Algebra of Monotonic Boolean Transformers

Viorel Preoteasa

May 26, 2024

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

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 $((\mathsf{dual}\ f)p = -f(-p))$ 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*:
$\bigsqcap M \circ f = (\bigsqcap m{\in}M.\ m \circ f)$
⟨*proof*⟩

lemma *INF-comp-fun*:
$(\bigsqcap a{\in}A.\ g\ a) \circ f = (\bigsqcap a{\in}A.\ g\ a \circ f)$
⟨*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*]:
mono $(\lambda\text{-}.\ c)$
⟨*proof*⟩

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

lemma (**in** *order*) *mono-comp* [*simp*]:
mono $f \Longrightarrow$ *mono* $g \Longrightarrow$ *mono* $(f \circ g)$
⟨*proof*⟩

lemma (**in** *bot*) *mono-bot* [*simp*]:
mono $\bot$
⟨*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 $A \subseteq \{f :: 'a \Rightarrow 'b :: \textit{complete-lattice}.\ \textit{mono}\ f\}$
shows *mono* $(\bigsqcap A)$
⟨*proof*⟩

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

typedef (**overloaded**) $'a$ *MonoTran* = $\{f :: 'a :: \textit{order} \Rightarrow 'a\ .\ \textit{mono}\ f\}$
⟨*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 ⊥
  ⟨*proof*⟩

instance
  ⟨*proof*⟩

end

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

lift-definition *top-MonoTran* :: *'a MonoTran*
  is ⊤
  ⟨*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* ⇒ *'a MonoTran*
is *Inf*
⟨*proof*⟩

lift-definition *Sup-MonoTran* :: *'a MonoTran set* ⇒ *'a MonoTran*
is *Sup*
⟨*proof*⟩

instance
⟨*proof*⟩

end

context includes *lifting-syntax*
begin

lemma [*transfer-rule*]:
(*rel-set A* ===> (*A* ===> *pcr-MonoTran HOL.eq*) ===> *pcr-MonoTran HOL.eq*) (λ*A f*. ⨅(*f* ‘ *A*)) (λ*A f*. ⨅(*f* ‘ *A*))
⟨*proof*⟩

lemma [*transfer-rule*]:
(*rel-set A* ===> (*A* ===> *pcr-MonoTran HOL.eq*) ===> *pcr-MonoTran HOL.eq*) (λ*A f*. ⨆(*f* ‘ *A*)) (λ*A f*. ⨆(*f* ‘ *A*))
⟨*proof*⟩

end

instance *MonoTran* :: (*complete-distrib-lattice*) *complete-distrib-lattice*
⟨*proof*⟩

definition
dual-fun (*f*::*'a*::*boolean-algebra* ⇒ *'a*) = *uminus* ∘ *f* ∘ *uminus*

lemma *dual-fun-apply* [*simp*]:
dual-fun f p = − *f* (− *p*)
⟨*proof*⟩

lemma *mono-dual-fun* [*simp*]:
mono f $\Longrightarrow$ *mono* (*dual-fun f*)
⟨*proof*⟩

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

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

lemma *mono-comp-fun*:
fixes *f* :: *'a*::*order* ⇒ *'b*::*order*
shows *mono f* $\Longrightarrow$ *mono* ((∘) *f*)
⟨*proof*⟩

definition
Omega-fun f g = *inf g* ∘ *comp f*

lemma *Omega-fun-apply* [*simp*]:
Omega-fun f g h p = (*g p* ⊓ *f* (*h p*))
⟨*proof*⟩

lemma *mono-Omega-fun* [*simp*]:
mono f $\Longrightarrow$ *mono* (*Omega-fun f g*)
⟨*proof*⟩

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

definition
omega-fun f = *lfp* (*Omega-fun f id*)

definition
star-fun f = *gfp* (*Omega-fun f id*)

lemma *mono-omega-fun* [*simp*]:
fixes *f* :: *'a*::*complete-lattice* ⇒ *'a*
assumes *mono f*
shows *mono* (*omega-fun f*)
⟨*proof*⟩

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

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

lemma *lfp-omega-lowerbound*:
mono f $\Longrightarrow$ *Omega-fun f g A* $\leq$ *A* $\Longrightarrow$ *omega-fun f* $\circ$ *g* $\leq$ *A*
⟨*proof*⟩

lemma *gfp-omega-upperbound*:
mono f $\Longrightarrow$ *A* $\leq$ *Omega-fun f g A* $\Longrightarrow$ *A* $\leq$ *star-fun f* $\circ$ *g*
⟨*proof*⟩

lemma *lfp-omega-greatest*:
assumes $\bigwedge u$. *Omega-fun f g u* $\leq$ *u* $\Longrightarrow$ *A* $\leq$ *u*
shows *A* $\leq$ *omega-fun f* $\circ$ *g*
⟨*proof*⟩

lemma *gfp-star-least*:
assumes $\bigwedge u$. *u* $\leq$ *Omega-fun f g u* $\Longrightarrow$ *u* $\leq$ *A*
shows *star-fun f* $\circ$ *g* $\leq$ *A*
⟨*proof*⟩

lemma *lfp-omega*:
mono f $\Longrightarrow$ *omega-fun f* $\circ$ *g* = *lfp* (*Omega-fun f g*)
⟨*proof*⟩

lemma *gfp-star*:
mono f $\Longrightarrow$ *star-fun f* $\circ$ *g* = *gfp* (*Omega-fun f g*)
⟨*proof*⟩

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

lemma *mono-assert-fun* [*simp*]:
mono (*assert-fun p*)
⟨*proof*⟩

lemma *assert-fun-le-id* [*simp*]: *assert-fun p* $\leq$ *id*
⟨*proof*⟩

lemma *assert-fun-disjunctive* [*simp*]: *assert-fun* (*p*::*'a*::*distrib-lattice*) $\in$ *Apply.disjunctive*
⟨*proof*⟩

definition
assertion-fun = *range assert-fun*

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

$(x \top)$
⟨*proof*⟩

lemma *assertion-fun-disj-less-one*: *assertion-fun* = *Apply.disjunctive* ∩ {*x*::'*a*::*boolean-algebra* ⇒ '*a* . *x* ≤ *id*}
⟨*proof*⟩

lemma *assert-fun-dual*: ((*assert-fun p*) *o* ⊤) ⊓ (*dual-fun* (*assert-fun p*)) = *assert-fun p*
⟨*proof*⟩

lemma *assertion-fun-dual*: *x* ∈ *assertion-fun* ⟹ (*x o* ⊤) ⊓ (*dual-fun x*) = *x*
⟨*proof*⟩

lemma *assertion-fun-MonoTran* [*simp*]: *x* ∈ *assertion-fun* ⟹ *mono x*
⟨*proof*⟩

lemma *assertion-fun-le-one* [*simp*]: *x* ∈ *assertion-fun* ⟹ *x* ≤ *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* ⇒ '*a* (- ⌢ *o* [*81*] *80*)

class *omega* =
fixes *omega*::'*a* ⇒ '*a* (- ⌢ ω [*81*] *80*)

class *star* =
fixes *star*::'*a* ⇒ '*a* ((- ⌢ ∗) [*81*] *80*)

class *dual-star* =
fixes *dual-star*::'*a* ⇒ '*a* ((- ⌢ ⊗) [*81*] *80*)

class *mbt-algebra* = *monoid-mult* + *dual* + *omega* + *distrib-lattice* + *order-top* + *order-bot* + *star* + *dual-star* +
assumes
dual-le: (*x* ≤ *y*) = (*y* ⌢ *o* ≤ *x* ⌢ *o*)

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

lemma *le-comp-right*: $x \leq y \Longrightarrow x * z \leq y * z$
⟨*proof*⟩

subclass *bounded-lattice*
⟨*proof*⟩

end

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

lift-definition *dual-MonoTran* :: *'a MonoTran* ⇒ *'a MonoTran*
is *dual-fun*
⟨*proof*⟩

lift-definition *omega-MonoTran* :: *'a MonoTran* ⇒ *'a MonoTran*
is *omega-fun*
⟨*proof*⟩

lift-definition *star-MonoTran* :: *'a MonoTran* ⇒ *'a MonoTran*
is *star-fun*
⟨*proof*⟩

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

instance ⟨*proof*⟩

end

context *mbt-algebra* **begin**

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

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

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

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

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

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

lemma *gt-one-comp*: $1 \leq x \Longrightarrow y \leq x * y$
⟨*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*: $(x * z) \sqcup y \leq z \Longrightarrow (x\hat{}\otimes) * y \leq z$
⟨*proof*⟩

lemma *omega-one* [*simp*]: $1 \frown \omega = \bot$
⟨*proof*⟩

lemma *omega-mono*: $x \leq y \Longrightarrow x \frown \omega \leq y \frown \omega$
⟨*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*: $x \in conjunctive \Longrightarrow x \frown o \in disjunctive$
⟨*proof*⟩

lemma *dual-disjunctive*: $x \in disjunctive \Longrightarrow x \frown o \in conjunctive$
⟨*proof*⟩

lemma *comp-pres-conj*: $x \in conjunctive \Longrightarrow y \in conjunctive \Longrightarrow x * y \in conjunctive$
⟨*proof*⟩

lemma *comp-pres-disj*: $x \in disjunctive \Longrightarrow y \in disjunctive \Longrightarrow x * y \in disjunctive$
⟨*proof*⟩

lemma *start-pres-conj*: $x \in conjunctive \Longrightarrow (x \frown *) \in conjunctive$
⟨*proof*⟩

lemma *dual-star-pres-disj*: $x \in disjunctive \Longrightarrow x\widehat{\otimes} \in disjunctive$
⟨*proof*⟩

3.1 Assertions

Usually, in Kleene algebra with tests or in other progrm 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 $\bot$ and 1. In this formalization assertions can be defined much simpler using the dual operator.

definition
$assertion = \{x \,.\, x \leq 1 \wedge (x * \top) \sqcap (x \frown o) = x\}$

lemma *assertion-prop*: $x \in assertion \Longrightarrow (x * \top) \sqcap 1 = x$
⟨*proof*⟩

lemma *dual-assertion-prop*: $x \in assertion \Longrightarrow ((x \frown o) * \bot) \sqcup 1 = x \frown o$
⟨*proof*⟩

lemma *assertion-disjunctive*: $x \in assertion \Longrightarrow x \in disjunctive$
⟨*proof*⟩

lemma *Abs-MonoTran-injective*: $mono\ x \Longrightarrow mono\ y \Longrightarrow Abs\text{-}MonoTran\ x = Abs\text{-}MonoTran\ y \Longrightarrow x = y$
⟨*proof*⟩
end

lemma *mbta-MonoTran-disjunctive*: *Rep-MonoTran* ‘ *disjunctive* = *Apply.disjunctive*
⟨*proof*⟩

lemma *assertion-MonoTran*: *assertion* = *Abs-MonoTran* ‘ *assertion-fun*
⟨*proof*⟩

context *mbt-algebra* **begin**
lemma *assertion-conjunctive*: $x \in assertion \Longrightarrow x \in conjunctive$
⟨*proof*⟩

lemma *dual-assertion-conjunctive*: $x \in assertion \Longrightarrow x\ \hat{}\ o \in conjunctive$
⟨*proof*⟩

lemma *dual-assertion-disjunct*: $x \in assertion \Longrightarrow x\ \hat{}\ o \in disjunctive$
⟨*proof*⟩

lemma [*simp*]: $x \in assertion \Longrightarrow y \in assertion \Longrightarrow x \sqcap y \leq x * y$
⟨*proof*⟩

lemma [*simp*]: $x \in assertion \Longrightarrow x * y \leq y$
⟨*proof*⟩

lemma [*simp*]: $x \in assertion \Longrightarrow y \in assertion \Longrightarrow x * y \leq x$
⟨*proof*⟩

lemma *assertion-inf-comp-eq*: $x \in assertion \Longrightarrow y \in assertion \Longrightarrow x \sqcap y = x * y$
⟨*proof*⟩

lemma *one-right-assertion* [*simp*]: $x \in assertion \Longrightarrow x * 1 = x$
⟨*proof*⟩

lemma [*simp*]: $x \in assertion \Longrightarrow x \sqcup 1 = 1$
⟨*proof*⟩

lemma [*simp*]: $x \in assertion \Longrightarrow 1 \sqcup x = 1$
⟨*proof*⟩

lemma [*simp*]: $x \in assertion \Longrightarrow x \sqcap 1 = x$
⟨*proof*⟩

lemma [*simp*]: $x \in \mathit{assertion} \Longrightarrow \mathit{1} \sqcap x = x$
⟨*proof*⟩

lemma [*simp*]: $x \in \mathit{assertion} \Longrightarrow x \leq x * \top$
⟨*proof*⟩

lemma [*simp*]: $x \in \mathit{assertion} \Longrightarrow x \leq \mathit{1}$
⟨*proof*⟩

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

lemma *sup-uminus*[*simp*]: $x \in \mathit{assertion} \Longrightarrow x \sqcup \mathit{neg\text{-}assert}\ x = \mathit{1}$
⟨*proof*⟩

lemma *inf-uminus*[*simp*]: $x \in \mathit{assertion} \Longrightarrow x \sqcap \mathit{neg\text{-}assert}\ x = \bot$
⟨*proof*⟩

lemma *uminus-assertion*[*simp*]: $x \in \mathit{assertion} \Longrightarrow \mathit{neg\text{-}assert}\ x \in \mathit{assertion}$
⟨*proof*⟩

lemma *uminus-uminus* [*simp*]: $x \in \mathit{assertion} \Longrightarrow \mathit{neg\text{-}assert}\ (\mathit{neg\text{-}assert}\ x) = x$
⟨*proof*⟩

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

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

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

lemma *inf-assertion* [*simp*]: $x \in \mathit{assertion} \Longrightarrow y \in \mathit{assertion} \Longrightarrow x \sqcap y \in \mathit{assertion}$
⟨*proof*⟩

lemma *comp-assertion* [*simp*]: $x \in \mathit{assertion} \Longrightarrow y \in \mathit{assertion} \Longrightarrow x * y \in \mathit{assertion}$
⟨*proof*⟩

lemma *sup-assertion* [*simp*]: $x \in \mathit{assertion} \Longrightarrow y \in \mathit{assertion} \Longrightarrow x \sqcup y \in \mathit{assertion}$
⟨*proof*⟩

lemma [*simp*]: $x \in \mathit{assertion} \Longrightarrow x * x = x$

⟨*proof*⟩

lemma [*simp*]: $x \in \mathit{assertion} \Longrightarrow (x \frown o) * (x \frown o) = x \frown o$
⟨*proof*⟩

lemma [*simp*]: $x \in \mathit{assertion} \Longrightarrow x * (x \frown o) = x$
⟨*proof*⟩

lemma [*simp*]: $x \in \mathit{assertion} \Longrightarrow (x \frown o) * x = x \frown o$
⟨*proof*⟩

lemma [*simp*]: $\bot \in \mathit{assertion}$
⟨*proof*⟩

lemma [*simp*]: $1 \in \mathit{assertion}$
⟨*proof*⟩

3.2 Weakest precondition of true

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

lemma *wpt-is-assertion* [*simp*]: $\mathit{wpt}\ x \in \mathit{assertion}$
⟨*proof*⟩

lemma *wpt-comp*: $(\mathit{wpt}\ x) * x = x$
⟨*proof*⟩

lemma *wpt-comp-2*: $\mathit{wpt}\ (x * y) = \mathit{wpt}\ (x * (\mathit{wpt}\ y))$
⟨*proof*⟩

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

lemma *wpt-le-assertion*: $x \in \mathit{assertion} \Longrightarrow x * y = y \Longrightarrow \mathit{wpt}\ y \leq x$
⟨*proof*⟩

lemma *wpt-choice*: $\mathit{wpt}\ (x \sqcap y) = \mathit{wpt}\ x \sqcap \mathit{wpt}\ y$
⟨*proof*⟩
end

context *lattice* **begin**
lemma [*simp*]: $x \leq y \Longrightarrow x \sqcap y = x$
⟨*proof*⟩
end

context *mbt-algebra* **begin**

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

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

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

lemma *assertion-commute*: $x \in assertion \Longrightarrow y \in conjunctive \Longrightarrow y * x = wpt(y * x) * y$
⟨*proof*⟩

lemma *wpt-mono*: $x \leq y \Longrightarrow wpt\ x \leq wpt\ y$
⟨*proof*⟩

lemma $a \in conjunctive \Longrightarrow x * a \leq a * y \Longrightarrow (x\ \widehat{}\ \omega) * a \leq a * (y\ \widehat{}\ \omega)$
⟨*proof*⟩

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

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

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

end

3.3 Monotonic Boolean trasformers algebra with post condition statement

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

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

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

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

class *post-mbt-algebra* = *mbt-algebra* +
  fixes *post* :: $'a \Rightarrow 'a$
  assumes *post-1*: $(post\ x) * x * \top = \top$
  and *post-2*: $y * x * \top \sqcap (post\ x) \leq y$

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

lift-definition *post-MonoTran* :: $'a$::*complete-boolean-algebra MonoTran* $\Rightarrow$ $'a$::*complete-boolean-algebra MonoTran*
  is $\lambda x.$ *post-fun* $(x\ \top)$
  ⟨*proof*⟩

instance ⟨*proof*⟩

end

3.4 Complete monotonic Boolean transformers algebra

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

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

context *complete-mbt-algebra* **begin**
lemma *dual-Inf*: $(Inf\ X)\ \hat{}\ o = (SUP\ x \in X\ .\ x\ \hat{}\ o)$
  ⟨*proof*⟩

lemma *dual-Sup*: $(Sup\ X)\ \hat{}\ o = (INF\ x \in X\ .\ x\ \hat{}\ o)$
  ⟨*proof*⟩

lemma *INF-comp*: $(\bigsqcap (f\ `\ A)) * z = (INF\ a \in A\ .\ (f\ a) * z)$
  ⟨*proof*⟩

lemma *dual-INF*: $(\bigsqcap (f\ `\ A))\ \hat{}\ o = (SUP\ a \in A\ .\ (f\ a)\ \hat{}\ o)$
  ⟨*proof*⟩

lemma *dual-SUP*: $(\bigsqcup (f\ `\ A))\ \hat{}\ o = (INF\ a \in A\ .\ (f\ a)\ \hat{}\ o)$
  ⟨*proof*⟩

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

lemma *SUP-comp*: $(\bigsqcup (f\ `\ A)) * z = (SUP\ a \in A\ .\ (f\ a) * z)$
  ⟨*proof*⟩

lemma *Sup-assertion* [*simp*]: $X \subseteq assertion \Longrightarrow Sup\ X \in assertion$

⟨*proof*⟩

lemma *Sup-range-assertion* [*simp*]: (!!*w* . *p w* ∈ *assertion*) ⟹ *Sup* (*range p*) ∈ *assertion*
⟨*proof*⟩

lemma *Sup-less-assertion* [*simp*]: (!!*w* . *p w* ∈ *assertion*) ⟹ *Sup-less p w* ∈ *assertion*
⟨*proof*⟩

theorem *omega-lfp*:
x ^ω ∗ *y* = *lfp* (λ *z* . (*x* ∗ *z*) ⊓ *y*)
⟨*proof*⟩
end

lemma [*simp*]: *mono* (λ (*t*::'*a*::*mbt-algebra*) . *x* ∗ *t* ⊓ *y*)
⟨*proof*⟩

class *mbt-algebra-fusion* = *mbt-algebra* +
assumes *fusion*: (∀ *t* . *x* ∗ *t* ⊓ *y* ⊓ *z* ≤ *u* ∗ (*t* ⊓ *z*) ⊓ *v*)
⟹ (*x* ^ω) ∗ *y* ⊓ *z* ≤ (*u* ^ω) ∗ *v*

lemma
class.mbt-algebra-fusion (*1*::'*a*::*complete-mbt-algebra*) ((∗)) (⊓) (≤) (<) (⊔) *dual dual-star omega star* ⊥ ⊤
⟨*proof*⟩

context *mbt-algebra-fusion*
begin

lemma *omega-star*: *x* ∈ *conjunctive* ⟹ *x* ^ω = *wpt* (*x* ^ω) ∗ (*x* ^∗)
⟨*proof*⟩

lemma *omega-pres-conj*: *x* ∈ *conjunctive* ⟹ *x* ^ω ∈ *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 instroduced 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* ⇒ 'a ({· - } [0] *1000*) **where**
{·p} = *Rep-Assertion p*

definition
abs-wpt x = *Abs-Assertion* (*wpt x*)

lemma [*simp*]: {·p} ∈ *assertion*
⟨*proof*⟩

lemma [*simp*]: *abs-wpt* ({·p}) = *p*
⟨*proof*⟩

lemma [*simp*]: *x* ∈ *assertion* ⟹ {·*Abs-Assertion x*} = *x*
⟨*proof*⟩

lemma [*simp*]: *x* ∈ *assertion* ⟹ {·*abs-wpt x*} = *x*
⟨*proof*⟩

lemma *assert-injective*: {·p} = {·q} ⟹ *p* = *q*
⟨*proof*⟩

instantiation *Assertion* :: (*mbt-algebra*) *boolean-algebra*
begin
definition
uminus-Assertion-def: − *p* = *abs-wpt*(*neg-assert* {·p})

definition
bot-Assertion-def: ⊥ = *abs-wpt* ⊥

definition
top-Assertion-def: ⊤ = *abs-wpt 1*

definition
inf-Assertion-def: *p* ⊓ *q* = *abs-wpt* ({·p} ⊓ {·q})

definition
sup-Assertion-def: *p* ⊔ *q* = *abs-wpt* ({·p} ⊔ {·q})

definition
less-eq-Assertion-def: (*p* ≤ *q*) = ({·p} ≤ {·q})

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

definition
minus-Assertion-def: $(p::'a\ Assertion) - q = p \sqcap - q$

instance
⟨*proof*⟩

end

lemma *assert-image* [*simp*]: *assert* ‘ $A \subseteq$ *assertion*
⟨*proof*⟩

instantiation *Assertion* :: (*complete-mbt-algebra*) *complete-lattice*
begin
definition
Sup-Assertion-def: *Sup* A = *abs-wpt* (*Sup* (*assert* ‘ A))

definition
Inf-Assertion-def: *Inf* $(A::('a\ Assertion)\ set) = -$ (*Sup* (*uminus* ‘ A))

lemma *Sup1*: $(x::'a\ Assertion) \in A \Longrightarrow x \leq$ *Sup* A
⟨*proof*⟩

lemma *Sup2*: $(\bigwedge x::'a\ Assertion\ .\ x \in A \Longrightarrow x \leq z) \Longrightarrow$ *Sup* $A \leq z$
⟨*proof*⟩

instance
⟨*proof*⟩
end

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

lemma *assert-Sup*: $\{\cdot$*Sup* $A\} =$ *Sup* (*assert* ‘ A)
⟨*proof*⟩

lemma *assert-Inf*: $\{\cdot$*Inf* $A\} =$ (*Inf* (*assert* ‘ A)) $\sqcap$ *1*
⟨*proof*⟩

lemma *assert-Inf-ne*: $A \neq \{\} \Longrightarrow \{\cdot$*Inf* $A\} =$ *Inf* (*assert* ‘ A)
⟨*proof*⟩

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

lemma *assert-Sup-less*: $\{\cdot$ *Sup-less p w* $\} =$ *Sup-less* (*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 refienment. 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* ⇒ *'a* ([· -] [*0*] *1000*) **where**
$[\cdot p] = \{\cdot p\} \frown o$

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

lemma *assert-false* [*simp*]: $\{\cdot \bot\} = \bot$
⟨*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*: $wp\ [\cdot p]\ q = -p \sqcup q$
⟨*proof*⟩

lemma *assert-commute*: $y \in conjunctive \Longrightarrow y * \{\cdot p\} = \{\cdot\ wp\ y\ p\ \} * y$
⟨*proof*⟩

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

lemma *wp-mono* [*simp*]: $mono\ (wp\ x)$
⟨*proof*⟩

lemma *wp-mono2*: $p \leq q \Longrightarrow wp\ x\ p \leq wp\ x\ q$
⟨*proof*⟩

lemma *wp-fun-mono* [*simp*]: $mono\ wp$
⟨*proof*⟩

lemma *wp-fun-mono2*: $x \leq y \Longrightarrow wp\ x\ p \leq wp\ y\ p$
⟨*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*: $wp\ (x\ ^\frown\omega)\ p = wp\ x\ (wp\ (x\ ^\frown\omega)\ p) \sqcap p$
⟨*proof*⟩

lemma *wp-omega-least*: $(wp\ x\ r) \sqcap p \leq r \Longrightarrow wp\ (x\ ^\frown\omega)\ p \leq r$
⟨*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

grd x = − (wp x ⊥)

lemma *grd-comp*: *[·grd x] ∗ x = x*
⟨*proof*⟩

lemma *assert-assume*: *{·p} ∗ [·p] = {· p}*
⟨*proof*⟩

lemma *dual-assume*: *[·p] ⌢ o = {·p}*
⟨*proof*⟩

lemma *assume-prop*: *([·p] ∗ ⊥) ⊔ 1 = [·p]*
⟨*proof*⟩

An alternative definition of a Hoare triple

definition *hoare1 p S q = ([· p] ∗ S ∗ [· −q] = ⊤)*

lemma *hoare1 p S q = hoare p S q*
⟨*proof*⟩

lemma *hoare-choice*: *hoare p (x ⊓ y) q = ((hoare p) x q & (hoare p y q))*
⟨*proof*⟩

definition
if-stm:: *'a*::*mbt-algebra Assertion ⇒ 'a ⇒ 'a ⇒ 'a ((If (-)/ then (-)/ else (-)) [0, 0, 10] 10)* **where**
if-stm b x y = (([· b] ∗ x) ⊓ ([· −b] ∗ y))

lemma *if-assertion*: *(If p then x else y) = {·p} ∗ x ⊔ {· −p} ∗ y*
⟨*proof*⟩

lemma *hoare-if*: *hoare p (If b then x else y) q = (hoare (p ⊓ b) x q ∧ hoare (p ⊓ −b) y q)*
⟨*proof*⟩

lemma *hoare-comp*: *hoare p (x ∗ y) q = (∃ r . (hoare p x r) ∧ (hoare r y q))*
⟨*proof*⟩

lemma *hoare-refinement*: *hoare p S q = ({·p} ∗ (post {·q}) ≤ S)*
⟨*proof*⟩

theorem *hoare-fixpoint-mbt*:
F x = x
⟹ (!! (w::'a::well-founded) f . (⋀v. v < w ⟹ hoare (p v) f q) ⟹ hoare (p w) (F f) q)
⟹ hoare (p u) x q
⟨*proof*⟩

lemma *hoare-Sup*: *hoare (Sup P) x q = (∀ p ∈ P . hoare p x q)*

⟨*proof*⟩

theorem *hoare-fixpoint-complete-mbt*:
F x = *x*
⟹ (!! *w f* . *hoare* (*Sup-less p w*) *f q* ⟹ *hoare* (*p w*) (*F f*) *q*)
⟹ *hoare* (*Sup* (*range p*)) *x q*
⟨*proof*⟩

definition
while:: *'a*::*mbt-algebra Assertion* ⇒ *'a* ⇒ *'a* ((*While* (-)/ *do* (-)) [*0*, *10*] *10*)
where
while p x = ([· *p*] ∗ *x*) ^ω ∗ [· −*p*]

lemma *while-false*: (*While* ⊥ *do x*) = *1*
⟨*proof*⟩

lemma *while-true*: (*While* ⊤ *do 1*) = ⊥
⟨*proof*⟩

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

lemma *hoare-comp-wp*: *hoare p* (*x* ∗ *y*) *q* = *hoare p x* (*wp y q*)
⟨*proof*⟩

lemma (**in** *mbt-algebra*) *hoare-assume*: *hoare p* [·*b*] *q* = (*p* ⊓ *b* ≤ *q*)
⟨*proof*⟩

lemma (**in** *mbt-algebra*) *hoare-assume-comp*: *hoare p* ([·*b*] ∗ *x*) *q* = *hoare* (*p* ⊓ *b*) *x q*
⟨*proof*⟩

lemma *hoare-while-mbt*:
(∀ (*w*::*'b*::*well-founded*) *r* . (∀ *v* . *v* < *w* ⟶ *p v* ≤ *r*) ⟶ *hoare* ((*p w*) ⊓ *b*) *x r*) ⟹
(∀ *u* . *p u* ≤ *q*) ⟹ *hoare* (*p w*) (*While b do x*) (*q* ⊓ −*b*)
⟨*proof*⟩

lemma *hoare-while-complete-mbt*:
(∀ *w*::*'b*::*well-founded* . *hoare* ((*p w*) ⊓ *b*) *x* (*Sup-less p w*)) ⟹
hoare (*Sup* (*range p*)) (*While b do x*) ((*Sup* (*range p*)) ⊓ −*b*)
⟨*proof*⟩

definition
datarefin S S1 D D1 = (*D* ∗ *S* ≤ *S1* ∗ *D1*)

lemma *hoare p S q* ⟹ *datarefin S S1 D D1* ⟹ *hoare* (*wp D p*) *S1* (*wp D1 q*)
⟨*proof*⟩

lemma *hoare p S q* $\Longrightarrow$ *datarefin* $(\{\cdot p\} * S)$ *S1 D D1* $\Longrightarrow$ *hoare* (*wp D p*) *S1* (*wp D1 q*)
⟨*proof*⟩

lemma *inf-pres-conj*: $x \in$ *conjunctive* $\Longrightarrow$ $y \in$ *conjunctive* $\Longrightarrow$ $x \sqcap y \in$ *conjunctive*
⟨*proof*⟩

lemma *sup-pres-disj*: $x \in$ *disjunctive* $\Longrightarrow$ $y \in$ *disjunctive* $\Longrightarrow$ $x \sqcup y \in$ *disjunctive*
⟨*proof*⟩

lemma *assumption-conjuncive* [*simp*]: $[\cdot p] \in$ *conjunctive*
⟨*proof*⟩

lemma *assumption-disjuncive* [*simp*]: $[\cdot p] \in$ *disjunctive*
⟨*proof*⟩

lemma *if-pres-conj*: $x \in$ *conjunctive* $\Longrightarrow$ $y \in$ *conjunctive* $\Longrightarrow$ (*If p then x else y*) $\in$ *conjunctive*
⟨*proof*⟩

lemma *if-pres-disj*: $x \in$ *disjunctive* $\Longrightarrow$ $y \in$ *disjunctive* $\Longrightarrow$ (*If p then x else y*) $\in$ *disjunctive*
⟨*proof*⟩

lemma *while-dual-star*: (*While p do* (x::'a::*mbt-algebra*)) = $((\{\cdot\ p\} * x)^{\frown\otimes} * \{\cdot -p\})$
⟨*proof*⟩

lemma *while-pres-disj*: (x::'a::*mbt-algebra*) $\in$ *disjunctive* $\Longrightarrow$ (*While p do x*) $\in$ *disjunctive*
⟨*proof*⟩

lemma *while-pres-conj*: (x::'a::*mbt-algebra-fusion*) $\in$ *conjunctive* $\Longrightarrow$ (*While p do x*) $\in$ *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.