

Algebra of Monotonic Boolean Transformers

Viorel Preoteasa

August 7, 2022

Abstract

Algebras of imperative programming languages have been successful in reasoning about programs. In general an algebra of programs is an algebraic structure with programs as elements and with program compositions (sequential composition, choice, skip) as algebra operations. Various versions of these algebras were introduced to model partial correctness, total correctness, refinement, demonic choice, and other aspects. We formalize here an algebra which can be used to model total correctness, refinement, demonic and angelic choice. The basic model of this algebra are monotonic Boolean transformers (monotonic functions from a Boolean algebra to itself).

Contents

1	Introduction	1
2	Monotonic Boolean Transformers	2
3	Algebra of Monotonic Boolean Transformers	9
3.1	Assertions	12
3.2	Weakest precondition of true	15
3.3	Monotonic Boolean transformers algebra with post condition statement	16
3.4	Complete monotonic Boolean transformers algebra	17
4	Boolean Algebra of Assertions	18
5	Program statements, Hoare and refinement rules	21

1 Introduction

Abstract algebra is a useful tool in mathematics. Rather than working with specific models like natural numbers and algebra of truth values, one could reason in a more abstract setting and obtain results which are more general

and applicable in different models. Algebras of logics are very important tools in studying various aspects of logical systems. Algebras of programming theories have also a significant contribution to the simplification of reasoning about programs. Programs are elements of an algebra and program compositions and program constants (sequential composition, choice, iteration, skip, fail) are the operations of the algebra. These operations satisfy a number of relations which are used for reasoning about programs. Kleene algebra with tests (KAT) [10] is an extension of Kleene algebra and it is suitable for reasoning about programs in a partial correctness framework. Various versions of Kleene algebras have been introduced, ranging from Kleene algebra with domain [7] and concurrent Kleene algebra [9] to an algebra for separation logic [6].

Refinement Calculus [1, 2, 5, 12] is a calculus based on (monotonic) predicate transformers suitable for program development in a total correctness framework. Within this calculus various aspects of imperative programming languages can be formalized. These include total correctness, partial correctness, demonic choice, and angelic choice. Demonic refinement algebra (DRA) was introduced in [15, 16] as a variation of KAT to allow also reasoning about total correctness. The intended model of DRA is the set of conjunctive predicate transformers and this algebra cannot represent angelic choice. General refinement algebra (GRA) was also introduced in [16], but few results were proved and they were mostly related to iteration. Although the intended model for GRA is the set of monotonic predicate transformers, GRA does not include the angelic choice operator. GRA has been further extended in [14] with enabledness and termination operators, and it was extended for probabilistic programs in [11].

This formalization is based on [13] where a different extension of GRA is introduced. In [13] GRA is extended with a dual operator [8, 3, 4, 5]. The intended model for this algebra is the set of monotonic Boolean transformers (monotonic functions from a Boolean algebra to itself).

This formalization is structured as follows. Section 2 introduces the monotonic Boolean transformers that are the basic model of the algebra. Section 3 introduces the monotonic Boolean transformers algebra and some of its properties. Section 4 introduces the Boolean algebra of assertions. In section 5 we introduce standard program statements and we prove their Hoare total correctness rules.

2 Monotonic Boolean Transformers

theory *Mono-Bool-Tran*

imports

LatticeProperties.Complete-Lattice-Prop

LatticeProperties.Conj-Disj

begin

The type of monotonic transformers is the type associated to the set of monotonic functions from a partially ordered set (poset) to itself. The type of monotonic transformers with the pointwise extended order is also a poset. The monotonic transformers with composition and identity form a monoid, and the monoid operation is compatible with the order.

Gradually we extend the algebraic structure of monotonic transformers to lattices, and complete lattices. We also introduce a dual operator $((\text{dual } f)p = -f(-p))$ on monotonic transformers over a boolean algebra. However the monotonic transformers over a boolean algebra are not closed to the pointwise extended negation operator.

Finally we introduce an iteration operator on monotonic transformers over a complete lattice.

unbundle *lattice-syntax*

lemma *Inf-comp-fun:*

$$\prod M \circ f = (\prod m \in M. m \circ f)$$

<proof>

lemma *INF-comp-fun:*

$$(\prod a \in A. g \ a) \circ f = (\prod a \in A. g \ a \circ f)$$

<proof>

lemma *Sup-comp-fun:*

$$\bigsqcup M \circ f = (\bigsqcup m \in M. m \circ f)$$

<proof>

lemma *SUP-comp-fun:*

$$(\bigsqcup a \in A. g \ a) \circ f = (\bigsqcup a \in A. g \ a \circ f)$$

<proof>

lemma *(in order) mono-const [simp]:*

$$\text{mono } (\lambda-. c)$$

<proof>

lemma *(in order) mono-id [simp]:*

$$\text{mono } \text{id}$$

<proof>

lemma *(in order) mono-comp [simp]:*

$$\text{mono } f \implies \text{mono } g \implies \text{mono } (f \circ g)$$

<proof>

lemma *(in bot) mono-bot [simp]:*

$$\text{mono } \perp$$

<proof>

lemma *(in top) mono-top [simp]:*

```

mono  $\top$ 
<proof>

lemma (in semilattice-inf) mono-inf [simp]:
  assumes mono f and mono g
  shows mono (f  $\sqcap$  g)
<proof>

lemma (in semilattice-sup) mono-sup [simp]:
  assumes mono f and mono g
  shows mono (f  $\sqcup$  g)
<proof>

lemma (in complete-lattice) mono-Inf [simp]:
  assumes  $A \subseteq \{f :: 'a \Rightarrow 'b :: complete-lattice. mono f\}$ 
  shows mono ( $\prod A$ )
<proof>

lemma (in complete-lattice) mono-Sup [simp]:
  assumes  $A \subseteq \{f :: 'a \Rightarrow 'b :: complete-lattice. mono f\}$ 
  shows mono ( $\bigsqcup A$ )
<proof>

typedef (overloaded) 'a MonoTran = {f :: 'a :: order  $\Rightarrow$  'a . mono f}
<proof>

lemma [simp]:
  mono (Rep-MonoTran f)
<proof>

setup-lifting type-definition-MonoTran

instantiation MonoTran :: (order) order
begin

lift-definition less-eq-MonoTran :: 'a MonoTran  $\Rightarrow$  'a MonoTran  $\Rightarrow$  bool
  is less-eq <proof>

lift-definition less-MonoTran :: 'a MonoTran  $\Rightarrow$  'a MonoTran  $\Rightarrow$  bool
  is less <proof>

instance
  <proof>

end

instantiation MonoTran :: (order) monoid-mult
begin

```

```

lift-definition one-MonoTran :: 'a MonoTran
  is id
  ⟨proof⟩

lift-definition times-MonoTran :: 'a MonoTran ⇒ 'a MonoTran ⇒ 'a MonoTran
  is comp
  ⟨proof⟩

instance
  ⟨proof⟩

end

instantiation MonoTran :: (order-bot) order-bot
begin

lift-definition bot-MonoTran :: 'a MonoTran
  is  $\perp$ 
  ⟨proof⟩

instance
  ⟨proof⟩

end

instantiation MonoTran :: (order-top) order-top
begin

lift-definition top-MonoTran :: 'a MonoTran
  is  $\top$ 
  ⟨proof⟩

instance
  ⟨proof⟩

end

instantiation MonoTran :: (lattice) lattice
begin

lift-definition inf-MonoTran :: 'a MonoTran ⇒ 'a MonoTran ⇒ 'a MonoTran
  is inf
  ⟨proof⟩

lift-definition sup-MonoTran :: 'a MonoTran ⇒ 'a MonoTran ⇒ 'a MonoTran
  is sup
  ⟨proof⟩

instance

```

```

    <proof>

end

instance MonoTran :: (distrib-lattice) distrib-lattice
    <proof>

instantiation MonoTran :: (complete-lattice) complete-lattice
begin

lift-definition Inf-MonoTran :: 'a MonoTran set  $\Rightarrow$  'a MonoTran
    is Inf
    <proof>

lift-definition Sup-MonoTran :: 'a MonoTran set  $\Rightarrow$  'a MonoTran
    is Sup
    <proof>

instance
    <proof>

end

context includes lifting-syntax
begin

lemma [transfer-rule]:
    (rel-set A  $\Longrightarrow$  (A  $\Longrightarrow$  pcr-MonoTran HOL.eq)  $\Longrightarrow$  pcr-MonoTran
    HOL.eq) ( $\lambda$ A f.  $\sqcap$ (f ' A)) ( $\lambda$ A f.  $\sqcap$ (f ' A))
    <proof>

lemma [transfer-rule]:
    (rel-set A  $\Longrightarrow$  (A  $\Longrightarrow$  pcr-MonoTran HOL.eq)  $\Longrightarrow$  pcr-MonoTran
    HOL.eq) ( $\lambda$ A f.  $\sqcup$ (f ' A)) ( $\lambda$ A f.  $\sqcup$ (f ' A))
    <proof>

end

instance MonoTran :: (complete-distrib-lattice) complete-distrib-lattice
    <proof>

definition
    dual-fun (f::'a::boolean-algebra  $\Rightarrow$  'a) = uminus  $\circ$  f  $\circ$  uminus

lemma dual-fun-apply [simp]:
    dual-fun f p = - f (- p)
    <proof>

```

lemma *mono-dual-fun* [*simp*]:
 $mono\ f \implies mono\ (dual\text{-}fun\ f)$
 ⟨*proof*⟩

lemma (*in order*) *mono-inf-fun* [*simp*]:
fixes $x :: 'b::semilattice\text{-}inf$
shows $mono\ (inf\ x)$
 ⟨*proof*⟩

lemma (*in order*) *mono-sup-fun* [*simp*]:
fixes $x :: 'b::semilattice\text{-}sup$
shows $mono\ (sup\ x)$
 ⟨*proof*⟩

lemma *mono-comp-fun*:
fixes $f :: 'a::order \Rightarrow 'b::order$
shows $mono\ f \implies mono\ ((\circ)\ f)$
 ⟨*proof*⟩

definition
 $\Omega\text{-}fun\ f\ g = inf\ g \circ comp\ f$

lemma *Omega-fun-apply* [*simp*]:
 $\Omega\text{-}fun\ f\ g\ h\ p = (g\ p \sqcap f\ (h\ p))$
 ⟨*proof*⟩

lemma *mono-Omega-fun* [*simp*]:
 $mono\ f \implies mono\ (\Omega\text{-}fun\ f\ g)$
 ⟨*proof*⟩

lemma *mono-mono-Omega-fun* [*simp*]:
fixes $f :: 'b::order \Rightarrow 'a::semilattice\text{-}inf$ **and** $g :: 'c::semilattice\text{-}inf \Rightarrow 'a$
shows $mono\ f \implies mono\ g \implies mono\text{-}mono\ (\Omega\text{-}fun\ f\ g)$
 ⟨*proof*⟩

definition
 $\omega\text{-}fun\ f = lfp\ (\Omega\text{-}fun\ f\ id)$

definition
 $star\text{-}fun\ f = gfp\ (\Omega\text{-}fun\ f\ id)$

lemma *mono-omega-fun* [*simp*]:
fixes $f :: 'a::complete\text{-}lattice \Rightarrow 'a$
assumes $mono\ f$
shows $mono\ (\omega\text{-}fun\ f)$
 ⟨*proof*⟩

lemma *mono-star-fun* [*simp*]:

fixes $f :: 'a::\text{complete-lattice} \Rightarrow 'a$
assumes $\text{mono } f$
shows $\text{mono } (\text{star-fun } f)$
 $\langle \text{proof} \rangle$

lemma $\text{lfm-omega-lowerbound}$:
 $\text{mono } f \Longrightarrow \text{Omega-fun } f \ g \ A \leq A \Longrightarrow \text{omega-fun } f \circ g \leq A$
 $\langle \text{proof} \rangle$

lemma $\text{gfm-omega-upperbound}$:
 $\text{mono } f \Longrightarrow A \leq \text{Omega-fun } f \ g \ A \Longrightarrow A \leq \text{star-fun } f \circ g$
 $\langle \text{proof} \rangle$

lemma $\text{lfm-omega-greatest}$:
assumes $\bigwedge u. \text{Omega-fun } f \ g \ u \leq u \Longrightarrow A \leq u$
shows $A \leq \text{omega-fun } f \circ g$
 $\langle \text{proof} \rangle$

lemma gfm-star-least :
assumes $\bigwedge u. u \leq \text{Omega-fun } f \ g \ u \Longrightarrow u \leq A$
shows $\text{star-fun } f \circ g \leq A$
 $\langle \text{proof} \rangle$

lemma lfm-omega :
 $\text{mono } f \Longrightarrow \text{omega-fun } f \circ g = \text{lfm } (\text{Omega-fun } f \ g)$
 $\langle \text{proof} \rangle$

lemma gfm-star :
 $\text{mono } f \Longrightarrow \text{star-fun } f \circ g = \text{gfm } (\text{Omega-fun } f \ g)$
 $\langle \text{proof} \rangle$

definition
 $\text{assert-fun } p \ q = (p \sqcap q :: 'a::\text{semilattice-inf})$

lemma mono-assert-fun [simp]:
 $\text{mono } (\text{assert-fun } p)$
 $\langle \text{proof} \rangle$

lemma assert-fun-le-id [simp]: $\text{assert-fun } p \leq \text{id}$
 $\langle \text{proof} \rangle$

lemma $\text{assert-fun-disjunctive}$ [simp]: $\text{assert-fun } (p::'a::\text{distrib-lattice}) \in \text{Apply.disjunctive}$
 $\langle \text{proof} \rangle$

definition
 $\text{assertion-fun} = \text{range } \text{assert-fun}$

lemma assert-cont :
 $(x :: 'a::\text{boolean-algebra} \Rightarrow 'a) \leq \text{id} \Longrightarrow x \in \text{Apply.disjunctive} \Longrightarrow x = \text{assert-fun}$

($x \top$)
 ⟨proof⟩

lemma *assert-fun-disj-less-one*: $\text{assert-fun} = \text{Apply.disjunctive} \cap \{x::'a::\text{boolean-algebra} \Rightarrow 'a . x \leq \text{id}\}$
 ⟨proof⟩

lemma *assert-fun-dual*: $((\text{assert-fun } p) \circ \top) \sqcap (\text{dual-fun } (\text{assert-fun } p)) = \text{assert-fun } p$
 ⟨proof⟩

lemma *assert-fun-dual*: $x \in \text{assert-fun} \implies (x \circ \top) \sqcap (\text{dual-fun } x) = x$
 ⟨proof⟩

lemma *assert-fun-MonoTran [simp]*: $x \in \text{assert-fun} \implies \text{mono } x$
 ⟨proof⟩

lemma *assert-fun-le-one [simp]*: $x \in \text{assert-fun} \implies x \leq \text{id}$
 ⟨proof⟩

end

3 Algebra of Monotonic Boolean Transformers

theory *Mono-Bool-Tran-Algebra*
imports *Mono-Bool-Tran*
begin

In this section we introduce the *algebra of monotonic boolean transformers*. This is a bounded distributive lattice with a monoid operation, a dual operator and an iteration operator. The standard model for this algebra is the set of monotonic boolean transformers introduced in the previous section.

class *dual* =
fixes *dual::'a* $\Rightarrow 'a (- \hat{\ } \circ [81] 80)$

class *omega* =
fixes *omega::'a* $\Rightarrow 'a (- \hat{\ } \omega [81] 80)$

class *star* =
fixes *star::'a* $\Rightarrow 'a ((- \hat{\ } *) [81] 80)$

class *dual-star* =
fixes *dual-star::'a* $\Rightarrow 'a ((- \hat{\ } \otimes) [81] 80)$

class *mbt-algebra* = *monoid-mult* + *dual* + *omega* + *distrib-lattice* + *order-top* + *order-bot* + *star* + *dual-star* +
assumes
dual-le: $(x \leq y) = (y \hat{\ } \circ \leq x \hat{\ } \circ)$

and *dual-dual* [*simp*]: $(x \hat{o}) \hat{o} = x$
and *dual-comp*: $(x * y) \hat{o} = x \hat{o} * y \hat{o}$
and *dual-one* [*simp*]: $1 \hat{o} = 1$
and *top-comp* [*simp*]: $\top * x = \top$
and *inf-comp*: $(x \sqcap y) * z = (x * z) \sqcap (y * z)$
and *le-comp*: $x \leq y \implies z * x \leq z * y$
and *dual-neg*: $(x * \top) \sqcap (x \hat{o} * \perp) = \perp$
and *omega-fix*: $x \hat{\omega} = (x * (x \hat{\omega})) \sqcap 1$
and *omega-least*: $(x * z) \sqcap y \leq z \implies (x \hat{\omega}) * y \leq z$
and *star-fix*: $x \hat{*} = (x * (x \hat{*})) \sqcap 1$
and *star-greatest*: $z \leq (x * z) \sqcap y \implies z \leq (x \hat{*}) * y$
and *dual-star-def*: $(x \hat{\otimes}) = (((x \hat{o}) \hat{*}) \hat{o})$
begin

lemma *le-comp-right*: $x \leq y \implies x * z \leq y * z$
<proof>

subclass *bounded-lattice*
<proof>

end

instantiation *MonoTran* :: (*complete-boolean-algebra*) *mbt-algebra*
begin

lift-definition *dual-MonoTran* :: '*a MonoTran* \Rightarrow '*a MonoTran*
is *dual-fun*
<proof>

lift-definition *omega-MonoTran* :: '*a MonoTran* \Rightarrow '*a MonoTran*
is *omega-fun*
<proof>

lift-definition *star-MonoTran* :: '*a MonoTran* \Rightarrow '*a MonoTran*
is *star-fun*
<proof>

definition *dual-star-MonoTran* :: '*a MonoTran* \Rightarrow '*a MonoTran*
where
 $(x :: ('a MonoTran)) \hat{\otimes} = ((x \hat{o}) \hat{*}) \hat{o}$

instance *<proof>*

end

context *mbt-algebra* **begin**

lemma *dual-top* [*simp*]: $\top \hat{o} = \perp$
<proof>

lemma *dual-bot* [*simp*]: $\perp \hat{\ } o = \top$
 ⟨*proof*⟩

lemma *dual-inf*: $(x \sqcap y) \hat{\ } o = (x \hat{\ } o) \sqcup (y \hat{\ } o)$
 ⟨*proof*⟩

lemma *dual-sup*: $(x \sqcup y) \hat{\ } o = (x \hat{\ } o) \sqcap (y \hat{\ } o)$
 ⟨*proof*⟩

lemma *sup-comp*: $(x \sqcup y) * z = (x * z) \sqcup (y * z)$
 ⟨*proof*⟩

lemma *dual-eq*: $x \hat{\ } o = y \hat{\ } o \implies x = y$
 ⟨*proof*⟩

lemma *dual-neg-top* [*simp*]: $(x \hat{\ } o * \perp) \sqcup (x * \top) = \top$
 ⟨*proof*⟩

lemma *bot-comp* [*simp*]: $\perp * x = \perp$
 ⟨*proof*⟩

lemma [*simp*]: $(x * \top) * y = x * \top$
 ⟨*proof*⟩

lemma [*simp*]: $(x * \perp) * y = x * \perp$
 ⟨*proof*⟩

lemma *gt-one-comp*: $1 \leq x \implies y \leq x * y$
 ⟨*proof*⟩

theorem *omega-comp-fix*: $x \hat{\ } \omega * y = (x * (x \hat{\ } \omega) * y) \sqcap y$
 ⟨*proof*⟩

theorem *dual-star-fix*: $x \hat{\ } \otimes = (x * (x \hat{\ } \otimes)) \sqcup 1$
 ⟨*proof*⟩

theorem *star-comp-fix*: $x \hat{\ } * * y = (x * (x \hat{\ } *) * y) \sqcap y$
 ⟨*proof*⟩

theorem *dual-star-comp-fix*: $x \hat{\ } \otimes * y = (x * (x \hat{\ } \otimes) * y) \sqcup y$
 ⟨*proof*⟩

theorem *dual-star-least*: $(x * z) \sqcup y \leq z \implies (x \hat{\ } \otimes) * y \leq z$
 ⟨*proof*⟩

lemma *omega-one* [*simp*]: $1 \hat{\ } \omega = \perp$
 ⟨*proof*⟩

lemma *omega-mono*: $x \leq y \implies x \hat{\ } \omega \leq y \hat{\ } \omega$
 ⟨*proof*⟩

end

sublocale *mbt-algebra* < *conjunctive inf inf times*
 ⟨*proof*⟩

sublocale *mbt-algebra* < *disjunctive sup sup times*
 ⟨*proof*⟩

context *mbt-algebra* **begin**

lemma *dual-conjunctive*: $x \in \text{conjunctive} \implies x \hat{\ } o \in \text{disjunctive}$
 ⟨*proof*⟩

lemma *dual-disjunctive*: $x \in \text{disjunctive} \implies x \hat{\ } o \in \text{conjunctive}$
 ⟨*proof*⟩

lemma *comp-pres-conj*: $x \in \text{conjunctive} \implies y \in \text{conjunctive} \implies x * y \in \text{conjunctive}$
 ⟨*proof*⟩

lemma *comp-pres-disj*: $x \in \text{disjunctive} \implies y \in \text{disjunctive} \implies x * y \in \text{disjunctive}$
 ⟨*proof*⟩

lemma *start-pres-conj*: $x \in \text{conjunctive} \implies (x \hat{\ } *) \in \text{conjunctive}$
 ⟨*proof*⟩

lemma *dual-star-pres-disj*: $x \in \text{disjunctive} \implies x \hat{\ } \otimes \in \text{disjunctive}$
 ⟨*proof*⟩

3.1 Assertions

Usually, in Kleene algebra with tests or in other program algebras, tests or assertions or assumptions are defined using an existential quantifier. An element of the algebra is a test if it has a complement with respect to \perp and 1 . In this formalization assertions can be defined much simpler using the dual operator.

definition

$$\text{assertion} = \{x . x \leq 1 \wedge (x * \top) \sqcap (x \hat{\ } o) = x\}$$

lemma *assertion-prop*: $x \in \text{assertion} \implies (x * \top) \sqcap 1 = x$
 ⟨*proof*⟩

lemma *dual-assertion-prop*: $x \in \text{assertion} \implies ((x \hat{\ } o) * \perp) \sqcup 1 = x \hat{\ } o$
 ⟨*proof*⟩

lemma *assertion-disjunctive*: $x \in \text{assertion} \implies x \in \text{disjunctive}$
<proof>

lemma *Abs-MonoTran-injective*: $\text{mono } x \implies \text{mono } y \implies \text{Abs-MonoTran } x = \text{Abs-MonoTran } y \implies x = y$
<proof>
end

lemma *mbta-MonoTran-disjunctive*: $\text{Rep-MonoTran } \text{'disjunctive} = \text{Apply.disjunctive}$
<proof>

lemma *assertion-MonoTran*: $\text{assertion} = \text{Abs-MonoTran } \text{'assertion-fun}$
<proof>

context *mbt-algebra* **begin**

lemma *assertion-conjunctive*: $x \in \text{assertion} \implies x \in \text{conjunctive}$
<proof>

lemma *dual-assertion-conjunctive*: $x \in \text{assertion} \implies x \hat{\ } o \in \text{conjunctive}$
<proof>

lemma *dual-assertion-disjunct*: $x \in \text{assertion} \implies x \hat{\ } o \in \text{disjunctive}$
<proof>

lemma [*simp*]: $x \in \text{assertion} \implies y \in \text{assertion} \implies x \sqcap y \leq x * y$
<proof>

lemma [*simp*]: $x \in \text{assertion} \implies x * y \leq y$
<proof>

lemma [*simp*]: $x \in \text{assertion} \implies y \in \text{assertion} \implies x * y \leq x$
<proof>

lemma *assertion-inf-comp-eq*: $x \in \text{assertion} \implies y \in \text{assertion} \implies x \sqcap y = x * y$
<proof>

lemma *one-right-assertion* [*simp*]: $x \in \text{assertion} \implies x * 1 = x$
<proof>

lemma [*simp*]: $x \in \text{assertion} \implies x \sqcup 1 = 1$
<proof>

lemma [*simp*]: $x \in \text{assertion} \implies 1 \sqcup x = 1$
<proof>

lemma [*simp*]: $x \in \text{assertion} \implies x \sqcap 1 = x$
<proof>

lemma [simp]: $x \in \text{assertion} \implies 1 \sqcap x = x$
(proof)

lemma [simp]: $x \in \text{assertion} \implies x \leq x * \top$
(proof)

lemma [simp]: $x \in \text{assertion} \implies x \leq 1$
(proof)

definition
 $\text{neg-assert } (x::'a) = (x \hat{=} o * \perp) \sqcap 1$

lemma sup-uminus[simp]: $x \in \text{assertion} \implies x \sqcup \text{neg-assert } x = 1$
(proof)

lemma inf-uminus[simp]: $x \in \text{assertion} \implies x \sqcap \text{neg-assert } x = \perp$
(proof)

lemma uminus-assertion[simp]: $x \in \text{assertion} \implies \text{neg-assert } x \in \text{assertion}$
(proof)

lemma uminus-uminus [simp]: $x \in \text{assertion} \implies \text{neg-assert } (\text{neg-assert } x) = x$
(proof)

lemma dual-comp-neg [simp]: $x \hat{=} o * y \sqcup (\text{neg-assert } x) * \top = x \hat{=} o * y$
(proof)

lemma [simp]: $(\text{neg-assert } x) \hat{=} o * y \sqcup x * \top = (\text{neg-assert } x) \hat{=} o * y$
(proof)

lemma [simp]: $x * \top \sqcup (\text{neg-assert } x) \hat{=} o * y = (\text{neg-assert } x) \hat{=} o * y$
(proof)

lemma inf-assertion [simp]: $x \in \text{assertion} \implies y \in \text{assertion} \implies x \sqcap y \in \text{assertion}$
(proof)

lemma comp-assertion [simp]: $x \in \text{assertion} \implies y \in \text{assertion} \implies x * y \in \text{assertion}$
(proof)

lemma sup-assertion [simp]: $x \in \text{assertion} \implies y \in \text{assertion} \implies x \sqcup y \in \text{assertion}$
(proof)

lemma [simp]: $x \in \text{assertion} \implies x * x = x$

<proof>

lemma [*simp*]: $x \in \text{assertion} \implies (x \hat{\ } o) * (x \hat{\ } o) = x \hat{\ } o$
<proof>

lemma [*simp*]: $x \in \text{assertion} \implies x * (x \hat{\ } o) = x$
<proof>

lemma [*simp*]: $x \in \text{assertion} \implies (x \hat{\ } o) * x = x \hat{\ } o$
<proof>

lemma [*simp*]: $\perp \in \text{assertion}$
<proof>

lemma [*simp*]: $1 \in \text{assertion}$
<proof>

3.2 Weakest precondition of true

definition

$$\text{wpt } x = (x * \top) \sqcap 1$$

lemma *wpt-is-assertion* [*simp*]: $\text{wpt } x \in \text{assertion}$
<proof>

lemma *wpt-comp*: $(\text{wpt } x) * x = x$
<proof>

lemma *wpt-comp-2*: $\text{wpt } (x * y) = \text{wpt } (x * (\text{wpt } y))$
<proof>

lemma *wpt-assertion* [*simp*]: $x \in \text{assertion} \implies \text{wpt } x = x$
<proof>

lemma *wpt-le-assertion*: $x \in \text{assertion} \implies x * y = y \implies \text{wpt } y \leq x$
<proof>

lemma *wpt-choice*: $\text{wpt } (x \sqcap y) = \text{wpt } x \sqcap \text{wpt } y$
<proof>

end

context *lattice* **begin**

lemma [*simp*]: $x \leq y \implies x \sqcap y = x$
<proof>

end

context *mbt-algebra* **begin**

lemma *wpt-dual-assertion-comp*: $x \in \text{assertion} \implies y \in \text{assertion} \implies \text{wpt} ((x \hat{\ } o) * y) = (\text{neg-assert } x) \sqcup y$
 ⟨proof⟩

lemma *le-comp-left-right*: $x \leq y \implies u \leq v \implies x * u \leq y * v$
 ⟨proof⟩

lemma *wpt-dual-assertion*: $x \in \text{assertion} \implies \text{wpt} (x \hat{\ } o) = 1$
 ⟨proof⟩

lemma *assertion-commute*: $x \in \text{assertion} \implies y \in \text{conjunctive} \implies y * x = \text{wpt}(y * x) * y$
 ⟨proof⟩

lemma *wpt-mono*: $x \leq y \implies \text{wpt } x \leq \text{wpt } y$
 ⟨proof⟩

lemma $a \in \text{conjunctive} \implies x * a \leq a * y \implies (x \hat{\ } \omega) * a \leq a * (y \hat{\ } \omega)$
 ⟨proof⟩

lemma [*simp*]: $x \leq 1 \implies y * x \leq y$
 ⟨proof⟩

lemma [*simp*]: $x \leq x * \top$
 ⟨proof⟩

lemma [*simp*]: $x * \perp \leq x$
 ⟨proof⟩

end

3.3 Monotonic Boolean transformers algebra with post condition statement

definition

post-fun ($p::'a::\text{order}$) $q = (\text{if } p \leq q \text{ then } (\top::'b::\{\text{order-bot,order-top}\}) \text{ else } \perp)$

lemma *mono-post-fun* [*simp*]: $\text{mono} (\text{post-fun } (p::-\::\{\text{order-bot,order-top}\}))$
 ⟨proof⟩

lemma *post-top* [*simp*]: $\text{post-fun } p \ p = \top$
 ⟨proof⟩

lemma *post-refin* [*simp*]: $\text{mono } S \implies ((S \ p)::'a::\text{bounded-lattice}) \sqcap (\text{post-fun } p) \ x \leq S \ x$
 ⟨proof⟩


```

class post-mbt-algebra = mbt-algebra +
  fixes post :: 'a ⇒ 'a
  assumes post-1: (post x) * x * ⊤ = ⊤
  and post-2: y * x * ⊤ ⊓ (post x) ≤ y

instantiation MonoTran :: (complete-boolean-algebra) post-mbt-algebra
begin

lift-definition post-MonoTran :: 'a::complete-boolean-algebra MonoTran ⇒ 'a::complete-boolean-algebra
MonoTran
  is λx. post-fun (x ⊤)
  ⟨proof⟩

instance ⟨proof⟩

end

```

3.4 Complete monotonic Boolean transformers algebra

```

class complete-mbt-algebra = post-mbt-algebra + complete-distrib-lattice +
  assumes Inf-comp: (Inf X) * z = (INF x ∈ X . (x * z))

```

```

instance MonoTran :: (complete-boolean-algebra) complete-mbt-algebra
  ⟨proof⟩

```

context *complete-mbt-algebra* **begin**

```

lemma dual-Inf: (Inf X) ^ o = (SUP x ∈ X . x ^ o)
  ⟨proof⟩

```

```

lemma dual-Sup: (Sup X) ^ o = (INF x ∈ X . x ^ o)
  ⟨proof⟩

```

```

lemma INF-comp: (⊓ (f ' A)) * z = (INF a ∈ A . (f a) * z)
  ⟨proof⟩

```

```

lemma dual-INF: (⊓ (f ' A)) ^ o = (SUP a ∈ A . (f a) ^ o)
  ⟨proof⟩

```

```

lemma dual-SUP: (⊔ (f ' A)) ^ o = (INF a ∈ A . (f a) ^ o)
  ⟨proof⟩

```

```

lemma Sup-comp: (Sup X) * z = (SUP x ∈ X . (x * z))
  ⟨proof⟩

```

```

lemma SUP-comp: (⊔ (f ' A)) * z = (SUP a ∈ A . (f a) * z)
  ⟨proof⟩

```

```

lemma Sup-assertion [simp]: X ⊆ assertion ⇒ Sup X ∈ assertion

```

<proof>

lemma *Sup-range-assertion* [*simp*]: ($!!w . p w \in \text{assertion}$) $\implies \text{Sup} (\text{range } p) \in \text{assertion}$
<proof>

lemma *Sup-less-assertion* [*simp*]: ($!!w . p w \in \text{assertion}$) $\implies \text{Sup-less } p w \in \text{assertion}$
<proof>

theorem *omega-lfp*:
 $x \hat{\omega} * y = \text{lfp} (\lambda z . (x * z) \sqcap y)$
<proof>
end

lemma [*simp*]: *mono* ($\lambda (t::'a::\text{mbt-algebra}) . x * t \sqcap y$)
<proof>

class *mbt-algebra-fusion* = *mbt-algebra* +
assumes *fusion*: ($\forall t . x * t \sqcap y \sqcap z \leq u * (t \sqcap z) \sqcap v$)
 $\implies (x \hat{\omega}) * y \sqcap z \leq (u \hat{\omega}) * v$

lemma
class.mbt-algebra-fusion ($1::'a::\text{complete-mbt-algebra}$) ($(*)$) (\sqcap) (\leq) ($<$) (\sqcup) *dual*
dual-star omega star $\perp \top$
<proof>

context *mbt-algebra-fusion*
begin

lemma *omega-star*: $x \in \text{conjunctive} \implies x \hat{\omega} = \text{wpt} (x \hat{\omega}) * (x \hat{*})$
<proof>

lemma *omega-pres-conj*: $x \in \text{conjunctive} \implies x \hat{\omega} \in \text{conjunctive}$
<proof>
end

end

4 Boolean Algebra of Assertions

theory *Assertion-Algebra*
imports *Mono-Bool-Tran-Algebra*
begin

This section introduces the boolean algebra of assertions. The type `Assertion` and the boolean operation are introduced based on the set `assertion` and the operations on the monotonic boolean transformers algebra. The type

Assertion over a complete monotonic boolean transformers algebra is a complete boolean algebra.

typedef (overloaded) ('a::mbt-algebra) *Assertion* = *assertion::'a set*
 ⟨*proof*⟩

definition

assert :: 'a::mbt-algebra *Assertion* \Rightarrow 'a ($\{\cdot -\}$ [0] 1000) **where**
 $\{\cdot p\} = \text{Rep-Assertion } p$

definition

abs-wpt $x = \text{Abs-Assertion } (\text{wpt } x)$

lemma [*simp*]: $\{\cdot p\} \in \text{assertion}$
 ⟨*proof*⟩

lemma [*simp*]: $\text{abs-wpt } (\{\cdot p\}) = p$
 ⟨*proof*⟩

lemma [*simp*]: $x \in \text{assertion} \Longrightarrow \{\cdot \text{Abs-Assertion } x\} = x$
 ⟨*proof*⟩

lemma [*simp*]: $x \in \text{assertion} \Longrightarrow \{\cdot \text{abs-wpt } x\} = x$
 ⟨*proof*⟩

lemma *assert-injective*: $\{\cdot p\} = \{\cdot q\} \Longrightarrow p = q$
 ⟨*proof*⟩

instantiation *Assertion* :: (mbt-algebra) boolean-algebra
begin

definition

uminus-Assertion-def: $\neg p = \text{abs-wpt}(\text{neg-assert } \{\cdot p\})$

definition

bot-Assertion-def: $\perp = \text{abs-wpt } \perp$

definition

top-Assertion-def: $\top = \text{abs-wpt } 1$

definition

inf-Assertion-def: $p \sqcap q = \text{abs-wpt } (\{\cdot p\} \sqcap \{\cdot q\})$

definition

sup-Assertion-def: $p \sqcup q = \text{abs-wpt } (\{\cdot p\} \sqcup \{\cdot q\})$

definition

less-eq-Assertion-def: $(p \leq q) = (\{\cdot p\} \leq \{\cdot q\})$

definition

less-Assertion-def: $(p < q) = (\{ \cdot p \} < \{ \cdot q \})$

definition

minus-Assertion-def: $(p :: 'a \text{ Assertion}) - q = p \sqcap - q$

instance

$\langle \text{proof} \rangle$

end

lemma *assert-image [simp]*: $\text{assert } 'A \subseteq \text{assertion}$

$\langle \text{proof} \rangle$

instantiation *Assertion* :: (complete-mbt-algebra) complete-lattice

begin**definition**

Sup-Assertion-def: $\text{Sup } A = \text{abs-wpt } (\text{Sup } (\text{assert } 'A))$

definition

Inf-Assertion-def: $\text{Inf } (A :: 'a \text{ Assertion}) \text{ set} = - (\text{Sup } (\text{uminus } 'A))$

lemma *Sup1*: $(x :: 'a \text{ Assertion}) \in A \implies x \leq \text{Sup } A$

$\langle \text{proof} \rangle$

lemma *Sup2*: $(\bigwedge x :: 'a \text{ Assertion} . x \in A \implies x \leq z) \implies \text{Sup } A \leq z$

$\langle \text{proof} \rangle$

instance

$\langle \text{proof} \rangle$

end

lemma *assert-top [simp]*: $\{ \cdot \top \} = 1$

$\langle \text{proof} \rangle$

lemma *assert-Sup*: $\{ \cdot \text{Sup } A \} = \text{Sup } (\text{assert } 'A)$

$\langle \text{proof} \rangle$

lemma *assert-Inf*: $\{ \cdot \text{Inf } A \} = (\text{Inf } (\text{assert } 'A)) \sqcap 1$

$\langle \text{proof} \rangle$

lemma *assert-Inf-ne*: $A \neq \{ \} \implies \{ \cdot \text{Inf } A \} = \text{Inf } (\text{assert } 'A)$

$\langle \text{proof} \rangle$

lemma *assert-Sup-range*: $\{ \cdot \text{Sup } (\text{range } p) \} = \text{Sup } (\text{range } (\text{assert } o p))$

$\langle \text{proof} \rangle$

lemma *assert-Sup-less*: $\{\cdot \text{ Sup-less } p \ w\} = \text{Sup-less } (\text{assert } o \ p) \ w$
 ⟨*proof*⟩

end

5 Program statements, Hoare and refinement rules

theory *Statements*

imports *Assertion-Algebra*

begin

In this section we introduce assume, if, and while program statements as well as Hoare triples, and data refinement. We prove Hoare correctness rules for the program statements and we prove some theorems linking Hoare correctness statement to (data) refinement. Most of the theorems assume a monotonic boolean transformers algebra. The theorem stating the equivalence between a Hoare correctness triple and a refinement statement holds under the assumption that we have a monotonic boolean transformers algebra with post condition statement.

definition

assume :: 'a::mbt-algebra Assertion \Rightarrow 'a ([\cdot -] [0] 1000) **where**
 $[\cdot p] = \{\cdot p\} \hat{\ } o$

lemma [*simp*]: $\{\cdot p\} * \top \sqcap [\cdot p] = \{\cdot p\}$
 ⟨*proof*⟩

lemma [*simp*]: $[\cdot p] * x \sqcup \{\cdot -p\} * \top = [\cdot p] * x$
 ⟨*proof*⟩

lemma [*simp*]: $\{\cdot p\} * \top \sqcup [\cdot -p] * x = [\cdot -p] * x$
 ⟨*proof*⟩

lemma *assert-sup*: $\{\cdot p \sqcup q\} = \{\cdot p\} \sqcup \{\cdot q\}$
 ⟨*proof*⟩

lemma *assert-inf*: $\{\cdot p \sqcap q\} = \{\cdot p\} \sqcap \{\cdot q\}$
 ⟨*proof*⟩

lemma *assert-neg*: $\{\cdot -p\} = \text{neg-assert } \{\cdot p\}$
 ⟨*proof*⟩

lemma *assert-false* [*simp*]: $\{\cdot \perp\} = \perp$
 ⟨*proof*⟩

lemma *if-Assertion-assumption*: $(\{\cdot p\} * x) \sqcup (\{\cdot -p\} * y) = ([\cdot p] * x) \sqcap ([\cdot -p] * y)$
 ⟨*proof*⟩

definition

$$wp\ x\ p = abs\text{-}wpt\ (x * \{\cdot p\})$$

lemma *wp-assume*: $wp\ [\cdot p]\ q = \neg p \sqcup q$
<proof>

lemma *assert-commute*: $y \in conjunctive \implies y * \{\cdot p\} = \{\cdot\ wp\ y\ p\} * y$
<proof>

lemma *wp-assert*: $wp\ \{\cdot p\}\ q = p \sqcap q$
<proof>

lemma *wp-mono [simp]*: $mono\ (wp\ x)$
<proof>

lemma *wp-mono2*: $p \leq q \implies wp\ x\ p \leq wp\ x\ q$
<proof>

lemma *wp-fun-mono [simp]*: $mono\ wp$
<proof>

lemma *wp-fun-mono2*: $x \leq y \implies wp\ x\ p \leq wp\ y\ p$
<proof>

lemma *wp-comp*: $wp\ (x * y)\ p = wp\ x\ (wp\ y\ p)$
<proof>

lemma *wp-choice*: $wp\ (x \sqcap y) = wp\ x \sqcap wp\ y$
<proof>

lemma *[simp]*: $wp\ 1 = id$
<proof>

lemma *wp-omega-fix*: $wp\ (x \hat{\omega})\ p = wp\ x\ (wp\ (x \hat{\omega})\ p) \sqcap p$
<proof>

lemma *wp-omega-least*: $(wp\ x\ r) \sqcap p \leq r \implies wp\ (x \hat{\omega})\ p \leq r$
<proof>

lemma *Assertion-wp*: $\{\cdot wp\ x\ p\} = (x * \{\cdot p\} * \top) \sqcap 1$
<proof>

definition

$$hoare\ p\ S\ q = (p \leq wp\ S\ q)$$

definition

$$\text{grad } x = - (\text{wp } x \perp)$$

lemma *grad-comp*: $[\cdot \text{grad } x] * x = x$
 $\langle \text{proof} \rangle$

lemma *assert-assume*: $\{\cdot p\} * [\cdot p] = \{\cdot p\}$
 $\langle \text{proof} \rangle$

lemma *dual-assume*: $[\cdot p] \wedge o = \{\cdot p\}$
 $\langle \text{proof} \rangle$

lemma *assume-prop*: $([\cdot p] * \perp) \sqcup 1 = [\cdot p]$
 $\langle \text{proof} \rangle$

An alternative definition of a Hoare triple

definition *hoare1* $p \ S \ q = ([\cdot p] * S * [\cdot -q] = \top)$

lemma *hoare1* $p \ S \ q = \text{hoare } p \ S \ q$
 $\langle \text{proof} \rangle$

lemma *hoare-choice*: $\text{hoare } p \ (x \sqcap y) \ q = ((\text{hoare } p) \ x \ q \ \& \ (\text{hoare } p \ y \ q))$
 $\langle \text{proof} \rangle$

definition

if-stm: $'a::\text{mbt-algebra } \text{Assertion} \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a \ ((\text{If } (-) / \text{ then } (-) / \text{ else } (-)) [0, 0, 10] 10)$ **where**

$$\text{if-stm } b \ x \ y = (([\cdot b] * x) \sqcap ([\cdot -b] * y))$$

lemma *if-assertion*: $(\text{If } p \ \text{then } x \ \text{else } y) = \{\cdot p\} * x \sqcup \{\cdot -p\} * y$
 $\langle \text{proof} \rangle$

lemma *hoare-if*: $\text{hoare } p \ (\text{If } b \ \text{then } x \ \text{else } y) \ q = (\text{hoare } (p \sqcap b) \ x \ q \ \wedge \ \text{hoare } (p \sqcap -b) \ y \ q)$
 $\langle \text{proof} \rangle$

lemma *hoare-comp*: $\text{hoare } p \ (x * y) \ q = (\exists r . (\text{hoare } p \ x \ r) \ \wedge \ (\text{hoare } r \ y \ q))$
 $\langle \text{proof} \rangle$

lemma *hoare-refinement*: $\text{hoare } p \ S \ q = (\{\cdot p\} * (\text{post } \{\cdot q\}) \leq S)$
 $\langle \text{proof} \rangle$

theorem *hoare-fixpoint-mbt*:

$$\begin{aligned} F \ x &= x \\ &\implies (!!(w::'a::\text{well-founded}) \ f . (\bigwedge v. v < w \implies \text{hoare } (p \ v) \ f \ q) \implies \text{hoare } (p \ w) \ (F \ f) \ q) \\ &\implies \text{hoare } (p \ u) \ x \ q \end{aligned}$$

$$\langle \text{proof} \rangle$$

lemma *hoare-Sup*: $\text{hoare } (\text{Sup } P) \ x \ q = (\forall p \in P . \text{hoare } p \ x \ q)$

$\langle \text{proof} \rangle$

theorem *hoare-fixpoint-complete-mbt*:

$F x = x$
 $\implies (\forall w f . \text{hoare } (\text{Sup-less } p w) f q \implies \text{hoare } (p w) (F f) q)$
 $\implies \text{hoare } (\text{Sup } (\text{range } p)) x q$
 $\langle \text{proof} \rangle$

definition

while:: 'a::mbt-algebra Assertion \Rightarrow 'a \Rightarrow 'a ((While (-)/ do (-)) [0, 10] 10)

where

while p x = ([· p] * x) $\hat{\omega}$ * [· -p]

lemma *while-false*: (While \perp do x) = 1

$\langle \text{proof} \rangle$

lemma *while-true*: (While \top do 1) = \perp

$\langle \text{proof} \rangle$

lemma *hoare-wp [simp]*: hoare (wp x q) x q

$\langle \text{proof} \rangle$

lemma *hoare-comp-wp*: hoare p (x * y) q = hoare p x (wp y q)

$\langle \text{proof} \rangle$

lemma (in mbt-algebra) *hoare-assume*: hoare p [·b] q = (p \sqcap b \leq q)

$\langle \text{proof} \rangle$

lemma (in mbt-algebra) *hoare-assume-comp*: hoare p ([·b] * x) q = hoare (p \sqcap b) x q

$\langle \text{proof} \rangle$

lemma *hoare-while-mbt*:

($\forall (w::'b::\text{well-founded}) r . (\forall v . v < w \implies p v \leq r) \implies \text{hoare } ((p w) \sqcap b) x r$) \implies

($\forall u . p u \leq q \implies \text{hoare } (p w) (\text{While } b \text{ do } x) (q \sqcap -b)$)

$\langle \text{proof} \rangle$

lemma *hoare-while-complete-mbt*:

($\forall w::'b::\text{well-founded} . \text{hoare } ((p w) \sqcap b) x (\text{Sup-less } p w) \implies \text{hoare } (\text{Sup } (\text{range } p)) (\text{While } b \text{ do } x) ((\text{Sup } (\text{range } p)) \sqcap -b)$)

$\langle \text{proof} \rangle$

definition

datarefin S S1 D D1 = (D * S \leq S1 * D1)

lemma *hoare p S q \implies datarefin S S1 D D1 \implies hoare (wp D p) S1 (wp D1 q)*

$\langle \text{proof} \rangle$

lemma *hoare* $p \ S \ q \implies \text{datarefin } (\{\cdot p\} * S) \ S1 \ D \ D1 \implies \text{hoare } (wp \ D \ p) \ S1 \ (wp \ D1 \ q)$

<proof>

lemma *inf-pres-conj*: $x \in \text{conjunctive} \implies y \in \text{conjunctive} \implies x \sqcap y \in \text{conjunctive}$

<proof>

lemma *sup-pres-disj*: $x \in \text{disjunctive} \implies y \in \text{disjunctive} \implies x \sqcup y \in \text{disjunctive}$

<proof>

lemma *assumption-conjunctive* [*simp*]: $[\cdot p] \in \text{conjunctive}$

<proof>

lemma *assumption-disjunctive* [*simp*]: $[\cdot p] \in \text{disjunctive}$

<proof>

lemma *if-pres-conj*: $x \in \text{conjunctive} \implies y \in \text{conjunctive} \implies (\text{If } p \ \text{then } x \ \text{else } y) \in \text{conjunctive}$

<proof>

lemma *if-pres-disj*: $x \in \text{disjunctive} \implies y \in \text{disjunctive} \implies (\text{If } p \ \text{then } x \ \text{else } y) \in \text{disjunctive}$

<proof>

lemma *while-dual-star*: $(\text{While } p \ \text{do } (x::'a::\text{mbt-algebra})) = ((\{\cdot p\} * x) \widehat{\otimes} * \{\cdot \neg p\})$

<proof>

lemma *while-pres-disj*: $(x::'a::\text{mbt-algebra}) \in \text{disjunctive} \implies (\text{While } p \ \text{do } x) \in \text{disjunctive}$

<proof>

lemma *while-pres-conj*: $(x::'a::\text{mbt-algebra-fusion}) \in \text{conjunctive} \implies (\text{While } p \ \text{do } x) \in \text{conjunctive}$

<proof>

unbundle *no-lattice-syntax*

end

References

- [1] R.-J. Back. *On the correctness of refinement in program development*. PhD thesis, Department of Computer Science, University of Helsinki, 1978.
- [2] R.-J. Back. *Correctness preserving program refinements: proof theory and applications*, volume 131 of *Mathematical Centre Tracts*. Mathe-

matisch Centrum, Amsterdam, 1980.

- [3] R.-J. Back and J. von Wright. A lattice-theoretical basis for a specification language. In *Proceedings of the International Conference on Mathematics of Program Construction, 375th Anniversary of the Groningen University*, pages 139–156, London, UK, 1989. Springer-Verlag.
- [4] R.-J. Back and J. von Wright. Duality in specification languages: a lattice-theoretical approach. *Acta Inf.*, 27:583–625, July 1990.
- [5] R.-J. Back and J. von Wright. *Refinement Calculus. A systematic Introduction*. Springer, 1998.
- [6] H.-H. Dang, P. Höfner, and B. Möller. Algebraic separation logic. *Journal of Logic and Algebraic Programming*, 80(6):221 – 247, 2011. Relations and Kleene Algebras in Computer Science.
- [7] J. Desharnais, B. Möller, and G. Struth. Kleene algebra with domain. *ACM Trans. Comput. Logic*, 7:798–833, October 2006.
- [8] P. Guerreiro. Another characterization of weakest preconditions. In M. Dezani-Ciancaglini and U. Montanari, editors, *International Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 164–177. Springer Berlin / Heidelberg, 1982. 10.1007/3-540-11494-7_12.
- [9] C. A. Hoare, B. Möller, G. Struth, and I. Wehrman. Concurrent kleene algebra. In *Proceedings of the 20th International Conference on Concurrency Theory, CONCUR 2009*, pages 399–414, Berlin, Heidelberg, 2009. Springer-Verlag.
- [10] D. Kozen. Kleene algebra with tests. *ACM Trans. Program. Lang. Syst.*, 19:427–443, May 1997.
- [11] L. Meinicke and K. Solin. Refinement algebra for probabilistic programs. *Formal Aspects of Computing*, 22:3–31, 2010. 10.1007/s00165-009-0111-1.
- [12] C. Morgan. *Programming from specifications*. Prentice-Hall, Inc., 1990.
- [13] V. Preoteasa. Algebra of monotonic boolean transformers. In *Proceedings of SBMF 2011*, Lecture Notes in Computer Science, pages 140–155, Berlin Heidelberg, 2011. Springer-Verlag.
- [14] K. Solin and J. von Wright. Enabledness and termination in refinement algebra. *Sci. Comput. Program.*, 74:654–668, June 2009.

- [15] J. von Wright. From kleene algebra to refinement algebra. In *Proceedings of the 6th International Conference on Mathematics of Program Construction*, MPC '02, pages 233–262, London, UK, UK, 2002. Springer-Verlag.
- [16] J. von Wright. Towards a refinement algebra. *Sci. Comput. Program.*, 51:23–45, May 2004.