Unbounded Separation Logic

Thibault Dardinier
Department of Computer Science, ETH Zurich, Switzerland

October 13, 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

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** ‹⊕› *63*)

begin

definition *compatible* :: $'a \Rightarrow 'a \Rightarrow$ *bool* (**infixl** ‹##› *60*) **where**
$a \ \#\# \ b \longleftrightarrow a \oplus b \neq$ *None*

definition *larger* :: $'a \Rightarrow 'a \Rightarrow$ *bool* (**infixl** ‹⪰› *55*) **where**
$a \succeq b \longleftrightarrow (\exists c.$ *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** ‹⊙› *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* :: $'a \Rightarrow bool$

assumes *commutative*: $a \oplus b = b \oplus a$
and *asso1*: $a \oplus b = Some\ ab \wedge b \oplus c = Some\ bc \Longrightarrow ab \oplus c = a \oplus bc$
and *asso2*: $a \oplus b = Some\ ab \wedge \neg\ b\ \#\#\ c \Longrightarrow \neg\ ab\ \#\#\ c$

and *sinv-inverse*: $smult\ p\ (sinv\ p) = one$
and *sone-neutral*: $smult\ one\ p = p$
and *sadd-comm*: $sadd\ p\ q = sadd\ q\ p$
and *smult-comm*: $smult\ p\ q = smult\ q\ p$
and *smult-distrib*: $smult\ p\ (sadd\ q\ r) = sadd\ (smult\ p\ q)\ (smult\ p\ r)$
and *smult-asso*: $smult\ (smult\ p\ q)\ r = smult\ p\ (smult\ q\ r)$

and *double-mult*: $p \odot (q \odot a) = (smult\ p\ q) \odot a$
and *plus-mult*: $Some\ a = b \oplus c \Longrightarrow Some\ (p \odot a) = (p \odot b) \oplus (p \odot c)$
and *distrib-mult*: $Some\ ((sadd\ p\ q) \odot x) = p \odot x \oplus q \odot x$
and *one-neutral*: $one \odot a = a$

and *valid-mono*: $valid\ a \wedge a \succeq b \Longrightarrow valid\ b$

begin

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

fun *sat* :: $'a \Rightarrow ('d \Rightarrow 'c) \Rightarrow ('d, 'c, 'a)\ interp \Rightarrow ('a, 'b, 'c, 'd)\ assertion \Rightarrow bool$
$(\langle\text{-}, \text{-}, \text{-} \models \text{-}\rangle\ [51, 65, 68, 66]\ 50)$ **where**
$\sigma, s, \Delta \models Mult\ p\ A \longleftrightarrow (\exists a.\ \sigma = p \odot a \wedge a, s, \Delta \models A)$
$|\ \sigma, s, \Delta \models Star\ A\ B \longleftrightarrow (\exists a\ b.\ Some\ \sigma = a \oplus b \wedge a, s, \Delta \models A \wedge b, s, \Delta \models B)$
$|\ \sigma, s, \Delta \models Wand\ A\ B \longleftrightarrow (\forall a\ \sigma'.\ a, s, \Delta \models A \wedge Some\ \sigma' = \sigma \oplus a \longrightarrow \sigma', s, \Delta \models B)$

$|\ \sigma, s, \Delta \models Sem\ b \longleftrightarrow b\ s\ \sigma$
$|\ \sigma, s, \Delta \models Imp\ A\ B \longleftrightarrow (\sigma, s, \Delta \models A \longrightarrow \sigma, s, \Delta \models B)$
$|\ \sigma, s, \Delta \models Or\ A\ B \longleftrightarrow (\sigma, s, \Delta \models A \vee \sigma, s, \Delta \models B)$
$|\ \sigma, s, \Delta \models And\ A\ B \longleftrightarrow (\sigma, s, \Delta \models A \wedge \sigma, s, \Delta \models B)$

$|\ \sigma, s, \Delta \models Exists\ x\ A \longleftrightarrow (\exists v.\ \sigma, s(x := v), \Delta \models A)$
$|\ \sigma, s, \Delta \models Forall\ x\ A \longleftrightarrow (\forall v.\ \sigma, s(x := v), \Delta \models A)$

$|\ \sigma, s, \Delta \models Pred \longleftrightarrow (\sigma \in \Delta\ s)$
$|\ \sigma, s, \Delta \models Bounded\ A \longleftrightarrow (valid\ \sigma \longrightarrow \sigma, s, \Delta \models A)$
$|\ \sigma, s, \Delta \models Wildcard\ A \longleftrightarrow (\exists a\ p.\ \sigma = p \odot a \wedge a, s, \Delta \models A)$

definition *intuitionistic* :: $('d \Rightarrow 'c) \Rightarrow ('d, 'c, 'a)\ interp \Rightarrow ('a, 'b, 'c, 'd)\ assertion \Rightarrow bool$ **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* (‹-, - ⊢ -› [*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* (‹-, - ≡ -› [*63*, *66*, *68*] *52*) **where**
$A, \Delta \equiv B \longleftrightarrow (A, \Delta \vdash B \land B, \Delta \vdash A)$

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 $\sigma, s, \Delta \models$ *Forall* $x\ A$
⟨*proof*⟩

lemma *intuitionisticI*:
assumes $\bigwedge a\ b.\ a \succeq b \land b, s, \Delta \models A \Longrightarrow a, s, \Delta \models A$
shows *intuitionistic* $s\ \Delta\ A$
⟨*proof*⟩

lemma *can-divide*:
assumes $p \odot a = p \odot b$
shows $a = b$
⟨*proof*⟩

lemma *unique-inv*:
$a = p \odot b \longleftrightarrow b = (sinv\ p) \odot a$
⟨*proof*⟩

lemma *entailsI*:
assumes $\bigwedge \sigma\ s.\ \sigma, s, \Delta \models A \Longrightarrow \sigma, s, \Delta \models B$
shows $A, \Delta \vdash B$
⟨*proof*⟩

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 $A, \Delta \equiv B$
⟨*proof*⟩

lemma *compatible-imp*:
assumes $a\ \#\#\ b$
shows $(p \odot a)\ \#\#\ (p \odot b)$
⟨*proof*⟩

lemma *compatible-iff*:
$a \ \#\# \ b \longleftrightarrow (p \odot a) \ \#\# \ (p \odot b)$
⟨*proof*⟩

lemma *sat-wand*:
assumes $\bigwedge a\ \sigma'.\ a,\ s,\ \Delta \models A \wedge Some\ \sigma' = \sigma \oplus a \Longrightarrow \sigma',\ s,\ \Delta \models B$
shows $\sigma,\ s,\ \Delta \models Wand\ A\ B$
⟨*proof*⟩

lemma *sat-imp*:
assumes $\sigma,\ s,\ \Delta \models A \Longrightarrow \sigma,\ s,\ \Delta \models B$
shows $\sigma,\ s,\ \Delta \models Imp\ A\ B$
⟨*proof*⟩

lemma *sat-mult*:
assumes $\bigwedge a.\ \sigma = p \odot a \Longrightarrow a,\ s,\ \Delta \models A$
shows $\sigma,\ s,\ \Delta \models Mult\ p\ A$
⟨*proof*⟩

lemma *larger-same*:
$a \succeq b \longleftrightarrow p \odot a \succeq p \odot b$
⟨*proof*⟩

lemma *asso3*:
assumes $\neg\ a \ \#\# \ b$
and $b \oplus c = Some\ bc$
shows $\neg\ a \ \#\# \ bc$
⟨*proof*⟩

lemma *compatible-smaller*:
assumes $a \succeq b$
and $x \ \#\# \ a$
shows $x \ \#\# \ b$
⟨*proof*⟩

lemma *compatible-multiples*:
assumes $p \odot a \ \#\# \ q \odot b$
shows $a \ \#\# \ b$
⟨*proof*⟩

lemma *move-sum*:
assumes $Some\ a = a1 \oplus a2$
and $Some\ b = b1 \oplus b2$
and $Some\ x = a \oplus b$
and $Some\ x1 = a1 \oplus b1$
and $Some\ x2 = a2 \oplus b2$
shows $Some\ x = x1 \oplus x2$
⟨*proof*⟩

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 $\exists a'.$ *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* :: $('a \times ('d \Rightarrow 'c))$ *command* $\Rightarrow ('a \times ('d \Rightarrow 'c)) \Rightarrow$ *bool* **where**
  safe $c\ \sigma \longleftrightarrow (\sigma,$ *None*$) \notin c$

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

definition *frame-property* :: $('a \times ('d \Rightarrow 'c))$ *command* $\Rightarrow$ *bool* **where**
  frame-property $c \longleftrightarrow (\forall \sigma\ \sigma 0\ r\ \sigma'\ s\ s'.$ *valid* $\sigma \wedge$ *valid* $\sigma' \wedge$ *safe* $c\ (\sigma 0, s) \wedge$ *Some* $\sigma = \sigma 0 \oplus r \wedge ((\sigma, s),$ *Some* $(\sigma', s')) \in c$
  $\longrightarrow (\exists \sigma 0'.$ *Some* $\sigma' = \sigma 0' \oplus r \wedge ((\sigma 0, s),$ *Some* $(\sigma 0', s')) \in c))$

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

lemma *valid-hoare-tripleI*:
  assumes $\bigwedge \sigma\ s.$ *valid* $\sigma \wedge \sigma, s, \Delta \models P \Longrightarrow$ *safe* $c\ (\sigma, s)$
    and $\bigwedge \sigma\ s\ \sigma'\ s'.$ *valid* $\sigma \wedge \sigma, s, \Delta \models P \Longrightarrow ((\sigma, s),$ *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* :: $('a \times ('d \Rightarrow 'c))$ *command* $\Rightarrow$ *bool* **where**
  valid-command $c \longleftrightarrow (\forall a\ b\ sa\ sb.\ ((a, sa),$ *Some* $(b, sb)) \in c \wedge$ *valid* $a \longrightarrow$ *valid*

b)

definition *modified* :: $('a \times ('d \Rightarrow 'c))$ *command* $\Rightarrow$ *'d set* **where**
modified c = $\{ x \mid x.\ \exists\sigma\ s\ \sigma'\ s'.\ ((\sigma, s), \mathit{Some}\ (\sigma', s')) \in c \wedge s\ x \neq s'\ x \}$

definition *equal-outside* :: $('d \Rightarrow 'c) \Rightarrow ('d \Rightarrow 'c) \Rightarrow$ *'d set* $\Rightarrow$ *bool* **where**
equal-outside s s' S $\longleftrightarrow (\forall x.\ x \notin S \longrightarrow s\ x = s'\ x)$

definition *not-in-fv* :: $('a, 'b, 'c, 'd)$ *assertion* $\Rightarrow$ *'d set* $\Rightarrow$ *bool* **where**
not-in-fv A S $\longleftrightarrow (\forall\sigma\ s\ \Delta\ s'.$ *equal-outside s s' S* $\longrightarrow (\sigma, s, \Delta \models A \longleftrightarrow \sigma, s', \Delta \models A))$

lemma *not-in-fv-mod*:
assumes *not-in-fv A* (*modified c*)
and $((\sigma, s), \mathit{Some}\ (\sigma', s')) \in c$
shows $x, s, \Delta \models A \longleftrightarrow x, s', \Delta \models A$
⟨*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 $\Delta \longleftrightarrow$ *valid-hoare-triple* (*Bounded P*) *c Q* Δ
⟨*proof*⟩

lemma *hoare-triple-output*:
assumes *valid-command c*
shows *valid-hoare-triple P c Q* $\Delta \longleftrightarrow$ *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*:
  $A, \Delta \vdash B \Longrightarrow (Wildcard\ A, \Delta \vdash Wildcard\ B)$
  ⟨*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, $\Delta \vdash$ *Mult one A*
⟨*proof*⟩

lemma *DotFull*:
Mult one A, $\Delta \equiv$ *A*
⟨*proof*⟩

3.10 DotExists

lemma *dot-exists1*:
Mult p (*Exists x A*), $\Delta \vdash$ *Exists x* (*Mult p A*)
⟨*proof*⟩

lemma *dot-exists2*:
Exists x (*Mult p A*), $\Delta \vdash$ *Mult p* (*Exists x A*)
⟨*proof*⟩

lemma *DotExists*:
Mult p (*Exists x A*), $\Delta \equiv$ *Exists x* (*Mult p A*)
⟨*proof*⟩

lemma *WildExists*:
Wildcard (*Exists x A*), $\Delta \equiv$ *Exists x* (*Wildcard A*)
⟨*proof*⟩

3.11 DotForall

lemma *dot-forall1*:
Mult p (*Forall x A*), $\Delta \vdash$ *Forall x* (*Mult p A*)
⟨*proof*⟩

lemma *dot-forall2*:
Forall x (*Mult p A*), $\Delta \vdash$ *Mult p* (*Forall x A*)
⟨*proof*⟩

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

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

3.12 Split

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

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
  $combinable\ \Delta\ A \longleftrightarrow (\forall p\ q.\ Star\ (Mult\ p\ A)\ (Mult\ q\ A),\ \Delta \vdash Mult\ (sadd\ p\ q)\ A)$

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 $x,\ s,\ \Delta \models Mult\ (sadd\ p\ q)\ A$
  ⟨*proof*⟩

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 $sadd\ p\ q = one$
shows $x,\ s,\ \Delta \models A$
$\langle proof \rangle$

lemma *combinableI-old*:
assumes $\bigwedge a\ b\ p\ q\ x\ \sigma\ s.\ a,\ s,\ \Delta \models A \wedge b,\ s,\ \Delta \models A \wedge Some\ \sigma = p \odot a \oplus q \odot b \wedge \sigma = (sadd\ p\ q) \odot x \Longrightarrow x,\ s,\ \Delta \models A$
shows *combinable* Δ A
$\langle proof \rangle$

lemma *combinableI*:
assumes $\bigwedge a\ b\ p\ q\ x\ \sigma\ s.\ a,\ s,\ \Delta \models A \wedge b,\ s,\ \Delta \models A \wedge Some\ x = p \odot a \oplus q \odot b \wedge sadd\ p\ q = one \Longrightarrow x,\ s,\ \Delta \models A$
shows *combinable* Δ A
$\langle proof \rangle$

lemma *combinable-wand*:
assumes *combinable* Δ B
shows *combinable* Δ (*Wand* A B)
$\langle proof \rangle$

lemma *combinable-star*:
assumes *combinable* Δ A
and *combinable* Δ B
shows *combinable* Δ (*Star* A B)
$\langle proof \rangle$

lemma *combinable-mult*:
assumes *combinable* Δ A
shows *combinable* Δ (*Mult* π A)
$\langle proof \rangle$

lemma *combinable-and*:
assumes *combinable* Δ A
and *combinable* Δ B
shows *combinable* Δ (*And* A B)
$\langle proof \rangle$

lemma *combinable-forall*:
assumes *combinable* Δ A
shows *combinable* Δ (*Forall* x A)
$\langle proof \rangle$

definition *unambiguous* **where**
unambiguous Δ *A x* $\longleftrightarrow$ ($\forall \sigma 1\ \sigma 2\ v1\ v2\ s.\ \sigma 1\ \#\#\ \sigma 2 \wedge \sigma 1,\ s(x := v1),\ \Delta \models A \wedge \sigma 2,\ s(x := v2),\ \Delta \models A \longrightarrow v1 = v2$)

lemma *unambiguousI*:
assumes $\bigwedge \sigma 1\ \sigma 2\ v1\ v2\ s.\ \sigma 1\ \#\#\ \sigma 2 \wedge \sigma 1,\ s(x := v1),\ \Delta \models A \wedge \sigma 2,\ s(x := v2),\ \Delta \models A \Longrightarrow v1 = v2$
shows *unambiguous* Δ *A x*
⟨*proof*⟩

lemma *unambiguous-star*:
assumes *unambiguous* Δ *A x*
shows *unambiguous* Δ (*Star A B*) *x*
⟨*proof*⟩

lemma *combinable-exists*:
assumes *combinable* Δ *A*
and *unambiguous* Δ *A x*
shows *combinable* Δ (*Exists x A*)
⟨*proof*⟩

lemma *combinable-pure*:
assumes *pure A*
shows *combinable* Δ *A*
⟨*proof*⟩

lemma *combinable-imp*:
assumes *pure A*
and *combinable* Δ *B*
shows *combinable* Δ (*Imp A B*)
⟨*proof*⟩

lemma *combinable-wildcard*:
assumes *combinable* Δ *A*
shows *combinable* Δ (*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* $=$ *nat* $\Rightarrow$ $('d, 'c, 'a)$ *interp*

context *logic*
begin

5.1 Definitions

definition *smaller-interp* :: $('d, 'c, 'a)$ *interp* $\Rightarrow$ $('d, 'c, 'a)$ *interp* $\Rightarrow$ *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* $\Delta\ \Delta'$
  $\langle proof \rangle$

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 $A\ \Delta\ s = \{\ a\ |a.\ a,\ s,\ \Delta \models A\ \}$

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

lemma *monotonicI*:
  assumes $\bigwedge \Delta\ \Delta'.$ *smaller-interp* $\Delta\ \Delta' \Longrightarrow$ *smaller-interp* $(f\ \Delta)\ (f\ \Delta')$
  shows *monotonic* f
  $\langle proof \rangle$

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

lemma *non-increasingI*:
  assumes $\bigwedge \Delta\ \Delta'.$ *smaller-interp* $\Delta\ \Delta' \Longrightarrow$ *smaller-interp* $(f\ \Delta')\ (f\ \Delta)$
  shows *non-increasing* f
  $\langle proof \rangle$

lemma *smaller-interp-refl*:
  smaller-interp $\Delta\ \Delta$

$\langle proof \rangle$

lemma *smaller-interp-applies-cons*:
assumes *smaller-interp* (*applies-eq* A Δ) (*applies-eq* A Δ')
and a, s, $\Delta \models A$
shows a, s, $\Delta' \models A$
$\langle proof \rangle$

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
$\langle proof \rangle$

lemma *smaller-empty*:
smaller-interp empty-interp x
$\langle proof \rangle$

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 Δ
$\langle proof \rangle$

lemma *set-closure-property-instantiate*:
assumes *set-closure-property* S Δ
and $a \in \Delta\ s$
and $b \in \Delta\ s$
and $x \in S\ a\ b$
shows $x \in \Delta\ s$
$\langle proof \rangle$

5.2 Everything preserves monotonicity

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

$\langle proof \rangle$

5.2.1 Monotonicity

lemma *mono-instantiate*:
  assumes *monotonic* (*applies-eq A*)
    and $x \in$ *applies-eq A* Δ *s*
    and *smaller-interp* Δ Δ'
  shows $x \in$ *applies-eq A* Δ' *s*
  $\langle proof \rangle$

lemma *mono-star*:
  assumes *monotonic* (*applies-eq A*)
    and *monotonic* (*applies-eq B*)
  shows *monotonic* (*applies-eq* (*Star A B*))
$\langle proof \rangle$

lemma *mono-wand*:
  assumes *non-increasing* (*applies-eq A*)
    and *monotonic* (*applies-eq B*)
  shows *monotonic* (*applies-eq* (*Wand A B*))
$\langle proof \rangle$

lemma *mono-and*:
  assumes *monotonic* (*applies-eq A*)
    and *monotonic* (*applies-eq B*)
  shows *monotonic* (*applies-eq* (*And A B*))
$\langle proof \rangle$

lemma *mono-or*:
  assumes *monotonic* (*applies-eq A*)
    and *monotonic* (*applies-eq B*)
  shows *monotonic* (*applies-eq* (*Or A B*))
$\langle proof \rangle$

lemma *mono-sem*:
  monotonic (*applies-eq* (*Sem B*))
  $\langle proof \rangle$

lemma *mono-interp*:
  monotonic (*applies-eq Pred*)
$\langle proof \rangle$

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 $x \in$ *applies-eq A* Δ' *s*
and *smaller-interp* Δ Δ'
shows $x \in$ *applies-eq A* Δ *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* (*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* (*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* (*Or* *A* *B*))
⟨*proof*⟩

lemma *non-increasing-sem*:
non-increasing (*applies-eq* (*Sem* *B*))
⟨*proof*⟩

lemma *non-increasing-mult*:
assumes *non-increasing* (*applies-eq* *A*)
shows *non-increasing* (*applies-eq* (*Mult* π *A*))
⟨*proof*⟩

lemma *non-increasing-wild*:
assumes *non-increasing* (*applies-eq* *A*)
shows *non-increasing* (*applies-eq* (*Wildcard* *A*))
⟨*proof*⟩

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

lemma *non-increasing-bounded*:
assumes *non-increasing* (*applies-eq* *A*)
shows *non-increasing* (*applies-eq* (*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) *interp* ⇒ ('d, 'c, 'a) *interp*) ⇒ ('d, 'c, 'a) *interp set*
where
D f = { Δ |Δ. *smaller-interp* Δ (*f* Δ) }

fun *GFP* :: (('d, 'c, 'a) *interp* ⇒ ('d, 'c, 'a) *interp*) ⇒ ('d, 'c, 'a) *interp* **where**
GFP f s = { σ |σ. ∃Δ ∈ *D f*. σ ∈ Δ *s* }

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

lemma *GFP-lub*:
assumes ⋀*x*. *x* ∈ *D f* ⟹ *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) *interp* ⇒ ('d, 'c, 'a) *interp*) ⇒ ('d, 'c, 'a) *interp set*
where
DD f = { Δ |Δ. *smaller-interp* (*f* Δ) Δ }

fun *LFP* :: (('d, 'c, 'a) *interp* ⇒ ('d, 'c, 'a) *interp*) ⇒ ('d, 'c, 'a) *interp* **where**
LFP f s = { σ |σ. ∀Δ ∈ *DD f*. σ ∈ Δ *s* }

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

lemma *LFP-glb*:
  assumes $\bigwedge x.\ x \in DD\ f \Longrightarrow$ *smaller-interp* $y\ x$
  shows *smaller-interp* y (*LFP* f)
$\langle proof \rangle$

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

5.4.1 Intuitionistic assertions

definition *sem-intui* :: $('d, 'c, 'a)$ *interp* $\Rightarrow$ *bool* **where**
  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 $\bigwedge s\ \sigma\ \sigma'.\ \sigma' \succeq \sigma \wedge \sigma \in \Delta\ s \Longrightarrow \sigma' \in \Delta\ s$
  shows *sem-intui* Δ
  $\langle proof \rangle$

lemma *instantiate-intui-applies*:
  assumes *intuitionistic* $s\ \Delta\ A$
      and $\sigma' \succeq \sigma$
      and $\sigma \in$ *applies-eq* $A\ \Delta\ s$
    shows $\sigma' \in$ *applies-eq* $A\ \Delta\ s$
  $\langle proof \rangle$

lemma *sem-intui-intuitionistic*:
  sem-intui (*applies-eq* $A\ \Delta$) $\longleftrightarrow$ ($\forall s.$ *intuitionistic* $s\ \Delta\ A$) (**is** *?A* $\longleftrightarrow$ *?B*)
$\langle proof \rangle$

lemma *intuitionistic-set-closure*:
  sem-intui = *set-closure-property* $(\lambda a\ b.\ \{\ \sigma\ |\sigma.\ \sigma \succeq a\})$
$\langle proof \rangle$

5.4.2 Combinable assertions

definition *sem-combinable* :: $('d, 'c, 'a)$ *interp* $\Rightarrow$ *bool* **where**
  sem-combinable $\Delta \longleftrightarrow (\forall s\ p\ q\ a\ b\ x.$ *sadd* $p\ q =$ *one* $\wedge\ a \in \Delta\ s \wedge b \in \Delta\ s \wedge$ *Some* $x = p \odot a \oplus q \odot b \longrightarrow x \in \Delta\ s)$

lemma *sem-combinableI*:
  assumes $\bigwedge s\ p\ q\ a\ b\ x.$ *sadd* $p\ q =$ *one* $\wedge\ a \in \Delta\ s \wedge b \in \Delta\ s \wedge$ *Some* $x = p \odot a \oplus q \odot b \Longrightarrow x \in \Delta\ s$
  shows *sem-combinable* Δ
  $\langle proof \rangle$

lemma *sem-combinableE*:
  assumes *sem-combinable* Δ

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 $x \in \Delta\ s$
⟨*proof*⟩

lemma *applies-eq-equiv*:
$x \in applies\text{-}eq\ A\ \Delta\ s \longleftrightarrow x,\ s,\ \Delta \models A$
⟨*proof*⟩

lemma *sem-combinable-appliesE*:
assumes $sem\text{-}combinable\ (applies\text{-}eq\ A\ \Delta)$
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 $x,\ s,\ \Delta \models A$
⟨*proof*⟩

lemma *sem-combinable-equiv*:
$sem\text{-}combinable\ (applies\text{-}eq\ A\ \Delta) \longleftrightarrow (combinable\ \Delta\ A)$ (**is** $?A \longleftrightarrow ?B$)
⟨*proof*⟩

lemma *combinable-set-closure*:
$sem\text{-}combinable = set\text{-}closure\text{-}property\ (\lambda a\ b.\ \{\ \sigma\ |\sigma\ p\ q.\ sadd\ p\ q = one \wedge Some\ \sigma = p \odot a \oplus q \odot b\})$
⟨*proof*⟩

5.5 Transfinite induction

definition $Inf :: ('d,\ 'c,\ 'a)\ interp\ set \Rightarrow ('d,\ 'c,\ 'a)\ interp$ **where**
$Inf\ S\ s = \{\ \sigma\ |\sigma.\ \forall \Delta \in S.\ \sigma \in \Delta\ s\}$

definition $Sup :: ('d,\ 'c,\ 'a)\ interp\ set \Rightarrow ('d,\ 'c,\ 'a)\ interp$ **where**
$Sup\ S\ s = \{\ \sigma\ |\sigma.\ \exists \Delta \in S.\ \sigma \in \Delta\ s\}$

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

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*:
$local.Inf\ \{\} = full\text{-}interp$
⟨*proof*⟩

lemma *sup-empty*:
$local.Sup\ \{\} = empty\text{-}interp$
⟨*proof*⟩

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

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

interpretation *complete-lattice Inf Sup inf smaller-interp less sup empty-interp full-interp*
⟨*proof*⟩

lemma *mono-same*:
$monotonic\ f \longleftrightarrow order\text{-}class.mono\ f$
⟨*proof*⟩

lemma $smaller\text{-}interp\ a\ b \longleftrightarrow a \leq b$
⟨*proof*⟩

lemma *set-closure-property-admissible*:
$ccpo.admissible\ Sup\text{-}class.Sup\ (\leq)\ (set\text{-}closure\text{-}property\ S)$
⟨*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 $\bigwedge a\ b\ s.\ a \in \Delta\ s \wedge b \in \Delta\ s \Longrightarrow (\exists x.\ a \succeq x \wedge b \succeq x \wedge x \in \Delta\ s)$
shows $supp\ \Delta$
⟨*proof*⟩

lemma *supp-admissible*:
ccpo.admissible Sup-class.Sup ($\leq$) *supp*
⟨*proof*⟩

lemma *Sup-class.Sup* {} = *empty-interp* ⟨*proof*⟩

lemma *set-closure-prop-empty-all*:
shows *set-closure-property S empty-interp*
and *set-closure-property S full-interp*
⟨*proof*⟩

lemma *LFP-preserves-set-closure-property-aux*:
assumes *monotonic f*
and *set-closure-property S empty-interp*
and $\bigwedge\Delta$. *set-closure-property S* $\Delta \Longrightarrow$ *set-closure-property S* (*f* Δ)
shows *set-closure-property S* (*ccpo-class.fixp f*)
⟨*proof*⟩

lemma *GFP-preserves-set-closure-property-aux*:
assumes *order-class.mono f*
and *set-closure-property S full-interp*
and $\bigwedge\Delta$. *set-closure-property S* $\Delta \Longrightarrow$ *set-closure-property S* (*f* Δ)
shows *set-closure-property S* (*complete-lattice-class.gfp f*)
⟨*proof*⟩

5.6 Theorems

5.6.1 Greatest Fixed Point

theorem *GFP-is-FP*:
assumes *monotonic f*
shows *f* (*GFP f*) = *GFP f*
⟨*proof*⟩

theorem *GFP-greatest*:
assumes *f u* = *u*
shows *smaller-interp u* (*GFP f*)
⟨*proof*⟩

lemma *same-GFP*:
assumes *monotonic f*
shows *complete-lattice-class.gfp f* = *GFP f*
⟨*proof*⟩

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 $\bigwedge\Delta.$ *set-closure-property* $S\ \Delta \Longrightarrow$ *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*:
  $Wand\ (Star\ A\ B)\ C,\ \Delta \vdash Wand\ A\ (Wand\ B\ C)$
⟨*proof*⟩

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

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

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

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

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**
  $syn\text{-}mult\ \pi\ (Star\ A\ B) = Star\ (syn\text{-}mult\ \pi\ A)\ (syn\text{-}mult\ \pi\ B)$
| $syn\text{-}mult\ \pi\ (Wand\ A\ B) = Wand\ (syn\text{-}mult\ \pi\ A)\ (syn\text{-}mult\ \pi\ B)$
| $syn\text{-}mult\ \pi\ (Or\ A\ B) = Or\ (syn\text{-}mult\ \pi\ A)\ (syn\text{-}mult\ \pi\ B)$
| $syn\text{-}mult\ \pi\ (And\ A\ B) = And\ (syn\text{-}mult\ \pi\ A)\ (syn\text{-}mult\ \pi\ B)$

| *syn-mult* π (*Imp A B*) = *Imp* (*syn-mult* π *A*) (*syn-mult* π *B*)
| *syn-mult* π (*Mult* α *A*) = *syn-mult* (*smult* α π) *A*
| *syn-mult* π (*Exists x A*) = *Exists x* (*syn-mult* π *A*)
| *syn-mult* π (*Forall x A*) = *Forall x* (*syn-mult* π *A*)
| *syn-mult* π (*Wildcard A*) = *Wildcard A*
| *syn-mult* π *A* = *Mult* π *A*

definition *div-state* **where**
div-state π σ = (*SOME r*. $\sigma = \pi \odot r$)

lemma *div-state-ok*:
$\sigma = \pi \odot$ (*div-state* π σ)
⟨*proof*⟩

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

theorem *syn-sen-mult-same*:
σ, *s*, $\Delta \models$ *syn-mult* π *A* $\longleftrightarrow$ σ, *s*, $\Delta \models$ *Mult* π *A*
⟨*proof*⟩

7.2 Monotonicity and fixed point

fun *pos-neg-rec-call* :: *bool* ⇒ (*'a*, *'b*, *'c*, *'d*) *assertion* ⇒ *bool* **where**
pos-neg-rec-call b Pred $\longleftrightarrow$ *b*
| *pos-neg-rec-call b* (*Mult - A*) $\longleftrightarrow$ *pos-neg-rec-call b A*
| *pos-neg-rec-call b* (*Exists - A*) $\longleftrightarrow$ *pos-neg-rec-call b A*
| *pos-neg-rec-call b* (*Forall - A*) $\longleftrightarrow$ *pos-neg-rec-call b A*
| *pos-neg-rec-call b* (*Star A B*) $\longleftrightarrow$ *pos-neg-rec-call b A* $\wedge$ *pos-neg-rec-call b B*
| *pos-neg-rec-call b* (*Or A B*) $\longleftrightarrow$ *pos-neg-rec-call b A* $\wedge$ *pos-neg-rec-call b B*
| *pos-neg-rec-call b* (*And A B*) $\longleftrightarrow$ *pos-neg-rec-call b A* $\wedge$ *pos-neg-rec-call b B*
| *pos-neg-rec-call b* (*Wand A B*) $\longleftrightarrow$ *pos-neg-rec-call* ($\neg$ *b*) *A* $\wedge$ *pos-neg-rec-call b B*
| *pos-neg-rec-call b* (*Imp A B*) $\longleftrightarrow$ *pos-neg-rec-call* ($\neg$ *b*) *A* $\wedge$ *pos-neg-rec-call b B*
| *pos-neg-rec-call* - (*Sem* -) $\longleftrightarrow$ *True*
| *pos-neg-rec-call b* (*Bounded A*) $\longleftrightarrow$ *pos-neg-rec-call b A*
| *pos-neg-rec-call b* (*Wildcard A*) $\longleftrightarrow$ *pos-neg-rec-call b A*

lemma *pos-neg-rec-call-mono*:
assumes *pos-neg-rec-call b A*
shows (*b* $\longrightarrow$ *monotonic* (*applies-eq A*)) $\wedge$ ($\neg$ *b* $\longrightarrow$ *non-increasing* (*applies-eq A*))
⟨*proof*⟩

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

theorem *exists-lfp-gfp*:
assumes *pos-neg-rec-call True A*
shows σ, *s*, *LFP* (*applies-eq A*) $\models$ *A* $\longleftrightarrow$ $\sigma \in$ *LFP* (*applies-eq A*) *s*
and σ, *s*, *GFP* (*applies-eq A*) $\models$ *A* $\longleftrightarrow$ $\sigma \in$ *GFP* (*applies-eq A*) *s*
⟨*proof*⟩

7.3 Combinability

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

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

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

7.4 Theorems

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

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

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

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 True A*
shows *syn-mult* π *A*, *LFP* (*applies-eq A*) $\vdash$ *Mult* π *Pred*
$\langle proof \rangle$

theorem *unfold-lfp*:
assumes *pos-neg-rec-call True A*

shows *Mult* π *Pred*, *LFP* (*applies-eq* A) $\vdash$ *syn-mult* π A
⟨*proof*⟩

theorem *fold-gfp*:
assumes *pos-neg-rec-call True* A
shows *syn-mult* π A, *GFP* (*applies-eq* A) $\vdash$ *Mult* π *Pred*
⟨*proof*⟩

theorem *unfold-gfp*:
assumes *pos-neg-rec-call True* A
shows *Mult* π *Pred*, *GFP* (*applies-eq* A) $\vdash$ *syn-mult* π A
⟨*proof*⟩

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

theorem *wf-assertion-combinable-lfp*:
assumes *wf-assertion* A
and *pos-neg-rec-call True* A
shows *sem-combinable* (*LFP* (*applies-eq* A))
⟨*proof*⟩

theorem *wf-assertion-combinable-gfp*:
assumes *wf-assertion* A
and *pos-neg-rec-call True* A
shows *sem-combinable* (*GFP* (*applies-eq* A))
⟨*proof*⟩

theorem *wf-combine*:
assumes *wf-assertion* A
and *pos-neg-rec-call True* A
shows *Star* (*Mult* α *Pred*) (*Mult* β *Pred*), *LFP* (*applies-eq* A) $\vdash$ *Mult* (*sadd* α β) *Pred*
and *Star* (*Mult* α *Pred*) (*Mult* β *Pred*), *GFP* (*applies-eq* A) $\vdash$ *Mult* (*sadd* α β) *Pred*
⟨*proof*⟩

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. Vizel, 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.