

# A Restricted Definition of the Magic Wand to Soundly Combine Fractions of a Wand

Thibault Dardinier

March 17, 2025

## Abstract

Many separation logics support *fractional permissions* [1] to distinguish between read and write access to a heap location, for instance, to allow concurrent reads while enforcing exclusive writes. The concept has been generalized to fractional assertions [2, 5, 6, 3].  $A^p$  (where  $A$  is a separation logic assertion and  $p$  a fraction between 0 and 1) represents a fraction  $p$  of  $A$ .  $A^p$  holds in a state  $\sigma$  iff there exists a state  $\sigma_A$  in which  $A$  holds and  $\sigma$  is obtained from  $\sigma_A$  by multiplying all permission amounts held by  $p$ .

While  $A^{p+q}$  can always be split into  $A^p * A^q$ , recombining  $A^p * A^q$  into  $A^{p+q}$  is not always sound. We say that  $A$  is *combinable* iff the entailment  $A^p * A^q \models A^{p+q}$  holds for any two positive fractions  $p$  and  $q$  such that  $p + q \leq 1$ . Combinable assertions are particularly useful to reason about concurrent programs, for instance, to combine the post-conditions of parallel branches when they terminate. Unfortunately, the magic wand assertion  $A \multimap B$ , commonly used to specify properties of partial data structures, is typically *not* combinable.

In this entry, we formalize a novel, restricted definition of the magic wand, described in a paper at CAV 22 [4], which we call the *combinable wand*. We prove some key properties of the combinable wand; in particular, a combinable wand is combinable if its right-hand side is combinable.

## Contents

<b>1 State Model with Fractional Permissions</b>	<b>2</b>
1.1 Non-negative rationals (permission amounts) . . . . .	2
1.1.1 Definitions . . . . .	2
1.1.2 Lemmas . . . . .	3
1.2 Permission masks: Maps from heap locations to permission amounts . . . . .	7
1.2.1 Definitions . . . . .	7
1.2.2 Lemmas . . . . .	9
1.3 Partial heaps: Partial maps from heap location to values . . .	14

1.3.1	Definitions . . . . .	14
1.3.2	Lemmas . . . . .	15
1.4	This state model corresponds to a separation algebra . . . . .	19
<b>2</b>	<b>Combinable Magic Wands</b>	<b>20</b>
2.1	Definitions . . . . .	20
2.2	Lemmas . . . . .	21
2.3	The combinable wand is stronger than the original wand . . .	23
2.4	The combinable wand is the same as the original wand when the left-hand side is binary . . . . .	23
2.5	The combinable wand is combinable . . . . .	23
2.6	Theorems . . . . .	23

## 1 State Model with Fractional Permissions

In this section, we define a concrete state model based on fractional permissions [1]. A state is a pair of a permission mask and a partial heap. A permission mask is a total map from heap locations to a rational between 0 and 1 (included), while a partial heap is a partial map from heap locations to values. We also define a partial addition on these states, and show that this state model corresponds to a separation algebra.

### 1.1 Non-negative rationals (permission amounts)

```
theory PosRat
  imports Main HOL.Rat
begin
```

#### 1.1.1 Definitions

```
typedef prat = { r :: rat |r. r ≥ 0} ⟨proof⟩
```

```
setup-lifting type-definition-prat
```

```
lift-definition pwrite :: prat ⇒ 1 ⟨proof⟩
lift-definition pnone :: prat ⇒ 0 ⟨proof⟩
lift-definition half :: prat ⇒ 1 / 2 ⟨proof⟩
```

```
lift-definition pgte :: prat ⇒ prat ⇒ bool is (≥) ⟨proof⟩
lift-definition pgt :: prat ⇒ prat ⇒ bool is (>) ⟨proof⟩
```

```
lift-definition lt :: prat ⇒ prat ⇒ bool is (<) ⟨proof⟩
lift-definition ppos :: prat ⇒ bool is λp. p > 0 ⟨proof⟩
```

```
lift-definition pmult :: prat ⇒ prat ⇒ prat is (*) ⟨proof⟩
lift-definition padd :: prat ⇒ prat ⇒ prat is (+) ⟨proof⟩
```

```

lift-definition pdiv :: prat  $\Rightarrow$  prat  $\Rightarrow$  prat is (/)  $\langle proof \rangle$ 

lift-definition pmin :: prat  $\Rightarrow$  prat  $\Rightarrow$  prat is (min)  $\langle proof \rangle$ 
lift-definition pmax :: prat  $\Rightarrow$  prat  $\Rightarrow$  prat is (max)  $\langle proof \rangle$ 

```

### 1.1.2 Lemmas

**lemma** pmin-comm:  
 $pmin\ a\ b = pmin\ b\ a$   
 $\langle proof \rangle$

**lemma** pmin-greater:  
 $pgte\ a\ (pmin\ a\ b)$   
 $\langle proof \rangle$

**lemma** pmin-is:  
**assumes** pgte a b  
**shows** pmin a b = b  
 $\langle proof \rangle$

**lemma** pmax-comm:  
 $pmax\ a\ b = pmax\ b\ a$   
 $\langle proof \rangle$

**lemma** pmax-smaller:  
 $pgte\ (pmax\ a\ b)\ a$   
 $\langle proof \rangle$

**lemma** pmax-is:  
**assumes** pgte a b  
**shows** pmax a b = a  
 $\langle proof \rangle$

**lemma** pmax-is-smaller:  
**assumes** pgte x a  
**and** pgte x b  
**shows** pgte x (pmax a b)  
 $\langle proof \rangle$

**lemma** half-between-0-1:  
 $ppos\ half \wedge pgt\ purite\ half$   
 $\langle proof \rangle$

**lemma** pgd-implies-pgt:  
**assumes** pgd a b  
**shows** pgte a b

$\langle proof \rangle$

**lemma** half-plus-half:  
padd half half = pwrite  
 $\langle proof \rangle$

**lemma** padd-comm:  
padd a b = padd b a  
 $\langle proof \rangle$

**lemma** padd-asso:  
padd (padd a b) c = padd a (padd b c)  
 $\langle proof \rangle$

**lemma** p-greater-exists:  
pgte a b  $\longleftrightarrow$  ( $\exists r$ . a = padd b r)  
 $\langle proof \rangle$

**lemma** pgte-antisym:  
**assumes** pgte a b  
and pgte b a  
**shows** a = b  
 $\langle proof \rangle$

**lemma** greater-sum-both:  
**assumes** pgte a (padd b c)  
**shows**  $\exists a1 a2$ . a = padd a1 a2  $\wedge$  pgte a1 b  $\wedge$  pgte a2 c  
 $\langle proof \rangle$

**lemma** padd-cancellative:  
**assumes** a = padd x b  
and a = padd y b  
**shows** x = y  
 $\langle proof \rangle$

**lemma** not-pgte-charact:  
 $\neg$  pgte a b  $\longleftrightarrow$  pgt b a  
 $\langle proof \rangle$

**lemma** pgte-pgt:  
**assumes** pgt a b  
and pgte c d  
**shows** pgt (padd a c) (padd b d)  
 $\langle proof \rangle$

```
lemma pmult-distr:  
  pmult a (padd b c) = padd (pmult a b) (pmult a c)  
  ⟨proof⟩
```

```
lemma pmult-comm:  
  pmult a b = pmult b a  
  ⟨proof⟩
```

```
lemma pmult-special:  
  pmult pwrite x = x  
  pmult pnone x = pnone  
  ⟨proof⟩
```

```
definition pinv where  
  pinv p = pdiv pwrite p
```

```
lemma pinv-double-half:  
  assumes ppos p  
  shows pmult half (pinv p) = pinv (padd p p)  
  ⟨proof⟩
```

```
lemma ppos-mult:  
  assumes ppos a  
    and ppos b  
  shows ppos (pmult a b)  
  ⟨proof⟩
```

```
lemma padd-zero:  
  pnone = padd a b  $\longleftrightarrow$  a = pnone  $\wedge$  b = pnone  
  ⟨proof⟩
```

```
lemma ppos-add:  
  assumes ppos a  
  shows ppos (padd a b)  
  ⟨proof⟩
```

```
lemma pinv-inverts:  
  assumes pgte a b  
    and ppos b  
  shows pgte (pinv b) (pinv a)  
  ⟨proof⟩
```

```
lemma pinv-pmult-ok:
```

```

assumes ppos p
shows pmult p (pinv p) = pwrite
⟨proof⟩

```

```

lemma pinv-pwrite:
  pinv pwrite = pwrite
  ⟨proof⟩

```

```

lemma pmult-ppos:
  assumes ppos a
    and ppos b
  shows ppos (pmult a b)
  ⟨proof⟩

```

```

lemma ppos-inv:
  assumes ppos p
  shows ppos (pinv p)
⟨proof⟩

```

```

lemma pmin-pmax:
  assumes pgte x (pmin a b)
  shows x = pmin (pmax x a) (pmax x b)
⟨proof⟩

```

```

definition comp-one where
  comp-one p = (SOME r. padd p r = pwrite)

```

```

lemma padd-comp-one:
  assumes pgte pwrite x
  shows padd x (comp-one x) = pwrite
  ⟨proof⟩

```

```

lemma ppos-eq-pnone:
  ppos p  $\longleftrightarrow$  p ≠ pnone
  ⟨proof⟩

```

```

lemma pmult-order:
  assumes pgte a b
  shows pgte (pmult p a) (pmult b p)
  ⟨proof⟩

```

```

lemma multiply-smaller-pwrite:
  assumes pgte pwrite a
    and pgte pwrite b

```

```

shows pgte pwrite (pmult a b)
⟨proof⟩

lemma pmult-pdiv-cancel:
assumes ppos a
shows pmult a (pdiv x a) = x
⟨proof⟩

lemma pmult-padd:
pmult a (padd (pmult b x) (pmult c y)) = padd (pmult (pmult a b) x) (pmult
(pmult a c) y)
⟨proof⟩

lemma pdiv-smaller:
assumes pgte a b
and ppos a
shows pgte pwrite (pdiv b a)
⟨proof⟩

lemma sum-coeff:
assumes ppos a
and ppos b
shows padd (pdiv a (padd a b)) (pdiv b (padd a b)) = pwrite
⟨proof⟩

lemma padd-one-ineq-sum:
assumes padd a b = pwrite
and pgte x aa
and pgte x bb
shows pgte x (padd (pmult a aa) (pmult b bb))
⟨proof⟩

end

```

## 1.2 Permission masks: Maps from heap locations to permission amounts

```

theory Mask
imports PosRat
begin

```

### 1.2.1 Definitions

```

type-synonym field = string
type-synonym address = nat
type-synonym heap-loc = address × field

```

```

type-synonym mask = heap-loc  $\Rightarrow$  prat
type-synonym bmask = heap-loc  $\Rightarrow$  bool

definition null where null = 0

definition full-mask :: mask where
  full-mask hl = (if fst hl = null then pnone else pwrite)

definition multiply-mask :: prat  $\Rightarrow$  mask  $\Rightarrow$  mask where
  multiply-mask p  $\pi$  hl = pmult p ( $\pi$  hl)

fun empty-mask where
  empty-mask hl = pnone

fun empty-bmask where
  empty-bmask hl = False

fun add-acc where add-acc  $\pi$  hl p =  $\pi$ (hl := padd ( $\pi$  hl) p)

inductive rm-acc where
   $\pi$  hl = padd p r  $\implies$  rm-acc  $\pi$  hl p ( $\pi$ (hl := r))

fun add-masks where
  add-masks  $\pi'$   $\pi$  hl = padd ( $\pi'$  hl) ( $\pi$  hl)

definition greater-mask where
  greater-mask  $\pi'$   $\pi$   $\longleftrightarrow$  ( $\exists$  r.  $\pi' =$  add-masks  $\pi$  r)

fun uni-mask where
  uni-mask hl p = empty-mask(hl := p)

fun valid-mask :: mask  $\Rightarrow$  bool where
  valid-mask  $\pi$   $\longleftrightarrow$  ( $\forall$  hl. pgte pwrite ( $\pi$  hl))  $\wedge$  ( $\forall$  f.  $\pi$  (null, f) = pnone)

definition valid-null :: mask  $\Rightarrow$  bool where
  valid-null  $\pi$   $\longleftrightarrow$  ( $\forall$  f.  $\pi$  (null, f) = pnone)

definition equal-on-mask where
  equal-on-mask  $\pi$  h h'  $\longleftrightarrow$  ( $\forall$  hl. ppos ( $\pi$  hl)  $\longrightarrow$  h hl = h' hl)

definition equal-on-bmask where
  equal-on-bmask  $\pi$  h h'  $\longleftrightarrow$  ( $\forall$  hl.  $\pi$  hl  $\longrightarrow$  h hl = h' hl)

definition big-add-masks where
  big-add-masks  $\Pi$   $\Pi'$  h = add-masks ( $\Pi$  h) ( $\Pi'$  h)

definition big-greater-mask where

```

```

big-greater-mask  $\Pi \Pi' \longleftrightarrow (\forall h. \text{greater-mask} (\Pi h) (\Pi' h))$ 

definition greater-bmask where
greater-bmask  $H H' \longleftrightarrow (\forall h. H' h \longrightarrow H h)$ 

definition update-dm where
update-dm  $dm \pi \pi' hl \longleftrightarrow (dm hl \vee \text{pgt} (\pi hl) (\pi' hl))$ 

fun pre-get-m where pre-get-m  $\varphi = \text{fst } \varphi$ 
fun pre-get-h where pre-get-h  $\varphi = \text{snd } \varphi$ 
fun srm-acc where srm-acc  $\varphi hl p = (\text{rm-acc} (\text{pre-get-m } \varphi) hl p, \text{pre-get-h } \varphi)$ 

```

**datatype** *val* = *Bool* (*the-bool*: *bool*) | *Address* (*the-address*: *address*) | *Rat* (*the-rat*: *prat*)

**definition** *upper-bounded* :: *mask*  $\Rightarrow$  *prat*  $\Rightarrow$  *bool* **where**  
*upper-bounded*  $\pi p \longleftrightarrow (\forall hl. \text{pgte} p (\pi hl))$

### 1.2.2 Lemmas

**lemma** *ssubsetI*:  
**assumes**  $\bigwedge \pi h. (\pi, h) \in A \implies (\pi, h) \in B$   
**shows**  $A \subseteq B$   
*(proof)*

**lemma** *double-inclusion*:  
**assumes**  $A \subseteq B$   
**and**  $B \subseteq A$   
**shows**  $A = B$   
*(proof)*

**lemma** *add-masks-comm*:  
*add-masks*  $a b = \text{add-masks } b a$   
*(proof)*

**lemma** *add-masks-asso*:  
*add-masks* (*add-masks*  $a b$ )  $c = \text{add-masks } a (\text{add-masks } b c)$   
*(proof)*

**lemma** *minus-empty*:  
 $\pi = \text{add-masks } \pi \text{ empty-mask}$   
*(proof)*

**lemma** *add-acc-uni-mask*:  
*add-acc*  $\pi hl p = \text{add-masks } \pi (\text{uni-mask } hl p)$   
*(proof)*

**lemma** *add-masks-equiv-valid-null*:

*valid-null* (*add-masks* *a* *b*)  $\longleftrightarrow$  *valid-null* *a*  $\wedge$  *valid-null* *b*  
*(proof)*

**lemma** *valid-maskI*:  
**assumes**  $\bigwedge hl. \text{pgte } \text{pwrite}(\pi \ hl)$   
**and**  $\bigwedge f. \pi(\text{null}, f) = \text{pnone}$   
**shows** *valid-mask*  $\pi$   
*(proof)*

**lemma** *greater-mask-equiv-def*:  
*greater-mask*  $\pi' \ \pi \longleftrightarrow (\forall hl. \text{pgte}(\pi' \ hl) \ (\pi \ hl))$   
**(is**  $?A \longleftrightarrow ?B$   
*(proof)*

**lemma** *greater-maskI*:  
**assumes**  $\bigwedge hl. \text{pgte}(\pi' \ hl) \ (\pi \ hl)$   
**shows** *greater-mask*  $\pi' \ \pi$   
*(proof)*

**lemma** *greater-mask-properties*:  
*greater-mask*  $\pi \ \pi$   
*greater-mask* *a* *b*  $\wedge$  *greater-mask* *b* *c*  $\implies$  *greater-mask* *a* *c*  
*greater-mask*  $\pi' \ \pi \wedge \text{greater-mask } \pi \ \pi' \implies \pi = \pi'$   
*(proof)*

**lemma** *greater-mask-decomp*:  
**assumes** *greater-mask* *a* (*add-masks* *b* *c*)  
**shows**  $\exists a1 \ a2. \ a = \text{add-masks } a1 \ a2 \wedge \text{greater-mask } a1 \ b \wedge \text{greater-mask } a2 \ c$   
*(proof)*

**lemma** *valid-empty*:  
*valid-mask* *empty-mask*  
*(proof)*

**lemma** *upper-valid-aux*:  
**assumes** *valid-mask* *a*  
**and** *a* = *add-masks* *b* *c*  
**shows** *valid-mask* *b*  
*(proof)*

**lemma** *upper-valid*:  
**assumes** *valid-mask* *a*  
**and** *a* = *add-masks* *b* *c*  
**shows** *valid-mask* *b*  $\wedge$  *valid-mask* *c*  
*(proof)*

**lemma** *equal-on-bmaskI*:  
**assumes**  $\bigwedge hl. \pi \ hl \implies h \ hl = h' \ hl$   
**shows** *equal-on-bmask*  $\pi \ h \ h'$

$\langle proof \rangle$

**lemma** *big-add-greater*:  
  *big-greater-mask* (*big-add-masks* *A* *B*) *B*  
   $\langle proof \rangle$

**lemma** *big-greater-iff*:  
  *big-greater-mask* *A* *B*  $\implies (\exists C. A = \text{big-add-masks } B C)$   
   $\langle proof \rangle$

**lemma** *big-add-masks-asso*:  
  *big-add-masks* *A* (*big-add-masks* *B* *C*) = *big-add-masks* (*big-add-masks* *A* *B*) *C*  
   $\langle proof \rangle$

**lemma** *big-add-mask-neutral*:  
  *big-add-masks*  $\Pi$  ( $\lambda\_. \text{empty-mask}$ ) =  $\Pi$   
   $\langle proof \rangle$

**lemma** *sym-equal-on-mask*:  
  *equal-on-mask*  $\pi$  *a* *b*  $\longleftrightarrow \text{equal-on-mask } \pi b a$   
   $\langle proof \rangle$

**lemma** *greater-mask-uni-equiv*:  
  *greater-mask*  $\pi$  (*uni-mask* *hl r*)  $\longleftrightarrow \text{pgte } (\pi hl) r$   
   $\langle proof \rangle$

**lemma** *greater-mask-uniI*:  
  **assumes** *pgte* ( $\pi hl$ ) *r*  
  **shows** *greater-mask*  $\pi$  (*uni-mask* *hl r*)  
   $\langle proof \rangle$

**lemma** *greater-bmask-refl*:  
  *greater-bmask* *H H*  
   $\langle proof \rangle$

**lemma** *greater-bmask-trans*:  
  **assumes** *greater-bmask* *A B*  
    **and** *greater-bmask* *B C*  
    **shows** *greater-bmask* *A C*  
   $\langle proof \rangle$

**lemma** *update-dm-same*:  
  *update-dm* *dm*  $\pi \pi = dm$   
   $\langle proof \rangle$

**lemma** *update-trans*:  
  **assumes** *greater-mask*  $\pi \pi'$   
    **and** *greater-mask*  $\pi' \pi''$   
    **shows** *update-dm* (*update-dm* *dm*  $\pi \pi')$   $\pi' \pi'' = \text{update-dm } dm \pi \pi''$

$\langle proof \rangle$

**lemma** *equal-on-bmask-greater*:  
  **assumes** *equal-on-bmask*  $\pi' h h'$   
    **and** *greater-bmask*  $\pi' \pi$   
  **shows** *equal-on-bmask*  $\pi h h'$   
 $\langle proof \rangle$

**lemma** *update-dm-equal-bmask*:  
  **assumes**  $\pi = add\text{-masks } \pi' m$   
  **shows** *equal-on-bmask* (*update-dm dm*  $\pi \pi')$   $h' h \longleftrightarrow equal\text{-on-mask } m h h' \wedge$   
*equal-on-bmask*  $dm h h'$   
 $\langle proof \rangle$

**lemma** *const-sum-mask-greater*:  
  **assumes** *add-masks*  $a b = add\text{-masks } c d$   
    **and** *greater-mask*  $a c$   
  **shows** *greater-mask*  $d b$   
 $\langle proof \rangle$

**lemma** *add-masks-cancellative*:  
  **assumes** *add-masks*  $b c = add\text{-masks } b d$   
  **shows**  $c = d$   
 $\langle proof \rangle$

**lemma** *equal-on-maskI*:  
  **assumes**  $\bigwedge hl. ppos(\pi hl) \implies h hl = h' hl$   
  **shows** *equal-on-mask*  $\pi h h'$   
 $\langle proof \rangle$

**lemma** *greater-equal-on-mask*:  
  **assumes** *equal-on-mask*  $\pi' h h'$   
    **and** *greater-mask*  $\pi' \pi$   
  **shows** *equal-on-mask*  $\pi h h'$   
 $\langle proof \rangle$

**lemma** *equal-on-mask-sum*:  
  *equal-on-mask*  $\pi_1 h h' \wedge equal\text{-on-mask } \pi_2 h h' \longleftrightarrow equal\text{-on-mask } (add\text{-masks }$   
 $\pi_1 \pi_2) h h'$   
 $\langle proof \rangle$

**lemma** *valid-larger-mask*:  
  *valid-mask*  $\pi \longleftrightarrow greater\text{-mask } full\text{-mask } \pi$   
 $\langle proof \rangle$

**lemma** *valid-mask-full-mask*:  
  *valid-mask* *full-mask*  
 $\langle proof \rangle$

```

lemma mult-greater:
  assumes greater-mask a b
  shows greater-mask (multiply-mask p a) (multiply-mask p b)
  ⟨proof⟩

lemma mult-write-mask:
  multiply-mask pwrite π = π
  ⟨proof⟩

lemma mult-distr-masks:
  multiply-mask a (add-masks b c) = add-masks (multiply-mask a b) (multiply-mask
  a c)
  ⟨proof⟩

lemma mult-add-states:
  multiply-mask (padd a b) π = add-masks (multiply-mask a π) (multiply-mask b
  π)
  ⟨proof⟩

lemma upper-boundedI:
  assumes  $\bigwedge hl. \text{pgte } p (\pi \text{ } hl)$ 
  shows upper-bounded π p
  ⟨proof⟩

lemma upper-bounded-smaller:
  assumes upper-bounded π a
  shows upper-bounded (multiply-mask p π) (pmult p a)
  ⟨proof⟩

lemma upper-bounded-bigger:
  assumes upper-bounded π a
  and pgte b a
  shows upper-bounded π b
  ⟨proof⟩

lemma valid-mult:
  assumes valid-mask π
  and pgte purite p
  shows valid-mask (multiply-mask p π)
  ⟨proof⟩

lemma valid-sum:
  assumes valid-mask a
  and valid-mask b
  and upper-bounded a ma
  and upper-bounded b mb
  and pgte pwrite (padd ma mb)
  shows valid-mask (add-masks a b)

```

```

    and upper-bounded (add-masks a b) (padd ma mb)
⟨proof⟩

lemma valid-multiply:
assumes valid-mask a
and upper-bounded a ma
and pgte pwrite (pmult ma p)
shows valid-mask (multiply-mask p a)
⟨proof⟩

lemma greater-mult:
assumes greater-mask a b
shows greater-mask (multiply-mask p a) (multiply-mask p b)
⟨proof⟩

end

```

### 1.3 Partial heaps: Partial maps from heap location to values

```

theory PartialHeapSA
imports Mask Package-logic.PackageLogic
begin

```

#### 1.3.1 Definitions

```

type-synonym heap = heap-loc → val
type-synonym pre-state = mask × heap

definition valid-heap :: mask ⇒ heap ⇒ bool where
valid-heap π h ↔ (∀ hl. ppos (π hl) → h hl ≠ None)

fun valid-state :: pre-state ⇒ bool where
valid-state (π, h) ↔ valid-mask π ∧ valid-heap π h

lemma valid-stateI:
assumes valid-mask π
and ∀hl. ppos (π hl) ==> h hl ≠ None
shows valid-state (π, h)
⟨proof⟩

definition empty-heap where empty-heap hl = None

lemma valid-pre-unit:
valid-state (empty-mask, empty-heap)
⟨proof⟩

typedef state = { φ |φ. valid-state φ }
⟨proof⟩

fun get-m :: state ⇒ mask where get-m a = fst (Rep-state a)

```

```

fun get-h :: state  $\Rightarrow$  heap where get-h a = snd (Rep-state a)

fun compatible-options where
  compatible-options (Some a) (Some b)  $\longleftrightarrow$  a = b
  | compatible-options - -  $\longleftrightarrow$  True

definition compatible-heaps :: heap  $\Rightarrow$  heap  $\Rightarrow$  bool where
  compatible-heaps h h'  $\longleftrightarrow$  ( $\forall$  hl. compatible-options (h hl) (h' hl))

definition compatible :: pre-state  $\Rightarrow$  pre-state  $\Rightarrow$  bool where
  compatible  $\varphi$   $\varphi'$   $\longleftrightarrow$  compatible-heaps (snd  $\varphi$ ) (snd  $\varphi'$ )  $\wedge$  valid-mask (add-masks
  (fst  $\varphi$ ) (fst  $\varphi'$ ))

fun add-states :: pre-state  $\Rightarrow$  pre-state  $\Rightarrow$  pre-state where
  add-states ( $\pi$ , h) ( $\pi'$ , h') = (add-masks  $\pi$   $\pi'$ , h ++ h')

definition larger-heap where
  larger-heap h' h  $\longleftrightarrow$  ( $\forall$  hl x. h hl = Some x  $\longrightarrow$  h' hl = Some x)

definition unit :: state where
  unit = Abs-state (empty-mask, empty-heap)

definition plus :: state  $\Rightarrow$  state  $\rightarrow$  state (infixl  $\langle \oplus \rangle$  63) where
  a  $\oplus$  b = (if compatible (Rep-state a) (Rep-state b) then Some (Abs-state (add-states
  (Rep-state a) (Rep-state b))) else None)

definition core :: state  $\Rightarrow$  state ( $\langle \mid \rangle$ ) where
  core  $\varphi$  = Abs-state (empty-mask, get-h  $\varphi$ )

definition stable :: state  $\Rightarrow$  bool where
  stable  $\varphi$   $\longleftrightarrow$  ( $\forall$  hl. ppos (get-m  $\varphi$  hl)  $\longleftrightarrow$  get-h  $\varphi$  hl  $\neq$  None)

```

### 1.3.2 Lemmas

```

lemma valid-heapI:
  assumes  $\bigwedge$  hl. ppos ( $\pi$  hl)  $\Longrightarrow$  h hl  $\neq$  None
  shows valid-heap  $\pi$  h
  ⟨proof⟩

lemma valid-state-decompose:
  assumes valid-state (add-masks a b, h)
  shows valid-state (a, h)
  ⟨proof⟩

lemma compatible-heapsI:
  assumes  $\bigwedge$  hl a b. h hl = Some a  $\Longrightarrow$  h' hl = Some b  $\Longrightarrow$  a = b
  shows compatible-heaps h h'
  ⟨proof⟩

```

```

lemma compatibleI-old:
  assumes  $\bigwedge hl\ x\ y.\ snd\ \varphi\ hl = Some\ x \wedge snd\ \varphi'\ hl = Some\ y \implies x = y$ 
    and valid-mask (add-masks (fst  $\varphi$ ) (fst  $\varphi'$ ))
  shows compatible  $\varphi\ \varphi'$ 
   $\langle proof \rangle$ 

lemma larger-heap-anti:
  assumes larger-heap a b
    and larger-heap b a
  shows a = b
   $\langle proof \rangle$ 

lemma larger-heapI:
  assumes  $\bigwedge hl\ x.\ h\ hl = Some\ x \implies h'\ hl = Some\ x$ 
  shows larger-heap h' h
   $\langle proof \rangle$ 

lemma larger-heap-refl:
  larger-heap h h
   $\langle proof \rangle$ 

lemma compatible-heaps-comm:
  assumes compatible-heaps a b
  shows a ++ b = b ++ a
   $\langle proof \rangle$ 

lemma larger-heaps-sum-ineq:
  assumes larger-heap a' a
    and larger-heap b' b
    and compatible-heaps a' b'
  shows larger-heap (a' ++ b') (a ++ b)
   $\langle proof \rangle$ 

lemma larger-heap-trans:
  assumes larger-heap a b
    and larger-heap b c
  shows larger-heap a c
   $\langle proof \rangle$ 

lemma larger-heap-comp:
  assumes larger-heap a b
    and compatible-heaps a c
  shows compatible-heaps b c
   $\langle proof \rangle$ 

lemma larger-heap-plus:
  assumes larger-heap a b
    and larger-heap a c
  shows larger-heap a (b ++ c)

```

$\langle proof \rangle$

**lemma** compatible-heaps-sum:  
  **assumes** compatible-heaps a b  
    **and** compatible-heaps a c  
  **shows** compatible-heaps a (b ++ c)  
 $\langle proof \rangle$

**lemma** larger-compatible-sum-heaps:  
  **assumes** larger-heap a x  
    **and** larger-heap b y  
    **and** compatible-heaps a b  
  **shows** compatible-heaps x y  
 $\langle proof \rangle$

**lemma** get-h-m:  
  Rep-state x = (get-m x, get-h x)  $\langle proof \rangle$

**lemma** get-pre:  
  get-h x = snd (Rep-state x)  
  get-m x = fst (Rep-state x)  
 $\langle proof \rangle$

**lemma** plus-ab-defined:  
   $\varphi \oplus \varphi' \neq \text{None} \longleftrightarrow \text{compatible-heaps}(\text{get-h } \varphi)(\text{get-h } \varphi') \wedge \text{valid-mask}(\text{add-masks}(\text{get-m } \varphi)(\text{get-m } \varphi'))$   
  (**is** ?A  $\longleftrightarrow$  ?B)  
 $\langle proof \rangle$

**lemma** plus-charact:  
  **assumes** a  $\oplus$  b = Some x  
  **shows** get-m x = add-masks (get-m a) (get-m b)  
    **and** get-h x = (get-h a) ++ (get-h b)  
 $\langle proof \rangle$

**lemma** commutative:  
  a  $\oplus$  b = b  $\oplus$  a  
 $\langle proof \rangle$

**lemma** asso1:  
  **assumes** a  $\oplus$  b = Some ab  $\wedge$  b  $\oplus$  c = Some bc  
  **shows** ab  $\oplus$  c = a  $\oplus$  bc  
 $\langle proof \rangle$

**lemma** asso2:  
  **assumes** a  $\oplus$  b = Some ab  $\wedge$  b  $\oplus$  c = None  
  **shows** ab  $\oplus$  c = None  
 $\langle proof \rangle$

```

lemma core-defined:
  get-h  $|\varphi| = \text{get-h } \varphi$ 
  get-m  $|\varphi| = \text{empty-mask}$ 
   $\langle\text{proof}\rangle$ 

lemma state-ext:
  assumes get-h  $a = \text{get-h } b$ 
  and get-m  $a = \text{get-m } b$ 
  shows  $a = b$ 
   $\langle\text{proof}\rangle$ 

lemma core-is-smaller:
  Some  $x = x \oplus |x|$ 
   $\langle\text{proof}\rangle$ 

lemma core-is-pure:
  Some  $|x| = |x| \oplus |x|$ 
   $\langle\text{proof}\rangle$ 

lemma core-sum:
  assumes Some  $c = a \oplus b$ 
  shows Some  $|c| = |a| \oplus |b|$ 
   $\langle\text{proof}\rangle$ 

lemma core-max:
  assumes Some  $x = x \oplus c$ 
  shows  $\exists r. \text{Some } |x| = c \oplus r$ 
   $\langle\text{proof}\rangle$ 

lemma positivity:
  assumes  $a \oplus b = \text{Some } c$ 
  and Some  $c = c \oplus c$ 
  shows Some  $a = a \oplus a$ 
   $\langle\text{proof}\rangle$ 

lemma cancellative:
  assumes Some  $a = b \oplus x$ 
  and Some  $a = b \oplus y$ 
  and  $|x| = |y|$ 
  shows  $x = y$ 
   $\langle\text{proof}\rangle$ 

lemma unit-charact:
  get-h unit = empty-heap
  get-m unit = empty-mask
   $\langle\text{proof}\rangle$ 

lemma empty-heap-neutral:
   $a ++ \text{empty-heap} = a$ 

```

```

⟨proof⟩

lemma unit-neutral:
  Some  $a = a \oplus \text{unit}$ 
⟨proof⟩

lemma stableI:
  assumes  $\bigwedge hl. ppos(\text{get-m } \varphi \text{ } hl) \longleftrightarrow \text{get-h } \varphi \text{ } hl \neq \text{None}$ 
  shows stable  $\varphi$ 
⟨proof⟩

lemma stable-unit:
  stable unit
⟨proof⟩

lemma stable-sum:
  assumes stable  $a$ 
    and stable  $b$ 
    and Some  $x = a \oplus b$ 
  shows stable  $x$ 
⟨proof⟩

lemma multiply-valid:
  assumes pgte pwrite  $p$ 
  shows valid-state (multiply-mask  $p$  (get-m  $\varphi$ ), get-h  $\varphi$ )
⟨proof⟩

```

#### 1.4 This state model corresponds to a separation algebra

**global-interpretation**  $\text{PartialSA}$ : package-logic plus core unit stable  
**defines** greater (infixl  $\langle \succeq \rangle$  50) =  $\text{PartialSA}.\text{greater}$   
 and add-set (infixl  $\langle \otimes \rangle$  60) =  $\text{PartialSA}.\text{add-set}$   
 and defined (infixl  $\langle |\#| \rangle$  60) =  $\text{PartialSA}.\text{defined}$   
 and greater-set (infixl  $\langle |\gg| \rangle$  50) =  $\text{PartialSA}.\text{greater-set}$   
 and minus (infixl  $\langle |\ominus| \rangle$  60) =  $\text{PartialSA}.\text{minus}$   
⟨proof⟩

**lemma** greaterI:
 assumes larger-heap (get-h  $a$ ) (get-h  $b$ )
 and greater-mask (get-m  $a$ ) (get-m  $b$ )
 shows  $a \succeq b$ 
⟨proof⟩

**lemma** larger-implies-greater-mask-hl:
 assumes  $a \succeq b$ 
 shows pgte (get-m  $a$   $hl$ ) (get-m  $b$   $hl$ )
⟨proof⟩

```

lemma larger-implies-larger-heap:
  assumes  $a \succeq b$ 
  shows larger-heap (get-h a) (get-h b)
  ⟨proof⟩

lemma compatibleI:
  assumes compatible-heaps (get-h a) (get-h b)
  and valid-mask (add-masks (get-m a) (get-m b))
  shows  $a \parallel b$ 
  ⟨proof⟩

end

```

## 2 Combinable Magic Wands

Note that, in this theory, assertions are represented as semantic assertions, i.e., as the set of states in which they hold.

```

theory CombinableWands
  imports PartialHeapSA
begin

```

### 2.1 Definitions

**type-synonym**  $\text{sem-assertion} = \text{state set}$

```

fun multiply :: prat  $\Rightarrow$  state  $\Rightarrow$  state where
  multiply  $p \varphi = \text{Abs-state}(\text{multiply-mask } p (\text{get-m } \varphi), \text{get-h } \varphi)$ 

```

Because we work in an intuitionistic setting, a fraction of an assertion is defined using the upper-closure operator.

```

fun multiply-sem-assertion :: prat  $\Rightarrow$  sem-assertion  $\Rightarrow$  sem-assertion where
  multiply-sem-assertion  $p P = \text{PartialSA.upper-closure}(\text{multiply } p ` P)$ 

```

```

definition combinable :: sem-assertion  $\Rightarrow$  bool where
  combinable  $P \longleftrightarrow (\forall \alpha \beta. \text{ppos } \alpha \wedge \text{ppos } \beta \wedge \text{pgte } (\text{padd } \alpha \beta) \longrightarrow$ 
   $(\text{multiply-sem-assertion } \alpha P) \otimes (\text{multiply-sem-assertion } \beta P) \subseteq \text{multiply-sem-assertion}$ 
   $(\text{padd } \alpha \beta) P)$ 

```

```

definition scaled where
  scaled  $\varphi = \{ \text{multiply } p \varphi \mid p. \text{ppos } p \wedge \text{pgte } \text{pwrite } p \}$ 

```

```

definition comp-min-mask :: mask  $\Rightarrow$  (mask  $\Rightarrow$  mask) where
  comp-min-mask  $b a hl = \text{pmin}(a hl) (\text{comp-one}(b hl))$ 

```

```

definition scalable where
  scalable  $w a \longleftrightarrow (\forall \varphi \in \text{scaled } w. \neg a \parallel \varphi)$ 

```

```

definition R where

```

$R a w = (\text{if scalable } w a \text{ then } w \text{ else Abs-state (comp-min-mask (get-m } a) (\text{get-m } w), \text{get-h } w))$

**definition cwand where**

$cwand A B = \{ w \mid w. \forall a x. a \in A \wedge \text{Some } x = R a w \oplus a \rightarrow x \in B \}$

**definition wand :: sem-assertion  $\Rightarrow$  sem-assertion  $\Rightarrow$  sem-assertion where**

$wand A B = \{ w \mid w. \forall a x. a \in A \wedge \text{Some } x = w \oplus a \rightarrow x \in B \}$

**definition intuitionistic where**

$\text{intuitionistic } A \longleftrightarrow (\forall a b. a \succeq b \wedge b \in A \rightarrow a \in A)$

**definition binary-mask :: mask  $\Rightarrow$  mask where**

$\text{binary-mask } \pi l = (\text{if } \pi l = \text{pwrite} \text{ then pwrite else pnone})$

**definition binary :: sem-assertion  $\Rightarrow$  bool where**

$\text{binary } A \longleftrightarrow (\forall \varphi \in A. \text{Abs-state (binary-mask (get-m } \varphi), \text{get-h } \varphi) \in A)$

## 2.2 Lemmas

**lemma wand-equiv-def:**

$wand A B = \{ \varphi \mid \varphi. A \otimes \{\varphi\} \subseteq B \}$

$\langle \text{proof} \rangle$

**lemma w-in-scaled:**

$w \in \text{scaled } w$

$\langle \text{proof} \rangle$

**lemma non-scalable-instantiate:**

**assumes**  $\neg \text{scalable } w a$

**shows**  $\exists p. \text{ppos } p \wedge \text{pgte } p \text{write } p \wedge a \mid\# \text{ multiply } p w$

$\langle \text{proof} \rangle$

**lemma compatible-same-mask:**

**assumes**  $\text{valid-mask (add-masks } a w)$

**shows**  $w = \text{comp-min-mask } a w$

$\langle \text{proof} \rangle$

**lemma R-smaller:**

$w \succeq R a w$

$\langle \text{proof} \rangle$

**lemma R-compatible-same:**

**assumes**  $a \mid\# w$

**shows**  $R a w = w$

$\langle \text{proof} \rangle$

**lemma in-cwand:**

**assumes**  $\bigwedge a x. a \in A \wedge \text{Some } x = R a w \oplus a \implies x \in B$

**shows**  $w \in \text{wand } A \text{ } B$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{wandI}:$   
**assumes**  $\bigwedge a x. a \in A \wedge \text{Some } x = a \oplus w \implies x \in B$   
**shows**  $w \in \text{wand } A \text{ } B$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{non-scalable-R-charact}:$   
**assumes**  $\neg \text{scalable } w a$   
**shows**  $\text{get-m } (R a w) = \text{comp-min-mask } (\text{get-m } a) (\text{get-m } w) \wedge \text{get-h } (R a w) =$   
 $\text{get-h } w$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{valid-bin}:$   
 $\text{valid-state } (\text{binary-mask } (\text{get-m } a), \text{get-h } a)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{in-multiply-sem}:$   
**assumes**  $x \in \text{multiply-sem-assertion } p A$   
**shows**  $\exists a \in A. x \succeq \text{multiply } p a$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{get-h-multiply}:$   
**assumes**  $\text{pgte } \text{pwrite } p$   
**shows**  $\text{get-h } (\text{multiply } p x) = \text{get-h } x$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{in-multiply-refl}:$   
**assumes**  $x \in A$   
**shows**  $\text{multiply } p x \in \text{multiply-sem-assertion } p A$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{get-m-smaller}:$   
**assumes**  $\text{pgte } \text{pwrite } p$   
**shows**  $\text{get-m } (\text{multiply } p a) \text{ hl} = \text{pmult } p (\text{get-m } a \text{ hl})$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{get-m-smaller-mask}:$   
**assumes**  $\text{pgte } \text{pwrite } p$   
**shows**  $\text{get-m } (\text{multiply } p a) = \text{multiply-mask } p (\text{get-m } a)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{multiply-order}:$   
**assumes**  $\text{pgte } \text{pwrite } p$   
**and**  $a \succeq b$   
**shows**  $\text{multiply } p a \succeq \text{multiply } p b$   
 $\langle \text{proof} \rangle$

```

lemma multiply-twice:
  assumes pgte pwrite a  $\wedge$  pgte pwrite b
  shows multiply a (multiply b x) = multiply (pmult a b) x
   $\langle proof \rangle$ 

lemma valid-mask-add-comp-min:
  assumes valid-mask a
    and valid-mask b
  shows valid-mask (add-masks (comp-min-mask b a) b)
   $\langle proof \rangle$ 

```

### 2.3 The combinable wand is stronger than the original wand

```

lemma cwand-stronger:
  cwand A B  $\subseteq$  wand A B
   $\langle proof \rangle$ 

```

### 2.4 The combinable wand is the same as the original wand when the left-hand side is binary

```

lemma binary-same:
  assumes binary A
    and intuitionistic B
  shows wand A B  $\subseteq$  cwand A B
   $\langle proof \rangle$ 

```

### 2.5 The combinable wand is combinable

```

lemma combinableI:
  assumes  $\bigwedge a b. \text{ppos } a \wedge \text{ppos } b \wedge \text{padd } a b = \text{pwrite} \implies (\text{multiply-sem-assertion } a (\text{cwand } A B)) \otimes (\text{multiply-sem-assertion } b (\text{cwand } A B)) \subseteq \text{cwand } A B$ 
  shows combinable (cwand A B)
   $\langle proof \rangle$ 

```

```

lemma combinable-cwand:
  assumes combinable B
    and intuitionistic B
  shows combinable (cwand A B)
   $\langle proof \rangle$ 

```

### 2.6 Theorems

The following theorem is crucial to use the package logic [4] to automatically compute footprints of combinable wands.

```

theorem R-mono-transformer:
  PartialSA.mono-transformer (R a)
   $\langle proof \rangle$ 

```

```

theorem properties-of-combinable-wands:

```

```

assumes intuitionistic B
shows combinable B ==> combinable (cwand A B)
  and cwand A B ⊆ wand A B
  and binary A ==> cwand A B = wand A B
⟨proof⟩

```

**end**

## References

- [1] J. Boyland. Checking interference with fractional permissions. In R. Cousot, editor, *Static Analysis (SAS)*, pages 55–72, 2003.
- [2] J. T. Boyland. Semantics of fractional permissions with nesting. *Transactions on Programming Languages and Systems (TOPLAS)*, 32(6):22:1–22:33, 2010.
- [3] 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.
- [4] 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.
- [5] B. Jacobs and F. Piessens. Expressive modular fine-grained concurrency specification. In *Principles of Programming Languages (POPL)*, pages 271–282, 2011.
- [6] X.-B. Le and A. Hobor. Logical reasoning for disjoint permissions. In A. Ahmed, editor, *European Symposium on Programming (ESOP)*, 2018.