

Formalization of a Framework for the Sound Automation of Magic Wands

Thibault Dardinier

May 14, 2024

Abstract

The magic wand \multimap (also called separating implication) is a separation logic [4] connective commonly used to specify properties of partial data structures, for instance during iterative traversals. A *footprint* of a magic wand formula $A \multimap B$ is a state that, combined with any state in which A holds, yields a state in which B holds. The key challenge of proving a magic wand (also called *packaging* a wand) is to find such a footprint. Existing package algorithms either have a high annotation overhead or are unsound.

In this entry, we formally define a framework for the sound automation of magic wands, described in a paper at CAV 2022 [2], and prove that it is sound and complete. This framework, called the *package logic*, precisely characterises a wide design space of possible package algorithms applicable to a large class of separation logics.

Contents

1	Separation Algebra	2
1.1	Definitions	2
1.2	First lemmata	3
1.2.1	plus	4
1.2.2	Pure	4
1.3	Succ is an order	4
1.4	Core (pure) and stabilize (stable)	5
1.5	Subtraction	6
1.6	Lifting the algebra to sets of states	8
1.7	Addition of more than two states	11
2	Package Logic	12
2.1	Definitions	12
2.2	Lemmas	15
2.3	Lemmas for completeness	17
2.4	Soundness	18
2.5	Completeness	18

1 Separation Algebra

In this section, we formalize the concept of a separation algebra [1, 3], on which our package logic is based.

```
theory SepAlgebra
  imports Main
begin
```

```
type-synonym 'a property = 'a  $\Rightarrow$  bool
```

```
locale sep-algebra =
```

```
  fixes plus :: 'a  $\Rightarrow$  'a  $\Rightarrow$  'a option (infixl  $\oplus$  63)
```

```
  fixes core :: 'a  $\Rightarrow$  'a (|-|)
```

```
  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:  $a \oplus b = \text{Some } ab \wedge b \oplus c = \text{None} \implies ab \oplus c = \text{None}$ 
```

```
    and core-is-smaller:  $\text{Some } x = x \oplus |x|$ 
```

```
    and core-is-pure:  $\text{Some } |x| = |x| \oplus |x|$ 
```

```
    and core-max:  $\text{Some } x = x \oplus c \implies (\exists r. \text{Some } |x| = c \oplus r)$ 
```

```
    and core-sum:  $\text{Some } c = a \oplus b \implies \text{Some } |c| = |a| \oplus |b|$ 
```

```
    and positivity:  $a \oplus b = \text{Some } c \implies \text{Some } c = c \oplus c \implies \text{Some } a = a \oplus a$ 
```

```
    and cancellative:  $\text{Some } a = b \oplus x \implies \text{Some } a = b \oplus y \implies |x| = |y| \implies x$   
=  $y$ 
```

```
begin
```

```
lemma asso3:
```

```
  assumes  $a \oplus b = \text{None}$ 
```

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

```
  shows  $a \oplus bc = \text{None}$ 
```

```
   $\langle$ proof $\rangle$ 
```

1.1 Definitions

```
definition defined :: 'a  $\Rightarrow$  'a  $\Rightarrow$  bool (infixl ## 62) where  
   $a \## b \longleftrightarrow a \oplus b \neq \text{None}$ 
```

```
definition greater :: 'a  $\Rightarrow$  'a  $\Rightarrow$  bool (infixl  $\succeq$  50) where  
   $a \succeq b \longleftrightarrow (\exists c. \text{Some } a = b \oplus c)$ 
```

```
definition pure :: 'a  $\Rightarrow$  bool where  
   $\text{pure } a \longleftrightarrow \text{Some } a = a \oplus a$ 
```

definition *minus* :: 'a \Rightarrow 'a \Rightarrow 'a (**infixl** \ominus 63)
where $b \ominus a = (\text{THE-default } b (\lambda x. \text{Some } b = a \oplus x \wedge x \succeq |b|))$

definition *add-set* :: 'a set \Rightarrow 'a set \Rightarrow 'a set (**infixl** \otimes 60) **where**
 $A \otimes B = \{ \varphi \mid \varphi \text{ a b. } a \in A \wedge b \in B \wedge \text{Some } \varphi = a \oplus b \}$

definition *greater-set* :: 'a set \Rightarrow 'a set \Rightarrow bool (**infixl** \gg 50) **where**
 $A \gg B \longleftrightarrow (\forall a \in A. \exists b \in B. a \succeq b)$

definition *up-closed* :: 'a set \Rightarrow bool **where**
 $\text{up-closed } A \longleftrightarrow (\forall \varphi'. (\exists \varphi \in A. \varphi' \succeq \varphi) \longrightarrow \varphi' \in A)$

definition *equiv* :: 'a set \Rightarrow 'a set \Rightarrow bool (**infixl** \sim 40) **where**
 $A \sim B \longleftrightarrow A \gg B \wedge B \gg A$

definition *setify* :: 'a property \Rightarrow ('a set \Rightarrow bool) **where**
 $\text{setify } P \ A \longleftrightarrow (\forall x \in A. P \ x)$

definition *mono-prop* :: 'a property \Rightarrow bool **where**
 $\text{mono-prop } P \longleftrightarrow (\forall x \ y. y \succeq x \wedge P \ x \longrightarrow P \ y)$

definition *under* :: 'a set \Rightarrow 'a \Rightarrow 'a set **where**
 $\text{under } A \ \omega = \{ \omega' \mid \omega'. \omega' \in A \wedge \omega \succeq \omega' \}$

definition *max-projection-prop* :: ('a \Rightarrow bool) \Rightarrow ('a \Rightarrow 'a) \Rightarrow bool **where**
 $\text{max-projection-prop } P \ f \longleftrightarrow (\forall x. x \succeq f \ x \wedge P \ (f \ x) \wedge$
 $(\forall p. P \ p \wedge x \succeq p \longrightarrow f \ x \succeq p))$

inductive *multi-plus* :: 'a list \Rightarrow 'a \Rightarrow bool **where**
MPSingle: $\text{multi-plus } [a] \ a$
MPConcat: $\llbracket \text{length } la > 0 ; \text{length } lb > 0 ; \text{multi-plus } la \ a ; \text{multi-plus } lb \ b ;$
 $\text{Some } \omega = a \oplus b \rrbracket \Longrightarrow \text{multi-plus } (la @ lb) \ \omega$

fun *splus* :: 'a option \Rightarrow 'a option \Rightarrow 'a option **where**
 $\text{splus } \text{None} \ - = \text{None}$
 $\mid \text{splus } \text{None} \ - = \text{None}$
 $\mid \text{splus } (\text{Some } a) \ (\text{Some } b) = a \oplus b$

1.2 First lemmata

lemma *greater-equiv*:
 $a \succeq b \longleftrightarrow (\exists c. \text{Some } a = c \oplus b)$
<proof>

lemma *smaller-compatible*:
assumes $a' \ \#\# \ b$
and $a' \succeq a$
shows $a \ \#\# \ b$

<proof>

lemma *bigger-sum-smaller:*

assumes *Some c = a ⊕ b*

and *a ≥ a'*

shows *∃ b'. b' ≥ b ∧ Some c = a' ⊕ b'*

<proof>

1.2.1 splus

lemma *splus-develop:*

assumes *Some a = b ⊕ c*

shows *a ⊕ d = splus (splus (Some b) (Some c)) (Some d)*

<proof>

lemma *splus-comm:*

splus a b = splus b a

<proof>

lemma *splus-asso:*

splus (splus a b) c = splus a (splus b c)

<proof>

1.2.2 Pure

lemma *pure-stable:*

assumes *pure a*

and *pure b*

and *Some c = a ⊕ b*

shows *pure c*

<proof>

lemma *pure-smaller:*

assumes *pure a*

and *a ≥ b*

shows *pure b*

<proof>

1.3 Succ is an order

lemma *succ-antisym:*

assumes *a ≥ b*

and *b ≥ a*

shows *a = b*

<proof>

lemma *succ-trans:*

assumes *a ≥ b*

and *b ≥ c*

shows $a \succeq c$
<proof>

lemma *succ-refl*:

$a \succeq a$
<proof>

1.4 Core (pure) and stabilize (stable)

lemma *max-projection-propI*:

assumes $\bigwedge x. x \succeq f x$
and $\bigwedge x. P (f x)$
and $\bigwedge x p. P p \wedge x \succeq p \implies f x \succeq p$
shows *max-projection-prop* $P f$
<proof>

lemma *max-projection-propE*:

assumes *max-projection-prop* $P f$
shows $\bigwedge x. x \succeq f x$
and $\bigwedge x. P (f x)$
and $\bigwedge x p. P p \wedge x \succeq p \implies f x \succeq p$
<proof>

lemma *max-projection-prop-pure-core*:

max-projection-prop *pure core*
<proof>

lemma *mpp-smaller*:

assumes *max-projection-prop* $P f$
shows $x \succeq f x$
<proof>

lemma *mpp-compatible*:

assumes *max-projection-prop* $P f$
and $a \#\# b$
shows $f a \#\# f b$
<proof>

lemma *mpp-prop*:

assumes *max-projection-prop* $P f$
shows $P (f x)$
<proof>

lemma *mppI*:

assumes *max-projection-prop* $P f$
and $a \succeq x$

and $P x$
and $x \succeq f a$
shows $x = f a$
 $\langle proof \rangle$

lemma *mpp-invo*:
assumes *max-projection-prop* $P f$
shows $f (f x) = f x$
 $\langle proof \rangle$

lemma *mpp-mono*:
assumes *max-projection-prop* $P f$
and $a \succeq b$
shows $f a \succeq f b$
 $\langle proof \rangle$

1.5 Subtraction

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

lemma *smaller-than-core*:
assumes $y \succeq x$
and *Some* $z = x \oplus |y|$
shows $|z| = |y|$
 $\langle proof \rangle$

lemma *extract-core*:
assumes *Some* $b = a \oplus x \wedge x \succeq |b|$
shows $|x| = |b|$
 $\langle proof \rangle$

lemma *minus-unique*:
assumes *Some* $b = a \oplus x \wedge x \succeq |b|$
and *Some* $b = a \oplus y \wedge y \succeq |b|$
shows $x = y$
 $\langle proof \rangle$

lemma *minus-exists*:
assumes $b \succeq a$
shows $\exists x. \text{Some } b = a \oplus x \wedge x \succeq |b|$
 $\langle proof \rangle$

lemma *minus-equiv-def*:

assumes $b \succeq a$

shows *Some* $b = a \oplus (b \ominus a) \wedge (b \ominus a) \succeq |b|$

<proof>

lemma *minus-default*:

assumes $\neg b \succeq a$

shows $b \ominus a = b$

<proof>

lemma *minusI*:

assumes *Some* $b = a \oplus x$

and $x \succeq |b|$

shows $x = b \ominus a$

<proof>

lemma *minus-core*:

$|a \ominus b| = |a|$

<proof>

lemma *minus-core-weaker*:

$|a \ominus b| = |a| \ominus |b|$

<proof>

lemma *minus-equiv-def-any-elem*:

assumes *Some* $x = a \oplus b$

shows *Some* $(x \ominus a) = b \oplus |x|$

<proof>

lemma *minus-bigger*:

assumes *Some* $x = a \oplus b$

shows $x \ominus a \succeq b$

<proof>

lemma *minus-smaller*:

assumes $x \succeq a$

shows $x \succeq x \ominus a$

<proof>

lemma *minus-sum*:

assumes *Some* $a = b \oplus c$

and $x \succeq a$

shows $x \ominus a = (x \ominus b) \ominus c$

<proof>

lemma *smaller-compatible-core*:

assumes $y \succeq x$

shows $x \#\# |y|$

<proof>

lemma *smaller-pure-sum-smaller*:

assumes $y \succeq a$
and $y \succeq b$
and *Some* $x = a \oplus b$
and *pure* b
shows $y \succeq x$

<proof>

lemma *greater-minus-trans*:

assumes $y \succeq x$
and $x \succeq a$
shows $y \ominus a \succeq x \ominus a$

<proof>

lemma *minus-and-plus*:

assumes *Some* $\omega' = \omega \oplus r$
and $\omega \succeq a$
shows *Some* $(\omega' \ominus a) = (\omega \ominus a) \oplus r$

<proof>

1.6 Lifting the algebra to sets of states

lemma *add-set-comm*:

$A \otimes B = B \otimes A$

<proof>

lemma *x-elem-set-product*:

$x \in A \otimes B \iff (\exists a b. a \in A \wedge b \in B \wedge \text{Some } x = a \oplus b)$

<proof>

lemma *x-elem-set-product-splus*:

$x \in A \otimes B \iff (\exists a b. a \in A \wedge b \in B \wedge \text{Some } x = \text{splus } (\text{Some } a) (\text{Some } b))$

<proof>

lemma *add-set-asso*:

$(A \otimes B) \otimes C = A \otimes (B \otimes C)$ (**is** $?A = ?B$)

<proof>

lemma *up-closedI*:

assumes $\bigwedge \varphi' \varphi. (\varphi' :: 'a) \succeq \varphi \wedge \varphi \in A \implies \varphi' \in A$
shows *up-closed* A

<proof>

lemma *up-closed-plus-UNIV*:

up-closed ($A \otimes UNIV$)
<proof>

lemma *succ-set-trans*:
assumes $A \gg B$
and $B \gg C$
shows $A \gg C$
<proof>

lemma *greater-setI*:
assumes $\bigwedge a. a \in A \implies (\exists b \in B. a \succeq b)$
shows $A \gg B$
<proof>

lemma *bigger-set*:
assumes $A' \gg A$
shows $A' \otimes B \gg A \otimes B$
<proof>

lemma *bigger-singleton*:
assumes $\varphi' \succeq \varphi$
shows $\{\varphi'\} \gg \{\varphi\}$
<proof>

lemma *add-set-elem*:
 $\varphi \in A \otimes B \iff (\exists a b. \text{Some } \varphi = a \oplus b \wedge a \in A \wedge b \in B)$
<proof>

lemma *up-closed-sum*:
assumes *up-closed* A
and *up-closed* B
shows *up-closed* $(A \otimes B)$
<proof>

lemma *up-closed-bigger-subset*:
assumes *up-closed* B
and $A \gg B$
shows $A \subseteq B$
<proof>

lemma *up-close-equiv*:
assumes *up-closed* A
and *up-closed* B
shows $A \sim B \iff A = B$
<proof>

lemma *equiv-stable-sum*:
assumes $A \sim B$
shows $A \otimes C \sim B \otimes C$

<proof>

lemma *equiv-up-closed-subset*:

assumes *up-closed A*

and *equiv B C*

shows $B \subseteq A \longleftrightarrow C \subseteq A$ (**is** $?B \longleftrightarrow ?C$)

<proof>

lemma *mono-propI*:

assumes $\bigwedge x y. y \succeq x \wedge P x \implies P y$

shows *mono-prop P*

<proof>

lemma *mono-prop-set*:

assumes $A \gg B$

and *setify P B*

and *mono-prop P*

shows *setify P A*

<proof>

lemma *mono-prop-set-equiv*:

assumes *mono-prop P*

and *equiv A B*

shows $\text{setify } P A \longleftrightarrow \text{setify } P B$

<proof>

lemma *setify-sum*:

setify P (A \otimes B) \longleftrightarrow ($\forall x \in A. \text{setify } P (\{x\} \otimes B)$) (**is** $?A \longleftrightarrow ?B$)

<proof>

lemma *setify-sum-image*:

setify P ((Set.image f A) \otimes B) \longleftrightarrow ($\forall x \in A. \text{setify } P (\{f x\} \otimes B)$)

<proof>

lemma *equivI*:

assumes $A \gg B$

and $B \gg A$

shows *equiv A B*

<proof>

lemma *sub-bigger*:

assumes $A \subseteq B$

shows $A \gg B$

<proof>

lemma *larger-set-refl*:

$A \gg A$

<proof>

definition *upper-closure* **where**

$$\text{upper-closure } A = \{ \varphi' \mid \varphi' \varphi. \varphi' \succeq \varphi \wedge \varphi \in A \}$$

lemma *upper-closure-up-closed*:

$$\text{up-closed } (\text{upper-closure } A)$$

<proof>

1.7 Addition of more than two states

lemma *multi-decompose*:

assumes *multi-plus* $l \ \omega$

shows $\text{length } l \geq 2 \implies (\exists a \ b \ la \ lb. l = la @ lb \wedge \text{length } la > 0 \wedge \text{length } lb > 0 \wedge \text{multi-plus } la \ a \wedge \text{multi-plus } lb \ b \wedge \text{Some } \omega = a \oplus b)$

<proof>

lemma *multi-take-drop*:

assumes *multi-plus* $l \ \omega$

and $\text{length } l \geq 2$

shows $\exists n \ a \ b. n > 0 \wedge n < \text{length } l \wedge \text{multi-plus } (\text{take } n \ l) \ a \wedge \text{multi-plus } (\text{drop } n \ l) \ b \wedge \text{Some } \omega = a \oplus b$

<proof>

lemma *multi-plus-single*:

assumes *multi-plus* $[v] \ a$

shows $a = v$

<proof>

lemma *multi-plus-two*:

assumes $\text{length } l \geq 2$

shows $\text{multi-plus } l \ \omega \longleftrightarrow (\exists a \ b \ la \ lb. l = (la @ lb) \wedge \text{length } la > 0 \wedge \text{length } lb > 0 \wedge \text{multi-plus } la \ a \wedge \text{multi-plus } lb \ b \wedge \text{Some } \omega = a \oplus b)$ (**is** $?A \longleftrightarrow ?B$)

<proof>

lemma *multi-plus-head-tail*:

$\text{length } l \leq n \wedge \text{length } l \geq 2 \longrightarrow (\text{multi-plus } l \ \omega \longleftrightarrow (\exists r. \text{Some } \omega = (\text{List.hd } l) \oplus r \wedge \text{multi-plus } (\text{List.tl } l) \ r))$

<proof>

lemma *not-multi-plus-empty*:

$\neg \text{multi-plus } [] \ \omega$

<proof>

lemma *multi-plus-deter*:

$\text{length } l \leq n \implies \text{multi-plus } l \ \omega \implies \text{multi-plus } l \ \omega' \implies \omega = \omega'$

<proof>

lemma *multi-plus-implies-multi-plus-of-drop*:
 assumes *multi-plus l ω*
 and *n < length l*
 shows $\exists a. \text{multi-plus } (\text{drop } n \ l) \ a \wedge \omega \succeq a$
<proof>

lemma *multi-plus-bigger-than-head*:
 assumes *length l > 0*
 and *multi-plus l ω*
 shows $\omega \succeq \text{List.hd } l$
<proof>

lemma *multi-plus-bigger*:
 assumes *i < length l*
 and *multi-plus l ω*
 shows $\omega \succeq (l ! i)$
<proof>

lemma *sum-then-singleton*:
 Some a = b ⊕ c ⟷ {a} = {b} ⊗ {c} (is ?A ⟷ ?B)
<proof>

lemma *empty-set-sum*:
 $\{\} \otimes A = \{\}$
<proof>

end

end

2 Package Logic

In this section, we define our package logic, as described in [2], and then prove that this logic is sound and complete for packaging magic wands.

theory *PackageLogic*
 imports *Main SepAlgebra*
begin

2.1 Definitions

type-synonym *'a abool = 'a ⇒ bool*

datatype *'a aassertion =*
 AStar 'a aassertion 'a aassertion

| *AImp* 'a abool 'a aassertion
 | *ASem* 'a abool

locale *package-logic* = *sep-algebra* +

fixes *unit* :: 'a
fixes *stable* :: 'a \Rightarrow bool

assumes *unit-neutral*: *Some* a = a \oplus *unit*
and *stable-sum*: *stable* a \Rightarrow *stable* b \Rightarrow *Some* x = a \oplus b \Rightarrow *stable* x
and *stable-unit*: *stable* *unit*

begin

fun *sat* :: 'a aassertion \Rightarrow 'a \Rightarrow bool **where**
sat (*AStar* A B) $\varphi \longleftrightarrow (\exists a b. \text{Some } \varphi = a \oplus b \wedge \text{sat } A a \wedge \text{sat } B b)$
 | *sat* (*AImp* b A) $\varphi \longleftrightarrow (b \varphi \longrightarrow \text{sat } A \varphi)$
 | *sat* (*ASem* A) $\varphi \longleftrightarrow A \varphi$

definition *mono-pure-cond* **where**

mono-pure-cond b $\longleftrightarrow (\forall \varphi. b \varphi \longleftrightarrow b |\varphi|) \wedge (\forall \varphi' \varphi r. \text{pure } r \wedge \text{Some } \varphi' = \varphi \oplus r \wedge \neg b \varphi \longrightarrow \neg b \varphi')$

definition *bool-conj* **where**

bool-conj a b x $\longleftrightarrow a x \wedge b x$

type-synonym 'c *pruner* = 'c \Rightarrow bool

definition *mono-pruner* :: 'a *pruner* \Rightarrow bool **where**

mono-pruner p $\longleftrightarrow (\forall \varphi' \varphi r. \text{pure } r \wedge p \varphi \wedge \text{Some } \varphi' = \varphi \oplus r \longrightarrow p \varphi')$

fun *wf-assertion* **where**

wf-assertion (*AStar* A B) $\longleftrightarrow \text{wf-assertion } A \wedge \text{wf-assertion } B$
 | *wf-assertion* (*AImp* b A) $\longleftrightarrow \text{mono-pure-cond } b \wedge \text{wf-assertion } A$
 | *wf-assertion* (*ASem* A) $\longleftrightarrow \text{mono-pruner } A$

type-synonym 'c *transformer* = 'c \Rightarrow 'c

type-synonym 'c *ext-state* = 'c \times 'c \times 'c *transformer*

inductive *package-rhs* ::

'a \Rightarrow 'a \Rightarrow 'a *ext-state* set \Rightarrow 'a abool \Rightarrow 'a aassertion \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a *ext-state* set \Rightarrow bool **where**

AStar: $\llbracket \text{package-rhs } \varphi f S pc A \varphi' f' S' ; \text{package-rhs } \varphi' f' S' pc B \varphi'' f'' S'' \rrbracket$
 $\Rightarrow \text{package-rhs } \varphi f S pc (\text{AStar } A B) \varphi'' f'' S''$
 | *AImp*: *package-rhs* $\varphi f S (\text{bool-conj } pc b) A \varphi' f' S' \Rightarrow \text{package-rhs } \varphi f S pc$

$(A \text{Imp } b \ A) \ \varphi' \ f' \ S'$

| $ASem$: $\llbracket \bigwedge a \ u \ T \ b. (a, u, T) \in S \implies pc \ a \implies b = witness \ (a, u, T) \implies a \succeq b \wedge B \ b ;$
 $S' = \{ (a, u, T) \mid a \ u \ T. (a, u, T) \in S \wedge \neg pc \ a \}$
 $\cup \{ (a \oplus b, the \ (u \oplus b), T) \mid a \ u \ T \ b. (a, u, T) \in S \wedge pc \ a \wedge b = witness \ (a, u, T) \}$ \rrbracket
 $\implies package\text{-}rhs \ \varphi \ f \ S \ pc \ (ASem \ B) \ \varphi \ f \ S'$

| $AddFromOutside$: $\llbracket Some \ \varphi = \varphi' \oplus m ; package\text{-}rhs \ \varphi' \ f' \ S' \ pc \ A \ \varphi'' \ f'' \ S'' ;$
 $stable \ m ; Some \ f' = f \oplus m ;$
 $S' = \{ (r, u, T) \mid a \ u \ T \ r. (a, u, T) \in S \wedge Some \ r = a \oplus (T \ f' \ominus T \ f) \wedge r \ \#\# \ u \}$ \rrbracket
 $\implies package\text{-}rhs \ \varphi \ f \ S \ pc \ A \ \varphi'' \ f'' \ S''$

definition *package-sat where*

$package\text{-}sat \ pc \ A \ a' \ u' \ u \longleftrightarrow (pc \ |a'| \longrightarrow (\exists x. Some \ x = |a'| \oplus (u' \ominus u) \wedge sat \ A \ x))$

definition *package-rhs-connection :: 'a \Rightarrow 'a \Rightarrow 'a ext-state set \Rightarrow 'a abool \Rightarrow 'a aassertion \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a ext-state set \Rightarrow bool where*

$package\text{-}rhs\text{-}connection \ \varphi \ f \ S \ pc \ A \ \varphi' \ f' \ S' \longleftrightarrow f' \succeq f \wedge \varphi \ \#\# \ f \wedge \varphi \oplus f = \varphi' \oplus f' \wedge stable \ f' \wedge$
 $(\forall (a, u, T) \in S. \exists au. Some \ au = a \oplus u \wedge (au \ \#\# \ (T \ f' \ominus T \ f) \longrightarrow$
 $(\exists a' \ u'. (a', u', T) \in S' \wedge |a'| \succeq |a| \wedge au \oplus (T \ f' \ominus T \ f) = a' \oplus u' \wedge u' \succeq u \wedge package\text{-}sat \ pc \ A \ a' \ u' \ u))$

definition *mono-transformer :: 'a transformer \Rightarrow bool where*

$mono\text{-}transformer \ T \longleftrightarrow (\forall \varphi \ \varphi'. \varphi' \succeq \varphi \longrightarrow T \ \varphi' \succeq T \ \varphi) \wedge T \ unit = unit$

definition *valid-package-set where*

$valid\text{-}package\text{-}set \ S \ f \longleftrightarrow (\forall (a, u, T) \in S. a \ \#\# \ u \wedge |a| \succeq |u| \wedge mono\text{-}transformer \ T \wedge a \succeq |T \ f|)$

definition *intuitionistic where*

$intuitionistic \ A \longleftrightarrow (\forall \varphi' \ \varphi. \varphi' \succeq \varphi \wedge A \ \varphi \longrightarrow A \ \varphi')$

definition *pure-remains where*

$pure\text{-}remains \ S \longleftrightarrow (\forall (a, u, p) \in S. pure \ a)$

definition *is-footprint-general :: 'a \Rightarrow ('a \Rightarrow 'a \Rightarrow 'a) \Rightarrow 'a aassertion \Rightarrow 'a aassertion \Rightarrow bool where*

$is\text{-}footprint\text{-}general \ w \ R \ A \ B \longleftrightarrow (\forall a \ b. sat \ A \ a \wedge Some \ b = a \oplus R \ a \ w \longrightarrow sat \ B \ b)$

definition *is-footprint-standard :: 'a \Rightarrow 'a aassertion \Rightarrow 'a aassertion \Rightarrow bool where*

$is\text{-}footprint\text{-}standard \ w \ A \ B \longleftrightarrow (\forall a \ b. sat \ A \ a \wedge Some \ b = a \oplus w \longrightarrow sat \ B \ b)$

2.2 Lemmas

lemma *is-footprint-generalI*:

assumes $\bigwedge a b. \text{sat } A \ a \wedge \text{Some } b = a \oplus R \ a \ w \implies \text{sat } B \ b$
shows *is-footprint-general* $w \ R \ A \ B$
 $\langle \text{proof} \rangle$

lemma *is-footprint-standardI*:

assumes $\bigwedge a b. \text{sat } A \ a \wedge \text{Some } b = a \oplus w \implies \text{sat } B \ b$
shows *is-footprint-standard* $w \ A \ B$
 $\langle \text{proof} \rangle$

lemma *mono-pure-condI*:

assumes $\bigwedge \varphi. b \ \varphi \longleftrightarrow b \ |\varphi|$
and $\bigwedge \varphi \ \varphi' \ r. \text{pure } r \wedge \text{Some } \varphi' = \varphi \oplus r \wedge \neg b \ \varphi \implies \neg b \ \varphi'$
shows *mono-pure-cond* b
 $\langle \text{proof} \rangle$

lemma *mono-pure-cond-conj*:

assumes *mono-pure-cond* pc
and *mono-pure-cond* b
shows *mono-pure-cond* $(\text{bool-conj } pc \ b)$
 $\langle \text{proof} \rangle$

lemma *bigger-sum*:

assumes *Some* $\varphi = a \oplus b$
and $\varphi' \succeq \varphi$
shows $\exists b'. b' \succeq b \wedge \text{Some } \varphi' = a \oplus b'$
 $\langle \text{proof} \rangle$

lemma *wf-assertion-sat-larger-pure*:

assumes *wf-assertion* A
and *sat* $A \ \varphi$
and *Some* $\varphi' = \varphi \oplus r$
and *pure* r
shows *sat* $A \ \varphi'$
 $\langle \text{proof} \rangle$

lemma *package-satI*:

assumes *pc* $|a'| \implies (\exists x. \text{Some } x = |a'| \oplus (u' \ominus u) \wedge \text{sat } A \ x)$
shows *package-sat* $pc \ A \ a' \ u' \ u$
 $\langle \text{proof} \rangle$

lemma *package-rhs-connection-instantiate*:

assumes *package-rhs-connection* $\varphi \ f \ S \ pc \ A \ \varphi' \ f' \ S'$
and $(a, u, T) \in S$
obtains au **where** *Some* $au = a \oplus u$

and $au \#\# (Tf' \ominus Tf) \implies \exists a' u'. (a', u', T) \in S' \wedge |a'| \succeq |a| \wedge au \oplus (Tf' \ominus Tf) = a' \oplus u' \wedge u' \succeq u \wedge \text{package-sat pc } A \ a' \ u' \ u$
 ⟨proof⟩

lemma *package-rhs-connectionI*:

assumes $\varphi \oplus f = \varphi' \oplus f'$
and *stable* f'
and $\varphi \#\# f$
and $f' \succeq f$
and $\bigwedge a \ u \ T. (a, u, T) \in S \implies (\exists au. \text{Some } au = a \oplus u \wedge (au \#\# (Tf' \ominus Tf)) \longrightarrow (\exists a' u'. (a', u', T) \in S' \wedge |a'| \succeq |a| \wedge au \oplus (Tf' \ominus Tf) = a' \oplus u' \wedge u' \succeq u \wedge \text{package-sat pc } A \ a' \ u' \ u))$
shows *package-rhs-connection* $\varphi \ f \ S \ pc \ A \ \varphi' \ f' \ S'$
 ⟨proof⟩

lemma *valid-package-setI*:

assumes $\bigwedge a \ u \ T. (a, u, T) \in S \implies a \#\# u \wedge |a| \succeq |u| \wedge \text{mono-transformer } T \wedge a \succeq |Tf|$
shows *valid-package-set* $S \ f$
 ⟨proof⟩

lemma *defined-sum-move*:

assumes $a \#\# b$
and *Some* $b = x \oplus y$
and *Some* $a' = a \oplus x$
shows $a' \#\# y$
 ⟨proof⟩

lemma *bigger-core-sum-defined*:

assumes $|a| \succeq b$
shows *Some* $a = a \oplus b$
 ⟨proof⟩

lemma *package-rhs-proof*:

assumes *package-rhs* $\varphi \ f \ S \ pc \ A \ \varphi' \ f' \ S'$
and *valid-package-set* $S \ f$
and *wf-assertion* A
and *mono-pure-cond* pc
and *stable* f
and $\varphi \#\# f$
shows *package-rhs-connection* $\varphi \ f \ S \ pc \ A \ \varphi' \ f' \ S' \wedge \text{valid-package-set } S' \ f'$
 ⟨proof⟩

lemma *unit-core*:

$|unit| = unit$
 ⟨proof⟩

lemma *unit-smaller*:

$\varphi \succeq \text{unit}$
 $\langle \text{proof} \rangle$

2.3 Lemmas for completeness

lemma *sat-star-exists-bigger*:

assumes $\text{sat } (A \text{Star } A \ B) \ \varphi$
and *wf-assertion* A
and *wf-assertion* B
shows $\exists a \ b. |a| \succeq |\varphi| \wedge |b| \succeq |\varphi| \wedge \text{Some } \varphi = a \oplus b \wedge \text{sat } A \ a \wedge \text{sat } B \ b$
 $\langle \text{proof} \rangle$

lemma *let-pair-instantiate*:

assumes $(a, b) = f \ x \ y$
shows $(\text{let } (a, b) = f \ x \ y \ \text{in } g \ a \ b) = g \ a \ b$
 $\langle \text{proof} \rangle$

lemma *greater-than-sum-exists*:

assumes $a \succeq b$
and $\text{Some } b = b1 \oplus b2$
shows $\exists r. \text{Some } a = r \oplus b2 \wedge |r| \succeq |a| \wedge r \succeq b1$
 $\langle \text{proof} \rangle$

lemma *bigger-the*:

assumes $\text{Some } a = x' \oplus y$
and $x' \succeq x$
shows $\text{the } (|a| \oplus x') \succeq \text{the } (|a| \oplus x)$
 $\langle \text{proof} \rangle$

lemma *wf-assertion-and-the*:

assumes $|a| \#\# b$
and $\text{sat } A \ b$
and *wf-assertion* A
shows $\text{sat } A \ (\text{the } (|a| \oplus b))$
 $\langle \text{proof} \rangle$

lemma *minus-some*:

assumes $a \succeq b$
shows $\text{Some } a = b \oplus (a \ominus b)$
 $\langle \text{proof} \rangle$

lemma *core-mono*:

assumes $a \succeq b$
shows $|a| \succeq |b|$
 $\langle \text{proof} \rangle$

lemma *prove-last-completeness*:

assumes $a' \succeq a$
and $\text{Some } a = \text{nf1} \oplus f2$
shows $a' \ominus \text{nf1} \succeq f2$
 ⟨proof⟩

lemma *completeness-aux*:

assumes $\bigwedge a u T. (a, u, T) \in S \implies |f1 a u T| \succeq |a| \wedge \text{Some } a = f1 a u T \oplus f2 a u T \wedge (\text{pc } |a| \longrightarrow \text{sat } A (\text{the } (|a| \oplus (f1 a u T))))$
and *valid-package-set* $S f$
and *wf-assertion* A
and *mono-pure-cond* pc
and *stable* $f \wedge \varphi \#\# f$
shows $\exists S'. \text{package-rhs } \varphi f S \text{ pc } A \varphi f S' \wedge (\forall (a', u', T) \in S'. \exists a u. (a, u, T) \in S \wedge a' \succeq f2 a u T \wedge |a'| = |a|)$
 ⟨proof⟩

2.4 Soundness

theorem *general-soundness*:

assumes *package-rhs* $\varphi \text{ unit } \{ (a, \text{unit}, T) \mid a T. (a, T) \in S \} (\lambda-. \text{True}) A \varphi' f S'$
and $\bigwedge a T. (a, T) \in S \implies \text{mono-transformer } T$
and *wf-assertion* A
and *intuitionistic* $(\text{sat } A) \vee \text{pure-remains } S'$
shows $\text{Some } \varphi = \varphi' \oplus f \wedge \text{stable } f \wedge (\forall (a, T) \in S. a \#\# T f \longrightarrow \text{sat } A (\text{the } (a \oplus T f)))$
 ⟨proof⟩

theorem *soundness*:

assumes *wf-assertion* B
and $\bigwedge a. \text{sat } A a \implies a \in S$
and $\bigwedge a. a \in S \implies \text{mono-transformer } (R a)$
and *package-rhs* $\sigma \text{ unit } \{ (a, \text{unit}, R a) \mid a. a \in S \} (\lambda-. \text{True}) B \sigma' w S'$
and *intuitionistic* $(\text{sat } B) \vee \text{pure-remains } S'$
shows $\text{stable } w \wedge \text{Some } \sigma = \sigma' \oplus w \wedge \text{is-footprint-general } w R A B$
 ⟨proof⟩

corollary *soundness-paper*:

assumes *wf-assertion* B
and $\bigwedge a. \text{sat } A a \implies a \in S$
and *package-rhs* $\sigma \text{ unit } \{ (a, \text{unit}, \text{id}) \mid a. a \in S \} (\lambda-. \text{True}) B \sigma' w S'$
and *intuitionistic* $(\text{sat } B) \vee \text{pure-remains } S'$
shows $\text{stable } w \wedge \text{Some } \sigma = \sigma' \oplus w \wedge \text{is-footprint-standard } w A B$
 ⟨proof⟩

2.5 Completeness

theorem *general-completeness*:

assumes $\bigwedge a u T x. (a, u, T) \in S \implies \text{Some } x = a \oplus T f \implies \text{sat } A x$
and $\text{Some } \varphi = \varphi' \oplus f$

and *stable* f
and *valid-package-set* S *unit*
and *wf-assertion* A
shows $\exists S'. \text{package-rhs } \varphi \text{ unit } S (\lambda-. \text{True}) A \varphi' f S'$
 $\langle \text{proof} \rangle$

theorem *completeness:*

assumes *wf-assertion* B
and *stable* $w \wedge \text{is-footprint-general } w R A B$
and *Some* $\sigma = \sigma' \oplus w$
and $\bigwedge a. \text{sat } A a \implies \text{mono-transformer } (R a)$
shows $\exists S'. \text{package-rhs } \sigma \text{ unit } \{(a, \text{unit}, R a) \mid a. \text{sat } A a\} (\lambda-. \text{True}) B \sigma' w$
 S'
 $\langle \text{proof} \rangle$

corollary *completeness-paper:*

assumes *wf-assertion* B
and *stable* $w \wedge \text{is-footprint-standard } w A B$
and *Some* $\sigma = \sigma' \oplus w$
shows $\exists S'. \text{package-rhs } \sigma \text{ unit } \{(a, \text{unit}, \text{id}) \mid a. \text{sat } A a\} (\lambda-. \text{True}) B \sigma' w S'$
 $\langle \text{proof} \rangle$

end

end

References

- [1] C. Calcagno, P. W. O’Hearn, and H. Yang. Local Action and Abstract Separation Logic. In *Logic in Computer Science (LICS)*, pages 366–375, 2007.
- [2] 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.
- [3] R. Dockins, A. Hobor, and A. W. Appel. A Fresh Look at Separation Algebras and Share Accounting. In Z. Hu, editor, *Asian Symposium on Programming Languages and Systems (APLAS)*, pages 161–177, 2009.
- [4] J. C. Reynolds. Separation Logic: A Logic for Shared Mutable Data Structures. In *Logic in Computer Science (LICS)*, pages 55–74. IEEE, 2002.