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 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 postconditions of parallel branches when they terminate. Unfortunately, the magic wand assertion $A \mathrel{-\!\!*} B$, 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 9
1.2.1 Definitions 9
1.2.2 Lemmas 11
1.3 Partial heaps: Partial maps from heap location to values . . . 20

1.3.1 Definitions 20
1.3.2 Lemmas 21
1.4 This state model corresponds to a separation algebra 31

2 Combinable Magic Wands **34**
2.1 Definitions 34
2.2 Lemmas 35
2.3 The combinable wand is stronger than the original wand . . . 40
2.4 The combinable wand is the same as the original wand when the left-hand side is binary 40
2.5 The combinable wand is combinable 43
2.6 Theorems 54

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*} **by** *fastforce*

setup-lifting *type-definition-prat*

lift-definition *pwrite* :: *prat* **is** *1* **by** *simp*
lift-definition *pnone* :: *prat* **is** *0* **by** *simp*
lift-definition *half* :: *prat* **is** *1* / *2* **by** *fastforce*

lift-definition *pgte* :: *prat* ⇒ *prat* ⇒ *bool* **is** (≥) **done**
lift-definition *pgt* :: *prat* ⇒ *prat* ⇒ *bool* **is** (>) **done**

lift-definition *lt* :: *prat* ⇒ *prat* ⇒ *bool* **is** (<) **done**
lift-definition *ppos* :: *prat* ⇒ *bool* **is** λ*p*. *p* > *0* **done**

lift-definition *pmult* :: *prat* ⇒ *prat* ⇒ *prat* **is** (∗) **by** *simp*
lift-definition *padd* :: *prat* ⇒ *prat* ⇒ *prat* **is** (+) **by** *simp*

lift-definition *pdiv* :: *prat* ⇒ *prat* ⇒ *prat* **is** (/) **by** *simp*

lift-definition *pmin* :: *prat* ⇒ *prat* ⇒ *prat* **is** (*min*) **by** *simp*
lift-definition *pmax* :: *prat* ⇒ *prat* ⇒ *prat* **is** (*max*) **by** *simp*

1.1.2 Lemmas

lemma *pmin-comm*:
  pmin a b = *pmin b a*
  by (*metis Rep-prat-inverse min.commute pmin.rep-eq*)

lemma *pmin-greater*:
  pgte a (*pmin a b*)
  by (*simp add*: *pgte.rep-eq pmin.rep-eq*)

lemma *pmin-is*:
  assumes *pgte a b*
  shows *pmin a b* = *b*
  by (*metis Rep-prat-inject assms min-absorb2 pgte.rep-eq pmin.rep-eq*)

lemma *pmax-comm*:
  pmax a b = *pmax b a*
  by (*metis Rep-prat-inverse max.commute pmax.rep-eq*)

lemma *pmax-smaller*:
  pgte (*pmax a b*) *a*
  by (*simp add*: *pgte.rep-eq pmax.rep-eq*)

lemma *pmax-is*:
  assumes *pgte a b*
  shows *pmax a b* = *a*
  by (*metis Rep-prat-inject assms max.absorb-iff1 pgte.rep-eq pmax.rep-eq*)

lemma *pmax-is-smaller*:
  assumes *pgte x a*
    and *pgte x b*
   shows *pgte x* (*pmax a b*)
proof (*cases pgte a b*)
  case *True*
  then show *?thesis*
   by (*simp add*: *assms*(*1*) *pmax-is*)
next
  case *False*
  then show *?thesis*
   using *assms*(*2*) *pgte.rep-eq pmax.rep-eq* **by** *auto*
qed

lemma *half-between-0-1*:
ppos half ∧ *pgt pwrite half*
by (*simp add*: *half.rep-eq pgt.rep-eq ppos.rep-eq pwrite.rep-eq*)

lemma *pgt-implies-pgte*:
assumes *pgt a b*
shows *pgte a b*
by (*meson assms less-imp-le pgt.rep-eq pgte.rep-eq*)

lemma *half-plus-half*:
padd half half = *pwrite*
by (*metis Rep-prat-inject divide-less-eq-numeral1*(*1*) *dual-order.irrefl half.rep-eq less-divide-eq-numeral1*(*1*) *linorder-neqE-linordered-idom mult.right-neutral one-add-one padd.rep-eq pwrite.rep-eq ring-class.ring-distribs*(*1*))

lemma *padd-comm*:
padd a b = *padd b a*
by (*metis Rep-prat-inject add.commute padd.rep-eq*)

lemma *padd-asso*:
padd (*padd a b*) *c* = *padd a* (*padd b c*)
by (*metis Rep-prat-inverse group-cancel.add1 padd.rep-eq*)

lemma *p-greater-exists*:
pgte a b ⟷ (∃ *r*. *a* = *padd b r*)
proof (*rule iffI*)
show *pgte a b* ⟹ ∃ *r*. *a* = *padd b r*
proof −
assume *asm*: *pgte a b*
obtain *aa bb* **where** *aa* = *Rep-prat a bb* = *Rep-prat b*
by *simp*
then have *aa* ≥ *bb* **using** *asm*
using *pgte.rep-eq* **by** *fastforce*
then obtain *rr* **where** *rr* = *aa* − *bb*
by *simp*
then have *a* = *padd b* (*Abs-prat rr*)
by (*metis Abs-prat-inverse Rep-prat-inject* ‹*aa* = *Rep-prat a*› ‹*bb* = *Rep-prat b*› ‹*bb* ≤ *aa*› *add.commute diff-add-cancel diff-ge-0-iff-ge mem-Collect-eq padd.rep-eq*)
then show ∃ *r*. *a* = *padd b r* **by** *blast*
qed
show ∃ *r*. *a* = *padd b r* ⟹ *pgte a b*
using *Rep-prat padd.rep-eq pgte.rep-eq* **by** *force*
qed

lemma *pgte-antisym*:
assumes *pgte a b*

and *pgte b a*
shows $a = b$
by (*metis Rep-prat-inverse assms*(*1*) *assms*(*2*) *leD le-less pgte.rep-eq*)

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 −
obtain *aa bb cc* **where** $aa =$ *Rep-prat a* $bb =$ *Rep-prat b* $cc =$ *Rep-prat c*
by *simp*
then have $aa \geq bb + cc$
using *assms padd.rep-eq pgte.rep-eq* **by** *auto*
then obtain *x* **where** $aa = bb + x$ $x \geq cc$
by (*metis add.commute add-le-cancel-left diff-add-cancel*)
then show *?thesis*
by (*metis assms order-refl p-greater-exists padd-asso pgte.rep-eq*)
qed

lemma *padd-cancellative*:
assumes $a =$ *padd x b*
and $a =$ *padd y b*
shows $x = y$
by (*metis Rep-prat-inject add-le-cancel-right assms*(*1*) *assms*(*2*) *leD less-eq-rat-def padd.rep-eq*)

lemma *not-pgte-charact*:
$\neg$ *pgte a b* $\longleftrightarrow$ *pgt b a*
by (*meson not-less pgt.rep-eq pgte.rep-eq*)

lemma *pgte-pgt*:
assumes *pgt a b*
and *pgte c d*
shows *pgt* (*padd a c*) (*padd b d*)
using *assms*(*1*) *assms*(*2*) *padd.rep-eq pgt.rep-eq pgte.rep-eq* **by** *auto*

lemma *pmult-distr*:
pmult a (*padd b c*) $=$ *padd* (*pmult a b*) (*pmult a c*)
by (*metis Rep-prat-inject distrib-left padd.rep-eq pmult.rep-eq*)

lemma *pmult-comm*:
pmult a b $=$ *pmult b a*
by (*metis Rep-prat-inject mult.commute pmult.rep-eq*)

lemma *pmult-special*:
pmult pwrite x $= x$
pmult pnone x $=$ *pnone*
apply (*metis Rep-prat-inverse comm-monoid-mult-class.mult-1 pmult.rep-eq pwrite.rep-eq*)

by (*metis Rep-prat-inverse mult-zero-left pmult.rep-eq pnone.rep-eq*)

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 −
have (*Fract 1 2*) ∗ ((*Fract 1 1*) / (*Rep-prat p*)) = (*Fract 1 1*) / (*Rep-prat p* + *Rep-prat p*)
by (*metis* (*no-types*, *lifting*) *One-rat-def comm-monoid-mult-class.mult-1 divide-rat mult-2 mult-rat rat-number-expand*(*3*) *times-divide-times-eq*)
then show *?thesis*
by (*metis Rep-prat-inject half.rep-eq mult-2 mult-numeral-1-right numeral-One padd.rep-eq pdiv.rep-eq pinv-def pmult.rep-eq pwrite.rep-eq times-divide-times-eq*)
qed

lemma *ppos-mult*:
assumes *ppos a*
and *ppos b*
shows *ppos* (*pmult a b*)
using *assms*(*1*) *assms*(*2*) *pmult.rep-eq ppos.rep-eq* **by** *auto*

lemma *padd-zero*:
pnone = *padd a b* ⟷ *a* = *pnone* ∧ *b* = *pnone*
by (*metis Rep-prat Rep-prat-inject add.right-neutral leD le-add-same-cancel2 less-eq-rat-def mem-Collect-eq padd.rep-eq pnone.rep-eq*)

lemma *ppos-add*:
assumes *ppos a*
shows *ppos* (*padd a b*)
by (*metis Rep-prat Rep-prat-inject assms dual-order.strict-iff-order mem-Collect-eq padd-zero pnone.rep-eq ppos.rep-eq*)

lemma *pinv-inverts*:
assumes *pgte a b*
and *ppos b*
shows *pgte* (*pinv b*) (*pinv a*)
proof −
have *Rep-prat a* ≥ *Rep-prat b*
using *assms*(*1*) *pgte.rep-eq* **by** *auto*
then have (*Fract 1 1*) / *Rep-prat b* ≥ (*Fract 1 1*) / *Rep-prat a*
by (*metis One-rat-def assms*(*2*) *frac-le le-numeral-extra*(*4*) *ppos.rep-eq zero-le-one*)
then show *?thesis*

by (*simp add*: *One-rat-def pdiv.rep-eq pgte.rep-eq pinv-def pwrite.rep-eq*)
qed

lemma *pinv-pmult-ok*:
assumes *ppos p*
shows *pmult p* (*pinv p*) = *pwrite*
proof −
obtain *r* **where** *r* = *Rep-prat p* **by** *simp*
then have *r* ∗ ((*Fract 1 1*) / *r*) = *Fract 1 1*
using *assms ppos.rep-eq* **by** *force*
then show *?thesis*
by (*metis One-rat-def Rep-prat-inject* ‹*r* = *Rep-prat p*› *pdiv.rep-eq pinv-def pmult.rep-eq pwrite.rep-eq*)
qed

lemma *pinv-pwrite*:
pinv pwrite = *pwrite*
by (*metis Rep-prat-inverse div-by-1 pdiv.rep-eq pinv-def pwrite.rep-eq*)

lemma *pmult-ppos*:
assumes *ppos a*
and *ppos b*
shows *ppos* (*pmult a b*)
using *assms*(*1*) *assms*(*2*) *pmult.rep-eq ppos.rep-eq* **by** *auto*

lemma *ppos-inv*:
assumes *ppos p*
shows *ppos* (*pinv p*)
by (*metis Rep-prat Rep-prat-inverse assms less-eq-rat-def mem-Collect-eq not-one-le-zero pinv-pmult-ok pmult-comm pmult-special*(*2*) *pnone.rep-eq ppos.rep-eq pwrite.rep-eq*)

lemma *pmin-pmax*:
assumes *pgte x* (*pmin a b*)
shows *x* = *pmin* (*pmax x a*) (*pmax x b*)
proof (*cases pgte x a*)
case *True*
then show *?thesis*
by (*metis pmax-is pmax-smaller pmin-comm pmin-is*)
next
case *False*
then show *?thesis*
by (*metis assms not-pgte-charact pgt-implies-pgte pmax-is pmax-smaller pmin-comm pmin-is*)
qed

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*
by (*metis* (*mono-tags*, *lifting*) *assms comp-one-def p-greater-exists someI-ex*)

lemma *ppos-eq-pnone*:
ppos p $\longleftrightarrow$ *p* $\neq$ *pnone*
by (*metis Rep-prat Rep-prat-inject dual-order.strict-iff-order mem-Collect-eq pnone.rep-eq ppos.rep-eq*)

lemma *pmult-order*:
assumes *pgte a b*
shows *pgte* (*pmult p a*) (*pmult b p*)
using *assms p-greater-exists pmult-comm pmult-distr* **by** *force*

lemma *multiply-smaller-pwrite*:
assumes *pgte pwrite a*
and *pgte pwrite b*
shows *pgte pwrite* (*pmult a b*)
by (*metis assms*(*1*) *assms*(*2*) *p-greater-exists padd-asso pmult-order pmult-special*(*1*))

lemma *pmult-pdiv-cancel*:
assumes *ppos a*
shows *pmult a* (*pdiv x a*) = *x*
by (*metis Rep-prat-inject assms divide-cancel-right dual-order.strict-iff-order nonzero-mult-div-cancel-left pdiv.rep-eq pmult.rep-eq ppos.rep-eq*)

lemma *pmult-padd*:
pmult a (*padd* (*pmult b x*) (*pmult c y*)) = *padd* (*pmult* (*pmult a b*) *x*) (*pmult* (*pmult a c*) *y*)
by (*metis Rep-prat-inject mult.assoc pmult.rep-eq pmult-distr*)

lemma *pdiv-smaller*:
assumes *pgte a b*
and *ppos a*
shows *pgte pwrite* (*pdiv b a*)
proof −
let *?a* = *Rep-prat a*
let *?b* = *Rep-prat b*
have *?b* / *?a* $\leq$ *1*
by (*meson assms*(*1*) *assms*(*2*) *divide-le-eq-1-pos pgte.rep-eq ppos.rep-eq*)

then show *?thesis*
by (*simp add*: *pdiv.rep-eq pgte.rep-eq pwrite.rep-eq*)
qed

lemma *sum-coeff*:
assumes *ppos a*
and *ppos b*
shows *padd* (*pdiv a* (*padd a b*)) (*pdiv b* (*padd a b*)) = *pwrite*
proof −
let *?a* = *Rep-prat a*
let *?b* = *Rep-prat b*
have (*?a* / (*?a* + *?b*)) + (*?b* / (*?a* + *?b*)) = *1*
by (*metis add-divide-distrib add-pos-pos assms*(*1*) *assms*(*2*) *less-irrefl ppos.rep-eq right-inverse-eq*)
then show *?thesis*
by (*metis Rep-prat-inject padd.rep-eq pdiv.rep-eq pwrite.rep-eq*)
qed

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*))
by (*metis* (*mono-tags*, *lifting*) *Rep-prat assms*(*1*) *assms*(*2*) *assms*(*3*) *convex-bound-le mem-Collect-eq padd.rep-eq pgte.rep-eq pmult.rep-eq pwrite.rep-eq*)

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* ⇒ *prat*
type-synonym *bmask* = *heap-loc* ⇒ *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* ⇒ *mask* ⇒ *mask* **where**
$\textit{multiply-mask}\ p\ \pi\ hl = \textit{pmult}\ p\ (\pi\ hl)$

fun *empty-mask* **where**
$\textit{empty-mask}\ hl = \textit{pnone}$

fun *empty-bmask* **where**
$\textit{empty-bmask}\ hl = \textit{False}$

fun *add-acc* **where** $\textit{add-acc}\ \pi\ hl\ p = \pi(hl := \textit{padd}\ (\pi\ hl)\ p)$

inductive *rm-acc* **where**
$\pi\ hl = \textit{padd}\ p\ r \Longrightarrow \textit{rm-acc}\ \pi\ hl\ p\ (\pi(hl := r))$

fun *add-masks* **where**
$\textit{add-masks}\ \pi'\ \pi\ hl = \textit{padd}\ (\pi'\ hl)\ (\pi\ hl)$

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

fun *uni-mask* **where**
$\textit{uni-mask}\ hl\ p = \textit{empty-mask}(hl := p)$

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

definition *valid-null* :: *mask* ⇒ *bool* **where**
$\textit{valid-null}\ \pi \longleftrightarrow (\forall f.\ \pi\ (\textit{null}, f) = \textit{pnone})$

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

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

definition *big-add-masks* **where**
$\textit{big-add-masks}\ \Pi\ \Pi'\ h = \textit{add-masks}\ (\Pi\ h)\ (\Pi'\ h)$

definition *big-greater-mask* **where**
$\textit{big-greater-mask}\ \Pi\ \Pi' \longleftrightarrow (\forall h.\ \textit{greater-mask}\ (\Pi\ h)\ (\Pi'\ h))$

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

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

fun *pre-get-m* **where** *pre-get-m* φ = *fst* φ
fun *pre-get-h* **where** *pre-get-h* φ = *snd* φ
fun *srm-acc* **where** *srm-acc* φ *hl p* = (*rm-acc* (*pre-get-m* φ) *hl p*, *pre-get-h* φ)

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

definition *upper-bounded* :: *mask* ⇒ *prat* ⇒ *bool* **where**
upper-bounded π *p* ⟷ (∀ *hl*. *pgte p* (π *hl*))

1.2.2 Lemmas

lemma *ssubsetI*:
assumes ⋀π *h*. (π, *h*) ∈ *A* ⟹ (π, *h*) ∈ *B*
shows *A* ⊆ *B*
using *assms* **by** *auto*

lemma *double-inclusion*:
assumes *A* ⊆ *B*
and *B* ⊆ *A*
shows *A* = *B*
using *assms* **by** *blast*

lemma *add-masks-comm*:
add-masks a b = *add-masks b a*
proof (*rule ext*)
fix *x* **show** *add-masks a b x* = *add-masks b a x*
by (*metis Rep-prat-inverse add.commute add-masks.simps padd.rep-eq*)
qed

lemma *add-masks-asso*:
add-masks (*add-masks a b*) *c* = *add-masks a* (*add-masks b c*)
proof (*rule ext*)
fix *x* **show** *add-masks* (*add-masks a b*) *c x* = *add-masks a* (*add-masks b c*) *x*
by (*metis Rep-prat-inverse add.assoc add-masks.simps padd.rep-eq*)
qed

lemma *minus-empty*:
π = *add-masks* π *empty-mask*
proof (*rule ext*)
fix *x* **show** π *x* = *add-masks* π *empty-mask x*
by (*metis Rep-prat-inverse add.right-neutral add-masks.simps empty-mask.simps padd.rep-eq pnone.rep-eq*)
qed

lemma *add-acc-uni-mask*:
add-acc π *hl p* = *add-masks* π (*uni-mask hl p*)
proof (*rule ext*)

fix *x* **show** *add-acc* π *hl p x* = *add-masks* π (*uni-mask hl p*) *x*
by (*metis* (*no-types*, *opaque-lifting*) *add-acc.simps add-masks.simps fun-upd-apply minus-empty uni-mask.simps*)
qed

lemma *add-masks-equiv-valid-null*:
valid-null (*add-masks a b*) $\longleftrightarrow$ *valid-null a* $\wedge$ *valid-null b*
by (*metis* (*mono-tags*, *lifting*) *add-masks.simps padd-zero valid-null-def*)

lemma *valid-maskI*:
assumes $\bigwedge$*hl. pgte pwrite* (π *hl*)
and $\bigwedge$*f.* π (*null*, *f*) = *pnone*
shows *valid-mask* π
by (*simp add*: *assms*(*1*) *assms*(*2*))

lemma *greater-mask-equiv-def*:
greater-mask π' π $\longleftrightarrow$ ($\forall$ *hl. pgte* (π' *hl*) (π *hl*))
(**is** *?A* $\longleftrightarrow$ *?B*)
proof (*rule iffI*)
show *?A* $\Longrightarrow$ *?B*
proof (*clarify*)
fix *hl* **assume** *greater-mask* π' π
then obtain *r* **where** π' = *add-masks* π *r*
using *greater-mask-def* **by** *blast*
then show *pgte* (π' *hl*) (π *hl*)
using *Rep-prat padd.rep-eq pgte.rep-eq* **by** *auto*
qed
show *?B* $\Longrightarrow$ *?A*
proof −
assume *?B*
let *?r* = λ*hl.* (*SOME p.* π' *hl* = *padd* (π *hl*) *p*)
have π' = *add-masks* π *?r*
proof (*rule ext*)
fix *hl*
have π' *hl* = *padd* (π *hl*) (*?r hl*)
by (*meson* ‹$\forall$ *hl. pgte* (π' *hl*) (π *hl*)› *p-greater-exists someI-ex*)
then show π' *hl* = *add-masks* π *?r hl*
by *auto*
qed
then show *?A*
using *greater-mask-def* **by** *blast*
qed
qed

lemma *greater-maskI*:
assumes $\bigwedge$*hl. pgte* (π' *hl*) (π *hl*)
shows *greater-mask* π' π
by (*simp add*: *assms greater-mask-equiv-def*)

lemma *greater-mask-properties*:
greater-mask π π
greater-mask a b $\wedge$ *greater-mask* b c $\Longrightarrow$ *greater-mask* a c
greater-mask π' π $\wedge$ *greater-mask* π π' $\Longrightarrow$ $\pi = \pi'$
apply (*simp add*: *greater-maskI pgte.rep-eq*)
apply (*metis add-masks-asso greater-mask-def*)
proof (*rule ext*)
fix x **assume** *greater-mask* π' π $\wedge$ *greater-mask* π π'
then show $\pi\ x = \pi'\ x$
by (*meson greater-mask-equiv-def pgte-antisym*)
qed

lemma *greater-mask-decomp*:
assumes *greater-mask* a (*add-masks* b c)
shows $\exists$ *a1 a2*. a = *add-masks a1 a2* $\wedge$ *greater-mask a1* b $\wedge$ *greater-mask a2* c
by (*metis add-masks-asso assms greater-mask-def greater-mask-properties*(*1*))

lemma *valid-empty*:
valid-mask empty-mask
by (*metis empty-mask.simps le-add-same-cancel1 p-greater-exists padd.rep-eq pgte.rep-eq pnone.rep-eq valid-mask.simps*)

lemma *upper-valid-aux*:
assumes *valid-mask* a
and a = *add-masks* b c
shows *valid-mask* b
proof (*rule valid-maskI*)
show $\bigwedge$*hl*. *pgte pwrite* (b *hl*)
using *assms*(*1*) *assms*(*2*) *p-greater-exists padd-asso* **by** *fastforce*
fix f **show** b (*null*, f) = *pnone*
by (*metis add-masks-comm assms*(*1*) *assms*(*2*) *empty-mask.simps greater-mask-def greater-mask-equiv-def minus-empty pgte-antisym valid-mask.simps*)
qed

lemma *upper-valid*:
assumes *valid-mask* a
and a = *add-masks* b c
shows *valid-mask* b $\wedge$ *valid-mask* c
using *add-masks-comm assms*(*1*) *assms*(*2*) *upper-valid-aux* **by** *blast*

lemma *equal-on-bmaskI*:
assumes $\bigwedge$*hl*. π *hl* $\Longrightarrow$ h *hl* = h' *hl*
shows *equal-on-bmask* π h h'
using *assms equal-on-bmask-def* **by** *blast*

lemma *big-add-greater*:
big-greater-mask (*big-add-masks* A B) B
by (*metis add-masks-comm big-add-masks-def big-greater-mask-def greater-mask-def*)

lemma *big-greater-iff*:
big-greater-mask A B $\Longrightarrow$ ($\exists$ *C*. *A* = *big-add-masks B C*)
proof –
assume *big-greater-mask A B*
let *?C* = λ*h*. *SOME r*. *A h* = *add-masks* (*B h*) *r*
have *A* = *big-add-masks B ?C*
proof (*rule ext*)
fix *x*
have *A x* = *add-masks* (*B x*) (*?C x*)
proof (*rule ext*)
fix *xa*
have *A x* = *add-masks* (*B x*) (*SOME r*. *A x* = *add-masks* (*B x*) *r*)
by (*metis* (*mono-tags*, *lifting*) ‹*big-greater-mask A B*› *big-greater-mask-def greater-mask-def someI-ex*)
then show *A x xa* = *add-masks* (*B x*) (*SOME r*. *A x* = *add-masks* (*B x*) *r*) *xa*
by *auto*
qed
then show *A x* = *big-add-masks B* (λ*h*. *SOME r*. *A h* = *add-masks* (*B h*) *r*) *x*
by (*metis* (*no-types*, *lifting*) *big-add-masks-def*)
qed
then show $\exists$ *C*. *A* = *big-add-masks B C*
by *fast*
qed

lemma *big-add-masks-asso*:
big-add-masks A (*big-add-masks B C*) = *big-add-masks* (*big-add-masks A B*) *C*
proof (*rule ext*)
fix *x* **show** *big-add-masks A* (*big-add-masks B C*) *x* = *big-add-masks* (*big-add-masks A B*) *C x*
by (*simp add*: *add-masks-asso big-add-masks-def*)
qed

lemma *big-add-mask-neutral*:
big-add-masks Π (λ-. *empty-mask*) = Π
proof (*rule ext*)
fix *x* **show** *big-add-masks* Π (λ-. *empty-mask*) *x* = Π *x*
by (*metis big-add-masks-def minus-empty*)
qed

lemma *sym-equal-on-mask*:
equal-on-mask π *a b* $\longleftrightarrow$ *equal-on-mask* π *b a*
proof –
have $\bigwedge$*a b*. *equal-on-mask* π *a b* $\Longrightarrow$ *equal-on-mask* π *b a*
by (*simp add*: *equal-on-mask-def*)
then show *?thesis* **by** *blast*
qed

lemma *greater-mask-uni-equiv*:

greater-mask π (*uni-mask hl r*) $\longleftrightarrow$ *pgte* (π *hl*) *r*
by (*metis add-masks-comm fun-upd-apply greater-mask-def greater-mask-equiv-def minus-empty uni-mask.simps*)

lemma *greater-mask-uniI*:
assumes *pgte* (π *hl*) *r*
shows *greater-mask* π (*uni-mask hl r*)
using *greater-mask-uni-equiv assms* **by** *metis*

lemma *greater-bmask-refl*:
greater-bmask H H
by (*simp add*: *greater-bmask-def*)

lemma *greater-bmask-trans*:
assumes *greater-bmask A B*
and *greater-bmask B C*
shows *greater-bmask A C*
by (*metis assms*(*1*) *assms*(*2*) *greater-bmask-def*)

lemma *update-dm-same*:
update-dm dm π π = *dm*
proof (*rule ext*)
fix *x* **show** *update-dm dm* π π *x* = *dm x*
by (*simp add*: *pgt.rep-eq update-dm-def*)
qed

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 (*rule ext*)
fix *hl* **show** *update-dm* (*update-dm dm* π π') π' π'' *hl* = *update-dm dm* π π'' *hl*
proof −
have *update-dm* (*update-dm dm* π π') π' π'' *hl* $\longleftrightarrow$ (*update-dm dm* π π') *hl* $\lor$ *pgt* (π' *hl*) (π'' *hl*)
using *update-dm-def* **by** *metis*
also have ... $\longleftrightarrow$ *dm hl* $\lor$ *pgt* (π *hl*) (π' *hl*) $\lor$ *pgt* (π' *hl*) (π'' *hl*)
using *update-dm-def* **by** *metis*
moreover have *update-dm dm* π π'' *hl* $\longleftrightarrow$ *dm hl* $\lor$ *pgt* (π *hl*) (π'' *hl*)
using *update-dm-def* **by** *metis*
moreover have *pgt* (π *hl*) (π' *hl*) $\lor$ *pgt* (π' *hl*) (π'' *hl*) $\longleftrightarrow$ *pgt* (π *hl*) (π'' *hl*)
proof
show *pgt* (π *hl*) (π' *hl*) $\lor$ *pgt* (π' *hl*) (π'' *hl*) $\Longrightarrow$ *pgt* (π *hl*) (π'' *hl*)
by (*metis assms*(*1*) *assms*(*2*) *greater-mask-equiv-def greater-mask-properties*(*2*) *not-pgte-charact pgte-antisym*)
show *pgt* (π *hl*) (π'' *hl*) $\Longrightarrow$ *pgt* (π *hl*) (π' *hl*) $\lor$ *pgt* (π' *hl*) (π'' *hl*)
by (*metis assms*(*1*) *greater-mask-equiv-def not-pgte-charact pgte-antisym*)
qed
ultimately show *?thesis* **by** *blast*

qed
qed

lemma *equal-on-bmask-greater*:
assumes *equal-on-bmask* π' *h* h'
and *greater-bmask* π' π
shows *equal-on-bmask* π *h* h'
by (*metis* (*mono-tags*, *lifting*) *assms*(*1*) *assms*(*2*) *equal-on-bmask-def greater-bmask-def*)

lemma *update-dm-equal-bmask*:
assumes π = *add-masks* π' *m*
shows *equal-on-bmask* (*update-dm dm* π π') h' *h* $\longleftrightarrow$ *equal-on-mask m h* h' $\wedge$ *equal-on-bmask dm h* h'
proof −
have *equal-on-bmask* (*update-dm dm* π π') h' *h* $\longleftrightarrow$ ($\forall$ *hl*. *update-dm dm* π π' *hl* $\longrightarrow$ h' *hl* = *h hl*)
by (*simp add*: *equal-on-bmask-def*)
moreover have $\bigwedge$*hl*. *update-dm dm* π π' *hl* $\longleftrightarrow$ *dm hl* $\vee$ *pgt* (π *hl*) (π' *hl*)
by (*simp add*: *update-dm-def*)
moreover have ($\forall$ *hl*. *update-dm dm* π π' *hl* $\longrightarrow$ h' *hl* = *h hl*) $\longleftrightarrow$ *equal-on-mask m h* h' $\wedge$ *equal-on-bmask dm h* h'
proof
show $\forall$ *hl*. *update-dm dm* π π' *hl* $\longrightarrow$ h' *hl* = *h hl* $\Longrightarrow$ *equal-on-mask m h* h' $\wedge$ *equal-on-bmask dm h* h'
by (*simp add*: *assms equal-on-bmask-def equal-on-mask-def padd.rep-eq pgt.rep-eq ppos.rep-eq update-dm-def*)
assume *equal-on-mask m h* h' $\wedge$ *equal-on-bmask dm h* h'
then have $\bigwedge$*hl*. *update-dm dm* π π' *hl* $\Longrightarrow$ h' *hl* = *h hl*
by (*metis* (*full-types*) *add.right-neutral add-masks.simps assms dual-order.strict-iff-order equal-on-bmask-def equal-on-mask-def padd.rep-eq pgt.rep-eq pnone.rep-eq ppos-eq-pnone update-dm-def*)
then show $\forall$ *hl*. *update-dm dm* π π' *hl* $\longrightarrow$ h' *hl* = *h hl*
by *simp*
qed
then show *?thesis*
by (*simp add*: *calculation*)
qed

lemma *const-sum-mask-greater*:
assumes *add-masks a b* = *add-masks c d*
and *greater-mask a c*
shows *greater-mask d b*
proof (*rule ccontr*)
assume ¬ *greater-mask d b*
then obtain *hl* **where** ¬ *pgte* (*d hl*) (*b hl*)
using *greater-mask-equiv-def* **by** *blast*
then have *pgt* (*b hl*) (*d hl*)
using *not-pgte-charact* **by** *auto*
then have *pgt* (*padd* (*a hl*) (*b hl*)) (*padd* (*c hl*) (*d hl*))

by (*metis assms*(*2*) *greater-mask-equiv-def padd-comm pgte-pgt*)
then show *False*
by (*metis add-masks.simps assms*(*1*) *not-pgte-charact order-refl pgte.rep-eq*)
qed

lemma *add-masks-cancellative*:
assumes *add-masks b c* = *add-masks b d*
shows *c* = *d*
proof (*rule ext*)
fix *x* **show** *c x* = *d x*
by (*metis assms*(*1*) *const-sum-mask-greater greater-mask-properties*(*1*) *greater-mask-properties*(*3*))
qed

lemma *equal-on-maskI*:
assumes $\bigwedge$*hl*. *ppos* (π *hl*) ⟹ *h hl* = *h′ hl*
shows *equal-on-mask* π *h h′*
by (*simp add*: *assms equal-on-mask-def*)

lemma *greater-equal-on-mask*:
assumes *equal-on-mask* π′ *h h′*
and *greater-mask* π′ π
shows *equal-on-mask* π *h h′*
proof (*rule equal-on-maskI*)
fix *hl* **assume** *asm*: *ppos* (π *hl*)
then show *h hl* = *h′ hl*
by (*metis assms*(*1*) *assms*(*2*) *equal-on-mask-def greater-mask-equiv-def less-le-trans pgte.rep-eq ppos.rep-eq*)
qed

lemma *equal-on-mask-sum*:
equal-on-mask π*1* *h h′* ∧ *equal-on-mask* π*2* *h h′* ⟷ *equal-on-mask* (*add-masks* π*1* π*2*) *h h′*
proof
show *equal-on-mask* (*add-masks* π*1* π*2*) *h h′* ⟹ *equal-on-mask* π*1* *h h′* ∧ *equal-on-mask* π*2* *h h′*
using *add-masks-comm greater-equal-on-mask greater-mask-def* **by** *blast*
assume *asm0*: *equal-on-mask* π*1* *h h′* ∧ *equal-on-mask* π*2* *h h′*
show *equal-on-mask* (*add-masks* π*1* π*2*) *h h′*
proof (*rule equal-on-maskI*)
fix *hl* **assume** *ppos* (*add-masks* π*1* π*2* *hl*)
then show *h hl* = *h′ hl*
proof (*cases ppos* (π*1* *hl*))
case *True*
then show *?thesis*
by (*meson asm0 equal-on-mask-def*)
next
case *False*
then show *?thesis*
by (*metis asm0* ‹*ppos* (*add-masks* π*1* π*2* *hl*)› *add-masks.simps equal-on-mask-def*

padd-zero ppos-eq-pnone)
    qed
  qed
qed

lemma *valid-larger-mask*:
  valid-mask $\pi \longleftrightarrow$ *greater-mask full-mask* π
   by (*metis fst-eqD full-mask-def greater-maskI greater-mask-def not-one-le-zero not-pgte-charact pgt-implies-pgte pgte.rep-eq pnone.rep-eq pwrite.rep-eq surjective-pairing upper-valid-aux valid-mask.elims*(*1*))

lemma *valid-mask-full-mask*:
  valid-mask full-mask
  using *greater-mask-properties*(*1*) *valid-larger-mask* **by** *blast*

lemma *mult-greater*:
  assumes *greater-mask a b*
  shows *greater-mask* (*multiply-mask p a*) (*multiply-mask p b*)
 by (*metis* (*full-types*) *assms greater-mask-equiv-def multiply-mask-def p-greater-exists pmult-distr*)

lemma *mult-write-mask*:
  multiply-mask pwrite $\pi = \pi$
proof (*rule ext*)
  fix *x* **show** *multiply-mask pwrite* π $x = \pi$ x
    by (*simp add*: *multiply-mask-def pmult-special*(*1*))
qed

lemma *mult-distr-masks*:
  multiply-mask a (*add-masks b c*) = *add-masks* (*multiply-mask a b*) (*multiply-mask a c*)
proof (*rule ext*)
  fix *x* **show** *multiply-mask a* (*add-masks b c*) *x* = *add-masks* (*multiply-mask a b*) (*multiply-mask a c*) *x*
    by (*simp add*: *multiply-mask-def pmult-distr*)
qed

lemma *mult-add-states*:
  multiply-mask (*padd a b*) π = *add-masks* (*multiply-mask a* π) (*multiply-mask b* π)
proof (*rule ext*)
   fix *x* **show** *multiply-mask* (*padd a b*) π *x* = *add-masks* (*multiply-mask a* π) (*multiply-mask b* π) *x*
    by (*simp add*: *multiply-mask-def pmult-comm pmult-distr*)
qed

lemma *upper-boundedI*:
  assumes $\bigwedge$*hl*. *pgte p* (π *hl*)
  shows *upper-bounded* π *p*

by (*simp add*: *assms upper-bounded-def*)

lemma *upper-bounded-smaller*:
assumes *upper-bounded* π *a*
shows *upper-bounded* (*multiply-mask p* π) (*pmult p a*)
by (*metis assms multiply-mask-def p-greater-exists pmult-distr upper-bounded-def*)

lemma *upper-bounded-bigger*:
assumes *upper-bounded* π *a*
and *pgte b a*
shows *upper-bounded* π *b*
by (*meson assms*(*1*) *assms*(*2*) *order-trans pgte.rep-eq upper-bounded-def*)

lemma *valid-mult*:
assumes *valid-mask* π
and *pgte pwrite p*
shows *valid-mask* (*multiply-mask p* π)
proof (*rule valid-maskI*)
have *upper-bounded* π *pwrite*
using *assms*(*1*) *upper-bounded-def* **by** *auto*
then have *upper-bounded* (*multiply-mask p* π) (*pmult p pwrite*)
by (*simp add*: *upper-bounded-smaller*)
then show $\bigwedge$*hl*. *pgte pwrite* (*multiply-mask p* π *hl*)
by (*metis assms*(*2*) *pmult-comm pmult-special*(*1*) *upper-bounded-bigger upper-bounded-def*)
show $\bigwedge$*f*. *multiply-mask p* π (*null*, *f*) = *pnone*
by (*metis Rep-prat-inverse add-0-left assms*(*1*) *multiply-mask-def padd.rep-eq padd-cancellative pmult-distr pnone.rep-eq valid-mask.elims*(*1*))
qed

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 (*rule valid-maskI*)
show $\bigwedge$*hl*. *pgte pwrite* (*add-masks a b hl*)
proof −
fix *hl*
have *pgte* (*padd ma mb*) (*add-masks a b hl*)
by (*metis* (*mono-tags*, *lifting*) *add-masks.simps add-mono-thms-linordered-semiring*(*1*) *assms*(*3*) *assms*(*4*) *padd.rep-eq pgte.rep-eq upper-bounded-def*)
then show *pgte pwrite* (*add-masks a b hl*)
by (*meson assms*(*5*) *dual-order.trans pgte.rep-eq*)
qed

show $\bigwedge f$. *add-masks a b* (*null*, *f*) = *pnone*
by (*metis Rep-prat-inverse add-0-left add-masks.simps assms*(*1*) *assms*(*2*) *padd.rep-eq pnone.rep-eq valid-mask.simps*)
show *upper-bounded* (*add-masks a b*) (*padd ma mb*)
using *add-mono-thms-linordered-semiring*(*1*) *assms*(*3*) *assms*(*4*) *padd.rep-eq pgte.rep-eq upper-bounded-def* **by** *fastforce*
qed

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*)
by (*metis* (*no-types*, *opaque-lifting*) *assms*(*1*) *assms*(*2*) *assms*(*3*) *multiply-mask-def pmult-comm pmult-special*(*2*) *upper-bounded-bigger upper-bounded-def upper-bounded-smaller valid-mask.elims*(*1*))

lemma *greater-mult*:
assumes *greater-mask a b*
shows *greater-mask* (*multiply-mask p a*) (*multiply-mask p b*)
by (*metis Rep-prat assms greater-mask-equiv-def mem-Collect-eq mult-left-mono multiply-mask-def pgte.rep-eq pmult.rep-eq*)

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 $\bigwedge$*hl*. *ppos* (π *hl*) ⟹ *h hl* ≠ *None*
shows *valid-state* (π, *h*)
using *assms*(*1*) *assms*(*2*) *valid-heap-def valid-state.simps* **by** *blast*

definition *empty-heap* **where** *empty-heap hl* = *None*

lemma *valid-pre-unit*:
  valid-state (*empty-mask*, *empty-heap*)
  using *pnone.rep-eq ppos.rep-eq valid-empty valid-stateI* **by** *fastforce*

typedef *state* = { φ |φ. *valid-state* φ }
  using *valid-pre-unit* **by** *blast*

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* $\rightharpoonup$ *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*) $\Longrightarrow$ *h hl* $\neq$ *None*
  shows *valid-heap* π *h*
  using *assms valid-heap-def* **by** *presburger*

lemma *valid-state-decompose*:
  assumes *valid-state* (*add-masks a b*, *h*)

shows *valid-state* (a, h)
proof (*rule valid-stateI*)
show *valid-mask a*
using *assms upper-valid-aux valid-state.simps* **by** *blast*
fix *hl* **assume** *ppos* (*a hl*) **then show** $h\ hl \neq None$
by (*metis add-masks.simps assms ppos-add valid-heap-def valid-state.simps*)
qed

lemma *compatible-heapsI*:
assumes $\bigwedge hl\ a\ b.\ h\ hl = Some\ a \Longrightarrow h'\ hl = Some\ b \Longrightarrow a = b$
shows *compatible-heaps* $h\ h'$
by (*metis assms compatible-heaps-def compatible-options.elims*(*3*))

lemma *compatibleI-old*:
assumes $\bigwedge hl\ x\ y.\ snd\ \varphi\ hl = Some\ x \wedge snd\ \varphi'\ hl = Some\ y \Longrightarrow x = y$
and *valid-mask* (*add-masks* (*fst* φ) (*fst* φ'))
shows *compatible* $\varphi\ \varphi'$
using *assms*(*1*) *assms*(*2*) *compatible-def compatible-heapsI* **by** *presburger*

lemma *larger-heap-anti*:
assumes *larger-heap a b*
and *larger-heap b a*
shows $a = b$
proof (*rule ext*)
fix *x* **show** $a\ x = b\ x$
proof (*cases a x*)
case *None*
then show *?thesis*
by (*metis assms*(*1*) *larger-heap-def not-None-eq*)
next
case (*Some a*)
then show *?thesis*
by (*metis assms*(*2*) *larger-heap-def*)
qed
qed

lemma *larger-heapI*:
assumes $\bigwedge hl\ x.\ h\ hl = Some\ x \Longrightarrow h'\ hl = Some\ x$
shows *larger-heap* $h'\ h$
by (*simp add*: *assms larger-heap-def*)

lemma *larger-heap-refl*:
larger-heap h h
using *larger-heap-def* **by** *blast*

lemma *compatible-heaps-comm*:
assumes *compatible-heaps a b*
shows $a ++ b = b ++ a$
proof (*rule ext*)

fix *x* **show** (*a* ++ *b*) *x* = (*b* ++ *a*) *x*
proof (*cases a x*)
case *None*
then show *?thesis*
by (*simp add*: *domIff map-add-dom-app-simps*(*2*) *map-add-dom-app-simps*(*3*))
next
case (*Some a*)
then show *?thesis*
by (*metis* (*no-types*, *lifting*) *assms compatible-heaps-def compatible-options.elims*(*2*) *map-add-None map-add-dom-app-simps*(*1*) *map-add-dom-app-simps*(*3*))
qed
qed

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*)
proof (*rule larger-heapI*)
fix *hl x* **assume** (*a* ++ *b*) *hl* = *Some x*
show (*a′* ++ *b′*) *hl* = *Some x*
proof (*cases a hl*)
case *None*
then show *?thesis*
by (*metis* ‹(*a* ++ *b*) *hl* = *Some x*› *assms*(*2*) *larger-heap-def map-add-SomeD map-add-find-right*)
next
case (*Some aa*)
then show *?thesis*
by (*metis* (*mono-tags*, *lifting*) ‹(*a* ++ *b*) *hl* = *Some x*› *assms*(*1*) *assms*(*2*) *assms*(*3*) *compatible-heaps-comm larger-heap-def map-add-Some-iff*)
qed
qed

lemma *larger-heap-trans*:
assumes *larger-heap a b*
and *larger-heap b c*
shows *larger-heap a c*
by (*metis* (*no-types*, *opaque-lifting*) *assms*(*1*) *assms*(*2*) *larger-heap-def*)

lemma *larger-heap-comp*:
assumes *larger-heap a b*
and *compatible-heaps a c*
shows *compatible-heaps b c*
proof (*rule compatible-heapsI*)
fix *hl a ba*
assume *b hl* = *Some a c hl* = *Some ba*
then show *a* = *ba*
by (*metis assms*(*1*) *assms*(*2*) *compatible-heaps-def compatible-options.simps*(*1*)

larger-heap-def)
qed

lemma *larger-heap-plus*:
  assumes *larger-heap a b*
    and *larger-heap a c*
   shows *larger-heap a* (*b* ++ *c*)
proof (*rule larger-heapI*)
  fix *hl x* **assume** (*b* ++ *c*) *hl* = *Some x*
  then show *a hl* = *Some x*
  proof (*cases b hl*)
   case *None*
   then show *?thesis*
    by (*metis* ‹(*b* ++ *c*) *hl* = *Some x*› *assms*(*2*) *larger-heap-def map-add-SomeD*)
  next
   case (*Some bb*)
   then show *?thesis*
     by (*metis* ‹(*b* ++ *c*) *hl* = *Some x*› *assms*(*1*) *assms*(*2*) *larger-heap-def map-add-SomeD*)
  qed
qed

lemma *compatible-heaps-sum*:
  assumes *compatible-heaps a b*
    and *compatible-heaps a c*
   shows *compatible-heaps a* (*b* ++ *c*)
 by (*metis* (*no-types*, *opaque-lifting*) *assms*(*1*) *assms*(*2*) *compatible-heaps-def map-add-dom-app-simps*(*1*) *map-add-dom-app-simps*(*3*))

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 (*rule compatible-heapsI*)
  fix *hl a b* **assume** *x hl* = *Some a y hl* = *Some b*
  then show *a* = *b*
   by (*metis assms*(*1*) *assms*(*2*) *assms*(*3*) *compatible-heaps-def compatible-options.simps*(*1*) *larger-heap-def*)
qed

lemma *get-h-m*:
  Rep-state x = (*get-m x*, *get-h x*) **by** *simp*

lemma *get-pre*:
  get-h x = *snd* (*Rep-state x*)
  get-m x = *fst* (*Rep-state x*)
  by *simp-all*

lemma *plus-ab-defined*:
$\varphi \oplus \varphi' \neq$ *None* $\longleftrightarrow$ *compatible-heaps* (*get-h* φ) (*get-h* φ') $\wedge$ *valid-mask* (*add-masks* (*get-m* φ) (*get-m* φ'))
(**is** *?A* $\longleftrightarrow$ *?B*)
proof
show *?A* $\Longrightarrow$ *?B*
by (*metis compatible-def get-pre*(*1*) *get-pre*(*2*) *plus-def*)
show *?B* $\Longrightarrow$ *?A*
using *compatible-def plus-def* **by** *auto*
qed

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 −
have *x* = (*Abs-state* (*add-states* (*Rep-state a*) (*Rep-state b*)))
by (*metis assms option.discI option.inject plus-def*)
moreover have *compatible* (*Rep-state a*) (*Rep-state b*)
using *assms*(*1*) *plus-def* **by** (*metis option.discI*)

moreover have *valid-state* (*add-states* (*Rep-state a*) (*Rep-state b*))
proof −
have *valid-state* (*add-masks* (*get-m a*) (*get-m b*), (*get-h a*) ++ (*get-h b*))
proof (*rule valid-stateI*)
show *valid-mask* (*add-masks* (*get-m a*) (*get-m b*))
using *calculation*(*2*) *compatible-def* **by** *fastforce*
fix *hl* **assume** *ppos* (*add-masks* (*get-m a*) (*get-m b*) *hl*)
then show (*get-h a* ++ *get-h b*) *hl* $\neq$ *None*
proof (*cases ppos* (*get-m a hl*))
case *True*
then show *?thesis*
by (*metis Rep-state get-h-m map-add-None mem-Collect-eq valid-heap-def valid-state.simps*)
next
case *False*
then have *ppos* (*get-m b hl*)
using ‹*ppos* (*add-masks* (*get-m a*) (*get-m b*) *hl*)› *padd.rep-eq ppos.rep-eq* **by** *auto*
then show *?thesis*
by (*metis Rep-state get-h-m map-add-None mem-Collect-eq valid-heap-def valid-state.simps*)
qed
qed
then show *?thesis*
using *add-states.simps get-h-m* **by** *presburger*
qed
ultimately show *get-m x* = *add-masks* (*get-m a*) (*get-m b*)
by (*metis Abs-state-inverse add-states.simps fst-conv get-h-m mem-Collect-eq*)

show *get-h x* = (*get-h a*) ++ (*get-h b*)
by (*metis Abs-state-inject CollectI Rep-state Rep-state-inverse* ‹*valid-state* (*add-states* (*Rep-state a*) (*Rep-state b*))› ‹*x* = *Abs-state* (*add-states* (*Rep-state a*) (*Rep-state b*))› *add-states.simps eq-snd-iff get-h.simps*)
qed

lemma *commutative*:
a ⊕ *b* = *b* ⊕ *a*
proof (*cases compatible-heaps* (*get-h a*) (*get-h b*) ∧ *valid-mask* (*add-masks* (*get-m a*) (*get-m b*)))
case *True*
then have *r0*: *compatible-heaps* (*get-h b*) (*get-h a*) ∧ *add-masks* (*get-m a*) (*get-m b*) = *add-masks* (*get-m b*) (*get-m a*)
by (*metis add-masks-comm compatible-heapsI compatible-heaps-def compatible-options.simps*(*1*))
then have (*get-h a*) ++ (*get-h b*) = (*get-h b*) ++ (*get-h a*)
by (*simp add*: *compatible-heaps-comm*)
then show *?thesis*
by (*metis True r0 add-states.simps get-h-m plus-ab-defined plus-def*)
next
case *False*
then show *?thesis*
by (*metis add-masks-comm compatible-heapsI compatible-heaps-def compatible-options.simps*(*1*) *plus-ab-defined*)
qed

lemma *asso1*:
assumes *a* ⊕ *b* = *Some ab* ∧ *b* ⊕ *c* = *Some bc*
shows *ab* ⊕ *c* = *a* ⊕ *bc*
proof (*cases ab* ⊕ *c*)
case *None*
then show *?thesis*
proof (*cases compatible-heaps* (*get-h ab*) (*get-h c*))
case *True*
then have ¬ *valid-mask* (*add-masks* (*add-masks* (*get-m a*) (*get-m b*)) (*get-m c*))
by (*metis None assms plus-ab-defined plus-charact*(*1*))
then show *?thesis*
by (*metis add-masks-asso assms plus-ab-defined plus-charact*(*1*))
next
case *False*
then have ¬ *compatible-heaps* (*get-h a* ++ *get-h b*) (*get-h c*)
using *assms plus-charact*(*2*) **by** *force*
then obtain *l x y* **where** (*get-h a* ++ *get-h b*) *l* = *Some x get-h c l* = *Some y* *x* ≠ *y*
using *compatible-heapsI* **by** *blast*
then have ¬ *compatible-heaps* (*get-h a*) (*get-h b* ++ *get-h c*)
proof (*cases get-h a l*)

case *None*
then show *?thesis*
by (*metis* ‹(*get-h a* ++ *get-h b*) *l* = *Some x*› ‹*get-h c l* = *Some y*› ‹*x* ≠ *y*› *assms compatible-heaps-comm map-add-dom-app-simps*(*1*) *map-add-dom-app-simps*(*3*) *map-add-find-right option.inject option.simps*(*3*) *plus-ab-defined*)
next
case (*Some aa*)
then show *?thesis*
by (*metis* ‹(*get-h a* ++ *get-h b*) *l* = *Some x*› ‹*get-h c l* = *Some y*› ‹*x* ≠ *y*› *assms commutative compatible-heaps-def compatible-options.elims*(*2*) *map-add-find-right option.inject option.simps*(*3*) *plus-charact*(*2*))
qed
then show *?thesis*
by (*metis None assms plus-ab-defined plus-charact*(*2*))
qed
next
case (*Some x*)
then have *compatible-heaps* (*get-h a* ++ *get-h b*) (*get-h c*)
by (*metis assms option.simps*(*3*) *plus-ab-defined plus-charact*(*2*))
then have *compatible-heaps* (*get-h a*) (*get-h b* ++ *get-h c*)
by (*metis* (*full-types*) *assms compatible-heaps-comm compatible-heaps-def compatible-heaps-sum compatible-options.simps*(*2*) *domIff map-add-dom-app-simps*(*1*) *option.distinct*(*1*) *plus-ab-defined*)
moreover have *valid-mask* (*add-masks* (*get-m a*) (*add-masks* (*get-m b*) (*get-m c*)))
by (*metis Some add-masks-asso assms option.distinct*(*1*) *plus-ab-defined plus-charact*(*1*))
ultimately obtain *y* **where** *Some y* = *a* ⊕ *bc*
by (*metis assms plus-ab-defined plus-charact*(*1*) *plus-charact*(*2*) *plus-def*)
then show *?thesis*
by (*metis* (*mono-tags*, *lifting*) *Some add-masks-asso add-states.simps assms get-h-m map-add-assoc option.distinct*(*1*) *plus-charact*(*1*) *plus-charact*(*2*) *plus-def*)
qed

lemma *asso2*:
assumes *a* ⊕ *b* = *Some ab* ∧ *b* ⊕ *c* = *None*
shows *ab* ⊕ *c* = *None*
proof (*cases valid-mask* (*add-masks* (*get-m b*) (*get-m c*)))
case *True*
then have ¬ *compatible-heaps* (*get-h b*) (*get-h c*)
using *assms plus-ab-defined* **by** *blast*
then obtain *l x y* **where** *get-h b l* = *Some x get-h c l* = *Some y x* ≠ *y*
using *compatible-heapsI* **by** *blast*
then have *get-h ab l* = *Some x*
by (*metis assms map-add-find-right plus-charact*(*2*))
then show *?thesis*
by (*metis* ‹*get-h c l* = *Some y*› ‹*x* ≠ *y*› *compatible-heaps-def compatible-options.simps*(*1*) *plus-ab-defined*)
next
case *False*

then obtain *l* **where** ¬ (*pgte pwrite* (*add-masks* (*get-m b*) (*get-m c*) *l*))
by (*metis Abs-state-cases Rep-state-cases Rep-state-inverse add-masks-equiv-valid-null get-h-m mem-Collect-eq valid-mask.simps valid-null-def valid-state.simps*)
then have ¬ (*pgte pwrite* (*add-masks* (*get-m ab*) (*get-m c*) *l*))
proof −
have *pgte* (*add-masks* (*get-m ab*) (*get-m c*) *l*) (*add-masks* (*get-m b*) (*get-m c*) *l*)
using *assms p-greater-exists padd-asso padd-comm plus-charact*(*1*) **by** *auto*
then show *?thesis*
by (*meson* ‹¬ *pgte pwrite* (*add-masks* (*get-m b*) (*get-m c*) *l*)› *order-trans pgte.rep-eq*)
qed
then show *?thesis*
using *plus-ab-defined valid-mask.simps* **by** *blast*
qed

lemma *core-defined*:
get-h $|\varphi|$ = *get-h* φ
get-m $|\varphi|$ = *empty-mask*
using *Abs-state-inverse core-def pnone.rep-eq ppos.rep-eq valid-empty valid-stateI*
apply *force*
by (*metis Abs-state-inverse CollectI core-def empty-mask.simps fst-conv get-pre*(*2*) *less-irrefl pnone.rep-eq ppos.rep-eq valid-empty valid-stateI*)

lemma *state-ext*:
assumes *get-h a* = *get-h b*
and *get-m a* = *get-m b*
shows *a* = *b*
by (*metis Rep-state-inverse assms*(*1*) *assms*(*2*) *get-h-m*)

lemma *core-is-smaller*:
Some x = $x \oplus |x|$
proof −
obtain *y* **where** *Some y* = $x \oplus |x|$
by (*metis Rep-state compatible-heapsI core-defined*(*1*) *core-defined*(*2*) *get-h-m mem-Collect-eq minus-empty option.collapse option.sel plus-ab-defined valid-state.simps*)
moreover have *y* = *x*
proof (*rule state-ext*)
have *get-h x* = *get-h x* ++ *get-h x*
by (*simp add*: *map-add-subsumed1*)
then show *get-h y* = *get-h x*
using *calculation core-defined*(*1*) *plus-charact*(*2*) **by** *presburger*
show *get-m y* = *get-m x*
by (*metis calculation core-defined*(*2*) *minus-empty plus-charact*(*1*))
qed
ultimately show *?thesis* **by** *blast*
qed

lemma *core-is-pure*:

Some $|x| = |x| \oplus |x|$
proof −
  obtain y **where** *Some* $y = |x| \oplus |x|$
    by (*metis core-def core-defined*(*1*) *core-is-smaller*)
  moreover have $y = |x|$
    by (*metis calculation core-def core-defined*(*1*) *core-is-smaller option.sel*)
  ultimately show *?thesis* **by** *blast*
qed

lemma *core-sum*:
  assumes *Some* $c = a \oplus b$
  shows *Some* $|c| = |a| \oplus |b|$
proof −
  obtain x **where** *Some* $x = |a| \oplus |b|$
   by (*metis assms core-defined*(*1*) *core-defined*(*2*) *minus-empty option.exhaust-sel plus-ab-defined valid-empty*)
  moreover have $x = |c|$
   by (*metis assms calculation core-defined*(*1*) *core-defined*(*2*) *minus-empty plus-charact*(*1*) *plus-charact*(*2*) *state-ext*)
  ultimately show *?thesis* **by** *blast*
qed

lemma *core-max*:
  assumes *Some* $x = x \oplus c$
  shows $\exists r.$ *Some* $|x| = c \oplus r$
proof −
  obtain y **where** *Some* $y = c \oplus |x|$
    by (*metis assms asso2 core-is-smaller plus-def*)
  moreover have $|x| = y$
    by (*metis* (*mono-tags*, *opaque-lifting*) *Rep-state-inverse add-masks-cancellative assms calculation commutative core-defined*(*1*) *core-sum get-h-m minus-empty option.inject plus-charact*(*1*))
  ultimately show *?thesis* **by** *blast*
qed

lemma *positivity*:
  assumes $a \oplus b =$ *Some* c
    and *Some* $c = c \oplus c$
  shows *Some* $a = a \oplus a$
proof −
  obtain x **where** *Some* $x = a \oplus a$
    by (*metis assms*(*1*) *assms*(*2*) *asso2 commutative option.exhaust-sel*)
  moreover have $x = a$
    by (*metis Rep-state-inverse add-masks-cancellative add-masks-comm assms*(*1*) *assms*(*2*) *calculation core-defined*(*1*) *core-defined*(*2*) *core-is-smaller get-h-m greater-mask-def greater-mask-properties*(*3*) *option.sel plus-charact*(*1*))
  ultimately show *?thesis* **by** *blast*
qed

lemma *cancellative*:
assumes *Some a* = *b* ⊕ *x*
and *Some a* = *b* ⊕ *y*
and |*x*| = |*y*|
shows *x* = *y*
by (*metis add-masks-cancellative assms*(*1*) *assms*(*2*) *assms*(*3*) *core-defined*(*1*) *plus-charact*(*1*) *state-ext*)

lemma *unit-charact*:
get-h unit = *empty-heap*
get-m unit = *empty-mask*
proof −
have *valid-state* (*empty-mask*, *empty-heap*)
using *valid-pre-unit* **by** *auto*
then show *get-h unit* = *empty-heap* **using** *unit-def*
by (*simp add*: ‹*unit* = *Abs-state* (*empty-mask*, *empty-heap*)› *Abs-state-inverse*)
show *get-m unit* = *empty-mask*
using ‹*valid-state* (*empty-mask*, *empty-heap*)› *unit-def Abs-state-inverse*
by *fastforce*
qed

lemma *empty-heap-neutral*:
a ++ *empty-heap* = *a*
proof (*rule ext*)
fix *x* **show** (*a* ++ *empty-heap*) *x* = *a x*
by (*simp add*: *domIff empty-heap-def map-add-dom-app-simps*(*3*))
qed

lemma *unit-neutral*:
Some a = *a* ⊕ *unit*
proof −
obtain *x* **where** *Some x* = *a* ⊕ *unit*
by (*metis Abs-state-cases Rep-state-cases Rep-state-inverse compatible-heapsI empty-heap-def fst-conv get-h-m mem-Collect-eq minus-empty option.distinct*(*1*) *option.exhaust-sel plus-ab-defined snd-conv unit-def valid-pre-unit valid-state.simps*)
moreover have *x* = *a*
proof (*rule state-ext*)
show *get-h x* = *get-h a*
using *calculation empty-heap-neutral plus-charact*(*2*) *unit-charact*(*1*) **by** *auto*
show *get-m x* = *get-m a*
by (*metis calculation minus-empty plus-charact*(*1*) *unit-charact*(*2*))
qed
ultimately show *?thesis* **by** *blast*
qed

lemma *stableI*:
assumes $\bigwedge$*hl. ppos* (*get-m* φ *hl*) ⟷ *get-h* φ *hl* ≠ *None*
shows *stable* φ
using *assms stable-def* **by** *blast*

lemma *stable-unit*:
stable unit
by (*metis empty-heap-def stable-def unit-charact*(*1*) *unit-charact*(*2*) *valid-heap-def valid-pre-unit valid-state.simps*)

lemma *stable-sum*:
assumes *stable a*
and *stable b*
and *Some x* = *a* ⊕ *b*
shows *stable x*
proof (*rule stableI*)
fix *hl*
show *ppos* (*get-m x hl*) = (*get-h x hl* ≠ *None*) (**is** *?A* ⟷ *?B*)
proof
show *?A* ⟹ *?B*
by (*metis add-le-same-cancel2 add-masks.simps assms*(*1*) *assms*(*2*) *assms*(*3*) *leI less-le-trans map-add-None padd.rep-eq plus-charact*(*1*) *plus-charact*(*2*) *ppos.rep-eq stable-def*)
show *?B* ⟹ *?A*
by (*metis add-masks.simps assms*(*1*) *assms*(*2*) *assms*(*3*) *map-add-None padd-comm plus-charact*(*1*) *plus-charact*(*2*) *ppos-add stable-def*)
qed
qed

lemma *multiply-valid*:
assumes *pgte pwrite p*
shows *valid-state* (*multiply-mask p* (*get-m* φ), *get-h* φ)
proof (*rule valid-stateI*)
show *valid-mask* (*multiply-mask p* (*get-m* φ))
by (*metis Rep-state assms*(*1*) *get-h-m mem-Collect-eq valid-mult valid-state.simps*)
fix *hl* **show** *ppos* (*multiply-mask p* (*get-m* φ) *hl*) ⟹ *get-h* φ *hl* ≠ *None*
by (*metis Abs-state-cases Rep-state-cases Rep-state-inverse get-h-m mem-Collect-eq multiply-mask-def pmult-comm pmult-special*(*2*) *ppos-eq-pnone valid-heap-def valid-state.simps*)
qed

1.4 This state model corresponds to a separation algebra

global-interpretation *PartialSA*: *package-logic plus core unit stable*
defines *greater* (**infixl** ‹⪰› *50*) = *PartialSA.greater*
and *add-set* (**infixl** ‹⊗› *60*) = *PartialSA.add-set*
and *defined* (**infixl** ‹|#|› *60*) = *PartialSA.defined*
and *greater-set* (**infixl** ‹|≫|› *50*) = *PartialSA.greater-set*
and *minus* (**infixl** ‹|⊖|› *60*) = *PartialSA.minus*
apply *standard*
apply (*simp add*: *commutative*)
using *asso1* **apply** *blast*
using *asso2* **apply** *blast*
using *core-is-smaller* **apply** *blast*

using *core-is-pure* **apply** *blast*
using *core-max* **apply** *blast*
using *core-sum* **apply** *blast*
using *positivity* **apply** *blast*
using *cancellative* **apply** *blast*
using *unit-neutral* **apply** *blast*
using *stable-sum* **apply** *blast*
using *stable-unit* **by** *blast*

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 −
let *?m* = *λl. SOME p. get-m a l* = *padd* (*get-m b l*) *p*
have *r0*: *get-m a* = *add-masks* (*get-m b*) *?m*
proof (*rule ext*)
fix *l*
have *pgte* (*get-m a l*) (*get-m b l*)
by (*meson assms*(*2*) *greater-mask-equiv-def*)
then have *get-m a l* = *padd* (*get-m b l*) (*SOME p. get-m a l* = *padd* (*get-m b l*) *p*)
by (*simp add*: *p-greater-exists verit-sko-ex′*)
then show *get-m a l* = *add-masks* (*get-m b*) (*λl. SOME p. get-m a l* = *padd* (*get-m b l*) *p*) *l*
by *simp*
qed
moreover have *valid-state* (*?m*, *get-h a*)
proof (*rule valid-stateI*)
show *valid-mask* (*λl. SOME p. get-m a l* = *padd* (*get-m b l*) *p*)
by (*metis* (*no-types*, *lifting*) *Rep-state calculation get-h-m mem-Collect-eq upper-valid valid-state.simps*)
fix *hl*
assume *asm0*: *ppos* (*SOME p. get-m a hl* = *padd* (*get-m b hl*) *p*)
then have *ppos* (*get-m a hl*)
by (*metis* (*no-types*, *lifting*) *add-masks.elims add-masks-comm calculation greater-mask-def ppos-add*)
then show *get-h a hl* ≠ *None*
by (*metis Rep-state get-h.simps get-pre*(*2*) *mem-Collect-eq prod.collapse valid-heap-def valid-state.simps*)
qed
moreover have *compatible-heaps* (*get-h b*) (*get-h a*)
by (*metis* (*mono-tags*, *lifting*) *assms*(*1*) *compatible-heapsI larger-heap-def option.inject*)
ultimately have *r2*: (*get-m a*, *get-h a*) = *add-states* (*get-m b*, *get-h b*) (*?m*, *get-h a*)
proof −
have *get-h b* ++ *get-h a* = *get-h a*

proof (*rule ext*)
fix *x* **show** (*get-h b* ++ *get-h a*) *x* = *get-h a x*
by (*metis assms*(*1*) *domIff larger-heap-def map-add-dom-app-simps*(*1*) *map-add-dom-app-simps*(*3*) *not-Some-eq*)
qed
then show *?thesis*
by (*metis r0 add-states.simps*)
qed
moreover have *r1*: *compatible-heaps* (*get-h b*) (*get-h a*) ∧ *valid-mask* (*add-masks* (*get-m b*) *?m*)
by (*metis Rep-state* ‹*compatible-heaps* (*get-h b*) (*get-h a*)› *r0 get-h-m mem-Collect-eq valid-state.simps*)
ultimately have *Some a* = *b* ⊕ *Abs-state* (*?m*, *get-h a*)
proof −
have *Rep-state* (*Abs-state* (*?m*, *get-h a*)) = (*?m*, *get-h a*)
using *Abs-state-inverse* ‹*valid-state* (λ*l*. *SOME p*. *get-m a l* = *padd* (*get-m b l*) *p*, *get-h a*)› **by** *blast*
moreover have *compatible* (*Rep-state b*) (*?m*, *get-h a*)
using *r1 compatible-def* **by** *auto*
moreover have *valid-state* (*add-states* (*Rep-state b*) (*?m*, *get-h a*))
by (*metis Rep-state r2 get-h-m mem-Collect-eq*)
ultimately show *?thesis*
by (*metis* (*no-types*, *lifting*) *Rep-state-inverse r2 get-h-m plus-def*)
qed
then show *?thesis*
by (*meson PartialSA.greater-def*)
qed

lemma *larger-implies-greater-mask-hl*:
assumes *a* ⪰ *b*
shows *pgte* (*get-m a hl*) (*get-m b hl*)
using *PartialSA.greater-def assms p-greater-exists plus-charact*(*1*) **by** *auto*

lemma *larger-implies-larger-heap*:
assumes *a* ⪰ *b*
shows *larger-heap* (*get-h a*) (*get-h b*)
by (*metis* (*full-types*) *PartialSA.greater-equiv assms larger-heapI map-add-find-right plus-charact*(*2*))

lemma *compatibleI*:
assumes *compatible-heaps* (*get-h a*) (*get-h b*)
and *valid-mask* (*add-masks* (*get-m a*) (*get-m b*))
shows *a* |#| *b*
using *PartialSA.defined-def assms*(*1*) *assms*(*2*) *plus-ab-defined* **by** *presburger*

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* ⇒ *state* ⇒ *state* **where**
multiply p φ = *Abs-state* (*multiply-mask p* (*get-m φ*), *get-h φ*)

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

fun *multiply-sem-assertion* :: *prat* ⇒ *sem-assertion* ⇒ *sem-assertion* **where**
multiply-sem-assertion p P = *PartialSA.upper-closure* (*multiply p* ‘ *P*)

definition *combinable* :: *sem-assertion* ⇒ *bool* **where**
combinable P ⟷ (∀ *α β*. *ppos α* ∧ *ppos β* ∧ *pgte pwrite* (*padd α β*) ⟶ (*multiply-sem-assertion α P*) ⊗ (*multiply-sem-assertion β P*) ⊆ *multiply-sem-assertion* (*padd α β*) *P*)

definition *scaled* **where**
scaled φ = { *multiply p φ* |*p*. *ppos p* ∧ *pgte pwrite p* }

definition *comp-min-mask* :: *mask* ⇒ (*mask* ⇒ *mask*) **where**
comp-min-mask b a hl = *pmin* (*a hl*) (*comp-one* (*b hl*))

definition *scalable* **where**
scalable w a ⟷ (∀ *φ* ∈ *scaled w*. ¬ *a* |#| *φ*)

definition *R* **where**
R a w = (*if scalable w a then w else Abs-state* (*comp-min-mask* (*get-m a*) (*get-m w*), *get-h w*))

definition *cwand* **where**
cwand A B = { *w* |*w*. ∀ *a x*. *a* ∈ *A* ∧ *Some x* = *R a w* ⊕ *a* ⟶ *x* ∈ *B* }

definition *wand* :: *sem-assertion* ⇒ *sem-assertion* ⇒ *sem-assertion* **where**
wand A B = { *w* |*w*. ∀ *a x*. *a* ∈ *A* ∧ *Some x* = *w* ⊕ *a* ⟶ *x* ∈ *B* }

definition *intuitionistic* **where**
intuitionistic A ⟷ (∀ *a b*. *a* ⪰ *b* ∧ *b* ∈ *A* ⟶ *a* ∈ *A*)

definition *binary-mask* :: *mask* ⇒ *mask* **where**

binary-mask π *l* = (*if* π *l* = *pwrite then pwrite else pnone*)

definition *binary* :: *sem-assertion* ⇒ *bool* **where**
binary A ⟷ (∀ φ ∈ *A*. *Abs-state* (*binary-mask* (*get-m* φ), *get-h* φ) ∈ *A*)

2.2 Lemmas

lemma *wand-equiv-def*:
wand A B = { φ |φ. *A* ⊗ {φ} ⊆ *B* }
proof
show *wand A B* ⊆ {φ |φ. *A* ⊗ {φ} ⊆ *B*}
proof
fix *w* **assume** *w* ∈ *wand A B*
have *A* ⊗ {*w*} ⊆ *B*
proof
fix *x* **assume** *x* ∈ *A* ⊗ {*w*}
then show *x* ∈ *B*
using *PartialSA.add-set-elem* ‹*w* ∈ *wand A B*› *commutative wand-def* **by** *auto*
qed
then show *w* ∈ {φ |φ. *A* ⊗ {φ} ⊆ *B*}
by *simp*
qed
show {φ |φ. *A* ⊗ {φ} ⊆ *B*} ⊆ *wand A B*
proof
fix *w* **assume** *w* ∈ {φ |φ. *A* ⊗ {φ} ⊆ *B*}
have ⋀*a x*. *a* ∈ *A* ∧ *Some x* = *w* ⊕ *a* ⟹ *x* ∈ *B*
proof −
fix *a x* **assume** *a* ∈ *A* ∧ *Some x* = *w* ⊕ *a*
then have *x* ∈ *A* ⊗ {*w*}
using *PartialSA.add-set-elem PartialSA.commutative* **by** *auto*
then show *x* ∈ *B*
using ‹*w* ∈ {φ |φ. *A* ⊗ {φ} ⊆ *B*}› **by** *blast*
qed
then show *w* ∈ *wand A B*
using *wand-def* **by** *force*
qed
qed

lemma *w-in-scaled*:
w ∈ *scaled w*
proof −
have *multiply pwrite w* = *w*
by (*simp add*: *Rep-state-inverse mult-write-mask*)
then show *?thesis*
by (*metis* (*mono-tags*, *lifting*) *half-between-0-1 half-plus-half mem-Collect-eq not-pgte-charact pgt-implies-pgte ppos-add scaled-def*)
qed

lemma *non-scalable-instantiate*:
  assumes ¬ *scalable w a*
  shows ∃ *p. ppos p* ∧ *pgte pwrite p* ∧ *a* |#| *multiply p w*
  using *assms scalable-def scaled-def* **by** *auto*

lemma *compatible-same-mask*:
  assumes *valid-mask* (*add-masks a w*)
  shows *w* = *comp-min-mask a w*
proof (*rule ext*)
  fix *x*
  have *pgte pwrite* (*padd* (*a x*) (*w x*))
    by (*metis add-masks.simps assms valid-mask.elims*(*1*))
  moreover have *padd* (*a x*) (*comp-one* (*a x*)) = *pwrite*
    by (*meson assms padd-comp-one upper-valid-aux valid-mask.elims*(*1*))
  then have *pgte* (*comp-one* (*a x*)) (*w x*)
    by (*metis add-le-cancel-left calculation padd.rep-eq pgte.rep-eq*)
  then show *w x* = *comp-min-mask a w x*
    by (*metis comp-min-mask-def pmin-comm pmin-is*)
qed

lemma *R-smaller*:
  w ⪰ *R a w*
proof (*cases scalable w a*)
  case *True*
  then show *?thesis*
    by (*simp add*: *PartialSA.succ-refl R-def*)
next
  case *False*
  then have *R a w* = *Abs-state* (*comp-min-mask* (*get-m a*) (*get-m w*), *get-h w*)
    by (*meson R-def*)
  moreover have *greater-mask* (*get-m w*) (*comp-min-mask* (*get-m a*) (*get-m w*))
  proof (*rule greater-maskI*)
    fix *hl* **show** *pgte* (*get-m w hl*) (*comp-min-mask* (*get-m a*) (*get-m w*) *hl*)
      by (*simp add*: *comp-min-mask-def pmin-greater*)
  qed
  ultimately show *?thesis*
      by (*metis Abs-state-cases larger-heap-refl Rep-state-cases Rep-state-inverse fst-conv get-h-m greaterI greater-mask-def mem-Collect-eq snd-conv valid-state-decompose*)
qed

lemma *R-compatible-same*:
  assumes *a* |#| *w*
  shows *R a w* = *w*
proof −
  have ¬ *scalable w a*
    using *assms scalable-def w-in-scaled* **by** *blast*
  then have *R a w* = *Abs-state* (*comp-min-mask* (*get-m a*) (*get-m w*), *get-h w*)
    using *R-def* **by** *auto*
  then show *?thesis*

```
    by (metis PartialSA.defined-def Rep-state-inverse assms compatible-same-mask
get-h.simps get-m.simps plus-ab-defined prod.collapse)
qed

lemma in-cwand:
  assumes ⋀a x. a ∈ A ∧ Some x = R a w ⊕ a ⟹ x ∈ B
  shows w ∈ cwand A B
  using assms cwand-def by force

lemma wandI:
  assumes ⋀a x. a ∈ A ∧ Some x = a ⊕ w ⟹ x ∈ B
  shows w ∈ wand A B
proof –
  have A ⊗ {w} ⊆ B
  proof (rule subsetI)
    fix x assume x ∈ A ⊗ {w}
    then obtain a where Some x = a ⊕ w a ∈ A
      using PartialSA.add-set-elem by auto
    then show x ∈ B
      using assms by blast
  qed
  then show ?thesis
    using wand-equiv-def by force
qed

lemma non-scalable-R-charact:
  assumes ¬ scalable w a
  shows get-m (R a w) = comp-min-mask (get-m a) (get-m w) ∧ get-h (R a w) =
get-h w
proof –
  have R a w = Abs-state (comp-min-mask (get-m a) (get-m w), get-h w)
    using R-def assms by auto
  moreover have valid-state (comp-min-mask (get-m a) (get-m w), get-h w)
  proof (rule valid-stateI)
    show valid-mask (comp-min-mask (get-m a) (get-m w))
    proof (rule valid-maskI)
      show ⋀f. comp-min-mask (get-m a) (get-m w) (null, f) = pnone
        by (metis (no-types, opaque-lifting) PartialSA.unit-neutral add-masks.simps
comp-min-mask-def option.distinct(1) p-greater-exists padd-zero plus-ab-defined pmin-greater
valid-mask.simps)
      fix hl show pgte pwrite (comp-min-mask (get-m a) (get-m w) hl)
      by (metis PartialSA.unit-neutral comp-min-mask-def greater-mask-def greater-mask-equiv-def
option.distinct(1) plus-ab-defined pmin-greater upper-valid-aux valid-mask.simps)
    qed
    fix hl assume ppos (comp-min-mask (get-m a) (get-m w) hl)
    show get-h w hl ≠ None
    by (metis Rep-state ‹ppos (comp-min-mask (get-m a) (get-m w) hl)› comp-min-mask-def
get-h.simps get-pre(2) mem-Collect-eq p-greater-exists pmin-greater ppos-add prod.collapse
valid-heap-def valid-state.simps)
```

qed
ultimately show *?thesis*
by (*metis Rep-state-cases Rep-state-inverse fst-conv get-h.simps get-m.simps mem-Collect-eq snd-conv*)
qed

lemma *valid-bin*:
valid-state (*binary-mask* (*get-m a*), *get-h a*)
proof (*rule valid-stateI*)
show *valid-mask* (*binary-mask* (*get-m a*))
by (*metis PartialSA.unit-neutral binary-mask-def minus-empty option.discI plus-ab-defined unit-charact*(*2*) *valid-mask.elims*(*2*) *valid-mask.elims*(*3*))
show $\bigwedge$*hl. ppos* (*binary-mask* (*get-m a*) *hl*) $\Longrightarrow$ *get-h a hl* $\neq$ *None*
by (*metis Rep-prat Rep-state binary-mask-def get-h.simps get-pre*(*2*) *leD mem-Collect-eq pnone.rep-eq ppos.rep-eq prod.collapse valid-heap-def valid-state.simps*)
qed

lemma *in-multiply-sem*:
assumes *x* $\in$ *multiply-sem-assertion p A*
shows $\exists$ *a* $\in$ *A*. *x* $\succeq$ *multiply p a*
using *PartialSA.sep-algebra-axioms assms greater-def sep-algebra.upper-closure-def*
by *fastforce*

lemma *get-h-multiply*:
assumes *pgte pwrite p*
shows *get-h* (*multiply p x*) = *get-h x*
using *Abs-state-inverse assms multiply-valid* **by** *auto*

lemma *in-multiply-refl*:
assumes *x* $\in$ *A*
shows *multiply p x* $\in$ *multiply-sem-assertion p A*
using *PartialSA.succ-refl PartialSA.upper-closure-def assms* **by** *fastforce*

lemma *get-m-smaller*:
assumes *pgte pwrite p*
shows *get-m* (*multiply p a*) *hl* = *pmult p* (*get-m a hl*)
using *Abs-state-inverse assms multiply-mask-def multiply-valid* **by** *auto*

lemma *get-m-smaller-mask*:
assumes *pgte pwrite p*
shows *get-m* (*multiply p a*) = *multiply-mask p* (*get-m a*)
using *Abs-state-inverse assms multiply-mask-def multiply-valid* **by** *auto*

lemma *multiply-order*:
assumes *pgte pwrite p*
and *a* $\succeq$ *b*
shows *multiply p a* $\succeq$ *multiply p b*
proof (*rule greaterI*)
show *larger-heap* (*get-h* (*multiply p a*)) (*get-h* (*multiply p b*))

using *assms(1) assms(2) get-h-multiply larger-implies-larger-heap* **by** *presburger*
show *greater-mask (get-m (multiply p a)) (get-m (multiply p b))*
by *(metis assms(1) assms(2) get-m-smaller-mask greater-maskI larger-implies-greater-mask-hl mult-greater)*
qed

lemma *multiply-twice*:
assumes *pgte pwrite a* $\wedge$ *pgte pwrite b*
shows *multiply a (multiply b x) = multiply (pmult a b) x*
proof −
have *get-h (multiply (pmult a b) x) = get-h x*
by *(metis assms get-h-multiply p-greater-exists padd-asso pmult-order pmult-special(1))*
moreover have *get-h (multiply a (multiply b x)) = get-h x*
using *assms get-h-multiply* **by** *presburger*
moreover have *get-m (multiply a (multiply b x)) = get-m (multiply (pmult a b) x)*
proof *(rule ext)*
fix *l*
have *pgte pwrite (pmult a b)* **using** *multiply-smaller-pwrite assms* **by** *simp*
then have *get-m (multiply (pmult a b) x) l = pmult (pmult a b) (get-m x l)*
using *get-m-smaller* **by** *blast*
then show *get-m (multiply a (multiply b x)) l = get-m (multiply (pmult a b) x) l*
by *(metis Rep-prat-inverse assms get-m-smaller mult.assoc pmult.rep-eq)*
qed
ultimately show *?thesis*
using *state-ext* **by** *presburger*
qed

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 *(rule valid-maskI)*
show $\bigwedge f$. *add-masks (comp-min-mask b a) b (null, f) = pnone*
proof −
fix *f*
have *comp-min-mask b a (null, f) = pnone*
by *(metis assms(1) comp-min-mask-def p-greater-exists padd-zero pmin-greater valid-mask.simps)*
then show *add-masks (comp-min-mask b a) b (null, f) = pnone*
by *(metis add-masks.simps assms(2) padd-zero valid-mask.simps)*
qed
fix *hl* **show** *pgte pwrite (add-masks (comp-min-mask b a) b hl)*
proof *(cases pgte (a hl) (comp-one (b hl)))*
case *True*
then have *add-masks (comp-min-mask b a) b hl = padd (comp-one (b hl)) (b hl)*
by *(simp add: comp-min-mask-def pmin-is)*

then have *add-masks* (*comp-min-mask b a*) *b hl* = *pwrite*
  by (*metis assms*(*2*) *padd-comm padd-comp-one valid-mask.simps*)
then show *?thesis*
  by (*simp add*: *pgte.rep-eq*)
next
case *False*
then have *comp-min-mask b a hl* = *a hl*
  by (*metis comp-min-mask-def not-pgte-charact pgt-implies-pgte pmin-comm pmin-is*)
then have *add-masks* (*comp-min-mask b a*) *b hl* = *padd* (*a hl*) (*b hl*)
  by *auto*
moreover have *pgte* (*padd* (*comp-one* (*b hl*)) (*b hl*)) (*padd* (*a hl*) (*b hl*))
  using *False padd.rep-eq pgte.rep-eq* **by** *force*
moreover have *padd* (*comp-one* (*b hl*)) (*b hl*) = *pwrite*
  by (*metis assms*(*2*) *padd-comm padd-comp-one valid-mask.simps*)
ultimately show *?thesis* **by** *simp*
qed
qed

2.3 The combinable wand is stronger than the original wand

lemma *cwand-stronger*:
  cwand A B ⊆ *wand A B*
proof
  fix *w* **assume** *asm0*: $w \in$ *cwand A B*
  then have *r*: $\bigwedge a\ x.\ a \in A \wedge Some\ x = R\ a\ w \oplus a \Longrightarrow x \in B$
    using *cwand-def* **by** *blast*
  show $w \in$ *wand A B*
  proof (*rule wandI*)
    fix *a x* **assume** *asm1*: $a \in A \wedge Some\ x = a \oplus w$
    then have $R\ a\ w = w$
      by (*metis PartialSA.defined-def R-compatible-same option.distinct*(*1*))
    then show $x \in B$
      by (*metis PartialSA.commutative asm1 r*)
  qed
qed

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* ⊆ *cwand A B*
proof (*rule subsetI*)
  fix *w* **assume** $w \in$ *wand A B*
  then have *asm0*: $A \otimes \{w\} \subseteq B$
    by (*simp add*: *wand-equiv-def*)
  show $w \in$ *cwand A B*

proof (*rule in-cwand*)
fix *a x* **assume** *asm1*: $a \in A \wedge$ *Some* $x = R\ a\ w \oplus a$
show $x \in B$
proof (*cases scalable w a*)
case *True*
then show *?thesis*
by (*metis PartialSA.commutative PartialSA.defined-def R-def asm1 option.distinct*(*1*) *scalable-def w-in-scaled*)
next
case *False*
then have *r0*: *get-m* (*R a w*) = *comp-min-mask* (*get-m a*) (*get-m w*) ∧ *get-h* (*R a w*) = *get-h w*
using *non-scalable-R-charact* **by** *blast*
moreover have *Abs-state* (*binary-mask* (*get-m a*), *get-h a*) ∈ *A*
using *asm1 assms*(*1*) *binary-def* **by** *blast*
moreover have *greater-mask* (*add-masks* (*comp-min-mask* (*get-m a*) (*get-m w*)) (*get-m a*))
(*add-masks* (*binary-mask* (*get-m a*)) (*get-m w*))
proof (*rule greater-maskI*)
fix *hl* **show** *pgte* (*add-masks* (*comp-min-mask* (*get-m a*) (*get-m w*)) (*get-m a*) *hl*) (*add-masks* (*binary-mask* (*get-m a*)) (*get-m w*) *hl*)
proof (*cases get-m a hl* = *pwrite*)
case *True*
obtain φ **where** φ ∈ *scaled w a* |#| φ **using** *False scalable-def*[*of w a*]
by *blast*
then obtain *p* **where** *ppos p pgte pwrite p multiply p w* |#| *a*
using *PartialSA.commutative PartialSA.defined-def mem-Collect-eq scaled-def* **by** *auto*
have *get-m w hl* = *pnone*
proof (*rule ccontr*)
assume *get-m w hl* ≠ *pnone*
then have *ppos* (*get-m w hl*)
by (*metis less-add-same-cancel1 not-pgte-charact p-greater-exists padd.rep-eq padd-zero pgt.rep-eq ppos.rep-eq*)
moreover have *get-m* (*multiply p* φ) = *multiply-mask p* (*get-m* φ)
using *multiply-valid*[*of p* φ] *multiply.simps*[*of p* φ]
by (*metis Rep-state-cases Rep-state-inverse* ‹*pgte pwrite p*› *fst-conv get-pre*(*2*) *mem-Collect-eq*)
then have *ppos* (*get-m* (*multiply p w*) *hl*) **using** *pmult-ppos*
by (*metis Rep-state-cases Rep-state-inverse* ‹*pgte pwrite p*› ‹*ppos p*› *calculation fst-conv get-pre*(*2*) *mem-Collect-eq multiply.simps multiply-mask-def multiply-valid*)
then have *pgt* (*padd* (*get-m* (*multiply p w*) *hl*) (*get-m a hl*)) *pwrite*
by (*metis True add-le-same-cancel2 leD not-pgte-charact padd.rep-eq pgte.rep-eq ppos.rep-eq*)
then have ¬ *valid-mask* (*add-masks* (*get-m* (*multiply p w*)) (*get-m a*))
by (*metis add-masks.elims not-pgte-charact valid-mask.elims*(*1*))
then show *False*
using *PartialSA.defined-def* ‹*multiply p w* |#| *a*› *plus-ab-defined* **by**

blast
qed
then show *?thesis*
by (*metis Rep-prat-inverse add.right-neutral add-masks.simps binary-mask-def p-greater-exists padd.rep-eq padd-comm pnone.rep-eq*)
next
case *False*
then have *add-masks* (*binary-mask* (*get-m a*)) (*get-m w*) *hl* = *get-m w hl*
by (*metis Rep-prat-inject add.right-neutral add-masks.simps binary-mask-def padd.rep-eq padd-comm pnone.rep-eq*)
then show *?thesis*
proof (*cases pgte* (*get-m w hl*) (*comp-one* (*get-m a hl*)))
case *True*
then have *comp-min-mask* (*get-m a*) (*get-m w*) *hl* = *comp-one* (*get-m a hl*)
using *comp-min-mask-def pmin-is* **by** *presburger*
then have *add-masks* (*comp-min-mask* (*get-m a*) (*get-m w*)) (*get-m a*) *hl* = *pwrite*
by (*metis PartialSA.unit-neutral add-masks.simps add-masks-comm minus-empty option.distinct*(*1*) *padd-comp-one plus-ab-defined unit-charact*(*2*) *valid-mask.simps*)
then show *?thesis*
by (*metis PartialSA.unit-neutral* ‹*add-masks* (*binary-mask* (*get-m a*)) (*get-m w*) *hl* = *get-m w hl*› *minus-empty option.distinct*(*1*) *plus-ab-defined unit-charact*(*2*) *valid-mask.simps*)
next
case *False*
then have *comp-min-mask* (*get-m a*) (*get-m w*) *hl* = *get-m w hl*
by (*metis comp-min-mask-def not-pgte-charact pgt-implies-pgte pmin-comm pmin-is*)
then show *?thesis*
using ‹*add-masks* (*binary-mask* (*get-m a*)) (*get-m w*) *hl* = *get-m w hl*› *p-greater-exists* **by** *auto*
qed
qed
qed
then have *valid-mask* (*add-masks* (*binary-mask* (*get-m a*)) (*get-m w*))
by (*metis asm1 calculation*(*1*) *greater-mask-def option.distinct*(*1*) *plus-ab-defined upper-valid-aux*)
moreover have *compatible-heaps* (*get-h a*) (*get-h w*)
by (*metis PartialSA.commutative asm1 r0 option.simps*(*3*) *plus-ab-defined*)
then obtain *xx* **where** *Some xx* = *Abs-state* (*binary-mask* (*get-m a*), *get-h a*) ⊕ *w*
using *Abs-state-inverse calculation compatible-def fst-conv plus-def valid-bin* **by** *auto*
then have *xx* ∈ *B* **using** *asm0*
by (*meson PartialSA.add-set-elem* ‹*Abs-state* (*binary-mask* (*get-m a*), *get-h a*) ∈ *A*› *singletonI subset-iff*)
moreover have *x* ⪰ *xx*
proof (*rule greaterI*)

show *greater-mask* (*get-m x*) (*get-m xx*)
using *Abs-state-inverse* ‹*Some xx* = *Abs-state* (*binary-mask* (*get-m a*), *get-h a*) ⊕ *w*› *asm1* ‹*greater-mask* (*add-masks* (*comp-min-mask* (*get-m a*) (*get-m w*)) (*get-m a*)) (*add-masks* (*binary-mask* (*get-m a*)) (*get-m w*))› *calculation*(*1*) *plus-charact*(*1*) *valid-bin* **by** *auto*
show *larger-heap* (*get-h x*) (*get-h xx*)
proof (*rule larger-heapI*)
fix *hl xa* **assume** *get-h xx hl* = *Some xa*
then show *get-h x hl* = *Some xa*
by (*metis PartialSA.commutative Rep-state-cases Rep-state-inverse* ‹*Some xx* = *Abs-state* (*binary-mask* (*get-m a*), *get-h a*) ⊕ *w*› *asm1 calculation*(*1*) *get-h.simps mem-Collect-eq plus-charact*(*2*) *snd-conv valid-bin*)
qed
qed
ultimately show *?thesis*
using *assms*(*2*) *intuitionistic-def* **by** *blast*
qed
qed
qed

2.5 The combinable wand is combinable

lemma *combinableI*:
assumes $\bigwedge$*a b*. *ppos a* ∧ *ppos b* ∧ *padd a b* = *pwrite* ⟹ (*multiply-sem-assertion a* (*cwand A B*)) ⊗ (*multiply-sem-assertion b* (*cwand A B*)) ⊆ *cwand A B*
shows *combinable* (*cwand A B*)
proof −
have $\bigwedge$*a b*. *ppos a* ∧ *ppos b* ∧ *pgte pwrite* (*padd a b*) ⟹ (*multiply-sem-assertion a* (*cwand A B*)) ⊗ (*multiply-sem-assertion b* (*cwand A B*)) ⊆ *multiply-sem-assertion* (*padd a b*) (*cwand A B*)
proof −
fix *a b* **assume** *asm0*: *ppos a* ∧ *ppos b* ∧ *pgte pwrite* (*padd a b*)
then have *pgte pwrite a* ∧ *pgte pwrite b*
using *padd.rep-eq pgte.rep-eq ppos.rep-eq* **by** *auto*
show (*multiply-sem-assertion a* (*cwand A B*)) ⊗ (*multiply-sem-assertion b* (*cwand A B*)) ⊆ *multiply-sem-assertion* (*padd a b*) (*cwand A B*)
proof
fix *x* **assume** *x* ∈ *multiply-sem-assertion a* (*cwand A B*) ⊗ *multiply-sem-assertion b* (*cwand A B*)
then obtain *xa xb* **where** *Some x* = *xa* ⊕ *xb xa* ∈ *multiply-sem-assertion a* (*cwand A B*) *xb* ∈ *multiply-sem-assertion b* (*cwand A B*)
by (*meson PartialSA.add-set-elem*)
then obtain *wa wb* **where** *wa* ∈ *cwand A B wb* ∈ *cwand A B xa* ⪰ *multiply a wa xb* ⪰ *multiply b wb*
by (*meson in-multiply-sem*)
let *?a* = *pdiv a* (*padd a b*)
let *?b* = *pdiv b* (*padd a b*)
have *r0*: *pgte pwrite ?a* ∧ *pgte pwrite ?b*
using *asm0 p-greater-exists padd-comm pdiv-smaller ppos-add* **by** *blast*

have *multiply ?a wa* |#| *multiply ?b wb*
proof (*rule compatibleI*)
show *compatible-heaps* (*get-h* (*multiply* (*pdiv a* (*padd a b*)) *wa*)) (*get-h* (*multiply* (*pdiv b* (*padd a b*)) *wb*))
proof −
have *compatible-heaps* (*get-h* (*multiply a wa*)) (*get-h* (*multiply b wb*))
by (*metis PartialSA.asso2 PartialSA.asso3 PartialSA.greater-equiv PartialSA.minus-some* ‹*Some x* = *xa* ⊕ *xb*› ‹*xa* ⪰ *multiply a wa*› ‹*xb* ⪰ *multiply b wb*› *option.simps*(*3*) *plus-ab-defined*)
moreover have *get-h* (*multiply* (*pdiv a* (*padd a b*)) *wa*) = *get-h* (*multiply a wa*) ∧ *get-h* (*multiply* (*pdiv b* (*padd a b*)) *wb*) = *get-h* (*multiply b wb*)
proof −
have *pgte pwrite a* ∧ *pgte pwrite b*
by (*metis asm0 p-greater-exists padd-asso padd-comm*)
moreover have *pgte pwrite ?a* ∧ *pgte pwrite ?b*
using *asm0 p-greater-exists padd-comm pdiv-smaller ppos-add* **by** *blast*
ultimately show *?thesis*
using *get-h-multiply* **by** *presburger*
qed
then show *?thesis*
using *calculation* **by** *presburger*
qed
show *valid-mask* (*add-masks* (*get-m* (*multiply* (*pdiv a* (*padd a b*)) *wa*)) (*get-m* (*multiply* (*pdiv b* (*padd a b*)) *wb*)))
proof (*rule valid-maskI*)
show $\bigwedge f$. *add-masks* (*get-m* (*multiply* (*pdiv a* (*padd a b*)) *wa*)) (*get-m* (*multiply* (*pdiv b* (*padd a b*)) *wb*)) (*null*, *f*) = *pnone*
by (*metis PartialSA.unit-neutral add-masks-equiv-valid-null option.distinct*(*1*) *plus-ab-defined valid-mask.simps valid-null-def*)
fix *hl* **have** *add-masks* (*get-m* (*multiply* (*pdiv a* (*padd a b*)) *wa*)) (*get-m* (*multiply* (*pdiv b* (*padd a b*)) *wb*)) *hl*
= *padd* (*pmult ?a* (*get-m wa hl*)) (*pmult ?b* (*get-m wb hl*))
proof −
have *get-m* (*multiply ?a wa*) *hl* = *pmult ?a* (*get-m wa hl*)
using *Abs-state-inverse r0 multiply-mask-def multiply-valid* **by** *auto*
moreover have *get-m* (*multiply ?b wb*) *hl* = *pmult ?b* (*get-m wb hl*)
using *Abs-state-inverse r0 multiply-mask-def multiply-valid* **by** *auto*
ultimately show *?thesis* **by** *simp*
qed
moreover have *pgte pwrite* (*padd* (*pmult ?a* (*get-m wa hl*)) (*pmult ?b* (*get-m wb hl*)))
proof (*rule padd-one-ineq-sum*)
show *pgte pwrite* (*get-m wa hl*)
by (*metis PartialSA.unit-neutral option.discI plus-ab-defined upper-valid-aux valid-mask.simps*)
show *pgte pwrite* (*get-m wb hl*)
by (*metis PartialSA.unit-neutral option.discI plus-ab-defined upper-valid-aux valid-mask.simps*)
show *padd* (*pdiv a* (*padd a b*)) (*pdiv b* (*padd a b*)) = *pwrite*

using *asm0 sum-coeff* **by** *blast*
qed
ultimately show *pgte pwrite* (*add-masks* (*get-m* (*multiply* (*pdiv a* (*padd a b*)) *wa*)) (*get-m* (*multiply* (*pdiv b* (*padd a b*)) *wb*)) *hl*)
by *presburger*
qed
qed
then obtain *xx* **where** *xx-def*: *Some xx* = *multiply ?a wa* ⊕ *multiply ?b wb*

using *PartialSA.defined-def* **by** *auto*
moreover have *inclusion*: (*multiply-sem-assertion ?a* (*cwand A B*)) ⊗ (*multiply-sem-assertion ?b* (*cwand A B*)) ⊆ *cwand A B*
proof (*rule assms*)
show *ppos* (*pdiv a* (*padd a b*)) ∧ *ppos* (*pdiv b* (*padd a b*)) ∧ *padd* (*pdiv a* (*padd a b*)) (*pdiv b* (*padd a b*)) = *pwrite*
using *asm0 padd.rep-eq pdiv.rep-eq ppos.rep-eq sum-coeff* **by** *auto*
qed
ultimately have *xx* ∈ *cwand A B*
proof −
have *multiply ?a wa* ∈ *multiply-sem-assertion ?a* (*cwand A B*)
using ‹*wa* ∈ *cwand A B*› *in-multiply-refl* **by** *presburger*
moreover have *multiply ?b wb* ∈ *multiply-sem-assertion ?b* (*cwand A B*)
by (*meson* ‹*wb* ∈ *cwand A B*› *in-multiply-refl*)
ultimately show *?thesis*
using *PartialSA.add-set-def xx-def inclusion* **by** *fastforce*
qed
moreover have *x* ⪰ *multiply* (*padd a b*) *xx*
proof (*rule greaterI*)
have *valid-state* (*multiply-mask* (*padd a b*) (*get-m xx*), *get-h xx*)
using *asm0 multiply-valid* **by** *blast*
show *larger-heap* (*get-h x*) (*get-h* (*multiply* (*padd a b*) *xx*))
proof −
have *get-h* (*multiply* (*padd a b*) *xx*) = *get-h xx*
using *asm0 get-h-multiply* **by** *blast*
moreover have *get-h xx* = *get-h wa* ++ *get-h wb*
by (*metis xx-def asm0 get-h-multiply p-greater-exists padd-comm plus-charact*(*2*) *sum-coeff*)
moreover have *get-h x* = *get-h xa* ++ *get-h xb*
using ‹*Some x* = *xa* ⊕ *xb*› *plus-charact*(*2*) **by** *presburger*
moreover have *get-h wa* = *get-h* (*multiply a wa*) ∧ *get-h wb* = *get-h* (*multiply b wb*)
by (*metis asm0 get-h-multiply order-trans p-greater-exists padd-comm pgte.rep-eq*)
moreover have *larger-heap* (*get-h xa*) (*get-h wa*) ∧ *larger-heap* (*get-h xb*) (*get-h wb*)
using ‹*xa* ⪰ *multiply a wa*› ‹*xb* ⪰ *multiply b wb*› *calculation*(*4*) *larger-implies-larger-heap* **by** *presburger*
ultimately show *?thesis*
by (*metis* ‹*Some x* = *xa* ⊕ *xb*› *larger-heaps-sum-ineq option.simps*(*3*)

plus-ab-defined)
qed
show *greater-mask* (*get-m x*) (*get-m* (*multiply* (*padd a b*) *xx*))
proof (*rule greater-maskI*)
fix *hl*
have *pgte* (*get-m x hl*) (*padd* (*get-m xa hl*) (*get-m xb hl*))
using ‹*Some x = xa ⊕ xb*› *pgte.rep-eq plus-charact*(*1*) **by** *auto*
moreover have *pgte* (*get-m xa hl*) (*get-m* (*multiply a wa*) *hl*) ∧ *pgte* (*get-m xb hl*) (*get-m* (*multiply b wb*) *hl*)
using ‹*xa* ⪰ *multiply a wa*› ‹*xb* ⪰ *multiply b wb*› *larger-implies-greater-mask-hl* **by** *blast*
moreover have *get-m* (*multiply* (*padd a b*) *xx*) *hl* = *pmult* (*padd a b*) (*get-m xx hl*)
by (*metis Rep-state-cases Rep-state-inverse* ‹*valid-state* (*multiply-mask* (*padd a b*) (*get-m xx*), *get-h xx*)› *fst-conv get-pre*(*2*) *mem-Collect-eq multiply.simps multiply-mask-def*)
moreover have ... = *padd* (*pmult* (*pmult* (*padd a b*) *?a*) (*get-m wa hl*)) (*pmult* (*pmult* (*padd a b*) *?b*) (*get-m wb hl*))
proof −
have *get-m* (*multiply ?a wa*) *hl* = *pmult ?a* (*get-m wa hl*)
by (*metis Abs-state-inverse asm0 fst-conv get-pre*(*2*) *mem-Collect-eq multiply.simps multiply-mask-def multiply-valid p-greater-exists sum-coeff*)
moreover have *get-m* (*multiply ?b wb*) *hl* = *pmult ?b* (*get-m wb hl*)
by (*metis Abs-state-inverse asm0 fst-conv get-pre*(*2*) *mem-Collect-eq multiply.simps multiply-mask-def multiply-valid p-greater-exists padd-comm pdiv-smaller ppos-add*)
ultimately have *get-m xx hl* = *padd* (*pmult ?a* (*get-m wa hl*)) (*pmult ?b* (*get-m wb hl*))
using *xx-def plus-charact*(*1*) **by** *fastforce*
then show *?thesis*
by (*simp add*: *pmult-padd*)
qed
moreover have ... = *padd* (*pmult a* (*get-m wa hl*)) (*pmult b* (*get-m wb hl*))
using *asm0 pmult-pdiv-cancel ppos-add* **by** *presburger*
moreover have *get-m* (*multiply a wa*) *hl* = *pmult a* (*get-m wa hl*) ∧ *get-m* (*multiply b wb*) *hl* = *pmult b* (*get-m wb hl*)
proof −
have *valid-mask* (*multiply-mask a* (*get-m wa*))
using *asm0 mult-add-states multiply-valid upper-valid-aux valid-state.simps* **by** *blast*
moreover have *valid-mask* (*multiply-mask b* (*get-m wb*))
using *asm0 mult-add-states multiply-valid upper-valid valid-state.simps* **by** *blast*
ultimately show *?thesis*
by (*metis* (*no-types, lifting*) *Abs-state-inverse asm0 fst-conv get-pre*(*2*) *mem-Collect-eq multiply.simps multiply-mask-def multiply-valid order-trans p-greater-exists padd-comm pgte.rep-eq*)
qed

ultimately show *pgte* (*get-m x hl*) (*get-m* (*multiply* (*padd a b*) *xx*) *hl*)
by (*simp add*: *padd.rep-eq pgte.rep-eq*)
qed
qed
ultimately show $x \in$ *multiply-sem-assertion* (*padd a b*) (*cwand A B*)
by (*metis PartialSA.up-closed-def PartialSA.upper-closure-up-closed in-multiply-refl multiply-sem-assertion.simps*)
qed
qed
then show *?thesis*
using *combinable-def* **by** *presburger*
qed

lemma *combinable-cwand*:
assumes *combinable B*
and *intuitionistic B*
shows *combinable* (*cwand A B*)
proof (*rule combinableI*)
fix α β **assume** *asm0*: *ppos* $\alpha \wedge$ *ppos* $\beta \wedge$ *padd* α β = *pwrite*
then have *pgte pwrite* $\alpha \wedge$ *pgte pwrite* β
by (*metis p-greater-exists padd-comm*)
show *multiply-sem-assertion* α (*cwand A B*) $\otimes$ *multiply-sem-assertion* β (*cwand A B*) $\subseteq$ *cwand A B*
proof
fix *w* **assume** *asm*: $w \in$ *multiply-sem-assertion* α (*cwand A B*) $\otimes$ *multiply-sem-assertion* β (*cwand A B*)
then obtain *xa xb* **where** *Some w* = *xa* $\oplus$ *xb* *xa* $\in$ *multiply-sem-assertion* α (*cwand A B*) *xb* $\in$ *multiply-sem-assertion* β (*cwand A B*)
by (*meson PartialSA.add-set-elem*)
then obtain *wa wb* **where** *wa* $\in$ *cwand A B* *wb* $\in$ *cwand A B* *xa* $\succeq$ *multiply* α *wa* *xb* $\succeq$ *multiply* β *wb*
by (*meson in-multiply-sem*)
then obtain *r*: $\bigwedge a\ x.\ a \in A \wedge$ *Some x* = *R a wa* $\oplus$ *a* $\Longrightarrow x \in B$ $\bigwedge a\ x.\ a \in A \wedge$ *Some x* = *R a wb* $\oplus$ *a* $\Longrightarrow x \in B$
using *cwand-def* **by** *blast*
show $w \in$ *cwand A B*
proof (*rule in-cwand*)
fix *a x* **assume** *asm1*: $a \in A \wedge$ *Some x* = *R a w* $\oplus$ *a*
have $\neg$ *scalable w a*
proof (*rule ccontr*)
assume $\neg$ $\neg$ *scalable w a*
then have *R a w* = *w* $\wedge$ $\neg$ *a* |#| *R a w*
by (*simp add*: *R-def scalable-def w-in-scaled*)
then show *False*
using *PartialSA.commutative PartialSA.defined-def asm1* **by** *auto*
qed
then have *r3*: *get-h* (*R a w*) = *get-h w* $\wedge$ *get-m* (*R a w*) = *comp-min-mask* (*get-m a*) (*get-m w*)
using *non-scalable-R-charact* **by** *blast*

moreover obtain *p* **where** *a* |#| *multiply p w ppos p* ∧ *pgte pwrite p*
using ‹¬ *scalable w a*› *non-scalable-instantiate* **by** *blast*
moreover have ¬ *scalable wa a*
proof −
have *a* |#| *multiply* (*pmult* α *p*) *wa*
proof −
have *w* ⪰ *xa* **using** ‹*Some w* = *xa* ⊕ *xb*› **using** *PartialSA.greater-def* **by** *blast*
then have *multiply p w* ⪰ *multiply p xa*
using *calculation*(*3*) *multiply-order* **by** *blast*
then have *multiply p w* ⪰ *multiply* (*pmult* α *p*) *wa*
proof −
have *multiply p w* ⪰ *multiply p* (*multiply* α *wa*)
using *PartialSA.succ-trans* ‹*w* ⪰ *xa*› ‹*xa* ⪰ *multiply* α *wa*› *calculation*(*3*) *multiply-order* **by** *blast*
then show *?thesis*
using ‹*pgte pwrite* α ∧ *pgte pwrite* β› *calculation*(*3*) *multiply-twice pmult-comm* **by** *auto*
qed
then show *?thesis*
using *PartialSA.asso3 PartialSA.defined-def PartialSA.minus-some calculation*(*2*) **by** *fastforce*
qed
moreover have *ppos* (*pmult* α *p*) ∧ *pgte pwrite* (*pmult* α *p*)
by (*metis Rep-prat-inverse* ‹*ppos p* ∧ *pgte pwrite p*› *add.right-neutral asm0 dual-order.strict-iff-order padd.rep-eq pgte.rep-eq pmult-comm pmult-ppos pmult-special*(*2*) *pnone.rep-eq ppos.rep-eq ppos-eq-pnone padd-one-ineq-sum*)
ultimately show *?thesis*
using *scalable-def scaled-def* **by** *auto*
qed
then have *r1*: *get-h* (*R a wa*) = *get-h wa* ∧ *get-m* (*R a wa*) = *comp-min-mask* (*get-m a*) (*get-m wa*)
using *non-scalable-R-charact* **by** *blast*
moreover have *R a wa* |#| *a*
proof (*rule compatibleI*)
have *larger-heap* (*get-h w*) (*get-h xa*) ∧ *larger-heap* (*get-h xa*) (*get-h wa*)
by (*metis PartialSA.commutative PartialSA.greater-equiv* ‹*Some w* = *xa* ⊕ *xb*› ‹*pgte pwrite* α ∧ *pgte pwrite* β› ‹*xa* ⪰ *multiply* α *wa*› *get-h-multiply larger-implies-larger-heap*)
then show *compatible-heaps* (*get-h* (*R a wa*)) (*get-h a*)
by (*metis asm1 calculation*(*1*) *calculation*(*4*) *larger-heap-comp option.distinct*(*1*) *plus-ab-defined*)
show *valid-mask* (*add-masks* (*get-m* (*R a wa*)) (*get-m a*))
by (*metis PartialSA.unit-neutral calculation*(*4*) *minus-empty option.distinct*(*1*) *plus-ab-defined unit-charact*(*2*) *valid-mask-add-comp-min*)
qed
then obtain *ba* **where** *Some ba* = *R a wa* ⊕ *a*
using *PartialSA.defined-def* **by** *auto*

moreover have ¬ *scalable wb a*
proof −
have *a* |#| *multiply* (*pmult* β *p*) *wb*
proof −
have *w* ⪰ *xb* **using** ‹*Some w* = *xa* ⊕ *xb*›
using *PartialSA.greater-equiv* **by** *blast*
then have *multiply p w* ⪰ *multiply p xb*
using *calculation*(*3*) *multiply-order* **by** *blast*
then have *multiply p w* ⪰ *multiply* (*pmult* β *p*) *wb*
proof −
have *multiply p w* ⪰ *multiply p* (*multiply* β *wb*)
using *PartialSA.succ-trans* ‹*w* ⪰ *xb*› ‹*xb* ⪰ *multiply* β *wb*› *calculation*(*3*) *multiply-order* **by** *blast*
then show *?thesis*
using ‹*pgte pwrite* α ∧ *pgte pwrite* β› *calculation*(*3*) *multiply-twice pmult-comm* **by** *auto*
qed
then show *?thesis*
using *PartialSA.asso3 PartialSA.defined-def PartialSA.minus-some calculation*(*2*) **by** *fastforce*
qed
moreover have *ppos* (*pmult* β *p*) ∧ *pgte pwrite* (*pmult* β *p*)
by (*simp add*: ‹*pgte pwrite* α ∧ *pgte pwrite* β› ‹*ppos p* ∧ *pgte pwrite p*› *asm0 multiply-smaller-pwrite pmult-ppos*)
ultimately show *?thesis*
using *scalable-def scaled-def* **by** *auto*
qed
then have *r2*: *get-h* (*R a wb*) = *get-h wb* ∧ *get-m* (*R a wb*) = *comp-min-mask* (*get-m a*) (*get-m wb*)
using *non-scalable-R-charact* **by** *blast*
moreover have *R a wb* |#| *a*
proof (*rule compatibleI*)
have *larger-heap* (*get-h w*) (*get-h xb*) ∧ *larger-heap* (*get-h xb*) (*get-h wb*)
using ‹*Some w* = *xa* ⊕ *xb*› ‹*pgte pwrite* α ∧ *pgte pwrite* β› ‹*xb* ⪰ *multiply* β *wb*› *get-h-multiply larger-heap-def larger-implies-larger-heap plus-charact*(*2*) **by** *fastforce*
then show *compatible-heaps* (*get-h* (*R a wb*)) (*get-h a*)
by (*metis asm1 calculation*(*1*) *calculation*(*6*) *larger-heap-comp option.simps*(*3*) *plus-ab-defined*)
show *valid-mask* (*add-masks* (*get-m* (*R a wb*)) (*get-m a*))
by (*metis PartialSA.unit-neutral calculation*(*6*) *minus-empty option.distinct*(*1*) *plus-ab-defined unit-charact*(*2*) *valid-mask-add-comp-min*)
qed
then obtain *bb* **where** *Some bb* = *R a wb* ⊕ *a*
using *PartialSA.defined-def* **by** *auto*

moreover obtain *ya* **where** *Some ya* = *R a wa* ⊕ *a*
using *calculation*(*5*) **by** *auto*
then have *ya* ∈ *B*

using *asm1 r*(*1*) **by** *blast*
then have *multiply* α *ya* ∈ *multiply-sem-assertion* α *B*
using *in-multiply-refl* **by** *blast*
moreover obtain *yb* **where** *Some yb* = *R a wb* ⊕ *a*
using *calculation*(*7*) **by** *auto*
then have *yb* ∈ *B*
using *asm1 r*(*2*) **by** *blast*
then have *multiply* β *yb* ∈ *multiply-sem-assertion* β *B*
using *in-multiply-refl* **by** *blast*
moreover have (*multiply* α *ya*) |#| (*multiply* β *yb*)
proof (*rule compatibleI*)
have *get-h ya* = *get-h wa* ++ *get-h a*
using ‹*Some ya* = *R a wa* ⊕ *a*› *r1 plus-charact*(*2*) **by** *presburger*
then have *get-h* (*multiply* α *ya*) = *get-h wa* ++ *get-h a*
using ‹*pgte pwrite* α ∧ *pgte pwrite* β› *get-h-multiply* **by** *presburger*
moreover have *get-h yb* = *get-h wb* ++ *get-h a*
using ‹*Some yb* = *R a wb* ⊕ *a*› *r2 plus-charact*(*2*) **by** *presburger*
then have *get-h* (*multiply* β *yb*) = *get-h wb* ++ *get-h a*
using ‹*pgte pwrite* α ∧ *pgte pwrite* β› *get-h-multiply* **by** *presburger*
moreover have *compatible-heaps* (*get-h wa*) (*get-h wb*)
proof (*rule compatible-heapsI*)
fix *hl a b* **assume** *get-h wa hl* = *Some a get-h wb hl* = *Some b*
then have *get-h xa hl* = *Some a get-h xb hl* = *Some b*
apply (*metis* (*full-types*) ‹*pgte pwrite* α ∧ *pgte pwrite* β› ‹*xa* ⪰ *multiply* α *wa*› *get-h-multiply larger-heap-def larger-implies-larger-heap*)
by (*metis* ‹*get-h wb hl* = *Some b*› ‹*pgte pwrite* α ∧ *pgte pwrite* β› ‹*xb* ⪰ *multiply* β *wb*› *get-h-multiply larger-heap-def larger-implies-larger-heap*)
moreover have *compatible-heaps* (*get-h xa*) (*get-h xb*)
by (*metis* ‹*Some w* = *xa* ⊕ *xb*› *option.simps*(*3*) *plus-ab-defined*)
ultimately show *a* = *b*
by (*metis compatible-heaps-def compatible-options.simps*(*1*))
qed
ultimately show *compatible-heaps* (*get-h* (*multiply* α *ya*)) (*get-h* (*multiply* β *yb*))
by (*metis PartialSA.commutative PartialSA.core-is-smaller* ‹*Some ya* = *R a wa* ⊕ *a*› ‹*Some yb* = *R a wb* ⊕ *a*›
r1 r2 compatible-heaps-sum core-defined(*1*) *core-defined*(*2*) *option.distinct*(*1*) *plus-ab-defined*)
show *valid-mask* (*add-masks* (*get-m* (*multiply* α *ya*)) (*get-m* (*multiply* β *yb*)))
proof (*rule valid-maskI*)
show ⋀*f*. *add-masks* (*get-m* (*multiply* α *ya*)) (*get-m* (*multiply* β *yb*)) (*null*, *f*) = *pnone*
by (*metis* (*no-types*, *opaque-lifting*) *PartialSA.core-is-smaller add-masks.simps core-defined*(*2*) *minus-empty not-None-eq plus-ab-defined valid-mask.simps*)
fix *hl*
have *add-masks* (*get-m* (*multiply* α *ya*)) (*get-m* (*multiply* β *yb*)) *hl* = *padd* (*pmult* α (*get-m ya hl*)) (*pmult* β (*get-m yb hl*))
using ‹*pgte pwrite* α ∧ *pgte pwrite* β› *get-m-smaller* **by** *auto*

moreover have *get-m ya hl* = *padd* (*get-m* (*R a wa*) *hl*) (*get-m a hl*) ∧ *get-m yb hl* = *padd* (*get-m* (*R a wb*) *hl*) (*get-m a hl*)
using ‹*Some ya* = *R a wa* ⊕ *a*› ‹*Some yb* = *R a wb* ⊕ *a*› *plus-charact*(*1*) **by** *auto*
ultimately show *pgte pwrite* (*add-masks* (*get-m* (*multiply* α *ya*)) (*get-m* (*multiply* β *yb*)) *hl*)
by (*metis PartialSA.unit-neutral asm0 option.distinct*(*1*) *padd-one-ineq-sum plus-ab-defined plus-charact*(*1*) *valid-mask.simps*)
qed
qed
then obtain *y* **where** *Some y* = *multiply* α *ya* ⊕ *multiply* β *yb*
using *PartialSA.defined-def* **by** *auto*
moreover have *x* ⪰ *y*
proof (*rule greaterI*)
have *get-h y* = *get-h ya* ++ *get-h yb*
using ‹*pgte pwrite* α ∧ *pgte pwrite* β› *calculation*(*10*) *get-h-multiply plus-charact*(*2*) **by** *presburger*
moreover have *get-h ya* = *get-h wa* ++ *get-h a*
using ‹*Some ya* = *R a wa* ⊕ *a*› *r1 plus-charact*(*2*) **by** *presburger*
moreover have *get-h yb* = *get-h wb* ++ *get-h a*
using ‹*Some yb* = *R a wb* ⊕ *a*› *r2 plus-charact*(*2*) **by** *presburger*
moreover have *larger-heap* (*get-h x*) (*get-h wa*)
proof −
have *larger-heap* (*get-h x*) (*get-h xa*)
by (*metis PartialSA.greater-def* ‹*Some w* = *xa* ⊕ *xb*› *r3 asm1 larger-heap-trans larger-implies-larger-heap*)
moreover have *larger-heap* (*get-h xa*) (*get-h wa*)
by (*metis* ‹*pgte pwrite* α ∧ *pgte pwrite* β› ‹*xa* ⪰ *multiply* α *wa*› *get-h-multiply larger-implies-larger-heap*)
ultimately show *?thesis*
using *larger-heap-trans* **by** *blast*
qed
moreover have *larger-heap* (*get-h x*) (*get-h wb*)
proof −
have *larger-heap* (*get-h x*) (*get-h xb*)
by (*metis PartialSA.greater-def PartialSA.greater-equiv* ‹*Some w* = *xa* ⊕ *xb*› *r3 asm1 larger-heap-trans larger-implies-larger-heap*)
moreover have *larger-heap* (*get-h xb*) (*get-h wb*)
by (*metis* ‹*pgte pwrite* α ∧ *pgte pwrite* β› ‹*xb* ⪰ *multiply* β *wb*› *get-h-multiply larger-implies-larger-heap*)
ultimately show *?thesis*
using *larger-heap-trans* **by** *blast*
qed
moreover have *larger-heap* (*get-h x*) (*get-h a*)
using *PartialSA.greater-equiv asm1 larger-implies-larger-heap* **by** *blast*
ultimately show *larger-heap* (*get-h x*) (*get-h y*)
by (*simp add*: *larger-heap-plus*)
show *greater-mask* (*get-m x*) (*get-m y*)
proof (*rule greater-maskI*)

fix *hl*
have *get-m x hl* = *padd* (*get-m* (*R a w*) *hl*) (*get-m a hl*)
using *asm1 plus-charact*(*1*) **by** *auto*
moreover have *get-m y hl* = *padd* (*pmult* α (*padd* (*get-m* (*R a wa*) *hl*) (*get-m a hl*))) (*pmult* β (*padd* (*get-m* (*R a wb*) *hl*) (*get-m a hl*)))
by (*metis* ‹*Some y* = *multiply* α *ya* ⊕ *multiply* β *yb*› ‹*Some ya* = *R a wa* ⊕ *a*› ‹*Some yb* = *R a wb* ⊕ *a*› ‹*pgte pwrite* α ∧ *pgte pwrite* β› *add-masks.simps get-m-smaller plus-charact*(*1*))

moreover have *equ*: *padd* (*pmult* α (*padd* (*get-m* (*R a wa*) *hl*) (*get-m a hl*))) (*pmult* β (*padd* (*get-m* (*R a wb*) *hl*) (*get-m a hl*)))
= *padd* (*padd* (*pmult* α (*get-m a hl*)) (*pmult* β (*get-m a hl*))) (*padd* (*pmult* α (*get-m* (*R a wa*) *hl*)) (*pmult* β (*get-m* (*R a wb*) *hl*)))
using *padd-asso padd-comm pmult-distr* **by** *force*

have *pgte* (*get-m* (*R a w*) *hl*) (*padd* (*pmult* α (*get-m* (*R a wa*) *hl*)) (*pmult* β (*get-m* (*R a wb*) *hl*)))
proof (*cases pgte* (*get-m w hl*) (*comp-one* (*get-m a hl*)))
case *True*
then have *get-m* (*R a w*) *hl* = (*comp-one* (*get-m a hl*))
using *r3 comp-min-mask-def pmin-is* **by** *presburger*
moreover have *pgte* (*comp-one* (*get-m a hl*)) (*get-m* (*R a wa*) *hl*)
by (*metis r1 comp-min-mask-def pmin-comm pmin-greater*)
then have *pgte* (*pmult* α (*comp-one* (*get-m a hl*))) (*pmult* α (*get-m* (*R a wa*) *hl*))
by (*metis pmult-comm pmult-order*)
moreover have *pgte* (*comp-one* (*get-m a hl*)) (*get-m* (*R a wb*) *hl*)
by (*metis r2 comp-min-mask-def pmin-comm pmin-greater*)
then have *pgte* (*pmult* β (*comp-one* (*get-m a hl*))) (*pmult* β (*get-m* (*R a wb*) *hl*))
by (*metis pmult-comm pmult-order*)
ultimately show *?thesis*
using ‹*pgte* (*comp-one* (*get-m a hl*)) (*get-m* (*R a wa*) *hl*)› ‹*pgte* (*comp-one* (*get-m a hl*)) (*get-m* (*R a wb*) *hl*)› *asm0 padd-one-ineq-sum* **by** *presburger*
next
case *False*
then have *get-m* (*R a w*) *hl* = *get-m w hl*
by (*metis r3 comp-min-mask-def not-pgte-charact pgt-implies-pgte pmin-comm pmin-is*)
moreover have *pgte* (*get-m w hl*) (*padd* (*pmult* α (*get-m wa hl*)) (*pmult* β (*get-m wb hl*)))
proof −
have *pgte* (*get-m w hl*) (*padd* (*get-m xa hl*) (*get-m xb hl*))
using ‹*Some w* = *xa* ⊕ *xb*› *not-pgte-charact pgt-implies-pgte plus-charact*(*1*) **by** *auto*
moreover have *pgte* (*get-m xa hl*) (*pmult* α (*get-m wa hl*))
by (*metis* ‹*pgte pwrite* α ∧ *pgte pwrite* β› ‹*xa* ⪰ *multiply* α *wa*› *get-m-smaller larger-implies-greater-mask-hl*)
moreover have *pgte* (*get-m xb hl*) (*pmult* β (*get-m wb hl*))

by (*metis* ‹*pgte pwrite* $\alpha \wedge$ *pgte pwrite* β› ‹*xb* $\succeq$ *multiply* β *wb*› *get-m-smaller larger-implies-greater-mask-hl*)
ultimately show *?thesis*
by (*simp add*: *padd.rep-eq pgte.rep-eq*)
qed
moreover have *pgte* (*pmult* α (*get-m wa hl*)) (*pmult* α (*get-m* (*R a wa*) *hl*))
by (*metis R-smaller larger-implies-greater-mask-hl pmult-comm pmult-order*)
moreover have *pgte* (*pmult* β (*get-m wb hl*)) (*pmult* β (*get-m* (*R a wb*) *hl*))
by (*metis R-smaller larger-implies-greater-mask-hl pmult-comm pmult-order*)
ultimately show *?thesis*
using *padd.rep-eq pgte.rep-eq* **by** *force*
qed
moreover have *get-m x hl* = *padd* (*get-m* (*R a w*) *hl*) (*get-m a hl*)
using *calculation*(*1*) **by** *auto*
moreover have *get-m y hl* = *padd* (*pmult* α (*get-m ya hl*)) (*pmult* β (*get-m yb hl*))
using ‹*Some y* = *multiply* α *ya* $\oplus$ *multiply* β *yb*› ‹*pgte pwrite* $\alpha \wedge$ *pgte pwrite* β› *get-m-smaller plus-charact*(*1*) **by** *auto*
moreover have *padd* (*pmult* α (*get-m ya hl*)) (*pmult* β (*get-m yb hl*)) = *padd* (*pmult* α (*padd* (*get-m* (*R a wa*) *hl*) (*get-m a hl*))) (*pmult* β (*padd* (*get-m* (*R a wb*) *hl*) (*get-m a hl*)))
using *calculation*(*2*) *calculation*(*5*) **by** *presburger*
moreover have ... = *padd* (*pmult* (*padd* α β) (*get-m a hl*)) (*padd* (*pmult* α (*get-m* (*R a wa*) *hl*)) (*pmult* β (*get-m* (*R a wb*) *hl*)))
by (*metis equ pmult-comm pmult-distr*)
ultimately show *pgte* (*get-m x hl*) (*get-m y hl*)
using *asm0 p-greater-exists padd-asso padd-comm pmult-special*(*1*) **by** *force*
qed
qed
ultimately have $y \in$ *multiply-sem-assertion* α *B* $\otimes$ *multiply-sem-assertion* β *B*
using *PartialSA.add-set-elem* **by** *blast*
then have $y \in$ *multiply-sem-assertion pwrite B*
by (*metis asm0 assms*(*1*) *combinable-def not-pgte-charact pgt-implies-pgte subsetD*)
then obtain *b* **where** $y \succeq$ *multiply pwrite b* $b \in B$
using *in-multiply-sem* **by** *blast*
then have *multiply pwrite b* = *b*
by (*metis Rep-state-inverse get-h-m mult-write-mask multiply.simps*)
then have $y \in B$
by (*metis* ‹$b \in B$› ‹$y \succeq$ *multiply pwrite b*› *assms*(*2*) *intuitionistic-def*)
show $x \in B$
using ‹$x \succeq y$› ‹$y \in B$› *assms*(*2*) *intuitionistic-def* **by** *blast*
qed

qed
qed

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 −
have *R a unit* = *unit*
by (*simp add*: *PartialSA.succ-antisym PartialSA.unit-smaller R-smaller*)
moreover have $\bigwedge\varphi\ \varphi'.\ \varphi' \succeq \varphi \Longrightarrow R\ a\ \varphi' \succeq R\ a\ \varphi$
proof −
fix $\varphi\ \varphi'$
assume $\varphi' \succeq \varphi$
show $R\ a\ \varphi' \succeq R\ a\ \varphi$
proof (*cases scalable* φ' *a*)
case *True*
then show *?thesis*
by (*metis PartialSA.succ-trans R-def R-smaller* ‹$\varphi' \succeq \varphi$›)
next
case *False*
then obtain *p* **where** *ppos p pgte pwrite p multiply p* φ' |#| *a*
by (*metis PartialSA.commutative PartialSA.defined-def non-scalable-instantiate*)
then have *multiply p* φ |#| *a*
using *PartialSA.smaller-compatible* ‹$\varphi' \succeq \varphi$› *multiply-order* **by** *blast*
then have ¬ *scalable* φ *a*
using *PartialSA.commutative PartialSA.defined-def* ‹*pgte pwrite p*› ‹*ppos p*› *scalable-def scaled-def* **by** *auto*

moreover have *greater-mask* (*comp-min-mask* (*get-m a*) (*get-m* φ')) (*comp-min-mask* (*get-m a*) (*get-m* φ))
proof (*rule greater-maskI*)
fix *hl* **show** *pgte* (*comp-min-mask* (*get-m a*) (*get-m* φ') *hl*) (*comp-min-mask* (*get-m a*) (*get-m* φ) *hl*)
proof (*cases pgte* (*get-m* φ' *hl*) (*comp-one* (*get-m a hl*)))
case *True*
then show *?thesis*
by (*metis comp-min-mask-def pmin-comm pmin-greater pmin-is*)
next
case *False*
then show *?thesis*
by (*metis PartialSA.succ-trans R-smaller* ‹$\varphi' \succeq \varphi$› *calculation comp-min-mask-def larger-implies-greater-mask-hl non-scalable-R-charact not-pgte-charact pgt-implies-pgte pmin-comm pmin-is*)
qed
qed
ultimately show *?thesis*

using *False* ‹$\varphi' \succeq \varphi$› *greaterI larger-implies-larger-heap non-scalable-R-charact*
by *presburger*
qed
qed
ultimately show *?thesis*
by (*simp add*: *PartialSA.mono-transformer-def*)
qed

theorem *properties-of-combinable-wands*:
assumes *intuitionistic B*
shows *combinable B* $\Longrightarrow$ *combinable* (*cwand A B*)
and *cwand A B* $\subseteq$ *wand A B*
and *binary A* $\Longrightarrow$ *cwand A B* = *wand A B*
by (*simp-all add*: *assms combinable-cwand cwand-stronger binary-same dual-order.eq-iff*)

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.