*

lemma *simp-nImage* [*simp*]: $\exists y$. *Rep-nImage* $x = n(y)$
using *Rep-nImage*
by *simp*

setup-lifting *type-definition-nImage*

— Theorem 15

instantiation *nImage* :: (*n-semiring*) *bounded-idempotent-semiring*

begin

lift-definition *plus-nImage* :: *'a nImage* ⇒ *'a nImage* ⇒ *'a nImage* **is** *plus*
by (*metis n-dist-add*)

lift-definition *times-nImage* :: *'a nImage* ⇒ *'a nImage* ⇒ *'a nImage* **is** *times*
by (*metis n-export*)

lift-definition *zero-nImage* :: *'a nImage* **is** *0*
by (*metis n-zero*)

lift-definition *one-nImage* :: *'a nImage* **is** *1*
  by (*metis n-L*)

lift-definition *T-nImage* :: *'a nImage* **is** *1*
  by (*metis n-L*)

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

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

instance
  apply *intro-classes*
  apply (*metis* (*mono-tags*) *Rep-nImage-inject add-associative plus-nImage.rep-eq*)
  apply (*metis* (*mono-tags*) *Rep-nImage-inject add-commutative plus-nImage.rep-eq*)
  apply (*metis* (*mono-tags*) *Rep-nImage-inject add-idempotent plus-nImage.rep-eq*)
  apply (*metis* (*mono-tags*) *Rep-nImage-inject less-eq-def less-eq-nImage.rep-eq plus-nImage.rep-eq*)
  apply (*metis less-eq-nImage.rep-eq less-nImage.rep-eq less-def*)
  apply (*smt2 zero-nImage.rep-eq Rep-nImage-inject add-left-zero plus-nImage.rep-eq*)
  apply (*metis* (*mono-tags*) *times-nImage.rep-eq dual-order.refl less-eq-nImage.rep-eq mult-left-dist-add plus-nImage.rep-eq*)
  apply (*metis* (*mono-tags*) *plus-nImage.rep-eq times-nImage.rep-eq Rep-nImage-inject mult-right-dist-add*)
  apply (*smt2 times-nImage.rep-eq zero-nImage.rep-eq Rep-nImage-inverse mult-left-zero*)
  apply (*smt2 Rep-nImage-inverse mult-left-one one-nImage.rep-eq times-nImage.rep-eq*)
  apply (*simp add*: *less-eq-nImage.rep-eq mult-right-one one-nImage.rep-eq times-nImage.rep-eq*)
  apply (*simp add*: *less-eq-nImage.rep-eq mult-associative times-nImage.rep-eq*)
  apply (*smt2 T-nImage.abs-eq T-nImage.rep-eq less-eq-def map-fun-apply n-sub-one plus-nImage-def simp-nImage*)
  apply (*metis* (*mono-tags*) *Rep-nImage-inverse mult-associative times-nImage.rep-eq*)
  apply (*smt2 one-nImage.rep-eq Rep-nImage-inverse mult-right-one times-nImage.rep-eq*)
  apply (*metis* (*mono-tags*) *plus-nImage.rep-eq times-nImage.rep-eq Rep-nImage-inject mult-left-dist-add*)
  apply (*smt2 times-nImage.rep-eq zero-nImage.rep-eq Rep-nImage-inverse n-mult-right-zero simp-nImage*)
  done

end

— Theorem 15

instantiation *nImage* :: (*n-semiring*) *bounded-distributive-lattice*

begin

lift-definition *meet-nImage* :: *'a nImage* ⇒ *'a nImage* ⇒ *'a nImage* **is** *times*
  by (*metis n-export*)

instance
  apply *intro-classes*
  apply (*metis* (*mono-tags*) *Rep-nImage-inject meet-nImage.rep-eq mult-associative*)
  apply (*metis* (*mono-tags*) *meet-nImage.rep-eq Rep-nImage-inverse simp-nImage n-mult-commutative*)
  apply (*metis* (*mono-tags*) *meet-nImage.rep-eq Rep-nImage-inverse simp-nImage n-mult-idempotent*)
  apply (*metis* (*mono-tags*) *meet-nImage.rep-eq Rep-nImage-inverse simp-nImage n-order less-eq-nImage.rep-eq*)
  apply (*metis less-def*)
  apply (*metis* (*mono-tags*) *T-nImage.abs-eq meet-nImage-def mult-left-one-1 one-nImage.abs-eq times-nImage-def*)
  apply (*metis* (*mono-tags*) *Rep-nImage-inject meet-nImage.rep-eq mult-left-dist-add plus-nImage.rep-eq*)
  apply (*metis* (*mono-tags*) *Rep-nImage-inject meet-nImage.rep-eq n-add-left-dist-mult plus-nImage.rep-eq simp-nImage*)
  apply (*metis* (*mono-tags*) *Rep-nImage-inject meet-nImage.rep-eq n-mult-left-absorb-add plus-nImage.rep-eq simp-nImage*)
  apply (*metis* (*mono-tags*) *Rep-nImage-inject meet-nImage.rep-eq n-add-left-absorb-mult plus-nImage.rep-eq simp-nImage*)
  done

end

class *n-itering* = *bounded-itering* + *n-semiring*

begin

lemma *mult-L-circ*: $(x\ ;\ L)^\circ = 1 + x\ ;\ L$
  by (*metis L-left-zero circ-mult mult-associative*)

lemma *mult-L-circ-mult*: $(x\ ;\ L)^\circ\ ;\ y = y + x\ ;\ L$
  by (*metis L-left-zero mult-L-circ mult-associative mult-left-one mult-right-dist-add*)

lemma *circ-L*: $L^\circ = L + 1$

by (*metis L-left-zero add-commutative circ-left-unfold*)

lemma *circ-n-L*: $x^\circ \ ; \ n(x) \ ; \ L = x^\circ \ ; \ 0$
by (*metis add-left-zero circ-left-unfold circ-plus-same mult-left-zero n-L-split n-dist-add n-mult-zero n-one ni-def ni-split*)

lemma *n-circ-left-unfold*: $n(x^\circ) = n(x \ ; \ x^\circ)$
by (*metis circ-n-L circ-plus-same n-mult n-mult-zero*)

lemma *ni-circ*: $ni(x)^\circ = 1 + ni(x)$
by (*metis mult-L-circ ni-def*)

lemma *circ-ni*: $x^\circ \ ; \ ni(x) = x^\circ \ ; \ 0$
by (*metis circ-n-L mult-associative ni-def*)

lemma *ni-circ-left-unfold*: $ni(x^\circ) = ni(x \ ; \ x^\circ)$
by (*metis n-circ-left-unfold ni-def*)

lemma *n-circ-import*: $n(y) \ ; \ x \leq x \ ; \ n(y) \longrightarrow n(y) \ ; \ x^\circ = n(y) \ ; \ (n(y) \ ; \ x)^\circ$
apply *rule*
apply (*rule antisym*)
apply (*metis circ-simulate circ-slide mult-associative n-mult-idempotent n-preserves-equation*)
apply (*metis circ-isotone mult-left-isotone mult-left-one mult-right-isotone n-sub-one*)
done

end

class *n-omega-itering* = *left-omega-conway-semiring* + *n-itering* +
assumes *circ-circ*: $x^{\circ\circ} = L + x^\star$

begin

lemma *L-below-one-circ*: $L \leq 1^\circ$
by (*metis add-left-divisibility circ-circ circ-one*)

lemma *circ-below-L-add-star*: $x^\circ \leq L + x^\star$
by (*metis circ-circ circ-increasing*)

lemma *L-add-circ-add-star*: $L + x^\circ = L + x^\star$
by (*smt add-associative add-commutative circ-below-L-add-star less-eq-def star-below-circ*)

lemma *circ-one-L*: $1^\circ = L + 1$
by (*metis L-add-circ-add-star L-below-one-circ less-eq-def star-one*)

lemma *one-circ-zero*: $L = 1^\circ \ ; \ 0$
by (*metis L-left-zero circ-L circ-ni circ-one-L circ-plus-same ni-L*)

lemma *circ-not-simulate*: $(\forall x \ y \ z \ . \ x \ ; \ z \leq z \ ; \ y \longrightarrow x^\circ \ ; \ z \leq z \ ; \ y^\circ) \longrightarrow 1 = 0$
by (*metis L-left-zero circ-one-L eq-iff mult-left-one mult-left-zero mult-right-sub-dist-add-left n-L n-zero zero-least*)

lemma *star-circ-L*: $x^{\star\circ} = L + x^\star$
by (*smt L-add-circ-add-star L-left-zero add-commutative circ-add-1 circ-mult-star circ-one-L circ-star star.circ-loop-fixpoint star.circ-plus-one star-sup-one*)

lemma *circ-circ-2*: $x^{\circ\circ} = L + x^\circ$
by (*metis circ-star star-circ-L*)

lemma *circ-add-6*: $L + (x + y)^\circ = (x^\circ \ ; \ y^\circ)^\circ$
by (*metis circ-circ-2 add-associative add-commutative circ-add-1 circ-circ-add circ-decompose-4*)

lemma *circ-add-7*: $(x^\circ \ ; \ y^\circ)^\circ = L + (x + y)^\star$
by (*metis L-add-circ-add-star circ-add-6*)

end

class *n-omega-algebra-2* = *bounded-left-zero-omega-algebra* + *n-semiring* + *Omega* +
assumes *Omega-def*: $x^\Omega = n(x^\omega) \ ; \ L + x^\star$

begin

lemma *mult-L-star*: $(x \;;\; L)^\star = 1 + x \;;\; L$
by (*metis L-left-zero mult-associative star.circ-mult*)

lemma *mult-L-omega*: $(x \;;\; L)^\omega = x \;;\; L$
by (*metis L-left-zero omega-slide*)

lemma *mult-L-add-star*: $(x \;;\; L + y)^\star = y^\star + y^\star \;;\; x \;;\; L$
by (*metis L-left-zero add-commutative mult-L-star mult-associative mult-left-dist-add mult-right-one star.circ-add-1*)

lemma *mult-L-add-omega*: $(x \;;\; L + y)^\omega = y^\omega + y^\star \;;\; x \;;\; L$
by (*smt L-left-zero add-commutative add-left-upper-bound less-eq-def mult-L-omega mult-L-star mult-associative mult-left-one mult-right-dist-add omega-decompose*)

lemma *mult-L-add-circ*: $(x \;;\; L + y)^\Omega = n(y^\omega) \;;\; L + y^\star + y^\star \;;\; x \;;\; L$
by (*smt add-associative add-commutative Omega-def less-eq-def mult-L-add-omega mult-L-add-star mult-right-dist-add n-L-decreasing n-dist-add*)

lemma *circ-add-n*: $(x^\Omega \;;\; y)^\Omega \;;\; x^\Omega = n((x^\star \;;\; y)^\omega) \;;\; L + ((x^\star \;;\; y)^\star \;;\; x^\star + (x^\star \;;\; y)^\star \;;\; n(x^\omega) \;;\; L)$
by (*smt L-left-zero add-associative add-commutative Omega-def mult-L-add-circ mult-associative mult-left-dist-add mult-right-dist-add*)

— Theorem 20.6

lemma *n-omega-induct*: $n(y) \le n(x \;;\; y + z) \longrightarrow n(y) \le n(x^\omega + x^\star \;;\; z)$
by (*smt add-commutative mult-associative n-dist-add n-galois n-mult omega-induct*)

lemma *n-Omega-left-unfold*: $1 + x \;;\; x^\Omega = x^\Omega$
proof –
have $1 + x \;;\; x^\Omega = 1 + x \;;\; n(x^\omega) \;;\; L + x \;;\; x^\star$
by (*metis add-associative add-left-zero mult-associative mult-left-dist-add mult-left-zero mult-L-add-circ mult-L-add-star*)
also have ... $= n(x \;;\; x^\omega) \;;\; L + (1 + x \;;\; x^\star)$
by (*metis add-associative add-commutative add-left-zero mult-left-dist-add n-L-split*)
also have ... $= n(x^\omega) \;;\; L + x^\star$
by (*metis omega-unfold star-left-unfold-equal*)
also have ... $= x^\Omega$
by (*metis Omega-def*)
finally show *?thesis*
by *metis*
qed

lemma *n-Omega-circ-add*: $(x + y)^\Omega = (x^\Omega \;;\; y)^\Omega \;;\; x^\Omega$
proof –
have $(x^\Omega \;;\; y)^\Omega \;;\; x^\Omega = n((x^\star \;;\; y)^\omega) \;;\; L + ((x^\star \;;\; y)^\star \;;\; x^\star + (x^\star \;;\; y)^\star \;;\; n(x^\omega) \;;\; L)$
by (*metis circ-add-n*)
also have ... $= n((x^\star \;;\; y)^\omega) \;;\; L + n((x^\star \;;\; y)^\star \;;\; x^\omega) \;;\; L + (x^\star \;;\; y)^\star \;;\; 0 + (x^\star \;;\; y)^\star \;;\; x^\star$
by (*smt n-L-split add-commutative add-associative mult-associative*)
also have ... $= n((x^\star \;;\; y)^\omega + (x^\star \;;\; y)^\star \;;\; x^\omega) \;;\; L + (x^\star \;;\; y)^\star \;;\; x^\star$
by (*smt2 add-associative add-left-zero mult-left-dist-add mult-right-dist-add n-dist-add*)
also have ... $= (x + y)^\Omega$
by (*metis Omega-def omega-decompose star.circ-add*)
finally show *?thesis*
..
qed

lemma *n-Omega-circ-simulate-right-plus*: $z \;;\; x \le y \;;\; y^\Omega \;;\; z + w \longrightarrow z \;;\; x^\Omega \le y^\Omega \;;\; (z + w \;;\; x^\Omega)$
proof
assume $z \;;\; x \le y \;;\; y^\Omega \;;\; z + w$
also have ... $= y \;;\; n(y^\omega) \;;\; L + y \;;\; y^\star \;;\; z + w$
by (*metis L-left-zero Omega-def add-commutative mult-associative mult-left-dist-add mult-right-dist-add*)
finally have *1*: $z \;;\; x \le n(y^\omega) \;;\; L + y \;;\; y^\star \;;\; z + w$
by (*metis add-associative add-commutative add-left-zero mult-associative mult-left-dist-add n-L-split omega-unfold*)
hence $(n(y^\omega) \;;\; L + y^\star \;;\; z + y^\star \;;\; w \;;\; n(x^\omega) \;;\; L + y^\star \;;\; w \;;\; x^\star) \;;\; x \le n(y^\omega) \;;\; L + y^\star \;;\; (n(y^\omega) \;;\; L + y \;;\; y^\star \;;\; z + w) + y^\star \;;\; w \;;\; n(x^\omega) \;;\; L + y^\star \;;\; w \;;\; x^\star$
by (*smt L-left-zero add-associative add-left-upper-bound add-right-upper-bound less-eq-def mult-associative mult-left-dist-add mult-right-dist-add star.circ-back-loop-fixpoint*)
also have ... $= n(y^\omega) \;;\; L + y^\star \;;\; n(y^\omega) \;;\; L + y^\star \;;\; y \;;\; y^\star \;;\; z + y^\star \;;\; w + y^\star \;;\; w \;;\; n(x^\omega) \;;\; L + y^\star \;;\; w \;;\; x^\star$
by (*metis add-associative mult-associative mult-left-dist-add*)
also have ... $= n(y^\omega) \;;\; L + y^\star \;;\; n(y^\omega) \;;\; L + y^\star \;;\; y \;;\; y^\star \;;\; z + y^\star \;;\; w \;;\; n(x^\omega) \;;\; L + y^\star \;;\; w \;;\; x^\star$
by (*smt2 add-associative add-commutative add-idempotent star.circ-back-loop-fixpoint*)

also have ... $= n(y^\omega)$; $L + y^\star$; y ; $y^\star$; $z + y^\star$; w ; $n(x^\omega)$; $L + y^\star$; w ; $x^\star$
by (*smt2 add-associative add-commutative add-idempotent mult-associative mult-left-dist-add n-L-split star-mult-omega*)
also have ... $\leq n(y^\omega)$; $L + y^\star$; $z + y^\star$; w ; $n(x^\omega)$; $L + y^\star$; w ; $x^\star$
by (*metis add-left-isotone add-right-isotone mult-left-isotone star.circ-plus-same star.circ-transitive-equal star.left-plus-below-circ*)
finally have *2*: z ; $x^\star \leq n(y^\omega)$; $L + y^\star$; $z + y^\star$; w ; $n(x^\omega)$; $L + y^\star$; w ; $x^\star$
by (*smt add-least-upper-bound add-left-upper-bound star.circ-loop-fixpoint star-right-induct*)
have z ; x ; $x^\omega \leq n(y^\omega)$; $L + y$; $y^\star$; z ; $x^\omega + w$; x^ω **using** *1*
by (*metis L-left-zero mult-associative mult-left-isotone mult-right-dist-add*)
hence $n(z$; x ; $x^\omega) \leq n(y$; $y^\star$; z ; $x^\omega + n(y^\omega)$; $L + w$; $x^\omega)$
by (*metis add-commutative n-isotone*)
hence $n(z$; $x^\omega) \leq n(y^\omega + y^\star$; w ; $x^\omega)$
by (*smt2 add-associative add-commutative left-plus-omega less-eq-def mult-associative mult-left-dist-add n-L-decreasing n-omega-induct omega-unfold star.left-plus-circ star-mult-omega*)
hence $n(z$; $x^\omega)$; $L \leq n(y^\omega)$; $L + y^\star$; w ; $n(x^\omega)$; L
by (*metis eq-iff mult-right-dist-add n-dist-add n-galois n-mult order-trans*)
hence z ; $n(x^\omega)$; $L \leq z$; $0 + n(y^\omega)$; $L + y^\star$; w ; $n(x^\omega)$; L
by (*metis add-associative add-right-isotone n-L-split*)
also have ... $\leq n(y^\omega)$; $L + y^\star$; $z + y^\star$; w ; $n(x^\omega)$; L
by (*smt2 add-commutative add-left-isotone add-left-upper-bound order-trans star.circ-loop-fixpoint zero-right-mult-decreasing*)
finally have z ; $n(x^\omega)$; $L \leq n(y^\omega)$; $L + y^\star$; $z + y^\star$; w ; $n(x^\omega)$; $L + y^\star$; w ; $x^\star$
by (*metis add-commutative add-left-upper-bound order-trans*)
thus z ; $x^\Omega \leq y^\Omega$; $(z + w$; $x^\Omega)$ **using** *2*
by (*smt L-left-zero Omega-def add-associative less-eq-def mult-associative mult-left-dist-add mult-right-dist-add*)
qed

lemma *n-Omega-circ-simulate-left-plus*: x ; $z \leq z$; $y^\Omega + w \longrightarrow x^\Omega$; $z \leq (z + x^\Omega$; $w)$; y^Ω
proof
assume *1*: x ; $z \leq z$; $y^\Omega + w$
have x ; $(z$; $n(y^\omega)$; $L + z$; $y^\star + n(x^\omega)$; $L + x^\star$; w ; $n(y^\omega)$; $L + x^\star$; w ; $y^\star) = x$; z ; $n(y^\omega)$; $L + x$; z ; $y^\star + n(x^\omega)$; $L + x$; $x^\star$; w ; $n(y^\omega)$; $L + x$; $x^\star$; w ; $y^\star$
by (*smt add-associative add-commutative mult-associative mult-left-dist-add n-L-split omega-unfold*)
also have ... $\leq (z$; $n(y^\omega)$; $L + z$; $y^\star + w)$; $n(y^\omega)$; $L + (z$; $n(y^\omega)$; $L + z$; $y^\star + w)$; $y^\star + n(x^\omega)$; $L + x^\star$; w ; $n(y^\omega)$; $L + x^\star$; w ; $y^\star$ **using** *1*
by (*smt Omega-def add-associative add-right-upper-bound less-eq-def mult-associative mult-left-dist-add mult-right-dist-add star.circ-loop-fixpoint*)
also have ... $= z$; $n(y^\omega)$; $L + z$; $y^\star$; $n(y^\omega)$; $L + w$; $n(y^\omega)$; $L + z$; $y^\star + w$; $y^\star + n(x^\omega)$; $L + x^\star$; w ; $n(y^\omega)$; $L + x^\star$; w ; $y^\star$
by (*smt2 L-left-zero add-associative add-commutative add-idempotent mult-associative mult-right-dist-add star.circ-transitive-equal*)
also have ... $= z$; $n(y^\omega)$; $L + w$; $n(y^\omega)$; $L + z$; $y^\star + w$; $y^\star + n(x^\omega)$; $L + x^\star$; w ; $n(y^\omega)$; $L + x^\star$; w ; $y^\star$
by (*smt add-associative add-commutative add-idempotent less-eq-def mult-associative n-L-split star-mult-omega zero-right-mult-decreasing*)
finally have x ; $(z$; $n(y^\omega)$; $L + z$; $y^\star + n(x^\omega)$; $L + x^\star$; w ; $n(y^\omega)$; $L + x^\star$; w ; $y^\star) \leq z$; $n(y^\omega)$; $L + z$; $y^\star + n(x^\omega)$; $L + x^\star$; w ; $n(y^\omega)$; $L + x^\star$; w ; $y^\star$
by (*smt2 add-associative add-commutative add-idempotent mult-associative star.circ-loop-fixpoint*)
thus x^Ω ; $z \leq (z + x^\Omega$; $w)$; y^Ω
by (*smt L-left-zero Omega-def add-associative add-least-upper-bound add-left-upper-bound mult-associative mult-left-dist-add mult-right-dist-add star.circ-back-loop-fixpoint star-left-induct*)
qed

end

— Theorem 2.6 and Theorem 19

sublocale *n-omega-algebra-2* < *nL-omega*!: *itering* **where** *circ* = *Omega*
apply *unfold-locales*
apply (*smt add-associative add-commutative add-left-zero circ-add-n Omega-def mult-left-dist-add mult-right-dist-add n-L-split n-dist-add omega-decompose star.circ-add-1 star.circ-slide*)
apply (*smt L-left-zero add-associative add-commutative add-left-zero Omega-def mult-associative mult-left-dist-add mult-right-dist-add n-L-split omega-slide star.circ-mult*)
apply (*metis n-Omega-circ-simulate-right-plus*)
apply (*metis n-Omega-circ-simulate-left-plus*)
done

sublocale *n-omega-algebra-2* < *nL-omega*!: *n-omega-itering* **where** *circ* = *Omega*
apply *unfold-locales*
apply (*smt2 Omega-def add-associative add-commutative less-eq-def mult-L-add-star mult-left-one n-L-split n-top ni-below-L ni-def star-involutive star-mult-omega star-omega-top zero-right-mult-decreasing*)

done

sublocale *n-omega-algebra-2* < *nL-omega*!: *left-zero-kleene-conway-semiring* **where** *circ* = *Omega* **..**

sublocale *n-omega-algebra-2* < *nL-star*!: *left-omega-conway-semiring* **where** *circ* = *star* **..**

context *n-omega-algebra-2*

begin

lemma *circ-add-8*: $n((x^\star ; y)^\star ; x^\omega) ; L \leq (x^\star ; y)^\Omega ; x^\Omega$
by (*metis add-left-upper-bound nL-omega.circ-add-4 Omega-def mult-left-isotone n-isotone omega-sum-unfold-3 order-trans*)

lemma *n-split-omega-omega*: $x^\omega \leq x^\omega ; 0 + n(x^\omega) ; T$
by (*metis n-split n-top-L omega-vector*)

— Theorem 20.1

lemma *n-below-n-star*: $n(x) \leq n(x^\star)$
by (*metis n-isotone star.circ-increasing*)

— Theorem 20.2

lemma *n-star-below-n-omega*: $n(x^\star) \leq n(x^\omega)$
by (*metis n-mult-left-upper-bound star-mult-omega*)

lemma *n-below-n-omega*: $n(x) \leq n(x^\omega)$
by (*metis n-mult-left-upper-bound omega-unfold*)

— Theorem 20.4

lemma *star-n-L*: $x^\star ; n(x) ; L = x^\star ; 0$
by (*metis add-left-zero mult-left-zero n-L-split n-dist-add n-mult-zero n-one ni-def ni-split star-left-unfold-equal star-plus*)

lemma *star-L-split*: $y \leq z \wedge x ; z ; L \leq x ; 0 + z ; L \longrightarrow x^\star ; y ; L \leq x^\star ; 0 + z ; L$
proof
assume *1*: $y \leq z \wedge x ; z ; L \leq x ; 0 + z ; L$
have $x ; (x^\star ; 0 + z ; L) \leq x^\star ; 0 + x ; z ; L$
by (*metis add-right-zero eq-iff mult-associative mult-left-dist-add star.circ-loop-fixpoint*)
also have ... $\leq x^\star ; 0 + x ; 0 + z ; L$ **using** *1*
by (*metis add-associative add-left-upper-bound less-eq-def*)
also have ... $= x^\star ; 0 + z ; L$
by (*metis add-commutative less-eq-def mult-right-dist-add star.circ-increasing*)
finally have $y ; L + x ; (x^\star ; 0 + z ; L) \leq x^\star ; 0 + z ; L$ **using** *1*
by (*metis add-least-upper-bound add-right-upper-bound mult-left-isotone order-trans*)
thus $x^\star ; y ; L \leq x^\star ; 0 + z ; L$
by (*metis star-left-induct mult-associative*)
qed

lemma *star-L-split-same*: $x ; y ; L \leq x ; 0 + y ; L \longrightarrow x^\star ; y ; L = x^\star ; 0 + y ; L$
by (*smt add-associative add-left-zero antisym less-eq-def mult-associative mult-left-dist-add mult-left-one mult-right-sub-dist-add-left order-refl star-L-split star.circ-right-unfold*)

lemma *star-n-L-split-equal*: $n(x ; y) \leq n(y) \longrightarrow x^\star ; n(y) ; L = x^\star ; 0 + n(y) ; L$
by (*metis n-mult-right-upper-bound star-L-split-same*)

lemma *n-star-mult*: $n(x ; y) \leq n(y) \longrightarrow n(x^\star ; y) = n(x^\star) + n(y)$
by (*metis n-dist-add n-mult n-mult-zero n-n-L star-n-L-split-equal*)

— Theorem 20.3

lemma *n-omega-mult*: $n(x^\omega ; y) = n(x^\omega)$
by (*smt add-commutative less-eq-def n-dist-add n-mult-left-upper-bound omega-sub-vector*)

lemma *n-star-left-unfold*: $n(x^\star) = n(x ; x^\star)$
by (*metis n-mult n-mult-zero star.circ-plus-same star-n-L*)

lemma *ni-star-below-ni-omega*: $ni(x^\star) \leq ni(x^\omega)$
by (*metis n-star-below-n-omega ni-n-order*)

lemma *ni-below-ni-omega*: $ni(x) \leq ni(x^{\omega})$
 by (*metis n-below-n-omega ni-n-order*)

lemma *ni-star*: $ni(x)^{\star} = 1 + ni(x)$
 by (*metis mult-L-star ni-def*)

lemma *ni-omega*: $ni(x)^{\omega} = ni(x)$
 by (*metis mult-L-omega ni-def*)

lemma *ni-omega-induct*: $ni(y) \leq ni(x ; y + z) \longrightarrow ni(y) \leq ni(x^{\omega} + x^{\star} ; z)$
 by (*metis n-omega-induct ni-n-order*)

lemma *star-ni*: $x^{\star} ; ni(x) = x^{\star} ; 0$
 by (*metis mult-associative ni-def star-n-L*)

lemma *star-ni-split-equal*: $ni(x ; y) \leq ni(y) \longrightarrow x^{\star} ; ni(y) = x^{\star} ; 0 + ni(y)$
 by (*metis mult-associative ni-def ni-n-order star-n-L-split-equal*)

lemma *ni-star-mult*: $ni(x ; y) \leq ni(y) \longrightarrow ni(x^{\star} ; y) = ni(x^{\star}) + ni(y)$
 by (*metis n-dist-add n-star-mult ni-dist-add ni-n-equal ni-n-order*)

lemma *ni-omega-mult*: $ni(x^{\omega} ; y) = ni(x^{\omega})$
 by (*metis n-omega-mult ni-def*)

lemma *ni-star-left-unfold*: $ni(x^{\star}) = ni(x ; x^{\star})$
 by (*metis n-star-left-unfold ni-def*)

lemma *n-star-import*: $n(y) ; x \leq x ; n(y) \longrightarrow n(y) ; x^{\star} = n(y) ; (n(y) ; x)^{\star}$
 apply (*rule impI*, *rule antisym*)
 defer
 apply (*metis mult-left-isotone mult-left-one mult-right-isotone n-sub-one star.circ-isotone*)
 proof −
  assume $n(y) ; x \leq x ; n(y)$
  hence $n(y) ; (n(y) ; x)^{\star} ; x \leq n(y) ; (n(y) ; x)^{\star}$
   by (*smt2 mult-associative mult-right-dist-add mult-right-sub-dist-add-left n-mult-idempotent n-preserves-equation star.circ-back-loop-fixpoint*)
  thus $n(y) ; x^{\star} \leq n(y) ; (n(y) ; x)^{\star}$
   by (*smt add-associative less-eq-def mult-left-dist-add mult-right-one star.circ-plus-one star-right-induct*)
 qed

lemma *n-omega-export*: $n(y) ; x \leq x ; n(y) \longrightarrow n(y) ; x^{\omega} = (n(y) ; x)^{\omega}$
 apply (*rule impI*, *rule antisym*)
 apply (*metis mult-associative mult-right-isotone n-mult-idempotent omega-simulation*)
 apply (*metis mult-right-isotone mult-right-one n-sub-one omega-isotone omega-slide*)
 done

lemma *n-omega-import*: $n(y) ; x \leq x ; n(y) \longrightarrow n(y) ; x^{\omega} = n(y) ; (n(y) ; x)^{\omega}$
 by (*metis mult-associative n-order n-omega-export order-refl*)

— Theorem 20.5

lemma *star-n-omega-top*: $x^{\star} ; n(x^{\omega}) ; T = x^{\star} ; 0 + n(x^{\omega}) ; T$
 by (*smt add-least-upper-bound add-right-divisibility antisym mult-associative nL-star.circ-mult-omega nL-star.star-zero-below-circ-mult n-top-split star.circ-loop-fixpoint*)

lemma *n-star-induct-add*: $n(z + x ; y) \leq n(y) \longrightarrow n(x^{\star} ; z) \leq n(y)$ **oops**

end

end

15 Approximation

```
theory Approximation

imports Semiring

begin

class apx =
  fixes apx :: 'a ⇒ 'a ⇒ bool (infix ⊑ 50)

class apx-order = apx +
  assumes apx-reflexive: x ⊑ x
  assumes apx-antisymmetric: x ⊑ y ∧ y ⊑ x ⟶ x = y
  assumes apx-transitive: x ⊑ y ∧ y ⊑ z ⟶ x ⊑ z

sublocale apx-order < apx!: order where less-eq = apx and less = λx y . x ⊑ y ∧ ¬ y ⊑ x
  apply unfold-locales
  apply rule
  apply (rule apx-reflexive)
  apply (metis apx-transitive)
  apply (metis apx-antisymmetric)
  done

context apx-order

begin

abbreviation the-apx-least-fixpoint    :: ('a ⇒ 'a) ⇒ 'a (κ - [201] 200)  where κ  f ≡ apx.the-least-fixpoint f
abbreviation the-apx-least-prefixpoint :: ('a ⇒ 'a) ⇒ 'a (pκ - [201] 200) where pκ f ≡ apx.the-least-prefixpoint f

definition is-apx-meet  :: 'a ⇒ 'a ⇒ 'a ⇒ bool          where is-apx-meet x y z ⟷ z ⊑ x ∧ z ⊑ y ∧ (∀ w . w ⊑ x ∧ w ⊑ y
⟶ w ⊑ z)
definition has-apx-meet :: 'a ⇒ 'a ⇒ bool               where has-apx-meet x y ⟷ (∃ z . is-apx-meet x y z)
definition the-apx-meet :: 'a ⇒ 'a ⇒ 'a (infixl △ 66) where x △ y = (THE z . is-apx-meet x y z)

lemma apx-meet-unique: has-apx-meet x y ⟶ (∃!z . is-apx-meet x y z)
  by (smt apx-antisymmetric has-apx-meet-def is-apx-meet-def)

lemma apx-meet: has-apx-meet x y ⟶ is-apx-meet x y (x △ y)
proof
  assume has-apx-meet x y
  hence is-apx-meet x y (THE z . is-apx-meet x y z)
    by (smt apx-meet-unique theI')
  thus is-apx-meet x y (x △ y)
    by (simp add: is-apx-meet-def the-apx-meet-def)
qed

lemma apx-greatest-lower-bound: has-apx-meet x y ⟶ (w ⊑ x ∧ w ⊑ y ⟷ w ⊑ x △ y)
  by (smt apx-meet apx-transitive is-apx-meet-def)

lemma apx-meet-same: is-apx-meet x y z ⟶ z = x △ y
  by (metis apx-meet apx-meet-unique has-apx-meet-def)

lemma apx-meet-char: is-apx-meet x y z ⟷ has-apx-meet x y ∧ z = x △ y
  by (metis apx-meet-same has-apx-meet-def)

end

class apx-biorder = apx-order + order

begin

lemma mu-below-kappa: has-least-fixpoint f ∧ apx.has-least-fixpoint f ⟶ μ f ≤ κ f
  by (metis apx.least-fixpoint apx.is-least-fixpoint-def is-least-fixpoint-def least-fixpoint)

lemma kappa-below-nu: has-greatest-fixpoint f ∧ apx.has-least-fixpoint f ⟶ κ f ≤ ν f
  by (metis apx.least-fixpoint greatest-fixpoint apx.is-least-fixpoint-def is-greatest-fixpoint-def)
```

lemma *kappa-apx-below-mu*: *has-least-fixpoint* $f \wedge$ *apx.has-least-fixpoint* $f \longrightarrow \kappa\ f \sqsubseteq \mu\ f$
  by (*metis apx.least-fixpoint apx.is-least-fixpoint-def is-least-fixpoint-def least-fixpoint*)

lemma *kappa-apx-below-nu*: *has-greatest-fixpoint* $f \wedge$ *apx.has-least-fixpoint* $f \longrightarrow \kappa\ f \sqsubseteq \nu\ f$
  by (*metis apx.least-fixpoint greatest-fixpoint apx.is-least-fixpoint-def is-greatest-fixpoint-def*)

end

class *apx-semiring* = *apx-biorder* + *idempotent-left-semiring* + L +
  assumes *apx-L-least*: $L \sqsubseteq x$
  assumes *add-apx-left-isotone*: $x \sqsubseteq y \longrightarrow x + z \sqsubseteq y + z$
  assumes *mult-apx-left-isotone*: $x \sqsubseteq y \longrightarrow x\ ;\ z \sqsubseteq y\ ;\ z$
  assumes *mult-apx-right-isotone*: $x \sqsubseteq y \longrightarrow z\ ;\ x \sqsubseteq z\ ;\ y$

begin

lemma *add-apx-right-isotone*: $x \sqsubseteq y \longrightarrow z + x \sqsubseteq z + y$
  by (*metis add-apx-left-isotone add-commutative*)

lemma *add-apx-isotone*: $w \sqsubseteq y \wedge x \sqsubseteq z \longrightarrow w + x \sqsubseteq y + z$
  by (*metis add-apx-left-isotone add-apx-right-isotone apx-transitive*)

lemma *mult-apx-isotone*: $w \sqsubseteq y \wedge x \sqsubseteq z \longrightarrow w\ ;\ x \sqsubseteq y\ ;\ z$
  by (*metis apx-transitive mult-apx-left-isotone mult-apx-right-isotone*)

lemma *affine-apx-isotone*: *apx.isotone* $(\lambda x\ .\ y\ ;\ x + z)$
  by (*smt add-apx-left-isotone apx.isotone-def mult-apx-right-isotone*)

end

end

16 RecursionStrict

theory *RecursionStrict*

imports *NSemiring Approximation*

begin

class *semiring-apx* = *n-semiring* + *apx* +
  assumes *apx-def*: $x \sqsubseteq y \longleftrightarrow x \le y + n(x) ; L \wedge y \le x + n(x) ; T$

begin

lemma *apx-n-order-reverse*: $y \sqsubseteq x \longrightarrow n(x) \le n(y)$
  by (*metis apx-def less-eq-def n-add-left-absorb-mult n-dist-add n-export*)

lemma *apx-n-order*: $x \sqsubseteq y \wedge y \sqsubseteq x \longrightarrow n(x) = n(y)$
  by (*metis apx-n-order-reverse eq-iff*)

lemma *apx-transitive*: $x \sqsubseteq y \wedge y \sqsubseteq z \longrightarrow x \sqsubseteq z$
proof
  assume *1*: $x \sqsubseteq y \wedge y \sqsubseteq z$
  hence $n(y) ; L \le n(x) ; L$
    by (*metis apx-def mult-left-isotone n-add-left-absorb-mult n-export n-dist-add n-isotone*)
  hence *2*: $x \le z + n(x) ; L$ **using** *1*
    by (*smt add-associative add-right-divisibility apx-def less-eq-def*)
  have $z \le x + n(x) ; T + n(x + n(x) ; T) ; T$ **using** *1*
    by (*smt2 add-left-isotone order-refl add-associative add-isotone apx-def mult-left-isotone n-isotone order-trans*)
  also have ... $= x + n(x) ; T$ **using** *1*
    by (*metis add-associative add-idempotent n-add-left-absorb-mult n-export n-dist-add*)
  finally show $x \sqsubseteq z$ **using** *2*
    by (*metis apx-def*)
qed

— Theorem 16.1

subclass *apx-biorder*
  apply *unfold-locales*
  apply (*metis add-left-upper-bound apx-def*)
  apply (*metis antisym add-least-upper-bound apx-def eq-refl less-eq-def n-galois apx-n-order*)
  apply (*rule apx-transitive*)
  done

lemma *add-apx-left-isotone*: $x \sqsubseteq y \longrightarrow x + z \sqsubseteq y + z$
proof
  assume $x \sqsubseteq y$
  hence $x \le y + n(x) ; L \wedge y \le x + n(x) ; T$
    by (*metis apx-def*)
  hence $z + x \le z + y + n(z + x) ; L \wedge z + y \le z + x + n(z + x) ; T$
    by (*metis add-associative add-right-isotone mult-right-sub-dist-add-right n-dist-add order-trans*)
  thus $x + z \sqsubseteq y + z$
    by (*metis apx-def add-commutative*)
qed

lemma *mult-apx-left-isotone*: $x \sqsubseteq y \longrightarrow x ; z \sqsubseteq y ; z$
proof
  assume *1*: $x \sqsubseteq y$
  hence $x \le y + n(x) ; L$
    by (*metis apx-def*)
  hence $x ; z \le y ; z + n(x) ; L$
    by (*metis mult-left-isotone mult-right-dist-add L-left-zero mult-associative*)
  hence *2*: $x ; z \le y ; z + n(x ; z) ; L$
    by (*metis add-right-isotone mult-left-isotone n-mult-left-upper-bound order-trans*)
  have $y ; z \le x ; z + n(x) ; T ; z$ **using** *1*
    by (*metis apx-def mult-left-isotone mult-right-dist-add*)
  hence $y ; z \le x ; z + n(x ; z) ; T$
    by (*metis add-right-isotone mult-associative mult-isotone n-mult-left-upper-bound order-trans top-greatest*)
  thus $x ; z \sqsubseteq y ; z$ **using** *2*
    by (*metis apx-def*)

qed

lemma *mult-apx-right-isotone*: $x \sqsubseteq y \longrightarrow z \; ; \; x \sqsubseteq z \; ; \; y$
proof
  assume *1*: $x \sqsubseteq y$
  hence $x \leq y + n(x) \; ; \; L$
    by (*metis apx-def*)
  hence *2*: $z \; ; \; x \leq z \; ; \; y + n(z \; ; \; x) \; ; \; L$
    by (*smt2 add-associative add-left-upper-bound add-right-zero mult-associative mult-left-dist-add mult-right-isotone n-L-split*)
  have $y \leq x + n(x) \; ; \; T$ **using** *1*
    by (*metis apx-def*)
  hence $z \; ; \; y \leq z \; ; \; x + z \; ; \; n(x) \; ; \; T$
    by (*smt2 mult-associative mult-left-dist-add mult-right-isotone*)
  also have ... $\leq z \; ; \; x + n(z \; ; \; x) \; ; \; T$
    by (*smt add-associative add-least-upper-bound add-left-upper-bound add-right-zero mult-left-dist-add n-L-split n-top-split order-trans*)
  finally show $z \; ; \; x \sqsubseteq z \; ; \; y$ **using** *2*
    by (*metis apx-def*)
qed

— Theorem 16.1 and Theorem 16.2

subclass *apx-semiring*
  apply *unfold-locales*
  apply (*metis add-right-top add-right-upper-bound apx-def mult-left-one n-L top-greatest*)
  apply (*rule add-apx-left-isotone*)
  apply (*rule mult-apx-left-isotone*)
  apply (*rule mult-apx-right-isotone*)
  done

— Theorem 16.2

lemma *ni-apx-isotone*: $x \sqsubseteq y \longrightarrow ni(x) \sqsubseteq ni(y)$
  by (*smt apx-def less-eq-def n-dist-add n-galois n-n-L ni-def*)

— Theorem 17

definition *kappa-apx-meet* :: $('a \Rightarrow 'a) \Rightarrow bool$
  where *kappa-apx-meet f* $\longleftrightarrow$ *apx.has-least-fixpoint f* $\wedge$ *has-apx-meet* $(\mu\ f)\ (\nu\ f) \wedge \kappa\ f = \mu\ f \bigtriangleup \nu\ f$

definition *kappa-mu-nu* :: $('a \Rightarrow 'a) \Rightarrow bool$
  where *kappa-mu-nu f* $\longleftrightarrow$ *apx.has-least-fixpoint f* $\wedge\ \kappa\ f = \mu\ f + n(\nu\ f) \; ; \; L$

definition *nu-below-mu-nu* :: $('a \Rightarrow 'a) \Rightarrow bool$
  where *nu-below-mu-nu f* $\longleftrightarrow \nu\ f \leq \mu\ f + n(\nu\ f) \; ; \; T$

definition *mu-nu-apx-nu* :: $('a \Rightarrow 'a) \Rightarrow bool$
  where *mu-nu-apx-nu f* $\longleftrightarrow \mu\ f + n(\nu\ f) \; ; \; L \sqsubseteq \nu\ f$

definition *mu-nu-apx-meet* :: $('a \Rightarrow 'a) \Rightarrow bool$
  where *mu-nu-apx-meet f* $\longleftrightarrow$ *has-apx-meet* $(\mu\ f)\ (\nu\ f) \wedge \mu\ f \bigtriangleup \nu\ f = \mu\ f + n(\nu\ f) \; ; \; L$

definition *apx-meet-below-nu* :: $('a \Rightarrow 'a) \Rightarrow bool$
  where *apx-meet-below-nu f* $\longleftrightarrow$ *has-apx-meet* $(\mu\ f)\ (\nu\ f) \wedge \mu\ f \bigtriangleup \nu\ f \leq \nu\ f$

lemma *mu-below-l*: $\mu\ f \leq \mu\ f + n(\nu\ f) \; ; \; L$
  by (*metis add-left-upper-bound*)

lemma *l-below-nu*: *has-least-fixpoint f* $\wedge$ *has-greatest-fixpoint f* $\longrightarrow \mu\ f + n(\nu\ f) \; ; \; L \leq \nu\ f$
  by (*metis add-least-upper-bound mu-below-nu n-L-decreasing*)

lemma *n-l-nu*: *has-least-fixpoint f* $\wedge$ *has-greatest-fixpoint f* $\longrightarrow n(\mu\ f + n(\nu\ f) \; ; \; L) = n(\nu\ f)$
  by (*metis less-eq-def mu-below-nu n-dist-add n-n-L*)

lemma *l-apx-mu*: *has-least-fixpoint f* $\wedge$ *has-greatest-fixpoint f* $\longrightarrow \mu\ f + n(\nu\ f) \; ; \; L \sqsubseteq \mu\ f$
  by (*smt add-associative add-left-upper-bound apx-def n-l-nu order-refl*)

— Theorem 17.4 implies Theorem 17.5

lemma *nu-below-mu-nu-mu-nu-apx-nu*: *has-least-fixpoint f* $\wedge$ *has-greatest-fixpoint f* $\wedge$ *nu-below-mu-nu f* $\longrightarrow$ *mu-nu-apx-nu f*
  by (*smt add-associative add-commutative add-left-isotone add-left-top apx-def mu-below-nu mu-nu-apx-nu-def mult-left-dist-add n-l-nu nu-below-mu-nu-def*)

— Theorem 17.5 implies Theorem 17.6

lemma *mu-nu-apx-nu-mu-nu-apx-meet*: *has-least-fixpoint f* $\wedge$ *has-greatest-fixpoint f* $\wedge$ *mu-nu-apx-nu f* $\longrightarrow$ *mu-nu-apx-meet f*
proof
  let $?l = \mu\ f + n(\nu\ f)\ ;\ L$
  assume *has-least-fixpoint f* $\wedge$ *has-greatest-fixpoint f* $\wedge$ *mu-nu-apx-nu f*
  hence *is-apx-meet* $(\mu\ f)\ (\nu\ f)\ ?l$
    by (*smt add-associative add-commutative add-left-upper-bound apx-def is-apx-meet-def l-below-nu mu-nu-apx-nu-def n-l-nu order-refl order-trans*)
  thus *mu-nu-apx-meet f*
    by (*smt apx-meet-char mu-nu-apx-meet-def*)
qed

— Theorem 17.6 implies Theorem 17.7

lemma *mu-nu-apx-meet-apx-meet-below-nu*: *has-least-fixpoint f* $\wedge$ *has-greatest-fixpoint f* $\wedge$ *mu-nu-apx-meet f* $\longrightarrow$ *apx-meet-below-nu f*
  by (*metis apx-meet-below-nu-def l-below-nu mu-nu-apx-meet-def*)

— Theorem 17.7 implies Theorem 17.4

lemma *apx-meet-below-nu-nu-below-mu-nu*: *apx-meet-below-nu f* $\longrightarrow$ *nu-below-mu-nu f*
proof –
  have $\forall\, m\ .\ m \sqsubseteq \mu\ f \wedge m \sqsubseteq \nu\ f \wedge m \le \nu\ f \longrightarrow \nu\ f \le \mu\ f + n(m)\ ;\ T$
    by (*metis add-associative add-left-isotone add-right-top apx-def mult-left-dist-add order-trans*)
  thus *?thesis*
    by (*smt2 add-right-isotone apx-greatest-lower-bound apx-meet-below-nu-def apx-reflexive mult-left-isotone n-isotone nu-below-mu-nu-def order-trans*)
qed

— Theorem 17.1 implies Theorem 17.2

lemma *has-apx-least-fixpoint-kappa-apx-meet*: *has-least-fixpoint f* $\wedge$ *has-greatest-fixpoint f* $\wedge$ *apx.has-least-fixpoint f* $\longrightarrow$ *kappa-apx-meet f*
proof
  assume *1*: *has-least-fixpoint f* $\wedge$ *has-greatest-fixpoint f* $\wedge$ *apx.has-least-fixpoint f*
  hence $\forall\, w\ .\ w \sqsubseteq \mu\ f \wedge w \sqsubseteq \nu\ f \longrightarrow w \sqsubseteq \kappa\ f$
    by (*smt2 add-left-isotone apx-def mu-below-kappa order-trans kappa-below-nu*)
  hence *is-apx-meet* $(\mu\ f)\ (\nu\ f)\ (\kappa\ f)$ **using** *1*
    by (*smt apx-meet-char is-apx-meet-def kappa-apx-below-mu kappa-apx-below-nu kappa-apx-meet-def*)
  thus *kappa-apx-meet f* **using** *1*
    by (*metis apx-meet-char kappa-apx-meet-def*)
qed

— Theorem 17.2 implies Theorem 17.7

lemma *kappa-apx-meet-apx-meet-below-nu*: *has-greatest-fixpoint f* $\wedge$ *kappa-apx-meet f* $\longrightarrow$ *apx-meet-below-nu f*
  by (*metis apx-meet-below-nu-def kappa-apx-meet-def kappa-below-nu*)

— Theorem 17.7 implies Theorem 17.3

lemma *apx-meet-below-nu-kappa-mu-nu*: *has-least-fixpoint f* $\wedge$ *has-greatest-fixpoint f* $\wedge$ *isotone f* $\wedge$ *apx.isotone f* $\wedge$ *apx-meet-below-nu f* $\longrightarrow$ *kappa-mu-nu f*
proof
  let $?l = \mu\ f + n(\nu\ f)\ ;\ L$
  let $?m = \mu\ f \bigtriangleup \nu\ f$
  assume *1*: *has-least-fixpoint f* $\wedge$ *has-greatest-fixpoint f* $\wedge$ *isotone f* $\wedge$ *apx.isotone f* $\wedge$ *apx-meet-below-nu f*
  hence *2*: $?l \sqsubseteq \nu\ f$
    by (*metis apx-meet-below-nu-nu-below-mu-nu mu-nu-apx-nu-def nu-below-mu-nu-mu-nu-apx-nu*)
  hence *3*: $?m = ?l$ **using** *1*
    by (*metis mu-nu-apx-meet-def mu-nu-apx-nu-def mu-nu-apx-nu-mu-nu-apx-meet*)
  have $\mu\ f \le f(?l)$ **using** *1*
    by (*metis add-left-upper-bound is-least-fixpoint-def isotone-def least-fixpoint*)
  hence *4*: $?l \le f(?l) + n(?l)\ ;\ L$ **using** *1*
    by (*metis add-left-isotone n-l-nu*)

have $f(?l) \le f(\nu\ f)$ **using** *1*
  by (*metis l-below-nu isotone-def*)
also have ... $\le\ ?l + n(?l)\ ;\ T$ **using** *1 2*
  by (*metis apx-def nu-unfold*)
finally have *5*: $?l \sqsubseteq f(?l)$ **using** *4*
  by (*metis apx-def*)
have *6*: $f(?l) \sqsubseteq \mu\ f$ **using** *1*
  by (*metis apx.isotone-def is-least-fixpoint-def least-fixpoint l-apx-mu*)
have $f(?l) \sqsubseteq \nu\ f$ **using** *1 2*
  by (*metis apx.isotone-def greatest-fixpoint is-greatest-fixpoint-def*)
hence $f(?l) \sqsubseteq\ ?l$ **using** *1 3 6*
  by (*metis apx-greatest-lower-bound apx-meet-below-nu-def*)
hence $f(?l) =\ ?l$ **using** *5*
  by (*metis apx-antisymmetric*)
thus *kappa-mu-nu f* **using** *1 2 4*
  by (*smt add-left-isotone apx-antisymmetric apx-def apx.least-fixpoint-char greatest-fixpoint apx.is-least-fixpoint-def is-greatest-fixpoint-def is-least-fixpoint-def least-fixpoint n-l-nu order-trans kappa-mu-nu-def*)
qed

— Theorem 17.3 implies Theorem 17.1

lemma *kappa-mu-nu-has-apx-least-fixpoint*: *kappa-mu-nu f* $\longrightarrow$ *apx.has-least-fixpoint f*
  by (*metis kappa-mu-nu-def*)

— Theorem 17.4 implies Theorem 17.3

lemma *nu-below-mu-nu-kappa-mu-nu*: *has-least-fixpoint f* $\wedge$ *has-greatest-fixpoint f* $\wedge$ *isotone f* $\wedge$ *apx.isotone f* $\wedge$ *nu-below-mu-nu f* $\longrightarrow$ *kappa-mu-nu f*
  by (*metis apx-meet-below-nu-kappa-mu-nu mu-nu-apx-meet-apx-meet-below-nu mu-nu-apx-nu-mu-nu-apx-meet nu-below-mu-nu-mu-nu-apx-nu*)

— Theorem 17.3 implies Theorem 17.4

lemma *kappa-mu-nu-nu-below-mu-nu*: *has-least-fixpoint f* $\wedge$ *has-greatest-fixpoint f* $\wedge$ *kappa-mu-nu f* $\longrightarrow$ *nu-below-mu-nu f*
  by (*metis apx-meet-below-nu-nu-below-mu-nu has-apx-least-fixpoint-kappa-apx-meet kappa-apx-meet-apx-meet-below-nu kappa-mu-nu-has-apx-least-fixpoint*)

definition *kappa-mu-nu-ni* :: $('a \Rightarrow 'a) \Rightarrow$ *bool*
  where *kappa-mu-nu-ni f* $\longleftrightarrow$ *apx.has-least-fixpoint f* $\wedge\ \kappa\ f = \mu\ f + ni(\nu\ f)$

lemma *kappa-mu-nu-ni-kappa-mu-nu*: *kappa-mu-nu-ni f* $\longleftrightarrow$ *kappa-mu-nu f*
  by (*metis ni-def kappa-mu-nu-def kappa-mu-nu-ni-def*)

lemma *nu-below-mu-nu-kappa-mu-nu-ni*: *has-least-fixpoint f* $\wedge$ *has-greatest-fixpoint f* $\wedge$ *isotone f* $\wedge$ *apx.isotone f* $\wedge$ *nu-below-mu-nu f* $\longrightarrow$ *kappa-mu-nu-ni f*
  by (*metis nu-below-mu-nu-kappa-mu-nu kappa-mu-nu-ni-kappa-mu-nu*)

lemma *kappa-mu-nu-ni-nu-below-mu-nu*: *has-least-fixpoint f* $\wedge$ *has-greatest-fixpoint f* $\wedge$ *kappa-mu-nu-ni f* $\longrightarrow$ *nu-below-mu-nu f*
  by (*metis kappa-mu-nu-ni-kappa-mu-nu kappa-mu-nu-nu-below-mu-nu*)

end

class *itering-apx* = *n-itering* + *semiring-apx*

begin

— Theorem 16.3

lemma *circ-apx-isotone*: $x \sqsubseteq y \longrightarrow x^\circ \sqsubseteq y^\circ$
proof
  assume $x \sqsubseteq y$
  hence *1*: $x \le y + n(x)\ ;\ L \wedge y \le x + n(x)\ ;\ T$
    by (*metis apx-def*)
  hence $y^\circ \le x^\circ + x^\circ\ ;\ n(x)\ ;\ T$
    by (*metis circ-isotone circ-left-top circ-unfold-sum mult-associative*)
  also have ... $\le x^\circ + n(x^\circ\ ;\ x)\ ;\ T$
    by (*smt2 add-least-upper-bound n-isotone n-top-split order-refl order-trans right-plus-below-circ zero-right-mult-decreasing*)
  also have ... $\le x^\circ + n(x^\circ)\ ;\ T$

by (*metis add-least-upper-bound mult-left-isotone n-isotone order-refl order-trans right-plus-below-circ*)
finally have *2*: $y^\circ \leq x^\circ + n(x^\circ) ; T$
by *metis*
have $x^\circ \leq y^\circ + y^\circ ; n(x) ; L$ **using** *1*
by (*metis L-left-zero circ-isotone circ-unfold-sum mult-associative*)
also have ... $= y^\circ + n(y^\circ ; x) ; L$
by (*metis add-associative add-right-zero mult-associative mult-zero-add-circ-2 n-L-split n-mult-right-zero*)
also have ... $\leq y^\circ + n(x^\circ ; x) ; L + n(x^\circ) ; n(T ; x) ; L$ **using** *2*
by (*metis add-associative add-right-isotone mult-associative mult-left-isotone mult-right-dist-add n-dist-add n-export n-isotone*)
finally have $x^\circ \leq y^\circ + n(x^\circ) ; L$
by (*metis add-associative circ-plus-same n-add-left-absorb-mult n-circ-left-unfold n-dist-add n-export ni-def ni-dist-add*)
thus $x^\circ \sqsubseteq y^\circ$ **using** *2*
by (*metis apx-def*)
qed

end

class *omega-algebra-apx* = *n-omega-algebra-2* + *semiring-apx*

sublocale *omega-algebra-apx* < *star*!: *itering-apx* **where** *circ* = *star* **..**

sublocale *omega-algebra-apx* < *nL-omega*!: *itering-apx* **where** *circ* = *Omega* **..**

context *omega-algebra-apx*

begin

— Theorem 16.4

lemma *omega-apx-isotone*: $x \sqsubseteq y \longrightarrow x^\omega \sqsubseteq y^\omega$
proof
assume $x \sqsubseteq y$
hence *1*: $x \leq y + n(x) ; L \wedge y \leq x + n(x) ; T$
by (*metis apx-def*)
hence $y^\omega \leq x^\star ; n(x) ; T ; (x^\star ; n(x) ; T)^\omega + x^\omega + x^\star ; n(x) ; T ; (x^\star ; n(x) ; T)^\star ; x^\omega$
by (*smt add-associative mult-associative mult-left-one mult-right-dist-add omega-decompose omega-isotone omega-unfold star-left-unfold-equal*)
also have ... $\leq x^\star ; n(x) ; T + x^\omega + x^\star ; n(x) ; T ; (x^\star ; n(x) ; T)^\star ; x^\omega$
by (*smt2 add-commutative add-right-isotone mult-associative mult-right-isotone top-greatest*)
also have ... $= x^\star ; n(x) ; T + x^\omega$
by (*metis add-associative add-commutative add-left-top mult-associative mult-left-dist-add*)
also have ... $\leq n(x^\star ; x) ; T + x^\star ; 0 + x^\omega$
by (*metis add-commutative add-left-isotone n-top-split*)
also have ... $\leq n(x^\star ; x) ; T + x^\omega$
by (*smt2 add-least-upper-bound add-left-isotone mult-right-isotone order-trans star-zero-below-omega top-greatest*)
finally have *2*: $y^\omega \leq x^\omega + n(x^\omega) ; T$
by (*metis add-commutative add-right-isotone mult-left-isotone n-star-below-n-omega n-star-left-unfold order-trans star.circ-plus-same*)
have $x^\omega \leq (y + n(x) ; L)^\omega$ **using** *1*
by (*metis omega-isotone*)
also have ... $= y^\star ; n(x) ; L ; (y^\star ; n(x) ; L)^\omega + y^\omega + y^\star ; n(x) ; L ; (y^\star ; n(x) ; L)^\star ; y^\omega$
by (*smt add-associative mult-associative mult-left-one mult-right-dist-add omega-decompose omega-isotone omega-unfold star-left-unfold-equal*)
also have ... $= y^\star ; n(x) ; L + y^\omega$
by (*metis L-left-zero add-associative add-commutative mult-associative add-idempotent*)
also have ... $\leq y^\omega + y^\star ; 0 + n(y^\star ; x) ; L$
by (*metis add-associative add-commutative eq-refl n-L-split*)
also have ... $\leq y^\omega + n(x^\star ; x) ; L + n(x^\star) ; n(T ; x) ; L$ **using** *1*
by (*metis add-right-isotone add-right-zero apx-def mult-associative mult-left-dist-add mult-left-isotone mult-right-dist-add n-dist-add n-export n-isotone star.circ-apx-isotone star-mult-omega add-associative*)
finally have $x^\omega \leq y^\omega + n(x^\omega) ; L$
by (*metis add-associative add-isotone mult-right-dist-add n-add-left-absorb-mult n-star-left-unfold ni-def ni-star-below-ni-omega order-refl order-trans star.circ-plus-same*)
thus $x^\omega \sqsubseteq y^\omega$ **using** *2*
by (*metis apx-def*)
qed

end

class *omega-algebra-apx-extra* = *omega-algebra-apx* +
assumes *n-split-omega*: $x^\omega \le x^\star \ ; \ 0 + n(x^\omega) \ ; \ T$

begin

lemma *omega-n-star*: $x^\omega + n(x^\star) \ ; \ T \le x^\star \ ; \ n(x^\omega) \ ; \ T$
proof –
have *1*: $n(x^\star) \ ; \ T \le n(x^\omega) \ ; \ T$
by (*smt2 mult-left-isotone n-star-below-n-omega*)
have ... $\le x^\star \ ; \ n(x^\omega) \ ; \ T$
by (*metis add-right-divisibility star-n-omega-top*)
thus *?thesis* **using** *1*
by (*metis add-least-upper-bound n-split-omega order-trans star-n-omega-top*)
qed

lemma *n-omega-zero*: $n(x^\omega) = 0 \longleftrightarrow n(x^\star) = 0 \wedge x^\omega \le x^\star \ ; \ 0$
by (*metis add-right-zero eq-iff mult-left-zero n-mult-zero n-split-omega star-zero-below-omega*)

lemma *n-split-nu-mu*: $y^\omega + y^\star \ ; \ z \le y^\star \ ; \ z + n(y^\omega + y^\star \ ; \ z) \ ; \ T$
proof –
have $y^\omega \le y^\star \ ; \ 0 + n(y^\omega + y^\star \ ; \ z) \ ; \ T$
by (*smt2 add-left-upper-bound add-right-isotone mult-left-isotone n-isotone n-split-omega order-trans*)
also have ... $\le y^\star \ ; \ z + n(y^\omega + y^\star \ ; \ z) \ ; \ T$
by (*metis add-left-isotone mult-right-isotone zero-least*)
finally show *?thesis*
by (*metis add-least-upper-bound add-left-upper-bound*)
qed

lemma *loop-exists*: $\nu \ (\lambda x \ . \ y \ ; \ x + z) \le \mu \ (\lambda x \ . \ y \ ; \ x + z) + n(\nu \ (\lambda x \ . \ y \ ; \ x + z)) \ ; \ T$
by (*metis n-split-nu-mu omega-loop-nu star-loop-mu*)

lemma *loop-isotone*: *isotone* $(\lambda x \ . \ y \ ; \ x + z)$
by (*smt add-commutative add-right-isotone isotone-def mult-right-isotone*)

lemma *loop-apx-isotone*: *apx.isotone* $(\lambda x \ . \ y \ ; \ x + z)$
by (*smt add-apx-left-isotone apx.isotone-def mult-apx-right-isotone*)

lemma *loop-has-least-fixpoint*: *has-least-fixpoint* $(\lambda x \ . \ y \ ; \ x + z)$
by (*metis has-least-fixpoint-def star-loop-is-least-fixpoint*)

lemma *loop-has-greatest-fixpoint*: *has-greatest-fixpoint* $(\lambda x \ . \ y \ ; \ x + z)$
by (*metis has-greatest-fixpoint-def omega-loop-is-greatest-fixpoint*)

lemma *loop-apx-least-fixpoint*: *apx.is-least-fixpoint* $(\lambda x \ . \ y \ ; \ x + z) \ (\mu \ (\lambda x \ . \ y \ ; \ x + z) + n(\nu \ (\lambda x \ . \ y \ ; \ x + z)) \ ; \ L)$
by (*metis apx.least-fixpoint-char loop-apx-isotone loop-exists loop-has-greatest-fixpoint loop-has-least-fixpoint loop-isotone nu-below-mu-nu-def nu-below-mu-nu-kappa-mu-nu kappa-mu-nu-def*)

lemma *loop-has-apx-least-fixpoint*: *apx.has-least-fixpoint* $(\lambda x \ . \ y \ ; \ x + z)$
by (*metis apx.has-least-fixpoint-def loop-apx-least-fixpoint*)

lemma *loop-semantics*: $\kappa \ (\lambda x \ . \ y \ ; \ x + z) = \mu \ (\lambda x \ . \ y \ ; \ x + z) + n(\nu \ (\lambda x \ . \ y \ ; \ x + z)) \ ; \ L$
by (*metis apx.least-fixpoint-char loop-apx-least-fixpoint*)

lemma *loop-apx-least-fixpoint-ni*: *apx.is-least-fixpoint* $(\lambda x \ . \ y \ ; \ x + z) \ (\mu \ (\lambda x \ . \ y \ ; \ x + z) + ni(\nu \ (\lambda x \ . \ y \ ; \ x + z)))$
by (*metis loop-apx-least-fixpoint ni-def*)

lemma *loop-semantics-ni*: $\kappa \ (\lambda x \ . \ y \ ; \ x + z) = \mu \ (\lambda x \ . \ y \ ; \ x + z) + ni(\nu \ (\lambda x \ . \ y \ ; \ x + z))$
by (*metis loop-semantics ni-def*)

— Theorem 18

lemma *loop-semantics-kappa-mu-nu*: $\kappa \ (\lambda x \ . \ y \ ; \ x + z) = n(y^\omega) \ ; \ L + y^\star \ ; \ z$
proof –
have $\kappa \ (\lambda x \ . \ y \ ; \ x + z) = y^\star \ ; \ z + n(y^\omega + y^\star \ ; \ z) \ ; \ L$
by (*metis loop-semantics omega-loop-nu star-loop-mu*)
thus *?thesis*
by (*smt add-associative add-commutative less-eq-def mult-right-dist-add n-L-decreasing n-dist-add*)
qed

end

class *omega-algebra-apx-extra-2* = *omega-algebra-apx* +
  assumes *omega-n-star*: $x^\omega \le x^\star \ ; \ n(x^\omega) \ ; \ T$

begin

subclass *omega-algebra-apx-extra*
  apply *unfold-locales*
  apply (*metis omega-n-star star-n-omega-top*)
  done

end

end

17 NSemiringBoolean

```
theory NSemiringBoolean

imports NSemiring

begin

class an =
  fixes an :: 'a ⇒ 'a

class an-semiring = bounded-idempotent-left-zero-semiring + L + n + an + neg +
  assumes an-complement: an(x) + n(x) = 1
  assumes an-dist-add  : an(x + y) = an(x) ; an(y)
  assumes an-export    : an(an(x) ; y) = n(x) + an(y)
  assumes an-mult-zero : an(x) = an(x ; 0)
  assumes an-L-split   : x ; n(y) ; L = x ; 0 + n(x ; y) ; L
  assumes an-split     : an(x ; L) ; x ≤ x ; 0
  assumes an-uminus    : −x = an(x ; L)

begin

— Theorem 21

lemma n-an-def: n(x) = an(an(x) ; L)
  by (metis add-right-zero an-export an-split antisym mult-left-one mult-right-one zero-least)

— Theorem 21

lemma an-complement-zero: an(x) ; n(x) = 0
  by (smt add-commutative an-dist-add an-split antisym mult-left-zero n-an-def zero-least)

— Theorem 21

lemma an-n-def: an(x) = n(an(x) ; L)
  by (smt add-commutative an-complement an-complement-zero mult-left-dist-add mult-right-dist-add mult-right-one n-an-def)

lemma an-case-split-left: an(z) ; x ≤ y ∧ n(z) ; x ≤ y ⟷ x ≤ y
  by (metis add-least-upper-bound an-complement mult-left-one mult-right-dist-add)

lemma an-case-split-right: x ; an(z) ≤ y ∧ x ; n(z) ≤ y ⟷ x ≤ y
  by (metis add-least-upper-bound an-complement mult-right-one mult-left-dist-add)

lemma split-sub: x ; y ≤ z + x ; T
  by (metis add-right-upper-bound mult-right-isotone order-trans top-greatest)

— Theorem 21

subclass n-semiring
  apply unfold-locales
  apply (metis add-left-zero an-complement-zero an-dist-add n-an-def)
  apply (metis add-left-top an-complement an-dist-add an-export mult-associative n-an-def)
  apply (metis an-dist-add an-export mult-associative n-an-def)
  apply (metis an-dist-add an-export an-n-def mult-right-dist-add n-an-def)
  apply (metis add-idempotent an-dist-add an-mult-zero n-an-def)
  apply (metis an-L-split)
  apply (metis add-left-upper-bound an-case-split-left an-split order-trans split-sub)
  done

lemma n-complement-zero: n(x) ; an(x) = 0
  by (metis an-complement-zero an-n-def n-an-def)

lemma an-zero: an(0) = 1
  by (metis add-right-zero an-complement n-zero)

lemma an-one: an(1) = 1
  by (metis add-right-zero an-complement n-one)

lemma an-L: an(L) = 0
```

by (*metis mult-left-one n-L n-complement-zero*)

lemma *an-top*: $an(T) = 0$
by (*metis mult-left-one n-complement-zero n-top*)

lemma *an-export-n*: $an(n(x) ; y) = an(x) + an(y)$
by (*metis an-export an-n-def n-an-def*)

lemma *n-export-an*: $n(an(x) ; y) = an(x) ; n(y)$
by (*metis an-n-def n-export*)

lemma *n-an-mult-commutative*: $n(x) ; an(y) = an(y) ; n(x)$
by (*metis add-commutative an-dist-add n-an-def*)

lemma *an-mult-commutative*: $an(x) ; an(y) = an(y) ; an(x)$
by (*metis add-commutative an-dist-add*)

lemma *an-mult-idempotent*: $an(x) ; an(x) = an(x)$
by (*metis add-idempotent an-dist-add*)

lemma *an-sub-one*: $an(x) \leq 1$
by (*metis add-left-upper-bound an-complement*)

— Theorem 21

lemma *an-antitone*: $x \leq y \longrightarrow an(y) \leq an(x)$
by (*metis an-dist-add an-sub-one less-eq-def mult-right-isotone mult-right-one*)

lemma *an-mult-left-upper-bound*: $an(x ; y) \leq an(x)$
by (*metis an-antitone an-mult-zero mult-right-isotone zero-least*)

lemma *an-mult-right-zero*: $an(x) ; 0 = 0$
by (*metis an-n-def n-mult-right-zero*)

lemma *n-mult-an*: $n(x ; an(y)) = n(x)$
by (*metis an-n-def n-mult-n*)

lemma *an-mult-n*: $an(x ; n(y)) = an(x)$
by (*metis an-n-def n-an-def n-mult-n*)

lemma *an-mult-an*: $an(x ; an(y)) = an(x)$
by (*metis an-mult-n an-n-def*)

lemma *an-mult-left-absorb-add*: $an(x) ; (an(x) + an(y)) = an(x)$
by (*metis an-n-def n-mult-left-absorb-add*)

lemma *an-mult-right-absorb-add*: $(an(x) + an(y)) ; an(y) = an(y)$
by (*metis add-commutative an-export-n an-mult-commutative an-mult-left-absorb-add*)

lemma *an-add-left-absorb-mult*: $an(x) + an(x) ; an(y) = an(x)$
by (*metis an-n-def n-add-left-absorb-mult*)

lemma *an-add-right-absorb-mult*: $an(x) ; an(y) + an(y) = an(y)$
by (*metis add-commutative an-add-left-absorb-mult an-mult-commutative*)

lemma *an-add-left-dist-mult*: $an(x) + an(y) ; an(z) = (an(x) + an(y)) ; (an(x) + an(z))$
by (*metis an-dist-add an-export-n mult-left-dist-add*)

lemma *an-add-right-dist-mult*: $an(x) ; an(y) + an(z) = (an(x) + an(z)) ; (an(y) + an(z))$
by (*metis add-commutative an-add-left-dist-mult*)

lemma *an-n-order*: $an(x) \leq an(y) \longleftrightarrow n(y) \leq n(x)$
by (*smt add-commutative an-dist-add an-mult-left-absorb-add an-n-def less-eq-def n-an-def n-dist-add*)

lemma *an-order*: $an(x) \leq an(y) \longleftrightarrow an(x) ; an(y) = an(x)$
by (*metis an-add-right-absorb-mult an-mult-left-absorb-add less-eq-def*)

lemma *an-mult-left-lower-bound*: $an(x) ; an(y) \leq an(x)$
by (*metis add-left-upper-bound an-antitone an-dist-add*)

lemma *an-mult-right-lower-bound*: $an(x) ; an(y) \le an(y)$
by (*metis an-add-right-absorb-mult less-eq-def*)

lemma *an-n-mult-left-lower-bound*: $an(x) ; n(y) \le an(x)$
by (*metis an-mult-left-lower-bound n-an-def*)

lemma *an-n-mult-right-lower-bound*: $an(x) ; n(y) \le n(y)$
by (*metis an-mult-right-lower-bound n-an-def*)

lemma *n-an-mult-left-lower-bound*: $n(x) ; an(y) \le n(x)$
by (*metis an-mult-left-lower-bound n-an-def*)

lemma *n-an-mult-right-lower-bound*: $n(x) ; an(y) \le an(y)$
by (*metis an-mult-right-lower-bound n-an-def*)

lemma *an-mult-least-upper-bound*: $an(x) \le an(y) \wedge an(x) \le an(z) \longleftrightarrow an(x) \le an(y) ; an(z)$
by (*smt an-dist-add an-mult-left-lower-bound an-order mult-associative*)

lemma *an-mult-left-divisibility*: $an(x) \le an(y) \longleftrightarrow (\exists z . an(x) = an(y) ; an(z))$
by (*metis an-mult-commutative an-mult-left-lower-bound an-order*)

lemma *an-mult-right-divisibility*: $an(x) \le an(y) \longleftrightarrow (\exists z . an(x) = an(z) ; an(y))$
by (*metis an-mult-commutative an-mult-left-divisibility*)

lemma *an-split-top*: $an(x ; L) ; x ; T \le x ; 0$
by (*metis an-split mult-associative mult-left-isotone mult-left-zero*)

lemma *an-n-L*: $an(n(x) ; L) = an(x)$
by (*metis an-n-def n-an-def*)

lemma *an-galois*: $an(y) \le an(x) \longleftrightarrow n(x) ; L \le y$
by (*metis an-n-order n-galois*)

lemma *an-mult*: $an(x ; n(y) ; L) = an(x ; y)$
by (*metis an-n-L n-mult*)

lemma *n-mult-top*: $an(x ; n(y) ; T) = an(x ; y)$
by (*metis an-n-L n-mult-top*)

lemma *an-n-equal*: $an(x) = an(y) \longleftrightarrow n(x) = n(y)$
by (*metis an-n-L n-an-def*)

lemma *an-top-L*: $an(x ; T) = an(x ; L)$
by (*metis an-n-equal n-top-L*)

lemma *an-case-split-left-equal*: $an(z) ; x = an(z) ; y \wedge n(z) ; x = n(z) ; y \longrightarrow x = y$
by (*metis an-complement case-split-left-equal*)

lemma *an-case-split-right-equal*: $x ; an(z) = y ; an(z) \wedge x ; n(z) = y ; n(z) \longrightarrow x = y$
by (*metis an-complement case-split-right-equal*)

lemma *an-equal-complement*: $n(x) + an(y) = 1 \wedge n(x) ; an(y) = 0 \longleftrightarrow an(x) = an(y)$
by (*metis add-commutative an-complement an-dist-add mult-left-one mult-right-dist-add n-complement-zero*)

lemma *n-equal-complement*: $n(x) + an(y) = 1 \wedge n(x) ; an(y) = 0 \longleftrightarrow n(x) = n(y)$
by (*metis an-equal-complement n-an-def*)

lemma *an-shunting*: $an(z) ; x \le y \longleftrightarrow x \le y + n(z) ; T$
apply (*rule iffI*)
apply (*metis an-case-split-left add-left-upper-bound dual-order.trans split-sub*)
apply (*metis add-right-zero an-case-split-left an-complement-zero mult-associative mult-left-dist-add mult-left-zero mult-right-isotone order-refl order-trans*)
done

lemma *an-shunting-an*: $an(z) ; an(x) \le an(y) \longleftrightarrow an(x) \le n(z) + an(y)$
apply (*rule iffI*)
apply (*smt2 add-left-upper-bound add-right-upper-bound an-case-split-left n-an-mult-left-lower-bound order-trans*)
apply (*metis add-left-zero add-right-upper-bound an-case-split-left an-complement-zero mult-left-dist-add mult-right-isotone order-trans*)

done

lemma *an-L-zero*: $an(x ; L) ; x = an(x ; L) ; x ; 0$
by (*smt2 an-order an-split antisym mult-associative mult-right-isotone order-refl zero-right-mult-decreasing*)

lemma *n-plus-complement-intro-n*: $n(x) + an(x) ; n(y) = n(x) + n(y)$
by (*metis add-commutative an-complement an-n-def mult-right-one n-add-right-dist-mult n-an-mult-commutative*)

lemma *n-plus-complement-intro-an*: $n(x) + an(x) ; an(y) = n(x) + an(y)$
by (*metis an-n-def n-plus-complement-intro-n*)

lemma *an-plus-complement-intro-n*: $an(x) + n(x) ; n(y) = an(x) + n(y)$
by (*metis an-n-def n-an-def n-plus-complement-intro-n*)

lemma *an-plus-complement-intro-an*: $an(x) + n(x) ; an(y) = an(x) + an(y)$
by (*metis an-n-def an-plus-complement-intro-n*)

lemma *n-mult-complement-intro-n*: $n(x) ; (an(x) + n(y)) = n(x) ; n(y)$
by (*metis add-left-zero mult-left-dist-add n-complement-zero*)

lemma *n-mult-complement-intro-an*: $n(x) ; (an(x) + an(y)) = n(x) ; an(y)$
by (*metis add-left-zero mult-left-dist-add n-complement-zero*)

lemma *an-mult-complement-intro-n*: $an(x) ; (n(x) + n(y)) = an(x) ; n(y)$
by (*metis add-left-zero an-complement-zero mult-left-dist-add*)

lemma *an-mult-complement-intro-an*: $an(x) ; (n(x) + an(y)) = an(x) ; an(y)$
by (*metis add-left-zero an-complement-zero mult-left-dist-add*)

lemma *an-preserves-equation*: $an(y) ; x \le x ; an(y) \longleftrightarrow an(y) ; x = an(y) ; x ; an(y)$
by (*metis an-n-def n-preserves-equation*)

lemma *wnf-lemma-1*: $(n(p;L) ; n(q;L) + an(p;L) ; an(r;L)) ; n(p;L) = n(p;L) ; n(q;L)$
by (*smt add-commutative an-n-def n-add-left-absorb-mult n-add-right-dist-mult n-export n-mult-commutative n-mult-complement-intro-n*)

lemma *wnf-lemma-2*: $(n(p;L) ; n(q;L) + an(r;L) ; an(q;L)) ; n(q;L) = n(p;L) ; n(q;L)$
by (*metis an-mult-commutative n-mult-commutative wnf-lemma-1*)

lemma *wnf-lemma-3*: $(n(p;L) ; n(r;L) + an(p;L) ; an(q;L)) ; an(p;L) = an(p;L) ; an(q;L)$
by (*smt an-add-right-dist-mult an-n-def n-add-left-absorb-mult n-an-def n-an-mult-commutative n-export n-mult-complement-intro-n n-plus-complement-intro-an*)

lemma *wnf-lemma-4*: $(n(r;L) ; n(q;L) + an(p;L) ; an(q;L)) ; an(q;L) = an(p;L) ; an(q;L)$
by (*metis an-mult-commutative n-mult-commutative wnf-lemma-3*)

lemma *wnf-lemma-5*: $n(p+q) ; (n(q) ; x + an(q) ; y) = n(q) ; x + an(q) ; n(p) ; y$
by (*smt add-right-zero mult-associative mult-left-dist-add n-an-mult-commutative n-complement-zero n-dist-add n-mult-right-absorb-add*)

definition *ani* :: $'a \Rightarrow 'a$
where *ani* $x = an(x) ; L$

lemma *ani-zero*: $ani(0) = L$
by (*metis an-zero ani-def mult-left-one*)

lemma *ani-one*: $ani(1) = L$
by (*metis an-one ani-def mult-left-one*)

lemma *ani-L*: $ani(L) = 0$
by (*metis an-L ani-def mult-left-zero*)

lemma *ani-top*: $ani(T) = 0$
by (*metis an-top ani-def mult-left-zero*)

lemma *ani-complement*: $ani(x) + ni(x) = L$
by (*metis an-complement ani-def mult-right-dist-add n-top ni-def ni-top*)

lemma *ani-mult-zero*: $ani(x) = ani(x ; 0)$

by (*metis an-mult-zero ani-def*)

lemma *ani-antitone*: $y \leq x \longrightarrow ani(x) \leq ani(y)$
by (*metis an-antitone ani-def mult-left-isotone*)

lemma *ani-mult-left-upper-bound*: $ani(x ; y) \leq ani(x)$
by (*metis an-mult-left-upper-bound ani-def mult-left-isotone*)

lemma *ani-involutive*: $ani(ani(x)) = ni(x)$
by (*metis ani-def n-an-def ni-def*)

lemma *ani-below-L*: $ani(x) \leq L$
by (*metis add-left-upper-bound ani-complement*)

lemma *ani-left-zero*: $ani(x) ; y = ani(x)$
by (*metis L-left-zero ani-def mult-associative*)

lemma *ani-top-L*: $ani(x ; T) = ani(x ; L)$
by (*metis an-top-L ani-def*)

lemma *ani-ni-order*: $ani(x) \leq ani(y) \longleftrightarrow ni(y) \leq ni(x)$
by (*metis an-galois an-n-L an-n-def ani-def mult-left-isotone n-isotone ni-def*)

lemma *ani-ni-equal*: $ani(x) = ani(y) \longleftrightarrow ni(x) = ni(y)$
by (*metis ani-ni-order antisym order-refl*)

lemma *ni-ani*: $ni(ani(x)) = ani(x)$
by (*metis an-n-def ani-def ni-def*)

lemma *ani-ni*: $ani(ni(x)) = ani(x)$
by (*metis ani-ni-equal ni-idempotent*)

lemma *ani-mult*: $ani(x ; ni(y)) = ani(x ; y)$
by (*metis ani-ni-equal ni-mult*)

lemma *ani-an-order*: $ani(x) \leq ani(y) \longleftrightarrow an(x) \leq an(y)$
by (*metis an-n-order ani-ni-order ni-n-order*)

lemma *ani-an-equal*: $ani(x) = ani(y) \longleftrightarrow an(x) = an(y)$
by (*metis an-n-def ani-def*)

lemma *n-mult-ani*: $n(x) ; ani(x) = 0$
by (*smt an-complement an-export-n an-zero ani-def ani-ni-equal n-an-def ni-ani ni-export ni-zero*)

lemma *an-mult-ni*: $an(x) ; ni(x) = 0$
by (*metis an-n-def ani-def n-an-def n-mult-ani ni-def*)

lemma *n-mult-ni*: $n(x) ; ni(x) = ni(x)$
by (*metis n-export n-order ni-def ni-export order-refl*)

lemma *an-mult-ani*: $an(x) ; ani(x) = ani(x)$
by (*metis an-n-def ani-def n-mult-ni ni-def*)

lemma *ani-ni-meet*: $x \leq ani(y) \wedge x \leq ni(y) \longrightarrow x = 0$
by (*metis an-case-split-left an-mult-ni antisym less-eq-def mult-left-sub-dist-add-left n-mult-ani zero-least*)

lemma *ani-galois*: $ani(x) \leq y \longleftrightarrow ni(x + y) = L$
by (*metis add-left-zero add-commutative an-L an-complement an-dist-add an-n-def an-shunting-an ani-def less-eq-def mult-left-one n-an-def ni-def ni-n-galois*)

lemma *an-ani*: $an(ani(x)) = n(x)$
by (*metis ani-def n-an-def*)

lemma *n-ani*: $n(ani(x)) = an(x)$
by (*metis an-n-def ani-def*)

lemma *an-ni*: $an(ni(x)) = an(x)$
by (*metis an-n-L ni-def*)

lemma *ani-an*: $ani(an(x)) = L$
  by (*metis an-mult-right-zero an-mult-zero an-zero ani-def mult-left-one*)

lemma *ani-n*: $ani(n(x)) = L$
  by (*metis an-ani ani-an*)

lemma *ni-an*: $ni(an(x)) = 0$
  by (*metis n-ani ni-n*)

lemma *ani-mult-n*: $ani(x ; n(y)) = ani(x)$
  by (*metis an-mult-n ani-an-equal*)

lemma *ani-mult-an*: $ani(x ; an(y)) = ani(x)$
  by (*metis an-mult-an ani-def*)

lemma *ani-export-n*: $ani(n(x) ; y) = ani(x) + ani(y)$
  by (*metis an-export-n ani-def mult-right-dist-add*)

lemma *ani-export-an*: $ani(an(x) ; y) = ni(x) + ani(y)$
  by (*metis an-export ani-def mult-right-dist-add ni-def*)

lemma *ni-export-an*: $ni(an(x) ; y) = an(x) ; ni(y)$
  by (*metis an-n-def ni-export*)

lemma *ani-mult-top*: $ani(x ; n(y) ; T) = ani(x ; y)$
  by (*metis ani-ni-equal ni-mult-top*)

lemma *ani-an-zero*: $ani(x) = 0 \longleftrightarrow an(x) = 0$
  by (*metis ani-def mult-left-zero n-ani n-zero*)

lemma *ani-an-L*: $ani(x) = L \longleftrightarrow an(x) = 1$
  by (*metis ani-def mult-left-one n-L n-ani*)

— Theorem 21

subclass *tests*
  apply *unfold-locales*
  apply (*metis mult-associative*)
  apply (*metis an-mult-commutative an-uminus*)
  apply (*smt an-add-left-dist-mult an-export-n an-n-L an-uminus n-an-def n-complement-zero n-export*)
  apply (*metis an-dist-add an-n-def an-uminus n-an-def*)
  apply (*rule the-equality*[*THEN sym*])
  apply (*metis an-complement-zero an-uminus n-an-def*)
  apply (*metis an-L an-uminus mult-left-one mult-left-zero*)
  apply (*metis an-uminus an-zero mult-left-zero*)
  apply (*metis an-export-n an-n-L an-uminus n-an-def n-export*)
  apply (*metis an-order an-uminus*)
  apply (*metis less-def*)
  done

end

class *an-itering* = *n-itering* + *an-semiring* + *while* +
  assumes *while-circ-def*: $p \star y = (p ; y)^\circ ; -p$

begin

subclass *test-itering*
  apply *unfold-locales*
  apply (*rule while-circ-def*)
  done

lemma *an-circ-left-unfold*: $an(x^\circ) = an(x ; x^\circ)$
  by (*metis an-dist-add an-one circ-left-unfold mult-left-one*)

lemma *an-circ-x-n-circ*: $an(x^\circ) ; x ; n(x^\circ) \le x ; 0$
  by (*metis an-circ-left-unfold an-mult an-split mult-associative n-mult-right-zero*)

lemma *an-circ-invariant*: $an(x^\circ) ; x \le x ; an(x^\circ)$

proof −
  have *1*: $an(x^\circ)$; x ; $an(x^\circ) \le x$; $an(x^\circ)$
    by (*metis an-case-split-left mult-associative order-refl*)
  have $an(x^\circ)$; x ; $n(x^\circ) \le x$; $an(x^\circ)$
    by (*metis an-circ-x-n-circ order-trans mult-right-isotone zero-least*)
  thus *?thesis* **using** *1*
    by (*metis an-case-split-right*)
qed

lemma *ani-circ*: $ani(x)^\circ = 1 + ani(x)$
  by (*metis ani-left-zero circ-plus-same circ-right-unfold*)

lemma *ani-circ-left-unfold*: $ani(x^\circ) = ani(x$; $x^\circ)$
  by (*metis an-circ-left-unfold ani-def*)

lemma *an-circ-import*: $an(y)$; $x \le x$; $an(y) \longrightarrow an(y)$; $x^\circ = an(y)$; $(an(y)$; $x)^\circ$
  by (*metis an-n-def n-circ-import*)

lemma *preserves-L*: *preserves L* $(-p)$
  by (*metis L-left-zero mult-associative preserves-equation-test*)

end

class *an-omega-algebra* = *n-omega-algebra-2* + *an-semiring* + *while* +
  assumes *while-Omega-def*: $p \star y = (p$; $y)^\Omega$; $-p$

begin

lemma *an-split-omega-omega*: $an(x^\omega)$; $x^\omega \le x^\omega$; 0
  by (*metis an-split an-top-L omega-vector*)

lemma *an-omega-below-an-star*: $an(x^\omega) \le an(x^\star)$
  by (*metis an-n-order n-star-below-n-omega*)

lemma *an-omega-below-an*: $an(x^\omega) \le an(x)$
  by (*metis an-n-order n-below-n-omega*)

lemma *an-omega-induct*: $an(x$; $y + z) \le an(y) \longrightarrow an(x^\omega + x^\star$; $z) \le an(y)$
  by (*metis an-n-order n-omega-induct*)

lemma *an-star-mult*: $an(y) \le an(x$; $y) \longrightarrow an(x^\star$; $y) = an(x^\star)$; $an(y)$
  by (*smt an-dist-add an-export-n an-n-equal less-eq-def n-dist-add n-export n-mult-left-lower-bound n-star-mult*)

lemma *an-omega-mult*: $an(x^\omega$; $y) = an(x^\omega)$
  by (*metis an-n-equal n-omega-mult*)

lemma *an-star-left-unfold*: $an(x^\star) = an(x$; $x^\star)$
  by (*metis an-n-equal n-star-left-unfold*)

lemma *an-star-x-n-star*: $an(x^\star)$; x ; $n(x^\star) \le x$; 0
  by (*metis add-right-zero an-case-split-left an-complement-zero mult-associative mult-left-zero n-export-an n-mult n-mult-right-zero n-split n-star-left-unfold order-refl order-trans*)

lemma *an-star-invariant*: $an(x^\star)$; $x \le x$; $an(x^\star)$
proof −
  have *1*: $an(x^\star)$; x ; $an(x^\star) \le x$; $an(x^\star)$
    by (*metis an-case-split-left mult-associative order-refl*)
  have $an(x^\star)$; x ; $n(x^\star) \le x$; $an(x^\star)$
    by (*metis an-star-x-n-star order-trans mult-right-isotone zero-least*)
  thus *?thesis* **using** *1*
    by (*metis an-case-split-right*)
qed

lemma *n-an-star-unfold-invariant*: $n(an(x^\star)$; $x^\omega) \le an(x)$; $n(x$; $an(x^\star)$; $x^\omega)$
proof −
  have $n(an(x^\star)$; $x^\omega) \le an(x)$
    by (*metis an-antitone an-n-mult-left-lower-bound n-export-an order-trans star.circ-increasing*)
  thus *?thesis*
    by (*smt an-star-invariant less-eq-def mult-associative mult-right-dist-add n-isotone n-order omega-unfold*)

qed

lemma *ani-omega-below-ani-star*: $ani(x^\omega) \leq ani(x^\star)$
  by (*metis an-omega-below-an-star ani-an-order*)

lemma *ani-omega-below-ani*: $ani(x^\omega) \leq ani(x)$
  by (*metis an-omega-below-an ani-an-order*)

lemma *ani-star*: $ani(x)^\star = 1 + ani(x)$
  by (*metis mult-L-star ani-def*)

lemma *ani-omega*: $ani(x)^\omega = ani(x) \ ; \ L$
  by (*metis mult-L-omega ani-def ani-left-zero*)

lemma *ani-omega-induct*: $ani(x \ ; \ y + z) \leq ani(y) \longrightarrow ani(x^\omega + x^\star \ ; \ z) \leq ani(y)$
  by (*metis an-omega-induct ani-an-order*)

lemma *ani-omega-mult*: $ani(x^\omega \ ; \ y) = ani(x^\omega)$
  by (*metis an-omega-mult ani-def*)

lemma *ani-star-left-unfold*: $ani(x^\star) = ani(x \ ; \ x^\star)$
  by (*metis an-star-left-unfold ani-def*)

lemma *an-star-import*: $an(y) \ ; \ x \leq x \ ; \ an(y) \longrightarrow an(y) \ ; \ x^\star = an(y) \ ; \ (an(y) \ ; \ x)^\star$
  by (*metis an-n-def n-star-import*)

lemma *an-omega-export*: $an(y) \ ; \ x \leq x \ ; \ an(y) \longrightarrow an(y) \ ; \ x^\omega = (an(y) \ ; \ x)^\omega$
  by (*metis an-n-def n-omega-export*)

lemma *an-omega-import*: $an(y) \ ; \ x \leq x \ ; \ an(y) \longrightarrow an(y) \ ; \ x^\omega = an(y) \ ; \ (an(y) \ ; \ x)^\omega$
  by (*metis an-n-def n-omega-import*)

end

— Theorem 22

sublocale *an-omega-algebra* < *nL-omega*!: *an-itering* **where** *circ* = *Omega*
  apply *unfold-locales*
  apply (*rule while-Omega-def*)
  done

context *an-omega-algebra*

begin

lemma *preserves-star*: *nL-omega.preserves* x $(-p) \longrightarrow$ *nL-omega.preserves* $(x^\star)$ $(-p)$
  by (*metis nL-omega.preserves-def star-simulation-right*)

end

end

18 NModal

theory *NModal*

imports *NSemiringBoolean*

begin

class *n-diamond-semiring* = *n-semiring* + *diamond* +
  assumes *ndiamond-def*: $|x{>}y = n(x ; y ; L)$

begin

lemma *diamond-x-0*: $|x{>}0 = n(x)$
  by (*metis n-mult n-mult-zero n-zero ndiamond-def*)

lemma *diamond-x-1*: $|x{>}1 = n(x ; L)$
  by (*metis n-L n-mult ndiamond-def*)

lemma *diamond-x-L*: $|x{>}L = n(x ; L)$
  by (*metis L-left-zero mult-associative ndiamond-def*)

lemma *diamond-x-top*: $|x{>}T = n(x ; L)$
  by (*metis mult-associative n-top-L ndiamond-def top-mult-top*)

lemma *diamond-x-n*: $|x{>}n(y) = n(x ; y)$
  by (*metis n-mult ndiamond-def*)

lemma *diamond-0-y*: $|0{>}y = 0$
  by (*metis mult-left-zero n-n-L n-one ndiamond-def*)

lemma *diamond-1-y*: $|1{>}y = n(y ; L)$
  by (*metis mult-left-one ndiamond-def*)

lemma *diamond-1-n*: $|1{>}n(y) = n(y)$
  by (*metis diamond-1-y n-n-L*)

lemma *diamond-L-y*: $|L{>}y = 1$
  by (*metis L-left-zero n-L ndiamond-def*)

lemma *diamond-top-y*: $|T{>}y = 1$
  by (*metis add-left-top add-right-top diamond-L-y mult-right-dist-add n-dist-add n-top ndiamond-def*)

lemma *diamond-n-y*: $|n(x){>}y = n(x) ; n(y ; L)$
  by (*metis mult-associative n-export ndiamond-def*)

lemma *diamond-n-0*: $|n(x){>}0 = 0$
  by (*metis diamond-x-n n-mult-right-zero n-zero*)

lemma *diamond-n-1*: $|n(x){>}1 = n(x)$
  by (*metis diamond-x-1 n-n-L*)

lemma *diamond-n-n*: $|n(x){>}n(y) = n(x) ; n(y)$
  by (*metis diamond-x-n n-export*)

lemma *diamond-n-n-same*: $|n(x){>}n(x) = n(x)$
  by (*metis diamond-n-n n-mult-idempotent*)

— Theorem 23.1

lemma *diamond-left-dist-add*: $|x + y{>}z = |x{>}z + |y{>}z$
  by (*metis mult-right-dist-add n-dist-add ndiamond-def*)

— Theorem 23.2

lemma *diamond-right-dist-add*: $|x{>}(y + z) = |x{>}y + |x{>}z$
  by (*metis mult-left-dist-add mult-right-dist-add n-dist-add ndiamond-def*)

— Theorem 23.3

lemma *diamond-associative*: $|x ; y{>}z = |x{>}(y ; z)$
  by (*metis mult-associative ndiamond-def*)

— Theorem 23.3

lemma *diamond-left-mult*: $|x ; y{>}z = |x{>}|y{>}z$
  by (*metis diamond-x-n mult-associative ndiamond-def*)

lemma *diamond-right-mult*: $|x{>}(y ; z) = |x{>}|y{>}z$
  by (*metis diamond-associative diamond-left-mult*)

lemma *diamond-n-export*: $|n(x) ; y{>}z = n(x) ; |y{>}z$
  by (*metis diamond-associative diamond-n-y ndiamond-def*)

lemma *diamond-diamond-export*: $||x{>}y{>}z = |x{>}y ; |z{>}1$
  by (*metis diamond-n-y diamond-x-1 ndiamond-def*)

lemma *diamond-left-isotone*: $x \le y \longrightarrow |x{>}z \le |y{>}z$
  by (*metis diamond-left-dist-add less-eq-def*)

lemma *diamond-right-isotone*: $y \le z \longrightarrow |x{>}y \le |x{>}z$
  by (*metis diamond-right-dist-add less-eq-def*)

lemma *diamond-isotone*: $w \le y \wedge x \le z \longrightarrow |w{>}x \le |y{>}z$
  by (*metis diamond-left-isotone diamond-right-isotone order-trans*)

definition *ndiamond-L* :: $'a \Rightarrow 'a \Rightarrow 'a$ (‖ - » - [*50*,*90*] *95*)
  where $\|x»y = n(x ; y) ; L$

lemma *ndiamond-to-L*: $\|x»y = |x{>}n(y) ; L$
  by (*metis diamond-x-n ndiamond-L-def*)

lemma *ndiamond-from-L*: $|x{>}y = n(\|x»(y ; L))$
  by (*metis mult-associative n-n-L ndiamond-L-def ndiamond-def*)

lemma *diamond-L-ni*: $\|x»y = ni(x ; y)$
  by (*metis ndiamond-L-def ni-def*)

lemma *diamond-L-associative*: $\|x ; y»z = \|x»(y ; z)$
  by (*metis mult-associative diamond-L-ni*)

lemma *diamond-L-left-mult*: $\|x ; y»z = \|x»\|y»z$
  by (*metis diamond-L-associative diamond-L-ni ni-mult*)

lemma *diamond-L-right-mult*: $\|x»(y ; z) = \|x»\|y»z$
  by (*metis diamond-L-associative diamond-L-left-mult*)

lemma *diamond-L-left-dist-add*: $\|x + y»z = \|x»z + \|y»z$
  by (*metis mult-right-dist-add diamond-L-ni ni-dist-add*)

lemma *diamond-L-x-ni*: $\|x»ni(y) = ni(x ; y)$
  by (*metis diamond-L-ni ni-mult*)

lemma *diamond-L-left-isotone*: $x \le y \longrightarrow \|x»z \le \|y»z$
  by (*metis diamond-L-left-dist-add less-eq-def*)

lemma *diamond-L-right-isotone*: $y \le z \longrightarrow \|x»y \le \|x»z$
  by (*metis mult-right-isotone ndiamond-L-def ni-def ni-isotone*)

lemma *diamond-L-isotone*: $w \le y \wedge x \le z \longrightarrow \|w»x \le \|y»z$
  by (*metis mult-isotone ndiamond-L-def ni-def ni-isotone*)

end

class *n-box-semiring* = *n-diamond-semiring* + *an-semiring* + *box* +
  assumes *nbox-def*: $|x]y = an(x ; an(y ; L) ; L)$

begin

— Theorem 23.8

lemma *box-diamond*: $|x]y = an(|x{>}an(y ; L) ; L)$
  by (*metis an-n-L nbox-def ndiamond-def*)

— Theorem 23.4

lemma *diamond-box*: $|x{>}y = an(|x]an(y ; L) ; L)$
  by (*metis diamond-associative diamond-right-mult diamond-x-n n-an-def nbox-def ndiamond-def*)

lemma *box-x-0*: $|x]0 = an(x ; L)$
  by (*metis an-L mult-right-one n-L n-an-def nbox-def*)

lemma *box-x-1*: $|x]1 = an(x)$
  by (*metis an-L an-n-L box-diamond diamond-x-0 n-L*)

lemma *box-x-L*: $|x]L = an(x)$
  by (*metis L-left-zero an-L an-n-L box-diamond diamond-x-0*)

lemma *box-x-top*: $|x]T = an(x)$
  by (*metis an-n-L an-top an-top-L box-diamond diamond-x-n n-mult-zero n-zero top-mult-top*)

lemma *box-x-n*: $|x]n(y) = an(x ; an(y) ; L)$
  by (*metis an-n-L nbox-def*)

lemma *box-x-an*: $|x]an(y) = an(x ; y)$
  by (*metis an-n-L box-diamond diamond-x-n n-an-def*)

lemma *box-0-y*: $|0]y = 1$
  by (*metis an-zero box-diamond diamond-0-y mult-left-zero*)

lemma *box-1-y*: $|1]y = n(y ; L)$
  by (*metis mult-left-one n-an-def nbox-def*)

lemma *box-1-n*: $|1]n(y) = n(y)$
  by (*metis box-1-y n-n-L*)

lemma *box-1-an*: $|1]an(y) = an(y)$
  by (*metis an-n-def box-1-y*)

lemma *box-L-y*: $|L]y = 0$
  by (*metis an-L box-1-an box-diamond box-x-an diamond-L-y*)

lemma *box-top-y*: $|T]y = 0$
  by (*metis an-L box-1-an box-diamond box-x-an diamond-top-y*)

lemma *box-n-y*: $|n(x)]y = an(x) + n(y ; L)$
  by (*metis an-export-n mult-associative n-an-def nbox-def*)

lemma *box-an-y*: $|an(x)]y = n(x) + n(y ; L)$
  by (*metis an-n-def box-n-y n-an-def*)

lemma *box-n-0*: $|n(x)]0 = an(x)$
  by (*metis an-n-L box-x-0*)

lemma *box-an-0*: $|an(x)]0 = n(x)$
  by (*metis box-x-0 n-an-def*)

lemma *box-n-1*: $|n(x)]1 = 1$
  by (*metis an-zero box-x-an n-mult-right-zero*)

lemma *box-an-1*: $|an(x)]1 = 1$
  by (*metis an-mult-right-zero an-zero box-x-an*)

lemma *box-n-n*: $|n(x)]n(y) = an(x) + n(y)$
  by (*metis box-n-y n-n-L*)

lemma *box-an-n*: $|an(x)]n(y) = n(x) + n(y)$
  by (*metis an-n-def box-n-n n-an-def*)

lemma *box-n-an*: $|n(x)]an(y) = an(x) + an(y)$
  by (*metis an-export-n box-x-an*)

lemma *box-an-an*: $|an(x)]an(y) = n(x) + an(y)$
  by (*metis an-export box-x-an*)

lemma *box-n-n-same*: $|n(x)]n(x) = 1$
  by (*metis an-complement box-n-n*)

lemma *box-an-an-same*: $|an(x)]an(x) = 1$
  by (*metis an-equal-complement box-an-an*)

— Theorem 23.5

lemma *box-left-dist-add*: $|x + y]z = |x]z ; |y]z$
  by (*metis an-dist-add mult-right-dist-add nbox-def*)

lemma *box-right-dist-add*: $|x](y + z) = an(x ; an(y ; L) ; an(z ; L) ; L)$
  by (*metis an-dist-add mult-associative mult-right-dist-add nbox-def*)

lemma *box-associative*: $|x ; y]z = an(x ; y ; an(z ; L) ; L)$
  by (*metis nbox-def*)

— Theorem 23.7

lemma *box-left-mult*: $|x ; y]z = |x]|y]z$
  by (*metis box-x-an mult-associative nbox-def*)

lemma *box-right-mult*: $|x](y ; z) = an(x ; an(y ; z ; L) ; L)$
  by (*metis nbox-def*)

— Theorem 23.6

lemma *box-right-mult-n-n*: $|x](n(y) ; n(z)) = |x]n(y) ; |x]n(z)$
  by (*smt an-dist-add an-export-n an-n-L mult-associative mult-left-dist-add mult-right-dist-add nbox-def*)

lemma *box-right-mult-an-n*: $|x](an(y) ; n(z)) = |x]an(y) ; |x]n(z)$
  by (*metis an-n-def box-right-mult-n-n*)

lemma *box-right-mult-n-an*: $|x](n(y) ; an(z)) = |x]n(y) ; |x]an(z)$
  by (*metis an-n-def box-right-mult-n-n*)

lemma *box-right-mult-an-an*: $|x](an(y) ; an(z)) = |x]an(y) ; |x]an(z)$
  by (*metis an-dist-add box-x-an mult-left-dist-add*)

lemma *box-n-export*: $|n(x) ; y]z = an(x) + |y]z$
  by (*metis box-1-y box-left-mult box-n-y mult-left-one*)

lemma *box-an-export*: $|an(x) ; y]z = n(x) + |y]z$
  by (*metis box-an-y box-left-mult box-n-an box-n-y n-an-def nbox-def*)

lemma *box-left-antitone*: $y \le x \longrightarrow |x]z \le |y]z$
  by (*smt an-mult-commutative an-order box-diamond box-left-dist-add less-eq-def*)

lemma *box-right-isotone*: $y \le z \longrightarrow |x]y \le |x]z$
  by (*metis an-antitone mult-left-isotone mult-right-isotone nbox-def*)

lemma *box-antitone-isotone*: $y \le w \land x \le z \longrightarrow |w]x \le |y]z$
  by (*metis box-left-antitone box-right-isotone order-trans*)

definition *nbox-L* :: $'a \Rightarrow 'a \Rightarrow 'a$ ($\|$ - $]\!]$ - [*50*,*90*] *95*)
  where $\|x]\!]y = an(x ; an(y) ; L) ; L$

lemma *nbox-to-L*: $\|x]\!]y = |x]n(y) ; L$
  by (*metis box-x-n nbox-L-def*)

lemma *nbox-from-L*: $|x]y = n(\|x]\!](y ; L))$
  by (*metis an-n-def nbox-L-def nbox-def*)

lemma *diamond-x-an*: $|x>an(y) = n(x ; an(y) ; L)$
  by (*metis ndiamond-def*)

lemma *diamond-1-an*: $|1>an(y) = an(y)$
  by (*metis an-n-def diamond-1-y*)

lemma *diamond-an-y*: $|an(x)>y = an(x) ; n(y ; L)$
  by (*metis mult-associative n-export-an ndiamond-def*)

lemma *diamond-an-0*: $|an(x)>0 = 0$
  by (*metis an-mult-right-zero diamond-x-n n-zero*)

lemma *diamond-an-1*: $|an(x)>1 = an(x)$
  by (*metis an-n-def diamond-x-1*)

lemma *diamond-an-n*: $|an(x)>n(y) = an(x) ; n(y)$
  by (*metis n-export-an n-mult ndiamond-def*)

lemma *diamond-n-an*: $|n(x)>an(y) = n(x) ; an(y)$
  by (*metis an-n-def diamond-n-y*)

lemma *diamond-an-an*: $|an(x)>an(y) = an(x) ; an(y)$
  by (*metis an-n-def diamond-an-n*)

lemma *diamond-an-an-same*: $|an(x)>an(x) = an(x)$
  by (*metis an-mult-idempotent an-n-def ndiamond-def*)

lemma *diamond-an-export*: $|an(x) ; y>z = an(x) ; |y>z$
  by (*metis mult-associative n-export-an ndiamond-def*)

lemma *box-ani*: $|x]y = an(x ; ani(y ; L))$
  by (*metis ani-def mult-associative nbox-def*)

lemma *box-x-n-ani*: $|x]n(y) = an(x ; ani(y))$
  by (*metis an-ani box-x-an*)

lemma *box-L-ani*: $\|x]\!]y = ani(x ; ani(y))$
  by (*metis ani-def mult-associative nbox-L-def*)

lemma *box-L-left-mult*: $\|x ; y]\!]z = \|x]\!]\|y]\!]z$
  by (*metis ani-def ani-mult mult-associative n-an-def nbox-L-def ni-def*)

lemma *diamond-x-an-ani*: $|x>an(y) = n(x ; ani(y))$
  by (*metis diamond-x-n n-ani*)

lemma *box-L-left-antitone*: $y \leq x \longrightarrow \|x]\!]z \leq \|y]\!]z$
  by (*metis ani-antitone ani-def mult-left-isotone nbox-L-def*)

lemma *box-L-right-isotone*: $y \leq z \longrightarrow \|x]\!]y \leq \|x]\!]z$
  by (*metis ani-antitone ani-def mult-associative mult-right-isotone nbox-L-def*)

lemma *box-L-antitone-isotone*: $y \leq w \wedge x \leq z \longrightarrow \|w]\!]x \leq \|y]\!]z$
  by (*metis box-L-left-antitone box-L-right-isotone order-trans*)

end

class *n-box-omega-algebra* = *n-box-semiring* + *an-omega-algebra*

begin

lemma *diamond-omega*: $|x^\omega>y = |x^\omega>z$
  by (*metis mult-associative n-omega-mult ndiamond-def*)

lemma *box-omega*: $|x^\omega]y = |x^\omega]z$
  by (*metis box-diamond diamond-omega*)

lemma *an-box-omega-induct*: $|x]an(y) ; n(z ; L) \leq an(y) \longrightarrow |x^\omega + x^\star]z \leq an(y)$
  by (*smt an-dist-add an-omega-induct an-omega-mult box-left-dist-add box-x-an mult-associative n-an-def nbox-def*)

lemma *n-box-omega-induct*: $|x]n(y)\ ;\ n(z\ ;\ L) \le n(y) \longrightarrow |x^\omega + x^\star]z \le n(y)$
  by (*metis an-box-omega-induct n-an-def*)

lemma *an-box-omega-induct-an*: $|x]an(y)\ ;\ an(z) \le an(y) \longrightarrow |x^\omega + x^\star]an(z) \le an(y)$
  by (*metis an-box-omega-induct an-n-def*)

— Theorem 23.13

lemma *n-box-omega-induct-n*: $|x]n(y)\ ;\ n(z) \le n(y) \longrightarrow |x^\omega + x^\star]n(z) \le n(y)$
  by (*metis n-box-omega-induct n-n-L*)

lemma *n-diamond-omega-induct*: $n(y) \le |x>n(y) + n(z\ ;\ L) \longrightarrow n(y) \le |x^\omega + x^\star>z$
  by (*smt n-dist-add n-omega-induct n-omega-mult diamond-left-dist-add diamond-x-n mult-associative ndiamond-def*)

lemma *an-diamond-omega-induct*: $an(y) \le |x>an(y) + n(z\ ;\ L) \longrightarrow an(y) \le |x^\omega + x^\star>z$
  by (*metis n-diamond-omega-induct an-n-def*)

— Theorem 23.9

lemma *n-diamond-omega-induct-n*: $n(y) \le |x>n(y) + n(z) \longrightarrow n(y) \le |x^\omega + x^\star>n(z)$
  by (*metis n-diamond-omega-induct n-n-L*)

lemma *an-diamond-omega-induct-an*: $an(y) \le |x>an(y) + an(z) \longrightarrow an(y) \le |x^\omega + x^\star>an(z)$
  by (*metis an-diamond-omega-induct an-n-def*)

lemma *box-segerberg-an*: $|x^\omega + x^\star]an(y) = an(y)\ ;\ |x^\omega + x^\star](n(y) + |x]an(y))$
proof (*rule antisym*)
  have $|x^\omega + x^\star]an(y) \le |x^\omega + x^\star]|x]an(y)$
    by (*smt box-left-dist-add box-left-mult box-omega add-right-isotone box-left-antitone mult-right-dist-add star.right-plus-below-circ*)
  hence $|x^\omega + x^\star]an(y) \le |x^\omega + x^\star](n(y) + |x]an(y))$
    by (*smt2 add-right-upper-bound box-right-isotone order-trans*)
  thus$|x^\omega + x^\star]an(y) \le an(y)\ ;\ |x^\omega + x^\star](n(y) + |x]an(y))$
    by (*metis add-least-upper-bound box-1-an box-left-antitone order-refl star-left-unfold-equal an-mult-least-upper-bound nbox-def*)
next
  have $an(y)\ ;\ |x](n(y) + |x^\omega + x^\star]an(y))\ ;\ (n(y) + |x]an(y)) = |x](\ |x^\omega + x^\star]an(y)\ ;\ an(y))\ ;\ an(y)$
    by (*smt add-left-zero an-export an-mult-commutative box-right-mult-an-an mult-associative mult-right-dist-add n-complement-zero nbox-def*)
  hence *1*: $an(y)\ ;\ |x](n(y) + |x^\omega + x^\star]an(y))\ ;\ (n(y) + |x]an(y)) \le n(y) + |x^\omega + x^\star]an(y)$
    by (*smt add-associative add-commutative add-right-upper-bound box-1-an box-left-dist-add box-left-mult mult-left-dist-add omega-unfold star-left-unfold-equal star.circ-plus-one*)
  have $n(y)\ ;\ |x](n(y) + |x^\omega + x^\star]an(y))\ ;\ (n(y) + |x]an(y)) \le n(y) + |x^\omega + x^\star]an(y)$
    by (*smt add-left-upper-bound an-n-def mult-left-isotone n-an-mult-left-lower-bound n-mult-left-absorb-add nbox-def order-trans*)
  thus $an(y)\ ;\ |x^\omega + x^\star](n(y) + |x]an(y)) \le |x^\omega + x^\star]an(y)$ **using** *1*
    by (*smt an-case-split-left an-shunting-an mult-associative n-box-omega-induct-n n-dist-add nbox-def nbox-from-L*)
qed

— Theorem 23.16

lemma *box-segerberg-n*: $|x^\omega + x^\star]n(y) = n(y)\ ;\ |x^\omega + x^\star](an(y) + |x]n(y))$
  by (*smt an-n-def box-segerberg-an n-an-def*)

lemma *diamond-segerberg-an*: $|x^\omega + x^\star>an(y) = an(y) + |x^\omega + x^\star>(n(y)\ ;\ |x>an(y))$
  by (*smt an-export an-n-L box-diamond box-segerberg-an diamond-box mult-associative n-an-def*)

— Theorem 23.12

lemma *diamond-segerberg-n*: $|x^\omega + x^\star>n(y) = n(y) + |x^\omega + x^\star>(an(y)\ ;\ |x>n(y))$
  by (*smt an-export an-n-L box-diamond box-segerberg-an diamond-box mult-associative n-an-def*)

— Theorem 23.11

lemma *diamond-star-unfold-n*: $|x^\star>n(y) = n(y) + |an(y)\ ;\ x>|x^\star>n(y)$
proof −
  have $|x^\star>n(y) = n(y) + n(y)\ ;\ |x\ ;\ x^\star>n(y) + |an(y)\ ;\ x\ ;\ x^\star>n(y)$
    by (*smt add-associative add-commutative add-right-zero an-complement an-complement-zero diamond-an-n diamond-left-dist-add diamond-n-export diamond-n-n-same mult-associative mult-left-one mult-right-dist-add*

star-left-unfold-equal)
  thus *?thesis*
    by (*metis diamond-left-mult diamond-x-n n-add-left-absorb-mult*)
qed

lemma *diamond-star-unfold-an*: $|x^\star>an(y) = an(y) + |n(y) ; x>|x^\star>an(y)$
  by (*metis an-n-def diamond-star-unfold-n n-an-def*)

— Theorem 23.15

lemma *box-star-unfold-n*: $|x^\star]n(y) = n(y) ; |n(y) ; x]|x^\star]n(y)$
  by (*smt an-export an-n-L box-diamond diamond-box diamond-star-unfold-an n-an-def n-export*)

lemma *box-star-unfold-an*: $|x^\star]an(y) = an(y) ; |an(y) ; x]|x^\star]an(y)$
  by (*metis an-n-def box-star-unfold-n*)

— Theorem 23.10

lemma *diamond-omega-unfold-n*: $|x^\omega + x^\star>n(y) = n(y) + |an(y) ; x>|x^\omega + x^\star>n(y)$
  by (*smt add-associative add-commutative diamond-an-export diamond-left-dist-add diamond-right-dist-add diamond-star-unfold-n diamond-x-n n-omega-mult n-plus-complement-intro-n omega-unfold*)

lemma *diamond-omega-unfold-an*: $|x^\omega + x^\star>an(y) = an(y) + |n(y) ; x>|x^\omega + x^\star>an(y)$
  by (*metis an-n-def diamond-omega-unfold-n n-an-def*)

— Theorem 23.14

lemma *box-omega-unfold-n*: $|x^\omega + x^\star]n(y) = n(y) ; |n(y) ; x]|x^\omega + x^\star]n(y)$
  by (*smt an-export an-n-L box-diamond diamond-box diamond-omega-unfold-an n-an-def n-export*)

lemma *box-omega-unfold-an*: $|x^\omega + x^\star]an(y) = an(y) ; |an(y) ; x]|x^\omega + x^\star]an(y)$
  by (*metis an-n-def box-omega-unfold-n*)

lemma *box-cut-iteration-an*: $|x^\omega + x^\star]an(y) = |(an(y) ; x)^\omega + (an(y) ; x)^\star]an(y)$
  by (*smt add-isotone an-box-omega-induct-an an-case-split-left an-mult-commutative antisym box-left-antitone box-omega-unfold-an nbox-def omega-isotone order-refl star.circ-isotone*)

lemma *box-cut-iteration-n*: $|x^\omega + x^\star]n(y) = |(n(y) ; x)^\omega + (n(y) ; x)^\star]n(y)$
  by (*metis n-an-def box-cut-iteration-an*)

lemma *diamond-cut-iteration-an*: $|x^\omega + x^\star>an(y) = |(n(y) ; x)^\omega + (n(y) ; x)^\star>an(y)$
  by (*metis box-cut-iteration-n diamond-box n-an-def*)

lemma *diamond-cut-iteration-n*: $|x^\omega + x^\star>n(y) = |(an(y) ; x)^\omega + (an(y) ; x)^\star>n(y)$
  by (*metis an-n-def diamond-cut-iteration-an n-an-def*)

lemma *ni-diamond-omega-induct*: $ni(y) \le \|x\gg ni(y) + ni(z) \longrightarrow ni(y) \le \|x^\omega + x^\star\gg z$
  by (*metis diamond-L-left-dist-add diamond-L-x-ni diamond-L-ni ni-dist-add ni-omega-induct ni-omega-mult*)

lemma *ani-diamond-omega-induct*: $ani(y) \le \|x\gg ani(y) + ni(z) \longrightarrow ani(y) \le \|x^\omega + x^\star\gg z$
  by (*metis ni-ani ni-diamond-omega-induct*)

lemma *n-diamond-omega-L*: $|n(x^\omega) ; L>y = |x^\omega>y$
  by (*metis L-left-zero mult-associative n-n-L n-omega-mult ndiamond-def*)

lemma *n-diamond-loop*: $|x^\Omega>y = |x^\omega + x^\star>y$
  by (*metis Omega-def diamond-left-dist-add n-diamond-omega-L*)

— Theorem 24.1

lemma *cut-iteration-loop*: $|x^\Omega>n(y) = |(an(y) ; x)^\Omega>n(y)$
  by (*metis n-diamond-loop diamond-cut-iteration-n*)

lemma *cut-iteration-while-loop*: $|x^\Omega>n(y) = |(an(y) ; x)^\Omega ; n(y)>n(y)$
  by (*metis cut-iteration-loop diamond-left-mult diamond-n-n-same*)

— Theorem 24.1

lemma *cut-iteration-while-loop-2*: $|x^\Omega>n(y) = |an(y) \star x>n(y)$

by (*metis cut-iteration-while-loop an-uminus n-an-def while-Omega-def*)

lemma *modal-while*: $-q ; -p ; L \leq x ; -p ; L \wedge -p \leq -q + -r \longrightarrow -p \leq |n((-q ; x)^\omega) ; L + (-q ; x)^\star ; --q>(-r)$
proof
assume *1*: $-q ; -p ; L \leq x ; -p ; L \wedge -p \leq -q + -r$
hence *2*: $--q ; -p \leq |-q ; x>(-p) + --q ; -r$
by (*smt add-associative add-commutative greatest-lower-bound leq-mult-zero less-eq-def lower-bound-right mult-associative plus-def sub-comm sub-mult-closed*)
have $-q ; -p = n(-q ; -q ; -p ; L)$
by (*metis an-uminus n-export-an mult-associative mult-right-one n-L mult-idempotent*)
also have ... $\leq n(-q ; x ; -p ; L)$ **using** *1*
by (*metis n-isotone mult-right-isotone mult-associative*)
also have ... $\leq |-q ; x>(-p) + --q ; -r$
by (*metis add-left-upper-bound ndiamond-def*)
finally have $-p \leq |-q ; x>(-p) + --q ; -r$ **using** *2*
by (*smt2 add-associative less-eq-def plus-cases sub-comm*)
thus $-p \leq |n((-q ; x)^\omega) ; L + (-q ; x)^\star ; --q>(-r)$
by (*smt L-left-zero an-diamond-omega-induct-an an-uminus diamond-left-dist-add mult-associative n-n-L n-omega-mult ndiamond-def sub-mult-closed*)
qed

lemma *modal-while-loop*: $-q ; -p ; L \leq x ; -p ; L \wedge -p \leq -q + -r \longrightarrow -p \leq |(-q ; x)^\Omega ; --q>(-r)$
by (*metis L-left-zero Omega-def modal-while mult-associative mult-right-dist-add*)

— Theorem 24.2

lemma *modal-while-loop-2*: $-q ; -p ; L \leq x ; -p ; L \wedge -p \leq -q + -r \longrightarrow -p \leq |-q \star x>(-r)$
by (*metis while-Omega-def modal-while-loop*)

lemma *modal-while-2*: $-p ; L \leq x ; -p ; L \longrightarrow -p \leq |n((-q ; x)^\omega) ; L + (-q ; x)^\star ; --q>(--q)$
proof −
have $-p ; L \leq x ; -p ; L \longrightarrow -p \leq |-q ; x>(-p) + --q$
by (*metis an-uminus double-negation n-an-def n-isotone ndiamond-def diamond-an-export add-associative add-commutative less-eq-def plus-compl-intro*)
thus *?thesis*
by (*smt L-left-zero an-diamond-omega-induct-an an-uminus diamond-left-dist-add mult-associative mult-idempotent n-n-L n-omega-mult ndiamond-def*)
qed

end

class *n-modal-omega-algebra* = *n-box-omega-algebra* +
assumes *n-star-induct*: $n(x ; y) \leq n(y) \longrightarrow n(x^\star ; y) \leq n(y)$

begin

lemma *n-star-induct-add*: $n(z + x ; y) \leq n(y) \longrightarrow n(x^\star ; z) \leq n(y)$
by (*metis an-dist-add an-mult-least-upper-bound an-n-order n-mult-right-upper-bound n-star-induct star-L-split*)

lemma *n-star-induct-star*: $n(x ; y) \leq n(y) \longrightarrow n(x^\star) \leq n(y)$
by (*metis n-mult-right-upper-bound n-star-induct*)

lemma *n-star-induct-iff*: $n(x ; y) \leq n(y) \longleftrightarrow n(x^\star ; y) \leq n(y)$
by (*metis mult-left-isotone n-isotone n-star-induct order-trans star.circ-increasing*)

lemma *n-star-zero*: $n(x) = 0 \longleftrightarrow n(x^\star) = 0$
by (*metis add-right-zero less-eq-def mult-right-one n-one n-star-induct-iff*)

lemma *n-diamond-star-induct*: $|x>n(y) \leq n(y) \longrightarrow |x^\star>n(y) \leq n(y)$
by (*metis diamond-x-n n-star-induct*)

lemma *n-diamond-star-induct-add*: $|x>n(y) + n(z) \leq n(y) \longrightarrow |x^\star>n(z) \leq n(y)$
by (*metis add-commutative diamond-x-n n-dist-add n-star-induct-add*)

lemma *n-diamond-star-induct-iff*: $|x>n(y) \leq n(y) \longleftrightarrow |x^\star>n(y) \leq n(y)$
by (*metis n-mult n-star-induct-iff ndiamond-def*)

lemma *an-star-induct*: $an(y) \leq an(x ; y) \longrightarrow an(y) \leq an(x^\star ; y)$
by (*metis an-n-order n-star-induct*)

lemma *an-star-induct-add*: $an(y) \le an(z + x ; y) \longrightarrow an(y) \le an(x^\star ; z)$
by (*metis an-n-order n-star-induct-add*)

lemma *an-star-induct-star*: $an(y) \le an(x ; y) \longrightarrow an(y) \le an(x^\star)$
by (*metis an-n-order n-star-induct-star*)

lemma *an-star-induct-iff*: $an(y) \le an(x ; y) \longleftrightarrow an(y) \le an(x^\star ; y)$
by (*metis an-n-order n-star-induct-iff*)

lemma *an-star-one*: $an(x) = 1 \longleftrightarrow an(x^\star) = 1$
by (*metis an-n-equal an-zero n-star-zero n-zero*)

lemma *an-box-star-induct*: $an(y) \le |x]an(y) \longrightarrow an(y) \le |x^\star]an(y)$
by (*metis an-star-induct box-x-an*)

lemma *an-box-star-induct-add*: $an(y) \le |x]an(y) ; an(z) \longrightarrow an(y) \le |x^\star]an(z)$
by (*metis add-commutative an-dist-add an-star-induct-add box-x-an*)

lemma *an-box-star-induct-iff*: $an(y) \le |x]an(y) \longleftrightarrow an(y) \le |x^\star]an(y)$
by (*metis an-star-induct-iff box-x-an*)

lemma *box-star-segerberg-an*: $|x^\star]an(y) = an(y) ; |x^\star](n(y) + |x]an(y))$
proof (*rule antisym*)
show $|x^\star]an(y) \le an(y) ; |x^\star](n(y) + |x]an(y))$
by (*metis add-right-upper-bound box-1-an box-left-dist-add box-left-mult box-right-isotone mult-right-isotone star.circ-right-unfold*)
next
have $an(y) ; |x^\star](n(y) + |x]an(y)) \le an(y) ; |x]an(y)$
by (*metis add-left-zero an-complement-zero box-an-an box-left-antitone box-x-an mult-left-dist-add mult-left-one mult-right-isotone star.circ-reflexive*)
thus $an(y) ; |x^\star](n(y) + |x]an(y)) \le |x^\star]an(y)$
by (*smt an-box-star-induct-add an-case-split-left an-dist-add an-mult-least-upper-bound box-left-antitone box-left-mult box-right-mult-an-an star.left-plus-below-circ nbox-def*)
qed

lemma *box-star-segerberg-n*: $|x^\star]n(y) = n(y) ; |x^\star](an(y) + |x]n(y))$
by (*metis box-diamond box-n-export box-star-segerberg-an box-x-an n-an-def nbox-def ndiamond-def*)

lemma *diamond-segerberg-an*: $|x^\star{>}an(y) = an(y) + |x^\star{>}(n(y) ; |x{>}an(y))$
by (*smt an-export an-n-L box-diamond box-star-segerberg-an diamond-box mult-associative n-an-def*)

lemma *diamond-star-segerberg-n*: $|x^\star{>}n(y) = n(y) + |x^\star{>}(an(y) ; |x{>}n(y))$
by (*smt an-export an-n-L box-diamond box-star-segerberg-an diamond-box mult-associative n-an-def*)

lemma *box-cut-star-iteration-an*: $|x^\star]an(y) = |(an(y) ; x)^\star]an(y)$
by (*smt an-box-star-induct-add an-mult-commutative an-mult-complement-intro-an antisym box-an-export box-star-unfold-an nbox-def order-refl*)

lemma *box-cut-star-iteration-n*: $|x^\star]n(y) = |(n(y) ; x)^\star]n(y)$
by (*metis box-cut-star-iteration-an n-an-def*)

lemma *diamond-cut-star-iteration-an*: $|x^\star{>}an(y) = |(n(y) ; x)^\star{>}an(y)$
by (*metis box-cut-star-iteration-n diamond-box n-an-def*)

lemma *diamond-cut-star-iteration-n*: $|x^\star{>}n(y) = |(an(y) ; x)^\star{>}n(y)$
by (*metis an-n-def diamond-cut-star-iteration-an n-an-def*)

lemma *ni-star-induct*: $ni(x ; y) \le ni(y) \longrightarrow ni(x^\star ; y) \le ni(y)$
by (*metis n-star-induct ni-n-order*)

lemma *ni-star-induct-add*: $ni(z + x ; y) \le ni(y) \longrightarrow ni(x^\star ; z) \le ni(y)$
by (*metis n-star-induct-add ni-n-order*)

lemma *ni-star-induct-star*: $ni(x ; y) \le ni(y) \longrightarrow ni(x^\star) \le ni(y)$
by (*metis n-star-induct-star ni-n-order*)

lemma *ni-star-induct-iff*: $ni(x ; y) \le ni(y) \longleftrightarrow ni(x^\star ; y) \le ni(y)$
by (*metis n-star-induct-iff ni-n-order*)

lemma *ni-star-zero*: $ni(x) = 0 \longleftrightarrow ni(x^\star) = 0$
 by (*metis n-star-zero ni-n-zero*)

lemma *ni-diamond-star-induct*: $\|x» ni(y) \le ni(y) \longrightarrow \|x^\star» ni(y) \le ni(y)$
 by (*metis diamond-L-x-ni ni-star-induct*)

lemma *ni-diamond-star-induct-add*: $\|x» ni(y) + ni(z) \le ni(y) \longrightarrow \|x^\star» ni(z) \le ni(y)$
 by (*metis add-commutative diamond-L-x-ni ni-dist-add ni-star-induct-add*)

lemma *ni-diamond-star-induct-iff*: $\|x» ni(y) \le ni(y) \longleftrightarrow \|x^\star» ni(y) \le ni(y)$
 by (*metis diamond-L-x-ni ni-star-induct-iff*)

lemma *ani-star-induct*: $ani(y) \le ani(x \ ; \ y) \longrightarrow ani(y) \le ani(x^\star \ ; \ y)$
 by (*metis an-star-induct ani-an-order*)

lemma *ani-star-induct-add*: $ani(y) \le ani(z + x \ ; \ y) \longrightarrow ani(y) \le ani(x^\star \ ; \ z)$
 by (*metis an-star-induct-add ani-an-order*)

lemma *ani-star-induct-star*: $ani(y) \le ani(x \ ; \ y) \longrightarrow ani(y) \le ani(x^\star)$
 by (*metis an-star-induct-star ani-an-order*)

lemma *ani-star-induct-iff*: $ani(y) \le ani(x \ ; \ y) \longleftrightarrow ani(y) \le ani(x^\star \ ; \ y)$
 by (*metis an-star-induct-iff ani-an-order*)

lemma *ani-star-L*: $ani(x) = L \longleftrightarrow ani(x^\star) = L$
 by (*metis an-star-one ani-an-L*)

lemma *ani-box-star-induct*: $ani(y) \le \|x]\!] ani(y) \longrightarrow ani(y) \le \|x^\star]\!] ani(y)$
 by (*metis an-ani ani-def ani-star-induct-iff n-ani box-L-ani*)

lemma *ani-box-star-induct-iff*: $ani(y) \le \|x]\!] ani(y) \longleftrightarrow ani(y) \le \|x^\star]\!] ani(y)$
 by (*smt an-ani ani-def ani-star-induct-iff n-ani box-L-ani*)

lemma *ani-box-star-induct-add*: $ani(y) \le \|x]\!] ani(y) \wedge ani(y) \le ani(z) \longrightarrow ani(y) \le \|x^\star]\!] ani(z)$
 by (*smt ani-box-star-induct-iff box-L-right-isotone order-trans*)

end

end

19 LatticeOrderedSemiring

```
theory LatticeOrderedSemiring

imports Semiring

begin

— Many results in this theory are taken from a joint paper with Rudolf Berghammer.

— M0-algebra

class lattice-ordered-pre-left-semiring = pre-left-semiring + bounded-distributive-lattice

begin

subclass bounded-pre-left-semiring
  apply unfold-locales
  apply (metis add-right-top-1)
  done

lemma top-mult-right-one: x ; T = x ; T ; 1
  by (metis add-commutative add-left-top less-eq-def mult-semi-associative mult-sub-right-one)

lemma mult-left-sub-dist-meet-left: x ; (y ⌢ z) ≤ x ; y
  by (metis meet.add-left-upper-bound mult-right-isotone)

lemma mult-left-sub-dist-meet-right: x ; (y ⌢ z) ≤ x ; z
  by (metis meet-commutative mult-left-sub-dist-meet-left)

lemma mult-right-sub-dist-meet-left: (x ⌢ y) ; z ≤ x ; z
  by (metis meet.add-left-upper-bound mult-left-isotone)

lemma mult-right-sub-dist-meet-right: (x ⌢ y) ; z ≤ y ; z
  by (metis meet.add-right-upper-bound mult-left-isotone)

lemma mult-right-sub-dist-meet: (x ⌢ y) ; z ≤ x ; z ⌢ y ; z
  by (metis meet.add-least-upper-bound mult-right-sub-dist-meet-left mult-right-sub-dist-meet-right)

— Figure 1: fundamental properties

definition total             :: 'a ⇒ bool where total x ⟷ x ; T = T
definition co-total          :: 'a ⇒ bool where co-total x ⟷ x ; 0 = 0
definition transitive        :: 'a ⇒ bool where transitive x ⟷ x ; x ≤ x
definition dense             :: 'a ⇒ bool where dense x ⟷ x ≤ x ; x
definition reflexive         :: 'a ⇒ bool where reflexive x ⟷ 1 ≤ x
definition co-reflexive      :: 'a ⇒ bool where co-reflexive x ⟷ x ≤ 1
definition idempotent        :: 'a ⇒ bool where idempotent x ⟷ x ; x = x
definition up-closed         :: 'a ⇒ bool where up-closed x ⟷ x ; 1 = x
definition add-distributive  :: 'a ⇒ bool where add-distributive x ⟷ (∀ y z . x ; (y + z) = x ; y + x ; z)
definition meet-distributive :: 'a ⇒ bool where meet-distributive x ⟷ (∀ y z . x ; (y ⌢ z) = x ; y ⌢ x ; z)
definition contact           :: 'a ⇒ bool where contact x ⟷ x ; x + 1 = x
definition kernel            :: 'a ⇒ bool where kernel x ⟷ x ; x ⌢ 1 = x ; 1
definition add-dist-contact  :: 'a ⇒ bool where add-dist-contact x ⟷ add-distributive x ∧ contact x
definition meet-dist-kernel  :: 'a ⇒ bool where meet-dist-kernel x ⟷ meet-distributive x ∧ kernel x
definition test              :: 'a ⇒ bool where test x ⟷ x ; T ⌢ 1 = x
definition co-test           :: 'a ⇒ bool where co-test x ⟷ x ; 0 + 1 = x
definition co-vector         :: 'a ⇒ bool where co-vector x ⟷ x ; 0 = x

— CPCP Theorem 5 / Figure 2: relations between properties

lemma reflexive-total: reflexive x ⟶ total x
  by (metis eq-iff mult-isotone mult-left-one meet.zero-least reflexive-def total-def)

lemma reflexive-dense: reflexive x ⟶ dense x
  by (metis mult-left-isotone mult-left-one reflexive-def dense-def)

lemma reflexive-transitive-up-closed: reflexive x ∧ transitive x ⟶ up-closed x
  by (metis antisym-conv mult-isotone mult-sub-right-one reflexive-def reflexive-dense transitive-def dense-def up-closed-def)
```

lemma *co-reflexive-co-total*: *co-reflexive x* $\longrightarrow$ *co-total x*
  by (*metis co-reflexive-def co-total-def eq-iff mult-left-isotone mult-left-one zero-least*)

lemma *co-reflexive-transitive*: *co-reflexive x* $\longrightarrow$ *transitive x*
  by (*metis co-reflexive-def mult-left-isotone mult-left-one transitive-def*)

lemma *idempotent-transitive-dense*: *idempotent x* $\longleftrightarrow$ *transitive x* $\wedge$ *dense x*
  by (*metis eq-iff transitive-def dense-def idempotent-def*)

lemma *contact-reflexive*: *contact x* $\longrightarrow$ *reflexive x*
  by (*metis contact-def add-right-upper-bound reflexive-def*)

lemma *contact-transitive*: *contact x* $\longrightarrow$ *transitive x*
  by (*metis contact-def add-left-upper-bound transitive-def*)

lemma *contact-dense*: *contact x* $\longrightarrow$ *dense x*
  by (*metis contact-reflexive reflexive-dense*)

lemma *contact-idempotent*: *contact x* $\longrightarrow$ *idempotent x*
  by (*metis contact-transitive contact-dense idempotent-transitive-dense*)

lemma *contact-up-closed*: *contact x* $\longrightarrow$ *up-closed x*
  by (*metis contact-def contact-idempotent dual-order.antisym mult-left-sub-dist-add-right mult-sub-right-one idempotent-def up-closed-def*)

lemma *contact-reflexive-idempotent-up-closed*: *contact x* $\longleftrightarrow$ *reflexive x* $\wedge$ *idempotent x* $\wedge$ *up-closed x*
  by (*metis contact-def contact-idempotent contact-up-closed add-commutative less-eq-def reflexive-def idempotent-def*)

lemma *kernel-co-reflexive*: *kernel x* $\longrightarrow$ *co-reflexive x*
  by (*metis co-reflexive-def kernel-def meet.add-least-upper-bound mult-sub-right-one*)

lemma *kernel-transitive*: *kernel x* $\longrightarrow$ *transitive x*
  by (*metis co-reflexive-transitive kernel-co-reflexive*)

lemma *kernel-dense*: *kernel x* $\longrightarrow$ *dense x*
  by (*metis kernel-def meet.add-least-upper-bound mult-sub-right-one dense-def*)

lemma *kernel-idempotent*: *kernel x* $\longrightarrow$ *idempotent x*
  by (*metis kernel-transitive kernel-dense idempotent-transitive-dense*)

lemma *kernel-up-closed*: *kernel x* $\longrightarrow$ *up-closed x*
  by (*metis co-reflexive-def kernel-co-reflexive kernel-def kernel-idempotent meet-less-eq-def idempotent-def up-closed-def*)

lemma *kernel-co-reflexive-idempotent-up-closed*: *kernel x* $\longleftrightarrow$ *co-reflexive x* $\wedge$ *idempotent x* $\wedge$ *up-closed x*
  by (*metis co-reflexive-def kernel-def kernel-idempotent kernel-up-closed meet.less-eq-def meet-commutative idempotent-def up-closed-def*)

lemma *test-co-reflexive*: *test x* $\longrightarrow$ *co-reflexive x*
  by (*metis co-reflexive-def meet.add-right-upper-bound test-def*)

lemma *test-up-closed*: *test x* $\longrightarrow$ *up-closed x*
  by (*metis eq-iff mult-left-one mult-sub-right-one mult-right-sub-dist-meet test-def top-mult-right-one up-closed-def*)

lemma *co-test-reflexive*: *co-test x* $\longrightarrow$ *reflexive x*
  by (*metis co-test-def add-right-upper-bound reflexive-def*)

lemma *co-test-transitive*: *co-test x* $\longrightarrow$ *transitive x*
  by (*smt2 co-test-def add-associative less-eq-def mult-left-one mult-left-zero mult-right-dist-add mult-semi-associative transitive-def*)

lemma *co-test-idempotent*: *co-test x* $\longrightarrow$ *idempotent x*
  by (*metis co-test-reflexive co-test-transitive reflexive-dense idempotent-transitive-dense*)

lemma *co-test-up-closed*: *co-test x* $\longrightarrow$ *up-closed x*
  by (*metis co-test-reflexive co-test-idempotent contact-def contact-up-closed add-commutative less-eq-def reflexive-def idempotent-def*)

lemma *co-test-contact*: *co-test x* $\longrightarrow$ *contact x*
  by (*metis co-test-reflexive co-test-idempotent co-test-up-closed contact-reflexive-idempotent-up-closed*)

lemma *vector-transitive*: *vector x ⟶ transitive x*
  by (*metis mult-right-isotone meet.zero-least vector-def transitive-def*)

lemma *vector-up-closed*: *vector x ⟶ up-closed x*
  by (*metis vector-def top-mult-right-one up-closed-def*)

— CPCP Theorem 8 / Figure 3: closure properties

— total

lemma *one-total*: *total 1*
  by (*metis mult-left-one total-def*)

lemma *top-total*: *total T*
  by (*metis top-mult-top total-def*)

lemma *add-total*: *total x ∧ total y ⟶ total (x + y)*
  by (*metis add-left-top mult-right-dist-add total-def*)

— co-total

lemma *zero-co-total*: *co-total 0*
  by (*metis co-total-def mult-left-zero*)

lemma *one-co-total*: *co-total 1*
  by (*metis co-total-def mult-left-one*)

lemma *add-co-total*: *co-total x ∧ co-total y ⟶ co-total (x + y)*
  by (*metis co-total-def add-right-zero mult-right-dist-add*)

lemma *meet-co-total*: *co-total x ∧ co-total y ⟶ co-total (x ⌢ y)*
  by (*metis co-total-def add-left-zero antisym-conv less-eq-def mult-right-sub-dist-meet-left*)

lemma *comp-co-total*: *co-total x ∧ co-total y ⟶ co-total (x ; y)*
  by (*metis co-total-def eq-iff mult-semi-associative zero-least*)

— sub-transitive

lemma *zero-transitive*: *transitive 0*
  by (*metis mult-left-zero zero-least transitive-def*)

lemma *one-transitive*: *transitive 1*
  by (*metis mult-left-one order-refl transitive-def*)

lemma *top-transitive*: *transitive T*
  by (*metis meet.zero-least transitive-def*)

lemma *meet-transitive*: *transitive x ∧ transitive y ⟶ transitive (x ⌢ y)*
  by (*smt2 meet.less-eq-def meet-associative meet-commutative mult-left-sub-dist-meet-left mult-right-sub-dist-meet-left transitive-def*)

— dense

lemma *zero-dense*: *dense 0*
  by (*metis zero-least dense-def*)

lemma *one-dense*: *dense 1*
  by (*metis mult-sub-right-one dense-def*)

lemma *top-dense*: *dense T*
  by (*metis top-left-mult-increasing dense-def*)

lemma *add-dense*: *dense x ∧ dense y ⟶ dense (x + y)*
proof
  assume *dense x ∧ dense y*
  hence $x \le x ; x \wedge y \le y ; y$
    by (*metis dense-def*)
  hence $x \le (x + y) ; (x + y) \wedge y \le (x + y) ; (x + y)$
    by (*metis add-left-upper-bound dual-order.trans mult-isotone add-right-upper-bound*)

hence $x + y \leq (x + y) ; (x + y)$
  by (*metis add-least-upper-bound*)
thus *dense* $(x + y)$
  by (*metis dense-def*)
qed

— reflexive

lemma *one-reflexive*: *reflexive 1*
  by (*metis order-refl reflexive-def*)

lemma *top-reflexive*: *reflexive T*
  by (*metis meet.zero-least reflexive-def*)

lemma *add-reflexive*: *reflexive* $x \wedge$ *reflexive* $y \longrightarrow$ *reflexive* $(x + y)$
  by (*metis add-associative less-eq-def reflexive-def*)

lemma *meet-reflexive*: *reflexive* $x \wedge$ *reflexive* $y \longrightarrow$ *reflexive* $(x \frown y)$
  by (*metis meet.add-least-upper-bound reflexive-def*)

lemma *comp-reflexive*: *reflexive* $x \wedge$ *reflexive* $y \longrightarrow$ *reflexive* $(x ; y)$
  by (*metis mult-left-isotone mult-left-one order-trans reflexive-def*)

— co-reflexive

lemma *zero-co-reflexive*: *co-reflexive 0*
  by (*metis co-reflexive-def zero-least*)

lemma *one-co-reflexive*: *co-reflexive 1*
  by (*metis co-reflexive-def order-refl*)

lemma *add-co-reflexive*: *co-reflexive* $x \wedge$ *co-reflexive* $y \longrightarrow$ *co-reflexive* $(x + y)$
  by (*metis co-reflexive-def add-least-upper-bound*)

lemma *meet-co-reflexive*: *co-reflexive* $x \wedge$ *co-reflexive* $y \longrightarrow$ *co-reflexive* $(x \frown y)$
  by (*metis co-reflexive-def meet.less-eq-def meet-associative*)

lemma *comp-co-reflexive*: *co-reflexive* $x \wedge$ *co-reflexive* $y \longrightarrow$ *co-reflexive* $(x ; y)$
  by (*metis co-reflexive-def mult-isotone mult-left-one*)

— idempotent

lemma *zero-idempotent*: *idempotent 0*
  by (*metis mult-left-zero idempotent-def*)

lemma *one-idempotent*: *idempotent 1*
  by (*metis mult-left-one idempotent-def*)

lemma *top-idempotent*: *idempotent T*
  by (*metis top-mult-top idempotent-def*)

— up-closed

lemma *zero-up-closed*: *up-closed 0*
  by (*metis mult-left-zero up-closed-def*)

lemma *one-up-closed*: *up-closed 1*
  by (*metis mult-left-one up-closed-def*)

lemma *top-up-closed*: *up-closed T*
  by (*metis top-mult-top vector-def vector-up-closed*)

lemma *add-up-closed*: *up-closed* $x \wedge$ *up-closed* $y \longrightarrow$ *up-closed* $(x + y)$
  by (*metis mult-right-dist-add up-closed-def*)

lemma *meet-up-closed*: *up-closed* $x \wedge$ *up-closed* $y \longrightarrow$ *up-closed* $(x \frown y)$
  by (*metis dual-order.antisym mult-sub-right-one mult-right-sub-dist-meet up-closed-def*)

lemma *comp-up-closed*: *up-closed* $x \wedge$ *up-closed* $y \longrightarrow$ *up-closed* $(x ; y)$

by (*metis dual-order.antisym mult-semi-associative mult-sub-right-one up-closed-def*)

— add-distributive

lemma *zero-add-distributive*: *add-distributive 0*
by (*metis add-distributive-def add-idempotent mult-left-zero*)

lemma *one-add-distributive*: *add-distributive 1*
by (*metis add-distributive-def mult-left-one*)

lemma *add-add-distributive*: *add-distributive* $x \wedge$ *add-distributive* $y \longrightarrow$ *add-distributive* $(x + y)$
by (*smt2 add-distributive-def add-associative add-commutative mult-right-dist-add*)

— meet-distributive

lemma *zero-meet-distributive*: *meet-distributive 0*
by (*metis meet-left-zero mult-left-zero meet-distributive-def*)

lemma *one-meet-distributive*: *meet-distributive 1*
by (*metis mult-left-one meet-distributive-def*)

— contact

lemma *one-contact*: *contact 1*
by (*metis contact-def add-idempotent mult-left-one*)

lemma *top-contact*: *contact T*
by (*metis contact-def add-left-top top-mult-top*)

lemma *meet-contact*: *contact* $x \wedge$ *contact* $y \longrightarrow$ *contact* $(x \frown y)$
by (*smt2 contact-def contact-reflexive contact-transitive contact-up-closed meet.less-eq-def meet-commutative meet-left-dist-add mult-left-sub-dist-add-right meet-transitive meet-up-closed reflexive-def transitive-def up-closed-def*)

— kernel

lemma *zero-kernel*: *kernel 0*
by (*metis kernel-co-reflexive-idempotent-up-closed zero-co-reflexive zero-idempotent zero-up-closed*)

lemma *one-kernel*: *kernel 1*
by (*metis kernel-def meet-idempotent mult-left-one*)

lemma *add-kernel*: *kernel* $x \wedge$ *kernel* $y \longrightarrow$ *kernel* $(x + y)$
by (*metis add-co-reflexive add-dense add-up-closed co-reflexive-transitive kernel-co-reflexive-idempotent-up-closed idempotent-transitive-dense*)

— add-distributive contact

lemma *one-add-dist-contact*: *add-dist-contact 1*
by (*metis add-dist-contact-def one-add-distributive one-contact*)

— meet-distributive kernel

lemma *zero-meet-dist-kernel*: *meet-dist-kernel 0*
by (*metis meet-dist-kernel-def zero-kernel zero-meet-distributive*)

lemma *one-meet-dist-kernel*: *meet-dist-kernel 1*
by (*metis meet-dist-kernel-def one-kernel one-meet-distributive*)

— test

lemma *zero-test*: *test 0*
by (*metis meet-commutative meet-right-zero mult-left-zero test-def*)

lemma *one-test*: *test 1*
by (*metis meet-left-top mult-left-one test-def*)

lemma *add-test*: *test* $x \wedge$ *test* $y \longrightarrow$ *test* $(x + y)$
by (*metis* (*no-types*, *lifting*) *meet-commutative meet-left-dist-add mult-right-dist-add test-def*)

lemma *meet-test*: *test x* ∧ *test y* ⟶ *test* (*x* ⌢ *y*)
by (*smt2 test-def meet-commutative meet.add-least-upper-bound meet.add-right-isotone mult-right-sub-dist-meet-left meet.add-left-upper-bound top-right-mult-increasing antisym*)

— co-test

lemma *one-co-test*: *co-test 1*
by (*metis co-test-def co-total-def add-left-zero one-co-total*)

lemma *add-co-test*: *co-test x* ∧ *co-test y* ⟶ *co-test* (*x* + *y*)
by (*smt2 co-test-contact co-test-def contact-def add-associative add-commutative add-left-zero mult-left-one mult-right-dist-add*)

— vector

lemma *zero-vector*: *vector 0*
by (*metis mult-left-zero vector-def*)

lemma *top-vector*: *vector T*
by (*metis top-mult-top vector-def*)

lemma *add-vector*: *vector x* ∧ *vector y* ⟶ *vector* (*x* + *y*)
by (*metis mult-right-dist-add vector-def*)

lemma *meet-vector*: *vector x* ∧ *vector y* ⟶ *vector* (*x* ⌢ *y*)
by (*metis antisym meet.add-least-upper-bound mult-right-sub-dist-meet-left mult-right-sub-dist-meet-right top-right-mult-increasing vector-def*)

lemma *comp-vector*: *vector y* ⟶ *vector* (*x* ; *y*)
by (*metis antisym-conv mult-semi-associative top-right-mult-increasing vector-def*)

end

class *lattice-ordered-pre-left-semiring-1* = *non-associative-left-semiring* + *bounded-distributive-lattice* +
assumes *mult-associative-one*: *x* ; (*y* ; *z*) = (*x* ; (*y* ; *1*)) ; *z*
assumes *mult-right-dist-meet-one*: (*x* ; *1* ⌢ *y* ; *1*) ; *z* = *x* ; *z* ⌢ *y* ; *z*

begin

subclass *pre-left-semiring*
apply *unfold-locales*
apply (*metis mult-associative-one mult-left-isotone mult-right-isotone mult-sub-right-one*)
done

subclass *lattice-ordered-pre-left-semiring*
..

lemma *mult-zero-associative*: *x* ; *0* ; *y* = *x* ; *0*
by (*smt mult-left-zero mult-associative-one*)

lemma *mult-zero-add-one-dist*: (*x* ; *0* + *1*) ; *z* = *x* ; *0* + *z*
by (*metis mult-left-one mult-right-dist-add mult-zero-associative*)

lemma *mult-zero-add-dist*: (*x* ; *0* + *y*) ; *z* = *x* ; *0* + *y* ; *z*
by (*metis mult-right-dist-add mult-zero-associative*)

lemma *vector-zero-meet-one-comp*: (*x* ; *0* ⌢ *1*) ; *y* = *x* ; *0* ⌢ *y*
by (*metis mult-left-one mult-right-dist-meet-one mult-zero-associative*)

— CPCP Theorem 5 / Figure 2: relations between properties

lemma *co-test-meet-distributive*: *co-test x* ⟶ *meet-distributive x*
by (*metis add-left-dist-meet co-test-def meet-distributive-def mult-zero-add-one-dist*)

lemma *co-test-add-distributive*: *co-test x* ⟶ *add-distributive x*
by (*smt2 add-associative add-commutative add-distributive-def add-left-upper-bound co-test-def less-eq-def mult-zero-add-one-dist*)

lemma *co-test-add-dist-contact*: *co-test x* ⟶ *add-dist-contact x*
by (*metis co-test-add-distributive add-dist-contact-def co-test-contact*)

— CPCP Theorem 8 / Figure 3: closure properties

— co-test

lemma *meet-co-test*: *co-test* $x \wedge$ *co-test* $y \longrightarrow$ *co-test* $(x \frown y)$
  by (*smt2 add-commutative add-left-dist-meet co-test-def co-test-up-closed up-closed-def mult-right-dist-meet-one*)

lemma *comp-co-test*: *co-test* $x \wedge$ *co-test* $y \longrightarrow$ *co-test* $(x ; y)$
  by (*metis add-associative co-test-def mult-zero-add-dist mult-zero-add-one-dist*)

end

class *lattice-ordered-pre-left-semiring-2* = *lattice-ordered-pre-left-semiring* +
  assumes *mult-sub-associative-one*: $x ; (y ; z) \leq (x ; (y ; 1)) ; z$
  assumes *mult-right-dist-meet-one-sub*: $x ; z \frown y ; z \leq (x ; 1 \frown y ; 1) ; z$

begin

subclass *lattice-ordered-pre-left-semiring-1*
  apply *unfold-locales*
  apply (*metis meet.eq-iff mult-sub-associative-one mult-sup-associative-one*)
  apply (*metis meet.antisym-conv mult-one-associative mult-right-dist-meet-one-sub mult-right-sub-dist-meet*)
  done

end

class *multirelation-algebra-1* = *lattice-ordered-pre-left-semiring* +
  assumes *mult-left-top*: $T ; x = T$

begin

— CPCP Theorem 8 / Figure 3: closure properties

lemma *top-add-distributive*: *add-distributive* T
  by (*metis add-distributive-def add-left-top mult-left-top*)

lemma *top-meet-distributive*: *meet-distributive* T
  by (*metis meet-idempotent meet-distributive-def mult-left-top*)

lemma *top-add-dist-contact*: *add-dist-contact* T
  by (*metis add-dist-contact-def top-add-distributive top-contact*)

lemma *top-co-test*: *co-test* T
  by (*metis co-test-def add-left-top mult-left-top*)

end

— M1-algebra

class *multirelation-algebra-2* = *multirelation-algebra-1* + *lattice-ordered-pre-left-semiring-2*

begin

lemma *mult-top-associative*: $x ; T ; y = x ; T$
  by (*metis mult-left-top mult-associative-one*)

lemma *vector-meet-one-comp*: $(x ; T \frown 1) ; y = x ; T \frown y$
  by (*metis mult-left-one mult-left-top mult-associative-one mult-right-dist-meet-one*)

lemma *vector-left-annihilator*: *vector* $x \longrightarrow x ; y = x$
  by (*metis mult-left-top vector-def mult-associative-one*)

— properties

lemma *test-comp-meet*: *test* $x \wedge$ *test* $y \longrightarrow x ; y = x \frown y$
  by (*smt2 meet-associative meet-commutative meet-idempotent test-def vector-meet-one-comp*)

— CPCP Theorem 5 / Figure 2: relations between properties

lemma *test-add-distributive*: *test x* $\longrightarrow$ *add-distributive x*
by (*metis add-distributive-def meet-left-dist-add test-def vector-meet-one-comp*)

lemma *test-meet-distributive*: *test x* $\longrightarrow$ *meet-distributive x*
by (*smt2 meet.less-eq-def meet-associative meet-commutative meet-distributive-def meet.add-right-upper-bound mult-left-one test-def vector-meet-one-comp*)

lemma *test-meet-dist-kernel*: *test x* $\longrightarrow$ *meet-dist-kernel x*
by (*metis kernel-co-reflexive-idempotent-up-closed meet-associative meet-dist-kernel-def meet-idempotent test-co-reflexive test-def test-up-closed idempotent-def vector-meet-one-comp test-meet-distributive*)

lemma *vector-idempotent*: *vector x* $\longrightarrow$ *idempotent x*
by (*metis idempotent-def vector-left-annihilator*)

lemma *vector-add-distributive*: *vector x* $\longrightarrow$ *add-distributive x*
by (*metis add-distributive-def add-idempotent vector-left-annihilator*)

lemma *vector-meet-distributive*: *vector x* $\longrightarrow$ *meet-distributive x*
by (*metis meet-distributive-def meet-idempotent vector-left-annihilator*)

lemma *vector-co-vector*: *vector x* $\longleftrightarrow$ *co-vector x*
by (*metis co-vector-def vector-def mult-zero-associative vector-left-annihilator*)

— CPCP Theorem 8 / Figure 3: closure properties

— test

lemma *comp-test*: *test x* $\wedge$ *test y* $\longrightarrow$ *test* (*x* ; *y*)
by (*metis meet-associative meet-distributive-def meet.add-right-zero test-def test-up-closed up-closed-def mult-associative-one test-meet-distributive*)

end

class *dual* =
fixes *dual* :: *'a* $\Rightarrow$ *'a* (-^d [*100*] *100*)

class *multirelation-algebra-3* = *lattice-ordered-pre-left-semiring* + *dual* +
assumes *dual-involutive*: $x^{dd} = x$
assumes *dual-dist-add*: $(x + y)^d = x^d \frown y^d$
assumes *dual-one*: $1^d = 1$

begin

lemma *dual-dist-meet*: $(x \frown y)^d = x^d + y^d$
by (*metis dual-dist-add dual-involutive*)

lemma *dual-antitone*: $x \leq y \longrightarrow y^d \leq x^d$
by (*metis dual-dist-meet add-left-divisibility meet.add-left-divisibility*)

lemma *dual-zero*: $0^d = T$
by (*metis dual-dist-meet add-right-top dual-involutive meet-left-zero*)

lemma *dual-top*: $T^d = 0$
by (*metis dual-zero dual-involutive*)

— CPCP Theorem 8 / Figure 3: closure properties

lemma *reflexive-co-reflexive-dual*: *reflexive x* $\longleftrightarrow$ *co-reflexive* (x^d)
by (*metis co-reflexive-def dual-antitone dual-involutive dual-one reflexive-def*)

end

class *multirelation-algebra-4* = *multirelation-algebra-3* +
assumes *dual-sub-dist-comp*: $(x ; y)^d \leq x^d ; y^d$

begin

subclass *multirelation-algebra-1*
apply *unfold-locales*

apply (*metis dual-zero dual-sub-dist-comp dual-involutive meet.less-eq-def meet-commutative meet-left-top mult-left-zero*)
done

lemma *dual-sub-dist-comp-one*: $(x\ ;\ y)^d \leq (x\ ;\ 1)^d\ ;\ y^d$
by (*metis dual-sub-dist-comp mult-one-associative*)

— CPCP Theorem 8 / Figure 3: closure properties

lemma *co-total-total-dual*: *co-total* $x \longrightarrow$ *total* (x^d)
by (*metis co-total-def dual-sub-dist-comp dual-zero meet.less-eq-def meet-commutative meet-left-top total-def*)

lemma *transitive-dense-dual*: *transitive* $x \longrightarrow$ *dense* (x^d)
by (*metis dual-antitone dual-sub-dist-comp order-trans transitive-def dense-def*)

end

— M2-algebra

class *multirelation-algebra-5* = *multirelation-algebra-3* +
assumes *dual-dist-comp-one*: $(x\ ;\ y)^d = (x\ ;\ 1)^d\ ;\ y^d$

begin

subclass *multirelation-algebra-4*
apply *unfold-locales*
apply (*metis dual-antitone mult-sub-right-one mult-left-isotone dual-dist-comp-one*)
done

lemma *strong-up-closed*: $x\ ;\ 1 \leq x \longrightarrow x^d\ ;\ y^d \leq (x\ ;\ y)^d$
by (*metis dual-dist-comp-one eq-iff mult-sub-right-one*)

lemma *strong-up-closed-2*: *up-closed* $x \longrightarrow (x\ ;\ y)^d = x^d\ ;\ y^d$
by (*metis dual-sub-dist-comp eq-iff strong-up-closed up-closed-def*)

subclass *lattice-ordered-pre-left-semiring-2*
apply *unfold-locales*
apply (*smt2 comp-up-closed dual-antitone dual-dist-comp-one dual-involutive dual-one mult-left-one mult-one-associative mult-semi-associative up-closed-def strong-up-closed-2*)
apply (*smt2 dual-dist-comp-one dual-dist-meet dual-involutive eq-refl mult-one-associative mult-right-dist-add*)
done

— CPCP Theorem 6

subclass *multirelation-algebra-2*
..

— CPCP Theorem 8 / Figure 3: closure properties

— up-closed

lemma *up-closed-dual*: *up-closed* $x \longleftrightarrow$ *up-closed* (x^d)
by (*metis dual-involutive dual-one up-closed-def strong-up-closed-2*)

— contact

lemma *contact-kernel-dual*: *contact* $x \longleftrightarrow$ *kernel* (x^d)
by (*metis contact-def contact-up-closed dual-dist-add dual-involutive dual-one kernel-def kernel-up-closed up-closed-def strong-up-closed-2*)

— add-distributive contact

lemma *add-dist-contact-meet-dist-kernel-dual*: *add-dist-contact* $x \longleftrightarrow$ *meet-dist-kernel* (x^d)
proof
assume *1*: *add-dist-contact* x
hence *2*: *up-closed* x
by (*metis add-dist-contact-def contact-up-closed*)
have *add-distributive* x **using** *1*
by (*metis add-dist-contact-def*)
hence *meet-distributive* (x^d) **using** *2*

by (*smt2 meet-distributive-def add-distributive-def dual-dist-add dual-involutive strong-up-closed-2*)
thus *meet-dist-kernel* (x^d) **using** *1*
by (*metis contact-kernel-dual add-dist-contact-def meet-dist-kernel-def*)
next
assume *3*: *meet-dist-kernel* (x^d)
hence *2*: *up-closed* (x^d)
by (*metis kernel-up-closed meet-dist-kernel-def*)
have *meet-distributive* (x^d) **using** *3*
by (*metis meet-dist-kernel-def*)
hence *add-distributive* (x^{dd}) **using** *2*
by (*smt2 meet-distributive-def add-distributive-def dual-dist-add dual-involutive strong-up-closed-2*)
thus *add-dist-contact* x **using** *3*
by (*metis contact-kernel-dual add-dist-contact-def meet-dist-kernel-def dual-involutive*)
qed

— test

lemma *test-co-test-dual*: *test* $x \longleftrightarrow$ *co-test* (x^d)
by (*smt2 co-test-def co-test-up-closed dual-dist-meet dual-involutive dual-one dual-top test-def test-up-closed strong-up-closed-2*)

— vector

lemma *vector-dual*: *vector* $x \longleftrightarrow$ *vector* (x^d)
by (*metis dual-dist-comp-one comp-vector dual-involutive dual-top vector-def zero-vector*)

end

class *multirelation-algebra-6* = *multirelation-algebra-4* +
assumes *dual-sub-dist-comp-one*: $(x\ ;\ 1)^d\ ;\ y^d \le (x\ ;\ y)^d$

begin

subclass *multirelation-algebra-5*
apply *unfold-locales*
apply (*metis dual-sub-dist-comp dual-sub-dist-comp-one meet.eq-iff mult-one-associative*)
done

lemma *dense* $x \wedge$ *co-reflexive* $x \longrightarrow$ *up-closed* x **nitpick** [*expect=genuine*] **oops**
lemma $x\ ;\ T \frown y\ ;\ z \le (x\ ;\ T \frown y)\ ;\ z$ **nitpick** [*expect=genuine,card=8*] **oops**

end

— M3-algebra

class *up-closed-multirelation-algebra* = *multirelation-algebra-3* +
assumes *dual-dist-comp*: $(x\ ;\ y)^d = x^d\ ;\ y^d$

begin

lemma *mult-right-dist-meet*: $(x \frown y)\ ;\ z = x\ ;\ z \frown y\ ;\ z$
by (*metis dual-dist-add dual-dist-comp dual-involutive mult-right-dist-add*)

— CPCP Theorem 7

subclass *idempotent-left-semiring*
apply *unfold-locales*
apply (*metis antisym dual-antitone dual-dist-comp dual-involutive mult-semi-associative*)
apply (*metis mult-left-one*)
apply (*metis dual-dist-add dual-dist-comp dual-involutive dual-one less-eq-def meet-absorb mult-sub-right-one*)
done

subclass *multirelation-algebra-6*
apply *unfold-locales*
apply (*metis dual-dist-comp eq-iff*)
apply (*metis dual-dist-comp eq-iff mult-right-one*)
done

lemma *vector-meet-comp*: $(x\ ;\ T \frown y)\ ;\ z = x\ ;\ T \frown y\ ;\ z$
by (*metis mult-associative mult-left-top mult-right-dist-meet*)

lemma *vector-zero-meet-comp*: $(x ; 0 \frown y) ; z = x ; 0 \frown y ; z$
by (*metis mult-associative mult-left-zero mult-right-dist-meet*)

— CPCP Theorem 8 / Figure 3: closure properties

— total

lemma *meet-total*: *total* $x \wedge$ *total* $y \longrightarrow$ *total* $(x \frown y)$
by (*metis meet-left-top total-def mult-right-dist-meet*)

lemma *comp-total*: *total* $x \wedge$ *total* $y \longrightarrow$ *total* $(x ; y)$
by (*metis mult-associative total-def*)

lemma *total-co-total-dual*: *total* $x \longleftrightarrow$ *co-total* (x^d)
by (*metis co-total-def dual-dist-comp dual-involutive dual-top total-def*)

— dense

lemma *transitive-iff-dense-dual*: *transitive* $x \longleftrightarrow$ *dense* (x^d)
by (*metis dense-def dual-antitone dual-dist-comp dual-involutive transitive-def*)

— idempotent

lemma *idempotent-dual*: *idempotent* $x \longleftrightarrow$ *idempotent* (x^d)
by (*metis dual-involutive idempotent-transitive-dense transitive-iff-dense-dual*)

— add-distributive

lemma *comp-add-distributive*: *add-distributive* $x \wedge$ *add-distributive* $y \longrightarrow$ *add-distributive* $(x ; y)$
by (*metis add-distributive-def mult-associative*)

lemma *add-meet-distributive-dual*: *add-distributive* $x \longleftrightarrow$ *meet-distributive* (x^d)
by (*metis* (*no-types*, *hide-lams*) *add-distributive-def dual-dist-add dual-dist-comp dual-involutive meet-distributive-def*)

— meet-distributive

lemma *meet-meet-distributive*: *meet-distributive* $x \wedge$ *meet-distributive* $y \longrightarrow$ *meet-distributive* $(x \frown y)$
by (*smt2 meet-distributive-def meet-associative meet-commutative mult-right-dist-meet*)

lemma *comp-meet-distributive*: *meet-distributive* $x \wedge$ *meet-distributive* $y \longrightarrow$ *meet-distributive* $(x ; y)$
by (*metis meet-distributive-def mult-associative*)

lemma *co-total* $x \wedge$ *transitive* $x \wedge$ *up-closed* $x \longrightarrow$ *co-reflexive* x **nitpick** [*expect=genuine*] **oops**
lemma *total* $x \wedge$ *dense* $x \wedge$ *up-closed* $x \longrightarrow$ *reflexive* x **nitpick** [*expect=genuine*] **oops**
lemma $x ; T \frown x^d ; 0 = 0$ **nitpick** [*expect=genuine*] **oops**

end

class *multirelation-algebra-7* = *multirelation-algebra-4* +
assumes *vector-meet-comp*: $(x ; T \frown y) ; z = x ; T \frown y ; z$

begin

lemma *vector-zero-meet-comp*: $(x ; 0 \frown y) ; z = x ; 0 \frown y ; z$
by (*metis vector-def comp-vector vector-meet-comp zero-vector*)

lemma *test-add-distributive*: *test* $x \longrightarrow$ *add-distributive* x
by (*metis add-distributive-def meet-left-dist-add mult-left-one test-def vector-meet-comp*)

lemma *test-meet-distributive*: *test* $x \longrightarrow$ *meet-distributive* x
by (*smt2 meet.less-eq-def meet-associative meet-commutative meet-distributive-def meet.add-right-upper-bound mult-left-one test-def vector-meet-comp*)

lemma *test-meet-dist-kernel*: *test* $x \longrightarrow$ *meet-dist-kernel* x
by (*metis kernel-co-reflexive-idempotent-up-closed meet-associative meet-dist-kernel-def meet-idempotent mult-left-one test-co-reflexive test-def test-up-closed idempotent-def vector-meet-comp test-meet-distributive*)

lemma *co-test-meet-distributive*: *co-test* $x \longrightarrow$ *meet-distributive* x
proof

```
  assume co-test x
  hence x = x ; 0 + 1
    by (metis co-test-def)
  hence ∀ y z . x ; y ⌢ x ; z = x ; (y ⌢ z)
    by (metis mult-left-one mult-left-top mult-right-dist-add meet.add-right-zero vector-zero-meet-comp add-left-dist-meet)
  thus meet-distributive x
    by (metis meet-distributive-def)
qed

lemma co-test-add-distributive: co-test x ⟶ add-distributive x
proof
  assume co-test x
  hence 1: x = x ; 0 + 1
    by (metis co-test-def)
  hence ∀ y z . x ; (y + z) = x ; y + x ; z
   by (metis add-associative add-commutative add-idempotent mult-left-one mult-left-top mult-right-dist-add meet.add-right-zero
vector-zero-meet-comp)
  thus add-distributive x
    by (metis add-distributive-def)
qed

lemma co-test-add-dist-contact: co-test x ⟶ add-dist-contact x
  by (metis co-test-add-distributive add-dist-contact-def co-test-contact)

end

end
```

20 NAlgebra

theory *NAlgebra*

imports *LatticeOrderedSemiring*

begin

class *left-n-algebra* = *bounded-idempotent-left-semiring* + *bounded-distributive-lattice* + *n* + *L* +
  assumes *n-dist-n-add* : $n(x) + n(y) = n(n(x) ; T + y)$
  assumes *n-export* : $n(x) ; n(y) = n(n(x) ; y)$
  assumes *n-isotone-idempotent*: $n(x) ; n(x + y) = n(x)$
  assumes *n-sub-nL-meet-one* : $n(x) \leq n(L) \frown 1$
  assumes *n-L-decreasing* : $n(x) ; L \leq x$
  assumes *n-nL-semi-commute* : $n(L) ; x \leq x ; n(L)$
  assumes *n-nL-meet-L-nL0* : $n(L) ; x = (x \frown L) + n(L ; 0) ; x$
  assumes *n-L-split-n-L-L* : $x ; L = x ; 0 + n(x ; L) ; L$
  assumes *n-n-top-split-n-top* : $x ; n(y) ; T \leq x ; 0 + n(x ; y) ; T$
  assumes *n-top-meet-L-below-L*: $x ; T ; y \frown L \leq x ; L ; y$

begin

subclass *lattice-ordered-pre-left-semiring* **..**

— Theorem 25.6

lemma *n-sub-one*: $n(x) \leq 1$
  by (*metis meet.add-least-upper-bound n-sub-nL-meet-one*)

— Theorem 25.2

lemma *n-mult-idempotent* : $n(x) ; n(x) = n(x)$
  by (*metis add-idempotent n-isotone-idempotent*)

lemma *n-L-increasing*: $n(x) \leq n(n(x) ; L)$
  by (*smt meet.add-least-upper-bound mult-right-isotone n-export n-mult-idempotent n-sub-nL-meet-one*)

— Theorem 25.35

lemma *meet-L-below-n-L*: $x \frown L \leq n(L) ; x$
  by (*metis add-left-divisibility n-nL-meet-L-nL0*)

— Theorem 25.30

lemma *n-vector-meet-L*: $x ; T \frown L \leq x ; L$
  by (*metis mult-right-one n-top-meet-L-below-L*)

— Theorem 25.7

lemma *n-mult-right-zero*: $n(x) ; 0 = 0$
  by (*metis antisym mult-left-isotone mult-left-one n-sub-one zero-least*)

lemma *n-mult-left-absorb-add*: $n(x) ; (n(x) + n(y)) = n(x)$
  by (*metis add-commutative n-dist-n-add n-isotone-idempotent*)

lemma *n-mult-right-absorb-add*: $(n(x) + n(y)) ; n(y) = n(y)$
  by (*metis less-eq-def mult-left-one mult-right-dist-add n-mult-idempotent n-sub-one*)

lemma *n-add-left-absorb-mult*: $n(x) + n(x) ; n(y) = n(x)$
  by (*metis add-commutative less-eq-def mult-left-sub-dist-add-right n-mult-left-absorb-add*)

lemma *n-add-right-absorb-mult*: $n(x) ; n(y) + n(y) = n(y)$
  by (*metis less-eq-def mult-left-one mult-right-dist-add n-sub-one*)

— Theorem 25.1

lemma *n-mult-commutative*: $n(x) ; n(y) = n(y) ; n(x)$
  by (*metis add-commutative mult-associative n-add-left-absorb-mult n-add-right-absorb-mult n-export n-mult-left-absorb-add n-mult-right-absorb-add*)

lemma *n-add-right-dist-mult*: $n(x) ; n(y) + n(z) = (n(x) + n(z)) ; (n(y) + n(z))$
by (*smt add-associative mult-right-dist-add n-dist-n-add n-mult-commutative n-mult-idempotent n-mult-right-absorb-add*)

lemma *n-add-left-dist-mult*: $n(x) + n(y) ; n(z) = (n(x) + n(y)) ; (n(x) + n(z))$
by (*metis add-commutative n-add-right-dist-mult*)

lemma *n-order*: $n(x) \leq n(y) \longleftrightarrow n(x) ; n(y) = n(x)$
by (*metis less-eq-def n-add-right-absorb-mult n-mult-left-absorb-add*)

— Theorem 25.3

lemma *n-mult-left-lower-bound*: $n(x) ; n(y) \leq n(x)$
by (*metis add-right-upper-bound n-add-left-absorb-mult*)

— Theorem 25.4

lemma *n-mult-right-lower-bound*: $n(x) ; n(y) \leq n(y)$
by (*metis add-commutative add-right-upper-bound n-add-right-absorb-mult*)

lemma *n-mult-least-upper-bound*: $n(x) \leq n(y) \wedge n(x) \leq n(z) \longleftrightarrow n(x) \leq n(y) ; n(z)$
by (*metis mult-left-isotone n-mult-left-lower-bound n-mult-right-lower-bound n-order order-trans*)

lemma *n-mult-left-divisibility*: $n(x) \leq n(y) \longleftrightarrow (\exists z . n(x) = n(y) ; n(z))$
by (*metis antisym mult-left-isotone n-mult-idempotent n-mult-left-lower-bound n-mult-right-lower-bound*)

lemma *n-mult-right-divisibility*: $n(x) \leq n(y) \longleftrightarrow (\exists z . n(x) = n(z) ; n(y))$
by (*metis n-mult-right-lower-bound n-order*)

— Theorem 25.5

lemma *n-left-upper-bound*: $n(x) \leq n(x + y)$
by (*metis n-isotone-idempotent n-order*)

lemma *n-right-upper-bound*: $n(x) \leq n(y + x)$
by (*metis add-commutative n-left-upper-bound*)

— Theorem 25

lemma *n-isotone*: $x \leq y \longrightarrow n(x) \leq n(y)$
by (*metis less-eq-def n-left-upper-bound*)

lemma *n-add-left-zero*: $n(0) + n(x) = n(x)$
by (*metis less-eq-def n-isotone zero-least*)

lemma *n-mult-left-zero*: $n(0) ; n(x) = n(0)$
by (*metis add-left-zero n-isotone-idempotent*)

— Theorem 25.8

lemma *n-mult-right-zero-n*: $n(x) ; n(0) = n(0)$
by (*metis add-commutative n-add-left-zero n-mult-right-absorb-add*)

lemma *n-mult-left-one*: $n(T) ; n(x) = n(x)$
by (*metis add-commutative add-right-top n-dist-n-add n-mult-right-absorb-add*)

lemma *n-mult-right-one*: $n(x) ; n(T) = n(x)$
by (*metis add-right-top n-isotone-idempotent*)

lemma *n-add-left-top*: $n(T) + n(x) = n(T)$
by (*metis n-add-left-absorb-mult n-mult-left-one*)

lemma *n-mult-left-dist-add*: $n(x) ; (n(y) + n(z)) = (n(x) ; n(y)) + (n(x) ; n(z))$
by (*metis mult-right-dist-add n-dist-n-add n-mult-commutative*)

— Theorem 25.11

lemma *n-n-L*: $n(n(x) ; L) = n(x)$
by (*metis antisym n-L-decreasing n-L-increasing n-isotone*)

— Theorem 25.40

lemma *n-galois*: $n(x) \leq n(y) \longleftrightarrow n(x)\ ;\ L \leq y$
  by (*metis mult-left-isotone n-L-decreasing n-L-increasing n-isotone order-trans*)

— Theorem 25.10

lemma *n-add-n-top*: $n(x + n(x)\ ;\ T) = n(x)$
  by (*metis add-commutative add-idempotent n-dist-n-add*)

— Theorem 25.37

lemma *n-less-eq-char-n*: $x \leq y \longleftrightarrow x \leq y + L \wedge n(L)\ ;\ x \leq y + n(y)\ ;\ T$
proof
  assume $x \leq y$
  thus $x \leq y + L \wedge n(L)\ ;\ x \leq y + n(y)\ ;\ T$
    by (*metis meet-less-eq-def meet-absorb mult-isotone mult-left-one n-sub-one order-trans*)
next
  assume *1*: $x \leq y + L \wedge n(L)\ ;\ x \leq y + n(y)\ ;\ T$
  hence $x \leq y + (x \frown L)$
    by (*metis meet-less-eq-def add-left-dist-meet add-right-upper-bound meet.add-left-isotone*)
  also have ... $\leq y + n(y)\ ;\ T$ **using** *1*
    by (*metis add-least-upper-bound add-left-upper-bound meet-L-below-n-L order-trans*)
  finally have $x \leq y + (L \frown n(y)\ ;\ T)$ **using** *1*
    by (*metis meet.add-least-upper-bound add-left-dist-meet*)
  thus $x \leq y$
    by (*metis add-idempotent add-least-upper-bound n-vector-meet-L less-eq-def meet-commutative n-L-decreasing*)
qed

— Theorem 25.39

lemma *n-preserves-equation*: $n(y)\ ;\ x \leq x\ ;\ n(y) \longleftrightarrow n(y)\ ;\ x = n(y)\ ;\ x\ ;\ n(y)$
  by (*metis n-mult-idempotent n-sub-one order-refl test-preserves-equation*)

— Theorem 25.35

lemma *n-L-decreasing-meet-L*: $n(x)\ ;\ L \leq x \frown L$
  by (*metis add-commutative meet-less-eq-def meet-absorb meet-commutative meet.add-least-upper-bound mult-right-sub-dist-add-left n-L-decreasing n-add-left-top n-sub-nL-meet-one*)

— Theorem 25.13

lemma *n-sub-nL*: $n(x) \leq n(L)$
  by (*metis meet.add-least-upper-bound n-sub-nL-meet-one*)

— Theorem 25.16

lemma *n-zero-L-zero*: $n(0)\ ;\ L = 0$
  by (*metis antisym n-L-decreasing zero-least*)

lemma *n-L-top-below-L*: $L\ ;\ T \leq L$
proof −
  have $n(L\ ;\ 0)\ ;\ L\ ;\ T \leq L\ ;\ 0$
    by (*metis mult-associative mult-left-isotone n-L-decreasing vector-def zero-vector*)
  hence $n(L\ ;\ 0)\ ;\ L\ ;\ T \leq L$
    by (*metis order-trans zero-right-mult-decreasing*)
  hence $n(L)\ ;\ L\ ;\ T \leq L$
    by (*metis add-least-upper-bound meet.add-right-upper-bound mult-associative n-nL-meet-L-nL0*)
  thus $L\ ;\ T \leq L$
    by (*metis add-least-upper-bound eq-iff meet-idempotent n-nL-meet-L-nL0 top-right-mult-increasing*)
qed

— Theorem 25.27

lemma *n-L-top-L*: $L\ ;\ T = L$
  by (*metis antisym n-L-top-below-L top-right-mult-increasing*)

— Theorem 25.28

lemma *n-L-below-L*: $L ; x \leq L$
  by (*metis add-right-top n-L-top-below-L mult-left-sub-dist-add-left order-trans*)

— Theorem 25.29

lemma *n-L-split-L*: $x ; L \leq x ; 0 + L$
  by (*metis add-commutative add-left-isotone meet.add-least-upper-bound n-L-decreasing-meet-L n-L-split-n-L-L*)

— Theorem 25.21

lemma *n-split-top*: $x ; n(y) ; T \leq x ; y + n(x ; y) ; T$
proof –
  have $x ; 0 + n(x ; y) ; T \leq x ; y + n(x ; y) ; T$
    by (*metis add-left-isotone mult-right-isotone zero-least*)
  thus *?thesis*
    by (*smt n-n-top-split-n-top order-trans*)
qed

— Theorem 25.22

lemma *n-top-split*: $n(x) ; T ; y \leq x ; y + n(x ; y) ; T$
proof –
  have $n(x) ; T ; y \frown L \leq x ; y$
    by (*metis mult-left-isotone n-L-decreasing n-top-meet-L-below-L order-trans*)
  thus *?thesis*
    by (*smt add-isotone mult-associative mult-isotone n-L-decreasing n-export n-isotone n-mult-commutative n-nL-meet-L-nL0 n-n-L top-greatest zero-least*)
qed

— Theorem 25.17

lemma *n-nL-nT*: $n(L) = n(T)$
  by (*metis antisym n-isotone n-sub-nL top-greatest*)

— Theorem 25.27

lemma *n-L-L-L*: $L ; L = L$
  by (*metis antisym n-vector-meet-L n-L-below-L meet-idempotent n-L-top-L*)

— Theorem 25.27

lemma *n-L-top-L-L*: $L ; T ; L = L$
  by (*metis n-L-L-L n-L-top-L*)

lemma *n-L-below-nL-top*: $L \leq n(L) ; T$
  by (*metis meet-L-below-n-L meet-left-top*)

— Theorem 25.12

lemma *n-n-nL*: $n(x) = n(x) ; n(L)$
  by (*metis n-order n-sub-nL*)

— Theorem 25.20

lemma *n-n-L-n*: $n(x ; n(y) ; L) \leq n(x ; y)$
  by (*metis mult-associative mult-right-isotone n-L-decreasing n-isotone*)

— Theorem 25.26

lemma *n-L-nL-L*: $L ; n(L) = L$
  by (*metis antisym n-vector-meet-L n-L-below-L meet-less-eq-def meet-commutative n-L-below-nL-top n-nL-semi-commute order-trans*)

lemma *n-L-nT-L*: $L ; n(T) = L$
  by (*metis n-L-nL-L n-nL-nT*)

— Theorem 25.26

lemma *n-L-L*: $n(L) ; L = L$

by (*metis add-left-zero mult-left-one n-L-split-n-L-L*)

lemma *n-top-L*: $n(T)$; $L = L$
by (*metis n-L-L n-nL-nT*)

— Theorem 25.24

lemma *n-n-L-split-n-L*: x ; $n(y)$; $L \le x$; $0 + n(x$; $y)$; L
by (*metis add-right-isotone n-L-top-below-L mult-associative mult-isotone n-L-split-n-L-L n-L-top-L n-mult-right-zero n-n-L-n*)

— Theorem 25.23

lemma *n-n-L-split-n-n-L-L*: x ; $n(y)$; $L = x$; $0 + n(x$; $n(y)$; $L)$; L
by (*metis mult-associative n-L-split-n-L-L n-mult-right-zero*)

— Theorem 25.25

lemma *n-nL-split-n-L-top*: $n(L)$; $x \le x$; $0 + n(x$; $L)$; T
by (*metis n-nL-semi-commute n-n-top-split-n-top order-trans top-right-mult-increasing*)

— Theorem 25.38

lemma *n-less-eq-char*: $x \le y \longleftrightarrow x \le y + L \wedge x \le y + n(y)$; T
by (*smt add-absorb add-associative add-idempotent n-less-eq-char-n meet-less-eq-def meet-left-dist-add n-add-n-top*)

— Theorem 25.32

lemma *n-top-meet-L-split-L*: x ; T ; $y \frown L \le x$; $0 + L$; y
proof −
  have x ; T ; $y \frown L \le x$; $0 + n(x$; $L)$; L ; y
    by (*smt n-top-meet-L-below-L mult-associative n-L-L-L n-L-split-n-L-L mult-right-dist-add mult-left-zero*)
  thus *?thesis*
    by (*metis n-sub-one mult-left-isotone add-right-isotone mult-left-one order-trans*)
qed

— Theorem 25.31

lemma *n-top-meet-L-L-meet-L*: x ; T ; $y \frown L = x$; L ; $y \frown L$
  apply (*rule antisym*)
  apply (*metis meet.add-least-upper-bound meet.add-right-upper-bound n-top-meet-L-below-L*)
  apply (*metis meet.add-left-isotone mult-left-isotone mult-right-isotone top-greatest*)
  done

lemma *n-n-top-below-n-L*: $n(x$; $T) \le n(x$; $L)$
by (*metis n-vector-meet-L n-L-decreasing-meet-L n-galois order-trans*)

— Theorem 25.18

lemma *n-n-top-n-L*: $n(x$; $T) = n(x$; $L)$
by (*metis antisym mult-right-isotone n-isotone n-n-top-below-n-L top-greatest*)

— Theorem 25.33

lemma *n-meet-L-0-below-0-meet-L*: $(x \frown L)$; $0 \le x$; $0 \frown L$
by (*metis meet.add-least-upper-bound mult-isotone order.refl zero-right-mult-decreasing*)

— Theorem 25.15

lemma *n-n-L-below-L*: $n(x)$; $L \le x$; L
by (*metis mult-associative mult-left-isotone n-L-L-L n-L-decreasing*)

lemma *n-n-L-below-n-L-L*: $n(x)$; $L \le n(x$; $L)$; L
by (*metis n-n-L-below-L mult-left-isotone n-galois*)

— Theorem 25.14

lemma *n-below-n-L*: $n(x) \le n(x$; $L)$
by (*metis n-n-L-below-L n-galois*)

— Theorem 25.34

lemma *n-n-meet-L-n-zero*: $n(x) = n(x) \frown L + n(x\ ;\ 0)$
  apply (*rule antisym*)
    apply (*smt add-right-isotone mult-associative mult-left-isotone n-L-decreasing n-export n-isotone n-mult-commutative n-nL-meet-L-nL0 n-n-L*)
  apply (*metis add-least-upper-bound meet.add-left-upper-bound n-isotone zero-right-mult-decreasing*)
  done

lemma *n-meet-L-below*: $n(x) \frown L \leq x$
  by (*metis meet.add-left-isotone n-L-decreasing n-vector-meet-L order-trans top-right-mult-increasing*)

— Theorem 25.9

lemma *n-below-n-zero*: $n(x) \leq x + n(x\ ;\ 0)$
  by (*metis add-left-isotone n-meet-L-below n-n-meet-L-n-zero*)

— Theorem 25.36

lemma *n-meet-L-top-below-n-L*: $(n(x) \frown L)\ ;\ T \leq n(x)\ ;\ L$
proof −
  have $(n(x) \frown L)\ ;\ T \leq n(x)\ ;\ T \frown L\ ;\ T$
    by (*metis meet.add-least-upper-bound meet.add-left-upper-bound meet-commutative mult-left-isotone*)
  thus *?thesis*
    by (*metis n-L-top-L n-vector-meet-L order-trans*)
qed

— Theorem 25.36

lemma *n-meet-L-top-below*: $(n(x) \frown L)\ ;\ T \leq x$
  by (*metis n-L-decreasing n-meet-L-top-below-n-L order-trans*)

— Theorem 25.34

lemma *n-n-meet-L*: $n(x) = n(x \frown L)$
  by (*metis add-absorb add-commutative less-eq-def n-L-decreasing-meet-L n-n-L n-right-upper-bound*)

— Theorem 25.19

lemma *n-n-top-split-n-L-n-zero-top*: $n(x)\ ;\ T = n(x)\ ;\ L + n(x\ ;\ 0)\ ;\ T$
  apply (*rule antisym*)
  apply (*metis add-left-isotone mult-right-dist-add n-meet-L-top-below-n-L n-n-meet-L-n-zero*)
  apply (*metis add-least-upper-bound mult-left-isotone mult-right-isotone n-isotone top-greatest zero-right-mult-decreasing*)
  done

end

typedef *'a nImage* = { *x*::*'a*::*left-n-algebra* . ($\exists$ *y*::*'a* . $x = n(y)$) }
  by *auto*

lemma *simp-nImage* [*simp*]: $\exists$ *y* . *Rep-nImage x* = $n(y)$
  using *Rep-nImage*
  by *simp*

setup-lifting *type-definition-nImage*

— Theorem 25

instantiation *nImage* :: (*left-n-algebra*) *bounded-idempotent-semiring*

begin

lift-definition *plus-nImage* :: *'a nImage* ⇒ *'a nImage* ⇒ *'a nImage* **is** *plus*
  by (*metis n-dist-n-add*)

lift-definition *times-nImage* :: *'a nImage* ⇒ *'a nImage* ⇒ *'a nImage* **is** *times*
  by (*metis n-export*)

lift-definition *zero-nImage* :: *'a nImage* **is** *n(0)*
  by *metis*

lift-definition *one-nImage* :: *'a nImage* **is** *n(T)*
  by *metis*

lift-definition *T-nImage* :: *'a nImage* **is** *n(T)*
  by *metis*

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

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

instance
  apply *intro-classes*
  apply (*metis* (*mono-tags*) *Rep-nImage-inject add-associative plus-nImage.rep-eq*)
  apply (*metis* (*mono-tags*) *Rep-nImage-inject add-commutative plus-nImage.rep-eq*)
  apply (*metis* (*mono-tags*) *Rep-nImage-inject add-idempotent plus-nImage.rep-eq*)
  apply (*metis* (*mono-tags*) *Rep-nImage-inject less-eq-def less-eq-nImage.rep-eq plus-nImage.rep-eq*)
  apply (*metis less-eq-nImage.rep-eq less-nImage.rep-eq less-def*)
  apply (*smt2 plus-nImage.rep-eq Rep-nImage-inverse n-add-left-zero simp-nImage zero-nImage.rep-eq*)
  apply (*metis* (*mono-tags*) *less-eq-nImage.rep-eq plus-nImage.rep-eq times-nImage.rep-eq mult-left-sub-dist-add*)
  apply (*metis* (*mono-tags*) *plus-nImage.rep-eq times-nImage.rep-eq Rep-nImage-inject mult-right-dist-add*)
  apply (*smt2 times-nImage.rep-eq Rep-nImage-inverse n-mult-left-zero zero-nImage.rep-eq simp-nImage*)
  apply (*smt2 one-nImage.rep-eq times-nImage.rep-eq Rep-nImage-inverse n-mult-left-one simp-nImage*)
  apply (*smt2 one-nImage.rep-eq eq-refl less-eq-nImage.rep-eq n-nL-nT n-n-nL times-nImage.rep-eq simp-nImage*)
  apply (*metis* (*mono-tags*) *less-eq-nImage.rep-eq times-nImage.rep-eq mult-associative order-refl*)
  apply (*smt2 T-nImage.rep-eq plus-nImage.rep-eq Rep-nImage-inverse add-commutative n-add-left-top simp-nImage*)
  apply (*metis* (*mono-tags*) *Rep-nImage-inject mult-associative times-nImage.rep-eq*)
  apply (*smt2 times-nImage.rep-eq Rep-nImage-inverse n-mult-right-one one-nImage.rep-eq simp-nImage*)
  apply (*metis* (*mono-tags*) *Rep-nImage-inject n-mult-left-dist-add plus-nImage.rep-eq times-nImage.rep-eq simp-nImage*)
  apply (*smt2 times-nImage.rep-eq Rep-nImage-inverse n-export n-mult-right-zero zero-nImage.rep-eq simp-nImage*)
  done

end

— Theorem 25

instantiation *nImage* :: (*left-n-algebra*) *bounded-distributive-lattice*

begin

lift-definition *meet-nImage* :: *'a nImage* ⇒ *'a nImage* ⇒ *'a nImage* **is** *times*
  by (*metis n-export*)

instance
  apply *intro-classes*
  apply (*metis* (*mono-tags*) *Rep-nImage-inject meet-nImage.rep-eq mult-associative*)
  apply (*metis* (*mono-tags*) *meet-nImage.rep-eq Rep-nImage-inverse simp-nImage n-mult-commutative*)
  apply (*metis* (*mono-tags*) *meet-nImage.rep-eq Rep-nImage-inverse simp-nImage n-mult-idempotent*)
  apply (*metis* (*mono-tags*) *meet-nImage.rep-eq Rep-nImage-inverse simp-nImage n-order less-eq-nImage.rep-eq*)
  apply (*metis less-def*)
  apply (*metis* (*mono-tags*) *T-nImage.abs-eq meet-nImage-def mult-left-one-1 one-nImage.abs-eq times-nImage-def*)
  apply (*metis meet-nImage-def mult-left-dist-add times-nImage-def*)
  apply (*metis* (*mono-tags*) *Rep-nImage-inject meet-nImage.rep-eq n-add-left-dist-mult plus-nImage.rep-eq simp-nImage*)
  apply (*metis* (*mono-tags*) *Rep-nImage-inject meet-nImage.rep-eq n-mult-left-absorb-add plus-nImage.rep-eq simp-nImage*)
  apply (*metis* (*mono-tags*) *Rep-nImage-inject meet-nImage.rep-eq n-add-left-absorb-mult plus-nImage.rep-eq simp-nImage*)
  done

end

class *n-algebra* = *left-n-algebra* + *idempotent-left-zero-semiring*

begin

— Theorem 25 counterexamples

lemma *n-zero*: *n(0)* = *0* **nitpick** [*expect=genuine*] **oops**
lemma *n-one*: *n(1)* = *0* **nitpick** [*expect=genuine*] **oops**

lemma *n-nL-one*: $n(L) = 1$ **nitpick** [*expect=genuine*] **oops**
lemma *n-nT-one*: $n(T) = 1$ **nitpick** [*expect=genuine*] **oops**
lemma *n-n-zero*: $n(x) = n(x \; ; \; 0)$ **nitpick** [*expect=genuine*] **oops**
lemma *n-dist-add*: $n(x) + n(y) = n(x + y)$ **nitpick** [*expect=genuine*] **oops**
lemma *n-L-split*: $x \; ; \; n(y) \; ; \; L = x \; ; \; 0 + n(x \; ; \; y) \; ; \; L$ **nitpick** [*expect=genuine*] **oops**
lemma *n-split*: $x \le x \; ; \; 0 + n(x \; ; \; L) \; ; \; T$ **nitpick** [*expect=genuine*] **oops**
lemma *n-mult-top-1*: $n(x \; ; \; y) \le n(x \; ; \; n(y) \; ; \; T)$ **nitpick** [*expect=genuine*] **oops**
lemma *l91-1*: $n(L) \; ; \; x \le n(x \; ; \; T) \; ; \; T$ **nitpick** [*expect=genuine*] **oops**
lemma *meet-domain-top*: $x \frown n(y) \; ; \; T = n(y) \; ; \; x$ **nitpick** [*expect=genuine*] **oops**
lemma *meet-domain-2*: $x \frown n(y) \; ; \; T \le n(L) \; ; \; x$ **nitpick** [*expect=genuine*] **oops**
lemma *n-nL-top-n-top-meet-L-top-2*: $n(L) \; ; \; x \; ; \; T \le n(x \; ; \; T \frown L) \; ; \; T$ **nitpick** [*expect=genuine*] **oops**
lemma *n-nL-top-n-top-meet-L-top-1*: $n(x \; ; \; T \frown L) \; ; \; T \le n(L) \; ; \; x \; ; \; T$ **nitpick** [*expect=genuine*] **oops**
lemma *l9*: $x \; ; \; 0 \frown L \le n(x \; ; \; L) \; ; \; L$ **nitpick** [*expect=genuine*] **oops**
lemma *l18-2*: $n(x \; ; \; L) \; ; \; L \le n(x) \; ; \; L$ **nitpick** [*expect=genuine*] **oops**
lemma *l51-1*: $n(x) \; ; \; L \le (x \frown L) \; ; \; 0$ **nitpick** [*expect=genuine*] **oops**
lemma *l51-2*: $(x \frown L) \; ; \; 0 \le n(x) \; ; \; L$ **nitpick** [*expect=genuine*] **oops**

lemma *n-split-equal*: $x + n(x \; ; \; L) \; ; \; T = x \; ; \; 0 + n(x \; ; \; L) \; ; \; T$ **nitpick** [*expect=genuine*] **oops**
lemma *n-split-top*: $x \; ; \; T \le x \; ; \; 0 + n(x \; ; \; L) \; ; \; T$ **nitpick** [*expect=genuine*] **oops**
lemma *n-mult*: $n(x \; ; \; n(y) \; ; \; L) = n(x \; ; \; y)$ **nitpick** [*expect=genuine*] **oops**
lemma *n-mult-1*: $n(x \; ; \; y) \le n(x \; ; \; n(y) \; ; \; L)$ **nitpick** [*expect=genuine*] **oops**
lemma *n-mult-top*: $n(x \; ; \; n(y) \; ; \; T) = n(x \; ; \; y)$ **nitpick** [*expect=genuine*] **oops**
lemma *n-mult-right-upper-bound*: $n(x \; ; \; y) \le n(z) \longleftrightarrow n(x) \le n(z) \wedge x \; ; \; n(y) \; ; \; L \le x \; ; \; 0 + n(z) \; ; \; L$ **nitpick** [*expect=genuine*] **oops**
lemma *meet-domain*: $x \frown n(y) \; ; \; z = n(y) \; ; \; (x \frown z)$ **nitpick** [*expect=genuine*] **oops**
lemma *meet-domain-1*: $x \frown n(y) \; ; \; z \le n(y) \; ; \; x$ **nitpick** [*expect=genuine*] **oops**
lemma *meet-domain-top-3*: $x \frown n(y) \; ; \; T \le n(y) \; ; \; x$ **nitpick** [*expect=genuine*] **oops**
lemma *n-n-top-n-top-split-n-n-top-top*: $n(x) \; ; \; T + x \; ; \; n(y) \; ; \; T = x \; ; \; 0 + n(x \; ; \; n(y) \; ; \; T) \; ; \; T$ **nitpick** [*expect=genuine*] **oops**
lemma *n-n-top-n-top-split-n-n-top-top-1*: $x \; ; \; 0 + n(x \; ; \; n(y) \; ; \; T) \; ; \; T \le n(x) \; ; \; T + x \; ; \; n(y) \; ; \; T$ **nitpick** [*expect=genuine*] **oops**
lemma *n-n-top-n-top-split-n-n-top-top-2*: $n(x) \; ; \; T + x \; ; \; n(y) \; ; \; T \le x \; ; \; 0 + n(x \; ; \; n(y) \; ; \; T) \; ; \; T$ **nitpick** [*expect=genuine*] **oops**
lemma *n-nL-top-n-top-meet-L-top*: $n(L) \; ; \; x \; ; \; T = n(x \; ; \; T \frown L) \; ; \; T$ **nitpick** [*expect=genuine*] **oops**
lemma *l18*: $n(x) \; ; \; L = n(x \; ; \; L) \; ; \; L$ **nitpick** [*expect=genuine*] **oops**
lemma *l22*: $x \; ; \; 0 \frown L = n(x) \; ; \; L$ **nitpick** [*expect=genuine*] **oops**
lemma *l22-1*: $x \; ; \; 0 \frown L = n(x \; ; \; L) \; ; \; L$ **nitpick** [*expect=genuine*] **oops**
lemma *l22-2*: $x \frown L = n(x) \; ; \; L$ **nitpick** [*expect=genuine*] **oops**
lemma *l22-3*: $x \frown L = n(x \; ; \; L) \; ; \; L$ **nitpick** [*expect=genuine*] **oops**
lemma *l22-4*: $x \frown L \le n(x) \; ; \; L$ **nitpick** [*expect=genuine*] **oops**
lemma *l22-5*: $x \; ; \; 0 \frown L \le n(x) \; ; \; L$ **nitpick** [*expect=genuine*] **oops**
lemma *l23*: $x \; ; \; T \frown L = n(x) \; ; \; L$ **nitpick** [*expect=genuine*] **oops**
lemma *l51*: $n(x) \; ; \; L = (x \frown L) \; ; \; 0$ **nitpick** [*expect=genuine*] **oops**
lemma *l91*: $x = x \; ; \; T \longrightarrow n(L) \; ; \; x \le n(x) \; ; \; T$ **nitpick** [*expect=genuine*] **oops**
lemma *l92*: $x = x \; ; \; T \longrightarrow n(L) \; ; \; x \le n(x \frown L) \; ; \; T$ **nitpick** [*expect=genuine*] **oops**
lemma $x \frown L \le n(x) \; ; \; T$ **nitpick** [*expect=genuine*] **oops**

lemma *n-meet-L-0-0-meet-L*: $(x \frown L) \; ; \; 0 = x \; ; \; 0 \frown L$ **oops**
lemma *n-meet-comp*: $n(x) \frown n(y) \le n(x) \; ; \; n(y)$ **oops**

end

end

21 Recursion

theory *Recursion*

imports *Approximation NAlgebra*

begin

class *n-algebra-apx* = *n-algebra* + *apx* +
  assumes *apx-def*: $x \sqsubseteq y \longleftrightarrow x \le y + L \wedge n(L) ; y \le x + n(x) ; T$

begin

lemma *apx-transitive-2*: $x \sqsubseteq y \wedge y \sqsubseteq z \longrightarrow x \sqsubseteq z$
proof
  assume *1*: $x \sqsubseteq y \wedge y \sqsubseteq z$
  hence $n(L) ; z \le n(L) ; y + n(L) ; n(y) ; T$
    by (*metis apx-def mult-associative mult-left-dist-add mult-right-isotone n-mult-idempotent*)
  also have ... $\le x + n(x) ; T + n(n(L) ; y) ; T$ **using** *1*
    by (*metis add-left-isotone apx-def n-export*)
  also have ... $\le x + n(x) ; T$ **using** *1*
    by (*metis add-associative add-idempotent add-right-isotone apx-def mult-left-isotone n-add-n-top n-isotone*)
  finally show $x \sqsubseteq z$ **using** *1*
    by (*smt add-associative add-commutative apx-def less-eq-def*)
qed

lemma *apx-meet-L*: $y \sqsubseteq x \longrightarrow x \frown L \le y \frown L$
proof
  assume *1*: $y \sqsubseteq x$
  have $n(L) ; (x \frown L) \le n(L) ; x \frown L$
    by (*metis eq-iff meet-L-below-n-L meet.add-least-upper-bound mult-left-sub-dist-meet-right n-L-decreasing*)
  also have ... $\le (y \frown L) + (n(y) ; T \frown L)$ **using** *1*
    by (*metis apx-def meet-commutative meet-left-dist-add meet.add-left-isotone*)
  also have ... $\le (y \frown L) + n(y \frown L) ; T$
    by (*metis add-least-upper-bound n-vector-meet-L meet.add-least-upper-bound n-L-decreasing order-refl order-trans*)
  finally show $x \frown L \le y \frown L$
    by (*metis n-less-eq-char-n less-eq-def meet.add-right-upper-bound*)
qed

— Theorem 26.1

subclass *apx-biorder*
  apply *unfold-locales*
  apply (*smt add-least-upper-bound add-left-upper-bound apx-def less-eq-def mult-left-one mult-right-dist-add n-sub-one*)
  apply (*metis add-same-context antisym apx-def apx-meet-L relative-equality*)
  apply (*metis apx-transitive-2*)
  done

lemma *add-apx-left-isotone-2*: $x \sqsubseteq y \longrightarrow x + z \sqsubseteq y + z$
proof
  assume *1*: $x \sqsubseteq y$
  hence *2*: $x + z \le y + z + L$
    by (*smt add-associative add-commutative add-left-isotone apx-def*)
  have $n(L) ; (y + z) = n(L) ; y + n(L) ; z$
    by (*metis mult-left-dist-add*)
  also have ... $\le n(L) ; y + z$
    by (*metis add-commutative add-least-upper-bound add-right-upper-bound n-sub-one mult-left-dist-add mult-left-isotone mult-left-one*)
  also have ... $\le x + n(x) ; T + z$ **using** *1*
    by (*metis add-left-isotone apx-def*)
  also have ... $\le x + z + n(x + z) ; T$
    by (*metis add-associative add-commutative add-right-isotone mult-left-isotone n-right-upper-bound*)
  finally show $x + z \sqsubseteq y + z$ **using** *2*
    by (*metis apx-def*)
qed

lemma *mult-apx-left-isotone-2*: $x \sqsubseteq y \longrightarrow x ; z \sqsubseteq y ; z$
proof
  assume *1*: $x \sqsubseteq y$

hence $x ; z \le y ; z + L ; z$
  by (*metis apx-def mult-left-isotone mult-right-dist-add*)
hence *2*: $x ; z \le y ; z + L$
  by (*metis add-commutative add-left-isotone n-L-below-L order-trans*)
have $n(L) ; y ; z \le x ; z + n(x) ; T ; z$ **using** *1*
  by (*metis apx-def mult-left-isotone mult-right-dist-add*)
also have ... $\le x ; z + n(x ; z) ; T$
  by (*metis add-least-upper-bound add-left-upper-bound n-top-split*)
finally show $x ; z \sqsubseteq y ; z$ **using** *2*
  by (*metis apx-def mult-associative*)
qed

lemma *mult-apx-right-isotone-2*: $x \sqsubseteq y \longrightarrow z ; x \sqsubseteq z ; y$
proof
assume *1*: $x \sqsubseteq y$
hence $z ; x \le z ; y + z ; L$
  by (*metis apx-def mult-left-dist-add mult-right-isotone*)
also have ... $\le z ; y + z ; 0 + L$
  by (*metis add-associative add-right-isotone n-L-split-L*)
finally have *2*: $z ; x \le z ; y + L$
  by (*metis add-right-zero mult-left-dist-add*)
have $n(L) ; z ; y \le z ; n(L) ; y$
  by (*metis n-nL-semi-commute mult-left-isotone*)
also have ... $\le z ; (x + n(x) ; T)$ **using** *1*
  by (*metis apx-def mult-associative mult-right-isotone*)
also have ... $= z ; x + z ; n(x) ; T$
  by (*metis mult-associative mult-left-dist-add*)
also have ... $\le z ; x + n(z ; x) ; T$
  by (*metis add-least-upper-bound add-left-upper-bound n-split-top*)
finally show $z ; x \sqsubseteq z ; y$ **using** *2*
  by (*metis apx-def mult-associative*)
qed

— Theorem 26.1 and Theorem 26.2

subclass *apx-semiring*
apply *unfold-locales*
apply (*metis add-least-upper-bound add-right-isotone add-right-upper-bound apx-def mult-right-isotone top-greatest*)
apply (*rule add-apx-left-isotone-2*)
apply (*rule mult-apx-left-isotone-2*)
apply (*rule mult-apx-right-isotone-2*)
done

— Theorem 26.2

lemma *meet-L-apx-isotone*: $x \sqsubseteq y \longrightarrow x \frown L \sqsubseteq y \frown L$
  by (*smt add-commutative add-idempotent add-left-dist-meet apx-def apx-meet-L n-less-eq-char-n meet-commutative meet.add-right-isotone*)

— Theorem 26.2

lemma *n-L-apx-isotone*: $x \sqsubseteq y \longrightarrow n(x) ; L \sqsubseteq n(y) ; L$
proof
assume $x \sqsubseteq y$
hence $n(L) ; n(y) ; L \le n(x) ; L + n(n(x) ; L) ; T$
  by (*metis add-left-upper-bound apx-def mult-left-isotone n-add-n-top n-export n-isotone order-trans*)
thus $n(x) ; L \sqsubseteq n(y) ; L$
  by (*metis apx-def less-eq-def meet.add-least-upper-bound mult-associative n-L-decreasing-meet-L*)
qed

— Theorem 27

definition *kappa-apx-meet* :: $('a \Rightarrow 'a) \Rightarrow$ *bool*
where *kappa-apx-meet f* $\longleftrightarrow$ *apx.has-least-fixpoint f* $\wedge$ *has-apx-meet* $(\mu\ f)\ (\nu\ f) \wedge \kappa\ f = \mu\ f \bigtriangleup \nu\ f$

definition *kappa-mu-nu* :: $('a \Rightarrow 'a) \Rightarrow$ *bool*
where *kappa-mu-nu f* $\longleftrightarrow$ *apx.has-least-fixpoint f* $\wedge\ \kappa\ f = \mu\ f + (\nu\ f \frown L)$

definition *nu-below-mu-nu* :: $('a \Rightarrow 'a) \Rightarrow$ *bool*

where *nu-below-mu-nu f* $\longleftrightarrow n(L) ; \nu f \le \mu f + (\nu f \frown L) + n(\nu f) ; T$

definition *nu-below-mu-nu-2* :: $('a \Rightarrow 'a) \Rightarrow$ *bool*
where *nu-below-mu-nu-2 f* $\longleftrightarrow n(L) ; \nu f \le \mu f + (\nu f \frown L) + n(\mu f + (\nu f \frown L)) ; T$

definition *mu-nu-apx-nu* :: $('a \Rightarrow 'a) \Rightarrow$ *bool*
where *mu-nu-apx-nu f* $\longleftrightarrow \mu f + (\nu f \frown L) \sqsubseteq \nu f$

definition *mu-nu-apx-meet* :: $('a \Rightarrow 'a) \Rightarrow$ *bool*
where *mu-nu-apx-meet f* $\longleftrightarrow$ *has-apx-meet* $(\mu f)\ (\nu f) \wedge \mu f \triangle \nu f = \mu f + (\nu f \frown L)$

definition *apx-meet-below-nu* :: $('a \Rightarrow 'a) \Rightarrow$ *bool*
where *apx-meet-below-nu f* $\longleftrightarrow$ *has-apx-meet* $(\mu f)\ (\nu f) \wedge \mu f \triangle \nu f \le \nu f$

lemma *mu-below-l*: $\mu f \le \mu f + (\nu f \frown L)$
by (*metis add-left-upper-bound*)

lemma *l-below-nu*: *has-least-fixpoint f* $\wedge$ *has-greatest-fixpoint f* $\longrightarrow \mu f + (\nu f \frown L) \le \nu f$
by (*metis add-least-upper-bound meet.add-left-upper-bound mu-below-nu*)

lemma *n-l-nu*: *has-least-fixpoint f* $\wedge$ *has-greatest-fixpoint f* $\longrightarrow (\mu f + (\nu f \frown L)) \frown L = \nu f \frown L$
by (*smt add-commutative add-left-dist-meet less-eq-def meet-absorb meet-associative meet-commutative mu-below-nu*)

lemma *l-apx-mu*: $\mu f + (\nu f \frown L) \sqsubseteq \mu f$
proof –
have *1*: $\mu f + (\nu f \frown L) \le \mu f + L$
by (*metis add-right-isotone meet.add-right-upper-bound*)
have *2*: $n(L) ; \mu f \le \mu f + (\nu f \frown L) + n(\mu f + (\nu f \frown L)) ; T$
by (*metis mult-left-isotone mult-left-one mult-left-sub-dist-add-left n-sub-one order-trans*)
thus *?thesis* **using** *1*
by (*metis apx-def*)
qed

— Theorem 27.4 implies Theorem 27.5

lemma *nu-below-mu-nu-nu-below-mu-nu-2*: *nu-below-mu-nu f* $\longrightarrow$ *nu-below-mu-nu-2 f*
proof
assume *1*: *nu-below-mu-nu f*
have $n(L) ; \nu f = n(L) ; (n(L) ; \nu f)$
by (*metis n-mult-idempotent mult-associative*)
also have ... $\le n(L) ; (\mu f + (\nu f \frown L) + n(\nu f) ; T)$ **using** *1*
by (*metis mult-right-isotone nu-below-mu-nu-def*)
also have ... $= n(L) ; (\mu f + (\nu f \frown L)) + n(L) ; n(\nu f) ; T$
by (*metis mult-associative mult-left-dist-add*)
also have ... $\le \mu f + (\nu f \frown L) + n(L) ; n(\nu f) ; T$
by (*metis add-left-isotone mult-left-isotone mult-left-one n-sub-one*)
also have ... $= \mu f + (\nu f \frown L) + n(n(\nu f) ; L) ; T$
by (*smt n-mult-commutative n-export*)
also have ... $\le \mu f + (\nu f \frown L) + n(\nu f \frown L) ; T$
by (*metis add-right-isotone mult-left-isotone n-L-decreasing-meet-L n-isotone*)
also have ... $\le \mu f + (\nu f \frown L) + n(\mu f + (\nu f \frown L)) ; T$
by (*metis add-right-isotone mult-left-isotone n-right-upper-bound*)
finally show *nu-below-mu-nu-2 f*
by (*metis nu-below-mu-nu-2-def*)
qed

— Theorem 27.5 implies Theorem 27.4

lemma *nu-below-mu-nu-2-nu-below-mu-nu*: *has-least-fixpoint f* $\wedge$ *has-greatest-fixpoint f* $\wedge$ *nu-below-mu-nu-2 f* $\longrightarrow$ *nu-below-mu-nu f*
proof
assume *1*: *has-least-fixpoint f* $\wedge$ *has-greatest-fixpoint f* $\wedge$ *nu-below-mu-nu-2 f*
hence $n(L) ; \nu f \le \mu f + (\nu f \frown L) + n(\mu f + (\nu f \frown L)) ; T$
by (*metis nu-below-mu-nu-2-def*)
also have ... $\le \mu f + (\nu f \frown L) + n(\nu f) ; T$ **using** *1*
by (*metis add-right-isotone l-below-nu mult-left-isotone n-isotone*)
finally show *nu-below-mu-nu f*
by (*metis nu-below-mu-nu-def*)
qed

lemma *nu-below-mu-nu-equivalent*: *has-least-fixpoint f* $\wedge$ *has-greatest-fixpoint f* $\longrightarrow$ (*nu-below-mu-nu f* $\longleftrightarrow$ *nu-below-mu-nu-2 f*)
  by (*metis nu-below-mu-nu-2-nu-below-mu-nu nu-below-mu-nu-nu-below-mu-nu-2*)

— Theorem 27.5 implies Theorem 27.6

lemma *nu-below-mu-nu-2-mu-nu-apx-nu*: *has-least-fixpoint f* $\wedge$ *has-greatest-fixpoint f* $\wedge$ *nu-below-mu-nu-2 f* $\longrightarrow$ *mu-nu-apx-nu f*

```
proof
  assume 1: has-least-fixpoint f ∧ has-greatest-fixpoint f ∧ nu-below-mu-nu-2 f
  hence $\mu\ f + (\nu\ f \frown L) \leq \nu\ f + L$
    by (metis add-commutative add-right-upper-bound l-below-nu order-trans)
  thus mu-nu-apx-nu f using 1
    by (metis apx-def mu-nu-apx-nu-def nu-below-mu-nu-2-def)
qed
```

— Theorem 27.6 implies Theorem 27.7

lemma *mu-nu-apx-nu-mu-nu-apx-meet*: *has-least-fixpoint f* $\wedge$ *has-greatest-fixpoint f* $\wedge$ *mu-nu-apx-nu f* $\longrightarrow$ *mu-nu-apx-meet f*

```
proof
  let $?l = \mu\ f + (\nu\ f \frown L)$
  assume has-least-fixpoint f ∧ has-greatest-fixpoint f ∧ mu-nu-apx-nu f
  hence is-apx-meet ($\mu$ f) ($\nu$ f) ?l
     by (smt add-apx-left-isotone add-commutative apx-meet-L is-apx-meet-def l-apx-mu less-eq-def meet.add-least-upper-bound
mu-nu-apx-nu-def)
  thus mu-nu-apx-meet f
    by (smt apx-meet-char mu-nu-apx-meet-def)
qed
```

— Theorem 27.7 implies Theorem 27.8

lemma *mu-nu-apx-meet-apx-meet-below-nu*: *has-least-fixpoint f* $\wedge$ *has-greatest-fixpoint f* $\wedge$ *mu-nu-apx-meet f* $\longrightarrow$ *apx-meet-below-nu f*
  by (*metis apx-meet-below-nu-def l-below-nu mu-nu-apx-meet-def*)

— Theorem 27.8 implies Theorem 27.5

lemma *apx-meet-below-nu-nu-below-mu-nu-2*: *apx-meet-below-nu f* $\longrightarrow$ *nu-below-mu-nu-2 f*

```
proof –
  let $?l = \mu\ f + (\nu\ f \frown L)$
  have $\forall m\ .\ m \sqsubseteq \mu\ f \wedge m \sqsubseteq \nu\ f \wedge m \leq \nu\ f \longrightarrow n(L)\ ;\ \nu\ f \leq\ ?l + n(?l)\ ;\ T$
  proof
    fix m
    show $m \sqsubseteq \mu\ f \wedge m \sqsubseteq \nu\ f \wedge m \leq \nu\ f \longrightarrow n(L)\ ;\ \nu\ f \leq\ ?l + n(?l)\ ;\ T$
    proof
      assume 1: $m \sqsubseteq \mu\ f \wedge m \sqsubseteq \nu\ f \wedge m \leq \nu\ f$
      hence $m \leq\ ?l$
        by (smt add-commutative add-left-dist-meet add-left-upper-bound apx-def meet-less-eq-def meet.add-least-upper-bound)
      hence $m + n(m)\ ;\ T \leq\ ?l + n(?l)\ ;\ T$
        by (metis add-isotone mult-left-isotone n-isotone)
      thus $n(L)\ ;\ \nu\ f \leq\ ?l + n(?l)\ ;\ T$ using 1
        by (smt apx-def order-trans)
    qed
  qed
  thus ?thesis
    by (smt apx-meet-below-nu-def apx-meet-same apx-meet-unique is-apx-meet-def nu-below-mu-nu-2-def)
qed
```

— Theorem 27.1 implies Theorem 27.2

lemma *has-apx-least-fixpoint-kappa-apx-meet*: *has-least-fixpoint f* $\wedge$ *has-greatest-fixpoint f* $\wedge$ *apx.has-least-fixpoint f* $\longrightarrow$ *kappa-apx-meet f*

```
proof
  assume 1: has-least-fixpoint f ∧ has-greatest-fixpoint f ∧ apx.has-least-fixpoint f
  hence 2: $\forall w\ .\ w \sqsubseteq \mu\ f \wedge w \sqsubseteq \nu\ f \longrightarrow n(L)\ ;\ \kappa\ f \leq w + n(w)\ ;\ T$
    by (metis apx-def mult-right-isotone order-trans kappa-below-nu)
  have $\forall w\ .\ w \sqsubseteq \mu\ f \wedge w \sqsubseteq \nu\ f \longrightarrow w \leq \kappa\ f + L$ using 1
    by (metis add-left-isotone apx-def mu-below-kappa order-trans)
  hence $\forall w\ .\ w \sqsubseteq \mu\ f \wedge w \sqsubseteq \nu\ f \longrightarrow w \sqsubseteq \kappa\ f$ using 2
```

```
    by (metis apx-def)
  hence is-apx-meet (μ f) (ν f) (κ f) using 1
    by (smt apx-meet-char is-apx-meet-def kappa-apx-below-mu kappa-apx-below-nu kappa-apx-meet-def)
  thus kappa-apx-meet f using 1
    by (metis apx-meet-char kappa-apx-meet-def)
qed
```

— Theorem 27.2 implies Theorem 27.8

```
lemma kappa-apx-meet-apx-meet-below-nu: has-greatest-fixpoint f ∧ kappa-apx-meet f ⟶ apx-meet-below-nu f
  by (metis apx-meet-below-nu-def kappa-apx-meet-def kappa-below-nu)
```

— Theorem 27.8 implies Theorem 27.3

```
lemma apx-meet-below-nu-kappa-mu-nu: has-least-fixpoint f ∧ has-greatest-fixpoint f ∧ isotone f ∧ apx.isotone f ∧
apx-meet-below-nu f ⟶ kappa-mu-nu f
proof
  let ?l = μ f + (ν f ⌢ L)
  let ?m = μ f △ ν f
  assume 1: has-least-fixpoint f ∧ has-greatest-fixpoint f ∧ isotone f ∧ apx.isotone f ∧ apx-meet-below-nu f
  hence 2: ?m = ?l
          by (metis apx-meet-below-nu-nu-below-mu-nu-2 mu-nu-apx-meet-def mu-nu-apx-nu-mu-nu-apx-meet
nu-below-mu-nu-2-mu-nu-apx-nu)
  have 3: ?l ≤ f(?l) + L
  proof -
    have ?l ≤ μ f + L
      by (metis add-right-isotone meet.add-right-upper-bound)
    also have ... = f(μ f) + L using 1
      by (metis is-least-fixpoint-def least-fixpoint)
    also have ... ≤ f(?l) + L using 1
      by (metis add-left-isotone add-left-upper-bound isotone-def)
    finally show ?l ≤ f(?l) + L
      by metis
  qed
  have n(L) ; f(?l) ≤ ?l + n(?l) ; T
  proof -
    have n(L) ; f(?l) ≤ n(L) ; f(ν f) using 1 2
      by (metis apx-meet-below-nu-def isotone-def mult-right-isotone)
    also have ... = n(L) ; ν f using 1
      by (metis greatest-fixpoint is-greatest-fixpoint-def)
    also have ... ≤ ?l + n(?l) ; T using 1
      by (metis apx-meet-below-nu-nu-below-mu-nu-2 nu-below-mu-nu-2-def)
    finally show n(L) ; f(?l) ≤ ?l + n(?l) ; T
      by metis
  qed
  hence 4: ?l ⊑ f(?l) using 3
    by (metis apx-def)
  have 5: f(?l) ⊑ μ f
  proof -
    have ?l ⊑ μ f
      by (metis l-apx-mu)
    thus f(?l) ⊑ μ f using 1
      by (metis apx.isotone-def is-least-fixpoint-def least-fixpoint)
  qed
  have 6: f(?l) ⊑ ν f
  proof -
    have ?l ⊑ ν f using 1 2
      by (metis apx-greatest-lower-bound apx-meet-below-nu-def apx-reflexive)
    thus f(?l) ⊑ ν f using 1
      by (metis apx.isotone-def greatest-fixpoint is-greatest-fixpoint-def)
  qed
  hence f(?l) ⊑ ?l using 1 2 5
    by (metis apx-greatest-lower-bound apx-meet-below-nu-def)
  hence 7: f(?l) = ?l using 4
    by (metis apx-antisymmetric)
  have ∀ y . f(y) = y ⟶ ?l ⊑ y
  proof
    fix y
    show f(y) = y ⟶ ?l ⊑ y
    proof
      assume 8: f(y) = y
      hence 9: ?l ≤ y + L using 1
        by (metis add-isotone is-least-fixpoint-def least-fixpoint meet.add-right-upper-bound)
      have y ≤ ν f using 1 8
        by (metis greatest-fixpoint is-greatest-fixpoint-def)
      hence n(L) ; y ≤ ?l + n(?l) ; T using 1 4 6
        by (smt apx-meet-below-nu-nu-below-mu-nu-2 mult-right-isotone nu-below-mu-nu-2-def order-trans)
      thus ?l ⊑ y using 9
        by (metis apx-def)
    qed
  qed
  thus kappa-mu-nu f using 1 2 7
    by (smt apx.least-fixpoint-same apx.has-least-fixpoint-def apx.is-least-fixpoint-def kappa-mu-nu-def)
qed
```

— Theorem 27.3 implies Theorem 27.1

```
lemma kappa-mu-nu-has-apx-least-fixpoint: kappa-mu-nu f ⟶ apx.has-least-fixpoint f
  by (metis kappa-mu-nu-def)
```

— Theorem 27.4 implies Theorem 27.3

```
lemma nu-below-mu-nu-kappa-mu-nu: has-least-fixpoint f ∧ has-greatest-fixpoint f ∧ isotone f ∧ apx.isotone f ∧
nu-below-mu-nu f ⟶ kappa-mu-nu f
    by (metis apx-meet-below-nu-kappa-mu-nu mu-nu-apx-meet-apx-meet-below-nu mu-nu-apx-nu-mu-nu-apx-meet
nu-below-mu-nu-nu-below-mu-nu-2 nu-below-mu-nu-2-mu-nu-apx-nu)
```

— Theorem 27.3 implies Theorem 27.4

```
lemma kappa-mu-nu-nu-below-mu-nu: has-least-fixpoint f ∧ has-greatest-fixpoint f ∧ kappa-mu-nu f ⟶ nu-below-mu-nu f
  by (metis apx-meet-below-nu-nu-below-mu-nu-2 has-apx-least-fixpoint-kappa-apx-meet nu-below-mu-nu-2-nu-below-mu-nu
kappa-apx-meet-apx-meet-below-nu kappa-mu-nu-has-apx-least-fixpoint)
```

— Theorem 28

```
definition kappa-mu-nu-L :: ('a ⇒ 'a) ⇒ bool
  where kappa-mu-nu-L f ⟷ apx.has-least-fixpoint f ∧ κ f = μ f + n(ν f) ; L

definition nu-below-mu-nu-L :: ('a ⇒ 'a) ⇒ bool
  where nu-below-mu-nu-L f ⟷ n(L) ; ν f ≤ μ f + n(ν f) ; T

definition mu-nu-apx-nu-L :: ('a ⇒ 'a) ⇒ bool
  where mu-nu-apx-nu-L f ⟷ μ f + n(ν f) ; L ⊑ ν f

definition mu-nu-apx-meet-L :: ('a ⇒ 'a) ⇒ bool
  where mu-nu-apx-meet-L f ⟷ has-apx-meet (μ f) (ν f) ∧ μ f △ ν f = μ f + n(ν f) ; L

lemma n-below-l: x + n(y) ; L ≤ x + (y ⌢ L)
  by (metis add-right-isotone n-L-decreasing-meet-L)

lemma n-equal-l: nu-below-mu-nu-L f ⟶ μ f + n(ν f) ; L = μ f + (ν f ⌢ L)
proof
  assume nu-below-mu-nu-L f
  hence ν f ⌢ L ≤ (μ f + n(ν f) ; T) ⌢ L
    by (smt meet-L-below-n-L meet.add-least-upper-bound meet.add-right-upper-bound nu-below-mu-nu-L-def order-trans)
  also have ... ≤ μ f + (n(ν f) ; T ⌢ L)
    by (metis add-left-dist-meet add-right-upper-bound meet.add-right-isotone)
  also have ... ≤ μ f + n(ν f) ; L
    by (metis add-right-isotone n-vector-meet-L)
  finally have μ f + (ν f ⌢ L) ≤ μ f + n(ν f) ; L
    by (metis add-least-upper-bound add-left-upper-bound)
  thus μ f + n(ν f) ; L = μ f + (ν f ⌢ L)
    by (metis antisym n-below-l)
qed
```

— Theorem 28.2 implies Theorem 27.4

```
lemma nu-below-mu-nu-L-nu-below-mu-nu: nu-below-mu-nu-L f ⟶ nu-below-mu-nu f
```

by (*metis add-associative add-right-top mult-left-dist-add n-equal-l nu-below-mu-nu-L-def nu-below-mu-nu-def*)

— Theorem 28.2 implies Theorem 28.1

lemma *nu-below-mu-nu-L-kappa-mu-nu-L*: *has-least-fixpoint f* $\wedge$ *has-greatest-fixpoint f* $\wedge$ *isotone f* $\wedge$ *apx.isotone f* $\wedge$ *nu-below-mu-nu-L f* $\longrightarrow$ *kappa-mu-nu-L f*
by (*metis n-equal-l nu-below-mu-nu-L-nu-below-mu-nu nu-below-mu-nu-kappa-mu-nu kappa-mu-nu-L-def kappa-mu-nu-def*)

— Theorem 28.2 implies Theorem 28.3

lemma *nu-below-mu-nu-L-mu-nu-apx-nu-L*: *has-least-fixpoint f* $\wedge$ *has-greatest-fixpoint f* $\wedge$ *nu-below-mu-nu-L f* $\longrightarrow$ *mu-nu-apx-nu-L f*
by (*metis mu-nu-apx-nu-L-def mu-nu-apx-nu-def n-equal-l nu-below-mu-nu-2-mu-nu-apx-nu nu-below-mu-nu-L-nu-below-mu-nu nu-below-mu-nu-nu-below-mu-nu-2*)

— Theorem 28.2 implies Theorem 28.4

lemma *nu-below-mu-nu-L-mu-nu-apx-meet-L*: *has-least-fixpoint f* $\wedge$ *has-greatest-fixpoint f* $\wedge$ *nu-below-mu-nu-L f* $\longrightarrow$ *mu-nu-apx-meet-L f*
by (*metis mu-nu-apx-meet-L-def mu-nu-apx-meet-def mu-nu-apx-nu-mu-nu-apx-meet n-equal-l nu-below-mu-nu-2-mu-nu-apx-nu nu-below-mu-nu-L-nu-below-mu-nu nu-below-mu-nu-nu-below-mu-nu-2*)

— Theorem 28.3 implies Theorem 28.2

lemma *mu-nu-apx-nu-L-nu-below-mu-nu-L*: *has-least-fixpoint f* $\wedge$ *has-greatest-fixpoint f* $\wedge$ *mu-nu-apx-nu-L f* $\longrightarrow$ *nu-below-mu-nu-L f*
proof
  let *?n* $= \mu\ f + n(\nu\ f)\ ;\ L$
  let *?l* $= \mu\ f + (\nu\ f \frown L)$
  assume *1*: *has-least-fixpoint f* $\wedge$ *has-greatest-fixpoint f* $\wedge$ *mu-nu-apx-nu-L f*
  hence $n(L)\ ;\ \nu\ f \le\ ?n + n(?n)\ ;\ T$
    by (*metis apx-def mu-nu-apx-nu-L-def*)
  also have ... $\le\ ?n + n(?l)\ ;\ T$
    by (*metis add-right-isotone n-isotone mult-left-isotone n-below-l*)
  also have ... $\le\ ?n + n(\nu\ f)\ ;\ T$ **using** *1*
    by (*metis add-right-isotone n-isotone l-below-nu mult-left-isotone*)
  finally show *nu-below-mu-nu-L f*
    by (*metis add-associative add-right-top mult-left-dist-add nu-below-mu-nu-L-def*)
qed

— Theorem 28.1 implies Theorem 28.3

lemma *kappa-mu-nu-L-mu-nu-apx-nu-L*: *has-greatest-fixpoint f* $\wedge$ *kappa-mu-nu-L f* $\longrightarrow$ *mu-nu-apx-nu-L f*
by (*metis mu-nu-apx-nu-L-def kappa-apx-below-nu kappa-mu-nu-L-def*)

— Theorem 28.4 implies Theorem 28.3

lemma *mu-nu-apx-meet-L-mu-nu-apx-nu-L*: *mu-nu-apx-meet-L f* $\longrightarrow$ *mu-nu-apx-nu-L f*
by (*smt apx-meet-same has-apx-meet-def is-apx-meet-def mu-nu-apx-meet-L-def mu-nu-apx-nu-L-def*)

— Theorem 28.1 implies Theorem 28.2

lemma *kappa-mu-nu-L-nu-below-mu-nu-L*: *has-least-fixpoint f* $\wedge$ *has-greatest-fixpoint f* $\wedge$ *kappa-mu-nu-L f* $\longrightarrow$ *nu-below-mu-nu-L f*
by (*metis mu-nu-apx-nu-L-nu-below-mu-nu-L kappa-mu-nu-L-mu-nu-apx-nu-L*)

— Theorem 28 counterexample

lemma *nu-below-mu-nu-nu-below-mu-nu-L*: *nu-below-mu-nu f* $\longrightarrow$ *nu-below-mu-nu-L f* **nitpick** [*expect=genuine*] **oops**

— Theorem 29.1

lemma *unfold-fold-1*: *isotone f* $\wedge$ *has-least-prefixpoint f* $\wedge$ *apx.has-least-fixpoint f* $\wedge$ $f(x) \le x \longrightarrow \kappa\ f \le x + L$
by (*metis add-left-isotone apx-def has-least-fixpoint-def is-least-prefixpoint-def least-prefixpoint-char least-prefixpoint-fixpoint order-trans pmu-mu kappa-apx-below-mu*)

— Theorem 29.2

lemma *unfold-fold-2*: *isotone f* $\wedge$ *apx.isotone f* $\wedge$ *has-least-prefixpoint f* $\wedge$ *has-greatest-fixpoint f* $\wedge$ *apx.has-least-fixpoint f* $\wedge$

$f(x) \le x \wedge \kappa\ f \frown L \le x \frown L \longrightarrow \kappa\ f \le x$
proof
  assume *1*: *isotone f* $\wedge$ *apx.isotone f* $\wedge$ *has-least-prefixpoint f* $\wedge$ *has-greatest-fixpoint f* $\wedge$ *apx.has-least-fixpoint f* $\wedge$ $f(x) \le x$ $\wedge$ $\kappa\ f \frown L \le x \frown L$
  hence $\kappa\ f \frown L = \nu\ f \frown L$
    by (*metis apx-meet-L meet.add-left-isotone meet.antisym kappa-apx-below-nu kappa-below-nu*)
  hence $\kappa\ f = (\kappa\ f \frown L) + \mu\ f$ **using** *1*
      by (*metis apx-meet-below-nu-kappa-mu-nu has-apx-least-fixpoint-kappa-apx-meet add-commutative least-fixpoint-char least-prefixpoint-fixpoint kappa-apx-meet-apx-meet-below-nu kappa-mu-nu-def*)
  thus $\kappa\ f \le x$ **using** *1*
    by (*metis add-least-upper-bound is-least-prefixpoint-def least-prefixpoint meet.add-least-upper-bound pmu-mu*)
qed

end

end

22 NOmegaAlgebra

theory *NOmegaAlgebra*

imports *OmegaAlgebra Recursion*

begin

class *itering-apx* = *bounded-itering* + *n-algebra-apx*

begin

lemma *circ-L*: $L^\circ = L + 1$
  by (*metis add-commutative mult-top-circ n-L-top-L*)

lemma *n-circ-import*: $n(y) ; x \leq x ; n(y) \longrightarrow n(y) ; x^\circ = n(y) ; (n(y) ; x)^\circ$
  by (*metis circ-import n-mult-idempotent n-sub-one order-refl*)

— Theorem 26.3 and Theorem 26.4

lemma *circ-apx-isotone*: $x \sqsubseteq y \longrightarrow x^\circ \sqsubseteq y^\circ$
proof
  assume $x \sqsubseteq y$
  hence *1*: $x \leq y + L \wedge n(L) ; y \leq x + n(x) ; T$
    by (*metis apx-def*)
  have $n(L) ; y^\circ \leq (n(L) ; y)^\circ$
    by (*smt circ-reflexive circ-transitive-equal n-sub-one n-circ-import n-nL-semi-commute mult-left-isotone order-trans*)
  also have ... $\leq x^\circ + x^\circ ; n(x) ; T$ **using** *1*
    by (*metis circ-isotone circ-left-top circ-unfold-sum mult-associative*)
  also have ... $\leq x^\circ + (x^\circ ; 0 + n(x^\circ ; x) ; T)$
    by (*smt add-right-isotone n-n-top-split-n-top*)
  also have ... $\leq x^\circ + (x^\circ ; 0 + n(x^\circ) ; T)$
    by (*metis add-right-isotone mult-left-isotone n-isotone right-plus-below-circ*)
  also have ... $= x^\circ + n(x^\circ) ; T$
    by (*smt add-associative add-commutative less-eq-def zero-right-mult-decreasing*)
  finally have *2*: $n(L) ; y^\circ \leq x^\circ + n(x^\circ) ; T$
    by *metis*
  have $x^\circ \leq y^\circ ; L^\circ$ **using** *1*
    by (*metis circ-add-1 circ-back-loop-fixpoint circ-isotone n-L-below-L less-eq-def mult-associative*)
  also have ... $= y^\circ + y^\circ ; L$
    by (*metis add-commutative circ-L mult-left-dist-add mult-right-one*)
  also have ... $\leq y^\circ + y^\circ ; 0 + L$
    by (*metis add-associative add-right-isotone n-L-split-L*)
  finally have $x^\circ \leq y^\circ + L$
    by (*metis add-commutative less-eq-def zero-right-mult-decreasing*)
  thus $x^\circ \sqsubseteq y^\circ$ **using** *2*
    by (*metis apx-def*)
qed

end

class *n-omega-algebra-1* = *bounded-left-zero-omega-algebra* + *n-algebra-apx* + *Omega* +
  assumes *Omega-def*: $x^\Omega = n(x^\omega) ; L + x^\star$

begin

lemma *n-omega-export*: $n(y) ; x \leq x ; n(y) \longrightarrow n(y) ; x^\omega = (n(y) ; x)^\omega$
  by (*metis mult-associative n-mult-idempotent omega-import omega-slide order-refl*)

— Theorem 30.1

lemma *L-mult-star*: $L ; x^\star = L$
  by (*metis less-eq-def mult-associative n-L-below-L star.circ-back-loop-fixpoint*)

— Theorem 30.2

lemma *mult-L-star*: $(x ; L)^\star = 1 + x ; L$
  by (*smt L-mult-star mult-associative star.circ-mult*)

lemma *mult-L-omega-below*: $(x\ ;\ L)^\omega \leq x\ ;\ L$
  by (*metis mult-right-isotone n-L-below-L omega-slide*)

— Theorem 30.5

lemma *mult-L-add-star*: $(x\ ;\ L + y)^\star = y^\star + y^\star\ ;\ x\ ;\ L$
  by (*metis L-mult-star mult-associative star.circ-add-1 star.circ-decompose-6 star.circ-unfold-sum*)

lemma *mult-L-add-omega-below*: $(x\ ;\ L + y)^\omega \leq y^\omega + y^\star\ ;\ x\ ;\ L$
proof –
  have $(x\ ;\ L + y)^\omega \leq y^\star\ ;\ x\ ;\ L + (y^\star\ ;\ x\ ;\ L)^\star\ ;\ y^\omega$
    by (*metis add-commutative mult-associative omega-decompose add-left-isotone mult-L-omega-below*)
  also have ... $\leq y^\omega + y^\star\ ;\ x\ ;\ L$
      by (*smt add-associative add-commutative less-eq-def mult-L-star mult-associative mult-left-dist-add mult-left-one mult-right-dist-add n-L-below-L order-refl*)
  finally show *?thesis*
    by *metis*
qed

lemma *n-Omega-isotone*: $x \leq y \longrightarrow x^\Omega \leq y^\Omega$
  by (*metis Omega-def add-isotone mult-left-isotone n-isotone omega-isotone star-isotone*)

lemma *n-star-below-Omega*: $x^\star \leq x^\Omega$
  by (*metis add-right-upper-bound Omega-def*)

— Theorem 30.4

lemma *mult-L-star-mult-below*: $(x\ ;\ L)^\star\ ;\ y \leq y + x\ ;\ L$
  by (*metis add-right-isotone mult-associative mult-right-isotone n-L-below-L star-left-induct*)

end

sublocale *n-omega-algebra-1* < *star*!: *itering-apx* **where** *circ* = *star* **..**

class *n-omega-algebra* = *n-omega-algebra-1* + *n-algebra-apx* +
  assumes *n-split-omega-mult*: $n(L)\ ;\ x^\omega \leq x^\star\ ;\ n(x^\omega)\ ;\ T$
  assumes *tarski*: $x\ ;\ L \leq x\ ;\ L\ ;\ x\ ;\ L$

begin

— Theorem 30.3

lemma *mult-L-omega*: $(x\ ;\ L)^\omega = x\ ;\ L$
  apply (*rule antisym*)
  apply (*rule mult-L-omega-below*)
  apply (*metis mult-associative omega-induct-mult tarski*)
  done

— Theorem 30.6

lemma *mult-L-add-omega*: $(x\ ;\ L + y)^\omega = y^\omega + y^\star\ ;\ x\ ;\ L$
  apply (*rule antisym*)
  apply (*rule mult-L-add-omega-below*)
  apply (*metis add-right-isotone add-right-upper-bound less-eq-def mult-L-omega mult-associative mult-isotone omega-sub-dist star.circ-sub-dist star-mult-omega*)
  done

— Theorem 30.3

lemma *tarski-mult-top-idempotent*: $x\ ;\ L = x\ ;\ L\ ;\ x\ ;\ L$
  by (*metis mult-L-omega mult-associative omega-unfold*)

— Theorem 30.7

lemma *n-below-n-omega*: $n(x) \leq n(x^\omega)$
proof –
  have $n(x)\ ;\ L \leq n(x)\ ;\ L\ ;\ n(x)\ ;\ L$
    by (*metis tarski*)
  also have ... $\leq x\ ;\ n(x)\ ;\ L$

  by (*metis mult-left-isotone n-L-decreasing*)
 finally have $n(x) ; L \leq x^\omega$
  by (*metis mult-associative omega-induct-mult*)
 thus *?thesis*
  by (*metis n-galois*)
qed

— Theorem 30.15

lemma *n-split-omega-add-zero*: $n(L) ; x^\omega \leq x^\star ; 0 + n(x^\omega) ; T$
proof –
 have $n(x^\omega) ; T + x ; (x^\star ; 0 + n(x^\omega) ; T) = n(x^\omega) ; T + x ; x^\star ; 0 + x ; n(x^\omega) ; T$
  by (*metis add-associative mult-associative mult-left-dist-add*)
 also have ... $\leq n(x^\omega) ; T + x ; x^\star ; 0 + x ; 0 + n(x^\omega) ; T$
  by (*metis add-associative add-right-isotone n-n-top-split-n-top omega-unfold*)
 also have ... $= x ; x^\star ; 0 + n(x^\omega) ; T$
  by (*smt add-associative add-commutative add-left-top add-right-zero mult-associative mult-left-dist-add*)
 also have ... $\leq x^\star ; 0 + n(x^\omega) ; T$
  by (*metis add-left-isotone mult-left-isotone star.left-plus-below-circ*)
 finally have $x^\star ; n(x^\omega) ; T \leq x^\star ; 0 + n(x^\omega) ; T$
  by (*metis mult-associative star-left-induct*)
 thus *?thesis*
  by (*metis n-split-omega-mult order-trans*)
qed

lemma *n-split-omega-add*: $n(L) ; x^\omega \leq x^\star + n(x^\omega) ; T$
 by (*metis add-left-isotone n-split-omega-add-zero order-trans zero-right-mult-decreasing*)

— Theorem 30.8

lemma *n-dist-omega-star*: $n(y^\omega + y^\star ; z) = n(y^\omega) + n(y^\star ; z)$
proof –
 have $n(y^\omega + y^\star ; z) \leq n(n(L) ; y^\omega + y^\star ; z)$
  by (*smt mult-associative mult-left-dist-add mult-left-isotone mult-left-one n-export n-mult-commutative n-n-nL n-sub-one*)
 also have ... $\leq n(y^\star ; 0 + n(y^\omega) ; T + y^\star ; z)$
  by (*metis add-commutative add-right-isotone n-isotone n-split-omega-add-zero*)
 also have ... $= n(y^\omega) + n(y^\star ; z)$
  by (*smt add-associative add-commutative add-right-zero mult-left-dist-add n-dist-n-add*)
 finally show *?thesis*
  by (*metis add-least-upper-bound n-left-upper-bound n-right-upper-bound antisym*)
qed

lemma *mult-L-add-circ-below*: $(x ; L + y)^\Omega \leq n(y^\omega) ; L + y^\star + y^\star ; x ; L$
proof –
 have $(x ; L + y)^\Omega \leq n(y^\omega + y^\star ; x ; L) ; L + (x ; L + y)^\star$
  by (*metis add-left-isotone mult-L-add-omega-below mult-left-isotone n-isotone Omega-def*)
 also have ... $= n(y^\omega) ; L + n(y^\star ; x ; L) ; L + (x ; L + y)^\star$
  by (*metis mult-associative mult-right-dist-add n-dist-omega-star*)
 also have ... $\leq n(y^\omega) ; L + y^\star + y^\star ; x ; L$
  by (*smt add-associative add-commutative add-idempotent add-right-isotone mult-L-add-star n-L-decreasing*)
 finally show *?thesis*
  by *metis*
qed

lemma *n-mult-omega-L-below-zero*: $n(y ; x^\omega) ; L \leq y ; x^\star ; 0 + y ; n(x^\omega) ; L$
proof –
 have $n(y ; x^\omega) ; L \leq n(L) ; y ; x^\omega \frown L$
  by (*metis n-L-decreasing-meet-L n-export n-mult-commutative n-n-nL mult-associative*)
 also have ... $\leq y ; n(L) ; x^\omega \frown L$
  by (*smt meet.add-left-isotone meet.add-right-divisibility mult-right-sub-dist-meet-right n-nL-semi-commute*)
 also have ... $\leq y ; (x^\star ; 0 + n(x^\omega) ; T) \frown L$
  by (*metis meet-commutative meet.add-right-isotone mult-associative mult-right-isotone n-split-omega-add-zero*)
 also have ... $= (y ; x^\star ; 0 \frown L) + (y ; n(x^\omega) ; T \frown L)$
  by (*metis meet-commutative meet-left-dist-add mult-associative mult-left-dist-add*)
 also have ... $\leq (y ; x^\star ; 0 \frown L) + y ; n(x^\omega) ; L$
  by (*metis add-right-isotone n-vector-meet-L*)
 also have ... $\leq y ; x^\star ; 0 + y ; n(x^\omega) ; L$
  by (*metis add-left-isotone meet.add-left-upper-bound*)
 finally show *?thesis*

  by *metis*
qed

— Theorem 30.14

lemma *n-mult-omega-L-star-zero*: $y\ ;\ x^\star\ ;\ 0 + n(y\ ;\ x^\omega)\ ;\ L = y\ ;\ x^\star\ ;\ 0 + y\ ;\ n(x^\omega)\ ;\ L$
apply (*rule antisym*)
apply (*metis add-least-upper-bound mult-associative mult-left-dist-add mult-left-sub-dist-add-left n-mult-omega-L-below-zero*)
apply (*smt add-associative add-commutative add-left-zero add-right-isotone mult-associative mult-left-dist-add n-n-L-split-n-L*)
done

— Theorem 30.11

lemma *n-mult-omega-L-star*: $y\ ;\ x^\star + n(y\ ;\ x^\omega)\ ;\ L = y\ ;\ x^\star + y\ ;\ n(x^\omega)\ ;\ L$
by (*metis zero-right-mult-decreasing n-mult-omega-L-star-zero add-relative-same-increasing*)

lemma *n-mult-omega-L-below*: $n(y\ ;\ x^\omega)\ ;\ L \leq y\ ;\ x^\star + y\ ;\ n(x^\omega)\ ;\ L$
by (*metis add-right-upper-bound n-mult-omega-L-star*)

lemma *n-omega-L-below-zero*: $n(x^\omega)\ ;\ L \leq x\ ;\ x^\star\ ;\ 0 + x\ ;\ n(x^\omega)\ ;\ L$
by (*smt omega-unfold n-mult-omega-L-below-zero add-left-isotone star.left-plus-below-circ order-trans*)

lemma *n-omega-L-below*: $n(x^\omega)\ ;\ L \leq x^\star + x\ ;\ n(x^\omega)\ ;\ L$
by (*metis omega-unfold n-mult-omega-L-below add-left-isotone star.left-plus-below-circ order-trans*)

— Theorem 30.13

lemma *n-omega-L-star-zero*: $x\ ;\ x^\star\ ;\ 0 + n(x^\omega)\ ;\ L = x\ ;\ x^\star\ ;\ 0 + x\ ;\ n(x^\omega)\ ;\ L$
by (*metis n-mult-omega-L-star-zero omega-unfold*)

— Theorem 30.10

lemma *n-omega-L-star*: $x^\star + n(x^\omega)\ ;\ L = x^\star + x\ ;\ n(x^\omega)\ ;\ L$
by (*metis star.circ-mult-upper-bound star.left-plus-below-circ zero-least n-omega-L-star-zero add-relative-same-increasing*)

— Theorem 30.12

lemma *n-omega-L-star-zero-star*: $x^\star\ ;\ 0 + n(x^\omega)\ ;\ L = x^\star\ ;\ 0 + x^\star\ ;\ n(x^\omega)\ ;\ L$
by (*metis n-mult-omega-L-star-zero star-mult-omega mult-associative star.circ-transitive-equal*)

— Theorem 30.9

lemma *n-omega-L-star-star*: $x^\star + n(x^\omega)\ ;\ L = x^\star + x^\star\ ;\ n(x^\omega)\ ;\ L$
by (*metis zero-right-mult-decreasing n-omega-L-star-zero-star add-relative-same-increasing*)

lemma *n-Omega-left-unfold*: $1 + x\ ;\ x^\Omega = x^\Omega$
by (*smt Omega-def add-associative add-commutative mult-associative mult-left-dist-add n-omega-L-star star.circ-left-unfold*)

lemma *n-Omega-left-slide*: $(x\ ;\ y)^\Omega\ ;\ x \leq x\ ;\ (y\ ;\ x)^\Omega$
proof –
have $(x\ ;\ y)^\Omega\ ;\ x \leq x\ ;\ y\ ;\ n((x\ ;\ y)^\omega)\ ;\ L + (x\ ;\ y)^\star\ ;\ x$
by (*smt Omega-def add-commutative add-left-isotone mult-associative mult-right-dist-add mult-right-isotone n-L-below-L n-omega-L-star*)
also have ... $\leq x\ ;\ (y\ ;\ 0 + n(y\ ;\ (x\ ;\ y)^\omega)\ ;\ L) + (x\ ;\ y)^\star\ ;\ x$
by (*smt add-associative add-commutative less-eq-def mult-associative mult-left-dist-add mult-left-sub-dist-add-left n-n-L-split-n-L star.circ-slide*)
also have ... $= x\ ;\ (y\ ;\ x)^\Omega$
by (*smt Omega-def add-associative add-commutative less-eq-def mult-associative mult-isotone mult-left-dist-add omega-slide star.circ-increasing star.circ-slide zero-least*)
finally show *?thesis*
by *metis*
qed

lemma *n-Omega-add-1*: $(x + y)^\Omega = x^\Omega\ ;\ (y\ ;\ x^\Omega)^\Omega$
proof –
have *1*: $(x + y)^\Omega = n((x^\star\ ;\ y)^\omega)\ ;\ L + n((x^\star\ ;\ y)^\star\ ;\ x^\omega)\ ;\ L + (x^\star\ ;\ y)^\star\ ;\ x^\star$
by (*smt Omega-def mult-right-dist-add n-dist-omega-star omega-decompose star.circ-add*)
have $n((x^\star\ ;\ y)^\omega)\ ;\ L \leq (x^\star\ ;\ y)^\star + x^\star\ ;\ (y\ ;\ n((x^\star\ ;\ y)^\omega)\ ;\ L)$
by (*metis n-omega-L-below mult-associative*)

also have ... $\leq (x^\star ; y)^\star + x^\star ; y ; 0 + x^\star ; n((y ; x^\star)^\omega) ; L$

by (*smt add-associative add-right-isotone mult-associative mult-left-dist-add mult-right-isotone n-n-L-split-n-L omega-slide*)

also have ... $= (x^\star ; y)^\star + x^\star ; n((y ; x^\star)^\omega) ; L$

by (*metis add-commutative less-eq-def star.circ-sub-dist-1 zero-right-mult-decreasing*)

also have ... $\leq x^\star ; (y ; x^\star)^\star + x^\star ; n((y ; x^\star)^\omega) ; L$

by (*metis add-left-isotone mult-right-isotone star.circ-increasing star.circ-isotone star-decompose-3*)

also have ... $\leq x^\star ; (y ; x^\Omega)^\Omega$

by (*metis Omega-def add-commutative mult-associative mult-left-dist-add mult-right-isotone n-Omega-isotone n-star-below-Omega*)

also have ... $\leq x^\Omega ; (y ; x^\Omega)^\Omega$

by (*metis n-star-below-Omega mult-left-isotone*)

finally have *2*: $n((x^\star ; y)^\omega) ; L \leq x^\Omega ; (y ; x^\Omega)^\Omega$

by *metis*

have $n((x^\star ; y)^\star ; x^\omega) ; L \leq n(x^\omega) ; L + x^\star ; (y ; x^\star)^\star + x^\star ; (y ; x^\star)^\star ; y ; n(x^\omega) ; L$

by (*smt add-associative add-commutative mult-left-one mult-right-dist-add n-mult-omega-L-below star.circ-mult star.circ-slide*)

also have ... $= n(x^\omega) ; L ; (y ; x^\Omega)^\star + x^\star ; (y ; x^\Omega)^\star$

by (*smt Omega-def add-associative mult-L-add-star mult-associative mult-left-dist-add L-mult-star*)

also have ... $\leq x^\Omega ; (y ; x^\Omega)^\Omega$

by (*metis mult-right-dist-add Omega-def n-star-below-Omega mult-right-isotone*)

finally have *3*: $n((x^\star ; y)^\star ; x^\omega) ; L \leq x^\Omega ; (y ; x^\Omega)^\Omega$

by *metis*

have $(x^\star ; y)^\star ; x^\star \leq x^\Omega ; (y ; x^\Omega)^\Omega$

by (*metis star-slide mult-isotone mult-right-isotone n-star-below-Omega order-trans star-isotone*)

hence *4*: $(x + y)^\Omega \leq x^\Omega ; (y ; x^\Omega)^\Omega$ **using** *1 2 3*

by (*metis add-least-upper-bound*)

have *5*: $x^\Omega ; (y ; x^\Omega)^\Omega \leq n(x^\omega) ; L + x^\star ; n((y ; x^\Omega)^\omega) ; L + x^\star ; (y ; x^\Omega)^\star$

by (*smt Omega-def add-associative add-left-isotone mult-associative mult-left-dist-add mult-right-dist-add mult-right-isotone n-L-below-L*)

have $n(x^\omega) ; L \leq n((x^\star ; y)^\star ; x^\omega) ; L$

by (*metis add-commutative add-left-upper-bound mult-left-isotone n-isotone star.circ-loop-fixpoint*)

hence *6*: $n(x^\omega) ; L \leq (x + y)^\Omega$ **using** *1*

by (*metis Omega-def add-left-upper-bound n-Omega-isotone order-trans*)

have $x^\star ; n((y ; x^\Omega)^\omega) ; L \leq x^\star ; n((y ; x^\star)^\omega + (y ; x^\star)^\star ; y ; n(x^\omega) ; L) ; L$

by (*metis Omega-def mult-L-add-omega-below mult-associative mult-left-dist-add mult-left-isotone mult-right-isotone n-isotone*)

also have ... $\leq x^\star ; 0 + n(x^\star ; ((y ; x^\star)^\omega + (y ; x^\star)^\star ; y ; n(x^\omega) ; L)) ; L$

by (*metis n-n-L-split-n-L*)

also have ... $\leq x^\star + n((x^\star ; y)^\omega + x^\star ; (y ; x^\star)^\star ; y ; n(x^\omega) ; L) ; L$

by (*smt add-left-isotone mult-associative mult-left-dist-add omega-slide zero-right-mult-decreasing*)

also have ... $\leq x^\star + n((x^\star ; y)^\omega + (x^\star ; y)^\star ; n(x^\omega) ; L) ; L$

by (*smt add-right-divisibility add-right-isotone mult-left-isotone n-isotone star.circ-mult*)

also have ... $\leq x^\star + n((x + y)^\omega) ; L$

by (*metis add-right-isotone mult-associative mult-left-isotone mult-right-isotone n-L-decreasing n-isotone omega-decompose*)

also have ... $\leq (x + y)^\Omega$

by (*metis add-left-isotone star.circ-sub-dist Omega-def add-commutative*)

finally have *7*: $x^\star ; n((y ; x^\Omega)^\omega) ; L \leq (x + y)^\Omega$

by *metis*

have $x^\star ; (y ; x^\Omega)^\star \leq (x^\star ; y)^\star ; x^\star + (x^\star ; y)^\star ; n(x^\omega) ; L$

by (*smt Omega-def add-right-isotone mult-L-add-star mult-associative mult-left-dist-add mult-left-isotone star.left-plus-below-circ star-slide*)

also have ... $\leq (x^\star ; y)^\star ; x^\star + n((x^\star ; y)^\star ; x^\omega) ; L$

by (*metis add-associative add-right-isotone add-right-zero mult-left-dist-add n-n-L-split-n-L*)

also have ... $\leq (x + y)^\Omega$

by (*smt Omega-def add-commutative add-right-isotone mult-left-isotone n-right-upper-bound omega-decompose star.circ-add*)

finally have $n(x^\omega) ; L + x^\star ; n((y ; x^\Omega)^\omega) ; L + x^\star ; (y ; x^\Omega)^\star \leq (x + y)^\Omega$ **using** *6 7*

by (*metis add-least-upper-bound*)

hence $x^\Omega ; (y ; x^\Omega)^\Omega \leq (x + y)^\Omega$ **using** *5*

by (*smt order-trans*)

thus *?thesis* **using** *4*

by (*metis antisym*)

qed

end

sublocale *n-omega-algebra* < *nL-omega!*: *left-zero-conway-semiring* **where** *circ* = *Omega*

apply *unfold-locales*

apply (*metis n-Omega-left-unfold*)

apply (*metis n-Omega-left-slide*)

apply (*metis n-Omega-add-1*)
done

context *n-omega-algebra*

begin

— Theorem 31.2

lemma *omega-apx-isotone*: $x \sqsubseteq y \longrightarrow x^\omega \sqsubseteq y^\omega$
proof
assume $x \sqsubseteq y$
hence *1*: $x \leq y + L \wedge n(L) ; y \leq x + n(x) ; T$
by (*metis apx-def*)
have $n(x) ; T + x ; (x^\omega + n(x^\omega) ; T) \leq n(x) ; T + x^\omega + n(x^\omega) ; T$
by (*smt add-associative mult-associative mult-left-dist-add add-right-isotone n-n-top-split-n-top add-right-zero omega-unfold*)
also have ... $\leq x^\omega + n(x^\omega) ; T$
by (*metis add-commutative add-right-isotone mult-left-isotone n-below-n-omega add-associative add-idempotent*)
finally have *2*: $x^\star ; n(x) ; T \leq x^\omega + n(x^\omega) ; T$
by (*metis mult-associative star-left-induct*)
have $n(L) ; y^\omega = (n(L) ; y)^\omega$
by (*metis n-omega-export n-nL-semi-commute*)
also have ... $\leq (x + n(x) ; T)^\omega$ **using** *1*
by (*metis omega-isotone*)
also have ... $= (x^\star ; n(x) ; T)^\omega + (x^\star ; n(x) ; T)^\star ; x^\omega$
by (*metis mult-associative omega-decompose*)
also have ... $\leq x^\star ; n(x) ; T + (x^\star ; n(x) ; T)^\star ; x^\omega$
by (*metis add-left-isotone mult-top-omega*)
also have ... $= x^\star ; n(x) ; T + (1 + x^\star ; n(x) ; T ; (x^\star ; n(x) ; T)^\star) ; x^\omega$
by (*metis mult-associative star.circ-left-top star.mult-top-circ*)
also have ... $\leq x^\omega + x^\star ; n(x) ; T$
by (*smt add-isotone add-least-upper-bound mult-associative mult-left-one mult-right-dist-add mult-right-isotone order-refl top-greatest*)
also have ... $\leq x^\omega + n(x^\omega) ; T$ **using** *2*
by (*metis add-least-upper-bound add-left-upper-bound*)
finally have *3*: $n(L) ; y^\omega \leq x^\omega + n(x^\omega) ; T$
by *metis*
have $x^\omega \leq (y + L)^\omega$ **using** *1*
by (*metis omega-isotone*)
also have ... $= (y^\star ; L)^\omega + (y^\star ; L)^\star ; y^\omega$
by (*metis omega-decompose*)
also have ... $= y^\star ; L ; (y^\star ; L)^\omega + (y^\star ; L)^\star ; y^\omega$
by (*metis omega-unfold*)
also have ... $\leq y^\star ; L + (y^\star ; L)^\star ; y^\omega$
by (*metis add-left-isotone n-L-below-L mult-associative mult-right-isotone*)
also have ... $= y^\star ; L + (1 + y^\star ; L ; (y^\star ; L)^\star) ; y^\omega$
by (*metis star.circ-left-unfold*)
also have ... $\leq y^\star ; L + y^\omega$
by (*metis add-commutative add-least-upper-bound add-right-upper-bound mult-L-star-mult-below mult-associative star.circ-mult star.circ-slide*)
also have ... $\leq y^\star ; 0 + L + y^\omega$
by (*metis add-left-isotone n-L-split-L*)
finally have $x^\omega \leq y^\omega + L$
by (*metis add-associative add-commutative less-eq-def star-zero-below-omega*)
thus $x^\omega \sqsubseteq y^\omega$ **using** *3*
by (*metis apx-def*)
qed

lemma *combined-apx-left-isotone*: $x \sqsubseteq y \longrightarrow n(x^\omega) ; L + x^\star ; z \sqsubseteq n(y^\omega) ; L + y^\star ; z$
by (*metis add-apx-isotone mult-apx-left-isotone omega-apx-isotone star.circ-apx-isotone n-L-apx-isotone*)

lemma *combined-apx-left-isotone-2*: $x \sqsubseteq y \longrightarrow (x^\omega \frown L) + x^\star ; z \sqsubseteq (y^\omega \frown L) + y^\star ; z$
by (*metis add-apx-isotone mult-apx-left-isotone omega-apx-isotone star.circ-apx-isotone meet-L-apx-isotone*)

lemma *combined-apx-right-isotone*: $y \sqsubseteq z \longrightarrow n(x^\omega) ; L + x^\star ; y \sqsubseteq n(x^\omega) ; L + x^\star ; z$
by (*metis add-apx-right-isotone mult-apx-right-isotone*)

lemma *combined-apx-right-isotone-2*: $y \sqsubseteq z \longrightarrow (x^\omega \frown L) + x^\star ; y \sqsubseteq (x^\omega \frown L) + x^\star ; z$
  by (*metis add-apx-right-isotone mult-apx-right-isotone*)

lemma *combined-apx-isotone*: $x \sqsubseteq y \wedge w \sqsubseteq z \longrightarrow n(x^\omega) ; L + x^\star ; w \sqsubseteq n(y^\omega) ; L + y^\star ; z$
  by (*metis add-apx-isotone mult-apx-isotone omega-apx-isotone star.circ-apx-isotone n-L-apx-isotone*)

lemma *combined-apx-isotone-2*: $x \sqsubseteq y \wedge w \sqsubseteq z \longrightarrow (x^\omega \frown L) + x^\star ; w \sqsubseteq (y^\omega \frown L) + y^\star ; z$
  by (*metis add-apx-isotone mult-apx-isotone omega-apx-isotone star.circ-apx-isotone meet-L-apx-isotone*)

— Theorem 30.16

lemma *n-split-nu-mu*: $n(L) ; (y^\omega + y^\star ; z) \leq y^\star ; z + n(y^\omega + y^\star ; z) ; T$
proof –
  have $n(L) ; (y^\omega + y^\star ; z) \leq n(L) ; y^\omega + y^\star ; z$
    by (*metis add-right-isotone mult-left-dist-add mult-left-isotone mult-left-one n-sub-one*)
  also have ... $\leq y^\star ; 0 + n(y^\omega) ; T + y^\star ; z$
    by (*metis add-commutative add-right-isotone n-split-omega-add-zero*)
  also have ... $\leq y^\star ; z + n(y^\omega + y^\star ; z) ; T$
    by (*smt add-associative add-commutative add-right-isotone add-right-zero mult-left-dist-add mult-left-isotone n-left-upper-bound*)
  finally show *?thesis*
    by *metis*
qed

lemma *n-split-nu-mu-2*: $n(L) ; (y^\omega + y^\star ; z) \leq y^\star ; z + ((y^\omega + y^\star ; z) \frown L) + n(y^\omega + y^\star ; z) ; T$
  by (*smt add-left-isotone add-left-upper-bound add-right-isotone add-right-upper-bound n-split-nu-mu order-trans*)

lemma *loop-exists*: $n(L) ; \nu\ (\lambda x . y ; x + z) \leq \mu\ (\lambda x . y ; x + z) + n(\nu\ (\lambda x . y ; x + z)) ; T$
  by (*metis n-split-nu-mu omega-loop-nu star-loop-mu*)

lemma *loop-exists-2*: $n(L) ; \nu\ (\lambda x . y ; x + z) \leq \mu\ (\lambda x . y ; x + z) + (\nu\ (\lambda x . y ; x + z) \frown L) + n(\nu\ (\lambda x . y ; x + z)) ; T$
  by (*metis n-split-nu-mu-2 omega-loop-nu star-loop-mu*)

lemma *loop-apx-least-fixpoint*: *apx.is-least-fixpoint* $(\lambda x . y ; x + z)\ (\mu\ (\lambda x . y ; x + z) + n(\nu\ (\lambda x . y ; x + z)) ; L)$
proof –
  have *kappa-mu-nu-L* $(\lambda x . y ; x + z)$
    by (*metis affine-apx-isotone loop-exists affine-has-greatest-fixpoint affine-has-least-fixpoint affine-isotone nu-below-mu-nu-L-def nu-below-mu-nu-L-kappa-mu-nu-L*)
  thus *?thesis*
    by (*smt apx.least-fixpoint-char kappa-mu-nu-L-def*)
qed

lemma *loop-apx-least-fixpoint-2*: *apx.is-least-fixpoint* $(\lambda x . y ; x + z)\ (\mu\ (\lambda x . y ; x + z) + (\nu\ (\lambda x . y ; x + z) \frown L))$
proof –
  have *kappa-mu-nu* $(\lambda x . y ; x + z)$
    by (*metis affine-apx-isotone affine-has-greatest-fixpoint affine-has-least-fixpoint affine-isotone loop-exists-2 nu-below-mu-nu-def nu-below-mu-nu-kappa-mu-nu*)
  thus *?thesis*
    by (*smt apx.least-fixpoint-char kappa-mu-nu-def*)
qed

lemma *loop-has-apx-least-fixpoint*: *apx.has-least-fixpoint* $(\lambda x . y ; x + z)$
  by (*metis apx.has-least-fixpoint-def loop-apx-least-fixpoint*)

lemma *loop-semantics*: $\kappa\ (\lambda x . y ; x + z) = \mu\ (\lambda x . y ; x + z) + n(\nu\ (\lambda x . y ; x + z)) ; L$
  by (*metis apx.least-fixpoint-char loop-apx-least-fixpoint*)

lemma *loop-semantics-2*: $\kappa\ (\lambda x . y ; x + z) = \mu\ (\lambda x . y ; x + z) + (\nu\ (\lambda x . y ; x + z) \frown L)$
  by (*metis apx.least-fixpoint-char loop-apx-least-fixpoint-2*)

— Theorem 31.1

lemma *loop-semantics-kappa-mu-nu*: $\kappa\ (\lambda x . y ; x + z) = n(y^\omega) ; L + y^\star ; z$
proof –
  have $\kappa\ (\lambda x . y ; x + z) = y^\star ; z + n(y^\omega + y^\star ; z) ; L$
    by (*metis loop-semantics omega-loop-nu star-loop-mu*)
  thus *?thesis*
    by (*smt n-dist-omega-star add-associative mult-right-dist-add add-commutative less-eq-def n-L-decreasing*)
qed

— Theorem 31.1

lemma *loop-semantics-kappa-mu-nu-2*: $\kappa\ (\lambda x\ .\ y\ ;\ x + z) = (y^\omega \frown L) + y^\star\ ;\ z$
proof –
  have $\kappa\ (\lambda x\ .\ y\ ;\ x + z) = y^\star\ ;\ z + ((y^\omega + y^\star\ ;\ z) \frown L)$
    by (*metis loop-semantics-2 omega-loop-nu star-loop-mu*)
  thus *?thesis*
    by (*smt add-absorb add-associative add-commutative add-left-dist-meet*)
qed

— Theorem 31.2

lemma *loop-semantics-apx-left-isotone*: $w \sqsubseteq y \longrightarrow \kappa\ (\lambda x\ .\ w\ ;\ x + z) \sqsubseteq \kappa\ (\lambda x\ .\ y\ ;\ x + z)$
  by (*metis loop-semantics-kappa-mu-nu combined-apx-left-isotone*)

— Theorem 31.2

lemma *loop-semantics-apx-right-isotone*: $w \sqsubseteq z \longrightarrow \kappa\ (\lambda x\ .\ y\ ;\ x + w) \sqsubseteq \kappa\ (\lambda x\ .\ y\ ;\ x + z)$
  by (*metis loop-semantics-kappa-mu-nu combined-apx-right-isotone*)

lemma *loop-semantics-apx-isotone*: $v \sqsubseteq y \wedge w \sqsubseteq z \longrightarrow \kappa\ (\lambda x\ .\ v\ ;\ x + w) \sqsubseteq \kappa\ (\lambda x\ .\ y\ ;\ x + z)$
  by (*metis loop-semantics-kappa-mu-nu combined-apx-isotone*)

end

end

23 NOmegaAlgebraBinaryItering

```
theory NOmegaAlgebraBinaryItering

imports NOmegaAlgebra BinaryIteringStrict

begin

sublocale extended-binary-itering < left-zero-conway-semiring where circ = (λx . x ⋆ 1)
  apply unfold-locales
  apply (metis while-left-unfold)
  apply (metis mult-right-one while-one-mult-below while-slide)
  apply (metis while-one-while while-sumstar-2)
  done

class binary-itering-apx = bounded-binary-itering + n-algebra-apx

begin

lemma n-while-import: n(y) ; x ≤ x ; n(y) ⟶ n(y) ; (x ⋆ z) = n(y) ; ((n(y) ; x) ⋆ z)
  by (metis while-import n-mult-idempotent n-sub-one order-refl)

lemma n-while-preserve: n(y) ; x ≤ x ; n(y) ⟶ n(y) ; (x ⋆ z) = n(y) ; (x ⋆ (n(y) ; z))
  by (metis while-preserve n-mult-idempotent n-sub-one order-refl)

lemma while-L-L: L ⋆ L = L
  by (metis n-L-top-L while-mult-star-exchange while-right-top)

lemma while-L-below-add: L ⋆ x ≤ x + L
  by (metis while-left-unfold add-right-isotone n-L-below-L)

lemma while-L-split: x ⋆ L ≤ (x ⋆ y) + L
proof −
  have x ⋆ L ≤ (x ⋆ 0) + L
    by (metis add-commutative add-left-zero mult-right-one n-L-split-L while-right-unfold while-simulate-left-plus while-zero)
  thus ?thesis
    by (metis add-commutative add-right-isotone order-trans while-right-isotone zero-least)
qed

lemma while-n-while-top-split: x ⋆ (n(x ⋆ y) ; T) ≤ (x ⋆ 0) + n(x ⋆ y) ; T
proof −
  have x ; n(x ⋆ y) ; T ≤ x ; 0 + n(x ; (x ⋆ y)) ; T
    by (metis n-n-top-split-n-top)
  also have ... ≤ n(x ⋆ y) ; T + x ; 0
    by (metis add-commutative add-right-isotone mult-left-isotone n-isotone while-left-plus-below)
  finally have x ⋆ (n(x ⋆ y) ; T) ≤ n(x ⋆ y) ; T + (x ⋆ (x ; 0))
    by (metis mult-associative mult-right-one while-simulate-left mult-left-zero while-left-top)
  also have ... ≤ (x ⋆ 0) + n(x ⋆ y) ; T
    by (metis add-least-upper-bound add-left-isotone while-right-plus-below)
  finally show ?thesis
    by metis
qed

lemma circ-apx-right-isotone: x ⊑ y ⟶ z ⋆ x ⊑ z ⋆ y
proof
  assume x ⊑ y
  hence 1: x ≤ y + L ∧ n(L) ; y ≤ x + n(x) ; T
    by (metis apx-def)
  hence z ⋆ x ≤ (z ⋆ y) + (z ⋆ L)
    by (metis while-left-dist-add while-right-isotone)
  hence 2: z ⋆ x ≤ (z ⋆ y) + L
    by (smt add-least-upper-bound add-left-upper-bound while-L-split order-trans)
  have z ⋆ (n(z ⋆ x) ; T) ≤ (z ⋆ 0) + n(z ⋆ x) ; T
    by (metis while-n-while-top-split)
  also have ... ≤ (z ⋆ x) + n(z ⋆ x) ; T
    by (metis add-left-isotone while-right-isotone zero-least)
  finally have 3: z ⋆ (n(x) ; T) ≤ (z ⋆ x) + n(z ⋆ x) ; T
    by (metis mult-left-isotone n-isotone order-trans while-increasing while-right-isotone)
  have n(L) ; (z ⋆ y) ≤ z ⋆ (n(L) ; y)
```

by (*metis n-nL-semi-commute while-simulate*)
also have ... $\leq (z \star x) + (z \star (n(x) ; T))$ **using** *1*
by (*metis while-left-dist-add while-right-isotone*)
also have ... $\leq (z \star x) + n(z \star x) ; T$ **using** *3*
by (*metis add-least-upper-bound add-left-upper-bound*)
finally show $z \star x \sqsubseteq z \star y$ **using** *2*
by (*metis apx-def*)
qed

end

class *extended-binary-itering-apx* = *binary-itering-apx* + *bounded-extended-binary-itering* +
assumes *n-below-while-zero*: $n(x) \leq n(x \star 0)$

begin

lemma *circ-apx-right-isotone*: $x \sqsubseteq y \longrightarrow x \star z \sqsubseteq y \star z$
proof
assume $x \sqsubseteq y$
hence *1*: $x \leq y + L \wedge n(L) ; y \leq x + n(x) ; T$
by (*metis apx-def*)
hence $x \star z \leq ((y \star 1) ; L) \star (y \star z)$
by (*metis while-left-isotone while-sumstar-3*)
also have ... $\leq (y \star z) + (y \star 1) ; L$
by (*metis while-productstar add-right-isotone mult-right-isotone n-L-below-L while-slide*)
also have ... $\leq (y \star z) + L$
by (*metis add-commutative add-least-upper-bound add-right-upper-bound order-trans while-L-split while-one-mult-below*)
finally have *2*: $x \star z \leq (y \star z) + L$
by *metis*
have $n(L) ; (y \star z) \leq (n(L) ; y) \star z$
by (*metis n-nL-semi-commute n-while-import n-sub-one mult-left-one mult-left-isotone*)
also have ... $\leq ((x \star 1) ; n(x) ; T) \star (x \star z)$ **using** *1*
by (*metis while-left-isotone mult-associative while-sumstar-3*)
also have ... $\leq (x \star z) + (x \star 1) ; n(x) ; T$
by (*metis while-productstar add-left-top add-right-isotone mult-associative mult-left-sub-dist-add-right while-slide*)
also have ... $\leq (x \star z) + (x \star (n(x) ; T))$
by (*metis add-right-isotone mult-associative while-one-mult-below*)
also have ... $\leq (x \star z) + (x \star (n(x \star z) ; T))$
by (*metis n-below-while-zero zero-least while-right-isotone n-isotone mult-left-isotone add-right-isotone order-trans*)
also have ... $\leq (x \star z) + n(x \star z) ; T$
by (*smt add-associative add-right-isotone while-n-while-top-split add-right-zero while-left-dist-add*)
finally show $x \star z \sqsubseteq y \star z$ **using** *2*
by (*metis apx-def*)
qed

lemma *while-top*: $T \star x = L + T ; x$ **nitpick** [*expect=genuine*] **oops**
lemma *while-one-top*: $1 \star x = L + x$ **nitpick** [*expect=genuine*] **oops**
lemma *while-unfold-below-1*: $x = y ; x \longrightarrow x \leq y \star 1$ **nitpick** [*expect=genuine*] **oops**

lemma *while-square-1*: $x \star 1 = (x ; x) \star (x + 1)$ **oops**
lemma *while-absorb-below-one*: $y ; x \leq x \longrightarrow y \star x \leq 1 \star x$ **oops**
lemma *while-mult-L*: $(x ; L) \star z = z + x ; L$ **oops**
lemma *tarski-top-omega-below-2*: $x ; L \leq (x ; L) \star 0$ **oops**
lemma *tarski-top-omega-2*: $x ; L = (x ; L) \star 0$ **oops**
lemma *while-separate-right-plus*: $y ; x \leq x ; (x \star (1 + y)) + 1 \longrightarrow y \star (x \star z) \leq x \star (y \star z)$ **oops**
lemma $y \star (x \star 1) \leq x \star (y \star 1) \longrightarrow (x + y) \star 1 = x \star (y \star 1)$ **oops**
lemma $y ; x \leq (1 + x) ; (y \star 1) \longrightarrow (x + y) \star 1 = x \star (y \star 1)$ **oops**

end

class *n-omega-algebra-binary* = *n-omega-algebra* + *while* +
assumes *while-def*: $x \star y = n(x^\omega) ; L + x^\star ; y$

begin

lemma *while-omega-meet-L-star*: $x \star y = (x^\omega \frown L) + x^\star ; y$
by (*metis loop-semantics-kappa-mu-nu loop-semantics-kappa-mu-nu-2 while-def*)

lemma *while-one-mult-while-below-1*: $(y \star 1) ; (y \star v) \leq y \star v$

proof –
have $(y \star 1) ; (y \star v) \le y \star (y \star v)$
by (*smt add-left-isotone mult-associative mult-right-dist-add mult-right-isotone n-L-below-L while-def mult-left-one*)
also have ... $= n(y^\omega) ; L + y^\star ; n(y^\omega) ; L + y^\star ; y^\star ; v$
by (*metis while-def mult-left-dist-add add-associative mult-associative*)
also have ... $= n(y^\omega) ; L + n(y^\star ; y^\omega) ; L + y^\star ; y^\star ; v$
by (*smt n-mult-omega-L-star-zero add-relative-same-increasing add-associative add-left-zero mult-left-sub-dist-add-left add-commutative*)
finally show *?thesis*
by (*metis add-idempotent star.circ-transitive-equal star-mult-omega while-def*)
qed

lemma *star-below-while*: $x^\star ; y \le x \star y$
by (*metis add-right-upper-bound while-def*)

subclass *bounded-binary-itering*
proof *unfold-locales*
fix *x y z*
have $z + x ; ((y ; x) \star (y ; z)) = x ; (y ; x)^\star ; y ; z + x ; n((y ; x)^\omega) ; L + z$
by (*smt add-associative add-commutative mult-associative mult-left-dist-add while-def*)
also have ... $= x ; (y ; x)^\star ; y ; z + n(x ; (y ; x)^\omega) ; L + z$
by (*metis mult-associative mult-right-isotone zero-least n-mult-omega-L-star-zero add-relative-same-increasing*)
also have ... $= (x ; y)^\star ; z + n(x ; (y ; x)^\omega) ; L$
by (*smt add-associative add-commutative mult-associative star.circ-loop-fixpoint star-slide*)
also have ... $= (x ; y) \star z$
by (*smt omega-slide while-def add-commutative*)
finally show $(x ; y) \star z = z + x ; ((y ; x) \star (y ; z))$
by *metis*
next
fix *x y z*
have $(x \star y) \star (x \star z) = n((n(x^\omega) ; L + x^\star ; y)^\omega) ; L + (n(x^\omega) ; L + x^\star ; y)^\star ; (x \star z)$
by (*metis while-def*)
also have ... $= n((x^\star ; y)^\omega + (x^\star ; y)^\star ; n(x^\omega) ; L) ; L + ((x^\star ; y)^\star + (x^\star ; y)^\star ; n(x^\omega) ; L) ; (x \star z)$
by (*metis mult-L-add-star mult-L-add-omega*)
also have ... $= n((x^\star ; y)^\omega) ; L + n((x^\star ; y)^\star ; n(x^\omega) ; L) ; L + (x^\star ; y)^\star ; (x \star z) + (x^\star ; y)^\star ; n(x^\omega) ; L ; (x \star z)$
by (*metis mult-associative n-dist-omega-star mult-right-dist-add add-associative*)
also have ... $= n((x^\star ; y)^\omega) ; L + n((x^\star ; y)^\star ; n(x^\omega) ; L) ; L + (x^\star ; y)^\star ; 0 + (x^\star ; y)^\star ; (x \star z) + (x^\star ; y)^\star ; n(x^\omega) ; L ; (x \star z)$
by (*smt add-associative add-left-zero mult-left-dist-add*)
also have ... $= n((x^\star ; y)^\omega) ; L + ((x^\star ; y)^\star ; n(x^\omega) ; L ; (x \star z) + (x^\star ; y)^\star ; n(x^\omega) ; L + (x^\star ; y)^\star ; (x \star z))$
by (*smt n-n-L-split-n-n-L-L add-commutative add-associative*)
also have ... $= n((x^\star ; y)^\omega) ; L + ((x^\star ; y)^\star ; n(x^\omega) ; L + (x^\star ; y)^\star ; (x \star z))$
by (*smt mult-L-omega omega-sub-vector less-eq-def*)
also have ... $= n((x^\star ; y)^\omega) ; L + (x^\star ; y)^\star ; (x \star z)$
by (*metis add-left-divisibility mult-associative mult-right-isotone while-def less-eq-def*)
also have ... $= (x^\star ; y)^\star ; x^\star ; z + (x^\star ; y)^\star ; n(x^\omega) ; L + n((x^\star ; y)^\omega) ; L$
by (*metis add-commutative mult-associative mult-left-dist-add while-def*)
also have ... $= (x^\star ; y)^\star ; x^\star ; z + n((x^\star ; y)^\star ; x^\omega) ; L + n((x^\star ; y)^\omega) ; L$
by (*metis add-right-zero mult-left-dist-add add-associative n-mult-omega-L-star-zero*)
also have ... $= (x + y) \star z$
by (*metis add-associative add-commutative omega-decompose star.circ-add while-def mult-right-dist-add n-dist-omega-star*)
finally show $(x + y) \star z = (x \star y) \star (x \star z)$
by *metis*
next
fix *x y z*
show $x \star (y + z) = (x \star y) + (x \star z)$
by (*smt add-associative add-commutative add-left-upper-bound less-eq-def mult-left-dist-add while-def*)
next
fix *x y z*
show $(x \star y) ; z \le x \star (y ; z)$
by (*smt add-left-isotone mult-associative mult-right-dist-add mult-right-isotone n-L-below-L while-def*)
next
fix *v w x y z*
show $x ; z \le z ; (y \star 1) + w \longrightarrow x \star (z ; v) \le z ; (y \star v) + (x \star (w ; (y \star v)))$
proof
assume *1*: $x ; z \le z ; (y \star 1) + w$
have $z ; v + x ; (z ; (y \star v) + x^\star ; (w ; (y \star v))) \le z ; v + x ; z ; (y \star v) + x^\star ; (w ; (y \star v))$
by (*metis add-associative add-right-isotone mult-associative mult-left-dist-add mult-left-isotone star.left-plus-below-circ*)
also have ... $\le z ; v + z ; (y \star 1) ; (y \star v) + w ; (y \star v) + x^\star ; (w ; (y \star v))$ **using** *1*

by (*metis add-associative add-left-isotone add-right-isotone mult-left-isotone mult-right-dist-add*)

also have ... $\leq z ; v + z ; (y \star v) + x^\star ; (w ; (y \star v))$

by (*smt add-least-upper-bound add-right-upper-bound less-eq-def mult-associative mult-left-dist-add star.circ-loop-fixpoint while-one-mult-while-below-1*)

also have ... $= z ; (y \star v) + x^\star ; (w ; (y \star v))$

by (*metis less-eq-def mult-left-dist-add mult-left-one mult-right-sub-dist-add-left order-trans star.circ-plus-one star-below-while*)

finally have $x^\star ; z ; v \leq z ; (y \star v) + x^\star ; (w ; (y \star v))$

by (*metis mult-associative star-left-induct*)

thus $x \star (z ; v) \leq z ; (y \star v) + (x \star (w ; (y \star v)))$

by (*smt add-associative add-commutative add-right-isotone mult-associative while-def*)

qed

next

fix $v\ w\ x\ y\ z$

show $z ; x \leq y ; (y \star z) + w \longrightarrow z ; (x \star v) \leq y \star (z ; v + w ; (x \star v))$

proof

assume $z ; x \leq y ; (y \star z) + w$

hence *1*: $z ; x \leq y ; y^\star ; z + (y ; n(y^\omega) ; L + w)$

by (*smt add-associative add-commutative mult-associative mult-left-dist-add while-def*)

hence $z ; x^\star \leq y^\star ; (z + (y ; n(y^\omega) ; L + w) ; x^\star)$

by (*metis star.circ-simulate-right-plus*)

also have ... $= y^\star ; z + y^\star ; y ; n(y^\omega) ; L + y^\star ; w ; x^\star$

by (*smt add-associative mult-associative mult-left-dist-add mult-right-dist-add L-mult-star*)

also have ... $= y^\star ; z + n(y^\star ; y ; y^\omega) ; L + y^\star ; w ; x^\star$

by (*metis add-relative-same-increasing mult-isotone n-mult-omega-L-star-zero star.left-plus-below-circ star.right-plus-circ zero-least*)

also have ... $= n(y^\omega) ; L + y^\star ; z + y^\star ; w ; x^\star$

by (*metis add-commutative omega-unfold right-plus-omega*)

finally have $z ; x^\star ; v \leq n(y^\omega) ; L ; v + y^\star ; z ; v + y^\star ; w ; x^\star ; v$

by (*smt less-eq-def mult-right-dist-add*)

also have ... $\leq n(y^\omega) ; L + y^\star ; (z ; v + w ; x^\star ; v)$

by (*metis n-L-below-L mult-associative mult-right-isotone add-left-isotone mult-left-dist-add add-associative*)

also have ... $\leq n(y^\omega) ; L + y^\star ; (z ; v + w ; (x \star v))$

by (*metis add-commutative add-right-isotone mult-associative mult-left-sub-dist-add-left mult-right-isotone while-def*)

finally have *2*: $z ; x^\star ; v \leq y \star (z ; v + w ; (x \star v))$

by (*metis while-def*)

have *3*: $y^\star ; y ; y^\star ; 0 \leq y^\star ; w ; x^\omega$

by (*metis add-commutative add-left-zero mult-associative mult-left-sub-dist-add-left star.circ-loop-fixpoint star.circ-transitive-equal*)

have $z ; x^\omega \leq y ; y^\star ; z ; x^\omega + (y ; n(y^\omega) ; L + w) ; x^\omega$ **using** *1*

by (*metis mult-associative mult-left-isotone mult-right-dist-add omega-unfold*)

hence $z ; x^\omega \leq y^\omega + y^\star ; y ; n(y^\omega) ; L ; x^\omega + y^\star ; w ; x^\omega$

by (*smt add-associative add-commutative left-plus-omega mult-associative mult-left-dist-add mult-right-dist-add omega-induct star.left-plus-circ*)

also have ... $\leq y^\omega + y^\star ; y ; n(y^\omega) ; L + y^\star ; w ; x^\omega$

by (*metis add-left-isotone add-right-isotone mult-associative mult-right-isotone n-L-below-L*)

also have ... $= y^\omega + n(y^\star ; y ; y^\omega) ; L + y^\star ; w ; x^\omega$ **using** *3*

by (*smt add-associative add-commutative add-relative-same-increasing n-mult-omega-L-star-zero*)

also have ... $= y^\omega + y^\star ; w ; x^\omega$

by (*metis mult-associative omega-unfold star-mult-omega add-commutative less-eq-def n-L-decreasing*)

finally have $n(z ; x^\omega) ; L \leq n(y^\omega) ; L + n(y^\star ; w ; x^\omega) ; L$

by (*metis mult-associative mult-left-isotone mult-right-dist-add n-dist-omega-star n-isotone*)

also have ... $\leq n(y^\omega) ; L + y^\star ; (w ; (n(x^\omega) ; L + x^\star ; 0))$

by (*smt add-commutative add-right-isotone mult-associative mult-left-dist-add n-mult-omega-L-below-zero*)

also have ... $\leq n(y^\omega) ; L + y^\star ; (w ; (n(x^\omega) ; L + x^\star ; v))$

by (*metis add-right-isotone mult-right-isotone zero-least*)

also have ... $\leq n(y^\omega) ; L + y^\star ; (z ; v + w ; (n(x^\omega) ; L + x^\star ; v))$

by (*metis add-right-isotone mult-left-sub-dist-add-right*)

finally have *4*: $n(z ; x^\omega) ; L \leq y \star (z ; v + w ; (x \star v))$

by (*metis while-def*)

have $z ; (x \star v) = z ; n(x^\omega) ; L + z ; x^\star ; v$

by (*metis while-def mult-left-dist-add mult-associative*)

also have ... $= n(z ; x^\omega) ; L + z ; x^\star ; v$

by (*metis add-commutative add-relative-same-increasing mult-right-isotone n-mult-omega-L-star-zero zero-least*)

finally show $z ; (x \star v) \leq y \star (z ; v + w ; (x \star v))$ **using** *2 4*

by (*metis add-least-upper-bound*)

qed

qed

lemma *while-top*: $T \star x = L + T ; x$
by (*metis n-top-L star.circ-top star-omega-top while-def*)

lemma *while-one-top*: $1 \star x = L + x$
by (*smt mult-left-one n-top-L omega-one star-one while-def*)

lemma *while-finite-associative*: $x^\omega = 0 \longrightarrow (x \star y) ; z = x \star (y ; z)$
by (*metis add-left-zero mult-associative n-zero-L-zero while-def*)

lemma *while-while-one*: $y \star (x \star 1) = n(y^\omega) ; L + y^\star ; n(x^\omega) ; L + y^\star ; x^\star$
by (*metis add-associative mult-left-dist-add mult-right-one while-def mult-associative*)

— Theorem 8.4 and Theorem 31.3

subclass *bounded-extended-binary-itering*
proof *unfold-locales*
fix *w x y z*
have $w ; (x \star y ; z) = n(w ; n(x^\omega) ; L) ; L + w ; x^\star ; y ; z$
by (*smt add-associative add-commutative add-left-zero mult-associative mult-left-dist-add n-n-L-split-n-n-L-L while-def*)
also have ... $\leq n((w ; n(x^\omega) ; L)^\omega) ; L + w ; x^\star ; y ; z$
by (*metis eq-refl mult-L-omega*)
also have ... $\leq n((w ; (x \star y))^\omega) ; L + w ; x^\star ; y ; z$
by (*smt add-left-isotone add-left-upper-bound mult-associative mult-left-isotone mult-right-isotone n-isotone omega-isotone while-def*)
also have ... $\leq n((w ; (x \star y))^\omega) ; L + w ; (x \star y) ; z$
by (*metis star-below-while mult-associative mult-left-isotone mult-right-isotone add-right-isotone*)
also have ... $\leq n((w ; (x \star y))^\omega) ; L + (w ; (x \star y))^\star ; (w ; (x \star y) ; z)$
by (*metis add-right-isotone add-right-upper-bound star.circ-loop-fixpoint*)
finally show $w ; (x \star y ; z) \leq (w ; (x \star y)) \star (w ; (x \star y) ; z)$
by (*metis while-def*)
qed

subclass *extended-binary-itering-apx*
apply *unfold-locales*
apply (*metis n-below-n-omega n-left-upper-bound n-n-L order-trans while-def*)
done

lemma *while-simulate-4-plus*: $y ; x \leq x ; (x \star (1 + y)) \longrightarrow y ; x ; x^\star \leq x ; (x \star (1 + y))$
proof
assume *1*: $y ; x \leq x ; (x \star (1 + y))$
have $x ; (x \star (1 + y)) = x ; n(x^\omega) ; L + x ; x^\star ; (1 + y)$
by (*metis mult-associative mult-left-dist-add while-def*)
also have ... $= n(x ; x^\omega) ; L + x ; x^\star ; (1 + y)$
by (*smt n-mult-omega-L-star-zero add-relative-same-increasing add-commutative add-right-zero mult-left-sub-dist-add-right*)
finally have *2*: $x ; (x \star (1 + y)) = n(x^\omega) ; L + x ; x^\star + x ; x^\star ; y$
by (*metis add-associative mult-left-dist-add mult-right-one omega-unfold*)
hence $x ; x^\star ; y ; x \leq x ; x^\star ; n(x^\omega) ; L + x ; x^\star ; x^\star ; x + x ; x^\star ; x ; x^\star ; y$ **using** *1*
by (*metis mult-associative mult-right-isotone mult-left-dist-add star-plus*)
also have ... $= n(x ; x^\star ; x^\omega) ; L + x ; x^\star ; x^\star ; x + x ; x^\star ; x ; x^\star ; y$
by (*smt n-mult-omega-L-star-zero add-relative-same-increasing add-commutative add-right-zero mult-left-sub-dist-add-right*)
also have ... $= n(x^\omega) ; L + x ; x^\star ; x + x ; x ; x^\star ; y$
by (*metis mult-associative omega-unfold star.circ-plus-same star.circ-transitive-equal star-mult-omega*)
also have ... $\leq n(x^\omega) ; L + x ; x^\star + x ; x^\star ; y$
by (*smt add-associative add-right-upper-bound less-eq-def mult-associative mult-right-dist-add star.circ-increasing star.circ-plus-same star.circ-transitive-equal*)
finally have *3*: $x ; x^\star ; y ; x \leq n(x^\omega) ; L + x ; x^\star + x ; x^\star ; y$
by *metis*
have $(n(x^\omega) ; L + x ; x^\star + x ; x^\star ; y) ; x \leq n(x^\omega) ; L + x ; x^\star ; x + x ; x^\star ; y ; x$
by (*metis mult-right-dist-add n-L-below-L mult-associative mult-right-isotone add-left-isotone*)
also have ... $\leq n(x^\omega) ; L + x ; x^\star + x ; x^\star ; y ; x$
by (*smt add-commutative add-left-isotone mult-associative mult-right-isotone star.left-plus-below-circ star-plus*)
also have ... $\leq n(x^\omega) ; L + x ; x^\star + x ; x^\star ; y$ **using** *3*
by (*metis add-least-upper-bound add-left-upper-bound*)
finally show $y ; x ; x^\star \leq x ; (x \star (1 + y))$ **using** *1 2*
by (*metis add-least-upper-bound star-right-induct*)
qed

lemma *while-simulate-4-omega*: $y ; x \leq x ; (x \star (1 + y)) \longrightarrow y ; x^\omega \leq x^\omega$
proof

assume *1*: $y ; x \le x ; (x \star (1 + y))$
have $x ; (x \star (1 + y)) = x ; n(x^\omega) ; L + x ; x^\star ; (1 + y)$
by (*metis mult-associative mult-left-dist-add while-def*)
also have ... $= n(x ; x^\omega) ; L + x ; x^\star ; (1 + y)$
by (*smt n-mult-omega-L-star-zero add-relative-same-increasing add-commutative add-right-zero mult-left-sub-dist-add-right*)
finally have $x ; (x \star (1 + y)) = n(x^\omega) ; L + x ; x^\star + x ; x^\star ; y$
by (*metis add-associative mult-left-dist-add mult-right-one omega-unfold*)
hence $y ; x^\omega \le n(x^\omega) ; L ; x^\omega + x ; x^\star ; x^\omega + x ; x^\star ; y ; x^\omega$ **using** *1*
by (*smt less-eq-def mult-associative mult-right-dist-add omega-unfold*)
also have ... $\le x ; x^\star ; (y ; x^\omega) + x^\omega$
by (*metis add-left-isotone mult-L-omega omega-sub-vector mult-associative omega-unfold star-mult-omega n-L-decreasing less-eq-def add-commutative*)
finally have $y ; x^\omega \le (x ; x^\star)^\omega + (x ; x^\star)^\star ; x^\omega$
by (*metis add-commutative omega-induct*)
thus $y ; x^\omega \le x^\omega$
by (*metis add-idempotent left-plus-omega star-mult-omega*)
qed

lemma *while-square-1*: $x \star 1 = (x ; x) \star (x + 1)$
by (*metis mult-right-one omega-square star-square-2 while-def*)

lemma *while-absorb-below-one*: $y ; x \le x \longrightarrow y \star x \le 1 \star x$
by (*metis star-left-induct-mult add-isotone n-galois n-sub-nL while-def while-one-top*)

lemma *while-mult-L*: $(x ; L) \star z = z + x ; L$
by (*metis add-right-zero mult-left-zero while-denest-5 while-one-top while-productstar while-sumstar*)

lemma *tarski-top-omega-below-2*: $x ; L \le (x ; L) \star 0$
by (*metis add-right-divisibility while-mult-L*)

lemma *tarski-top-omega-2*: $x ; L = (x ; L) \star 0$
by (*metis add-left-zero while-mult-L*)

lemma *while-sub-mult-one*: $x ; (1 \star y) \le 1 \star x$ **nitpick** [*expect=genuine*] **oops**
lemma *while-unfold-below*: $x = z + y ; x \longrightarrow x \le y \star z$ **nitpick** [*expect=genuine*] **oops**
lemma *while-loop-is-greatest-postfixpoint*: *is-greatest-postfixpoint* $(\lambda x . y ; x + z) (y \star z)$ **nitpick** [*expect=genuine*] **oops**
lemma *while-loop-is-greatest-fixpoint*: *is-greatest-fixpoint* $(\lambda x . y ; x + z) (y \star z)$ **nitpick** [*expect=genuine*] **oops**
lemma *while-denest-3*: $(x \star w) \star x^\omega = (x \star w)^\omega$ **nitpick** [*expect=genuine*] **oops**
lemma *while-mult-top*: $(x ; T) \star z = z + x ; T$ **nitpick** [*expect=genuine*] **oops**
lemma *tarski-below-top-omega*: $x \le (x ; L)^\omega$ **nitpick** [*expect=genuine*] **oops**
lemma *tarski-mult-omega-omega*: $(x ; y^\omega)^\omega = x ; y^\omega$ **nitpick** [*expect=genuine*] **oops**
lemma *tarski-below-top-omega-2*: $x \le (x ; L) \star 0$ **nitpick** [*expect=genuine*] **oops**
lemma $1 = (x ; 0) \star 1$ **nitpick** [*expect=genuine*] **oops**
lemma *tarski*: $x = 0 \vee T ; x ; T = T$ **nitpick** [*expect=genuine*] **oops**
lemma $(x + y) \star z = ((x \star 1) ; y) \star ((x \star 1) ; z)$ **nitpick** [*expect=genuine*] **oops**
lemma *while-top-2*: $T \star z = T ; z$ **nitpick** [*expect=genuine*] **oops**
lemma *while-mult-top-2*: $(x ; T) \star z = z + x ; T ; z$ **nitpick** [*expect=genuine*] **oops**
lemma *while-one-mult*: $(x \star 1) ; x = x \star x$ **nitpick** [*expect=genuine*] **oops**
lemma $(x \star 1) ; y = x \star y$ **nitpick** [*expect=genuine*] **oops**
lemma *while-associative*: $(x \star y) ; z = x \star (y ; z)$ **nitpick** [*expect=genuine*] **oops**
lemma *while-back-loop-is-fixpoint*: *is-fixpoint* $(\lambda x . x ; y + z) (z ; (y \star 1))$ **nitpick** [*expect=genuine*] **oops**
lemma $1 + x ; 0 = x \star 1$ **nitpick** [*expect=genuine*] **oops**
lemma $x = x ; (x \star 1)$ **nitpick** [*expect=genuine*] **oops**
lemma $x ; (x \star 1) = x \star 1$ **nitpick** [*expect=genuine*] **oops**
lemma $x \star 1 = x \star (1 \star 1)$ **nitpick** [*expect=genuine*] **oops**
lemma $(x + y) \star 1 = (x \star (y \star 1)) \star 1$ **nitpick** [*expect=genuine*] **oops**
lemma $z + y ; x = x \longrightarrow y \star z \le x$ **nitpick** [*expect=genuine*] **oops**
lemma $y ; x = x \longrightarrow y \star x \le x$ **nitpick** [*expect=genuine*] **oops**
lemma $z + x ; y = x \longrightarrow z ; (y \star 1) \le x$ **nitpick** [*expect=genuine*] **oops**
lemma $x ; y = x \longrightarrow x ; (y \star 1) \le x$ **nitpick** [*expect=genuine*] **oops**
lemma $x ; z = z ; y \longrightarrow x \star z \le z ; (y \star 1)$ **nitpick** [*expect=genuine*] **oops**

lemma *while-unfold-below-1*: $x = y ; x \longrightarrow x \le y \star 1$ **nitpick** [*expect=genuine*] **oops**
lemma $x^\omega \le x^\omega ; x^\omega$ **oops**
lemma *tarski-omega-idempotent*: $x^{\omega\omega} = x^\omega$ **oops**

end

class *n-omega-algebra-binary-strict* = *n-omega-algebra-binary* + *circ* +

assumes *L-left-zero*: $L \; ; \; x = L$
assumes *circ-def*: $x^\circ = n(x^\omega) \; ; \; L + x^\star$

begin

— Theorem 2.7 and Theorem 50.5

subclass *strict-binary-itering*
apply *unfold-locales*
apply (*metis while-def mult-associative L-left-zero mult-right-dist-add*)
apply (*metis circ-def while-def mult-right-one*)
done

end

end

24 RelationAlgebra

theory *RelationAlgebra*

imports *Fixpoint*

begin

context *boolean-algebra*

begin

notation
  inf (**infixl** ⊓ *70*) **and**
  sup (**infixl** ⊔ *65*) **and**
  uminus (- ′ [*80*] *80*)

— We follow Roger Maddux's 1996 paper.
Maddux Axioms (Ba1) *sup-assoc* (Ba2) *sup-commute*
Maddux Theorem 3 (ii) *double-compl* (vi) *sup-idem* (vii) *inf-idem* (viii) *inf-commute* (ix) *inf-assoc* (xiv) *sup-inf-absorb* (xv) *inf-sup-distrib1* (xvi) *compl-sup* (xvii) *compl-inf* (xviii) *sup-inf-distrib1*
Maddux Theorem 5 (i) *sup-compl-top* (ii) *inf-compl-bot* (iii) *compl-top-eq* (iv) *compl-bot-eq* (v) *sup-top-right* (vi) *inf-bot-right* (vii) *sup-bot-right* (viii) *inf-top-right*

— Maddux Theorem 7(v)

lemma *shunting-1*: $x \leq y \longleftrightarrow x \sqcap y\ ' = bot$
  apply *rule*
  apply (*smt compl-inf-bot inf-absorb1 inf-bot-right inf-commute inf-left-commute*)
  apply (*metis inf-commute inf-sup-distrib1 inf-top-left le-iff-inf sup-commute sup-bot-left sup-compl-top*)
  done

lemma *shunting-2*: $x \leq y \longleftrightarrow x\ ' \sqcup y = top$
  by (*metis compl-bot-eq compl-inf double-compl shunting-1*)

— Maddux Definition 13

definition *conjugate* :: $('a \Rightarrow 'a) \Rightarrow ('a \Rightarrow 'a) \Rightarrow bool$
  where $conjugate\ f\ g \longleftrightarrow (\forall x\ y\ .\ f\ x \sqcap y = bot \longleftrightarrow x \sqcap g\ y = bot)$

— Maddux Theorem 14

lemma *conjugate-unique*: $conjugate\ f\ g \wedge conjugate\ f\ h \longrightarrow g = h$
proof
  assume $conjugate\ f\ g \wedge conjugate\ f\ h$
  hence $\forall x\ y\ .\ g\ y \leq x\ ' \longleftrightarrow h\ y \leq x\ '$
    by (*smt double-compl inf-commute shunting-1 conjugate-def*)
  hence $\forall y\ .\ g\ y = h\ y$
    by (*metis double-compl eq-iff*)
  thus $g = h$
    by (*metis lifted-antisymmetric lifted-less-eq-def order-refl*)
qed

lemma *conjugate-symmetric*: $conjugate\ f\ g \longrightarrow conjugate\ g\ f$
  by (*smt conjugate-def inf-commute*)

— Maddux Definition 15(iii)

definition *additive* :: $('a \Rightarrow 'a) \Rightarrow bool$
  where $additive\ f \longleftrightarrow (\forall x\ y\ .\ f\ (x \sqcup y) = f\ x \sqcup f\ y)$

— part of Maddux Theorem 17(i)

lemma *additive-isotone*: $additive\ f \longrightarrow isotone\ f$
  by (*metis additive-def isotone-def le-iff-sup*)

— part of Maddux Theorem 18(ii)

lemma *conjugate-additive*: $conjugate\ f\ g \longrightarrow additive\ f$

```
proof
  assume 1: conjugate f g
  have 2: ∀ x y z . f (x ⊔ y) ≤ z ⟷ f x ≤ z ∧ f y ≤ z
  proof
    fix x
    show ∀ y z . f (x ⊔ y) ≤ z ⟷ f x ≤ z ∧ f y ≤ z
    proof
      fix y
      show ∀ z . f (x ⊔ y) ≤ z ⟷ f x ≤ z ∧ f y ≤ z
      proof
        fix z
        have (f (x ⊔ y) ≤ z) = (f (x ⊔ y) ⊓ z ′ = bot)
          by (metis shunting-1)
        also have ... = ((x ⊔ y) ⊓ g(z ′) = bot) using 1
          by (metis conjugate-def)
        also have ... = (x ⊔ y ≤ (g(z ′))′)
          by (metis double-compl shunting-1)
        also have ... = (x ≤ (g(z ′))′ ∧ y ≤ (g(z ′))′)
          by (metis le-sup-iff)
        also have ... = (x ⊓ g(z ′) = bot ∧ y ⊓ g(z ′) = bot)
          by (metis double-compl shunting-1)
        also have ... = (f x ⊓ z ′ = bot ∧ f y ⊓ z ′ = bot) using 1
          by (metis conjugate-def)
        also have ... = (f x ≤ z ∧ f y ≤ z)
          by (metis shunting-1)
        finally show f (x ⊔ y) ≤ z ⟷ f x ≤ z ∧ f y ≤ z
          by metis
      qed
    qed
  qed
  have ∀ x y . f (x ⊔ y) = f x ⊔ f y
  proof
    fix x
    show ∀ y . f (x ⊔ y) = f x ⊔ f y
    proof
      fix y
      have f(x ⊔ y) ≤ f(x) ⊔ f(y) using 2
        by (metis sup-ge1 sup-ge2)
      thus f (x ⊔ y) = f x ⊔ f y using 2
        by (metis le-supI order-refl antisym)
    qed
  qed
  thus additive f
    by (metis additive-def)
qed

lemma conjugate-isotone: conjugate f g ⟶ isotone f
  by (metis additive-isotone conjugate-additive)
```

— Maddux Theorem 19

```
lemma conjugate-char-1: conjugate f g ⟷ (∀ x y . f(x ⊓ (g y)′) ≤ f x ⊓ y ′ ∧ g(y ⊓ (f x)′) ≤ g y ⊓ x ′)
proof
  assume 1: conjugate f g
  show ∀ x y . f(x ⊓ (g y)′) ≤ f x ⊓ y ′ ∧ g(y ⊓ (f x)′) ≤ g y ⊓ x ′
  proof
    fix x
    show ∀ y . f(x ⊓ (g y)′) ≤ f x ⊓ y ′ ∧ g(y ⊓ (f x)′) ≤ g y ⊓ x ′
    proof
      fix y
      have f(x ⊓ (g y)′) ≤ y ′ using 1
        by (smt compl-inf-bot conjugate-def double-compl inf-assoc inf-bot-right shunting-1)
      hence 2: f(x ⊓ (g y)′) ≤ f x ⊓ y ′ using 1
        by (metis conjugate-isotone inf-le1 isotone-def le-inf-iff)
      have g(y ⊓ (f x)′) ≤ x ′ using 1
        by (smt compl-inf-bot conjugate-def double-compl inf-assoc inf-bot-right shunting-1 inf-commute)
      hence g(y ⊓ (f x)′) ≤ g y ⊓ x ′ using 1
        by (metis conjugate-isotone inf-le1 isotone-def le-inf-iff conjugate-symmetric)
      thus f(x ⊓ (g y)′) ≤ f x ⊓ y ′ ∧ g(y ⊓ (f x)′) ≤ g y ⊓ x ′ using 2
      by metis
    qed
  qed
next
  assume ∀ x y . f(x ⊓ (g y)′) ≤ f x ⊓ y ′ ∧ g(y ⊓ (f x)′) ≤ g y ⊓ x ′
  thus conjugate f g
    by (smt conjugate-def double-compl inf-commute inf-le2 le-iff-inf shunting-1)
qed

lemma conjugate-char-2: conjugate f g ⟷ f bot = bot ∧ g bot = bot ∧ (∀ x y . f x ⊓ y ≤ f(x ⊓ g y) ∧ g y ⊓ x ≤ g(y ⊓ f x))
proof
  assume 1: conjugate f g
  show f bot = bot ∧ g bot = bot ∧ (∀ x y . f x ⊓ y ≤ f(x ⊓ g y) ∧ g y ⊓ x ≤ g(y ⊓ f x))
  proof
    show f bot = bot using 1
      by (metis conjugate-def inf-idem inf-bot-left)
  next
    show g bot = bot ∧ (∀ x y . f x ⊓ y ≤ f(x ⊓ g y) ∧ g y ⊓ x ≤ g(y ⊓ f x))
    proof
      show g bot = bot using 1
        by (metis conjugate-def inf-idem inf-bot-right)
    next
      show ∀ x y . f x ⊓ y ≤ f(x ⊓ g y) ∧ g y ⊓ x ≤ g(y ⊓ f x)
      proof
        fix x
        show ∀ y . f x ⊓ y ≤ f(x ⊓ g y) ∧ g y ⊓ x ≤ g(y ⊓ f x)
        proof
          fix y
          show f x ⊓ y ≤ f(x ⊓ g y) ∧ g y ⊓ x ≤ g(y ⊓ f x)
          proof
            have f x ⊓ y = (f(x ⊓ g y) ⊔ f(x ⊓ (g y)′)) ⊓ y using 1
              by (metis additive-def conjugate-additive inf-sup-distrib1 inf-top-right sup-compl-top)
            also have ... ≤ (f(x ⊓ g y) ⊔ (f x ⊓ y ′)) ⊓ y using 1
              by (metis conjugate-char-1 inf-mono order-refl sup-mono)
            also have ... ≤ f(x ⊓ g y)
              by (smt inf-idem inf-assoc inf-commute inf-compl-bot inf-sup-distrib1 le-iff-inf sup-commute sup-bot-left)
            finally show f x ⊓ y ≤ f(x ⊓ g y)
              by metis
          next
            have g y ⊓ x = (g(y ⊓ f x) ⊔ g(y ⊓ (f x)′)) ⊓ x using 1
              by (metis additive-def conjugate-additive conjugate-symmetric inf-sup-distrib1 inf-top-right sup-compl-top)
            also have ... ≤ (g(y ⊓ f x) ⊔ (g y ⊓ x ′)) ⊓ x using 1
              by (metis conjugate-char-1 inf-mono order-refl sup-mono)
            also have ... ≤ g(y ⊓ f x)
              by (smt inf-idem inf-assoc inf-commute inf-compl-bot inf-sup-distrib1 le-iff-inf sup-commute sup-bot-left)
            finally show g y ⊓ x ≤ g(y ⊓ f x)
              by metis
          qed
        qed
      qed
    qed
  qed
next
  assume f bot = bot ∧ g bot = bot ∧ (∀ x y . f x ⊓ y ≤ f(x ⊓ g y) ∧ g y ⊓ x ≤ g(y ⊓ f x))
  thus conjugate f g
    by (smt conjugate-def inf-commute le-bot)
qed

end

class conv =
  fixes conv :: ′a ⇒ ′a (-˘ [100] 100)
```

— Maddux Axioms (Ra1)-(Ra7)

```
class relation-algebra = boolean-algebra + mult + one + conv +
  assumes comp-associative   : (x ; y) ; z = x ; (y ; z)
  assumes comp-right-dist-sup: (x ⊔ y) ; z = (x ; z) ⊔ (y ; z)
  assumes comp-right-one     : x ; 1 = x
  assumes conv-involutive    : x⌣⌣ = x
  assumes conv-dist-sup      : (x ⊔ y)⌣ = x⌣ ⊔ y⌣
  assumes conv-dist-comp     : (x ; y)⌣ = y⌣ ; x⌣
  assumes conv-complement-sub: x⌣ ; (x ; y)′ ⊔ y ′ = y ′

begin

— most of Maddux Theorem 24 and a few other facts

lemma conv-order: x ≤ y ⟷ x⌣ ≤ y⌣
  by (metis conv-dist-sup conv-involutive le-iff-sup)

lemma conv-zero: bot⌣ = bot
  by (metis conv-dist-sup conv-involutive sup-bot-right sup-eq-bot-iff)

lemma conv-top: top⌣ = top
  by (metis conv-involutive conv-order eq-iff top-greatest)

lemma conv-complement-0: x⌣ ⊔ (x ′)⌣ = top
  by (metis conv-dist-sup conv-top sup-compl-top)

lemma conv-complement-1: (x⌣)′ ⊔ (x ′)⌣ = (x ′)⌣
 by (smt compl-sup-top conv-dist-sup conv-top inf-compl-bot sup-idem sup-bot-right sup-commute sup-inf-distrib1 sup-top-right)

lemma conv-complement: (x ′)⌣ = (x⌣)′
  by (metis conv-complement-1 conv-dist-sup conv-involutive sup-commute)

lemma conv-dist-inf: (x ⊓ y)⌣ = x⌣ ⊓ y⌣
  by (smt conv-complement compl-inf double-compl conv-dist-sup)

lemma conv-meet-zero-iff: bot = x⌣ ⊓ y ⟷ bot = x ⊓ y⌣
  by (metis conv-dist-inf conv-involutive conv-zero)

lemma conv-one: 1⌣ = 1
  by (metis comp-right-one conv-dist-comp conv-involutive)

lemma comp-left-dist-sup: (x ; y) ⊔ (x ; z) = x ; (y ⊔ z)
  by (metis comp-right-dist-sup conv-involutive conv-dist-sup conv-dist-comp)

lemma comp-right-isotone: x ≤ y ⟶ z ; x ≤ z ; y
  by (metis comp-left-dist-sup le-iff-sup)

lemma comp-left-isotone: x ≤ y ⟶ x ; z ≤ y ; z
  by (metis comp-right-dist-sup le-iff-sup)

lemma comp-left-conjugate: conjugate (λy . x ; y) (λy . x⌣ ; y)
proof –
  let ?f = λy . x ; y
  let ?g = λy . x⌣ ; y
  have ∀ z y . ?f(z ⊓ (?g y)′) ≤ ?f z ⊓ y ′ ∧ ?g(y ⊓ (?f z)′) ≤ ?g y ⊓ z ′
  proof
    fix z
    show ∀ y . ?f(z ⊓ (?g y)′) ≤ ?f z ⊓ y ′ ∧ ?g(y ⊓ (?f z)′) ≤ ?g y ⊓ z ′
    proof
      fix y
      show ?f(z ⊓ (?g y)′) ≤ ?f z ⊓ y ′ ∧ ?g(y ⊓ (?f z)′) ≤ ?g y ⊓ z ′
      proof
        have ?f(z ⊓ (?g y)′) ≤ ?f(z) ⊓ ?f((?g y)′)
          by (metis comp-right-isotone inf-greatest inf-le1 inf-le2)
        also have ... ≤ ?f(z) ⊓ y ′
          by (metis conv-complement-sub conv-involutive inf-mono le-iff-sup order-refl)
        finally show ?f(z ⊓ (?g y)′) ≤ ?f(z) ⊓ y ′
          by metis
      next
        have ?g(y ⊓ (?f z)′) ≤ ?g(y) ⊓ ?g((?f z)′)
          by (metis comp-right-isotone inf-greatest inf-le1 inf-le2)
        also have ... ≤ ?g(y) ⊓ z ′
          by (metis conv-complement-sub inf-mono le-iff-sup order-refl)
        finally show ?g(y ⊓ (?f z)′) ≤ ?g y ⊓ z ′
        by metis
      qed
    qed
  qed
  thus conjugate ?f ?g
    by (metis conjugate-char-1)
qed

lemma complement-conv-sub: (y ; x)′ ; x˘ ≤ y ′
  by (smt conv-complement-sub conv-order conv-involutive conv-dist-comp conv-complement le-iff-sup)

lemma comp-right-conjugate: conjugate (λy . y ; x) (λy . y ; x˘)
proof –
  let ?f = λy . y ; x
  let ?g = λy . y ; x˘
  have ∀ z y . ?f(z ⊓ (?g y)′) ≤ ?f z ⊓ y ′ ∧ ?g(y ⊓ (?f z)′) ≤ ?g y ⊓ z ′
  proof
    fix z
    show ∀ y . ?f(z ⊓ (?g y)′) ≤ ?f z ⊓ y ′ ∧ ?g(y ⊓ (?f z)′) ≤ ?g y ⊓ z ′
    proof
      fix y
      show ?f(z ⊓ (?g y)′) ≤ ?f z ⊓ y ′ ∧ ?g(y ⊓ (?f z)′) ≤ ?g y ⊓ z ′
      proof
        have ?f(z ⊓ (?g y)′) ≤ ?f(z) ⊓ ?f((?g y)′)
          by (metis comp-left-isotone inf-greatest inf-le1 inf-le2)
        also have ... ≤ ?f(z) ⊓ y ′
          by (metis complement-conv-sub conv-involutive inf-mono order-refl)
        finally show ?f(z ⊓ (?g y)′) ≤ ?f(z) ⊓ y ′
          by metis
      next
        have ?g(y ⊓ (?f z)′) ≤ ?g(y) ⊓ ?g((?f z)′)
          by (metis comp-left-isotone inf-greatest inf-le1 inf-le2)
        also have ... ≤ ?g(y) ⊓ z ′
          by (metis complement-conv-sub inf-mono order-refl)
        finally show ?g(y ⊓ (?f z)′) ≤ ?g y ⊓ z ′
          by metis
      qed
    qed
  qed
  thus conjugate ?f ?g
    by (smt conjugate-char-1)
qed

lemma schroeder-1: x ; y ⊓ z = bot ⟷ x˘ ; z ⊓ y = bot
  by (smt comp-left-conjugate conjugate-def inf-commute)

lemma schroeder-2: x ; y ⊓ z = bot ⟷ z ; y˘ ⊓ x = bot
  by (smt comp-right-conjugate conjugate-def inf-commute)

lemma schroeder-3: x ; y ≤ z ⟷ x˘ ; z ′ ≤ y ′
  by (metis double-compl schroeder-1 shunting-1)

lemma schroeder-4: x ; y ≤ z ⟷ z ′ ; y˘ ≤ x ′
  by (metis double-compl schroeder-2 shunting-1)

lemma dedekind-1: x ; y ⊓ z ≤ x ; (y ⊓ (x˘ ; z))
  by (metis comp-left-conjugate conjugate-char-2)

lemma dedekind-2: y ; x ⊓ z ≤ (y ⊓ (z ; x˘)) ; x
  by (smt comp-right-conjugate conjugate-char-2)

lemma comp-left-zero: bot ; x = bot
  by (metis comp-right-conjugate conjugate-char-2)

lemma comp-right-zero: x ; bot = bot
  by (metis comp-left-conjugate conjugate-char-2)

lemma comp-left-one: 1 ; x = x
  by (metis comp-right-one conv-dist-comp conv-involutive)
```

lemma *comp-right-top-increasing*: $x \le x ; top$
  by (*metis comp-right-isotone comp-right-one top-greatest*)

lemma *comp-left-top-increasing*: $x \le top ; x$
  by (*metis comp-left-isotone comp-left-one top-greatest*)

lemma *top-top*: $top ; top = top$
  by (*metis comp-left-top-increasing top-unique*)

lemma *theorem24xxiii*: $x ; y \sqcap (x ; z)' = x ; (y \sqcap z ') \sqcap (x ; z)'$
proof –
  have $x ; y \sqcap (x ; z)' \le x ; (y \sqcap (x^{\smile} ; (x ; z)'))$
    by (*metis dedekind-1*)
  also have ... $\le x ; (y \sqcap z ')$
    by (*metis comp-right-isotone conv-complement-sub inf-mono le-iff-sup order-refl*)
  finally have $x ; y \sqcap (x ; z)' \le x ; (y \sqcap z ') \sqcap (x ; z)'$
    by (*metis inf-le2 le-inf-iff*)
  thus *?thesis*
    by (*metis comp-right-isotone eq-iff inf-commute inf-le1 le-infI le-infI2*)
qed

lemma *theorem24xxiv*: $(x ; y)' \sqcup (x ; z) = (x ; (y \sqcap z '))' \sqcup (x ; z)$
  by (*metis compl-inf double-compl theorem24xxiii*)

lemma *vector-complement*: $x = x ; top \longrightarrow x ' = x ' ; top$
  by (*metis comp-right-top-increasing complement-conv-sub conv-top eq-iff*)

lemma *vector-meet-comp*: $x = x ; top \longrightarrow (x \sqcap y) ; z = x \sqcap (y ; z)$
proof
  assume *1*: $x = x ; top$
  hence $(x \sqcap y) ; z \le x \sqcap (y ; z)$
    by (*metis comp-left-isotone comp-right-isotone inf-assoc inf-commute inf-le2 le-iff-inf le-infI top-greatest*)
  thus $(x \sqcap y) ; z = x \sqcap (y ; z)$ **using** *1*
    by (*smt antisym comp-left-isotone comp-right-isotone dedekind-2 inf-commute inf-mono order-refl order-trans top-greatest*)
qed

lemma *vector-meet*: $x = x ; top \wedge y = y ; top \longrightarrow x \sqcap y = (x \sqcap y) ; top$
  by (*metis vector-meet-comp*)

lemma *vector-meet-one-comp*: $x = x ; top \longrightarrow (x \sqcap 1) ; y = x \sqcap y$
  by (*metis comp-left-one vector-meet-comp*)

lemma *covector-meet-comp-1*: $x = x ; top \longrightarrow (y \sqcap x^{\smile}) ; z = (y \sqcap x^{\smile}) ; (x \sqcap z)$
proof
  assume *1*: $x = x ; top$
  have $(y \sqcap x^{\smile}) ; z \le (y \sqcap x^{\smile}) ; (z \sqcap ((y^{\smile} \sqcap x) ; top))$
    by (*metis inf-top-right dedekind-1 conv-dist-inf conv-involutive*)
  also have ... $\le (y \sqcap x^{\smile}) ; (x \sqcap z)$ **using** *1*
    by (*metis comp-left-isotone comp-right-isotone inf-le2 inf-mono order-refl inf-commute*)
  finally show $(y \sqcap x^{\smile}) ; z = (y \sqcap x^{\smile}) ; (x \sqcap z)$
    by (*metis comp-right-isotone eq-iff inf-le2*)
qed

lemma *covector-meet-comp-2*: $x = x ; top \longrightarrow y ; (x \sqcap z) = (y \sqcap x^{\smile}) ; (x \sqcap z)$
proof
  assume *1*: $x = x ; top$
  have $y ; (x \sqcap z) \le (y \sqcap (top ; (x \sqcap z)^{\smile})) ; (x \sqcap z)$
    by (*metis dedekind-2 inf-top-right*)
  also have ... $\le (y \sqcap x^{\smile}) ; (x \sqcap z)$ **using** *1*
    by (*metis comp-left-isotone conv-dist-comp conv-order conv-top eq-refl inf-le1 inf-mono*)
  finally show $y ; (x \sqcap z) = (y \sqcap x^{\smile}) ; (x \sqcap z)$
    by (*metis comp-left-isotone eq-iff inf-le1*)
qed

lemma *coreflexive-conv*: $x \le 1 \longrightarrow x^{\smile} = x$
proof
  assume *1*: $x \le 1$
  hence $x \le x ; (1 \sqcap (x^{\smile} ; 1))$
    by (*metis comp-right-one le-iff-inf dedekind-1*)

also have ... $\leq x^{\smile}$ **using** *1*
  by (*metis comp-left-isotone comp-right-one conv-dist-inf conv-one inf-absorb2 comp-left-one*)
thus $x^{\smile} = x$
  by (*metis antisym calculation conv-involutive conv-order order-trans*)
qed

lemma *coreflexive-comp-top-meet*: $x \leq 1 \longrightarrow x \ ; \ top \sqcap y = x \ ; \ y$
proof
assume *1*: $x \leq 1$
hence $x \ ; \ top \sqcap y \leq x \ ; \ y$
  by (*metis comp-left-isotone comp-left-one coreflexive-conv dedekind-1 inf-top-left order-trans*)
thus $x \ ; \ top \sqcap y = x \ ; \ y$ **using** *1*
  by (*metis antisym comp-left-isotone comp-left-one comp-right-isotone le-inf-iff top-greatest*)
qed

lemma *coreflexive-comp-top-complement-meet-one*: $x \leq 1 \longrightarrow (x \ ; \ top)' \sqcap 1 = x\ ' \sqcap 1$
proof
assume *1*: $x \leq 1$
hence *2*: $x \ ; \ x^{\smile} \ ; \ (x\ ' \sqcap 1) \leq 1 \ ; \ 1 \ ; \ x\ '$
  by (*metis comp-left-one coreflexive-comp-top-meet coreflexive-conv inf-commute inf-idem le-iff-inf le-infI2*)
have *3*: $x \ ; \ x^{\smile} \ ; \ (x\ ' \sqcap 1) \leq x \ ; \ 1 \ ; \ 1$ **using** *1*
  by (*metis comp-left-isotone comp-right-isotone inf-le2 order-trans coreflexive-conv*)
have $x\ ' \sqcap 1 \sqcap (x \ ; \ top) \leq x \ ; \ x^{\smile} \ ; \ (x\ ' \sqcap 1)$
  by (*metis dedekind-1 inf-commute comp-associative inf-top-left*)
also have ... $\leq bot$ **using** *2 3*
  by (*metis le-inf-iff comp-left-one comp-right-one compl-inf-bot*)
finally have $x\ ' \sqcap 1 \leq (x \ ; \ top)' \sqcap 1$
  by (*metis bot-unique double-compl shunting-1 inf-le2 le-inf-iff*)
thus $(x \ ; \ top)' \sqcap 1 = x\ ' \sqcap 1$
  by (*metis antisym comp-right-top-increasing compl-le-compl-iff inf-mono order-refl*)
qed

lemma *coreflexive-comp-meet*: $x \leq 1 \wedge y \leq 1 \longrightarrow x \ ; \ y = x \sqcap y$
  by (*smt comp-right-one coreflexive-comp-top-meet inf-absorb1 inf-left-commute*)

lemma *coreflexive-comp-meet-comp*: $x \leq 1 \wedge y \leq 1 \longrightarrow (x \ ; \ z) \sqcap (y \ ; \ z) = (x \sqcap y) \ ; \ z$
  by (*smt comp-associative comp-left-isotone comp-right-one coreflexive-comp-top-meet inf-left-commute le-iff-inf*)

lemma *coreflexive-comp-meet-complement*: $x \leq 1 \longrightarrow (x \ ; \ y) \sqcap z\ ' = (x \ ; \ y) \sqcap (x \ ; \ z)'$
  by (*smt compl-le-compl-iff coreflexive-comp-top-meet inf-assoc inf-commute inf-left-idem inf-top-left le-iff-inf theorem24xxiii*)

lemma *vector-export-comp*: $(x \ ; \ top \sqcap 1) \ ; \ y = x \ ; \ top \sqcap y$
  by (*metis comp-associative top-top vector-meet-one-comp*)

— states with infinite executions of non-strict computations

abbreviation N :: $'a \Rightarrow 'a$
  where $N(x) \equiv ((x\ ') \ ; \ top)' \sqcap 1$

lemma *N-comp*: $N(x) \ ; \ y = ((x\ ') \ ; \ top)' \sqcap y$
  by (*metis comp-associative top-top vector-complement vector-export-comp*)

lemma *N-comp-top*: $N(x) \ ; \ top = ((x\ ') \ ; \ top)'$
  by (*metis N-comp inf-top-right*)

lemma *vector-N*: $x = x \ ; \ top \longrightarrow N(x) = x \sqcap 1$
  by (*metis double-compl vector-complement*)

lemma *N-vector*: $N(x \ ; \ top) = x \ ; \ top \sqcap 1$
  by (*metis comp-associative top-top vector-N*)

lemma *N-vector-top*: $N(x \ ; \ top) \ ; \ top = x \ ; \ top$
  by (*metis N-vector inf-top-right vector-export-comp*)

lemma *N-below-meet-one*: $N(x) \leq x \sqcap 1$
  by (*metis comp-right-top-increasing compl-le-swap2 inf-commute inf-le1 inf-mono le-inf-iff*)

lemma *N-below*: $N(x) \leq x$
  by (*metis N-below-meet-one le-infE*)

lemma *N-comp-N*: $N(x)\ ;\ N(y) = ((x\ ')\ ;\ top)' \sqcap ((y\ ')\ ;\ top)' \sqcap 1$
  by (*metis N-comp inf-assoc*)

lemma *N-zero*: $N(bot) = bot$
  by (*metis compl-bot-eq compl-top-eq inf-bot-left top-top*)

lemma *N-top*: $N(top) = 1$
  by (*metis inf-top-left top-top vector-N*)

lemma *n-split-omega-mult*: $xs\ ;\ xo = xo \wedge xo\ ;\ top = xo \longrightarrow N(top)\ ;\ xo = xs\ ;\ N(xo)\ ;\ top$
  by (*metis N-top N-vector-top comp-associative comp-left-one*)

end

end

25 NAlgebraRelationAlgebra

```
theory NAlgebraRelationAlgebra

imports NOmegaAlgebra RelationAlgebra

begin

sublocale relation-algebra < bounded-idempotent-semiring where plus = sup and zero = bot and T = top
  apply unfold-locales
  apply (metis sup-assoc)
  apply (metis sup-commute)
  apply (metis sup-idem)
  apply (metis le-iff-sup)
  apply (metis less-le-not-le)
  apply (metis sup-bot-left)
  apply (metis comp-left-dist-sup order-refl)
  apply (metis comp-right-dist-sup)
  apply (metis comp-left-zero)
  apply (metis comp-right-one conv-dist-comp conv-involutive)
  apply (metis comp-right-one order-refl)
  apply (metis comp-associative order-refl)
  apply (metis sup-top-right)
  apply (metis comp-associative)
  apply (metis comp-right-one)
  apply (metis comp-left-dist-sup)
  apply (metis comp-right-zero)
  done

sublocale relation-algebra < lattice-ordered-pre-left-semiring where plus = sup and zero = bot and T = top and meet =
inf
  apply unfold-locales
  apply (metis inf-assoc)
  apply (metis inf-commute)
  apply (metis inf-idem)
  apply (metis inf.order-iff)
  apply (metis less-le-not-le)
  apply (metis inf-top-left)
  apply (metis inf-sup-distrib1)
  apply (metis sup-inf-distrib1)
  apply (metis inf-sup-absorb)
  apply (metis sup-inf-absorb)
  done
```

— Theorem 37

```
sublocale relation-algebra < n-algebra where plus = sup and zero = bot and T = top and meet = inf and n = N and L
= top
  apply unfold-locales
  apply (metis N-comp-top comp-associative compl-inf double-compl inf-sup-distrib2 top-top vector-meet-comp)
  apply (metis N-comp compl-sup double-compl mult-associative mult-right-dist-add top-top N-comp-N)
  apply (metis N-comp-N compl-inf compl-sup meet-absorb mult-right-dist-add)
  apply (metis N-top inf-idem meet.add-right-upper-bound)
  apply (metis N-comp-top compl-le-swap2 top-right-mult-increasing)
  apply (metis N-top eq-refl mult-left-one mult-right-one)
  apply (metis N-top N-zero comp-right-zero mult-left-one mult-left-zero meet.add-right-zero sup-bot-right)
  apply (metis N-vector-top comp-right-zero sup-bot-left)
 apply (metis N-comp-top conv-complement-sub double-compl le-supI2 less-eq-def mult-associative mult-left-isotone schroeder-3)
  apply (metis meet.add-left-upper-bound)
  done

sublocale relation-algebra < n-algebra-apx where plus = sup and zero = bot and T = top and meet = inf and n = N
and L = top and apx = greater-eq
  apply unfold-locales
  apply (metis N-top mult-left-one n-less-eq-char sup-top-right top-greatest)
  done

class left-residuated-relation-algebra = relation-algebra + inverse +
  assumes lres-def: x / y = (x ′ ; y⌣)′
```

— Theorem 32.1

sublocale *left-residuated-relation-algebra* < *residuated-pre-left-semiring* **where** *plus* = *sup* **and** *zero* = *bot*
  apply *unfold-locales*
  apply (*metis compl-le-swap1 lres-def schroeder-4*)
  done

context *left-residuated-relation-algebra*

begin

— Theorem 32.3

lemma *lres-mult-lres-lres*: $x\ /\ (z\ ;\ y) = (x\ /\ y)\ /\ z$
  by (*metis conv-dist-comp double-compl lres-def mult-associative*)

— Theorem 32.5

lemma *lres-dist-meet*: $(x \sqcap y)\ /\ z = (x\ /\ z) \sqcap (y\ /\ z)$
  by (*metis compl-inf compl-sup lres-def mult-right-dist-add*)

— Theorem 32.6

lemma *lres-add-export-vector*: *vector* $x \longrightarrow (x \sqcup y)\ /\ z = x \sqcup (y\ /\ z)$
proof
  assume *1*: *vector* x
  have $(x \sqcup y)\ /\ z = ((x\ ' \sqcap y\ ')\ ;\ z^{\smile})'$
    by (*metis lres-def compl-sup*)
  also have ... $= (x\ ' \sqcap (y\ '\ ;\ z^{\smile}))'$ **using** *1*
    by (*metis vector-complement vector-def vector-meet-comp*)
  also have ... $= x \sqcup (y\ /\ z)$
    by (*metis compl-inf double-compl lres-def*)
  finally show $(x \sqcup y)\ /\ z = x \sqcup (y\ /\ z)$
    .
qed

— Theorem 32.7

lemma *lres-top-vector*: *vector* $(x\ /\ top)$
  by (*metis eq-iff lres-inverse top-right-mult-increasing top-top vector-def lres-mult-lres-lres*)

— Theorem 32.10

lemma *lres-top-export-meet-mult*: $((x\ /\ top) \sqcap y)\ ;\ z = (x\ /\ top) \sqcap (y\ ;\ z)$
  by (*metis vector-def vector-meet-comp lres-top-vector*)

lemma *N-lres*: $N(x) = x\ /\ top \sqcap 1$
  by (*metis conv-top lres-def*)

end

class *complete-relation-algebra* = *relation-algebra* + *complete-lattice*

begin

definition *mu* :: $('a \Rightarrow 'a) \Rightarrow 'a$ **where** $mu\, f = Inf\ \{\ y\ .\ f\, y \le y\ \}$
definition *nu* :: $('a \Rightarrow 'a) \Rightarrow 'a$ **where** $nu\, f = Sup\ \{\ y\ .\ y \le f\, y\ \}$

lemma *mu-lower-bound*: $f\, x \le x \longrightarrow mu\, f \le x$
  by (*auto simp add*: *mu-def intro*: *Inf-lower*)

lemma *mu-greatest-lower-bound*: $(\forall\, y\ .\ f\, y \le y \longrightarrow x \le y) \longrightarrow x \le mu\, f$
  by (*auto simp add*: *mu-def intro*: *Inf-greatest*)

lemma *mu-unfold-1*: *isotone* $f \longrightarrow f\ (mu\, f) \le mu\, f$
  by (*metis mu-greatest-lower-bound order-trans mu-lower-bound isotone-def*)

lemma *mu-unfold-2*: *isotone* $f \longrightarrow mu\, f \le f\ (mu\, f)$
  by (*metis mu-unfold-1 isotone-def mu-lower-bound*)

lemma *mu-unfold*: *isotone f* $\longrightarrow$ *mu f* = *f* (*mu f*)
  by (*metis antisym mu-unfold-1 mu-unfold-2*)

lemma *mu-const*: *mu* (λx . *y*) = *y*
  by (*metis isotone-def mu-unfold order-refl*)

lemma *mu-lpfp*: *isotone f* $\longrightarrow$ *is-least-prefixpoint f* (*mu f*)
  by (*metis is-least-prefixpoint-def mu-lower-bound mu-unfold-1*)

lemma *mu-lfp*: *isotone f* $\longrightarrow$ *is-least-fixpoint f* (*mu f*)
  by (*metis is-least-fixpoint-def mu-lower-bound mu-unfold order-refl*)

lemma *mu-pmu*: *isotone f* $\longrightarrow$ *p*μ *f* = *mu f*
  by (*metis least-prefixpoint-char mu-lpfp*)

lemma *mu-mu*: *isotone f* $\longrightarrow$ μ *f* = *mu f*
  by (*metis least-fixpoint-char mu-lfp*)

end

class *omega-relation-algebra* = *relation-algebra* + *star* + *omega* +
  assumes *ra-star-left-unfold* : $1 \sqcup y \ ; \ y^\star \leq y^\star$
  assumes *ra-star-left-induct* : $z \sqcup y \ ; \ x \leq x \longrightarrow y^\star \ ; \ z \leq x$
  assumes *ra-star-right-induct*: $z \sqcup x \ ; \ y \leq x \longrightarrow z \ ; \ y^\star \leq x$
  assumes *ra-omega-unfold*: $y^\omega = y \ ; \ y^\omega$
  assumes *ra-omega-induct*: $x \leq z \sqcup y \ ; \ x \longrightarrow x \leq y^\omega \sqcup y^\star \ ; \ z$

sublocale *omega-relation-algebra* < *bounded-omega-algebra* **where** *plus* = *sup* **and** *zero* = *bot* **and** *T* = *top*
  apply *unfold-locales*
  apply (*metis ra-star-left-unfold*)
  apply (*metis ra-star-left-induct*)
  apply (*metis ra-star-right-induct*)
  apply (*metis ra-omega-unfold*)
  apply (*metis ra-omega-induct*)
  done

— Theorem 38

sublocale *omega-relation-algebra* < *n-omega-algebra* **where** *plus* = *sup* **and** *zero* = *bot* **and** *T* = *top* **and** *meet* = *inf* **and** *n* = *N* **and** *L* = *top* **and** *apx* = *greater-eq* **and** *Omega* = λx . $N(x^\omega)$; *top* $\sqcup$ $x^\star$
  apply *unfold-locales*
  apply *simp*
  apply (*metis n-split-omega-mult omega-vector order.refl star-mult-omega*)
  apply (*metis inf.absorb1 meet.eq-refl mult-associative top.extremum top-left-mult-increasing top-top vector-meet-comp*)
  done

end

26 Domain

```
theory Domain

imports Semiring Tests

begin

class left-zero-domain-semiring = idempotent-left-zero-semiring + d +
  assumes d-restrict: x + d(x) ; x = d(x) ; x
  assumes d-mult-d  : d(x ; y) = d(x ; d(y))
  assumes d-plus-one: d(x) + 1 = 1
  assumes d-zero    : d(0) = 0
  assumes d-dist-add: d(x + y) = d(x) + d(y)

begin
```

— Many lemmas in this class are taken from Georg Struth's theories.

```
lemma d-restrict-equals: x = d(x) ; x
  by (metis add-commutative d-plus-one d-restrict mult-left-one mult-right-dist-add)

lemma d-involutive: d(d(x)) = d(x)
  by (metis d-mult-d mult-left-one)

lemma d-fixpoint: (∃ y . x = d(y)) ⟷ x = d(x)
  by (metis d-involutive)

lemma d-type: ∀ P . (∀ x . x = d(x) ⟶ P(x)) ⟷ (∀ x . P(d(x)))
  by (metis d-involutive)

lemma d-mult-sub: d(x ; y) ≤ d(x)
  by (metis d-dist-add d-mult-d d-plus-one less-eq-def mult-left-sub-dist-add-left mult-right-one)

lemma d-sub-one: x ≤ 1 ⟶ x ≤ d(x)
  by (metis d-restrict-equals mult-right-isotone mult-right-one)

lemma d-strict: d(x) = 0 ⟷ x = 0
  by (metis d-restrict-equals d-zero mult-left-zero)

lemma d-one: d(1) = 1
  by (metis d-restrict-equals mult-right-one)

lemma d-below-one: d(x) ≤ 1
  by (metis d-plus-one less-eq-def)

lemma d-isotone: x ≤ y ⟶ d(x) ≤ d(y)
  by (metis d-dist-add less-eq-def)

lemma d-plus-left-upper-bound: d(x) ≤ d(x + y)
  by (metis add-left-upper-bound d-isotone)

lemma d-export: d(d(x) ; y) = d(x) ; d(y)
  by (smt add-commutative antisym d-mult-d d-mult-sub d-plus-left-upper-bound d-plus-one d-restrict d-sub-one mult-isotone
mult-left-one mult-left-sub-dist-add-right mult-right-dist-add mult-right-one)

lemma d-idempotent: d(x) ; d(x) = d(x)
  by (metis d-export d-restrict-equals)

lemma d-commutative: d(x) ; d(y) = d(y) ; d(x)
  by (smt antisym d-export d-mult-d d-mult-sub d-one d-restrict-equals mult-isotone mult-left-one)

lemma d-least-left-preserver: x ≤ d(y) ; x ⟷ d(x) ≤ d(y)
  by (metis d-below-one d-involutive d-mult-sub d-restrict-equals eq-iff mult-left-isotone mult-left-one)

lemma d-weak-locality: x ; y = 0 ⟷ x ; d(y) = 0
  by (metis d-mult-d d-strict)

lemma d-add-closed: d(d(x) + d(y)) = d(x) + d(y)
```

by (*metis d-dist-add d-involutive*)

lemma *d-mult-closed*: $d(d(x) ; d(y)) = d(x) ; d(y)$
by (*metis d-export d-mult-d*)

lemma *d-mult-left-lower-bound*: $d(x) ; d(y) \leq d(x)$
by (*metis d-export d-involutive d-mult-sub*)

lemma *d-mult-greatest-lower-bound*: $d(x) \leq d(y) ; d(z) \longleftrightarrow d(x) \leq d(y) \wedge d(x) \leq d(z)$
by (*metis d-commutative d-idempotent d-mult-left-lower-bound mult-isotone order-trans*)

lemma *d-mult-left-absorb-add*: $d(x) ; (d(x) + d(y)) = d(x)$
by (*metis add-commutative d-idempotent d-plus-one mult-left-dist-add mult-right-one*)

lemma *d-add-left-absorb-mult*: $d(x) + d(x) ; d(y) = d(x)$
by (*metis add-commutative d-mult-left-lower-bound less-eq-def*)

lemma *d-add-left-dist-mult*: $d(x) + d(y) ; d(z) = (d(x) + d(y)) ; (d(x) + d(z))$
by (*smt add-associative d-commutative d-idempotent d-mult-left-absorb-add mult-left-dist-add mult-right-dist-add*)

lemma *d-order*: $d(x) \leq d(y) \longleftrightarrow d(x) = d(x) ; d(y)$
by (*metis d-mult-greatest-lower-bound d-mult-left-absorb-add less-eq-def order-refl*)

lemma *d-mult-below*: $d(x) ; y \leq y$
by (*metis add-left-divisibility d-plus-one mult-left-one mult-right-dist-add*)

lemma *d-preserves-equation*: $d(y) ; x \leq x ; d(y) \longleftrightarrow d(y) ; x = d(y) ; x ; d(y)$
apply *rule*
apply (*metis antisym d-below-one d-idempotent mult-associative mult-right-isotone mult-right-one*)
apply (*metis d-below-one mult-associative mult-left-isotone mult-left-one*)
done

end

class *left-zero-antidomain-semiring* = *idempotent-left-zero-semiring* + *d* + *neg* +
assumes *a-restrict* : $-x ; x = 0$
assumes *a-plus-mult-d*: $-(x ; y) + -(x ; --y) = -(x ; --y)$
assumes *a-complement* : $--x + -x = 1$
assumes *d-def* : $d(x) = --x$

begin

— Many lemmas in this class are taken from Georg Struth's theories.

notation
uminus (a)

lemma *a-greatest-left-absorber*: $a(x) ; y = 0 \longleftrightarrow a(x) \leq a(y)$
apply *rule*
apply (*metis a-complement a-plus-mult-d a-restrict add-left-zero add-right-zero mult-left-dist-add mult-left-one mult-right-one mult-right-sub-dist-add-right*)
apply (*metis a-restrict add-right-zero less-eq-def mult-right-dist-add*)
done

lemma *a-mult-d*: $a(x ; y) = a(x ; d(y))$
by (*metis a-complement a-greatest-left-absorber a-plus-mult-d d-def less-eq-def mult-associative mult-left-one mult-left-zero mult-right-dist-add a-restrict add-commutative*)

lemma *a-d-closed*: $d(a(x)) = a(x)$
by (*metis a-mult-d d-def mult-left-one*)

lemma *a-plus-left-lower-bound*: $a(x + y) \leq a(x)$
by (*metis a-greatest-left-absorber eq-iff mult-left-sub-dist-add-left zero-least*)

lemma *a-idempotent*: $a(x) ; a(x) = a(x)$
by (*metis a-complement a-d-closed a-restrict add-right-zero d-def mult-left-dist-add mult-right-one*)

lemma *a-below-one*: $a(x) \leq 1$
by (*metis a-complement add-commutative add-left-upper-bound*)

lemma *a-mult-add*: $a(x) ; (y + x) = a(x) ; y$
  by (*metis a-restrict add-right-zero mult-left-dist-add*)

lemma *a-3*: $a(x) ; a(y) ; d(x + y) = 0$
  by (*metis a-below-one a-greatest-left-absorber a-mult-d a-restrict add-right-zero less-eq-def mult-associative mult-left-dist-add mult-left-isotone mult-left-one*)

lemma *a-dist-add*: $a(x) ; a(y) = a(x + y)$
proof −
  have $a(x) ; a(y) = a(x) ; a(y) ; a(x + y)$
    by (*metis a-3 a-complement add-left-zero d-def mult-left-dist-add mult-right-one*)
  hence $a(x) ; a(y) \leq a(x + y)$
    by (*metis a-below-one mult-left-isotone mult-left-one order-trans*)
  thus *?thesis*
    by (*metis a-idempotent a-plus-left-lower-bound add-commutative antisym mult-left-isotone mult-right-isotone order-trans*)
qed

lemma *a-export*: $a(a(x) ; y) = d(x) + a(y)$
proof −
  have $a(a(x) ; y) = a(a(x) ; y) ; d(y) + a(a(x) ; y) ; a(y)$
    by (*metis a-complement d-def mult-left-dist-add mult-right-one*)
  hence $a(a(x) ; y) \leq a(a(x) ; y) ; d(y) + a(y)$
    by (*metis a-below-one a-dist-add less-eq-def mult-left-isotone mult-left-one*)
  hence $a(a(x) ; y) \leq a(a(x) ; y) ; (a(x) + d(x)) ; d(y) + a(y)$
    by (*metis a-complement add-commutative d-def mult-associative mult-left-one*)
  hence $a(a(x) ; y) \leq a(a(x) ; y) ; d(x) ; d(y) + a(y)$
    by (*smt a-mult-d a-restrict add-left-zero mult-associative mult-left-dist-add mult-right-dist-add*)
  hence $a(a(x) ; y) \leq d(x) + a(y)$
    by (*metis a-dist-add a-plus-left-lower-bound add-commutative add-right-isotone d-def order-trans*)
  thus *?thesis*
    by (*metis a-restrict a-greatest-left-absorber a-dist-add add-commutative mult-left-zero d-def add-least-upper-bound mult-associative antisym*)
qed

subclass *left-zero-domain-semiring*
  apply *unfold-locales*
  apply (*smt a-complement a-d-closed a-idempotent a-restrict case-split-left-equal d-def eq-refl less-eq-def mult-associative*)
  apply (*metis a-mult-d d-def*)
  apply (*metis a-below-one d-def less-eq-def*)
  apply (*metis a-3 a-d-closed a-dist-add d-def*)
  apply (*metis a-dist-add a-export d-def*)
  done

subclass *tests*
  apply *unfold-locales*
  apply (*metis mult-associative*)
  apply (*metis a-dist-add add-commutative*)
  apply (*metis a-complement a-d-closed a-dist-add d-def mult-left-dist-add mult-right-one*)
  apply (*metis a-d-closed a-dist-add d-def*)
  apply (*rule the-equality*[*THEN sym*])
  apply (*metis a-d-closed a-restrict d-def*)
  apply (*metis a-d-closed a-restrict d-def*)
  apply (*metis a-complement a-restrict add-right-zero mult-right-one*)
  apply (*metis a-d-closed a-export d-def*)
  apply (*smt a-d-closed a-dist-add a-plus-left-lower-bound add-commutative d-def less-eq-def*)
  apply (*metis less-def*)
  done

lemma *a-fixpoint*: $\forall x . (a(x) = x \longrightarrow (\forall y . y = 0))$
  by (*metis a-complement a-restrict add-idempotent mult-left-one mult-left-zero*)

lemma *a-strict*: $a(x) = 1 \longleftrightarrow x = 0$
  by (*metis a-complement a-restrict add-right-zero mult-left-one mult-right-one*)

lemma *d-complement-zero*: $d(x) ; a(x) = 0$
  by (*metis a-restrict d-def*)

lemma *a-complement-zero*: $a(x) ; d(x) = 0$
  by (*metis d-def zero-def*)

lemma *a-shunting-zero*: $a(x)\ ;\ d(y) = 0 \longleftrightarrow a(x) \leq a(y)$
 by (*metis a-greatest-left-absorber d-weak-locality*)

lemma *a-antitone*: $x \leq y \longrightarrow a(y) \leq a(x)$
 by (*metis a-plus-left-lower-bound less-eq-def*)

lemma *a-mult-deMorgan*: $a(a(x)\ ;\ a(y)) = d(x + y)$
 by (*metis a-dist-add d-def*)

lemma *a-mult-deMorgan-1*: $a(a(x)\ ;\ a(y)) = d(x) + d(y)$
 by (*metis a-mult-deMorgan d-dist-add*)

lemma *a-mult-deMorgan-2*: $a(d(x)\ ;\ d(y)) = a(x) + a(y)$
 by (*metis d-def plus-def*)

lemma *a-plus-deMorgan*: $a(a(x) + a(y)) = d(x)\ ;\ d(y)$
 by (*metis a-dist-add d-def*)

lemma *a-plus-deMorgan-1*: $a(d(x) + d(y)) = a(x)\ ;\ a(y)$
 by (*metis a-mult-deMorgan-1 sub-mult-closed*)

lemma *a-mult-left-upper-bound*: $a(x) \leq a(x\ ;\ y)$
 by (*metis a-greatest-left-absorber d-def d-mult-sub leq-mult-zero sub-comm*)

lemma *d-a-closed*: $a(d(x)) = a(x)$
 by (*metis a-d-closed d-def*)

lemma *a-export-d*: $a(d(x)\ ;\ y) = a(x) + a(y)$
 by (*metis a-mult-d a-mult-deMorgan-2*)

lemma *a-7*: $d(x)\ ;\ a(d(y) + d(z)) = d(x)\ ;\ a(y)\ ;\ a(z)$
 by (*metis a-plus-deMorgan-1 mult-associative*)

lemma *d-a-shunting*: $d(x)\ ;\ a(y) \leq d(z) \longleftrightarrow d(x) \leq d(z) + d(y)$
 by (*smt a-dist-add d-def plus-closed shunting sub-comm*)

lemma *d-d-shunting*: $d(x)\ ;\ d(y) \leq d(z) \longleftrightarrow d(x) \leq d(z) + a(y)$
 by (*metis d-a-closed d-a-shunting d-def*)

lemma *d-cancellation-1*: $d(x) \leq d(y) + (d(x)\ ;\ a(y))$
 by (*metis a-dist-add add-commutative add-left-upper-bound d-def plus-compl-intro*)

lemma *d-cancellation-2*: $(d(z) + d(y))\ ;\ a(y) \leq d(z)$
 by (*metis d-a-shunting d-dist-add eq-refl*)

lemma *a-add-closed*: $d(a(x) + a(y)) = a(x) + a(y)$
 by (*metis d-def plus-closed*)

lemma *a-mult-closed*: $d(a(x)\ ;\ a(y)) = a(x)\ ;\ a(y)$
 by (*metis d-def sub-mult-closed*)

lemma *d-a-shunting-zero*: $d(x)\ ;\ a(y) = 0 \longleftrightarrow d(x) \leq d(y)$
 by (*metis a-greatest-left-absorber d-def*)

lemma *d-d-shunting-zero*: $d(x)\ ;\ d(y) = 0 \longleftrightarrow d(x) \leq a(y)$
 by (*metis d-def leq-mult-zero*)

lemma *d-compl-intro*: $d(x) + d(y) = d(x) + a(x)\ ;\ d(y)$
 by (*metis add-commutative d-def plus-compl-intro*)

lemma *a-compl-intro*: $a(x) + a(y) = a(x) + d(x)\ ;\ a(y)$
 by (*smt a-dist-add add-commutative d-def mult-right-one plus-compl plus-distr-mult-left*)

lemma *kat-2*: $y\ ;\ a(z) \leq a(x)\ ;\ y \longrightarrow d(x)\ ;\ y\ ;\ a(z) = 0$
 by (*smt a-export a-plus-left-lower-bound add-least-upper-bound d-d-shunting-zero d-export d-strict less-eq-def mult-associative*)

lemma *kat-3*: $d(x)\ ;\ y\ ;\ a(z) = 0 \longrightarrow d(x)\ ;\ y = d(x)\ ;\ y\ ;\ d(z)$
 by (*metis add-left-zero d-def mult-left-dist-add mult-right-one plus-compl*)

lemma *kat-4*: $d(x) ; y = d(x) ; y ; d(z) \longrightarrow d(x) ; y \leq y ; d(z)$
by (*metis a-below-one d-def mult-left-isotone mult-left-one*)

lemma *kat-2-equiv*: $y ; a(z) \leq a(x) ; y \longleftrightarrow d(x) ; y ; a(z) = 0$
by (*smt2 kat-2 a-below-one a-complement add-left-zero d-def mult-left-one mult-right-dist-add mult-right-isotone mult-right-one*)

lemma *kat-4-equiv*: $d(x) ; y = d(x) ; y ; d(z) \longleftrightarrow d(x) ; y \leq y ; d(z)$
apply *rule*
apply (*metis kat-4*)
apply (*rule antisym*)
apply (*metis d-idempotent less-eq-def mult-associative mult-left-dist-add*)
apply (*metis d-plus-one less-eq-def mult-left-dist-add mult-right-one*)
done

lemma *kat-3-equiv-opp*: $a(z) ; y ; d(x) = 0 \longleftrightarrow y ; d(x) = d(z) ; y ; d(x)$
by (*metis a-complement a-restrict add-left-zero d-a-closed d-def mult-associative mult-left-one mult-left-zero mult-right-dist-add*)

lemma *kat-4-equiv-opp*: $y ; d(x) = d(z) ; y ; d(x) \longleftrightarrow y ; d(x) \leq d(z) ; y$
by (*metis d-def double-negation kat-2-equiv kat-3-equiv-opp*)

lemma *d-restrict-iff*: $(x \leq y) \longleftrightarrow (x \leq d(x) ; y)$
by (*metis add-least-upper-bound d-below-one d-restrict-equals less-eq-def mult-left-dist-add mult-left-isotone mult-left-one*)

lemma *d-restrict-iff-1*: $(d(x) ; y \leq z) \longleftrightarrow (d(x) ; y \leq d(x) ; z)$
by (*metis add-commutative d-export d-mult-left-lower-bound d-plus-one d-restrict-iff mult-left-isotone mult-left-one mult-right-sub-dist-add-right order-trans*)

end

end

27 DomainIteration

theory *DomainIteration*

imports *Domain LatticeOrderedSemiring OmegaAlgebra*

begin

class *domain-semiring-lattice* = *left-zero-domain-semiring* + *lattice-ordered-pre-left-semiring*

begin

subclass *bounded-idempotent-left-zero-semiring* **..**

lemma *d-top*: $d(T) = 1$
  by (*metis add-left-top d-dist-add d-one d-plus-one*)

lemma *mult-domain-top*: $x \; ; \; d(y) \; ; \; T \leq d(x \; ; \; y) \; ; \; T$
  by (*smt d-mult-d d-restrict-equals mult-associative mult-right-isotone top-greatest*)

lemma *domain-meet-domain*: $d(x \frown d(y) \; ; \; z) \leq d(y)$
  by (*metis d-export d-isotone d-mult-left-lower-bound meet.add-right-upper-bound order-trans*)

lemma *meet-domain*: $x \frown d(y) \; ; \; z = d(y) \; ; \; (x \frown z)$
  apply (*rule antisym*)
  apply (*metis domain-meet-domain add-commutative add-right-divisibility d-plus-one d-restrict-equals meet.add-right-isotone mult-left-isotone mult-left-one mult-right-isotone order-trans*)
  apply (*metis d-plus-one meet.add-least-upper-bound mult-left-one mult-left-sub-dist-meet-right mult-right-sub-dist-add-left*)
  done

lemma *meet-intro-domain*: $x \frown y = d(y) \; ; \; x \frown y$
  by (*metis d-restrict-equals meet-commutative meet-domain*)

lemma *meet-domain-top*: $x \frown d(y) \; ; \; T = d(y) \; ; \; x$
  by (*metis meet.add-right-zero meet-domain*)

lemma $d(x) = x \; ; \; T \frown 1$ **nitpick** [*expect=genuine*] **oops**

lemma *d-galois*: $d(x) \leq d(y) \longleftrightarrow x \leq d(y) \; ; \; T$
  by (*metis d-isotone d-least-left-preserver meet-domain-top meet.add-least-upper-bound*)

lemma *vector-meet*: $x \; ; \; T \frown y \leq d(x) \; ; \; y$
  by (*metis d-galois d-mult-sub meet-commutative meet-domain-top meet.add-right-isotone*)

end

class *domain-semiring-lattice-L* = *domain-semiring-lattice* + *L* +
  assumes *l1*: $x \; ; \; L = x \; ; \; 0 + d(x) \; ; \; L$
  assumes *l2*: $d(L) \; ; \; x \leq x \; ; \; d(L)$
  assumes *l3*: $d(L) \; ; \; T \leq L + d(L \; ; \; 0) \; ; \; T$
  assumes *l4*: $L \; ; \; T \leq L$
  assumes *l5*: $x \; ; \; 0 \frown L \leq (x \frown L) \; ; \; 0$

begin

lemma *l8*: $(x \frown L) \; ; \; 0 \leq x \; ; \; 0 \frown L$
  by (*metis meet-commutative meet.add-least-upper-bound mult-right-sub-dist-meet-right zero-right-mult-decreasing*)

lemma *l9*: $x \; ; \; 0 \frown L \leq d(x \; ; \; 0) \; ; \; L$
  by (*metis d-restrict-equals meet-commutative meet-domain meet.add-right-upper-bound*)

lemma *l10*: $L \; ; \; L = L$
  by (*metis d-restrict-equals l1 less-eq-def zero-right-mult-decreasing*)

lemma *l11*: $d(x) \; ; \; L \leq x \; ; \; L$
  by (*metis add-right-upper-bound l1*)

lemma *l12*: $d(x \; ; \; 0) \; ; \; L \leq x \; ; \; 0$
  by (*metis add-right-divisibility l1 mult-associative mult-left-zero*)

lemma *l13*: $d(x ; 0) ; L \le x$
by (*metis l12 order-trans zero-right-mult-decreasing*)

lemma *l14*: $x ; L \le x ; 0 + L$
by (*metis add-right-isotone l1 meet-domain-top meet.add-left-upper-bound*)

lemma *l15*: $x ; d(y) ; L = x ; 0 + d(x ; y) ; L$
by (*metis d-commutative d-mult-d d-zero l1 mult-associative mult-left-zero*)

lemma *l16*: $x ; T \frown L \le x ; L$
by (*metis l11 order-trans vector-meet*)

lemma *l17*: $d(x) ; L \le d(x ; L) ; L$
by (*smt d-restrict-equals l11 meet-domain-top meet.add-least-upper-bound meet.add-left-divisibility order-trans*)

lemma *l18*: $d(x) ; L = d(x ; L) ; L$
by (*metis antisym d-mult-sub l17 mult-left-isotone*)

lemma *l19*: $d(x ; T ; 0) ; L \le d(x ; L) ; L$
by (*metis d-mult-sub l18 mult-associative mult-left-isotone*)

lemma *l20*: $x \le y \longleftrightarrow x \le y + L \wedge x \le y + d(y ; 0) ; T$
apply *rule*
apply (*metis add-isotone add-right-zero zero-least*)
apply (*smt add-commutative add-left-dist-meet l13 less-eq-def meet-domain-top*)
done

lemma *l21*: $d(x ; 0) ; L \le x ; 0 \frown L$
by (*metis l12 meet-domain-top meet.add-least-upper-bound meet.add-left-upper-bound*)

lemma *l22*: $x ; 0 \frown L = d(x ; 0) ; L$
by (*metis antisym l9 l21*)

lemma *l23*: $x ; T \frown L = d(x) ; L$
apply (*rule antisym*)
apply (*metis vector-meet*)
apply (*metis add-least-upper-bound d-mult-below l1 meet.add-least-upper-bound mult-right-isotone top-greatest*)
done

lemma *l29*: $L ; d(L) = L$
by (*metis d-preserves-equation d-restrict-equals l2*)

lemma *l30*: $d(L) ; x \le (x \frown L) + d(L ; 0) ; x$
by (*smt l3 less-eq-def meet-domain-top meet-left-dist-add*)

lemma *l31*: $d(L) ; x = (x \frown L) + d(L ; 0) ; x$
by (*smt add-least-upper-bound antisym d-mult-sub d-restrict-equals l30 meet-domain mult-left-isotone mult-left-sub-dist-meet-left*)

lemma *l40*: $L ; x \le L$
by (*metis add-right-top l4 mult-left-sub-dist-add-left order-trans*)

lemma *l41*: $L ; T = L$
by (*metis antisym l4 top-right-mult-increasing*)

lemma *l50*: $x ; 0 \frown L = (x \frown L) ; 0$
by (*metis eq-iff l5 l8*)

lemma *l51*: $d(x ; 0) ; L = (x \frown L) ; 0$
by (*metis l22 l50*)

lemma *l90*: $L ; T ; L = L$
by (*metis add-commutative d-restrict-equals d-top l1 l51 l31 meet-absorb meet-commutative mult-associative mult-left-dist-add mult-left-zero mult-right-one*)

lemma *l91*: $x = x ; T \longrightarrow d(L ; 0) ; x \le d(x ; 0) ; T$
proof −
have $d(L ; 0) ; x \le d(d(L ; 0) ; x) ; T$
by (*metis d-galois order-refl*)

also have ... $= d(d(L \; ; \; 0) \; ; \; d(x)) \; ; \; T$
  by (*metis d-mult-d*)
also have ... $= d(d(x) \; ; \; L \; ; \; 0) \; ; \; T$
  by (*metis d-commutative d-mult-d mult-associative*)
also have ... $\leq d(x \; ; \; L \; ; \; 0) \; ; \; T$
  by (*metis d-isotone l11 mult-left-isotone*)
also have ... $\leq d(x \; ; \; T \; ; \; 0) \; ; \; T$
  by (*metis d-isotone mult-left-isotone mult-right-isotone top-greatest*)
finally show *?thesis*
  by *metis*
qed

lemma *l92*: $x = x \; ; \; T \longrightarrow d(L \; ; \; 0) \; ; \; x \leq d((x \frown L) \; ; \; 0) \; ; \; T$
proof
assume *1*: $x = x \; ; \; T$
have $d(L \; ; \; 0) \; ; \; x = d(L) \; ; \; d(L \; ; \; 0) \; ; \; x$
  by (*metis d-commutative d-mult-sub d-order*)
also have ... $\leq d(L) \; ; \; d(x \; ; \; 0) \; ; \; T$ **using** *1*
  by (*metis eq-iff l91 mult-associative mult-isotone*)
also have ... $= d(d(x \; ; \; 0) \; ; \; L) \; ; \; T$
  by (*metis d-commutative d-export*)
also have ... $\leq d((x \frown L) \; ; \; 0) \; ; \; T$
  by (*metis l51 order-refl*)
finally show $d(L \; ; \; 0) \; ; \; x \leq d((x \frown L) \; ; \; 0) \; ; \; T$
  by *metis*
qed

end

class *domain-itering-lattice-L* = *bounded-itering* + *domain-semiring-lattice-L*

begin

lemma *mult-L-circ*: $(x \; ; \; L)^\circ = 1 + x \; ; \; L$
by (*metis circ-back-loop-fixpoint circ-mult l40 less-eq-def mult-associative*)

lemma *mult-L-circ-mult-below*: $(x \; ; \; L)^\circ \; ; \; y \leq y + x \; ; \; L$
by (*smt add-right-isotone l40 mult-L-circ mult-associative mult-left-one mult-right-dist-add mult-right-isotone*)

lemma *circ-L*: $L^\circ = L + 1$
by (*metis add-commutative l10 mult-L-circ*)

lemma *circ-d0-L*: $x^\circ \; ; \; d(x \; ; \; 0) \; ; \; L = x^\circ \; ; \; 0$
by (*metis add-right-zero circ-loop-fixpoint circ-plus-same d-zero l15 mult-associative mult-left-zero*)

lemma *d0-circ-left-unfold*: $d(x^\circ \; ; \; 0) = d(x \; ; \; x^\circ \; ; \; 0)$
by (*metis add-commutative add-left-zero circ-loop-fixpoint mult-associative*)

lemma *d-circ-import*: $d(y) \; ; \; x \leq x \; ; \; d(y) \longrightarrow d(y) \; ; \; x^\circ = d(y) \; ; \; (d(y) \; ; \; x)^\circ$
apply *rule*
apply (*rule antisym*)
apply (*metis circ-simulate circ-slide mult-associative d-idempotent d-preserves-equation*)
apply (*metis circ-isotone mult-left-isotone mult-left-one mult-right-isotone d-below-one*)
done

end

class *domain-omega-algebra-lattice-L* = *bounded-left-zero-omega-algebra* + *domain-semiring-lattice-L*

begin

lemma *mult-L-star*: $(x \; ; \; L)^\star = 1 + x \; ; \; L$
by (*metis l40 less-eq-def mult-associative star.circ-back-loop-fixpoint star.circ-mult*)

lemma *mult-L-omega*: $(x \; ; \; L)^\omega \leq x \; ; \; L$
by (*smt l41 less-eq-def mult-associative mult-left-dist-add omega-unfold top-greatest*)

lemma *mult-L-add-star*: $(x \; ; \; L + y)^\star = y^\star + y^\star \; ; \; x \; ; \; L$
proof (*rule antisym*)

have $(x ; L + y) ; (y^\star + y^\star ; x ; L) = x ; L ; (y^\star + y^\star ; x ; L) + y ; (y^\star + y^\star ; x ; L)$
by (*metis mult-associative mult-right-dist-add*)
also have ... $\leq x ; L + y ; (y^\star + y^\star ; x ; L)$
by (*metis add-left-isotone l40 mult-associative mult-right-isotone*)
also have ... $\leq x ; L + y ; y^\star + y^\star ; x ; L$
by (*smt add-associative add-commutative add-right-upper-bound mult-associative mult-left-dist-add star.circ-loop-fixpoint*)
also have ... $\leq x ; L + y^\star + y^\star ; x ; L$
by (*metis add-left-isotone add-right-isotone star.left-plus-below-circ*)
also have ... $= y^\star + y^\star ; x ; L$
by (*metis add-associative add-commutative mult-associative star.circ-loop-fixpoint star.circ-reflexive star.circ-sup-one-right-unfold star-involutive*)
finally have $1 + (x ; L + y) ; (y^\star + y^\star ; x ; L) \leq y^\star + y^\star ; x ; L$
by (*metis add-commutative add-least-upper-bound add-right-divisibility star.circ-left-unfold*)
thus $(x ; L + y)^\star \leq y^\star + y^\star ; x ; L$
by (*metis mult-right-one star-left-induct*)
next
show $y^\star + y^\star ; x ; L \leq (x ; L + y)^\star$
by (*metis add-commutative add-least-upper-bound mult-associative star.circ-increasing star.circ-mult-upper-bound star.circ-sub-dist*)
qed

lemma *mult-L-add-omega*: $(x ; L + y)^\omega \leq y^\omega + y^\star ; x ; L$
proof −
have *1*: $(y^\star ; x ; L)^\omega \leq y^\omega + y^\star ; x ; L$
by (*metis add-least-upper-bound add-right-isotone mult-L-omega*)
have $(y^\star ; x ; L)^\star ; y^\omega \leq y^\omega + y^\star ; x ; L$
by (*metis add-right-isotone l40 mult-associative mult-right-isotone star-left-induct*)
thus *?thesis* **using** *1*
by (*smt add-associative add-commutative less-eq-def mult-associative omega-decompose*)
qed

end

sublocale *domain-omega-algebra-lattice-L* < *dL-star*!: *itering* **where** *circ* = *star* ..

sublocale *domain-omega-algebra-lattice-L* < *dL-star*!: *domain-itering-lattice-L* **where** *circ* = *star* ..

context *domain-omega-algebra-lattice-L*

begin

lemma *d0-star-below-d0-omega*: $d(x^\star ; 0) \leq d(x^\omega ; 0)$
by (*metis d-isotone star-zero-below-omega-zero*)

lemma *d0-below-d0-omega*: $d(x ; 0) \leq d(x^\omega ; 0)$
by (*metis d0-star-below-d0-omega d-isotone mult-left-isotone order-trans star.circ-increasing*)

lemma *star-L-split*: $y \leq z \wedge x ; z ; L \leq x ; 0 + z ; L \longrightarrow x^\star ; y ; L \leq x^\star ; 0 + z ; L$
proof
assume *1*: $y \leq z \wedge x ; z ; L \leq x ; 0 + z ; L$
have $x ; (x^\star ; 0 + z ; L) \leq x^\star ; 0 + x ; z ; L$
by (*metis add-right-zero eq-iff mult-associative mult-left-dist-add star.circ-loop-fixpoint*)
also have ... $\leq x^\star ; 0 + x ; 0 + z ; L$ **using** *1*
by (*metis add-associative add-left-upper-bound less-eq-def*)
also have ... $= x^\star ; 0 + z ; L$
by (*metis add-commutative less-eq-def mult-right-dist-add star.circ-increasing*)
finally have $y ; L + x ; (x^\star ; 0 + z ; L) \leq x^\star ; 0 + z ; L$ **using** *1*
by (*metis add-least-upper-bound add-right-upper-bound mult-left-isotone order-trans*)
thus $x^\star ; y ; L \leq x^\star ; 0 + z ; L$
by (*metis star-left-induct mult-associative*)
qed

lemma *star-L-split-same*: $x ; y ; L \leq x ; 0 + y ; L \longrightarrow x^\star ; y ; L = x^\star ; 0 + y ; L$
by (*smt add-associative add-left-zero antisym less-eq-def mult-associative mult-left-dist-add mult-left-one mult-right-sub-dist-add-left order-refl star-L-split star.circ-right-unfold*)

lemma *star-d-L-split-equal*: $d(x ; y) \leq d(y) \longrightarrow x^\star ; d(y) ; L = x^\star ; 0 + d(y) ; L$
by (*metis add-right-isotone l15 less-eq-def mult-right-sub-dist-add-left star-L-split-same*)

lemma *d0-omega-mult*: $d(x^\omega ; y ; 0) = d(x^\omega ; 0)$
  apply (*rule antisym*)
  apply (*metis d-isotone mult-left-isotone omega-sub-vector*)
  apply (*metis d-isotone mult-associative mult-right-isotone zero-least*)
  done

lemma *d-omega-export*: $d(y) ; x \leq x ; d(y) \longrightarrow d(y) ; x^\omega = (d(y) ; x)^\omega$
  apply (*rule impI*)
  apply (*rule antisym*)
  apply (*metis d-preserves-equation omega-simulation order-refl*)
  apply (*smt less-eq-def mult-left-dist-add omega-simulation-2 omega-slide*)
  done

lemma *d-omega-import*: $d(y) ; x \leq x ; d(y) \longrightarrow d(y) ; x^\omega = d(y) ; (d(y) ; x)^\omega$
  by (*metis d-idempotent d-omega-export mult-associative omega-slide*)

lemma *star-d-omega-top*: $x^\star ; d(x^\omega) ; T = x^\star ; 0 + d(x^\omega) ; T$
  apply (*rule antisym*)
  apply (*metis add-right-divisibility dual-order.trans mult-domain-top star-mult-omega*)
    apply (*smt2 add-associative add-commutative add-left-divisibility add-left-zero mult-associative mult-left-dist-add star.circ-loop-fixpoint*)
  done

lemma *omega-meet-L*: $x^\omega \frown L = d(x^\omega) ; L$
  by (*metis l23 omega-vector*)

lemma *d-star-mult*: $d(x ; y) \leq d(y) \longrightarrow d(x^\star ; y) = d(x^\star ; 0) + d(y)$ **oops**
lemma *d0-split-omega-omega*: $x^\omega \leq x^\omega ; 0 + d(x^\omega \frown L) ; T$ **nitpick** [*expect=genuine*] **oops**

end

end

28 DomainRecursion

theory *DomainRecursion*

imports *DomainIteration Approximation*

begin

class *domain-semiring-lattice-apx* = *domain-semiring-lattice-L* + *apx* +
  assumes *apx-def*: $x \sqsubseteq y \longleftrightarrow x \leq y + L \wedge d(L) ; y \leq x + d(x ; 0) ; T$

begin

lemma *apx-transitive*: $x \sqsubseteq y \wedge y \sqsubseteq z \longrightarrow x \sqsubseteq z$
proof
  assume *1*: $x \sqsubseteq y \wedge y \sqsubseteq z$
  hence *2*: $x \leq z + L$
    by (*smt add-associative add-commutative apx-def less-eq-def*)
  have $d(d(L) ; y ; 0) ; T \leq d((x + d(x ; 0) ; T) ; 0) ; T$ **using** *1*
    by (*metis apx-def d-isotone mult-left-isotone*)
  also have ... $\leq d(x ; 0) ; T$
    by (*metis add-least-upper-bound d-galois mult-left-isotone mult-right-dist-add order-refl zero-right-mult-decreasing*)
  finally have *3*: $d(d(L) ; y ; 0) ; T \leq d(x ; 0) ; T$
    by *metis*
  have $d(L) ; z = d(L) ; (d(L) ; z)$
    by (*metis d-idempotent mult-associative*)
  also have ... $\leq d(L) ; y + d(d(L) ; y ; 0) ; T$ **using** *1*
    by (*metis apx-def d-export mult-associative mult-left-dist-add mult-right-isotone*)
  also have ... $\leq x + d(x ; 0) ; T$ **using** *1 3*
    by (*smt add-least-upper-bound add-right-upper-bound apx-def order-trans*)
  finally show $x \sqsubseteq z$ **using** *2*
    by (*metis apx-def*)
qed

lemma *apx-meet-L*: $y \sqsubseteq x \longrightarrow x \frown L \leq y \frown L$
proof
  assume *1*: $y \sqsubseteq x$
  have $x \frown L = d(L) ; x \frown L$
    by (*metis d-restrict-equals meet-commutative meet-domain*)
  also have ... $\leq (y + d(y ; 0) ; T) \frown L$ **using** *1*
    by (*metis apx-def meet.add-left-isotone*)
  also have ... $\leq y$
    by (*metis add-least-upper-bound l13 meet-commutative meet-domain meet-left-dist-add meet.add-right-upper-bound meet.add-right-zero*)
  finally show $x \frown L \leq y \frown L$
    by (*metis meet-associative meet-idempotent meet.add-left-isotone*)
qed

lemma *add-apx-left-isotone*: $x \sqsubseteq y \longrightarrow x + z \sqsubseteq y + z$
proof
  assume *1*: $x \sqsubseteq y$
  hence *2*: $x + z \leq y + z + L$
    by (*smt add-associative add-commutative add-left-isotone apx-def*)
  have $d(L) ; (y + z) = d(L) ; y + d(L) ; z$
    by (*metis mult-left-dist-add*)
  also have ... $\leq d(L) ; y + z$
    by (*metis add-commutative add-least-upper-bound add-right-upper-bound d-below-one mult-left-dist-add mult-left-isotone mult-left-one*)
  also have ... $\leq x + d(x ; 0) ; T + z$ **using** *1*
    by (*metis add-left-isotone apx-def*)
  also have ... $\leq x + z + d((x + z) ; 0) ; T$
    by (*smt add-associative add-commutative add-right-isotone d-isotone mult-left-isotone mult-right-sub-dist-add-left*)
  finally show $x + z \sqsubseteq y + z$ **using** *2*
    by (*metis apx-def*)
qed

subclass *apx-biorder*
  apply *unfold-locales*
  apply (*metis add-least-upper-bound add-left-upper-bound apx-def d-plus-one mult-left-one mult-right-dist-add*)

apply (*metis add-same-context antisym apx-def apx-meet-L relative-equality*)
apply (*rule apx-transitive*)
done

lemma *mult-apx-left-isotone*: $x \sqsubseteq y \longrightarrow x ; z \sqsubseteq y ; z$
proof
assume *1*: $x \sqsubseteq y$
hence $x ; z \leq y ; z + L ; z$
by (*metis apx-def mult-left-isotone mult-right-dist-add*)
hence *2*: $x ; z \leq y ; z + L$
by (*metis add-commutative add-left-isotone l40 order-trans*)
have $d(L) ; y ; z \leq x ; z + d(x ; 0) ; T ; z$ **using** *1*
by (*metis apx-def mult-left-isotone mult-right-dist-add*)
also have ... $\leq x ; z + d(x ; z ; 0) ; T$
by (*metis add-right-isotone d-isotone mult-associative mult-isotone mult-right-isotone top-greatest zero-least*)
finally show $x ; z \sqsubseteq y ; z$ **using** *2*
by (*metis apx-def mult-associative*)
qed

lemma *mult-apx-right-isotone*: $x \sqsubseteq y \longrightarrow z ; x \sqsubseteq z ; y$
proof
assume *1*: $x \sqsubseteq y$
hence $z ; x \leq z ; y + z ; L$
by (*metis apx-def mult-left-dist-add mult-right-isotone*)
also have ... $\leq z ; y + z ; 0 + L$
by (*metis add-associative add-right-isotone l14*)
finally have *2*: $z ; x \leq z ; y + L$
by (*metis add-right-zero mult-left-dist-add*)
have $d(L) ; z ; y \leq z ; d(L) ; y$
by (*metis l2 mult-left-isotone*)
also have ... $\leq z ; (x + d(x ; 0) ; T)$ **using** *1*
by (*metis apx-def mult-associative mult-right-isotone*)
also have ... $= z ; x + z ; d(x ; 0) ; T$
by (*metis mult-associative mult-left-dist-add*)
also have ... $\leq z ; x + d(z ; x ; 0) ; T$
by (*metis add-right-isotone mult-associative mult-domain-top*)
finally show $z ; x \sqsubseteq z ; y$ **using** *2*
by (*metis apx-def mult-associative*)
qed

subclass *apx-semiring*
apply *unfold-locales*
apply (*metis add-right-upper-bound apx-def l3 mult-right-isotone order-trans top-greatest*)
apply (*rule add-apx-left-isotone*)
apply (*rule mult-apx-left-isotone*)
apply (*rule mult-apx-right-isotone*)
done

lemma *meet-L-apx-isotone*: $x \sqsubseteq y \longrightarrow x \frown L \sqsubseteq y \frown L$
by (*smt add-absorb add-commutative apx-def apx-meet-L d-restrict-equals l20 meet-commutative meet-domain meet.add-left-upper-bound*)

definition *kappa-apx-meet* :: $('a \Rightarrow 'a) \Rightarrow$ *bool*
where *kappa-apx-meet f* $\longleftrightarrow$ *apx.has-least-fixpoint f* $\wedge$ *has-apx-meet* $(\mu\ f)\ (\nu\ f) \wedge \kappa\ f = \mu\ f \bigtriangleup \nu\ f$

definition *kappa-mu-nu* :: $('a \Rightarrow 'a) \Rightarrow$ *bool*
where *kappa-mu-nu f* $\longleftrightarrow$ *apx.has-least-fixpoint f* $\wedge\ \kappa\ f = \mu\ f + (\nu\ f \frown L)$

definition *nu-below-mu-nu* :: $('a \Rightarrow 'a) \Rightarrow$ *bool*
where *nu-below-mu-nu f* $\longleftrightarrow d(L) ; \nu\ f \leq \mu\ f + (\nu\ f \frown L) + d(\nu\ f ; 0) ; T$

definition *nu-below-mu-nu-2* :: $('a \Rightarrow 'a) \Rightarrow$ *bool*
where *nu-below-mu-nu-2 f* $\longleftrightarrow d(L) ; \nu\ f \leq \mu\ f + (\nu\ f \frown L) + d((\mu\ f + (\nu\ f \frown L)) ; 0) ; T$

definition *mu-nu-apx-nu* :: $('a \Rightarrow 'a) \Rightarrow$ *bool*
where *mu-nu-apx-nu f* $\longleftrightarrow \mu\ f + (\nu\ f \frown L) \sqsubseteq \nu\ f$

definition *mu-nu-apx-meet* :: $('a \Rightarrow 'a) \Rightarrow$ *bool*
where *mu-nu-apx-meet f* $\longleftrightarrow$ *has-apx-meet* $(\mu\ f)\ (\nu\ f) \wedge \mu\ f \bigtriangleup \nu\ f = \mu\ f + (\nu\ f \frown L)$

```
definition apx-meet-below-nu :: ('a ⇒ 'a) ⇒ bool
  where apx-meet-below-nu f ⟷ has-apx-meet (μ f) (ν f) ∧ μ f △ ν f ≤ ν f

lemma mu-below-l: μ f ≤ μ f + (ν f ⌢ L)
  by (metis add-left-upper-bound)

lemma l-below-nu: has-least-fixpoint f ∧ has-greatest-fixpoint f ⟶ μ f + (ν f ⌢ L) ≤ ν f
  by (metis add-least-upper-bound meet.add-left-upper-bound mu-below-nu)

lemma n-l-nu: has-least-fixpoint f ∧ has-greatest-fixpoint f ⟶ (μ f + (ν f ⌢ L)) ⌢ L = ν f ⌢ L
  by (smt add-commutative add-left-dist-meet less-eq-def meet-absorb meet-associative meet-commutative mu-below-nu)

lemma l-apx-mu: μ f + (ν f ⌢ L) ⊑ μ f
  by (metis add-right-isotone apx-def meet-absorb meet-domain-top meet.add-least-upper-bound meet.add-left-upper-bound
meet.add-right-upper-bound)

lemma nu-below-mu-nu-nu-below-mu-nu-2: nu-below-mu-nu f ⟶ nu-below-mu-nu-2 f
proof
  assume 1: nu-below-mu-nu f
  have d(L) ; ν f = d(L) ; (d(L) ; ν f)
    by (metis d-idempotent mult-associative)
  also have ... ≤ d(L) ; (μ f + (ν f ⌢ L) + d(ν f ; 0) ; T) using 1
    by (metis mult-right-isotone nu-below-mu-nu-def)
  also have ... = d(L) ; (μ f + (ν f ⌢ L)) + d(L) ; d(ν f ; 0) ; T
    by (metis mult-associative mult-left-dist-add)
  also have ... ≤ μ f + (ν f ⌢ L) + d(L) ; d(ν f ; 0) ; T
    by (metis add-left-isotone meet-domain-top meet.add-left-upper-bound)
  also have ... = μ f + (ν f ⌢ L) + d(d(ν f ; 0) ; L) ; T
    by (smt d-commutative d-export)
  also have ... = μ f + (ν f ⌢ L) + d((ν f ⌢ L) ; 0) ; T
    by (metis l51)
  also have ... ≤ μ f + (ν f ⌢ L) + d((μ f + (ν f ⌢ L)) ; 0) ; T
    by (metis add-right-isotone add-right-upper-bound d-dist-add mult-right-dist-add)
  finally show nu-below-mu-nu-2 f
    by (metis nu-below-mu-nu-2-def)
qed

lemma nu-below-mu-nu-2-nu-below-mu-nu: has-least-fixpoint f ∧ has-greatest-fixpoint f ∧ nu-below-mu-nu-2 f ⟶
nu-below-mu-nu f
proof
  assume 1: has-least-fixpoint f ∧ has-greatest-fixpoint f ∧ nu-below-mu-nu-2 f
  hence d(L) ; ν f ≤ μ f + (ν f ⌢ L) + d((μ f + (ν f ⌢ L)) ; 0) ; T
    by (metis nu-below-mu-nu-2-def)
  also have ... ≤ μ f + (ν f ⌢ L) + d(ν f ; 0) ; T using 1
    by (smt add-absorb add-associative add-commutative d-dist-add l-below-nu less-eq-def meet-absorb mult-right-dist-add)
  finally show nu-below-mu-nu f
    by (metis nu-below-mu-nu-def)
qed

lemma nu-below-mu-nu-equivalent: has-least-fixpoint f ∧ has-greatest-fixpoint f ⟶ (nu-below-mu-nu f ⟷ nu-below-mu-nu-2
f)
  by (metis nu-below-mu-nu-2-nu-below-mu-nu nu-below-mu-nu-nu-below-mu-nu-2)

lemma nu-below-mu-nu-2-mu-nu-apx-nu: has-least-fixpoint f ∧ has-greatest-fixpoint f ∧ nu-below-mu-nu-2 f ⟶ mu-nu-apx-nu
f
proof
  assume 1: has-least-fixpoint f ∧ has-greatest-fixpoint f ∧ nu-below-mu-nu-2 f
  hence μ f + (ν f ⌢ L) ≤ ν f + L
    by (metis add-commutative add-right-upper-bound l-below-nu order-trans)
  thus mu-nu-apx-nu f using 1
    by (metis apx-def mu-nu-apx-nu-def nu-below-mu-nu-2-def)
qed

lemma mu-nu-apx-nu-mu-nu-apx-meet: has-least-fixpoint f ∧ has-greatest-fixpoint f ∧ mu-nu-apx-nu f ⟶ mu-nu-apx-meet f
proof
  let ?l = μ f + (ν f ⌢ L)
  assume has-least-fixpoint f ∧ has-greatest-fixpoint f ∧ mu-nu-apx-nu f
  hence is-apx-meet (μ f) (ν f) ?l
     by (smt add-apx-left-isotone add-commutative apx-meet-L is-apx-meet-def l-apx-mu less-eq-def meet.add-least-upper-bound
mu-nu-apx-nu-def)
  thus mu-nu-apx-meet f
    by (smt apx-meet-char mu-nu-apx-meet-def)
qed

lemma mu-nu-apx-meet-apx-meet-below-nu: has-least-fixpoint f ∧ has-greatest-fixpoint f ∧ mu-nu-apx-meet f ⟶
apx-meet-below-nu f
  by (metis apx-meet-below-nu-def l-below-nu mu-nu-apx-meet-def)

lemma apx-meet-below-nu-nu-below-mu-nu-2: apx-meet-below-nu f ⟶ nu-below-mu-nu-2 f
proof –
  let ?l = μ f + (ν f ⌒ L)
  have ∀ m . m ⊑ μ f ∧ m ⊑ ν f ∧ m ≤ ν f ⟶ d(L) ; ν f ≤ ?l + d(?l ; 0) ; T
  proof
    fix m
    show m ⊑ μ f ∧ m ⊑ ν f ∧ m ≤ ν f ⟶ d(L) ; ν f ≤ ?l + d(?l ; 0) ; T
    proof
      assume 1: m ⊑ μ f ∧ m ⊑ ν f ∧ m ≤ ν f
      hence m ≤ ?l
        by (smt add-commutative add-left-dist-meet add-left-upper-bound apx-def meet.less-eq-def meet.add-least-upper-bound)
      hence m + d(m ; 0) ; T ≤ ?l + d(?l ; 0) ; T
        by (metis add-isotone d-dist-add less-eq-def mult-right-dist-add)
      thus d(L) ; ν f ≤ ?l + d(?l ; 0) ; T using 1
        by (smt apx-def order-trans)
    qed
  qed
  thus ?thesis
    by (smt apx-meet-below-nu-def apx-meet-same apx-meet-unique is-apx-meet-def nu-below-mu-nu-2-def)
qed

lemma has-apx-least-fixpoint-kappa-apx-meet: has-least-fixpoint f ∧ has-greatest-fixpoint f ∧ apx.has-least-fixpoint f ⟶
kappa-apx-meet f
proof
  assume 1: has-least-fixpoint f ∧ has-greatest-fixpoint f ∧ apx.has-least-fixpoint f
  hence 2: ∀ w . w ⊑ μ f ∧ w ⊑ ν f ⟶ d(L) ; κ f ≤ w + d(w ; 0) ; T
    by (metis apx-def mult-right-isotone order-trans kappa-below-nu)
  have ∀ w . w ⊑ μ f ∧ w ⊑ ν f ⟶ w ≤ κ f + L using 1
    by (metis add-left-isotone apx-def mu-below-kappa order-trans)
  hence ∀ w . w ⊑ μ f ∧ w ⊑ ν f ⟶ w ⊑ κ f using 2
    by (metis apx-def)
  hence is-apx-meet (μ f) (ν f) (κ f) using 1
    by (smt apx-meet-char is-apx-meet-def kappa-apx-below-mu kappa-apx-below-nu kappa-apx-meet-def)
  thus kappa-apx-meet f using 1
    by (metis apx-meet-char kappa-apx-meet-def)
qed

lemma kappa-apx-meet-apx-meet-below-nu: has-greatest-fixpoint f ∧ kappa-apx-meet f ⟶ apx-meet-below-nu f
  by (metis apx-meet-below-nu-def kappa-apx-meet-def kappa-below-nu)

lemma apx-meet-below-nu-kappa-mu-nu: has-least-fixpoint f ∧ has-greatest-fixpoint f ∧ isotone f ∧ apx.isotone f ∧
apx-meet-below-nu f ⟶ kappa-mu-nu f
proof
  let ?l = μ f + (ν f ⌒ L)
  let ?m = μ f △ ν f
  assume 1: has-least-fixpoint f ∧ has-greatest-fixpoint f ∧ isotone f ∧ apx.isotone f ∧ apx-meet-below-nu f
  hence 2: ?m = ?l
                  by   (metis   apx-meet-below-nu-nu-below-mu-nu-2   mu-nu-apx-meet-def   mu-nu-apx-nu-mu-nu-apx-meet
nu-below-mu-nu-2-mu-nu-apx-nu)
  have 3: ?l ≤ f(?l) + L
  proof –
    have ?l ≤ μ f + L
      by (metis add-right-isotone meet.add-right-upper-bound)
    also have ... = f(μ f) + L using 1
      by (metis is-least-fixpoint-def least-fixpoint)
    also have ... ≤ f(?l) + L using 1
      by (metis add-left-isotone add-left-upper-bound isotone-def)
    finally show ?l ≤ f(?l) + L
      by metis
  qed
  have d(L) ; f(?l) ≤ ?l + d(?l ; 0) ; T
  proof −
    have d(L) ; f(?l) ≤ d(L) ; f(ν f) using 1 2
      by (metis apx-meet-below-nu-def isotone-def mult-right-isotone)
    also have ... = d(L) ; ν f using 1
      by (metis greatest-fixpoint is-greatest-fixpoint-def)
    also have ... ≤ ?l + d(?l ; 0) ; T using 1
      by (metis apx-meet-below-nu-nu-below-mu-nu-2 nu-below-mu-nu-2-def)
    finally show d(L) ; f(?l) ≤ ?l + d(?l ; 0) ; T
      by metis
  qed
  hence 4: ?l ⊑ f(?l) using 3
    by (metis apx-def)
  have 5: f(?l) ⊑ μ f
  proof −
    have ?l ⊑ μ f
      by (metis l-apx-mu)
    thus f(?l) ⊑ μ f using 1
      by (metis apx.isotone-def is-least-fixpoint-def least-fixpoint)
  qed
  have 6: f(?l) ⊑ ν f
  proof −
    have ?l ⊑ ν f using 1 2
      by (metis apx-greatest-lower-bound apx-meet-below-nu-def apx-reflexive)
    thus f(?l) ⊑ ν f using 1
      by (metis apx.isotone-def greatest-fixpoint is-greatest-fixpoint-def)
  qed
  hence f(?l) ⊑ ?l using 1 2 5
    by (metis apx-greatest-lower-bound apx-meet-below-nu-def)
  hence 7: f(?l) = ?l using 4
    by (metis apx-antisymmetric)
  have ∀ y . f(y) = y ⟶ ?l ⊑ y
  proof
    fix y
    show f(y) = y ⟶ ?l ⊑ y
    proof
      assume 8: f(y) = y
      hence 9: ?l ≤ y + L using 1
        by (metis add-isotone is-least-fixpoint-def least-fixpoint meet.add-right-upper-bound)
      have y ≤ ν f using 1 8
        by (metis greatest-fixpoint is-greatest-fixpoint-def)
      hence d(L) ; y ≤ ?l + d(?l ; 0) ; T using 4 6
        by (metis apx-def apx-transitive mult-right-isotone order-trans)
      thus ?l ⊑ y using 9
        by (metis apx-def)
    qed
  qed
  thus kappa-mu-nu f using 1 2 7
    by (smt apx.least-fixpoint-same apx.has-least-fixpoint-def apx.is-least-fixpoint-def kappa-mu-nu-def)
qed

lemma kappa-mu-nu-has-apx-least-fixpoint: kappa-mu-nu f ⟶ apx.has-least-fixpoint f
  by (metis kappa-mu-nu-def)

lemma nu-below-mu-nu-kappa-mu-nu: has-least-fixpoint f ∧ has-greatest-fixpoint f ∧ isotone f ∧ apx.isotone f ∧
nu-below-mu-nu f ⟶ kappa-mu-nu f
    by (metis apx-meet-below-nu-kappa-mu-nu mu-nu-apx-meet-apx-meet-below-nu mu-nu-apx-nu-mu-nu-apx-meet
nu-below-mu-nu-nu-below-mu-nu-2 nu-below-mu-nu-2-mu-nu-apx-nu)

lemma kappa-mu-nu-nu-below-mu-nu: has-least-fixpoint f ∧ has-greatest-fixpoint f ∧ kappa-mu-nu f ⟶ nu-below-mu-nu f
  by (metis apx-meet-below-nu-nu-below-mu-nu-2 has-apx-least-fixpoint-kappa-apx-meet nu-below-mu-nu-2-nu-below-mu-nu
kappa-apx-meet-apx-meet-below-nu kappa-mu-nu-has-apx-least-fixpoint)

definition kappa-mu-nu-L :: ('a ⇒ 'a) ⇒ bool
  where kappa-mu-nu-L f ⟷ apx.has-least-fixpoint f ∧ κ f = μ f + d(ν f ; 0) ; L

definition nu-below-mu-nu-L :: ('a ⇒ 'a) ⇒ bool
  where nu-below-mu-nu-L f ⟷ d(L) ; ν f ≤ μ f + d(ν f ; 0) ; T
```

definition *mu-nu-apx-nu-L* :: $('a \Rightarrow 'a) \Rightarrow$ *bool*
  where *mu-nu-apx-nu-L f* $\longleftrightarrow \mu\ f + d(\nu\ f\ ;\ 0)\ ;\ L \sqsubseteq \nu\ f$

definition *mu-nu-apx-meet-L* :: $('a \Rightarrow 'a) \Rightarrow$ *bool*
  where *mu-nu-apx-meet-L f* $\longleftrightarrow$ *has-apx-meet* $(\mu\ f)\ (\nu\ f) \wedge \mu\ f \vartriangle \nu\ f = \mu\ f + d(\nu\ f\ ;\ 0)\ ;\ L$

lemma *n-below-l*: $x + d(y\ ;\ 0)\ ;\ L \leq x + (y \frown L)$
  by (*metis add-right-isotone d-mult-below l13 meet.add-least-upper-bound*)

lemma *n-equal-l*: *nu-below-mu-nu-L f* $\longrightarrow \mu\ f + d(\nu\ f\ ;\ 0)\ ;\ L = \mu\ f + (\nu\ f \frown L)$
proof
  assume *nu-below-mu-nu-L f*
  hence $\nu\ f \frown L \leq (\mu\ f + d(\nu\ f\ ;\ 0)\ ;\ T) \frown L$
    by (*smt meet-associative meet-intro-domain meet.add-right-divisibility nu-below-mu-nu-L-def*)
  also have ... $\leq \mu\ f + d(\nu\ f\ ;\ 0)\ ;\ L$
    by (*smt add-left-dist-meet add-right-divisibility meet-commutative meet-domain-top meet.add-left-isotone*)
  finally have $\mu\ f + (\nu\ f \frown L) \leq \mu\ f + d(\nu\ f\ ;\ 0)\ ;\ L$
    by (*metis add-least-upper-bound add-left-upper-bound*)
  thus $\mu\ f + d(\nu\ f\ ;\ 0)\ ;\ L = \mu\ f + (\nu\ f \frown L)$
    by (*metis antisym n-below-l*)
qed

lemma *nu-below-mu-nu-L-nu-below-mu-nu*: *nu-below-mu-nu-L f* $\longrightarrow$ *nu-below-mu-nu f*
  by (*metis add-associative add-right-top mult-left-dist-add n-equal-l nu-below-mu-nu-L-def nu-below-mu-nu-def*)

lemma *nu-below-mu-nu-L-kappa-mu-nu-L*: *has-least-fixpoint f* $\wedge$ *has-greatest-fixpoint f* $\wedge$ *isotone f* $\wedge$ *apx.isotone f* $\wedge$ *nu-below-mu-nu-L f* $\longrightarrow$ *kappa-mu-nu-L f*
  by (*metis n-equal-l nu-below-mu-nu-L-nu-below-mu-nu nu-below-mu-nu-kappa-mu-nu kappa-mu-nu-L-def kappa-mu-nu-def*)

lemma *nu-below-mu-nu-L-mu-nu-apx-nu-L*: *has-least-fixpoint f* $\wedge$ *has-greatest-fixpoint f* $\wedge$ *nu-below-mu-nu-L f* $\longrightarrow$ *mu-nu-apx-nu-L f*
  by (*metis mu-nu-apx-nu-L-def mu-nu-apx-nu-def n-equal-l nu-below-mu-nu-2-mu-nu-apx-nu nu-below-mu-nu-L-nu-below-mu-nu nu-below-mu-nu-nu-below-mu-nu-2*)

lemma *nu-below-mu-nu-L-mu-nu-apx-meet-L*: *has-least-fixpoint f* $\wedge$ *has-greatest-fixpoint f* $\wedge$ *nu-below-mu-nu-L f* $\longrightarrow$ *mu-nu-apx-meet-L f*
  by (*metis mu-nu-apx-meet-L-def mu-nu-apx-meet-def mu-nu-apx-nu-mu-nu-apx-meet n-equal-l nu-below-mu-nu-2-mu-nu-apx-nu nu-below-mu-nu-L-nu-below-mu-nu nu-below-mu-nu-nu-below-mu-nu-2*)

lemma *mu-nu-apx-nu-L-nu-below-mu-nu-L*: *has-least-fixpoint f* $\wedge$ *has-greatest-fixpoint f* $\wedge$ *mu-nu-apx-nu-L f* $\longrightarrow$ *nu-below-mu-nu-L f*
proof
  let $?n = \mu\ f + d(\nu\ f\ ;\ 0)\ ;\ L$
  let $?l = \mu\ f + (\nu\ f \frown L)$
  assume *1*: *has-least-fixpoint f* $\wedge$ *has-greatest-fixpoint f* $\wedge$ *mu-nu-apx-nu-L f*
  hence $d(L)\ ;\ \nu\ f \leq\ ?n + d(?n\ ;\ 0)\ ;\ T$
    by (*metis apx-def mu-nu-apx-nu-L-def*)
  also have ... $\leq\ ?n + d(?l\ ;\ 0)\ ;\ T$
    by (*metis add-right-isotone d-isotone mult-left-isotone n-below-l*)
  also have ... $\leq\ ?n + d(\nu\ f\ ;\ 0)\ ;\ T$ **using** *1*
    by (*metis add-right-isotone d-isotone l-below-nu mult-left-isotone*)
  finally show *nu-below-mu-nu-L f*
    by (*metis add-associative add-right-top mult-left-dist-add nu-below-mu-nu-L-def*)
qed

lemma *kappa-mu-nu-L-mu-nu-apx-nu-L*: *has-greatest-fixpoint f* $\wedge$ *kappa-mu-nu-L f* $\longrightarrow$ *mu-nu-apx-nu-L f*
  by (*metis mu-nu-apx-nu-L-def kappa-apx-below-nu kappa-mu-nu-L-def*)

lemma *mu-nu-apx-meet-L-mu-nu-apx-nu-L*: *mu-nu-apx-meet-L f* $\longrightarrow$ *mu-nu-apx-nu-L f*
  by (*smt apx-meet-same has-apx-meet-def is-apx-meet-def mu-nu-apx-meet-L-def mu-nu-apx-nu-L-def*)

lemma *kappa-mu-nu-L-nu-below-mu-nu-L*: *has-least-fixpoint f* $\wedge$ *has-greatest-fixpoint f* $\wedge$ *kappa-mu-nu-L f* $\longrightarrow$ *nu-below-mu-nu-L f*
  by (*metis mu-nu-apx-nu-L-nu-below-mu-nu-L kappa-mu-nu-L-mu-nu-apx-nu-L*)

end

class *itering-apx* = *domain-itering-lattice-L* + *domain-semiring-lattice-apx*

begin

lemma *circ-apx-isotone*: $x \sqsubseteq y \longrightarrow x^\circ \sqsubseteq y^\circ$
proof
  assume $x \sqsubseteq y$
  hence *1*: $x \le y + L \wedge d(L) ; y \le x + d(x ; 0) ; T$
    by (*metis apx-def*)
  have $d(L) ; y^\circ \le (d(L) ; y)^\circ$
    by (*smt circ-reflexive circ-transitive-equal d-below-one d-circ-import l2 mult-left-isotone order-trans*)
  also have ... $\le x^\circ ; (d(x ; 0) ; T ; x^\circ)^\circ$ **using** *1*
    by (*metis circ-add-1 circ-isotone*)
  also have ... $= x^\circ + x^\circ ; d(x ; 0) ; T$
    by (*metis circ-left-top mult-associative mult-left-dist-add mult-right-one mult-top-circ*)
  also have ... $\le x^\circ + d(x^\circ ; x ; 0) ; T$
    by (*metis add-right-isotone mult-associative mult-domain-top*)
  finally have *2*: $d(L) ; y^\circ \le x^\circ + d(x^\circ ; 0) ; T$
    by (*metis circ-plus-same d0-circ-left-unfold*)
  have $x^\circ \le y^\circ ; L^\circ$ **using** *1*
    by (*metis circ-add-1 circ-back-loop-fixpoint circ-isotone l40 less-eq-def mult-associative*)
  also have ... $= y^\circ + y^\circ ; L$
    by (*metis add-commutative circ-L mult-left-dist-add mult-right-one*)
  also have ... $\le y^\circ + y^\circ ; 0 + L$
    by (*metis add-associative add-right-isotone l14*)
  finally have $x^\circ \le y^\circ + L$
    by (*metis add-commutative less-eq-def zero-right-mult-decreasing*)
  thus $x^\circ \sqsubseteq y^\circ$ **using** *2*
    by (*metis apx-def*)
qed

end

class *omega-algebra-apx* = *domain-omega-algebra-lattice-L* + *domain-semiring-lattice-apx*

sublocale *omega-algebra-apx* < *star*!: *itering-apx* **where** *circ* = *star* **..**

context *omega-algebra-apx*

begin

lemma *omega-apx-isotone*: $x \sqsubseteq y \longrightarrow x^\omega \sqsubseteq y^\omega$
proof
  assume $x \sqsubseteq y$
  hence *1*: $x \le y + L \wedge d(L) ; y \le x + d(x ; 0) ; T$
    by (*metis apx-def*)
  have $d(L) ; y^\omega = (d(L) ; y)^\omega$
    by (*metis d-omega-export l2*)
  also have ... $\le (x + d(x ; 0) ; T)^\omega$ **using** *1*
    by (*metis omega-isotone*)
  also have ... $= (x^\star ; d(x ; 0) ; T)^\omega + (x^\star ; d(x ; 0) ; T)^\star ; x^\omega$
    by (*metis mult-associative omega-decompose*)
  also have ... $\le x^\star ; d(x ; 0) ; T + (x^\star ; d(x ; 0) ; T)^\star ; x^\omega$
    by (*metis add-left-isotone mult-top-omega*)
  also have ... $= x^\star ; d(x ; 0) ; T + (1 + x^\star ; d(x ; 0) ; T ; (x^\star ; d(x ; 0) ; T)^\star) ; x^\omega$
    by (*metis mult-associative star.circ-left-top star.mult-top-circ*)
  also have ... $\le x^\omega + x^\star ; d(x ; 0) ; T$
    by (*smt add-isotone add-least-upper-bound mult-associative mult-left-one mult-right-dist-add mult-right-isotone order-refl top-greatest*)
  also have ... $\le x^\omega + d(x^\star ; x ; 0) ; T$
    by (*metis add-right-isotone mult-associative mult-domain-top*)
  also have ... $\le x^\omega + d(x^\star ; 0) ; T$
    by (*metis dL-star.d0-circ-left-unfold eq-refl star.circ-plus-same*)
  finally have *2*: $d(L) ; y^\omega \le x^\omega + d(x^\omega ; 0) ; T$
    by (*smt add-right-isotone d0-star-below-d0-omega mult-left-isotone order-trans*)
  have $x^\omega \le (y + L)^\omega$ **using** *1*
    by (*metis omega-isotone*)
  also have ... $= (y^\star ; L)^\omega + (y^\star ; L)^\star ; y^\omega$
    by (*metis omega-decompose*)
  also have ... $= y^\star ; L ; (y^\star ; L)^\omega + (y^\star ; L)^\star ; y^\omega$
    by (*metis omega-unfold*)

also have ... $\leq y^\star$; $L + (y^\star$; $L)^\star$; y^ω
by (*metis add-left-isotone l40 mult-associative mult-right-isotone*)
also have ... $= y^\star$; $L + (1 + y^\star$; L ; $(y^\star$; $L)^\star)$; y^ω
by (*metis star.circ-left-unfold*)
also have ... $\leq y^\star$; $L + y^\omega$
by (*metis add-commutative add-least-upper-bound add-right-upper-bound dL-star.mult-L-circ-mult-below mult-associative star.circ-mult star.circ-slide*)
also have ... $\leq y^\star$; $0 + L + y^\omega$
by (*metis add-left-isotone l14*)
finally have $x^\omega \leq y^\omega + L$
by (*metis add-associative add-commutative less-eq-def star-zero-below-omega*)
thus $x^\omega \sqsubseteq y^\omega$ **using** *2*
by (*metis apx-def*)
qed

lemma *combined-apx-isotone*: $x \sqsubseteq y \longrightarrow (x^\omega \frown L) + x^\star$; $z \sqsubseteq (y^\omega \frown L) + y^\star$; z
by (*metis add-apx-isotone mult-apx-left-isotone omega-apx-isotone star.circ-apx-isotone meet-L-apx-isotone*)

lemma *d-split-nu-mu*: $d(L)$; $(y^\omega + y^\star$; $z) \leq y^\star$; $z + ((y^\omega + y^\star$; $z) \frown L) + d((y^\omega + y^\star$; $z)$; $0)$; T
proof –
have $d(L)$; $y^\omega \leq (y^\omega \frown L) + d(y^\omega$; $0)$; T
by (*metis add-right-isotone l31 l91 omega-vector*)
hence $d(L)$; $(y^\omega + y^\star$; $z) \leq y^\star$; $z + (y^\omega \frown L) + d(y^\omega$; $0)$; T
by (*smt add-associative add-commutative add-isotone d-mult-below mult-left-dist-add*)
thus *?thesis*
by (*smt add-commutative add-isotone add-right-isotone add-right-upper-bound d-isotone meet-commutative meet.add-right-isotone mult-left-isotone order-trans*)
qed

lemma *loop-exists*: $d(L)$; ν $(\lambda x$. y ; $x + z) \leq \mu$ $(\lambda x$. y ; $x + z) + (\nu$ $(\lambda x$. y ; $x + z) \frown L) + d(\nu$ $(\lambda x$. y ; $x + z)$; $0)$; T
by (*metis d-split-nu-mu omega-loop-nu star-loop-mu*)

lemma *loop-isotone*: *isotone* $(\lambda x$. y ; $x + z)$
by (*smt add-commutative add-right-isotone isotone-def mult-right-isotone*)

lemma *loop-apx-isotone*: *apx.isotone* $(\lambda x$. y ; $x + z)$
by (*smt add-apx-left-isotone apx.isotone-def mult-apx-right-isotone*)

lemma *loop-has-least-fixpoint*: *has-least-fixpoint* $(\lambda x$. y ; $x + z)$
by (*metis has-least-fixpoint-def star-loop-is-least-fixpoint*)

lemma *loop-has-greatest-fixpoint*: *has-greatest-fixpoint* $(\lambda x$. y ; $x + z)$
by (*metis has-greatest-fixpoint-def omega-loop-is-greatest-fixpoint*)

lemma *loop-apx-least-fixpoint*: *apx.is-least-fixpoint* $(\lambda x$. y ; $x + z)$ $(\mu$ $(\lambda x$. y ; $x + z) + (\nu$ $(\lambda x$. y ; $x + z) \frown L))$
using *apx.least-fixpoint-char loop-apx-isotone loop-exists loop-has-greatest-fixpoint loop-has-least-fixpoint loop-isotone nu-below-mu-nu-def nu-below-mu-nu-kappa-mu-nu kappa-mu-nu-def*
by *auto*

lemma *loop-has-apx-least-fixpoint*: *apx.has-least-fixpoint* $(\lambda x$. y ; $x + z)$
by (*metis apx.has-least-fixpoint-def loop-apx-least-fixpoint*)

lemma *loop-semantics*: κ $(\lambda x$. y ; $x + z) = \mu$ $(\lambda x$. y ; $x + z) + (\nu$ $(\lambda x$. y ; $x + z) \frown L)$
by (*metis apx.least-fixpoint-char loop-apx-least-fixpoint*)

lemma *loop-semantics-kappa-mu-nu*: κ $(\lambda x$. y ; $x + z) = (y^\omega \frown L) + y^\star$; z
proof –
have κ $(\lambda x$. y ; $x + z) = y^\star$; $z + ((y^\omega + y^\star$; $z) \frown L)$
by (*metis loop-semantics omega-loop-nu star-loop-mu*)
thus *?thesis*
by (*smt add-absorb add-associative add-commutative add-left-dist-meet*)
qed

lemma *loop-semantics-kappa-mu-nu-domain*: κ $(\lambda x$. y ; $x + z) = d(y^\omega)$; $L + y^\star$; z
by (*metis loop-semantics-kappa-mu-nu omega-meet-L*)

lemma *loop-semantics-apx-isotone*: $w \sqsubseteq y \longrightarrow \kappa$ $(\lambda x$. w ; $x + z) \sqsubseteq \kappa$ $(\lambda x$. y ; $x + z)$
by (*metis loop-semantics-kappa-mu-nu combined-apx-isotone*)

end

end

29 ExtendedDesigns

theory *ExtendedDesigns*

imports *OmegaAlgebra Domain*

begin

class *domain-semiring-L-below* = *left-zero-domain-semiring* + *L* +
  assumes *L-left-zero-below*: $L \; ; \; x \leq L$
  assumes *mult-L-split*: $x \; ; \; L = x \; ; \; 0 + d(x) \; ; \; L$

begin

lemma *d-zero-mult-L*: $d(x \; ; \; 0) \; ; \; L \leq x$
  by (*metis add-least-upper-bound mult-L-split mult-associative mult-left-zero zero-right-mult-decreasing*)

lemma *mult-L*: $x \; ; \; L \leq x \; ; \; 0 + L$
  by (*metis add-right-isotone d-mult-below mult-L-split*)

lemma *d-mult-L*: $d(x) \; ; \; L \leq x \; ; \; L$
  by (*metis add-right-divisibility mult-L-split*)

lemma *d-L-split*: $x \; ; \; d(y) \; ; \; L = x \; ; \; 0 + d(x \; ; \; y) \; ; \; L$
  by (*metis d-commutative d-mult-d d-zero mult-L-split mult-associative mult-left-zero*)

lemma *d-mult-mult-L*: $d(x \; ; \; y) \; ; \; L \leq x \; ; \; d(y) \; ; \; L$
  by (*metis add-right-divisibility d-L-split*)

lemma *L-L*: $L \; ; \; L = L$
  by (*metis d-restrict-equals less-eq-def mult-L-split zero-right-mult-decreasing*)

end

class *antidomain-semiring-L* = *left-zero-antidomain-semiring* + *L* +
  assumes *d-zero-mult-L*: $d(x \; ; \; 0) \; ; \; L \leq x$
  assumes *d-L-zero*   : $d(L \; ; \; 0) = 1$
  assumes *mult-L*   : $x \; ; \; L \leq x \; ; \; 0 + L$

begin

lemma *L-left-zero*: $L \; ; \; x = L$
  by (*metis d-L-zero d-zero-mult-L less-def less-le mult-associative mult-left-one mult-left-zero zero-right-mult-decreasing*)

subclass *domain-semiring-L-below*
  apply *unfold-locales*
  apply (*metis L-left-zero order-refl*)
  apply (*rule antisym*)
  apply (*smt d-restrict-equals less-eq-def mult-L mult-associative mult-left-dist-add*)
   apply (*metis add-least-upper-bound d-L-zero d-mult-d d-zero-mult-L mult-associative mult-right-isotone mult-right-one zero-least*)
  done

end

class *ed-below* = *bounded-left-zero-omega-algebra* + *domain-semiring-L-below* + *Omega* +
  assumes *Omega-def*: $x^\Omega = d(x^\omega) \; ; \; L + x^\star$

begin

lemma *Omega-isotone*: $x \leq y \longrightarrow x^\Omega \leq y^\Omega$
  by (*metis Omega-def add-isotone d-isotone mult-left-isotone omega-isotone star.circ-isotone*)

lemma *star-below-Omega*: $x^\star \leq x^\Omega$
  by (*metis Omega-def add-right-upper-bound*)

lemma *one-below-Omega*: $1 \leq x^\Omega$
  by (*metis add-least-upper-bound star.circ-plus-one star-below-Omega*)

lemma *L-left-zero-star*: $L ; x^\star = L$
  by (*metis L-left-zero-below add-right-upper-bound antisym star.circ-back-loop-fixpoint*)

lemma *L-left-zero-Omega*: $L ; x^\Omega = L$
  by (*metis L-left-zero-below L-left-zero-star Omega-def less-eq-def mult-left-dist-add*)

lemma *mult-L-star*: $(x ; L)^\star = 1 + x ; L$
  by (*metis L-left-zero-star mult-associative star.circ-left-unfold*)

lemma *mult-L-omega-below*: $(x ; L)^\omega \leq x ; L$
  by (*metis L-left-zero-below mult-right-isotone omega-slide*)

lemma *mult-L-add-star*: $(x ; L + y)^\star = y^\star + y^\star ; x ; L$
  by (*metis L-left-zero-star add-commutative mult-associative star.circ-unfold-sum*)

lemma *mult-L-add-omega-below*: $(x ; L + y)^\omega \leq y^\omega + y^\star ; x ; L$
proof −
  have $(x ; L + y)^\omega = (y^\star ; x ; L)^\omega + (y^\star ; x ; L)^\star ; y^\omega$
    by (*metis add-commutative mult-associative omega-decompose*)
  also have ... $\leq y^\star ; x ; L + (y^\star ; x ; L)^\star ; y^\omega$
    by (*metis add-left-isotone mult-L-omega-below*)
  also have ... $= y^\star ; x ; L + y^\star ; x ; L ; y^\omega + y^\omega$
    by (*smt L-left-zero-star add-associative add-commutative mult-associative star.circ-loop-fixpoint*)
  also have ... $\leq y^\omega + y^\star ; x ; L$
    by (*metis L-left-zero-star add-commutative eq-refl mult-associative star.circ-back-loop-fixpoint*)
  finally show *?thesis*
    .
qed

lemma *mult-L-add-circ*: $(x ; L + y)^\Omega = d(y^\omega) ; L + y^\star + y^\star ; x ; L$
proof −
  have $(x ; L + y)^\Omega = d((x ; L + y)^\omega) ; L + (x ; L + y)^\star$
    by (*metis Omega-def*)
  also have ... $\leq d(y^\omega + y^\star ; x ; L) ; L + (x ; L + y)^\star$
    by (*metis add-left-isotone d-isotone mult-L-add-omega-below mult-left-isotone*)
  also have ... $= d(y^\omega) ; L + d(y^\star ; x ; L) ; L + (x ; L + y)^\star$
    by (*metis d-dist-add mult-right-dist-add*)
  also have ... $\leq d(y^\omega) ; L + y^\star ; x ; L ; L + (x ; L + y)^\star$
    by (*metis add-left-isotone add-right-isotone d-mult-L*)
  also have ... $= d(y^\omega) ; L + y^\star + y^\star ; x ; L$
    by (*smt L-L add-associative add-commutative less-eq-def mult-L-add-star mult-associative order-refl*)
  finally have *1*: $(x ; L + y)^\Omega \leq d(y^\omega) ; L + y^\star + y^\star ; x ; L$
    .
  have *2*: $d(y^\omega) ; L \leq (x ; L + y)^\Omega$
    by (*metis Omega-def add-left-upper-bound add-right-upper-bound d-isotone mult-left-isotone omega-isotone order-trans*)
  have $y^\star + y^\star ; x ; L \leq (x ; L + y)^\Omega$
    by (*metis Omega-def add-right-upper-bound mult-L-add-star*)
  hence $d(y^\omega) ; L + y^\star + y^\star ; x ; L \leq (x ; L + y)^\Omega$ **using** *2*
    by (*metis Omega-def add-least-upper-bound add-right-upper-bound mult-L-add-star*)
  thus *?thesis* **using** *1*
    by (*metis antisym*)
qed

lemma *circ-add-d*: $(x^\Omega ; y)^\Omega ; x^\Omega = d((x^\star ; y)^\omega) ; L + ((x^\star ; y)^\star ; x^\star + (x^\star ; y)^\star ; d(x^\omega) ; L)$
proof −
  have $(x^\Omega ; y)^\Omega ; x^\Omega = ((d(x^\omega) ; L + x^\star) ; y)^\Omega ; x^\Omega$
    by (*metis Omega-def*)
  also have ... $= (d(x^\omega) ; L ; y + x^\star ; y)^\Omega ; x^\Omega$
    by (*metis mult-right-dist-add*)
  also have ... $\leq (d(x^\omega) ; L + x^\star ; y)^\Omega ; x^\Omega$
    by (*metis L-left-zero-below Omega-isotone add-left-isotone mult-associative mult-left-isotone mult-right-isotone*)
  also have ... $= (d((x^\star ; y)^\omega) ; L + (x^\star ; y)^\star + (x^\star ; y)^\star ; d(x^\omega) ; L) ; x^\Omega$
    by (*metis mult-L-add-circ*)
  also have ... $= d((x^\star ; y)^\omega) ; L ; x^\Omega + (x^\star ; y)^\star ; x^\Omega + (x^\star ; y)^\star ; d(x^\omega) ; L ; x^\Omega$
    by (*metis mult-right-dist-add*)
  also have ... $= d((x^\star ; y)^\omega) ; L + (x^\star ; y)^\star ; x^\Omega + (x^\star ; y)^\star ; d(x^\omega) ; L$
    by (*smt L-left-zero-Omega mult-associative*)
  also have ... $= d((x^\star ; y)^\omega) ; L + ((x^\star ; y)^\star ; x^\star + (x^\star ; y)^\star ; d(x^\omega) ; L)$
    by (*smt Omega-def add-associative add-commutative add-idempotent mult-associative mult-left-dist-add*)

finally have *1*: $(x^\Omega ; y)^\Omega ; x^\Omega \le d((x^\star ; y)^\omega) ; L + ((x^\star ; y)^\star ; x^\star + (x^\star ; y)^\star ; d(x^\omega) ; L)$
.
have $d((x^\star ; y)^\omega) ; L \le (x^\Omega ; y)^\Omega$
by (*metis Omega-def Omega-isotone add-commutative add-right-upper-bound mult-left-isotone order-trans*)
also have ... $\le (x^\Omega ; y)^\Omega ; x^\Omega$
by (*metis mult-right-isotone mult-right-one one-below-Omega*)
finally have *2*: $d((x^\star ; y)^\omega) ; L \le (x^\Omega ; y)^\Omega ; x^\Omega$
.
have *3*: $(x^\star ; y)^\star ; x^\star \le (x^\Omega ; y)^\Omega ; x^\Omega$
by (*metis Omega-isotone mult-left-isotone mult-right-isotone order-trans star-below-Omega*)
have $(x^\star ; y)^\star ; d(x^\omega) ; L \le (x^\star ; y)^\star ; x^\Omega$
by (*metis Omega-def add-commutative mult-associative mult-left-sub-dist-add-right*)
also have ... $\le (x^\Omega ; y)^\Omega ; x^\Omega$
by (*metis Omega-isotone mult-left-isotone order-trans star-below-Omega*)
finally have $d((x^\star ; y)^\omega) ; L + ((x^\star ; y)^\star ; x^\star + (x^\star ; y)^\star ; d(x^\omega) ; L) \le (x^\Omega ; y)^\Omega ; x^\Omega$ **using** *2 3*
by (*smt add-associative less-eq-def*)
thus *?thesis* **using** *1*
by (*metis antisym*)
qed

lemma *mult-L-omega*: $(x ; L)^\omega = x ; L$ **nitpick** [*expect=genuine*] **oops**
lemma *mult-L-add-omega*: $(x ; L + y)^\omega = y^\omega + y^\star ; x ; L$ **nitpick** [*expect=genuine*] **oops**
lemma *d-Omega-circ-simulate-right-plus*: $z ; x \le y ; y^\Omega ; z + w \longrightarrow z ; x^\Omega \le y^\Omega ; (z + w ; x^\Omega)$ **nitpick** [*expect=genuine*] **oops**
lemma *d-Omega-circ-simulate-left-plus*: $x ; z \le z ; y^\Omega + w \longrightarrow x^\Omega ; z \le (z + x^\Omega ; w) ; y^\Omega$ **nitpick** [*expect=genuine*] **oops**

end

class *ed* = *ed-below* +
assumes *L-left-zero*: $L ; x = L$

begin

lemma *mult-L-omega*: $(x ; L)^\omega = x ; L$
by (*metis L-left-zero omega-slide*)

lemma *mult-L-add-omega*: $(x ; L + y)^\omega = y^\omega + y^\star ; x ; L$
by (*smt L-left-zero add-commutative add-left-upper-bound less-eq-def mult-L-omega mult-L-star mult-associative mult-left-one mult-right-dist-add omega-decompose*)

lemma *d-Omega-circ-simulate-right-plus*: $z ; x \le y ; y^\Omega ; z + w \longrightarrow z ; x^\Omega \le y^\Omega ; (z + w ; x^\Omega)$
proof
assume $z ; x \le y ; y^\Omega ; z + w$
hence $z ; x \le y ; d(y^\omega) ; L ; z + y ; y^\star ; z + w$
by (*metis Omega-def mult-associative mult-left-dist-add mult-right-dist-add*)
also have ... $\le y ; d(y^\omega) ; L + y ; y^\star ; z + w$
by (*metis L-left-zero-below add-commutative add-right-isotone mult-associative mult-right-isotone*)
also have ... $= y ; 0 + d(y ; y^\omega) ; L + y ; y^\star ; z + w$
by (*metis d-L-split*)
also have ... $= d(y^\omega) ; L + y ; y^\star ; z + w$
by (*smt add-associative add-commutative add-left-zero mult-associative mult-left-dist-add omega-unfold*)
finally have *1*: $z ; x \le d(y^\omega) ; L + y ; y^\star ; z + w$
.
have $(d(y^\omega) ; L + y^\star ; z + y^\star ; w ; d(x^\omega) ; L + y^\star ; w ; x^\star) ; x = d(y^\omega) ; L ; x + y^\star ; z ; x + y^\star ; w ; d(x^\omega) ; L ; x + y^\star ; w ; x^\star ; x$
by (*metis mult-right-dist-add*)
also have ... $\le d(y^\omega) ; L + y^\star ; z ; x + y^\star ; w ; d(x^\omega) ; L ; x + y^\star ; w ; x^\star ; x$
by (*metis L-left-zero-below add-left-isotone mult-associative mult-right-isotone*)
also have ... $\le d(y^\omega) ; L + y^\star ; z ; x + y^\star ; w ; d(x^\omega) ; L + y^\star ; w ; x^\star ; x$
by (*metis L-left-zero-below add-commutative add-left-isotone mult-associative mult-right-isotone*)
also have ... $\le d(y^\omega) ; L + y^\star ; z ; x + y^\star ; w ; d(x^\omega) ; L + y^\star ; w ; x^\star$
by (*metis add-left-upper-bound add-right-isotone star.circ-back-loop-fixpoint*)
also have ... $\le d(y^\omega) ; L + y^\star ; (d(y^\omega) ; L + y ; y^\star ; z + w) + y^\star ; w ; d(x^\omega) ; L + y^\star ; w ; x^\star$ **using** *1*
by (*smt add-left-isotone add-right-isotone less-eq-def mult-associative mult-left-dist-add*)
also have ... $= d(y^\omega) ; L + y^\star ; y ; y^\star ; z + y^\star ; w ; d(x^\omega) ; L + y^\star ; w ; x^\star$
by (*smt2 add-associative add-commutative add-idempotent mult-associative mult-left-dist-add d-L-split star.circ-back-loop-fixpoint star-mult-omega*)
also have ... $\le d(y^\omega) ; L + y^\star ; z + y^\star ; w ; d(x^\omega) ; L + y^\star ; w ; x^\star$
by (*metis add-left-isotone add-right-isotone mult-left-isotone star.circ-plus-same star.circ-transitive-equal*

star.left-plus-below-circ)
finally have *2*: $z ; x^\star \leq d(y^\omega) ; L + y^\star ; z + y^\star ; w ; d(x^\omega) ; L + y^\star ; w ; x^\star$
by (*smt add-least-upper-bound add-left-upper-bound star.circ-loop-fixpoint star-right-induct*)
have $z ; x ; x^\omega \leq y ; y^\star ; z ; x^\omega + d(y^\omega) ; L ; x^\omega + w ; x^\omega$ **using** *1*
by (*metis add-commutative mult-left-isotone mult-right-dist-add*)
also have ... $\leq y ; y^\star ; z ; x^\omega + d(y^\omega) ; L + w ; x^\omega$
by (*metis L-left-zero-below add-commutative add-right-isotone mult-associative mult-right-isotone*)
finally have $z ; x^\omega \leq y^\omega + y^\star ; d(y^\omega) ; L + y^\star ; w ; x^\omega$
by (*smt add-associative add-commutative left-plus-omega mult-associative mult-left-dist-add omega-induct omega-unfold star.left-plus-circ*)
hence $z ; x^\omega \leq y^\omega + y^\star ; d(y^\omega) ; L + y^\star ; w ; x^\omega$
by (*smt add-associative add-commutative left-plus-omega mult-associative mult-left-dist-add omega-induct omega-unfold star.left-plus-circ*)
hence $z ; x^\omega \leq y^\omega + y^\star ; w ; x^\omega$
by (*metis add-commutative d-mult-L less-eq-def mult-associative mult-right-isotone omega-sub-vector order-trans star-mult-omega*)
hence $d(z ; x^\omega) ; L \leq d(y^\omega) ; L + y^\star ; w ; d(x^\omega) ; L$
by (*smt add-associative add-commutative d-L-split d-dist-add less-eq-def mult-right-dist-add*)
hence $z ; d(x^\omega) ; L \leq z ; 0 + d(y^\omega) ; L + y^\star ; w ; d(x^\omega) ; L$
by (*metis add-associative add-right-isotone d-L-split*)
also have ... $\leq y^\star ; z + d(y^\omega) ; L + y^\star ; w ; d(x^\omega) ; L$
by (*smt2 add-commutative add-left-isotone add-left-upper-bound order-trans star.circ-loop-fixpoint zero-right-mult-decreasing*)
finally have $z ; d(x^\omega) ; L \leq d(y^\omega) ; L + y^\star ; z + y^\star ; w ; d(x^\omega) ; L + y^\star ; w ; x^\star$
by (*smt2 add-commutative add-left-upper-bound order-trans*)
thus $z ; x^\Omega \leq y^\Omega ; (z + w ; x^\Omega)$ **using** *2*
by (*smt L-left-zero Omega-def add-associative less-eq-def mult-associative mult-left-dist-add mult-right-dist-add*)
qed

lemma *d-Omega-circ-simulate-left-plus*: $x ; z \leq z ; y^\Omega + w \longrightarrow x^\Omega ; z \leq (z + x^\Omega ; w) ; y^\Omega$
proof
assume *1*: $x ; z \leq z ; y^\Omega + w$
have $x ; (z ; d(y^\omega) ; L + z ; y^\star + d(x^\omega) ; L + x^\star ; w ; d(y^\omega) ; L + x^\star ; w ; y^\star) = x ; z ; d(y^\omega) ; L + x ; z ; y^\star + d(x^\omega) ; L + x ; x^\star ; w ; d(y^\omega) ; L + x ; x^\star ; w ; y^\star$
by (*smt add-associative add-commutative mult-associative mult-left-dist-add d-L-split omega-unfold*)
also have ... $\leq (z ; d(y^\omega) ; L + z ; y^\star + w) ; d(y^\omega) ; L + (z ; d(y^\omega) ; L + z ; y^\star + w) ; y^\star + d(x^\omega) ; L + x^\star ; w ; d(y^\omega) ; L + x^\star ; w ; y^\star$ **using** *1*
by (*smt Omega-def add-associative add-right-upper-bound less-eq-def mult-associative mult-left-dist-add mult-right-dist-add star.circ-loop-fixpoint*)
also have ... $= z ; d(y^\omega) ; L + z ; y^\star ; d(y^\omega) ; L + w ; d(y^\omega) ; L + z ; y^\star + w ; y^\star + d(x^\omega) ; L + x^\star ; w ; d(y^\omega) ; L + x^\star ; w ; y^\star$
by (*smt2 L-left-zero add-associative add-commutative add-idempotent mult-associative mult-right-dist-add star.circ-transitive-equal*)
also have ... $= z ; d(y^\omega) ; L + w ; d(y^\omega) ; L + z ; y^\star + w ; y^\star + d(x^\omega) ; L + x^\star ; w ; d(y^\omega) ; L + x^\star ; w ; y^\star$
by (*smt add-associative add-commutative add-idempotent less-eq-def mult-associative d-L-split star-mult-omega zero-right-mult-decreasing*)
finally have $x ; (z ; d(y^\omega) ; L + z ; y^\star + d(x^\omega) ; L + x^\star ; w ; d(y^\omega) ; L + x^\star ; w ; y^\star) \leq z ; d(y^\omega) ; L + z ; y^\star + d(x^\omega) ; L + x^\star ; w ; d(y^\omega) ; L + x^\star ; w ; y^\star$
by (*smt2 add-associative add-commutative add-idempotent mult-associative star.circ-loop-fixpoint*)
thus $x^\Omega ; z \leq (z + x^\Omega ; w) ; y^\Omega$
by (*smt L-left-zero Omega-def add-associative add-least-upper-bound add-left-upper-bound mult-associative mult-left-dist-add mult-right-dist-add star.circ-back-loop-fixpoint star-left-induct*)
qed

end

— Theorem 2.5 and Theorem 50.4

sublocale *ed* < *ed-omega*!: *itering* **where** *circ* = *Omega*
apply *unfold-locales*
apply (*smt add-associative add-commutative add-left-zero circ-add-d Omega-def mult-left-dist-add mult-right-dist-add d-L-split d-dist-add omega-decompose star.circ-add-1 star.circ-slide*)
apply (*smt L-left-zero add-associative add-commutative add-left-zero Omega-def mult-associative mult-left-dist-add mult-right-dist-add d-L-split omega-slide star.circ-mult*)
apply (*metis d-Omega-circ-simulate-right-plus*)
apply (*metis d-Omega-circ-simulate-left-plus*)
done

sublocale *ed* < *ed-star*!: *itering* **where** *circ* = *star* **..**

```
class ed-2 = ed-below + antidomain-semiring-L + Omega

begin

subclass ed
  apply unfold-locales
  apply (rule L-left-zero)
  done

end

end
```

30 Precondition

theory *Precondition*

imports *Tests*

begin

class *pre* =
  fixes *pre* :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixr** « *55*)

class *precondition* = *tests* + *pre* +
  assumes *pre-closed*: $x«-q = --(x«-q)$
  assumes *pre-seq*: $x;y«-q = x«y«-q$
  assumes *pre-lower-bound-right*: $x«-p;-q \le x«-q$
  assumes *pre-one-increasing*: $-q \le 1«-q$

begin

— Theorem 39.2

lemma *pre-sub-distr*: $x«-p;-q \le (x«-p);(x«-q)$
  by (*smt greatest-lower-bound pre-closed pre-lower-bound-right sub-comm sub-mult-closed*)

— Theorem 39.5

lemma *pre-below-one*: $x«-p \le 1$
  by (*metis one-greatest pre-closed*)

lemma *pre-lower-bound-left*: $x«-p;-q \le x«-p$
  by (*smt lower-bound-left pre-closed pre-sub-distr sub-mult-closed transitive*)

— Theorem 39.1

lemma *pre-iso*: $-p \le -q \longrightarrow x«-p \le x«-q$
  by (*metis leq-def pre-lower-bound-right*)

— Theorem 39.4 and Theorem 40.9

lemma *pre-below-pre-one*: $x«-p \le x«1$
  by (*metis one-def one-greatest pre-iso*)

— Theorem 39.3

lemma *pre-seq-below-pre-one*: $x;y«1 \le x«1$
  by (*metis one-def pre-below-pre-one pre-closed pre-seq*)

— Theorem 39.6

lemma *pre-compose*: $-p \le x«-q \wedge -q \le y«-r \longrightarrow -p \le x;y«-r$
  by (*metis pre-closed pre-iso transitive pre-seq*)

lemma *pre-test-test*: $-p;(-p«-q) = -p;-q$ **nitpick** [*expect=genuine*] **oops**
lemma *pre-test-promote*: $-p«-q = -p«-p;-q$ **nitpick** [*expect=genuine*] **oops**
lemma *pre-test*: $-p«-q = --p+-q$ **nitpick** [*expect=genuine*] **oops**
lemma *pre-test*: $-p«-q = -p;-q$ **nitpick** [*expect=genuine*] **oops**
lemma *pre-distr-mult*: $x«-p;-q = (x«-p);(x«-q)$ **nitpick** [*expect=genuine*] **oops**
lemma *pre-distr-plus*: $x«-p+-q = (x«-p);(x«-q)$ **nitpick** [*expect=genuine*] **oops**

end

class *precondition-test-test* = *precondition* +
  assumes *pre-test-test*: $-p;(-p«-q) = -p;-q$

begin

lemma *pre-one*: $1«-p = -p$
  by (*metis bs-mult-left-one one-def pre-closed pre-test-test*)

lemma *pre-import*: −p;(x«−q) = −p;(−p;x«−q)
  by (*metis pre-closed pre-seq pre-test-test*)

lemma *pre-import-composition*: −p;(−p;x;y«−q) = −p;(x«y«−q)
  by (*metis pre-closed pre-seq pre-import*)

lemma *pre-import-equiv*: −p ≤ x«−q ⟷ −p ≤ −p;x«−q
  by (*metis leq-def pre-closed pre-import*)

lemma *pre-import-equiv-mult*: −p;−q ≤ x«−s ⟷ −p;−q ≤ −q;x«−s
  by (*smt leq-def pre-closed sub-assoc sub-mult-closed pre-import*)

lemma *pre-test-promote*: −p«−q = −p«−p;−q **nitpick** [*expect=genuine*] **oops**
lemma *pre-test*: −p«−q = −−p+−q **nitpick** [*expect=genuine*] **oops**
lemma *pre-test*: −p«−q = −p;−q **nitpick** [*expect=genuine*] **oops**
lemma *pre-distr-mult*: x«−p;−q = (x«−p);(x«−q) **nitpick** [*expect=genuine*] **oops**
lemma *pre-distr-plus*: x«−p+−q = (x«−p);(x«−q) **nitpick** [*expect=genuine*] **oops**

end

class *precondition-promote* = *precondition* +
  assumes *pre-test-promote*: −p«−q = −p«−p;−q

begin

lemma *pre-mult-test-promote*: x;−p«−q = x;−p«−p;−q
  by (*metis pre-seq pre-test-promote sub-mult-closed*)

lemma *pre-test-test*: −p;(−p«−q) = −p;−q **nitpick** [*expect=genuine*] **oops**
lemma *pre-test*: −p«−q = −−p+−q **nitpick** [*expect=genuine*] **oops**
lemma *pre-test*: −p«−q = −p;−q **nitpick** [*expect=genuine*] **oops**
lemma *pre-distr-mult*: x«−p;−q = (x«−p);(x«−q) **nitpick** [*expect=genuine*] **oops**
lemma *pre-distr-plus*: x«−p+−q = (x«−p);(x«−q) **nitpick** [*expect=genuine*] **oops**

end

class *precondition-test-box* = *precondition* +
  assumes *pre-test*: −p«−q = −−p+−q

begin

lemma *pre-test-neg*: −−p;(−p«−q) = −−p
  by (*metis mult-absorb pre-test*)

lemma *pre-zero*: 0«−q = 1
  by (*metis one-compl one-def plus-left-one pre-test*)

lemma *pre-export*: −p;x«−q = −−p+(x«−q)
  by (*metis pre-closed pre-seq pre-test*)

lemma *pre-neg-mult*: −−p ≤ −p;x«−q
  by (*metis leq-def pre-closed pre-seq pre-test-neg*)

lemma *pre-test-test-same*: −p«−p = 1
  by (*metis plus-comm plus-compl pre-test*)

lemma *test-below-pre-test-mult*: −q ≤ −p«−p;−q
  by (*metis pre-test reflexive shunting sub-mult-closed*)

lemma *test-below-pre-test*: −q ≤ −p«−q
  by (*metis pre-test upper-bound-right*)

lemma *test-below-pre-test-2*: −−p ≤ −p«−q
  by (*metis pre-test upper-bound-left*)

lemma *pre-test-zero*: −p«0 = −−p
  by (*metis one-compl plus-right-zero pre-test*)

lemma *pre-test-one*: −p«1 = 1

```
  by (metis one-def plus-right-one pre-test)

subclass precondition-test-test
  apply unfold-locales
  apply (metis mult-compl-intro pre-test)
  done

subclass precondition-promote
  apply unfold-locales
  apply (metis plus-comm plus-compl-intro pre-test sub-mult-closed)
  done

lemma pre-test: −p«−q = −p;−q nitpick [expect=genuine] oops
lemma pre-distr-mult: x«−p;−q = (x«−p);(x«−q) nitpick [expect=genuine] oops
lemma pre-distr-plus: x«−p+−q = (x«−p);(x«−q) nitpick [expect=genuine] oops

end

class precondition-test-diamond = precondition +
  assumes pre-test: −p«−q = −p;−q

begin

lemma pre-test-neg: −−p;(−p«−q) = 0
  by (metis bs-mult-right-zero mult-compl pre-test sub-assoc sub-comm)

lemma pre-zero: 0«−q = 0
  by (metis bs-mult-left-zero one-compl pre-test)

lemma pre-export: −p;x«−q = −p;(x«−q)
  by (metis pre-closed pre-seq pre-test)

lemma pre-neg-mult: −p;x«−q ≤ −p
  by (metis lower-bound-left pre-closed pre-export)

lemma pre-test-test-same: −p«−p = −p
  by (metis mult-idempotent pre-test)

lemma test-above-pre-test-plus: −−p«−p+−q ≤ −q
  by (metis double-negation lower-bound-left mult-compl-intro plus-closed pre-test sub-comm)

lemma test-above-pre-test: −p«−q ≤ −q
  by (metis lower-bound-right pre-test)

lemma test-above-pre-test-2: −p«−q ≤ −p
  by (metis lower-bound-left pre-test)

lemma pre-test-zero: −p«0 = 0
  by (metis bs-mult-right-zero one-compl pre-test)

lemma pre-test-one: −p«1 = −p
  by (metis bs-mult-right-one one-def pre-test)

subclass precondition-test-test
  apply unfold-locales
  apply (metis mult-idempotent pre-export pre-test)
  done

subclass precondition-promote
  apply unfold-locales
  apply (metis mult-idempotent pre-seq pre-test)
  done

lemma pre-test: −p«−q = −−p+−q nitpick [expect=genuine] oops
lemma pre-distr-mult: x«−p;−q = (x«−p);(x«−q) nitpick [expect=genuine,card=6] oops
lemma pre-distr-plus: x«−p+−q = (x«−p);(x«−q) nitpick [expect=genuine] oops

end
```

class *precondition-distr-mult* = *precondition* +
  assumes *pre-distr-mult*: $x«-p;-q = (x«-p);(x«-q)$

begin

lemma *pre-test-test*: $-p;(-p«-q) = -p;-q$ **nitpick** [*expect=genuine*] **oops**
lemma *pre-test-promote*: $-p«-q = -p«-p;-q$ **nitpick** [*expect=genuine*] **oops**
lemma *pre-test*: $-p«-q = --p+-q$ **nitpick** [*expect=genuine*] **oops**
lemma *pre-test*: $-p«-q = -p;-q$ **nitpick** [*expect=genuine*] **oops**
lemma *pre-distr-plus*: $x«-p+-q = (x«-p);(x«-q)$ **nitpick** [*expect=genuine*] **oops**

end

class *precondition-distr-plus* = *precondition* +
  assumes *pre-distr-plus*: $x«-p+-q = (x«-p)+(x«-q)$

begin

lemma *pre-test-test*: $-p;(-p«-q) = -p;-q$ **nitpick** [*expect=genuine*] **oops**
lemma *pre-test-promote*: $-p«-q = -p«-p;-q$ **nitpick** [*expect=genuine*] **oops**
lemma *pre-test*: $-p«-q = --p+-q$ **nitpick** [*expect=genuine*] **oops**
lemma *pre-test*: $-p«-q = -p;-q$ **nitpick** [*expect=genuine*] **oops**
lemma *pre-distr-mult*: $x«-p;-q = (x«-p);(x«-q)$ **nitpick** [*expect=genuine*] **oops**

end

end

31 CompleteTests

```
theory CompleteTests

imports Tests

begin

class complete-tests = tests + Sup + Inf +
  assumes sup-test: test-set A ⟶ Sup A = −−Sup A
  assumes sup-upper: test-set A ∧ x ∈ A ⟶ x ≤ Sup A
  assumes sup-least: test-set A ∧ (∀ x∈A . x ≤ −y) ⟶ Sup A ≤ −y

begin

lemma Sup-isotone: test-set B ∧ A ⊆ B ⟶ Sup A ≤ Sup B
  by (smt subsetD sup-least sup-test sup-upper test-set-closed)

lemma mult-right-dist-sup: test-set A ⟶ Sup A ; −p = Sup { x;−p | x . x ∈ A }
proof
  assume 1: test-set A
  hence 2: test-set { x;−p | x . x ∈ A }
    by (simp add: mult-right-dist-test-set)
  have 3: Sup { x;−p | x . x ∈ A } ≤ Sup A ; −p using 1
    by (smt mem-Collect-eq mult-iso-left sub-mult-closed sup-test sup-least sup-upper test-set-def)
  have ∀ x∈A . x ≤ −−(−−Sup { x;−p | x . x ∈ A } + −−p)
  proof
    fix x
    assume 4: x ∈ A
    hence x;−p + −−p ≤ Sup { x;−p | x . x ∈ A } + −−p using 1 2
      by (smt mem-Collect-eq plus-iso-left sub-mult-closed sup-upper test-set-def sup-test)
    thus x ≤ −−(−−Sup { x;−p | x . x ∈ A } + −−p) using 1 2 4
      by (smt plus-closed plus-compl-intro sub-comm test-set-def transitive upper-bound-left sup-test)
  qed
  hence Sup A ≤ −−(−−Sup { x;−p | x . x ∈ A } + −−p) using 1
    by (simp add: sup-least)
  hence Sup A ; −p ≤ Sup { x;−p | x . x ∈ A } using 1 2
    by (smt plus-closed plus-comm shunting sub-comm sup-test)
  thus Sup A ; −p = Sup { x;−p | x . x ∈ A } using 1 2 3
    by (smt antisymmetric sub-mult-closed sup-test)
qed

lemma mult-left-dist-sup: test-set A ⟶ −p ; Sup A = Sup { −p;x | x . x ∈ A }
proof
  assume 1: test-set A
  hence 2: Sup A ; −p = Sup { x;−p | x . x ∈ A }
    by (simp add: mult-right-dist-sup)
  have 3: −p ; Sup A = Sup A ; −p using 1
    by (metis sub-comm sup-test)
  have { −p;x | x . x ∈ A } = { x;−p | x . x ∈ A }
    by (rule set-eqI, simp, metis 1 sub-comm test-set-def)
  thus −p ; Sup A = Sup { −p;x | x . x ∈ A } using 2 3
    by simp
qed

definition Sum :: (nat ⇒ 'a) ⇒ 'a
  where Sum f = Sup { f n | n::nat . True }

lemma Sum-test: test-seq t ⟶ Sum t = −−Sum t
  by (metis Sum-def sup-test test-seq-test-set)

lemma Sum-upper: test-seq t ⟶ t x ≤ Sum t
  by (smt Sum-def mem-Collect-eq sup-upper test-seq-test-set)

lemma Sum-least: test-seq t ∧ (∀ n . t n ≤ −p) ⟶ Sum t ≤ −p
  by (smt Sum-def mem-Collect-eq sup-least test-seq-test-set)

lemma mult-right-dist-Sum: test-seq t ∧ (∀ n . t n;−p ≤ −q) ⟶ Sum t;−p ≤ −q
  by (smt Sum-def mem-Collect-eq mult-right-dist-sup sub-mult-closed sup-least test-seq-test-set test-set-def)
```

```
lemma mult-left-dist-Sum: test-seq t ∧ (∀ n . −p;t n ≤ −q) ⟶ −p;Sum t ≤ −q
  by (smt Sum-def mem-Collect-eq mult-left-dist-sup sub-mult-closed sup-least test-seq-test-set test-set-def)

lemma pSum-below-Sum: test-seq t ⟶ pSum t m ≤ Sum t
  by (smt Sum-test Sum-upper bs-mult-right-one one-def pSum-below pSum-test test-seq-def)

lemma pSum-sup: test-seq t ⟶ pSum t m = Sup { t i | i . i ∈ {..<m} }
proof
  assume 1: test-seq t
  hence 2: test-set { t i | i . i ∈ {..<m} }
    by (smt mem-Collect-eq test-seq-def test-set-def)
  have ∀ y∈{ t i | i . i ∈ {..<m} } . y ≤ −−pSum t m
    by (simp, smt 1 pSum-test pSum-upper)
  hence 3: Sup { t i | i . i ∈ {..<m} } ≤ −−pSum t m using 2
    by (simp add: sup-least)
  have pSum t m ≤ Sup { t i | i . i ∈ {..<m} }
    apply (induct m)
    apply simp
    apply (smt sup-test test-set-def emptyE zero-least-test)
    proof −
      fix n
      assume 4: pSum t n ≤ Sup {t i |i. i ∈ {..<n}}
      have 5: test-set {t i |i. i ∈ {..<n}} using 1
        by (smt mem-Collect-eq test-seq-def test-set-def)
      have 6: test-set {t i |i. i < Suc n} using 1
        by (smt mem-Collect-eq test-seq-def test-set-def)
      hence 7: Sup {t i |i. i < Suc n} = −−Sup {t i |i. i < Suc n}
        by (smt sup-test)
      hence ∀ x∈{t i |i. i ∈ {..<n}} . x ≤ −−Sup {t i |i. i < Suc n} using 6
        apply simp
        apply rule+
        apply (rule mp)
        apply (rule sup-upper)
        apply simp
        by smt
      hence 8: Sup {t i |i. i ∈ {..<n}} ≤ −−Sup {t i |i. i < Suc n} using 5
        by (simp add: sup-least)
      have t n ∈ {t i |i. i < Suc n}
        by (simp, metis lessI)
      hence t n ≤ Sup {t i |i. i < Suc n} using 6
        by (smt sup-upper)
      hence pSum t n + t n ≤ Sup {t i |i. i <Suc n} using 1 4 5 7 8
        by (smt least-upper-bound test-seq-def pSum-test transitive sup-test)
      thus pSum t (Suc n) ≤ Sup {t i |i. i ∈ {..<Suc n}}
        by simp
    qed
  thus pSum t m = Sup { t i | i . i ∈ {..<m} } using 1 2 3
    by (smt antisymmetric sup-test pSum-test)
qed

definition Prod :: (nat ⇒ 'a) ⇒ 'a
  where Prod f = Inf { f n | n::nat . True }

lemma Sum-range: Sum f = Sup (range f)
  by (simp add: Sum-def image-def)

lemma Prod-range: Prod f = Inf (range f)
  by (simp add: Prod-def image-def)

end

end
```

32 Hoare

theory *Hoare*

imports *CompleteTests Precondition*

begin

class *ite* =
fixes *ite* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ (- $\lhd$ - $\rhd$ - [*58*,*58*,*58*] *57*)

class *hoare-triple* =
fixes *hoare-triple* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$ (- $\{\!|$ - $|\!\}$ - [*54*,*54*,*54*] *53*)

class *ifthenelse* = *precondition* + *ite* +
assumes *ite-pre*: $x\lhd-p\rhd y\ll-q = -p;(x\ll-q) + --p;(y\ll-q)$

begin

— Theorem 40.2

lemma *ite-pre-then*: $-p;(x\lhd-p\rhd y\ll-q) = -p;(x\ll-q)$
proof −
have $-p;(x\lhd-p\rhd y\ll-q) = -p;(x\ll-q) + 0;(y\ll-q)$
by (*smt ite-pre plus-absorb plus-distr-mult-left pre-closed sub-assoc sub-mult-closed zero-def*)
thus *?thesis*
by (*smt plus-absorb plus-right-zero pre-closed sub-mult-closed zero-def*)
qed

— Theorem 40.3

lemma *ite-pre-else*: $--p;(x\lhd-p\rhd y\ll-q) = --p;(y\ll-q)$
proof −
have $--p;(x\lhd-p\rhd y\ll-q) = 0;(x\ll-q) + --p;(y\ll-q)$
by (*smt ite-pre mult-distr-plus-left mult-idempotent pre-closed sub-assoc sub-mult-closed zero-def*)
thus *?thesis*
by (*smt mult-idempotent plus-left-zero pre-closed sub-assoc sub-mult-closed zero-def*)
qed

lemma *ite-import-mult-then*: $-p;-q \le x\ll-r \longrightarrow -p;-q \le x\lhd-p\rhd y\ll-r$
by (*smt ite-pre-then leq-def pre-closed sub-assoc sub-comm sub-mult-closed*)

lemma *ite-import-mult-else*: $--p;-q \le y\ll-r \longrightarrow --p;-q \le x\lhd-p\rhd y\ll-r$
by (*smt ite-pre-else leq-def pre-closed sub-assoc sub-comm sub-mult-closed*)

— Theorem 40.1

lemma *ite-import-mult*: $-p;-q \le x\ll-r \wedge --p;-q \le y\ll-r \longrightarrow -q \le x\lhd-p\rhd y\ll-r$
by (*metis ite-import-mult-then ite-import-mult-else leq-cases pre-closed*)

end

class *whiledo* = *ifthenelse* + *while* +
assumes *while-pre*: $-p\star x\ll-q = -p;(x\ll-p\star x\ll-q) + --p;-q$
assumes *while-post*: $-p\star x\ll-q = -p\star x\ll--p;-q$

begin

— Theorem 40.4

lemma *while-pre-then*: $-p;(-p\star x\ll-q) = -p;(x\ll-p\star x\ll-q)$
by (*smt pre-closed sub-comm while-pre wnf-lemma-1*)

— Theorem 40.5

lemma *while-pre-else*: $--p;(-p\star x\ll-q) = --p;-q$
by (*smt pre-closed sub-comm while-pre wnf-lemma-3*)

— Theorem 40.6

```
lemma while-pre-sub-1: −p⋆x«−q ≤ x;(−p⋆x)⊲−p⊳1«−q
  by (smt ite-pre-else ite-pre-then mult-iso-right plus-cases plus-iso-right pre-closed pre-one-increasing pre-seq sub-comm
sub-mult-closed while-pre)

— Theorem 40.7

lemma while-pre-sub-2: −p⋆x«−q ≤ x⊲−p⊳1«−p⋆x«−q
  by (smt ite-pre-else ite-pre-then mult-iso-right plus-cases plus-iso-right pre-closed pre-one-increasing sub-comm sub-mult-closed
while-pre while-pre-else)

— Theorem 40.8

lemma while-pre-compl: −−p ≤ −p⋆x«−−p
  by (metis lower-bound-right mult-idempotent pre-closed while-pre-else)

lemma while-pre-compl-one: −−p ≤ −p⋆x«1
  by (metis bs-mult-right-one lower-bound-right one-def pre-closed while-pre-else)

— Theorem 40.10

lemma while-export-equiv: −q ≤ −p⋆x«1 ⟷ −p;−q ≤ −p⋆x«1
  by (smt bs-mult-left-one leq-plus lower-bound-right one-def pre-closed shunting sub-comm while-pre-else)

lemma nat-test-pre: nat-test t s ∧ −q ≤ s ∧ (∀n . t n;−p;−q ≤ x«pSum t n;−q) ⟶ −q ≤ −p⋆x«−−p;−q
proof
  assume 1: nat-test t s ∧ −q ≤ s ∧ (∀n . t n;−p;−q ≤ x«pSum t n;−q)
  have 2: −q;−−p ≤ −p⋆x«−−p;−q
    by (smt leq-def mult-idempotent pre-closed sub-assoc sub-comm sub-mult-closed while-pre-else)
  have ∀n . t n;−p;−q ≤ −p⋆x«−−p;−q
  proof
    fix n
    show t n;−p;−q ≤ −p⋆x«−−p;−q
    proof (induct n rule: nat-less-induct)
      fix n
      have 3: t n = −−(t n) using 1
        by (smt nat-test-def)
      assume ∀m<n . t m;−p;−q ≤ −p⋆x«−−p;−q
      hence ∀m<n . t m;−p;−q + t m;−−p;−q ≤ −p⋆x«−−p;−q using 1 2
        by (smt least-upper-bound leq-def nat-test-def pre-closed sub-assoc sub-comm sub-mult-closed)
      hence ∀m<n . t m;−q ≤ −p⋆x«−−p;−q using 1
        by (smt bs-mult-right-one mult-distr-plus-left mult-distr-plus-right nat-test-def plus-compl sub-mult-closed)
      hence pSum t n;−q ≤ −p⋆x«−−p;−q using 1
        by (smt pSum-below-nat pre-closed sub-mult-closed)
      hence t n;−p;−q;(−p⋆x«−−p;−q) = t n;−p;−q using 1 3
        by (smt leq-def pSum-test-nat pre-closed pre-sub-distr sub-assoc sub-comm sub-mult-closed transitive while-pre-then)
      thus t n;−p;−q ≤ −p⋆x«−−p;−q using 3
        by (smt lower-bound-right pre-closed sub-mult-closed)
    qed
  qed
  hence −q;−p ≤ −p⋆x«−−p;−q using 1
    by (smt leq-def nat-test-def pre-closed sub-assoc sub-comm sub-mult-closed)
  thus −q ≤ −p⋆x«−−p;−q using 2
    by (smt bs-mult-right-one leq-def mult-distr-plus-left mult-distr-plus-right plus-compl pre-closed sub-mult-closed)
qed

lemma nat-test-pre-1: nat-test t s ∧ −r ≤ s ∧ −r ≤ −q ∧ (∀n . t n;−p;−q ≤ x«pSum t n;−q) ⟶ −r ≤ −p⋆x«−−p;−q
proof
  let ?qs = −q;s
  assume 1: nat-test t s ∧ −r ≤ s ∧ −r ≤ −q ∧ (∀n . t n;−p;−q ≤ x«pSum t n;−q)
  hence 2: −r ≤ ?qs
    by (metis greatest-lower-bound nat-test-def)
  have ∀n . t n;−p;?qs ≤ x«pSum t n;?qs using 1
    by (smt leq-def lower-bound-left nat-test-def pSum-below-sum pSum-test-nat sub-assoc sub-mult-closed transitive)
  hence ?qs ≤ −p⋆x«−−p;?qs using 1
    by (smt lower-bound-left lower-bound-right nat-test-def nat-test-pre pSum-test-nat pre-closed sub-assoc sub-mult-closed tran-
sitive)
  thus −r ≤ −p⋆x«−−p;−q using 1 2
    by (smt lower-bound-left nat-test-def pre-closed pre-iso sub-assoc sub-mult-closed transitive)
qed
```

```
lemma nat-test-pre-2: nat-test t s ∧ −r ≤ s ∧ (∀ n . t n;−p ≤ x«pSum t n) ⟶ −r ≤ −p⋆x«1
proof
  assume 1: nat-test t s ∧ −r ≤ s ∧ (∀ n . t n;−p ≤ x«pSum t n)
  hence −r ≤ −p⋆x«−−p;s
    by (smt leq-def nat-test-def nat-test-pre-1 pSum-below-sum pSum-test-nat sub-assoc sub-comm)
  thus −r ≤ −p⋆x«1 using 1
    by (smt nat-test-def one-def pre-below-pre-one pre-closed sub-mult-closed transitive)
qed

lemma nat-test-pre-3: nat-test t s ∧ −q ≤ s ∧ (∀ n . t n;−p;−q ≤ x«pSum t n;−q) ⟶ −q ≤ −p⋆x«1
proof −
  have −p⋆x«−−p;−q ≤ −p⋆x«1
    by (metis pre-below-pre-one sub-mult-closed)
  thus ?thesis
    by (smt nat-test-pre one-double-compl pre-closed sub-mult-closed transitive)
qed

definition aL :: 'a
  where aL ≡ 1⋆1«1

lemma aL-test: aL = −−aL
  by (metis aL-def one-def pre-closed)

end

class atoms = tests +
  fixes Atomic-program :: 'a set
  fixes Atomic-test :: 'a set
  assumes one-atomic-program: 1 ∈ Atomic-program
  assumes zero-atomic-test: 0 ∈ Atomic-test
  assumes atomic-test-test: p ∈ Atomic-test ⟶ p = −−p

class while-program = whiledo + atoms + power

begin

inductive-set Test-expression :: 'a set
  where atom-test: p ∈ Atomic-test ⟹ p ∈ Test-expression
      | neg-test:  p ∈ Test-expression ⟹ −p ∈ Test-expression
      | conj-test: p ∈ Test-expression ∧ q ∈ Test-expression ⟹ p;q ∈ Test-expression

lemma test-expression-test: p ∈ Test-expression ⟶ p = −−p
  apply rule
  apply (induct rule: Test-expression.induct)
  apply (metis atomic-test-test)
  apply simp
  apply (metis sub-mult-closed)
  done

lemma disj-test: p ∈ Test-expression ∧ q ∈ Test-expression ⟶ p+q ∈ Test-expression
  by (smt conj-test neg-test plus-def test-expression-test)

lemma zero-test-expression: 0 ∈ Test-expression
  by (metis atom-test zero-atomic-test)

lemma one-test-expression: 1 ∈ Test-expression
  by (metis neg-test one-def zero-test-expression)

lemma pSum-test-expression: (∀ n . t n ∈ Test-expression) ⟶ pSum t m ∈ Test-expression
  apply rule
  apply (induct m)
  apply (metis pSum.simps(1) zero-test-expression)
  apply (metis disj-test pSum.simps(2))
  done

inductive-set While-program :: 'a set
  where atom-prog:  x ∈ Atomic-program ⟹ x ∈ While-program
      | seq-prog:   x ∈ While-program ∧ y ∈ While-program ⟹ x;y ∈ While-program
      | cond-prog:  p ∈ Test-expression ∧ x ∈ While-program ∧ y ∈ While-program ⟹ x◁p▷y ∈ While-program
    | while-prog: p ∈ Test-expression ∧ x ∈ While-program ⟹ p⋆x ∈ While-program

lemma one-while-program: 1 ∈ While-program
  by (metis atom-prog one-atomic-program)

lemma power-while-program: x ∈ While-program ⟶ xˆm ∈ While-program
  apply rule
  apply (induct m)
  apply (metis one-while-program power-0)
  apply (metis seq-prog power-Suc)
  done

inductive-set Pre-expression :: 'a set
  where test-pre: p ∈ Test-expression ⟹ p ∈ Pre-expression
    | neg-pre:  p ∈ Pre-expression ⟹ −p ∈ Pre-expression
    | conj-pre: p ∈ Pre-expression ∧ q ∈ Pre-expression ⟹ p;q ∈ Pre-expression
    | pre-pre:  p ∈ Pre-expression ∧ x ∈ While-program ⟹ x«p ∈ Pre-expression

lemma pre-expression-test: p ∈ Pre-expression ⟶ p = −−p
  apply rule
  apply (induct rule: Pre-expression.induct)
  apply (metis test-expression-test)
  apply simp
  apply (metis sub-mult-closed)
  apply (metis pre-closed)
  done

lemma disj-pre: p ∈ Pre-expression ∧ q ∈ Pre-expression ⟶ p+q ∈ Pre-expression
  by (smt conj-pre neg-pre plus-def pre-expression-test)

lemma zero-pre-expression: 0 ∈ Pre-expression
  by (metis test-pre zero-test-expression)

lemma one-pre-expression: 1 ∈ Pre-expression
  by (metis test-pre one-test-expression)

lemma pSum-pre-expression: (∀ n . t n ∈ Pre-expression) ⟶ pSum t m ∈ Pre-expression
  apply rule
  apply (induct m)
  apply (metis pSum.simps(1) zero-pre-expression)
  apply (metis disj-pre pSum.simps(2))
  done

lemma aL-pre-expression: aL ∈ Pre-expression
  by (metis aL-def one-pre-expression one-test-expression one-while-program pre-pre while-prog)

end

class hoare-calculus = while-program + complete-tests

begin

definition tfun :: 'a ⇒ 'a ⇒ 'a ⇒ 'a ⇒ 'a
  where tfun p x q r ≡ p + (x«q;r)

lemma tfun-test: p = −−p ∧ q = −−q ∧ r = −−r ⟶ tfun p x q r = −−tfun p x q r
  by (smt tfun-def sub-mult-closed pre-closed plus-closed)

lemma tfun-pre-expression: x ∈ While-program ∧ p ∈ Pre-expression ∧ q ∈ Pre-expression ∧ r ∈ Pre-expression ⟶ tfun p
x q r ∈ Pre-expression
  by (metis tfun-def conj-pre disj-pre pre-pre)

lemma tfun-iso: p = −−p ∧ q = −−q ∧ r = −−r ∧ s = −−s ∧ r ≤ s ⟶ tfun p x q r ≤ tfun p x q s
  by (smt tfun-def mult-iso-right pre-iso sub-mult-closed plus-iso-right pre-closed)

definition tseq :: 'a ⇒ 'a ⇒ 'a ⇒ 'a ⇒ nat ⇒ 'a
  where tseq p x q r m ≡ (tfun p x q ˆ m) r

lemma tseq-test: p = −−p ∧ q = −−q ∧ r = −−r ⟶ tseq p x q r m = −−tseq p x q r m
```

  apply (*induct m*)
  apply (*smt tseq-def tfun-test power-zero-id id-def*)
  apply (*metis tseq-def tfun-test power-succ-unfold-ext*)
  done

lemma *tseq-test-seq*: *p* = −−*p* ∧ *q* = −−*q* ∧ *r* = −−*r* ⟶ *test-seq* (*tseq p x q r*)
  by (*metis test-seq-def tseq-test*)

lemma *tseq-pre-expression*: *x* ∈ *While-program* ∧ *p* ∈ *Pre-expression* ∧ *q* ∈ *Pre-expression* ∧ *r* ∈ *Pre-expression* ⟶ *tseq p x q r m* ∈ *Pre-expression*
  apply (*induct m*)
  apply (*smt tseq-def id-def power-zero-id*)
  apply (*smt tseq-def power-succ-unfold-ext tfun-pre-expression*)
  done

definition *tsum* :: ′*a* ⇒ ′*a* ⇒ ′*a* ⇒ ′*a* ⇒ ′*a*
  where *tsum p x q r* ≡ *Sum* (*tseq p x q r*)

lemma *tsum-test*: *p* = −−*p* ∧ *q* = −−*q* ∧ *r* = −−*r* ⟶ *tsum p x q r* = −−*tsum p x q r*
  by (*metis Sum-test tseq-test-seq tsum-def*)

lemma *t-fun-test*: *q* = −−*q* ⟶ *tfun* (−*p*) *x* (*p*⋆*x*«*q*) (−*p*+(*x*«(*p*⋆*x*«*q*);*aL*)) = −−*tfun* (−*p*) *x* (*p*⋆*x*«*q*) (−*p*+(*x*«(*p*⋆*x*«*q*);*aL*))
  by (*smt aL-test double-negation plus-closed pre-closed sub-mult-closed tfun-test*)

lemma *t-fun-pre-expression*: *x* ∈ *While-program* ∧ *p* ∈ *Test-expression* ∧ *q* ∈ *Pre-expression* ⟶ *tfun* (−*p*) *x* (*p*⋆*x*«*q*) (−*p*+(*x*«(*p*⋆*x*«*q*);*aL*)) ∈ *Pre-expression*
  by (*metis aL-pre-expression conj-pre disj-pre neg-pre pre-pre test-pre tfun-pre-expression while-prog*)

lemma *t-seq-test*: *q* = −−*q* ⟶ *tseq* (−*p*) *x* (*p*⋆*x*«*q*) (−*p*+(*x*«(*p*⋆*x*«*q*);*aL*)) *m* = −−*tseq* (−*p*) *x* (*p*⋆*x*«*q*) (−*p*+(*x*«(*p*⋆*x*«*q*);*aL*)) *m*
  by (*smt aL-test double-negation plus-closed pre-closed sub-mult-closed tseq-test*)

lemma *t-seq-test-seq*: *q* = −−*q* ⟶ *test-seq* (*tseq* (−*p*) *x* (*p*⋆*x*«*q*) (−*p*+(*x*«(*p*⋆*x*«*q*);*aL*)))
  by (*smt aL-test double-negation plus-closed pre-closed sub-mult-closed tseq-test-seq*)

lemma *t-seq-pre-expression*: *x* ∈ *While-program* ∧ *p* ∈ *Test-expression* ∧ *q* ∈ *Pre-expression* ⟶ *tseq* (−*p*) *x* (*p*⋆*x*«*q*) (−*p*+(*x*«(*p*⋆*x*«*q*);*aL*)) *m* ∈ *Pre-expression*
  by (*metis aL-pre-expression conj-pre disj-pre neg-pre pre-pre test-pre tseq-pre-expression while-prog*)

lemma *t-sum-test*: *q* = −−*q* ⟶ *tsum* (−*p*) *x* (*p*⋆*x*«*q*) (−*p*+(*x*«(*p*⋆*x*«*q*);*aL*)) = −−*tsum* (−*p*) *x* (*p*⋆*x*«*q*) (−*p*+(*x*«(*p*⋆*x*«*q*);*aL*))
  by (*smt aL-test double-negation plus-closed pre-closed sub-mult-closed tsum-test*)

definition *tfun2* :: ′*a* ⇒ ′*a* ⇒ ′*a* ⇒ ′*a* ⇒ ′*a* ⇒ ′*a*
  where *tfun2 p q x r s* ≡ *p* + *q*;(*x*«*r*;*s*)

lemma *tfun2-test*: *p* = −−*p* ∧ *q* = −−*q* ∧ *r* = −−*r* ∧ *s* = −−*s* ⟶ *tfun2 p q x r s* = −−*tfun2 p q x r s*
  by (*smt tfun2-def sub-mult-closed pre-closed plus-closed*)

lemma *tfun2-pre-expression*: *x* ∈ *While-program* ∧ *p* ∈ *Pre-expression* ∧ *q* ∈ *Pre-expression* ∧ *r* ∈ *Pre-expression* ∧ *s* ∈ *Pre-expression* ⟶ *tfun2 p q x r s* ∈ *Pre-expression*
  by (*metis tfun2-def conj-pre disj-pre pre-pre*)

lemma *tfun2-iso*: *p* = −−*p* ∧ *q* = −−*q* ∧ *r* = −−*r* ∧ *s1* = −−*s1* ∧ *s2* = −−*s2* ∧ *s1* ≤ *s2* ⟶ *tfun2 p q x r s1* ≤ *tfun2 p q x r s2*
  by (*smt tfun2-def mult-iso-right pre-iso sub-mult-closed plus-iso-right pre-closed*)

definition *tseq2* :: ′*a* ⇒ ′*a* ⇒ ′*a* ⇒ ′*a* ⇒ ′*a* ⇒ *nat* ⇒ ′*a*
  where *tseq2 p q x r s m* ≡ (*tfun2 p q x r* ˆ *m*) *s*

lemma *tseq2-test*: *p* = −−*p* ∧ *q* = −−*q* ∧ *r* = −−*r* ∧ *s* = −−*s* ⟶ *tseq2 p q x r s m* = −−*tseq2 p q x r s m*
  apply (*induct m*)
  apply (*smt tseq2-def power-zero-id id-def*)
  apply (*smt tseq2-def tfun2-test power-succ-unfold-ext*)
  done

lemma *tseq2-test-seq*: *p* = −−*p* ∧ *q* = −−*q* ∧ *r* = −−*r* ∧ *s* = −−*s* ⟶ *test-seq* (*tseq2 p q x r s*)
  by (*metis test-seq-def tseq2-test*)

```
lemma tseq2-pre-expression: x ∈ While-program ∧ p ∈ Pre-expression ∧ q ∈ Pre-expression ∧ r ∈ Pre-expression ∧ s ∈
Pre-expression ⟶ tseq2 p q x r s m ∈ Pre-expression
  apply (induct m)
  apply (smt tseq2-def id-def power-zero-id)
  apply (smt tseq2-def power-succ-unfold-ext tfun2-pre-expression)
  done

definition tsum2 :: 'a ⇒ 'a ⇒ 'a ⇒ 'a ⇒ 'a ⇒ 'a
  where tsum2 p q x r s ≡ Sum (tseq2 p q x r s)

lemma tsum2-test: p = −−p ∧ q = −−q ∧ r = −−r ∧ s = −−s ⟶ tsum2 p q x r s = −−tsum2 p q x r s
  by (metis Sum-test tseq2-test-seq tsum2-def)

lemma t-fun2-test: p = −−p ∧ q = −−q ⟶ tfun2 (−p;q) p x (p⋆x«q) (−p;q+p;(x«(p⋆x«q);aL)) = −−tfun2 (−p;q) p x
(p⋆x«q) (−p;q+p;(x«(p⋆x«q);aL))
  by (smt aL-test double-negation plus-closed pre-closed sub-mult-closed tfun2-test)

lemma t-fun2-pre-expression: x ∈ While-program ∧ p ∈ Test-expression ∧ q ∈ Pre-expression ⟶ tfun2 (−p;q) p x (p⋆x«q)
(−p;q+p;(x«(p⋆x«q);aL)) ∈ Pre-expression
  by (metis aL-pre-expression conj-pre disj-pre neg-pre pre-pre test-pre tfun2-pre-expression while-prog)

lemma t-seq2-test: p = −−p ∧ q = −−q ⟶ tseq2 (−p;q) p x (p⋆x«q) (−p;q+p;(x«(p⋆x«q);aL)) m = −−tseq2 (−p;q) p x
(p⋆x«q) (−p;q+p;(x«(p⋆x«q);aL)) m
  by (smt aL-test double-negation plus-closed pre-closed sub-mult-closed tseq2-test)

lemma t-seq2-test-seq: p = −−p ∧ q = −−q ⟶ test-seq (tseq2 (−p;q) p x (p⋆x«q) (−p;q+p;(x«(p⋆x«q);aL)))
  by (smt aL-test double-negation plus-closed pre-closed sub-mult-closed tseq2-test-seq)

lemma t-seq2-pre-expression: x ∈ While-program ∧ p ∈ Test-expression ∧ q ∈ Pre-expression ⟶ tseq2 (−p;q) p x (p⋆x«q)
(−p;q+p;(x«(p⋆x«q);aL)) m ∈ Pre-expression
  by (metis aL-pre-expression conj-pre disj-pre neg-pre pre-pre test-pre tseq2-pre-expression while-prog)

lemma t-sum2-test: p = −−p ∧ q = −−q ⟶ tsum2 (−p;q) p x (p⋆x«q) (−p;q+p;(x«(p⋆x«q);aL)) = −−tsum2 (−p;q) p x
(p⋆x«q) (−p;q+p;(x«(p⋆x«q);aL))
  by (smt aL-test double-negation plus-closed pre-closed sub-mult-closed tsum2-test)

lemma t-seq2-below-t-seq: p ∈ Test-expression ∧ q ∈ Pre-expression ∧ x ∈ While-program ⟶ tseq2 (−p;q) p x (p⋆x«q)
(−p;q+p;(x«(p⋆x«q);aL)) m ≤ tseq (−p) x (p⋆x«q) (−p+(x«(p⋆x«q);aL)) m
proof
  let ?t2 = tseq2 (−p;q) p x (p⋆x«q) (−p;q+p;(x«(p⋆x«q);aL))
  let ?t = tseq (−p) x (p⋆x«q) (−p+(x«(p⋆x«q);aL))
  assume 1: p ∈ Test-expression ∧ q ∈ Pre-expression ∧ x ∈ While-program
  show ?t2 m ≤ ?t m
  proof (induct m)
    show ?t2 0 ≤ ?t 0 using 1
     by (smt aL-test id-def lower-bound-left lower-bound-right plus-iso power-zero-id pre-closed pre-expression-test sub-mult-closed
test-pre tseq2-def tseq-def)
  next
    fix m
    assume ?t2 m ≤ ?t m
    hence 2: ?t2 (Suc m) ≤ tfun2 (− p ; q) p x (p ⋆ x « q) (?t m) using 1
     by (smt power-succ-unfold-ext pre-closed pre-expression-test sub-mult-closed t-seq2-test t-seq-test test-pre tfun2-iso tseq2-def)
    have ... ≤ ?t (Suc m) using 1
        by (smt lower-bound-left lower-bound-right plus-iso power-succ-unfold-ext pre-closed pre-expression-test sub-mult-closed
t-seq-test test-pre tfun2-def tfun-def tseq-def)
    thus ?t2 (Suc m) ≤ ?t (Suc m) using 1 2
      by (smt pre-closed pre-expression-test sub-mult-closed t-seq2-test t-seq-test test-pre tfun2-test transitive)
  qed
qed

lemma t-seq2-below-t-sum: p ∈ Test-expression ∧ q ∈ Pre-expression ∧ x ∈ While-program ⟶ tseq2 (−p;q) p x (p⋆x«q)
(−p;q+p;(x«(p⋆x«q);aL)) m ≤ tsum (−p) x (p⋆x«q) (−p+(x«(p⋆x«q);aL))
   by (smt Sum-upper pre-expression-test t-seq2-below-t-seq t-seq2-test t-seq-test t-sum-test test-pre test-seq-def transitive
tsum-def)

lemma t-sum2-below-t-sum: p ∈ Test-expression ∧ q ∈ Pre-expression ∧ x ∈ While-program ⟶ tsum2 (−p;q) p x (p⋆x«q)
(−p;q+p;(x«(p⋆x«q);aL)) ≤ tsum (−p) x (p⋆x«q) (−p+(x«(p⋆x«q);aL))
  by (smt Sum-least pre-expression-test t-seq2-below-t-sum t-seq2-test t-sum-test test-pre test-seq-def tsum2-def)
```

```
lemma t-seq2-below-w: p ∈ Test-expression ∧ q ∈ Pre-expression ∧ x ∈ While-program ⟶ tseq2 (−p;q) p x (p⋆x«q)
(−p;q+p;(x«(p⋆x«q);aL)) m ≤ p⋆x«q
  apply (cases m)
  apply (smt aL-test id-def lower-bound-left mult-iso-right plus-comm plus-iso-right power-zero-id pre-closed pre-expression-test
pre-iso sub-mult-closed test-pre tseq2-def while-pre)
  apply simp
  unfolding tseq2-def power-succ-unfold-ext
   apply (smt lower-bound-left mult-iso-right plus-comm plus-iso-right pre-closed pre-expression-test pre-iso sub-mult-closed
t-seq2-test test-pre tseq2-def while-pre tfun2-def)
  done

lemma t-sum2-below-w: p ∈ Test-expression ∧ q ∈ Pre-expression ∧ x ∈ While-program ⟶ tsum2 (−p;q) p x (p⋆x«q)
(−p;q+p;(x«(p⋆x«q);aL)) ≤ p⋆x«q
  by (smt Sum-least pre-closed pre-expression-test t-seq2-below-w t-seq2-test-seq test-pre tsum2-def)

lemma t-sum2-w: aL = 1 ∧ p ∈ Test-expression ∧ q ∈ Pre-expression ∧ x ∈ While-program ⟶ tsum2 (−p;q) p x (p⋆x«q)
(−p;q+p;(x«(p⋆x«q);aL)) = p⋆x«q
proof
  let ?w = p⋆x«q
  let ?s = −p;q+p;(x«?w;aL)
  assume 1: aL = 1 ∧ p ∈ Test-expression ∧ q ∈ Pre-expression ∧ x ∈ While-program
  have ?w = tseq2 (−p;q) p x ?w ?s 0 using 1
    by (smt bs-mult-right-one id-def plus-comm power-zero-id pre-closed pre-expression-test sub-mult-closed test-expression-test
tseq2-def while-pre)
  hence ?w ≤ tsum2 (−p;q) p x ?w ?s using 1
    by (smt Sum-upper pre-expression-test t-seq2-test-seq test-pre tsum2-def)
  thus tsum2 (−p;q) p x ?w ?s = ?w using 1
    by (smt antisymmetric pre-closed pre-expression-test t-sum2-test t-sum2-below-w test-pre)
qed

inductive derived-hoare-triple :: 'a ⇒ 'a ⇒ 'a ⇒ bool (- ⦇ - ⦈ - [54,54,54] 53)
  where atom-trip: p ∈ Pre-expression ∧ x ∈ Atomic-program ⟹ x«p⦇x⦈p
      | seq-trip:  p⦇x⦈q ∧ q⦇y⦈r ⟹ p⦇x;y⦈r
      | cond-trip:  p ∈ Test-expression ∧ q ∈ Pre-expression ∧ p;q⦇x⦈r ∧ −p;q⦇y⦈r ⟹ q⦇x◁p▷y⦈r
     | while-trip: p ∈ Test-expression ∧ q ∈ Pre-expression ∧ test-seq t ∧ q ≤ Sum t ∧ t 0;p;q⦇x⦈aL;q ∧ (∀n>0 . t n;p;q⦇x⦈pSum
t n;q) ⟹ q⦇p⋆x⦈−p;q
      | cons-trip:  p ∈ Pre-expression ∧ s ∈ Pre-expression ∧ p ≤ q ∧ q⦇x⦈r ∧ r ≤ s ⟹ p⦇x⦈s

lemma derived-type: p⦇x⦈q ⟹ p ∈ Pre-expression ∧ q ∈ Pre-expression ∧ x ∈ While-program
  apply (induct rule: derived-hoare-triple.induct)
  apply (metis atom-prog pre-pre)
  apply (metis seq-prog)
  apply (metis cond-prog)
  apply (metis conj-pre neg-pre test-pre while-prog)
  apply metis
  done

lemma cons-pre-trip: p ∈ Pre-expression ∧ q⦇y⦈r ⟶ p;q⦇y⦈r
  by (smt conj-pre cons-trip derived-type lower-bound-right reflexive pre-expression-test)

lemma cons-post-trip: q ∈ Pre-expression ∧ r ∈ Pre-expression ∧ p⦇y⦈q;r ⟶ p⦇y⦈r
  by (smt cons-trip derived-type lower-bound-right reflexive pre-expression-test)

definition valid-hoare-triple :: 'a ⇒ 'a ⇒ 'a ⇒ bool (- ⟨ - ⟩ - [54,54,54] 53)
  where p⟨x⟩q ≡ (p ∈ Pre-expression ∧ q ∈ Pre-expression ∧ x ∈ While-program ∧ p ≤ x«q)

end

class hoare-calculus-sound = hoare-calculus +
  assumes while-soundness: −p;−q ≤ x«−q ⟶ aL;−q ≤ −p⋆x«−q

begin

lemma while-soundness-0: −p;−q ≤ x«−q ⟶ −q;aL ≤ −p⋆x«−−p;−q
  by (smt while-soundness aL-test sub-comm while-post)

lemma while-soundness-1: test-seq t ∧ −q ≤ Sum t ∧ t 0;−p;−q ≤ x«aL;−q ∧ (∀n>0 . t n;−p;−q ≤ x«pSum t n;−q) ⟶
−q ≤ −p⋆x«−−p;−q
proof
```

```
  assume 1: test-seq t ∧ −q ≤ Sum t ∧ t 0;−p;−q ≤ x«aL;−q ∧ (∀ n>0 . t n;−p;−q ≤ x«pSum t n;−q)
  hence ∀ n . t n;−p;−q ≤ x«−q
    by (smt aL-test pSum-test pre-closed pre-lower-bound-right sub-mult-closed test-seq-def transitive)
  hence 2: −p;−q ≤ x«−q using 1
    by (smt Sum-test leq-def mult-right-dist-Sum pre-closed sub-assoc sub-comm sub-mult-closed test-seq-def)
  have ∀ n . t n;−q ≤ −p⋆x«−−p;−q ∧ pSum t n;−q ≤ −p⋆x«−−p;−q
  proof
    fix n
    show t n;−q ≤ −p⋆x«−−p;−q ∧ pSum t n;−q ≤ −p⋆x«−−p;−q
    proof (induct n rule: nat-less-induct)
      fix n
      assume ∀ m<n . t m;−q ≤ −p⋆x«−−p;−q ∧ pSum t m;−q ≤ −p⋆x«−−p;−q
      hence 3: pSum t n;−q ≤ −p⋆x«−−p;−q using 1
        apply (cases n)
        apply (smt bs-mult-left-zero pSum.simps(1) pre-closed sub-mult-closed zero-least-test)
        apply (smt least-upper-bound mult-distr-plus-right pSum.simps(2) pSum-test pre-closed sub-mult-closed test-seq-def)
        done
      hence x«pSum t n;−q ≤ x«−p⋆x«−−p;−q using 1
        by (smt pSum-test pre-closed pre-iso sub-mult-closed)
      hence 4: −p;(t n;−q) ≤ −p;(−p⋆x«−−p;−q) using 1 2
        apply (cases n)
              apply (smt aL-test leq-def mult-idempotent mult-iso-right pre-closed pre-lower-bound-left sub-assoc sub-comm
sub-mult-closed test-seq-def transitive while-pre-then while-soundness-0)
         apply (smt greatest-lower-bound lower-bound-left pSum-test pre-closed sub-assoc sub-comm sub-mult-closed test-seq-def
transitive while-pre-then)
        done
      have −−p;(t n;−q) ≤ −−p;(−p⋆x«−−p;−q) using 1
        by (smt leq-def lower-bound-right sub-assoc sub-comm sub-mult-closed test-seq-def while-pre-else)
      thus t n;−q ≤ −p⋆x«−−p;−q ∧ pSum t n;−q ≤ −p⋆x«−−p;−q using 1 3 4
        by (smt leq-cases-2 pre-closed sub-mult-closed test-seq-def)
    qed
  qed
  thus −q ≤ −p⋆x«−−p;−q using 1
    by (smt Sum-test leq-def mult-right-dist-Sum pre-closed sub-comm sub-mult-closed)
qed

lemma while-soundness-2: test-seq t ∧ −r ≤ Sum t ∧ (∀ n . t n;−p ≤ x«pSum t n) ⟶ −r ≤ −p⋆x«1
proof
  assume 1: test-seq t ∧ −r ≤ Sum t ∧ (∀ n . t n;−p ≤ x«pSum t n)
  hence 2: ∀ n>0 . t n;−p;Sum t ≤ x«pSum t n;Sum t
    by (smt Sum-test leq-def lower-bound-left pSum-below-Sum pSum-test pre-closed sub-mult-closed test-seq-def transitive)
  have 3: t 0;−p;Sum t ≤ x«0 using 1
    by (smt Sum-test Sum-upper leq-def sub-assoc sub-comm test-seq-def pSum.simps(1))
  have x«0 ≤ x«aL;Sum t using 1
    by (smt Sum-test aL-test pre-iso sub-mult-closed zero-double-compl zero-least-test)
  hence t 0;−p;Sum t ≤ x«aL;Sum t using 1 3
    by (smt Sum-test aL-test pre-closed sub-mult-closed test-seq-def transitive zero-double-compl)
  hence Sum t ≤ −p⋆x«−−p;Sum t using 1 2
    by (smt Sum-test reflexive while-soundness-1)
  thus −r ≤ −p⋆x«1 using 1
    by (smt Sum-test one-def pre-below-pre-one pre-closed sub-mult-closed transitive)
qed

theorem soundness: p⦇x⦈q ⟹ p⟨x⟩q
  apply (induct rule: derived-hoare-triple.induct)
  apply (metis atom-prog pre-pre valid-hoare-triple-def pre-closed reflexive pre-expression-test)
  apply (metis valid-hoare-triple-def pre-expression-test pre-compose seq-prog)
  apply (metis valid-hoare-triple-def ite-import-mult pre-expression-test cond-prog test-pre)
  apply (smt valid-hoare-triple-def pre-expression-test conj-pre neg-pre test-pre while-prog while-soundness-1)
  apply (metis pre-expression-test pre-iso pre-pre transitive valid-hoare-triple-def)
  done

end

class hoare-calculus-pre-complete = hoare-calculus +
  assumes aL-pre-import: (x«−q);aL ≤ x«−q;aL
  assumes pre-right-dist-Sum: x ∈ While-program ∧ ascending-chain t ∧ test-seq t ⟶ x«Sum t = Sum (λn . x«t n)

begin
```

lemma *aL-pre-import-equal*: $(x«-q);aL = (x«-q;aL);aL$
proof $-$
have $(x«-q);aL \leq (x«-q;aL);aL$
by (*smt aL-pre-import aL-test greatest-lower-bound lower-bound-right pre-closed sub-mult-closed*)
thus *?thesis*
by (*smt aL-test antisymmetric lower-bound-left mult-iso-left pre-closed pre-iso sub-mult-closed*)
qed

lemma *aL-pre-below-t-seq2*: $p \in$ *Test-expression* $\wedge$ $q \in$ *Pre-expression* $\wedge$ $x \in$ *While-program* $\longrightarrow (p\star x«q);aL \leq$ *tseq2* $(-p;q)$ p x $(p\star x«q)$ $(-p;q+p;(x«(p\star x«q);aL))$ 0
by (*smt aL-pre-import aL-test id-def lower-bound-left mult-distr-plus-right mult-iso-right plus-comm plus-iso power-zero-id pre-closed pre-expression-test sub-assoc sub-mult-closed test-pre tseq2-def while-pre*)

lemma *t-seq2-ascending*: $p \in$ *Test-expression* $\wedge$ $q \in$ *Pre-expression* $\wedge$ $x \in$ *While-program* $\longrightarrow$ *tseq2* $(-p;q)$ p x $(p\star x«q)$ $(-p;q+p;(x«(p\star x«q);aL))$ $m \leq$ *tseq2* $(-p;q)$ p x $(p\star x«q)$ $(-p;q+p;(x«(p\star x«q);aL))$ (*Suc m*)
apply (*induct m*)
apply (*smt aL-pre-below-t-seq2 aL-test greatest-lower-bound id-def lower-bound-left mult-iso-right plus-closed plus-iso-right power-succ-unfold-ext power-zero-id pre-closed pre-expression-test pre-iso sub-mult-closed test-pre tfun2-def tseq2-def*)
apply (*smt mult-iso-right plus-iso-right power-succ-unfold-ext pre-closed pre-expression-test pre-iso sub-mult-closed t-seq2-test test-pre tfun2-def tseq2-def*)
done

lemma *t-seq2-ascending-chain*: $p \in$ *Test-expression* $\wedge$ $q \in$ *Pre-expression* $\wedge$ $x \in$ *While-program* $\longrightarrow$ *ascending-chain* (*tseq2* $(-p;q)$ p x $(p\star x«q)$ $(-p;q+p;(x«(p\star x«q);aL)))$
by (*metis t-seq2-ascending ascending-chain-def*)

end

class *hoare-calculus-complete* = *hoare-calculus-pre-complete* +
assumes *while-completeness*: $-p;(x«-q) \leq -q \longrightarrow -p\star x«-q \leq -q+aL$

begin

lemma *while-completeness-var*: $-p;(x«-q)+-r \leq -q \longrightarrow -p\star x«-r \leq -q+aL$
proof
assume *1*: $-p;(x«-q)+-r \leq -q$
hence *2*: $-p\star x«-q \leq -q+aL$
by (*smt least-upper-bound pre-closed sub-mult-closed while-completeness*)
have $-p\star x«-r \leq -p\star x«-q$ **using** *1*
by (*smt least-upper-bound pre-closed pre-iso sub-mult-closed*)
thus $-p\star x«-r \leq -q+aL$ **using** *2*
by (*smt transitive pre-closed aL-test plus-closed*)
qed

lemma *while-completeness-sum*: $p \in$ *Test-expression* $\wedge$ $q \in$ *Pre-expression* $\wedge$ $x \in$ *While-program* $\longrightarrow p\star x«q \leq$ *tsum* $(-p)$ x $(p\star x«q)$ $(-p+(x«(p\star x«q);aL))$
proof
let $?w = p\star x«q$
let $?r = -p;q+p;(x«?w;aL)$
let $?t =$ *tseq2* $(-p;q)$ p x $?w$ $?r$
let $?ts =$ *tsum2* $(-p;q)$ p x $?w$ $?r$
assume *1*: $p \in$ *Test-expression* $\wedge$ $q \in$ *Pre-expression* $\wedge$ $x \in$ *While-program*
hence *2*: $?w = --?w$
by (*metis pre-expression-test pre-closed*)
have *3*: $?r = --?r$ **using** *1*
by (*smt aL-test sub-mult-closed pre-closed plus-closed pre-expression-test test-pre*)
have *4*: $?ts = --?ts$ **using** *1*
by (*metis t-sum2-test pre-expression-test test-pre*)
have *5*: *test-seq* $?t$ **using** *1*
by (*metis pre-expression-test t-seq2-test-seq test-expression-test*)
have $-p;q \leq ?r$ **using** *1*
by (*smt aL-test pre-closed pre-expression-test sub-mult-closed test-pre upper-bound-left*)
hence *6*: $-p;q \leq ?ts$ **using** *1 2 3 4*
by (*smt Sum-upper id-def transitive power-zero-id pre-expression-test sub-mult-closed test-pre tseq2-def tseq2-test-seq tsum2-def*)
have $\forall n\ .\ p;(x«?t\ n) \leq ?ts$ **using** *1 4 5*
by (*smt Sum-upper leq-def power-succ-unfold-ext pre-closed pre-expression-test sub-comm sub-mult-closed t-seq2-below-w test-pre test-seq-def tfun2-def transitive tseq2-def tsum2-def upper-bound-right*)
hence $p;(x«?ts) \leq ?ts$ **using** *1 4 5*

```
    by (smt mult-left-dist-Sum pre-closed pre-right-dist-Sum t-seq2-ascending-chain test-expression-test test-seq-def tsum2-def)
  hence p;(x«?ts)+−p;q ≤ ?ts using 1 4 6
    by (smt least-upper-bound pre-closed pre-expression-test sub-mult-closed test-pre)
  hence ?w ≤ ?ts+aL using 1 2 4
    by (smt pre-expression-test while-post sub-mult-closed t-sum2-below-t-sum t-sum-test test-pre transitive while-completeness-var)
  hence ?w ≤ ?ts using 1 2 3 4
    by (smt Sum-upper aL-pre-below-t-seq2 aL-test id-def leq-def leq-plus lower-bound-right mult-distr-plus-left plus-closed plus-comm plus-iso-left power-zero-id pre-expression-test sub-mult-closed test-pre transitive tseq2-def tseq2-test-seq tsum2-def)
  thus ?w ≤ tsum (−p) x ?w (−p+(x«?w;aL)) using 1 2 4
    by (smt pre-expression-test t-sum2-below-t-sum t-sum-test transitive)
qed

lemma while-complete: p ∈ Test-expression ∧ q ∈ Pre-expression ∧ x ∈ While-program ∧ (∀ r∈Pre-expression . x«r⦇x⦈r) ⟶ p⋆x«q⦇p⋆x⦈q
proof
  let ?w = p⋆x«q
  let ?t = tseq (−p) x ?w (−p+(x«?w;aL))
  assume 1: p ∈ Test-expression ∧ q ∈ Pre-expression ∧ x ∈ While-program ∧ (∀ r∈Pre-expression . x«r⦇x⦈r)
  hence 2: ?w ∈ Pre-expression
    by (metis pre-pre while-prog)
  have 3: test-seq ?t using 1
    by (metis t-seq-test-seq pre-expression-test)
  hence 4: ?w ≤ Sum ?t using 1
    by (metis tsum-def while-completeness-sum)
  have 5: ?t 0;p;?w⦇x⦈aL;?w using 1 2
    by (smt aL-pre-expression conj-pre cons-pre-trip id-def mult-compl-intro plus-closed power-zero-id pre-closed pre-expression-test sub-comm sub-mult-closed test-pre tseq-def)
  have ∀ n>0 . ?t n;p;?w⦇x⦈pSum ?t n;?w
  proof (rule, rule)
    fix n
    assume 0<(n::nat) then obtain m where 6: n = Suc m
      by (auto dest: less-imp-Suc-add)
    hence ?t m;?w ≤ pSum ?t n;?w using 2 3
      by (metis (lifting, full-types) mult-iso-left pSum.simps(2) pSum-test pre-expression-test test-seq-def upper-bound-right)
    thus ?t n;p;?w⦇x⦈pSum ?t n;?w using 1 2 6
      by (smt conj-pre cons-trip lower-bound-left mult-compl-intro pSum-pre-expression power-succ-unfold-ext pre-closed pre-expression-test sub-assoc sub-comm t-seq-pre-expression test-pre tfun-def tseq-def)
  qed
  hence ?w⦇p⋆x⦈−p;?w using 1 2 3 4 5
    by (smt while-trip)
  thus ?w⦇p⋆x⦈q using 1
    by (smt cons-post-trip neg-pre pre-expression-test test-pre while-pre-else)
qed

lemma pre-completeness: x ∈ While-program ⟹ q ∈ Pre-expression ⟹ x«q⦇x⦈q
  apply (induct arbitrary: q rule: While-program.induct)
  apply (metis atom-trip)
  apply (metis pre-pre pre-seq seq-trip pre-expression-test)
  apply (smt cond-prog cond-trip cons-pre-trip ite-pre-else ite-pre-then neg-pre pre-pre pre-expression-test test-pre)
  apply (metis while-complete)
  done

theorem completeness: p⟨x⟩q ⟶ p⦇x⦈q
  by (metis valid-hoare-triple-def pre-completeness reflexive pre-expression-test cons-trip)

end

class hoare-calculus-sound-complete = hoare-calculus-sound + hoare-calculus-complete

begin

— Theorem 41

theorem soundness-completeness: p⦇x⦈q ⟷ p⟨x⟩q
  by (smt soundness completeness)

end
```

class *hoare-rules* = *whiledo* + *complete-tests* + *hoare-triple* +
  assumes *rule-pre*: $x \ll\!- q\{\!|x|\!\}-q$
  assumes *rule-seq*: $-p\{\!|x|\!\}-q \wedge -q\{\!|y|\!\}-r \longrightarrow -p\{\!|x;y|\!\}-r$
  assumes *rule-cond*: $-p;-q\{\!|x|\!\}-r \wedge --p;-q\{\!|y|\!\}-r \longrightarrow -q\{\!|x \lhd -p \rhd y|\!\}-r$
  assumes *rule-while*: $\mathit{test\text{-}seq}\ t \wedge -q \le \mathit{Sum}\ t \wedge t\ 0;-p;-q\{\!|x|\!\}\mathit{aL};-q \wedge (\forall n{>}0\ .\ t\ n;-p;-q\{\!|x|\!\}\mathit{pSum}\ t\ n;-q) \longrightarrow -q\{\!|-p\star x|\!\}--p;-q$
  assumes *rule-cons*: $-p \le -q \wedge -q\{\!|x|\!\}-r \wedge -r \le -s \longrightarrow -p\{\!|x|\!\}-s$
  assumes *rule-disj*: $-p\{\!|x|\!\}-r \wedge -q\{\!|x|\!\}-s \longrightarrow -p+-q\{\!|x|\!\}-r+-s$

begin

lemma *rule-cons-pre*: $-p \le -q \wedge -q\{\!|x|\!\}-r \longrightarrow -p\{\!|x|\!\}-r$
  by (*metis rule-cons reflexive*)

lemma *rule-cons-pre-mult*: $-q\{\!|x|\!\}-r \longrightarrow -p;-q\{\!|x|\!\}-r$
  by (*metis rule-cons-pre lower-bound-left sub-comm sub-mult-closed*)

lemma *rule-cons-pre-plus*: $-p+-q\{\!|x|\!\}-r \longrightarrow -p\{\!|x|\!\}-r$
  by (*metis rule-cons-pre upper-bound-left plus-closed*)

lemma *rule-cons-post*: $-q\{\!|x|\!\}-r \wedge -r \le -s \longrightarrow -q\{\!|x|\!\}-s$
  by (*metis rule-cons reflexive*)

lemma *rule-cons-post-mult*: $-q\{\!|x|\!\}-r;-s \longrightarrow -q\{\!|x|\!\}-s$
  by (*metis rule-cons-post lower-bound-left sub-comm sub-mult-closed*)

lemma *rule-cons-post-plus*: $-q\{\!|x|\!\}-r \longrightarrow -q\{\!|x|\!\}-r+-s$
  by (*metis rule-cons-post upper-bound-left plus-closed*)

lemma *rule-disj-pre*: $-p\{\!|x|\!\}-r \wedge -q\{\!|x|\!\}-r \longrightarrow -p+-q\{\!|x|\!\}-r$
  by (*metis rule-disj plus-idempotent*)

end

class *hoare-calculus-valid* = *hoare-calculus-sound-complete* + *hoare-triple* +
  assumes *hoare-triple-valid*: $-p\{\!|x|\!\}-q \longleftrightarrow -p \le x \ll\!- q$

begin

lemma *valid-hoare-triple-same*: $p \in \mathit{Pre\text{-}expression} \wedge q \in \mathit{Pre\text{-}expression} \wedge x \in \mathit{While\text{-}program} \longrightarrow p\{\!|x|\!\}q = p\langle x\rangle q$
  by (*metis valid-hoare-triple-def hoare-triple-valid pre-expression-test*)

lemma *derived-hoare-triple-same*: $p \in \mathit{Pre\text{-}expression} \wedge q \in \mathit{Pre\text{-}expression} \wedge x \in \mathit{While\text{-}program} \longrightarrow p\{\!|x|\!\}q = p(\!|x|\!)q$
  by (*metis valid-hoare-triple-same soundness-completeness*)

lemma *valid-rule-disj*: $-p\{\!|x|\!\}-r \wedge -q\{\!|x|\!\}-s \longrightarrow -p+-q\{\!|x|\!\}-r+-s$
proof
  assume *1*: $-p\{\!|x|\!\}-r \wedge -q\{\!|x|\!\}-s$
  have $x \ll\!- r \le x \ll\!- r+-s \wedge x \ll\!- s \le x \ll\!- r+-s$
    by (*smt plus-closed pre-iso upper-bound-left upper-bound-right*)
  thus $-p+-q\{\!|x|\!\}-r+-s$ **using** *1*
    by (*smt hoare-triple-valid least-upper-bound plus-closed pre-closed transitive*)
qed

subclass *hoare-rules*
  apply *unfold-locales*
  apply (*smt hoare-triple-valid pre-closed reflexive*)
  apply (*smt hoare-triple-valid pre-compose*)
  apply (*smt hoare-triple-valid ite-import-mult sub-mult-closed*)
  apply (*smt hoare-triple-valid aL-test pSum-test plus-closed sub-mult-closed test-seq-def while-soundness-1*)
  apply (*smt hoare-triple-valid pre-iso transitive pre-closed*)
  apply (*smt valid-rule-disj*)
  done

lemma *nat-test-rule-while*: $\mathit{nat\text{-}test}\ t\ s \wedge -q \le s \wedge (\forall n\ .\ t\ n;-p;-q\{\!|x|\!\}\mathit{pSum}\ t\ n;-q) \longrightarrow -q\{\!|-p\star x|\!\}--p;-q$
  by (*smt hoare-triple-valid nat-test-def nat-test-pre pSum-test-nat sub-mult-closed*)

lemma *test-seq-rule-while*: $\mathit{test\text{-}seq}\ t \wedge -q \le \mathit{Sum}\ t \wedge t\ 0;-p;-q\{\!|x|\!\}\mathit{aL};-q \wedge (\forall n{>}0\ .\ t\ n;-p;-q\{\!|x|\!\}\mathit{pSum}\ t\ n;-q) \longrightarrow -q\{\!|-p\star x|\!\}--p;-q$

```
  by (smt hoare-triple-valid aL-test pSum-test sub-mult-closed test-seq-def while-soundness-1)

lemma rule-zero: 0⦃x⦄−p
  by (metis hoare-triple-valid one-compl pre-closed zero-least-test)

lemma rule-skip: −p⦃1⦄−p
  by (metis pre-closed pre-one-increasing rule-cons-pre rule-pre)

lemma rule-example-4: test-seq t ∧ Sum t = 1 ∧ t 0;−p1;−p3 = 0 ∧ −p1⦃z1⦄−p1;−p2 ∧ (∀ n>0 . t
n;−p1;−p2;−p3⦃z2⦄pSum t n;−p1;−p2) ⟶ −p1⦃z1;(−p3⋆z2)⦄−p2;−−p3
proof
   assume 1: test-seq t ∧ Sum t = 1 ∧ t 0;−p1;−p3 = 0 ∧ −p1⦃z1⦄−p1;−p2 ∧ (∀ n>0 . t n;−p1;−p2;−p3⦃z2⦄pSum t
n;−p1;−p2)
  hence 2: t 0;−p3;(−p1;−p2)⦃z2⦄aL;(−p1;−p2)
    by (smt aL-test bs-mult-left-zero rule-zero sub-assoc sub-comm sub-mult-closed test-seq-def)
  have ∀ n>0 . t n;−p3;(−p1;−p2)⦃z2⦄pSum t n;(−p1;−p2) using 1
    by (smt lower-bound-left pSum-test rule-cons-pre sub-assoc sub-comm sub-mult-closed test-seq-def)
  hence −p1;−p2⦃−p3⋆z2⦄−−p3;(−p1;−p2) using 1 2
    by (smt one-greatest rule-while sub-mult-closed)
  thus −p1⦃z1;(−p3⋆z2)⦄−p2;−−p3 using 1
    by (smt lower-bound-left rule-cons-post rule-seq sub-assoc sub-comm sub-mult-closed)
qed

end

class hoare-calculus-pc-2 = hoare-calculus-sound + hoare-calculus-pre-complete +
  assumes aL-one: aL = 1

begin

subclass hoare-calculus-sound-complete
  apply unfold-locales
  apply (smt aL-one plus-right-one one-greatest pre-closed)
  done

lemma while-soundness-pc: −p;−q ≤ x«−q ⟶ −q ≤ −p⋆x«−−p;−q
proof
  assume 1: −p;−q ≤ x«−q
  let ?t = λx . 1
  have 2: test-seq ?t
    by (metis test-seq-def one-double-compl)
  hence 3: −q ≤ Sum ?t
    by (metis Sum-test Sum-upper antisymmetric one-double-compl one-greatest)
  have 4: ?t 0;−p;−q ≤ x«aL;−q using 1 2
    by (metis aL-one bs-mult-left-one)
  have ∀ n>0 . ?t n;−p;−q ≤ x«pSum ?t n;−q using 1 2
    by (metis bs-mult-left-one gr0-implies-Suc pSum.simps(2) pSum-test plus-right-one)
  thus −q ≤ −p⋆x«−−p;−q using 2 3 4
    by (smt while-soundness-1)
qed

end

class hoare-calculus-pc = hoare-calculus-sound + hoare-calculus-pre-complete +
  assumes pre-one-one: x«1 = 1

begin

subclass hoare-calculus-pc-2
  apply unfold-locales
  apply (metis aL-def pre-one-one)
  done

end

class hoare-calculus-pc-valid = hoare-calculus-pc + hoare-calculus-valid

begin
```

```
lemma rule-while-pc: −p;−q⦃x⦄−q ⟶ −q⦃−p⋆x⦄−−p;−q
  by (metis hoare-triple-valid sub-mult-closed while-soundness-pc)

lemma rule-alternation: −p⦃x⦄−q ∧ −q⦃y⦄−p ⟶ −p⦃−r⋆x;y⦄−−r;−p
  by (metis rule-seq rule-cons-pre-mult rule-while-pc)

lemma rule-alternation-context: −p⦃v⦄−p ∧ −p⦃w⦄−q ∧ −q⦃x⦄−q ∧ −q⦃y⦄−p ∧ −p⦃z⦄−p ⟶ −p⦃−r⋆v;w;x;y;z⦄−−r;−p
  by (metis rule-seq rule-cons-pre-mult rule-while-pc)

lemma rule-example-3: −p;−q⦃x⦄−−p;−q ∧ −−p;−r⦃x⦄−p;−r ∧ −p;−r⦃y⦄−p;−q ∧ −−p;−q⦃z⦄−−p;−r ⟶
−p;−q+−−p;−r⦃−s⋆x;(y◁−p▷z)⦄−−s;(−p;−q+−−p;−r)
proof (rule, (erule conjE)+)
  assume −p;−q⦃x⦄−−p;−q and −−p;−r⦃x⦄−p;−r
  hence t1: −p;−q+−−p;−r⦃x⦄−−p;−q+−p;−r
    by (smt rule-disj sub-mult-closed)
  assume −p;−r⦃y⦄−p;−q
  hence −p;−r⦃y⦄−p;−q+−−p;−r
    by (smt rule-cons-post-plus sub-mult-closed)
  hence t2: −p;(−−p;−q+−p;−r)⦃y⦄−p;−q+−−p;−r
    by (metis mult-compl mult-distr-plus-left mult-idempotent plus-left-zero sub-assoc sub-mult-closed)
  assume −−p;−q⦃z⦄−−p;−r
  hence −−p;−q⦃z⦄−p;−q+−−p;−r
    by (smt plus-comm rule-cons-post-plus sub-mult-closed)
  hence −−p;(−−p;−q+−p;−r)⦃z⦄−p;−q+−−p;−r
    by (metis mult-compl mult-distr-plus-left mult-idempotent plus-right-zero sub-assoc sub-mult-closed)
  hence −−p;−q+−p;−r⦃y◁−p▷z⦄−p;−q+−−p;−r using t2
    by (smt plus-closed rule-cond sub-mult-closed)
  hence −s;(−p;−q+−−p;−r)⦃x;(y◁−p▷z)⦄−p;−q+−−p;−r using t1
    by (smt plus-closed rule-cons-pre-mult rule-seq sub-mult-closed)
  thus −p;−q+−−p;−r⦃−s⋆x;(y◁−p▷z)⦄−−s;(−p;−q+−−p;−r)
    by (smt plus-closed rule-while-pc sub-mult-closed)
qed

end

class hoare-calculus-tc = hoare-calculus + precondition-test-test + precondition-distr-mult +
  assumes while-bnd: p ∈ Test-expression ∧ q ∈ Pre-expression ∧ x ∈ While-program ⟶ p⋆x«q ≤ Sum (λn . (p;x)ˆn«0)

begin

lemma p ∈ Test-expression ∧ q ∈ Pre-expression ∧ x ∈ While-program ⟶ p⋆x«q ≤ tsum (−p) x (p⋆x«q)
(−p+(x«(p⋆x«q);aL))
proof
  let ?w = p⋆x«q
  let ?s = −p+(x«?w;aL)
  let ?t = tseq (−p) x ?w ?s
  let ?b = λn . (p;x)ˆn«0
  assume 1: p ∈ Test-expression ∧ q ∈ Pre-expression ∧ x ∈ While-program
  hence 2: test-seq ?t
    by (metis t-seq-test-seq pre-expression-test)
  have 3: test-seq ?b
    by (smt zero-double-compl pre-closed test-seq-def)
  have 4: ?w = −−?w using 1
    by (metis pre-expression-test pre-closed)
  have ?w ≤ Sum ?b using 1
    by (metis while-bnd)
  hence 5: ?w = Sum ?b;?w using 3 4
    by (smt Sum-test leq-def sub-comm)
  have ∀ n . ?b n;?w ≤ ?t n
  proof
    fix n
    show ?b n;?w ≤ ?t n
    proof (induct n)
      show ?b 0;?w ≤ ?t 0 using 2 4
        by (smt bs-mult-left-zero power-0 pre-one test-seq-def zero-double-compl zero-least-test)
    next
      fix n
      assume 6: ?b n;?w ≤ ?t n
      have −p ≤ ?t (Suc n)
      apply (simp only: power-succ-unfold-ext tseq-def) using 1
        by (smt pre-expression-test t-seq-test pre-closed sub-mult-closed tfun-def tseq-def upper-bound-left)
      hence 7: −p;?b (Suc n);?w ≤ ?t (Suc n) using 2 3 4
        by (smt lower-bound-left sub-mult-closed test-seq-def transitive)
      have 8: p;?b (Suc n);?w ≤ x«?w;(?b n;?w) using 1
       by (smt lower-bound-right mult-idempotent power-Suc pre-closed pre-distr-mult pre-expression-test pre-import-composition sub-assoc sub-comm sub-mult-closed test-expression-test while-pre-then zero-double-compl)
      have 9: ... ≤ x«?w;?t n using 2 3 4 6
        by (smt mult-iso-right pre-iso sub-mult-closed test-seq-def)
      have ... ≤ ?t (Suc n) using 2 4
        by (smt power-succ-unfold-ext pre-closed sub-mult-closed test-seq-def tfun-def tseq-def upper-bound-right)
      hence p;?b (Suc n);?w ≤ ?t (Suc n) using 1 2 3 4 8 9
        by (smt pre-closed sub-mult-closed test-expression-test test-seq-def transitive)
      thus ?b (Suc n);?w ≤ ?t (Suc n) using 1 2 3 4 7
        by (smt leq-cases sub-assoc sub-mult-closed test-expression-test test-seq-def)
    qed
  qed
  hence Sum ?b;?w ≤ tsum (−p) x ?w ?s using 1 3 4
    by (smt Sum-upper mult-right-dist-Sum pre-expression-test sub-mult-closed t-seq-test t-sum-test test-seq-def transitive tsum-def)
  thus ?w ≤ tsum (−p) x ?w ?s using 5
    by metis
qed

end

class complete-pre = complete-tests + precondition + power

begin

definition bnd :: 'a ⇒ 'a
  where bnd x = Sup { x^n«0 | n::nat . True }

lemma bnd-test-set: test-set { x^n«0 | n::nat . True }
  by (smt mem-Collect-eq one-compl pre-closed test-set-def)

lemma bnd-test: bnd x = −−bnd x
  by (metis bnd-def bnd-test-set sup-test)

lemma bnd-upper: x^m«0 ≤ bnd x
proof −
  have x^m«0 ∈ { x^m«0 | m::nat . True }
    by (smt mem-Collect-eq)
  thus ?thesis
    by (metis bnd-def bnd-test-set sup-upper)
qed

lemma bnd-least: (∀ n . x^n«0 ≤ −p) ⟶ bnd x ≤ −p
proof
  assume ∀ n . x^n«0 ≤ −p
  hence ∀ y∈{ x^n«0 | n::nat . True } . y ≤ −p
    by (smt mem-Collect-eq)
  thus bnd x ≤ −p
    by (metis bnd-def bnd-test-set sup-least)
qed

lemma mult-right-dist-bnd: (∀ n . (x^n«0);−p ≤ −q) ⟶ bnd x;−p ≤ −q
proof
  assume ∀ n . (x^n«0);−p ≤ −q
  hence Sup { y;−p | y . y ∈ { x^n«0 | n::nat . True } } ≤ −q
    by (smt mem-Collect-eq one-compl pre-closed sub-mult-closed sup-least test-set-def)
  thus bnd x;−p ≤ −q using bnd-test-set bnd-def mult-right-dist-sup
    by simp
qed

lemma tests-complete: nat-test (λn . (−p;x)^n«0) (bnd(−p;x))
  by (smt bnd-test bnd-upper mult-right-dist-bnd nat-test-def one-compl pre-closed)

end
```

end

33 PrePost

theory *PrePost*

imports *Precondition Semiring*

begin

class *pre-post* =
  fixes *pre-post* :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infix** ⊣ *55*)

class *pre-post-spec-greatest* = *bounded-idempotent-left-semiring* + *precondition* + *pre-post* +
  assumes *pre-post-galois*: $-p \le x«-q \longleftrightarrow x \le -p⊣-q$

begin

— Theorem 42.1

lemma *post-pre-left-antitone*: $x \le y \longrightarrow y«-q \le x«-q$
  by (*smt order-refl order-trans pre-closed pre-post-galois*)

lemma *pre-left-sub-dist*: $x+y«-q \le x«-q$
  by (*metis add-left-upper-bound post-pre-left-antitone*)

— Theorem 42.2

lemma *pre-post-left-antitone*: $-p \le -q \longrightarrow -q⊣-r \le -p⊣-r$
  by (*metis order-refl order-trans pre-post-galois*)

lemma *pre-post-left-sub-dist*: $-p+-q⊣-r \le -p⊣-r$
  by (*metis add-left-upper-bound plus-closed pre-post-left-antitone*)

lemma *pre-post-left-sup-dist*: $-p⊣-r \le -p;-q⊣-r$
  by (*metis lower-bound-left pre-post-left-antitone sub-mult-closed*)

— Theorem 42.5

lemma *pre-pre-post*: $x \le (x«-p)⊣-p$
  by (*metis order-refl pre-closed pre-post-galois*)

— Theorem 42.6

lemma *pre-post-pre*: $-p \le (-p⊣-q)«-q$
  by (*metis eq-refl pre-post-galois*)

— Theorem 42.8

lemma *pre-post-zero-top*: $0⊣-q = T$
  by (*metis eq-iff pre-post-galois top-greatest zero-double-compl zero-least*)

— Theorem 42.7

lemma *pre-post-pre-one*: $(1⊣-q)«-q = 1$
  by (*metis add-left-zero leq-plus-right-one one-compl one-def pre-closed pre-post-pre*)

— Theorem 42.3

lemma *pre-post-right-isotone*: $-p \le -q \longrightarrow -r⊣-p \le -r⊣-q$
  by (*metis order-trans pre-iso pre-post-galois pre-post-pre*)

lemma *pre-post-right-sub-dist*: $-r⊣-p \le -r⊣-p+-q$
  by (*metis add-left-upper-bound plus-closed pre-post-right-isotone*)

lemma *pre-post-right-sup-dist*: $-r⊣-p;-q \le -r⊣-p$
  by (*metis lower-bound-left pre-post-right-isotone sub-mult-closed*)

— Theorem 42.7

lemma *pre-post-reflexive*: $1 \le -p⊣-p$

by (*metis pre-one-increasing pre-post-galois*)

— Theorem 42.9

lemma *pre-post-compose*: $-q \le -r \longrightarrow (-p\dashv-q);(-r\dashv-s) \le -p\dashv-s$
by (*metis pre-compose pre-post-galois pre-post-left-antitone pre-post-pre*)

— Theorem 42.10

lemma *pre-post-compose-1*: $(-p\dashv-q);(-q\dashv-r) \le -p\dashv-r$
by (*metis pre-post-compose reflexive*)

— Theorem 42.11

lemma *pre-post-compose-2*: $(-p\dashv-p);(-p\dashv-q) = -p\dashv-q$
by (*metis antisym mult-left-isotone mult-left-one pre-post-compose-1 pre-post-reflexive*)

— Theorem 42.12

lemma *pre-post-compose-3*: $(-p\dashv-q);(-q\dashv-q) = -p\dashv-q$
by (*metis antisym mult-right-isotone mult-right-one pre-post-compose-1 pre-post-reflexive*)

— Theorem 42.13

lemma *pre-post-compose-4*: $(-p\dashv-p);(-p\dashv-p) = -p\dashv-p$
by (*metis pre-post-compose-2*)

— Theorem 42.14

lemma *pre-post-one-one*: $x«1 = 1 \longleftrightarrow x \le 1\dashv 1$
by (*metis eq-iff one-def pre-below-one pre-post-galois*)

— Theorem 42.4

lemma *post-pre-left-dist-add*: $x+y«-q = (x«-q);(y«-q)$
apply (*rule antisym*)
apply (*smt add-commutative greatest-lower-bound pre-closed pre-left-sub-dist*)
apply (*smt pre-pre-post pre-closed pre-post-left-sup-dist sub-comm order-trans add-least-upper-bound pre-post-galois sub-mult-closed*)
done

lemma *pre-post-right-dist-add*: $-p\dashv-q+-r = (-p\dashv-q) + (-p\dashv-r)$ **nitpick** [*expect=genuine*] **oops**

end

class *pre-post-spec-greatest-2* = *pre-post-spec-greatest* + *precondition-test-test*

begin

subclass *precondition-test-box*
apply *unfold-locales*
apply (*metis add-commutative bs-mult-right-one double-negation eq-iff mult-left-one mult-right-dist-add one-def plus-compl plus-compl-intro pre-below-one pre-import pre-post-galois pre-test-test zero-def zero-least*)
done

lemma *pre-post-seq-sub-associative*: $(-p\dashv-q);-r \le -p\dashv-q;-r$
by (*smt mult-right-isotone mult-right-one one-greatest pre-mult-test-promote pre-post-galois sub-comm sub-mult-closed*)

lemma *pre-post-right-import-mult*: $(-p\dashv-q);-r = (-p\dashv-q;-r);-r$
by (*metis antisym mult-associative mult-idempotent mult-left-isotone pre-post-right-sup-dist pre-post-seq-sub-associative*)

lemma *seq-pre-post-sub-associative*: $-r;(-p\dashv-q) \le --r+-p\dashv-q$
by (*metis add-least-upper-bound leq-def mult-left-isotone mult-left-one mult-right-one one-def plus-closed pre-neg-mult pre-post-galois*)

lemma *pre-post-left-import-add*: $-r;(-p\dashv-q) = -r;(--r+-p\dashv-q)$
by (*metis add-commutative antisym mult-associative mult-idempotent mult-right-isotone pre-post-left-sub-dist seq-pre-post-sub-associative*)

lemma *pre-post-import-same*: $-p;(-p\dashv-q) = -p;(1\dashv-q)$
  by (*metis double-negation plus-compl pre-post-left-import-add*)

lemma *pre-post-import-complement*: $--p;(-p\dashv-q) = --p;T$
  by (*metis mult-idempotent plus-cases plus-closed pre-post-left-import-add pre-post-zero-top zero-def zero-double-compl*)

lemma *pre-post-export*: $-p\dashv-q = (1\dashv-q) + --p;T$
proof (*rule antisym*)
  have *1*: $-p;(-p\dashv-q) \le (1\dashv-q) + --p;T$
    by (*metis add-left-upper-bound mult-left-one mult-right-sub-dist-add-right order-trans plus-left-one pre-post-import-same*)
  have $--p;(-p\dashv-q) \le (1\dashv-q) + --p;T$
    by (*metis add-right-upper-bound pre-post-import-complement*)
  thus $-p\dashv-q \le (1\dashv-q) + --p;T$ **using** *1*
    by (*smt case-split-left eq-refl plus-compl*)
next
  show $(1\dashv-q) + --p;T \le -p\dashv-q$
    by (*metis add-least-upper-bound double-negation one-greatest pre-neg-mult pre-post-galois pre-post-pre-one*)
qed

lemma *pre-post-left-dist-mult*: $-p;-q\dashv-r = (-p\dashv-r) + (-q\dashv-r)$
proof –
  have $\forall p\ q\ .\ -p;(-p;-q\dashv-r) = -p;(-q\dashv-r)$
    by (*metis add-commutative plus-compl-intro pre-post-left-import-add sub-mult-closed*)
  hence *1*: $(-p+-q);(-p;-q\dashv-r) \le (-p\dashv-r) + (-q\dashv-r)$
    by (*metis add-commutative add-least-upper-bound add-right-upper-bound mult-left-one mult-right-dist-add plus-left-one sub-comm*)
  have $-(-p+-q);(-p;-q\dashv-r) = -(-p+-q);T$
    by (*smt add-associative add-commutative one-compl plus-absorb plus-closed plus-right-zero pre-post-left-import-add pre-post-zero-top sub-mult-closed*)
  hence $-(-p+-q);(-p;-q\dashv-r) \le (-p\dashv-r) + (-q\dashv-r)$
    by (*metis add-left-upper-bound mult-left-one mult-right-sub-dist-add-right order-trans plus-left-one mult-associative plus-deMorgan pre-post-import-complement sub-comm*)
  thus *?thesis* **using** *1*
    by (*smt add-least-upper-bound antisym case-split-left order-refl plus-closed plus-compl pre-post-left-sup-dist sub-comm*)
qed

lemma *pre-post-left-import-mult*: $-r;(-p\dashv-q) = -r;(-r;-p\dashv-q)$
  by (*metis add-commutative plus-compl-intro pre-post-left-import-add sub-mult-closed*)

lemma *pre-post-right-import-add*: $(-p\dashv-q);-r = (-p\dashv-q+--r);-r$
  by (*smt bs-mult-right-one case-duality plus-closed plus-comm plus-compl pre-post-right-import-mult sub-comm wnf-lemma-1*)

lemma *pre-post-shunting*: $x \le -p;-q\dashv-r \longleftrightarrow -p;x \le -q\dashv-r$
proof –
  have $--p;x \le -p;-q\dashv-r$
    by (*metis double-negation order-trans pre-neg-mult pre-post-galois pre-post-left-sup-dist*)
  hence *1*: $-p;x \le -q\dashv-r \longrightarrow x \le -p;-q\dashv-r$
    by (*smt case-split-left eq-refl order-trans plus-compl pre-post-left-sup-dist sub-comm*)
  have $-p;(-p;-q\dashv-r) \le -q\dashv-r$
    by (*metis mult-left-isotone mult-left-one one-greatest pre-post-left-import-mult*)
  thus *?thesis* **using** *1*
    by (*smt mult-right-isotone order-trans*)
qed

lemma *pre-post-right-dist-add*: $-p\dashv-q+-r = (-p\dashv-q) + (-p\dashv-r)$ **oops**

end

class *left-zero-pre-post-spec-greatest-2* = *pre-post-spec-greatest-2* + *bounded-idempotent-left-zero-semiring*

begin

lemma *pre-post-right-dist-add*: $-p\dashv-q+-r = (-p\dashv-q) + (-p\dashv-r)$
proof –
  have *1*: $(-p\dashv-q+-r);-q \le (-p\dashv-q) + (-p\dashv-r)$
    by (*smt add-left-upper-bound mult-absorb mult-left-sub-dist-add-right mult-right-one order-trans plus-closed plus-left-one pre-post-right-import-mult sub-comm*)
  have $(-p\dashv-q+-r);--q = (-p\dashv-r);--q$
    by (*smt plus-def pre-post-right-import-mult mult-compl-intro mult-distr-plus-right mult-left-dist-add unique-zero*)

hence $(-p\dashv-q+-r);--q \le (-p\dashv-q) + (-p\dashv-r)$
  by (*metis add-right-upper-bound mult-left-sub-dist-add-right mult-right-one order-trans plus-left-one*)
thus *?thesis* **using** *1*
  by (*metis add-least-upper-bound antisym case-split-right one-greatest plus-comm plus-compl pre-post-right-sub-dist*)
qed

end

class *havoc* =
  fixes $H :: 'a$

class *idempotent-left-semiring-H* = *bounded-idempotent-left-semiring* + *havoc* +
  assumes *H-zero* : $H ; 0 = 0$
  assumes *H-split*: $x \le x ; 0 + H$

begin

lemma *H-galois*: $x ; 0 \le y \longleftrightarrow x \le y + H$
  by (*smt H-split H-zero add-associative add-commutative add-left-zero add-right-isotone less-eq-def mult-right-dist-add mult-right-sub-dist-add-left zero-right-mult-decreasing*)

lemma *H-greatest-finite*: $x ; 0 = 0 \longleftrightarrow x \le H$
  by (*metis H-galois add-left-zero eq-iff zero-least*)

lemma *H-reflexive*: $1 \le H$
  by (*metis H-greatest-finite mult-left-one*)

lemma *H-transitive*: $H = H ; H$
  by (*metis H-greatest-finite H-reflexive H-zero antisym-conv mult-associative mult-right-isotone mult-right-one*)

lemma *T-split-H*: $T ; 0 + H = T$
  by (*metis H-split add-left-top less-eq-def*)

lemma $H ; (x + y) = H ; x + H ; y$ **nitpick** [*expect=genuine*] **oops**

end

class *pre-post-spec-least* = *bounded-idempotent-left-semiring* + *precondition-test-test* + *precondition-promote* + *pre-post* +
  assumes *test-mult-right-distr-add*: $-p ; (x + y) = -p ; x + -p ; y$
  assumes *pre-post-galois*: $-p \le x \text{«} -q \longleftrightarrow -p\dashv-q \le x$

begin

lemma *shunting-T*: $-p ; x \le y \longleftrightarrow x \le y + --p ; T$
proof
  assume $-p ; x \le y$
  thus $x \le y + --p ; T$
    by (*smt mult-left-one mult-right-dist-add plus-compl add-isotone mult-right-isotone top-greatest*)
next
  assume $x \le y + --p ; T$
  hence $-p ; x \le -p ; y$
    by (*metis add-right-zero mult-associative mult-compl mult-left-zero mult-right-isotone test-mult-right-distr-add*)
  thus $-p ; x \le y$
    by (*metis mult-left-isotone mult-left-one one-greatest order-trans*)
qed

lemma *post-pre-left-isotone*: $x \le y \longrightarrow x\text{«}-q \le y\text{«}-q$
  by (*smt order-refl order-trans pre-closed pre-post-galois*)

lemma *pre-left-sub-dist*: $x\text{«}-q \le x+y\text{«}-q$
  by (*metis add-left-upper-bound post-pre-left-isotone*)

lemma *pre-post-left-isotone*: $-p \le -q \longrightarrow -p\dashv-r \le -q\dashv-r$
  by (*metis order-refl order-trans pre-post-galois*)

lemma *pre-post-left-sub-dist*: $-p\dashv-r \le -p+-q\dashv-r$
  by (*metis add-left-upper-bound plus-closed pre-post-left-isotone*)

lemma *pre-post-left-sup-dist*: $-p;-q\dashv-r \le -p\dashv-r$

by (*metis lower-bound-left pre-post-left-isotone sub-mult-closed*)

lemma *pre-pre-post*: $(x \ll -p) \dashv -p \le x$
by (*metis order-refl pre-closed pre-post-galois*)

lemma *pre-post-pre*: $-p \le (-p \dashv -q) \ll -q$
by (*metis eq-refl pre-post-galois*)

lemma *pre-post-zero-top*: $0 \dashv -q = 0$
by (*metis eq-iff pre-post-galois zero-double-compl zero-least*)

lemma *pre-post-pre-one*: $(1 \dashv -q) \ll -q = 1$
by (*metis add-left-zero leq-plus-right-one one-compl one-def pre-closed pre-post-pre*)

lemma *pre-post-right-antitone*: $-p \le -q \longrightarrow -r \dashv -q \le -r \dashv -p$
by (*metis order-trans pre-iso pre-post-galois pre-post-pre*)

lemma *pre-post-right-sub-dist*: $-r \dashv -p + -q \le -r \dashv -p$
by (*metis add-left-upper-bound plus-closed pre-post-right-antitone*)

lemma *pre-post-right-sup-dist*: $-r \dashv -p \le -r \dashv -p;-q$
by (*metis lower-bound-left pre-post-right-antitone sub-mult-closed*)

lemma *pre-top*: $T \ll -q = 1$
by (*metis antisym one-def pre-below-one pre-post-galois top-greatest*)

lemma *pre-mult-top-increasing*: $-p \le -p;T \ll -q$
by (*metis one-greatest pre-import-equiv pre-top*)

lemma *pre-post-below-mult-top*: $-p \dashv -q \le -p;T$
by (*metis pre-mult-top-increasing pre-post-galois*)

lemma *pre-post-import-complement*: $--p;(-p \dashv -q) = 0$
proof −
have $--p;(-p \dashv -q) \le --p;(-p;T)$
by (*metis mult-right-isotone pre-post-below-mult-top*)
thus *?thesis*
by (*metis mult-associative mult-left-zero sub-comm zero-def antisym zero-least*)
qed

lemma *pre-post-import-same*: $-p;(-p \dashv -q) = -p \dashv -q$
proof −
have $-p \dashv -q = -p;(-p \dashv -q) + --p;(-p \dashv -q)$
by (*metis mult-left-one mult-right-dist-add plus-compl*)
thus *?thesis*
by (*metis add-right-zero pre-post-import-complement*)
qed

lemma *pre-post-export*: $-p \dashv -q = -p;(1 \dashv -q)$
apply (*rule antisym*)
apply (*metis one-greatest pre-import-equiv pre-post-galois pre-post-pre-one*)
apply (*smt add-commutative add-left-upper-bound leq-plus-right-one less-eq-def order-trans pre-closed pre-mult-top-increasing pre-post-galois shunting-T*)
done

lemma *pre-post-seq-associative*: $-r;(-p \dashv -q) = -r;-p \dashv -q$
by (*metis mult-associative pre-post-export sub-mult-closed*)

lemma *pre-post-left-import-mult*: $-r;(-p \dashv -q) = -r;(-r;-p \dashv -q)$
by (*metis mult-associative mult-idempotent pre-post-seq-associative*)

lemma *seq-pre-post-sub-associative*: $-r;(-p \dashv -q) \le --r+-p \dashv -q$
by (*metis add-commutative mult-left-one mult-right-sub-dist-add-left order-trans plus-compl pre-post-left-sub-dist*)

lemma *pre-post-left-import-add*: $-r;(-p \dashv -q) = -r;(--r+-p \dashv -q)$
by (*metis mult-compl-intro plus-closed pre-post-seq-associative*)

lemma *pre-post-left-dist-add*: $-p+-q \dashv -r = (-p \dashv -r) + (-q \dashv -r)$
by (*metis mult-right-dist-add plus-closed pre-post-export*)

lemma *pre-post-reflexive*: $-p\dashv-p \le 1$
  by (*metis pre-one pre-post-galois reflexive*)

lemma *pre-post-compose*: $-q \le -r \longrightarrow -p\dashv-s \le (-p\dashv-q);(-r\dashv-s)$
  by (*metis pre-compose pre-post-galois pre-post-left-isotone pre-post-pre*)

lemma *pre-post-compose-1*: $-p\dashv-r \le (-p\dashv-q);(-q\dashv-r)$
  by (*metis pre-post-compose reflexive*)

lemma *pre-post-compose-2*: $(-p\dashv-p);(-p\dashv-q) = -p\dashv-q$
  by (*metis antisym mult-left-isotone mult-left-one pre-post-compose-1 pre-post-reflexive*)

lemma *pre-post-compose-3*: $(-p\dashv-q);(-q\dashv-q) = -p\dashv-q$
  by (*metis antisym mult-right-isotone mult-right-one pre-post-compose-1 pre-post-reflexive*)

lemma *pre-post-compose-4*: $(-p\dashv-p);(-p\dashv-p) = -p\dashv-p$
  by (*metis pre-post-compose-2*)

lemma *pre-post-one-one*: $x«1 = 1 \longleftrightarrow 1\dashv1 \le x$
  by (*metis eq-iff one-def pre-below-one pre-post-galois*)

lemma *pre-one-right*: $-p«1 = -p$
  by (*metis antisym mult-right-one one-def plus-compl pre-left-sub-dist pre-mult-top-increasing pre-one pre-seq pre-test-promote pre-top*)

lemma *pre-pre-one*: $x«-q = x;-q«1$
  by (*metis one-def pre-one-right pre-seq*)

subclass *precondition-test-diamond*
  apply *unfold-locales*
  apply (*metis pre-one-right pre-pre-one sub-mult-closed*)
  done

lemma *pre-post-shunting*: $x \le -p;-q\dashv-r \longleftrightarrow -p;x \le -q\dashv-r$ **nitpick** [*expect=genuine*] **oops**
lemma $(-p\dashv-q);-r = (-p\dashv-q+-r);-r$ **nitpick** [*expect=genuine*] **oops**
lemma $(-p\dashv-q);-r = (-p\dashv-q+--r);-r$ **nitpick** [*expect=genuine*] **oops**
lemma $(-p\dashv-q);-r = (-p\dashv-q;-r);-r$ **nitpick** [*expect=genuine*] **oops**
lemma $(-p\dashv-q);-r = (-p\dashv-q;--r);-r$ **nitpick** [*expect=genuine*] **oops**
lemma $-p\dashv-q+-r = (-p\dashv-q) + (-p\dashv-r)$ **nitpick** [*expect=genuine*] **oops**
lemma $-p\dashv-q+-r = (-p\dashv-q) ; (-p\dashv-r)$ **nitpick** [*expect=genuine*] **oops**
lemma *pre-post-right-dist-mult*: $-p\dashv-q;-r = (-p\dashv-q) ; (-p\dashv-r)$ **oops**
lemma *pre-post-right-dist-mult*: $-p\dashv-q;-r = (-p\dashv-q) + (-p\dashv-r)$ **oops**
lemma *post-pre-left-dist-add*: $x+y«-q = (x«-q) + (y«-q)$ **oops**

end

class *havoc-dual* =
  fixes *Hd* :: *'a*

class *idempotent-left-semiring-Hd* = *bounded-idempotent-left-semiring* + *havoc-dual* +
  assumes *Hd-total*: $Hd ; T = T$
  assumes *Hd-least*: $x ; T = T \longrightarrow Hd \le x$

begin

lemma *Hd-least-total*: $x ; T = T \longleftrightarrow Hd \le x$
  by (*metis Hd-least Hd-total antisym mult-left-isotone top-greatest*)

lemma *Hd-reflexive*: $Hd \le 1$
  by (*metis Hd-least mult-left-one*)

lemma *Hd-transitive*: $Hd = Hd ; Hd$
  by (*metis Hd-least-total eq-iff less-eq-def mult-associative mult-left-one mult-right-sub-dist-add-left*)

end

class *pre-post-spec-least-Hd* = *idempotent-left-semiring-Hd* + *pre-post-spec-least* +
  assumes *pre-one-mult-top*: $(x«1);T = x;T$

```
begin

lemma Hd-pre-one: Hd«1 = 1
  by (metis Hd-total pre-seq pre-top)

lemma pre-post-below-Hd: 1⊣1 ≤ Hd
  by (metis Hd-pre-one pre-post-one-one)

lemma Hd-pre-post: Hd = 1⊣1
  by (metis Hd-least Hd-pre-one Hd-total eq-iff pre-one-mult-top pre-post-one-one)

lemma T-left-zero: T;x = T
  by (metis mult-associative mult-left-one mult-left-zero pre-closed pre-one-mult-top pre-seq pre-top)

lemma test-dual-test: (−p+−−p;T);−p = −p+−−p;T
  by (metis T-left-zero mult-associative mult-idempotent mult-right-dist-add)

lemma pre-zero: 0«−q = 0
  by (metis add-right-zero less-eq-def mult-left-zero pre-below-pre-one pre-one-mult-top top-right-mult-increasing)

lemma pre-zero-mult-top: (x«0);T = x;0
  by (metis mult-associative mult-left-zero one-def pre-one-mult-top pre-seq pre-zero)

lemma pre-one-mult-Hd: (x«1);Hd ≤ x
  by (metis Hd-pre-post one-def pre-closed pre-post-export pre-pre-post)

lemma Hd-mult-pre-one: Hd;(x«1) ≤ x
proof −
  have 1: −(x«1);Hd;(x«1) ≤ x
    by (metis Hd-reflexive less-eq-def mult-associative mult-isotone mult-left-one one-def pre-closed pre-one-mult-top shunting-T
top-right-mult-increasing)
  have (x«1);Hd;(x«1) ≤ x
    by (metis mult-isotone mult-right-one one-def pre-below-one pre-one-mult-Hd)
  thus ?thesis using 1
    by (metis add-idempotent case-split-left less-eq-def mult-associative one-def plus-compl pre-closed)
qed

lemma pre-post-one-def-1: 1 ≤ x«−q ⟶ Hd;(−q+−−q;T) ≤ x
proof
  assume 1 ≤ x«−q
  hence Hd;(−q+−−q;T) ≤ x;−q;(−q+−−q;T)
    by (metis Hd-pre-post antisym pre-below-one pre-post-one-one pre-pre-one mult-left-isotone)
  thus Hd;(−q+−−q;T) ≤ x
    by (metis mult-associative mult-compl mult-left-sub-dist-add-left mult-left-zero mult-right-one plus-compl
test-mult-right-distr-add order-trans)
qed

lemma pre-post-one-def: 1⊣−q = Hd;(−q+−−q;T)
proof (rule antisym)
  have 1 ≤ (1⊣1);(−q+−−q)«1
    by (metis pre-post-pre one-def mult-right-one plus-compl)
  also have ... ≤ (1⊣1);(−q+−−q;T)«−q
    by (metis add-right-isotone mult-right-isotone mult-right-one one-def post-pre-left-isotone pre-seq pre-test-promote
test-dual-test top-right-mult-increasing)
  finally show 1⊣−q ≤ Hd;(−q+−−q;T)
    by (metis Hd-pre-post one-def pre-post-galois)
next
  show Hd;(−q+−−q;T) ≤ 1⊣−q
    by (metis pre-post-pre one-def pre-post-one-def-1)
qed

lemma pre-post-def: −p⊣−q = −p;Hd;(−q+−−q;T)
  by (metis mult-associative pre-post-export pre-post-one-def)

end

end
```

34 RelativeDomain

theory *RelativeDomain*

imports *Semiring Tests*

begin

class $Z =$
  fixes $Z :: 'a$

class *relative-domain-semiring* = *idempotent-left-semiring* + *d* + *Z* +
  assumes *d-restrict* : $x \le d(x) \mathbin{;} x + Z$
  assumes *d-mult-d* : $d(x \mathbin{;} y) = d(x \mathbin{;} d(y))$
  assumes *d-below-one*: $d(x) \le 1$
  assumes *d-Z* : $d(Z) = 0$
  assumes *d-dist-add* : $d(x + y) = d(x) + d(y)$
  assumes *d-export* : $d(d(x) \mathbin{;} y) = d(x) \mathbin{;} d(y)$

begin

lemma *d-plus-one*: $d(x) + 1 = 1$
  by (*metis d-below-one less-eq-def*)

— Theorem 44.2

lemma *d-zero*: $d(0) = 0$
  by (*metis d-Z d-export mult-left-zero*)

— Theorem 44.3

lemma *d-involutive*: $d(d(x)) = d(x)$
  by (*metis d-mult-d mult-left-one*)

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

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

— Theorem 44.4

lemma *d-mult-sub*: $d(x \mathbin{;} y) \le d(x)$
  by (*metis add-commutative d-below-one d-dist-add d-mult-d less-eq-def mult-left-sub-dist-add-right mult-right-one*)

lemma *d-sub-one*: $x \le 1 \longrightarrow x \le d(x) + Z$
  by (*metis add-left-isotone d-restrict mult-right-isotone mult-right-one order-trans*)

lemma *d-one*: $d(1) + Z = 1 + Z$
  by (*smt add-associative add-commutative d-plus-one d-restrict less-eq-def mult-right-one*)

— Theorem 44.8

lemma *d-strict*: $d(x) = 0 \longleftrightarrow x \le Z$
  by (*metis add-commutative add-right-zero d-Z d-dist-add d-restrict less-eq-def mult-left-zero*)

— Theorem 44.1

lemma *d-isotone*: $x \le y \longrightarrow d(x) \le d(y)$
  by (*metis d-dist-add less-eq-def*)

lemma *d-plus-left-upper-bound*: $d(x) \le d(x + y)$
  by (*metis add-left-upper-bound d-isotone*)

lemma *d-idempotent*: $d(x) \mathbin{;} d(x) = d(x)$
  by (*smt add-commutative add-right-zero d-Z d-dist-add d-export d-involutive d-mult-sub d-restrict less-eq-def*)

— Theorem 44.12

lemma *d-least-left-preserver*: $x \leq d(y)\ ;\ x + Z \longleftrightarrow d(x) \leq d(y)$
  apply *rule*
  apply (*smt add-associative add-left-divisibility add-right-zero d-Z d-dist-add d-involutive d-mult-sub less-eq-def*)
  apply (*smt add-associative add-commutative d-restrict less-eq-def mult-right-dist-add*)
  done

— Theorem 44.9

lemma *d-weak-locality*: $x\ ;\ y \leq Z \longleftrightarrow x\ ;\ d(y) \leq Z$
  by (*metis d-mult-d d-strict*)

lemma *d-add-closed*: $d(d(x) + d(y)) = d(x) + d(y)$
  by (*metis d-dist-add d-involutive*)

lemma *d-mult-closed*: $d(d(x)\ ;\ d(y)) = d(x)\ ;\ d(y)$
  by (*metis d-export d-mult-d*)

lemma *d-mult-left-lower-bound*: $d(x)\ ;\ d(y) \leq d(x)$
  by (*metis d-export d-involutive d-mult-sub*)

lemma *d-mult-left-absorb-add*: $d(x)\ ;\ (d(x) + d(y)) = d(x)$
  by (*smt d-add-closed d-export d-idempotent d-involutive d-mult-sub eq-iff mult-left-sub-dist-add-left*)

lemma *d-add-left-absorb-mult*: $d(x) + d(x)\ ;\ d(y) = d(x)$
  by (*metis add-commutative d-mult-left-lower-bound less-eq-def*)

lemma *d-commutative*: $d(x)\ ;\ d(y) = d(y)\ ;\ d(x)$
  by (*metis add-commutative antisym d-add-left-absorb-mult d-below-one d-export d-mult-left-absorb-add mult-associative mult-left-isotone mult-left-one*)

lemma *d-mult-greatest-lower-bound*: $d(x) \leq d(y)\ ;\ d(z) \longleftrightarrow d(x) \leq d(y) \wedge d(x) \leq d(z)$
  by (*metis d-commutative d-idempotent d-mult-left-lower-bound mult-isotone order-trans*)

lemma *d-add-left-dist-mult*: $d(x) + d(y)\ ;\ d(z) = (d(x) + d(y))\ ;\ (d(x) + d(z))$
  by (*metis add-associative d-commutative d-dist-add d-idempotent d-mult-left-absorb-add mult-right-dist-add*)

lemma *d-order*: $d(x) \leq d(y) \longleftrightarrow d(x) = d(x)\ ;\ d(y)$
  by (*metis d-mult-greatest-lower-bound d-mult-left-absorb-add less-eq-def order-refl*)

— Theorem 44.6

lemma *Z-mult-decreasing*: $Z\ ;\ x \leq Z$
  by (*metis add-left-zero d-Z d-least-left-preserver d-mult-sub mult-left-zero*)

— Theorem 44.5

lemma *d-below-d-one*: $d(x) \leq d(1)$
  by (*metis d-mult-sub mult-left-one*)

— Theorem 44.7

lemma *d-relative-Z*: $d(x)\ ;\ x + Z = x + Z$
  by (*metis add-left-upper-bound add-same-context d-below-one d-restrict mult-isotone mult-left-one*)

lemma *Z-left-zero-above-one*: $1 \leq x \longrightarrow Z\ ;\ x = Z$
  by (*metis Z-mult-decreasing eq-iff mult-right-isotone mult-right-one*)

— Theorem 44.11

lemma *kat-4*: $d(x)\ ;\ y = d(x)\ ;\ y\ ;\ d(z) \longrightarrow d(x)\ ;\ y \leq y\ ;\ d(z)$
  by (*metis d-below-one mult-left-isotone mult-left-one*)

lemma *kat-4-equiv*: $d(x)\ ;\ y = d(x)\ ;\ y\ ;\ d(z) \longleftrightarrow d(x)\ ;\ y \leq y\ ;\ d(z)$
  apply *rule*
  apply (*metis kat-4*)
  apply (*rule antisym*)
  apply (*metis d-idempotent mult-associative mult-right-isotone*)
  apply (*metis d-below-one mult-right-isotone mult-right-one*)
  done

```
lemma kat-4-equiv-opp: y ; d(x) = d(z) ; y ; d(x) ⟷ y ; d(x) ≤ d(z) ; y
  apply rule
  apply (metis d-below-one mult-right-isotone mult-right-one)
  apply (rule antisym)
  apply (metis d-idempotent mult-associative mult-left-isotone)
  apply (metis d-below-one mult-left-isotone mult-left-one)
  done

— Theorem 44.10

lemma d-restrict-iff-1: d(x) ; y ≤ z ⟷ d(x) ; y ≤ d(x) ; z
  by (smt d-below-one d-idempotent mult-associative mult-left-isotone mult-left-one mult-right-isotone order-trans)



end

typedef 'a dImage = { x::'a::relative-domain-semiring . (∃ y::'a . x = d(y)) }
  by auto

lemma simp-dImage [simp]: ∃ y . Rep-dImage x = d(y)
  using Rep-dImage
  by simp

setup-lifting type-definition-dImage

— Theorem 44

instantiation dImage :: (relative-domain-semiring) bounded-distributive-lattice

begin

lift-definition plus-dImage :: 'a dImage ⇒ 'a dImage ⇒ 'a dImage is plus
  by (metis d-dist-add)

lift-definition meet-dImage :: 'a dImage ⇒ 'a dImage ⇒ 'a dImage is times
  by (metis d-export)

lift-definition zero-dImage :: 'a dImage is 0
  by (metis d-zero)

lift-definition T-dImage :: 'a dImage is d(1)
  by metis

lift-definition less-eq-dImage :: 'a dImage ⇒ 'a dImage ⇒ bool is less-eq .

lift-definition less-dImage :: 'a dImage ⇒ 'a dImage ⇒ bool is less .

instance
  apply intro-classes
  apply (metis (mono-tags) Rep-dImage-inject add-associative plus-dImage.rep-eq)
  apply (metis (mono-tags) Rep-dImage-inject add-commutative plus-dImage.rep-eq)
  apply (metis (mono-tags) Rep-dImage-inject add-idempotent plus-dImage.rep-eq)
  apply (metis (mono-tags) Rep-dImage-inject less-eq-def less-eq-dImage.rep-eq plus-dImage.rep-eq)
  apply (metis less-eq-dImage.rep-eq less-dImage.rep-eq less-def)
  apply (smt2 zero-dImage.rep-eq Rep-dImage-inject add-left-zero plus-dImage.rep-eq)
  apply (metis (mono-tags) Rep-dImage-inverse mult-associative meet-dImage.rep-eq)
  apply (metis (mono-tags) meet-dImage.rep-eq Rep-dImage-inverse simp-dImage d-commutative)
  apply (metis (mono-tags) meet-dImage.rep-eq Rep-dImage-inverse simp-dImage d-idempotent)
  apply (metis (mono-tags) meet-dImage.rep-eq Rep-dImage-inverse simp-dImage d-order less-eq-dImage.rep-eq)
  apply (smt2 T-dImage.rep-eq Rep-dImage-inject d-below-d-one d-commutative d-order meet-dImage.rep-eq simp-dImage)
  apply (smt2 Rep-dImage-inject d-commutative meet-dImage.rep-eq mult-right-dist-add plus-dImage.rep-eq simp-dImage)
  apply (metis (mono-tags) Rep-dImage-inject meet-dImage.rep-eq d-add-left-dist-mult plus-dImage.rep-eq simp-dImage)
  apply (metis (mono-tags) Rep-dImage-inject meet-dImage.rep-eq d-mult-left-absorb-add plus-dImage.rep-eq simp-dImage)
  apply (metis (mono-tags) Rep-dImage-inject meet-dImage.rep-eq d-add-left-absorb-mult plus-dImage.rep-eq simp-dImage)
  done

end
```

```
class bounded-relative-domain-semiring = relative-domain-semiring + bounded-idempotent-left-semiring

begin

lemma Z-top: Z ; T = Z
  by (metis Z-mult-decreasing eq-iff top-right-mult-increasing)

lemma d-restrict-T: x ≤ d(x) ; T + Z
  by (metis add-left-isotone d-restrict mult-right-isotone order-trans top-greatest)

lemma d-one-one: d(1) = 1 nitpick [expect=genuine] oops

end

class relative-domain-semiring-split = relative-domain-semiring +
  assumes split-Z: x ; (y + Z) ≤ x ; y + Z

begin

lemma d-restrict-iff: (x ≤ y + Z) ⟷ (x ≤ d(x) ; y + Z)
proof −
  have x ≤ y + Z ⟶ x ≤ d(x) ; (y + Z) + Z
    by (smt add-left-isotone d-restrict less-eq-def mult-left-sub-dist-add-left order-trans)
  hence x ≤ y + Z ⟶ x ≤ d(x) ; y + Z
    by (metis add-isotone add-right-zero add-same-context d-strict d-zero mult-left-sub-dist-add-left split-Z)
  thus ?thesis
    by (smt d-below-one mult-left-isotone add-left-isotone mult-left-one order-trans)
qed

end

class relative-antidomain-semiring = idempotent-left-semiring + d + Z + neg +
  assumes a-restrict  : −x ; x ≤ Z
  assumes a-mult-d    : −(x ; y) = −(x ; −−y)
  assumes a-complement: −x ; −−x = 0
  assumes a-Z         : −Z = 1
  assumes a-export    : −(−−x ; y) = −x + −y
  assumes a-dist-add  : −(x + y) = −x ; −y
  assumes d-def       : d(x) = −−x

begin

notation
  uminus (a)

— Theorem 45.7

lemma a-complement-one: −−x + −x = 1
  by (metis a-Z a-complement a-export a-mult-d mult-left-one)

— Theorem 45.5 and Theorem 45.6

lemma a-d-closed: d(a(x)) = a(x)
  by (metis a-mult-d d-def mult-left-one)

lemma a-below-one: a(x) ≤ 1
  by (metis a-complement-one add-right-divisibility)

lemma a-export-a: a(a(x) ; y) = d(x) + a(y)
  by (metis a-d-closed a-export d-def)

lemma a-add-absorb: (x + a(y)) ; a(a(y)) = x ; a(a(y))
  by (metis a-complement add-right-zero mult-right-dist-add)

— Theorem 45.10

lemma a-greatest-left-absorber: a(x) ; y ≤ Z ⟷ a(x) ≤ a(y)
  apply rule
  apply (smt a-Z a-add-absorb a-dist-add a-export-a a-mult-d add-commutative d-def less-eq-def mult-left-one)
```

apply (*metis a-restrict le-less-trans le-neq-trans less-eq-def less-imp-le mult-right-sub-dist-add-left*)
done

lemma *a-plus-left-lower-bound*: $a(x + y) \leq a(x)$
by (*metis a-greatest-left-absorber a-restrict add-commutative mult-left-sub-dist-add-right order-trans*)

— Theorem 45.2

subclass *relative-domain-semiring*
apply *unfold-locales*
apply (*smt a-Z a-complement-one a-restrict add-commutative add-left-upper-bound case-split-left d-def order-trans*)
apply (*metis a-mult-d d-def*)
apply (*metis a-below-one d-def*)
apply (*metis a-Z a-complement d-def mult-left-one*)
apply (*metis a-dist-add a-export-a d-def*)
apply (*metis a-dist-add a-export d-def*)
done

— Theorem 45.1

subclass *tests*
apply *unfold-locales*
apply (*metis mult-associative*)
apply (*metis a-dist-add add-commutative*)
apply (*smt a-complement a-d-closed a-export-a add-right-zero d-add-left-dist-mult*)
apply (*metis a-d-closed a-dist-add d-def*)
apply (*rule the-equality*[*THEN sym*])
apply (*metis a-complement*)
apply (*metis a-complement*)
apply (*metis a-Z a-d-closed d-Z d-def*)
apply (*metis a-d-closed a-export d-def*)
apply (*smt a-d-closed a-dist-add a-plus-left-lower-bound add-commutative d-def less-eq-def*)
apply (*metis less-def*)
done

lemma *a-plus-mult-d*: $-(x \mathbin{;} y) + -(x \mathbin{;} --y) = -(x \mathbin{;} --y)$
by (*metis a-mult-d add-idempotent*)

lemma *a-mult-d-2*: $a(x \mathbin{;} y) = a(x \mathbin{;} d(y))$
by (*metis a-mult-d d-def*)

lemma *a-idempotent*: $a(x) \mathbin{;} a(x) = a(x)$
by (*metis a-dist-add add-idempotent*)

lemma *a-3*: $a(x) \mathbin{;} a(y) \mathbin{;} d(x + y) = 0$
by (*metis a-complement a-dist-add d-def*)

lemma *a-fixpoint*: $\forall x \,.\, (a(x) = x \longrightarrow (\forall y \,.\, y = 0))$
by (*metis a-idempotent mult-left-one mult-left-zero one-def zero-def*)

— Theorem 45.9

lemma *a-strict*: $a(x) = 1 \longleftrightarrow x \leq Z$
by (*metis d-def d-strict double-negation one-compl one-def*)

lemma *d-complement-zero*: $d(x) \mathbin{;} a(x) = 0$
by (*metis d-def sub-comm zero-def*)

lemma *a-complement-zero*: $a(x) \mathbin{;} d(x) = 0$
by (*metis d-def zero-def*)

lemma *a-shunting-zero*: $a(x) \mathbin{;} d(y) = 0 \longleftrightarrow a(x) \leq a(y)$
by (*metis d-def leq-mult-zero*)

lemma *a-antitone*: $x \leq y \longrightarrow a(y) \leq a(x)$
by (*metis a-plus-left-lower-bound less-eq-def*)

lemma *a-mult-deMorgan*: $a(a(x) \mathbin{;} a(y)) = d(x + y)$
by (*metis a-dist-add d-def*)

lemma *a-mult-deMorgan-1*: $a(a(x) ; a(y)) = d(x) + d(y)$
  by (*metis a-mult-deMorgan d-dist-add*)

lemma *a-mult-deMorgan-2*: $a(d(x) ; d(y)) = a(x) + a(y)$
  by (*metis d-def plus-def*)

lemma *a-plus-deMorgan*: $a(a(x) + a(y)) = d(x) ; d(y)$
  by (*metis a-dist-add d-def*)

lemma *a-plus-deMorgan-1*: $a(d(x) + d(y)) = a(x) ; a(y)$
  by (*metis a-mult-deMorgan-1 sub-mult-closed*)

— Theorem 45.8

lemma *a-mult-left-upper-bound*: $a(x) \leq a(x ; y)$
  by (*metis a-antitone d-def d-mult-sub double-negation*)

— Theorem 45.6

lemma *d-a-closed*: $a(d(x)) = a(x)$
  by (*metis a-d-closed d-def*)

lemma *a-export-d*: $a(d(x) ; y) = a(x) + a(y)$
  by (*metis a-export d-def*)

lemma *a-7*: $d(x) ; a(d(y) + d(z)) = d(x) ; a(y) ; a(z)$
  by (*metis a-plus-deMorgan-1 mult-associative*)

lemma *d-a-shunting*: $d(x) ; a(y) \leq d(z) \longleftrightarrow d(x) \leq d(z) + d(y)$
  by (*smt a-dist-add d-def plus-closed shunting sub-comm*)

lemma *d-d-shunting*: $d(x) ; d(y) \leq d(z) \longleftrightarrow d(x) \leq d(z) + a(y)$
  by (*metis d-a-closed d-a-shunting d-def*)

lemma *d-cancellation-1*: $d(x) \leq d(y) + (d(x) ; a(y))$
  by (*metis a-dist-add add-commutative add-left-upper-bound d-def plus-compl-intro*)

lemma *d-cancellation-2*: $(d(z) + d(y)) ; a(y) \leq d(z)$
  by (*metis d-a-shunting d-dist-add eq-refl*)

lemma *a-add-closed*: $d(a(x) + a(y)) = a(x) + a(y)$
  by (*metis d-def plus-closed*)

lemma *a-mult-closed*: $d(a(x) ; a(y)) = a(x) ; a(y)$
  by (*metis d-def sub-mult-closed*)

lemma *d-a-shunting-zero*: $d(x) ; a(y) = 0 \longleftrightarrow d(x) \leq d(y)$
  by (*metis d-def double-negation leq-mult-zero*)

lemma *d-d-shunting-zero*: $d(x) ; d(y) = 0 \longleftrightarrow d(x) \leq a(y)$
  by (*metis d-def leq-mult-zero*)

lemma *d-compl-intro*: $d(x) + d(y) = d(x) + a(x) ; d(y)$
  by (*metis add-commutative d-def plus-compl-intro*)

lemma *a-compl-intro*: $a(x) + a(y) = a(x) + d(x) ; a(y)$
  by (*smt a-dist-add add-commutative d-def mult-right-one plus-compl plus-distr-mult-left*)

lemma *kat-2*: $y ; a(z) \leq a(x) ; y \longrightarrow d(x) ; y ; a(z) = 0$
  by (*metis d-complement-zero eq-iff mult-associative mult-left-zero mult-right-isotone zero-least*)

— Theorem 45.4

lemma *kat-2-equiv*: $y ; a(z) \leq a(x) ; y \longleftrightarrow d(x) ; y ; a(z) = 0$
  apply *rule*
  apply (*metis kat-2*)
   apply (*metis a-Z a-below-one a-complement-one case-split-left d-def mult-associative mult-right-isotone mult-right-one zero-least*)
  done

lemma *kat-3-equiv-opp*: $a(z)\ ;\ y\ ;\ d(x) = 0 \longleftrightarrow y\ ;\ d(x) = d(z)\ ;\ y\ ;\ d(x)$
by (*metis a-complement-one add-left-zero d-def mult-associative mult-left-one mult-left-zero mult-right-dist-add unique-zero zero-double-compl*)

— Theorem 45.4

lemma *kat-3-equiv-opp-2*: $d(z)\ ;\ y\ ;\ a(x) = 0 \longleftrightarrow y\ ;\ a(x) = a(z)\ ;\ y\ ;\ a(x)$
by (*metis a-d-closed kat-3-equiv-opp d-def*)

lemma *kat-equiv-6*: $d(x)\ ;\ y\ ;\ a(z) = d(x)\ ;\ y\ ;\ 0 \longleftrightarrow d(x)\ ;\ y\ ;\ a(z) \leq y\ ;\ 0$
by (*metis a-d-closed antisym d-idempotent kat-4 mult-associative mult-right-isotone mult-right-one one-def zero-least-test*)

lemma *a-one*: $a(1) = 0$
by (*metis one-compl*)

lemma *d-one-one*: $d(1) = 1$
by (*metis d-def one-double-compl*)

lemma *case-split-left-add*: $-p\ ;\ x \leq y \wedge --p\ ;\ x \leq z \longrightarrow x \leq y + z$
by (*metis a-complement a-dist-add add-isotone mult-left-one mult-right-dist-add one-def plus-closed*)

lemma *test-mult-left-sub-dist-shunt*: $-p\ ;\ (--p\ ;\ x + Z) \leq Z$
by (*metis a-Z a-dist-add a-export a-greatest-left-absorber add-commutative add-left-upper-bound mult-left-one*)

lemma *test-mult-left-dist-shunt*: $-p\ ;\ (--p\ ;\ x + Z) = -p\ ;\ Z$
by (*smt add-commutative antisym mult-associative mult-idempotent mult-left-sub-dist-add-left mult-right-isotone test-mult-left-sub-dist-shunt*)

end

typedef *'a aImage* = { *x::'a::relative-antidomain-semiring* . (∃ *y::'a* . $x = a(y)$) }
by *auto*

lemma *simp-aImage* [*simp*]: ∃ *y* . *Rep-aImage x* = $a(y)$
using *Rep-aImage*
by *simp*

setup-lifting *type-definition-aImage*

— Theorem 45.3

instantiation *aImage* :: (*relative-antidomain-semiring*) *boolean-algebra*

begin

lift-definition *sup-aImage* :: *'a aImage* ⇒ *'a aImage* ⇒ *'a aImage* **is** *plus*
by (*metis plus-closed*)

lift-definition *inf-aImage* :: *'a aImage* ⇒ *'a aImage* ⇒ *'a aImage* **is** *times*
by (*metis a-dist-add*)

lift-definition *minus-aImage* :: *'a aImage* ⇒ *'a aImage* ⇒ *'a aImage* **is** $\lambda x\ y\ .\ x\ ;\ a(y)$
by (*metis a-dist-add*)

lift-definition *uminus-aImage* :: *'a aImage* ⇒ *'a aImage* **is** *a*
by *metis*

lift-definition *bot-aImage* :: *'a aImage* **is** *0*
by (*metis a-one*)

lift-definition *top-aImage* :: *'a aImage* **is** *1*
by (*metis a-Z*)

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

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

```
instance
  apply intro-classes
  apply (metis less-eq-aImage.rep-eq less-aImage.rep-eq less-def)
  apply (metis less-eq-aImage.rep-eq simp-aImage reflexive)
  apply (metis (mono-tags) less-eq-aImage.rep-eq simp-aImage transitive)
  apply (metis Rep-aImage-inject antisymmetric less-eq-aImage.rep-eq simp-aImage)
  apply (metis (mono-tags) inf-aImage.rep-eq less-eq-aImage.rep-eq lower-bound-left simp-aImage)
  apply (metis (mono-tags) inf-aImage.rep-eq less-eq-aImage.rep-eq lower-bound-right simp-aImage)
  apply (smt2 inf-aImage.rep-eq leq-def less-eq-aImage.rep-eq simp-aImage sub-assoc)
  apply (metis (mono-tags) less-eq-aImage.rep-eq simp-aImage sup-aImage.rep-eq upper-bound-left)
  apply (metis (mono-tags) less-eq-aImage.rep-eq simp-aImage sup-aImage.rep-eq upper-bound-right)
  apply (smt2 leq-plus less-eq-aImage.rep-eq plus-assoc simp-aImage sup-aImage.rep-eq)
  apply (smt2 bot-aImage.rep-eq less-eq-aImage.rep-eq simp-aImage zero-least-test)
  apply (smt2 less-eq-aImage.rep-eq one-greatest simp-aImage top-aImage.rep-eq)
  apply (metis (mono-tags, hide-lams) Rep-aImage-inject inf-aImage.rep-eq plus-distr-mult-left sup-aImage.rep-eq simp-aImage)
  apply (smt2 Rep-aImage-inject inf-aImage.rep-eq bot-aImage.rep-eq uminus-aImage.rep-eq zero-def simp-aImage)
  apply (smt2 Rep-aImage-inject sup-aImage.rep-eq top-aImage.rep-eq plus-compl uminus-aImage.rep-eq simp-aImage)
  apply (metis (mono-tags) Rep-aImage-inject inf-aImage.rep-eq minus-aImage.rep-eq uminus-aImage.rep-eq)
  done

end

class bounded-relative-antidomain-semiring = relative-antidomain-semiring + bounded-idempotent-left-semiring

begin

subclass bounded-relative-domain-semiring ..

lemma a-T: a(T) = 0
  by (metis a-dist-add a-one add-right-top mult-left-zero)

lemma d-T: d(T) = 1
  by (metis a-dist-add add-left-top d-def one-def zero-def)

lemma shunting-T-1: −p ; x ≤ y ⟶ x ≤ −−p ; T + y
  by (metis add-commutative case-split-left-add mult-right-isotone top-greatest)

lemma shunting-Z: −p ; x ≤ Z ⟷ x ≤ −−p ; T + Z
  apply rule
  apply (metis add-commutative case-split-left-add mult-right-isotone top-greatest)
  apply (smt a-T a-Z a-antitone a-dist-add a-export a-greatest-left-absorber add-commutative add-right-zero mult-left-one)
  done

lemma a-left-dist-add: −p ; (y + z) = −p ; y + −p ; z nitpick [expect=genuine,card=7] oops
lemma shunting-T: −p ; x ≤ y ⟷ x ≤ −−p ; T + y nitpick [expect=genuine,card=7] oops

end

class relative-left-zero-antidomain-semiring = relative-antidomain-semiring + idempotent-left-zero-semiring

begin

lemma kat-3: d(x) ; y ; a(z) = 0 ⟶ d(x) ; y = d(x) ; y ; d(z)
  by (metis add-left-zero d-def mult-left-dist-add mult-right-one plus-compl)

lemma a-a-below: a(a(x)) ; y ≤ y
  by (metis a-complement-one mult-left-one mult-right-sub-dist-add-left)

lemma kat-equiv-5: d(x) ; y ≤ y ; d(z) ⟷ d(x) ; y ; a(z) = d(x) ; y ; 0
proof
  assume d(x) ; y ≤ y ; d(z)
  thus d(x) ; y ; a(z) = d(x) ; y ; 0
    by (metis d-complement-zero kat-4-equiv mult-associative)
next
  assume d(x) ; y ; a(z) = d(x) ; y ; 0
  hence a(a(x)) ; y ; a(z) ≤ y ; a(a(z))
    by (smt2 a-a-below d-def mult-isotone zero-least)
  thus d(x) ; y ≤ y ; d(z)
    by (metis a-a-below a-complement-one case-split-right d-def mult-isotone order-refl)
```

qed

lemma *case-split-right-add*: $x \; ; -p \leq y \wedge x \; ; --p \leq z \longrightarrow x \leq y + z$
  by (*metis a-complement a-dist-add add-isotone mult-left-dist-add mult-right-one one-def plus-closed*)

end

class *bounded-relative-left-zero-antidomain-semiring* = *relative-left-zero-antidomain-semiring* + *bounded-idempotent-left-zero-semiring*

begin

lemma *shunting-T*: $-p \; ; x \leq y \longleftrightarrow x \leq --p \; ; T + y$
  apply *rule*
  apply (*metis add-commutative case-split-left-add mult-right-isotone top-greatest*)
  apply (*metis a-complement add-left-zero add-right-divisibility mult-associative mult-left-dist-add mult-left-one mult-left-zero mult-right-dist-add mult-right-isotone order-trans plus-left-one*)
  done

end

end

35 RelativeModal

theory *RelativeModal*

imports *RelativeDomain*

begin

class *relative-diamond-semiring* = *relative-domain-semiring* + *diamond* +
  assumes *diamond-def*: $|x{>}y = d(x\ ;\ y)$

begin

lemma *diamond-x-1*: $|x{>}1 = d(x)$
  by (*metis diamond-def mult-right-one*)

lemma *diamond-x-d*: $|x{>}d(y) = d(x\ ;\ y)$
  by (*metis d-mult-d diamond-def*)

lemma *diamond-x-und*: $|x{>}d(y) = |x{>}y$
  by (*metis diamond-def diamond-x-d*)

lemma *diamond-d-closed*: $|x{>}y = d(\ |x{>}y)$
  by (*metis d-fixpoint diamond-def*)

— Theorem 46.11

lemma *diamond-0-y*: $|0{>}y = 0$
  by (*metis d-zero diamond-def mult-left-zero*)

lemma *diamond-1-y*: $|1{>}y = d(y)$
  by (*metis diamond-def mult-left-one*)

— Theorem 46.12

lemma *diamond-1-d*: $|1{>}d(y) = d(y)$
  by (*metis diamond-1-y diamond-x-und*)

— Theorem 46.10

lemma *diamond-d-y*: $|d(x){>}y = d(x)\ ;\ d(y)$
  by (*metis d-export diamond-def*)

— Theorem 46.11

lemma *diamond-d-0*: $|d(x){>}0 = 0$
  by (*metis d-commutative diamond-0-y diamond-d-y diamond-x-1*)

— Theorem 46.12

lemma *diamond-d-1*: $|d(x){>}1 = d(x)$
  by (*metis diamond-d-closed diamond-x-1*)

lemma *diamond-d-d*: $|d(x){>}d(y) = d(x)\ ;\ d(y)$
  by (*metis d-mult-closed diamond-def*)

— Theorem 46.12

lemma *diamond-d-d-same*: $|d(x){>}d(x) = d(x)$
  by (*metis d-idempotent diamond-d-d*)

— Theorem 46.2

lemma *diamond-left-dist-add*: $|x + y{>}z = |x{>}z + |y{>}z$
  by (*metis d-dist-add diamond-def mult-right-dist-add*)

— Theorem 46.3

lemma *diamond-right-sub-dist-add*: $|x{>}y + |x{>}z \leq |x{>}(y + z)$

by (*smt add-associative d-plus-left-upper-bound diamond-def less-eq-def mult-left-sub-dist-add-left mult-left-sub-dist-add-right*)

— Theorem 46.4

lemma *diamond-associative*: $|x ; y>z = |x>(y ; z)$
by (*metis diamond-def mult-associative*)

— Theorem 46.4

lemma *diamond-left-mult*: $|x ; y>z = |x>|y>z$
by (*metis diamond-def diamond-x-d mult-associative*)

lemma *diamond-right-mult*: $|x>(y ; z) = |x>|y>z$
by (*metis diamond-associative diamond-left-mult*)

— Theorem 46.6

lemma *diamond-d-export*: $|d(x) ; y>z = d(x) ; |y>z$
by (*metis diamond-associative diamond-d-closed diamond-d-y diamond-right-mult*)

lemma *diamond-diamond-export*: $||x>y>z = |x>y ; |z>1$
by (*metis diamond-d-d diamond-def diamond-x-1 diamond-x-und*)

— Theorem 46.1

lemma *diamond-left-isotone*: $x \leq y \longrightarrow |x>z \leq |y>z$
by (*metis diamond-left-dist-add less-eq-def*)

— Theorem 46.1

lemma *diamond-right-isotone*: $y \leq z \longrightarrow |x>y \leq |x>z$
by (*metis d-isotone diamond-def mult-right-isotone*)

lemma *diamond-isotone*: $w \leq y \wedge x \leq z \longrightarrow |w>x \leq |y>z$
by (*metis diamond-left-isotone diamond-right-isotone order-trans*)

lemma *diamond-left-upper-bound*: $|x>y \leq |x+z>y$
by (*metis add-left-upper-bound diamond-left-dist-add*)

lemma *diamond-right-upper-bound*: $|x>y \leq |x>(y+z)$
by (*metis add-left-upper-bound diamond-right-isotone*)

lemma *diamond-lower-bound-right*: $|x>(d(y) ; d(z)) \leq |x>d(y)$
by (*metis d-mult-left-lower-bound diamond-right-isotone*)

lemma *diamond-lower-bound-left*: $|x>(d(y) ; d(z)) \leq |x>d(z)$
by (*metis d-commutative diamond-lower-bound-right*)

— Theorem 46.5

lemma *diamond-right-sub-dist-mult*: $|x>(d(y) ; d(z)) \leq |x>d(y) ; |x>d(z)$
by (*metis d-mult-greatest-lower-bound diamond-def diamond-lower-bound-left diamond-lower-bound-right*)

— Theorem 46.13

lemma *diamond-demodalisation-1*: $d(x) ; |y>z \leq Z \longleftrightarrow d(x) ; y ; d(z) \leq Z$
by (*smt d-strict diamond-associative diamond-right-mult diamond-x-1 diamond-x-und*)

— Theorem 46.14

lemma *diamond-demodalisation-3*: $|x>y \leq d(z) \longleftrightarrow x ; d(y) \leq d(z) ; x + Z$
apply *rule*
apply (*metis add-commutative add-right-isotone d-below-one d-restrict diamond-def diamond-x-und mult-left-isotone mult-right-isotone mult-right-one order-trans*)
apply (*smt add-commutative add-left-zero d-Z d-commutative d-dist-add d-involutive d-mult-sub d-plus-left-upper-bound diamond-d-y diamond-def diamond-x-und less-eq-def order-trans*)
done

— Theorem 46.6

lemma *diamond-d-export-2*: $|d(x) ; y{>}z = d(x) ; |d(x) ; y{>}z$
  by (*metis diamond-d-export diamond-left-mult d-idempotent*)

— Theorem 46.7

lemma *diamond-d-promote*: $|x ; d(y){>}z = |x ; d(y){>}(d(y) ; z)$
  by (*metis d-idempotent diamond-def mult-associative*)

— Theorem 46.8

lemma *diamond-d-import-iff*: $d(x) \leq |y{>}z \longleftrightarrow d(x) \leq |d(x) ; y{>}z$
  by (*metis diamond-d-export diamond-d-y d-order diamond-def eq-iff*)

— Theorem 46.9

lemma *diamond-d-import-iff-2*: $d(x) ; d(y) \leq |z{>}w \longleftrightarrow d(x) ; d(y) \leq |d(y) ; z{>}w$
  apply *rule*
  apply (*metis diamond-associative d-export d-mult-greatest-lower-bound diamond-def order.refl*)
  apply (*metis diamond-d-y d-mult-greatest-lower-bound diamond-def mult-associative*)
  done

end

class *relative-box-semiring* = *relative-diamond-semiring* + *relative-antidomain-semiring* + *box* +
  assumes *box-def*: $|x]y = a(x ; a(y))$

begin

— Theorem 47.1

lemma *box-diamond*: $|x]y = a(\ |x{>}a(y))$
  by (*metis box-def d-a-closed diamond-def*)

— Theorem 47.2

lemma *diamond-box*: $|x{>}y = a(\ |x]a(y))$
  by (*metis box-diamond d-def diamond-d-closed diamond-def diamond-x-d*)

lemma *box-x-0*: $|x]0 = a(x)$
  by (*metis box-def mult-right-one one-def*)

lemma *box-x-1*: $|x]1 = a(x ; 0)$
  by (*metis box-def one-compl*)

lemma *box-x-d*: $|x]d(y) = a(x ; a(y))$
  by (*metis box-def d-a-closed*)

lemma *box-x-und*: $|x]d(y) = |x]y$
  by (*metis box-def box-x-d*)

lemma *box-x-a*: $|x]a(y) = a(x ; y)$
  by (*metis a-mult-d box-def*)

— Theorem 47.15

lemma *box-0-y*: $|0]y = 1$
  by (*metis box-def mult-left-zero one-def*)

lemma *box-1-y*: $|1]y = d(y)$
  by (*metis box-def d-def mult-left-one*)

— Theorem 47.16

lemma *box-1-d*: $|1]d(y) = d(y)$
  by (*metis box-1-y d-involutive*)

lemma *box-1-a*: $|1]a(y) = a(y)$
  by (*metis a-d-closed box-1-y*)

lemma *box-d-y*: $|d(x)]y = a(x) + d(y)$
  by (*metis a-dist-add box-def box-x-a diamond-box diamond-x-1 mult-right-one plus-closed*)

lemma *box-a-y*: $|a(x)]y = d(x) + d(y)$
  by (*metis a-mult-deMorgan-1 box-def*)

— Theorem 47.14

lemma *box-d-0*: $|d(x)]0 = a(x)$
  by (*metis box-x-0 d-a-closed*)

lemma *box-a-0*: $|a(x)]0 = d(x)$
  by (*metis box-x-0 d-def*)

— Theorem 47.15

lemma *box-d-1*: $|d(x)]1 = 1$
  by (*metis box-diamond diamond-d-0 one-compl one-def*)

lemma *box-a-1*: $|a(x)]1 = 1$
  by (*metis box-x-1 bs-mult-right-zero one-def*)

— Theorem 47.13

lemma *box-d-d*: $|d(x)]d(y) = a(x) + d(y)$
  by (*metis box-d-y box-x-und*)

lemma *box-a-d*: $|a(x)]d(y) = d(x) + d(y)$
  by (*metis a-mult-deMorgan-1 box-x-d*)

lemma *box-d-a*: $|d(x)]a(y) = a(x) + a(y)$
  by (*metis a-export-d box-x-a*)

lemma *box-a-a*: $|a(x)]a(y) = d(x) + a(y)$
  by (*metis a-export-a box-x-a*)

— Theorem 47.15

lemma *box-d-d-same*: $|d(x)]d(x) = 1$
  by (*metis box-d-y d-a-closed d-def plus-compl*)

lemma *box-a-a-same*: $|a(x)]a(x) = 1$
  by (*metis box-def mult-compl one-def*)

— Theorem 47.16

lemma *box-d-below-box*: $d(x) \leq |d(y)]d(x)$
  by (*metis box-d-y box-x-und add-right-divisibility*)

lemma *box-d-closed*: $|x]y = d(|x]y)$
  by (*metis box-1-a box-1-y box-def*)

lemma *box-deMorgan-1*: $a(|x]y) = |x>a(y)$
  by (*metis box-def d-def diamond-def*)

lemma *box-deMorgan-2*: $a(|x>y) = |x]a(y)$
  by (*metis box-def diamond-box double-negation*)

— Theorem 47.5

lemma *box-left-dist-add*: $|x + y]z = |x]z \ ; |y]z$
  by (*metis a-dist-add box-def mult-right-dist-add*)

lemma *box-right-dist-add*: $|x](y + z) = a(x \ ; a(y) \ ; a(z))$
  by (*metis a-dist-add box-def mult-associative*)

lemma *box-associative*: $|x \ ; y]z = a(x \ ; y \ ; a(z))$
  by (*metis box-def*)

— Theorem 47.6

lemma *box-left-mult*: $|x\ ;\ y]z = |x]|y]z$
  by (*metis box-def box-x-a mult-associative*)

lemma *box-right-mult*: $|x](y\ ;\ z) = a(x\ ;\ a(y\ ;\ z))$
  by (*metis box-def*)

— Theorem 47.7

lemma *box-right-submult-d-d*: $|x](d(y)\ ;\ d(z)) \leq |x]d(y)\ ;\ |x]d(z)$
  by (*smt a-antitone a-dist-add a-export-d box-diamond d-a-closed diamond-def mult-left-sub-dist-add*)

lemma *box-right-submult-a-d*: $|x](a(y)\ ;\ d(z)) \leq |x]a(y)\ ;\ |x]d(z)$
  by (*metis box-d-closed box-right-submult-d-d box-x-0*)

lemma *box-right-submult-d-a*: $|x](d(y)\ ;\ a(z)) \leq |x]d(y)\ ;\ |x]a(z)$
  by (*metis box-a-0 box-left-mult box-right-submult-d-d box-x-0 box-x-und*)

lemma *box-right-submult-a-a*: $|x](a(y)\ ;\ a(z)) \leq |x]a(y)\ ;\ |x]a(z)$
  by (*metis a-d-closed box-right-submult-a-d*)

— Theorem 47.8

lemma *box-d-export*: $|d(x)\ ;\ y]z = a(x) + |y]z$
  by (*metis a-d-closed box-d-y box-def box-left-mult*)

lemma *box-a-export*: $|a(x)\ ;\ y]z = d(x) + |y]z$
  by (*metis a-d-closed box-d-a box-def box-left-mult d-def*)

— Theorem 47.4

lemma *box-left-antitone*: $y \leq x \longrightarrow |x]z \leq |y]z$
  by (*metis a-antitone box-def mult-left-isotone*)

— Theorem 47.3

lemma *box-right-isotone*: $y \leq z \longrightarrow |x]y \leq |x]z$
  by (*metis a-antitone box-def mult-right-isotone*)

lemma *box-antitone-isotone*: $y \leq w \wedge x \leq z \longrightarrow |w]x \leq |y]z$
  by (*metis box-left-antitone box-right-isotone order-trans*)

lemma *diamond-1-a*: $|1{>}a(y) = a(y)$
  by (*metis a-d-closed diamond-1-y*)

lemma *diamond-a-y*: $|a(x){>}y = a(x)\ ;\ d(y)$
  by (*metis a-mult-closed d-def d-mult-d diamond-def*)

lemma *diamond-a-0*: $|a(x){>}0 = 0$
  by (*metis box-a-1 box-deMorgan-1 one-compl*)

lemma *diamond-a-1*: $|a(x){>}1 = a(x)$
  by (*metis a-d-closed diamond-x-1*)

lemma *diamond-a-d*: $|a(x){>}d(y) = a(x)\ ;\ d(y)$
  by (*metis diamond-a-y diamond-x-und*)

lemma *diamond-d-a*: $|d(x){>}a(y) = d(x)\ ;\ a(y)$
  by (*metis a-d-closed diamond-d-y*)

lemma *diamond-a-a*: $|a(x){>}a(y) = a(x)\ ;\ a(y)$
  by (*metis a-mult-closed diamond-def*)

lemma *diamond-a-a-same*: $|a(x){>}a(x) = a(x)$
  by (*metis a-idempotent diamond-a-a*)

lemma *diamond-a-export*: $|a(x)\ ;\ y{>}z = a(x)\ ;\ |y{>}z$
  by (*metis diamond-a-a diamond-box diamond-left-mult*)

lemma *a-box-a-a*: $a(p)$; $|a(p)]a(q) = a(p)$; $a(q)$
  by (*metis box-x-a double-negation mult-compl-intro plus-def*)

lemma *box-left-lower-bound*: $|x+y]z \le |x]z$
  by (*metis add-left-upper-bound box-left-antitone*)

lemma *box-right-upper-bound*: $|x]y \le |x](y+z)$
  by (*metis add-left-upper-bound box-right-isotone*)

lemma *box-lower-bound-right*: $|x](d(y)$; $d(z)) \le |x]d(y)$
  by (*metis box-right-isotone d-mult-left-lower-bound*)

lemma *box-lower-bound-left*: $|x](d(y)$; $d(z)) \le |x]d(z)$
  by (*metis box-lower-bound-right d-commutative*)

— Theorem 47.9

lemma *box-d-import*: $d(x)$; $|y]z = d(x)$; $|d(x)$; $y]z$
  by (*metis a-box-a-a box-left-mult box-def d-def*)

— Theorem 47.10

lemma *box-d-promote*: $|x$; $d(y)]z = |x$; $d(y)](d(y)$; $z)$
  by (*metis a-box-a-a box-left-mult a-mult-d box-def d-def*)

— Theorem 47.11

lemma *box-d-import-iff*: $d(x) \le |y]z \longleftrightarrow d(x) \le |d(x)$; $y]z$
  by (*metis box-d-closed box-d-import d-order*)

— Theorem 47.12

lemma *box-d-import-iff-2*: $d(x)$; $d(y) \le |z]w \longleftrightarrow d(x)$; $d(y) \le |d(y)$; $z]w$
  apply *rule*
  apply (*metis box-d-import d-commutative d-restrict-iff-1*)
  apply (*smt2 box-d-closed box-d-import d-mult-closed d-order mult-associative*)
  done

— Theorem 47.20

lemma *box-demodalisation-2*: $-p \le |y](-q) \longleftrightarrow -p$; y ; $--q \le Z$
  by (*metis a-greatest-left-absorber box-def mult-associative*)

lemma *box-right-sub-dist-add*: $|x]d(y) + |x]d(z) \le |x](d(y) + d(z))$
  by (*metis add-commutative add-least-upper-bound box-right-upper-bound*)

lemma *box-diff-var*: $|x](d(y) + a(z))$; $|x]d(z) \le |x]d(z)$
  by (*metis box-def lower-bound-right*)

— Theorem 47.19

lemma *diamond-demodalisation-2*: $|x>y \le d(z) \longleftrightarrow a(z)$; x ; $d(y) \le Z$
  by (*metis a-mult-d box-def d-a-shunting-zero d-strict diamond-a-y diamond-box diamond-x-1 mult-associative mult-right-one sub-comm*)

— Theorem 47.17

lemma *box-below-Z*: ($|x]y)$; x ; $a(y) \le Z$
  by (*metis a-restrict box-def mult-associative*)

— Theorem 47.18

lemma *box-partial-correctness*: $|x]1 = 1 \longleftrightarrow x$; $0 \le Z$
  by (*metis box-x-a a-strict one-def*)

lemma *diamond-split*: $|x>y = d(z)$; $|x>y + a(z)$; $|x>y$
  by (*metis a-export-d a-restrict add-commutative d-def d-strict mult-left-one mult-right-dist-add one-def*)

lemma *box-import-shunting*: $-p;-q \le |x](-r) \longleftrightarrow -q \le |-p;x](-r)$

by (*smt box-demodalisation-2 mult-associative sub-comm sub-mult-closed*)

lemma *box-dist-mult*: $|x](d(y) ; d(z)) = |x](d(y)) ; |x](d(z))$ **nitpick** [*expect=genuine,card=6*] **oops**
lemma *box-demodalisation-3*: $d(x) \le |y]d(z) \longrightarrow d(x) ; y \le y ; d(z) + Z$ **nitpick** [*expect=genuine,card=6*] **oops**
lemma *fbox-diff*: $|x](d(y) + a(z)) \le |x]y + a(|x]z)$ **nitpick** [*expect=genuine,card=6*] **oops**
lemma *diamond-diff*: $|x>y ; a(|x>z) \le |x>(d(y) ; a(z))$ **nitpick** [*expect=genuine,card=6*] **oops**
lemma *diamond-diff-var*: $|x>d(y) \le |x>(d(y) ; a(z)) + |x>d(z)$ **nitpick** [*expect=genuine,card=6*] **oops**

end

class *relative-left-zero-diamond-semiring* = *relative-diamond-semiring* + *relative-domain-semiring* + *idempotent-left-zero-semiring*

begin

lemma *diamond-right-dist-add*: $|x>(y + z) = |x>y + |x>z$
 by (*metis d-dist-add diamond-def mult-left-dist-add*)

end

class *relative-left-zero-box-semiring* = *relative-box-semiring* + *relative-left-zero-antidomain-semiring*

begin

subclass *relative-left-zero-diamond-semiring* **..**

lemma *box-right-mult-d-d*: $|x](d(y) ; d(z)) = |x]d(y) ; |x]d(z)$
 by (*smt a-dist-add box-x-a diamond-box diamond-x-1 mult-left-dist-add*)

lemma *box-right-mult-a-d*: $|x](a(y) ; d(z)) = |x]a(y) ; |x]d(z)$
 by (*metis box-d-closed box-right-mult-d-d box-x-0*)

lemma *box-right-mult-d-a*: $|x](d(y) ; a(z)) = |x]d(y) ; |x]a(z)$
 by (*metis box-a-0 box-left-mult box-right-mult-d-d box-x-0 box-x-und*)

lemma *box-right-mult-a-a*: $|x](a(y) ; a(z)) = |x]a(y) ; |x]a(z)$
 by (*metis a-dist-add box-x-a mult-left-dist-add*)

lemma *box-demodalisation-3*: $d(x) \le |y]d(z) \longrightarrow d(x) ; y \le y ; d(z) + Z$
proof –
 have $d(x) \le |y]d(z) \longrightarrow d(x) ; y ; a(z) \le Z$
  by (*metis mult-left-isotone a-mult-d a-restrict box-def d-def mult-associative order-trans*)
 thus *?thesis*
  by (*metis add-commutative case-split-right-add d-def d-restrict-iff-1 eq-refl mult-associative*)
qed

lemma *fbox-diff*: $|x](d(y) + a(z)) \le |x]y + a(|x]z)$
 by (*smt a-compl-intro a-dist-add a-mult-d a-plus-left-lower-bound add-commutative box-def d-def mult-left-dist-add shunting*)

lemma *diamond-diff-var*: $|x>d(y) \le |x>(d(y) ; a(z)) + |x>d(z)$
 by (*smt2 a-dist-add add-commutative box-def box-right-mult-a-a diamond-box diamond-right-upper-bound diamond-x-1 double-negation mult-compl-intro mult-right-one one-def plus-closed sub-comm*)

lemma *diamond-diff*: $|x>y ; a(|x>z) \le |x>(d(y) ; a(z))$
 by (*metis d-a-shunting d-involutive diamond-def diamond-diff-var diamond-x-und*)

end

end

36 CompleteDomain

theory *CompleteDomain*

imports *RelativeDomain CompleteTests*

begin

class *complete-antidomain-semiring* = *relative-antidomain-semiring* + *complete-tests* +
  assumes *a-dist-Sum*: *ascending-chain f* $\longrightarrow$ $-$(*Sum f*) = *Prod* (λn . $-f$ n)
  assumes *a-dist-Prod*: *descending-chain f* $\longrightarrow$ $-$(*Prod f*) = *Sum* (λn . $-f$ n)

begin

lemma *a-ascending-chain*: *ascending-chain f* $\longrightarrow$ *descending-chain* (λn . $-f$ n)
  by (*smt ascending-chain-def descending-chain-def a-antitone*)

lemma *a-descending-chain*: *descending-chain f* $\longrightarrow$ *ascending-chain* (λn . $-f$ n)
  by (*smt ascending-chain-def descending-chain-def a-antitone*)

lemma *d-dist-Sum*: *ascending-chain f* $\longrightarrow$ *d*(*Sum f*) = *Sum* (λn . *d*(*f n*))
  unfolding *d-def*
  apply (*metis a-dist-Sum a-dist-Prod a-ascending-chain*)
  done

lemma *d-dist-Prod*: *descending-chain f* $\longrightarrow$ *d*(*Prod f*) = *Prod* (λn . *d*(*f n*))
  unfolding *d-def*
  apply (*metis a-dist-Sum a-dist-Prod a-descending-chain*)
  done

end

end

37 HoareModal

```
theory HoareModal

imports CompleteDomain Hoare KleeneAlgebra RelativeModal

begin

class box-precondition = relative-box-semiring + pre +
  assumes pre-def: x«p = |x]p

begin

— Theorem 47

subclass precondition
  apply unfold-locales
  apply (metis box-def double-negation pre-def)
  apply (metis box-left-mult pre-def)
  apply (metis a-dist-add box-deMorgan-2 box-right-submult-a-a greatest-lower-bound pre-def)
  apply (metis box-1-a pre-def reflexive)
  done

subclass precondition-test-test
  apply unfold-locales
  apply (metis a-box-a-a pre-def)
  done

subclass precondition-promote
  apply unfold-locales
  apply (metis box-def box-x-a pre-def pre-test-test)
  done

subclass precondition-test-box
  apply unfold-locales
  apply (metis box-a-a d-def pre-def)
  done

lemma pre-Z: −p ≤ x«−q ⟷ −p ; x ; −−q ≤ Z
  by (metis box-demodalisation-2 pre-def)

lemma pre-left-dist-add: x+y«−q = (x«−q) ; (y«−q)
  by (metis box-left-dist-add pre-def)

lemma pre-left-antitone: x ≤ y ⟶ y«−q ≤ x«−q
  by (metis box-left-antitone pre-def)

lemma pre-promote-neg: (x«−q) ; x ; −−q ≤ Z
  by (metis order-refl pre-Z pre-closed)

lemma pre-pc-Z: x«1 = 1 ⟷ x ; 0 ≤ Z
  by (metis a-strict box-x-1 pre-def)

lemma pre-sub-promote: (x«−q) ; x ≤ (x«−q) ; x ; −q + Z nitpick [expect=genuine,card=6] oops
lemma pre-promote: (x«−q) ; x + Z = (x«−q) ; x ; −q + Z nitpick [expect=genuine,card=6] oops
lemma pre-mult-sub-promote: (x;y«−q) ; x ≤ (x;y«−q) ; x ; (y«−q) + Z nitpick [expect=genuine,card=6] oops
lemma pre-mult-promote: (x;y«−q) ; x ; (y«−q) + Z = (x;y«−q) ; x + Z nitpick [expect=genuine,card=6] oops

end

class left-zero-box-precondition = box-precondition + relative-left-zero-antidomain-semiring

begin

lemma pre-sub-promote: (x«−q) ; x ≤ (x«−q) ; x ; −q + Z
  by (metis case-split-right-add order-refl pre-Z pre-closed)

lemma pre-promote: (x«−q) ; x + Z = (x«−q) ; x ; −q + Z
  by (smt a-below-one add-left-upper-bound add-same-context mult-right-isotone mult-right-one order-trans pre-sub-promote)
```

lemma *pre-mult-sub-promote*: $(x;y \ll -q) ; x \le (x;y \ll -q) ; x ; (y \ll -q) + Z$
  by (*metis pre-closed pre-seq pre-sub-promote*)

lemma *pre-mult-promote-sub*: $(x;y \ll -q) ; x ; (y \ll -q) \le (x;y \ll -q) ; x$
  by (*metis mult-right-isotone mult-right-one pre-below-one*)

lemma *pre-mult-promote*: $(x;y \ll -q) ; x ; (y \ll -q) + Z = (x;y \ll -q) ; x + Z$
  by (*metis add-left-upper-bound add-same-context order-trans pre-mult-sub-promote pre-mult-promote-sub*)

end

class *diamond-precondition* = *relative-box-semiring* + *pre* +
  assumes *pre-def*: $x \ll p = |x>p$

begin

— Theorem 47

subclass *precondition*
  apply *unfold-locales*
  apply (*metis d-def diamond-d-closed pre-def*)
  apply (*metis diamond-left-mult pre-def*)
  apply (*smt diamond-right-isotone lower-bound-right pre-def*)
  apply (*metis diamond-1-a pre-def reflexive*)
  done

subclass *precondition-test-test*
  apply *unfold-locales*
  apply (*metis diamond-a-a-same diamond-a-export diamond-associative diamond-right-mult pre-def*)
  done

subclass *precondition-promote*
  apply *unfold-locales*
  apply (*metis box-deMorgan-1 diamond-a-a pre-def pre-test-test*)
  done

subclass *precondition-test-diamond*
  apply *unfold-locales*
  apply (*metis diamond-a-a pre-def*)
  done

lemma *pre-left-dist-add*: $x+y \ll -q = (x \ll -q) + (y \ll -q)$
  by (*metis d-dist-add diamond-def mult-right-dist-add pre-def*)

lemma *pre-left-isotone*: $x \le y \longrightarrow x \ll -q \le y \ll -q$
  by (*metis diamond-left-isotone pre-def*)

end

class *box-while* = *box-precondition* + *bounded-left-conway-semiring* + *ite* + *while* +
  assumes *ite-def*: $x \lhd p \rhd y = p ; x + -p ; y$
  assumes *while-def*: $p \star x = (p ; x)^\circ ; -p$

begin

subclass *bounded-relative-antidomain-semiring* **..**

lemma *Z-circ-left-zero*: $Z ; x^\circ = Z$
  by (*metis Z-left-zero-above-one circ-reflexive*)

subclass *ifthenelse*
  apply *unfold-locales*
  apply (*smt a-d-closed box-a-export box-left-dist-add box-x-a case-duality d-def ite-def pre-def*)
  done

— Theorem 48.1

subclass *whiledo*
  apply *unfold-locales*

apply (*smt circ-loop-fixpoint ite-def ite-pre mult-associative mult-right-one pre-one pre-seq while-def*)
apply (*metis pre-mult-test-promote while-def*)
done

lemma *pre-while-1*: $-p;(-p\star x)«1 = -p\star x«1$
proof −
have $--p;(-p;(-p\star x)«1) = --p;(-p\star x«1)$
by (*metis a-mult-left-upper-bound box-def bs-mult-right-one leq-def mult-associative one-def pre-def while-pre-else*)
thus *?thesis*
by (*smt eq-cases one-def pre-closed pre-import*)
qed

lemma *aL-one-circ*: $aL = a(1^\circ;0)$
by (*metis a-one box-0-y box-left-mult box-x-1 mult-left-one pre-def while-def aL-def*)

end

class *diamond-while* = *diamond-precondition* + *bounded-left-conway-semiring* + *ite* + *while* +
assumes *ite-def*: $x\lhd p\rhd y = p\ ;\ x + -p\ ;\ y$
assumes *while-def*: $p\star x = (p\ ;\ x)^\circ\ ;\ -p$

begin

subclass *bounded-relative-antidomain-semiring* **..**

lemma *Z-circ-left-zero*: $Z\ ;\ x^\circ = Z$
by (*metis Z-left-zero-above-one circ-reflexive*)

subclass *ifthenelse*
apply *unfold-locales*
apply (*metis ite-def pre-def diamond-left-dist-add diamond-a-export*)
done

— Theorem 48.2

subclass *whiledo*
apply *unfold-locales*
apply (*smt circ-loop-fixpoint ite-def ite-pre mult-associative mult-right-one pre-one pre-seq while-def*)
apply (*metis pre-mult-test-promote while-def*)
done

lemma *aL-one-circ*: $aL = d(1^\circ;0)$
by (*metis aL-def a-one diamond-x-1 mult-left-one pre-def while-def*)

end

class *box-while-program* = *box-while* + *atoms*

begin

subclass *while-program* **..**

end

class *diamond-while-program* = *diamond-while* + *atoms*

begin

subclass *while-program* **..**

end

class *box-hoare-calculus* = *box-while-program* + *complete-antidomain-semiring*

begin

subclass *hoare-calculus* **..**

end

```
class diamond-hoare-calculus = diamond-while-program + complete-antidomain-semiring

begin

subclass hoare-calculus ..

end

class box-hoare-sound = box-hoare-calculus + relative-domain-semiring-split + left-kleene-conway-semiring +
  assumes aL-circ: aL ; x° ≤ x⋆

begin

lemma aL-circ-ext: |x⋆]y ≤ |aL ; x°]y
  by (metis aL-circ box-left-antitone)

lemma box-star-induct: −p ≤ |x](−p) ⟶ −p ≤ |x⋆](−p)
proof
  assume −p ≤ |x](−p)
  hence 1: x;−−p;T ≤ Z + −−p;T
    by (metis Z-top add-commutative box-demodalisation-2 mult-associative mult-left-isotone shunting-Z)
  have x;(Z + −−p;T) ≤ x;−−p;T + Z
    by (smt add-commutative mult-associative split-Z)
  also have ... ≤ Z + −−p;T using 1
    by (smt add-commutative add-least-upper-bound add-right-upper-bound)
  finally have x;(Z + −−p;T) + −−p ≤ Z + −−p;T
    by (smt add-commutative add-least-upper-bound mult-left-sub-dist-add order-trans split-Z top-right-mult-increasing)
  thus −p ≤ |x⋆](−p)
    by (metis add-commutative box-demodalisation-2 mult-associative shunting-Z star-left-induct)
qed

lemma box-circ-induct: −p ≤ |x](−p) ⟶ −p;aL ≤ |x°](−p)
  by (smt aL-circ-ext aL-test box-left-mult box-star-induct order-trans plus-comm pre-closed pre-def pre-test shunting-right)

lemma a-while-soundness: −p;−q ≤ |x](−q) ⟶ aL;−q ≤ |(−p;x)°;−−p](−q)
proof −
  have |(−p;x)°](−q) ≤ |(−p;x)°;−−p](−q)
    by (smt add-right-upper-bound box-def box-right-dist-add box-right-isotone)
  thus ?thesis
    by (smt box-import-shunting box-circ-induct order-trans sub-comm aL-test)
qed

subclass hoare-calculus-sound
  apply unfold-locales
  apply (metis a-while-soundness while-def pre-def)
  done

end

class diamond-hoare-sound = diamond-hoare-calculus + left-kleene-conway-semiring +
  assumes aL-circ: aL ; x° ≤ x⋆

begin

lemma aL-circ-equal: aL ; x° = aL ; x⋆
  by (smt aL-circ aL-one-circ antisym d-restrict-iff-1 mult-right-isotone star-below-circ)

lemma aL-zero: aL = 0
  by (smt aL-circ-equal aL-one-circ d-export d-idempotent diamond-d-0 diamond-def mult-associative mult-right-one star-one)

subclass hoare-calculus-sound
  apply unfold-locales
  apply (metis aL-zero bs-mult-left-zero zero-least)
  done

end

class box-hoare-complete = box-hoare-calculus + left-kleene-conway-semiring +
  assumes box-circ-induct-2: −p;|x](−q) ≤ −q ⟶ |x°](−p) ≤ −q+aL
```

assumes *aL-zero-or-one*: $aL = 0 \vee aL = 1$
assumes *while-mult-left-dist-Prod*: $x \in$ *While-program* $\wedge$ *descending-chain* $t \wedge$ *test-seq* $t \longrightarrow x;Prod\ t = Prod\ (\lambda n\ .\ x;t\ n)$

begin

subclass *hoare-calculus-complete*
apply *unfold-locales*
prefer *3*
apply (*smt box-circ-induct-2 double-negation least-upper-bound lower-bound-left mult-distr-plus-right pre-closed pre-def pre-import pre-seq pre-test sub-mult-closed while-def*)
apply (*metis aL-zero-or-one bs-mult-right-zero mult-right-one order-refl pre-closed zero-least*)
unfolding *pre-def box-def*
apply (*metis a-ascending-chain a-dist-Prod a-dist-Sum descending-chain-left-mult while-mult-left-dist-Prod test-seq-def*)
done

end

class *diamond-hoare-complete* = *diamond-hoare-calculus* + *relative-domain-semiring-split* + *left-kleene-conway-semiring* +
assumes *dL-circ*: $-aL;x^\circ \leq x^\star$
assumes *aL-zero-or-one*: $aL = 0 \vee aL = 1$
assumes *while-mult-left-dist-Sum*: $x \in$ *While-program* $\wedge$ *ascending-chain* $t \wedge$ *test-seq* $t \longrightarrow x;Sum\ t = Sum\ (\lambda n\ .\ x;t\ n)$

begin

lemma *diamond-star-induct-var*: $|x\rangle(d\ p) \leq d\ p \longrightarrow |x^\star\rangle(d\ p) \leq d\ p$
proof
assume $|x\rangle(d\ p) \leq d\ p$
hence $x\ ;\ (d\ p\ ;\ x^\star + Z) \leq d\ p\ ;\ x\ ;\ x^\star + Z\ ;\ x^\star + Z$
by (*metis add-left-isotone d-mult-d diamond-def diamond-demodalisation-3 mult-associative mult-left-isotone mult-right-dist-add order-trans split-Z*)
also have ... $\leq d\ p\ ;\ x^\star + Z$
by (*smt Z-mult-decreasing add-associative add-left-isotone less-eq-def mult-associative mult-right-isotone star.left-plus-below-circ*)
finally show $|x^\star\rangle(d\ p) \leq d\ p$
by (*smt add-commutative add-least-upper-bound add-right-upper-bound d-mult-d diamond-def diamond-demodalisation-3 order-trans star.circ-back-loop-prefixpoint star-left-induct*)
qed

lemma *diamond-star-induct*: $d\ q + |x\rangle(d\ p) \leq d\ p \longrightarrow |x^\star\rangle(d\ q) \leq d\ p$
by (*metis add-least-upper-bound diamond-star-induct-var diamond-right-isotone order-trans*)

lemma *while-completeness-1*: $-p;(x«-q) \leq -q \longrightarrow -p\star x«-q \leq -q+aL$
proof
assume $-p;(x«-q) \leq -q$
hence $--p;-q + |-p;x\rangle(-q) \leq -q$
by (*metis add-least-upper-bound diamond-a-export lower-bound-right pre-def*)
hence $|(-p;x)^\star\rangle(--p;-q) \leq -q$
by (*smt diamond-star-induct d-def sub-mult-closed double-negation*)
hence $|-aL;(-p;x)^\circ\rangle(--p;-q) \leq -q$
by (*smt dL-circ diamond-left-isotone order-trans*)
thus $-p\star x«-q \leq -q+aL$
by (*smt aL-test diamond-a-export diamond-def mult-associative plus-comm pre-closed pre-def shunting while-def*)
qed

subclass *hoare-calculus-complete*
apply *unfold-locales*
prefer *3*
apply (*rule while-completeness-1*)
apply (*metis aL-zero-or-one bs-mult-right-zero mult-right-one order-refl pre-closed zero-least*)
unfolding *pre-def diamond-def*
apply (*metis while-mult-left-dist-Sum d-dist-Sum ascending-chain-left-mult*)
done

end

class *box-hoare-valid* = *box-hoare-sound* + *box-hoare-complete* + *hoare-triple* +
assumes *hoare-triple-def*: $p⦃x⦄q \longleftrightarrow p \leq |x]q$

begin

— Theorem 49.2

subclass *hoare-calculus-valid*
  apply *unfold-locales*
  apply (*metis hoare-triple-def pre-def*)
  done

lemma *rule-skip-valid*: $-p\{\!|1|\!\}-p$
  by (*metis box-1-a hoare-triple-def reflexive*)

end

class *diamond-hoare-valid* = *diamond-hoare-sound* + *diamond-hoare-complete* + *hoare-triple* +
  assumes *hoare-triple-def*: $p\{\!|x|\!\}q \longleftrightarrow p \leq |x>q$

begin

lemma *circ-star-equal*: $x^\circ = x^\star$
  by (*metis aL-zero antisym dL-circ mult-left-one one-def star-below-circ*)

— Theorem 49.1

subclass *hoare-calculus-valid*
  apply *unfold-locales*
  apply (*metis hoare-triple-def pre-def*)
  done

end

class *diamond-hoare-sound-2* = *diamond-hoare-calculus* + *left-kleene-conway-semiring* +
  assumes *diamond-circ-induct-2*: $--p;-q \leq |x>(-q) \longrightarrow aL;-q \leq |x^\circ>(-p)$

begin

subclass *hoare-calculus-sound*
  apply *unfold-locales*
    apply (*smt a-export diamond-associative diamond-circ-induct-2 double-negation mult-compl-intro pre-def pre-import-equiv-mult sub-comm sub-mult-closed while-def*)
  done

end

class *diamond-hoare-valid-2* = *diamond-hoare-sound-2* + *diamond-hoare-complete* + *hoare-triple* +
  assumes *hoare-triple-def*: $p\{\!|x|\!\}q \longleftrightarrow p \leq |x>q$

begin

subclass *hoare-calculus-valid*
  apply *unfold-locales*
  apply (*metis hoare-triple-def pre-def*)
  done

end

end

38 PrePostModal

theory *PrePostModal*

imports *PrePost HoareModal*

begin

class *pre-post-spec-whiledo* = *pre-post-spec-greatest* + *whiledo*

begin

lemma *nat-test-pre-post*: *nat-test t s* $\wedge$ $-q \leq s$ $\wedge$ $(\forall n\ .\ x \leq t\ n;-p;-q\dashv(pSum\ t\ n;-q)) \longrightarrow -p\star x \leq -q\dashv--p;-q$
  by (*smt nat-test-def nat-test-pre pSum-test-nat pre-post-galois sub-mult-closed*)

lemma *nat-test-pre-post-2*: *nat-test t s* $\wedge$ $-r \leq s$ $\wedge$ $(\forall n\ .\ x \leq t\ n;-p\dashv(pSum\ t\ n)) \longrightarrow -p\star x \leq -r\dashv 1$
  by (*smt nat-test-def nat-test-pre-2 one-def pSum-test-nat pre-post-galois sub-mult-closed*)

end

class *pre-post-spec-hoare* = *pre-post-spec-whiledo* + *hoare-calculus-sound*

begin

lemma *pre-post-while*: $x \leq -p;-q\dashv-q \longrightarrow -p\star x \leq aL;-q\dashv-q$
  by (*smt aL-test pre-post-galois sub-mult-closed while-soundness*)

— Theorem 43.1

lemma *while-soundness-3*: *test-seq t* $\wedge$ $-q \leq Sum\ t$ $\wedge$ $x \leq t\ 0;-p;-q\dashv aL;-q$ $\wedge$ $(\forall n{>}0\ .\ x \leq t\ n;-p;-q\dashv pSum\ t\ n;-q) \longrightarrow$
$-p\star x \leq -q\dashv--p;-q$
  by (*smt aL-test pSum-test plus-closed pre-post-galois sub-mult-closed test-seq-def while-soundness-1*)

— Theorem 43.2

lemma *while-soundness-4*: *test-seq t* $\wedge$ $-r \leq Sum\ t$ $\wedge$ $(\forall n\ .\ x \leq t\ n;-p\dashv pSum\ t\ n) \longrightarrow -p\star x \leq -r\dashv 1$
  by (*smt one-def pSum-test pre-post-galois sub-mult-closed test-seq-def while-soundness-2*)

end

class *pre-post-spec-hoare-pc-2* = *pre-post-spec-hoare* + *hoare-calculus-pc-2*

begin

— Theorem 43.3

lemma *pre-post-while-pc*: $x \leq -p;-q\dashv-q \longrightarrow -p\star x \leq -q\dashv--p;-q$
  by (*metis pre-post-galois sub-mult-closed while-soundness-pc*)

end

class *pre-post-spec-hoare-pc* = *pre-post-spec-hoare* + *hoare-calculus-pc*

begin

subclass *pre-post-spec-hoare-pc-2* **..**

lemma *pre-post-one-one-top*: $1\dashv 1 = T$
  by (*metis add-left-top less-eq-def pre-one-one pre-post-one-one*)

end

class *pre-post-spec-H* = *pre-post-spec-greatest* + *box-precondition* + *havoc* +
  assumes *H-zero-2*: $H\ ;\ 0 = 0$
  assumes *H-split-2*: $x \leq x\ ;\ -q\ ;\ T + H\ ;\ --q$

begin

subclass *idempotent-left-semiring-H*

apply *unfold-locales*
apply (*rule H-zero-2*)
apply (*metis H-split-2 a-one mult-associative mult-left-zero mult-right-one one-def*)
done

lemma *pre-post-def-iff*: $-p ; x ; --q \leq Z \longleftrightarrow x \leq Z + --p ; T + H ; -q$
proof (*rule iffI*)
assume $-p ; x ; --q \leq Z$
hence $x ; --q ; T \leq Z + --p ; T$
by (*smt Z-left-zero-above-one case-split-left-add mult-associative mult-left-isotone mult-right-dist-add mult-right-isotone top-greatest top-mult-top*)
thus $x \leq Z + --p ; T + H ; -q$
by (*metis add-left-isotone order-trans H-split-2 double-negation*)
next
assume $x \leq Z + --p ; T + H ; -q$
hence $-p ; x ; --q \leq -p ; (Z ; --q + --p ; T ; --q + H ; -q ; --q)$
by (*metis mult-isotone reflexive mult-associative mult-right-dist-add*)
thus $-p ; x ; --q \leq Z$
by (*metis H-zero-2 Z-mult-decreasing add-commutative add-left-zero mult-associative mult-right-dist-add mult-right-isotone order-trans test-mult-left-dist-shunt test-mult-left-sub-dist-shunt zero-def*)
qed

lemma *pre-post-def*: $-p\dashv-q = Z + --p;T + H;-q$
by (*metis eq-iff pre-Z pre-post-def-iff pre-post-galois*)

end

class *pre-post-L* = *pre-post-spec-greatest* + *box-while* + *left-conway-semiring-L* + *left-kleene-conway-semiring* +
assumes *circ-below-L-add-star*: $x^\circ \leq L + x^\star$

begin

— a loop does not abort if its body does not abort
— this avoids abortion from all states; alternatively from states in -r if -r is an invariant

lemma *body-abort-loop*: $Z = L \wedge x \leq -p\dashv 1 \longrightarrow -p\star x \leq 1\dashv 1$
proof
assume *1*: $Z = L \wedge x \leq -p\dashv 1$
hence $-p ; x ; 0 \leq L$
by (*metis a-one one-def pre-Z pre-post-galois*)
hence $(-p ; x)^\star ; 0 \leq L$
by (*metis L-split add-left-zero less-eq-def star-left-induct*)
hence $(-p ; x)^\circ ; 0 \leq L$
by (*smt L-left-zero L-split add-commutative circ-below-L-add-star less-eq-def mult-right-dist-add*)
thus $-p\star x \leq 1\dashv 1$ **using** *1*
by (*metis a-one a-strict box-def bs-mult-right-zero mult-associative pre-def pre-post-one-one while-def*)
qed

end

class *pre-post-spec-Hd* = *pre-post-spec-least* + *diamond-precondition* + *idempotent-left-semiring-Hd* +
assumes *d-mult-top*: $d(x) ; T = x ; T$

begin

subclass *pre-post-spec-least-Hd*
apply *unfold-locales*
apply (*metis d-mult-top diamond-x-1 pre-def*)
done

end

end

39 MonotonicBooleanTransformers

theory *MonotonicBooleanTransformers*

imports *Base MonoBoolTranAlgebra/Assertion-Algebra*

begin

— This theory requires LatticeProperties and MonoBoolTranAlgebra from the Archive of Formal Proofs.

context *mbt-algebra*

begin

lemma *directed-left-mult*: *directed* $Y \longrightarrow$ *directed* (*op* ; *x* ‘ *Y*)
  unfolding *directed-def*
  apply *simp*
  apply (*metis le-comp*)
  done

lemma *neg-assertion*: *neg-assert* $x \in$ *assertion*
  unfolding *assertion-def*
  apply *rule*
   apply (*smt dual-comp dual-dual dual-neg dual-one dual-sup dual-top inf-commute inf-le2 inf-sup-distrib1 mult.assoc mult.left-neutral neg-assert-def sup-bot-left sup-comp top-comp*)
  done

lemma *assertion-neg-assert*: $x \in$ *assertion* $\longleftrightarrow x =$ *neg-assert* (*neg-assert* x)
  by (*metis neg-assertion uminus-uminus*)

— extend and dualise part of Viorel Preoteasa's theory

definition
  assumption $= \{x\ .\ 1 \leq x \wedge (x * \bot) \sqcup (x\ \hat{}\ o) = x\}$

definition
  neg-assume $(x{::}'a) = (x\ \hat{}\ o * top) \sqcup 1$

lemma *neg-assume-assert*: *neg-assume* $x =$ (*neg-assert* $(x\ \hat{}\ o)$) $\hat{}\ o$
  by (*metis dual-bot dual-comp dual-dual dual-inf dual-one neg-assert-def neg-assume-def*)

lemma *assert-iff-assume*: $x \in$ *assertion* $\longleftrightarrow x\ \hat{}\ o \in$ *assumption*
  by (*smt assertion-def assumption-def dual-bot dual-comp dual-dual dual-inf dual-le dual-one mem-Collect-eq*)

lemma *assertion-iff-assumption-subseteq*: $X \subseteq$ *assertion* $\longleftrightarrow$ *dual* ‘ $X \subseteq$ *assumption*
  unfolding *subset-eq*
  apply *simp*
  by (*metis assert-iff-assume*)

lemma *assumption-iff-assertion-subseteq*: $X \subseteq$ *assumption* $\longleftrightarrow$ *dual* ‘ $X \subseteq$ *assertion*
  unfolding *subset-eq*
  apply *simp*
  by (*metis dual-dual assert-iff-assume*)

lemma *assumption-prop*: $x \in$ *assumption* $\Longrightarrow (x * bot) \sqcup 1 = x$
  by (*smt assert-iff-assume assertion-prop dual-comp dual-dual dual-neg-top dual-one dual-sup dual-top*)

lemma *neg-assumption*: *neg-assume* $x \in$ *assumption*
  unfolding *assumption-def*
  apply *rule*
   by (*smt dual-comp dual-dual dual-neg-top dual-one dual-sup dual-top inf-commute inf-sup-distrib1 le-iff-inf mult.assoc mult.left-neutral neg-assume-def sup-bot-right sup-comp sup-inf-absorb sup-inf-distrib1 sup-left-commute top-comp*)

lemma *assumption-neg-assume*: $x \in$ *assumption* $\longleftrightarrow x =$ *neg-assume* (*neg-assume* x)
  by (*smt assert-iff-assume assertion-neg-assert dual-dual neg-assume-assert*)

lemma *assumption-sup-comp-eq*: $x \in$ *assumption* $\Longrightarrow y \in$ *assumption* $\Longrightarrow x \sqcup y = x * y$
  by (*smt assert-iff-assume assertion-inf-comp-eq dual-comp dual-dual dual-sup*)

lemma *sup-uminus-assume*[*simp*]: $x \in assumption \Longrightarrow x \sqcap neg\text{-}assume\ x = 1$
  by (*smt assert-iff-assume dual-dual dual-one dual-sup neg-assume-assert sup-uminus*)

lemma *inf-uminus-assume*[*simp*]: $x \in assumption \Longrightarrow x \sqcup neg\text{-}assume\ x = top$
  by (*smt assert-iff-assume dual-dual dual-sup dual-top inf-uminus neg-assume-assert sup-bot-right*)

lemma *uminus-assumption*[*simp*]: $x \in assumption \Longrightarrow neg\text{-}assume\ x \in assumption$
  by (*smt assert-iff-assume dual-dual neg-assume-assert uminus-assertion*)

lemma *uminus-uminus-assume*[*simp*]: $x \in assumption \Longrightarrow neg\text{-}assume\ (neg\text{-}assume\ x) = x$
  by (*smt assert-iff-assume dual-dual neg-assume-assert uminus-uminus*)

lemma *sup-assumption*[*simp*]: $x \in assumption \Longrightarrow y \in assumption \Longrightarrow x \sqcup y \in assumption$
  by (*smt assert-iff-assume dual-dual dual-sup inf-assertion*)

lemma *comp-assumption*[*simp*]: $x \in assumption \Longrightarrow y \in assumption \Longrightarrow x * y \in assumption$
  by (*smt assert-iff-assume comp-assertion dual-comp dual-dual*)

lemma *inf-assumption*[*simp*]: $x \in assumption \Longrightarrow y \in assumption \Longrightarrow x \sqcap y \in assumption$
  by (*smt assert-iff-assume dual-dual dual-inf sup-assertion*)

lemma [*simp*]: $x \in assumption \Longrightarrow x * x = x$
  by (*simp add*: *assumption-sup-comp-eq* [*THEN sym*])

lemma [*simp*]: $x \in assumption \Longrightarrow (x\ \hat{}\ o) * (x\ \hat{}\ o) = x\ \hat{}\ o$
  apply (*rule dual-eq*)
  by (*simp add*: *dual-comp assumption-sup-comp-eq* [*THEN sym*])

lemma [*simp*]: $top \in assumption$
  by (*unfold assumption-def*, *simp*)

lemma [*simp*]: $1 \in assumption$
  by (*unfold assumption-def*, *simp*)

lemma *assert-top*: $neg\text{-}assert\ (neg\text{-}assert\ p)\ \hat{}\ o\ ;\ bot = neg\text{-}assert\ p\ ;\ top$
  by (*smt bot-comp dual-comp dual-dual dual-top inf-comp inf-top-right mult.assoc mult.left-neutral neg-assert-def*)

lemma *assume-bot*: $neg\text{-}assume\ (neg\text{-}assume\ p)\ \hat{}\ o\ ;\ top = neg\text{-}assume\ p\ ;\ bot$
  by (*smt dual-bot dual-comp dual-one dual-sup dual-top mult.assoc mult.left-neutral neg-assert-def neg-assume-assert neg-assume-def sup-bot-right sup-comp top-comp*)

definition
  $wpb\ x = (x * bot) \sqcup 1$

lemma *wpt-iff-wpb*: $wpb\ x = wpt\ (x\ \hat{}\ o)\ \hat{}\ o$
  by (*smt dual-comp dual-dual dual-one dual-sup dual-top wpb-def wpt-def*)

lemma *wpb-is-assumption*[*simp*]: $wpb\ x \in assumption$
  by (*smt assert-iff-assume wpt-iff-wpb wpt-is-assertion*)

lemma *wpb-comp*: $(wpb\ x) * x = x$
  by (*smt dual-comp dual-dual dual-neg-top dual-sup wpt-comp wpt-iff-wpb*)

lemma *wpb-comp-2*: $wpb\ (x * y) = wpb\ (x * (wpb\ y))$
  by (*smt dual-comp dual-dual wpt-comp-2 wpt-iff-wpb*)

lemma *wpb-assumption*[*simp*]: $x \in assumption \Longrightarrow wpb\ x = x$
  by (*smt assert-iff-assume dual-dual wpt-assertion wpt-iff-wpb*)

lemma *wpb-choice*: $wpb\ (x \sqcup y) = wpb\ x \sqcup wpb\ y$
  by (*smt dual-inf dual-sup wpt-choice wpt-iff-wpb*)

lemma *wpb-dual-assumption*: $x \in assumption \Longrightarrow wpb\ (x\ \hat{}\ o) = 1$
  by (*smt assert-iff-assume dual-dual dual-one wpt-dual-assertion wpt-iff-wpb*)

lemma *wpb-mono*: $x \le y \Longrightarrow wpb\ x \le wpb\ y$
  by (*metis le-iff-sup wpb-choice*)

lemma *assumption-disjunctive*: $x \in assumption \Longrightarrow x \in disjunctive$

by (*smt assert-iff-assume assertion-conjunctive dual-comp dual-conjunctive dual-dual*)

lemma *assumption-conjunctive*: $x \in assumption \Longrightarrow x \in conjunctive$
by (*smt assert-iff-assume assertion-disjunctive dual-comp dual-disjunctive dual-dual*)

lemma *wpb-le-assumption*: $x \in assumption \Longrightarrow x * y = y \Longrightarrow x \leq wpb\ y$
by (*metis comp-assumption le-comp mult.right-neutral sup.commute sup-ge1 wpb-assumption wpb-comp-2 wpb-def wpb-is-assumption*)

definition *dual-omega* :: $'a \Rightarrow 'a$ ((- ^ $\mho$) [81] 80)
where $(x\ \hat{}\ \mho) = (((x\ \hat{}\ o)\ \hat{}\ \omega)\ \hat{}\ o)$

lemma *dual-omega-fix*: $x\hat{}\mho = (x * (x\hat{}\mho)) \sqcup 1$
by (*smt dual-comp dual-dual dual-omega-def dual-one dual-sup omega-fix*)

lemma *dual-omega-comp-fix*: $x\hat{}\mho * y = (x * (x\hat{}\mho) * y) \sqcup y$
apply (*subst dual-omega-fix*)
by (*simp add*: *sup-comp*)

lemma *dual-omega-greatest*: $z \leq (x * z) \sqcup y \Longrightarrow z \leq (x\hat{}\mho) * y$
by (*smt dual-comp dual-dual dual-le dual-neg-top dual-omega-def dual-sup omega-least*)

end

context *post-mbt-algebra*

begin

lemma *post-antitone*: $x \leq y \longrightarrow post\ y \leq post\ x$
proof
assume $x \leq y$
hence $post\ y \leq post\ x\ ;\ y\ ;\ top \sqcap post\ y$
by (*metis inf-top-left post-1 inf-mono le-comp-left-right order-refl*)
thus $post\ y \leq post\ x$
by (*metis post-2 order-trans*)
qed

lemma *post-assumption-below-one*: $q \in assumption \longrightarrow post\ q \leq post\ 1$
by (*metis post-antitone sup.commute sup-ge1 wpb-assumption wpb-def*)

lemma *post-assumption-above-one*: $q \in assumption \longrightarrow post\ 1 \leq post\ (q\ \hat{}\ o)$
by (*metis dual-le dual-one post-antitone sup.commute sup-ge1 wpb-assumption wpb-def*)

lemma *post-assumption-below-dual*: $q \in assumption \longrightarrow post\ q \leq post\ (q\ \hat{}\ o)$
by (*metis order-trans post-assumption-above-one post-assumption-below-one*)

lemma *assumption-assertion-absorb*: $q \in assumption \longrightarrow q\ ;\ (q\ \hat{}\ o) = q$
by (*smt CollectE assumption-def assumption-prop bot-comp mult.left-neutral mult-assoc sup-comp*)

lemma *post-dual-below-post-one*: $q \in assumption \longrightarrow post\ (q\ \hat{}\ o) \leq post\ 1\ ;\ q$
proof
assume $q \in assumption$
hence $post\ (q\ \hat{}\ o) \leq post\ 1\ ;\ q\ ;\ (q\ \hat{}\ o)\ ;\ top \sqcap post\ (q\ \hat{}\ o)$
by (*metis assumption-assertion-absorb gt-one-comp inf-le1 inf-top-left mult-assoc order-refl post-1 sup-uminus-assume top-unique*)
thus $post\ (q\ \hat{}\ o) \leq post\ 1\ ;\ q$
by (*metis post-2 order-trans*)
qed

lemma *post-below-post-one*: $q \in assumption \longrightarrow post\ q \leq post\ 1\ ;\ q$
by (*metis order-trans post-assumption-below-dual post-dual-below-post-one*)

end

context *complete-mbt-algebra*

begin

lemma *Inf-assumption*[*simp*]: $X \subseteq assumption \Longrightarrow Inf\ X \in assumption$

```
  by (metis SUP-def Sup-assertion assert-iff-assume assumption-iff-assertion-subseteq dual-Inf dual-dual)

definition continuous x ⟷ (∀ Y . directed Y ⟶ x ; (SUP y:Y . y) = (SUP y:Y . x ; y))

definition Continuous = { x . continuous x }

lemma continuous-Continuous: continuous x ⟷ x ∈ Continuous
  by (simp add: Continuous-def)
```

— Theorem 53.1

```
lemma one-continuous: 1 ∈ Continuous
  by (simp add: Continuous-def continuous-def SUP-def image-def)

lemma continuous-dist-ascending-chain: x ∈ Continuous ∧ ascending-chain f ⟶ x ; (SUP n::nat . f n) = (SUP n::nat . x ;
f n)
proof
  assume 1: x ∈ Continuous ∧ ascending-chain f
  hence directed (range f)
    by (metis ascending-chain-directed)
  hence x ; (SUP n::nat . f n) = (SUP y:range f . x ; y) using 1
    by (smt2 Sup.SUP-def Sup.SUP-identity-eq continuous-Continuous continuous-def)
  thus x ; (SUP n::nat . f n) = (SUP n::nat . x ; f n)
    by simp
qed
```

— Theorem 53.1

```
lemma assertion-continuous: x ∈ assertion ⟶ x ∈ Continuous
proof
  assume x ∈ assertion
  hence 1: x = (x ; top) ⊓ 1
    by (metis assertion-prop)
  have ∀ Y . directed Y ⟶ x ; (SUP y:Y . y) = (SUP y:Y . x ; y)
  proof (rule,rule)
    fix Y
    assume directed Y
    have x ; (SUP y:Y . y) = (x ; top) ⊓ (SUP y:Y . y) using 1
      by (smt inf-comp mult.assoc mult.left-neutral top-comp)
    also have ... = (SUP y:Y . (x ; top) ⊓ y)
      by (smt inf-SUP SUP-cong)
    finally show x ; (SUP y:Y . y) = (SUP y:Y . x ; y) using 1
      by (smt inf-comp mult.left-neutral mult.assoc top-comp SUP-cong)
  qed
  thus x ∈ Continuous
    by (simp add: continuous-def Continuous-def)
qed
```

— Theorem 53.1

```
lemma assumption-continuous: x ∈ assumption ⟶ x ∈ Continuous
proof
  assume x ∈ assumption
  hence 1: x = (x ; bot) ⊔ 1
    by (metis assumption-prop)
  have ∀ Y . directed Y ⟶ x ; (SUP y:Y . y) = (SUP y:Y . x ; y)
  proof (rule,rule)
    fix Y
    assume 2: directed Y
    have x ; (SUP y:Y . y) = (x ; bot) ⊔ (SUP y:Y . y) using 1
      by (smt sup-comp mult.assoc mult.left-neutral bot-comp)
    also have ... = (SUP y:Y . (x ; bot) ⊔ y) using 2
      by (smt sup-SUP SUP-cong directed-def)
    finally show x ; (SUP y:Y . y) = (SUP y:Y . x ; y) using 1
      by (smt sup-comp mult.left-neutral mult.assoc bot-comp SUP-cong)
  qed
  thus x ∈ Continuous
    by (simp add: continuous-def Continuous-def)
qed
```

— Theorem 53.1

```
lemma mult-continuous: x ∈ Continuous ∧ y ∈ Continuous ⟶ x ; y ∈ Continuous
proof
  assume 1: x ∈ Continuous ∧ y ∈ Continuous
  have ∀ Y. directed Y ⟶ x ; y ; (SUP y:Y . y) = (SUP z:Y . x ; y ; z)
  proof (rule,rule)
    fix Y
    assume directed Y
    hence x ; y ; (SUP w:Y . w) = (SUP z:Y . x ; (y ; z)) using 1
      by (metis Sup-image-eq continuous-Continuous continuous-def directed-left-mult image-ident image-image mult-assoc)
    thus x ; y ; (SUP y:Y . y) = (SUP z:Y . x ; y ; z)
      by (smt SUP-cong mult.assoc)
  qed
  thus x ; y ∈ Continuous
    by (metis continuous-Continuous continuous-def)
qed
```

— Theorem 53.1

```
lemma sup-continuous: x ∈ Continuous ∧ y ∈ Continuous ⟶ x ⊔ y ∈ Continuous
  by (smt SUP-cong SUP-sup-distrib continuous-Continuous continuous-def sup-comp)
```

— Theorem 53.1

```
lemma inf-continuous: x ∈ Continuous ∧ y ∈ Continuous ⟶ x ⊓ y ∈ Continuous
proof
  assume 1: x ∈ Continuous ∧ y ∈ Continuous
  have ∀ Y. directed Y ⟶ (x ⊓ y) ; (SUP y:Y . y) = (SUP z:Y . (x ⊓ y) ; z)
  proof (rule,rule)
    fix Y
    assume 2: directed Y
    have 3: (SUP w:Y . SUP z:Y . (x ; w) ⊓ (y ; z)) ≤ (SUP z:Y . (x ; z) ⊓ (y ; z))
      apply (rule SUP-least)
      apply (rule SUP-least)
    proof -
      fix w z
      assume w ∈ Y and z ∈ Y
      with 2 obtain v where 4: v∈Y ∧ w ≤ v ∧ z ≤ v
        by (metis directed-def)
      hence x ; w ⊓ (y ; z) ≤ (x ; v) ⊓ (y ; v)
        by (metis inf-mono le-comp-left-right order-refl)
      thus x ; w ⊓ (y ; z) ≤ (SUP z:Y . (x ; z) ⊓ (y ; z)) using 4
        by (smt SUP-upper imageI order-trans)
    qed
    have (SUP z:Y . (x ; z) ⊓ (y ; z)) ≤ (SUP w:Y . SUP z:Y . (x ; w) ⊓ (y ; z))
      apply (rule SUP-least)
      apply (smt SUP-upper order-trans)
      done
    hence (SUP w:Y . SUP z:Y . (x ; w) ⊓ (y ; z)) = (SUP z:Y . (x ⊓ y) ; z) using 3
      by (smt antisym SUP-cong inf-comp)
    thus (x ⊓ y) ; (SUP y:Y . y) = (SUP z:Y . (x ⊓ y) ; z) using 1 2
      by (metis inf-comp continuous-Continuous continuous-def SUP-inf-distrib2)
  qed
  thus x ⊓ y ∈ Continuous
    by (metis continuous-Continuous continuous-def)
qed
```

— Theorem 53.1

```
lemma dual-star-continuous: x ∈ Continuous ⟶ x ^ ⊗ ∈ Continuous
proof
  assume 1: x ∈ Continuous
  have ∀ Y. directed Y ⟶ (x ^ ⊗) ; (SUP y:Y . y) = (SUP z:Y . (x ^ ⊗) ; z)
  proof (rule,rule)
    fix Y
    assume directed Y
    hence directed (op ; (x ^ ⊗) ‘ Y)
      by (metis directed-left-mult)
    hence x ; (SUP y:Y . (x ˆ ⊗) ; y) = (SUP y:Y . x ; ((x ˆ ⊗) ; y)) using 1
      by (metis continuous-Continuous continuous-def Sup-image-eq image-ident image-image)
    also have ... = (SUP y:Y . x ; (x ˆ ⊗) ; y)
      by (metis mult-assoc)
    also have ... ≤ (SUP y:Y . (x ˆ ⊗) ; y)
      apply (rule SUP-least)
      apply (metis SUP-upper dual-star-comp-fix order-trans sup-ge1)
      done
    finally have x ; (SUP y:Y . (x ˆ ⊗) ; y) ⊔ (SUP y:Y . y) ≤ (SUP y:Y . (x ˆ ⊗) ; y)
      apply (rule sup-least)
      apply (smt SUP-least SUP-upper dual-star-comp-fix order-trans sup-ge2)
      done
    thus (x ˆ ⊗) ; (SUP y:Y . y) = (SUP z:Y . (x ˆ ⊗) ; z)
      by (smt SUP-least SUP-upper antisym dual-star-least le-comp)
  qed
  thus x ˆ ⊗ ∈ Continuous
    by (metis continuous-Continuous continuous-def)
qed
```

— Theorem 53.1

```
lemma omega-continuous: x ∈ Continuous ⟶ x ˆ ω ∈ Continuous
proof
  assume 1: x ∈ Continuous
  have ∀ Y. directed Y ⟶ (x ˆ ω) ; (SUP y:Y . y) = (SUP z:Y . (x ˆ ω) ; z)
  proof (rule,rule)
    fix Y
    assume 2: directed Y
    hence directed (op ; (x ˆ ω) ‘ Y)
      by (metis directed-left-mult)
    hence x ; (SUP y:Y . (x ˆ ω) ; y) = (SUP y:Y . x ; ((x ˆ ω) ; y)) using 1
      by (metis continuous-Continuous continuous-def Sup-image-eq image-ident image-image)
    hence 3: x ; (SUP y:Y . (x ˆ ω) ; y) = (SUP y:Y . x ; (x ˆ ω) ; y)
      by (simp add: mult-assoc)
    have (SUP y:Y . x ; (x ˆ ω) ; y) ⊓ (SUP y:Y . y) = (SUP w:Y . SUP z:Y . (x ; (x ˆ ω) ; w) ⊓ z)
      using SUP-inf-distrib2 by blast
    hence x ; (SUP y:Y . (x ˆ ω) ; y) ⊓ (SUP y:Y . y) = (SUP w:Y . SUP z:Y . (x ; (x ˆ ω) ; w) ⊓ z) using 3
      by metis
    also have ... ≤ (SUP y:Y . (x ˆ ω) ; y)
      apply (rule SUP-least)
      apply (rule SUP-least)
    proof −
      fix w z
      assume w ∈ Y and z ∈ Y
      with 2 obtain v where 4: v∈Y ∧ w ≤ v ∧ z ≤ v
        by (metis directed-def)
      hence x ; x ˆ ω ; w ⊓ z ≤ x ˆ ω ; v
        by (metis inf-mono le-comp-left-right order-refl omega-comp-fix)
      thus x ; x ˆ ω ; w ⊓ z ≤ (SUP y:Y . (x ˆ ω) ; y) using 4
        by (smt SUP-upper imageI order-trans)
    qed
    finally show (x ˆ ω) ; (SUP y:Y . y) = (SUP z:Y . (x ˆ ω) ; z)
      by (smt SUP-least SUP-upper antisym omega-least le-comp)
  qed
  thus x ˆ ω ∈ Continuous
    by (metis continuous-Continuous continuous-def)
qed

definition cocontinuous x ⟷ (∀ Y . codirected Y ⟶ x ; (INF y:Y . y) = (INF y:Y . x ; y))

definition Cocontinuous = { x . cocontinuous x }

lemma directed-dual: directed X ⟷ codirected (dual ‘ X)
  by (simp add: directed-def codirected-def dual-le[THEN sym])

lemma dual-dual-image: dual ‘ dual ‘ X = X
  unfolding image-def
  apply (auto intro: dual-dual)
  done
```

```
lemma continuous-dual: continuous x ⟷ cocontinuous (x ^ o)
  unfolding continuous-def cocontinuous-def
  proof
    assume 1: ∀ Y. directed Y ⟶ x ; (SUP y:Y . y) = (SUP y:Y . x ; y)
    show ∀ Y. codirected Y ⟶ x ^ o ; (INF y:Y . y) = (INF y:Y . x ^ o ; y)
    proof (rule,rule)
      fix Y
      assume codirected Y
      hence x ^ o ; (INF y:Y . y) = (INF y:(dual ‘ Y) . (x ; y) ^ o) using 1
        by (metis dual-dual-image dual-SUP Inf-image-eq image-ident image-image dual-comp directed-dual)
      also have ... = (INF y:(dual ‘ Y) . x ^ o ; y ^ o)
        by (metis dual-comp)
      also have ... = (INF y:Y . x ^ o ; y)
        by (metis dual-dual-image Inf-image-eq image-image)
      finally show x ^ o ; (INF y:Y . y) = (INF y:Y . x ^ o ; y)
        by metis
     qed
  next
    assume 2: ∀ Y. codirected Y ⟶ x ^ o ; (INF y:Y . y) = (INF y:Y . x ^ o ; y)
    show ∀ Y. directed Y ⟶ x ; (SUP y:Y . y) = (SUP y:Y . x ; y)
    proof (rule,rule)
      fix Y
      assume directed Y
      hence x ; (SUP y:Y . y) = (SUP y:(dual ‘ Y) . (x ^ o ; y) ^ o) using 2
        by (metis dual-dual-image dual-INF Sup-image-eq image-ident image-image dual-comp dual-dual directed-dual)
      also have ... = (SUP y:(dual ‘ Y) . x ; y ^ o)
        by (metis dual-comp dual-dual)
      also have ... = (SUP y:Y . x ; y)
        by (metis dual-dual-image Sup-image-eq image-image)
      finally show x ; (SUP y:Y . y) = (SUP y:Y . x ; y)
        by metis
     qed
qed

lemma cocontinuous-Cocontinuous: cocontinuous x ⟷ x ∈ Cocontinuous
  by (simp add: Cocontinuous-def)
```

— Theorem 53.1 and Theorem 53.2

```
lemma Continuous-dual: x ∈ Continuous ⟷ x ^ o ∈ Cocontinuous
  by (metis cocontinuous-Cocontinuous continuous-Continuous continuous-dual)
```

— Theorem 53.2

```
lemma one-cocontinuous: 1 ∈ Cocontinuous
  by (smt Continuous-dual dual-one one-continuous)

lemma ascending-chain-dual: ascending-chain f ⟷ descending-chain (dual o f)
  by (metis ascending-chain-def descending-chain-def o-def dual-le)

lemma cocontinuous-dist-descending-chain: x ∈ Cocontinuous ∧ descending-chain f ⟶ x ; (INF n::nat . f n) = (INF n::nat
. x ; f n)
proof
  assume x ∈ Cocontinuous ∧ descending-chain f
  hence x ^ o ; (SUP n::nat . (dual o f) n) = (SUP n::nat . x ^ o ; (dual o f) n)
     by (smt Continuous-dual SUP-cong ascending-chain-dual continuous-dist-ascending-chain descending-chain-def dual-dual
o-def)
  thus x ; (INF n::nat . f n) = (INF n::nat . x ; f n)
    by (smt INF-cong dual-SUP dual-comp dual-dual o-def)
qed
```

— Theorem 53.2

```
lemma assertion-cocontinuous: x ∈ assertion ⟶ x ∈ Cocontinuous
  by (smt Continuous-dual assert-iff-assume assumption-continuous dual-dual)
```

— Theorem 53.2

```
lemma assumption-cocontinuous: x ∈ assumption ⟶ x ∈ Cocontinuous
```

```
  by (smt Continuous-dual assert-iff-assume assertion-continuous dual-dual)

— Theorem 53.2

lemma mult-cocontinuous: x ∈ Cocontinuous ∧ y ∈ Cocontinuous ⟶ x ; y ∈ Cocontinuous
  by (smt Continuous-dual dual-comp dual-dual mult-continuous)

— Theorem 53.2

lemma sup-cocontinuous: x ∈ Cocontinuous ∧ y ∈ Cocontinuous ⟶ x ⊔ y ∈ Cocontinuous
  by (smt Continuous-dual dual-sup dual-dual inf-continuous)

— Theorem 53.2

lemma inf-cocontinuous: x ∈ Cocontinuous ∧ y ∈ Cocontinuous ⟶ x ⊓ y ∈ Cocontinuous
  by (smt Continuous-dual dual-inf dual-dual sup-continuous)

— Theorem 53.2

lemma dual-omega-cocontinuous: x ∈ Cocontinuous ⟶ x ^ ℧ ∈ Cocontinuous
  by (smt Continuous-dual dual-omega-def dual-dual omega-continuous)

— Theorem 53.2

lemma star-cocontinuous: x ∈ Cocontinuous ⟶ x ^ ∗ ∈ Cocontinuous
  by (smt Continuous-dual dual-star-def dual-dual dual-star-continuous)

lemma dual-omega-iterate: y ∈ Cocontinuous ⟶ y ^ ℧ ∗ z = (INF n::nat . ((λx . y ∗ x ⊔ z) ^ n) top)
  apply rule
  apply (rule antisym)
  apply (rule INF-greatest)
  apply simp
  proof –
    fix n
    show y ^ ℧ ; z ≤ ((λx. y ; x ⊔ z) ^ n) top
      apply (induct n)
      apply (metis power-zero-id id-def top-greatest)
      apply (smt dual-omega-comp-fix le-comp mult-assoc order-refl sup-mono power-succ-unfold-ext)
      done
  next
    assume 1: y ∈ Cocontinuous
    have 2: descending-chain (λn . ((λx. y ; x ⊔ z) ^ n) top)
    proof (subst descending-chain-def, rule)
      fix n
      show ((λx. y ; x ⊔ z) ^ Suc n) top ≤ ((λx. y ; x ⊔ z) ^ n) top
        apply (induct n)
        apply (metis power-zero-id id-def top-greatest)
        apply (smt power-succ-unfold-ext sup-mono order-refl le-comp)
        done
    qed
    have (INF n. ((λx. y ; x ⊔ z) ^ n) top) ≤ (INF n. ((λx. y ; x ⊔ z) ^ Suc n) top)
      apply (rule INF-greatest)
      unfolding power-succ-unfold-ext
      apply (smt power-succ-unfold-ext INF-lower UNIV-I)
      done
    thus (INF n. ((λx. y ; x ⊔ z) ^ n) top) ≤ y ^ ℧ ; z using 1 2
      by (smt INF-cong cocontinuous-dist-descending-chain power-succ-unfold-ext sup-INF sup-commute dual-omega-greatest)
  qed

lemma dual-omega-iterate-one: y ∈ Cocontinuous ⟶ y ^ ℧ = (INF n::nat . ((λx . y ∗ x ⊔ 1) ^ n) top)
  by (metis dual-omega-iterate mult.right-neutral)

subclass ccpo
  apply unfold-locales
  apply (metis Sup-upper)
  apply (metis Sup-least)
  done

end
```

```
class post-mbt-algebra-ext = post-mbt-algebra +
  assumes post-sub-fusion: post 1 ; neg-assume q ≤ post (neg-assume q ^ o)

begin

lemma post-fusion: post (neg-assume q ^ o) = post 1 ; neg-assume q
  by (metis antisym post-dual-below-post-one neg-assumption post-sub-fusion)

lemma post-dual-post-one: q ∈ assumption ⟶ post 1 ; q ≤ post (q ^ o)
  by (metis assumption-neg-assume post-sub-fusion)

end

instance MonoTran :: (complete-boolean-algebra) post-mbt-algebra-ext
proof
  fix q::'a MonoTran
  show post 1 ; neg-assume q ≤ post (neg-assume q ^ o)
      apply (simp add: neg-assume-def post-MonoTran-def dual-MonoTran-def times-MonoTran-def top-MonoTran-def
one-MonoTran-def bot-MonoTran-def inf-MonoTran-def sup-MonoTran-def less-eq-MonoTran-def Abs-MonoTran-inverse)
    apply (simp add: dual-fun-def le-fun-def post-fun-def)
    apply (smt inf-compl-bot inf-top-right sup-bot-right sup-commute sup-ge2 sup-inf-distrib1 top-le)
    done
qed

class complete-mbt-algebra-ext = complete-mbt-algebra + post-mbt-algebra-ext

instance MonoTran :: (complete-boolean-algebra) complete-mbt-algebra-ext ..

end
```

40 MonotonicBooleanTransformersInstances

theory *MonotonicBooleanTransformersInstances*

imports *MonotonicBooleanTransformers PrePostModal GeneralRefinementAlgebra*

begin

sublocale *mbt-algebra* < *mbta*!: *bounded-idempotent-left-semiring* **where** *plus* = *sup* **and** *zero* = *bot* **and** *T* = *top*
  apply *unfold-locales*
  apply (*metis sup-assoc*)
  apply (*metis sup-commute*)
  apply (*metis sup-idem*)
  apply (*metis le-iff-sup*)
  apply (*metis less-le-not-le*)
  apply (*metis sup-bot-left*)
  apply (*metis le-comp le-supI sup-ge1 sup-ge2*)
  apply (*metis sup-comp*)
  apply (*metis bot-comp*)
  apply (*metis mult.left-neutral*)
  apply (*metis mult-1-right order-refl*)
  apply (*metis mult.assoc order-refl*)
  apply (*metis sup-top-right*)
  apply (*metis mult.assoc*)
  apply (*metis mult.right-neutral*)
  done

sublocale *mbt-algebra* < *mbta-dual*!: *bounded-idempotent-left-semiring* **where** *less* = *greater* **and** *less-eq* = *greater-eq* **and** *plus* = *inf* **and** *zero* = *top* **and** *T* = *bot*
  apply *unfold-locales*
  apply (*metis inf-assoc*)
  apply (*metis inf-commute*)
  apply (*metis inf-idem*)
  apply (*metis inf.commute le-iff-inf*)
  apply (*metis less-le-not-le*)
  apply (*metis inf-top-left*)
  apply (*metis le-comp le-infI inf-le1 inf-le2*)
  apply (*metis inf-comp*)
  apply (*metis top-comp*)
  apply (*metis mult.left-neutral*)
  apply (*metis mult.right-neutral order-refl*)
  apply (*metis mult.assoc order-refl*)
  apply (*metis inf-bot-right*)
  done

sublocale *mbt-algebra* < *mbta*!: *bounded-general-refinement-algebra* **where** *plus* = *sup* **and** *star* = *dual-star* **and** *zero* = *bot* **and** *Omega* = *dual-omega* **and** *T* = *top*
  apply *unfold-locales*
  apply (*metis dual-star-fix eq-refl sup.commute*)
  apply (*metis dual-star-least sup.commute*)
  apply (*metis dual-omega-fix eq-refl sup.commute*)
  apply (*metis dual-omega-greatest sup.commute*)
  done

sublocale *mbt-algebra* < *mbta-dual*!: *bounded-general-refinement-algebra* **where** *less* = *greater* **and** *less-eq* = *greater-eq* **and** *plus* = *inf* **and** *zero* = *top* **and** *Omega* = *omega* **and** *T* = *bot*
  apply *unfold-locales*
  apply (*metis eq-refl inf.commute star-fix*)
  apply (*metis inf.commute star-greatest*)
  apply (*metis eq-refl inf.commute omega-fix*)
  apply (*metis inf.commute omega-least*)
  done

— Theorem 50.9(b)

sublocale *mbt-algebra* < *mbta*!: *left-conway-semiring-L* **where** *circ* = *dual-star* **and** *plus* = *sup* **and** *zero* = *bot* **and** *L* = *bot*
  apply *unfold-locales*
  apply (*metis mbta.add-left-zero mbta.star-one mult.left-neutral*)
  apply (*metis eq-refl mbta.add-right-zero*)

done

— Theorem 50.8(a)

sublocale *mbt-algebra* < *mbta-dual*!: *left-conway-semiring-L* **where** *circ* = *omega* **and** *less* = *greater* **and** *less-eq* = *greater-eq* **and** *plus* = *inf* **and** *zero* = *top* **and** *L* = *bot*
  apply *unfold-locales*
  apply (*metis bot-comp inf-bot-left mbta-dual.Omega-one*)
  apply (*metis bot-least inf-bot-right*)
  done

— Theorem 50.8(b)

sublocale *mbt-algebra* < *mbta-fix*!: *left-conway-semiring-L* **where** *circ* = *dual-omega* **and** *plus* = *sup* **and** *zero* = *bot* **and** *L* = *top*
  apply *unfold-locales*
  apply (*metis mbta.Omega-one mbta.add-left-top mbta.top-left-zero*)
  apply (*metis mbta.add-right-top top-greatest*)
  done

— Theorem 50.9(a)

sublocale *mbt-algebra* < *mbta-fix-dual*!: *left-conway-semiring-L* **where** *circ* = *star* **and** *less* = *greater* **and** *less-eq* = *greater-eq* **and** *plus* = *inf* **and** *zero* = *top* **and** *L* = *top*
  apply *unfold-locales*
  apply (*metis inf-top-left mbta-dual.star-one mult.left-neutral*)
  apply (*metis eq-refl inf-top-right*)
  done

sublocale *mbt-algebra* < *mbta*!: *left-kleene-conway-semiring* **where** *circ* = *dual-star* **and** *plus* = *sup* **and** *star* = *dual-star* **and** *zero* = *bot* **..**

sublocale *mbt-algebra* < *mbta-dual*!: *left-kleene-conway-semiring* **where** *circ* = *omega* **and** *less* = *greater* **and** *less-eq* = *greater-eq* **and** *plus* = *inf* **and** *zero* = *top* **..**

sublocale *mbt-algebra* < *mbta-fix*!: *left-kleene-conway-semiring* **where** *circ* = *dual-omega* **and** *plus* = *sup* **and** *star* = *dual-star* **and** *zero* = *bot* **..**

sublocale *mbt-algebra* < *mbta-fix-dual*!: *left-kleene-conway-semiring* **where** *circ* = *star* **and** *less* = *greater* **and** *less-eq* = *greater-eq* **and** *plus* = *inf* **and** *zero* = *top* **..**

sublocale *mbt-algebra* < *mbta*!: *tests* **where** *plus* = *sup* **and** *uminus* = *neg-assert* **and** *zero* = *bot*
  apply *unfold-locales*
  apply (*metis mult.assoc*)
  apply (*metis neg-assertion assertion-inf-comp-eq inf-commute*)
  apply (*simp add*: *dual-inf dual-comp dual-sup inf-comp sup-comp neg-assert-def*)
  apply (*metis inf-assoc dual-neg sup-bot-right sup-inf-distrib1*)
  apply (*metis comp-assertion neg-assertion uminus-uminus*)
  apply (*rule the-equality*[*THEN sym*])
  apply (*metis assertion-inf-comp-eq inf-uminus neg-assertion*)
  apply (*metis bot-comp dual-top inf-bot-left neg-assert-def*)
  apply (*metis dual-bot inf-top-left neg-assert-def top-comp*)
  apply (*simp add*: *dual-inf dual-comp dual-sup inf-comp sup-comp neg-assert-def*)
  apply (*metis inf-sup-distrib2*)
  apply (*metis assertion-inf-comp-eq le-iff-inf neg-assertion*)
  apply (*metis less-le-not-le*)
  done

sublocale *mbt-algebra* < *mbta-dual*!: *tests* **where** *less* = *greater* **and** *less-eq* = *greater-eq* **and** *plus* = *inf* **and** *uminus* = *neg-assume* **and** *zero* = *top*
  apply *unfold-locales*
  apply (*metis mult.assoc*)
  apply (*metis neg-assumption assumption-sup-comp-eq sup-commute*)
  apply (*simp add*: *dual-inf dual-comp dual-sup inf-comp sup-comp neg-assume-def*)
  apply (*smt dual-comp dual-dual dual-inf dual-neg dual-top inf-sup-distrib1 inf-top-right sup.commute sup.left-commute*)
  apply (*metis comp-assumption neg-assumption uminus-uminus-assume*)
  apply (*rule the-equality*[*THEN sym*])
  apply (*metis assumption-sup-comp-eq inf-uminus-assume neg-assumption*)
  apply (*metis top-comp dual-bot sup-top-left neg-assume-def*)

apply (*metis dual-top sup-bot-left neg-assume-def bot-comp*)
apply (*simp add*: *dual-inf dual-comp dual-sup inf-comp sup-comp neg-assume-def*)
apply (*metis sup-inf-distrib2*)
apply (*metis assumption-sup-comp-eq le-iff-sup neg-assumption sup.commute*)
apply (*metis less-le-not-le*)
done

— Theorem 51.2

sublocale *mbt-algebra* < *mbta*!: *bounded-relative-antidomain-semiring* **where** *d* = λ*x* . (*x* ∗ *top*) ⊓ *1* **and** *plus* = *sup* **and** *uminus* = *neg-assert* **and** *zero* = *bot* **and** *T* = *top* **and** *Z* = *bot*
apply *unfold-locales*
apply (*simp-all add*: *dual-inf dual-comp dual-sup inf-comp sup-comp neg-assert-def*)
apply (*metis dual-neg eq-refl inf.commute inf-mono mbta.top-right-mult-increasing*)
apply (*metis mbta.add-left-zero mbta.mult-right-dist-add mult.assoc mult.left-neutral sup.commute*)
apply (*metis dual-neg inf.commute inf.left-commute inf-bot-left*)
apply (*metis inf.commute inf-sup-distrib1*)
apply (*metis inf.assoc*)
done

— Theorem 51.1

sublocale *mbt-algebra* < *mbta-dual*!: *bounded-relative-antidomain-semiring* **where** *d* = λ*x* . (*x* ∗ *bot*) ⊔ *1* **and** *less* = *greater* **and** *less-eq* = *greater-eq* **and** *plus* = *inf* **and** *uminus* = *neg-assume* **and** *zero* = *top* **and** *T* = *bot* **and** *Z* = *top*
apply *unfold-locales*
apply (*simp-all add*: *dual-inf dual-comp dual-sup inf-comp sup-comp neg-assume-def*)
apply (*metis dual-dual dual-neg-top mbta.add-isotone mbta.zero-right-mult-decreasing mbta-dual.order-refl sup.commute*)
apply (*smt bot-comp dual-bot dual-comp dual-one dual-sup mbta.add-right-zero mbta.mult-right-dist-add mult.assoc mult.left-neutral*)
apply (*metis dual-dual dual-neg-top mbta.add-right-top sup.commute sup.left-commute*)
apply (*metis sup.commute sup-inf-distrib1*)
apply (*smt sup.assoc*)
done

sublocale *mbt-algebra* < *mbta*!: *relative-domain-semiring-split* **where** *d* = λ*x* . (*x* ∗ *top*) ⊓ *1* **and** *plus* = *sup* **and** *zero* = *bot* **and** *Z* = *bot*
apply *unfold-locales*
apply (*metis eq-refl mbta.add-right-zero*)
done

sublocale *mbt-algebra* < *mbta-dual*!: *relative-domain-semiring-split* **where** *d* = λ*x* . (*x* ∗ *bot*) ⊔ *1* **and** *less* = *greater* **and** *less-eq* = *greater-eq* **and** *plus* = *inf* **and** *zero* = *top* **and** *Z* = *top*
apply *unfold-locales*
apply (*metis eq-refl inf-top-right*)
done

sublocale *mbt-algebra* < *mbta*!: *diamond-while* **where** *box* = λ*x y* . *neg-assert* (*x* ∗ *neg-assert y*) **and** *circ* = *dual-star* **and** *d* = λ*x* . (*x* ∗ *top*) ⊓ *1* **and** *diamond* = λ*x y* . (*x* ∗ *y* ∗ *top*) ⊓ *1* **and** *ite* = λ*x p y* . (*p* ∗ *x*) ⊔ (*neg-assert p* ∗ *y*) **and** *plus* = *sup* **and** *pre* = λ*x y* . *wpt* (*x* ∗ *y*) **and** *uminus* = *neg-assert* **and** *while* = λ*p x* . ((*p* ∗ *x*) ˆ ⊗) ∗ *neg-assert p* **and** *zero* = *bot* **and** *T* = *top* **and** *Z* = *bot*
apply *unfold-locales*
apply *simp-all*
apply (*metis wpt-def*)
done

sublocale *mbt-algebra* < *mbta-dual*!: *box-while* **where** *box* = λ*x y* . *neg-assume* (*x* ∗ *neg-assume y*) **and** *circ* = *omega* **and** *d* = λ*x* . (*x* ∗ *bot*) ⊔ *1* **and** *diamond* = λ*x y* . (*x* ∗ *y* ∗ *bot*) ⊔ *1* **and** *ite* = λ*x p y* . (*p* ∗ *x*) ⊓ (*neg-assume p* ∗ *y*) **and** *less* = *greater* **and** *less-eq* = *greater-eq* **and** *plus* = *inf* **and** *pre* = λ*x y* . *wpb* (*x* ˆ *o* ; *y*) **and** *uminus* = *neg-assume* **and** *while* = λ*p x* . ((*p* ∗ *x*) ˆ ω) ∗ *neg-assume p* **and** *zero* = *top* **and** *T* = *bot* **and** *Z* = *top*
apply *unfold-locales*
apply *simp-all*
apply (*metis bot-comp dual-comp dual-dual dual-top mbta.add-left-zero mbta.mult-right-dist-add mult.assoc mult.left-neutral neg-assume-def sup.commute wpb-def*)
done

sublocale *mbt-algebra* < *mbta-fix*!: *diamond-while* **where** *box* = λ*x y* . *neg-assert* (*x* ∗ *neg-assert y*) **and** *circ* = *dual-omega* **and** *d* = λ*x* . (*x* ∗ *top*) ⊓ *1* **and** *diamond* = λ*x y* . (*x* ∗ *y* ∗ *top*) ⊓ *1* **and** *ite* = λ*x p y* . (*p* ∗ *x*) ⊔ (*neg-assert p* ∗ *y*) **and** *plus* = *sup* **and** *pre* = λ*x y* . *wpt* (*x* ∗ *y*) **and** *uminus* = *neg-assert* **and** *while* = λ*p x* . ((*p* ∗ *x*) ˆ ℧) ∗ *neg-assert p* **and** *zero* = *bot* **and** *T* = *top* **and** *Z* = *bot*

```
  apply unfold-locales
  apply simp-all
  done

sublocale mbt-algebra < mbta-fix-dual!: box-while where box = λx y . neg-assume (x ∗ neg-assume y) and circ = star and
d = λx . (x ∗ bot) ⊔ 1 and diamond = λx y . (x ∗ y ∗ bot) ⊔ 1 and ite = λx p y . (p ∗ x) ⊓ (neg-assume p ∗ y) and less =
greater and less-eq = greater-eq and plus = inf and pre = λx y . wpb (x ˆ o ; y) and uminus = neg-assume and while =
λp x . ((p ∗ x) ˆ ∗) ∗ neg-assume p and zero = top and T = bot and Z = top
  apply unfold-locales
  apply simp-all
  done

sublocale mbt-algebra < mbta-pre!: box-while where box = λx y . neg-assert (x ∗ neg-assert y) and circ = dual-star and d
= λx . (x ∗ top) ⊓ 1 and diamond = λx y . (x ∗ y ∗ top) ⊓ 1 and ite = λx p y . (p ∗ x) ⊔ (neg-assert p ∗ y) and plus =
sup and pre = λx y . wpt (x ˆ o ∗ y) and uminus = neg-assert and while = λp x . ((p ∗ x) ˆ ⊗) ∗ neg-assert p and zero =
bot and T = top and Z = bot
  apply unfold-locales
  apply simp-all
   apply (smt dual-bot dual-comp dual-dual dual-inf mbta.add-left-zero mbta.mult-associative mbta.mult-right-dist-add
mbta-dual.Z-top mult.left-neutral neg-assert-def sup.commute wpt-def)
  done

sublocale mbt-algebra < mbta-pre-dual!: diamond-while where box = λx y . neg-assume (x ∗ neg-assume y) and circ =
omega and d = λx . (x ∗ bot) ⊔ 1 and diamond = λx y . (x ∗ y ∗ bot) ⊔ 1 and ite = λx p y . (p ∗ x) ⊓ (neg-assume p ∗
y) and less = greater and less-eq = greater-eq and plus = inf and pre = λx y . wpb (x ; y) and uminus = neg-assume and
while = λp x . ((p ∗ x) ˆ ω) ∗ neg-assume p and zero = top and T = bot and Z = top
  apply unfold-locales
  apply simp-all
  apply (metis wpb-def)
  done

sublocale mbt-algebra < mbta-pre-fix!: box-while where box = λx y . neg-assert (x ∗ neg-assert y) and circ = dual-omega
and d = λx . (x ∗ top) ⊓ 1 and diamond = λx y . (x ∗ y ∗ top) ⊓ 1 and ite = λx p y . (p ∗ x) ⊔ (neg-assert p ∗ y) and
plus = sup and pre = λx y . wpt (x ˆ o ∗ y) and uminus = neg-assert and while = λp x . ((p ∗ x) ˆ ℧) ∗ neg-assert p and
zero = bot and T = top and Z = bot
  apply unfold-locales
  apply simp-all
  done

sublocale mbt-algebra < mbta-pre-fix-dual!: diamond-while where box = λx y . neg-assume (x ∗ neg-assume y) and circ =
star and d = λx . (x ∗ bot) ⊔ 1 and diamond = λx y . (x ∗ y ∗ bot) ⊔ 1 and ite = λx p y . (p ∗ x) ⊓ (neg-assume p ∗ y)
and less = greater and less-eq = greater-eq and plus = inf and pre = λx y . wpb (x ; y) and uminus = neg-assume and
while = λp x . ((p ∗ x) ˆ ∗) ∗ neg-assume p and zero = top and T = bot and Z = top
  apply unfold-locales
  apply simp-all
  done

sublocale post-mbt-algebra < mbta!: pre-post-spec-Hd where box = λx y . neg-assert (x ∗ neg-assert y) and d = λx . (x ∗
top) ⊓ 1 and diamond = λx y . (x ∗ y ∗ top) ⊓ 1 and plus = sup and pre = λx y . wpt (x ∗ y) and pre-post = λp q . p ∗
post q and uminus = neg-assert and zero = bot and Hd = post 1 and T = top and Z = bot
  apply unfold-locales
  apply (metis mult.assoc mult.left-neutral post-1)
  apply (metis inf.commute inf-top-right mult.assoc mult.left-neutral post-2)
  apply (metis neg-assertion assertion-disjunctive disjunctiveD)
  apply rule
  defer
  apply (smt mbta.a-d-closed post-1 mult-assoc mbta.diamond-left-isotone wpt-def)
  apply (metis inf.commute inf-comp inf-top-left mult.assoc mult.left-neutral)
  proof −
    fix p x q
    let ?pt = neg-assert p
    let ?qt = neg-assert q
    assume ?pt ≤ wpt (x ; ?qt)
    hence ?pt ; post ?qt ≤ x ; ?qt ; top ; post ?qt ⊓ post ?qt
      by (metis mbta.mult-left-isotone wpt-def inf-comp mult.left-neutral)
    thus ?pt ; post ?qt ≤ x
      by (smt mbta.top-left-zero mult.assoc post-2 order-trans)
  qed
```

```
sublocale post-mbt-algebra < mbta-dual!: pre-post-spec-H where box = λx y . neg-assume (x ∗ neg-assume y) and d = λx . (x ∗ bot) ⊔ 1 and diamond = λx y . (x ∗ y ∗ bot) ⊔ 1 and less = greater and less-eq = greater-eq and plus = inf and pre = λx y . wpb (x ˆ o ; y) and pre-post = λp q . (p ˆ o) ∗ post (q ˆ o) and uminus = neg-assume and zero = top and H = post 1 and T = bot and Z = top
  apply unfold-locales
  prefer 2
  apply (metis mbta.mult-right-one post-1)
  proof
    fix p x q
    let ?pt = neg-assume p
    let ?qt = neg-assume q
    assume wpb (x ˆ o ; ?qt) ≤ ?pt
    hence ?pt ˆ o ; post (?qt ˆ o) ≤ (x ; (?qt ˆ o) ; top ⊓ 1) ; post (?qt ˆ o)
      by (smt wpb-def dual-le dual-comp dual-dual dual-one dual-sup dual-top mbta.mult-left-isotone)
    thus ?pt ˆ o ; post (?qt ˆ o) ≤ x
      by (smt inf-comp mult-assoc top-comp mult.left-neutral post-2 order-trans)
  next
    fix p x q
    let ?pt = neg-assume p
    let ?qt = neg-assume q
    assume 1: ?pt ˆ o ; post (?qt ˆ o) ≤ x
    have ?pt ˆ o = ?pt ˆ o ; post (?qt ˆ o) ; (?qt ˆ o) ; top ⊓ 1
      by (metis assert-iff-assume assertion-prop dual-dual mult-assoc neg-assumption post-1)
    thus wpb (x ˆ o ; ?qt) ≤ ?pt using 1
      by (smt dual-comp dual-dual dual-le dual-one dual-sup dual-top wpb-def mbta.diamond-left-isotone)
  next
    fix x q
    let ?qt = neg-assume q
    have x ; ?qt ; bot ⊓ (post 1 ; neg-assume ?qt) = (x ; neg-assume ?qt ˆ o ; top ⊓ post 1) ; neg-assume ?qt
      by (smt inf-comp mbta.add-right-zero mbta.mult-left-one mbta.mult-right-dist-add mbta-dual.d-def mult-assoc neg-assume-def top-comp)
    also have ... ≤ x ; neg-assume ?qt ˆ o
      by (smt assumption-assertion-absorb dual-comp dual-dual mbta.mult-left-isotone mult.right-neutral mult-assoc neg-assumption post-2)
    also have ... ≤ x
      by (metis dual-comp dual-dual dual-le mbta.mult-left-sub-dist-add-left mult.right-neutral neg-assume-def sup.commute)
    finally show x ; ?qt ; bot ⊓ (post 1 ; neg-assume ?qt) ≤ x
      by metis
  qed

sublocale post-mbt-algebra < mbta-pre!: pre-post-spec-H where box = λx y . neg-assert (x ∗ neg-assert y) and d = λx . (x ∗ top) ⊓ 1 and diamond = λx y . (x ∗ y ∗ top) ⊓ 1 and plus = sup and pre = λx y . wpt (x ˆ o ∗ y) and pre-post = λp q . p ˆ o ∗ (post q ˆ o) and uminus = neg-assert and zero = bot and H = post 1 ˆ o and T = top and Z = bot
  apply unfold-locales
  prefer 2
  apply (metis dual-comp dual-top mbta.Hd-total)
  proof
    fix p x q
    let ?pt=neg-assert p
    let ?qt=neg-assert q
    assume ?pt ≤ wpt (x ˆ o ; ?qt)
    hence ?pt ; post ?qt ≤ (x ˆ o ; ?qt ; top ⊓ 1) ; post ?qt
      by (metis wpt-def mbta.mult-left-isotone)
    also have ... ≤ x ˆ o
      by (smt inf-comp mult.left-neutral mult-assoc post-2 top-comp)
    finally show x ≤ ?pt ˆ o ; (post ?qt ˆ o)
      by (metis dual-le dual-comp dual-dual)
  next
    fix p x q
    let ?pt=neg-assert p
    let ?qt=neg-assert q
    assume x ≤ ?pt ˆ o ; (post ?qt ˆ o)
    hence x ; ?qt ˆ o ; bot ⊔ 1 ≤ (?pt ; post ?qt ; ?qt ; top ⊓ 1) ˆ o
      by (smt dual-comp dual-inf dual-one dual-top mbta.add-left-isotone mbta.mult-left-isotone)
    also have ... = ?pt ˆ o
      by (metis post-1 mult-assoc assertion-prop neg-assertion)
    finally show ?pt ≤ wpt (x ˆ o ; ?qt)
      by (smt dual-comp dual-dual dual-le dual-neg-top dual-one dual-sup dual-top wpt-def)
  next
```

```
    fix x q
    let ?qt=neg-assert q
    have x ˆ o ; ?qt ˆ o ; bot ⊓ (post 1 ; neg-assert ?qt ˆ o) ≤ x ˆ o ; neg-assert ?qt ; neg-assert ?qt ˆ o
        by (smt bot-comp inf.commute inf-comp inf-top-left mbta.mult-left-isotone mult.left-neutral mult-assoc neg-assert-def
post-2)
    also have ... ≤ x ˆ o
      by (smt assert-iff-assume assumption-assertion-absorb dual-comp dual-dual le-comp mbta.a-below-one mbta.mult-right-one
mult-assoc neg-assertion)
    finally show x ≤ x ; ?qt ; top ⊔ post 1 ˆ o ; neg-assert ?qt
      by (smt dual-comp dual-dual dual-inf dual-le dual-top)
  qed

sublocale post-mbt-algebra < mbta-pre-dual!: pre-post-spec-Hd where box = λx y . neg-assume (x ∗ neg-assume y) and d =
λx . (x ∗ bot) ⊔ 1 and diamond = λx y . (x ∗ y ∗ bot) ⊔ 1 and less = greater and less-eq = greater-eq and plus = inf and
pre = λx y . wpb (x ; y) and pre-post = λp q . p ∗ (post (q ˆ o) ˆ o) and uminus = neg-assume and zero = top and Hd =
post 1 ˆ o and T = bot and Z = top
  apply unfold-locales
  apply (metis mbta-pre.H-zero)
  apply (metis mbta-pre.H-greatest-finite)
  apply (metis neg-assumption assumption-conjunctive conjunctiveD)
  apply rule
  defer
  apply (smt dual-comp dual-dual dual-top mbta-dual.a-d-closed mbta-dual.diamond-left-isotone mult-assoc post-1 wpb-def)
  apply (metis bot-comp mbta.add-right-zero mbta.mult-right-dist-add mult.assoc mult.left-neutral)
  proof −
    fix p x q
    let ?pt=neg-assume p
    let ?qt=neg-assume q
    assume wpb (x ; ?qt) ≤ ?pt
    hence ?pt ˆ o ; post (?qt ˆ o) ≤ (x ˆ o ; ?qt ˆ o ; top ⊓ 1) ; post (?qt ˆ o)
      by (smt dual-comp dual-dual dual-le dual-one dual-sup dual-top le-comp-right wpb-def)
    also have ... ≤ x ˆ o
      by (smt inf-comp mult.left-neutral mult-assoc post-2 top-comp)
    finally show x ≤ ?pt ; post (?qt ˆ o) ˆ o
      by (smt dual-comp dual-dual dual-le)
  qed

sublocale post-mbt-algebra < mbta-dual!: pre-post-spec-whiledo where ite = λx p y . (p ∗ x) ⊓ (neg-assume p ∗ y) and less
= greater and less-eq = greater-eq and plus = inf and pre = λx y . wpb (x ˆ o ; y) and pre-post = λp q . (p ˆ o) ∗ post (q
ˆ o) and uminus = neg-assume and while = λp x . ((p ∗ x) ˆ ω) ∗ neg-assume p and zero = top and T = bot ..

sublocale post-mbt-algebra < mbta-fix-dual!: pre-post-spec-whiledo where ite = λx p y . (p ∗ x) ⊓ (neg-assume p ∗ y) and
less = greater and less-eq = greater-eq and plus = inf and pre = λx y . wpb (x ˆ o ; y) and pre-post = λp q . (p ˆ o) ∗ post
(q ˆ o) and uminus = neg-assume and while = λp x . ((p ∗ x) ˆ ∗) ∗ neg-assume p and zero = top and T = bot ..

sublocale post-mbt-algebra < mbta-pre!: pre-post-spec-whiledo where ite = λx p y . (p ∗ x) ⊔ (neg-assert p ∗ y) and plus =
sup and pre = λx y . wpt (x ˆ o ∗ y) and pre-post = λp q . p ˆ o ∗ (post q ˆ o) and uminus = neg-assert and while = λp x
. ((p ∗ x) ˆ ⊗) ∗ neg-assert p and zero = bot and T = top ..

sublocale post-mbt-algebra < mbta-pre-fix!: pre-post-spec-whiledo where ite = λx p y . (p ∗ x) ⊔ (neg-assert p ∗ y) and plus
= sup and pre = λx y . wpt (x ˆ o ∗ y) and pre-post = λp q . p ˆ o ∗ (post q ˆ o) and uminus = neg-assert and while =
λp x . ((p ∗ x) ˆ ℧) ∗ neg-assert p and zero = bot and T = top ..

sublocale post-mbt-algebra < mbta-dual!: pre-post-L where box = λx y . neg-assume (x ∗ neg-assume y) and circ = omega
and d = λx . (x ∗ bot) ⊔ 1 and diamond = λx y . (x ∗ y ∗ bot) ⊔ 1 and ite = λx p y . (p ∗ x) ⊓ (neg-assume p ∗ y) and
less = greater and less-eq = greater-eq and plus = inf and pre = λx y . wpb (x ˆ o ; y) and pre-post = λp q . (p ˆ o) ∗ post
(q ˆ o) and uminus = neg-assume and while = λp x . ((p ∗ x) ˆ ω) ∗ neg-assume p and zero = top and L = bot and T =
bot and Z = top
  apply unfold-locales
  apply (metis bot-least inf-bot-left)
  done

sublocale post-mbt-algebra < mbta-fix-dual!: pre-post-L where box = λx y . neg-assume (x ∗ neg-assume y) and circ = star
and d = λx . (x ∗ bot) ⊔ 1 and diamond = λx y . (x ∗ y ∗ bot) ⊔ 1 and ite = λx p y . (p ∗ x) ⊓ (neg-assume p ∗ y) and
less = greater and less-eq = greater-eq and plus = inf and pre = λx y . wpb (x ˆ o ; y) and pre-post = λp q . (p ˆ o) ∗ post
(q ˆ o) and uminus = neg-assume and while = λp x . ((p ∗ x) ˆ ∗) ∗ neg-assume p and zero = top and L = top and T =
bot and Z = top
  apply unfold-locales
  apply (metis eq-refl inf-top-left)
```

done

sublocale *post-mbt-algebra* < *mbta-pre*!: *pre-post-L* **where** *box* = λ*x y* . *neg-assert* (*x* ∗ *neg-assert y*) **and** *circ* = *dual-star* **and** *d* = λ*x* . (*x* ∗ *top*) ⊓ *1* **and** *diamond* = λ*x y* . (*x* ∗ *y* ∗ *top*) ⊓ *1* **and** *ite* = λ*x p y* . (*p* ∗ *x*) ⊔ (*neg-assert p* ∗ *y*) **and** *plus* = *sup* **and** *pre* = λ*x y* . *wpt* (*x* ˆ *o* ∗ *y*) **and** *pre-post* = λ*p q* . *p* ˆ *o* ∗ (*post q* ˆ *o*) **and** *star* = *dual-star* **and** *uminus* = *neg-assert* **and** *while* = λ*p x* . ((*p* ∗ *x*) ˆ ⊗) ∗ *neg-assert p* **and** *zero* = *bot* **and** *L* = *bot* **and** *T* = *top* **and** *Z* = *bot*
apply *unfold-locales*
apply (*metis mbta.add-right-upper-bound*)
done

sublocale *post-mbt-algebra* < *mbta-pre-fix*!: *pre-post-L* **where** *box* = λ*x y* . *neg-assert* (*x* ∗ *neg-assert y*) **and** *circ* = *dual-omega* **and** *d* = λ*x* . (*x* ∗ *top*) ⊓ *1* **and** *diamond* = λ*x y* . (*x* ∗ *y* ∗ *top*) ⊓ *1* **and** *ite* = λ*x p y* . (*p* ∗ *x*) ⊔ (*neg-assert p* ∗ *y*) **and** *plus* = *sup* **and** *pre* = λ*x y* . *wpt* (*x* ˆ *o* ∗ *y*) **and** *pre-post* = λ*p q* . *p* ˆ *o* ∗ (*post q* ˆ *o*) **and** *star* = *dual-star* **and** *uminus* = *neg-assert* **and** *while* = λ*p x* . ((*p* ∗ *x*) ˆ ℧) ∗ *neg-assert p* **and** *zero* = *bot* **and** *L* = *top* **and** *T* = *top* **and** *Z* = *bot*
apply *unfold-locales*
apply (*metis mbta.add-left-top top-greatest*)
done

sublocale *complete-mbt-algebra* < *mbta*!: *complete-tests* **where** *plus* = *sup* **and** *uminus* = *neg-assert* **and** *zero* = *bot*
apply *unfold-locales*
apply (*smt mbta.test-set-def neg-assertion subset-eq Sup-assertion assertion-neg-assert*)
apply (*metis Sup-upper*)
apply (*metis Sup-least*)
done

sublocale *complete-mbt-algebra* < *mbta-dual*!: *complete-tests* **where** *less* = *greater* **and** *less-eq* = *greater-eq* **and** *plus* = *inf* **and** *uminus* = *neg-assume* **and** *zero* = *top* **and** *Inf* = *Sup* **and** *Sup* = *Inf*
apply *unfold-locales*
apply (*smt mbta-dual.test-set-def neg-assumption subset-eq Inf-assumption assumption-neg-assume*)
apply (*metis Inf-lower*)
apply (*metis Inf-greatest*)
done

sublocale *complete-mbt-algebra* < *mbta*!: *complete-antidomain-semiring* **where** *d* = λ*x* . (*x* ∗ *top*) ⊓ *1* **and** *plus* = *sup* **and** *uminus* = *neg-assert* **and** *zero* = *bot* **and** *Z* = *bot*
apply *unfold-locales*
apply *rule*
unfolding *mbta.Sum-def mbta.Prod-def neg-assert-def dual-Inf dual-Sup INF-def SUP-def Inf-comp Sup-comp*
unfolding *inf-commute*
apply (*subst inf-Inf*)
apply (*metis* (*mono-tags*) *empty-Collect-eq image-is-empty*)
apply (*rule arg-cong*[**where** *f*=*Inf*])
apply *auto*
unfolding *inf-Sup SUP-def*
apply (*rule arg-cong*[**where** *f*=*Sup*])
apply *auto*
apply (*smt2 comp-apply image-eqI mem-Collect-eq*)
done

sublocale *complete-mbt-algebra* < *mbta-dual*!: *complete-antidomain-semiring* **where** *d* = λ*x* . (*x* ∗ *bot*) ⊔ *1* **and** *less* = *greater* **and** *less-eq* = *greater-eq* **and** *plus* = *inf* **and** *uminus* = *neg-assume* **and** *zero* = *top* **and** *Inf* = *Sup* **and** *Sup* = *Inf* **and** *Z* = *top*
apply *unfold-locales*
apply *rule*
unfolding *mbta-dual.Sum-def mbta-dual.Prod-def neg-assume-def dual-Inf dual-Sup INF-def SUP-def Inf-comp Sup-comp*
unfolding *sup-commute*
apply (*subst sup-Sup*)
apply (*metis* (*mono-tags*) *empty-Collect-eq image-is-empty*)
apply (*rule arg-cong*[**where** *f*=*Sup*])
apply *auto*
unfolding *sup-Inf INF-def*
apply (*rule arg-cong*[**where** *f*=*Inf*])
apply *auto*
apply (*smt2 comp-apply image-eqI mem-Collect-eq*)
done

sublocale *complete-mbt-algebra* < *mbta*!: *diamond-while-program* **where** *box* = λ*x y* . *neg-assert* (*x* ∗ *neg-assert y*) **and** *circ* = *dual-star* **and** *d* = λ*x* . (*x* ∗ *top*) ⊓ *1* **and** *diamond* = λ*x y* . (*x* ∗ *y* ∗ *top*) ⊓ *1* **and** *ite* = λ*x p y* . (*p* ∗ *x*) ⊔ (*neg-assert p* ∗ *y*) **and** *plus* = *sup* **and** *pre* = λ*x y* . *wpt* (*x* ∗ *y*) **and** *uminus* = *neg-assert* **and** *while* = λ*p x* . ((*p* ∗ *x*) ˆ ⊗) ∗ *neg-assert*

p **and** *zero* = *bot* **and** *Atomic-program* = *Continuous* **and** *Atomic-test* = *assertion* **and** *T* = *top* **and** *Z* = *bot*
  apply *unfold-locales*
  apply (*metis one-continuous*)
  apply *simp-all*
  done

sublocale *complete-mbt-algebra* < *mbta-dual*!: *box-while-program* **where** *box* = λ*x y* . *neg-assume* (*x* ∗ *neg-assume y*) **and** *circ* = *omega* **and** *d* = λ*x* . (*x* ∗ *bot*) ⊔ *1* **and** *diamond* = λ*x y* . (*x* ∗ *y* ∗ *bot*) ⊔ *1* **and** *ite* = λ*x p y* . (*p* ∗ *x*) ⊓ (*neg-assume p* ∗ *y*) **and** *less* = *greater* **and** *less-eq* = *greater-eq* **and** *plus* = *inf* **and** *pre* = λ*x y* . *wpb* (*x* ˆ *o* ; *y*) **and** *uminus* = *neg-assume* **and** *while* = λ*p x* . ((*p* ∗ *x*) ˆ ω) ∗ *neg-assume p* **and** *zero* = *top* **and** *Atomic-program* = *Continuous* **and** *Atomic-test* = *assumption* **and** *T* = *bot* **and** *Z* = *top*
  apply *unfold-locales*
  apply (*metis one-continuous*)
  apply *simp-all*
  done

sublocale *complete-mbt-algebra* < *mbta-fix*!: *diamond-while-program* **where** *box* = λ*x y* . *neg-assert* (*x* ∗ *neg-assert y*) **and** *circ* = *dual-omega* **and** *d* = λ*x* . (*x* ∗ *top*) ⊓ *1* **and** *diamond* = λ*x y* . (*x* ∗ *y* ∗ *top*) ⊓ *1* **and** *ite* = λ*x p y* . (*p* ∗ *x*) ⊔ (*neg-assert p* ∗ *y*) **and** *plus* = *sup* **and** *pre* = λ*x y* . *wpt* (*x* ∗ *y*) **and** *uminus* = *neg-assert* **and** *while* = λ*p x* . ((*p* ∗ *x*) ˆ ℧) ∗ *neg-assert p* **and** *zero* = *bot* **and** *Atomic-program* = *Cocontinuous* **and** *Atomic-test* = *assertion* **and** *T* = *top* **and** *Z* = *bot*
  apply *unfold-locales*
  apply (*metis one-cocontinuous*)
  apply *simp-all*
  done

sublocale *complete-mbt-algebra* < *mbta-fix-dual*!: *box-while-program* **where** *box* = λ*x y* . *neg-assume* (*x* ∗ *neg-assume y*) **and** *circ* = *star* **and** *d* = λ*x* . (*x* ∗ *bot*) ⊔ *1* **and** *diamond* = λ*x y* . (*x* ∗ *y* ∗ *bot*) ⊔ *1* **and** *ite* = λ*x p y* . (*p* ∗ *x*) ⊓ (*neg-assume p* ∗ *y*) **and** *less* = *greater* **and** *less-eq* = *greater-eq* **and** *plus* = *inf* **and** *pre* = λ*x y* . *wpb* (*x* ˆ *o* ; *y*) **and** *uminus* = *neg-assume* **and** *while* = λ*p x* . ((*p* ∗ *x*) ˆ ∗) ∗ *neg-assume p* **and** *zero* = *top* **and** *Atomic-program* = *Cocontinuous* **and** *Atomic-test* = *assumption* **and** *T* = *bot* **and** *Z* = *top*
  apply *unfold-locales*
  apply (*metis one-cocontinuous*)
  apply *simp-all*
  done

sublocale *complete-mbt-algebra* < *mbta-pre*!: *box-while-program* **where** *box* = λ*x y* . *neg-assert* (*x* ∗ *neg-assert y*) **and** *circ* = *dual-star* **and** *d* = λ*x* . (*x* ∗ *top*) ⊓ *1* **and** *diamond* = λ*x y* . (*x* ∗ *y* ∗ *top*) ⊓ *1* **and** *ite* = λ*x p y* . (*p* ∗ *x*) ⊔ (*neg-assert p* ∗ *y*) **and** *plus* = *sup* **and** *pre* = λ*x y* . *wpt* (*x* ˆ *o* ∗ *y*) **and** *uminus* = *neg-assert* **and** *while* = λ*p x* . ((*p* ∗ *x*) ˆ ⊗) ∗ *neg-assert p* **and** *zero* = *bot* **and** *Atomic-program* = *Continuous* **and** *Atomic-test* = *assertion* **and** *T* = *top* **and** *Z* = *bot* **..**

sublocale *complete-mbt-algebra* < *mbta-pre-dual*!: *diamond-while-program* **where** *box* = λ*x y* . *neg-assume* (*x* ∗ *neg-assume y*) **and** *circ* = *omega* **and** *d* = λ*x* . (*x* ∗ *bot*) ⊔ *1* **and** *diamond* = λ*x y* . (*x* ∗ *y* ∗ *bot*) ⊔ *1* **and** *ite* = λ*x p y* . (*p* ∗ *x*) ⊓ (*neg-assume p* ∗ *y*) **and** *less* = *greater* **and** *less-eq* = *greater-eq* **and** *plus* = *inf* **and** *pre* = λ*x y* . *wpb* (*x* ; *y*) **and** *uminus* = *neg-assume* **and** *while* = λ*p x* . ((*p* ∗ *x*) ˆ ω) ∗ *neg-assume p* **and** *zero* = *top* **and** *Atomic-program* = *Continuous* **and** *Atomic-test* = *assumption* **and** *T* = *bot* **and** *Z* = *top* **..**

sublocale *complete-mbt-algebra* < *mbta-pre-fix*!: *box-while-program* **where** *box* = λ*x y* . *neg-assert* (*x* ∗ *neg-assert y*) **and** *circ* = *dual-omega* **and** *d* = λ*x* . (*x* ∗ *top*) ⊓ *1* **and** *diamond* = λ*x y* . (*x* ∗ *y* ∗ *top*) ⊓ *1* **and** *ite* = λ*x p y* . (*p* ∗ *x*) ⊔ (*neg-assert p* ∗ *y*) **and** *plus* = *sup* **and** *pre* = λ*x y* . *wpt* (*x* ˆ *o* ∗ *y*) **and** *uminus* = *neg-assert* **and** *while* = λ*p x* . ((*p* ∗ *x*) ˆ ℧) ∗ *neg-assert p* **and** *zero* = *bot* **and** *Atomic-program* = *Cocontinuous* **and** *Atomic-test* = *assertion* **and** *T* = *top* **and** *Z* = *bot* **..**

sublocale *complete-mbt-algebra* < *mbta-pre-fix-dual*!: *diamond-while-program* **where** *box* = λ*x y* . *neg-assume* (*x* ∗ *neg-assume y*) **and** *circ* = *star* **and** *d* = λ*x* . (*x* ∗ *bot*) ⊔ *1* **and** *diamond* = λ*x y* . (*x* ∗ *y* ∗ *bot*) ⊔ *1* **and** *ite* = λ*x p y* . (*p* ∗ *x*) ⊓ (*neg-assume p* ∗ *y*) **and** *less* = *greater* **and** *less-eq* = *greater-eq* **and** *plus* = *inf* **and** *pre* = λ*x y* . *wpb* (*x* ; *y*) **and** *uminus* = *neg-assume* **and** *while* = λ*p x* . ((*p* ∗ *x*) ˆ ∗) ∗ *neg-assume p* **and** *zero* = *top* **and** *Atomic-program* = *Cocontinuous* **and** *Atomic-test* = *assumption* **and** *T* = *bot* **and** *Z* = *top* **..**

— Theorem 52

sublocale *complete-mbt-algebra* < *mbta*!: *diamond-hoare-sound* **where** *box* = λ*x y* . *neg-assert* (*x* ∗ *neg-assert y*) **and** *circ* = *dual-star* **and** *d* = λ*x* . (*x* ∗ *top*) ⊓ *1* **and** *diamond* = λ*x y* . (*x* ∗ *y* ∗ *top*) ⊓ *1* **and** *ite* = λ*x p y* . (*p* ∗ *x*) ⊔ (*neg-assert p* ∗ *y*) **and** *plus* = *sup* **and** *pre* = λ*x y* . *wpt* (*x* ∗ *y*) **and** *star* = *dual-star* **and** *uminus* = *neg-assert* **and** *while* = λ*p x* . ((*p* ∗ *x*) ˆ ⊗) ∗ *neg-assert p* **and** *zero* = *bot* **and** *Atomic-program* = *Continuous* **and** *Atomic-test* = *assertion* **and** *T* = *top* **and** *Z* = *bot*
  apply *unfold-locales*
  apply (*metis bot-comp bot-least mbta.aL-one-circ mbta.d-Z mbta.one-circ-L*)
  done

— Theorem 52

sublocale complete-mbt-algebra < mbta-dual!: box-hoare-sound **where** box = λx y . neg-assume (x ∗ neg-assume y) **and** circ = omega **and** d = λx . (x ∗ bot) ⊔ 1 **and** diamond = λx y . (x ∗ y ∗ bot) ⊔ 1 **and** ite = λx p y . (p ∗ x) ⊓ (neg-assume p ∗ y) **and** less = greater **and** less-eq = greater-eq **and** plus = inf **and** pre = λx y . wpb (x ˆ o ; y) **and** uminus = neg-assume **and** while = λp x . ((p ∗ x) ˆ ω) ∗ neg-assume p **and** zero = top **and** Atomic-program = Continuous **and** Atomic-test = assumption **and** Inf = Sup **and** Sup = Inf **and** T = bot **and** Z = top
  apply unfold-locales
  apply (metis bot-comp mbta.top-left-zero mbta-dual.Omega-one mbta-dual.aL-one-circ mbta-dual.a-T top-greatest)
  done

— Theorem 52

sublocale complete-mbt-algebra < mbta-fix!: diamond-hoare-sound-2 **where** box = λx y . neg-assert (x ∗ neg-assert y) **and** circ = dual-omega **and** d = λx . (x ∗ top) ⊓ 1 **and** diamond = λx y . (x ∗ y ∗ top) ⊓ 1 **and** ite = λx p y . (p ∗ x) ⊔ (neg-assert p ∗ y) **and** plus = sup **and** pre = λx y . wpt (x ∗ y) **and** star = dual-star **and** uminus = neg-assert **and** while = λp x . ((p ∗ x) ˆ ℧) ∗ neg-assert p **and** zero = bot **and** Atomic-program = Cocontinuous **and** Atomic-test = assertion **and** T = top **and** Z = bot
  apply unfold-locales
  proof
    fix p q x
    let ?pt = neg-assert p
    let ?qt = neg-assert q
    assume neg-assert ?pt ; ?qt ≤ x ; ?qt ; top ⊓ 1
    hence ?qt ; top ≤ x ˆ ℧ ; ?pt ; top
      by (smt mbta.Omega-induct mbta.d-def mbta.d-mult-top mbta.mult-left-isotone mbta.shunting-T-1 mult.assoc)
    thus mbta-fix.aL ; ?qt ≤ x ˆ ℧ ; ?pt ; top ⊓ 1
      by (smt inf.commute inf-top-left mbta.Omega-one mbta.d-T mbta-dual.Omega.circ-isotone mbta-dual.Omega.mult-top-circ-1 mbta-dual.Z-top mbta-dual.mult-L-circ-mult mbta-fix.aL-one-circ mult.assoc mult.left-neutral neg-assert-def)
  qed

— Theorem 52

sublocale complete-mbt-algebra < mbta-fix-dual!: box-hoare-sound **where** box = λx y . neg-assume (x ∗ neg-assume y) **and** circ = star **and** d = λx . (x ∗ bot) ⊔ 1 **and** diamond = λx y . (x ∗ y ∗ bot) ⊔ 1 **and** ite = λx p y . (p ∗ x) ⊓ (neg-assume p ∗ y) **and** less = greater **and** less-eq = greater-eq **and** plus = inf **and** pre = λx y . wpb (x ˆ o ; y) **and** uminus = neg-assume **and** while = λp x . ((p ∗ x) ˆ ∗) ∗ neg-assume p **and** zero = top **and** Atomic-program = Cocontinuous **and** Atomic-test = assumption **and** Inf = Sup **and** Sup = Inf **and** T = bot **and** Z = top
  apply unfold-locales
  apply (metis eq-refl mbta-dual.a-Z mbta-fix-dual.aL-one-circ mbta-fix-dual.one-circ-L mult.left-neutral)
  done

— Theorem 52

sublocale complete-mbt-algebra < mbta-pre!: box-hoare-sound **where** box = λx y . neg-assert (x ∗ neg-assert y) **and** circ = dual-star **and** d = λx . (x ∗ top) ⊓ 1 **and** diamond = λx y . (x ∗ y ∗ top) ⊓ 1 **and** ite = λx p y . (p ∗ x) ⊔ (neg-assert p ∗ y) **and** plus = sup **and** pre = λx y . wpt (x ˆ o ∗ y) **and** star = dual-star **and** uminus = neg-assert **and** while = λp x . ((p ∗ x) ˆ ⊗) ∗ neg-assert p **and** zero = bot **and** Atomic-program = Continuous **and** Atomic-test = assertion **and** T = top **and** Z = bot
  apply unfold-locales
  apply (metis eq-refl mbta.a-Z mbta.one-circ-L mbta-pre.aL-one-circ mult.left-neutral)
  done

— Theorem 52

sublocale complete-mbt-algebra < mbta-pre-dual!: diamond-hoare-sound-2 **where** box = λx y . neg-assume (x ∗ neg-assume y) **and** circ = omega **and** d = λx . (x ∗ bot) ⊔ 1 **and** diamond = λx y . (x ∗ y ∗ bot) ⊔ 1 **and** ite = λx p y . (p ∗ x) ⊓ (neg-assume p ∗ y) **and** less = greater **and** less-eq = greater-eq **and** plus = inf **and** pre = λx y . wpb (x ; y) **and** uminus = neg-assume **and** while = λp x . ((p ∗ x) ˆ ω) ∗ neg-assume p **and** zero = top **and** Atomic-program = Continuous **and** Atomic-test = assumption **and** Inf = Sup **and** Sup = Inf **and** T = bot **and** Z = top
  apply unfold-locales
  proof
    fix p q x
    let ?pt = neg-assume p
    let ?qt = neg-assume q
    assume x ; ?qt ; bot ⊔ 1 ≤ neg-assume ?pt ; ?qt
    hence x ; ?qt ; bot ⊓ ?pt ≤ ?qt
      by (smt inf-commute inf-le1 le-supE mbta-dual.a-compl-intro mbta-dual.add-right-isotone mbta-dual.d-def order-trans)
    hence (x ; ?qt ; bot ⊓ ?pt) ; bot ≤ ?qt ; bot
      by (smt mbta.mult-left-isotone)

  hence *x ˆ ω ; ?pt ; bot* ⊔ *1* ≤ *?qt*
    by (*smt bot-comp inf-comp mbta.add-left-isotone mbta-dual.a-d-closed mult-assoc omega-least*)
  thus *x ˆ ω ; ?pt ; bot* ⊔ *1* ≤ *mbta-pre-dual.aL ; ?qt*
    by (*metis bot-comp mbta.add-right-zero mbta-dual.Omega-one mbta-pre-dual.aL-one-circ mult.left-neutral sup.commute*)
 qed

— Theorem 52

sublocale *complete-mbt-algebra* < *mbta-pre-fix*!: *box-hoare-sound* **where** *box* = λ*x y* . *neg-assert* (*x* ∗ *neg-assert y*) **and** *circ* = *dual-omega* **and** *d* = λ*x* . (*x* ∗ *top*) ⊓ *1* **and** *diamond* = λ*x y* . (*x* ∗ *y* ∗ *top*) ⊓ *1* **and** *ite* = λ*x p y* . (*p* ∗ *x*) ⊔ (*neg-assert p* ∗ *y*) **and** *plus* = *sup* **and** *pre* = λ*x y* . *wpt* (*x* ˆ *o* ∗ *y*) **and** *star* = *dual-star* **and** *uminus* = *neg-assert* **and** *while* = λ*p x* . ((*p* ∗ *x*) ˆ ℧) ∗ *neg-assert p* **and** *zero* = *bot* **and** *Atomic-program* = *Cocontinuous* **and** *Atomic-test* = *assertion* **and** *T* = *top* **and** *Z* = *bot*
 apply *unfold-locales*
 apply (*metis bot-comp bot-least mbta.Omega-one mbta.a-T mbta-dual.Z-top mbta-pre-fix.aL-one-circ*)
 done

— Theorem 52

sublocale *complete-mbt-algebra* < *mbta-pre-fix-dual*!: *diamond-hoare-sound* **where** *box* = λ*x y* . *neg-assume* (*x* ∗ *neg-assume y*) **and** *circ* = *star* **and** *d* = λ*x* . (*x* ∗ *bot*) ⊔ *1* **and** *diamond* = λ*x y* . (*x* ∗ *y* ∗ *bot*) ⊔ *1* **and** *ite* = λ*x p y* . (*p* ∗ *x*) ⊓ (*neg-assume p* ∗ *y*) **and** *less* = *greater* **and** *less-eq* = *greater-eq* **and** *plus* = *inf* **and** *pre* = λ*x y* . *wpb* (*x* ; *y*) **and** *uminus* = *neg-assume* **and** *while* = λ*p x* . ((*p* ∗ *x*) ˆ ∗) ∗ *neg-assume p* **and** *zero* = *top* **and** *Atomic-program* = *Cocontinuous* **and** *Atomic-test* = *assumption* **and** *Inf* = *Sup* **and** *Sup* = *Inf* **and** *T* = *bot* **and** *Z* = *top*
 apply *unfold-locales*
 apply (*metis mbta.T-left-zero mbta.add-left-top mbta-pre-fix-dual.aL-one-circ mbta-fix-dual.L-def top-greatest*)
 done

— Theorem 52

sublocale *complete-mbt-algebra* < *mbta*!: *diamond-hoare-valid* **where** *box* = λ*x y* . *neg-assert* (*x* ∗ *neg-assert y*) **and** *circ* = *dual-star* **and** *d* = λ*x* . (*x* ∗ *top*) ⊓ *1* **and** *diamond* = λ*x y* . (*x* ∗ *y* ∗ *top*) ⊓ *1* **and** *hoare-triple* = λ*p x q* . *p* ≤ *wpt*(*x* ∗ *q*) **and** *ite* = λ*x p y* . (*p* ∗ *x*) ⊔ (*neg-assert p* ∗ *y*) **and** *plus* = *sup* **and** *pre* = λ*x y* . *wpt* (*x* ∗ *y*) **and** *star* = *dual-star* **and** *uminus* = *neg-assert* **and** *while* = λ*p x* . ((*p* ∗ *x*) ˆ ⊗) ∗ *neg-assert p* **and** *zero* = *bot* **and** *Atomic-program* = *Continuous* **and** *Atomic-test* = *assertion* **and** *T* = *top* **and** *Z* = *bot*
 apply *unfold-locales*
 apply (*metis mbta.aL-zero mbta.circ-circ-mult mbta.one-def mbta.star-involutive mbta.star-one order-refl*)
 apply (*metis mbta.aL-one-circ mbta.d-Z mbta.one-circ-L*)
 defer
 apply (*metis wpt-def*)
 unfolding *mbta.Sum-range SUP-def*[*THEN sym*]
 proof
  fix *x t*
  assume *1*: *x* ∈ *while-program.While-program op* ; *neg-assert Continuous assertion* (λ*p x* . (*p* ; *x*) ˆ ⊗ ; *neg-assert p*) (λ*x p y* . *p* ; *x* ⊔ *neg-assert p* ; *y*) ∧ *ascending-chain t* ∧ *tests.test-seq neg-assert t*
  have *x* ∈ *Continuous*
   apply (*induct x rule*: *while-program.While-program.induct*[**where** *pre*=λ*x y* . *wpt* (*x* ∗ *y*) **and** *while*=λ*p x* . ((*p* ∗ *x*) ˆ ⊗) ∗ *neg-assert p*])
   apply *unfold-locales*
   apply (*metis 1*)
   apply *metis*
   apply (*metis mult-continuous*)
   apply (*metis assertion-continuous mbta.test-expression-test mult-continuous neg-assertion sup-continuous*)
   apply (*metis assertion-continuous dual-star-continuous mbta.test-expression-test mult-continuous neg-assertion*)
   done
  thus *x* ; (*SUP n*::*nat* . *t n*) = (*SUP n*::*nat* . *x* ; *t n*) **using** *1*
   by (*smt continuous-dist-ascending-chain SUP-cong*)
 qed

— Theorem 52

sublocale *complete-mbt-algebra* < *mbta-dual*!: *box-hoare-valid* **where** *box* = λ*x y* . *neg-assume* (*x* ∗ *neg-assume y*) **and** *circ* = *omega* **and** *d* = λ*x* . (*x* ∗ *bot*) ⊔ *1* **and** *diamond* = λ*x y* . (*x* ∗ *y* ∗ *bot*) ⊔ *1* **and** *hoare-triple* = λ*p x q* . *wpb*(*x* ˆ *o* ∗ *q*) ≤ *p* **and** *ite* = λ*x p y* . (*p* ∗ *x*) ⊓ (*neg-assume p* ∗ *y*) **and** *less* = *greater* **and** *less-eq* = *greater-eq* **and** *plus* = *inf* **and** *pre* = λ*x y* . *wpb* (*x* ˆ *o* ; *y*) **and** *uminus* = *neg-assume* **and** *while* = λ*p x* . ((*p* ∗ *x*) ˆ ω) ∗ *neg-assume p* **and** *zero* = *top* **and** *Atomic-program* = *Continuous* **and** *Atomic-test* = *assumption* **and** *Inf* = *Sup* **and** *Sup* = *Inf* **and** *T* = *bot* **and** *Z* = *top*
 apply *unfold-locales*
 apply *rule*
 defer

```
  apply (metis bot-comp mbta-dual.Omega-one mbta-dual.aL-one-circ mbta-dual.a-T)
  prefer 2
  apply (metis mbta-dual.pre-def)
  unfolding mbta-dual.Prod-range SUP-def[THEN sym]
  proof
    fix x t
    assume 1: x ∈ while-program.While-program op ; neg-assume Continuous assumption (λp x . (p ; x) ˆ ω ; neg-assume p) (λx p y . (p ; x) ⊓ (neg-assume p ; y)) ∧ ord.descending-chain greater-eq t ∧ tests.test-seq neg-assume t
    have x ∈ Continuous
      apply (induct x rule: while-program.While-program.induct[where pre=λx y . wpb (x ˆ o ; y) and while=λp x . ((p ∗ x) ˆ ω) ∗ neg-assume p])
      apply unfold-locales
      apply (metis 1)
      apply metis
      apply (metis mult-continuous)
      apply (metis assumption-continuous mbta-dual.test-expression-test mult-continuous neg-assumption inf-continuous)
      apply (metis assumption-continuous omega-continuous mbta-dual.test-expression-test mult-continuous neg-assumption)
      done
    thus x ; (SUP n::nat . t n) = (SUP n::nat . x ; t n) using 1
      by (smt ord.descending-chain-def ascending-chain-def continuous-dist-ascending-chain SUP-cong)
  next
    fix p x q
    let ?pt = neg-assume p
    let ?qt = neg-assume q
    assume ?qt ≤ ?pt ; neg-assume (x ; neg-assume ?qt)
    also have ... ≤ x ˆ o ; ?qt ⊔ ?pt
      by (smt assumption-sup-comp-eq mbta.add-left-isotone mbta.zero-right-mult-decreasing mbta-dual.pre-def neg-assume-def neg-assumption sup.commute sup.left-commute sup.left-idem wpb-def)
    finally show ?qt ⊓ mbta-dual.aL ≤ neg-assume (x ˆ ω ; neg-assume ?pt)
      by (smt dual-dual dual-omega-def dual-omega-greatest le-infI1 mbta-dual.a-d-closed mbta-dual.d-isotone mbta-dual.pre-def wpb-def)
  qed
```

— Theorem 52

```
sublocale complete-mbt-algebra < mbta-pre-fix-dual!: diamond-hoare-valid where box = λx y . neg-assume (x ∗ neg-assume y) and circ = star and d = λx . (x ∗ bot) ⊔ 1 and diamond = λx y . (x ∗ y ∗ bot) ⊔ 1 and hoare-triple = λp x q . wpb(x ∗ q) ≤ p and ite = λx p y . (p ∗ x) ⊓ (neg-assume p ∗ y) and less = greater and less-eq = greater-eq and plus = inf and pre = λx y . wpb (x ; y) and uminus = neg-assume and while = λp x . ((p ∗ x) ˆ ∗) ∗ neg-assume p and zero = top and Atomic-program = Cocontinuous and Atomic-test = assumption and Inf = Sup and Sup = Inf and T = bot and Z = top
  apply unfold-locales
  apply (smt gt-one-comp mbta.add-commutative mbta.add-left-upper-bound neg-assume-def)
  apply (metis mbta.T-left-zero mbta.add-left-top mbta-pre-fix-dual.aL-one-circ mbta-fix-dual.L-def)
  defer
  apply (metis wpb-def)
  unfolding mbta-dual.Sum-range INF-def[THEN sym]
  proof
    fix x t
    assume 1: x ∈ while-program.While-program op ; neg-assume Cocontinuous assumption (λp x . (p ; x) ˆ ∗ ; neg-assume p) (λx p y . (p ; x) ⊓ (neg-assume p ; y)) ∧ ord.ascending-chain greater-eq t ∧ tests.test-seq neg-assume t
    have x ∈ Cocontinuous
      apply (induct x rule: while-program.While-program.induct[where pre=λx y . wpb (x ; y) and while=λp x . ((p ∗ x) ˆ ∗) ∗ neg-assume p])
      apply unfold-locales
      apply (metis 1)
      apply metis
      apply (metis mult-cocontinuous)
      apply (metis assumption-cocontinuous mbta-dual.test-expression-test mult-cocontinuous neg-assumption inf-cocontinuous)
      apply (metis assumption-cocontinuous star-cocontinuous mbta-dual.test-expression-test mult-cocontinuous neg-assumption)
      done
    thus x ; (INF n::nat . t n) = (INF n::nat . x ; t n) using 1
      by (smt descending-chain-def ord.ascending-chain-def cocontinuous-dist-descending-chain INF-cong)
  qed
```

— Theorem 52

```
sublocale complete-mbt-algebra < mbta-pre-fix!: box-hoare-valid where box = λx y . neg-assert (x ∗ neg-assert y) and circ = dual-omega and d = λx . (x ∗ top) ⊓ 1 and diamond = λx y . (x ∗ y ∗ top) ⊓ 1 and hoare-triple = λp x q . p ≤ wpt(x ˆ o ∗ q) and ite = λx p y . (p ∗ x) ⊔ (neg-assert p ∗ y) and plus = sup and pre = λx y . wpt (x ˆ o ∗ y) and star =
dual-star and uminus = neg-assert and while = λp x . ((p ∗ x) ^ ℧) ∗ neg-assert p and zero = bot and Atomic-program =
Cocontinuous and Atomic-test = assertion and T = top and Z = bot
  apply unfold-locales
  apply rule
  defer
  apply (metis mbta.Omega-one mbta.T-left-zero mbta.a-T mbta-pre-fix.aL-one-circ)
  prefer 2
  apply (metis mbta-pre.pre-def)
  unfolding mbta.Prod-range INF-def[THEN sym]
  proof
    fix x t
    assume 1: x ∈ while-program.While-program op ; neg-assert Cocontinuous assertion (λp x . (p ; x) ^ ℧ ; neg-assert p) (λx
p y . p ; x ⊔ neg-assert p ; y) ∧ descending-chain t ∧ tests.test-seq neg-assert t
    have x ∈ Cocontinuous
      apply (induct x rule: while-program.While-program.induct[where pre=λx y . wpt (x ^ o ; y) and while=λp x . ((p ∗ x)
^ ℧) ∗ neg-assert p])
      apply unfold-locales
      apply (metis 1)
      apply metis
      apply (metis mult-cocontinuous)
      apply (metis assertion-cocontinuous mbta.test-expression-test mult-cocontinuous neg-assertion sup-cocontinuous)
      apply (metis assertion-cocontinuous dual-omega-cocontinuous mbta.test-expression-test mult-cocontinuous neg-assertion)
      done
    thus x ; (INF n::nat . t n) = (INF n::nat . x ; t n) using 1
      by (smt descending-chain-def cocontinuous-dist-descending-chain INF-cong)
  next
    fix p x q
    let ?pt = neg-assert p
    let ?qt = neg-assert q
    assume 1: ?pt ; neg-assert (x ; neg-assert ?qt) ≤ ?qt
    have x ^ o ; ?qt ⊓ ?pt ≤ ?pt ; neg-assert (x ; neg-assert ?qt)
          by (smt inf-comp mbta.sub-comm mbta.top-right-mult-increasing mbta-dual.add-left-isotone mbta-pre.pre-def
mult.left-neutral mult-assoc top-comp wpt-def)
    also have ... ≤ ?qt using 1
      by metis
    finally have (x ^ o) ^ ω ; ?pt ; top ≤ ?qt ; top
      by (metis mbta.mult-left-isotone omega-least mult-assoc)
    hence neg-assert (x ^ ℧ ; neg-assert ?pt) ≤ ?qt
      by (smt dual-omega-def inf-mono mbta.d-a-closed mbta.d-def mbta-pre.pre-def order-refl wpt-def mbta.a-d-closed)
    thus neg-assert (x ^ ℧ ; neg-assert ?pt) ≤ ?qt ⊔ mbta-pre-fix.aL
      by (smt mbta.add-left-upper-bound order-trans)
  qed

sublocale complete-mbt-algebra < mbta-dual!: pre-post-spec-hoare where ite = λx p y . (p ∗ x) ⊓ (neg-assume p ∗ y) and
less = greater and less-eq = greater-eq and plus = inf and pre = λx y . wpb (x ^ o ; y) and pre-post = λp q . (p ^ o) ∗ post
(q ^ o) and uminus = neg-assume and while = λp x . ((p ∗ x) ^ ω) ∗ neg-assume p and zero = top and Atomic-program =
Continuous and Atomic-test = assumption and Inf = Sup and Sup = Inf and T = bot ..

sublocale complete-mbt-algebra < mbta-fix-dual!: pre-post-spec-hoare where ite = λx p y . (p ∗ x) ⊓ (neg-assume p ∗ y) and
less = greater and less-eq = greater-eq and plus = inf and pre = λx y . wpb (x ^ o ; y) and pre-post = λp q . (p ^ o) ∗ post
(q ^ o) and uminus = neg-assume and while = λp x . ((p ∗ x) ^ ∗) ∗ neg-assume p and zero = top and Atomic-program =
Cocontinuous and Atomic-test = assumption and Inf = Sup and Sup = Inf and T = bot ..

sublocale complete-mbt-algebra < mbta-pre!: pre-post-spec-hoare where ite = λx p y . (p ∗ x) ⊔ (neg-assert p ∗ y) and plus
= sup and pre = λx y . wpt (x ^ o ∗ y) and pre-post = λp q . p ^ o ∗ (post q ^ o) and uminus = neg-assert and while =
λp x . ((p ∗ x) ^ ⊗) ∗ neg-assert p and zero = bot and Atomic-program = Continuous and Atomic-test = assertion and T
= top ..

sublocale complete-mbt-algebra < mbta-pre-fix!: pre-post-spec-hoare where ite = λx p y . (p ∗ x) ⊔ (neg-assert p ∗ y) and
plus = sup and pre = λx y . wpt (x ^ o ∗ y) and pre-post = λp q . p ^ o ∗ (post q ^ o) and uminus = neg-assert and while
= λp x . ((p ∗ x) ^ ℧) ∗ neg-assert p and zero = bot and Atomic-program = Cocontinuous and Atomic-test = assertion and
T = top ..

end
```