

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

Thibault Dardinier

June 17, 2024

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 σ iff there exists a state σ_A in which A holds and σ is obtained from σ_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

1	State Model with Fractional Permissions	2
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
2	Combinable Magic Wands	20
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 is 1 <proof>
```

```
lift-definition pnone :: prat is 0 <proof>
```

```
lift-definition half :: prat is 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 $half-between-0-1:$

$ppos\ half \wedge pgt\ pwrite\ half$
 $\langle proof \rangle$

lemma $pgt-implies-pgte:$

assumes $pgt\ a\ b$
shows $pgte\ a\ b$

<proof>

lemma *half-plus-half*:
padd half half = pwrite
<proof>

lemma *padd-comm*:
padd a b = padd b a
<proof>

lemma *padd-asso*:
padd (padd a b) c = padd a (padd b c)
<proof>

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

lemma *pgte-antisym*:
assumes *pgte a b*
and *pgte b a*
shows *a = b*
<proof>

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

lemma *padd-cancellative*:
assumes *a = padd x b*
and *a = padd y b*
shows *x = y*
<proof>

lemma *not-pgte-charact*:
 $\neg pgte a b \longleftrightarrow pgt b a$
<proof>

lemma *pgte-pgt*:
assumes *pgt a b*
and *pgte c d*
shows *pgt (padd a c) (padd b d)*
<proof>

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*:

$pnone = padd\ a\ b \iff a = pnone \wedge b = pnone$
<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 $pmult\ p\ (pinv\ p) = pwrite$
 $\langle proof \rangle$

lemma $pinv-pwrite$:
 $pinv\ pwrite = pwrite$
 $\langle proof \rangle$

lemma $pmult-ppos$:
assumes $ppos\ a$
and $ppos\ b$
shows $ppos\ (pmult\ a\ b)$
 $\langle proof \rangle$

lemma $ppos-inv$:
assumes $ppos\ p$
shows $ppos\ (pinv\ p)$
 $\langle proof \rangle$

lemma $pmin-pmax$:
assumes $pgte\ x\ (pmin\ a\ b)$
shows $x = pmin\ (pmax\ x\ a)\ (pmax\ x\ b)$
 $\langle proof \rangle$

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

lemma $padd-comp-one$:
assumes $pgte\ pwrite\ x$
shows $padd\ x\ (comp-one\ x) = pwrite$
 $\langle proof \rangle$

lemma $ppos-eq-pnone$:
 $ppos\ p \longleftrightarrow p \neq pnone$
 $\langle proof \rangle$

lemma $pmult-order$:
assumes $pgte\ a\ b$
shows $pgte\ (pmult\ p\ a)\ (pmult\ b\ p)$
 $\langle proof \rangle$

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

shows $pgte\ pwrite\ (pmult\ a\ b)$
 $\langle proof \rangle$

lemma *pmult-pdiv-cancel*:
assumes $ppos\ a$
shows $pmult\ a\ (pdiv\ x\ a) = x$
 $\langle proof \rangle$

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

lemma *pdiv-smaller*:
assumes $pgte\ a\ b$
and $ppos\ a$
shows $pgte\ pwrite\ (pdiv\ b\ a)$
 $\langle proof \rangle$

lemma *sum-coeff*:
assumes $ppos\ a$
and $ppos\ b$
shows $padd\ (pdiv\ a\ (padd\ a\ b))\ (pdiv\ b\ (padd\ a\ b)) = pwrite$
 $\langle proof \rangle$

lemma *padd-one-ineq-sum*:
assumes $padd\ a\ b = pwrite$
and $pgte\ x\ aa$
and $pgte\ x\ bb$
shows $pgte\ x\ (padd\ (pmult\ a\ aa)\ (pmult\ b\ bb))$
 $\langle proof \rangle$

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 $heap-loc = address \times field$

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 $\bigwedge hl. \text{pgte } pwrite (\pi \ hl)$
 and $\bigwedge f. \pi (\text{null}, f) = \text{pnone}$
 shows *valid-mask* π
(*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 $a \ b \wedge \text{greater-mask } b \ c \implies \text{greater-mask } a \ c$
 greater-mask $\pi' \pi \wedge \text{greater-mask } \pi \ \pi' \implies \pi = \pi'$
(*proof*)

lemma *greater-mask-decomp*:
 assumes *greater-mask* $a (\text{add-masks } b \ c)$
 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 $a = \text{add-masks } b \ c$
 shows *valid-mask* b
(*proof*)

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

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

<proof>

lemma *big-add-greater:*

big-greater-mask (big-add-masks A B) B

<proof>

lemma *big-greater-iff:*

big-greater-mask A B \implies ($\exists C. A = \text{big-add-masks } B C$)

<proof>

lemma *big-add-masks-asso:*

big-add-masks A (big-add-masks B C) = big-add-masks (big-add-masks A B) C

<proof>

lemma *big-add-mask-neutral:*

big-add-masks Π ($\lambda \cdot \text{empty-mask}$) = Π

<proof>

lemma *sym-equal-on-mask:*

equal-on-mask $\pi a b \iff \text{equal-on-mask } \pi b a$

<proof>

lemma *greater-mask-uni-equiv:*

greater-mask π (uni-mask hl r) \iff pgte (π hl) r

<proof>

lemma *greater-mask-uniI:*

assumes *pgte (π hl) r*

shows *greater-mask π (uni-mask hl r)*

<proof>

lemma *greater-bmask-refl:*

greater-bmask H H

<proof>

lemma *greater-bmask-trans:*

assumes *greater-bmask A B*

and *greater-bmask B C*

shows *greater-bmask A C*

<proof>

lemma *update-dm-same:*

update-dm dm π π = dm

<proof>

lemma *update-trans:*

assumes *greater-mask π π'*

and *greater-mask π' π''*

shows *update-dm (update-dm dm π π') π' π'' = update-dm dm π π''*

<proof>

lemma *equal-on-bmask-greater*:
 assumes *equal-on-bmask* π' h h'
 and *greater-bmask* π' π
 shows *equal-on-bmask* π h h'
<proof>

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

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

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

lemma *equal-on-maskI*:
 assumes $\bigwedge hl. \text{ppos } (\pi hl) \implies h hl = h' hl$
 shows *equal-on-mask* $\pi h h'$
<proof>

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

lemma *equal-on-mask-sum*:
 equal-on-mask $\pi 1 h h' \wedge \text{equal-on-mask } \pi 2 h h' \longleftrightarrow \text{equal-on-mask } (\text{add-masks } \pi 1 \pi 2) h h'$
<proof>

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

lemma *valid-mask-full-mask*:
 valid-mask full-mask
<proof>

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 $\pi = \pi$

<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) $\pi = add-masks (multiply-mask a \pi) (multiply-mask b \pi)$

<proof>

lemma *upper-boundedI*:

assumes $\bigwedge hl. pgte p (\pi hl)$

shows *upper-bounded πp*

<proof>

lemma *upper-bounded-smaller*:

assumes *upper-bounded πa*

shows *upper-bounded (multiply-mask p π) (pmult p a)*

<proof>

lemma *upper-bounded-bigger*:

assumes *upper-bounded πa*

and *pgte b a*

shows *upper-bounded πb*

<proof>

lemma *valid-mult*:

assumes *valid-mask π*

and *pgte pwrite 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* \rightarrow *val*
type-synonym *pre-state* = *mask* \times *heap*

definition *valid-heap* :: *mask* \Rightarrow *heap* \Rightarrow *bool* **where**
valid-heap π h \longleftrightarrow (\forall hl. *ppos* (π hl) \longrightarrow h hl \neq *None*)

fun *valid-state* :: *pre-state* \Rightarrow *bool* **where**
valid-state (π , h) \longleftrightarrow *valid-mask* π \wedge *valid-heap* π h

lemma *valid-stateI*:
assumes *valid-mask* π
and \bigwedge hl. *ppos* (π hl) \Longrightarrow h hl \neq *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* \Rightarrow *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 φ φ' \longleftrightarrow *compatible-heaps* (*snd* φ) (*snd* φ') \wedge *valid-mask* (*add-masks*
 (*fst* φ) (*fst* φ'))

fun *add-states* :: *pre-state* \Rightarrow *pre-state* \Rightarrow *pre-state* **where**
 add-states (π , *h*) (π' , *h'*) = (*add-masks* π π' , *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** \oplus 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* (|-|) **where**
 core φ = *Abs-state* (*empty-mask*, *get-h* φ)

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

1.3.2 Lemmas

lemma *valid-heapI*:
 assumes \bigwedge *hl*. *ppos* (π *hl*) \implies *h* *hl* \neq *None*
 shows *valid-heap* π *h*
 \langle *proof* \rangle

lemma *valid-state-decompose*:
 assumes *valid-state* (*add-masks* *a* *b*, *h*)
 shows *valid-state* (*a*, *h*)
 \langle *proof* \rangle

lemma *compatible-heapsI*:
 assumes \bigwedge *hl* *a* *b*. *h* *hl* = *Some* *a* \implies *h'* *hl* = *Some* *b* \implies *a* = *b*
 shows *compatible-heaps* *h* *h'*
 \langle *proof* \rangle

lemma *compatibleI-old*:
assumes $\bigwedge hl\ x\ y. \text{snd } \varphi\ hl = \text{Some } x \wedge \text{snd } \varphi'\ hl = \text{Some } y \implies x = y$
and *valid-mask* (*add-masks* (*fst* φ) (*fst* φ'))
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 $\bigwedge hl\ x. h\ hl = \text{Some } x \implies h'\ hl = \text{Some } x$
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* $a\ (b\ ++\ c)$

<proof>

lemma *compatible-heaps-sum:*

assumes *compatible-heaps a b*
and *compatible-heaps a c*
shows *compatible-heaps a (b ++ c)*
<proof>

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*
<proof>

lemma *get-h-m:*

Rep-state x = (get-m x, get-h x) <proof>

lemma *get-pre:*

get-h x = snd (Rep-state x)
get-m x = fst (Rep-state x)
<proof>

lemma *plus-ab-defined:*

$\varphi \oplus \varphi' \neq \text{None} \longleftrightarrow \text{compatible-heaps (get-h } \varphi) (\text{get-h } \varphi') \wedge \text{valid-mask (add-masks (get-m } \varphi) (\text{get-m } \varphi'))$
(is ?A \longleftrightarrow ?B)
<proof>

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)*
<proof>

lemma *commutative:*

a \oplus b = b \oplus a
<proof>

lemma *asso1:*

assumes *a \oplus b = Some ab \wedge b \oplus c = Some bc*
shows *ab \oplus c = a \oplus bc*
<proof>

lemma *asso2:*

assumes *a \oplus b = Some ab \wedge b \oplus c = None*
shows *ab \oplus c = None*
<proof>

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 $\text{unit} = \text{empty-heap}$

get-m $\text{unit} = \text{empty-mask}$

$\langle \text{proof} \rangle$

lemma *empty-heap-neutral*:

$a ++ \text{empty-heap} = a$

<proof>

lemma *unit-neutral*:

Some a = a \oplus unit

<proof>

lemma *stableI*:

assumes $\bigwedge hl. ppos (get-m \varphi hl) \longleftrightarrow get-h \varphi hl \neq None$

shows *stable φ*

<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 φ), get-h φ)*

<proof>

1.4 This state model corresponds to a separation algebra

global-interpretation *PartialSA*: *package-logic plus core unit stable*

defines *greater (infixl \succeq 50) = PartialSA.greater*

and *add-set (infixl \otimes 60) = PartialSA.add-set*

and *defined (infixl $|\#|$ 60) = PartialSA.defined*

and *greater-set (infixl $|\gg|$ 50) = PartialSA.greater-set*

and *minus (infixl $|\ominus|$ 60) = PartialSA.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)
 \langle *proof* \rangle

lemma *compatibleI*:
assumes *compatible-heaps* (*get-h* a) (*get-h* b)
and *valid-mask* (*add-masks* (*get-m* a) (*get-m* b))
shows $a \mid\# \mid b$
 \langle *proof* \rangle

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 *sem-assertion* = *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 $P \iff (\forall \alpha \ \beta. \text{ppos } \alpha \wedge \text{ppos } \beta \wedge \text{pgte } p\text{write } (\text{padd } \alpha \ \beta) \longrightarrow (\text{multiply-sem-assertion } \alpha \ P) \otimes (\text{multiply-sem-assertion } \beta \ P) \subseteq \text{multiply-sem-assertion } (\text{padd } \alpha \ \beta) \ P)$

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

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 $w \ a \iff (\forall \varphi \in \text{scaled } w. \neg a \mid\# \mid \varphi)$

definition *R* **where**

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

definition *cwand where*

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

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

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

definition *intuitionistic where*

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

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

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

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

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

2.2 Lemmas

lemma *wand-equiv-def:*

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

<proof>

lemma *w-in-scaled:*

$w \in \text{scaled } w$

<proof>

lemma *non-scalable-instantiate:*

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

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

<proof>

lemma *compatible-same-mask:*

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

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

<proof>

lemma *R-smaller:*

$w \succeq R a w$

<proof>

lemma *R-compatible-same:*

assumes $a \mid\# \mid w$

shows $R a w = w$

<proof>

lemma *in-cwand:*

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

shows $w \in \text{cwand } A \ B$
<proof>

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

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

lemma *valid-bin*:
valid-state (*binary-mask* (*get-m* *a*), *get-h* *a*)
<proof>

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

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

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

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

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

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

lemma *multiply-twice*:
 assumes *pgte pwrite a* \wedge *pgte pwrite b*
 shows *multiply a (multiply b x) = multiply (pmult a b) x*
<proof>

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)*
<proof>

2.3 The combinable wand is stronger than the original wand

lemma *cwand-stronger*:
cwand A B \subseteq *wand A B*
<proof>

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*
<proof>

2.5 The combinable wand is combinable

lemma *combinableI*:
 assumes $\bigwedge a b. ppos a \wedge ppos b \wedge padd a b = pwrite \implies (multiply-sem-assertion a (cwand A B)) \otimes (multiply-sem-assertion b (cwand A B)) \subseteq cwand A B$
 shows *combinable (cwand A B)*
<proof>

lemma *combinable-cwand*:
 assumes *combinable B*
 and *intuitionistic B*
 shows *combinable (cwand A B)*
<proof>

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)
<proof>

theorem *properties-of-combinable-wands*:

assumes *intuitionistic B*
shows *combinable B* \implies *combinable (cwand A B)*
and *cwand A B* \subseteq *wand A B*
and *binary A* \implies *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.