Formalization of a Framework for the Sound Automation of Magic Wands

Thibault Dardinier

May 14, 2024

Abstract

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

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

Contents

1 Separation Algebra **2**
- 1.1 Definitions . . . 2
- 1.2 First lemmata . . . 3
  - 1.2.1 splus . . . 4
  - 1.2.2 Pure . . . 5
- 1.3 Succ is an order . . . 6
- 1.4 Core (pure) and stabilize (stable) . . . 7
- 1.5 Subtraction . . . 8
- 1.6 Lifting the algebra to sets of states . . . 13
- 1.7 Addition of more than two states . . . 18

2 Package Logic **24**
- 2.1 Definitions . . . 24
- 2.2 Lemmas . . . 26
- 2.3 Lemmas for completeness . . . 39
- 2.4 Soundness . . . 52
- 2.5 Completeness . . . 55

1 Separation Algebra

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

theory *SepAlgebra*
imports *Main*
begin

type-synonym *'a property* = *'a* ⇒ *bool*

locale *sep-algebra* =

fixes *plus* :: *'a* ⇒ *'a* ⇒ *'a option* (**infixl** ⊕ *63*)

fixes *core* :: *'a* ⇒ *'a* (|-|)

assumes *commutative*: *a* ⊕ *b* = *b* ⊕ *a*
and *asso1*: *a* ⊕ *b* = *Some ab* ∧ *b* ⊕ *c* = *Some bc* ⟹ *ab* ⊕ *c* = *a* ⊕ *bc*
and *asso2*: *a* ⊕ *b* = *Some ab* ∧ *b* ⊕ *c* = *None* ⟹ *ab* ⊕ *c* = *None*

and *core-is-smaller*: *Some x* = *x* ⊕ |*x*|
and *core-is-pure*: *Some* |*x*| = |*x*| ⊕ |*x*|
and *core-max*: *Some x* = *x* ⊕ *c* ⟹ (∃ *r*. *Some* |*x*| = *c* ⊕ *r*)
and *core-sum*: *Some c* = *a* ⊕ *b* ⟹ *Some* |*c*| = |*a*| ⊕ |*b*|

and *positivity*: *a* ⊕ *b* = *Some c* ⟹ *Some c* = *c* ⊕ *c* ⟹ *Some a* = *a* ⊕ *a*
and *cancellative*: *Some a* = *b* ⊕ *x* ⟹ *Some a* = *b* ⊕ *y* ⟹ |*x*| = |*y*| ⟹ *x* = *y*

begin

lemma *asso3*:
assumes *a* ⊕ *b* = *None*
and *b* ⊕ *c* = *Some bc*
shows *a* ⊕ *bc* = *None*
by (*metis assms*(*1*) *assms*(*2*) *sep-algebra.asso2 sep-algebra.commutative sep-algebra-axioms*)

1.1 Definitions

definition *defined* :: *'a* ⇒ *'a* ⇒ *bool* (**infixl** ## *62*) **where**
a ## *b* ⟷ *a* ⊕ *b* ≠ *None*

definition *greater* :: *'a* ⇒ *'a* ⇒ *bool* (**infixl** ≽ *50*) **where**
a ≽ *b* ⟷ (∃ *c*. *Some a* = *b* ⊕ *c*)

definition *pure* :: *'a* ⇒ *bool* **where**
pure a ⟷ *Some a* = *a* ⊕ *a*

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

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

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

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

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

definition *setify* :: *'a property* ⇒ (*'a set* ⇒ *bool*) **where**
$setify\ P\ A \longleftrightarrow (\forall\, x \in A.\ P\ x)$

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

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

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

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

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

1.2 First lemmata

lemma *greater-equiv*:
$a \succeq b \longleftrightarrow (\exists\, c.\ Some\ a = c \oplus b)$
using *commutative greater-def* **by** *auto*

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

by (*metis* (*full-types*) *assms*(*1*) *assms*(*2*) *asso3 commutative defined-def greater-def*)

lemma *bigger-sum-smaller*:
assumes *Some* $c = a \oplus b$
and $a \succeq a'$
shows $\exists\, b'.\ b' \succeq b \wedge$ *Some* $c = a' \oplus b'$
proof −
obtain r **where** *Some* $a = a' \oplus r$
using *assms*(*2*) *greater-def* **by** *auto*
then obtain br **where** *Some* $br = r \oplus b$
by (*metis assms*(*1*) *asso2 domD domIff option.discI*)
then have *Some* $c = a' \oplus br$
by (*metis* ‹*Some* $a = a' \oplus r$› *assms*(*1*) *asso1*)
then show *?thesis*
using ‹*Some* $br = r \oplus b$› *commutative greater-def* **by** *force*
qed

1.2.1 splus

lemma *splus-develop*:
assumes *Some* $a = b \oplus c$
shows $a \oplus d =$ *splus* (*splus* (*Some* b) (*Some* c)) (*Some* d)
by (*metis assms splus.simps*(*3*))

lemma *splus-comm*:
splus $a\ b =$ *splus* $b\ a$
apply (*cases* a)
apply (*cases* b)
apply *simp-all*
apply (*cases* b)
by (*simp-all add*: *commutative*)

lemma *splus-asso*:
splus (*splus* $a\ b$) $c =$ *splus* a (*splus* $b\ c$)
proof (*cases* a)
case *None*
then show *?thesis*
by *simp*
next
case (*Some* a')
then have $a =$ *Some* a' **by** *simp*
then show *?thesis*
proof (*cases* b)
case *None*
then show *?thesis* **by** (*simp add*: *Some*)
next
case (*Some* b')
then have $b =$ *Some* b' **by** *simp*
then show *?thesis*

```
    proof (cases c)
      case None
      then show ?thesis by (simp add: splus-comm)
    next
      case (Some c')
      then have c = Some c' by simp
      then show ?thesis
      proof (cases a' ⊕ b')
        case None
        then have a' ⊕ b' = None by simp
        then show ?thesis
        proof (cases b' ⊕ c')
          case None
          then show ?thesis
            by (simp add: Some ‹a = Some a'› ‹a' ⊕ b' = None› ‹b = Some b'›)
        next
          case (Some bc)
          then show ?thesis
            by (metis (full-types) None ‹a = Some a'› ‹b = Some b'› ‹c = Some c'›
sep-algebra.asso2 sep-algebra-axioms splus.simps(2) splus.simps(3) splus-comm)
        qed
      next
        case (Some ab)
        then have Some ab = a' ⊕ b' by simp
        then show ?thesis
        proof (cases b' ⊕ c')
          case None
          then show ?thesis
              by (metis Some ‹a = Some a'› ‹b = Some b'› ‹c = Some c'› asso2
splus.simps(2) splus.simps(3))
        next
          case (Some bc)
          then show ?thesis
            by (metis ‹Some ab = a' ⊕ b'› ‹a = Some a'› ‹b = Some b'› ‹c = Some
c'› sep-algebra.asso1 sep-algebra-axioms splus.simps(3))
        qed
      qed
    qed
  qed
qed
```

1.2.2 Pure

```
lemma pure-stable:
  assumes pure a
      and pure b
      and Some c = a ⊕ b
    shows pure c
  by (metis assms asso1 commutative pure-def)
```

lemma *pure-smaller*:
assumes *pure a*
and $a \succeq b$
shows *pure b*
by (*metis assms greater-def positivity pure-def*)

1.3 Succ is an order

lemma *succ-antisym*:
assumes $a \succeq b$
and $b \succeq a$
shows $a = b$
proof −
obtain *ra* **where** *Some a* = *b* ⊕ *ra*
using *assms*(*1*) *greater-def* **by** *auto*
obtain *rb* **where** *Some b* = *a* ⊕ *rb*
using *assms*(*2*) *greater-def* **by** *auto*
then have *Some a* = *splus* (*Some a*) (*splus* (*Some ra*) (*Some rb*))
proof −
have *Some b* = *splus* (*Some a*) (*Some rb*)
by (*simp add*: ‹*Some b* = *a* ⊕ *rb*›)
then show *?thesis*
by (*metis* (*full-types*) ‹*Some a* = *b* ⊕ *ra*› *sep-algebra.splus.simps*(*3*) *sep-algebra-axioms splus-asso splus-comm*)
qed
moreover have *Some b* = *splus* (*Some b*) (*splus* (*Some ra*) (*Some rb*))
by (*metis* ‹*Some a* = *b* ⊕ *ra*› ‹*Some b* = *a* ⊕ *rb*› *sep-algebra.splus.simps*(*3*) *sep-algebra-axioms splus-asso*)
moreover have *pure ra* ∧ *pure rb*
proof −
obtain *rab* **where** *Some rab* = *ra* ⊕ *rb*
by (*metis calculation*(*2*) *splus.elims splus.simps*(*3*))
then have |*a*| ≽ *rab*
by (*metis calculation*(*1*) *core-max greater-def splus.simps*(*3*))
then have *pure rab*
using *core-is-pure pure-def pure-smaller* **by** *blast*
moreover have *rab* ≽ *ra* ∧ *rab* ≽ *rb*
using ‹*Some rab* = *ra* ⊕ *rb*› *greater-def greater-equiv* **by** *blast*
ultimately have *pure ra* **using** *pure-smaller*
by *blast*
moreover have *pure rb*
using ‹*pure rab*› ‹*rab* ≽ *ra* ∧ *rab* ≽ *rb*› *pure-smaller* **by** *blast*
ultimately show *?thesis*
by *blast*
qed
ultimately show *?thesis*
by (*metis* ‹*Some b* = *a* ⊕ *rb*› *option.inject pure-def sep-algebra.splus.simps*(*3*)

sep-algebra-axioms splus-asso)
qed

lemma *succ-trans*:
assumes $a \succeq b$
and $b \succeq c$
shows $a \succeq c$
using *assms*(*1*) *assms*(*2*) *bigger-sum-smaller greater-def* **by** *blast*

lemma *succ-refl*:
$a \succeq a$
using *core-is-smaller greater-def* **by** *blast*

1.4 Core (pure) and stabilize (stable)

lemma *max-projection-propI*:
assumes $\bigwedge x.\ x \succeq f\ x$
and $\bigwedge x.\ P\ (f\ x)$
and $\bigwedge x\ p.\ P\ p \wedge x \succeq p \Longrightarrow f\ x \succeq p$
shows *max-projection-prop P f*
by (*simp add*: *assms*(*1*) *assms*(*2*) *assms*(*3*) *max-projection-prop-def*)

lemma *max-projection-propE*:
assumes *max-projection-prop P f*
shows $\bigwedge x.\ x \succeq f\ x$
and $\bigwedge x.\ P\ (f\ x)$
and $\bigwedge x\ p.\ P\ p \wedge x \succeq p \Longrightarrow f\ x \succeq p$
using *assms max-projection-prop-def* **by** *auto*

lemma *max-projection-prop-pure-core*:
max-projection-prop pure core
proof (*rule max-projection-propI*)
fix x
show $x \succeq |x|$
using *core-is-smaller greater-equiv* **by** *blast*
show *pure* $|x|$
by (*simp add*: *core-is-pure pure-def*)
show $\bigwedge p.$ *pure* $p \wedge x \succeq p \Longrightarrow |x| \succeq p$
proof −
fix p **assume** *pure* $p \wedge x \succeq p$
then obtain r **where** *Some* $x = p \oplus r$
using *greater-def* **by** *blast*
then show $|x| \succeq p$
by (*metis* ‹*pure* $p \wedge x \succeq p$› *asso1 commutative core-max greater-equiv pure-def*)
qed
qed

lemma *mpp-smaller*:
assumes *max-projection-prop P f*

shows $x \succeq f\ x$
using *assms max-projection-propE*(*1*) **by** *auto*

lemma *mpp-compatible*:
assumes *max-projection-prop P f*
and $a \#\# b$
shows $f\ a \#\# f\ b$
by (*metis* (*mono-tags*, *opaque-lifting*) *assms*(*1*) *assms*(*2*) *commutative defined-def max-projection-prop-def smaller-compatible*)

lemma *mpp-prop*:
assumes *max-projection-prop P f*
shows $P\ (f\ x)$
by (*simp add*: *assms max-projection-propE*(*2*))

lemma *mppI*:
assumes *max-projection-prop P f*
and $a \succeq x$
and $P\ x$
and $x \succeq f\ a$
shows $x = f\ a$
proof −
have $f\ a \succeq x$
using *assms max-projection-propE*(*3*) **by** *auto*
then show *?thesis*
by (*simp add*: *assms*(*4*) *succ-antisym*)
qed

lemma *mpp-invo*:
assumes *max-projection-prop P f*
shows $f\ (f\ x) = f\ x$
using *assms max-projection-prop-def succ-antisym* **by** *auto*

lemma *mpp-mono*:
assumes *max-projection-prop P f*
and $a \succeq b$
shows $f\ a \succeq f\ b$
by (*metis assms max-projection-prop-def succ-trans*)

1.5 Subtraction

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

by (*metis assms asso1 bigger-sum-smaller greater-def*)

lemma *smaller-than-core*:
assumes $y \succeq x$
and *Some* $z = x \oplus |y|$
shows $|z| = |y|$
proof −
have *Some* $|z| = |x| \oplus |y|$
using *assms*(*2*) *core-sum max-projection-prop-pure-core mpp-invo* **by** *fastforce*
then have *Some* $|z| = |x| \oplus |y|$
by *simp*
moreover have $|z| \succeq |y|$
using *calculation greater-equiv* **by** *blast*
ultimately show *?thesis*
by (*meson addition-bigger assms*(*1*) *assms*(*2*) *core-is-smaller core-sum greater-def succ-antisym*)
qed

lemma *extract-core*:
assumes *Some* $b = a \oplus x \wedge x \succeq |b|$
shows $|x| = |b|$
proof −
obtain r **where** *Some* $x = r \oplus |b|$
using *assms greater-equiv* **by** *auto*
show *?thesis*
proof (*rule smaller-than-core*)
show *Some* $x = r \oplus |b|$
using ‹*Some* $x = r \oplus |b|$› **by** *auto*
show $b \succeq r$
by (*metis* ‹*Some* $x = r \oplus |b|$› *assms commutative greater-def succ-trans*)
qed
qed

lemma *minus-unique*:
assumes *Some* $b = a \oplus x \wedge x \succeq |b|$
and *Some* $b = a \oplus y \wedge y \succeq |b|$
shows $x = y$
proof −
have $|x| = |b|$
using *assms*(*1*) *extract-core* **by** *blast*
moreover have $|y| = |b|$
using *assms*(*2*) *extract-core* **by** *blast*
ultimately show *?thesis*
using *assms*(*1*) *assms*(*2*) *cancellative* **by** *auto*
qed

lemma *minus-exists*:

assumes $b \succeq a$
shows $\exists x.\ Some\ b = a \oplus x \wedge x \succeq |b|$
using *assms bigger-sum-smaller core-is-smaller* **by** *blast*

lemma *minus-equiv-def*:
assumes $b \succeq a$
shows $Some\ b = a \oplus (b \ominus a) \wedge (b \ominus a) \succeq |b|$
proof −
let $?x = THE\text{-}default\ b\ (\lambda x.\ Some\ b = a \oplus x \wedge x \succeq |b|\ )$
have $(\lambda x.\ Some\ b = a \oplus x \wedge x \succeq |b|\ )\ ?x$
proof (*rule THE-defaultI'*)
show $\exists! x.\ Some\ b = a \oplus x \wedge x \succeq |b|$
using *assms local.minus-unique minus-exists* **by** *blast*
qed
then show *?thesis* **by** (*metis minus-def*)
qed

lemma *minus-default*:
assumes $\neg\ b \succeq a$
shows $b \ominus a = b$
using *THE-default-none assms greater-def minus-def* **by** *fastforce*

lemma *minusI*:
assumes $Some\ b = a \oplus x$
and $x \succeq |b|$
shows $x = b \ominus a$
using *assms*(*1*) *assms*(*2*) *greater-def local.minus-unique minus-equiv-def* **by** *blast*

lemma *minus-core*:
$|a \ominus b| = |a|$
proof (*cases* $a \succeq b$)
case *True*
then have $Some\ a = b \oplus (a \ominus b) \wedge a \ominus b \succeq |a|$
using *minus-equiv-def* **by** *auto*
then show *?thesis*
using *extract-core* **by** *blast*
next
case *False*
then show *?thesis* **by** (*simp add*: *minus-default*)
qed

lemma *minus-core-weaker*:
$|a \ominus b| = |a| \ominus |b|$
proof (*cases* $a \succeq b$)
case *True*
then show *?thesis*
by (*metis greater-equiv max-projection-prop-pure-core minus-core minus-default minus-equiv-def mpp-invo succ-antisym*)

next
case *False*
then show *?thesis*
by (*metis greater-equiv max-projection-prop-pure-core minus-default minus-equiv-def mpp-invo succ-antisym*)
qed

lemma *minus-equiv-def-any-elem*:
assumes *Some* $x = a \oplus b$
shows *Some* $(x \ominus a) = b \oplus |x|$
proof −
obtain r **where** *Some* $r = b \oplus |x|$
by (*metis assms core-is-smaller domD domIff option.simps*(3) *sep-algebra.asso2 sep-algebra-axioms*)
have $r = x \ominus a$
proof (*rule minusI*)
show *Some* $x = a \oplus r$
by (*metis* ‹*Some* $r = b \oplus |x|$› *assms asso1 core-is-smaller*)
moreover show $r \succeq |x|$
using ‹*Some* $r = b \oplus |x|$› *greater-equiv* **by** *blast*
qed
then show *?thesis*
using ‹*Some* $r = b \oplus |x|$› **by** *blast*
qed

lemma *minus-bigger*:
assumes *Some* $x = a \oplus b$
shows $x \ominus a \succeq b$
using *assms greater-def minus-equiv-def-any-elem* **by** *blast*

lemma *minus-smaller*:
assumes $x \succeq a$
shows $x \succeq x \ominus a$
using *assms greater-equiv minus-equiv-def* **by** *blast*

lemma *minus-sum*:
assumes *Some* $a = b \oplus c$
and $x \succeq a$
shows $x \ominus a = (x \ominus b) \ominus c$
proof (*rule minusI*)
obtain r **where** *Some* $r = c \oplus (x \ominus a)$
by (*metis assms*(1) *assms*(2) *asso2 minus-equiv-def option.exhaust-sel*)
have $r = (x \ominus b)$
proof (*rule minusI*)
show *Some* $x = b \oplus r$
by (*metis* ‹*Some* $r = c \oplus (x \ominus a)$› *assms*(1) *assms*(2) *asso1 minus-equiv-def*)
moreover show $r \succeq |x|$
by (*meson* ‹*Some* $r = c \oplus (x \ominus a)$› *assms*(2) *greater-equiv sep-algebra.minus-equiv-def sep-algebra-axioms succ-trans*)

qed
then show *Some* $(x \ominus b) = c \oplus (x \ominus a)$
using ‹*Some* $r = c \oplus (x \ominus a)$› **by** *blast*
moreover show $x \ominus a \succeq |x \ominus b|$
by (*simp add*: *assms*(*2*) *minus-core minus-equiv-def*)
qed

lemma *smaller-compatible-core*:
assumes $y \succeq x$
shows $x \#\# |y|$
by (*metis assms asso2 core-is-smaller defined-def greater-equiv option.discI*)

lemma *smaller-pure-sum-smaller*:
assumes $y \succeq a$
and $y \succeq b$
and *Some* $x = a \oplus b$
and *pure* b
shows $y \succeq x$
proof −
obtain r **where** *Some* $y = r \oplus b$ $r \succeq a$
by (*metis assms*(*1*) *assms*(*2*) *assms*(*4*) *asso1 greater-equiv pure-def*)
then show *?thesis*
using *addition-bigger assms*(*3*) **by** *blast*
qed

lemma *greater-minus-trans*:
assumes $y \succeq x$
and $x \succeq a$
shows $y \ominus a \succeq x \ominus a$
proof −
obtain r **where** *Some* $y = x \oplus r$
using *assms*(*1*) *greater-def* **by** *blast*
then obtain *ra* **where** *Some* $x = a \oplus ra$
using *assms*(*2*) *greater-def* **by** *blast*
then have *Some* $(x \ominus a) = ra \oplus |x|$
by (*simp add*: *minus-equiv-def-any-elem*)
then obtain *yy* **where** *Some* $yy = (x \ominus a) \oplus r$
by (*metis* (*full-types*) ‹*Some* $y = x \oplus r$› *assms*(*2*) *asso3 commutative minus-equiv-def not-Some-eq*)
then obtain *Some* $x = a \oplus (x \ominus a)$ $x \ominus a \succeq |x|$
by (*simp-all add*: *assms*(*2*) *sep-algebra.minus-equiv-def sep-algebra-axioms*)
then obtain y' **where** *Some* $y' = a \oplus yy$
using ‹*Some* $y = x \oplus r$› ‹*Some* $yy = x \ominus a \oplus r$› *asso1*
by *metis*
moreover have $y \succeq y'$
by (*metis* ‹*Some* $x = a \oplus (x \ominus a)$› ‹*Some* $y = x \oplus r$› ‹*Some* $yy = x \ominus a \oplus r$› *asso1 calculation option.inject succ-refl*)
moreover obtain x' **where** *Some* $x' = (x \ominus a) \oplus a$
using *assms*(*2*) *commutative minus-equiv-def* **by** *fastforce*

then have $y \succeq x'$
by (*metis assms*(*1*) *assms*(*2*) *commutative minus-equiv-def option.sel*)
moreover have $x' \succeq x \ominus a$
using ‹*Some* $x' = x \ominus a \oplus a$› *greater-def* **by** *auto*
ultimately show *?thesis*
using ‹*Some* $x' = x \ominus a \oplus a$› ‹*Some* $y = x \oplus r$› *assms*(*2*) *asso1 commutative greater-equiv minus-bigger minus-equiv-def option.sel sep-algebra.succ-trans sep-algebra-axioms*
proof −
have *f1*: *Some* $y' = a \oplus yy$
by (*simp add*: ‹*Some* $y' = a \oplus yy$› *commutative*)
then have $y = y'$
by (*metis* ‹*Some* $y = x \oplus r$› ‹*Some* $yy = x \ominus a \oplus r$› ‹$x \succeq a$› *asso1 minus-equiv-def option.sel*)
then show *?thesis*
using *f1* **by** (*metis* (*no-types*) ‹*Some* $yy = x \ominus a \oplus r$› *commutative greater-equiv minus-bigger sep-algebra.succ-trans sep-algebra-axioms*)
qed
qed

lemma *minus-and-plus*:
assumes *Some* $\omega' = \omega \oplus r$
and $\omega \succeq a$
shows *Some* $(\omega' \ominus a) = (\omega \ominus a) \oplus r$
proof −
have $\omega \succeq \omega \ominus a$
by (*simp add*: *assms*(*2*) *minus-smaller*)
then have $(\omega \ominus a)$ *##* r
by (*metis* (*full-types*) *assms*(*1*) *defined-def option.discI sep-algebra.smaller-compatible sep-algebra-axioms*)
then obtain x **where** *Some* $x = (\omega \ominus a) \oplus r$
using *defined-def* **by** *auto*
then have *Some* $\omega' = a \oplus x \wedge x \succeq |\omega'|$
by (*metis* (*no-types*, *lifting*) *assms asso1 core-sum max-projection-prop-pure-core minus-core minus-equiv-def mpp-smaller option.inject*)
then have $x = \omega' \ominus a$
by (*simp add*: *minusI*)
then show *?thesis*
using ‹*Some* $x = \omega \ominus a \oplus r$› **by** *blast*
qed

1.6 Lifting the algebra to sets of states

lemma *add-set-commm*:
$A \otimes B = B \otimes A$
proof

show $A \otimes B \subseteq B \otimes A$
using *add-set-def sep-algebra.commutative sep-algebra-axioms* **by** *fastforce*
show $B \otimes A \subseteq A \otimes B$
using *add-set-def commutative* **by** *fastforce*
qed

lemma *x-elem-set-product*:
$x \in A \otimes B \longleftrightarrow (\exists\, a\ b.\ a \in A \wedge b \in B \wedge \mathit{Some}\ x = a \oplus b)$
using *sep-algebra.add-set-def sep-algebra-axioms* **by** *fastforce*

lemma *x-elem-set-product-splus*:
$x \in A \otimes B \longleftrightarrow (\exists\, a\ b.\ a \in A \wedge b \in B \wedge \mathit{Some}\ x = \mathit{splus}\ (\mathit{Some}\ a)\ (\mathit{Some}\ b))$
using *sep-algebra.add-set-def sep-algebra-axioms* **by** *fastforce*

lemma *add-set-asso*:
$(A \otimes B) \otimes C = A \otimes (B \otimes C)$ (**is** *?A = ?B*)
proof −
have *?A* $\subseteq$ *?B*
proof (*rule subsetI*)
fix x **assume** $x \in$ *?A*
then obtain *ab c* **where** $\mathit{Some}\ x = ab \oplus c\ ab \in A \otimes B\ c \in C$
using *x-elem-set-product* **by** *auto*
then obtain *a b* **where** $\mathit{Some}\ ab = a \oplus b\ a \in A\ b \in B$
using *x-elem-set-product* **by** *auto*
then obtain *bc* **where** $\mathit{Some}\ bc = b \oplus c$
by (*metis* ‹*Some x = ab ⊕ c*› *asso2 option.exhaust*)
then show $x \in$ *?B*
by (*metis* ‹*Some ab = a ⊕ b*› ‹*Some x = ab ⊕ c*› ‹$a \in A$› ‹$b \in B$› ‹$c \in C$›
asso1 x-elem-set-product)
qed
moreover have *?B* $\subseteq$ *?A*
proof (*rule subsetI*)
fix x **assume** $x \in$ *?B*
then obtain *a bc* **where** $\mathit{Some}\ x = a \oplus bc\ a \in A\ bc \in B \otimes C$
using *x-elem-set-product* **by** *auto*
then obtain *b c* **where** $\mathit{Some}\ bc = b \oplus c\ c \in C\ b \in B$
using *x-elem-set-product* **by** *auto*
then obtain *ab* **where** $\mathit{Some}\ ab = a \oplus b$
by (*metis* ‹*Some x = a ⊕ bc*› *asso3 option.collapse*)
then show $x \in$ *?A*
by (*metis* ‹*Some bc = b ⊕ c*› ‹*Some x = a ⊕ bc*› ‹$a \in A$› ‹$b \in B$› ‹$c \in C$›
asso1 x-elem-set-product)
qed
ultimately show *?thesis* **by** *blast*
qed

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

using *assms up-closed-def* **by** *blast*

lemma *up-closed-plus-UNIV*:
up-closed $(A \otimes UNIV)$
proof (*rule up-closedI*)
fix φ φ'
assume *asm*: $\varphi' \succeq \varphi \wedge \varphi \in A \otimes UNIV$
then obtain r a b **where** *Some* $\varphi' = \varphi \oplus r$ *Some* $\varphi = a \oplus b$ $a \in A$
using *greater-def x-elem-set-product* **by** *auto*
then obtain *br* **where** *Some* $br = b \oplus r$
by (*metis asso2 option.exhaust-sel*)
then have *Some* $\varphi' = a \oplus br$
by (*metis* ‹*Some* $\varphi = a \oplus b$› ‹*Some* $\varphi' = \varphi \oplus r$› *splus.simps*(*3*) *splus-asso*)
then show $\varphi' \in A \otimes UNIV$
using ‹$a \in A$› *x-elem-set-product* **by** *auto*
qed

lemma *succ-set-trans*:
assumes $A \gg B$
and $B \gg C$
shows $A \gg C$
by (*meson assms*(*1*) *assms*(*2*) *greater-set-def succ-trans*)

lemma *greater-setI*:
assumes $\bigwedge a.\ a \in A \Longrightarrow (\exists\, b \in B.\ a \succeq b)$
shows $A \gg B$
by (*simp add*: *assms greater-set-def*)

lemma *bigger-set*:
assumes $A' \gg A$
shows $A' \otimes B \gg A \otimes B$
proof (*rule greater-setI*)
fix x **assume** $x \in A' \otimes B$
then obtain a' b **where** *Some* $x = a' \oplus b$ $a' \in A'$ $b \in B$
using *x-elem-set-product* **by** *auto*
then obtain a **where** $a' \succeq a$ $a \in A$
using *assms greater-set-def* **by** *blast*
then obtain *ab* **where** *Some* $ab = a \oplus b$
by (*metis* ‹*Some* $x = a' \oplus b$› *asso2 domD domIff greater-equiv*)
then show $\exists\, ab \in A \otimes B.\ x \succeq ab$
using ‹*Some* $x = a' \oplus b$› ‹$a \in A$› ‹$a' \succeq a$› ‹$b \in B$› *addition-bigger x-elem-set-product*
by *blast*
qed

lemma *bigger-singleton*:
assumes $\varphi' \succeq \varphi$
shows $\{\varphi'\} \gg \{\varphi\}$
by (*simp add*: *assms greater-set-def*)

lemma *add-set-elem*:
  $\varphi \in A \otimes B \longleftrightarrow (\exists\, a\ b.\ \mathit{Some}\ \varphi = a \oplus b \wedge a \in A \wedge b \in B)$
  using *add-set-def* **by** *auto*

lemma *up-closed-sum*:
  assumes *up-closed A*
    shows *up-closed* $(A \otimes B)$
proof (*rule up-closedI*)
  fix $\varphi'\ \varphi$ **assume** *asm*: $\varphi' \succeq \varphi \wedge \varphi \in A \otimes B$
  then obtain $a\ b$ **where** $a \in A\ b \in B\ \mathit{Some}\ \varphi = a \oplus b$
    using *add-set-elem* **by** *auto*
  moreover obtain r **where** $\mathit{Some}\ \varphi' = \varphi \oplus r$
    using *asm greater-def* **by** *blast*
  then obtain ar **where** $\mathit{Some}\ ar = a \oplus r$
   by (*metis asso2 calculation*(3) *commutative option.exhaust-sel option.simps*(3))
  then have $ar \in A$
   by (*meson assms calculation*(1) *greater-def sep-algebra.up-closed-def sep-algebra-axioms*)
  then show $\varphi' \in A \otimes B$
    by (*metis* ‹*Some* $\varphi' = \varphi \oplus r$› ‹*Some* $ar = a \oplus r$› *add-set-elem asso1 calculation*(2) *calculation*(3) *commutative*)
qed

lemma *up-closed-bigger-subset*:
  assumes *up-closed B*
      and $A \gg B$
    shows $A \subseteq B$
 by (*meson assms*(1) *assms*(2) *greater-set-def sep-algebra.up-closed-def sep-algebra-axioms subsetI*)

lemma *up-close-equiv*:
  assumes *up-closed A*
      and *up-closed B*
    shows $A \sim B \longleftrightarrow A = B$
proof −
  have $A \sim B \longleftrightarrow A \gg B \wedge B \gg A$
    using *local.equiv-def* **by** *auto*
  also have ... $\longleftrightarrow A \subseteq B \wedge B \subseteq A$
   by (*metis assms*(1) *assms*(2) *greater-set-def set-eq-subset succ-refl up-closed-bigger-subset*)
  ultimately show *?thesis*
    by *blast*
qed

lemma *equiv-stable-sum*:
  assumes $A \sim B$
  shows $A \otimes C \sim B \otimes C$
  using *assms bigger-set local.equiv-def* **by** *auto*

lemma *equiv-up-closed-subset*:

assumes *up-closed A*
and *equiv B C*
shows $B \subseteq A \longleftrightarrow C \subseteq A$ (**is** $?B \longleftrightarrow ?C$)
proof −
have $?B \Longrightarrow ?C$
by (*meson greater-set-def up-closed-def equiv-def assms*(*1*) *assms*(*2*) *subsetD subsetI*)
moreover have $?C \Longrightarrow ?B$
by (*meson greater-set-def up-closed-def equiv-def assms*(*1*) *assms*(*2*) *subsetD subsetI*)
ultimately show *?thesis* **by** *blast*
qed

lemma *mono-propI*:
assumes $\bigwedge x\ y.\ y \succeq x \wedge P\ x \Longrightarrow P\ y$
shows *mono-prop P*
using *assms mono-prop-def* **by** *blast*

lemma *mono-prop-set*:
assumes $A \gg B$
and *setify P B*
and *mono-prop P*
shows *setify P A*
using *assms*(*1*) *assms*(*2*) *assms*(*3*) *greater-set-def local.setify-def mono-prop-def*
by *auto*

lemma *mono-prop-set-equiv*:
assumes *mono-prop P*
and *equiv A B*
shows *setify P A* $\longleftrightarrow$ *setify P B*
by (*meson assms*(*1*) *assms*(*2*) *local.equiv-def sep-algebra.mono-prop-set sep-algebra-axioms*)

lemma *setify-sum*:
setify P $(A \otimes B) \longleftrightarrow (\forall\, x \in A.$ *setify P* $(\{x\} \otimes B))$ (**is** $?A \longleftrightarrow ?B$)
proof −
have $?A \Longrightarrow ?B$
using *local.setify-def sep-algebra.add-set-elem sep-algebra-axioms singletonD* **by** *fastforce*
moreover have $?B \Longrightarrow ?A$
using *add-set-elem local.setify-def* **by** *fastforce*
ultimately show *?thesis* **by** *blast*
qed

lemma *setify-sum-image*:
setify P $((Set.image\ f\ A) \otimes B) \longleftrightarrow (\forall\, x \in A.$ *setify P* $(\{f\ x\} \otimes B))$
proof
show *setify P* $(f\ `\ A \otimes B) \Longrightarrow \forall\, x{\in}A.$ *setify P* $(\{f\ x\} \otimes B)$
by (*meson rev-image-eqI sep-algebra.setify-sum sep-algebra-axioms*)
show $\forall\, x{\in}A.$ *setify P* $(\{f\ x\} \otimes B) \Longrightarrow$ *setify P* $(f\ `\ A \otimes B)$

by (*metis* (*mono-tags*, *lifting*) *image-iff sep-algebra.setify-sum sep-algebra-axioms*)
qed

lemma *equivI*:
assumes $A \gg B$
and $B \gg A$
shows *equiv A B*
by (*simp add*: *assms*(*1*) *assms*(*2*) *local.equiv-def*)

lemma *sub-bigger*:
assumes $A \subseteq B$
shows $A \gg B$
by (*meson assms greater-set-def in-mono succ-refl*)

lemma *larger-set-refl*:
$A \gg A$
by (*simp add*: *sub-bigger*)

definition *upper-closure* **where**
upper-closure A = { φ' |φ' φ. $\varphi' \succeq \varphi \wedge \varphi \in A$ }

lemma *upper-closure-up-closed*:
up-closed (*upper-closure A*)
proof (*rule up-closedI*)
fix φ' φ
assume *asm0*: $\varphi' \succeq \varphi \wedge \varphi \in$ *upper-closure A*
then obtain *a* **where** $a \in A \wedge \varphi \succeq a$
using *sep-algebra.upper-closure-def sep-algebra-axioms* **by** *fastforce*
then have $\varphi' \succeq a$
using *asm0 succ-trans* **by** *blast*
then show $\varphi' \in$ *upper-closure A*
using ‹$a \in A \wedge \varphi \succeq a$› *upper-closure-def* **by** *auto*
qed

1.7 Addition of more than two states

lemma *multi-decompose*:
assumes *multi-plus l* ω
shows *length l* $\geq$ *2* $\Longrightarrow$ ($\exists$ *a b la lb*. *l* = *la* @ *lb* $\wedge$ *length la* > *0* $\wedge$ *length lb* > *0* $\wedge$ *multi-plus la a* $\wedge$ *multi-plus lb b* $\wedge$ *Some* ω = *a* $\oplus$ *b*)
using *assms*
apply (*rule multi-plus.cases*)
by *auto*[*2*]

lemma *multi-take-drop*:
assumes *multi-plus l ω*
and *length l ≥ 2*
shows ∃ *n a b*. *n > 0 ∧ n < length l ∧ multi-plus* (*take n l*) *a ∧ multi-plus* (*drop n l*) *b ∧ Some ω = a ⊕ b*
proof −
obtain *a b la lb* **where** *asm0*: *l = la @ lb ∧ length la > 0 ∧ length lb > 0 ∧ multi-plus la a ∧ multi-plus lb b ∧ Some ω = a ⊕ b*
using *assms*(*1*) *assms*(*2*) *multi-decompose* **by** *blast*
let *?n = length la*
have *la = take ?n l*
by (*simp add*: *asm0*)
moreover have *lb = drop ?n l*
by (*simp add*: *asm0*)
ultimately show *?thesis*
by (*metis asm0 length-drop zero-less-diff*)
qed

lemma *multi-plus-single*:
assumes *multi-plus* [*v*] *a*
shows *a = v*
using *assms*
apply (*cases*)
apply *simp*
by (*metis* (*no-types*, *lifting*) *Nil-is-append-conv butlast.simps*(*2*) *butlast-append length-greater-0-conv*)

lemma *multi-plus-two*:
assumes *length l ≥ 2*
shows *multi-plus l ω* ⟷ (∃ *a b la lb*. *l = (la @ lb) ∧ length la > 0 ∧ length lb > 0 ∧ multi-plus la a ∧ multi-plus lb b ∧ Some ω = a ⊕ b*) (**is** *?A* ⟷ *?B*)
by (*meson MPConcat assms multi-decompose*)

lemma *multi-plus-head-tail*:
length l ≤ n ∧ length l ≥ 2 ⟶ (*multi-plus l ω* ⟷ (∃ *r*. *Some ω = (List.hd l) ⊕ r ∧ multi-plus* (*List.tl l*) *r*))
proof (*induction n arbitrary*: *l ω*)
case *0*
then show *?case* **by** *auto*
next
case (*Suc n*)
then have *IH*: ⋀(*l* :: *'a list*) *ω*. *length l ≤ n ∧ length l ≥ 2* ⟶ *multi-plus l ω = (∃ r. Some ω = hd l ⊕ r ∧ multi-plus* (*tl l*) *r*)
by *blast*
then show *?case*
proof (*cases n = 0*)
case *True*
then have *n = 0* **by** *simp*
then show *?thesis* **by** *linarith*

next
case *False*
then have *length* $(l :: 'a\ list) \geq 2 \wedge length\ l \leq n + 1 \Longrightarrow multi\text{-}plus\ l\ \omega \longleftrightarrow (\exists r.\ Some\ \omega = hd\ l \oplus r \wedge multi\text{-}plus\ (tl\ l)\ r)$
(**is** *length* $l \geq 2\ \wedge length\ l \leq n + 1 \Longrightarrow ?A \longleftrightarrow ?B$)
proof −
assume *asm*: *length* $(l :: 'a\ list) \geq 2 \wedge length\ l \leq n + 1$
have $?B \Longrightarrow ?A$
proof −
assume $?B$
then obtain r **where** $Some\ \omega = hd\ l \oplus r \wedge multi\text{-}plus\ (tl\ l)\ r$ **by** *blast*
then have *multi-plus* $[hd\ l]\ (hd\ l)$
using *MPSingle* **by** *blast*
moreover have $[hd\ l]\ @\ (tl\ l) = l$
by (*metis Suc-le-length-iff asm append-Cons list.collapse list.simps*(3) *numeral-2-eq-2 self-append-conv2*)
ultimately show $?A$
by (*metis* (*no-types*, *lifting*) *MPConcat Suc-1 Suc-le-mono asm* ‹*Some* $\omega = hd\ l \oplus r \wedge multi\text{-}plus\ (tl\ l)\ r$› *append-Nil2 length-Cons length-greater-0-conv list.size*(3) *not-one-le-zero zero-less-Suc*)
qed
moreover have $?A \Longrightarrow ?B$
proof −
assume $?A$
then obtain *la lb a b* **where** $l = la\ @\ lb\ length\ la > 0\ length\ lb > 0$ *multi-plus la a multi-plus lb b Some* $\omega = a \oplus b$
using *asm multi-decompose* **by** *blast*
then have *r0*: $length\ la \leq n \wedge length\ la \geq 2 \longrightarrow multi\text{-}plus\ la\ a = (\exists r.\ Some\ a = hd\ la \oplus r \wedge multi\text{-}plus\ (tl\ la)\ r)$
using *IH* **by** *blast*
then show $?B$
proof (*cases length la* ≥ 2)
case *True*
then obtain *ra* **where** *Some a* $= (hd\ la) \oplus ra$ *multi-plus* $(tl\ la)\ ra$
by (*metis Suc-eq-plus1* ‹*0* $<$ *length lb*› ‹*l* $=$ *la* @ *lb*› *r0* ‹*multi-plus la a*› *append-eq-conv-conj asm drop-eq-Nil le-add1 le-less-trans length-append length-greater-0-conv less-Suc-eq-le order.not-eq-order-implies-strict*)
moreover obtain *rab* **where** *Some rab* $= ra \oplus b$
by (*metis* ‹*Some* $\omega = a \oplus b$› *calculation*(1) *asso2 option.exhaust-sel*)
then have *multi-plus* $((tl\ la)\ @\ lb)\ rab$
by (*metis* (*no-types*, *lifting*) *Nil-is-append-conv* ‹*multi-plus lb b*› *calculation*(2) *length-greater-0-conv list.simps*(3) *multi-plus.cases sep-algebra.MPConcat sep-algebra-axioms*)
moreover have *Some* $\omega = hd\ la \oplus rab$
by (*metis* ‹*Some* $\omega = a \oplus b$› ‹*Some rab* $= ra \oplus b$› *asso1 calculation*(1))
ultimately show $?B$
using ‹*0* $<$ *length la*› ‹*l* $=$ *la* @ *lb*› **by** *auto*
next
case *False*

```
        then have length la = 1
          using ‹0 < length la› by linarith
        then have la = [a]
       by (metis Nitpick.size-list-simp(2) One-nat-def Suc-le-length-iff ‹multi-plus
la a› diff-Suc-1 le-numeral-extra(4) length-0-conv list.sel(3) sep-algebra.multi-plus-single
sep-algebra-axioms)
        then show ?thesis
          using ‹Some ω = a ⊕ b› ‹l = la @ lb› ‹multi-plus lb b› by auto
      qed
    qed
    then show ?thesis using calculation by blast
  qed
  then show ?thesis by (metis (no-types, lifting) Suc-eq-plus1)
 qed
qed

lemma not-multi-plus-empty:
 ¬ multi-plus [] ω
 by (metis Nil-is-append-conv length-greater-0-conv list.simps(3) sep-algebra.multi-plus.simps
sep-algebra-axioms)

lemma multi-plus-deter:
 length l ≤ n ⟹ multi-plus l ω ⟹ multi-plus l ω' ⟹ ω = ω'
proof (induction n arbitrary: l ω ω')
 case 0
 then show ?case
  using multi-plus.cases by auto
next
 case (Suc n)
 then show ?case
 proof (cases length l ≥ 2)
  case True
  then obtain r where Some ω = (List.hd l) ⊕ r ∧ multi-plus (List.tl l) r
    using Suc.prems(2) multi-plus-head-tail by blast
   moreover obtain r' where Some ω' = (List.hd l) ⊕ r' ∧ multi-plus (List.tl
l) r'
    using Suc.prems(3) True multi-plus-head-tail by blast
  ultimately have r = r'
    by (metis Suc.IH Suc.prems(1) drop-Suc drop-eq-Nil)
  then show ?thesis
    by (metis ‹Some ω = hd l ⊕ r ∧ multi-plus (tl l) r› ‹Some ω' = hd l ⊕ r' ∧
multi-plus (tl l) r'› option.inject)
 next
  case False
  then have length l ≤ 1
    by simp
  then show ?thesis
  proof (cases length l = 0)
    case True
```

then show *?thesis*
using *Suc.prems(2) sep-algebra.not-multi-plus-empty sep-algebra-axioms* **by** *fastforce*
next
case *False*
then show *?thesis*
by (*metis One-nat-def Suc.prems(2) Suc.prems(3) Suc-length-conv ‹length l ≤ 1› le-SucE le-zero-eq length-greater-0-conv less-numeral-extra(3) sep-algebra.multi-plus-single sep-algebra-axioms*)
qed
qed
qed

lemma *multi-plus-implies-multi-plus-of-drop*:
assumes *multi-plus l ω*
and *n < length l*
shows ∃ *a*. *multi-plus* (*drop n l*) *a* ∧ *ω* ≽ *a*
using *assms*
proof (*induction n arbitrary*: *l ω*)
case *0*
then show *?case* **using** *succ-refl* **by** *fastforce*
next
case (*Suc n*)
then have *length l* ≥ *2*
by *linarith*
then obtain *r* **where** *Some ω* = (*List.hd l*) ⊕ *r* ∧ *multi-plus* (*List.tl l*) *r*
using *Suc.prems(1) multi-plus-head-tail* **by** *blast*
then obtain *a* **where** *multi-plus* (*drop n* (*List.tl l*)) *a* ∧ *r* ≽ *a*
using *Suc.IH Suc.prems(2)* **by** *fastforce*
then show *?case*
by (*metis ‹Some ω = hd l ⊕ r ∧ multi-plus* (*tl l*) *r› bigger-sum-smaller commutative drop-Suc greater-def*)
qed

lemma *multi-plus-bigger-than-head*:
assumes *length l > 0*
and *multi-plus l ω*
shows *ω* ≽ *List.hd l*
proof (*cases length l* ≥ *2*)
case *True*
then obtain *r* **where** *Some ω* = (*List.hd l*) ⊕ *r* ∧ *multi-plus* (*List.tl l*) *r*
using *assms(1) assms(2) multi-plus-head-tail* **by** *blast*
then show *?thesis*
using *greater-def* **by** *blast*
next
case *False*
then show *?thesis*
by (*metis Cons-nth-drop-Suc MPSingle assms(1) assms(2) drop-0 drop-eq-Nil hd-conv-nth length-greater-0-conv not-less-eq-eq numeral-2-eq-2 sep-algebra.multi-plus-deter*

sep-algebra-axioms succ-refl)
qed

lemma *multi-plus-bigger*:
assumes $i < length\ l$
and *multi-plus* $l\ \omega$
shows $\omega \succeq (l\ !\ i)$
proof −
obtain a **where** *multi-plus* $(drop\ i\ l)\ a \wedge \omega \succeq a$
using *assms*(*1*) *assms*(*2*) *multi-plus-implies-multi-plus-of-drop order.strict-trans*
by *blast*
moreover have *List.hd* $(drop\ i\ l) = l\ !\ i$
by (*simp add*: *assms*(*1*) *hd-drop-conv-nth*)
then show *?thesis*
by (*metis* (*no-types*, *lifting*) *succ-trans assms*(*1*) *assms*(*2*) *drop-eq-Nil leD length-greater-0-conv multi-plus-bigger-than-head multi-plus-implies-multi-plus-of-drop*)
qed

lemma *sum-then-singleton*:
Some $a = b \oplus c \longleftrightarrow \{a\} = \{b\} \otimes \{c\}$ (**is** *?A* $\longleftrightarrow$ *?B*)
proof −
have *?A* $\Longrightarrow$ *?B*
proof −
assume *?A*
then have $\{a\} \subseteq \{b\} \otimes \{c\}$
using *add-set-elem* **by** *auto*
moreover have $\{b\} \otimes \{c\} \subseteq \{a\}$
proof (*rule subsetI*)
fix x **assume** $x \in \{b\} \otimes \{c\}$
then show $x \in \{a\}$
by (*metis* ‹*Some* $a = b \oplus c$› *option.sel sep-algebra.add-set-elem sep-algebra-axioms singleton-iff*)
qed
ultimately show *?thesis* **by** *blast*
qed
moreover have *?B* $\Longrightarrow$ *?A*
using *add-set-elem* **by** *auto*
ultimately show *?thesis* **by** *blast*
qed

lemma *empty-set-sum*:
$\{\} \otimes A = \{\}$
by (*simp add*: *add-set-def*)

end

end

2 Package Logic

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

theory *PackageLogic*
imports *Main SepAlgebra*
begin

2.1 Definitions

type-synonym *'a abool* = *'a* ⇒ *bool*

datatype *'a aassertion* =
AStar 'a aassertion 'a aassertion
| *AImp 'a abool 'a aassertion*
| *ASem 'a abool*

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

fixes *unit* :: *'a*
fixes *stable* :: *'a* ⇒ *bool*

assumes *unit-neutral*: *Some a* = *a* ⊕ *unit*
and *stable-sum*: *stable a* ⟹ *stable b* ⟹ *Some x* = *a* ⊕ *b* ⟹ *stable x*
and *stable-unit*: *stable unit*

begin

fun *sat* :: *'a aassertion* ⇒ *'a* ⇒ *bool* **where**
sat (*AStar A B*) *φ* ⟷ (∃ *a b*. *Some φ* = *a* ⊕ *b* ∧ *sat A a* ∧ *sat B b*)
| *sat* (*AImp b A*) *φ* ⟷ (*b φ* ⟶ *sat A φ*)
| *sat* (*ASem A*) *φ* ⟷ *A φ*

definition *mono-pure-cond* **where**
mono-pure-cond b ⟷ (∀ *φ*. *b φ* ⟷ *b* |*φ*|) ∧ (∀ *φ′ φ r*. *pure r* ∧ *Some φ′* = *φ* ⊕ *r* ∧ ¬ *b φ* ⟶ ¬ *b φ′*)

definition *bool-conj* **where**
bool-conj a b x ⟷ *a x* ∧ *b x*

type-synonym *'c pruner* = *'c* ⇒ *bool*

definition *mono-pruner* :: *'a pruner* ⇒ *bool* **where**
mono-pruner p ⟷ (∀ *φ′ φ r*. *pure r* ∧ *p φ* ∧ *Some φ′* = *φ* ⊕ *r* ⟶ *p φ′*)

fun *wf-assertion* **where**
wf-assertion (*AStar A B*) ⟷ *wf-assertion A* ∧ *wf-assertion B*

| *wf-assertion* (*AImp b A*) $\longleftrightarrow$ *mono-pure-cond b* $\wedge$ *wf-assertion A*
| *wf-assertion* (*ASem A*) $\longleftrightarrow$ *mono-pruner A*

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

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

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

AStar: $[\![$ *package-rhs* φ *f S pc A* φ' *f' S'* ; *package-rhs* φ' *f' S' pc B* φ'' *f'' S''* $]\!]$ $\Longrightarrow$ *package-rhs* φ *f S pc* (*AStar A B*) φ'' *f'' S''*
| *AImp*: *package-rhs* φ *f S* (*bool-conj pc b*) *A* φ' *f' S'* $\Longrightarrow$ *package-rhs* φ *f S pc* (*AImp b A*) φ' *f' S'*

| *ASem*: $[\![$ $\bigwedge$*a u T b*. (*a*, *u*, *T*) $\in$ *S* $\Longrightarrow$ *pc a* $\Longrightarrow$ *b* = *witness* (*a*, *u*, *T*) $\Longrightarrow$ *a* $\succeq$ *b* $\wedge$ *B b* ;
S' = { (*a*, *u*, *T*) |*a u T*. (*a*, *u*, *T*) $\in$ *S* $\wedge$ $\neg$ *pc a* }
$\cup$ { (*a* $\ominus$ *b*, *the* (*u* $\oplus$ *b*), *T*) |*a u T b*. (*a*, *u*, *T*) $\in$ *S* $\wedge$ *pc a* $\wedge$ *b* = *witness* (*a*, *u*, *T*) } $]\!]$
$\Longrightarrow$ *package-rhs* φ *f S pc* (*ASem B*) φ *f S'*

| *AddFromOutside*: $[\![$ *Some* φ = φ' $\oplus$ *m* ; *package-rhs* φ' *f' S' pc A* φ'' *f'' S''* ; *stable m* ; *Some f'* = *f* $\oplus$ *m* ;
S' = { (*r*, *u*, *T*) |*a u T r*. (*a*, *u*, *T*) $\in$ *S* $\wedge$ *Some r* = *a* $\oplus$ (*T f'* $\ominus$ *T f*) $\wedge$ *r* ## *u* } $]\!]$
$\Longrightarrow$ *package-rhs* φ *f S pc A* φ'' *f'' S''*

definition *package-sat* **where**
package-sat pc A a' u' u $\longleftrightarrow$ (*pc* |*a'*| $\longrightarrow$ ($\exists$ *x*. *Some x* = |*a'*| $\oplus$ (*u'* $\ominus$ *u*) $\wedge$ *sat A x*))

definition *package-rhs-connection* :: *'a* $\Rightarrow$ *'a* $\Rightarrow$ *'a ext-state set* $\Rightarrow$ *'a abool* $\Rightarrow$ *'a aassertion* $\Rightarrow$ *'a* $\Rightarrow$ *'a* $\Rightarrow$ *'a ext-state set* $\Rightarrow$ *bool* **where**
package-rhs-connection φ *f S pc A* φ' *f' S'* $\longleftrightarrow$ *f'* $\succeq$ *f* $\wedge$ φ ## *f* $\wedge$ φ $\oplus$ *f* = φ' $\oplus$ *f'* $\wedge$ *stable f'* $\wedge$
($\forall$ (*a*, *u*, *T*) $\in$ *S*. $\exists$ *au*. *Some au* = *a* $\oplus$ *u* $\wedge$ (*au* ## (*T f'* $\ominus$ *T f*) $\longrightarrow$
($\exists$ *a' u'*. (*a'*, *u'*, *T*) $\in$ *S'* $\wedge$ |*a'*| $\succeq$ |*a*| $\wedge$ *au* $\oplus$ (*T f'* $\ominus$ *T f*) = *a'* $\oplus$ *u'* $\wedge$ *u'* $\succeq$ *u* $\wedge$ *package-sat pc A a' u' u*)))

definition *mono-transformer* :: *'a transformer* $\Rightarrow$ *bool* **where**
mono-transformer T $\longleftrightarrow$ ($\forall$ φ φ'. φ' $\succeq$ φ $\longrightarrow$ *T* φ' $\succeq$ *T* φ) $\wedge$ *T unit* = *unit*

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

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

definition *pure-remains* **where**
pure-remains S $\longleftrightarrow$ $(\forall\,(a,\ u,\ p) \in S.\ pure\ a)$

definition *is-footprint-general* :: $'a \Rightarrow ('a \Rightarrow 'a \Rightarrow 'a) \Rightarrow 'a\ aassertion \Rightarrow 'a\ aassertion \Rightarrow bool$ **where**
is-footprint-general w R A B $\longleftrightarrow$ $(\forall\ a\ b.\ sat\ A\ a \wedge Some\ b = a \oplus R\ a\ w \longrightarrow sat\ B\ b)$

definition *is-footprint-standard* :: $'a \Rightarrow 'a\ aassertion \Rightarrow 'a\ aassertion \Rightarrow bool$ **where**
is-footprint-standard w A B $\longleftrightarrow$ $(\forall\ a\ b.\ sat\ A\ a \wedge Some\ b = a \oplus w \longrightarrow sat\ B\ b)$

2.2 Lemmas

lemma *is-footprint-generalI*:
assumes $\bigwedge a\ b.\ sat\ A\ a \wedge Some\ b = a \oplus R\ a\ w \Longrightarrow sat\ B\ b$
shows *is-footprint-general w R A B*
using *assms is-footprint-general-def* **by** *blast*

lemma *is-footprint-standardI*:
assumes $\bigwedge a\ b.\ sat\ A\ a \wedge Some\ b = a \oplus w \Longrightarrow sat\ B\ b$
shows *is-footprint-standard w A B*
using *assms is-footprint-standard-def* **by** *blast*

lemma *mono-pure-condI*:
assumes $\bigwedge \varphi.\ b\ \varphi \longleftrightarrow b\ |\varphi|$
and $\bigwedge \varphi\ \varphi'\ r.\ pure\ r \wedge Some\ \varphi' = \varphi \oplus r \wedge \neg\ b\ \varphi \Longrightarrow \neg\ b\ \varphi'$
shows *mono-pure-cond b*
using *assms*(*1*) *assms*(*2*) *mono-pure-cond-def* **by** *blast*

lemma *mono-pure-cond-conj*:
assumes *mono-pure-cond pc*
and *mono-pure-cond b*
shows *mono-pure-cond* (*bool-conj pc b*)
proof (*rule mono-pure-condI*)
show $\bigwedge \varphi.$ *bool-conj pc b* φ = *bool-conj pc b* $|\varphi|$
by (*metis assms*(*1*) *assms*(*2*) *bool-conj-def mono-pure-cond-def*)
show $\bigwedge \varphi\ \varphi'\ r.$ *pure r* $\wedge$ *Some* φ' = $\varphi \oplus r \wedge \neg$ *bool-conj pc b* $\varphi \Longrightarrow \neg$ *bool-conj pc b* φ'
by (*metis assms*(*1*) *assms*(*2*) *bool-conj-def mono-pure-cond-def*)
qed

lemma *bigger-sum*:
assumes *Some* $\varphi = a \oplus b$
and $\varphi' \succeq \varphi$

shows $\exists\, b'.\ b' \succeq b \wedge \mathit{Some}\ \varphi' = a \oplus b'$
proof −
obtain r **where** $\mathit{Some}\ \varphi' = \varphi \oplus r$
using *assms*(2) *greater-def* **by** *blast*
then obtain b' **where** $\mathit{Some}\ b' = b \oplus r$
by (*metis assms*(1) *asso2 domD domI domIff*)
then show *?thesis*
by (*metis* ‹$\mathit{Some}\ \varphi' = \varphi \oplus r$› *assms*(1) *asso1 commutative sep-algebra.greater-equiv sep-algebra-axioms*)
qed

lemma *wf-assertion-sat-larger-pure*:
assumes *wf-assertion A*
and *sat A* φ
and $\mathit{Some}\ \varphi' = \varphi \oplus r$
and *pure r*
shows *sat A* φ'
using *assms*
proof (*induct arbitrary*: φ φ' *r rule*: *wf-assertion.induct*)
case (*1 A B*)
then obtain *a b* **where** $\mathit{Some}\ \varphi = a \oplus b$ *sat A a sat B b* **by** (*meson sat.simps*(1))
then obtain b' **where** $\mathit{Some}\ b' = b \oplus r$
by (*metis 1.prems*(3) *asso2 option.collapse*)
moreover obtain a' **where** $\mathit{Some}\ a' = a \oplus r$
by (*metis 1.prems*(3) ‹$\mathit{Some}\ \varphi = a \oplus b$› *asso2 commutative option.collapse*)
ultimately show *?case*
using *1*
by (*metis* ‹$\mathit{Some}\ \varphi = a \oplus b$› ‹*sat A a*› ‹*sat B b*› *asso1 sat.simps*(1) *wf-assertion.simps*(1))
next
case (*2 b A*)
then show *?case*
by (*metis mono-pure-cond-def sat.simps*(2) *wf-assertion.simps*(2))
next
case (*3 A*)
then show *?case*
by (*metis mono-pruner-def sat.simps*(3) *wf-assertion.simps*(3))
qed

lemma *package-satI*:
assumes *pc* $|a'| \Longrightarrow (\exists\, x.\ \mathit{Some}\ x = |a'| \oplus (u' \ominus u\ ) \wedge \mathit{sat}\ A\ x)$
shows *package-sat pc A* a' u' *u*
by (*simp add*: *assms package-sat-def*)

lemma *package-rhs-connection-instantiate*:
assumes *package-rhs-connection* φ *f S pc A* φ' f' S'
and $(a,\ u,\ T) \in S$
obtains *au* **where** $\mathit{Some}\ au = a \oplus u$

and $au \mathbin{\#\#} (T\, f' \ominus T\, f) \Longrightarrow \exists\, a'\, u'.\ (a',\ u',\ T) \in S' \wedge |a'| \succeq |a| \wedge au \oplus (T\, f' \ominus T\, f) = a' \oplus u' \wedge u' \succeq u \wedge$ *package-sat pc A* $a'\ u'\ u$
using *assms(1) assms(2) package-rhs-connection-def* **by** *fastforce*

lemma *package-rhs-connectionI*:
assumes $\varphi \oplus f = \varphi' \oplus f'$
and *stable* f'
and $\varphi \mathbin{\#\#} f$
and $f' \succeq f$
and $\bigwedge a\ u\ T.\ (a,\ u,\ T) \in S \Longrightarrow (\exists\, au.$ *Some* $au = a \oplus u \wedge (au \mathbin{\#\#} (T\, f' \ominus T\, f) \longrightarrow$
$(\exists\, a'\, u'.\ (a',\ u',\ T) \in S' \wedge |a'| \succeq |a| \wedge au \oplus (T\, f' \ominus T\, f) = a' \oplus u' \wedge u' \succeq u \wedge$ *package-sat pc A* $a'\ u'\ u)))$
shows *package-rhs-connection* $\varphi\ f\ S$ *pc A* $\varphi'\ f'\ S'$
using *package-rhs-connection-def assms* **by** *auto*

lemma *valid-package-setI*:
assumes $\bigwedge a\ u\ T.\ (a,\ u,\ T) \in S \Longrightarrow a \mathbin{\#\#} u \wedge |a| \succeq |u| \wedge$ *mono-transformer* $T \wedge a \succeq |T\, f|$
shows *valid-package-set* $S\ f$
using *assms valid-package-set-def* **by** *auto*

lemma *defined-sum-move*:
assumes $a \mathbin{\#\#} b$
and *Some* $b = x \oplus y$
and *Some* $a' = a \oplus x$
shows $a' \mathbin{\#\#} y$
by (*metis assms defined-def sep-algebra.asso1 sep-algebra-axioms*)

lemma *bigger-core-sum-defined*:
assumes $|a| \succeq b$
shows *Some* $a = a \oplus b$
by (*metis* (*no-types*, *lifting*) *assms asso1 core-is-smaller greater-equiv max-projection-prop-pure-core mpp-prop pure-def pure-smaller*)

lemma *package-rhs-proof*:
assumes *package-rhs* $\varphi\ f\ S$ *pc A* $\varphi'\ f'\ S'$
and *valid-package-set* $S\ f$
and *wf-assertion A*
and *mono-pure-cond pc*
and *stable f*
and $\varphi \mathbin{\#\#} f$
shows *package-rhs-connection* $\varphi\ f\ S$ *pc A* $\varphi'\ f'\ S' \wedge$ *valid-package-set* $S'\ f'$
using *assms*
proof (*induct rule*: *package-rhs.induct*)
case (*AImp* $\varphi\ f\ S$ *pc b A* $\varphi'\ f'\ S'$)
then have *asm0*: *package-rhs-connection* $\varphi\ f\ S$ (*bool-conj pc b*) *A* $\varphi'\ f'\ S' \wedge$ *valid-package-set* $S'\ f'$

using *mono-pure-cond-conj wf-assertion.simps(2)* **by** *blast*
let *?pc* = *bool-conj pc b*
obtain $\varphi \oplus f = \varphi' \oplus f'$ *stable* f' $\varphi \mathbin{\#\#} f$ $f' \succeq f$
and §: $\bigwedge a\ u\ T.\ (a,\ u,\ T) \in S \Longrightarrow (\exists\, au.\ \mathit{Some}\ au = a \oplus u \wedge (au \mathbin{\#\#} (T\ f' \ominus T\ f) \longrightarrow$
$(\exists\, a'\ u'.\ (a',\ u',\ T) \in S' \wedge |a'| \succeq |a| \wedge au \oplus (T\ f' \ominus T\ f) = a' \oplus u' \wedge u' \succeq u \wedge \mathit{package\text{-}sat}\ ?pc\ A\ a'\ u'\ u)))$
using *asm0 package-rhs-connection-def* **by** *force*

have *package-rhs-connection* φ f S *pc* (*AImp b A*) φ' f' S'
proof (*rule package-rhs-connectionI*)
show $\varphi \mathbin{\#\#} f$
by (*simp add*: ‹$\varphi \mathbin{\#\#} f$›)
show $\varphi \oplus f = \varphi' \oplus f'$ **by** (*simp add*: ‹$\varphi \oplus f = \varphi' \oplus f'$›)
show *stable* f' **using** ‹*stable* f'› **by** *simp*
show $f' \succeq f$ **by** (*simp add*: ‹$f' \succeq f$›)
fix $a\ u\ T$ **assume** *asm1*: $(a,\ u,\ T) \in S$
then obtain *au* **where** *asm2*: *Some* $au = a \oplus u \wedge (au \mathbin{\#\#} (T\ f' \ominus T\ f) \longrightarrow$
$(\exists\, a'\ u'.\ (a',\ u',\ T) \in S' \wedge |a'| \succeq |a| \wedge au \oplus (T\ f' \ominus T\ f) = a' \oplus u' \wedge u' \succeq u \wedge \mathit{package\text{-}sat}\ ?pc\ A\ a'\ u'\ u))$
using § **by** *presburger*

then have $au \mathbin{\#\#} (T\ f' \ominus T\ f) \Longrightarrow$
$(\exists\, a'\ u'.\ (a',\ u',\ T) \in S' \wedge |a'| \succeq |a| \wedge au \oplus (T\ f' \ominus T\ f) = a' \oplus u' \wedge u' \succeq u \wedge \mathit{package\text{-}sat}\ \mathit{pc}\ (\mathit{AImp}\ b\ A)\ a'\ u'\ u)$
proof −
assume *asm3*: $au \mathbin{\#\#} (T\ f' \ominus T\ f)$
then obtain $a'\ u'$ **where** *au'*: $(a',\ u',\ T) \in S' \wedge |a'| \succeq |a| \wedge au \oplus (T\ f' \ominus T\ f) = a' \oplus u' \wedge u' \succeq u \wedge \mathit{package\text{-}sat}\ ?pc\ A\ a'\ u'\ u$
using *asm2* **by** *blast*
have (*the* ($|a'| \oplus (u' \ominus u)$)) $\succeq |a'|$
proof −
have $u' \succeq u' \ominus u$
by (*metis minus-default minus-smaller succ-refl*)
then have $a' \mathbin{\#\#} (u' \ominus u)$
by (*metis au' asm3 asso3 defined-def minus-exists*)
then show *?thesis*
by (*metis core-is-smaller defined-def greater-def option.exhaust-sel sep-algebra.asso2 sep-algebra-axioms*)
qed
have *package-sat pc* (*AImp b A*) $a'\ u'\ u$
proof (*rule package-satI*)
assume *pc* $|a'|$
then show $\exists\, x.\ \mathit{Some}\ x = |a'| \oplus (u' \ominus u) \wedge \mathit{sat}\ (\mathit{AImp}\ b\ A)\ x$
proof (*cases b* $|a'|$)
case *True*
then have *?pc* $|a'|$
by (*simp add*: ‹*pc* $|a'|$› *bool-conj-def*)

then show *?thesis*
by (*metis au' package-logic.package-sat-def package-logic-axioms sat.simps*(*2*))
next
case *False*
then have ¬ *b* (*the* (|*a'*| ⊕ (*u'* ⊖ *u*)))
using *AImp.prems*(*2*) ‹*the* (|*a'*| ⊕ (*u'* ⊖ *u*)) ≽ |*a'*|› *core-sum max-projection-prop-def max-projection-prop-pure-core minus-exists mono-pure-cond-def wf-assertion.simps*(*2*)
by *metis*
moreover obtain *x* **where** *Some x* = |*a'*| ⊕ (*u'* ⊖ *u*)
by (*metis au' asm3 asso2 commutative core-is-smaller defined-def minus-and-plus option.collapse*)
ultimately show *?thesis* **by** (*metis option.sel sat.simps*(*2*))
qed
qed
then show ∃ *a' u'*. (*a'*, *u'*, *T*) ∈ *S'* ∧ |*a'*| ≽ |*a*| ∧ *au* ⊕ (*T f'* ⊖ *T f*) = *a'* ⊕ *u'* ∧ *u'* ≽ *u* ∧ *package-sat pc* (*AImp b A*) *a' u' u*
using *au'* **by** *blast*
qed
then show ∃ *au*. *Some au* = *a* ⊕ *u* ∧ (*au* ## (*T f'* ⊖ *T f*) ⟶ (∃ *a' u'*. (*a'*, *u'*, *T*) ∈ *S'* ∧ |*a'*| ≽ |*a*| ∧ *au* ⊕ (*T f'* ⊖ *T f*) = *a'* ⊕ *u'* ∧ *u'* ≽ *u* ∧ *package-sat pc* (*AImp b A*) *a' u' u*))
using *asm2* **by** *auto*
qed
then show *?case*
using ‹*package-rhs-connection* φ *f S* (*bool-conj pc b*) *A* φ' *f' S'* ∧ *valid-package-set S' f'*› **by** *blast*
next
case (*AStar* φ *f S pc A* φ' *f' S' B* φ'' *f'' S''*)
then have *r1*: *package-rhs-connection* φ *f S pc A* φ' *f' S'* ∧ *valid-package-set S' f'*
using *wf-assertion.simps*(*1*) **by** *blast*
moreover have φ' ## *f'* **using** *r1 package-rhs-connection-def*[*of* φ *f S pc A* φ' *f' S'*] *defined-def*
by *fastforce*
ultimately have *r2*: *package-rhs-connection* φ' *f' S' pc B* φ'' *f'' S''* ∧ *valid-package-set S'' f''*
using *AStar.hyps*(*4*) *AStar.prems*(*2*) *AStar.prems*(*3*) *package-rhs-connection-def*
by *force*

moreover obtain *fa-def*: φ ⊕ *f* = φ' ⊕ *f' stable f'* φ ## *f f'* ≽ *f*
and ∗∗: ⋀*a u T*. (*a*, *u*, *T*) ∈ *S* ⟹ (∃ *au*. *Some au* = *a* ⊕ *u* ∧ (*au* ## (*T f'* ⊖ *T f*) ⟶ (∃ *a' u'*. (*a'*, *u'*, *T*) ∈ *S'* ∧ |*a'*| ≽ |*a*| ∧ *au* ⊕ (*T f'* ⊖ *T f*) = *a'* ⊕ *u'* ∧ *u'* ≽ *u* ∧ *package-sat pc A a' u' u*)))
using *r1 package-rhs-connection-def* **by** *fastforce*
then obtain *fb-def*: φ' ⊕ *f'* = φ'' ⊕ *f''* *stable f''* φ' ## *f' f''* ≽ *f'*
and ⋀*a u T*. (*a*, *u*, *T*) ∈ *S'* ⟹ (∃ *au*. *Some au* = *a* ⊕ *u* ∧ (*au* ## (*T f''* ⊖ *T f'*) ⟶

$(\exists\, a'\ u'.\ (a',\ u',\ T) \in S'' \wedge |a'| \succeq |a| \wedge au \oplus (T\ f'' \ominus T\ f') = a' \oplus u' \wedge u' \succeq u \wedge package\text{-}sat\ pc\ B\ a'\ u'\ u)))$
using *r2 package-rhs-connection-def* **by** *fastforce*

moreover have *package-rhs-connection* $\varphi\ f\ S\ pc\ (AStar\ A\ B)\ \varphi''\ f''\ S''$
proof (*rule package-rhs-connectionI*)
show $\varphi \oplus f = \varphi'' \oplus f''$ **by** (*simp add*: *fa-def*(*1*) *fb-def*(*1*))
show *stable* f'' **by** (*simp add*: *fb-def*(*2*))
show $\varphi\ \#\#\ f$ **using** *fa-def*(*3*) **by** *auto*
show $f'' \succeq f$ **using** *fa-def*(*4*) *fb-def*(*4*) *succ-trans* **by** *blast*

fix $a\ u\ T$ **assume** *asm0*: $(a,\ u,\ T) \in S$
then have *f-def*: *Some* $(T\ f'' \ominus T\ f) = (T\ f'' \ominus T\ f') \oplus (T\ f' \ominus T\ f)$
proof −
have *mono-transformer* T **using** *valid-package-set-def asm0* ‹*valid-package-set* $S\ f$› **by** *fastforce*
then have $T\ f'' \succeq T\ f'$
by (*simp add*: *fb-def*(*4*) *mono-transformer-def*)
moreover have $T\ f' \succeq T\ f$
using ‹*mono-transformer* T› *fa-def*(*4*) *mono-transformer-def* **by** *blast*
ultimately show *?thesis*
using *commutative minus-and-plus minus-equiv-def* **by** *presburger*
qed

then obtain *au* **where** *au-def*: *Some* $au = a \oplus u$
$au\ \#\#\ (T\ f' \ominus T\ f) \Longrightarrow (\exists\, a'\ u'.\ (a',\ u',\ T) \in S' \wedge |a'| \succeq |a| \wedge au \oplus (T\ f' \ominus T\ f) = a' \oplus u' \wedge u' \succeq u \wedge package\text{-}sat\ pc\ A\ a'\ u'\ u)$
using $**$ *asm0* **by** *blast*
then show $\exists\, au.\ Some\ au = a \oplus u \wedge (au\ \#\#\ (T\ f'' \ominus T\ f) \longrightarrow (\exists\, a'\ u'.\ (a',\ u',\ T) \in S'' \wedge |a'| \succeq |a| \wedge au \oplus (T\ f'' \ominus T\ f) = a' \oplus u' \wedge u' \succeq u \wedge package\text{-}sat\ pc\ (AStar\ A\ B)\ a'\ u'\ u))$
proof (*cases* $au\ \#\#\ (T\ f'' \ominus T\ f)$)
case *True*
moreover have *mono-transformer* T **using** ‹*valid-package-set* $S\ f$› *valid-package-set-def asm0* **by** *fastforce*
ultimately have $au\ \#\#\ (T\ f'' \ominus T\ f') \wedge au\ \#\#\ (T\ f' \ominus T\ f)$ **using** *asso3 commutative defined-def f-def*
by *metis*
then obtain $a'\ u'$ **where** *r3*: $(a',\ u',\ T) \in S' \wedge |a'| \succeq |a| \wedge au \oplus (T\ f' \ominus T\ f) = a' \oplus u' \wedge u' \succeq u \wedge package\text{-}sat\ pc\ A\ a'\ u'\ u$
using *au-def*(*2*) **by** *blast*

then obtain au' **where** au'*-def*: *Some* $au' = a' \oplus u'$
$au'\ \#\#\ (T\ f'' \ominus T\ f') \Longrightarrow (\exists\, a''\ u''.\ (a'',\ u'',\ T) \in S'' \wedge |a''| \succeq |a'| \wedge au' \oplus (T\ f'' \ominus T\ f') = a'' \oplus u'' \wedge u'' \succeq u' \wedge package\text{-}sat\ pc\ B\ a''\ u''\ u')$
by (*meson package-logic.package-rhs-connection-instantiate package-logic-axioms r2*)

moreover have $au' \mathbin{\#\#} T\,f'' \ominus T\,f'$
using *True r3 calculation(1) commutative defined-sum-move f-def* **by** *fastforce*
ultimately obtain $a''\ u''$ **where** *r4*: $(a'', u'', T) \in S'' \wedge |a''| \succeq |a'| \wedge au' \oplus (T\,f'' \ominus T\,f') = a'' \oplus u'' \wedge u'' \succeq u' \wedge$ *package-sat pc B* $a''\ u''\ u'$
by *blast*

then have $au \oplus (T\,f'' \ominus T\,f) = a'' \oplus u''$
proof −
have $au \oplus (T\,f'' \ominus T\,f) =$ *splus* (*Some au*) (*Some* $(T\,f'' \ominus T\,f)$)
by *simp*
also have ... = *splus* (*Some au*) (*splus* (*Some* $(T\,f'' \ominus T\,f')$) (*Some* $(T\,f' \ominus T\,f)$))
using *f-def* **by** *auto*
finally show *?thesis*
by (*metis* (*full-types*) *r3 r4 au'-def*(1) *splus.simps*(3) *splus-asso splus-comm*)
qed
moreover have *package-sat pc* (*AStar A B*) $a''\ u''\ u$
proof (*rule package-satI*)
assume *pc* $|a''|$
then have *pc* $|a'|$
by (*metis AStar.prems*(3) *r4 greater-equiv minus-core minus-core-weaker minus-equiv-def mono-pure-cond-def pure-def*)
then obtain x **where** *Some* $x = |a'| \oplus (u' \ominus u) \wedge$ *sat A x*
using *r3 package-sat-def* **by** *fastforce*
then obtain x' **where** *Some* $x' = |a''| \oplus (u'' \ominus u') \wedge$ *sat B* x'
using ‹*pc* $|a''|$› *package-sat-def r4* **by** *presburger*

have $u'' \succeq u'' \ominus u$
by (*metis minus-default minus-smaller succ-refl*)
moreover have $a'' \mathbin{\#\#} u''$
using *True* ‹$au \oplus (T\,f'' \ominus T\,f) = a'' \oplus u''$› *defined-def* **by** *auto*
ultimately obtain x'' **where** *Some* $x'' = |a''| \oplus (u'' \ominus u)$
by (*metis commutative defined-def max-projection-prop-pure-core mpp-smaller not-None-eq smaller-compatible*)
moreover have *Some* $(u'' \ominus u) = (u'' \ominus u') \oplus (u' \ominus u)$
using *r4* ‹$(a', u', T) \in S' \wedge |a'| \succeq |a| \wedge au \oplus (T\,f' \ominus T\,f) = a' \oplus u' \wedge u' \succeq u \wedge$ *package-sat pc A* $a'\ u'\ u$› *commutative minus-and-plus minus-equiv-def* **by** *presburger*
moreover have $|a''| \succeq |a'|$
using *r4* **by** *blast*
moreover have *Some* $|a''| = |a'| \oplus |a''|$
by (*metis* (*no-types*, *lifting*) *calculation*(3) *core-is-pure sep-algebra.asso1 sep-algebra.minus-exists sep-algebra-axioms*)
ultimately have *Some* $x'' = x' \oplus x$
using *asso1*[*of - - x'*] ‹*Some* $x = |a'| \oplus (u' \ominus u) \wedge$ *sat A x*› ‹*Some* $x' = |a''| \oplus (u'' \ominus u') \wedge$ *sat B* x'› *commutative*
by *metis*
then show $\exists x.$ *Some* $x = |a''| \oplus (u'' \ominus u) \wedge$ *sat* (*AStar A B*) x

```
        using ‹Some x = |a'| ⊕ (u' ⊖ u) ∧ sat A x› ‹Some x' = |a''| ⊕ (u'' ⊖ u')
∧ sat B x'› ‹Some x'' = |a''| ⊕ (u'' ⊖ u)› commutative by fastforce
      qed
      ultimately show ?thesis
        using r3 r4 au-def(1) succ-trans by blast
    next
      case False
      then show ?thesis
        using au-def(1) by blast
    qed
  qed
  ultimately show ?case by blast
next
  case (ASem S pc witness B S' φ f)
  have valid-package-set S' f
  proof (rule valid-package-setI)
    fix a' u' T assume asm0: (a', u', T) ∈ S'
    then show a' ## u' ∧ |a'| ≽ |u'| ∧ mono-transformer T ∧ a' ≽ |T f|
    proof (cases (a', u', T) ∈ S)
      case True
      then show ?thesis
        using ASem.prems(1) valid-package-set-def by auto
    next
      case False
      then have (a', u', T) ∈ {(a ⊖ b, the (u ⊕ b), T) |a u T b. (a, u, T) ∈ S ∧
pc a ∧ b = witness (a, u, T)}
        using ASem.hyps(2) asm0 by blast
      then obtain a u b where (a, u, T) ∈ S pc a b = witness (a, u, T) a' = a
⊖ b u' = the (u ⊕ b) by blast
      then have a ≽ b ∧ B b
        using ASem.hyps(1) by presburger
      have a ## u
        using ASem.prems(1) ‹(a, u, T) ∈ S› valid-package-set-def by fastforce
      then have Some u' = u ⊕ b
          by (metis ‹a ≽ b ∧ B b› ‹u' = the (u ⊕ b)› commutative defined-def
option.exhaust-sel smaller-compatible)
      moreover have Some a = a' ⊕ b
       using ‹a ≽ b ∧ B b› ‹a' = a ⊖ b› commutative minus-equiv-def by presburger

      ultimately have a' ## u'
        by (metis ‹a ## u› asso1 commutative defined-def)
      moreover have |a'| ≽ |u'|
      proof −
        have |a| ≽ |u|
          using ASem.prems(1) ‹(a, u, T) ∈ S› valid-package-set-def by fastforce
        moreover have |a'| ≽ |a|
          by (simp add: ‹a' = a ⊖ b› minus-core succ-refl)
        moreover have |a'| ≽ |b|
         using ‹a ≽ b ∧ B b› ‹a' = a ⊖ b› max-projection-prop-pure-core minus-core
```

mpp-mono **by** *presburger*
ultimately show *?thesis*
by (*metis* ‹*Some* $u' = u \oplus b$› ‹$a' = a \ominus b$› *core-is-pure core-sum minus-core pure-def smaller-pure-sum-smaller*)
qed
moreover have $a' \succeq |T\ f|$
proof −
have $a \succeq |T\ f|$ **using** ‹$(a,\ u,\ T) \in S$› ‹*valid-package-set S f*› *valid-package-set-def*
by *fastforce*
then show *?thesis*
by (*metis* ‹$a' = a \ominus b$› *max-projection-prop-pure-core minus-core mpp-mono mpp-smaller sep-algebra.mpp-invo sep-algebra.succ-trans sep-algebra-axioms*)
qed
ultimately show *?thesis* **using** ‹$(a,\ u,\ T) \in S$› ‹*valid-package-set S f*› *valid-package-set-def*
by *fastforce*
qed
qed
moreover have *package-rhs-connection* $\varphi\ f\ S\ pc$ (*ASem B*) $\varphi\ f\ S'$
proof (*rule package-rhs-connectionI*)
show $\varphi \oplus f = \varphi \oplus f$
by *simp*
show *stable f* **by** (*simp add*: *ASem.prems*(*4*))
show φ ## f **by** (*simp add*: *ASem.prems*(*5*))
show $f \succeq f$ **by** (*simp add*: *succ-refl*)

fix $a\ u\ T$ **assume** *asm0*: $(a,\ u,\ T) \in S$

then obtain *au* **where** *Some au* = $a \oplus u$ **using** ‹*valid-package-set S f*› *valid-package-set-def defined-def* **by** *auto*
then have *r0*: $(\exists\, a'\ u'.\ (a',\ u',\ T) \in S' \wedge |a'| \succeq |a| \wedge$ *Some au* = $a' \oplus u' \wedge u' \succeq u \wedge$ *package-sat pc* (*ASem B*) $a'\ u'\ u$)
proof −
let *?b* = *witness* $(a,\ u,\ T)$
let *?a* = $a \ominus$ *?b*
let *?u* = *the* $(u \oplus$ *?b*$)$
show $\exists\, a'\ u'.\ (a',\ u',\ T) \in S' \wedge |a'| \succeq |a| \wedge$ *Some au* = $a' \oplus u' \wedge u' \succeq u \wedge$ *package-sat pc* (*ASem B*) $a'\ u'\ u$
proof (*cases pc a*)
case *True*
then have $(?a,\ ?u,\ T) \in S'$ **using** *ASem.hyps*(*2*) *asm0* **by** *blast*
then have $a \succeq ?b \wedge B\ ?b$ **using** *ASem.hyps*(*1*) *True asm0* **by** *blast*
moreover have *r1*: $(?a,\ ?u,\ T) \in S' \wedge |?a| \succeq |a| \wedge$ *Some au* = *?a* $\oplus$ *?u* $\wedge$ *?u* $\succeq u$
proof
show $(a \ominus$ *witness* $(a,\ u,\ T),$ *the* $(u \oplus$ *witness* $(a,\ u,\ T)),\ T) \in S'$
by (*simp add*: ‹$(a \ominus$ *witness* $(a,\ u,\ T),$ *the* $(u \oplus$ *witness* $(a,\ u,\ T)),\ T) \in S'$›)
have $|a \ominus$ *witness* $(a,\ u,\ T)| \succeq |a|$

by (*simp add*: *minus-core succ-refl*)
moreover have *Some au* = *a* ⊖ *witness* (*a*, *u*, *T*) ⊕ *the* (*u* ⊕ *witness* (*a*, *u*, *T*))
using ‹*Some au* = *a* ⊕ *u*› ‹*a* ≽ *witness* (*a*, *u*, *T*) ∧ *B* (*witness* (*a*, *u*, *T*))›
asso1[*of a* ⊖ *witness* (*a*, *u*, *T*) *witness* (*a*, *u*, *T*) *a u the* (*u* ⊕ *witness* (*a*, *u*, *T*))]
commutative option.distinct(*1*) *option.exhaust-sel asso3 minus-equiv-def*
by *metis*
moreover have *the* (*u* ⊕ *witness* (*a*, *u*, *T*)) ≽ *u*
using ‹*Some au* = *a* ⊕ *u*› ‹*a* ≽ *witness* (*a*, *u*, *T*) ∧ *B* (*witness* (*a*, *u*, *T*))› *commutative*
greater-def option.distinct(*1*) *option.exhaust-sel asso3*[*of u witness* (*a*, *u*, *T*)]
by *metis*
ultimately show |*a* ⊖ *witness* (*a*, *u*, *T*)| ≽ |*a*| ∧ *Some au* = *a* ⊖ *witness* (*a*, *u*, *T*) ⊕ *the* (*u* ⊕ *witness* (*a*, *u*, *T*)) ∧ *the* (*u* ⊕ *witness* (*a*, *u*, *T*)) ≽ *u*
by *blast*
qed
moreover have *package-sat pc* (*ASem B*) *?a ?u u*
proof (*rule package-satI*)
assume *pc* |*a* ⊖ *witness* (*a*, *u*, *T*)|
have *Some ?u* = *u* ⊕ *?b*
by (*metis* (*no-types*, *lifting*) ‹*Some au* = *a* ⊕ *u*› *calculation*(*1*) *commutative minus-equiv-def option.distinct*(*1*) *option.exhaust-sel sep-algebra.asso3 sep-algebra-axioms*)
moreover have *?a* ## *?u*
by (*metis r1 defined-def option.distinct*(*1*))
moreover have *?u* ≽ *?u* ⊖ *u*
using *r1 minus-smaller* **by** *blast*
ultimately obtain *x* **where** *Some x* = |*a* ⊖ *?b*| ⊕ (*?u* ⊖ *u*)
by (*metis* (*no-types*, *opaque-lifting*) ‹*a* ≽ *witness* (*a*, *u*, *T*) ∧ *B* (*witness* (*a*, *u*, *T*))› *commutative defined-def minus-core minus-equiv-def option.exhaust smaller-compatible*)
moreover have *x* ≽ *?b*
proof −
have *?u* ⊖ *u* ≽ *?b*
using ‹*Some* (*the* (*u* ⊕ *witness* (*a*, *u*, *T*))) = *u* ⊕ *witness* (*a*, *u*, *T*)› *minus-bigger* **by** *blast*
then show *?thesis*
using *calculation greater-equiv succ-trans* **by** *blast*
qed
ultimately show ∃*x*. *Some x* = |*a* ⊖ *witness* (*a*, *u*, *T*)| ⊕ (*the* (*u* ⊕ *witness* (*a*, *u*, *T*)) ⊖ *u*) ∧ *sat* (*ASem B*) *x*
using *ASem.prems*(*2*) ‹*Some* (*the* (*u* ⊕ *witness* (*a*, *u*, *T*))) = *u* ⊕ *witness* (*a*, *u*, *T*)›
‹*a* ≽ *witness* (*a*, *u*, *T*) ∧ *B* (*witness* (*a*, *u*, *T*))› *commutative max-projection-prop-def*[*of pure core*]
max-projection-prop-pure-core minus-equiv-def-any-elem mono-pruner-def[*of B*]
sat.simps(*3*)[*of B*] *wf-assertion.simps*(*3*)[*of B*]

```
        by metis
      qed
      ultimately show ?thesis by blast
    next
      case False
      then have package-sat pc (ASem B) a u u
        by (metis ASem.prems(3) mono-pure-cond-def package-sat-def)
      moreover have (a, u, T) ∈ S'
        using False ASem.hyps(2) asm0 by blast
      ultimately show ?thesis
        using ‹Some au = a ⊕ u› succ-refl by blast
    qed
  qed
  moreover have au ⊕ (T f ⊖ T f) = Some au
  proof −
  have a ⪰ |T f| using ‹(a, u, T) ∈ S› ‹valid-package-set S f› valid-package-set-def
by fastforce
    then have |a| ⪰ T f ⊖ T f
     using core-is-smaller max-projection-prop-def max-projection-prop-pure-core
minusI by presburger
    then have |au| ⪰ T f ⊖ T f
      using ‹Some au = a ⊕ u› core-sum greater-def succ-trans by blast
    then show ?thesis using bigger-core-sum-defined by force
  qed
  ultimately show ∃ au. Some au = a ⊕ u ∧ (au ## (T f ⊖ T f) ⟶ (∃ a' u'.
(a', u', T) ∈ S' ∧ |a'| ⪰ |a| ∧ au ⊕ (T f ⊖ T f) = a' ⊕ u' ∧ u' ⪰ u ∧ package-sat
pc (ASem B) a' u' u))
    using ‹Some au = a ⊕ u› by fastforce
 qed
 ultimately show ?case by blast
next
 case (AddFromOutside φ φ' m f' S' pc A φ'' f'' S'' f S)
 have valid-package-set S' f'
 proof (rule valid-package-setI)
  fix a' u T assume asm0: (a', u, T) ∈ S'
  then obtain a where (a, u, T) ∈ S a' ## u Some a' = a ⊕ (T f' ⊖ T f)
    using AddFromOutside.hyps(6) by blast
 then have |a| ⪰ |u| ∧ mono-transformer T ∧ a ⪰ |T f| using ‹valid-package-set
S f› valid-package-set-def
    by fastforce
  moreover have a' ⪰ |T f'|
   by (metis (no-types, opaque-lifting) ‹Some a' = a ⊕ (T f' ⊖ T f)› commutative
greater-equiv minus-core minus-equiv-def minus-smaller succ-trans unit-neutral)
  ultimately show a' ## u ∧ |a'| ⪰ |u| ∧ mono-transformer T ∧ a' ⪰ |T f'|
  using ‹Some a' = a ⊕ (T f' ⊖ T f)› ‹a' ## u› greater-def max-projection-prop-pure-core
mpp-mono succ-trans by blast
 qed
 then have r: package-rhs-connection φ' f' S' pc A φ'' f'' S'' ∧ valid-package-set
S'' f''
```

by (*metis AddFromOutside.hyps*(1) *AddFromOutside.hyps*(3) *AddFromOutside.hyps*(4) *AddFromOutside.hyps*(5) *AddFromOutside.prems*(2) *AddFromOutside.prems*(3) *AddFromOutside.prems*(4) *AddFromOutside.prems*(5) *asso1 commutative defined-def stable-sum*)
then obtain *r2*: $\varphi' \oplus f' = \varphi'' \oplus f''$ *stable* f'' $\varphi' \#\# f'$ $f'' \succeq f'$
$\bigwedge a\ u\ T.\ (a,\ u,\ T) \in S' \Longrightarrow (\exists\, au.\ Some\ au = a \oplus u \wedge (au \#\# (T\ f'' \ominus T\ f')$
$\longrightarrow$
$(\exists\, a'\ u'.\ (a',\ u',\ T) \in S'' \wedge |a'| \succeq |a| \wedge au \oplus (T\ f'' \ominus T\ f') = a' \oplus u' \wedge u' \succeq u \wedge package\text{-}sat\ pc\ A\ a'\ u'\ u)))$
using *package-rhs-connection-def* **by** *fastforce*

moreover have *package-rhs-connection* $\varphi\ f\ S\ pc\ A\ \varphi''\ f''\ S''$
proof (*rule package-rhs-connectionI*)
show $\varphi \oplus f = \varphi'' \oplus f''$
by (*metis AddFromOutside.hyps*(1) *AddFromOutside.hyps*(5) *asso1 commutative r2*(1))
show *stable* f''
using *AddFromOutside.hyps*(4) *calculation*(4) *r2*(2) *stable-sum* **by** *blast*
show $\varphi \#\# f$
by (*simp add*: *AddFromOutside.prems*(5))
show $f'' \succeq f$
using *AddFromOutside.hyps*(5) *bigger-sum greater-def r2*(4) **by** *blast*

fix $a\ u\ T$
assume *asm0*: $(a,\ u,\ T) \in S$
then obtain au **where** $Some\ au = a \oplus u$ **using** ‹*valid-package-set* $S\ f$› *valid-package-set-def defined-def*
by *fastforce*
moreover have $au \#\# (T\ f'' \ominus T\ f) \Longrightarrow (\exists\, a'\ u'.\ (a',\ u',\ T) \in S'' \wedge |a'| \succeq |a| \wedge au \oplus (T\ f'' \ominus T\ f) = a' \oplus u' \wedge u' \succeq u \wedge package\text{-}sat\ pc\ A\ a'\ u'\ u)$
proof −
assume *asm1*: $au \#\# (T\ f'' \ominus T\ f)$
moreover have *mono-transformer* T **using** ‹*valid-package-set* $S\ f$› *valid-package-set-def asm0*
by *fastforce*
then have $Some\ (T\ f'' \ominus T\ f) = (T\ f'' \ominus T\ f') \oplus (T\ f' \ominus T\ f)$
by (*metis AddFromOutside.hyps*(5) *commutative greater-equiv minus-and-plus minus-equiv-def mono-transformer-def r2*(4))
ultimately have $a \#\# (T\ f' \ominus T\ f)$
using ‹$Some\ au = a \oplus u$› *asso2 commutative defined-def minus-exists*
by *metis*
then obtain a' **where** $Some\ a' = a \oplus (T\ f' \ominus T\ f)$
by (*meson defined-def option.collapse*)
moreover have $a' \#\# u$
proof −
have $T\ f'' \ominus T\ f \succeq T\ f' \ominus T\ f$
using ‹$Some\ (T\ f'' \ominus T\ f) = T\ f'' \ominus T\ f' \oplus (T\ f' \ominus T\ f)$› *greater-equiv* **by** *blast*
then show *?thesis*

using ‹*Some au* = *a* ⊕ *u*› *asm1 asso1*[*of u a au T f'* ⊖ *T f a'*] *asso2*[*of*] *calculation commutative*
defined-def[*of*] *greater-equiv*[*of T f''* ⊖ *T f T f'* ⊖ *T f*]
by *metis*
qed

ultimately have (*a'*, *u*, *T*) ∈ *S'*
using *AddFromOutside.hyps*(6) *asm0* **by** *blast*

moreover have *au* ## (*T f''* ⊖ *T f'*)
by (*metis* ‹*Some* (*T f''* ⊖ *T f*) = *T f''* ⊖ *T f'* ⊕ (*T f'* ⊖ *T f*)› *asm1 asso3 defined-def*)

then have ∃ *au*. *Some au* = *a'* ⊕ *u* ∧ (*au* ## (*T f''* ⊖ *T f'*) ⟶ (∃ *a'a u'*. (*a'a*, *u'*, *T*) ∈ *S''* ∧ |*a'a*| ≽ |*a'*| ∧ *au* ⊕ (*T f''* ⊖ *T f'*) = *a'a* ⊕ *u'* ∧ *u'* ≽ *u* ∧ *package-sat pc A a'a u' u*))
using *r2*(5) *calculation* **by** *blast*

then obtain *au' a'' u'* **where** *r3*: *Some au'* = *a'* ⊕ *u au'* ## (*T f''* ⊖ *T f'*) ⟹ (*a''*, *u'*, *T*) ∈ *S''* ∧ |*a''*| ≽ |*a'*| ∧ *au'* ⊕ (*T f''* ⊖ *T f'*) = *a''* ⊕ *u'* ∧ *u'* ≽ *u* ∧ *package-sat pc A a'' u' u*
using ‹*au* ## (*T f''* ⊖ *T f'*)› **by** *blast*
moreover have *au'* ## (*T f''* ⊖ *T f'*) **using** ‹*au* ## (*T f''* ⊖ *T f*)› ‹*Some au* = *a* ⊕ *u*› *r3*(1)
‹*Some* (*T f''* ⊖ *T f*) = (*T f''* ⊖ *T f'*) ⊕ (*T f'* ⊖ *T f*)›
‹*Some a'* = *a* ⊕ (*T f'* ⊖ *T f*)› *asso1*[*of u a au T f'* ⊖ *T f a'*] *commutative defined-sum-move*[*of au T f''* ⊖ *T f*]
by *metis*
ultimately have *r4*: (*a''*, *u'*, *T*) ∈ *S''* ∧ |*a''*| ≽ |*a'*| ∧ *au'* ⊕ (*T f''* ⊖ *T f'*) = *a''* ⊕ *u'* ∧ *u'* ≽ *u* ∧ *package-sat pc A a'' u' u*
by *blast*
moreover have |*a''*| ≽ |*a*|
proof −
have |*a'*| ≽ |*a*|
using ‹*Some a'* = *a* ⊕ (*T f'* ⊖ *T f*)› *core-sum greater-def* **by** *blast*
then show *?thesis*
using *r4 succ-trans* **by** *blast*
qed
ultimately show ∃ *a' u'*. (*a'*, *u'*, *T*) ∈ *S''* ∧ |*a'*| ≽ |*a*| ∧ *au* ⊕ (*T f''* ⊖ *T f*) = *a'* ⊕ *u'* ∧ *u'* ≽ *u* ∧ *package-sat pc A a' u' u*
using ‹*Some* (*T f''* ⊖ *T f*) = *T f''* ⊖ *T f'* ⊕ (*T f'* ⊖ *T f*)› ‹*Some a'* = *a* ⊕ (*T f'* ⊖ *T f*)› ‹*Some au* = *a* ⊕ *u*›
commutative r3(1) *asso1 splus.simps*(3) *splus-asso* **by** *metis*
qed
ultimately show ∃ *au*. *Some au* = *a* ⊕ *u* ∧ (*au* ## (*T f''* ⊖ *T f*) ⟶ (∃ *a' u'*. (*a'*, *u'*, *T*) ∈ *S''* ∧ |*a'*| ≽ |*a*| ∧ *au* ⊕ (*T f''* ⊖ *T f*) = *a'* ⊕ *u'* ∧ *u'* ≽ *u* ∧ *package-sat pc A a' u' u*))
by *blast*
qed

ultimately show *?case* **using** *r* **by** *blast*
qed

lemma *unit-core*:
|unit| = unit
by (*meson core-is-pure max-projection-prop-pure-core sep-algebra.cancellative sep-algebra.mpp-invo sep-algebra-axioms unit-neutral*)

lemma *unit-smaller*:
φ ≽ unit
using *greater-equiv unit-neutral* **by** *auto*

2.3 Lemmas for completeness

lemma *sat-star-exists-bigger*:
assumes *sat* (*AStar A B*) *φ*
and *wf-assertion A*
and *wf-assertion B*
shows ∃ *a b*. *|a| ≽ |φ| ∧ |b| ≽ |φ| ∧ Some φ = a ⊕ b ∧ sat A a ∧ sat B b*
proof −
obtain *a b* **where** *Some φ = a ⊕ b sat A a sat B b*
using *assms sat.simps*(*1*) **by** *blast*
then obtain *a′ b′* **where** *Some a′ = a ⊕ |φ| Some b′ = b ⊕ |φ|*
by (*meson defined-def greater-def greater-equiv option.collapse smaller-compatible-core*)
then have *a′ ≽ a ∧ b′ ≽ b*
using *greater-def* **by** *blast*
then have *sat A a′ ∧ sat B b′*
by (*metis ‹Some a′ = a ⊕ |φ|› ‹Some b′ = b ⊕ |φ|› ‹sat A a› ‹sat B b› assms*(*2*) *assms*(*3*) *max-projection-prop-pure-core mpp-prop package-logic.wf-assertion-sat-larger-pure package-logic-axioms*)
moreover have *Some φ = a′ ⊕ b′*
by (*metis* (*no-types*, *lifting*) *‹Some φ = a ⊕ b› ‹Some a′ = a ⊕ |φ|› ‹Some b′ = b ⊕ |φ|› asso1 commutative core-is-smaller*)
ultimately show *?thesis*
by (*metis ‹Some a′ = a ⊕ |φ|› ‹Some b′ = b ⊕ |φ|› commutative extract-core greater-equiv max-projection-prop-pure-core mpp-mono*)
qed

lemma *let-pair-instantiate*:
assumes (*a*, *b*) = *f x y*
shows (*let* (*a*, *b*) = *f x y in g a b*) = *g a b*
by (*metis assms case-prod-conv*)

lemma *greater-than-sum-exists*:
assumes *a ≽ b*
and *Some b = b1 ⊕ b2*
shows ∃ *r*. *Some a = r ⊕ b2 ∧ |r| ≽ |a| ∧ r ≽ b1*
proof −

obtain r **where** *Some* $a = r \oplus b2 \wedge r \succeq b1$
by (*metis assms*(*1*) *assms*(*2*) *bigger-sum commutative*)
then obtain r' **where** *Some* $r' = r \oplus |a|$
by (*metis defined-def greater-def option.exhaust smaller-compatible-core*)
then have *Some* $a = r' \oplus b2$
by (*metis* ‹*Some* $a = r \oplus b2 \wedge r \succeq b1$› *commutative core-is-smaller sep-algebra.asso1 sep-algebra-axioms*)
then show *?thesis*
by (*metis* ‹*Some* $a = r \oplus b2 \wedge r \succeq b1$› ‹*Some* $r' = r \oplus |a|$› *core-is-pure greater-def smaller-than-core succ-trans*)
qed

lemma *bigger-the*:
assumes *Some* $a = x' \oplus y$
and $x' \succeq x$
shows *the* ($|a|$ $\oplus$ x') $\succeq$ *the* ($|a|$ $\oplus$ x)
proof −
have $a \succeq x'$
using *assms*(*1*) *greater-def* **by** *blast*
then have $|a|$ $\#\#$ x'
using *commutative defined-def smaller-compatible-core* **by** *auto*
moreover have $|a|$ $\#\#$ x
by (*metis assms*(*2*) *calculation defined-def sep-algebra.asso3 sep-algebra.minus-exists sep-algebra-axioms*)
ultimately show *?thesis*
using *addition-bigger assms*(*2*) *commutative defined-def* **by** *force*
qed

lemma *wf-assertion-and-the*:
assumes $|a| \#\# b$
and *sat* A b
and *wf-assertion* A
shows *sat* A (*the* ($|a| \oplus b$))
by (*metis assms*(*1*) *assms*(*2*) *assms*(*3*) *commutative defined-def max-projection-prop-pure-core option.collapse sep-algebra.mpp-prop sep-algebra-axioms wf-assertion-sat-larger-pure*)

lemma *minus-some*:
assumes $a \succeq b$
shows *Some* $a = b \oplus (a \ominus b)$
using *assms commutative minus-equiv-def* **by** *force*

lemma *core-mono*:
assumes $a \succeq b$
shows $|a| \succeq |b|$
using *assms max-projection-prop-pure-core mpp-mono* **by** *auto*

lemma *prove-last-completeness*:
assumes $a' \succeq a$
and *Some* $a = nf1 \oplus f2$

shows $a' \ominus nf1 \succeq f2$
by (*meson assms*(*1*) *assms*(*2*) *greater-def greater-minus-trans minus-bigger succ-trans*)

lemma *completeness-aux*:
assumes $\bigwedge a\ u\ T.\ (a,\ u,\ T) \in S \Longrightarrow |f1\ a\ u\ T| \succeq |a| \wedge Some\ a = f1\ a\ u\ T \oplus f2\ a\ u\ T \wedge (pc\ |a| \longrightarrow sat\ A\ (the\ (\ |a| \oplus (f1\ a\ u\ T))))$
and *valid-package-set S f*
and *wf-assertion A*
and *mono-pure-cond pc*
and *stable f* $\wedge$ φ ## *f*
shows $\exists S'.$ *package-rhs* φ *f S pc A* φ *f S'* $\wedge$ $(\forall (a',\ u',\ T) \in S'.\ \exists a\ u.\ (a,\ u,\ T) \in S \wedge a' \succeq f2\ a\ u\ T \wedge |a'| = |a|\ )$
using *assms*
proof (*induct A arbitrary*: *S pc f1 f2*)
case (*AImp b A*)
let *?pc* = *bool-conj pc b*

have *r0*: $\exists S'.$ *package-rhs* φ *f S* (*bool-conj pc b*) *A* φ *f S'* $\wedge$ $(\forall a{\in}S'.$ *case a of* $(a',\ u',\ T) \Rightarrow \exists a\ u.\ (a,\ u,\ T) \in S \wedge a' \succeq f2\ a\ u\ T \wedge |a'| = |a|\ )$
proof (*rule AImp*(*1*))
show *valid-package-set S f*
by (*simp add*: *AImp.prems*(*2*))
show *wf-assertion A* **using** *AImp.prems*(*3*) **by** *auto*
show *mono-pure-cond* (*bool-conj pc b*)
by (*meson AImp.prems*(*3*) *AImp.prems*(*4*) *mono-pure-cond-conj wf-assertion.simps*(*2*))
show *stable f* $\wedge$ φ ## *f* **using** ‹*stable f* $\wedge$ φ ## *f*› **by** *simp*

fix *a u T*
assume *asm0*: $(a,\ u,\ T) \in S$
then have *Some a* = *f1 a u T* $\oplus$ *f2 a u T*
using *AImp.prems*(*1*) **by** *blast*
moreover have *bool-conj pc b* $|a| \Longrightarrow$ *sat A* (*the* ($|a| \oplus$ *f1 a u T*))
proof −
assume *bool-conj pc b* $|a|$
then have *pc* $|a|$
by (*meson bool-conj-def*)
then have $|f1\ a\ u\ T| \succeq |a| \wedge Some\ a = f1\ a\ u\ T \oplus f2\ a\ u\ T \wedge sat\ (AImp\ b\ A)\ (the\ (\ |a| \oplus f1\ a\ u\ T))$
using *AImp.prems*(*1*) *asm0*(*1*) **by** *blast*
moreover have *b* (*the* ($|a| \oplus$ *f1 a u T*))
proof −
have $|a|$ ## *f1 a u T* $\wedge$ $|a| \succeq |f1\ a\ u\ T|$
by (*metis calculation commutative core-is-smaller defined-def greater-def max-projection-prop-pure-core mpp-mono option.discI succ-antisym*)
then obtain *x* **where** *Some x* = $|a| \oplus$ *f1 a u T*
by (*meson defined-def option.collapse*)
then have $|x| = |a|$
by (*metis* ‹*Some x* = $|a| \oplus$ *f1 a u T*› ‹$|a|$ ## *f1 a u T* $\wedge$ $|a| \succeq |f1\ a\ u\ T|$› *commutative core-is-pure core-sum max-projection-prop-pure-core mpp-smaller*

smaller-than-core)
then show *?thesis*
by (*metis AImp.prems*(*3*) ‹*Some x* = |*a*| ⊕ *f1 a u T*› ‹*bool-conj pc b* |*a*|› *bool-conj-def mono-pure-cond-def option.sel wf-assertion.simps*(*2*))
qed
ultimately show *sat A* (*the* (|*a*| ⊕ *f1 a u T*)) by (*metis sat.simps*(*2*))
qed
ultimately show |*f1 a u T*| ≽ |*a*| ∧ *Some a* = *f1 a u T* ⊕ *f2 a u T* ∧ (*bool-conj pc b* |*a*| ⟶ *sat A* (*the* (|*a*| ⊕ *f1 a u T*)))
by (*metis AImp.prems*(*1*) *asm0*)
qed
then obtain *S′* where *r*: *package-rhs φ f S* (*bool-conj pc b*) *A φ f S′* ⋀*a′ u′ T*. (*a′*, *u′*, *T*) ∈ *S′* ⟹ (∃ *a u*. (*a*, *u*, *T*) ∈ *S* ∧ *a′* ≽ *f2 a u T*)
by *fast*
moreover have *package-rhs φ f S pc* (*AImp b A*) *φ f S′*
by (*simp add*: *package-rhs.AImp r*(*1*))
ultimately show *?case*
using *r0 package-rhs.AImp* by *blast*
next
case (*ASem A*)
let *?witness* = λ(*a*, *u*, *T*). *the* (|*a*| ⊕ *f1 a u T*)

obtain *S′* where *S′-def*: *S′* = { (*a*, *u*, *T*) |*a u T*. (*a*, *u*, *T*) ∈ *S* ∧ ¬ *pc a* }
∪ { (*a* ⊖ *b*, *the* (*u* ⊕ *b*), *T*) |*a u T b*. (*a*, *u*, *T*) ∈ *S* ∧ *pc a* ∧ *b* = *?witness* (*a*, *u*, *T*) }
by *blast*

have *package-rhs φ f S pc* (*ASem A*) *φ f S′*
proof (*rule package-rhs.ASem*)
show *S′* = {(*a*, *u*, *T*) |*a u T*. (*a*, *u*, *T*) ∈ *S* ∧ ¬ *pc a*} ∪ {(*a* ⊖ *b*, *the* (*u* ⊕ *b*), *T*) |*a u T b*. (*a*, *u*, *T*) ∈ *S* ∧ *pc a* ∧ *b* = *?witness* (*a*, *u*, *T*)}
using *S′-def* by *blast*
fix *a u T b*
assume *asm0*: (*a*, *u*, *T*) ∈ *S pc a b* = (*case* (*a*, *u*, *T*) *of* (*a*, *u*, *T*) ⇒ *the* (|*a*| ⊕ *f1 a u T*))
then have *b* = *the* (|*a*| ⊕ *f1 a u T*) by *fastforce*
moreover have *pc* |*a*|
by (*meson ASem.prems*(*4*) *asm0*(*2*) *mono-pure-cond-def*)
then obtain |*f1 a u T*| ≽ |*a*| *Some a* = *f1 a u T* ⊕ *f2 a u T sat* (*ASem A*) (*the* (|*a*| ⊕ *f1 a u T*))
using *ASem.prems*(*1*) *asm0*(*1*) by *blast*
then have *Some b* = |*a*| ⊕ *f1 a u T* by (*metis calculation commutative defined-def minus-bigger minus-core option.exhaust-sel smaller-compatible-core*)
moreover have *a* ≽ *b*
proof −
have *a* ≽ *f1 a u T*
using ‹*Some a* = *f1 a u T* ⊕ *f2 a u T*› *greater-def* by *blast*
then show *?thesis*
by (*metis calculation*(*2*) *commutative max-projection-prop-pure-core mpp-smaller*

sep-algebra.mpp-prop sep-algebra-axioms smaller-pure-sum-smaller)
qed
ultimately show $a \succeq b \wedge A\ b$
using ‹*sat* (*ASem A*) (*the* ($|a| \oplus$ *f1 a u T*))› *sat.simps*(*3*) **by** *blast*
qed

moreover have *r0*: $\bigwedge a'\ u'\ T.\ (a',\ u',\ T) \in S' \Longrightarrow (\exists\, a\ u.\ (a,\ u,\ T) \in S \wedge a' \succeq$ *f2 a u T* $\wedge\ |a'| = |a|\ )$
proof −
fix $a'\ u'\ T$ **assume** *asm0*: $(a',\ u',\ T) \in S'$
then show $\exists\, a\ u.\ (a,\ u,\ T) \in S \wedge a' \succeq$ *f2 a u T* $\wedge\ |a'| = |a|$
proof (*cases* $(a',\ u',\ T) \in \{(a,\ u,\ T)\ |a\ u\ T.\ (a,\ u,\ T) \in S \wedge \neg$ *pc a*$\}$)
case *True*
then show *?thesis* **using** *ASem.prems*(*1*) *greater-equiv* **by** *fastforce*
next
case *False*
then have $(a',\ u',\ T) \in \{\ (a \ominus b,$ *the* $(u \oplus b),\ T)\ |a\ u\ T\ b.\ (a,\ u,\ T) \in S \wedge$ *pc a* $\wedge\ b =$ *?witness* $(a,\ u,\ T)\ \}$
using S'*-def asm0* **by** *blast*
then obtain *a u b* **where** $a' = a \ominus b\ u' =$ *the* $(u \oplus b)\ (a,\ u,\ T) \in S$ *pc a b* $=$ *?witness* $(a,\ u,\ T)$
by *blast*
then have $a' \succeq$ *f2 a u T*
proof −
have $a \succeq b$
proof −
have $a \succeq$ *f1 a u T*
using *ASem.prems*(*1*) ‹$(a,\ u,\ T) \in S$› *greater-def* **by** *blast*
moreover have *Some* $b = |a| \oplus$ *f1 a u T*
by (*metis* ‹$b =$ (*case* $(a,\ u,\ T)$ *of* $(a,\ u,\ T) \Rightarrow$ *the* ($|a| \oplus$ *f1 a u T*))› *calculation case-prod-conv commutative defined-def option.exhaust-sel smaller-compatible-core*)
ultimately show *?thesis*
by (*metis commutative max-projection-prop-pure-core mpp-smaller sep-algebra.mpp-prop sep-algebra-axioms smaller-pure-sum-smaller*)
qed
then show *?thesis*
using *ASem.prems*(*1*)[*of a u T*]
‹$(a,\ u,\ T) \in S$› ‹$a' = a \ominus b$› ‹$b =$ (*case* $(a,\ u,\ T)$ *of* $(a,\ u,\ T) \Rightarrow$ *the* ($|a| \oplus$ *f1 a u T*))›
commutative core-is-smaller minus-bigger option.exhaust-sel option.simps(*3*)
asso1[*of f2 a u T f1 a u T a* $|a|$ *the* ($|a| \oplus$ *f1 a u T*)] *asso2*[*of f2 a u T f1 a u T*]
split-conv
by *metis*
qed
then show *?thesis*
using ‹$(a,\ u,\ T) \in S$› ‹$a' = a \ominus b$› *minus-core* **by** *blast*
qed
qed

ultimately show *?case* **by** *blast*
next
case (*AStar A B*)

let *?fA* = $\lambda a\ u\ T.$ *SOME x.* $\exists$ *y. Some* (*f1 a u T*) = $x \oplus y \wedge |x| \succeq |f1\ a\ u\ T| \wedge |y| \succeq |a| \wedge (pc\ |a| \longrightarrow sat\ A\ (the\ (\ |a| \oplus x)) \wedge sat\ B\ (the\ (\ |a| \oplus y)))$
let *?fB* = $\lambda a\ u\ T.$ *SOME y. Some* (*f1 a u T*) = *?fA a u T* $\oplus$ *y* $\wedge$ $|y| \succeq |a| \wedge (pc\ |a| \longrightarrow sat\ B\ (the\ (\ |a| \oplus y)))$
let *?f2* = $\lambda a\ u\ T.$ *the* (*?fB a u T* $\oplus$ *f2 a u T*)

have *r*: $\bigwedge a\ u\ T.\ (a,\ u,\ T) \in S \Longrightarrow$ *Some* (*f1 a u T*) = *?fA a u T* $\oplus$ *?fB a u T* $\wedge$ $|$*?fA a u T*$| \succeq |f1\ a\ u\ T| \wedge |$*?fB a u T*$| \succeq |a| \wedge (pc\ |a| \longrightarrow sat\ A\ (the\ (\ |a| \oplus$ *?fA a u T*$)) \wedge sat\ B\ (the\ (\ |a| \oplus$ *?fB a u T*$)))$
$\wedge$ *Some* (*?f2 a u T*) = *?fB a u T* $\oplus$ *f2 a u T*
proof −
fix *a u T* **assume** *asm0*: (*a*, *u*, *T*) $\in$ *S*
then have *r0*: *Some a* = *f1 a u T* $\oplus$ *f2 a u T* $\wedge$ $(pc\ |a| \longrightarrow sat\ (AStar\ A\ B)\ (the\ (\ |a| \oplus f1\ a\ u\ T)))$
using *AStar.prems*(*1*) **by** *blast*
then have $\exists$ *x y. Some* (*the* ($|a| \oplus$ *f1 a u T*)) = $x \oplus y \wedge (pc\ |a| \longrightarrow sat\ A\ x) \wedge (pc\ |a| \longrightarrow sat\ B\ y) \wedge$
$x \succeq |(the\ (\ |a| \oplus f1\ a\ u\ T))| \wedge y \succeq |(the\ (\ |a| \oplus f1\ a\ u\ T))|$
proof (*cases pc* |*a*|)
case *True*
then show *?thesis*
using *AStar.prems*(*3*) *r0*
max-projection-prop-def[*of pure core*] *max-projection-prop-pure-core*
sat-star-exists-bigger[*of A B* (*the* (|*a*| $\oplus$ *f1 a u T*))]
succ-trans[*of*] *wf-assertion.simps*(*1*)[*of A B*]
by *blast*
next
case *False*
then have *Some* (*the* ($|a| \oplus$ *f1 a u T*)) = *the* ($|a| \oplus$ *f1 a u T*) $\oplus$ $|the\ (\ |a| \oplus f1\ a\ u\ T\ )|$
by (*simp add*: *core-is-smaller*)
then show *?thesis* **by** (*metis False max-projection-prop-pure-core mpp-smaller succ-refl*)
qed
then obtain *x y* **where** *Some* (*the* ($|a| \oplus$ *f1 a u T*)) = $x \oplus y\ pc\ |a| \longrightarrow sat\ A\ x\ pc\ |a| \longrightarrow sat\ B\ y$
$x \succeq |(the\ (\ |a| \oplus f1\ a\ u\ T))|\ y \succeq |(the\ (\ |a| \oplus f1\ a\ u\ T))|$ **by** *blast*
moreover obtain *af* **where** *Some af* = $|a| \oplus f1\ a\ u\ T$
by (*metis r0 commutative defined-def minus-bigger minus-core option.exhaust-sel smaller-compatible-core*)
ultimately have *Some* (*f1 a u T*) = $x \oplus y$
by (*metis AStar.prems*(*1*) *r0 asm0 commutative core-is-smaller greater-def max-projection-prop-pure-core mpp-mono option.sel succ-antisym*)
moreover have $|a|$ ## $x \wedge |a|$ ## *y*

by (*metis ‹Some af = |a| ⊕ f1 a u T›* *calculation commutative defined-def option.discI sep-algebra.asso3 sep-algebra-axioms*)
then have *the (|a| ⊕ x) ⪰ x ∧ the (|a| ⊕ y) ⪰ y*
using *commutative defined-def greater-def* **by** *auto*

ultimately have *pc-implies-sat*: *pc |a| ⟹ sat A (the (|a| ⊕ x)) ∧ sat B (the (|a| ⊕ y))*
by (*metis AStar.prems(3) ‹pc |a| ⟶ sat A x› ‹pc |a| ⟶ sat B y› ‹|a| ## x ∧ |a| ## y› commutative defined-def max-projection-prop-pure-core option.exhaust-sel package-logic.wf-assertion.simps(1) package-logic-axioms sep-algebra.mpp-prop sep-algebra-axioms wf-assertion-sat-larger-pure*)

have *r1*: *∃ y. Some (f1 a u T) = ?fA a u T ⊕ y ∧ |?fA a u T| ⪰ |f1 a u T| ∧ |y| ⪰ |a| ∧ (pc |a| ⟶ sat A (the (|a| ⊕ ?fA a u T)) ∧ sat B (the (|a| ⊕ y)))*
proof (*rule someI-ex*)
show *∃ x y. Some (f1 a u T) = x ⊕ y ∧ |x| ⪰ |f1 a u T| ∧ |y| ⪰ |a| ∧ (pc |a| ⟶ sat A (the (|a| ⊕ x)) ∧ sat B (the (|a| ⊕ y)))*
using *‹Some (f1 a u T) = x ⊕ y› ‹Some (the (|a| ⊕ f1 a u T)) = x ⊕ y› pc-implies-sat ‹x ⪰ |the (|a| ⊕ f1 a u T)|› ‹y ⪰ |the (|a| ⊕ f1 a u T)|› core-is-pure max-projection-propE(3) max-projection-prop-pure-core option.sel pure-def*
by (*metis AStar.prems(1) asm0 minusI minus-core*)
qed
then obtain *yy* **where** *yy-prop*: *Some (f1 a u T) = ?fA a u T ⊕ yy ∧ |?fA a u T| ⪰ |f1 a u T| ∧ |yy| ⪰ |a| ∧ (pc |a| ⟶ sat A (the (|a| ⊕ ?fA a u T)) ∧ sat B (the (|a| ⊕ yy)))*
by *blast*
moreover have *r2*: *Some (f1 a u T) = ?fA a u T ⊕ ?fB a u T ∧ |?fB a u T| ⪰ |a| ∧ (pc |a| ⟶ sat B (the (|a| ⊕ ?fB a u T)))*
proof (*rule someI-ex*)
show *∃ y. Some (f1 a u T) = ?fA a u T ⊕ y ∧ |y| ⪰ |a| ∧ (pc |a| ⟶ sat B (the (|a| ⊕ y)))*
using *r1* **by** *blast*
qed
ultimately have *?fB a u T ⊕ f2 a u T ≠ None*
using *r0*
option.distinct(1) [of] option.exhaust-sel[of ?fB a u T ⊕ f2 a u T] asso2[of ?fA a u T ?fB a u T f1 a u T f2 a u T]
by *metis*
then show *Some (f1 a u T) = ?fA a u T ⊕ ?fB a u T ∧ |?fA a u T| ⪰ |f1 a u T| ∧ |?fB a u T| ⪰ |a|*
∧ (pc |a| ⟶ sat A (the (|a| ⊕ ?fA a u T)) ∧ sat B (the (|a| ⊕ ?fB a u T)))
∧ Some (?f2 a u T) = ?fB a u T ⊕ f2 a u T
using *r0 r2 yy-prop*
option.distinct(1) option.exhaust-sel[of ?fB a u T ⊕ f2 a u T] asso2[of ?fA a u T ?fB a u T f1 a u T f2 a u T]
by *simp*
qed
have *ih1*: *∃ S′. package-rhs φ f S pc A φ f S′ ∧ (∀ a∈S′. case a of (a′, u′, T) ⇒ ∃ a u. (a, u, T) ∈ S ∧ a′ ⪰ ?f2 a u T ∧ |a′| = |a|)*

proof (*rule AStar*(*1*))
show *valid-package-set S f*
by (*simp add*: *AStar.prems*(*2*))
show *wf-assertion A*
using *AStar.prems*(*3*) **by** *auto*
show *mono-pure-cond pc*
by (*simp add*: *AStar.prems*(*4*))
show *stable f* ∧ *φ* ## *f* **using** ‹*stable f* ∧ *φ* ## *f*› **by** *simp*

fix *a u T*
assume *asm0*: (*a*, *u*, *T*) ∈ *S*
then have *b*: *Some* (*f1 a u T*) = *?fA a u T* ⊕ *?fB a u T* ∧ |*?fA a u T*| ≽ |*f1 a u T*| ∧ |*?fB a u T*| ≽ |*a*| ∧ (*pc* |*a*| ⟶ *sat A* (*the* (|*a*| ⊕ *?fA a u T*)) ∧ *sat B* (*the* (|*a*| ⊕ *?fB a u T*)))
using *r* **by** *fast*
show |*?fA a u T*| ≽ |*a*| ∧ *Some a* = *?fA a u T* ⊕ *?f2 a u T* ∧ (*pc* |*a*| ⟶ *sat A* (*the* (|*a*| ⊕ *?fA a u T*)))
proof −
have |*?fA a u T*| ≽ |*a*|
using *AStar.prems*(*1*)[*of a u T*] *asm0 b asso1*[*of ?fA a u T ?fB a u T f1 a u T*]
asso2[*of ?fA a u T ?fB a u T*] *option.sel succ-trans*[*of* |*?fA a u T*| - |*a*|]
by *blast*
moreover have *Some a* = *?fA a u T* ⊕ *?f2 a u T*
using *AStar.prems*(*1*)[*of a u T*] *asm0 b asso1*[*of ?fA a u T ?fB a u T f1 a u T f2 a u T ?f2 a u T*]
asso2[*of ?fA a u T ?fB a u T f1 a u T f2 a u T*] *option.sel option.exhaust-sel*[*of ?fB a u T* ⊕ *f2 a u T Some a* = *?fA a u T* ⊕ *?f2 a u T*]
by *force*
moreover have *pc* |*a*| ⟶ *sat A* (*the* (|*a*| ⊕ *?fA a u T*))
using *AStar.prems*(*1*)[*of a u T*] *asm0 b*
asso2[*of ?fA a u T ?fB a u T*] *option.sel succ-trans*[*of* |*?fA a u T*| - |*a*|]
by *blast*
ultimately show *?thesis*
by *blast*
qed
qed
then obtain *S′* **where** *r′*: *package-rhs φ f S pc A φ f S′* ⋀*a′ u′ T*. (*a′*, *u′*, *T*) ∈ *S′* ⟹ ∃ *a u*. (*a*, *u*, *T*) ∈ *S* ∧ *a′* ≽ *?f2 a u T* ∧ |*a′*| = |*a*|
by *fast*

let *?project* = λ*a′ T*. (*SOME r*. ∃ *a u*. *r* = (*a*, *u*) ∧ (*a*, *u*, *T*) ∈ *S* ∧ *a′* ≽ *?f2 a u T* ∧ |*a′*| = |*a*|)
have *project-prop*: ⋀*a′ u′ T*. (*a′*, *u′*, *T*) ∈ *S′* ⟹ ∃ *a u*. *?project a′ T* = (*a*, *u*) ∧ (*a*, *u*, *T*) ∈ *S* ∧ *a′* ≽ *?f2 a u T* ∧ |*a′*| = |*a*|
proof −
fix *a′ u′ T* **assume** (*a′*, *u′*, *T*) ∈ *S′*
then obtain *a u* **where** (*a*, *u*, *T*) ∈ *S* ∧ *a′* ≽ *?f2 a u T* ∧ |*a′*| = |*a*|
using *r′* **by** *blast*

moreover show $\exists\, a\ u.\ ?project\ a'\ T = (a,\ u) \wedge (a,\ u,\ T) \in S \wedge a' \succeq ?f2\ a\ u\ T \wedge |a'| = |a|$
proof (*rule someI-ex*)
show $\exists\, r\ a\ u.\ r = (a,\ u) \wedge (a,\ u,\ T) \in S \wedge a' \succeq ?f2\ a\ u\ T \wedge |a'| = |a|$ **using** *calculation* **by** *blast*
qed
qed

let *?nf1* = $\lambda a'\ u'\ T.$ *let* $(a,\ u)$ = *?project* $a'\ T$ *in* (*SOME r. Some r* = $|a'| \oplus$ *?fB a u T*)
let *?nf2* = $\lambda a'\ u'\ T.\ a' \ominus$ *?nf1* $a'\ u'\ T$

have $\exists\, S''.$ *package-rhs* $\varphi\ f\ S'$ *pc B* $\varphi\ f\ S'' \wedge (\forall\, a{\in}S''.$ *case a of* $(a',\ u',\ T) \Rightarrow \exists\, a\ u.\ (a,\ u,\ T) \in S' \wedge a' \succeq ?nf2\ a\ u\ T \wedge |a'| = |a|$)
proof (*rule AStar*(*2*))
show *stable f* $\wedge\ \varphi$ *##* f **using** ‹*stable f* $\wedge\ \varphi$ *##* f› **by** *simp*

then show *valid-package-set* $S'\ f$
using *AStar.prems* ‹*package-rhs* $\varphi\ f\ S$ *pc A* $\varphi\ f\ S'$› *package-logic.package-rhs-proof package-logic.wf-assertion.simps*(*1*) *package-logic-axioms*
by *metis*
show *wf-assertion B*
using *AStar.prems*(*3*) **by** *auto*
show *mono-pure-cond pc*
by (*simp add*: *AStar.prems*(*4*))
fix $a'\ u'\ T$ **assume** $(a',\ u',\ T) \in S'$
then obtain $a\ u$ **where** *a-u-def*: $(a,\ u)$ = *?project* $a'\ T$ $(a,\ u,\ T) \in S\ a' \succeq ?f2\ a\ u\ T\ |a'| = |a|$
using *project-prop* **by** *force*
define *nf1* **where** *nf1* = *?nf1* $a'\ u'\ T$
define *nf2* **where** *nf2* = *?nf2* $a'\ u'\ T$
moreover have *rnf1-def*: *Some nf1* = $|a'| \oplus$ *?fB a u T*
proof −
let *?x* = (*SOME r. Some r* = $|a'| \oplus$ *?fB a u T*)
have *Some ?x* = $|a'| \oplus$ *?fB a u T*
proof (*rule someI-ex*)
have *Some* (*f1 a u T*) = *?fA a u T* $\oplus$ *?fB a u T* $\wedge$ $|$*?fA a u T*$| \succeq |f1\ a\ u\ T| \wedge |$*?fB a u T*$| \succeq |a|$
$\wedge$ (*pc* $|a| \longrightarrow$ *sat A* (*the* ($|a| \oplus$ *?fA a u T*)) $\wedge$ *sat B* (*the* ($|a| \oplus$ *?fB a u T*)))
using *r a-u-def* **by** *blast*
then have *Some* (*?f2 a u T*) = *?fB a u T* $\oplus$ *f2 a u T*
by (*metis* (*no-types*, *lifting*) *AStar.prems*(*1*) *a-u-def*(*2*) *asso2 option.distinct*(*1*) *option.exhaust-sel*)
moreover have $a' \succeq$ (*?f2 a u T*) **using** ‹$a' \succeq$ *?f2 a u T*› **by** *blast*
ultimately have $a' \succeq$ *?fB a u T* **using** *succ-trans greater-def*
by *blast*
then obtain r **where** *Some r* = $|a'| \oplus$ *?fB a u T*
using

```
        commutative
        greater-equiv[of a' ?fB a u T]
        minus-equiv-def-any-elem[of a']
       by fastforce
     then show ∃ r. Some r = |a'| ⊕ ?fB a u T by blast
   qed
   moreover have ?nf1 a' u' T = ?x
    using let-pair-instantiate[of a u - a' T λa u. (SOME r. Some r = |a'| ⊕
?fB a u T )] a-u-def
     by fast
   ultimately show ?thesis using nf1-def by argo
 qed

 moreover have rnf2-def: Some a' = nf1 ⊕ nf2
 proof −
   have nf2 = a' ⊖ nf1 using nf1-def nf2-def by blast
   moreover have a' ≽ nf1
   proof −
     have ?f2 a u T ≽ nf1
     proof −
      have Some (?f2 a u T) = ?fB a u T ⊕ f2 a u T using r ‹(a, u, T) ∈ S›
by blast
       then have ?f2 a u T ≽ ?fB a u T
        using greater-def by blast
       moreover have ?f2 a u T ≽ |a'|
       proof −
        have |?f2 a u T| ≽ |a|
        proof −
            have |?f2 a u T| ≽ |?fB a u T| using ‹?f2 a u T ≽ ?fB a u T›
core-mono by blast
          moreover have |?fB a u T| ≽ |a| using r ‹(a, u, T) ∈ S› by blast
          ultimately show ?thesis using succ-trans ‹|a'| = |a|› by blast
        qed
        then show ?thesis
          using a-u-def(4)
            bigger-core-sum-defined[of ?f2 a u T]
            greater-equiv[of - |a|]
          by auto
       qed
       ultimately show ?thesis using
       core-is-pure[of a'] commutative pure-def[of |a'|] smaller-pure-sum-smaller[of
- - |a'|] rnf1-def
        by (metis (no-types, lifting))
     qed
     then show ?thesis using ‹a' ≽ ?f2 a u T› succ-trans by blast
   qed
   ultimately show ?thesis using minus-some nf2-def by blast
 qed
```

moreover have *pc* $|a'| \Longrightarrow$ *sat B* (*the* ($|a'| \oplus$ *?nf1 a' u' T*))
proof −
assume *pc* $|a'|$
moreover have $|a'| = |a|$
by (*simp add*: *a-u-def*(*4*))
then have *pc* $|a|$ **using** ‹*pc* $|a'|$› **by** *simp*
ultimately have *sat B* (*the* ($|a| \oplus$ *?fB a u T*))
using *r a-u-def* **by** *blast*
then have *sat B* (*the* ($|a'| \oplus$ *?fB a u T*)) **using** ‹$|a'| = |a|$› **by** *simp*

then show *sat B* (*the* ($|a'| \oplus$ *?nf1 a' u' T*))
proof −
have *nf1* $\succeq |a'|$ **using** *rnf1-def*
using *greater-def* **by** *blast*
then have *Some nf1* $= |a'| \oplus$ *nf1*
by (*metis bigger-core-sum-defined commutative core-mono max-projection-prop-pure-core mpp-invo*)
then show *?thesis* **using** *nf1-def rnf1-def* ‹*sat B* (*the* ($|a'| \oplus$ *?fB a u T*))› **by** *argo*
qed
qed
moreover have $|$*?nf1 a' u' T*$| \succeq |a'|$
proof −
have *?nf1 a' u' T* $\succeq |a'|$ **using** *nf1-def greater-def rnf1-def* **by** *blast*
then show *?thesis*
using *max-projection-propE*(*3*) *max-projection-prop-pure-core sep-algebra.mpp-prop sep-algebra-axioms* **by** *fastforce*
qed
ultimately show $|$*?nf1 a' u' T*$| \succeq |a'| \wedge$ *Some a'* $=$ *?nf1 a' u' T* $\oplus$ *?nf2 a' u' T* $\wedge$ (*pc* $|a'| \longrightarrow$ *sat B* (*the* ($|a'| \oplus$ *?nf1 a' u' T*)))
using *nf1-def*
by *blast*
qed

then obtain *S''* **where** *S''-prop*: *package-rhs* φ *f S' pc B* φ *f S''* $\bigwedge$*a' u' T*. (*a'*, *u'*, *T*) $\in$ *S''* $\Longrightarrow \exists$ *a u*. (*a*, *u*, *T*) $\in$ *S'* $\wedge$ *a'* $\succeq$ *?nf2 a u T* $\wedge |a'| = |a|$
by *fast*

then have *package-rhs* φ *f S pc* (*AStar A B*) φ *f S''*
using ‹*package-rhs* φ *f S pc A* φ *f S'*› *package-rhs.AStar* **by** *presburger*
moreover have $\bigwedge$*a'' u'' T*. (*a''*, *u''*, *T*) $\in$ *S''* $\Longrightarrow \exists$ *a u*. (*a*, *u*, *T*) $\in$ *S* $\wedge$ *a''* $\succeq$ *f2 a u T* $\wedge |a''| = |a|$
proof −
fix *a'' u'' T* **assume** *asm0*: (*a''*, *u''*, *T*) $\in$ *S''*

then obtain *a' u'* **where** (*a'*, *u'*, *T*) $\in$ *S'* $\wedge$ *a''* $\succeq$ *?nf2 a' u' T* $\wedge |a''| = |a'|$
using *S''-prop* **by** *blast*

```
    then obtain a u where a-u-def: (a, u) = ?project a' T (a, u, T) ∈ S a' ⪰
?f2 a u T |a'| = |a|
      using project-prop by force

    define nf1 where nf1 = ?nf1 a' u' T
    define nf2 where nf2 = ?nf2 a' u' T

    moreover have rnf1-def: Some nf1 = |a'| ⊕ ?fB a u T
    proof −
      let ?x = (SOME r. Some r = |a'| ⊕ ?fB a u T )
      have Some ?x = |a'| ⊕ ?fB a u T
      proof (rule someI-ex)
        have Some (f1 a u T) = ?fA a u T ⊕ ?fB a u T ∧ |?fA a u T| ⪰ |f1 a u
T| ∧ |?fB a u T| ⪰ |a|
          ∧ (pc |a| ⟶ sat A (the ( |a| ⊕ ?fA a u T)) ∧ sat B (the ( |a| ⊕ ?fB a u
T)))
          using r a-u-def by blast
        then have Some (?f2 a u T) = ?fB a u T ⊕ f2 a u T
       by (metis (no-types, lifting) AStar.prems(1) a-u-def(2) asso2 option.distinct(1)
option.exhaust-sel)
        moreover have a' ⪰ (?f2 a u T) using ‹a' ⪰ ?f2 a u T› by blast
        ultimately have a' ⪰ ?fB a u T using succ-trans greater-def
          by blast
        then obtain r where Some r = |a'| ⊕ ?fB a u T
       using commutative greater-equiv[of a' ?fB a u T] minus-equiv-def-any-elem[of
a' ] by fastforce
        then show ∃ r. Some r = |a'| ⊕ ?fB a u T by blast
      qed
      moreover have ?nf1 a' u' T = ?x
        using let-pair-instantiate[of a u - a' T λa u. (SOME r. Some r = |a'| ⊕
?fB a u T )] a-u-def
        by fast
      ultimately show ?thesis using nf1-def by argo
    qed

    moreover have rnf2-def: a' ⪰ nf1 ∧ ?nf2 a' u' T ⪰ f2 a u T
    proof −
      have nf2 = a' ⊖ nf1 using nf1-def nf2-def by blast
      moreover have a' ⪰ nf1 ∧ a' ⊖ nf1 ⪰ f2 a u T
      proof −
        have ?f2 a u T ⪰ nf1
        proof −
          have Some (?f2 a u T) = ?fB a u T ⊕ f2 a u T using r ‹(a, u, T) ∈ S›
by blast
          then have ?f2 a u T ⪰ ?fB a u T
            using greater-def by blast
          moreover have ?f2 a u T ⪰ |a'|
          proof −
            have |?f2 a u T| ⪰ |a|
```

```
        proof −
            have |?f2 a u T| ≽ |?fB a u T| using ‹?f2 a u T ≽ ?fB a u T›
core-mono by blast
          moreover have |?fB a u T| ≽ |a| using r ‹(a, u, T) ∈ S› by blast
          ultimately show ?thesis using succ-trans ‹|a'| = |a|› by blast
        qed
        then show ?thesis
          using a-u-def(4)
            bigger-core-sum-defined
            greater-equiv[of ?f2 a u T |a'|]
          by auto
      qed
      ultimately show ?thesis using
      core-is-pure[of a'] commutative pure-def[of |a'|] smaller-pure-sum-smaller[of
?f2 a u T - |a'|] rnf1-def
        by simp
    qed
    then have r1: a' ≽ nf1 using ‹a' ≽ ?f2 a u T› succ-trans by blast
     then have Some a' = nf1 ⊕ nf2 using minus-some nf2-def ‹nf2 = a' ⊖
nf1› by presburger
    have r2: a' ⊖ nf1 ≽ f2 a u T
      using ‹a' ≽ ?f2 a u T›
    proof (rule prove-last-completeness)

      have Some (?f2 a u T) = ?fB a u T ⊕ f2 a u T
        using r ‹(a, u, T) ∈ S› by blast
      moreover have Some nf1 = |a'| ⊕ ?fB a u T using rnf1-def by blast

      have Some (?f2 a u T) = ?fB a u T ⊕ f2 a u T using r ‹(a, u, T) ∈ S›
by blast
      then have ?f2 a u T ≽ ?fB a u T
        using greater-def by blast
      moreover have ?f2 a u T ≽ |a'|
      proof −
        have |?f2 a u T| ≽ |a|
        proof −
            have |?f2 a u T| ≽ |?fB a u T| using ‹?f2 a u T ≽ ?fB a u T›
core-mono by blast
          moreover have |?fB a u T| ≽ |a| using r ‹(a, u, T) ∈ S› by blast
          ultimately show ?thesis using succ-trans ‹|a'| = |a|› by blast
        qed
        then show ?thesis
          using a-u-def(4)
            bigger-core-sum-defined[of - |a|]
            greater-equiv[of ?f2 a u T |a|]
          by auto
      qed
      ultimately show Some (?f2 a u T) = nf1 ⊕ f2 a u T
        using asso1[of |a'| ?fB a u T nf1 f2 a u T ?f2 a u T]
```

asso1[*of* $|a'|$ $|a'|$ $|a'|$] *core-is-pure*[*of* a'] *greater-def*[*of ?f2 a u T* $|a'|$] *rnf1-def*
by (*metis* (*no-types*, *lifting*))
qed
then show *?thesis* **using** ‹$a' \succeq$ *?f2 a u T*› *succ-trans* **using** *r1* **by** *force*
qed
ultimately show *?thesis* **using** *nf2-def* **by** *argo*
qed
ultimately have $(a, u, T) \in S \wedge a' \succeq$ *?f2 a u T* $\wedge$ *?nf2* $a' u' T \succeq$ *f2 a u T* **using** *nf1-def nf2-def*
a-u-def **by** *blast*
then have $a'' \succeq$ *f2 a u T* $\wedge$ $|a''| = |a'|$ **using** ‹$(a', u', T) \in S' \wedge a'' \succeq$ *?nf2* $a' u' T \wedge |a''| = |a'|$ ›
using *succ-trans* **by** *blast*
then show $\exists\, a\ u.\ (a, u, T) \in S \wedge a'' \succeq$ *f2 a u T* $\wedge |a''| = |a|$ **using** r'
using *a-u-def*(*2*) *a-u-def*(*4*) **by** *auto*
qed
ultimately show *?case* **by** *blast*
qed

2.4 Soundness

theorem *general-soundness*:
assumes *package-rhs* φ *unit* { $(a, unit, T)\ |a\ T.\ (a, T) \in S$ } (λ-. *True*) *A* φ' *f* S'
and $\bigwedge a\ T.\ (a, T) \in S \Longrightarrow$ *mono-transformer T*
and *wf-assertion A*
and *intuitionistic* (*sat A*) $\vee$ *pure-remains* S'
shows *Some* $\varphi = \varphi' \oplus f \wedge$ *stable f* $\wedge$ ($\forall (a, T) \in S.\ a$ *##* $T f \longrightarrow$ *sat A* (*the* $(a \oplus T f)$))
proof −
let *?S* = { $(a, unit, p)\ |a\ p.\ (a, p) \in S$ }
let *?pc* = λ-. *True*
have *package-rhs-connection* φ *unit ?S ?pc A* φ' *f* $S' \wedge$ *valid-package-set* S' *f*
proof (*rule package-rhs-proof*)
show *package-rhs* φ *unit* {$(a, unit, p)\ |a\ p.\ (a, p) \in S$} ($\lambda$-. *True*) *A* φ' *f* S'
using *assms*(*1*) **by** *auto*
show *valid-package-set* {$(a, unit, p)\ |a\ p.\ (a, p) \in S$} *unit*
proof (*rule valid-package-setI*)
fix *a u T*
assume $(a, u, T) \in$ {$(a, unit, p)\ |a\ p.\ (a, p) \in S$}
then have $u = unit$ **by** *blast*
moreover have $|T\ unit| = unit$
using ‹$(a, u, T) \in$ {$(a, unit, p)\ |a\ p.\ (a, p) \in S$}› *assms*(*2*) *mono-transformer-def unit-core* **by** *fastforce*
then show *a* *##* *u* $\wedge\ |a| \succeq |u| \wedge$ *mono-transformer T* $\wedge$ *a* $\succeq\ |T\ unit|$
using ‹$(a, u, T) \in$ {$(a, unit, p)\ |a\ p.\ (a, p) \in S$}› *assms*(*2*) *defined-def unit-core unit-neutral unit-smaller* **by** *auto*
qed

show *wf-assertion A* **by** (*simp add*: *assms*(*3*))
show *mono-pure-cond* (λ-. *True*)
using *mono-pure-cond-def* **by** *auto*
show *stable unit* **by** (*simp add*: *stable-unit*)
show φ ## *unit*
using *defined-def unit-neutral* **by** *auto*
qed

then obtain *r*: $\varphi \oplus unit = \varphi' \oplus f$ *stable f*
$\bigwedge a\ u\ T.\ (a,\ u,\ T) \in ?S \Longrightarrow (\exists\, au.\ Some\ au = a \oplus u \wedge (au\ \#\#\ (T\ f \ominus T\ unit) \longrightarrow$
$(\exists\, a'\ u'.\ (a',\ u',\ T) \in S' \wedge |a'| \succeq |a| \wedge au \oplus (T\ f \ominus T\ unit) = a' \oplus u' \wedge u' \succeq u \wedge package\text{-}sat\ ?pc\ A\ a'\ u'\ u)))$
using *package-rhs-connection-def* **by** *force*

moreover have $\bigwedge a\ T\ x.\ (a,\ T) \in S \wedge Some\ x = a \oplus T\ f \Longrightarrow sat\ A\ x$
proof −
fix *a T x* **assume** *asm0*: $(a,\ T) \in S \wedge Some\ x = a \oplus T\ f$
then have $T\ f \ominus T\ unit = T\ f$
by (*metis assms*(*2*) *commutative minus-equiv-def mono-transformer-def option.sel unit-neutral unit-smaller*)
then obtain *au* **where** *au-def*: $Some\ au = a \oplus unit \wedge (au\ \#\#\ T\ f \longrightarrow$
$(\exists\, a'\ u'.\ (a',\ u',\ T) \in S' \wedge |a'| \succeq |a| \wedge au \oplus T\ f = a' \oplus u' \wedge u' \succeq unit \wedge package\text{-}sat\ ?pc\ A\ a'\ u'\ unit))$
using *r asm0* **by** *fastforce*
then have *au* = *a* **by** (*metis option.inject unit-neutral*)
then have $(\exists\, a'\ u'.\ (a',\ u',\ T) \in S' \wedge |a'| \succeq |a| \wedge a \oplus T\ f = a' \oplus u' \wedge package\text{-}sat\ ?pc\ A\ a'\ u'\ unit)$
using *au-def asm0 defined-def*
by *auto*
then obtain *a' u'* **where** *r0*: $(a',\ u',\ T) \in S' \wedge |a'| \succeq |a| \wedge a \oplus T\ f = a' \oplus u' \wedge package\text{-}sat\ ?pc\ A\ a'\ u'\ unit$
by *presburger*
then obtain *y* **where** $Some\ y = |a'| \oplus (u' \ominus unit)\ sat\ A\ y$
using *package-sat-def* **by** *auto*
then have $Some\ y = |a'| \oplus u'$
by (*metis commutative minus-equiv-def splus.simps*(*3*) *unit-neutral unit-smaller*)
then have $x \succeq y$
by (*metis r0 addition-bigger asm0 max-projection-prop-pure-core mpp-smaller*)
then show *sat A x*
proof (*cases intuitionistic* (*sat A*))
case *True*
then show *?thesis* **by** (*meson* ‹$Some\ y = |a'| \oplus (u' \ominus unit)$› ‹*sat A y*› ‹$x \succeq y$› *intuitionistic-def*)
next
case *False*
then have *pure-remains S'* **using** *assms*(*4*) **by** *auto*
then have *pure a'* **using** *pure-remains-def r0*
by *fast*

then show *?thesis* **using** *r0* ‹*Some* $y = |a'| \oplus (u' \ominus unit)$› ‹*sat* A y› ‹*Some* $y = |a'| \oplus u'$› *asm0 core-is-smaller*
core-max option.sel pure-def asso1[*of* a'] **by** *metis*
qed
qed
then have $(\forall (a,\ T) \in S.\ a\ \#\#\ T\ f \longrightarrow sat\ A\ (the\ (a \oplus T\ f)))$
using *sep-algebra.defined-def sep-algebra-axioms* **by** *fastforce*
moreover have *Some* $\varphi = \varphi' \oplus f \wedge stable\ f$
using *r*(*1*) *r*(*2*) *unit-neutral* **by** *auto*
ultimately show *?thesis* **by** *blast*
qed

theorem *soundness*:
assumes *wf-assertion B*
and $\bigwedge a.\ sat\ A\ a \Longrightarrow a \in S$
and $\bigwedge a.\ a \in S \Longrightarrow mono\text{-}transformer\ (R\ a)$
and *package-rhs* σ *unit* $\{\ (a,\ unit,\ R\ a)\ |a.\ a \in S\ \}$ $(\lambda\text{-}.\ True)$ $B\ \sigma'\ w\ S'$
and *intuitionistic* $(sat\ B) \vee pure\text{-}remains\ S'$
shows *stable* $w \wedge$ *Some* $\sigma = \sigma' \oplus w \wedge$ *is-footprint-general* $w\ R\ A\ B$
proof −
let *?S* $= \{\ (a,\ R\ a)\ |a.\ a \in S\}$
have *r*: *Some* $\sigma = \sigma' \oplus w \wedge stable\ w \wedge (\forall (a,\ T) \in \{(a,\ R\ a)\ |a.\ a \in S\}.\ a\ \#\#\ T\ w \longrightarrow sat\ B\ (the\ (a \oplus T\ w)))$
proof (*rule general-soundness*)
show *package-rhs* σ *unit* $\{(a,\ unit,\ T)\ |a\ T.\ (a,\ T) \in \{(a,\ R\ a)\ |a.\ a \in S\}\}$ $(\lambda\text{-}.\ True)$ $B\ \sigma'\ w\ S'$
using *assms*(*4*) **by** *auto*
show $\bigwedge a\ T.\ (a,\ T) \in \{(a,\ R\ a)\ |a.\ a \in S\} \Longrightarrow mono\text{-}transformer\ T$ **using** *assms*(*3*) **by** *blast*
show *wf-assertion B* **by** (*simp add*: *assms*(*1*))
show *intuitionistic* $(sat\ B) \vee pure\text{-}remains\ S'$ **by** (*simp add*: *assms*(*5*))
qed
moreover have *is-footprint-general w R A B*
proof (*rule is-footprint-generalI*)
fix *a b* **assume** *asm*: *sat A a* $\wedge$ *Some b* $= a \oplus R\ a\ w$
then have $(a,\ R\ a) \in ?S$
using *assms*(*2*) **by** *blast*
then have *sat B* (*the* $(a \oplus R\ a\ w)$) **using** *r* **using** *asm defined-def* **by** *fastforce*
then show *sat B b* **by** (*metis asm option.sel*)
qed
ultimately show *?thesis* **by** *blast*
qed

corollary *soundness-paper*:
assumes *wf-assertion B*
and $\bigwedge a.\ sat\ A\ a \Longrightarrow a \in S$
and *package-rhs* σ *unit* $\{\ (a,\ unit,\ id)\ |a.\ a \in S\ \}$ $(\lambda\text{-}.\ True)$ $B\ \sigma'\ w\ S'$
and *intuitionistic* $(sat\ B) \vee pure\text{-}remains\ S'$
shows *stable* $w \wedge$ *Some* $\sigma = \sigma' \oplus w \wedge$ *is-footprint-standard* $w\ A\ B$

proof −
have *stable w* ∧ *Some σ* = *σ′* ⊕ *w* ∧ *is-footprint-general w* (λ-. *id*) *A B*
using *assms soundness*[*of B A S* λ-. *id σ σ′ w S′*]
by (*simp add*: *mono-transformer-def*)
then show *?thesis*
using *is-footprint-general-def is-footprint-standardI* **by** *fastforce*
qed

2.5 Completeness

theorem *general-completeness*:
assumes ⋀*a u T x*. (*a*, *u*, *T*) ∈ *S* ⟹ *Some x* = *a* ⊕ *T f* ⟹ *sat A x*
and *Some φ* = *φ′* ⊕ *f*
and *stable f*
and *valid-package-set S unit*
and *wf-assertion A*
shows ∃ *S′*. *package-rhs φ unit S* (λ-. *True*) *A φ′ f S′*
proof −
define *S′* **where** *S′* = { (*r*, *u*, *T*) |*a u T r*. (*a*, *u*, *T*) ∈ *S* ∧ *Some r* = *a* ⊕ (*T f* ⊖ *T unit*) ∧ *r* ## *u* }
let *?pc* = λ-. *True*
have ∃ *S″*. *package-rhs φ′ f S′ ?pc A φ′ f S″*
proof −
let *?f2* = λ*a u T*. *unit*
let *?f1* = λ*a u T*. *a*
have ∃ *S″*. *package-rhs φ′ f S′ ?pc A φ′ f S″* ∧ (∀ (*a′*, *u′*, *T*) ∈ *S″*. ∃ *a u*. (*a*, *u*, *T*) ∈ *S′* ∧ *a′* ⪰ *?f2 a u T* ∧ |*a′*| = |*a*|)
proof (*rule completeness-aux*)
show *mono-pure-cond* (λ-. *True*) **by** (*simp add*: *mono-pure-cond-def*)
show *wf-assertion A* **by** (*simp add*: *assms*(*5*))
show *valid-package-set S′ f*
proof (*rule valid-package-setI*)
fix *a′ u′ T*
assume (*a′*, *u′*, *T*) ∈ *S′*
then obtain *a* **where** *asm*: (*a*, *u′*, *T*) ∈ *S* ∧ *Some a′* = *a* ⊕ (*T f* ⊖ *T unit*) ∧ *a′* ## *u′*
using *S′-def* **by** *blast*
then have *a* ## *u′* ∧ |*a*| ⪰ |*u′*| ∧ *mono-transformer T*
using *assms*(*4*) *valid-package-set-def* **by** *fastforce*
moreover have |*T f* ⊖ *T unit*| = |*T f*|
by (*simp add*: *minus-core*)
ultimately show *a′* ## *u′* ∧ |*a′*| ⪰ |*u′*| ∧ *mono-transformer T* ∧ *a′* ⪰ |*T f*|
by (*meson asm core-sum greater-def greater-equiv minus-equiv-def mono-transformer-def succ-trans unit-neutral*)
qed
show *stable f* ∧ *φ′* ## *f*
by (*metis assms*(*2*) *assms*(*3*) *defined-def domI domIff*)
fix *a u T*

assume $(a,\ u,\ T) \in S'$
then obtain $a'\ u'$ **where** $(a',\ u',\ T) \in S$ *Some* $a = a' \oplus (T\ f \ominus T\ unit)$ **using** *S'-def* **by** *blast*
moreover have $T\ f \ominus T\ unit = T\ f$
proof −
have *mono-transformer T* **using** ‹*valid-package-set S unit*› *valid-package-set-def* ‹$(a',\ u',\ T) \in S$› **by** *auto*
then show *?thesis*
by (*metis commutative minus-default minus-equiv-def mono-transformer-def option.sel unit-neutral*)
qed

then have *sat A* (*the* ($|a|$ $\oplus$ *a*))
by (*metis assms*(*1*) *calculation*(*1*) *calculation*(*2*) *commutative core-is-smaller option.sel*)
then show $|a| \succeq |a| \wedge$ *Some a* = *a* $\oplus$ *unit* $\wedge$ (*True* $\longrightarrow$ *sat A* (*the* ($|a|$ $\oplus$ *a*)))
by (*simp add*: *succ-refl unit-neutral*)
qed
then show *?thesis* **by** *auto*
qed
then obtain *S''* **where** *package-rhs* $\varphi'\ f\ S'$ *?pc A* $\varphi'\ f\ S''$ **by** *blast*
have *package-rhs* φ *unit S ?pc A* $\varphi'\ f\ S''$
using *assms*(*2*)
proof (*rule package-rhs.AddFromOutside*)
show *package-rhs* $\varphi'\ f\ S'$ *?pc A* $\varphi'\ f\ S''$
by (*simp add*: ‹*package-rhs* $\varphi'\ f\ S'$ *?pc A* $\varphi'\ f\ S''$›)
show *stable f* **using** *assms*(*3*) **by** *simp*
show *Some f* = *unit* $\oplus$ *f*
by (*simp add*: *commutative unit-neutral*)
show $S' = \{\ (r,\ u,\ T)\ |a\ u\ T\ r.\ (a,\ u,\ T) \in S \wedge$ *Some r* = *a* $\oplus$ $(T\ f \ominus T\ unit) \wedge r\ \#\#\ u\ \}$
using *S'-def* **by** *blast*
qed
then show *?thesis*
by *blast*
qed

theorem *completeness*:
assumes *wf-assertion B*
and *stable w* $\wedge$ *is-footprint-general w R A B*
and *Some* $\sigma = \sigma' \oplus w$
and $\bigwedge a.$ *sat A a* $\Longrightarrow$ *mono-transformer* (*R a*)
shows $\exists S'.$ *package-rhs* σ *unit* $\{(a,\ unit,\ R\ a)\ |a.\ sat\ A\ a\}$ $(\lambda\text{-}.\ True)$ *B* σ' *w S'*
proof −
let *?S* = $\{(a,\ unit,\ R\ a)\ |a.\ sat\ A\ a\}$
have $\exists S'.$ *package-rhs* σ *unit* $\{(a,\ unit,\ R\ a)\ |a.\ sat\ A\ a\}$ $(\lambda\text{-}.\ True)$ *B* σ' *w S'*
proof (*rule general-completeness*[*of ?S w B* σ σ'])

show $\bigwedge a\ u\ T\ x.\ (a,\ u,\ T) \in \{(a,\ unit,\ R\ a)\ |a.\ sat\ A\ a\} \Longrightarrow Some\ x = a \oplus T\ w \Longrightarrow sat\ B\ x$
using *assms(2) is-footprint-general-def* **by** *blast*
show $Some\ \sigma = \sigma' \oplus w$ **by** (*simp add*: *assms(3)*)
show *stable w* **by** (*simp add*: *assms(2)*)
show *wf-assertion B* **by** (*simp add*: *assms(1)*)

show *valid-package-set* $\{(a,\ unit,\ R\ a)\ |a.\ sat\ A\ a\}$ *unit*
proof (*rule valid-package-setI*)
fix *a u T* **assume** *asm0*: $(a,\ u,\ T) \in \{(a,\ unit,\ R\ a)\ |a.\ sat\ A\ a\}$
then have $u = unit \wedge T = R\ a \wedge sat\ A\ a$ **by** *fastforce*
then show $a\ \#\#\ u \wedge\ |a|\ \succeq\ |u|\ \wedge\ mono\text{-}transformer\ T \wedge a \succeq\ |T\ unit|$
using *assms(4) defined-def mono-transformer-def unit-core unit-neutral unit-smaller* **by** *auto*
qed
qed
then show *?thesis* **by** *meson*
qed

corollary *completeness-paper*:
assumes *wf-assertion B*
and *stable w* $\wedge$ *is-footprint-standard w A B*
and $Some\ \sigma = \sigma' \oplus w$
shows $\exists S'.\ package\text{-}rhs\ \sigma\ unit\ \{(a,\ unit,\ id)\ |a.\ sat\ A\ a\}\ (\lambda\text{-}.\ True)\ B\ \sigma'\ w\ S'$
proof −
have $\exists S'.\ package\text{-}rhs\ \sigma\ unit\ \{(a,\ unit,\ (\lambda\text{-}.\ id)\ a)\ |a.\ sat\ A\ a\}\ (\lambda\text{-}.\ True)\ B\ \sigma'\ w\ S'$
using *assms(1)*
proof (*rule completeness*)
show *stable w* $\wedge$ *is-footprint-general w* $(\lambda a.\ id)$ *A B*
using *assms(2) is-footprint-general-def is-footprint-standard-def* **by** *force*
show $Some\ \sigma = \sigma' \oplus w$ **by** (*simp add*: *assms(3)*)
show $\bigwedge a.\ sat\ A\ a \Longrightarrow mono\text{-}transformer\ id$ **using** *mono-transformer-def* **by** *auto*
qed
then show *?thesis* **by** *meson*
qed

end

end

References

[1] C. Calcagno, P. W. O'Hearn, and H. Yang. Local Action and Abstract Separation Logic. In *Logic in Computer Science (LICS)*, pages 366–375, 2007.

[2] T. Dardinier, G. Parthasarathy, N. Weeks, P. Müller, and A. J. Summers. Sound automation of magic wands. In S. Shoham and Y. Vizel, editors, *Computer Aided Verification*, pages 130–151, Cham, 2022. Springer International Publishing.

[3] R. Dockins, A. Hobor, and A. W. Appel. A Fresh Look at Separation Algebras and Share Accounting. In Z. Hu, editor, *Asian Symposium on Programming Languages and Systems (APLAS)*, pages 161–177, 2009.

[4] J. C. Reynolds. Separation Logic: A Logic for Shared Mutable Data Structures. In *Logic in Computer Science (LICS)*, pages 55–74. IEEE, 2002.