

Unbounded Separation Logic

Thibault Dardinier

Department of Computer Science, ETH Zurich, Switzerland

March 19, 2025

Abstract

Many separation logics [11] support fractional permissions [3, 2] to distinguish between read and write access to a heap location, for instance, to allow concurrent reads while enforcing exclusive writes. Fractional permissions extend to composite assertions such as (co)inductive predicates and magic wands by allowing those to be multiplied [8, 4, 6] by a fraction. Typical separation logic proofs require that this multiplication has three key properties: it needs to distribute over assertions, it should permit fractions to be factored out from assertions, and two fractions of the same assertion should be combinable into one larger fraction.

Existing formal semantics incorporating fractional assertions into a separation logic define multiplication semantically (via models), resulting in a semantics in which distributivity and combinability do not hold for key resource assertions such as magic wands, and fractions cannot be factored out from a separating conjunction. By contrast, existing automatic separation logic verifiers [9, 7, 10, 1] define multiplication syntactically, resulting in a different semantics for which it is unknown whether distributivity and combinability hold for all assertions.

In this entry, we present and formalize an *unbounded* version of separation logic [5], a novel semantics for separation logic assertions that allows states to hold more than a full permission to a heap location during the evaluation of an assertion. By reimposing upper bounds on the permissions held per location at statement boundaries, we retain key properties of separation logic, in particular, we prove that the frame rule still holds. We also prove that our assertion semantics unifies semantic and syntactic multiplication and thereby reconciles the discrepancy between separation logic theory and tools and enjoys distributivity, factorisability, and combinability.

Contents

1	Unbounded Separation Logic	3
1.1	Assertions and state model	3
1.2	Useful lemmas	5

2	Frame rule	7
3	Distributivity and Factorisability	9
3.1	DotPos	9
3.2	DotDot	9
3.3	DotStar	10
3.4	DotWand	10
3.5	DotOr	10
3.6	DotAnd	11
3.7	DotImp	11
3.8	DotPure	11
3.9	DotFull	12
3.10	DotExists	12
3.11	DotForall	12
3.12	Split	13
4	Combinability	13
5	(Co)Inductive Predicates	16
5.1	Definitions	16
5.2	Everything preserves monotonicity	17
5.2.1	Monotonicity	18
5.2.2	Non-increasing	19
5.3	Tarski's fixed points	21
5.3.1	Greatest Fixed Point	21
5.3.2	Least Fixed Point	21
5.4	Combinability and (an assertion being) intuitionistic are set- closure properties	22
5.4.1	Intuitionistic assertions	22
5.4.2	Combinable assertions	22
5.5	Transfinite induction	23
5.6	Theorems	25
5.6.1	Greatest Fixed Point	25
5.6.2	Least Fixed Point	25
6	Properties of Magic Wands	26
7	Fractional Predicates and Magic Wands in Automatic Sep- aration Logic Verifiers	27
7.1	Syntactic multiplication	27
7.2	Monotonicity and fixed point	28
7.3	Combinability	29
7.4	Theorems	29

1 Unbounded Separation Logic

```
theory UnboundedLogic
  imports Main
begin
```

1.1 Assertions and state model

We define our assertion language as described in Section 2.3 of the paper [5].

```
datatype ('a, 'b, 'c, 'd) assertion =
  Sem ('d  $\Rightarrow$  'c)  $\Rightarrow$  'a  $\Rightarrow$  bool
  | Mult 'b ('a, 'b, 'c, 'd) assertion
  | Star ('a, 'b, 'c, 'd) assertion ('a, 'b, 'c, 'd) assertion
  | Wand ('a, 'b, 'c, 'd) assertion ('a, 'b, 'c, 'd) assertion
  | Or ('a, 'b, 'c, 'd) assertion ('a, 'b, 'c, 'd) assertion
  | And ('a, 'b, 'c, 'd) assertion ('a, 'b, 'c, 'd) assertion
  | Imp ('a, 'b, 'c, 'd) assertion ('a, 'b, 'c, 'd) assertion
  | Exists 'd ('a, 'b, 'c, 'd) assertion
  | Forall 'd ('a, 'b, 'c, 'd) assertion
  | Pred
  | Bounded ('a, 'b, 'c, 'd) assertion
  | Wildcard ('a, 'b, 'c, 'd) assertion

type-synonym 'a command = ('a  $\times$  'a option) set

locale pre-logic =
  fixes plus :: 'a  $\Rightarrow$  'a  $\Rightarrow$  'a option (infixl <math>\oplus\Rightarrow 'a  $\Rightarrow$  bool (infixl <math>\langle\#\#\rangle 60) where
  a  $\langle\#\#\rangle$  b  $\longleftrightarrow$  a  $\oplus$  b  $\neq$  None

definition larger :: 'a  $\Rightarrow$  'a  $\Rightarrow$  bool (infixl <math>\langle\succ\rangle 55) where
  a  $\langle\succ\rangle$  b  $\longleftrightarrow$  ( $\exists$  c. Some a = b  $\oplus$  c)

end

type-synonym ('a, 'b, 'c) interp = ('a  $\Rightarrow$  'b)  $\Rightarrow$  'c set

The following locale captures the state model described in Section 2.2 of the
paper [5].

locale logic = pre-logic +

  fixes mult :: 'b  $\Rightarrow$  'a  $\Rightarrow$  'a (infixl <math>\langle\odot\rangle 64)

  fixes smult :: 'b  $\Rightarrow$  'b  $\Rightarrow$  'b
  fixes sadd :: 'b  $\Rightarrow$  'b  $\Rightarrow$  'b
  fixes sinv :: 'b  $\Rightarrow$  'b
```

fixes *one* :: 'b

fixes *valid* :: 'a ⇒ bool

assumes *commutative*: $a \oplus b = b \oplus a$

and *asso1*: $a \oplus b = \text{Some } ab \wedge b \oplus c = \text{Some } bc \implies ab \oplus c = a \oplus bc$

and *asso2*: $a \oplus b = \text{Some } ab \wedge \neg b \#\# c \implies \neg ab \#\# c$

and *sinv-inverse*: $\text{smult } p (\text{sinv } p) = \text{one}$

and *sone-neutral*: $\text{smult } \text{one } p = p$

and *sadd-comm*: $\text{sadd } p q = \text{sadd } q p$

and *smult-comm*: $\text{smult } p q = \text{smult } q p$

and *smult-distrib*: $\text{smult } p (\text{sadd } q r) = \text{sadd } (\text{smult } p q) (\text{smult } p r)$

and *smult-asso*: $\text{smult } (\text{smult } p q) r = \text{smult } p (\text{smult } q r)$

and *double-mult*: $p \odot (q \odot a) = (\text{smult } p q) \odot a$

and *plus-mult*: $\text{Some } a = b \oplus c \implies \text{Some } (p \odot a) = (p \odot b) \oplus (p \odot c)$

and *distrib-mult*: $\text{Some } ((\text{sadd } p q) \odot x) = p \odot x \oplus q \odot x$

and *one-neutral*: $\text{one} \odot a = a$

and *valid-mono*: $\text{valid } a \wedge a \succeq b \implies \text{valid } b$

begin

The validity of assertions corresponds to Figure 3 of the paper [5].

fun *sat* :: 'a ⇒ ('d ⇒ 'c) ⇒ ('d, 'c, 'a) *interp* ⇒ ('a, 'b, 'c, 'd) *assertion* ⇒ bool

(←, -, - ⊨ -) [51, 65, 68, 66] 50) **where**

| $\sigma, s, \Delta \models \text{Mult } p A \longleftrightarrow (\exists a. \sigma = p \odot a \wedge a, s, \Delta \models A)$

| $\sigma, s, \Delta \models \text{Star } A B \longleftrightarrow (\exists a b. \text{Some } \sigma = a \oplus b \wedge a, s, \Delta \models A \wedge b, s, \Delta \models B)$

| $\sigma, s, \Delta \models \text{Wand } A B \longleftrightarrow (\forall a \sigma'. a, s, \Delta \models A \wedge \text{Some } \sigma' = \sigma \oplus a \longrightarrow \sigma', s, \Delta \models B)$

| $\sigma, s, \Delta \models \text{Sem } b \longleftrightarrow b s \sigma$

| $\sigma, s, \Delta \models \text{Imp } A B \longleftrightarrow (\sigma, s, \Delta \models A \longrightarrow \sigma, s, \Delta \models B)$

| $\sigma, s, \Delta \models \text{Or } A B \longleftrightarrow (\sigma, s, \Delta \models A \vee \sigma, s, \Delta \models B)$

| $\sigma, s, \Delta \models \text{And } A B \longleftrightarrow (\sigma, s, \Delta \models A \wedge \sigma, s, \Delta \models B)$

| $\sigma, s, \Delta \models \text{Exists } x A \longleftrightarrow (\exists v. \sigma, s(x := v), \Delta \models A)$

| $\sigma, s, \Delta \models \text{Forall } x A \longleftrightarrow (\forall v. \sigma, s(x := v), \Delta \models A)$

| $\sigma, s, \Delta \models \text{Pred} \longleftrightarrow (\sigma \in \Delta s)$

| $\sigma, s, \Delta \models \text{Bounded } A \longleftrightarrow (\text{valid } \sigma \longrightarrow \sigma, s, \Delta \models A)$

| $\sigma, s, \Delta \models \text{Wildcard } A \longleftrightarrow (\exists a p. \sigma = p \odot a \wedge a, s, \Delta \models A)$

definition *intuitionistic* :: ('d ⇒ 'c) ⇒ ('d, 'c, 'a) *interp* ⇒ ('a, 'b, 'c, 'd) *assertion*

⇒ bool **where**

intuitionistic $s \Delta A \longleftrightarrow (\forall a b. a \succeq b \wedge b, s, \Delta \models A \longrightarrow a, s, \Delta \models A)$

definition *entails* :: ('a, 'b, 'c, 'd) assertion \Rightarrow ('d, 'c, 'a) interp \Rightarrow ('a, 'b, 'c, 'd) assertion \Rightarrow bool ($\langle -, - \vdash - \rangle$ [63, 66, 68] 52) **where**
 $A, \Delta \vdash B \longleftrightarrow (\forall \sigma \ s. \ \sigma, s, \Delta \models A \longrightarrow \sigma, s, \Delta \models B)$

definition *equivalent* :: ('a, 'b, 'c, 'd) assertion \Rightarrow ('d, 'c, 'a) interp \Rightarrow ('a, 'b, 'c, 'd) assertion \Rightarrow bool ($\langle -, - \equiv - \rangle$ [63, 66, 68] 52) **where**
 $A, \Delta \equiv B \longleftrightarrow (A, \Delta \vdash B \wedge B, \Delta \vdash A)$

definition *pure* :: ('a, 'b, 'c, 'd) assertion \Rightarrow bool **where**
 $pure\ A \longleftrightarrow (\forall \sigma \ \sigma' \ s \ \Delta \ \Delta'. \ \sigma, s, \Delta \models A \longleftrightarrow \sigma', s, \Delta' \models A)$

1.2 Useful lemmas

lemma *sat-forall*:

assumes $\bigwedge v. \ \sigma, s(x := v), \Delta \models A$
shows $\sigma, s, \Delta \models \text{Forall } x\ A$
 $\langle \text{proof} \rangle$

lemma *intuitionisticI*:

assumes $\bigwedge a \ b. \ a \succeq b \wedge b, s, \Delta \models A \Longrightarrow a, s, \Delta \models A$
shows *intuitionistic* $s \ \Delta \ A$
 $\langle \text{proof} \rangle$

lemma *can-divide*:

assumes $p \odot a = p \odot b$
shows $a = b$
 $\langle \text{proof} \rangle$

lemma *unique-inv*:

$a = p \odot b \longleftrightarrow b = (\text{sinv } p) \odot a$
 $\langle \text{proof} \rangle$

lemma *entailsI*:

assumes $\bigwedge \sigma \ s. \ \sigma, s, \Delta \models A \Longrightarrow \sigma, s, \Delta \models B$
shows $A, \Delta \vdash B$
 $\langle \text{proof} \rangle$

lemma *equivalentI*:

assumes $\bigwedge \sigma \ s. \ \sigma, s, \Delta \models A \Longrightarrow \sigma, s, \Delta \models B$
and $\bigwedge \sigma \ s. \ \sigma, s, \Delta \models B \Longrightarrow \sigma, s, \Delta \models A$
shows $A, \Delta \equiv B$
 $\langle \text{proof} \rangle$

lemma *compatible-imp*:

assumes $a \#\# b$
shows $(p \odot a) \#\# (p \odot b)$
 $\langle \text{proof} \rangle$

lemma compatible-iff:

$$a \#\# b \longleftrightarrow (p \odot a) \#\# (p \odot b)$$

\langle proof \rangle

lemma sat-wand:

$$\text{assumes } \bigwedge a \sigma'. a, s, \Delta \models A \wedge \text{Some } \sigma' = \sigma \oplus a \implies \sigma', s, \Delta \models B$$

$$\text{shows } \sigma, s, \Delta \models \text{Wand } A B$$

\langle proof \rangle

lemma sat-imp:

$$\text{assumes } \sigma, s, \Delta \models A \implies \sigma, s, \Delta \models B$$

$$\text{shows } \sigma, s, \Delta \models \text{Imp } A B$$

\langle proof \rangle

lemma sat-mult:

$$\text{assumes } \bigwedge a. \sigma = p \odot a \implies a, s, \Delta \models A$$

$$\text{shows } \sigma, s, \Delta \models \text{Mult } p A$$

\langle proof \rangle

lemma larger-same:

$$a \succeq b \longleftrightarrow p \odot a \succeq p \odot b$$

\langle proof \rangle

lemma asso3:

$$\text{assumes } \neg a \#\# b$$

$$\text{and } b \oplus c = \text{Some } bc$$

$$\text{shows } \neg a \#\# bc$$

\langle proof \rangle

lemma compatible-smaller:

$$\text{assumes } a \succeq b$$

$$\text{and } x \#\# a$$

$$\text{shows } x \#\# b$$

\langle proof \rangle

lemma compatible-multiples:

$$\text{assumes } p \odot a \#\# q \odot b$$

$$\text{shows } a \#\# b$$

\langle proof \rangle

lemma move-sum:

$$\text{assumes } \text{Some } a = a1 \oplus a2$$

$$\text{and } \text{Some } b = b1 \oplus b2$$

$$\text{and } \text{Some } x = a \oplus b$$

$$\text{and } \text{Some } x1 = a1 \oplus b1$$

$$\text{and } \text{Some } x2 = a2 \oplus b2$$

$$\text{shows } \text{Some } x = x1 \oplus x2$$

\langle proof \rangle

lemma *sum-both-larger*:
assumes *Some* $x' = a' \oplus b'$
and *Some* $x = a \oplus b$
and $a' \succeq a$
and $b' \succeq b$
shows $x' \succeq x$
 \langle *proof* \rangle

lemma *larger-first-sum*:
assumes *Some* $y = a \oplus b$
and $x \succeq y$
shows $\exists a'. \text{Some } x = a' \oplus b \wedge a' \succeq a$
 \langle *proof* \rangle

lemma *larger-implies-compatible*:
assumes $x \succeq y$
shows $x \#\# y$
 \langle *proof* \rangle

2 Frame rule

This section corresponds to Section 2.5 of the paper [5].

definition *safe* $:: ('a \times ('d \Rightarrow 'c)) \text{ command} \Rightarrow ('a \times ('d \Rightarrow 'c)) \Rightarrow \text{bool}$ **where**
safe $c \sigma \longleftrightarrow (\sigma, \text{None}) \notin c$

definition *safety-monotonicity* $:: ('a \times ('d \Rightarrow 'c)) \text{ command} \Rightarrow \text{bool}$ **where**
safety-monotonicity $c \longleftrightarrow (\forall \sigma \sigma' s. \text{valid } \sigma' \wedge \sigma' \succeq \sigma \wedge \text{safe } c (\sigma, s) \longrightarrow \text{safe } c (\sigma', s))$

definition *frame-property* $:: ('a \times ('d \Rightarrow 'c)) \text{ command} \Rightarrow \text{bool}$ **where**
frame-property $c \longleftrightarrow (\forall \sigma \sigma \theta r \sigma' s s'. \text{valid } \sigma \wedge \text{valid } \sigma' \wedge \text{safe } c (\sigma \theta, s) \wedge \text{Some } \sigma = \sigma \theta \oplus r \wedge ((\sigma, s), \text{Some } (\sigma', s')) \in c \longrightarrow (\exists \sigma \theta'. \text{Some } \sigma' = \sigma \theta' \oplus r \wedge ((\sigma \theta, s), \text{Some } (\sigma \theta', s')) \in c))$

definition *valid-hoare-triple* $:: ('a, 'b, 'c, 'd) \text{ assertion} \Rightarrow ('a \times ('d \Rightarrow 'c)) \text{ command} \Rightarrow ('a, 'b, 'c, 'd) \text{ assertion} \Rightarrow ('d, 'c, 'a) \text{ interp} \Rightarrow \text{bool}$ **where**
valid-hoare-triple $P c Q \Delta \longleftrightarrow (\forall \sigma s. \text{valid } \sigma \wedge \sigma, s, \Delta \models P \longrightarrow \text{safe } c (\sigma, s) \wedge (\forall \sigma' s'. ((\sigma, s), \text{Some } (\sigma', s')) \in c \longrightarrow \sigma', s', \Delta \models Q))$

lemma *valid-hoare-tripleI*:
assumes $\bigwedge \sigma s. \text{valid } \sigma \wedge \sigma, s, \Delta \models P \Longrightarrow \text{safe } c (\sigma, s)$
and $\bigwedge \sigma s \sigma' s'. \text{valid } \sigma \wedge \sigma, s, \Delta \models P \Longrightarrow ((\sigma, s), \text{Some } (\sigma', s')) \in c \Longrightarrow \sigma', s', \Delta \models Q$
shows *valid-hoare-triple* $P c Q \Delta$
 \langle *proof* \rangle

definition *valid-command* $:: ('a \times ('d \Rightarrow 'c)) \text{ command} \Rightarrow \text{bool}$ **where**
valid-command $c \longleftrightarrow (\forall a b \text{ sa sb}. ((a, \text{sa}), \text{Some } (b, \text{sb})) \in c \wedge \text{valid } a \longrightarrow \text{valid } c)$

b)

definition *modified* :: ('a × ('d ⇒ 'c)) command ⇒ 'd set **where**
modified c = { x | x. ∃ σ s σ' s'. ((σ, s), Some (σ', s')) ∈ c ∧ s x ≠ s' x }

definition *equal-outside* :: ('d ⇒ 'c) ⇒ ('d ⇒ 'c) ⇒ 'd set ⇒ bool **where**
equal-outside s s' S ↔ (∀ x. x ∉ S → s x = s' x)

definition *not-in-fv* :: ('a, 'b, 'c, 'd) assertion ⇒ 'd set ⇒ bool **where**
not-in-fv A S ↔ (∀ σ s Δ s'. *equal-outside* s s' S → (σ, s, Δ ⊨ A ↔ σ, s', Δ ⊨ A))

lemma *not-in-fv-mod*:
 assumes *not-in-fv* A (*modified* c)
 and ((σ, s), Some (σ', s')) ∈ c
 shows x, s, Δ ⊨ A ↔ x, s', Δ ⊨ A
 ⟨*proof*⟩

This theorem corresponds to Theorem 2 of the paper [5].

theorem *frame-rule*:
 assumes *valid-command* c
 and *safety-monotonicity* c
 and *frame-property* c
 and *valid-hoare-triple* P c Q Δ
 and *not-in-fv* R (*modified* c)
 shows *valid-hoare-triple* (Star P R) c (Star Q R) Δ
 ⟨*proof*⟩

lemma *hoare-triple-input*:
 valid-hoare-triple P c Q Δ ↔ *valid-hoare-triple* (Bounded P) c Q Δ
 ⟨*proof*⟩

lemma *hoare-triple-output*:
 assumes *valid-command* c
 shows *valid-hoare-triple* P c Q Δ ↔ *valid-hoare-triple* P c (Bounded Q) Δ
 ⟨*proof*⟩

end

end

3 Distributivity and Factorisability

This section corresponds to Section 2.4 and Figure 4 of the paper [5].

```
theory Distributivity
  imports UnboundedLogic
begin
```

```
context logic
begin
```

3.1 DotPos

```
lemma DotPos:
   $A, \Delta \vdash B \longleftrightarrow (Mult \ \pi \ A, \Delta \vdash Mult \ \pi \ B)$  (is  $?A \longleftrightarrow ?B$ )
  <proof>
```

Only one direction holds with a wildcard

```
lemma WildPos:
   $A, \Delta \vdash B \implies (Wildcard \ A, \Delta \vdash Wildcard \ B)$ 
  <proof>
```

3.2 DotDot

```
lemma dot-mult1:
   $Mult \ p \ (Mult \ q \ A), \Delta \vdash Mult \ (smult \ p \ q) \ A$ 
  <proof>
```

```
lemma dot-mult2:
   $Mult \ (smult \ p \ q) \ A, \Delta \vdash Mult \ p \ (Mult \ q \ A)$ 
  <proof>
```

```
lemma DotDot:
   $Mult \ p \ (Mult \ q \ A), \Delta \equiv Mult \ (smult \ p \ q) \ A$ 
  <proof>
```

```
lemma can-factorize:
   $\exists r. q = smult \ r \ p$ 
  <proof>
```

```
lemma WildDot:
   $Wildcard \ (Mult \ p \ A), \Delta \equiv Wildcard \ A$ 
  <proof>
```

```
lemma DotWild:
   $Mult \ p \ (Wildcard \ A), \Delta \equiv Wildcard \ A$ 
  <proof>
```

```
lemma WildWild:
   $Wildcard \ (Wildcard \ A), \Delta \equiv Wildcard \ A$ 
```

$\langle proof \rangle$

3.3 DotStar

lemma *dot-star1*:

$Mult\ p\ (Star\ A\ B), \Delta \vdash Star\ (Mult\ p\ A)\ (Mult\ p\ B)$
 $\langle proof \rangle$

lemma *dot-star2*:

$Star\ (Mult\ p\ A)\ (Mult\ p\ B), \Delta \vdash Mult\ p\ (Star\ A\ B)$
 $\langle proof \rangle$

lemma *DotStar*:

$Mult\ p\ (Star\ A\ B), \Delta \equiv Star\ (Mult\ p\ A)\ (Mult\ p\ B)$
 $\langle proof \rangle$

lemma *WildStar1*:

$Wildcard\ (Star\ A\ B), \Delta \vdash Star\ (Wildcard\ A)\ (Wildcard\ B)$
 $\langle proof \rangle$

3.4 DotWand

lemma *dot-wand1*:

$Mult\ p\ (Wand\ A\ B), \Delta \vdash Wand\ (Mult\ p\ A)\ (Mult\ p\ B)$
 $\langle proof \rangle$

lemma *dot-wand2*:

$Wand\ (Mult\ p\ A)\ (Mult\ p\ B), \Delta \vdash Mult\ p\ (Wand\ A\ B)$
 $\langle proof \rangle$

lemma *DotWand*:

$Mult\ p\ (Wand\ A\ B), \Delta \equiv Wand\ (Mult\ p\ A)\ (Mult\ p\ B)$
 $\langle proof \rangle$

3.5 DotOr

lemma *dot-or1*:

$Mult\ p\ (Or\ A\ B), \Delta \vdash Or\ (Mult\ p\ A)\ (Mult\ p\ B)$
 $\langle proof \rangle$

lemma *dot-or2*:

$Or\ (Mult\ p\ A)\ (Mult\ p\ B), \Delta \vdash Mult\ p\ (Or\ A\ B)$
 $\langle proof \rangle$

lemma *DotOr*:

$Mult\ p\ (Or\ A\ B), \Delta \equiv Or\ (Mult\ p\ A)\ (Mult\ p\ B)$
 $\langle proof \rangle$

lemma *WildOr*:

Wildcard (Or A B), Δ ≡ Or (Wildcard A) (Wildcard B)
⟨proof⟩

3.6 DotAnd

lemma *dot-and1*:
Mult p (And A B), Δ ⊢ And (Mult p A) (Mult p B)
⟨proof⟩

lemma *dot-and2*:
And (Mult p A) (Mult p B), Δ ⊢ Mult p (And A B)
⟨proof⟩

lemma *DotAnd*:
And (Mult p A) (Mult p B), Δ ≡ Mult p (And A B)
⟨proof⟩

lemma *WildAnd*:
Wildcard (And A B), Δ ⊢ And (Wildcard A) (Wildcard B)
⟨proof⟩

3.7 DotImp

lemma *dot-imp1*:
Imp (Mult p A) (Mult p B), Δ ⊢ Mult p (Imp A B)
⟨proof⟩

lemma *dot-imp2*:
Mult p (Imp A B), Δ ⊢ Imp (Mult p A) (Mult p B)
⟨proof⟩

lemma *DotImp*:
Mult p (Imp A B), Δ ≡ Imp (Mult p A) (Mult p B)
⟨proof⟩

3.8 DotPure

lemma *pure-mult1*:
assumes *pure A*
shows *Mult p A, Δ ⊢ A*
⟨proof⟩

lemma *pure-mult2*:
assumes *pure A*
shows *A, Δ ⊢ Mult p A*
⟨proof⟩

lemma *DotPure*:
assumes *pure A*
shows *Mult p A, Δ ≡ A*

<proof>

lemma *WildPure*:
 assumes *pure A*
 shows *Wildcard A, $\Delta \equiv A$*
<proof>

3.9 DotFull

lemma *mult-one-same1*:
 Mult one A, $\Delta \vdash A$
<proof>

lemma *mult-one-same2*:
 A, $\Delta \vdash \text{Mult one } A$
<proof>

lemma *DotFull*:
 Mult one A, $\Delta \equiv A$
<proof>

3.10 DotExists

lemma *dot-exists1*:
 Mult p (Exists x A), $\Delta \vdash \text{Exists } x (\text{Mult } p A)$
<proof>

lemma *dot-exists2*:
 Exists x (Mult p A), $\Delta \vdash \text{Mult } p (\text{Exists } x A)$
<proof>

lemma *DotExists*:
 Mult p (Exists x A), $\Delta \equiv \text{Exists } x (\text{Mult } p A)$
<proof>

lemma *WildExists*:
 Wildcard (Exists x A), $\Delta \equiv \text{Exists } x (\text{Wildcard } A)$
<proof>

3.11 DotForall

lemma *dot-forall1*:
 Mult p (Forall x A), $\Delta \vdash \text{Forall } x (\text{Mult } p A)$
<proof>

lemma *dot-forall2*:
 Forall x (Mult p A), $\Delta \vdash \text{Mult } p (\text{Forall } x A)$
<proof>

lemma *DotForall*:
 $Mult\ p\ (Forall\ x\ A), \Delta \equiv Forall\ x\ (Mult\ p\ A)$
 $\langle proof \rangle$

lemma *WildForall*:
 $Wildcard\ (Forall\ x\ A), \Delta \vdash Forall\ x\ (Wildcard\ A)$
 $\langle proof \rangle$

3.12 Split

lemma *split*:
 $Mult\ (sadd\ a\ b)\ A, \Delta \vdash Star\ (Mult\ a\ A)\ (Mult\ b\ A)$
 $\langle proof \rangle$

end

end

4 Combinability

This section corresponds to Section 3 of the paper [5].

theory *Combinability*
imports *UnboundedLogic*
begin

context *logic*
begin

The definition of combinable assertions corresponds to Definition 4 of the paper [5].

definition *combinable* :: $('d, 'c, 'a)\ interp \Rightarrow ('a, 'b, 'c, 'd)\ assertion \Rightarrow bool$
where
 $combinable\ \Delta\ A \iff (\forall p\ q.\ Star\ (Mult\ p\ A)\ (Mult\ q\ A), \Delta \vdash Mult\ (sadd\ p\ q)\ A)$

lemma *combinable-instantiate*:
assumes $combinable\ \Delta\ A$
and $a, s, \Delta \models A$
and $b, s, \Delta \models A$
and $Some\ x = p \odot a \oplus q \odot b$
shows $x, s, \Delta \models Mult\ (sadd\ p\ q)\ A$
 $\langle proof \rangle$

lemma *combinable-instantiate-one*:
assumes $combinable\ \Delta\ A$
and $a, s, \Delta \models A$
and $b, s, \Delta \models A$

and *Some* $x = p \odot a \oplus q \odot b$
and *sadd* $p \ q = \text{one}$
shows $x, s, \Delta \models A$
<proof>

lemma *combinableI-old:*

assumes $\bigwedge a \ b \ p \ q \ x \ \sigma \ s. a, s, \Delta \models A \wedge b, s, \Delta \models A \wedge \text{Some } \sigma = p \odot a \oplus q \odot b \wedge \sigma = (\text{sadd } p \ q) \odot x \implies x, s, \Delta \models A$
shows *combinable* $\Delta \ A$
<proof>

lemma *combinableI:*

assumes $\bigwedge a \ b \ p \ q \ x \ \sigma \ s. a, s, \Delta \models A \wedge b, s, \Delta \models A \wedge \text{Some } x = p \odot a \oplus q \odot b \wedge \text{sadd } p \ q = \text{one} \implies x, s, \Delta \models A$
shows *combinable* $\Delta \ A$
<proof>

lemma *combinable-wand:*

assumes *combinable* $\Delta \ B$
shows *combinable* $\Delta \ (\text{Wand } A \ B)$
<proof>

lemma *combinable-star:*

assumes *combinable* $\Delta \ A$
and *combinable* $\Delta \ B$
shows *combinable* $\Delta \ (\text{Star } A \ B)$
<proof>

lemma *combinable-mult:*

assumes *combinable* $\Delta \ A$
shows *combinable* $\Delta \ (\text{Mult } \pi \ A)$
<proof>

lemma *combinable-and:*

assumes *combinable* $\Delta \ A$
and *combinable* $\Delta \ B$
shows *combinable* $\Delta \ (\text{And } A \ B)$
<proof>

lemma *combinable-forall:*

assumes *combinable* $\Delta \ A$
shows *combinable* $\Delta \ (\text{Forall } x \ A)$
<proof>

definition *unambiguous where*

unambiguous $\Delta A x \longleftrightarrow (\forall \sigma 1 \ \sigma 2 \ v 1 \ v 2 \ s. \ \sigma 1 \ \#\# \ \sigma 2 \wedge \sigma 1, s(x := v 1), \Delta \models A \wedge \sigma 2, s(x := v 2), \Delta \models A \longrightarrow v 1 = v 2)$

lemma *unambiguousI:*

assumes $\bigwedge \sigma 1 \ \sigma 2 \ v 1 \ v 2 \ s. \ \sigma 1 \ \#\# \ \sigma 2 \wedge \sigma 1, s(x := v 1), \Delta \models A \wedge \sigma 2, s(x := v 2), \Delta \models A \implies v 1 = v 2$
shows *unambiguous* $\Delta A x$
<proof>

lemma *unambiguous-star:*

assumes *unambiguous* $\Delta A x$
shows *unambiguous* $\Delta (\text{Star } A \ B) x$
<proof>

lemma *combinable-exists:*

assumes *combinable* ΔA
and *unambiguous* $\Delta A x$
shows *combinable* $\Delta (\text{Exists } x \ A)$
<proof>

lemma *combinable-pure:*

assumes *pure* A
shows *combinable* ΔA
<proof>

lemma *combinable-imp:*

assumes *pure* A
and *combinable* ΔB
shows *combinable* $\Delta (\text{Imp } A \ B)$
<proof>

lemma *combinable-wildcard:*

assumes *combinable* ΔA
shows *combinable* $\Delta (\text{Wildcard } A)$
<proof>

end

end

5 (Co)Inductive Predicates

This subsection corresponds to Section 4 of the paper [5].

theory *FixedPoint*

imports *Distributivity Combinability*

begin

type-synonym $('d, 'c, 'a)$ *chain* = $\text{nat} \Rightarrow ('d, 'c, 'a)$ *interp*

context *logic*

begin

5.1 Definitions

definition *smaller-interp* :: $('d, 'c, 'a)$ *interp* $\Rightarrow ('d, 'c, 'a)$ *interp* $\Rightarrow \text{bool}$ **where**
smaller-interp $\Delta \Delta' \longleftrightarrow (\forall s. \Delta s \subseteq \Delta' s)$

lemma *smaller-interpI*:

assumes $\bigwedge s x. x \in \Delta s \Longrightarrow x \in \Delta' s$

shows *smaller-interp* $\Delta \Delta'$

<proof>

definition *indep-interp* **where**

indep-interp $A \longleftrightarrow (\forall x s \Delta \Delta'. x, s, \Delta \models A \longleftrightarrow x, s, \Delta' \models A)$

fun *applies-eq* :: $('a, 'b, 'c, 'd)$ *assertion* $\Rightarrow ('d, 'c, 'a)$ *interp* $\Rightarrow ('d, 'c, 'a)$ *interp*
where

applies-eq $A \Delta s = \{ a \mid a, s, \Delta \models A \}$

definition *monotonic* :: $(('d, 'c, 'a)$ *interp* $\Rightarrow ('d, 'c, 'a)$ *interp*) $\Rightarrow \text{bool}$ **where**
monotonic $f \longleftrightarrow (\forall \Delta \Delta'. \text{smaller-interp } \Delta \Delta' \longrightarrow \text{smaller-interp } (f \Delta) (f \Delta'))$

lemma *monotonicI*:

assumes $\bigwedge \Delta \Delta'. \text{smaller-interp } \Delta \Delta' \Longrightarrow \text{smaller-interp } (f \Delta) (f \Delta')$

shows *monotonic* f

<proof>

definition *non-increasing* :: $(('d, 'c, 'a)$ *interp* $\Rightarrow ('d, 'c, 'a)$ *interp*) $\Rightarrow \text{bool}$ **where**
non-increasing $f \longleftrightarrow (\forall \Delta \Delta'. \text{smaller-interp } \Delta \Delta' \longrightarrow \text{smaller-interp } (f \Delta') (f \Delta))$

lemma *non-increasingI*:

assumes $\bigwedge \Delta \Delta'. \text{smaller-interp } \Delta \Delta' \Longrightarrow \text{smaller-interp } (f \Delta') (f \Delta)$

shows *non-increasing* f

<proof>

lemma *smaller-interp-refl*:

smaller-interp $\Delta \Delta$

<proof>

lemma *smaller-interp-applies-cons:*

assumes *smaller-interp* (*applies-eq* $A \Delta$) (*applies-eq* $A \Delta'$)
and $a, s, \Delta \models A$
shows $a, s, \Delta' \models A$

<proof>

definition *empty-interp where*

empty-interp $s = \{\}$

definition *full-interp :: ('d, 'c, 'a) interp where*

full-interp $s = UNIV$

lemma *smaller-interp-trans:*

assumes *smaller-interp* $a b$
and *smaller-interp* $b c$
shows *smaller-interp* $a c$

<proof>

lemma *smaller-empty:*

smaller-interp empty-interp x

<proof>

The definition of set-closure properties corresponds to Definition 8 of the paper [5].

definition *set-closure-property :: ('a \Rightarrow 'a \Rightarrow 'a set) \Rightarrow ('d, 'c, 'a) interp \Rightarrow bool*
where

set-closure-property $S \Delta \longleftrightarrow (\forall a b s. a \in \Delta s \wedge b \in \Delta s \longrightarrow S a b \subseteq \Delta s)$

lemma *set-closure-propertyI:*

assumes $\bigwedge a b s. a \in \Delta s \wedge b \in \Delta s \Longrightarrow S a b \subseteq \Delta s$
shows *set-closure-property* $S \Delta$

<proof>

lemma *set-closure-property-instantiate:*

assumes *set-closure-property* $S \Delta$
and $a \in \Delta s$
and $b \in \Delta s$
and $x \in S a b$
shows $x \in \Delta s$

<proof>

5.2 Everything preserves monotonicity

lemma *indep-implies-non-increasing:*

assumes *indep-interp* A
shows *non-increasing* (*applies-eq* A)

<proof>

5.2.1 Monotonicity

lemma *mono-instantiate*:

assumes *monotonic* (*applies-eq* *A*)

and $x \in \text{applies-eq } A \ \Delta \ s$

and *smaller-interp* $\Delta \ \Delta'$

shows $x \in \text{applies-eq } A \ \Delta' \ s$

<proof>

lemma *mono-star*:

assumes *monotonic* (*applies-eq* *A*)

and *monotonic* (*applies-eq* *B*)

shows *monotonic* (*applies-eq* (*Star* *A* *B*))

<proof>

lemma *mono-wand*:

assumes *non-increasing* (*applies-eq* *A*)

and *monotonic* (*applies-eq* *B*)

shows *monotonic* (*applies-eq* (*Wand* *A* *B*))

<proof>

lemma *mono-and*:

assumes *monotonic* (*applies-eq* *A*)

and *monotonic* (*applies-eq* *B*)

shows *monotonic* (*applies-eq* (*And* *A* *B*))

<proof>

lemma *mono-or*:

assumes *monotonic* (*applies-eq* *A*)

and *monotonic* (*applies-eq* *B*)

shows *monotonic* (*applies-eq* (*Or* *A* *B*))

<proof>

lemma *mono-sem*:

monotonic (*applies-eq* (*Sem* *B*))

<proof>

lemma *mono-interp*:

monotonic (*applies-eq* *Pred*)

<proof>

lemma *mono-mult*:

assumes *monotonic* (*applies-eq* *A*)

shows *monotonic* (*applies-eq* (*Mult* π *A*))
(*proof*)

lemma *mono-wild*:
assumes *monotonic* (*applies-eq* *A*)
shows *monotonic* (*applies-eq* (*Wildcard* *A*))
(*proof*)

lemma *mono-imp*:
assumes *non-increasing* (*applies-eq* *A*)
and *monotonic* (*applies-eq* *B*)
shows *monotonic* (*applies-eq* (*Imp* *A* *B*))
(*proof*)

lemma *mono-bounded*:
assumes *monotonic* (*applies-eq* *A*)
shows *monotonic* (*applies-eq* (*Bounded* *A*))
(*proof*)

lemma *mono-exists*:
assumes *monotonic* (*applies-eq* *A*)
shows *monotonic* (*applies-eq* (*Exists* *v* *A*))
(*proof*)

lemma *mono-forall*:
assumes *monotonic* (*applies-eq* *A*)
shows *monotonic* (*applies-eq* (*Forall* *v* *A*))
(*proof*)

5.2.2 Non-increasing

lemma *non-increasing-instantiate*:
assumes *non-increasing* (*applies-eq* *A*)
and $x \in \text{applies-eq } A \Delta' s$
and *smaller-interp* $\Delta \Delta'$
shows $x \in \text{applies-eq } A \Delta s$
(*proof*)

lemma *non-inc-star*:
assumes *non-increasing* (*applies-eq* *A*)
and *non-increasing* (*applies-eq* *B*)
shows *non-increasing* (*applies-eq* (*Star* *A* *B*))
(*proof*)

lemma *non-increasing-wand*:
assumes *monotonic* (*applies-eq* *A*)

and *non-increasing* (*applies-eq* B)
shows *non-increasing* (*applies-eq* ($\text{Wand } A \ B$))
(*proof*)

lemma *non-increasing-and*:
assumes *non-increasing* (*applies-eq* A)
and *non-increasing* (*applies-eq* B)
shows *non-increasing* (*applies-eq* ($\text{And } A \ B$))
(*proof*)

lemma *non-increasing-or*:
assumes *non-increasing* (*applies-eq* A)
and *non-increasing* (*applies-eq* B)
shows *non-increasing* (*applies-eq* ($\text{Or } A \ B$))
(*proof*)

lemma *non-increasing-sem*:
non-increasing (*applies-eq* ($\text{Sem } B$))
(*proof*)

lemma *non-increasing-mult*:
assumes *non-increasing* (*applies-eq* A)
shows *non-increasing* (*applies-eq* ($\text{Mult } \pi \ A$))
(*proof*)

lemma *non-increasing-wild*:
assumes *non-increasing* (*applies-eq* A)
shows *non-increasing* (*applies-eq* ($\text{Wildcard } A$))
(*proof*)

lemma *non-increasing-imp*:
assumes *monotonic* (*applies-eq* A)
and *non-increasing* (*applies-eq* B)
shows *non-increasing* (*applies-eq* ($\text{Imp } A \ B$))
(*proof*)

lemma *non-increasing-bounded*:
assumes *non-increasing* (*applies-eq* A)
shows *non-increasing* (*applies-eq* ($\text{Bounded } A$))
(*proof*)

lemma *non-increasing-exists*:
assumes *non-increasing* (*applies-eq* A)

shows *non-increasing* (*applies-eq* (*Exists v A*))
 ⟨*proof*⟩

lemma *non-increasing-forall*:
assumes *non-increasing* (*applies-eq A*)
shows *non-increasing* (*applies-eq* (*Forall v A*))
 ⟨*proof*⟩

5.3 Tarski's fixed points

5.3.1 Greatest Fixed Point

definition $D :: (('d, 'c, 'a) \text{interp} \Rightarrow ('d, 'c, 'a) \text{interp}) \Rightarrow ('d, 'c, 'a) \text{interp}$ set
where

$$D f = \{ \Delta \mid \Delta. \text{smaller-interp } \Delta (f \Delta) \}$$

fun $GFP :: (('d, 'c, 'a) \text{interp} \Rightarrow ('d, 'c, 'a) \text{interp}) \Rightarrow ('d, 'c, 'a) \text{interp}$ **where**
 $GFP f s = \{ \sigma \mid \sigma. \exists \Delta \in D f. \sigma \in \Delta s \}$

lemma *smaller-interp-D*:
assumes $x \in D f$
shows *smaller-interp* $x (GFP f)$
 ⟨*proof*⟩

lemma *GFP-lub*:
assumes $\bigwedge x. x \in D f \implies \text{smaller-interp } x y$
shows *smaller-interp* ($GFP f$) y
 ⟨*proof*⟩

lemma *smaller-interp-antisym*:
assumes *smaller-interp* $a b$
and *smaller-interp* $b a$
shows $a = b$
 ⟨*proof*⟩

5.3.2 Least Fixed Point

definition $DD :: (('d, 'c, 'a) \text{interp} \Rightarrow ('d, 'c, 'a) \text{interp}) \Rightarrow ('d, 'c, 'a) \text{interp}$ set
where

$$DD f = \{ \Delta \mid \Delta. \text{smaller-interp } (f \Delta) \Delta \}$$

fun $LFP :: (('d, 'c, 'a) \text{interp} \Rightarrow ('d, 'c, 'a) \text{interp}) \Rightarrow ('d, 'c, 'a) \text{interp}$ **where**
 $LFP f s = \{ \sigma \mid \sigma. \forall \Delta \in DD f. \sigma \in \Delta s \}$

lemma *smaller-interp-DD*:
assumes $x \in DD f$
shows *smaller-interp* ($LFP f$) x
 ⟨*proof*⟩

lemma *LFP-glb*:

assumes $\bigwedge x. x \in DD f \implies \text{smaller-interp } y x$
shows $\text{smaller-interp } y (LFP f)$

<proof>

5.4 Combinability and (an assertion being) intuitionistic are set-closure properties

5.4.1 Intuitionistic assertions

definition *sem-intui* :: $('d, 'c, 'a) \text{interp} \implies \text{bool}$ **where**

$\text{sem-intui } \Delta \longleftrightarrow (\forall s \sigma \sigma'. \sigma' \succeq \sigma \wedge \sigma \in \Delta s \longrightarrow \sigma' \in \Delta s)$

lemma *sem-intuiI*:

assumes $\bigwedge s \sigma \sigma'. \sigma' \succeq \sigma \wedge \sigma \in \Delta s \implies \sigma' \in \Delta s$

shows $\text{sem-intui } \Delta$

<proof>

lemma *instantiate-intui-applies*:

assumes *intuitionistic* $s \Delta A$

and $\sigma' \succeq \sigma$

and $\sigma \in \text{applies-eq } A \Delta s$

shows $\sigma' \in \text{applies-eq } A \Delta s$

<proof>

lemma *sem-intui-intuitionistic*:

$\text{sem-intui } (\text{applies-eq } A \Delta) \longleftrightarrow (\forall s. \text{intuitionistic } s \Delta A) \text{ (is } ?A \longleftrightarrow ?B)$

<proof>

lemma *intuitionistic-set-closure*:

$\text{sem-intui} = \text{set-closure-property } (\lambda a b. \{ \sigma \mid \sigma \succeq a \})$

<proof>

5.4.2 Combinable assertions

definition *sem-combinable* :: $('d, 'c, 'a) \text{interp} \implies \text{bool}$ **where**

$\text{sem-combinable } \Delta \longleftrightarrow (\forall s p q a b x. \text{sadd } p q = \text{one} \wedge a \in \Delta s \wedge b \in \Delta s \wedge \text{Some } x = p \odot a \oplus q \odot b \longrightarrow x \in \Delta s)$

lemma *sem-combinableI*:

assumes $\bigwedge s p q a b x. \text{sadd } p q = \text{one} \wedge a \in \Delta s \wedge b \in \Delta s \wedge \text{Some } x = p \odot a \oplus q \odot b \implies x \in \Delta s$

shows $\text{sem-combinable } \Delta$

<proof>

lemma *sem-combinableE*:

assumes $\text{sem-combinable } \Delta$

and $a \in \Delta s$
and $b \in \Delta s$
and *Some* $x = p \odot a \oplus q \odot b$
and *sadd* $p q = one$
shows $x \in \Delta s$
 $\langle proof \rangle$

lemma *applies-eq-equiv*:

$x \in \text{applies-eq } A \Delta s \longleftrightarrow x, s, \Delta \models A$
 $\langle proof \rangle$

lemma *sem-combinable-appliesE*:

assumes *sem-combinable* (*applies-eq* $A \Delta$)
and $a, s, \Delta \models A$
and $b, s, \Delta \models A$
and *Some* $x = p \odot a \oplus q \odot b$
and *sadd* $p q = one$
shows $x, s, \Delta \models A$
 $\langle proof \rangle$

lemma *sem-combinable-equiv*:

sem-combinable (*applies-eq* $A \Delta$) \longleftrightarrow (*combinable* ΔA) (**is** $?A \longleftrightarrow ?B$)
 $\langle proof \rangle$

lemma *combinable-set-closure*:

sem-combinable = *set-closure-property* ($\lambda a b. \{ \sigma \mid \sigma p q. \text{sadd } p q = one \wedge \text{Some } \sigma = p \odot a \oplus q \odot b \}$)
 $\langle proof \rangle$

5.5 Transfinite induction

definition *Inf* :: ($'d, 'c, 'a$) *interp set* \Rightarrow ($'d, 'c, 'a$) *interp where*

Inf $S s = \{ \sigma \mid \sigma. \forall \Delta \in S. \sigma \in \Delta s \}$

definition *Sup* :: ($'d, 'c, 'a$) *interp set* \Rightarrow ($'d, 'c, 'a$) *interp where*

Sup $S s = \{ \sigma \mid \sigma. \exists \Delta \in S. \sigma \in \Delta s \}$

definition *inf* :: ($'d, 'c, 'a$) *interp* \Rightarrow ($'d, 'c, 'a$) *interp* \Rightarrow ($'d, 'c, 'a$) *interp where*

inf $\Delta \Delta' s = \Delta s \cap \Delta' s$

definition *less where*

less $a b \longleftrightarrow \text{smaller-interp } a b \wedge a \neq b$

definition *sup* :: ($'d, 'c, 'a$) *interp* \Rightarrow ($'d, 'c, 'a$) *interp* \Rightarrow ($'d, 'c, 'a$) *interp where*

sup $\Delta \Delta' s = \Delta s \cup \Delta' s$

lemma *smaller-full*:

smaller-interp x full-interp
<proof>

lemma *inf-empty*:
local.Inf {} = full-interp
<proof>

lemma *sup-empty*:
local.Sup {} = empty-interp
<proof>

lemma *test-axiom-inf*:
assumes $\bigwedge x. x \in A \implies \text{smaller-interp } z \ x$
shows *smaller-interp z (local.Inf A)*
<proof>

lemma *test-axiom-sup*:
assumes $\bigwedge x. x \in A \implies \text{smaller-interp } x \ z$
shows *smaller-interp (local.Sup A) z*
<proof>

interpretation *complete-lattice Inf Sup inf smaller-interp less sup empty-interp*
full-interp
<proof>

lemma *mono-same*:
monotonic f \longleftrightarrow order-class.mono f
<proof>

lemma *smaller-interp a b \longleftrightarrow a \leq b*
<proof>

lemma *set-closure-property-admissible*:
ccpo.admissible Sup-class.Sup (\leq) (set-closure-property S)
<proof>

definition *supp* :: *('d, 'c, 'a) interp \Rightarrow bool* **where**
supp $\Delta \longleftrightarrow (\forall a \ b \ s. a \in \Delta \ s \wedge b \in \Delta \ s \longrightarrow (\exists x. a \succeq x \wedge b \succeq x \wedge x \in \Delta \ s))$

lemma *suppI*:
assumes $\bigwedge a \ b \ s. a \in \Delta \ s \wedge b \in \Delta \ s \implies (\exists x. a \succeq x \wedge b \succeq x \wedge x \in \Delta \ s)$
shows *supp Δ*
<proof>

lemma *supp-admissible*:
 ccpo.admissible Sup-class.Sup (\leq) *supp*
 \langle *proof* \rangle

lemma *Sup-class.Sup* $\{\}$ = *empty-interp* \langle *proof* \rangle

lemma *set-closure-prop-empty-all*:
 shows *set-closure-property S empty-interp*
 and *set-closure-property S full-interp*
 \langle *proof* \rangle

lemma *LFP-preserves-set-closure-property-aux*:
 assumes *monotonic f*
 and *set-closure-property S empty-interp*
 and $\bigwedge \Delta. \textit{set-closure-property S } \Delta \implies \textit{set-closure-property S (f } \Delta)$
 shows *set-closure-property S (ccpo-class.fixp f)*
 \langle *proof* \rangle

lemma *GFP-preserves-set-closure-property-aux*:
 assumes *order-class.mono f*
 and *set-closure-property S full-interp*
 and $\bigwedge \Delta. \textit{set-closure-property S } \Delta \implies \textit{set-closure-property S (f } \Delta)$
 shows *set-closure-property S (complete-lattice-class.gfp f)*
 \langle *proof* \rangle

5.6 Theorems

5.6.1 Greatest Fixed Point

theorem *GFP-is-FP*:
 assumes *monotonic f*
 shows $f \text{ (GFP } f) = \text{GFP } f$
 \langle *proof* \rangle

theorem *GFP-greatest*:
 assumes $f u = u$
 shows *smaller-interp u (GFP f)*
 \langle *proof* \rangle

lemma *same-GFP*:
 assumes *monotonic f*
 shows *complete-lattice-class.gfp f = GFP f*
 \langle *proof* \rangle

5.6.2 Least Fixed Point

theorem *LFP-is-FP*:
 assumes *monotonic f*

shows $f (LFP f) = LFP f$
<proof>

theorem *LFP-least*:
assumes $f u = u$
shows *smaller-interp* $(LFP f) u$
<proof>

lemma *same-LFP*:
assumes *monotonic* f
shows *complete-lattice-class.lfp* $f = LFP f$
<proof>

lemma *LFP-same*:
assumes *monotonic* f
shows *ccpo-class.fixp* $f = LFP f$
<proof>

The following theorem corresponds to Theorem 5 of the paper [5].

theorem *FP-preserves-set-closure-property*:
assumes *monotonic* f
and $\bigwedge \Delta. \text{set-closure-property } S \Delta \implies \text{set-closure-property } S (f \Delta)$
shows *set-closure-property* $S (GFP f)$
and *set-closure-property* $S (LFP f)$
<proof>

end

end

6 Properties of Magic Wands

theory *WandProperties*
imports *Distributivity*
begin

context *logic*
begin

lemma *modus-ponens*:
Star $P (Wand P Q), \Delta \vdash Q$
<proof>

lemma *transitivity*:
Star $(Wand A B) (Wand B C), \Delta \vdash Wand A C$
<proof>

lemma *currying1*:
 Wand (Star A B) C, $\Delta \vdash$ Wand A (Wand B C)
 \langle proof \rangle

lemma *currying2*:
 Wand A (Wand B C), $\Delta \vdash$ Wand (Star A B) C
 \langle proof \rangle

lemma *distribution*:
 Star (Wand A B) C, $\Delta \vdash$ Wand A (Star B C)
 \langle proof \rangle

lemma *adjunct1*:
 assumes A, $\Delta \vdash$ Wand B C
 shows Star A B, $\Delta \vdash$ C
 \langle proof \rangle

lemma *adjunct2*:
 assumes Star A B, $\Delta \vdash$ C
 shows A, $\Delta \vdash$ Wand B C
 \langle proof \rangle

end

end

7 Fractional Predicates and Magic Wands in Automatic Separation Logic Verifiers

This section corresponds to Section 5 of the paper [5].

theory *AutomaticVerifiers*
 imports *FixedPoint WandProperties*
 begin

context *logic*
 begin

7.1 Syntactic multiplication

The following definition corresponds to Figure 6 of the paper [5].

fun *syn-mult* :: 'b \Rightarrow ('a, 'b, 'c, 'd) assertion \Rightarrow ('a, 'b, 'c, 'd) assertion **where**
syn-mult π (Star A B) = Star (*syn-mult* π A) (*syn-mult* π B)
 | *syn-mult* π (Wand A B) = Wand (*syn-mult* π A) (*syn-mult* π B)
 | *syn-mult* π (Or A B) = Or (*syn-mult* π A) (*syn-mult* π B)
 | *syn-mult* π (And A B) = And (*syn-mult* π A) (*syn-mult* π B)

$| \text{syn-mult } \pi (\text{Imp } A B) = \text{Imp } (\text{syn-mult } \pi A) (\text{syn-mult } \pi B)$
 $| \text{syn-mult } \pi (\text{Mult } \alpha A) = \text{syn-mult } (\text{smult } \alpha \pi) A$
 $| \text{syn-mult } \pi (\text{Exists } x A) = \text{Exists } x (\text{syn-mult } \pi A)$
 $| \text{syn-mult } \pi (\text{Forall } x A) = \text{Forall } x (\text{syn-mult } \pi A)$
 $| \text{syn-mult } \pi (\text{Wildcard } A) = \text{Wildcard } A$
 $| \text{syn-mult } \pi A = \text{Mult } \pi A$

definition *div-state* **where**

$\text{div-state } \pi \sigma = (\text{SOME } r. \sigma = \pi \odot r)$

lemma *div-state-ok*:

$\sigma = \pi \odot (\text{div-state } \pi \sigma)$

$\langle \text{proof} \rangle$

The following theorem corresponds to Theorem 6 of the paper [5].

theorem *syn-sen-mult-same*:

$\sigma, s, \Delta \models \text{syn-mult } \pi A \longleftrightarrow \sigma, s, \Delta \models \text{Mult } \pi A$

$\langle \text{proof} \rangle$

7.2 Monotonicity and fixed point

fun *pos-neg-rec-call* :: $\text{bool} \Rightarrow ('a, 'b, 'c, 'd) \text{ assertion} \Rightarrow \text{bool}$ **where**

$\text{pos-neg-rec-call } b \text{ Pred} \longleftrightarrow b$

$| \text{pos-neg-rec-call } b (\text{Mult } - A) \longleftrightarrow \text{pos-neg-rec-call } b A$

$| \text{pos-neg-rec-call } b (\text{Exists } - A) \longleftrightarrow \text{pos-neg-rec-call } b A$

$| \text{pos-neg-rec-call } b (\text{Forall } - A) \longleftrightarrow \text{pos-neg-rec-call } b A$

$| \text{pos-neg-rec-call } b (\text{Star } A B) \longleftrightarrow \text{pos-neg-rec-call } b A \wedge \text{pos-neg-rec-call } b B$

$| \text{pos-neg-rec-call } b (\text{Or } A B) \longleftrightarrow \text{pos-neg-rec-call } b A \wedge \text{pos-neg-rec-call } b B$

$| \text{pos-neg-rec-call } b (\text{And } A B) \longleftrightarrow \text{pos-neg-rec-call } b A \wedge \text{pos-neg-rec-call } b B$

$| \text{pos-neg-rec-call } b (\text{Wand } A B) \longleftrightarrow \text{pos-neg-rec-call } (\neg b) A \wedge \text{pos-neg-rec-call } b B$

$| \text{pos-neg-rec-call } b (\text{Imp } A B) \longleftrightarrow \text{pos-neg-rec-call } (\neg b) A \wedge \text{pos-neg-rec-call } b B$

$| \text{pos-neg-rec-call } - (\text{Sem } -) \longleftrightarrow \text{True}$

$| \text{pos-neg-rec-call } b (\text{Bounded } A) \longleftrightarrow \text{pos-neg-rec-call } b A$

$| \text{pos-neg-rec-call } b (\text{Wildcard } A) \longleftrightarrow \text{pos-neg-rec-call } b A$

lemma *pos-neg-rec-call-mono*:

assumes $\text{pos-neg-rec-call } b A$

shows $(b \longrightarrow \text{monotonic } (\text{applies-eq } A)) \wedge (\neg b \longrightarrow \text{non-increasing } (\text{applies-eq } A))$

$\langle \text{proof} \rangle$

The following theorem corresponds to Theorem 7 of the paper [5].

theorem *exists-lfp-gfp*:

assumes $\text{pos-neg-rec-call } \text{True } A$

shows $\sigma, s, \text{LFP } (\text{applies-eq } A) \models A \longleftrightarrow \sigma \in \text{LFP } (\text{applies-eq } A) s$

and $\sigma, s, \text{GFP } (\text{applies-eq } A) \models A \longleftrightarrow \sigma \in \text{GFP } (\text{applies-eq } A) s$

$\langle \text{proof} \rangle$

7.3 Combinability

definition *combinable-sem* :: ($'d \Rightarrow 'c \Rightarrow 'a \Rightarrow \text{bool} \Rightarrow \text{bool}$) $\Rightarrow \text{bool}$ **where**
combinable-sem $B \longleftrightarrow (\forall a b x s \alpha \beta. B s a \wedge B s b \wedge \text{sadd } \alpha \beta = \text{one} \wedge \text{Some } x = \alpha \odot a \oplus \beta \odot b \longrightarrow B s x)$

fun *wf-assertion* :: ($'a, 'b, 'c, 'd$) *assertion* $\Rightarrow \text{bool}$ **where**
wf-assertion $\text{Pred} \longleftrightarrow \text{True}$
wf-assertion $(\text{Sem } B) \longleftrightarrow \text{combinable-sem } B$
wf-assertion $(\text{Mult } - A) \longleftrightarrow \text{wf-assertion } A$
wf-assertion $(\text{Forall } - A) \longleftrightarrow \text{wf-assertion } A$
wf-assertion $(\text{Exists } x A) \longleftrightarrow \text{wf-assertion } A \wedge (\forall \Delta. \text{unambiguous } \Delta A x)$
wf-assertion $(\text{Star } A B) \longleftrightarrow \text{wf-assertion } A \wedge \text{wf-assertion } B$
wf-assertion $(\text{And } A B) \longleftrightarrow \text{wf-assertion } A \wedge \text{wf-assertion } B$
wf-assertion $(\text{Wand } A B) \longleftrightarrow \text{wf-assertion } B$
wf-assertion $(\text{Imp } A B) \longleftrightarrow \text{pure } A \wedge \text{wf-assertion } B$
wf-assertion $(\text{Wildcard } A) \longleftrightarrow \text{wf-assertion } A$
wf-assertion $- \longleftrightarrow \text{False}$

lemma *wf-implies-combinable*:
assumes *wf-assertion* A
and *sem-combinable* Δ
shows *combinable* ΔA
 $\langle \text{proof} \rangle$

7.4 Theorems

The following two theorems correspond to the rules shown in Section 5.1 of the paper [5].

theorem *apply-wand*:
 $\text{Star } (\text{syn-mult } \pi A) (\text{Mult } \pi (\text{Wand } A B)), \Delta \vdash \text{syn-mult } \pi B$
 $\langle \text{proof} \rangle$

theorem *package-wand*:
assumes $\text{Star } F (\text{syn-mult } \pi A), \Delta \vdash \text{syn-mult } \pi B$
shows $F, \Delta \vdash \text{Mult } \pi (\text{Wand } A B)$
 $\langle \text{proof} \rangle$

The following four theorems correspond to the rules shown in Section 5.2 of the paper [5].

theorem *fold-lfp*:
assumes *pos-neg-rec-call* $\text{True } A$
shows $\text{syn-mult } \pi A, \text{LFP } (\text{applies-eq } A) \vdash \text{Mult } \pi \text{Pred}$
 $\langle \text{proof} \rangle$

theorem *unfold-lfp*:
assumes *pos-neg-rec-call* $\text{True } A$

shows $Mult\ \pi\ Pred, LFP\ (applies\text{-}eq\ A) \vdash syn\text{-}mult\ \pi\ A$
 $\langle proof \rangle$

theorem *fold-gfp*:

assumes $pos\text{-}neg\text{-}rec\text{-}call\ True\ A$
shows $syn\text{-}mult\ \pi\ A, GFP\ (applies\text{-}eq\ A) \vdash Mult\ \pi\ Pred$
 $\langle proof \rangle$

theorem *unfold-gfp*:

assumes $pos\text{-}neg\text{-}rec\text{-}call\ True\ A$
shows $Mult\ \pi\ Pred, GFP\ (applies\text{-}eq\ A) \vdash syn\text{-}mult\ \pi\ A$
 $\langle proof \rangle$

The following theorems correspond to the rule shown in Section 5.3 of the paper [5].

theorem *wf-assertion-combinable-lfp*:

assumes $wf\text{-}assertion\ A$
and $pos\text{-}neg\text{-}rec\text{-}call\ True\ A$
shows $sem\text{-}combinable\ (LFP\ (applies\text{-}eq\ A))$
 $\langle proof \rangle$

theorem *wf-assertion-combinable-gfp*:

assumes $wf\text{-}assertion\ A$
and $pos\text{-}neg\text{-}rec\text{-}call\ True\ A$
shows $sem\text{-}combinable\ (GFP\ (applies\text{-}eq\ A))$
 $\langle proof \rangle$

theorem *wf-combine*:

assumes $wf\text{-}assertion\ A$
and $pos\text{-}neg\text{-}rec\text{-}call\ True\ A$
shows $Star\ (Mult\ \alpha\ Pred)\ (Mult\ \beta\ Pred), LFP\ (applies\text{-}eq\ A) \vdash Mult\ (sadd\ \alpha\ \beta)\ Pred$
and $Star\ (Mult\ \alpha\ Pred)\ (Mult\ \beta\ Pred), GFP\ (applies\text{-}eq\ A) \vdash Mult\ (sadd\ \alpha\ \beta)\ Pred$
 $\langle proof \rangle$

end

end

References

- [1] S. Blom, S. Darabi, M. Huisman, and W. Oortwijn. The VerCors tool set: Verification of parallel and concurrent software. In N. Polikarpova and S. Schneider, editors, *Integrated Formal Methods*, pages 102–110, Cham, 2017. Springer International Publishing.

- [2] R. Bornat, C. Calcagno, P. W. O’Hearn, and M. J. Parkinson. Permission accounting in separation logic. In J. Palsberg and M. Abadi, editors, *Principle of Programming Languages (POPL)*, pages 259–270. ACM, 2005.
- [3] J. Boyland. Checking interference with fractional permissions. In R. Cousot, editor, *Static Analysis (SAS)*, pages 55–72, 2003.
- [4] J. Brotherston, D. Costa, A. Hobor, and J. Wickerson. Reasoning over permissions regions in concurrent separation logic. In S. K. Lahiri and C. Wang, editors, *Computer Aided Verification (CAV)*, 2020.
- [5] T. Dardinier, P. Müller, and A. J. Summers. Fractional resources in unbounded separation logic. *Proc. ACM Program. Lang.*, 6(OOPSLA2), oct 2022.
- [6] T. Dardinier, G. Parthasarathy, N. Weeks, P. Müller, and A. J. Summers. Sound automation of magic wands. In S. Shoham and Y. Vizek, editors, *Computer Aided Verification*, pages 130–151, Cham, 2022. Springer International Publishing.
- [7] B. Jacobs, J. Smans, P. Philippaerts, F. Vogels, W. Penninckx, and F. Piessens. VeriFast: A powerful, sound, predictable, fast verifier for C and Java. In M. G. Bobaru, K. Havelund, G. J. Holzmann, and R. Joshi, editors, *NASA Formal Methods (NFM)*, volume 6617 of *Lecture Notes in Computer Science*, pages 41–55. Springer, 2011.
- [8] X.-B. Le and A. Hobor. Logical reasoning for disjoint permissions. In A. Ahmed, editor, *European Symposium on Programming (ESOP)*, 2018.
- [9] K. R. M. Leino, P. Müller, and J. Smans. Verification of concurrent programs with Chalice. In *Foundations of Security Analysis and Design V*, volume 5705 of *Lecture Notes in Computer Science*, pages 195–222. Springer, 2009.
- [10] P. Müller, M. Schwerhoff, and A. J. Summers. Viper: A verification infrastructure for permission-based reasoning. In B. Jobstmann and K. R. M. Leino, editors, *Verification, Model Checking, and Abstract Interpretation (VMCAI)*, volume 9583, pages 41–62. Springer, 2016.
- [11] J. C. Reynolds. Separation Logic: A Logic for Shared Mutable Data Structures. In *Logic in Computer Science (LICS)*, pages 55–74. IEEE, 2002.