

Formalization of CommCSL: A Relational Concurrent Separation Logic for Proving Information Flow Security in Concurrent Programs

Thibault Dardinier
Department of Computer Science
ETH Zurich, Switzerland

April 18, 2024

Abstract

Information flow security ensures that the secret data manipulated by a program does not influence its observable output. Proving information flow security is especially challenging for concurrent programs, where operations on secret data may influence the execution time of a thread and, thereby, the interleaving between threads. Such internal timing channels may affect the observable outcome of a program even if an attacker does not observe execution times. Existing verification techniques for information flow security in concurrent programs attempt to prove that secret data does not influence the relative timing of threads. However, these techniques are often restrictive (for instance because they disallow branching on secret data) and make strong assumptions about the execution platform (ignoring caching, processor instructions with data-dependent execution time, and other common features that affect execution time).

In this entry, we formalize and prove the soundness of COMM-CSL [1], a novel relational concurrent separation logic for proving secure information flow in concurrent programs that lifts these restrictions and does not make any assumptions about timing behavior. The key idea is to prove that all mutating operations performed on shared data commute, such that different thread interleavings do not influence its final value. Crucially, commutativity is required only for an abstraction of the shared data that contains the information that will be leaked to a public output. Abstract commutativity is satisfied by many more operations than standard commutativity, which makes our technique widely applicable.

Contents

| | | |
|----------|--|-----------|
| 1 | State Model | 3 |
| 1.1 | Partial Heaps | 3 |
| 1.2 | Fractional Permissions | 5 |
| 1.3 | Permission Heaps | 10 |
| 1.4 | Extended Heaps | 15 |
| 2 | Imperative Concurrent Language | 34 |
| 2.1 | Language Syntax and Semantics | 34 |
| 2.1.1 | Semantics of expressions | 35 |
| 2.1.2 | Semantics of commands | 35 |
| 2.1.3 | Abort semantics | 36 |
| 2.2 | Useful Definitions and Results | 36 |
| 3 | CommCSL | 45 |
| 3.1 | Assertion Language | 45 |
| 3.2 | Rules of the Logic | 61 |
| 4 | Soundness of CommCSL | 63 |
| 4.1 | Abstract Commutativity | 63 |
| 4.2 | Consistency | 98 |
| 4.3 | Safety and Hoare Triples | 113 |
| 4.3.1 | Preliminaries | 113 |
| 4.3.2 | Safety | 116 |
| 4.3.3 | Useful results about safety | 120 |
| 4.3.4 | Hoare triples | 126 |
| 4.4 | Soundness of the Rules | 127 |
| 4.4.1 | Skip | 127 |
| 4.4.2 | Assign | 128 |
| 4.4.3 | Alloc | 129 |
| 4.4.4 | Write | 138 |
| 4.4.5 | Read | 146 |
| 4.4.6 | Share | 151 |
| 4.4.7 | Atomic | 164 |
| 4.4.8 | Parallel | 192 |
| 4.4.9 | If | 205 |
| 4.4.10 | Sequential composition | 210 |
| 4.4.11 | Frame rule | 213 |
| 4.4.12 | Consequence | 220 |
| 4.4.13 | Existential | 220 |
| 4.4.14 | While loops | 222 |
| 4.4.15 | CommCSL is sound | 236 |
| 4.5 | Corollaries | 236 |

1 State Model

1.1 Partial Heaps

theory *PartialMap*

imports *Main*

begin

type-synonym (*'a*, *'b*) *map* = *'a* \rightarrow *'b*

fun *compatible-options* :: (*'a* \Rightarrow *'a* \Rightarrow *bool*) \Rightarrow *'a option* \Rightarrow *'a option* \Rightarrow *bool* **where**
 compatible-options *f* (*Some a*) (*Some b*) \longleftrightarrow *f a b*
 | *compatible-options* - - - \longleftrightarrow *True*

fun *merge-option* :: (*'b* \Rightarrow *'b* \Rightarrow *'b*) \Rightarrow *'b option* \Rightarrow *'b option* \Rightarrow *'b option* **where**
 merge-option - *None None* = *None*
 | *merge-option* - (*Some a*) *None* = *Some a*
 | *merge-option* - *None* (*Some b*) = *Some b*
 | *merge-option* *f* (*Some a*) (*Some b*) = *Some (f a b)*

definition *merge-options* :: (*'c* \Rightarrow *'c* \Rightarrow *'c*) \Rightarrow (*'b*, *'c*) *map* \Rightarrow (*'b*, *'c*) *map* \Rightarrow (*'b*,
'c) *map* **where**
 merge-options *f a b p* = *merge-option* *f* (*a p*) (*b p*)

definition *compatible-maps* :: (*'b* \Rightarrow *'b* \Rightarrow *bool*) \Rightarrow (*'a*, *'b*) *map* \Rightarrow (*'a*, *'b*) *map* \Rightarrow
bool **where**
 compatible-maps *f h1 h2* \longleftrightarrow (\forall *hl*. *compatible-options* *f* (*h1 hl*) (*h2 hl*))

lemma *compatible-mapsI*:

assumes $\bigwedge x a b. h1 x = \text{Some } a \wedge h2 x = \text{Some } b \implies f a b$

shows *compatible-maps* *f h1 h2*

by (*metis* *assms compatible-maps-def compatible-options.elims*(3))

definition *map-included* :: (*'a*, *'b*) *map* \Rightarrow (*'a*, *'b*) *map* \Rightarrow *bool* **where**
 map-included *h1 h2* \longleftrightarrow ($\forall x. h1 x \neq \text{None} \longrightarrow h1 x = h2 x$)

lemma *map-includedI*:

assumes $\bigwedge x r. h1 x = \text{Some } r \implies h2 x = \text{Some } r$

shows *map-included* *h1 h2*

by (*metis* *assms map-included-def option.exhaust*)

lemma *compatible-maps-empty*:

compatible-maps *f h* (*Map.empty*)

by (*simp* *add: compatible-maps-def*)

lemma *compatible-maps-comm*:

compatible-maps (=) *h1 h2* \longleftrightarrow *compatible-maps* (=) *h2 h1*

proof -

have $\bigwedge a b. \text{compatible-maps } (=) a b \implies \text{compatible-maps } (=) b a$

by (*metis* (*mono-tags*, *lifting*) *compatible-mapsI compatible-maps-def* *compati-*

ble-options.simps(1)

then show *?thesis*

by *auto*

qed

lemma *add-heaps-asso*:

$(h1 ++ h2) ++ h3 = h1 ++ (h2 ++ h3)$

by *auto*

lemma *compatible-maps-same*:

assumes *compatible-maps (=) ha hb*

and *ha x = Some y*

shows $(ha ++ hb) x = Some y$

proof (*cases hb x*)

case *None*

then show *?thesis*

by (*simp add: assms(2) map-add-Some-iff*)

next

case (*Some a*)

then show *?thesis*

by (*metis (mono-tags) assms(1) assms(2) compatible-maps-def compatible-options.simps(1) map-add-def option.simps(5)*)

qed

lemma *compatible-maps-reft*:

compatible-maps (=) h h

using *compatible-maps-def compatible-options.elims(3)* **by** *fastforce*

lemma *map-invo*:

$h ++ h = h$

by (*simp add: map-add-subsumed2*)

lemma *included-then-compatible-maps*:

assumes *map-included h1 h*

and *map-included h2 h*

shows *compatible-maps (=) h1 h2*

proof (*rule compatible-mapsI*)

fix *x a b* **assume** $h1 x = Some a \wedge h2 x = Some b$

show $a = b$

by (*metis <h1 x = Some a & h2 x = Some b> assms(1) assms(2) map-included-def option.inject option.simps(3)*)

qed

lemma *commut-charact*:

assumes *compatible-maps (=) h1 h2*

shows $h1 ++ h2 = h2 ++ h1$

proof (*rule ext*)

fix *x*

show $(h1 ++ h2) x = (h2 ++ h1) x$

```

proof (cases h1 x)
  case None
  then show ?thesis
  by (simp add: domIff map-add-dom-app-simps(2) map-add-dom-app-simps(3))
next
  case (Some a)
  then show ?thesis
  by (simp add: assms compatible-maps-same)
qed
qed

end

```

1.2 Fractional Permissions

```

theory PosRat
  imports Main HOL.Rat
begin

typedef prat = { r :: rat | r. r > 0 } by fastforce

setup-lifting type-definition-prat

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

lift-definition pgte :: prat  $\Rightarrow$  prat  $\Rightarrow$  bool is ( $\geq$ ) done
lift-definition pgt :: prat  $\Rightarrow$  prat  $\Rightarrow$  bool is ( $>$ ) done
lift-definition lt :: prat  $\Rightarrow$  prat  $\Rightarrow$  bool is ( $<$ ) done

lift-definition pmult :: prat  $\Rightarrow$  prat  $\Rightarrow$  prat is (*) by simp
lift-definition padd :: prat  $\Rightarrow$  prat  $\Rightarrow$  prat is (+) by simp

lift-definition pdiv :: prat  $\Rightarrow$  prat  $\Rightarrow$  prat is (/) by simp

lift-definition pmin :: prat  $\Rightarrow$  prat  $\Rightarrow$  prat is (min) by simp
lift-definition pmax :: prat  $\Rightarrow$  prat  $\Rightarrow$  prat is (max) by simp

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

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

lemma pmin-is:
  assumes pgte a b
  shows pmin a b = b

```

by (*metis Rep-prat-inject assms min-absorb2 pgte.rep-eq pmin.rep-eq*)

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

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

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

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

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

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

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

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

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

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

lemma *sum-larger*:
 $pgt (padd a b) a$
using *Rep-prat padd.rep-eq pgt.rep-eq* **by** *auto*

lemma *greater-sum-both*:
assumes *pgte a (padd b c)*
shows $\exists a1 a2. a = padd a1 a2 \wedge pgte a1 b \wedge pgte a2 c$
proof –
obtain *aa bb cc* **where** $aa = Rep-prat a \ bb = Rep-prat b \ cc = Rep-prat c$
by *simp*
then have $aa \geq bb + cc$
using *assms padd.rep-eq pgte.rep-eq* **by** *auto*
then obtain *x* **where** $aa = bb + x \ x \geq cc$
by (*metis add.commute add-le-cancel-left diff-add-cancel*)
then show *?thesis*
by (*metis (no-types, lifting) Abs-prat-inverse Rep-prat Rep-prat-inverse <aa = Rep-prat a> <bb = Rep-prat b> <cc = Rep-prat c> dual-order.trans eq-onp-same-args le-less mem-Collect-eq min-absorb2 min-def order-refl padd.abs-eq pgte.rep-eq*)
qed

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

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

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

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

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

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

definition *pinv where*
 $pinv\ p = pdiv\ pwrite\ p$

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

lemma *pinv-inverts*:
assumes *pgte a b*
shows *pgte (pinv b) (pinv a)*
proof –
have $Rep-prat\ a \geq Rep-prat\ b$
using *assms(1) pgte.rep-eq by auto*
then have $(Fract\ 1\ 1) / Rep-prat\ b \geq (Fract\ 1\ 1) / Rep-prat\ a$
by (*metis One-rat-def Rep-prat frac-le le-numeral-extra(4) mem-Collect-eq zero-le-one*)
then show *?thesis*
by (*simp add: One-rat-def pdiv.rep-eq pgte.rep-eq pinv-def pwrite.rep-eq*)
qed

lemma *pinv-pmult-ok*:
 $pmult\ p\ (pinv\ p) = pwrite$
proof –
obtain *r where r = Rep-prat p by simp*

then have $r * ((\text{Fract } 1 \ 1) / r) = \text{Fract } 1 \ 1$
by (*metis Rep-prat less-numeral-extra*(3) *mem-Collect-eq nonzero-mult-div-cancel-left times-divide-eq-right*)
then show *?thesis*
by (*metis One-rat-def Rep-prat-inject* $\langle r = \text{Rep-prat } p \rangle$ *pdiv.rep-eq pinv-def pmult.rep-eq pwrite.rep-eq*)
qed

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

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

lemma *pmin-sum*:
 $\text{padd } (\text{pmin } a \ b) \ c = \text{pmin } (\text{padd } a \ c) \ (\text{padd } b \ c)$
by (*metis not-pgte-charact pgt-implies-pgte pgte-pgt pmin-comm pmin-is*)

lemma *pmin-sum-larger*:
 $\text{pgte } (\text{pmin } (\text{padd } a1 \ b1) \ (\text{padd } a2 \ b2)) \ (\text{padd } (\text{pmin } a1 \ a2) \ (\text{pmin } b1 \ b2))$
proof (*cases pgte (padd a1 b1) (padd a2 b2)*)
case *True*
then have $\text{pmin } (\text{padd } a1 \ b1) \ (\text{padd } a2 \ b2) = \text{padd } a2 \ b2$
by (*simp add: pmin-is*)
moreover have $\text{pgte } a2 \ (\text{pmin } a1 \ a2) \wedge \text{pgte } b2 \ (\text{pmin } b1 \ b2)$
by (*metis pmin-comm pmin-greater*)
ultimately show *?thesis*
by (*simp add: padd.rep-eq pgte.rep-eq*)
next
case *False*
then have $\text{pmin } (\text{padd } a1 \ b1) \ (\text{padd } a2 \ b2) = \text{padd } a1 \ b1$
by (*metis not-pgte-charact pgt-implies-pgte pmin-comm pmin-is*)
moreover have $\text{pgte } a1 \ (\text{pmin } a1 \ a2) \wedge \text{pgte } b1 \ (\text{pmin } b1 \ b2)$
by (*metis pmin-greater*)
ultimately show *?thesis*
by (*simp add: padd.rep-eq pgte.rep-eq*)

qed

end

1.3 Permission Heaps

theory *FractionalHeap*

imports *Main PosRat PartialMap*

begin

type-synonym $(l, 'v)$ *fract-heap* = $l \rightarrow \text{prat} \times 'v$

definition *compatible-fractions* :: $(l, 'v)$ *fract-heap* \Rightarrow $(l, 'v)$ *fract-heap* \Rightarrow *bool*

where

compatible-fractions h $h' \longleftrightarrow$

$(\forall l p p'. h l = \text{Some } p \wedge h' l = \text{Some } p' \longrightarrow \text{pgte } p \text{ write } (\text{padd } (\text{fst } p) (\text{fst } p')))$

definition *same-values* :: $(l, 'v)$ *fract-heap* \Rightarrow $(l, 'v)$ *fract-heap* \Rightarrow *bool* **where**

same-values h $h' \longleftrightarrow (\forall l p p'. h l = \text{Some } p \wedge h' l = \text{Some } p' \longrightarrow \text{snd } p = \text{snd } p')$

fun *fadd-options* :: $(\text{prat} \times 'v)$ *option* \Rightarrow $(\text{prat} \times 'v)$ *option* \Rightarrow $(\text{prat} \times 'v)$ *option*

where

fadd-options *None* $x = x$

| *fadd-options* x *None* = x

| *fadd-options* $(\text{Some } x)$ $(\text{Some } y) = \text{Some } (\text{padd } (\text{fst } x) (\text{fst } y), \text{snd } x)$

lemma *fadd-options-cancellative*:

assumes *fadd-options* a $x = \text{fadd-options } b$ x

shows $a = b$

proof (*cases* x)

case *None*

then show *?thesis*

by (*metis* *assms* *fadd-options.elims* *option.simps(3)*)

next

case $(\text{Some } xx)$

then have $x = \text{Some } xx$ **by** *simp*

then show *?thesis*

apply (*cases* a)

apply (*cases* b)

apply *simp*

apply (*metis* *assms* *fadd-options.simps(1)* *fadd-options.simps(3)* *fst-conv* *not-pgte-charact* *option.sel* *padd-comm* *pgt-implies-pgte* *sum-larger*)

apply (*cases* b)

apply (*metis* *assms* *fadd-options.simps(1)* *fadd-options.simps(3)* *fst-conv* *not-pgte-charact* *option.sel* *padd-comm* *pgt-implies-pgte* *sum-larger*)

proof –

```

fix aa bb assume a = Some aa b = Some bb
then have snd aa = snd bb
  using Some assms by auto
moreover have fst aa = fst bb
  using padd-cancellative[of padd (fst aa) (fst xx) fst bb fst xx fst aa]
    Some ⟨a = Some aa⟩ ⟨b = Some bb⟩ assms fadd-options.simps(3) fst-conv
option.inject
  by auto
ultimately show a = b
  by (simp add: ⟨a = Some aa⟩ ⟨b = Some bb⟩ prod-eq-iff)
qed
qed

```

definition compatible-fract-heaps :: ('l, 'v) fract-heap ⇒ ('l, 'v) fract-heap ⇒ bool
where

compatible-fract-heaps h h' ⟷ compatible-fractions h h' ∧ same-values h h'

lemma compatible-fract-heapsI:

assumes $\bigwedge l p p'. h l = \text{Some } p \wedge h' l = \text{Some } p' \implies \text{pgte } p \text{ write } (\text{padd } (\text{fst } p))$
 (fst p')

and $\bigwedge l p p'. h l = \text{Some } p \wedge h' l = \text{Some } p' \implies \text{snd } p = \text{snd } p'$

shows compatible-fract-heaps h h'

by (simp add: assms(1) assms(2) compatible-fract-heaps-def compatible-fractions-def same-values-def)

lemma compatible-fract-heapsE:

assumes compatible-fract-heaps h h'

and $h l = \text{Some } p \wedge h' l = \text{Some } p'$

shows pgte p write (padd (fst p)) (fst p')

and $\text{snd } p = \text{snd } p'$

apply (meson assms(1) assms(2) compatible-fract-heaps-def compatible-fractions-def)

by (meson assms(1) assms(2) compatible-fract-heaps-def same-values-def)

lemma compatible-fract-heaps-comm:

assumes compatible-fract-heaps h h'

shows compatible-fract-heaps h' h

proof (rule compatible-fract-heapsI)

show $\bigwedge l p p'. h' l = \text{Some } p \wedge h l = \text{Some } p' \implies \text{pgte } p \text{ write } (\text{padd } (\text{fst } p))$
 (fst p')

by (metis assms compatible-fract-heapsE(1) padd-comm)

show $\bigwedge l p p'. h' l = \text{Some } p \wedge h l = \text{Some } p' \implies \text{snd } p = \text{snd } p'$

using assms compatible-fract-heapsE(2) **by** fastforce

qed

The following definition only makes sense if h and h' are compatible

definition add-fh :: ('l, 'v) fract-heap ⇒ ('l, 'v) fract-heap ⇒ ('l, 'v) fract-heap
where

add-fh h h' l = fadd-options (h l) (h' l)

definition *full-ownership* :: (*l*, *v*) *fract-heap* \Rightarrow *bool* **where**
full-ownership *h* $\longleftrightarrow (\forall l\ p. h\ l = \text{Some } p \longrightarrow \text{fst } p = \text{pwrite})$

lemma *full-ownershipI*:
assumes $\bigwedge l\ p. h\ l = \text{Some } p \implies \text{fst } p = \text{pwrite}$
shows *full-ownership* *h*
by (*simp add: assms full-ownership-def*)

fun *apply-opt* **where**
apply-opt *f* *None* = *None*
| *apply-opt* *f* (*Some* *x*) = *Some* (*f* *x*)

definition *normalize* :: (*l*, *v*) *fract-heap* \Rightarrow (*l* \rightarrow *v*) **where**
normalize *h* *l* = *apply-opt snd* (*h* *l*)

lemma *normalize-eq*:
normalize *h* *l* = *None* $\longleftrightarrow h\ l = \text{None}$
normalize *h* *l* = *Some* *v* $\longleftrightarrow (\exists p. h\ l = \text{Some } (p, v))$ (**is** *?A* \longleftrightarrow *?B*)
apply (*metis FractionalHeap.normalize-def apply-opt.elims option.distinct(1)*)

proof

show *?B* \implies *?A*
by (*metis FractionalHeap.normalize-def apply-opt.simps(2) snd-eqD*)
assume *?A* **then have** *h* *l* \neq *None*
by (*metis FractionalHeap.normalize-def apply-opt.simps(1) option.distinct(1)*)
then obtain *p* **where** *h* *l* = *Some* *p*
by *blast*
then show *?B*
by (*metis FractionalHeap.normalize-def <FractionalHeap.normalize h l = Some v> <h l \neq None> apply-opt.elims option.inject prod.exhaust-sel*)
qed

definition *fpdom* **where**
fpdom *h* = $\{x. \exists v. h\ x = \text{Some } (\text{pwrite}, v)\}$

lemma *compatible-then-dom-disjoint*:
assumes *compatible-fract-heaps* *h1* *h2*
shows $\text{dom } h1 \cap \text{fpdom } h2 = \{\}$
and $\text{dom } h2 \cap \text{fpdom } h1 = \{\}$

proof –

have *r*: $\bigwedge h1\ h2. \text{compatible-fract-heaps } h1\ h2 \implies \text{dom } h1 \cap \text{fpdom } h2 = \{\}$

proof –

fix *h1* *h2* **assume** *asm0*: *compatible-fract-heaps* *h1* *h2*

show $\text{dom } h1 \cap \text{fpdom } h2 = \{\}$

proof

show $\text{dom } h1 \cap \text{fpdom } h2 \subseteq \{\}$

proof

fix *x* **assume** $x \in \text{dom } h1 \cap \text{fpdom } h2$

```

then have  $x \in \text{dom } h1 \wedge x \in \text{fpdom } h2$  by auto
then have  $h1\ x \neq \text{None} \wedge h2\ x \neq \text{None}$ 
  using  $\text{domIff fpdom-def[of } h2] \text{ mem-Collect-eq option.discI}$ 
  by auto
then obtain  $a\ b$  where  $h1\ x = \text{Some } a\ h2\ x = \text{Some } b$  by auto
then have  $\text{fst } b = \text{pwrite} \wedge \text{pgte } \text{pwrite } (\text{padd } (\text{fst } a) (\text{fst } b))$ 
  using  $\langle x \in \text{dom } h1 \wedge x \in \text{fpdom } h2 \rangle \text{ asm0 compatible-fract-heapsE}(1)$ 
 $\text{fpdom-def[of } h2] \text{ fst-conv mem-Collect-eq option.sel}$ 
  by fastforce
then show  $x \in \{\}$ 
  by  $(\text{metis not-pgte-charact padd-comm sum-larger})$ 
qed
qed  $(\text{simp})$ 
qed
then show  $\text{dom } h1 \cap \text{fpdom } h2 = \{\}$ 
  using  $\text{assms}$  by blast
show  $\text{dom } h2 \cap \text{fpdom } h1 = \{\}$ 
  by  $(\text{simp add: assms compatible-fract-heaps-comm } r)$ 
qed

```

lemma *compatible-dom-sum:*

```

assumes  $\text{compatible-fract-heaps } h1\ h2$ 
shows  $\text{dom } (\text{add-fh } h1\ h2) = \text{dom } h1 \cup \text{dom } h2$  (is ?A = ?B)
proof
show  $?B \subseteq ?A$ 
proof
  fix  $x$  assume  $x \in ?B$ 
  show  $x \in ?A$ 
  proof  $(\text{cases } x \in \text{dom } h1)$ 
    case True
      then show  $?thesis$  using  $\text{add-fh-def[of } h1\ h2] \text{ domI domIff fadd-options.elims}$ 
      by metis
    next
      case False
        then have  $x \in \text{dom } h2$ 
        using  $\langle x \in \text{dom } h1 \cup \text{dom } h2 \rangle$  by auto
        then show  $?thesis$  using  $\text{add-fh-def[of } h1\ h2] \text{ domI domIff fadd-options.elims}$ 
        by metis
  qed
qed
show  $?A \subseteq ?B$ 
  using  $\text{UnI1[of - dom } h1\ \text{dom } h2] \text{ UnI2[of - dom } h1\ \text{dom } h2] \text{ add-fh-def[of } h1$ 
 $h2] \text{ domIff fadd-options.simps}(1) \text{ subset-iff[of } ?A\ ?B]$ 
   $\text{dom-map-add map-add-None}$ 
  by metis
qed

```

lemma *add-fh-asso:*

```

 $\text{add-fh } (\text{add-fh } a\ b)\ c = \text{add-fh } a\ (\text{add-fh } b\ c)$ 

```

```

proof (rule ext)
  fix x
  show add-fh (add-fh a b) c x = add-fh a (add-fh b c) x
  proof (cases a x)
    case None
    then show ?thesis
    by (simp add: add-fh-def)
  next
    case (Some aa)
    then have a x = Some aa by simp
    then show ?thesis
    proof (cases b x)
      case None
      then show ?thesis
      by (simp add: Some add-fh-def)
    next
      case (Some bb)
      then have b x = Some bb by simp
      then show ?thesis
      proof (cases c x)
        case None
        then show ?thesis
        by (simp add: Some ⟨a x = Some aa⟩ add-fh-def)
      next
        case (Some cc)
        then have add-fh (add-fh a b) c x = Some (padd (padd (fst aa) (fst bb))
(fst cc), snd aa)
        by (simp add: ⟨a x = Some aa⟩ ⟨b x = Some bb⟩ add-fh-def)
        moreover have add-fh a (add-fh b c) x = Some (padd (fst aa) (padd (fst
bb) (fst cc)), snd aa)
        by (simp add: Some ⟨a x = Some aa⟩ ⟨b x = Some bb⟩ add-fh-def)
        ultimately show ?thesis
        by (simp add: padd-asso)
      qed
    qed
  qed
qed
qed

lemma add-fh-update:
  assumes b x = None
  shows add-fh (a(x ↦ p)) b = (add-fh a b)(x ↦ p)
proof (rule ext)
  fix l show add-fh (a(x ↦ p)) b l = ((add-fh a b)(x ↦ p)) l
  apply (cases l = x)
  apply (simp add: add-fh-def assms)
  by (simp add: add-fh-def)
qed

end

```

1.4 Extended Heaps

```
theory StateModel
  imports FractionalHeap HOL-Library.Multiset
begin
```

```
type-synonym loc = nat
type-synonym val = nat
```

We store the initial value with the unique guard

```
type-synonym f-heap = (loc, val) fract-heap
type-synonym 'a gs-heap = (prat × 'a multiset) option
type-synonym ('i, 'a) gu-heap = 'i ⇒ 'a list option
```

```
type-synonym ('i, 'a) heap = f-heap × 'a gs-heap × ('i, 'a) gu-heap
```

```
type-synonym var = string
type-synonym normal-heap = (nat → nat)
type-synonym store = (var ⇒ nat)
```

```
fun get-fh where get-fh x = fst x
fun get-gs where get-gs x = fst (snd x)
fun get-gu where get-gu x = snd (snd x)
```

Two "heaps" are compatible iff: 1. The fractional heaps have the same common values and sum to at most 1 2. The unique guard heaps are disjoint 3. The shared guards permissions sum to at most 1

```
definition compatible :: ('i, 'a) heap ⇒ ('i, 'a) heap ⇒ bool (infixl ## 60) where
  h ## h' ↔ compatible-fract-heaps (get-fh h) (get-fh h') ∧ (∀ k. get-gu h k =
  None ∨ get-gu h' k = None)
  ∧ (∀ p p'. get-gs h = Some p ∧ get-gs h' = Some p' → pgte pwrite (padd (fst p)
  (fst p'))))
```

lemma *compatibleI*:

```
  assumes compatible-fract-heaps (get-fh h) (get-fh h')
    and ∧k. get-gu h k = None ∨ get-gu h' k = None
    and ∧p p'. get-gs h = Some p ∧ get-gs h' = Some p' ⇒ pgte pwrite (padd
  (fst p) (fst p'))
  shows h ## h'
  using assms(1) assms(2) assms(3) compatible-def by blast
```

```
fun add-gu-single where
  add-gu-single None x = x
| add-gu-single x None = x
```

```
definition add-gu where
  add-gu u1 u2 k = add-gu-single (u1 k) (u2 k)
```

```

lemma comp-add-gu-comm:
  assumes  $\bigwedge k. h\ k = \text{None} \vee h'\ k = \text{None}$ 
  shows  $\text{add-gu } h\ h' = \text{add-gu } h'\ h$ 
proof (rule ext)
  fix  $k$  show  $\text{add-gu } h\ h'\ k = \text{add-gu } h'\ h\ k$ 
  by (metis add-gu-def add-gu-single.simps(1) add-gu-single.simps(2) assms not-None-eq)
qed

```

```

fun add-gs ::  $(\text{pratt} \times 'a \text{ multiset}) \text{ option} \Rightarrow (\text{pratt} \times 'a \text{ multiset}) \text{ option} \Rightarrow (\text{pratt} \times 'a \text{ multiset}) \text{ option}$ 
where
   $\text{add-gs } \text{None } x = x$ 
|  $\text{add-gs } x\ \text{None} = x$ 
|  $\text{add-gs } (\text{Some } p)\ (\text{Some } p') = \text{Some } (\text{padd } (\text{fst } p)\ (\text{fst } p'), \text{snd } p + \text{snd } p')$ 

```

```

lemma add-gs-cancellative:
  assumes  $\text{add-gs } a\ x = \text{add-gs } b\ x$ 
  shows  $a = b$ 
  apply (cases x)
  apply (metis add-gs.elims assms not-None-eq)
  apply (cases a)
  apply (cases b)
  apply simp
  apply (metis add-gs.simps(1) add-gs.simps(3) assms fst-conv not-pgte-character option.sel padd-comm pgt-implies-pgte sum-larger)
  apply (cases b)
  apply (metis add-gs.simps(1) add-gs.simps(3) assms fst-conv not-pgte-character option.sel padd-comm pgt-implies-pgte sum-larger)
proof –
  fix  $xx\ aa\ bb$  assume  $x = \text{Some } xx\ a = \text{Some } aa\ b = \text{Some } bb$ 
  then have  $\text{fst } aa = \text{fst } bb$ 
  using assms padd-cancellative[of padd (fst aa) (fst xx)]
  Pair-inject add-gs.simps(3) option.inject by auto
  moreover have  $\text{snd } aa = \text{snd } bb$ 
  using add-left-cancel[of snd xx snd aa snd bb]
  using  $\langle a = \text{Some } aa \rangle \langle b = \text{Some } bb \rangle \langle x = \text{Some } xx \rangle$  assms by auto
  ultimately show  $a = b$ 
  by (simp add:  $\langle a = \text{Some } aa \rangle \langle b = \text{Some } bb \rangle \text{prod-eq-iff}$ )
qed

```

```

lemma add-gs-comm:
   $\text{add-gs } a\ b = \text{add-gs } b\ a$ 
proof (cases a)
  case None
  then show ?thesis
  by (metis add-gs.elims add-gs.simps(1) add-gs.simps(2))
next
  case (Some aa)

```



```

then have  $a = \text{Some } aa$  by simp
then show ?thesis
proof (cases b)
  case None
    then show ?thesis
    using Some by force
  next
    case (Some bb)
    moreover have  $\text{padd } (fst\ aa) (fst\ bb) = \text{padd } (fst\ bb) (fst\ aa) \wedge \text{snd } aa + \text{snd } bb = \text{snd } bb + \text{snd } aa$ 
    using padd-comm by force
    ultimately show ?thesis
    using  $\langle a = \text{Some } aa \rangle$  by force
qed
qed

```

```

lemma compatible-fheaps-comm:
  assumes compatible-fract-heaps a b
  shows  $\text{add-fh } a\ b = \text{add-fh } b\ a$ 
proof (rule ext)
  fix  $x$  show  $\text{add-fh } a\ b\ x = \text{add-fh } b\ a\ x$ 
  proof (cases a x)
    case None
      then show ?thesis
      by (metis add-fh-def add-fh-def fadd-options.simps(1) fadd-options.simps(2) option.exhaust-sel)
    next
      case (Some aa)
      then have  $a\ x = \text{Some } aa$  by simp
      then show ?thesis
      proof (cases b x)
        case None
          then show ?thesis
          by (simp add: Some add-fh-def)
        next
          case (Some bb)
          then show ?thesis
          using  $\langle a\ x = \text{Some } aa \rangle$  add-fh-def[of a b] add-fh-def[of b a] assms compatible-fract-heapsE(2) fadd-options.simps(3) padd-comm
          by (metis (full-types))
      qed
    qed
  qed

```

```

fun plus :: ('i, 'a) heap option  $\Rightarrow$  ('i, 'a) heap option  $\Rightarrow$  ('i, 'a) heap option (infixl
 $\oplus$  63) where
  None  $\oplus$  - = None
| -  $\oplus$  None = None
| Some h1  $\oplus$  Some h2 = (if h1 ## h2 then Some (add-fh (get-fh h1) (get-fh h2)),

```

$add\text{-}gs\ (get\text{-}gs\ h1)\ (get\text{-}gs\ h2),\ add\text{-}gu\ (get\text{-}gu\ h1)\ (get\text{-}gu\ h2))\ else\ None)$

lemma *plus-extract*:

assumes $Some\ x = Some\ a \oplus Some\ b$
shows $get\text{-}fh\ x = add\text{-}fh\ (get\text{-}fh\ a)\ (get\text{-}fh\ b)$
and $get\text{-}gs\ x = add\text{-}gs\ (get\text{-}gs\ a)\ (get\text{-}gs\ b)$
and $get\text{-}gu\ x = add\text{-}gu\ (get\text{-}gu\ a)\ (get\text{-}gu\ b)$
apply (*metis* *assms* *eqfst-iff* *get-fh.simps* *option.inject* *option.simps*(3) *plus.simps*(3))
apply (*metis* *assms* *fst-eqD* *get-gs.simps* *option.distinct*(1) *option.inject* *plus.simps*(3)
snd-conv)
by (*metis* *assms* *get-gu.elims* *option.distinct*(1) *option.sel* *plus.simps*(3) *snd-conv*)

lemma *compatible-eq*:

$Some\ a \oplus Some\ b = None \longleftrightarrow \neg a \#\# b$
by *simp*

lemma *compatible-comm*:

$a \#\# b \longleftrightarrow b \#\# a$
proof –
have $\bigwedge a\ b.\ a \#\# b \implies b \#\# a$
proof –
fix $a\ b$ **assume** *asm0*: $a \#\# b$
show $b \#\# a$
proof (*rule compatibleI*)
show *compatible-fract-heaps* (*get-fh* b) (*get-fh* a)
using *asm0* *compatible-def* *compatible-fract-heaps-comm* **by** *blast*
show $\bigwedge k.\ get\text{-}gu\ b\ k = None \vee get\text{-}gu\ a\ k = None$
by (*meson* *asm0* *compatible-def*)
show $\bigwedge p\ p'.\ get\text{-}gs\ b = Some\ p \wedge get\text{-}gs\ a = Some\ p' \implies pgte\ pwrite\ (padd\ (fst\ p)\ (fst\ p'))$
by (*metis* *asm0* *compatible-def* *padd-comm*)
qed
qed
then show *?thesis*
by *blast*
qed

lemma *heap-ext*:

assumes $get\text{-}fh\ a = get\text{-}fh\ b$
and $get\text{-}gs\ a = get\text{-}gs\ b$
and $get\text{-}gu\ a = get\text{-}gu\ b$
shows $a = b$
by (*metis* *assms*(1) *assms*(2) *assms*(3) *get-fh.simps* *get-gs.simps* *get-gu.elims* *prod.expand*)

lemma *plus-comm*:

$a \oplus b = b \oplus a$
proof –
have $r:\ \bigwedge x\ a\ b.\ Some\ x = Some\ a \oplus Some\ b \implies Some\ x = Some\ b \oplus Some\ a$

```

proof –
  fix  $x\ a\ b$  assume  $asm0: \text{Some } x = \text{Some } a \oplus \text{Some } b$ 
  then obtain  $y$  where  $\text{Some } y = \text{Some } b \oplus \text{Some } a$ 
    by ( $metis\ compatible-comm\ plus.simps(3)$ )
  have  $x = y$ 
  proof ( $rule\ heap-ext$ )
    show  $get-fh\ x = get-fh\ y$ 
      by ( $metis\ \langle \text{Some } y = \text{Some } b \oplus \text{Some } a \rangle\ asm0\ compatible-def\ compatible-eq\ compatible-fheaps-comm\ plus-extract(1)$ )
    show  $get-gs\ x = get-gs\ y$ 
      by ( $metis\ \langle \text{Some } y = \text{Some } b \oplus \text{Some } a \rangle\ add-gs-comm\ asm0\ plus-extract(2)$ )
    show  $get-gu\ x = get-gu\ y$  using  $comp-add-gu-comm[of\ get-gu\ x\ get-gu\ y]$ 
      by ( $metis\ \langle \text{Some } y = \text{Some } b \oplus \text{Some } a \rangle\ asm0\ comp-add-gu-comm\ compatible-def\ compatible-eq\ plus-extract(3)$ )
    qed
  then show  $\text{Some } x = \text{Some } b \oplus \text{Some } a$ 
    by ( $simp\ add: \langle \text{Some } y = \text{Some } b \oplus \text{Some } a \rangle$ )
  qed
then show  $?thesis$ 
proof ( $cases\ a \oplus b$ )
  case  $None$ 
    then show  $?thesis$ 
      by ( $metis\ (no-types,\ opaque-lifting)\ compatible-comm\ compatible-eq\ plus.elims$ )
  next
  case ( $\text{Some } ab$ )
  then have  $a = \text{Some } (the\ a) \wedge b = \text{Some } (the\ b)$ 
    by ( $metis\ option.collapse\ option.distinct(1)\ plus.simps(1)\ plus.simps(2)$ )
  then show  $?thesis$ 
    by ( $metis\ \langle \bigwedge x\ b\ a.\ \text{Some } x = \text{Some } a \oplus \text{Some } b \implies \text{Some } x = \text{Some } b \oplus \text{Some } a \rangle\ plus.elims$ )
  qed
qed

```

lemma $asso2$:

```

assumes  $\text{Some } a \oplus \text{Some } b = \text{Some } ab$ 
and  $\neg b \#\# c$ 
shows  $\neg ab \#\# c$ 
proof ( $cases\ compatible-fract-heaps\ (get-fh\ b)\ (get-fh\ c)$ )
case  $True$ 
then have  $(\exists k.\ get-gu\ b\ k \neq None \wedge get-gu\ c\ k \neq None)$ 
 $\vee (\exists p\ p'. get-gs\ b = \text{Some } p \wedge get-gs\ c = \text{Some } p' \wedge pgt\ (padd\ (fst\ p)\ (fst\ p'))$ 
 $pwrite)$ 
by ( $metis\ assms(2)\ compatible-def\ not-pgte-charact$ )
then show  $?thesis$ 
proof ( $cases\ \exists k.\ get-gu\ b\ k \neq None \wedge get-gu\ c\ k \neq None$ )
case  $True$ 
then obtain  $k$  where  $get-gu\ b\ k \neq None \wedge get-gu\ c\ k \neq None$ 
by  $auto$ 

```

```

then have get-gu ab k ≠ None
  using add-gu-def[of get-gu a get-gu b] add-gu-single.simps(1) assms(1) compatible-def compatible-eq option.distinct(1) plus-extract(3)
  by metis
  then show ?thesis
  by (meson ⟨get-gu b k ≠ None ∧ get-gu c k ≠ None⟩ compatible-def)
next
  case False
  then obtain p p' where get-gs b = Some p ∧ get-gs c = Some p' ∧ pgt (padd (fst p) (fst p')) pwrite
  using ⟨(∃ k. get-gu b k ≠ None ∧ get-gu c k ≠ None) ∨ (∃ p p'. get-gs b = Some p ∧ get-gs c = Some p' ∧ pgt (padd (fst p) (fst p')) pwrite)⟩ by blast
  moreover have get-gs ab = add-gs (get-gs a) (Some p)
  by (metis assms(1) calculation plus-extract(2))
  then show ?thesis
  proof (cases get-gs a)
  case None
  then show ?thesis
  by (metis ⟨get-gs ab = add-gs (get-gs a) (Some p)⟩ add-gs.simps(1) calculation compatible-def not-pgte-charact)
  next
  case (Some pa)
  then have get-gs ab = Some (padd (fst pa) (fst p), snd pa + snd p)
  using ⟨get-gs ab = add-gs (get-gs a) (Some p)⟩ by auto
  then have pgte (padd (fst pa) (fst p)) (fst p)
  using padd-comm pgt-implies-pgte sum-larger by presburger
  then have pgt (padd (padd (fst pa) (fst p)) (fst p')) pwrite
  using calculation padd.rep-eq pgt.rep-eq pgte.rep-eq by auto
  then show ?thesis
  by (metis ⟨get-gs ab = Some (padd (fst pa) (fst p), snd pa + snd p)⟩ calculation compatible-def fst-conv not-pgte-charact)
  qed
  qed
next
  case False
  then show ?thesis
  proof (cases compatible-fractions (get-fh b) (get-fh c))
  case True
  then have ¬ same-values (get-fh b) (get-fh c)
  using False compatible-fract-heaps-def by blast
  then obtain l pb pc where get-fh b l = Some pb get-fh c l = Some pc snd pc ≠ snd pb
  using same-values-def by fastforce
  then obtain pab where get-fh ab l = Some pab snd pab = snd pb
  apply (cases get-fh a l)
  apply (metis (no-types, lifting) add-fh-def assms(1) fadd-options.simps(2) plus-comm plus-extract(1))
  using add-fh-def[of get-fh b get-fh a l] assms(1) fadd-options.simps(3) plus-comm plus-extract(1) snd-conv

```

```

    by metis
  then show ?thesis
    by (metis ⟨get-fh c l = Some pc⟩ ⟨snd pc ≠ snd pb⟩ compatible-def compatible-
    fract-heapsE(2))
  next
    case False
    then obtain pb pc l where get-fh b l = Some pb get-fh c l = Some pc pgt
    (padd (fst pb) (fst pc)) pwrite
      using compatible-fractions-def not-pgte-charact by blast

  then show ?thesis
  proof (cases get-fh a l)
    case None
    then have get-fh ab l = Some pb
      by (metis (no-types, lifting) ⟨get-fh b l = Some pb⟩ add-fh-def assms(1)
      fadd-options.simps(1) plus-extract(1))
    then show ?thesis
      by (meson ⟨get-fh c l = Some pc⟩ ⟨pgt (padd (fst pb) (fst pc)) pwrite⟩
      compatible-def compatible-fract-heaps-def compatible-fractions-def not-pgte-charact)
    next
      case (Some pa)
      then obtain pab where get-fh ab l = Some pab fst pab = padd (fst pa) (fst
      pb)
        by (metis (mono-tags, opaque-lifting) ⟨get-fh b l = Some pb⟩ add-fh-def
        assms(1) fadd-options.simps(3) fst-conv plus-extract(1))
      then have pgte (fst pab) (fst pb)
        by (metis padd-comm pgt-implies-pgte sum-larger)
      then have pgt (padd (fst pab) (fst pc)) pwrite
        using ⟨pgt (padd (fst pb) (fst pc)) pwrite⟩ padd.rep-eq pgt.rep-eq pgte.rep-eq
      by force
      then show ?thesis
        by (meson ⟨get-fh ab l = Some pab⟩ ⟨get-fh c l = Some pc⟩ compatible-def
        compatible-fract-heapsE(1) not-pgte-charact)
    qed
  qed
qed

```

lemma *plus-extract-point-fh*:

```

  assumes Some x = Some a ⊕ Some b
    and get-fh a l = Some pa
    and get-fh b l = Some pb
  shows snd pa = snd pb ∧ pgte pwrite (padd (fst pa) (fst pb)) ∧ get-fh x l =
  Some (padd (fst pa) (fst pb), snd pa)
  using add-fh-def[of get-fh a get-fh b] assms(1) assms(2) assms(3) compati-
  ble-def[of a b] compatible-eq
    compatible-fract-heapsE(1)[of get-fh a get-fh b] compatible-fract-heapsE(2)[of
  get-fh a get-fh b]
    fadd-options.simps(3)[of pa pb] option.distinct(1) plus-extract(1)[of x a b]
  by metis

```

lemma *asso1*:

assumes $\text{Some } a \oplus \text{Some } b = \text{Some } ab$
and $\text{Some } b \oplus \text{Some } c = \text{Some } bc$
shows $\text{Some } ab \oplus \text{Some } c = \text{Some } a \oplus \text{Some } bc$

proof (cases $\text{Some } ab \oplus \text{Some } c$)
case *None*
then show *?thesis*
proof (cases *compatible-fract-heaps* (*get-fh* *ab*) (*get-fh* *c*))
case *True*
then have $r: (\exists k. \text{get-gu } ab \ k \neq \text{None} \wedge \text{get-gu } c \ k \neq \text{None}) \vee (\exists p \ p'. \text{get-gs } ab = \text{Some } p \wedge \text{get-gs } c = \text{Some } p' \wedge \text{pgt } (\text{padd } (\text{fst } p) (\text{fst } p')) \text{ pwrite})$
by (*metis* *None compatible-def compatible-eq not-pgte-charact*)
then show *?thesis*
proof (cases $\exists k. \text{get-gu } ab \ k \neq \text{None} \wedge \text{get-gu } c \ k \neq \text{None}$)
case *True*
then obtain k **where** $\text{get-gu } ab \ k \neq \text{None} \wedge \text{get-gu } c \ k \neq \text{None}$
by *presburger*
then have $\text{get-gu } a \ k \neq \text{None} \vee \text{get-gu } b \ k \neq \text{None}$
by (*metis* (*no-types, lifting*) *add-gu-def add-gu-single.simps(1) assms(1) plus-extract(3)*)
then show *?thesis*
by (*metis* $\langle \text{get-gu } ab \ k \neq \text{None} \wedge \text{get-gu } c \ k \neq \text{None} \rangle$ *assms(2) asso2 compatible-def compatible-eq option.discI plus-comm*)
next
case *False*
then obtain $pab \ pc$ **where** $\text{get-gs } ab = \text{Some } pab \wedge \text{get-gs } c = \text{Some } pc \wedge \text{pgt } (\text{padd } (\text{fst } pab) (\text{fst } pc)) \text{ pwrite}$
using r **by** *blast*
then show *?thesis*
apply (cases *get-gs* a)
apply (*metis* *add-gs.simps(1) assms(1) assms(2) compatible-def compatible-eq not-pgte-charact option.discI plus-extract(2)*)
apply (cases *get-gs* b)
apply (*metis* *add-gs.simps(1) add-gs.simps(2) assms(1) assms(2) compatible-def compatible-eq not-pgte-charact plus-extract(2)*)
proof –
fix $pa \ pb$ **assume** $\text{get-gs } ab = \text{Some } pab \wedge \text{get-gs } c = \text{Some } pc \wedge \text{pgt } (\text{padd } (\text{fst } pab) (\text{fst } pc)) \text{ pwrite}$
 $\text{get-gs } a = \text{Some } pa \ \text{get-gs } b = \text{Some } pb$
then have $pab = (\text{padd } (\text{fst } pa) (\text{fst } pb), \text{snd } pa + \text{snd } pb)$
by (*metis* *add-gs.simps(3) assms(1) option.sel plus-extract(2)*)
then show $\text{Some } ab \oplus \text{Some } c = \text{Some } a \oplus \text{Some } bc$
using $\text{None } \langle \text{get-gs } a = \text{Some } pa \rangle \langle \text{get-gs } ab = \text{Some } pab \wedge \text{get-gs } c = \text{Some } pc \wedge \text{pgt } (\text{padd } (\text{fst } pab) (\text{fst } pc)) \text{ pwrite} \rangle \langle \text{get-gs } b = \text{Some } pb \rangle$ *add-gs.simps(3) assms(2) compatible-def[of a bc] compatible-eq fst-conv not-pgte-charact[of pwrite padd (fst pab) (fst pc)] padd-asso plus-extract(2)*

```

      by metis
    qed
  qed
next
  case False
  then show ?thesis
  proof (cases compatible-fractions (get-fh ab) (get-fh c))
    case True
    then have  $\neg$ same-values (get-fh ab) (get-fh c)
      using False compatible-fract-heaps-def
      by blast

    then obtain l pab pc where get-fh ab l = Some pab get-fh c l = Some pc snd
      pab  $\neq$  snd pc
      using same-values-def by blast

    then show ?thesis
      apply (cases get-fh a l)

      apply (metis (no-types, lifting) add-fh-def assms(1) assms(2) compatible-def
        compatible-eq compatible-fract-heapsE(2) fadd-options.simps(1) option.distinct(1)
        plus-extract(1))

      proof -
        fix pa assume get-fh ab l = Some pab get-fh c l = Some pc snd pab  $\neq$  snd
          pc get-fh a l = Some pa
          moreover have same-values (get-fh a) (get-fh b)
            by (metis assms(1) compatible-def compatible-fract-heaps-def option.discI
              plus.simps(3))
          ultimately have snd pa = snd pab
            apply (cases get-fh b l)
            apply (metis (no-types, lifting) add-fh-def assms(1) fadd-options.simps(2)
              option.inject plus-extract(1))
            by (metis (no-types, lifting) add-fh-def assms(1) fadd-options.simps(3)
              option.sel plus-extract(1) snd-eqD)
          then show ?thesis
            by (metis (full-types) None  $\langle$ get-fh a l = Some pa $\rangle$   $\langle$ get-fh c l =
              Some pc $\rangle$   $\langle$ snd pab  $\neq$  snd pc $\rangle$  assms(2) asso2 compatible-def compatible-eq com-
              patible-fract-heapsE(2) plus-comm)
          qed
      next
      case False
      then obtain l pab pc where get-fh ab l = Some pab get-fh c l = Some pc pgt
        (padd (fst pab) (fst pc)) pwrite
        using compatible-fractions-def not-pgte-charact by blast
      then show ?thesis
      proof (cases get-fh a l)
        case None
        then have get-fh b l = Some pab

```

```

      by (metis (no-types, lifting) ⟨get-fh ab l = Some pab⟩ add-fh-def assms(1)
fadd-options.simps(1) plus-extract(1))
    then show ?thesis
      by (metis ⟨get-fh c l = Some pc⟩ ⟨pgt (padd (fst pab) (fst pc)) pwrite⟩
assms(2) compatible-def compatible-fract-heapsE(1) not-pgte-charact option.simps(3)
plus.simps(3))
    next
      case (Some pa)
      then have get-fh a l = Some pa by simp
      then show ?thesis
      proof (cases get-fh b l)
        case None
        then have pa = pab
          by (metis (no-types, lifting) Some ⟨get-fh ab l = Some pab⟩ add-fh-def
assms(1) fadd-options.simps(2) option.inject plus-extract(1))
        then show ?thesis
          by (metis Some ⟨get-fh ab l = Some pab⟩ ⟨get-fh c l = Some pc⟩ ⟨pgt (padd
(fst pab) (fst pc)) pwrite⟩ assms(2) asso2 compatible-def compatible-fract-heapsE(1)
not-pgte-charact padd-comm plus.simps(3) plus-comm)
      next
        case (Some pb)
        then have fst pab = padd (fst pa) (fst pb)
          using ⟨get-fh a l = Some pa⟩ ⟨get-fh ab l = Some pab⟩ add-fh-def[of
get-fh a get-fh b] assms(1) compatible-def compatible-eq
compatible-fract-heapsE(2)[of get-fh a get-fh b] fadd-options.simps(3)
fst-apfst option.discI option.sel plus-extract(1)[of ab a b] prod.collapse
snd-apfst
          by force
        then have pgt (padd (fst pa) (padd (fst pb) (fst pc))) pwrite
          using ⟨pgt (padd (fst pab) (fst pc)) pwrite⟩ padd-asso by auto
        moreover obtain pbc where get-fh bc l = Some pbc fst pbc = padd (fst
pb) (fst pc)
        by (metis (no-types, opaque-lifting) Some ⟨get-fh c l = Some pc⟩ add-fh-def
assms(2) fadd-options.simps(3) fst-conv plus-extract(1))
        ultimately show ?thesis
          by (metis None ⟨get-fh a l = Some pa⟩ compatible-def compatible-eq
compatible-fract-heapsE(1) not-pgte-charact)
      qed
    qed
  qed
next
case (Some x)
then have Some ab ⊕ Some c = Some x by simp
have a ## bc
proof (rule compatibleI)
show compatible-fract-heaps (get-fh a) (get-fh bc)
proof (rule compatible-fract-heapsI)
fix l pa pbc assume asm0: get-fh a l = Some pa ∧ get-fh bc l = Some pbc

```



```

have pgte pwrite (padd (fst pa) (fst pbc)) ∧ snd pa = snd pbc
proof (cases get-fh c l)
  case None
  then have get-fh b l = Some pbc
    by (metis (no-types, lifting) add-fh-def asm0 assms(2) fadd-options.elims
option.discI plus-extract(1))
  then show ?thesis
    by (metis (no-types, lifting) asm0 assms(1) compatible-def compatible-eq
compatible-fract-heapsE(1) compatible-fract-heapsE(2) option.discI)
  next
  case (Some pc)
  then have get-fh c l = Some pc by simp
  then show ?thesis
  proof (cases get-fh b l)
    case None
    then have get-fh ab l = Some pa
    by (metis (no-types, lifting) add-fh-def asm0 assms(1) fadd-options.simps(2)
plus-extract(1))
    moreover have pbc = pc
    by (metis (no-types, lifting) None Some add-fh-def asm0 assms(2)
fadd-options.simps(2) option.inject plus-comm plus-extract(1))
    ultimately show ?thesis
    by (metis (no-types, lifting) Some ⟨Some ab ⊕ Some c = Some x⟩ compatible-
def compatible-eq compatible-fract-heapsE(1) compatible-fract-heapsE(2) option.discI)
    next
    case (Some pb)
    then obtain pab where get-fh ab l = Some pab fst pab = padd (fst pa)
(fst pb) snd pab = snd pa
    by (metis (mono-tags, opaque-lifting) add-fh-def asm0 assms(1) fadd-options.simps(3)
fst-conv plus-extract(1) snd-conv)
    then have pgte pwrite (padd (padd (fst pa) (fst pb)) (fst pc))
    by (metis ⟨Some ab ⊕ Some c = Some x⟩ ⟨get-fh c l = Some pc⟩
compatible-def compatible-eq compatible-fract-heapsE(1) option.distinct(1))
    then have pgte pwrite (padd (fst pa) (fst pbc))
    by (metis (no-types, lifting) Some ⟨get-fh c l = Some pc⟩ add-fh-def asm0
assms(2) fadd-options.simps(3) fst-conv option.sel padd-asso plus-extract(1))
    moreover have snd pa = snd pb
    by (metis Some asm0 assms(1) compatible-def compatible-fract-heapsE(2)
option.simps(3) plus.simps(3))
    then have snd pa = snd pbc
    by (metis (no-types, opaque-lifting) Some ⟨get-fh c l = Some pc⟩ add-fh-def
asm0 assms(2) fadd-options.simps(3) option.sel plus-extract(1) snd-conv)
    ultimately show ?thesis by blast
  qed
qed
then show pgte pwrite (padd (fst pa) (fst pbc))
by auto
show snd pa = snd pbc
by (simp add: ⟨pgte pwrite (padd (fst pa) (fst pbc)) ∧ snd pa = snd pbc⟩)

```

```

qed

show  $\bigwedge k. \text{get-gu } a \ k = \text{None} \vee \text{get-gu } bc \ k = \text{None}$ 
proof -
  fix k show  $\text{get-gu } a \ k = \text{None} \vee \text{get-gu } bc \ k = \text{None}$ 
  proof (cases  $\text{get-gu } a \ k$ )
    case (Some aa)
      then have  $\text{get-gu } b \ k = \text{None} \vee \text{get-gu } c \ k = \text{None}$ 
        by (metis  $\text{assms}(2)$  compatible-def compatible-eq option.discI)
      then show ?thesis
        using Some  $\langle \text{Some } ab \oplus \text{Some } c = \text{Some } x \rangle$  add-gu-def[of  $\text{get-gu } a \ \text{get-gu } b$ ]
          add-gu-def[of  $\text{get-gu } b \ \text{get-gu } c$ ] add-gu-single.simps(1) add-gu-single.simps(2)
          assms(1) assms(2) compatible-def compatible-eq option.distinct(1)
        plus-extract(3)
        by metis
      qed (simp)
    qed
  fix pa pbc assume  $\text{get-gs } a = \text{Some } pa \wedge \text{get-gs } bc = \text{Some } pbc$ 
  show  $\text{pgte } pwrite \ (\text{padd } (\text{fst } pa) \ (\text{fst } pbc))$ 
  proof (cases  $\text{get-gs } b$ )
    case None
      then show ?thesis by (metis Some  $\langle \text{get-gs } a = \text{Some } pa \wedge \text{get-gs } bc = \text{Some } pbc \rangle$ 
        add-gs.simps(1) add-gs.simps(2) assms(1) assms(2) compatible-def compatible-eq
        option.discI plus-extract(2))
    next
      case (Some pb)
      then have  $\text{get-gs } b = \text{Some } pb$  by simp
      then show ?thesis
        proof (cases  $\text{get-gs } c$ )
          case None
            then show ?thesis
              by (metis Some  $\langle \text{get-gs } a = \text{Some } pa \wedge \text{get-gs } bc = \text{Some } pbc \rangle$  add-gs.simps(2)
                assms(1) assms(2) compatible-def compatible-eq option.distinct(1) plus-extract(2))
          next
            case (Some pc)
            then have  $\text{padd } (\text{fst } pa) \ (\text{fst } pbc) = \text{padd } (\text{fst } pa) \ (\text{padd } (\text{fst } pb) \ (\text{fst } pc))$ 
              by (metis (no-types, lifting)  $\langle \text{get-gs } a = \text{Some } pa \wedge \text{get-gs } bc = \text{Some } pbc \rangle$ 
                 $\langle \text{get-gs } b = \text{Some } pb \rangle$  add-gs.simps(3) assms(2) fst-conv option.sel plus-extract(2))
            also have  $\dots = \text{padd } (\text{padd } (\text{fst } pa) \ (\text{fst } pb)) \ (\text{fst } pc)$ 
              using padd-asso by force
            moreover obtain pab where  $\text{get-gs } ab = \text{Some } pab$ 
              by (metis  $\langle \text{get-gs } a = \text{Some } pa \wedge \text{get-gs } bc = \text{Some } pbc \rangle$   $\langle \text{get-gs } b = \text{Some } pb \rangle$ 
                add-gs.simps(3) assms(1) plus-extract(2))
            then have  $\text{pgte } pwrite \ (\text{padd } (\text{fst } pab) \ (\text{fst } pc))$ 
              by (metis Some  $\langle \text{Some } ab \oplus \text{Some } c = \text{Some } x \rangle$  compatible-def compatible-eq
                option.simps(3))
            ultimately show ?thesis
              by (metis (no-types, lifting)  $\langle \text{get-gs } a = \text{Some } pa \wedge \text{get-gs } bc = \text{Some } pbc \rangle$ 

```

```

⟨get-gs ab = Some pab⟩ ⟨get-gs b = Some pb⟩ add-gs.simps(3) assms(1) fst-conv
option.sel plus-extract(2))
  qed
  qed

qed
then obtain y where Some y = Some a ⊕ Some bc
  by simp
moreover have x = y
proof (rule heap-ext)
  show get-gu x = get-gu y
  proof (rule ext)
    fix k show get-gu x k = get-gu y k
    apply (cases get-gu a k)
    using Some add-gu-def[of get-gu a] add-gu-def[of get-gu b] add-gu-def[of
get-gu ab]
      add-gu-single.simps(1) assms(1) assms(2) calculation
      plus-extract(3)[of ab a b] plus-extract(3)[of bc b c] plus-extract(3)[of y a
bc] plus-extract(3)[of x ab c]
    apply simp
    apply (cases get-gu b k)
    using Some add-gu-def[of get-gu a] add-gu-def[of get-gu b] add-gu-def[of
get-gu ab]
      add-gu-single.simps(1) assms(1) assms(2) calculation
      plus-extract(3)[of ab a b] plus-extract(3)[of bc b c] plus-extract(3)[of y a
bc] plus-extract(3)[of x ab c]
      add-gu-single.simps(1) add-gu-single.simps(2) assms(1) assms(2) calcu-
lation
    apply simp
    by (metis assms(1) compatible-def compatible-eq option.simps(3))
  qed
show get-gs x = get-gs y
  apply (cases get-gs a)
  apply (metis (mono-tags, lifting) Some add-gs.simps(1) assms(1) assms(2)
calculation plus-extract(2))
  apply (cases get-gs b)
  apply (metis (mono-tags, lifting) Some add-gs.simps(1) add-gs.simps(2)
assms(1) assms(2) calculation plus-extract(2))
  apply (cases get-gs c)
  apply (metis Some add-gs.simps(1) assms(1) assms(2) calculation plus-comm
plus-extract(2))
proof -
  fix ga gb gc assume asm0: get-gs a = Some ga get-gs b = Some gb get-gs c
= Some gc
  then obtain gab gbc where r: get-gs ab = Some gab get-gs bc = gbc
  by (metis add-gs.simps(3) assms(1) plus-extract(2))
  then have get-gs x = Some (padd (padd (fst ga) (fst gb)) (fst gc), (snd ga +
snd gb) + snd gc)
  by (metis (no-types, lifting) Some add-gs.simps(3) asm0(1) asm0(2))

```

```

asm0(3) assms(1) fst-conv plus-extract(2) snd-conv)
  moreover have get-gs y = Some (padd (fst ga) (padd (fst gb) (fst gc)), snd
ga + (snd gb + snd gc))
    by (metis (mono-tags, opaque-lifting) ‹Some y = Some a  $\oplus$  Some bc›
add-gs.simps(3) asm0(1) asm0(2) asm0(3) assms(2) fst-conv plus-extract(2) prod.exhaust-sel
snd-conv)
  ultimately show get-gs x = get-gs y
    by (simp add: padd-asso)
  qed
  show get-fh x = get-fh y
    by (metis Some add-fh-asso assms(1) assms(2) calculation plus-extract(1))
  qed
  ultimately show ?thesis using Some by presburger
qed

```

```

lemma simpler-asso:
  (Some a  $\oplus$  Some b)  $\oplus$  Some c = Some a  $\oplus$  (Some b  $\oplus$  Some c)
proof (cases Some a  $\oplus$  Some b)
  case None
  then show ?thesis
    by (metis (no-types, opaque-lifting) asso2 compatible-eq option.exhaust plus.simps(1)
plus-comm)
  next
  case (Some ab)
  then have ab: Some ab = Some a  $\oplus$  Some b by simp
  then show ?thesis
  proof (cases Some b  $\oplus$  Some c)
    case None
    then show ?thesis
      by (metis Some asso2 compatible-eq plus.simps(2))
    next
    case (Some bc)
    then show ?thesis
      by (metis ab asso1)
  qed
qed

```

```

lemma plus-asso:
  (a  $\oplus$  b)  $\oplus$  c = a  $\oplus$  (b  $\oplus$  c)
proof (cases a)
  case (Some aa)
  then have aa: a = Some aa by simp
  then show ?thesis
  proof (cases b)
    case (Some bb)
    then have bb: b = Some bb by simp
    then show ?thesis
  proof (cases c)
    case None

```

```

    then show ?thesis
      by (simp add: plus-comm)
  next
    case (Some cc)
    then show ?thesis
      using aa bb simpler-asso by blast
  qed
qed (simp)
qed (simp)

```

definition *larger* :: $(i, 'a)$ heap \Rightarrow $(i, 'a)$ heap \Rightarrow bool (infixl \succeq 55) where
 $a \succeq b \iff (\exists c. \text{Some } a = \text{Some } b \oplus \text{Some } c)$

lemma *larger-trans*:

```

assumes  $a \succeq b$ 
  and  $b \succeq c$ 
shows  $a \succeq c$ 

```

proof –

```

obtain r1 where Some a = Some b  $\oplus$  Some r1
  using assms(1) larger-def by blast
moreover obtain r2 where Some b = Some c  $\oplus$  Some r2
  using assms(2) larger-def by blast
moreover obtain r where Some r = Some r1  $\oplus$  Some r2
  by (metis (no-types, opaque-lifting) calculation(1) calculation(2) not-Some-eq
plus.simps(1) plus-asso plus-comm)
ultimately show ?thesis
  by (metis larger-def plus-comm simpler-asso)
qed

```

lemma *comp-smaller*:

```

assumes  $a \#\# b$ 
  and Some b = Some c  $\oplus$  Some d
shows  $a \#\# c$ 
by (metis assms(1) assms(2) option.distinct(1) plus.simps(1) plus.simps(3)
plus-asso)

```

lemma *full-sguard-sum-same*:

```

assumes get-gs a = Some (pwrite, sargs)
  and Some h = Some a  $\oplus$  Some b
shows get-gs h = Some (pwrite, sargs)

```

proof (cases get-gs b)

case None

then show ?thesis

```

  by (metis add-gs.simps(2) assms(1) assms(2) fst-conv get-gs.elims option.sel
option.simps(3) plus.simps(3) snd-eqD)

```

next

case (Some a)

then show ?thesis

```

  by (metis assms(1) assms(2) compatible-def compatible-eq fst-eqD not-pgte-charact)

```

option.simps(3) sum-larger
qed

lemma *full-uguard-sum-same:*

assumes *get-gu a k = Some uargs*
and *Some h = Some a \oplus Some b*
shows *get-gu h k = Some uargs*
proof (*cases get-gu b k*)
case *None*
then show *?thesis*
by (*metis (no-types, lifting) add-gu-def add-gu-single.simps(2) assms(1) assms(2) plus-extract(3)*)
next
case (*Some a*)
then show *?thesis*
by (*metis assms(1) assms(2) compatible-def compatible-eq option.simps(3)*)
qed

lemma *smaller-more-compatible:*

assumes *a $\#\#$ b*
and *a \succeq c*
shows *c $\#\#$ b*
by (*meson assms(1) assms(2) comp-smaller compatible-comm larger-def*)

lemma *equiv-sum-get-fh:*

assumes *get-fh a = get-fh a'*
and *get-fh b = get-fh b'*
and *Some x = Some a \oplus Some b*
and *Some x' = Some a' \oplus Some b'*
shows *get-fh x = get-fh x'*
by (*metis assms(1) assms(2) assms(3) assms(4) fst-eqD get-fh.elims option.discI option.sel plus.simps(3)*)

lemma *addition-cancellative:*

assumes *Some a = Some b \oplus Some c*
and *Some a = Some b' \oplus Some c*
shows *b = b'*
proof (*rule heap-ext*)
show *get-gu b = get-gu b'*
proof (*rule ext*)
fix *k show* *get-gu b k = get-gu b' k*
apply (*cases get-gu a k*)
apply (*metis assms(1) assms(2) full-uguard-sum-same not-Some-eq*)
apply (*cases get-gu b k*)
using *add-gu-def[of get-gu b get-gu c]*
add-gu-single.simps(1)[of get-gu c k] assms(1) assms(2) compatible-def[of b
c] compatible-def[of b' c]
option.inject option.simps(3) plus.elims plus-extract(3)[of a b c]

```

    apply metis
  proof -
    fix ga gb assume get-gu a k = Some ga get-gu b k = Some gb
    then have get-gu c k = None
      by (metis assms(1) compatible-def compatible-eq option.simps(3))
    then show get-gu b k = get-gu b' k
      by (metis (no-types, opaque-lifting) add-gu-def add-gu-single.simps(1) assms(1)
        assms(2) plus-comm plus-extract(3))
    qed
  qed
  show get-gs b = get-gs b'
    by (metis add-gs-cancellative assms(1) assms(2) plus-extract(2))
  show get-fh b = get-fh b'
  proof (rule ext)
    fix l show get-fh b l = get-fh b' l
    proof (cases get-fh a l)
      case None
      then have get-fh b l = None
        by (metis (no-types, lifting) add-fh-def assms(1) fadd-options.elims option.distinct(1) plus-extract(1))
      then show ?thesis
        by (metis (no-types, opaque-lifting) None add-fh-def assms(2) fadd-options.elims option.distinct(1) plus-extract(1))
    next
      case (Some aa)
      then have get-fh a l = Some aa by simp
      then show ?thesis
        proof (cases get-fh c l)
          case None
          then show ?thesis
            by (metis (no-types, lifting) add-fh-def assms(1) assms(2) fadd-options.simps(1) plus-comm plus-extract(1))
        next
          case (Some cc)
          then have get-fh c l = Some cc by simp
          then show ?thesis using fadd-options-cancellative
            by (metis (no-types, opaque-lifting) add-fh-def assms(1) assms(2) plus-extract(1))
        qed
      qed
    qed
  qed

```

lemma *addition-cancellative3*:

```

  assumes Some x = Some a  $\oplus$  Some b  $\oplus$  Some c
  and Some x = Some a'  $\oplus$  Some b  $\oplus$  Some c
  shows a = a'

```

proof –

```

  obtain ab ab' where Some ab = Some a  $\oplus$  Some b Some ab' = Some a'  $\oplus$  Some
  b

```

by (*metis* *assms(1)* *assms(2)* *not-Some-eq plus.simps(1)*)
then have $ab = ab'$
by (*metis* *addition-cancellative* *assms(1)* *assms(2)*)
then show *?thesis*
using $\langle \text{Some } ab = \text{Some } a \oplus \text{Some } b \rangle \langle \text{Some } ab' = \text{Some } a' \oplus \text{Some } b \rangle$
addition-cancellative **by** *blast*
qed

lemma *larger3*:
assumes $\text{Some } x = \text{Some } a \oplus \text{Some } b \oplus \text{Some } c$
shows $x \succeq b$
proof –
obtain ab **where** $\text{Some } ab = \text{Some } a \oplus \text{Some } b$
by (*metis* *assms* *not-Some-eq plus.simps(1)*)
then show *?thesis*
by (*metis* (*no-types, opaque-lifting*) *assms* *larger-def* *larger-trans* *plus-comm*)
qed

lemma *add-get-fh*:
assumes $\text{Some } x = \text{Some } a \oplus \text{Some } b$
shows $\text{get-fh } x = \text{add-fh } (\text{get-fh } a) (\text{get-fh } b)$
by (*metis* *assms* *fst-conv* *get-fh.elims* *option.discI* *option.sel* *plus.simps(3)*)

lemma *sum-gs-one-none*:
assumes $\text{Some } x = \text{Some } a \oplus \text{Some } b$
and $\text{get-gs } b = \text{None}$
shows $\text{get-gs } x = \text{get-gs } a$
by (*metis* *add-gs.simps(1)* *assms(1)* *assms(2)* *plus-comm* *plus-extract(2)*)

lemma *sum-gs-one-some*:
assumes $\text{Some } x = \text{Some } a \oplus \text{Some } b$
and $\text{get-gs } a = \text{Some } (pa, ma)$
and $\text{get-gs } b = \text{Some } (pb, mb)$
shows $\text{get-gs } x = \text{Some } (\text{padd } pa \ pb, ma + mb)$
by (*metis* *add-gs.simps(3)* *assms(1)* *assms(2)* *assms(3)* *fst-conv* *plus-extract(2)* *snd-conv*)

definition *empty-heap* :: $('i, 'a)$ *heap* **where**
empty-heap = (*Map.empty*, *None*, $\lambda k. \text{None}$)

lemma *dom-normalize*:
 $\text{dom } h = \text{dom } (\text{normalize } h)$

by (meson FractionalHeap.normalize-eq(1) domIff subsetI subset-antisym)

lemma *sum-second-none-get-fh*:

assumes $\text{Some } x = \text{Some } a \oplus \text{Some } b$

and $\text{get-fh } b \ l = \text{None}$

shows $\text{get-fh } x \ l = \text{get-fh } a \ l$

proof (cases $\text{get-fh } a \ l$)

case *None*

then show ?thesis

by (metis (no-types, opaque-lifting) add-fh-def add-get-fh assms(1) assms(2) fadd-options.simps(1))

next

case (Some aa)

then show ?thesis

by (metis (no-types, lifting) add-fh-def add-get-fh assms(1) assms(2) fadd-options.simps(2))

qed

lemma *sum-first-none-get-fh*:

assumes $\text{Some } x = \text{Some } a \oplus \text{Some } b$

and $\text{get-fh } a \ l = \text{None}$

shows $\text{get-fh } x \ l = \text{get-fh } b \ l$

by (metis assms(1) assms(2) plus-comm sum-second-none-get-fh)

lemma *dom-sum-two*:

assumes $\text{Some } x = \text{Some } a \oplus \text{Some } b$

shows $\text{dom } (\text{get-fh } x) = \text{dom } (\text{get-fh } a) \cup \text{dom } (\text{get-fh } b)$

by (metis add-get-fh assms compatible-def compatible-dom-sum compatible-eq option.distinct(1))

lemma *dom-three-sum*:

assumes $\text{Some } x = \text{Some } a \oplus \text{Some } b \oplus \text{Some } c$

shows $\text{dom } (\text{get-fh } x) = \text{dom } (\text{get-fh } a) \cup \text{dom } (\text{get-fh } b) \cup \text{dom } (\text{get-fh } c)$

proof –

obtain *ab* where $\text{Some } ab = \text{Some } a \oplus \text{Some } b$

by (metis assms not-Some-eq plus.simps(1))

then have $\text{Some } x = \text{Some } ab \oplus \text{Some } c$

using *assms* by presburger

then have $\text{dom } (\text{get-fh } x) = \text{dom } (\text{get-fh } ab) \cup \text{dom } (\text{get-fh } c)$

by (meson dom-sum-two)

then show ?thesis

by (metis ⟨Some ab = Some a ⊕ Some b⟩ dom-sum-two)

qed

lemma *addition-smaller-domain*:

assumes $\text{Some } a = \text{Some } b \oplus \text{Some } c$

shows $\text{dom } (\text{get-fh } b) \subseteq \text{dom } (\text{get-fh } a)$

by (metis (no-types, opaque-lifting) Un-subset-iff assms dom-sum-two order-refl)

```

lemma one-value-sum-same:
  assumes Some x = Some a  $\oplus$  Some b
    and get-fh a l = Some ( $\pi$ , v)
    shows  $\exists \pi'. \text{get-fh } x \text{ l} = \text{Some } (\pi', v)$ 
    using assms(1) assms(2) not-Some-eq plus-extract-point-fh[of x a - l ( $\pi$ , v)]
snd-eqD sum-second-none-get-fh[of x a]
  by metis

lemma compatible-last-two:
  assumes Some x = Some a  $\oplus$  Some b  $\oplus$  Some c
  shows b  $\#\#$  c
  by (metis assms compatible-eq option.discI plus.simps(2) plus-asso)

lemma add-fh-map-empty:
  add-fh h Map.empty = h
proof (rule ext)
  fix x show add-fh h Map.empty x = h x
  by (metis add-fh-def fadd-options.simps(1) fadd-options.simps(2) not-None-eq)
qed

end

```

2 Imperative Concurrent Language

This file defines the syntax and semantics of a concurrent programming language, based on Viktor Vafeiadis' Isabelle soundness proof of CSL [2], and adapted to Isabelle 2016-1 by Qin Yu and James Brotherston (see <https://people.mpi-sws.org/viktor/cslsound/>).

```

theory Lang
imports Main StateModel
begin

```

2.1 Language Syntax and Semantics

```

type-synonym state = store  $\times$  normal-heap

```

```

datatype exp =
  Evar var
  | Enum nat
  | Eplus exp exp

```

```

datatype bexp =
  Beq exp exp
  | Band bexp bexp
  | Bnot bexp
  | Btrue

```

```

datatype cmd =

```

```

  Cskip
| Cassign var exp
| Cread var exp
| Cwrite exp exp
| Calloc var exp
| Cdispose exp
| Cseq cmd cmd
| Cpar cmd cmd
| Cif bexp cmd cmd
| Cwhile bexp cmd
| Catomic cmd

```

Arithmetic expressions (exp) consist of variables, constants, and arithmetic operations. Boolean expressions ($bexp$) consist of comparisons between arithmetic expressions. Commands (cmd) include the empty command, variable assignments, memory reads, writes, allocations and deallocations, sequential and parallel composition, conditionals, while loops, local variable declarations, and atomic statements.

2.1.1 Semantics of expressions

Denotational semantics for arithmetic and boolean expressions.

primrec

$edenot :: exp \Rightarrow store \Rightarrow nat$

where

```

  edenot (Evar v) s      = s v
| edenot (Enum n) s      = n
| edenot (Eplus e1 e2) s = edenot e1 s + edenot e2 s

```

primrec

$bdenot :: bexp \Rightarrow store \Rightarrow bool$

where

```

  bdenot (Beq e1 e2) s = (edenot e1 s = edenot e2 s)
| bdenot (Band b1 b2) s = (bdenot b1 s \wedge bdenot b2 s)
| bdenot (Bnot b) s     = (\neg bdenot b s)
| bdenot Btrue -       = True

```

2.1.2 Semantics of commands

We give a standard small-step operational semantics to commands with configurations being command-state pairs.

inductive

$red :: cmd \Rightarrow state \Rightarrow cmd \Rightarrow state \Rightarrow bool$

and $red\text{-}rtrans :: cmd \Rightarrow state \Rightarrow cmd \Rightarrow state \Rightarrow bool$

where

```

  red-Seq1[intro]: red (Cseq Cskip C) \sigma C \sigma
| red-Seq2[elim]: red C1 \sigma C1' \sigma' \Longrightarrow red (Cseq C1 C2) \sigma (Cseq C1' C2) \sigma'

```

$| \text{red-If1[intro]}: \text{bdenot } B \text{ (fst } \sigma) \implies \text{red } (\text{Cif } B \text{ } C1 \text{ } C2) \sigma \text{ } C1 \text{ } \sigma$
 $| \text{red-If2[intro]}: \neg \text{bdenot } B \text{ (fst } \sigma) \implies \text{red } (\text{Cif } B \text{ } C1 \text{ } C2) \sigma \text{ } C2 \text{ } \sigma$
 $| \text{red-Atomic[intro]}: \text{red-rtrans } C \sigma \text{ } Cskip \sigma' \implies \text{red } (\text{Catomic } C) \sigma \text{ } Cskip \sigma'$
 $| \text{red-Par1[elim]}: \text{red } C1 \sigma \text{ } C1' \sigma' \implies \text{red } (\text{Cpar } C1 \text{ } C2) \sigma \text{ } (\text{Cpar } C1' \text{ } C2) \sigma'$
 $| \text{red-Par2[elim]}: \text{red } C2 \sigma \text{ } C2' \sigma' \implies \text{red } (\text{Cpar } C1 \text{ } C2) \sigma \text{ } (\text{Cpar } C1 \text{ } C2') \sigma'$
 $| \text{red-Par3[intro]}: \text{red } (\text{Cpar } Cskip \text{ } Cskip) \sigma \text{ } (Cskip) \sigma$
 $| \text{red-Loop[intro]}: \text{red } (\text{Cwhile } B \text{ } C) \sigma \text{ } (\text{Cif } B \text{ } (\text{Cseq } C \text{ } (\text{Cwhile } B \text{ } C)) \text{ } Cskip) \sigma$
 $| \text{red-Assign[intro]}: \llbracket \sigma = (s, h); \sigma' = (s(x := \text{edenot } E \text{ } s), h) \rrbracket \implies \text{red } (\text{Cassign } x \text{ } E) \sigma \text{ } Cskip \sigma'$
 $| \text{red-Read[intro]}: \llbracket \sigma = (s, h); h(\text{edenot } E \text{ } s) = \text{Some } v; \sigma' = (s(x := v), h) \rrbracket \implies \text{red } (\text{Cread } x \text{ } E) \sigma \text{ } Cskip \sigma'$
 $| \text{red-Write[intro]}: \llbracket \sigma = (s, h); \sigma' = (s, h(\text{edenot } E \text{ } s \mapsto \text{edenot } E' \text{ } s)) \rrbracket \implies \text{red } (\text{Cwrite } E \text{ } E') \sigma \text{ } Cskip \sigma'$
 $| \text{red-Alloc[intro]}: \llbracket \sigma = (s, h); v \notin \text{dom } h; \sigma' = (s(x := v), h(v \mapsto \text{edenot } E \text{ } s)) \rrbracket \implies \text{red } (\text{Calloc } x \text{ } E) \sigma \text{ } Cskip \sigma'$
 $| \text{red-Free[intro]}: \llbracket \sigma = (s, h); \sigma' = (s, h(\text{edenot } E \text{ } s := \text{None})) \rrbracket \implies \text{red } (\text{Cdispose } E) \sigma \text{ } Cskip \sigma'$

$| \text{NoStep}: \text{red-rtrans } C \sigma \text{ } C \sigma$
 $| \text{OneMoreStep}: \llbracket \text{red } C \sigma \text{ } C' \sigma'; \text{red-rtrans } C' \sigma' \text{ } C'' \sigma'' \rrbracket \implies \text{red-rtrans } C \sigma \text{ } C'' \sigma''$

inductive-cases *red-par-cases*: $\text{red } (\text{Cpar } C1 \text{ } C2) \sigma \text{ } C' \sigma'$
inductive-cases *red-atomic-cases*: $\text{red } (\text{Catomic } C) \sigma \text{ } C' \sigma'$

2.1.3 Abort semantics

inductive

aborts :: *cmd* \Rightarrow *state* \Rightarrow *bool*

where

$| \text{aborts-Seq[intro]}: \text{aborts } C1 \sigma \implies \text{aborts } (\text{Cseq } C1 \text{ } C2) \sigma$
 $| \text{aborts-Atomic[intro]}: \llbracket \text{red-rtrans } C \sigma \text{ } C' \sigma'; \text{aborts } C' \sigma' \rrbracket \implies \text{aborts } (\text{Catomic } C) \sigma$
 $| \text{aborts-Par1[intro]}: \text{aborts } C1 \sigma \implies \text{aborts } (\text{Cpar } C1 \text{ } C2) \sigma$
 $| \text{aborts-Par2[intro]}: \text{aborts } C2 \sigma \implies \text{aborts } (\text{Cpar } C1 \text{ } C2) \sigma$
 $| \text{aborts-Read[intro]}: \text{edenot } E \text{ (fst } \sigma) \notin \text{dom } (\text{snd } \sigma) \implies \text{aborts } (\text{Cread } x \text{ } E) \sigma$
 $| \text{aborts-Write[intro]}: \text{edenot } E \text{ (fst } \sigma) \notin \text{dom } (\text{snd } \sigma) \implies \text{aborts } (\text{Cwrite } E \text{ } E') \sigma$
 $| \text{aborts-Free[intro]}: \text{edenot } E \text{ (fst } \sigma) \notin \text{dom } (\text{snd } \sigma) \implies \text{aborts } (\text{Cdispose } E) \sigma$

inductive-cases *abort-atomic-cases*: $\text{aborts } (\text{Catomic } C) \sigma$

2.2 Useful Definitions and Results

The free variables of expressions, boolean expressions, and commands are defined as expected:

primrec

fvE :: *exp* \Rightarrow *var set*

where

$fvE (Evar v) = \{v\}$
 $| fvE (Enum n) = \{\}$
 $| fvE (Eplus e1 e2) = (fvE e1 \cup fvE e2)$

primrec

$fvB :: bexp \Rightarrow var set$

where

$fvB (Beq e1 e2) = (fvE e1 \cup fvE e2)$
 $| fvB (Band b1 b2) = (fvB b1 \cup fvB b2)$
 $| fvB (Bnot b) = (fvB b)$
 $| fvB Btrue = \{\}$

primrec

$fvC :: cmd \Rightarrow var set$

where

$fvC (Cskip) = \{\}$
 $| fvC (Cassign v E) = (\{v\} \cup fvE E)$
 $| fvC (Cread v E) = (\{v\} \cup fvE E)$
 $| fvC (Cwrite E1 E2) = (fvE E1 \cup fvE E2)$
 $| fvC (Calloc v E) = (\{v\} \cup fvE E)$
 $| fvC (Cdispose E) = (fvE E)$
 $| fvC (Cseq C1 C2) = (fvC C1 \cup fvC C2)$
 $| fvC (Cpar C1 C2) = (fvC C1 \cup fvC C2)$
 $| fvC (Cif B C1 C2) = (fvB B \cup fvC C1 \cup fvC C2)$
 $| fvC (Cwhile B C) = (fvB B \cup fvC C)$
 $| fvC (Catomic C) = (fvC C)$

primrec

$wrC :: cmd \Rightarrow var set$

where

$wrC (Cskip) = \{\}$
 $| wrC (Cassign v E) = \{v\}$
 $| wrC (Cread v E) = \{v\}$
 $| wrC (Cwrite E1 E2) = \{\}$
 $| wrC (Calloc v E) = \{v\}$
 $| wrC (Cdispose E) = \{\}$
 $| wrC (Cseq C1 C2) = (wrC C1 \cup wrC C2)$
 $| wrC (Cpar C1 C2) = (wrC C1 \cup wrC C2)$
 $| wrC (Cif B C1 C2) = (wrC C1 \cup wrC C2)$
 $| wrC (Cwhile B C) = (wrC C)$
 $| wrC (Catomic C) = (wrC C)$

primrec

$subE :: var \Rightarrow exp \Rightarrow exp \Rightarrow exp$

where

$subE x E (Evar y) = (if x = y then E else Evar y)$
 $| subE x E (Enum n) = Enum n$
 $| subE x E (Eplus e1 e2) = Eplus (subE x E e1) (subE x E e2)$

primrec

$subB :: var \Rightarrow exp \Rightarrow bexp \Rightarrow bexp$

where

$subB\ x\ E\ (Beq\ e1\ e2) = Beq\ (subE\ x\ E\ e1)\ (subE\ x\ E\ e2)$
| $subB\ x\ E\ (Band\ b1\ b2) = Band\ (subB\ x\ E\ b1)\ (subB\ x\ E\ b2)$
| $subB\ x\ E\ (Bnot\ b) = Bnot\ (subB\ x\ E\ b)$
| $subB\ x\ E\ Btrue = Btrue$

Basic properties of substitutions:

lemma *subE-assign*:

$edenot\ (subE\ x\ E\ e)\ s = edenot\ e\ (s(x := edenot\ E\ s))$
by (*induct e, simp-all*)

lemma *subB-assign*:

$bdenot\ (subB\ x\ E\ b)\ s = bdenot\ b\ (s(x := edenot\ E\ s))$

proof (*induct b*)

case (*Beq x1 x2*)

then show *?case*

using *bdenot.simps(1) subB.simps(1) subE-assign* **by** *presburger*

qed (*simp-all*)

inductive-cases *red-skip-cases*: $red\ Cskip\ \sigma\ C'\ \sigma'$

inductive-cases *aborts-skip-cases*: $aborts\ Cskip\ \sigma$

lemma *skip-simps[simp]*:

$\neg\ red\ Cskip\ \sigma\ C'\ \sigma'$

$\neg\ aborts\ Cskip\ \sigma$

using *red-skip-cases* **apply** *blast*

using *aborts-skip-cases* **by** *blast*

definition

$agrees :: 'a\ set \Rightarrow ('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'b) \Rightarrow bool$

where

$agrees\ X\ s\ s' \equiv \forall x \in X. s\ x = s'\ x$

lemma *agrees-union*:

$agrees\ (A \cup B)\ s\ s' \longleftrightarrow agrees\ A\ s\ s' \wedge agrees\ B\ s\ s'$

by (*meson Un-iff agrees-def*)

Proposition 4.1: Properties of basic properties of *red*.

lemma *agreesI*:

assumes $\bigwedge x. x \in X \implies s\ x = s'\ x$

shows $agrees\ X\ s\ s'$

using *agrees-def assms* **by** *blast*

lemma *red-properties*:

$red\ C\ \sigma\ C'\ \sigma' \implies fvC\ C' \subseteq fvC\ C \wedge wrC\ C' \subseteq wrC\ C \wedge agrees\ (-\ wrC\ C)\ (fst\ \sigma')$
 $red\ rtrans\ C\ \sigma\ C'\ \sigma' \implies fvC\ C' \subseteq fvC\ C \wedge wrC\ C' \subseteq wrC\ C \wedge agrees\ (-\ wrC\ C)\ (fst\ \sigma')\ (fst\ \sigma)$
proof (*induct rule: red-red-rtrans.inducts*)
case (*OneMoreStep* $C\ \sigma\ C'\ \sigma'\ C''\ \sigma''$)
then have $fvC\ C'' \subseteq fvC\ C$
by *blast*
moreover have $wrC\ C'' \subseteq wrC\ C$
using *OneMoreStep.hyps(2) OneMoreStep.hyps(4)* **by** *blast*
moreover have $agrees\ (-\ wrC\ C)\ (fst\ \sigma'')\ (fst\ \sigma)$
proof (*rule agreesI*)
fix x **assume** $x \in -\ wrC\ C$
then have $x \in -\ wrC\ C' \wedge x \in -\ wrC\ C''$
using *OneMoreStep.hyps(2) OneMoreStep.hyps(4)* **by** *blast*
then show $fst\ \sigma''\ x = fst\ \sigma\ x$
by (*metis OneMoreStep.hyps(2) OneMoreStep.hyps(4) $\langle x \in -\ wrC\ C \rangle$*
agrees-def)
qed
ultimately show *?case* **by** *simp*
qed (*auto simp add: agrees-def*)

Proposition 4.2: Semantics does not depend on variables not free in the term

lemma *exp-agrees*: $agrees\ (fvE\ E)\ s\ s' \implies edenot\ E\ s = edenot\ E\ s'$
by (*simp add: agrees-def, induct E, auto*)

lemma *bexp-agrees*:

$agrees\ (fvB\ B)\ s\ s' \implies bdenot\ B\ s = bdenot\ B\ s'$
proof (*induct B*)
case (*Beq* $x1\ x2$)
then have $agrees\ (fvE\ x1)\ s\ s' \wedge agrees\ (fvE\ x2)\ s\ s'$
by (*simp add: agrees-def*)
then show *?case* **using** *exp-agrees*
by *force*
next
case (*Band* $B1\ B2$)
then show *?case*
by (*simp add: agrees-def*)
qed (*simp-all*)

lemma *red-not-in-fv-not-touched*:

$red\ C\ \sigma\ C'\ \sigma' \implies x \notin fvC\ C \implies fst\ \sigma\ x = fst\ \sigma'\ x$
 $red\ rtrans\ C\ \sigma\ C'\ \sigma' \implies x \notin fvC\ C \implies fst\ \sigma\ x = fst\ \sigma'\ x$
proof (*induct arbitrary: rule: red-red-rtrans.inducts*)
case (*OneMoreStep* $C\ \sigma\ C'\ \sigma'\ C''\ \sigma''$)
then show $fst\ \sigma\ x = fst\ \sigma''\ x$
by (*metis red-properties(1) subsetD*)
qed (*auto*)

lemma *agrees-update1*:
assumes *agrees* X s s'
shows *agrees* X $(s(x := v))$ $(s'(x := v))$
proof (*rule agreesI*)
fix y **show** $y \in X \implies (s(x := v)) y = (s'(x := v)) y$
apply (*cases* $y = x$)
apply *simp*
using *agrees-def* *assms* **by** *fastforce*
qed

lemma *agrees-update2*:
assumes *agrees* X s s'
and $x \notin X$
shows *agrees* X $(s(x := v))$ $(s'(x := v'))$
proof (*rule agreesI*)
fix y **show** $y \in X \implies (s(x := v)) y = (s'(x := v')) y$
apply (*cases* $y = x$)
using *assms*(2) **apply** *blast*
using *agrees-def* *assms*(1) **by** *fastforce*
qed

lemma *red-agrees-aux*:
 $red\ C\ \sigma\ C'\ \sigma' \implies (\forall s\ h.\ agrees\ X\ (fst\ \sigma)\ s \wedge snd\ \sigma = h \wedge fvC\ C \subseteq X \longrightarrow$
 $(\exists s'\ h'. red\ C\ (s,\ h)\ C'\ (s',\ h') \wedge agrees\ X\ (fst\ \sigma')\ s' \wedge snd\ \sigma' = h'))$
 $red\ rtrans\ C\ \sigma\ C'\ \sigma' \implies (\forall s\ h.\ agrees\ X\ (fst\ \sigma)\ s \wedge snd\ \sigma = h \wedge fvC\ C \subseteq X$
 \longrightarrow
 $(\exists s'\ h'. red\ rtrans\ C\ (s,\ h)\ C'\ (s',\ h') \wedge agrees\ X\ (fst\ \sigma')\ s' \wedge snd\ \sigma' = h'))$
proof (*induct* *rule: red-red-rtrans.inducts*)
case (*red-If1* B σ $C1$ $C2$)
then show *?case*
proof (*clarify*)
fix $X\ s\ h$
assume *asm0*: $bdnot\ B\ (fst\ \sigma)\ agrees\ X\ (fst\ \sigma)\ s\ fvC\ (Cif\ B\ C1\ C2) \subseteq X$
then have $bdnot\ B\ s$
using *Un-iff* *agrees-def*[*of* $X\ fst\ \sigma\ s$] *bexp-agrees* *fvC.simps*(9) *in-mono*
agrees-def[*of* $fvB\ B$]
by *fastforce*
then show $\exists s'\ h'. red\ (Cif\ B\ C1\ C2)\ (s,\ snd\ \sigma)\ C1\ (s',\ h') \wedge agrees\ X\ (fst\ \sigma)\ s' \wedge snd\ \sigma = h'$
by (*metis* *asm0*(2) *fst-eqD* *red-red-rtrans.red-If1*)
qed
next
case (*red-If2* B σ $C1$ $C2$)
then show *?case*
proof (*clarify*)
fix $X\ s\ h$
assume *asm0*: $\neg bdnot\ B\ (fst\ \sigma)\ agrees\ X\ (fst\ \sigma)\ s\ fvC\ (Cif\ B\ C1\ C2) \subseteq X$
then have $\neg bdnot\ B\ s$
using *Un-subset-iff* *agrees-def*[*of* X] *agrees-def*[*of* $fvB\ B$] *bexp-agrees* *fvC.simps*(9)

in-mono
by *metis*
then show $\exists s' h'. \text{red} (Cif B C1 C2) (s, \text{snd } \sigma) C2 (s', h') \wedge \text{agrees } X (\text{fst } \sigma) s' \wedge \text{snd } \sigma = h'$
by (*metis asm0(2) fst-eqD red-red-rtrans.red-If2*)
qed
next
case (*red-Assign* σ *ss hh* $\sigma' x E$)
then show *?case*
proof (*clarify*)
fix $X s h$
assume *asm0*: $\sigma' = (\text{ss}(x := \text{edenot } E \text{ ss}), \text{hh}) \sigma = (\text{ss}, \text{hh}) \text{agrees } X (\text{fst} (\text{ss}, \text{hh})) s \text{fvC} (Cassign x E) \subseteq X$
then have $\text{edenot } E s = \text{edenot } E \text{ ss}$
using *exp-agrees fst-conv fvC.simps(2)*
by (*metis (mono-tags, lifting) Un-subset-iff agrees-def in-mono*)
then have $\text{red} (Cassign x E) (\text{ss}, \text{snd} (s, h)) Cskip (\text{ss}(x := \text{edenot } E s), h)$
by force
moreover have $\text{agrees } X (\text{fst} (s(x := \text{edenot } E s), h)) (\text{ss}(x := \text{edenot } E s))$
proof (*rule agreesI*)
fix y **assume** $y \in X$
show $\text{fst} (s(x := \text{edenot } E s), h) y = (\text{ss}(x := \text{edenot } E s)) y$
apply (*cases x = y*)
apply *simp*
by (*metis* $\langle y \in X \rangle \text{agrees-def asm0(3) fstI fun-upd-other}$)
qed
ultimately show $\exists s' h'. \text{red} (Cassign x E) (s, \text{snd} (\text{ss}, \text{hh})) Cskip (s', h') \wedge \text{agrees } X (\text{fst} (\text{ss}(x := \text{edenot } E \text{ ss}), \text{hh})) s' \wedge \text{snd} (\text{ss}(x := \text{edenot } E \text{ ss}), \text{hh}) = h'$
using $\langle \text{edenot } E s = \text{edenot } E \text{ ss} \rangle$
by (*metis agrees-update1 asm0(3) fst-conv red-red-rtrans.red-Assign snd-conv*)
qed
next
case (*red-Read* σ *ss hh* $E v \sigma' x$)
have $\bigwedge s h. \text{agrees } X (\text{fst } \sigma) s \wedge \text{snd } \sigma = h \wedge \text{fvC} (Cread x E) \subseteq X \implies (\exists s' h'. \text{red} (Cread x E) (s, h) Cskip (s', h') \wedge \text{agrees } X (\text{fst } \sigma') s' \wedge \text{snd } \sigma' = h')$
proof –
fix $s h$ **assume** *asm0*: $\text{agrees } X (\text{fst } \sigma) s \wedge \text{snd } \sigma = h \wedge \text{fvC} (Cread x E) \subseteq X$
then have $\text{hh} (\text{edenot } E s) = \text{Some } v$
using *red-Read(1) red-Read(2) exp-agrees fstI fvC.simps(3) Un-subset-iff agrees-def[of fvE E] in-mono agrees-def[of X]* **by** *metis*
then have $\text{agrees } X (\text{fst } \sigma') (s(x := v))$
by (*metis asm0(1) agrees-update1 fstI red-Read.hyps(1) red-Read.hyps(3) red-Read.prem*)
then show $\exists s' h'. \text{red} (Cread x E) (s, h) Cskip (s', h') \wedge \text{agrees } X (\text{fst } \sigma') s' \wedge \text{snd } \sigma' = h'$
using $\langle \text{hh} (\text{edenot } E s) = \text{Some } v \rangle \text{red-Read.hyps(1) red-Read.hyps(3)}$
red-Read.prem
by (*metis asm0 red-red-rtrans.red-Read snd-conv*)

qed
then show *?case by blast*
next
case (*red-Write* σ *ss* *hh* σ' *E* *E'*)
have $\bigwedge s h. \text{ agrees } X \text{ (fst } \sigma) s \wedge \text{ snd } \sigma = h \wedge \text{ fvC } (C\text{write } E E') \subseteq X \implies (\exists s' h'. \text{ red } (C\text{write } E E') (s, h) \text{ Cskip } (s', h') \wedge \text{ agrees } X \text{ (fst } \sigma') s' \wedge \text{ snd } \sigma' = h')$
proof –
fix *s h* **assume** *asm0*: $\text{ agrees } X \text{ (fst } \sigma) s \wedge \text{ snd } \sigma = h \wedge \text{ fvC } (C\text{write } E E') \subseteq X$
then have $\text{ edenot } E \text{ ss} = \text{ edenot } E s \wedge \text{ edenot } E' \text{ ss} = \text{ edenot } E' s$
using *red-Write(1) exp-agrees fstI fvC.simps(4)*
by (*metis (mono-tags, lifting) Un-subset-iff agrees-def in-mono*)
then show $\exists s' h'. \text{ red } (C\text{write } E E') (s, h) \text{ Cskip } (s', h') \wedge \text{ agrees } X \text{ (fst } \sigma') s' \wedge \text{ snd } \sigma' = h'$
by (*metis fst-conv asm0 red-Write.hyps(1) red-Write.hyps(2) red-Write.premis red-red-rtrans.red-Write snd-conv*)
qed
then show *?case by blast*
next
case (*red-Alloc* σ *ss* *hh* *v* σ' *x* *E*)
have $\bigwedge s h. \text{ agrees } X \text{ (fst } \sigma) s \wedge \text{ snd } \sigma = h \wedge \text{ fvC } (C\text{alloc } x E) \subseteq X \implies (\exists s' h'. \text{ red } (C\text{alloc } x E) (s, h) \text{ Cskip } (s', h') \wedge \text{ agrees } X \text{ (fst } \sigma') s' \wedge \text{ snd } \sigma' = h')$
proof –
fix *s h* **assume** *asm0*: $\text{ agrees } X \text{ (fst } \sigma) s \wedge \text{ snd } \sigma = h \wedge \text{ fvC } (C\text{alloc } x E) \subseteq X$
then have $\text{ edenot } E \text{ ss} = \text{ edenot } E s$
using *red-Alloc(1) exp-agrees fst-conv fvC.simps(5)*
by (*metis (mono-tags, lifting) Un-iff agrees-def in-mono*)
then have $\text{ agrees } X \text{ (fst } \sigma') (s(x := v))$
by (*metis agrees-update1 asm0 fstI red-Alloc.hyps(1) red-Alloc.hyps(3) red-Alloc.premis*)
then show $\exists s' h'. \text{ red } (C\text{alloc } x E) (s, h) \text{ Cskip } (s', h') \wedge \text{ agrees } X \text{ (fst } \sigma') s' \wedge \text{ snd } \sigma' = h'$
by (*metis <edenot E ss = edenot E s> red-Alloc.hyps(1) red-Alloc.hyps(2) red-Alloc.hyps(3) red-Alloc.premis red-red-rtrans.red-Alloc snd-eqD asm0*)
qed
then show *?case by blast*
next
case (*red-Free* σ *ss* *hh* σ' *E*)
have $\bigwedge s h. \text{ agrees } X \text{ (fst } \sigma) s \wedge \text{ snd } \sigma = h \wedge \text{ fvC } (C\text{dispose } E) \subseteq X \implies (\exists s' h'. \text{ red } (C\text{dispose } E) (s, h) \text{ Cskip } (s', h') \wedge \text{ agrees } X \text{ (fst } \sigma') s' \wedge \text{ snd } \sigma' = h')$
proof –
fix *s h* **assume** *asm0*: $\text{ agrees } X \text{ (fst } \sigma) s \wedge \text{ snd } \sigma = h \wedge \text{ fvC } (C\text{dispose } E) \subseteq X$
then have $\text{ edenot } E \text{ ss} = \text{ edenot } E s$
using *red-Free(1) exp-agrees fst-eqD fvC.simps(6)*
by (*metis agrees-def in-mono*)
then show $\exists s' h'. \text{ red } (C\text{dispose } E) (s, h) \text{ Cskip } (s', h') \wedge \text{ agrees } X \text{ (fst } \sigma') s' \wedge \text{ snd } \sigma' = h'$
using *red-Free.hyps(1) red-Free.hyps(2) red-Free.premis asm0 by fastforce*
qed
then show *?case by blast*

```

next
  case (NoStep C σ)
  then show ?case
    using red-red-rtrans.NoStep by blast
next
  case (OneMoreStep C σ C' σ' C'' σ'')
  have  $\bigwedge s h. \text{ agrees } X (fst \sigma) s \wedge snd \sigma = h \wedge fvC C \subseteq X \implies (\exists s' h'. \text{ red-rtrans } C (s, h) C'' (s', h') \wedge \text{ agrees } X (fst \sigma') s' \wedge snd \sigma'' = h')$ 
  proof -
    fix s h assume asm0:  $\text{ agrees } X (fst \sigma) s \wedge snd \sigma = h \wedge fvC C \subseteq X$ 
    then obtain h' s' where  $\text{ red } C (s, h) C' (s', h') \wedge \text{ agrees } X (fst \sigma') s' \wedge snd \sigma' = h'$ 
    using OneMoreStep(2) by auto
    then obtain h'' s'' where  $\text{ red-rtrans } C' (s', h') C'' (s'', h'') \wedge \text{ agrees } X (fst \sigma'') s'' \wedge snd \sigma'' = h''$ 
    using OneMoreStep.hyps(4) asm0 red-properties(1) by fastforce
    then show  $\exists s' h'. \text{ red-rtrans } C (s, h) C'' (s', h') \wedge \text{ agrees } X (fst \sigma'') s' \wedge snd \sigma'' = h'$ 
    using  $\langle \text{ red } C (s, h) C' (s', h') \wedge \text{ agrees } X (fst \sigma') s' \wedge snd \sigma' = h' \rangle$ 
    red-red-rtrans.OneMoreStep by blast
  qed
  then show ?case by blast
qed (fastforce+)

```

lemma *red-agrees[rule-format]*:

```

red C σ C' σ'  $\implies \forall X s. \text{ agrees } X (fst \sigma) s \longrightarrow snd \sigma = h \longrightarrow fvC C \subseteq X \longrightarrow$ 
 $(\exists s' h'. \text{ red } C (s, h) C' (s', h') \wedge \text{ agrees } X (fst \sigma') s' \wedge snd \sigma' = h')$ 
using red-agrees-aux(1) by blast

```

lemma *aborts-agrees*:

```

assumes aborts C σ
  and agrees (fvC C) (fst σ) s
  and snd σ = h
shows aborts C (s, h)
using assms

```

proof (*induct arbitrary: s h rule: aborts.induct*)

```

case (aborts-Atomic C σ C' σ')

```

```

  then obtain s' where  $\text{ red-rtrans } C (s, h) C' (s', snd \sigma') \wedge \text{ agrees } (fvC C) (fst \sigma') s'$ 

```

```

  by (metis dual-order.refl fvC.simps(11) red-agrees-aux(2))

```

```

  moreover have  $\text{ agrees } (fvC C') (fst \sigma') s'$ 

```

```

  using calculation red-properties(2)

```

```

  by (meson agrees-def in-mono)

```

```

  ultimately show ?case

```

```

  using aborts-Atomic.hyps(3) by blast

```

next

```

case (aborts-Read E σ x)

```

```

then show ?case

```

```

  using aborts.aborts-Read[of E σ x] exp-agrees[of E fst σ s] fst-conv fvC.simps(3)

```

```

snd-conv
  by (simp add: aborts.aborts-Read agrees-def)
next
  case (aborts-Write E σ E')
  then show ?case
    using aborts.aborts-Write[of E σ E'] exp-agrees[of - fst σ s] fst-conv fvC.simps(4)[of
E E'] snd-conv
    by (simp add: aborts.aborts-Write agrees-def)
next
  case (aborts-Free E σ)
  then show ?case
    using exp-agrees by auto
next
  case (aborts-Par1 C1 σ C2)
  then have agrees (fvC C1) (fst σ) s
    by (simp add: agrees-def)
  then show ?case using aborts.aborts-Par1
    by (simp add: aborts-Par1.hyps(2) aborts-Par1.premis(2))
next
  case (aborts-Par2 C2 σ C1)
  then have agrees (fvC C2) (fst σ) s
    by (simp add: agrees-def)
  then show ?case using aborts.aborts-Par2
    by (simp add: aborts-Par2.hyps(2) aborts-Par2.premis(2))
next
  case (aborts-Seq C1 σ C2)
  then have agrees (fvC C1) (fst σ) s
    by (simp add: agrees-def)
  then show ?case
    by (simp add: aborts.aborts-Seq aborts-Seq.hyps(2) aborts-Seq.premis(2))
qed

```

corollary *exp-agrees2[simp]*:
 $x \notin \text{fv}E E \implies \text{edenot } E (s(x := v)) = \text{edenot } E s$
 by (rule *exp-agrees*, simp add: *agrees-def*)

lemma *agrees-update*:
 assumes $a \notin S$
 shows $\text{agrees } S s (s(a := v))$
 by (simp add: *agrees-def* *assms*)

lemma *agrees-comm*:
 $\text{agrees } S s s' \longleftrightarrow \text{agrees } S s' s$
 by (*metis* *agrees-def*)

lemma *not-in-dom*:
 assumes $x \notin \text{dom } s$
 shows $s x = \text{None}$

using *assms* by *auto*

lemma *agrees-minusD*:

agrees $(-X) x y \implies X \cap Y = \{\} \implies \text{agrees } Y x y$

by (*metis Int-Un-eq(2) agrees-union compl-le-swap1 inf.order-iff inf-shunt*)

end

3 CommCSL

theory *CommCSL*

imports *Lang StateModel*

begin

definition *no-guard* :: $('i, 'a) \text{ heap} \Rightarrow \text{bool}$ **where**

no-guard $h \longleftrightarrow \text{get-gs } h = \text{None} \wedge (\forall k. \text{get-gu } h k = \text{None})$

typedef $'a \text{ precondition} = \{ \text{pre} :: ('a \Rightarrow 'a \Rightarrow \text{bool}) \mid \text{pre}. \forall a b. \text{pre } a b \longrightarrow (\text{pre } b a \wedge \text{pre } a a) \}$

using *Sup2-E* by *auto*

lemma *charact-rep-prec*:

assumes *Rep-precondition* $\text{pre } a b$

shows *Rep-precondition* $\text{pre } b a \wedge \text{Rep-precondition } \text{pre } a a$

using *Rep-precondition assms* by *fastforce*

typedef $('i, 'a) \text{ indexed-precondition} = \{ \text{pre} :: ('i \Rightarrow 'a \Rightarrow 'a \Rightarrow \text{bool}) \mid \text{pre}. \forall a b k. \text{pre } k a b \longrightarrow (\text{pre } k b a \wedge \text{pre } k a a) \}$

using *Sup2-E* by *auto*

lemma *charact-rep-indexed-prec*:

assumes *Rep-indexed-precondition* $\text{pre } k a b$

shows *Rep-indexed-precondition* $\text{pre } k b a \wedge \text{Rep-indexed-precondition } \text{pre } k a a$

by (*metis (no-types, lifting) Abs-indexed-precondition-cases Rep-indexed-precondition-cases Rep-indexed-precondition-inverse assms mem-Collect-eq*)

type-synonym $'a \text{ list-exp} = \text{store} \Rightarrow 'a \text{ list}$

3.1 Assertion Language

datatype $('i, 'a, 'v) \text{ assertion} =$

Bool $b \text{exp}$

| *Emp*

| *And* $('i, 'a, 'v) \text{ assertion } ('i, 'a, 'v) \text{ assertion}$

| *Star* $('i, 'a, 'v) \text{ assertion } ('i, 'a, 'v) \text{ assertion} \quad (- * - 70)$

| *Low* $b \text{exp}$

| *LowExp* exp

| *PointsTo* $\text{exp } \text{prat } \text{exp}$

| *Exists var* ('i, 'a, 'v) *assertion*

| *EmptyFullGuards*

| *PreSharedGuards* 'a *precondition*

| *PreUniqueGuards* ('i, 'a) *indexed-precondition*

| *View normal-heap* \Rightarrow 'v ('i, 'a, 'v) *assertion store* \Rightarrow 'v

| *SharedGuard* *prat store* \Rightarrow 'a *multiset*

| *UniqueGuard* 'i 'a *list-exp*

| *Imp bexp* ('i, 'a, 'v) *assertion*

| *NoGuard*

inductive *PRE-shared-simpler* :: ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a *multiset* \Rightarrow 'a *multiset* \Rightarrow bool **where**

Empty: *PRE-shared-simpler spre* {#} {#}

| *Step*: \llbracket *PRE-shared-simpler spre* a b ; *spre* xa xb $\rrbracket \Longrightarrow$ *PRE-shared-simpler spre* (a + {# xa #}) (b + {# xb #})

definition *PRE-unique* :: ('b \Rightarrow 'b \Rightarrow bool) \Rightarrow 'b *list* \Rightarrow 'b *list* \Rightarrow bool **where**

PRE-unique upre uargs uargs' \longleftrightarrow length uargs = length uargs' \wedge ($\forall i. i \geq 0 \wedge i < \text{length } uargs' \longrightarrow \text{upre } (uargs ! i) (uargs' ! i)$)

fun *hyper-sat* :: (store \times ('i, 'a) *heap*) \Rightarrow (store \times ('i, 'a) *heap*) \Rightarrow ('i, 'a, nat) *assertion* \Rightarrow bool (-, - \models - [51, 65, 66] 50) **where**

 (s, -), (s', -) \models Bool b \longleftrightarrow bdenot b s \wedge bdenot b s'

| (-, h), (-, h') \models Emp \longleftrightarrow dom (get-fh h) = {} \wedge dom (get-fh h') = {}

| $\sigma, \sigma' \models$ And A B \longleftrightarrow $\sigma, \sigma' \models A \wedge \sigma, \sigma' \models B$

| (s, h), (s', h') \models Star A B \longleftrightarrow ($\exists h1 h2 h1' h2'. \text{Some } h = \text{Some } h1 \oplus \text{Some } h2 \wedge \text{Some } h' = \text{Some } h1' \oplus \text{Some } h2'$)

\wedge (s, h1), (s', h1') $\models A \wedge$ (s, h2), (s', h2') $\models B$)

| (s, h), (s', h') \models Low e \longleftrightarrow bdenot e s = bdenot e s'

| (s, h), (s', h') \models PointsTo loc p x \longleftrightarrow get-fh h (edenot loc s) = Some (p, edenot x s) \wedge get-fh h' (edenot loc s') = Some (p, edenot x s')

\wedge dom (get-fh h) = {edenot loc s} \wedge dom (get-fh h') = {edenot loc s'}

| (s, h), (s', h') \models Exists x A \longleftrightarrow ($\exists v v'. (s(x := v), h), (s'(x := v'), h') \models A$)

| (s, h), (s', h') \models EmptyFullGuards \longleftrightarrow (get-gs h = Some (pwrite, {#}) \wedge ($\forall k. \text{get-gu } h k = \text{Some } []$) \wedge get-gs h' = Some (pwrite, {#}) \wedge ($\forall k. \text{get-gu } h' k = \text{Some } []$))

| (s, h), (s', h') \models PreSharedGuards spre \longleftrightarrow

 (\exists sargs sargs'. get-gs h = Some (pwrite, sargs) \wedge get-gs h' = Some (pwrite, sargs') \wedge *PRE-shared-simpler* (*Rep-precondition spre*) sargs sargs')

$\wedge \text{get-fh } h = \text{Map.empty} \wedge \text{get-fh } h' = \text{Map.empty}$
 $| (s, h), (s', h') \models \text{PreUniqueGuards } \text{upre} \longleftrightarrow$
 $(\exists \text{uargs } \text{uargs}'. (\forall k. \text{get-gu } h \ k = \text{Some } (\text{uargs } k)) \wedge (\forall k. \text{get-gu } h' \ k = \text{Some}$
 $(\text{uargs}' k)) \wedge (\forall k. \text{PRE-unique } (\text{Rep-indexed-precondition } \text{upre } k) (\text{uargs } k) (\text{uargs}'$
 $k)) \wedge \text{get-fh } h = \text{Map.empty} \wedge \text{get-fh } h' = \text{Map.empty}$)

$| (s, h), (s', h') \models \text{View } f \ J \ e \longleftrightarrow ((s, h), (s', h') \models J \wedge f \ (\text{normalize } (\text{get-fh } h)))$
 $= e \ s \wedge f \ (\text{normalize } (\text{get-fh } h')) = e \ s'$
 $| (s, h), (s', h') \models \text{SharedGuard } \pi \ ms \longleftrightarrow ((\forall k. \text{get-gu } h \ k = \text{None} \wedge \text{get-gu } h' \ k$
 $= \text{None}) \wedge \text{get-gs } h = \text{Some } (\pi, ms \ s) \wedge \text{get-gs } h' = \text{Some } (\pi, ms \ s')$
 $\wedge \text{get-fh } h = \text{Map.empty} \wedge \text{get-fh } h' = \text{Map.empty})$

$| (s, h), (s', h') \models \text{UniqueGuard } k \ \text{lexp} \longleftrightarrow (\text{get-gs } h = \text{None} \wedge \text{get-gu } h \ k = \text{Some}$
 $(\text{lexp } s) \wedge \text{get-gu } h' \ k = \text{Some } (\text{lexp } s') \wedge \text{get-gs } h' = \text{None}$
 $\wedge \text{get-fh } h = \text{Map.empty} \wedge \text{get-fh } h' = \text{Map.empty} \wedge (\forall k'. k' \neq k \longrightarrow$
 $\text{get-gu } h \ k' = \text{None} \wedge \text{get-gu } h' \ k' = \text{None}))$

$| (s, h), (s', h') \models \text{LowExp } e \longleftrightarrow \text{edenot } e \ s = \text{edenot } e \ s'$

$| (s, h), (s', h') \models \text{Imp } b \ A \longleftrightarrow \text{bdenot } b \ s = \text{bdenot } b \ s' \wedge (\text{bdenot } b \ s \longrightarrow (s, h),$
 $(s', h') \models A)$

$| (s, h), (s', h') \models \text{NoGuard} \longleftrightarrow (\text{get-gs } h = \text{None} \wedge (\forall k. \text{get-gu } h \ k = \text{None}) \wedge$
 $\text{get-gs } h' = \text{None} \wedge (\forall k. \text{get-gu } h' \ k = \text{None}))$

lemma *sat-PreUniqueE*:

assumes $(s, h), (s', h') \models \text{PreUniqueGuards } \text{upre}$
shows $\exists \text{uargs } \text{uargs}'. (\forall k. \text{get-gu } h \ k = \text{Some } (\text{uargs } k)) \wedge (\forall k. \text{get-gu } h' \ k =$
 $\text{Some } (\text{uargs}' k)) \wedge (\forall k. \text{PRE-unique } (\text{Rep-indexed-precondition } \text{upre } k) (\text{uargs } k)$
 $(\text{uargs}' k))$
using *assms by auto*

lemma *decompose-heap-triple*:

$h = (\text{get-fh } h, \text{get-gs } h, \text{get-gu } h)$
by *simp*

definition *depends-only-on* :: $(\text{store} \Rightarrow 'v) \Rightarrow \text{var set} \Rightarrow \text{bool}$ **where**
 $\text{depends-only-on } e \ S \longleftrightarrow (\forall s \ s'. \text{agrees } S \ s \ s' \longrightarrow e \ s = e \ s')$

lemma *depends-only-onI*:

assumes $\bigwedge s \ s' :: \text{store. agrees } S \ s \ s' \Longrightarrow e \ s = e \ s'$
shows *depends-only-on* $e \ S$
using *assms depends-only-on-def by blast*

definition *fvS* :: $(\text{store} \Rightarrow 'v) \Rightarrow \text{var set}$ **where**
 $\text{fvS } e = (\text{SOME } S. \text{depends-only-on } e \ S)$

```

lemma fvSE:
  assumes agrees (fvS e) s s'
  shows  $e\ s = e\ s'$ 
proof –
  have depends-only-on e UNIV
  proof (rule depends-only-onI)
    fix s s' :: store assume agrees UNIV s s'
    have  $s = s'$ 
    proof (rule ext)
      fix x :: var have  $x \in UNIV$ 
      by auto
      then show  $s\ x = s'\ x$ 
      by (meson  $\langle$ agrees UNIV s s' $\rangle$  agrees-def)
    qed
  then show  $e\ s = e\ s'$  by simp
qed
then show ?thesis
  by (metis assms depends-only-on-def fvS-def someI-ex)
qed

```

```

fun fvA :: ('i, 'a, 'v) assertion  $\Rightarrow$  var set where
  fvA (Bool b) = fvB b
| fvA (And A B) = fvA A  $\cup$  fvA B
| fvA (Star A B) = fvA A  $\cup$  fvA B
| fvA (Low e) = fvB e
| fvA Emp = {}
| fvA (PointsTo v va vb) = fvE v  $\cup$  fvE vb
| fvA (Exists x A) = fvA A - {x}
| fvA (SharedGuard - e) = fvS e
| fvA (UniqueGuard - e) = fvS e
| fvA (View view A e) = fvA A  $\cup$  fvS e
| fvA (PreSharedGuards -) = {}
| fvA (PreUniqueGuards -) = {}
| fvA EmptyFullGuards = {}
| fvA (LowExp x) = fvE x
| fvA (Imp b A) = fvB b  $\cup$  fvA A

```

```

definition subS :: var  $\Rightarrow$  exp  $\Rightarrow$  (store  $\Rightarrow$  'v)  $\Rightarrow$  (store  $\Rightarrow$  'v) where
  subS x E e = ( $\lambda s. e\ (s(x := \text{edenot } E\ s))$ )

```

```

lemma subS-assign:
  subS x E e s  $\longleftrightarrow e\ (s(x := \text{edenot } E\ s))$ 
  by (simp add: subS-def)

```

```

fun collect-existentials :: ('i, 'a, nat) assertion  $\Rightarrow$  var set where
  collect-existentials (And A B) = collect-existentials A  $\cup$  collect-existentials B
| collect-existentials (Star A B) = collect-existentials A  $\cup$  collect-existentials B
| collect-existentials (Exists x A) = collect-existentials A  $\cup$  {x}

```



```

| collect-existentials (View view A e) = collect-existentials A
| collect-existentials (Imp - A) = collect-existentials A
| collect-existentials - = {}

```

```

fun subA :: var ⇒ exp ⇒ ('i, 'a, nat) assertion ⇒ ('i, 'a, nat) assertion where
  subA x E (And A B) = And (subA x E A) (subA x E B)
| subA x E (Star A B) = Star (subA x E A) (subA x E B)
| subA x E (Bool B) = Bool (subB x E B)
| subA x E (Low e) = Low (subB x E e)
| subA x E (LowExp e) = LowExp (subE x E e)
| subA x E (UniqueGuard k e) = UniqueGuard k (subS x E e)
| subA x E (SharedGuard π e) = SharedGuard π (subS x E e)
| subA x E (View view A e) = View view (subA x E A) (subS x E e)
| subA x E (PointsTo loc π e) = PointsTo (subE x E loc) π (subE x E e)
| subA x E (Exists y A) = (if x = y then Exists y A else Exists y (subA x E A))
| subA x E (Imp b A) = Imp (subB x E b) (subA x E A)
| subA - - A = A

```

lemma subA-assign:

```

assumes collect-existentials A ∩ fvE E = {}
shows (s, h), (s', h') ⊨ subA x E A ⟷ (s(x := edenot E s), h), (s'(x := edenot
E s^), h') ⊨ A
using assms
proof (induct A arbitrary: s h s' h')
  case (And A1 A2)
  then show ?case
    by (simp add: disjoint-iff-not-equal)
  next
  case (Star A1 A2)
  then show ?case
    by (simp add: disjoint-iff-not-equal)
  next
  case (Bool x)
  then show ?case
    by (metis hyper-sat.simps(1) subA.simps(3) subB-assign)
  next
  case (Low x2)
  then show ?case
    by (metis (no-types, lifting) hyper-sat.simps(5) subA.simps(4) subB-assign)
  next
  case (LowExp x2)
  then show ?case
    by (metis (no-types, lifting) hyper-sat.simps(14) subA.simps(5) subE-assign)
  next
  case (PointsTo x1a x2 x3)
  then show ?case
    by (metis (no-types, lifting) hyper-sat.simps(6) subA.simps(9) subE-assign)
  next
  case (Exists y A)

```

```

then have asm0: collect-existentials  $A \cap \text{fv}E E = \{\}$ 
  by auto
show ?case (is ?A  $\longleftrightarrow$  ?B)
proof
  show ?A  $\implies$  ?B
  proof –
    assume ?A
    show ?B
    proof (cases  $x = y$ )
      case True
        then show ?thesis by (metis (no-types, opaque-lifting)  $\langle ?A \rangle$  fun-upd-upd
hyper-sat.simps(7) subA.simps(10))
      next
        case False
          then obtain  $v v'$  where  $(s(y := v), h), (s'(y := v'), h') \models \text{sub}A x E A$ 
            using  $\langle (s, h), (s', h') \models \text{sub}A x E (\text{Exists } y A) \rangle$  by auto
          then have  $((s(y := v))(x := \text{edenot } E (s(y := v))), h), ((s'(y := v'))(x :=$ 
edenot  $E (s'(y := v'))), h') \models A$ 
            using Exists asm0 by blast
          moreover have  $y \notin \text{fv}E E$ 
            using Exists.prems by force
          then have  $\text{edenot } E (s(y := v)) = \text{edenot } E s \wedge \text{edenot } E (s'(y := v')) =$ 
edenot  $E s'$ 
            by (metis agrees-update exp-agrees)
          moreover have  $(s(y := v))(x := \text{edenot } E s) = (s(x := \text{edenot } E s))(y :=$ 
v)
             $\wedge (s'(y := v'))(x := \text{edenot } E s') = (s'(x := \text{edenot } E s'))(y := v')$ 
            by (simp add: False fun-upd-twist)
          ultimately show ?thesis using False hyper-sat.simps(7)
            by metis
        qed
      qed
    assume ?B
    show ?A
    proof (cases  $x = y$ )
      case True
        then show ?thesis
          using  $\langle (s(x := \text{edenot } E s), h), (s'(x := \text{edenot } E s'), h') \models \text{Exists } y A \rangle$  by
fastforce
      next
        case False
          then obtain  $v v'$  where  $((s(x := \text{edenot } E s))(y := v), h), ((s'(x := \text{edenot}$ 
E s')
 $(y := v'), h') \models A$ 
            using  $\langle (s(x := \text{edenot } E s), h), (s'(x := \text{edenot } E s'), h') \models \text{Exists } y A \rangle$ 
hyper-sat.simps(7) by blast
          moreover have  $(s(y := v))(x := \text{edenot } E s) = (s(x := \text{edenot } E s))(y := v)$ 
 $\wedge (s'(y := v'))(x := \text{edenot } E s') = (s'(x := \text{edenot } E s'))(y := v')$ 
            by (simp add: False fun-upd-twist)
          then have  $((s(y := v))(x := \text{edenot } E (s(y := v))), h), ((s'(y := v'))(x :=$ 

```

```

edenot E (s'(y := v^)), h') ⊨ A
  using Exists.premis calculation by auto
  then show ?thesis
  by (metis Exists.hyps False asm0 hyper-sat.simps(7) subA.simps(10))
qed
qed
next
  case (View x1a A x3)
  then show ?case
  by (metis (mono-tags, lifting) collect-existentials.simps(4) hyper-sat.simps(11)
subA.simps(8) subS-def)
next
  case (SharedGuard x1a x2)
  then show ?case
  by (metis (mono-tags, lifting) hyper-sat.simps(12) subA.simps(7) subS-def)
next
  case (UniqueGuard x)
  then show ?case
  by (metis (mono-tags, lifting) hyper-sat.simps(13) subA.simps(6) subS-def)
next
  case (Imp b A)
  then show ?case
  by (metis collect-existentials.simps(5) hyper-sat.simps(15) subA.simps(11)
subB-assign)
qed (auto)

```

```

lemma PRE-uniqueI:
  assumes length uargs = length uargs'
  and  $\bigwedge i. i \geq 0 \wedge i < \text{length } uargs' \implies \text{upre } (uargs ! i) (uargs' ! i)$ 
  shows PRE-unique upre uargs uargs'
  using assms PRE-unique-def by blast

```

```

lemma PRE-unique-implies-tl:
  assumes PRE-unique upre (ta # qa) (tb # qb)
  shows PRE-unique upre qa qb
proof (rule PRE-uniqueI)
  show length qa = length qb
  by (metis PRE-unique-def assms diff-Suc-1 length-Cons)
  fix i assume  $0 \leq i \wedge i < \text{length } qb$ 
  then have upre ((ta # qa) ! (i + 1)) ((tb # qb) ! (i + 1))
  using assms PRE-unique-def [of upre ⟨ta # qa⟩ ⟨tb # qb⟩]
  by (auto simp add: less-Suc-eq-le dest: spec [of - ⟨Suc i⟩])
  then show upre (qa ! i) (qb ! i)
  by simp
qed

```

```

lemma charact-PRE-unique:
  assumes PRE-unique (Rep-indexed-precondition pre k) a b

```

shows $PRE\text{-unique} (Rep\text{-indexed-precondition } pre \ k) \ b \ a \wedge PRE\text{-unique} (Rep\text{-indexed-precondition } pre \ k) \ a \ a$
using *assms*
proof (*induct length a arbitrary: a b*)
case 0
then show *?case*
by (*simp add: PRE-unique-def*)
next
case (*Suc n*)
then obtain *ha ta hb tb* **where** $a = ha \# ta \ b = hb \# tb$
by (*metis One-nat-def PRE-unique-def Suc-le-length-iff le-add1 plus-1-eq-Suc*)
then have $n = length \ ta$
using *Suc.hyps(2)* **by** *auto*
then have $PRE\text{-unique} (Rep\text{-indexed-precondition } pre \ k) \ tb \ ta \wedge PRE\text{-unique} (Rep\text{-indexed-precondition } pre \ k) \ ta \ ta$
by (*metis PRE-unique-implies-tl Suc.hyps(1) Suc.prem1 <a = ha # ta> <b = hb # tb>*)
then show *?case*
by (*metis PRE-unique-def Suc.prem1 charact-rep-indexed-prec*)
qed

lemma *charact-PRE-shared-simpler*:
assumes $PRE\text{-shared-simpler } rpre \ a \ b$
and $Rep\text{-precondition } pre = rpre$
shows $PRE\text{-shared-simpler} (Rep\text{-precondition } pre) \ b \ a \wedge PRE\text{-shared-simpler} (Rep\text{-precondition } pre) \ a \ a$
using *assms*
proof (*induct arbitrary: pre rule: PRE-shared-simpler.induct*)
case (*Empty spre*)
then show *?case*
by (*simp add: PRE-shared-simpler.Empty*)
next
case (*Step spre a b xa xb*)
then have $spre \ xb \ xa \wedge spre \ xa \ xa$ **using** *charact-rep-prec* **by** *metis*
then show *?case*
by (*metis PRE-shared-simpler.Step Step.hyps(2) Step.prem1*)
qed

lemma *always-sat-refl-aux*:
assumes $(s, h), (s', h') \models A$
shows $(s, h), (s, h) \models A$
using *assms*
proof (*induct A arbitrary: s h s' h'*)
case (*Star A B*)
then obtain $ha \ hb \ ha' \ hb'$ **where** $Some \ h = Some \ ha \oplus Some \ hb \ Some \ h' = Some \ ha' \oplus Some \ hb'$
 $(s, ha), (s', ha') \models A \ (s, hb), (s', hb') \models B$
by *auto*

```

then have (s, ha), (s, ha) ⊨ A ∧ (s, hb), (s, hb) ⊨ B
  using Star.hyps(1) Star.hyps(2) by blast
then show ?case
  using ⟨Some h = Some ha ⊕ Some hb⟩ hyper-sat.simps(4) by blast
next
  case (Exists x A)
  then show ?case
    by (meson hyper-sat.simps(7))
next
  case (PreSharedGuards x)
  then show ?case
    using charact-PRE-shared-simpler by force
next
  case (PreUniqueGuards upre)
  then obtain uargs uargs' where (∀ k. get-gu h k = Some (uargs k)) ∧
    (∀ k. get-gu h' k = Some (uargs' k)) ∧ (∀ k. PRE-unique (Rep-indexed-precondition
upre k) (uargs k) (uargs' k)) ∧ get-fh h = Map.empty ∧ get-fh h' = Map.empty
    using hyper-sat.simps(10)[of s h s' h' upre] by blast
  then show (s, h), (s, h) ⊨ PreUniqueGuards upre
    using charact-PRE-unique hyper-sat.simps(10)[of s h s h upre]
    by metis
qed (auto)

lemma always-sat-refl:
  assumes σ, σ' ⊨ A
  shows σ, σ ⊨ A
  by (metis always-sat-refl-aux assms prod.exhaust-sel)

lemma agrees-same-aux:
  assumes agrees (fvA A) s s''
  and (s, h), (s', h') ⊨ A
  shows (s'', h), (s', h') ⊨ A
  using assms
proof (induct A arbitrary: s s' s'' h h')
  case (Bool b)
  then show ?case by (simp add: bexp-agrees)
next
  case (And A1 A2)
  then show ?case using fvA.simps(2) hyper-sat.simps(3)
  by (metis (mono-tags, lifting) UnCI agrees-def)
next
  case (Star A B)
  then obtain ha hb ha' hb' where Some h = Some ha ⊕ Some hb Some h' =
Some ha' ⊕ Some hb'
  (s, ha), (s', ha') ⊨ A (s, hb), (s', hb') ⊨ B
  by auto
  then have (s'', ha), (s', ha') ⊨ A ∧ (s'', hb), (s', hb') ⊨ B
  using Star.hyps[of s s'' - s' -] Star.prem(1)
  by (simp add: agrees-def)

```

```

then show ?case
  using ⟨Some h = Some ha ⊕ Some hb⟩ ⟨Some h' = Some ha' ⊕ Some hb'⟩
hyper-sat.simps(4) by blast
next
  case (Low e)
  then have bdenot e s = bdenot e s''
    by (metis bexp-agrees fvA.simps(4))
  then show ?case using Low by simp
next
  case (LowExp e)
  then have edenot e s = edenot e s''
    by (metis exp-agrees fvA.simps(14))
  then show ?case using LowExp by simp
next
  case (PointsTo l π v)
  then have edenot l s = edenot l s'' ∧ edenot v s = edenot v s''
    by (simp add: agrees-def exp-agrees)
  then show ?case using PointsTo by auto
next
  case (Exists x A)
  then obtain v v' where (s(x := v), h), (s'(x := v'), h') ⊨ A
    by auto
  moreover have agrees (fvA A) (s(x := v)) (s''(x := v))
  proof (rule agreesI)
    fix y show y ∈ fvA A ⇒ (s(x := v)) y = (s''(x := v)) y
      apply (cases y = x)
      apply simp
    using Diff-iff[of y fvA A {x}] Exists.prem(1) agrees-def empty-iff fun-upd-apply[of
s x v] fun-upd-apply[of s'' x v] fvA.simps(7) insert-iff
      by metis
  qed
  ultimately have (s''(x := v), h), (s'(x := v'), h') ⊨ A
    using Exists.hyps by blast
  then show ?case by auto
next
  case (View x1a A e)
  then have (s'', h), (s', h') ⊨ A ∧ e s = e s''
    using fvA.simps(10) fvSE hyper-sat.simps(11) agrees-union
    by metis
  then show ?case
    using View.prem(2) by auto
next
  case (SharedGuard x1a x2)
  then show ?case using fvSE by auto
next
  case (UniqueGuard x)
  then show ?case using fvSE by auto
next
  case (Imp b A)

```

then show *?case*
by (*metis agrees-union bexp-agrees fvA.simps(15) hyper-sat.simps(15)*)
qed (*auto*)

lemma *agrees-same*:

assumes *agrees (fvA A) s s''*
shows $(s, h), (s', h') \models A \longleftrightarrow (s'', h), (s', h') \models A$
by (*metis (mono-tags, lifting) agrees-def agrees-same-aux assms*)

lemma *sat-comm-aux*:

$(s, h), (s', h') \models A \implies (s', h'), (s, h) \models A$
proof (*induct A arbitrary: s h s' h'*)
case (*Star A B*)
then obtain *ha hb ha' hb'* **where** *Some h = Some ha \oplus Some hb* *Some h' = Some ha' \oplus Some hb'*
 $(s, ha), (s', ha') \models A$ $(s, hb), (s', hb') \models B$
by *auto*
then have $(s', ha'), (s, ha) \models A \wedge (s', hb'), (s, hb) \models B$
using *Star.hyps(1) Star.hyps(2)* **by** *presburger*
then show *?case*
using $\langle \text{Some } h = \text{Some } ha \oplus \text{Some } hb \rangle \langle \text{Some } h' = \text{Some } ha' \oplus \text{Some } hb' \rangle$
hyper-sat.simps(4) **by** *blast*

next

case (*Exists x A*)
then obtain *v v'* **where** $(s(x := v), h), (s'(x := v'), h') \models A$
by *auto*
then have $(s'(x := v'), h'), (s(x := v), h) \models A$
using *Exists.hyps* **by** *blast*
then show *?case* **by** *auto*

next

case (*PreSharedGuards x*)
then show *?case*
by (*meson charact-PRE-shared-simpler hyper-sat.simps(9)*)

next

case (*PreUniqueGuards upre*)
then obtain *uargs uargs'* **where** $(\forall k. \text{get-gu } h \ k = \text{Some } (uargs \ k)) \wedge$
 $(\forall k. \text{get-gu } h' \ k = \text{Some } (uargs' \ k)) \wedge (\forall k. \text{PRE-unique } (\text{Rep-indexed-precondition}$
upre k) (uargs k) (uargs' k)) \wedge get-fh h = Map.empty \wedge get-fh h' = Map.empty
using *hyper-sat.simps(10)[of s h s' h' upre]* **by** *blast*
then show $(s', h'), (s, h) \models \text{PreUniqueGuards upre}$
using *charact-PRE-unique hyper-sat.simps(10)[of s' h' s h upre]*
by *metis*
qed (*auto*)

lemma *sat-comm*:

$\sigma, \sigma' \models A \longleftrightarrow \sigma', \sigma \models A$
using *sat-comm-aux surj-pair* **by** *metis*

definition *precise* **where**

precise $J \longleftrightarrow (\forall s1\ H1\ h1\ h1'\ s2\ H2\ h2\ h2'.\ H1 \succeq h1 \wedge H1 \succeq h1' \wedge H2 \succeq h2 \wedge H2 \succeq h2' \wedge (s1, h1), (s2, h2) \models J \wedge (s1, h1'), (s2, h2') \models J \longrightarrow h1' = h1 \wedge h2' = h2)$

lemma *preciseI*:

assumes $\bigwedge s1\ H1\ h1\ h1'\ s2\ H2\ h2\ h2'.\ H1 \succeq h1 \wedge H1 \succeq h1' \wedge H2 \succeq h2 \wedge H2 \succeq h2' \implies$

$(s1, h1), (s2, h2) \models J \implies (s1, h1'), (s2, h2') \models J \implies h1' = h1 \wedge h2' = h2$

shows *precise* J

using *assms precise-def by blast*

lemma *preciseE*:

assumes *precise* J

and $H1 \succeq h1 \wedge H1 \succeq h1' \wedge H2 \succeq h2 \wedge H2 \succeq h2'$

and $(s1, h1), (s2, h2) \models J \wedge (s1, h1'), (s2, h2') \models J$

shows $h1' = h1 \wedge h2' = h2$

using *assms(1) assms(2) assms(3) precise-def by blast*

definition *unary where*

unary $J \longleftrightarrow (\forall s\ h\ s'\ h'.\ (s, h), (s, h) \models J \wedge (s', h'), (s', h') \models J \longrightarrow (s, h), (s', h') \models J)$

lemma *unaryI*:

assumes $\bigwedge s1\ h1\ s2\ h2.\ (s1, h1), (s1, h1) \models J \wedge (s2, h2), (s2, h2) \models J \implies (s1, h1), (s2, h2) \models J$

shows *unary* J

using *assms unary-def by blast*

lemma *unary-smallerI*:

assumes $\bigwedge \sigma1\ \sigma2.\ \sigma1, \sigma1 \models J \wedge \sigma2, \sigma2 \models J \implies \sigma1, \sigma2 \models J$

shows *unary* J

using *assms unary-def by blast*

lemma *unaryE*:

assumes *unary* J

and $(s, h), (s, h) \models J \wedge (s', h'), (s', h') \models J$

shows $(s, h), (s', h') \models J$

using *assms(1) assms(2) unary-def by blast*

definition *entails* $:: ('i, 'a, nat)\ \text{assertion} \Rightarrow ('i, 'a, nat)\ \text{assertion} \Rightarrow \text{bool}$ **where**

entails $A\ B \longleftrightarrow (\forall \sigma\ \sigma'.\ \sigma, \sigma' \models A \longrightarrow \sigma, \sigma' \models B)$

lemma *entailsI*:

assumes $\bigwedge x\ y.\ x, y \models A \implies x, y \models B$

shows *entails* $A\ B$

using *assms entails-def* **by** *blast*

lemma *sat-points-to*:

assumes $(s, h :: ('i, 'a) \text{ heap}), (s, h) \models \text{PointsTo } a \ \pi \ e$

shows $\text{get-fh } h = [\text{edenot } a \ s \mapsto (\pi, \text{edenot } e \ s)]$

proof –

have $\text{get-fh } h (\text{edenot } a \ s) = \text{Some } (\pi, \text{edenot } e \ s) \wedge \text{dom } (\text{get-fh } h) = \{\text{edenot } a \ s\}$

using *assms* **by** *auto*

then show *?thesis*

by *fastforce*

qed

lemma *unary-inv-then-view*:

assumes *unary J*

shows *unary (View f J e)*

proof (*rule unaryI*)

fix $s \ h \ s' \ h'$

assume *asm0*: $(s, h), (s, h) \models \text{View } f \ J \ e \wedge (s', h'), (s', h') \models \text{View } f \ J \ e$

then show $(s, h), (s', h') \models \text{View } f \ J \ e$

by (*meson assms hyper-sat.simps(11) unaryE*)

qed

lemma *precise-inv-then-view*:

assumes *precise J*

shows *precise (View f J e)*

proof (*rule preciseI*)

fix $s1 \ H1 \ h1 \ h1' \ s2 \ H2 \ h2 \ h2'$

assume *asm0*: $H1 \succeq h1 \wedge H1 \succeq h1' \wedge H2 \succeq h2 \wedge H2 \succeq h2' \ (s1, h1), (s2, h2) \models \text{View } f \ J \ e$

$(s1, h1'), (s2, h2') \models \text{View } f \ J \ e$

then show $h1' = h1 \wedge h2' = h2$

by (*meson assms hyper-sat.simps(11) preciseE*)

qed

fun *syntactic-unary* :: $('i, 'a, \text{nat}) \text{ assertion} \Rightarrow \text{bool}$ **where**

syntactic-unary (Bool b) \longleftrightarrow True

| *syntactic-unary (And A B) \longleftrightarrow syntactic-unary A \wedge syntactic-unary B*

| *syntactic-unary (Star A B) \longleftrightarrow syntactic-unary A \wedge syntactic-unary B*

| *syntactic-unary (Low e) \longleftrightarrow False*

| *syntactic-unary Emp \longleftrightarrow True*

| *syntactic-unary (PointsTo v va vb) \longleftrightarrow True*

| *syntactic-unary (Exists x A) \longleftrightarrow syntactic-unary A*

| *syntactic-unary (SharedGuard - e) \longleftrightarrow True*

| *syntactic-unary (UniqueGuard - e) \longleftrightarrow True*

| *syntactic-unary (View view A e) \longleftrightarrow syntactic-unary A*

| *syntactic-unary (PreSharedGuards -) \longleftrightarrow False*

```

| syntactic-unary (PreUniqueGuards -)  $\longleftrightarrow$  False
| syntactic-unary EmptyFullGuards  $\longleftrightarrow$  True
| syntactic-unary (LowExp x)  $\longleftrightarrow$  False
| syntactic-unary (Imp b A)  $\longleftrightarrow$  False

```

lemma *syntactic-unary-implies-unary*:

```

assumes syntactic-unary A
shows unary A
using assms
proof (induct A)
  case (And A1 A2)
  show ?case
  proof (rule unary-smallerI)
    fix  $\sigma 1 \sigma 2$ 
    assume  $\sigma 1, \sigma 1 \models \text{And } A1 \ A2 \wedge \sigma 2, \sigma 2 \models \text{And } A1 \ A2$ 
    then have  $\sigma 1, \sigma 2 \models A1 \wedge \sigma 1, \sigma 2 \models A2$ 
    using And unary-def
    by (metis hyper-sat.simps(3) prod.exhaust-sel syntactic-unary.simps(2))
    then show  $\sigma 1, \sigma 2 \models \text{And } A1 \ A2$ 
    using hyper-sat.simps(3) by blast
  qed
next
  case (Star A B)
  then have unary A  $\wedge$  unary B by simp
  show ?case
  proof (rule unaryI)
    fix  $s1 \ h1 \ s2 \ h2$ 
    assume asm0:  $(s1, h1), (s1, h1) \models \text{Star } A \ B \wedge (s2, h2), (s2, h2) \models \text{Star } A$ 
    B
    then obtain  $a1 \ b1 \ a2 \ b2$  where Some  $h1 = \text{Some } a1 \oplus \text{Some } b1$   $(s1, a1),$ 
 $(s1, a1) \models A$   $(s1, b1), (s1, b1) \models B$ 
    Some  $h2 = \text{Some } a2 \oplus \text{Some } b2$   $(s2, a2), (s2, a2) \models A$   $(s2, b2), (s2, b2) \models$ 
    B
    by (meson always-sat-refl hyper-sat.simps(4))
    then have  $(s1, a1), (s2, a2) \models A \wedge (s1, b1), (s2, b2) \models B$ 
    using  $\langle \text{unary } A \wedge \text{unary } B \rangle$  unaryE by blast
    then show  $(s1, h1), (s2, h2) \models \text{Star } A \ B$ 
    using  $\langle \text{Some } h1 = \text{Some } a1 \oplus \text{Some } b1 \rangle \langle \text{Some } h2 = \text{Some } a2 \oplus \text{Some } b2 \rangle$ 
    hyper-sat.simps(4) by blast
  qed
next
  case (Exists x A)
  then have unary A by simp
  show ?case
  proof (rule unaryI)
    fix  $s1 \ h1 \ s2 \ h2$ 
    assume  $(s1, h1), (s1, h1) \models \text{Exists } x \ A \wedge (s2, h2), (s2, h2) \models \text{Exists } x \ A$ 
    then obtain  $v1 \ v2$  where  $(s1(x := v1), h1), (s1(x := v1), h1) \models A \wedge (s2(x$ 
 $:= v2), h2), (s2(x := v2), h2) \models A$ 

```

```

    by (meson always-sat-refl hyper-sat.simps(7))
  then have (s1(x := v1), h1), (s2(x := v2), h2) ⊨ A
    using ⟨unary A⟩ unary-def by blast
  then show (s1, h1), (s2, h2) ⊨ Exists x A by auto
qed
next
case (View view A x)
then have unary A by simp
show ?case
proof (rule unaryI)
  fix s1 h1 s2 h2
  assume asm0: (s1, h1), (s1, h1) ⊨ View view A x ∧ (s2, h2), (s2, h2) ⊨
View view A x
  then have (s1, h1), (s2, h2) ⊨ A
    by (meson ⟨unary A⟩ hyper-sat.simps(11) unaryE)
  then show (s1, h1), (s2, h2) ⊨ View view A x
    using asm0 by fastforce
qed
qed (auto simp add: unary-def)

```

```

record ('i, 'a, 'v) single-context =
  view :: (loc → val) ⇒ 'v
  abstract-view :: 'v ⇒ 'v
  saction :: 'v ⇒ 'a ⇒ 'v
  uaction :: 'i ⇒ 'v ⇒ 'a ⇒ 'v
  invariant :: ('i, 'a, 'v) assertion

```

type-synonym ('i, 'a, 'v) cont = ('i, 'a, 'v) single-context option

definition no-guard-assertion **where**

no-guard-assertion $A \longleftrightarrow (\forall s1\ h1\ s2\ h2. (s1, h1), (s2, h2) \models A \longrightarrow \text{no-guard } h1 \wedge \text{no-guard } h2)$

Axiom that says that view only depends on the part of the heap described by inv

definition view-function-of-inv :: ('i, 'a, nat) single-context ⇒ bool **where**

view-function-of-inv $\Gamma \longleftrightarrow (\forall (h :: ('i, 'a) \text{ heap}) (h' :: ('i, 'a) \text{ heap}) s. (s, h), (s, h) \models \text{invariant } \Gamma \wedge (h' \succeq h) \longrightarrow \text{view } \Gamma (\text{normalize } (\text{get-fh } h)) = \text{view } \Gamma (\text{normalize } (\text{get-fh } h')))$

definition wf-indexed-precondition :: ('i ⇒ 'a ⇒ 'a ⇒ bool) ⇒ bool **where**

wf-indexed-precondition $\text{pre} \longleftrightarrow (\forall a\ b\ k. \text{pre } k\ a\ b \longrightarrow (\text{pre } k\ b\ a \wedge \text{pre } k\ a\ a))$

definition wf-precondition :: ('a ⇒ 'a ⇒ bool) ⇒ bool **where**

wf-precondition $\text{pre} \longleftrightarrow (\forall a\ b. \text{pre } a\ b \longrightarrow (\text{pre } b\ a \wedge \text{pre } a\ a))$

lemma wf-precondition-rep-prec:

assumes wf-precondition pre

shows $\text{Rep-precondition } (\text{Abs-precondition } pre) = pre$
using $\text{Abs-precondition-inverse}[of\ pre]\ \text{assms mem-Collect-eq wf-precondition-def}[of\ pre]$
by *blast*

lemma *wf-indexed-precondition-rep-prec*:
assumes *wf-indexed-precondition pre*
shows $\text{Rep-indexed-precondition } (\text{Abs-indexed-precondition } pre) = pre$
using $\text{Abs-indexed-precondition-inverse}[of\ pre]\ \text{assms mem-Collect-eq wf-indexed-precondition-def}[of\ pre]$
by *blast*

definition *LowView where*

$\text{LowView } f\ A\ x = (\text{Exists } x\ (\text{And } (\text{View } f\ A\ (\lambda s. s\ x))\ (\text{LowExp } (\text{Evar } x))))$

lemma *LowViewE*:

assumes $(s, h), (s', h') \models \text{LowView } f\ A\ x$

and $x \notin \text{fv } A$

shows $(s, h), (s', h') \models A \wedge f\ (\text{normalize } (\text{get-fh } h)) = f\ (\text{normalize } (\text{get-fh } h'))$

proof –

obtain $v\ v'$ **where** $(s(x := v), h), (s'(x := v'), h') \models \text{And } (\text{View } f\ A\ (\lambda s. s\ x))\ (\text{LowExp } (\text{Evar } x))$

by (*metis LowView-def assms(1) hyper-sat.simps(7)*)

then obtain $(s(x := v), h), (s'(x := v'), h') \models \text{View } f\ A\ (\lambda s. s\ x)$

$(s(x := v), h), (s'(x := v'), h') \models \text{LowExp } (\text{Evar } x)$

using *hyper-sat.simps(3)* **by** *blast*

then obtain $(s(x := v), h), (s'(x := v'), h') \models A\ v = v'$

$f\ (\text{normalize } (\text{get-fh } h)) = f\ (\text{normalize } (\text{get-fh } h'))$

by *simp*

moreover have $(s, h), (s', h') \models A$

by (*meson agrees-same agrees-update assms(2) calculation(1) sat-comm-aux*)

ultimately show *?thesis* **by** *blast*

qed

lemma *LowViewI*:

assumes $(s, h), (s', h') \models A$

and $f\ (\text{normalize } (\text{get-fh } h)) = f\ (\text{normalize } (\text{get-fh } h'))$

and $x \notin \text{fv } A$

shows $(s, h), (s', h') \models \text{LowView } f\ A\ x$

proof –

let $?s = s(x := f\ (\text{normalize } (\text{get-fh } h)))$

let $?s' = s'(x := f\ (\text{normalize } (\text{get-fh } h')))$

have $(?s, h), (?s', h') \models A$

by (*meson agrees-same-aux agrees-update assms(1) assms(3) sat-comm-aux*)

then have $(?s, h), (?s', h') \models \text{And } (\text{View } f\ A\ (\lambda s. s\ x))\ (\text{LowExp } (\text{Evar } x))$

using *assms(2)* **by** *auto*

then show *?thesis using LowView-def*
by (*metis hyper-sat.simps(7)*)
qed

definition *disjoint* :: ('a set) ⇒ ('a set) ⇒ bool
where *disjoint* h1 h2 = (h1 ∩ h2 = {})

definition *unambiguous where*

unambiguous A x ⇔ (∀ s1 h1 s2 h2 v1 v2 v1' v2'. (s1(x := v1), h1), (s2(x := v2), h2) ⊨ A
 ∧ (s1(x := v1'), h1), (s2(x := v2'), h2) ⊨ A → v1 = v1' ∧ v2 = v2')

definition *all-axioms* :: ('v ⇒ 'w) ⇒ ('v ⇒ 'a ⇒ 'v) ⇒ ('a ⇒ 'a ⇒ bool) ⇒ ('i
 ⇒ 'v ⇒ 'b ⇒ 'v) ⇒ ('i ⇒ 'b ⇒ 'b ⇒ bool) ⇒ bool **where**
all-axioms α *sact spre uact upre* ⇔

— Every action's relational precondition is sufficient to preserve the low-ness of the abstract view of the resource value:

(∀ v v' sarg sarg'. α v = α v' ∧ *spre sarg sarg'* → α (*sact v sarg*) = α (*sact v' sarg'*)) ∧
 (∀ v v' uarg uarg' i. α v = α v' ∧ *upre i uarg uarg'* → α (*uact i v uarg*) = α (*uact i v' uarg'*)) ∧

(∀ sarg sarg'. *spre sarg sarg'* → *spre sarg' sarg'*) ∧
 (∀ uarg uarg' i. *upre i uarg uarg'* → *upre i uarg' uarg'*) ∧

— All relevant pairs of actions commute w.r.t. the abstract view:

(∀ v v' sarg sarg'. α v = α v' ∧ *spre sarg sarg* ∧ *spre sarg' sarg'* → α (*sact (sact v sarg) sarg'*) = α (*sact (sact v' sarg') sarg*)) ∧
 (∀ v v' sarg uarg i. α v = α v' ∧ *spre sarg sarg* ∧ *upre i uarg uarg* → α (*sact (uact i v uarg) sarg*) = α (*uact i (sact v' sarg) uarg*)) ∧
 (∀ v v' uarg uarg' i i'. i ≠ i' ∧ α v = α v' ∧ *upre i uarg uarg* ∧ *upre i' uarg' uarg'*
 → α (*uact i' (uact i v uarg) uarg'*) = α (*uact i (uact i' v' uarg') uarg*))

3.2 Rules of the Logic

inductive *CommCSL* :: ('i, 'a, nat) cont ⇒ ('i, 'a, nat) assertion ⇒ cmd ⇒ ('i, 'a, nat) assertion ⇒ bool

(- ⊢ {-} - {-} [51,0,0] 81) **where**

RuleSkip: Δ ⊢ {P} *Cskip* {P}

| *RuleAssign*: [[ΛΓ. Δ = Some Γ ⇒ x ∉ fvA (invariant Γ) ; collect-existentials P ∩ fvE E = {}] ⇒ Δ ⊢ {subA x E P} *Cassign* x E {P}

| *RuleNew*: [[x ∉ fvE E; ΛΓ. Δ = Some Γ ⇒ x ∉ fvA (invariant Γ) ∧ view-function-of-inv Γ] ⇒ Δ ⊢ {Emp} *Calloc* x E {PointsTo (Evar x) pwrite E}

| *RuleWrite*: [[ΛΓ. Δ = Some Γ ⇒ view-function-of-inv Γ ; v ∉ fvE loc] ⇒ Δ ⊢ {Exists v (PointsTo loc pwrite (Evar v))} *Cwrite* loc E {PointsTo loc pwrite E}

| [[ΛΓ. Δ = Some Γ ⇒ x ∉ fvA (invariant Γ) ∧ view-function-of-inv Γ ; x ∉ fvE

$E \cup \text{fv}E \ e \] \Longrightarrow$
 $\Delta \vdash \{ \text{PointsTo } E \ \pi \ e \} \ \text{Cread } x \ E \ \{ \text{And } (\text{PointsTo } E \ \pi \ e) \ (\text{Bool } (\text{Beq } (\text{Evar } x) \ e)) \}$
 $| \ \text{RuleShare}: \llbracket \Gamma = () \ \text{view} = f, \ \text{abstract-view} = \alpha, \ \text{saction} = \text{sact}, \ \text{uaction} = \text{uact}, \ \text{invariant} = J \ \rrbracket ; \ \text{all-axioms } \alpha \ \text{sact} \ \text{spre} \ \text{uact} \ \text{upre} ;$
 $\ \text{Some } \Gamma \vdash \{ \text{Star } P \ \text{EmptyFullGuards} \} \ C \ \{ \text{Star } Q \ (\text{And } (\text{PreSharedGuards } (\text{Abs-precondition} \ \text{spre})) \ (\text{PreUniqueGuards } (\text{Abs-indexed-precondition} \ \text{upre}))) \};$
 $\ \text{view-function-of-inv } \Gamma ; \ \text{unary } J ; \ \text{precise } J ; \ \text{wf-indexed-precondition} \ \text{upre} ;$
 $\ \text{wf-precondition} \ \text{spre} ; \ x \notin \text{fv}A \ J ;$
 $\ \text{no-guard-assertion } (\text{Star } P \ (\text{LowView } (\alpha \circ f) \ J \ x)) \] \Longrightarrow \ \text{None} \vdash \{ \text{Star } P \ (\text{LowView } (\alpha \circ f) \ J \ x) \} \ C \ \{ \text{Star } Q \ (\text{LowView } (\alpha \circ f) \ J \ x) \}$
 $| \ \text{RuleAtomicUnique}: \llbracket \Gamma = () \ \text{view} = f, \ \text{abstract-view} = \alpha, \ \text{saction} = \text{sact}, \ \text{uaction} = \text{uact}, \ \text{invariant} = J \ \rrbracket ;$
 $\ \text{no-guard-assertion } P ; \ \text{no-guard-assertion } Q ;$
 $\ \text{None} \vdash \{ \text{Star } P \ (\text{View } f \ J \ (\lambda s. \ s \ x)) \} \ C \ \{ \text{Star } Q \ (\text{View } f \ J \ (\lambda s. \ \text{uact } \text{index } (s \ x) \ (\text{map-to-arg } (s \ \text{uarg})))) \};$
 $\ \text{precise } J ; \ \text{unary } J ; \ \text{view-function-of-inv } \Gamma ; \ x \notin \text{fv}C \ C \cup \text{fv}A \ P \cup \text{fv}A \ Q \cup \text{fv}A \ J ; \ \text{uarg} \notin \text{fv}C \ C ;$
 $\ l \notin \text{fv}C \ C ; \ x \notin \text{fv}S \ (\lambda s. \ \text{map-to-list } (s \ l)) ; \ x \notin \text{fv}S \ (\lambda s. \ \text{map-to-arg } (s \ \text{uarg}) \ \# \ \text{map-to-list } (s \ l)) \]$
 $\Longrightarrow \ \text{Some } \Gamma \vdash \{ \text{Star } P \ (\text{UniqueGuard } \text{index } (\lambda s. \ \text{map-to-list } (s \ l))) \} \ \text{Catomic } C \ \{ \text{Star } Q \ (\text{UniqueGuard } \text{index } (\lambda s. \ \text{map-to-arg } (s \ \text{uarg}) \ \# \ \text{map-to-list } (s \ l))) \}$
 $| \ \text{RuleAtomicShared}: \llbracket \Gamma = () \ \text{view} = f, \ \text{abstract-view} = \alpha, \ \text{saction} = \text{sact}, \ \text{uaction} = \text{uact}, \ \text{invariant} = J \ \rrbracket ; \ \text{no-guard-assertion } P ; \ \text{no-guard-assertion } Q ;$
 $\ \text{None} \vdash \{ \text{Star } P \ (\text{View } f \ J \ (\lambda s. \ s \ x)) \} \ C \ \{ \text{Star } Q \ (\text{View } f \ J \ (\lambda s. \ \text{sact } (s \ x) \ (\text{map-to-arg } (s \ \text{sarg})))) \};$
 $\ \text{precise } J ; \ \text{unary } J ; \ \text{view-function-of-inv } \Gamma ; \ x \notin \text{fv}C \ C \cup \text{fv}A \ P \cup \text{fv}A \ Q \cup \text{fv}A \ J ; \ \text{sarg} \notin \text{fv}C \ C ;$
 $\ \text{ms} \notin \text{fv}C \ C ; \ x \notin \text{fv}S \ (\lambda s. \ \text{map-to-multiset } (s \ \text{ms})) ; \ x \notin \text{fv}S \ (\lambda s. \ \{ \# \ \text{map-to-arg } (s \ \text{sarg}) \ \# \} + \ \text{map-to-multiset } (s \ \text{ms})) \]$
 $\Longrightarrow \ \text{Some } \Gamma \vdash \{ \text{Star } P \ (\text{SharedGuard } \pi \ (\lambda s. \ \text{map-to-multiset } (s \ \text{ms}))) \} \ \text{Catomic } C \ \{ \text{Star } Q \ (\text{SharedGuard } \pi \ (\lambda s. \ \{ \# \ \text{map-to-arg } (s \ \text{sarg}) \ \# \} + \ \text{map-to-multiset } (s \ \text{ms}))) \}$
 $| \ \text{RulePar}: \llbracket \Delta \vdash \{ P1 \} \ C1 \ \{ Q1 \} ; \ \Delta \vdash \{ P2 \} \ C2 \ \{ Q2 \} ; \ \text{disjoint } (\text{fv}A \ P1 \cup \text{fv}C \ C1 \cup \text{fv}A \ Q1) \ (\text{wr}C \ C2) ;$
 $\ \text{disjoint } (\text{fv}A \ P2 \cup \text{fv}C \ C2 \cup \text{fv}A \ Q2) \ (\text{wr}C \ C1) ; \ \bigwedge \Gamma. \ \Delta = \text{Some } \Gamma \Longrightarrow \ \text{disjoint } (\text{fv}A \ (\text{invariant } \Gamma)) \ (\text{wr}C \ C2) ;$
 $\ \bigwedge \Gamma. \ \Delta = \text{Some } \Gamma \Longrightarrow \ \text{disjoint } (\text{fv}A \ (\text{invariant } \Gamma)) \ (\text{wr}C \ C1) ; \ \text{precise } P1 \ \vee \ \text{precise } P2 \]$
 $\Longrightarrow \ \Delta \vdash \{ \text{Star } P1 \ P2 \} \ \text{Cpar } C1 \ C2 \ \{ \text{Star } Q1 \ Q2 \}$
 $| \ \text{RuleIf1}: \llbracket \Delta \vdash \{ \text{And } P \ (\text{Bool } b) \} \ C1 \ \{ Q \} ; \ \Delta \vdash \{ \text{And } P \ (\text{Bool } (\text{Bnot } b)) \} \ C2 \ \{ Q \} \]$
 $\Longrightarrow \ \Delta \vdash \{ \text{And } P \ (\text{Low } b) \} \ \text{Cif } b \ C1 \ C2 \ \{ Q \}$
 $| \ \text{RuleIf2}: \llbracket \Delta \vdash \{ \text{And } P \ (\text{Bool } b) \} \ C1 \ \{ Q \} ; \ \Delta \vdash \{ \text{And } P \ (\text{Bool } (\text{Bnot } b)) \} \ C2 \ \{ Q \} ; \ \text{unary } Q \]$
 $\Longrightarrow \ \Delta \vdash \{ P \} \ \text{Cif } b \ C1 \ C2 \ \{ Q \}$
 $| \ \text{RuleSeq}: \llbracket \Delta \vdash \{ P \} \ C1 \ \{ R \} ; \ \Delta \vdash \{ R \} \ C2 \ \{ Q \} \] \Longrightarrow \ \Delta \vdash \{ P \} \ \text{Cseq } C1 \ C2 \ \{ Q \}$
 $| \ \text{RuleFrame}: \llbracket \Delta \vdash \{ P \} \ C \ \{ Q \} ; \ \text{disjoint } (\text{fv}A \ R) \ (\text{wr}C \ C) ; \ \text{precise } P \ \vee \ \text{precise } R \]$

$\implies \Delta \vdash \{Star\ P\ R\} C \{Star\ Q\ R\}$
| *RuleCons*: $\llbracket \Delta \vdash \{P'\} C \{Q'\} ; entails\ P\ P' ; entails\ Q'\ Q \rrbracket \implies \Delta \vdash \{P\} C \{Q\}$
| *RuleExists*: $\llbracket \Delta \vdash \{P\} C \{Q\} ; x \notin fvC\ C ; \wedge \Gamma. \Delta = Some\ \Gamma \implies x \notin fvA\ (invariant\ \Gamma) ; unambiguous\ P\ x \rrbracket$
 $\implies \Delta \vdash \{Exists\ x\ P\} C \{Exists\ x\ Q\}$
| *RuleWhile1*: $\Delta \vdash \{And\ I\ (Bool\ b)\} C \{And\ I\ (Low\ b)\} \implies \Delta \vdash \{And\ I\ (Low\ b)\} Cwhile\ b\ C \{And\ I\ (Bool\ (Bnot\ b))\}$
| *RuleWhile2*: $\llbracket unary\ I ; \Delta \vdash \{And\ I\ (Bool\ b)\} C \{I\} \rrbracket \implies \Delta \vdash \{I\} Cwhile\ b\ C \{And\ I\ (Bool\ (Bnot\ b))\}$

end

4 Soundness of CommCSL

4.1 Abstract Commutativity

theory *AbstractCommutativity*
imports *Main CommCSL HOL-Library.Multiset*
begin

datatype (*'i, 'a, 'b*) *action* = *Shared (get-s: 'a) | Unique (get-i: 'i) (get-u: 'b)*

We consider a family of unique actions indexed by *'i*

lemma *sabstract*:

assumes *all-axioms* α *sact spre uact upre*
shows $\alpha\ v = \alpha\ v' \wedge spre\ sarg\ sarg' \implies \alpha\ (sact\ v\ sarg) = \alpha\ (sact\ v'\ sarg')$
using *all-axioms-def*[of α *sact spre uact upre*] *assms* **by** *fast*

lemma *uabstract*:

assumes *all-axioms* α *sact spre uact upre*
shows $\alpha\ v = \alpha\ v' \wedge upre\ i\ uarg\ uarg' \implies \alpha\ (uact\ i\ v\ uarg) = \alpha\ (uact\ i\ v'\ uarg')$
using *all-axioms-def*[of α *sact spre uact upre*] *assms* **by** *fast*

lemma *spre-refl*:

assumes *all-axioms* α *sact spre uact upre*
shows *spre sarg sarg' \implies spre sarg' sarg'*
using *all-axioms-def*[of α *sact spre uact upre*] *assms* **by** *fast*

lemma *upre-refl*:

assumes *all-axioms* α *sact spre uact upre*
shows *upre i uarg uarg' \implies upre i uarg' uarg'*
using *all-axioms-def*[of α *sact spre uact upre*] *assms* **by** *fast*

lemma *ss-com*:

assumes *all-axioms* α *sact spre uact upre*
shows $\alpha\ v = \alpha\ v' \implies spre\ sarg\ sarg \wedge spre\ sarg'\ sarg' \implies \alpha\ (sact\ (sact\ v\ sarg)\ sarg') = \alpha\ (sact\ (sact\ v'\ sarg')\ sarg)$
using *all-axioms-def*[of α *sact spre uact upre*] *assms* **by** *blast*

lemma *su-com*:

assumes *all-axioms* α *sact spre uact upre*
shows $\alpha v = \alpha v' \implies spre\ sarg\ sarg \wedge upre\ i\ uarg\ uarg \implies \alpha (sact\ (uact\ i\ v\ uarg)\ sarg) = \alpha (uact\ i\ (sact\ v'\ sarg)\ uarg)$
using *all-axioms-def*[of α *sact spre uact upre*] *assms* **by** *blast*

lemma *uu-com*:

assumes *all-axioms* α *sact spre uact upre*
and $i \neq i'$
and $\alpha v = \alpha v'$
and *upre* $i'\ uarg'\ uarg'$
and *upre* $i\ uarg\ uarg$
shows $\alpha (uact\ i'\ (uact\ i\ v\ uarg)\ uarg') = \alpha (uact\ i\ (uact\ i'\ v'\ uarg')\ uarg)$

proof –

have $\bigwedge v\ v'\ uarg\ uarg'\ i\ i'. i \neq i' \wedge \alpha v = \alpha v' \wedge upre\ i\ uarg\ uarg \wedge upre\ i'\ uarg'\ uarg' \longrightarrow \alpha (uact\ i'\ (uact\ i\ v\ uarg)\ uarg') = \alpha (uact\ i\ (uact\ i'\ v'\ uarg')\ uarg)$
using *all-axioms-def*[of α *sact spre uact upre*] *assms*(1)
by *blast*
then show *?thesis*
using *assms*(2) *assms*(3) *assms*(4) *assms*(5) **by** *blast*

qed

definition *PRE-shared* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a\ multiset \Rightarrow 'a\ multiset \Rightarrow bool$
where

PRE-shared spre sargs sargs' $\longleftrightarrow (\exists ms.\ image\ mset\ fst\ ms = sargs \wedge image\ mset\ snd\ ms = sargs' \wedge (\forall x \in \# ms.\ spre\ (fst\ x)\ (snd\ x)))$

lemma *PRE-shared-same-size*:

assumes *PRE-shared spre sargs sargs'*
shows $size\ sargs = size\ sargs'$
proof –
obtain *ms* **where** $image\ mset\ fst\ ms = sargs \wedge image\ mset\ snd\ ms = sargs' \wedge (\forall x \in \# ms.\ spre\ (fst\ x)\ (snd\ x))$
by (*metis PRE-shared-def assms*)
then have $size\ sargs = size\ ms \wedge size\ ms = size\ sargs'$
by *force*
then show *?thesis*
by *simp*

qed

definition *is-Unique* :: $('i, 'a, 'b)\ action \Rightarrow bool$ **where**

is-Unique $a \longleftrightarrow \neg is\ Shared\ a$

definition *is-Unique-i* :: $'i \Rightarrow ('i, 'a, 'b)\ action \Rightarrow bool$ **where**

is-Unique-i $i\ a \longleftrightarrow is\ Unique\ a \wedge get\ i\ a = i$

definition *possible-sequence* :: 'a multiset \Rightarrow ('i \Rightarrow 'b list) \Rightarrow ('i, 'a, 'b) action list \Rightarrow bool **where**

possible-sequence sargs uargs s \longleftrightarrow ((\forall i. uargs i = map get-u (filter (is-Unique-i i) s)) \wedge sargs = image-mset get-s (filter-mset is-Shared (mset s)))

lemma *possible-sequenceI*:

assumes \bigwedge i. uargs i = map get-u (filter (is-Unique-i i) s)
and sargs = image-mset get-s (filter-mset is-Shared (mset s))
shows *possible-sequence* sargs uargs s
using *assms(1)* *assms(2)* *possible-sequence-def* **by** blast

fun *remove-at-index* :: nat \Rightarrow 'd list \Rightarrow 'd list **where**

remove-at-index - [] = []
| *remove-at-index* 0 (x # xs) = xs
| *remove-at-index* (Suc n) (x # xs) = x # (*remove-at-index* n xs)

lemma *remove-at-index*:

assumes $n < \text{length } l$
shows $\text{length } (\text{remove-at-index } n \ l) = \text{length } l - 1$
and $i \geq 0 \wedge i < n \implies \text{remove-at-index } n \ l ! i = l ! i$
and $i \geq n \wedge i < \text{length } l - 1 \implies \text{remove-at-index } n \ l ! i = l ! (i + 1)$
using *assms*

proof (*induct l arbitrary: n i*)

case (*Cons a l*)
{
case 1
then show ?*case*
proof (*cases n*)
case (*Suc k*)
then show ?*thesis*
using 1 *Cons.hyps(1)* **by** force
qed (*simp*)

next

case 2
then show ?*case*
proof (*cases n*)
case (*Suc k*)
then show ?*thesis*
using 2.*prems(1)* 2.*prems(2)* *Cons.hyps(2)* *Suc-less-eq2* *less-Suc-eq-0-disj*

by *auto*

qed (*simp*)

next

case 3
then show ?*case*
proof (*cases n*)
case (*Suc k*)
then have *remove-at-index* (*Suc k*) (a # l) ! i = (a # l) ! (i + 1)
apply (*cases i*)
using 3.*prems(1)* **apply** blast

```

    using 3.prem1 Cons.hyps(3) Suc-less-eq2 by auto
  then show remove-at-index n (a # l) ! i = (a # l) ! (i + 1)
    using Suc by blast
  qed (simp)
}
qed (auto)

```

```

fun insert-at :: nat ⇒ 'd ⇒ 'd list ⇒ 'd list where
  insert-at 0 x l = x # l
| insert-at - x [] = [x]
| insert-at (Suc n) x (h # xs) = h # (insert-at n x xs)

```

```

lemma insert-at-index:
  assumes n ≤ length l
  shows length (insert-at n x l) = length l + 1
    and i ≥ 0 ∧ i < n ⇒ insert-at n x l ! i = l ! i
    and n ≥ 0 ⇒ insert-at n x l ! n = x
    and i > n ∧ i < length l + 1 ⇒ insert-at n x l ! i = l ! (i - 1)
  using assms
proof (induct l arbitrary: n i)
  case (Cons a l)
  {
    case 1
    then show ?case by (cases n) (simp-all add: Cons.hyps(1))
  next
    case 2
    then show ?case apply (cases n)
      apply blast
      using Cons.hyps(2) less-Suc-eq-0-disj by force
  next
    case 3
    then show ?case apply (cases n)
      apply simp
      by (simp add: Cons.hyps(3))
  next
    case 4
    then show ?case apply (cases n)
      apply simp
      by (metis (no-types, lifting) 4.prem1 4.prem2 Cons.hyps(4) Nat.add-0-right
        One-nat-def Suc-le-length-iff Suc-less-eq Suc-pred add-Suc-right bot-nat-0.not-eq-extremum
        insert-at.simps(3) less-zeroE list.inject list.size(4) nat.simps(3) nth-Cons' nth-Cons-Suc)
  }
qed (simp-all)

```

```

lemma list-ext:
  assumes length a = length b
    and ∧i. i ≥ 0 ∧ i < length a ⇒ a ! i = b ! i
  shows a = b
by (meson assms(1) assms(2) bot-nat-0.extremum nth-equalityI)

```

```

lemma mset-remove-index:
  assumes  $i \geq 0 \wedge i < \text{length } l$ 
  shows  $mset\ l = mset\ (\text{remove-at-index } i\ l) + \{\#\ l!\ i\ \#\}$ 
  using assms
proof (induct l arbitrary: i)
  case (Cons a l)
  then show ?case
  proof (cases i)
    case (Suc k)
    then show ?thesis
      using Cons.hyps Cons.prem by force
    qed (auto)
  qed (simp)

lemma filter-remove:
  assumes  $k \geq 0 \wedge k < \text{length } s$ 
  and  $\neg P\ (s!\ k)$ 
  shows  $\text{filter } P\ (\text{remove-at-index } k\ s) = \text{filter } P\ s$ 
  using assms
proof (induct k arbitrary: s)
  case 0
  then have  $s = \text{hd } s \# \text{tl } s$ 
  by simp
  then show ?case
    by (metis 0.prem(2) filter.simps(2) nth-Cons-0 remove-at-index.simps(2))
  next
  case (Suc k)
  then show ?case
    by (cases s simp-all)
  qed

lemma exists-index-in-sequence-shared:
  assumes  $a \in \# \text{sargs}$ 
  and possible-sequence sargs uargs s
  shows  $\exists i. i \geq 0 \wedge i < \text{length } s \wedge s!\ i = \text{Shared } a \wedge \text{possible-sequence } (\text{sargs} - \{\# a \#\})\ uargs\ (\text{remove-at-index } i\ s)$ 
proof –
  have  $a \in \# \text{image-mset get-s } (\text{filter-mset is-Shared } (mset\ s))$ 
  by (metis assms(1) assms(2) possible-sequence-def)
  then have  $\text{Shared } a \in \text{set } s$ 
  by fastforce
  then obtain  $i$  where  $i \geq 0 \wedge i < \text{length } s \wedge s!\ i = \text{Shared } a$ 
  by (meson bot-nat-0.extremum in-set-conv-nth)
  let  $?s = \text{remove-at-index } i\ s$ 
  have  $\text{sargs} - \{\# a \#\} = \text{image-mset get-s } (\text{filter-mset is-Shared } (mset\ ?s))$ 
proof –
  have  $\text{sargs} = \text{image-mset get-s } (\text{filter-mset is-Shared } (mset\ s))$ 
  using possible-sequence-def assms(2) by blast

```

moreover have $mset\ s = mset\ ?s + \{\#\ Shared\ a\ \#\}$
by (*metis* $\langle 0 \leq i \wedge i < length\ s \wedge s ! i = Shared\ a \rangle\ mset\ remove\ index$)
ultimately show *?thesis*
by *simp*
qed
moreover have $\bigwedge i. uargs\ i = map\ get\ u\ (filter\ (is\ Unique\ i\ i)\ ?s)$
by (*metis* $\langle 0 \leq i \wedge i < length\ s \wedge s ! i = Shared\ a \rangle\ action.disc(1)\ assms(2)$)
filter-remove is-Unique-def is-Unique-i-def possible-sequence-def
ultimately show *?thesis*
using $\langle 0 \leq i \wedge i < length\ s \wedge s ! i = Shared\ a \rangle\ possible-sequence-def$ **by** *blast*
qed

lemma *head-possible-exists-first-unique*:
assumes $a = hd\ (uargs\ j)$
and $uargs\ j \neq []$
and *possible-sequence sargs uargs s*
shows $\exists i. i \geq 0 \wedge i < length\ s \wedge s ! i = Unique\ j\ a \wedge (\forall k. k \geq 0 \wedge k < i$
 $\longrightarrow \neg is\ Unique\ i\ j\ (s ! k))$
using *assms*
proof (*induct s arbitrary: sargs uargs*)
case *Nil*
then show *?case by (simp add: possible-sequence-def)*
next
case (*Cons x xs*)
then show $\exists i \geq 0. i < length\ (x \# xs) \wedge (x \# xs) ! i = Unique\ j\ a \wedge (\forall k. 0 \leq$
 $k \wedge k < i \longrightarrow \neg is\ Unique\ i\ j\ ((x \# xs) ! k))$
proof (*cases x*)
case (*Shared sarg*)
moreover have *possible-sequence (sargs - {# sarg #}) uargs xs*
proof (*rule possible-sequenceI*)
show $sargs - \{\#sarg\#\} = image\ mset\ get\ s\ (filter\ mset\ is\ Shared\ (mset\ xs))$
using *Cons.prem(3) action.disc(1) action.sel(1) add-mset-remove-trivial[of*
sarg]
calculation
filter-mset-add-mset image-mset-add-mset mset.simps(2) possible-sequence-def[of
sargs uargs x # xs]
by *auto*
fix i **show** $uargs\ i = map\ get\ u\ (filter\ (is\ Unique\ i\ i)\ xs)$
using *Cons.prem(3) action.disc(1) calculation filter-remove is-Unique-def*
is-Unique-i-def
le-numeral-extra(3) length-greater-0-conv list.discI nth-Cons-0 possi-
ble-sequence-def[of sargs uargs x # xs]
remove-at-index.simps(2)
by *metis*
qed
then obtain i **where** $i \geq 0 \wedge i < length\ xs \wedge xs ! i = Unique\ j\ a \wedge (\forall k. 0 \leq$
 $k \wedge k < i \longrightarrow \neg is\ Unique\ i\ j\ (xs ! k))$
using *Cons.hyps[of uargs] Cons.prem(1) Cons.prem(2) by auto*
moreover have $\bigwedge k. 0 \leq k \wedge k < i + 1 \longrightarrow \neg is\ Unique\ i\ j\ ((x \# xs) ! k)$

```

proof
  fix  $k$  assume  $0 \leq k \wedge k < i + 1$ 
  then show  $\neg \text{is-Unique-}i\ j\ ((x \# xs) ! k)$ 
    apply (cases  $k$ )
    apply (simp add: Shared is-Unique-def is-Unique-i-def)
    by (simp add: calculation(2))
qed
ultimately show ?thesis
by (metis Suc-eq-plus1 Suc-less-eq bot-nat-0.extremum length-Cons nth-Cons-Suc)
next
case (Unique  $k$  uarg)
then show ?thesis
proof (cases  $j = k$ )
  case True
  then have  $uargs\ j = \text{map}\ \text{get-}u\ (\text{filter}\ (\text{is-Unique-}i\ j)\ (x \# xs))$ 
    by (meson Cons.prem(3) possible-sequence-def)
  then have  $uarg = a$ 
    by (simp add: True Unique Cons.prem(1) is-Unique-def is-Unique-i-def)
  then show ?thesis
    using True Unique by fastforce
next
case False
moreover have  $\text{possible-sequence}\ \text{sargs}\ (uargs(k := \text{tl}\ (uargs\ k)))\ xs$ 
proof (rule possible-sequenceI)
  show  $\text{sargs} = \text{image-mset}\ \text{get-}s\ (\text{filter-mset}\ \text{is-Shared}\ (\text{mset}\ xs))$ 
    by (metis (mono-tags, lifting) Cons.prem(3) Unique action.disc(2))
  filter-mset-add-mset mset.simp(2) possible-sequence-def
  fix  $i$  show  $(uargs(k := \text{tl}\ (uargs\ k)))\ i = \text{map}\ \text{get-}u\ (\text{filter}\ (\text{is-Unique-}i\ i)$ 
xs)
proof (cases  $i = k$ )
  case True
  then show ?thesis
    using Cons.prem(3) Unique action.disc(2) action.sel(2) filter.simp(2)
fun-upd-same
     $\text{is-Unique-def is-Unique-i-def list.sel(3) map-tl possible-sequence-def[of}$ 
sargs uargs x # xs]
    by metis
next
case False
then show ?thesis
    using Cons.prem(3) Unique action.sel(2) filter-remove fun-upd-apply
is-Unique-i-def
     $\text{le-numeral-extra(3) length-greater-0-conv list.discI nth-Cons-0 possi-}$ 
ble-sequence-def[of sargs uargs x # xs]
     $\text{remove-at-index.simp(2)}$  by metis
qed
qed
then obtain  $i$  where  $i \geq 0 \wedge i < \text{length}\ xs \wedge xs ! i = \text{Unique}\ j\ a \wedge (\forall k. 0$ 
 $\leq k \wedge k < i \longrightarrow \neg \text{is-Unique-}i\ j\ (xs ! k))$ 

```

```

by (metis Cons.hyps Cons.prem1 Cons.prem2 calculation fun-upd-other)
moreover have  $\bigwedge k. 0 \leq k \wedge k < i + 1 \longrightarrow \neg \text{is-Unique-}i\ j\ ((x \# xs) ! k)$ 
proof
  fix k assume  $0 \leq k \wedge k < i + 1$ 
  then show  $\neg \text{is-Unique-}i\ j\ ((x \# xs) ! k)$ 
  apply (cases k)
  apply (metis False Unique action.sel(2) is-Unique-i-def nth-Cons-0)
  by (simp add: calculation(2))
qed
ultimately show ?thesis
by (metis Suc-eq-plus1 Suc-less-eq bot-nat-0.extremum length-Cons nth-Cons-Suc)
qed
qed
qed

```

lemma *remove-at-index-filter:*

```

assumes  $i \geq 0 \wedge i < \text{length } s \wedge P\ (s ! i)$ 
and  $\bigwedge j. j \geq 0 \wedge j < i \implies \neg P\ (s ! j)$ 
shows  $\text{tl } (\text{map get-u } (\text{filter } P\ s)) = \text{map get-u } (\text{filter } P\ (\text{remove-at-index } i\ s))$ 
using assms
proof (induct s arbitrary: i)
  case (Cons a s)
  then show ?case
  proof (cases i)
    case 0
    then show ?thesis
    using Cons.prem1 by auto
  next
  case (Suc k)
  then have  $\text{tl } (\text{map get-u } (\text{filter } P\ s)) = \text{map get-u } (\text{filter } P\ (\text{remove-at-index } k\ s))$ 
  apply (cases s)
  apply simp
  by (metis Cons.hyps Cons.prem1 Cons.prem2 Suc-less-eq bot-nat-0.extremum length-Cons nth-Cons-Suc)
  then show ?thesis
  by (metis Cons.prem2 Suc bot-nat-0.extremum filter.simps(2) nth-Cons-0 remove-at-index.simps(3) zero-less-Suc)
  qed
qed (simp)

```

definition *tail-kth where*

tail-kth uargs k = uargs(k := tl (uargs k))

lemma *exists-index-in-sequence-unique:*

```

assumes  $a = \text{hd } (\text{uargs } k)$ 
and  $\text{uargs } k \neq []$ 
and possible-sequence sargs uargs s
shows  $\exists i. i \geq 0 \wedge i < \text{length } s \wedge s ! i = \text{Unique } k\ a \wedge \text{possible-sequence } sargs$ 

```

$(tail-kth\ uargs\ k)\ (remove-at-index\ i\ s)$
 $\wedge (\forall j. j \geq 0 \wedge j < i \longrightarrow \neg is-Unique-i\ k\ (s!\ j))$

proof –

obtain i **where** $i \geq 0 \wedge i < length\ s \wedge s!\ i = Unique\ k\ a \wedge (\forall j. j \geq 0 \wedge j < i \longrightarrow \neg is-Unique-i\ k\ (s!\ j))$
by $(metis\ assms(1)\ assms(2)\ assms(3)\ head-possible-exists-first-unique)$
let $?s = remove-at-index\ i\ s$
have $sargs = image-mset\ get-s\ (filter-mset\ is-Shared\ (mset\ ?s))$
by $(metis\ \langle 0 \leq i \wedge i < length\ s \wedge s!\ i = Unique\ k\ a \wedge (\forall j. 0 \leq j \wedge j < i \longrightarrow \neg is-Unique-i\ k\ (s!\ j)) \rangle\ action.disc(2)\ add.right-neutral\ assms(3)\ filter-single-mset\ filter-union-mset\ mset-remove-index\ possible-sequence-def)$
moreover have $tl\ (uargs\ k) = map\ get-u\ (filter\ (is-Unique-i\ k)\ ?s)$

proof –

have $uargs\ k = map\ get-u\ (filter\ (is-Unique-i\ k)\ s)$
by $(meson\ assms(3)\ possible-sequence-def)$
then show $?thesis$
by $(metis\ \langle 0 \leq i \wedge i < length\ s \wedge s!\ i = Unique\ k\ a \wedge (\forall j. 0 \leq j \wedge j < i \longrightarrow \neg is-Unique-i\ k\ (s!\ j)) \rangle\ action.disc(2)\ action.sel(2)\ is-Unique-def\ is-Unique-i-def\ remove-at-index-filter)$

qed

moreover have $possible-sequence\ sargs\ (tail-kth\ uargs\ k)\ (remove-at-index\ i\ s)$
proof $(rule\ possible-sequenceI)$
show $sargs = image-mset\ get-s\ (filter-mset\ is-Shared\ (mset\ (remove-at-index\ i\ s)))$
by $(simp\ add:\ calculation(1))$
fix ia **show** $tail-kth\ uargs\ k\ ia = map\ get-u\ (filter\ (is-Unique-i\ ia)\ (remove-at-index\ i\ s))$
by $(metis\ (mono-tags,\ lifting)\ \langle 0 \leq i \wedge i < length\ s \wedge s!\ i = Unique\ k\ a \wedge (\forall j. 0 \leq j \wedge j < i \longrightarrow \neg is-Unique-i\ k\ (s!\ j)) \rangle\ action.sel(2)\ assms(3)\ calculation(2)\ filter-remove\ fun-upd-other\ fun-upd-same\ is-Unique-i-def\ possible-sequence-def\ tail-kth-def)$

qed

ultimately show $?thesis$
using $\langle 0 \leq i \wedge i < length\ s \wedge s!\ i = Unique\ k\ a \wedge (\forall j. 0 \leq j \wedge j < i \longrightarrow \neg is-Unique-i\ k\ (s!\ j)) \rangle$ **by** $blast$

qed

lemma $possible-sequence-where-is-unique:$

assumes $possible-sequence\ sargs\ uargs\ (Unique\ k\ a\ \# s)$
shows $a = hd\ (uargs\ k)$

proof –

let $?s = Unique\ k\ a\ \# s$
have $Unique\ k\ a = hd\ (filter\ is-Unique\ ?s)$
by $(simp\ add:\ is-Unique-def)$
then have $a = hd\ (map\ get-u\ (filter\ is-Unique\ ?s))$
by $(simp\ add:\ is-Unique-def)$
then show $?thesis$
using $action.disc(2)\ action.sel(2)\ assms\ filter.simps(2)\ hd-map\ is-Unique-def\ is-Unique-i-def\ list.discI\ list.sel(1)\ possible-sequence-def[of\ sargs\ uargs\ Unique\ k\ a\ \# s]$

by metis
qed

lemma possible-sequence-where-is-shared:

assumes possible-sequence sargs uargs (Shared a # s)
shows $a \in \# \text{ sargs}$

proof –

let ?s = Shared a # s

have $a \in \text{set} (\text{map } \text{get-s} (\text{filter } \text{is-Shared } ?s))$

by simp

then show ?thesis

by (metis (no-types, lifting) assms mset-filter mset-map possible-sequence-def set-mset-mset)

qed

lemma PRE-unique-tII:

assumes PRE-unique upre qa qb

and upre ta tb

shows PRE-unique upre (ta # qa) (tb # qb)

proof (rule PRE-uniqueI)

show $\text{length} (ta \# qa) = \text{length} (tb \# qb)$

using PRE-unique-def assms(1) by auto

fix i

show $0 \leq i \wedge i < \text{length} (tb \# qb) \implies \text{upre} ((ta \# qa) ! i) ((tb \# qb) ! i)$

proof (cases i)

case 0

then show ?thesis

using assms(2) by auto

next

case (Suc k)

assume $0 \leq i \wedge i < \text{length} (tb \# qb)$

then have $(ta \# qa) ! i = qa ! k \wedge (tb \# qb) ! i = qb ! k$

by (simp add: Suc)

then show ?thesis using assms(1) PRE-unique-def

using Suc $\langle 0 \leq i \wedge i < \text{length} (tb \# qb) \rangle$ by auto

qed

qed

fun abstract-pre :: $('a \Rightarrow 'a \Rightarrow \text{bool}) \Rightarrow ('i \Rightarrow 'b \Rightarrow 'b \Rightarrow \text{bool}) \Rightarrow ('i, 'a, 'b) \text{ action} \Rightarrow ('i, 'a, 'b) \text{ action} \Rightarrow \text{bool}$ **where**

abstract-pre spre upre (Shared sarg) (Shared sarg') \longleftrightarrow spre sarg sarg'

| abstract-pre spre upre (Unique k uarg) (Unique k' uarg') \longleftrightarrow $k = k' \wedge \text{upre } k \text{ uarg uarg}'$

| abstract-pre spre upre - - \longleftrightarrow False

definition PRE-sequence :: $('a \Rightarrow 'a \Rightarrow \text{bool}) \Rightarrow ('i \Rightarrow 'b \Rightarrow 'b \Rightarrow \text{bool}) \Rightarrow ('i, 'a, 'b) \text{ action list} \Rightarrow ('i, 'a, 'b) \text{ action list} \Rightarrow \text{bool}$ **where**

PRE-sequence spre upre s s' \longleftrightarrow $\text{length } s = \text{length } s' \wedge (\forall i. i \geq 0 \wedge i < \text{length}$

$s \longrightarrow \text{abstract-pre spre upre } (s ! i) (s' ! i)$

lemma *PRE-sequenceE*:

assumes *PRE-sequence spre upre s s'*
and $i \geq 0 \wedge i < \text{length } s$
shows *abstract-pre spre upre (s ! i) (s' ! i)*
using *PRE-sequence-def assms(1) assms(2)* **by** *blast*

lemma *PRE-sequenceI*:

assumes $\text{length } s = \text{length } s'$
and $\bigwedge i. i \geq 0 \wedge i < \text{length } s \implies \text{abstract-pre spre upre } (s ! i) (s' ! i)$
shows *PRE-sequence spre upre s s'*
by (*simp add: PRE-sequence-def assms(1) assms(2)*)

lemma *PRE-sequenceI-rec*:

assumes *PRE-sequence spre upre s s'*
and *abstract-pre spre upre a b*
shows *PRE-sequence spre upre (a # s) (b # s')*
using *PRE-sequence-def[of spre upre a # s b # s'] PRE-sequence-def[of spre upre s s']*
assms(1) assms(2) less-Suc-eq-0-disj length-Cons less-Suc-eq-le nth-Cons-0 nth-Cons-Suc
by *force*

lemma *PRE-sequenceE-rec*:

assumes *PRE-sequence spre upre (a # s) (b # s')*
shows *PRE-sequence spre upre s s'*
and *abstract-pre spre upre a b*
using *PRE-sequence-def[of spre upre a # s b # s'] PRE-sequence-def[of spre upre s s']*
apply (*metis Suc-less-eq assms bot-nat-0.extremum diff-Suc-1 length-Cons nth-Cons-Suc*)
by (*metis PRE-sequenceE assms length-Cons list.size(3) nth-Cons-0 remdups-adj.simps(1) remdups-adj-length zero-less-Suc*)

fun *compute* :: $('v \Rightarrow 'a \Rightarrow 'v) \Rightarrow ('i \Rightarrow 'v \Rightarrow 'b \Rightarrow 'v) \Rightarrow 'v \Rightarrow ('i, 'a, 'b) \text{ action list} \Rightarrow 'v$ **where**

compute sact uact v0 [] = v0
| *compute sact uact v0 (Shared sarg # s) = sact (compute sact uact v0 s) sarg*
| *compute sact uact v0 (Unique k uarg # s) = uact k (compute sact uact v0 s) uarg*

lemma *obtain-other-elem-ms*:

assumes *PRE-shared spre sargs sargs'*
and $sarg \in \# sargs$
shows $\exists sarg'. sarg' \in \# sargs' \wedge spre sarg sarg' \wedge PRE-shared spre (sargs - \{\# sarg \# \}) (sargs' - \{\# sarg' \# \})$
proof –
obtain *ms where asm: image-mset fst ms = sargs \wedge image-mset snd ms = sargs'*
 $\wedge (\forall x \in \# ms. spre (fst x) (snd x))$
using *PRE-shared-def assms(1)* **by** *blast*

then obtain x **where** $x \in \# ms$ $fst\ x = sarg$
using $assms(2)$ **by** $auto$
then have $snd\ x \in \# sargs' \wedge spre\ sarg\ (snd\ x)$
using asm **by** $force$
moreover have $PRE\text{-shared}\ spre\ (sargs - \{\# sarg\ \})\ (sargs' - \{\# snd\ x\ \})$
proof $-$
let $?ms = ms - \{\# x\ \}$
have $image\text{-mset}\ fst\ ?ms = (sargs - \{\# sarg\ \}) \wedge image\text{-mset}\ snd\ ?ms =$
 $(sargs' - \{\# snd\ x\ \})$
by $(simp\ add: \langle fst\ x = sarg \rangle \langle x \in \# ms \rangle asm\ image\text{-mset}\text{-Diff})$
moreover have $\bigwedge y. y \in \# ?ms \implies spre\ (fst\ y)\ (snd\ y)$
by $(meson\ asm\ in\text{-diff}D)$
ultimately show $?thesis$
using $PRE\text{-shared}\text{-def}$ **by** $blast$
qed
ultimately show $?thesis$
by $blast$
qed

lemma $exists\text{-aligned}\text{-sequence}$:

assumes $possible\text{-sequence}\ sargs\ uargs\ s$
and $possible\text{-sequence}\ sargs'\ uargs'\ s'$

and $PRE\text{-shared}\ spre\ sargs\ sargs'$
and $\bigwedge k. PRE\text{-unique}\ (upre\ k)\ (uargs\ k)\ (uargs'\ k)$

shows $\exists s''. possible\text{-sequence}\ sargs'\ uargs'\ s'' \wedge PRE\text{-sequence}\ spre\ upre\ s\ s''$
using $assms$
proof $(induct\ s\ arbitrary: sargs\ uargs\ sargs'\ uargs'\ s')$
case Nil
then have $sargs = mset\ [] \wedge (\forall k. uargs\ k = [])$
by $(simp\ add: possible\text{-sequence}\text{-def})$
then have $sargs' = \{\#\} \wedge (\forall k. uargs'\ k = [])$
by $(metis\ Nil.prem(3)\ Nil.prem(4)\ PRE\text{-shared}\text{-same}\text{-size}\ PRE\text{-unique}\text{-def}\ length\text{-0}\text{-conv}\ mset.simps(1)\ size\text{-eq}\text{-0}\text{-iff}\text{-empty})$
then show $\exists s''. possible\text{-sequence}\ sargs'\ uargs'\ s'' \wedge PRE\text{-sequence}\ spre\ upre\ []\ s''$
by $(simp\ add: PRE\text{-sequence}\text{-def}\ possible\text{-sequence}\text{-def})$
next
case $(Cons\ act\ s)$

show $\exists s''. possible\text{-sequence}\ sargs'\ uargs'\ s'' \wedge PRE\text{-sequence}\ spre\ upre\ (act\ \#\ s)\ s''$
proof $(cases\ act)$
case $(Shared\ sarg)$

then have $sarg \in \# sargs$
by $(metis\ Cons.prem(1)\ possible\text{-sequence}\text{-where}\text{-is}\text{-shared})$
then obtain $sarg'$ **where** $sarg' \in \# sargs'\ spre\ sarg\ sarg'\ PRE\text{-shared}\ spre$

```

(sargs - {# sarg #}) (sargs' - {# sarg' #})
  by (metis Cons.prem3 obtain-other-elem-ms)
let ?sargs = sargs - {# sarg #}
let ?sargs' = sargs' - {# sarg' #}
have possible-sequence ?sargs uargs s
proof (rule possible-sequenceI)
  show  $\bigwedge i. \text{uargs } i = \text{map } \text{get-u } (\text{filter } (\text{is-Unique-i } i) s)$ 
    using Cons.prem1 Shared action.disc(1) filter.simps(2) is-Unique-def
is-Unique-i-def
  possible-sequence-def[of sargs uargs act # s]
  by metis
  have sargs = image-mset get-s (filter-mset is-Shared (mset(Shared sarg # s)))
    using Cons.prem1 Shared possible-sequence-def by blast
  then show sargs - {#sarg#} = image-mset get-s (filter-mset is-Shared (mset
s)) by simp
qed

obtain i where  $i \geq 0 \wedge i < \text{length } s' \wedge s' ! i = \text{Shared sarg}' \wedge \text{possible-sequence } ?sargs' \text{ uargs}'$ 
(remove-at-index i s')
  by (meson Cons.prem2) <sarg'  $\in$  # sargs'> exists-index-in-sequence-shared

then obtain s'' where possible-sequence ?sargs' uargs' s''  $\wedge$  PRE-sequence spre upre s s''
  using Cons.hyps Cons.prem4 <PRE-shared spre (sargs - {#sarg#}) (sargs' - {#sarg'#})> <possible-sequence (sargs - {#sarg'#}) uargs s> by blast

let ?s'' = Shared sarg' # s''

have possible-sequence sargs' uargs' ?s''
proof (rule possible-sequenceI)
  show  $\bigwedge i. \text{uargs}' i = \text{map } \text{get-u } (\text{filter } (\text{is-Unique-i } i) (\text{Shared sarg}' \# s''))$ 
    by (metis <possible-sequence (sargs' - {#sarg'#}) uargs' s''  $\wedge$  PRE-sequence spre upre s s''> action.disc(1) filter.simps(2) is-Unique-def is-Unique-i-def possible-sequence-def)
  show sargs' = image-mset get-s (filter-mset is-Shared (mset (Shared sarg' # s'')))
    using <possible-sequence (sargs' - {#sarg'#}) uargs' s''  $\wedge$  PRE-sequence spre upre s s''> <sarg'  $\in$  # sargs'>
    insert-DiffM[of sarg' sargs'] possible-sequence-def[of sargs' - {#sarg'#} uargs' s'']
    action.disc(1) action.sel(1) filter-mset-add-mset msed-map-invL mset.simps(2)
    by auto
qed

moreover have PRE-sequence spre upre (act # s) ?s''
  by (simp add: PRE-sequenceI-rec Shared <possible-sequence (sargs' - {#sarg'#}) uargs' s''  $\wedge$  PRE-sequence spre upre s s''> <spre sarg sarg'>)

ultimately show  $\exists s''. \text{possible-sequence } sargs' \text{ uargs}' s'' \wedge \text{PRE-sequence spre}$ 

```

```

upre (act # s) s'' by auto
next
  case (Unique k uarg)
  then have hd (uargs k) = uarg
    by (metis Cons.prem(1) possible-sequence-where-is-unique)
  moreover have uargs k ≠ []
    by (metis (no-types, lifting) Cons.prem(1) Unique action.disc(2) action.sel(2)
dropWhile-eq-Cons-conv dropWhile-eq-self-iff filter.simps(2) is-Unique-def is-Unique-i-def
list.map-disc-iff possible-sequence-def)
  ultimately have uargs k = uarg # tl (uargs k)
    by fastforce
  moreover have uargs' k = hd (uargs' k) # tl (uargs' k)
    by (metis Cons.prem(4) PRE-unique-def calculation length-Cons list.exhaust-sel
list.size(3) nat.simps(3))
  ultimately have upre k uarg (hd (uargs' k))
    by (metis Cons.prem(4) PRE-unique-def length-greater-0-conv list.simps(3)
list.size(3) nth-Cons-0 remdups-adj.simps(1) remdups-adj-length)
  moreover have PRE-unique (upre k) (tl (uargs k)) (tl (uargs' k))
    by (metis Cons.prem(4) PRE-unique-implies-tl ⟨uargs k = uarg # tl (uargs
k)⟩ ⟨uargs' k = hd (uargs' k) # tl (uargs' k)⟩)
  moreover have possible-sequence sargs (tail-kth uargs k) s
  proof (rule possible-sequenceI)
    show  $\bigwedge i. \text{tail-kth } uargs \ k \ i = \text{map } \text{get-u } (\text{filter } (\text{is-Unique-i } i) \ s)$ 
  proof -
    fix i
    obtain ii where asms:  $ii \geq 0 \wedge ii < \text{length } (act \ \# \ s) \wedge$ 
       $(act \ \# \ s) ! \ ii = \text{Unique } k \ uarg \wedge$ 
       $\text{possible-sequence } sargs \ (\text{tail-kth } uargs \ k) \ (\text{remove-at-index } ii \ (act \ \# \ s)) \wedge$ 
       $(\forall j. 0 \leq j \wedge j < ii \longrightarrow \neg \text{is-Unique-i } k \ ((act \ \# \ s) ! \ j))$ 
    by (metis Cons.prem(1) ⟨hd (uargs k) = uarg⟩ ⟨uargs k ≠ []⟩ ex-
ists-index-in-sequence-unique)
    then show tail-kth uargs k i = map get-u (filter (is-Unique-i i) s)
    by (metis Unique action.disc(2) action.sel(2) bot-nat-0.extremum bot-nat-0.not-eq-extremum
is-Unique-def is-Unique-i-def nth-Cons-0 possible-sequence-def remove-at-index.simps(2))
  qed
  show sargs = image-mset get-s (filter-mset is-Shared (mset s))
    by (metis (mono-tags, lifting) Cons.prem(1) Unique action.disc(2) fil-
ter-mset-add-mset mset.simps(2) possible-sequence-def)
  qed
  let ?uarg' = hd (uargs' k)

  obtain i where  $i \geq 0 \wedge i < \text{length } s' \wedge s' ! \ i = \text{Unique } k \ ?uarg' \wedge \text{possi-}$ 
     $\text{ble-sequence } sargs' \ (\text{tail-kth } uargs' \ k) \ (\text{remove-at-index } i \ s')$ 
  by (metis Cons.prem(2) ⟨uargs' k = hd (uargs' k) # tl (uargs' k)⟩ ex-
ists-index-in-sequence-unique list.discI)
  then obtain s'' where possible-sequence sargs' (tail-kth uargs' k) s''  $\wedge$  PRE-sequence
spre upre s s''
  using Cons.hyps[of sargs tail-kth uargs k sargs' tail-kth uargs' k] Cons.prem(3)
  ⟨possible-sequence sargs (tail-kth uargs k) s⟩ calculation(2)

```

*Cons.prem*s(4) *fun-upd-other fun-upd-same tail-kth-def*
by metis

let $?s'' = \text{Unique } k \text{ (hd (uargs' k)) \# } s''$
have *PRE-sequence spre upre (act \# s) ?s''*
by (*simp add: PRE-sequenceI-rec Unique <possible-sequence sargs' (tail-kth uargs' k) s'' \wedge PRE-sequence spre upre s s''> calculation(1)*)
moreover have *possible-sequence sargs' uargs' ?s''*
proof (*rule possible-sequenceI*)
show $\bigwedge i. \text{uargs' } i = \text{map get-u (filter (is-Unique-i i) (Unique k (hd (uargs' k)) \# s''))}$
proof –
fix i
obtain ii **where** *ii-def: $ii \geq 0 \wedge ii < \text{length } s' \wedge s' ! ii = \text{Unique } k \text{ (hd (uargs' k))} \wedge \text{possible-sequence sargs' (tail-kth uargs' k) (remove-at-index ii s')} \wedge (\forall j. 0 \leq j \wedge j < ii \longrightarrow \neg \text{is-Unique-i } k \text{ (s' ! j)})$*
by (*metis Cons.prem*s(2) $\langle \text{uargs' } k = \text{hd (uargs' k)} \# \text{tl (uargs' k)} \rangle$ *exists-index-in-sequence-unique list.discI*)
then show $\text{uargs' } i = \text{map get-u (filter (is-Unique-i i) (Unique k (hd (uargs' k)) \# s''))}$
using *filter-remove[of ii s' is-Unique-i i] remove-at-index-filter[of ii s' is-Unique-i i]*
*Cons.prem*s(2) $\langle \text{possible-sequence sargs' (tail-kth uargs' k) s''} \wedge \text{PRE-sequence spre upre s s''} \rangle$
 $\langle \text{uargs' } k = \text{hd (uargs' k)} \# \text{tl (uargs' k)} \rangle$ *action.sel(2) action.sel(3)*
filter.simps(2)[of is-Unique-i i Unique k (hd (uargs' k)) s'']
is-Unique-i-def list.simps(9)[of get-u]
possible-sequence-def[of sargs' tail-kth uargs' k remove-at-index ii s']
possible-sequence-def[of sargs' uargs' s']
possible-sequence-def[of sargs' tail-kth uargs' k s''] ii-def
by metis

qed
show $\text{sargs' } = \text{image-mset get-s (filter-mset is-Shared (mset (Unique k (hd (uargs' k)) \# s'')))}$
using $\langle \text{possible-sequence sargs' (tail-kth uargs' k) s''} \wedge \text{PRE-sequence spre upre s s''} \rangle$ *possible-sequence-def by auto*

qed
ultimately show $\exists s''. \text{possible-sequence sargs' uargs' s''} \wedge \text{PRE-sequence spre upre (act \# s) s''}$ **by blast**

qed
qed

lemma *insert-remove-same-list:*
assumes $k \geq 0 \wedge k < \text{length } s$
and $s ! k = x$
shows $s = \text{insert-at } k \ x \text{ (remove-at-index } k \ s)$
proof (*rule list-ext*)
show $\text{length } s = \text{length (insert-at } k \ x \text{ (remove-at-index } k \ s))$

```

    by (metis One-nat-def Suc-pred add.commute assms(1) insert-at-index(1) length-greater-0-conv
less-Suc-eq-le linorder-not-le list.size(3) plus-1-eq-Suc remove-at-index(1))
  fix i assume asm0:  $0 \leq i \wedge i < \text{length } s$ 
  show  $s ! i = \text{insert-at } k \ x \ (\text{remove-at-index } k \ s) ! i$ 
  proof (cases  $i < k$ )
    case True
      then show ?thesis
        by (metis (no-types, lifting) One-nat-def Suc-pred asm0 assms(1) insert-at-index(2)
less-Suc-eq-le order-le-less-trans remove-at-index(1) remove-at-index(2))
    next
      case False
        then have  $i \geq k$  by simp
        then show ?thesis
          proof (cases  $i = k$ )
            case True
              then show ?thesis
                by (metis (no-types, lifting) One-nat-def Suc-pred assms(1) assms(2) in-
sert-at-index(3) less-Suc-eq-le order-le-less-trans remove-at-index(1))
            next
              case False
                then have  $i > k$ 
                  using  $\langle k \leq i \rangle$  nless-le by blast
                then show  $s ! i = \text{insert-at } k \ x \ (\text{remove-at-index } k \ s) ! i$ 
                  apply (cases i)
                  apply blast
                  using Groups.add-ac(2) One-nat-def Suc-less-eq Suc-pred asm0 assms(1)
insert-at-index(4)[of k - i x]
less-Suc-eq-le order-le-less-trans plus-1-eq-Suc remove-at-index(1)[of k s]
remove-at-index(3)[of k s ]
                  by fastforce
          qed
        qed
      qed
    qed
  qed

```

lemma *swap-works*:

```

  assumes  $\text{length } s = \text{length } s'$ 
    and  $k < \text{length } s - 1$ 
    and  $\bigwedge i. i \geq 0 \wedge i < \text{length } s \wedge i \neq k \wedge i \neq k + 1 \implies s ! i = s' ! i$ 
    and  $s ! k = s' ! (k + 1)$ 
    and  $s' ! k = s ! (k + 1)$ 
    and PRE-sequence spre upre s s
    and  $\alpha v0 = \alpha v0'$ 
    and  $\neg (\exists k'. \text{is-Unique-}i \ k' \ (s ! k) \wedge \text{is-Unique-}i \ k' \ (s' ! (k + 1)))$ 
    and all-axioms  $\alpha$  sact spre uact upre
  shows  $\alpha (\text{compute sact uact } v0 \ s) = \alpha (\text{compute sact uact } v0' \ s')$  (is ?A = ?B)
  using assms
  proof (induct k arbitrary: s s')
    case 0
      then obtain  $x1 \ x2 \ xs$  where  $s = x1 \ \# \ x2 \ \# \ xs$ 

```

by (*metis Suc-length-conv Suc-pred add-0 le-add-diff-inverse less-diff-conv less-imp-le-nat plus-1-eq-Suc*)
then have $hd\ s' = x2$
by (*metis 0.premis(1) 0.premis(2) 0.premis(5) One-nat-def add-0 hd-conv-nth length-greater-0-conv length-tl list.sel(2) nth-Cons-0 nth-Cons-Suc*)
moreover have $hd\ (tl\ s') = x1$
by (*metis 0.premis(1) 0.premis(2) 0.premis(4) Suc-eq-plus1 ‹s = x1 # x2 # xs› hd-conv-nth length-greater-0-conv length-tl nth-Cons-0 nth-tl*)
ultimately obtain xs' **where** $s' = x2 \# x1 \# xs'$
by (*metis 0.premis(1) 0.premis(2) length-greater-0-conv length-tl list.collapse list.sel(2)*)
moreover have $xs = xs'$
proof (*rule list-ext*)
show $length\ xs = length\ xs'$
using *0.premis(1) ‹s = x1 # x2 # xs› calculation by auto*
fix i **assume** $0 \leq i \wedge i < length\ xs$
then show $xs\ !\ i = xs'\ !\ i$
by (*metis 0.premis(3) Suc-eq-plus1 Suc-less-eq ‹s = x1 # x2 # xs› bot-nat-0.extremum calculation diff-Suc-1 length-Cons nat.simps(3) nth-Cons-Suc*)
qed
have *PRE-sequence spre upre xs xs*
apply (*cases x1*) **apply** (*cases x2*)
using *0.premis(6) ‹s = x1 # x2 # xs› PRE-sequenceE-rec(1) by blast+*
then have $\alpha\ (compute\ sact\ uact\ v0\ xs) = \alpha\ (compute\ sact\ uact\ v0'\ xs)$
using *assms(7)*
proof (*induct xs*)
case *Nil*
then show *?case by simp*
next
case (*Cons a xs*)
then have $\alpha\ (compute\ sact\ uact\ v0\ xs) = \alpha\ (compute\ sact\ uact\ v0'\ xs)$
using *PRE-sequenceE-rec(1) by blast*
then show $\alpha\ (compute\ sact\ uact\ v0\ (a \# xs)) = \alpha\ (compute\ sact\ uact\ v0'\ (a \# xs))$
proof (*cases a*)
case (*Shared x1*)
then show *?thesis*
by (*metis ‹all-axioms ‹sact spre uact upre› Cons.premis(1) PRE-sequenceE-rec(2) ‹‹sact uact v0 xs› = ‹sact uact v0' xs›› abstract-pre.simps(1) compute.simps(2) sabstract*)
next
case (*Unique x2*)
then show *?thesis*
by (*metis ‹all-axioms ‹sact spre uact upre› Cons.premis(1) PRE-sequenceE-rec(2) ‹‹sact uact v0 xs› = ‹sact uact v0' xs›› abstract-pre.simps(2) compute.simps(3) uabstract*)
qed
qed
then show *?case*

```

proof (cases x1)
  case (Shared sarg1)
  then have x1 = Shared sarg1 by simp
  then show ?thesis
  proof (cases x2)
    case (Shared sarg2)
    then show  $\alpha$  (compute sact uact v0 s) =  $\alpha$  (compute sact uact v0' s')
    using ⟨all-axioms  $\alpha$  sact spre uact upre⟩ 0.premis(2) 0.premis(5) 0.premis(6)
One-nat-def
  PRE-sequenceE-rec(2)[of spre upre x1 x2 # xs x1 x2 # xs]
  PRE-sequence-def[of spre upre s s] Suc-eq-plus1
  ⟨ $\alpha$  (compute sact uact v0 xs) =  $\alpha$  (compute sact uact v0' xs)⟩ ⟨s = x1 #
x2 # xs⟩
  ⟨x1 = Shared sarg1⟩ ⟨xs = xs'⟩
  abstract-pre.simps(1)[of spre upre sarg2 sarg2]
  abstract-pre.simps(1)[of spre upre sarg1 sarg1]
  calculation compute.simps(2)[of sact uact v0]
  calculation compute.simps(2)[of sact uact v0']
  nth-Cons-0 ss-com[of  $\alpha$  sact spre uact upre] zero-less-diff zero-less-one-class.zero-le-one
  by metis
next
  case (Unique uarg2)
  then show ?thesis
  using ⟨all-axioms  $\alpha$  sact spre uact upre⟩ 0.premis(6) PRE-sequenceE-rec(1)[of
spre upre x1 x2 # xs x1 x2 # xs]
  PRE-sequenceE-rec(2)[of spre upre ]
  Shared ⟨ $\alpha$  (compute sact uact v0 xs) =  $\alpha$  (compute sact uact v0' xs)⟩ ⟨s =
x1 # x2 # xs⟩ ⟨xs = xs'⟩
  abstract-pre.simps(1)[of spre upre] abstract-pre.simps(2)[of spre upre]
  calculation
  compute.simps(2)[of sact uact ] compute.simps(3)[of sact uact]
  su-com[of  $\alpha$  sact spre uact upre]
  by metis
qed
next
  case (Unique k1 uarg1)
  then have x1 = Unique k1 uarg1 by simp
  then show ?thesis
  proof (cases x2)
    case (Shared sarg2)
    then have spre sarg2 sarg2  $\wedge$  upre k1 uarg1 uarg1
    by (metis 0.premis(6) PRE-sequenceE-rec(1) PRE-sequenceE-rec(2) Unique
⟨s = x1 # x2 # xs⟩ abstract-pre.simps(1) abstract-pre.simps(2))
    then show ?thesis
    using ⟨all-axioms  $\alpha$  sact spre uact upre⟩ Unique ⟨ $\alpha$  (compute sact uact v0 xs)
=  $\alpha$  (compute sact uact v0' xs)⟩
    ⟨s = x1 # x2 # xs⟩ ⟨s' = x2 # x1 # xs'⟩ ⟨xs = xs'⟩ compute.simps(2)[of
sact uact]
    compute.simps(3)[of sact uact] su-com[of  $\alpha$  sact spre uact upre]

```



```

    by (metis Shared)
  next
    case (Unique k2 uarg2)
    then have k1 ≠ k2
      by (metis 0.prem5 0.prem8 Suc-eq-plus1 ‹∧thesis. (∧xs'. s' = x2 #
x1 # xs' ⇒ thesis) ⇒ thesis› ‹s = x1 # x2 # xs› ‹x1 = Unique k1 uarg1›
action.disc(2) action.sel(2) is-Unique-def is-Unique-i-def nth-Cons-0)
      then have upre k2 uarg2 uarg2 ∧ upre k1 uarg1 uarg1
        by (metis 0.prem6 PRE-sequenceE-rec(1) PRE-sequenceE-rec(2) Unique
‹s = x1 # x2 # xs› ‹x1 = Unique k1 uarg1› abstract-pre.simps(2))

      then show ?thesis
        using ‹all-axioms α sact spre uact upre› Unique ‹α (compute sact uact v0 xs)
= α (compute sact uact v0' xs)›
          ‹s = x1 # x2 # xs› ‹s' = x2 # x1 # xs'› ‹xs = xs'› compute.simps(2)[of
sact uact]
          uu-com[of α sact spre uact upre k1 k2 compute sact uact v0' xs compute sact
uact v0 xs]
          ‹k1 ≠ k2› ‹x1 = Unique k1 uarg1› compute.simps(3)
        by auto
      qed
    qed
  next
    case (Suc k)
    then obtain x xs x' xs' where s = x # xs s' = x' # xs'
      by (metis diff-0-eq-0 length-0-conv neq-Nil-conv not-less-zero)
    then have x = x'
      using Suc.prem3 by force
    moreover have α (compute sact uact v0 (tl s)) = α (compute sact uact v0' (tl
s'))
      proof (rule Suc(1))
        show length (tl s) = length (tl s')
          by (simp add: Suc.prem1)
        show k < length (tl s) - 1
          using Suc.prem2 by auto
        show ‹∧i. 0 ≤ i ∧ i < length (tl s) ∧ i ≠ k ∧ i ≠ k + 1 ⇒ tl s ! i = tl s' ! i›
          by (metis Suc.prem3 Suc-eq-plus1 ‹length (tl s) = length (tl s')› length-tl
less-diff-conv nat.inject nat-le-linear not-less-eq-eq nth-tl)
        show tl s ! k = tl s' ! (k + 1)
          by (metis Suc.prem4 Suc-eq-plus1 ‹s = x # xs› ‹s' = x' # xs'› add-diff-cancel-right'
add-gr-0 le-neq-implies-less list.sel(3) not-one-le-zero nth-Cons-pos zero-less-one-class.zero-le-one)
        show tl s' ! k = tl s ! (k + 1)
          by (metis Suc.prem5 Suc-eq-plus1 ‹s = x # xs› ‹s' = x' # xs'› list.sel(3)
nth-Cons-Suc)
        show PRE-sequence spre upre (tl s) (tl s)
          by (metis Suc.prem6 ‹s = x # xs› PRE-sequenceE-rec(1) list.sel(3))
        show α v0 = α v0'
          by (simp add: assms(7))
        show ¬ (∃ k'. is-Unique-i k' (tl s ! k) ∧ is-Unique-i k' (tl s ! (k + 1)))

```

```

    using Suc.premis(8) ⟨s = x # xs⟩ by force
  show all-axioms α sact spre uact upre
    by (simp add: Suc.premis(9))
qed
ultimately show ?case
proof (cases x)
  case (Shared x1)
  then show ?thesis
    using ⟨all-axioms α sact spre uact upre⟩ PRE-sequenceE-rec(2) Suc.premis(6)
    ⟨α (compute sact uact v0 (tl s)) = α (compute sact uact v0' (tl s'))⟩ ⟨s = x # xs⟩
    ⟨s' = x' # xs'⟩ ⟨x = x'⟩ sabstract
    by fastforce
  next
  case (Unique x2)
  then show ?thesis
    using ⟨all-axioms α sact spre uact upre⟩ PRE-sequenceE-rec(2) Suc.premis(6)
    ⟨α (compute sact uact v0 (tl s)) = α (compute sact uact v0' (tl s'))⟩ ⟨s = x
# xs⟩ ⟨s' = x' # xs'⟩
    ⟨x = x'⟩ uabstract[of α sact spre uact upre]
    by fastforce
  qed
qed

```

```

lemma mset-remove:
  assumes k ≥ 0 ∧ k < length s
  shows mset s = mset (remove-at-index k s) + {# s ! k #}
  using assms
proof (induct s arbitrary: k)
  case Nil
  then show ?case
    by simp
  next
  case (Cons a s)
  then show ?case
    using less-Suc-eq-0-disj by auto
qed

```

```

lemma abstract-pre-refl:
  assumes abstract-pre spre upre a b
  and all-axioms α sact spre uact upre
  shows abstract-pre spre upre b b
  apply (cases a)
  apply (cases b)
  using abstract-pre.simps(1) assms spre-refl apply metis
  using assms apply force
  apply (cases b)
  using assms apply force
  using abstract-pre.simps(2) assms upre-refl by metis

```

```

lemma PRE-sequence-refl:
  assumes PRE-sequence spre upre s s'
    and all-axioms  $\alpha$  sact spre uact upre
  shows PRE-sequence spre upre s' s'
proof (rule PRE-sequenceI)
  show length s' = length s'
    by simp
  fix i assume  $0 \leq i \wedge i < \text{length } s'$ 
  then show abstract-pre spre upre (s' ! i) (s' ! i)
    by (metis PRE-sequence-def abstract-pre-refl assms)
qed

lemma PRE-sequence-removes:
  assumes PRE-sequence spre upre s s
  shows PRE-sequence spre upre (remove-at-index n s) (remove-at-index n s)
  using assms
proof (induct n arbitrary: s)
  case 0
  then show ?case
    by (metis PRE-sequenceE-rec(1) nat.simps(3) remove-at-index.elims)
next
  case (Suc n)
  then show ?case
    apply (cases s)
    apply force
    by (metis PRE-sequenceE-rec(1) PRE-sequenceE-rec(2) PRE-sequenceI-rec remove-at-index.simps(3))
qed

lemma PRE-sequence-insert:
  assumes abstract-pre spre upre x x
    and PRE-sequence spre upre s s
  shows PRE-sequence spre upre (insert-at n x s) (insert-at n x s)
  using assms
proof (induct n arbitrary: s)
  case 0
  then show ?case
    by (simp add: PRE-sequenceI-rec)
next
  case (Suc n)
  then show ?case
    apply (cases s)
    apply (simp add: PRE-sequenceI-rec)
    by (metis PRE-sequenceE-rec(1) PRE-sequenceE-rec(2) PRE-sequenceI-rec insert-at.simps(3))
qed

lemma empty-possible-sequence:
  assumes possible-sequence sargs uargs []

```

```

    and possible-sequence sargs uargs s'
  shows s' = []
proof (rule ccontr)
  assume s' ≠ []
  then obtain x q where s' = x # q
    by (meson neq-Nil-conv)
  then show False
proof (cases x)
  case (Shared x1)
  then show ?thesis
    by (metis ‹s' = x # q› assms(1) assms(2) exists-index-in-sequence-shared
less-zeroE list.size(3) possible-sequence-where-is-shared)
  next
  case (Unique k uarg)
  then have uargs k ≠ []
    by (metis (no-types, lifting) ‹s' = x # q› action.disc(2) action.sel(2) assms(2)
filter.simps(2) is-Unique-def is-Unique-i-def list.discI list.map-disc-iff possible-sequence-def)
  then show ?thesis
    by (metis assms(1) exists-index-in-sequence-unique less-nat-zero-code list.size(3))
qed
qed

```

lemma *it-all-commutes*:

```

  assumes possible-sequence sargs uargs s
    and possible-sequence sargs uargs s'
    and  $\alpha v0 = \alpha v0'$ 
    and PRE-sequence spre upre s s
    and PRE-sequence spre upre s' s'
    and all-axioms  $\alpha$  sact spre uact upre
  shows  $\alpha$  (compute sact uact v0 s) =  $\alpha$  (compute sact uact v0' s')
  using assms
proof (induct size s arbitrary: sargs uargs s s')
  case 0
  then have s = []  $\wedge$  s' = []
    by (simp add: empty-possible-sequence)
  then show ?case
    by (simp add: 0.premis(1) 0.premis(2) assms(3))
  next
  case (Suc n)
  moreover obtain x s1 where s = x # s1
    by (meson Suc.hyps(2) Suc-length-conv)
  then have abstract-pre spre upre x x
    using Suc.premis(4) PRE-sequenceE-rec(2) by blast
  then show ?case
proof (cases x)
  case (Shared sarg)
  then have Shared sarg  $\in$  set s'
    by (metis Suc.premis(1) Suc.premis(2) ‹s = x # s1› exists-index-in-sequence-shared
nth-mem possible-sequence-where-is-shared)

```

then obtain k **where** $k \geq 0 \wedge k < \text{length } s' \wedge s' ! k = x$
by (*metis Shared bot-nat-0.extremum in-set-conv-nth*)

let $?s' = \text{remove-at-index } k \ s'$
have $\text{length } ?s' = \text{length } s' - 1$
by (*simp add: <0 ≤ k ∧ k < length s' ∧ s' ! k = x> remove-at-index(1)*)
moreover have $k < \text{length } s'$
by (*simp add: <0 ≤ k ∧ k < length s' ∧ s' ! k = x>*)
then have $s' = \text{insert-at } k \ x \ ?s'$
by (*simp add: <0 ≤ k ∧ k < length s' ∧ s' ! k = x> insert-remove-same-list*)
define $i :: \text{nat}$ **where** $i = k$
have $i \geq 0 \wedge i \leq k \implies \alpha (\text{compute sact uact } v0' (\text{insert-at } (k - i) \ x \ ?s')) =$
 $\alpha (\text{compute sact uact } v0' \ s')$
proof (*induct i*)
case 0
then show $?case$
using $\langle s' = \text{insert-at } k \ x \ (\text{remove-at-index } k \ s') \rangle$ **by** *auto*
next
case (*Suc i*)
then have $\alpha (\text{compute sact uact } v0' (\text{insert-at } (k - i) \ x \ (\text{remove-at-index } k$
 $s')) = \alpha (\text{compute sact uact } v0' \ s')$
using *Suc-leD* **by** *blast*
moreover have $\alpha (\text{compute sact uact } v0' (\text{insert-at } (k - \text{Suc } i) \ x \ (\text{remove-at-index } k$
 $s')) = \alpha (\text{compute sact uact } v0' (\text{insert-at } (k - i) \ x \ (\text{remove-at-index } k \ s')))$
proof (*rule swap-works*)
show $\text{length } (\text{insert-at } (k - \text{Suc } i) \ x \ (\text{remove-at-index } k \ s')) = \text{length}$
 $(\text{insert-at } (k - i) \ x \ (\text{remove-at-index } k \ s'))$
by (*metis (no-types, lifting) Suc-pred' <0 ≤ k ∧ k < length s' ∧ s' ! k = x> <length (remove-at-index k s') = length s' - 1> diff-le-self insert-at-index(1) less-Suc-eq-le order-le-less-trans*)
show $\text{PRE-sequence spre upre } (\text{insert-at } (k - \text{Suc } i) \ x \ (\text{remove-at-index } k$
 $s')) (\text{insert-at } (k - \text{Suc } i) \ x \ (\text{remove-at-index } k \ s'))$
proof $-$
have $\text{PRE-sequence spre upre } (\text{remove-at-index } k \ s') (\text{remove-at-index } k$
 $s')$ **using** $\langle \text{PRE-sequence spre upre } s' \ s' \rangle$
using *PRE-sequence-removes* **by** *auto*
then show $?thesis$ **using** *PRE-sequence-insert* $\langle \text{abstract-pre spre upre } x$
 $x \rangle$ **by** *blast*
qed
show $\alpha \ v0' = \alpha \ v0'$ **by** *simp*
let $?k = k - \text{Suc } i$

show $?k < \text{length } (\text{insert-at } (k - \text{Suc } i) \ x \ (\text{remove-at-index } k \ s')) - 1$
using *One-nat-def Suc.prems Suc-diff-Suc Suc-le-lessD* $\langle k < \text{length } s' \rangle$
 $\langle \text{length } (\text{remove-at-index } k \ s') = \text{length } s' - 1 \rangle$
 $\langle s' = \text{insert-at } k \ x \ (\text{remove-at-index } k \ s') \rangle$ *diff-le-self diff-zero*
 $\text{insert-at-index}(1)[\text{of } k - \text{Suc } i - x] \ \text{insert-at-index}(1)[\text{of } k - x]$ *less-Suc-eq-le*
order-le-less-trans
by *simp*

show $\text{insert-at } (k - \text{Suc } i) \ x \ (\text{remove-at-index } k \ s') \ ! \ ?k = \text{insert-at } (k - i) \ x \ (\text{remove-at-index } k \ s') \ ! \ (?k + 1)$
apply $(\text{cases } k)$
using $\text{Suc.prem.s apply blast}$
apply $(\text{cases } ?k)$
apply $(\text{metis } (\text{no-types, lifting}) \ \text{Suc.prem.s Suc-eq-plus1 Suc-leI } \langle k - \text{Suc } i < \text{length } (\text{insert-at } (k - \text{Suc } i) \ x \ (\text{remove-at-index } k \ s')) - 1 \rangle \ \text{add-diff-cancel-right}' \ \text{diff-diff-cancel} \ \text{diff-zero} \ \text{insert-at-index}(1) \ \text{insert-at-index}(3) \ \text{le-numeral-extra}(3) \ \text{length-greater-0-conv} \ \text{list.size}(3) \ \text{nat-less-le} \ \text{plus-1-eq-Suc})$
proof –
fix $\text{nat } \text{nata}$ **assume** $r: k = \text{Suc } \text{nat } k - \text{Suc } i = \text{Suc } \text{nata}$
moreover have $\text{insert-at } (k - i) \ x \ (\text{remove-at-index } k \ s') \ ! \ (k - i) = x$
by $(\text{metis } \text{Suc-pred}' \ \langle k < \text{length } s' \rangle \ \langle \text{length } (\text{remove-at-index } k \ s') = \text{length } s' - 1 \rangle \ \text{bot-nat-0.extremum} \ \text{diff-le-self} \ \text{insert-at-index}(3) \ \text{less-Suc-eq-le} \ \text{order-le-less-trans})$
moreover have $\bigwedge x. \text{insert-at } (k - \text{Suc } i) \ x \ (\text{remove-at-index } k \ s') \ ! \ (k - \text{Suc } i) = x$
by $(\text{metis } \text{Suc-leI} \ \text{Suc-le-mono} \ \text{Suc-pred}' \ \langle 0 \leq k \wedge k < \text{length } s' \wedge s' ! \ k = x \rangle \ \langle \text{length } (\text{remove-at-index } k \ s') = \text{length } s' - 1 \rangle \ \text{bot-nat-0.extremum} \ \text{diff-le-self} \ \text{insert-at-index}(3) \ \text{order-le-less-trans})$
ultimately show $?thesis$
by $(\text{metis } \text{Suc.prem.s Suc-diff-Suc Suc-eq-plus1 Suc-le-lessD})$
qed
show $\text{insert-at } (k - i) \ x \ (\text{remove-at-index } k \ s') \ ! \ ?k = \text{insert-at } (k - \text{Suc } i) \ x \ (\text{remove-at-index } k \ s') \ ! \ (?k + 1)$
proof –
have $\text{insert-at } (k - i) \ x \ (\text{remove-at-index } k \ s') \ ! \ (k - \text{Suc } i) = \text{remove-at-index } k \ s' ! \ (k - \text{Suc } i)$
by $(\text{metis } (\text{no-types, lifting}) \ \text{Suc.prem.s Suc-diff-Suc Suc-eq-plus1 Suc-leI} \ \text{Suc-le-lessD} \ \langle k < \text{length } s' \rangle \ \langle \text{length } (\text{remove-at-index } k \ s') = \text{length } s' - 1 \rangle \ \text{add-leE} \ \text{insert-at-index}(2) \ \text{le-add-diff-inverse2} \ \text{le-add-same-cancel2} \ \text{lessI} \ \text{less-Suc-eq-le})$
moreover have $\text{length } (\text{insert-at } (k - \text{Suc } i) \ x \ (\text{remove-at-index } k \ s')) = \text{length } (\text{remove-at-index } k \ s') + 1$
by $(\text{metis } \text{Suc-eq-plus1} \ \langle 0 \leq k \wedge k < \text{length } s' \wedge s' ! \ k = x \rangle \ \langle \text{length } (\text{remove-at-index } k \ s') = \text{length } s' - 1 \rangle \ \text{add-le-imp-le-diff} \ \text{insert-at-index}(1) \ \text{less-eq-Suc-le} \ \text{less-imp-diff-less})$
then have $\text{insert-at } (k - \text{Suc } i) \ x \ (\text{remove-at-index } k \ s') \ ! \ (k - \text{Suc } i + 1) = \text{remove-at-index } k \ s' ! \ (k - \text{Suc } i + 1 - 1)$
by $(\text{metis } \langle k - \text{Suc } i < \text{length } (\text{insert-at } (k - \text{Suc } i) \ x \ (\text{remove-at-index } k \ s')) - 1 \rangle \ \text{add-diff-cancel-right}' \ \text{insert-at-index}(4) \ \text{less-add-one} \ \text{less-diff-conv} \ \text{less-imp-le-nat})$
ultimately show $?thesis$
by simp
qed
show $\neg (\exists k'. \text{is-Unique-}i \ k' \ (\text{insert-at } (k - \text{Suc } i) \ x \ (\text{remove-at-index } k \ s') \ ! \ (k - \text{Suc } i)) \wedge \text{is-Unique-}i \ k' \ (\text{insert-at } (k - \text{Suc } i) \ x \ (\text{remove-at-index } k \ s') \ ! \ (k - \text{Suc } i + 1)))$
by $(\text{metis } (\text{no-types, lifting}) \ \text{One-nat-def} \ \text{Shared} \ \text{Suc.prem.s Suc-diff-Suc} \ \langle k$

$\langle \text{length } s' \rangle \langle \text{length } (\text{remove-at-index } k \ s') = \text{length } s' - 1 \rangle \text{action.disc}(1) \text{add-leE}$
 $\text{diff-zero insert-at-index}(3) \text{is-Unique-def is-Unique-i-def le-add-diff-inverse2 le-add-same-cancel2}$
 $\text{less-Suc-eq-le order-le-less-trans}$)

show $\bigwedge j. 0 \leq j \wedge j < \text{length } (\text{insert-at } (k - \text{Suc } i) \ x \ (\text{remove-at-index } k \ s')) \wedge j \neq ?k \wedge j \neq ?k + 1 \implies$

$\text{insert-at } (k - \text{Suc } i) \ x \ (\text{remove-at-index } k \ s') ! j = \text{insert-at } (k - i) \ x \ (\text{remove-at-index } k \ s') ! j$

proof (*clarify*)

fix j **assume** $0 \leq j \wedge j < \text{length } (\text{insert-at } (k - \text{Suc } i) \ x \ (\text{remove-at-index } k \ s')) \wedge j \neq k - \text{Suc } i \wedge j \neq k - \text{Suc } i + 1$

moreover have $\text{length } (\text{insert-at } (k - \text{Suc } i) \ x \ (\text{remove-at-index } k \ s')) = \text{length } (\text{remove-at-index } k \ s') + 1$

by (*metis (no-types, lifting) One-nat-def Suc.premS Suc-diff-Suc* $\langle k < \text{length } s' \rangle \langle \text{length } (\text{remove-at-index } k \ s') = \text{length } s' - 1 \rangle \text{add-leE diff-zero insert-at-index}(1) \text{le-add-diff-inverse2 less-Suc-eq-le order-le-less-trans}$)

moreover have $k - \text{Suc } i \leq \text{length } (\text{remove-at-index } k \ s')$

using $\langle k - \text{Suc } i < \text{length } (\text{insert-at } (k - \text{Suc } i) \ x \ (\text{remove-at-index } k \ s')) - 1 \rangle \text{calculation}(5)$ **by force**

ultimately show $\text{insert-at } (k - \text{Suc } i) \ x \ (\text{remove-at-index } k \ s') ! j = \text{insert-at } (k - i) \ x \ (\text{remove-at-index } k \ s') ! j$

apply (*cases* $j < k - \text{Suc } i$)

using $\text{insert-at-index}(2)[\text{of } k - \text{Suc } i \ \text{remove-at-index } k \ s' \ j \ x] \ \text{insert-at-index}(2)[\text{of } k - i \ \text{remove-at-index } k \ s' \ j \ x]$

apply (*metis Suc.premS Suc-diff-le Suc-eq-plus1 Suc-leI* $\langle k - \text{Suc } i < \text{length } (\text{insert-at } (k - \text{Suc } i) \ x \ (\text{remove-at-index } k \ s')) - 1 \rangle \text{diff-Suc-1 diff-Suc-Suc less-Suc-eq}$)

by (*simp add: insert-at-index}(4) nat-neg-iff*)

qed

show *all-axioms* $\alpha \ \text{sact spre uact upre}$

by (*simp add: assms}(6)*)

qed

ultimately show *?case*

by *presburger*

qed

then have $\alpha \ (\text{compute sact uact } v0' \ (x \ \# \ ?s')) = \alpha \ (\text{compute sact uact } v0' \ s')$

using *i-def* **by force**

moreover have $\alpha \ (\text{compute sact uact } v0 \ s1) = \alpha \ (\text{compute sact uact } v0' \ ?s')$

proof (*rule Suc}(1)*)

show $n = \text{length } s1$

using *Suc.hyps}(2) \langle s = x \ \# \ s1 \rangle* **by auto**

show $\alpha \ v0 = \alpha \ v0'$

using *assms}(3)* **by auto**

show *PRE-sequence spre upre* $s1 \ s1$

using *PRE-sequenceE-rec}(1) Suc.premS}(4) \langle s = x \ \# \ s1 \rangle* **by blast**

show *possible-sequence* ($\text{sargs} - \{\# \ \text{sarg} \ \#\}$) $\text{uargs } s1$

proof (*rule possible-sequenceI*)

show $\bigwedge i. \ \text{uargs } i = \text{map get-u } (\text{filter } (\text{is-Unique-i } i) \ s1)$

by (*metis (mono-tags, lifting) Shared Suc.hyps}(2) Suc.premS}(1) \langle s = x \ \#*

$s1 \triangleright$ *action.disc(1) filter-remove is-Unique-def is-Unique-i-def le-numeral-extra(3)*
nth-Cons-0 possible-sequence-def remove-at-index.simps(2) zero-less-Suc
show $sargs - \{\#sarg\#\} = image-mset\ get-s\ (filter-mset\ is-Shared\ (mset\ s1))$
using *Shared Suc.prem(1) $\langle s = x \# s1 \rangle$ action.disc(1)[of sarg] action.sel(1)[of sarg] add-mset-diff-bothsides diff-empty*
filter-mset-add-mset msed-map-invL mset.simps(2) possible-sequence-def[of sargs uargs s]
by simp
qed

show *possible-sequence (sargs - $\{\#sarg\#\})$ uargs (remove-at-index k s')*
proof *(rule possible-sequenceI)*
show $\bigwedge i. uargs\ i = map\ get-u\ (filter\ (is-Unique-i\ i)\ (remove-at-index\ k\ s'))$
proof *(rule list-ext)*
have *filter is-Unique (remove-at-index k s') = filter is-Unique s'*
by *(simp add: Shared $\langle 0 \leq k \wedge k < length\ s' \wedge s' ! k = x \rangle$ filter-remove is-Unique-def)*
then show $\bigwedge i. length\ (uargs\ i) = length\ (map\ get-u\ (filter\ (is-Unique-i\ i)\ (remove-at-index\ k\ s')))$
by *(metis Shared Suc.prem(2) $\langle 0 \leq k \wedge k < length\ s' \wedge s' ! k = x \rangle$ action.disc(1) filter-remove is-Unique-def is-Unique-i-def possible-sequence-def)*
show $\bigwedge i\ ia. 0 \leq ia \wedge ia < length\ (uargs\ i) \implies uargs\ i ! ia = map\ get-u\ (filter\ (is-Unique-i\ i)\ (remove-at-index\ k\ s')) ! ia$
by *(metis Shared Suc.prem(2) $\langle 0 \leq k \wedge k < length\ s' \wedge s' ! k = x \rangle$ action.disc(1) filter-remove is-Unique-def is-Unique-i-def possible-sequence-def)*
qed
have $sargs = image-mset\ get-s\ (filter-mset\ is-Shared\ (mset\ s'))$
using *Suc.prem(2) possible-sequence-def by blast*
show $sargs - \{\#sarg\#\} = image-mset\ get-s\ (filter-mset\ is-Shared\ (mset\ (remove-at-index\ k\ s')))$
proof $-$
have $mset\ s' = mset\ (remove-at-index\ k\ s') + \{\#x\#\}$
using $\langle 0 \leq k \wedge k < length\ s' \wedge s' ! k = x \rangle$ *mset-remove-index by blast*
then show *?thesis*
by *(simp add: Shared $\langle sargs = image-mset\ get-s\ (filter-mset\ is-Shared\ (mset\ s')) \rangle$)*
qed
qed
show *PRE-sequence spre upre (remove-at-index k s') (remove-at-index k s')*
using *Suc.prem(5) PRE-sequence-removes by blast*
show *all-axioms α sact spre uact upre by (simp add: assms(6))*
qed
ultimately show *?thesis*
using $\langle all-axioms\ \alpha\ sact\ spre\ uact\ upre \rangle$ *PRE-sequenceE-rec(2) Shared Suc.prem(4) $\langle s = x \# s1 \rangle$ abstract-pre.simps(1) compute.simps(2) sabtract*
by fastforce
next
case *(Unique ind uarg)*


```

let ?uargs = uargs ind
have hd ?uargs = uarg
by (metis Unique ⟨s = x # s1⟩ calculation(3) possible-sequence-where-is-unique)
moreover have ?uargs ≠ []
  by (metis (no-types, opaque-lifting) Suc.prem(1) Unique ⟨s = x # s1⟩ ac-
tion.disc(2) action.sel(2) filter.simps(2) is-Unique-def is-Unique-i-def list.distinct(1)
list.map-disc-iff possible-sequence-def)
ultimately have ?uargs = uarg # tl ?uargs
  by force
then obtain k where k ≥ 0 ∧ k < length s' ∧ s' ! k = x ∧ j. j ≥ 0 ∧ j < k
⇒ ¬ is-Unique-i ind (s' ! j)
  by (metis Suc.prem(2) Unique ⟨hd (uargs ind) = uarg⟩ ⟨uargs ind ≠ []⟩
exists-index-in-sequence-unique)
let ?s' = remove-at-index k s'
have length ?s' = length s' - 1
  by (simp add: ⟨0 ≤ k ∧ k < length s' ∧ s' ! k = x⟩ remove-at-index(1))
moreover have k < length s'
  by (simp add: ⟨0 ≤ k ∧ k < length s' ∧ s' ! k = x⟩)
then have s' = insert-at k x ?s'
  by (simp add: ⟨0 ≤ k ∧ k < length s' ∧ s' ! k = x⟩ insert-remove-same-list)
define i :: nat where i = k
have i ≥ 0 ∧ i ≤ k ⇒ α (compute sact uact v0' (insert-at (k - i) x ?s')) =
α (compute sact uact v0' s')
proof (induct i)
  case 0
  then show ?case
    using ⟨s' = insert-at k x (remove-at-index k s')⟩ by auto
next
  case (Suc i)
  then have α (compute sact uact v0' (insert-at (k - i) x (remove-at-index k
s'))) = α (compute sact uact v0' s')
    using Suc-leD by blast
  moreover have α (compute sact uact v0' (insert-at (k - Suc i) x (remove-at-index
k s'))) = α (compute sact uact v0' (insert-at (k - i) x (remove-at-index k s')))
    proof (rule swap-works)
      show length (insert-at (k - Suc i) x (remove-at-index k s')) = length
(insert-at (k - i) x (remove-at-index k s'))
        by (metis (no-types, lifting) Suc-pred' ⟨0 ≤ k ∧ k < length s' ∧ s' ! k =
x⟩ ⟨length (remove-at-index k s') = length s' - 1⟩ insert-at-index(1) less-Suc-eq-le
less-imp-diff-less order-le-less-trans)
      show PRE-sequence spre upre (insert-at (k - Suc i) x (remove-at-index k
s')) (insert-at (k - Suc i) x (remove-at-index k s'))
        proof -
          have PRE-sequence spre upre (remove-at-index k s') (remove-at-index k s')
            using ⟨PRE-sequence spre upre s' s'⟩
            by (simp add: PRE-sequence-removes)
          then show ?thesis
            using ⟨abstract-pre spre upre x x⟩ PRE-sequence-insert by blast
        qed
    end
end

```

show $\alpha v0' = \alpha v0'$ **by** *simp*
let $?k = k - \text{Suc } i$

show $?k < \text{length } (\text{insert-at } (k - \text{Suc } i) x (\text{remove-at-index } k s')) - 1$
using *One-nat-def Suc.premS Suc-diff-Suc Suc-le-lessD* $\langle k < \text{length } s' \rangle$
 $\langle \text{length } (\text{remove-at-index } k s') = \text{length } s' - 1 \rangle \langle s' = \text{insert-at } k x$
 $(\text{remove-at-index } k s') \rangle$
diff-le-self diff-zero insert-at-index(1)[of k remove-at-index k s' x] in-
sert-at-index(1)[of k - Suc i remove-at-index k s' x]
less-Suc-eq-le order-le-less-trans
by *simp*
show $\text{insert-at } (k - \text{Suc } i) x (\text{remove-at-index } k s') ! ?k = \text{insert-at } (k -$
 $i) x (\text{remove-at-index } k s') ! (?k + 1)$
proof $-$
have $\text{insert-at } (k - \text{Suc } i) x (\text{remove-at-index } k s') ! (k - \text{Suc } i) = x$
by (*metis Suc-pred'* $\langle k < \text{length } s' \rangle \langle \text{length } (\text{remove-at-index } k s')$
 $= \text{length } s' - 1 \rangle$ *bot-nat-0.extremum diff-self-eq-0 insert-at-index(3) less-Suc-eq-le*
less-imp-diff-less)
moreover **have** $\text{insert-at } (k - i) x (\text{remove-at-index } k s') ! (k - i) = x$
by (*metis Suc-pred'* $\langle k < \text{length } s' \rangle \langle \text{length } (\text{remove-at-index } k s') = \text{length } s'$
 $- 1 \rangle$ *bot-nat-0.extremum insert-at-index(3) less-Suc-eq-le less-imp-diff-less less-nat-zero-code*
not-gr-zero)
ultimately **show** *?thesis*
by (*simp add: Suc.premS Suc-diff-Suc Suc-le-lessD*)
qed
have $\text{insert-at } (k - i) x (\text{remove-at-index } k s') ! (k - \text{Suc } i) = \text{remove-at-index}$
 $k s' ! (k - \text{Suc } i)$
by (*metis (no-types, lifting) Suc.premS Suc-diff-Suc Suc-eq-plus1 Suc-leI*
Suc-le-lessD $\langle k < \text{length } s' \rangle \langle \text{length } (\text{remove-at-index } k s') = \text{length } s' - 1 \rangle$ *add-leE*
insert-at-index(2) le-add-diff-inverse2 le-add-same-cancel2 lessI less-Suc-eq-le)
then
show $\text{insert-at } (k - i) x (\text{remove-at-index } k s') ! ?k = \text{insert-at } (k - \text{Suc}$
 $i) x (\text{remove-at-index } k s') ! (?k + 1)$
using *One-nat-def Suc.premS Suc-diff-Suc* $\langle k - \text{Suc } i < \text{length } (\text{insert-at}$
 $(k - \text{Suc } i) x (\text{remove-at-index } k s')) - 1 \rangle$
 $\langle k < \text{length } s' \rangle \langle \text{length } (\text{remove-at-index } k s') = \text{length } s' - 1 \rangle$
add-diff-cancel-right'
add-leE diff-zero insert-at-index(1)[of k - Suc i remove-at-index k s' x]
insert-at-index(4)[of k - Suc i remove-at-index k s']
le-add-diff-inverse2 less-Suc-eq-le
less-add-same-cancel1 less-diff-conv order-le-less-trans zero-less-one
by *simp*
have $\text{insert-at } (k - \text{Suc } i) x (\text{remove-at-index } k s') ! (k - \text{Suc } i + 1) =$
 $\text{remove-at-index } k s' ! (k - \text{Suc } i + 1 - 1)$
using $\langle \text{insert-at } (k - i) x (\text{remove-at-index } k s') ! (k - \text{Suc } i) = \text{insert-at}$
 $(k - \text{Suc } i) x (\text{remove-at-index } k s') ! (k - \text{Suc } i + 1) \rangle \langle \text{insert-at } (k - i) x$

$(\text{remove-at-index } k \ s') ! (k - \text{Suc } i) = \text{remove-at-index } k \ s' ! (k - \text{Suc } i)$ **by** *auto*
then have $\neg \text{is-Unique-}i \ \text{ind } (\text{insert-at } (k - \text{Suc } i) \ x \ (\text{remove-at-index } k \ s'))$
 $! (?k + 1))$
by (*metis One-nat-def Suc.premS Suc-le-lessD* $\langle \wedge j. 0 \leq j \wedge j < k \implies \neg$
 $\text{is-Unique-}i \ \text{ind } (s' ! j) \rangle \langle k < \text{length } s' \rangle$ *add-diff-cancel-right'* *add-leE* *diff-Suc-less*
le-add2 *le-add-same-cancel2* *plus-1-eq-Suc* *remove-at-index(2)*)
then show $\neg (\exists k'. \text{is-Unique-}i \ k' \ (\text{insert-at } (k - \text{Suc } i) \ x \ (\text{remove-at-index}$
 $k \ s') ! (k - \text{Suc } i)) \wedge$
 $\text{is-Unique-}i \ k' \ (\text{insert-at } (k - \text{Suc } i) \ x \ (\text{remove-at-index } k \ s') ! (k - \text{Suc } i$
 $+ 1)))$
by (*metis (no-types, lifting) One-nat-def Suc.premS Suc-diff-Suc Unique* $\langle k$
 $< \text{length } s' \rangle \langle \text{length } (\text{remove-at-index } k \ s') = \text{length } s' - 1 \rangle$ *action.sel(2)* *diff-zero*
insert-at-index(3) *is-Unique-}i-def* *le-add2* *le-add-diff-inverse* *le-add-same-cancel2*
less-Suc-eq-le *order-le-less-trans*)
show $\wedge j. 0 \leq j \wedge j < \text{length } (\text{insert-at } (k - \text{Suc } i) \ x \ (\text{remove-at-index } k$
 $s')) \wedge j \neq ?k \wedge j \neq ?k + 1 \implies$
 $\text{insert-at } (k - \text{Suc } i) \ x \ (\text{remove-at-index } k \ s') ! j = \text{insert-at } (k - i) \ x$
 $(\text{remove-at-index } k \ s') ! j$
proof –
fix j **assume** $0 \leq j \wedge j < \text{length } (\text{insert-at } (k - \text{Suc } i) \ x \ (\text{remove-at-index}$
 $k \ s')) \wedge j \neq ?k \wedge j \neq ?k + 1$
moreover have $k - \text{Suc } i \leq \text{length } (\text{remove-at-index } k \ s')$
using $\langle 0 \leq k \wedge k < \text{length } s' \wedge s' ! k = x \rangle \langle \text{length } (\text{remove-at-index } k$
 $s') = \text{length } s' - 1 \rangle$ **by** *force*
moreover have $k - i \leq \text{length } (\text{remove-at-index } k \ s')$
using $\langle k < \text{length } s' \rangle \langle \text{length } (\text{remove-at-index } k \ s') = \text{length } s' - 1 \rangle$ **by**
linarith
then show $\text{insert-at } (k - \text{Suc } i) \ x \ (\text{remove-at-index } k \ s') ! j = \text{insert-at}$
 $(k - i) \ x \ (\text{remove-at-index } k \ s') ! j$
apply (*cases* $j < k - i$)
apply (*metis Suc.premS Suc-diff-Suc Suc-le-lessD* *calculation(1)* *calcu-*
lation(2) *insert-at-index(2)* *less-Suc-eq*)
by (*metis Suc.premS Suc-diff-Suc Suc-eq-plus1 Suc-le-lessD* *calcu-*
lation(1) *calculation(2)* *insert-at-index(1)* *insert-at-index(4)* *linorder-le-less-linear*
linorder-neqE-nat)
qed
show *all-axioms* α *sact spre uact upre* **by** (*simp add: assms(6)*)
qed
ultimately show *?case*
by *presburger*
qed
then have α (*compute* *sact uact* $v0' (x \# ?s')$) = α (*compute* *sact uact* $v0' s'$)
using *i-def* **by** *force*
moreover have α (*compute* *sact uact* $v0 \ s1$) = α (*compute* *sact uact* $v0' \ ?s'$)
proof (*rule* *Suc(1)*)
show *all-axioms* α *sact spre uact upre* **by** (*simp add: assms(6)*)
show $n = \text{length } s1$
using *Suc.hyps(2)* $\langle s = x \# s1 \rangle$ **by** *auto*
show $\alpha \ v0 = \alpha \ v0'$

```

using assms(3) by auto
show PRE-sequence spre upre s1 s1
  using Suc.prems(4)  $\langle s = x \# s1 \rangle$  PRE-sequenceE-rec(1) by blast
show possible-sequence sargs (tail-kth uargs ind) s1
proof (rule possible-sequenceI)
  show  $\bigwedge i. \text{tail-kth uargs ind } i = \text{map get-u (filter (is-Unique-i } i) s1)$ 
  proof –
    fix i show tail-kth uargs ind i = map get-u (filter (is-Unique-i } i) s1)
    proof (cases i = ind)
      case True
        then have tail-kth uargs ind i = tl ?uargs
          by (simp add: tail-kth-def)
        then show ?thesis using exists-index-in-sequence-unique[of uarg uargs
ind sargs s]
          by (metis Suc.prems(1) Unique  $\langle \text{hd (uargs ind) = uarg} \rangle \langle s = x$ 
 $\# s1 \rangle \langle \text{uargs ind} \neq [] \rangle$  action.disc(2) action.sel(2) is-Unique-def is-Unique-i-def
le-eq-less-or-eq nth-Cons-0 possible-sequence-def remove-at-index.simps(2))
        next
          case False
            then show ?thesis
              using Suc.hyps(2) Suc.prems(1) Unique  $\langle s = x \# s1 \rangle$  action.sel(2)
filter-remove
              fun-upd-apply is-Unique-i-def le-numeral-extra(3) nth-Cons-0[of x s1]
possible-sequence-def[of sargs uargs s]
              remove-at-index.simps(2)[of x s1] tail-kth-def zero-less-Suc
              by metis
            qed
          qed
        show sargs = image-mset get-s (filter-mset is-Shared (mset s1))
        by (metis Suc.prems(1) Unique  $\langle s = x \# s1 \rangle$  action.disc(2) filter-mset-add-mset
mset.simps(2) possible-sequence-def)
        qed
        show possible-sequence sargs (tail-kth uargs ind) (remove-at-index k s')
        proof (rule possible-sequenceI)
        show  $\bigwedge i. \text{tail-kth uargs ind } i = \text{map get-u (filter (is-Unique-i } i) (\text{remove-at-index$ 
 $k s'))$ 
          using Suc.prems(1) Suc.prems(2) Unique  $\langle 0 \leq k \wedge k < \text{length } s' \wedge s' ! k$ 
 $= x \rangle \langle \bigwedge j. 0 \leq j \wedge j < k \implies \neg \text{is-Unique-i ind } (s' ! j) \rangle$ 
           $\langle \text{possible-sequence sargs (tail-kth uargs ind) } s1 \rangle \langle s = x \# s1 \rangle$  action.sel(2)
          filter.simps(2)
          filter-remove fun-upd-same is-Unique-i-def possible-sequence-def[of sargs
tail-kth uargs ind s1]
          possible-sequence-def[of sargs uargs s] possible-sequence-def[of sargs uargs
s]
          remove-at-index-filter tail-kth-def
          by metis
        show sargs = image-mset get-s (filter-mset is-Shared (mset (remove-at-index
 $k s')))$ 
        by (metis Suc.prems(2) Unique  $\langle 0 \leq k \wedge k < \text{length } s' \wedge s' ! k = x \rangle$ 

```

```

action.disc(2) filter-remove mset-filter possible-sequence-def)
  qed
  show PRE-sequence spre upre (remove-at-index k s') (remove-at-index k s')
  using PRE-sequence-removes Suc.prem(5) by auto
  qed
  ultimately show ?thesis
  using Unique ⟨abstract-pre spre upre x x⟩ ⟨s = x # s1⟩ abstract-pre.simps(2)[of
]
  assms(6) compute.simps(3)[of sact uact] uabstract[of α sact spre uact upre ]
  by metis
  qed
qed

lemma PRE-sequence-same-abstract:
  assumes PRE-sequence spre upre s s'
  and α v0 = α v0'
  and all-axioms α sact spre uact upre
  shows α (compute sact uact v0 s) = α (compute sact uact v0' s')
  using assms
proof (induct s' arbitrary: s v0 v0')
  case Nil
  then show ?case
  by (simp add: PRE-sequence-def)
next
  case (Cons act' s')
  then show ?case
  proof (cases act')
    case (Shared sarg')
    then obtain sarg s0 where s = Shared sarg # s0 spre sarg sarg' PRE-sequence
spre upre s0 s'
    by (metis Cons.prem(1) PRE-sequenceE-rec(1) PRE-sequenceE-rec(2) PRE-sequence-def
abstract-pre.simps(1) abstract-pre.simps(3) action.exhaust length-0-conv neq-Nil-conv)
    then show ?thesis
    using Cons.hyps Cons.prem(2) Cons.prem(3) Shared sabstract by fastforce
  next
    case (Unique k uarg')
    then obtain uarg s0 where s = Unique k uarg # s0 upre k uarg uarg'
PRE-sequence spre upre s0 s'
    by (metis Cons.prem(1) PRE-sequenceE-rec(1) PRE-sequenceE-rec(2) PRE-sequence-def
abstract-pre.simps(2) abstract-pre.simps(4) action.exhaust length-0-conv neq-Nil-conv)
    then show ?thesis
    using Cons.hyps Cons.prem(2) Unique assms(3) uabstract by fastforce
  qed
qed

lemma simple-possible-PRE-seq:
  assumes possible-sequence sargs uargs s
  and possible-sequence sargs' uargs' s'
  and PRE-shared spre sargs sargs'

```

and $\bigwedge k$. *PRE-unique* (*upre* k) (*uargs* k) (*uargs'* k)
and *all-axioms* α *sact spre uact upre*
shows *PRE-sequence spre upre s' s'*
proof (*rule PRE-sequenceI*)
show *length s' = length s'* **by** *simp*
fix i **assume** $0 \leq i \wedge i < \text{length } s'$
then show *abstract-pre spre upre (s' ! i) (s' ! i)*
proof (*cases s' ! i*)
case (*Shared sarg'*)
then have *Shared sarg' ∈ # filter-mset is-Shared (mset s')*
using $\langle 0 \leq i \wedge i < \text{length } s' \rangle$ *nth-mem-mset* **by** *fastforce*
then have *sarg' ∈ # sargs'*
by (*metis (mono-tags, lifting) action.sel(1) assms(2) imageI possible-sequence-def set-image-mset*)
moreover obtain ms **where** *image-mset fst ms = sargs* \wedge *image-mset snd ms*
 $=$ *sargs' \wedge ($\forall x \in \# ms$. spre (fst x) (snd x))*
using *PRE-shared-def assms(3)* **by** *blast*
then obtain x **where** $x \in \# ms$ *snd x = sarg'*
using *calculation* **by** *fastforce*
then show *?thesis*
using *Shared \langle image-mset fst ms = sargs \wedge image-mset snd ms = sargs' \wedge*
 $(\forall x \in \# ms$. *spre (fst x) (snd x)* \rangle *spre-refl*
by (*metis abstract-pre.simps(1) assms(5)*)
next
case (*Unique k uarg'*)
then have *Unique k uarg' ∈ set (filter is-Unique s')*
by (*metis $\langle 0 \leq i \wedge i < \text{length } s' \rangle$ is-Unique-def action.disc(2) filter-set member-filter nth-mem*)
then have *uarg' ∈ set (map get-u (filter (is-Unique-i k) s'))*
by (*metis (no-types, lifting) action.sel(2) action.sel(3) filter-set image-eqI is-Unique-i-def list.set-map member-filter*)
then obtain i **where** $i \geq 0 \wedge i < \text{length } (uargs' k) \wedge uarg' = (uargs' k) ! i$
by (*metis assms(2) gr-implies-not-zero in-set-conv-nth linorder-le-less-linear possible-sequence-def*)
then have *upre k ((uargs k) ! i) ((uargs' k) ! i)*
using *PRE-unique-def assms(4)* **by** *blast*
then show *?thesis*
using *Unique $\langle 0 \leq i \wedge i < \text{length } (uargs' k) \wedge uarg' = uargs' k ! i \rangle$ assms(5)*
upre-refl **by** *fastforce*
qed
qed

lemma *main-lemma*:

assumes *possible-sequence sargs uargs s*
and *possible-sequence sargs' uargs' s'*

and *PRE-shared spre sargs sargs'*
and $\bigwedge k$. *PRE-unique* (*upre* k) (*uargs* k) (*uargs'* k)

and $\alpha v0 = \alpha v0'$
and *all-axioms* α *sact spre uact upre*

shows α (*compute sact uact v0 s*) = α (*compute sact uact v0' s'*)

proof –

obtain s'' **where** *possible-sequence sargs' uargs' s''* \wedge *PRE-sequence spre upre s s''*

using *assms(1) assms(2) assms(3) assms(4) exists-aligned-sequence* **by** *blast*

have α (*compute sact uact v0' s''*) = α (*compute sact uact v0' s'*)

proof (*rule it-all-commutes*)

show *possible-sequence sargs' uargs' s''*

by (*simp add: <possible-sequence sargs' uargs' s''* \wedge *PRE-sequence spre upre s s''*)

show *possible-sequence sargs' uargs' s'*

by (*simp add: assms(2)*)

show $\alpha v0' = \alpha v0'$

by *simp*

show *PRE-sequence spre upre s'' s''*

using *<possible-sequence sargs' uargs' s''* \wedge *PRE-sequence spre upre s s''*

PRE-sequence-refl assms(6) **by** *blast*

show *PRE-sequence spre upre s' s'*

using *simple-possible-PRE-seq assms(1) assms(2) assms(3) assms(4) assms(6)*

by *blast*

show *all-axioms* α *sact spre uact upre*

using *assms(6)* **by** *auto*

qed

moreover **have** α (*compute sact uact v0' s''*) = α (*compute sact uact v0 s*)

using *PRE-sequence-same-abstract <possible-sequence sargs' uargs' s''* \wedge *PRE-sequence spre upre s s''* *assms(1) assms(5) assms(6)* **by** *metis*

ultimately show *?thesis*

by *auto*

qed

inductive *reachable-value* :: ($'v \Rightarrow 'a \Rightarrow 'v$) \Rightarrow ($'i \Rightarrow 'v \Rightarrow 'b \Rightarrow 'v$) \Rightarrow $'v \Rightarrow 'a$
multiset \Rightarrow ($'i \Rightarrow 'b$ *list*) \Rightarrow $'v \Rightarrow$ *bool* **where**

Self: reachable-value sact uact v0 {#} ($\lambda k. []$) v0
| *SharedStep: reachable-value sact uact v0 sargs uargs v1* \Longrightarrow *reachable-value sact uact v0 (sargs + {# sarg #}) uargs (sact v1 sarg)*
| *UniqueStep: reachable-value sact uact v0 sargs uargs v1* \Longrightarrow *reachable-value sact uact v0 sargs (uargs(k := uarg # uargs k)) (uact k v1 uarg)*

lemma *reachable-then-possible-sequence-and-compute:*

assumes *reachable-value sact uact v0 sargs uargs v1*

shows $\exists s.$ *possible-sequence sargs uargs s* \wedge $v1 =$ *compute sact uact v0 s*

using *assms*

proof (*induct rule: reachable-value.induct*)

case (*Self sact uact v0*)

have *possible-sequence {#} ($\lambda k. []$) []* \wedge $v0 =$ *compute sact uact v0 []*

by (*simp add: possible-sequenceI*)

```

then show ?case by blast
next
  case (SharedStep sact uact v0 sargs uargs v1 sarg)
  then obtain s where possible-sequence sargs uargs s  $\wedge$  v1 = compute sact uact
v0 s by blast
  let ?s = Shared sarg # s
  have possible-sequence (sargs + {#sarg#}) uargs ?s
  proof (rule possible-sequenceI)
    show  $\bigwedge i.$  uargs i = map get-u (filter (is-Unique-i i) (Shared sarg # s))
    by (metis  $\langle$ possible-sequence sargs uargs s  $\wedge$  v1 = compute sact uact v0 s $\rangle$ 
action.disc(1) filter.simps(2) is-Unique-def is-Unique-i-def possible-sequence-def)
    show sargs + {#sarg#} = image-mset get-s (filter-mset is-Shared (mset (Shared
sarg # s)))
    using  $\langle$ possible-sequence sargs uargs s  $\wedge$  v1 = compute sact uact v0 s $\rangle$  possi-
ble-sequence-def by auto
  qed
  then show ?case
  using  $\langle$ possible-sequence sargs uargs s  $\wedge$  v1 = compute sact uact v0 s $\rangle$  by auto
next
  case (UniqueStep sact uact v0 sargs uargs v1 k uarg)
  then obtain s where possible-sequence sargs uargs s  $\wedge$  v1 = compute sact uact
v0 s by blast
  let ?s = Unique k uarg # s
  have possible-sequence sargs (uargs(k := uarg # uargs k)) ?s
  proof (rule possible-sequenceI)
    show  $\bigwedge i.$  (uargs(k := uarg # uargs k)) i = map get-u (filter (is-Unique-i i)
(Unique k uarg # s))
    proof –
    fix i show (uargs(k := uarg # uargs k)) i = map get-u (filter (is-Unique-i i)
(Unique k uarg # s))
    proof (cases i = k)
      case True
      then show ?thesis
      using Cons-eq-map-conv  $\langle$ possible-sequence sargs uargs s  $\wedge$  v1 = compute
sact uact v0 s $\rangle$ 
        action.disc(2) action.sel(2) action.sel(3) filter.simps(2) fun-upd-same
is-Unique-def
        is-Unique-i-def possible-sequence-def[of sargs uargs s]
      by fastforce
    next
    case False
    then show ?thesis
    by (metis  $\langle$ possible-sequence sargs uargs s  $\wedge$  v1 = compute sact uact v0 s $\rangle$ 
action.sel(2) filter.simps(2) fun-upd-other is-Unique-i-def possible-sequence-def)
  qed
  qed
  show sargs = image-mset get-s (filter-mset is-Shared (mset (Unique k uarg #
s)))
  using  $\langle$ possible-sequence sargs uargs s  $\wedge$  v1 = compute sact uact v0 s $\rangle$  possi-

```


ble-sequence-def **by force**
qed
then show *?case* **using** $\langle \text{possible-sequence sargs uargs } s \wedge v1 = \text{compute sact uact } v0 \ s \rangle$
by (*metis compute.simps(3)*)
qed

lemma *PRE-shared-simpler-implies:*
assumes *PRE-shared-simpler spre a b*
shows *PRE-shared spre a b*
using *assms*
proof (*induct rule: PRE-shared-simpler.induct*)
case (*Empty spre*)
then show *?case*
by (*simp add: PRE-shared-def*)
next
case (*Step spre a b xa xb*)
then obtain *ms where image-mset fst ms = a \wedge image-mset snd ms = b \wedge ($\forall x \in \#ms. \text{ spre } (fst \ x) \ (snd \ x)$)*
by (*metis PRE-shared-def*)
then have *image-mset fst (ms + $\{\#(xa, xb) \#\}$) = (a + $\{\#xa\#\}$) \wedge image-mset snd (ms + $\{\#(xa, xb) \#\}$) = (b + $\{\#xb\#\}$) \wedge ($\forall x \in \#(ms + \{\#(xa, xb) \#\})$).
*spre (fst x) (snd x)**
using *Step.hyps(3)* **by auto**
then show *?case* **using** *PRE-shared-def* **by blast**
qed

theorem *main-result:*
assumes *reachable-value sact uact v0 sargs uargs v*
and *reachable-value sact uact v0' sargs' uargs' v'*
and *PRE-shared-simpler spre sargs sargs'*
and $\bigwedge k. \text{ PRE-unique } (upre \ k) \ (uargs \ k) \ (uargs' \ k)$
and $\alpha \ v0 = \alpha \ v0'$
and *all-axioms α sact spre uact upre*
shows $\alpha \ v = \alpha \ v'$
proof –
obtain *s s' where possible-sequence sargs uargs s \wedge v = compute sact uact v0 s*
possible-sequence sargs' uargs' s' \wedge v' = compute sact uact v0' s'
using *assms(1) assms(2) reachable-then-possible-sequence-and-compute*
by *metis*
then show *?thesis*
by (*meson PRE-shared-simpler-implies assms(3) assms(4) assms(5) assms(6)*
main-lemma)
qed

end

4.2 Consistency

theory *Guards*

imports *StateModel CommCSL AbstractCommutativity*

begin

A state is "consistent" iff: 1. All its permissions are full 2. Has unique guards iff has shared guard 3. The values in the fractional heaps are "reachable" wrt to the sequence and multiset of actions 4. Has exactly guards for the names in "scope"

definition *reachable* :: ('i, 'a, 'v) *single-context* \Rightarrow 'v \Rightarrow ('i, 'a) *heap* \Rightarrow *bool* **where**
reachable scont v0 h \longleftrightarrow (\forall sargs uargs. *get-gs* h = *Some* (pwrite, sargs) \wedge (\forall k. *get-gu* h k = *Some* (uargs k))
 \longrightarrow *reachable-value* (*saction scont*) (*uaction scont*) v0 sargs uargs (*view scont* (*normalize* (*get-fh* h))))

lemma *reachableI*:

assumes \bigwedge sargs uargs. *get-gs* h = *Some* (pwrite, sargs) \wedge (\forall k. *get-gu* h k = *Some* (uargs k))
 \implies *reachable-value* (*saction scont*) (*uaction scont*) v0 sargs uargs (*view scont* (*normalize* (*get-fh* h)))
shows *reachable scont* v0 h
by (*metis* *assms* *reachable-def*)

lemma *reachableE*:

assumes *reachable scont* v0 h
and *get-gs* h = *Some* (pwrite, sargs)
and \bigwedge k. *get-gu* h k = *Some* (uargs k)
shows *reachable-value* (*saction scont*) (*uaction scont*) v0 sargs uargs (*view scont* (*normalize* (*get-fh* h)))
by (*meson* *assms* *reachable-def*)

definition *all-guards* :: ('i, 'a) *heap* \Rightarrow *bool* **where**

all-guards h \longleftrightarrow (\exists v. *get-gs* h = *Some* (pwrite, v)) \wedge (\forall k. *get-gu* h k \neq *None*)

lemma *no-guardI*:

assumes *get-gs* h = *None*
and \bigwedge k. *get-gu* h k = *None*
shows *no-guard* h
using *assms*(1) *assms*(2) *no-guard-def* **by** *blast*

definition *semi-consistent* :: ('i, 'a, 'v) *single-context* \Rightarrow 'v \Rightarrow ('i, 'a) *heap* \Rightarrow *bool*
where

semi-consistent Γ v0 h \longleftrightarrow *all-guards* h \wedge *reachable* Γ v0 h

lemma *semi-consistentE*:

assumes *semi-consistent* Γ v0 h
shows \exists sargs uargs. *get-gs* h = *Some* (pwrite, sargs) \wedge (\forall k. *get-gu* h k = *Some* (uargs k))

\wedge *reachable-value* (*saction* Γ) (*uaction* Γ) *v0* *sargs* *uargs* (*view* Γ (*normalize* (*get-fh* *h*)))
proof –
let *?uargs* = $\lambda k. (SOME\ x. \text{get-gu } h\ k = Some\ x)$
have $\wedge k. \text{get-gu } h\ k = Some\ (?uargs\ k)$
proof –
fix *k* **have** $\exists x. \text{get-gu } h\ k = Some\ x$
by (*meson all-guards-def assms option.exhaust-sel semi-consistent-def*)
then show $\text{get-gu } h\ k = Some\ (?uargs\ k)$
by *fastforce*
qed
moreover obtain *sargs* **where** $\text{get-gs } h = Some\ (pwrite, sargs)$
by (*meson all-guards-def assms semi-consistent-def*)
ultimately have *reachable-value* (*saction* Γ) (*uaction* Γ) *v0* *sargs* *?uargs* (*view* Γ (*normalize* (*get-fh* *h*)))
by (*meson assms reachableE semi-consistent-def*)
then show *?thesis*
using $\langle \wedge k. \text{get-gu } h\ k = Some\ (SOME\ x. \text{get-gu } h\ k = Some\ x) \rangle \langle \text{get-gs } h = Some\ (pwrite, sargs) \rangle$ **by** *fastforce*
qed

lemma *semi-consistentI*:
assumes *all-guards* *h*
and *reachable* Γ *v0* *h*
shows *semi-consistent* Γ *v0* *h*
by (*simp add: assms(1) assms(2) semi-consistent-def*)

lemma *no-guard-then-smaller-same*:
assumes $Some\ h = Some\ a \oplus Some\ b$
and *no-guard* *h*
shows *no-guard* *a*
proof (*rule no-guardI*)
show $\text{get-gs } a = None$
by (*metis add-gs.elims assms(1) assms(2) no-guard-def option.simps(3) plus-extract(2)*)
fix *k*
have $\text{get-gu } h\ k = None$
by (*meson assms(2) no-guard-def*)
then show $\text{get-gu } a\ k = None$
by (*metis assms(1) full-uguard-sum-same option.exhaust*)
qed

lemma *all-guardsI*:
assumes $\wedge k. \text{get-gu } h\ k \neq None$
and $\exists v. \text{get-gs } h = Some\ (pwrite, v)$
shows *all-guards* *h*
using *all-guards-def assms(1) assms(2)* **by** *blast*

lemma *all-guards-same*:
assumes *all-guards* *a*

and $\text{Some } h = \text{Some } a \oplus \text{Some } b$
shows $\text{all-guards } h$
proof (*rule all-guardsI*)
show $\exists v. \text{get-gs } h = \text{Some } (\text{pwrite}, v)$
using $\text{all-guards-def } \text{assms}(1) \text{ assms}(2) \text{ full-sguard-sum-same}$ **by** *blast*
fix k **have** $\text{get-gu } a \ k \neq \text{None}$
by (*meson all-guards-def assms(1)*)
then show $\text{get-gu } h \ k \neq \text{None}$
apply (*cases get-gu b k*)
apply (*metis assms(2) full-uguard-sum-same not-Some-eq*)
by (*metis assms(2) full-uguard-sum-same option.discI plus-comm*)
qed

definition $\text{empty-unique where}$
 $\text{empty-unique } - = \text{None}$

definition $\text{remove-guards} :: ('i, 'a) \text{ heap} \Rightarrow ('i, 'a) \text{ heap}$ **where**
 $\text{remove-guards } h = (\text{get-fh } h, \text{None}, \text{empty-unique})$

lemma $\text{remove-guards-smaller}$:

$h \succeq \text{remove-guards } h$

proof –

have $\text{remove-guards } h \ \#\# \ (\text{Map.empty}, \text{get-gs } h, \text{get-gu } h)$

proof (*rule compatibleI*)

show $\text{compatible-fract-heaps } (\text{get-fh } (\text{remove-guards } h)) \ (\text{get-fh } (\text{Map.empty}, \text{get-gs } h, \text{get-gu } h))$

using $\text{compatible-fract-heapsI}$ **by** *force*

show $\bigwedge k. \text{get-gu } (\text{remove-guards } h) \ k = \text{None} \vee \text{get-gu } (\text{Map.empty}, \text{get-gs } h, \text{get-gu } h) \ k = \text{None}$

by (*simp add: empty-unique-def remove-guards-def*)

show $\bigwedge p \ p'. \text{get-gs } (\text{remove-guards } h) = \text{Some } p \wedge \text{get-gs } (\text{Map.empty}, \text{get-gs } h, \text{get-gu } h) = \text{Some } p' \implies \text{pgte } \text{pwrite } (\text{padd } (\text{fst } p) \ (\text{fst } p'))$

by (*simp add: remove-guards-def*)

qed

then obtain x **where** $\text{Some } x = \text{Some } (\text{remove-guards } h) \oplus \text{Some } (\text{Map.empty}, \text{get-gs } h, \text{get-gu } h)$

by *auto*

moreover have $x = h$

proof (*rule heap-ext*)

show $\text{get-fh } x = \text{get-fh } h$

by (*metis add-fh-map-empty add-get-fh calculation fst-conv get-fh.elims remove-guards-def*)

show $\text{get-gs } x = \text{get-gs } h$

by (*metis calculation fst-eqD get-gs.elims plus-comm remove-guards-def snd-eqD sum-gs-one-none*)

show $\text{get-gu } x = \text{get-gu } h$

proof (*rule ext*)

fix k

have $\text{get-gu } (\text{remove-guards } h) \ k = \text{None}$

```

    by (simp add: empty-unique-def remove-guards-def)
  then show  $get-gu\ x\ k = get-gu\ h\ k$ 
    by (metis (mono-tags, lifting) add-gu-def add-gu-single.simps(1) calculation
  get-gu.elims plus-extract(3) snd-eqD)
  qed
  qed
  ultimately show ?thesis
    using larger-def by blast
  qed

```

```

lemma no-guard-remove:
  assumes  $Some\ a = Some\ b \oplus Some\ c$ 
    and no-guard c
  shows  $get-gs\ a = get-gs\ b$ 
    and  $get-gu\ a = get-gu\ b$ 
  using assms(1) assms(2) no-guard-def sum-gs-one-none apply blast
proof (rule ext)
  fix k
  have  $get-gu\ c\ k = None$ 
    by (meson assms(2) no-guard-def)
  then show  $get-gu\ a\ k = get-gu\ b\ k$ 
    by (metis (no-types, lifting) add-gu-def add-gu-single.simps(1) assms(1) plus-comm
  plus-extract(3))
  qed

```

```

lemma full-guard-comp-then-no:
  assumes  $a \#\# b$ 
    and all-guards a
  shows no-guard b
proof (rule no-guardI)
  show  $\bigwedge k. get-gu\ b\ k = None$ 
    by (meson all-guards-def assms(1) assms(2) compatible-def)
  show  $get-gs\ b = None$ 
  proof (rule ccontr)
    assume  $get-gs\ b \neq None$ 
    then obtain gb where  $get-gs\ b = Some\ gb$ 
      by blast
    moreover obtain v where  $get-gs\ a = Some\ (pwrite, v)$ 
      by (meson all-guards-def assms(2))
    moreover have pgt (padd pwrite (fst gb)) pwrite
      using sum-larger by auto
    ultimately show False
      by (metis assms(1) compatible-def fst-eqD not-pgte-charact)
  qed
  qed

```

```

lemma sum-of-no-guards:
  assumes no-guard a
    and no-guard b

```

and $\text{Some } x = \text{Some } a \oplus \text{Some } b$
shows *no-guard* x
by (*metis* *assms*(1) *assms*(2) *assms*(3) *no-guard-def* *no-guard-remove*(1) *no-guard-remove*(2))

lemma *no-guard-remove-guards*:
no-guard (*remove-guards* h)
by (*simp* *add*: *empty-unique-def* *no-guard-def* *remove-guards-def*)

lemma *get-fh-remove-guards*:
get-fh (*remove-guards* h) = *get-fh* h
by (*simp* *add*: *remove-guards-def*)

definition *pair-sat* :: (store \times ('i, 'a) heap) set \Rightarrow (store \times ('i, 'a) heap) set \Rightarrow
('i, 'a, nat) assertion \Rightarrow bool **where**
pair-sat $S S' Q \longleftrightarrow (\forall \sigma \sigma'. \sigma \in S \wedge \sigma' \in S' \longrightarrow \sigma, \sigma' \models Q)$

lemma *pair-satI*:
assumes $\bigwedge s h s' h'. (s, h) \in S \wedge (s', h') \in S' \Longrightarrow (s, h), (s', h') \models Q$
shows *pair-sat* $S S' Q$
by (*simp* *add*: *assms* *pair-sat-def*)

lemma *pair-sat-smallerI*:
assumes $\bigwedge \sigma \sigma'. \sigma \in S \wedge \sigma' \in S' \Longrightarrow \sigma, \sigma' \models Q$
shows *pair-sat* $S S' Q$
by (*simp* *add*: *assms* *pair-sat-def*)

lemma *pair-satE*:
assumes *pair-sat* $S S' Q$
and $(s, h) \in S \wedge (s', h') \in S'$
shows $(s, h), (s', h') \models Q$
using *assms*(1) *assms*(2) *pair-sat-def* **by** *blast*

definition *add-states* :: (store \times ('i, 'a) heap) set \Rightarrow (store \times ('i, 'a) heap) set \Rightarrow
(store \times ('i, 'a) heap) set **where**
add-states $S1 S2 = \{(s, H) \mid s H h1 h2. \text{Some } H = \text{Some } h1 \oplus \text{Some } h2 \wedge (s, h1) \in S1 \wedge (s, h2) \in S2\}$

lemma *add-states-sat-star*:
assumes *pair-sat* $SA SA' A$
and *pair-sat* $SB SB' B$
shows *pair-sat* (*add-states* $SA SB$) (*add-states* $SA' SB'$) (*Star* $A B$)
proof (*rule* *pair-satI*)
fix $s h s' h'$
assume *asm0*: $(s, h) \in \text{add-states } SA SB \wedge (s', h') \in \text{add-states } SA' SB'$
then obtain $ha hb ha' hb'$ **where** $(s, ha) \in SA (s, hb) \in SB (s', ha') \in SA' (s', hb') \in SB'$
 $\text{Some } h = \text{Some } ha \oplus \text{Some } hb \text{ Some } h' = \text{Some } ha' \oplus \text{Some } hb'$
using *add-states-def*[of $SA SB$] *add-states-def*[of $SA' SB'$] *fst-eqD* *mem-Collect-eq* *snd-conv*

by *auto*
 then show $(s, h), (s', h') \models \text{Star } A \ B$
 by (*meson* *assms*(1) *assms*(2) *hyper-sat.simps*(4) *pair-sat-def*)
 qed

lemma *add-states-subset*:
 assumes $S1 \subseteq S1'$
 shows $\text{add-states } S1 \ S2 \subseteq \text{add-states } S1' \ S2$
proof
 fix x assume $x \in \text{add-states } S1 \ S2$
 then show $x \in \text{add-states } S1' \ S2$
 using *add-states-def*[of $S1 \ S2$] *add-states-def*[of $S1' \ S2$] *assms mem-Collect-eq*[of
 x] *subsetD*[of $S1 \ S1'$]
 by *blast*
 qed

lemma *add-states-comm*:
 $\text{add-states } S1 \ S2 = \text{add-states } S2 \ S1$
proof –
 have $\bigwedge S1 \ S2. \text{add-states } S1 \ S2 \subseteq \text{add-states } S2 \ S1$
proof –
 fix $S1 \ S2$
 show $\text{add-states } S1 \ S2 \subseteq \text{add-states } S2 \ S1$
proof
 fix x assume $x \in \text{add-states } S1 \ S2$
 then obtain $h1 \ h2$ where $\text{Some } (snd \ x) = \text{Some } h1 \oplus \text{Some } h2$ (*fst* $x, h1$)
 $\in S1$ (*fst* $x, h2$) $\in S2$
 using *add-states-def*[of $S1 \ S2$] *fst-conv mem-Collect-eq*[of x] *snd-eqD*
 by *auto*
 moreover have $\text{Some } (snd \ x) = \text{Some } h2 \oplus \text{Some } h1$
 by (*simp add: calculation*(1) *plus-comm*)
 ultimately show $x \in \text{add-states } S2 \ S1$
 using *add-states-def*[of $S2 \ S1$] *mem-Collect-eq*[of x] *surjective-pairing*[of x]
 by *blast*
 qed
 qed
 then show *?thesis* by *blast*
 qed

lemma *magic-lemma*:
 assumes $\text{Some } x1 = \text{Some } a1 \oplus \text{Some } j1$
 and $\text{Some } x2 = \text{Some } a2 \oplus \text{Some } j2$
 and $(s1, x1), (s2, x2) \models \text{Star } A \ J$
 and $(s1, j1), (s2, j2) \models J$
 and *precise* J
 shows $(s1, a1), (s2, a2) \models A$
proof –
 obtain $a1' \ a2' \ j1' \ j2'$ where $\text{Some } x1 = \text{Some } a1' \oplus \text{Some } j1'$
 $\text{Some } x2 = \text{Some } a2' \oplus \text{Some } j2'$ ($s1, j1'$), ($s2, j2'$) $\models J$ ($s1, a1'$), ($s2, a2'$)

```

|= A
  using assms(3) hyper-sat.simps(4) by blast
  have j1 = j1' ∧ j2 = j2'
  using assms(5)
  proof (rule preciseE)
    show x1 ≥ j1' ∧ x1 ≥ j1 ∧ x2 ≥ j2' ∧ x2 ≥ j2
      by (metis ‹Some x1 = Some a1' ⊕ Some j1'› ‹Some x2 = Some a2' ⊕ Some
j2'› assms(1) assms(2) larger-def plus-comm)
    show (s1, j1'), (s2, j2') ⊨ J ∧ (s1, j1), (s2, j2) ⊨ J
      by (simp add: ‹(s1, j1'), (s2, j2') ⊨ J› assms(4))
  qed
  then have a1 = a1' ∧ a2 = a2'
  using ‹Some x1 = Some a1' ⊕ Some j1'› ‹Some x2 = Some a2' ⊕ Some j2'›
addition-cancellative assms(1) assms(2) by blast
  then show ?thesis
  using ‹(s1, a1'), (s2, a2') ⊨ A› by blast
qed

```

lemma *full-no-guard-same-normalize*:

```

  assumes full-ownership (get-fh h) ∧ no-guard h
    and full-ownership (get-fh h') ∧ no-guard h'
    and normalize (get-fh h) = normalize (get-fh h')
  shows h = h'
proof (rule heap-ext)
  show get-gu h = get-gu h'
  apply (rule ext)
  by (metis assms(1) assms(2) no-guard-def)
  show get-gs h = get-gs h'
  by (metis assms(1) assms(2) no-guard-def)
  show get-fh h = get-fh h'
  proof (rule ext)
    fix l show get-fh h l = get-fh h' l
    apply (cases get-fh h l)
    apply (metis FractionalHeap.normalize-eq(1) assms(3))
    apply (cases get-fh h' l)
    apply (metis FractionalHeap.normalize-eq(1) assms(3))
  by (metis FractionalHeap.normalize-def apply-opt.simps(2) assms(1) assms(2)
assms(3) full-ownership-def prod.collapse)
  qed
qed

```

lemma *get-fh-same-then-remove-guards-same*:

```

  assumes get-fh a = get-fh b
  shows remove-guards a = remove-guards b
  by (metis assms remove-guards-def)

```

lemma *remove-guards-sum*:

assumes $\text{Some } x = \text{Some } a \oplus \text{Some } b$
shows $\text{Some } (\text{remove-guards } x) = \text{Some } (\text{remove-guards } a) \oplus \text{Some } (\text{remove-guards } b)$
proof –
have $\text{remove-guards } a \#\#\text{remove-guards } b$
by (*metis* (*no-types*, *lifting*) *assms compatible-def compatible-eq get-fh-remove-guards no-guard-def no-guard-remove-guards option.distinct(1)*)
then obtain y **where** $\text{Some } y = \text{Some } (\text{remove-guards } a) \oplus \text{Some } (\text{remove-guards } b)$
by *auto*
moreover have $\text{remove-guards } x = y$
by (*metis* (*no-types*, *lifting*) $\langle \text{remove-guards } a \#\#\text{remove-guards } b \rangle$ *add-get-fh assms calculation get-fh-remove-guards get-gu.simps no-guard-def no-guard-remove(1) no-guard-remove(2) no-guard-remove-guards option.inject plus.simps(3) plus-extract(2) remove-guards-def snd-eqD*)
ultimately show *?thesis* **by** *blast*
qed

lemma *no-guard-smaller*:
assumes $a \succeq b$
shows $\text{remove-guards } a \succeq \text{remove-guards } b$
using *assms larger-def remove-guards-sum* **by** *blast*

definition *add-empty-guards* :: $('i, 'a) \text{ heap} \Rightarrow ('i, 'a) \text{ heap}$ **where**
 $\text{add-empty-guards } h = (\text{get-fh } h, \text{Some } (\text{pwrite}, \{\#\}), (\lambda_. \text{Some } []))$

lemma *no-guard-map-empty-compatible*:
assumes *no-guard* a
and $\text{get-fh } b = \text{Map.empty}$
shows $a \#\#\text{ } b$
by (*metis* (*no-types*, *lifting*) *assms(1) assms(2) compatible-def compatible-fract-heapsI no-guard-def option.simps(3)*)

lemma *no-guard-add-empty-is-add*:
assumes *no-guard* h
shows $\text{Some } (\text{add-empty-guards } h) = \text{Some } h \oplus \text{Some } (\text{Map.empty}, \text{Some } (\text{pwrite}, \{\#\}), (\lambda_. \text{Some } []))$
proof –
obtain x **where** $\text{Some } x = \text{Some } h \oplus \text{Some } (\text{Map.empty}, \text{Some } (\text{pwrite}, \{\#\}), (\lambda_. \text{Some } []))$
by (*simp add: assms no-guard-map-empty-compatible*)
moreover have $\text{add-empty-guards } h = x$
proof (*rule heap-ext*)
show $\text{get-fh } (\text{add-empty-guards } h) = \text{get-fh } x$
by (*metis add-empty-guards-def add-fh-map-empty add-get-fh calculation fst-conv get-fh.elims*)
show $\text{get-gs } (\text{add-empty-guards } h) = \text{get-gs } x$
by (*metis add-empty-guards-def assms calculation get-gs.elims no-guard-remove(1) plus-comm snd-eqD*)

```

  show get-gu (add-empty-guards h) = get-gu x
  by (metis add-empty-guards-def assms calculation get-gu.elims no-guard-remove(2)
plus-comm snd-eqD)
qed
ultimately show ?thesis by blast
qed

```

lemma *no-guard-and-sat-p-empty-guards:*

```

  assumes (s, h), (s', h') ⊨ A
  and no-guard h ∧ no-guard h'
  shows (s, add-empty-guards h), (s', add-empty-guards h') ⊨ Star A EmptyFull-
Guards
proof -
  have (s, (Map.empty, Some (pwrite, {#}), (λ-. Some []))), (s', (Map.empty,
Some (pwrite, {#}), (λ-. Some []))) ⊨ EmptyFullGuards
  by simp
  then show ?thesis
  using assms(1) assms(2) hyper-sat.simps(4) no-guard-add-empty-is-add by
blast
qed

```

lemma *no-guard-add-empty-guards-sum:*

```

  assumes no-guard x
  and Some x = Some a ⊕ Some b
  shows Some (add-empty-guards x) = Some (add-empty-guards a) ⊕ Some b
  using assms(1) assms(2) no-guard-add-empty-is-add[of a] no-guard-add-empty-is-add[of
x]
no-guard-then-smaller-same[of x a b] plus-asso plus-comm
  by (metis (no-types, lifting))

```

lemma *semi-consistent-empty-no-guard-initial-value:*

```

  assumes no-guard h
  shows semi-consistent Γ (view Γ (FractionalHeap.normalize (get-fh h))) (add-empty-guards
h)
proof (rule semi-consistentI)
  show all-guards (add-empty-guards h)
  by (simp add: add-empty-guards-def all-guards-def)
  show reachable Γ (view Γ (FractionalHeap.normalize (get-fh h))) (add-empty-guards
h)
  proof (rule reachableI)
    fix sargs uargs
    assume asm0: get-gs (add-empty-guards h) = Some (pwrite, sargs) ∧ (∀ k.
get-gu (add-empty-guards h) k = Some (uargs k))
    then have sargs = {#} ∧ uargs = (λk. [])
    by (metis add-empty-guards-def fst-conv get-gs.simps get-gu.simps option.sel
snd-conv)
    then show reachable-value (saction Γ) (uaction Γ) (view Γ (FractionalHeap.normalize
(get-fh h))) sargs uargs
    (view Γ (FractionalHeap.normalize (get-fh (add-empty-guards h))))
  qed

```

by (*simp add: Self add-empty-guards-def*)
qed
qed

lemma *no-guards-remove-same*:

assumes *no-guard h*
shows $h = \text{remove-guards } (\text{add-empty-guards } h)$
by (*metis add-empty-guards-def addition-cancellative assms fst-conv get-fh.elims get-fh-remove-guards no-guard-add-empty-is-add no-guard-remove-guards*)

lemma *no-guards-remove*:

$\text{no-guard } h \longleftrightarrow h = \text{remove-guards } h$
by (*metis get-fh-remove-guards no-guard-remove-guards no-guards-remove-same remove-guards-def*)

definition *add-sguard-to-no-guard* :: $('i, 'a) \text{ heap} \Rightarrow \text{prat} \Rightarrow 'a \text{ multiset} \Rightarrow ('i, 'a) \text{ heap}$ **where**

$\text{add-sguard-to-no-guard } h \ \pi \ ms = (\text{get-fh } h, \text{Some } (\pi, ms), (\lambda-. \text{None}))$

lemma *get-fh-add-sguard*:

$\text{get-fh } (\text{add-sguard-to-no-guard } h \ \pi \ ms) = \text{get-fh } h$
by (*simp add: add-sguard-to-no-guard-def*)

lemma *add-sguard-as-sum*:

assumes *no-guard h*
shows $\text{Some } (\text{add-sguard-to-no-guard } h \ \pi \ ms) = \text{Some } h \oplus \text{Some } (\text{Map.empty}, \text{Some } (\pi, ms), (\lambda-. \text{None}))$

proof –

obtain x **where** $\text{Some } x = \text{Some } h \oplus \text{Some } (\text{Map.empty}, \text{Some } (\pi, ms), (\lambda-. \text{None}))$

by (*simp add: assms no-guard-map-empty-compatible*)

moreover have $x = \text{add-sguard-to-no-guard } h \ \pi \ ms$

proof (*rule heap-ext*)

show $\text{get-fh } x = \text{get-fh } (\text{add-sguard-to-no-guard } h \ \pi \ ms)$

by (*metis add-fh-map-empty add-get-fh calculation fst-conv get-fh.elims get-fh-add-sguard*)

show $\text{get-gs } x = \text{get-gs } (\text{add-sguard-to-no-guard } h \ \pi \ ms)$

by (*metis add-sguard-to-no-guard-def assms calculation get-gs.elims no-guard-def plus-comm snd-eqD sum-gs-one-none*)

show $\text{get-gu } x = \text{get-gu } (\text{add-sguard-to-no-guard } h \ \pi \ ms)$

by (*metis add-sguard-to-no-guard-def assms calculation get-gu.simps no-guard-remove(2) plus-comm snd-conv*)

qed

ultimately show *?thesis* **by** *blast*

qed

definition *add-uguard-to-no-guard* :: $'i \Rightarrow ('i, 'a) \text{ heap} \Rightarrow 'a \text{ list} \Rightarrow ('i, 'a) \text{ heap}$ **where**

$\text{add-uguard-to-no-guard } k \ h \ l = (\text{get-fh } h, \text{None}, (\lambda-. \text{None}))(k := \text{Some } l)$

lemma *get-fh-add-uguard*:
 $get\text{-}fh\ (add\text{-}uguard\text{-}to\text{-}no\text{-}guard\ k\ h\ l) = get\text{-}fh\ h$
by (*simp add: add-uguard-to-no-guard-def*)

lemma *prove-sum*:
assumes $a \#\# b$
and $\bigwedge x. Some\ x = Some\ a \oplus Some\ b \implies x = y$
shows $Some\ y = Some\ a \oplus Some\ b$
using *assms(1) assms(2)* **by** *fastforce*

lemma *add-uguard-as-sum*:
assumes *no-guard h*
shows $Some\ (add\text{-}uguard\text{-}to\text{-}no\text{-}guard\ k\ h\ l) = Some\ h \oplus Some\ (Map.empty,$
None, (\lambda-. None)(k := Some l))
proof (*rule prove-sum*)
show $h \#\# (Map.empty, None, [k \mapsto l])$
by (*simp add: assms no-guard-map-empty-compatible*)
fix x **assume** *asm0: Some x = Some h \oplus Some (Map.empty, None, [k \mapsto l])*
show $x = add\text{-}uguard\text{-}to\text{-}no\text{-}guard\ k\ h\ l$
proof (*rule heap-ext*)
show $get\text{-}fh\ x = get\text{-}fh\ (add\text{-}uguard\text{-}to\text{-}no\text{-}guard\ k\ h\ l)$
by (*metis add-fh-map-empty add-get-fh asm0 fst-conv get-fh.elims get-fh-add-uguard*)
show $get\text{-}gs\ x = get\text{-}gs\ (add\text{-}uguard\text{-}to\text{-}no\text{-}guard\ k\ h\ l)$
by (*metis add-uguard-to-no-guard-def asm0 assms get-gs.elims no-guard-def*
plus-comm snd-eqD sum-gs-one-none)
show $get\text{-}gu\ x = get\text{-}gu\ (add\text{-}uguard\text{-}to\text{-}no\text{-}guard\ k\ h\ l)$
by (*metis add-uguard-to-no-guard-def asm0 assms get-gu.elims no-guard-remove(2)*
plus-comm snd-eqD)
qed
qed

lemma *no-guard-and-no-heap*:
assumes $Some\ h = Some\ p \oplus Some\ g$
and *no-guard p*
and $get\text{-}fh\ g = Map.empty$
shows $remove\text{-}guards\ h = p$
proof (*rule heap-ext*)
show $get\text{-}fh\ (remove\text{-}guards\ h) = get\text{-}fh\ p$
proof –
have $get\text{-}fh\ (remove\text{-}guards\ h) = get\text{-}fh\ h$
using *get-fh-remove-guards* **by** *blast*
moreover **have** $get\text{-}fh\ h = add\text{-}fh\ (get\text{-}fh\ p)\ (get\text{-}fh\ g)$
using *add-get-fh assms(1)* **by** *blast*
ultimately **show** *?thesis*
by (*metis assms(1) assms(3) ext get-fh.simps sum-second-none-get-fh*)
qed

```

show get-gs (remove-guards h) = get-gs p
  by (metis assms(2) no-guard-def no-guard-remove-guards)
show get-gu (remove-guards h) = get-gu p
  by (metis ⟨get-fh (remove-guards h) = get-fh p⟩ assms(2) get-fh-remove-guards
no-guards-remove remove-guards-def)
qed

```

```

lemma decompose-guard-remove-easy:
  Some h = Some (remove-guards h)  $\oplus$  Some (Map.empty, get-gs h, get-gu h)
proof (rule prove-sum)
  show remove-guards h  $\#\#$  (Map.empty, get-gs h, get-gu h)
    by (simp add: no-guard-map-empty-compatible no-guard-remove-guards)
  fix x assume asm0: Some x = Some (remove-guards h)  $\oplus$  Some (Map.empty,
get-gs h, get-gu h)
  show x = h
  proof (rule heap-ext)
    show get-fh x = get-fh h
    by (metis add-fh-map-empty add-get-fh asm0 fst-conv get-fh.elims get-fh-remove-guards)
    show get-gs x = get-gs h
    by (metis asm0 fst-conv get-gs.simps no-guard-remove(1) no-guard-remove-guards
plus-comm snd-conv)
    show get-gu x = get-gu h
    by (metis asm0 get-gu.elims no-guard-remove(2) no-guard-remove-guards
plus-comm snd-eqD)
  qed
qed

```

```

lemma all-guards-no-guard-propagates:
  assumes all-guards x
    and Some x = Some a  $\oplus$  Some b
    and no-guard a
  shows all-guards b
  by (metis all-guards-def assms(1) assms(2) assms(3) no-guard-def no-guard-remove(2)
plus-comm sum-gs-one-none)

```

```

lemma all-guards-exists-uargs:
  assumes all-guards x
  shows  $\exists$  uargs.  $\forall$  k. get-gu x k = Some (uargs k)
proof –
  let ?uargs =  $\lambda$ k. the (get-gu x k)
  have  $\bigwedge$ k. get-gu x k = Some (?uargs k)
    by (metis all-guards-def assms option.collapse)
  then show ?thesis
    by fastforce
qed

```

```

lemma all-guards-sum-known-one:

```

```

assumes Some x = Some a ⊕ Some b
  and all-guards x
  and  $\bigwedge k. \text{get-gu } a \ k = \text{None}$ 
  and get-gs a = Some (π, ms)
  shows  $\exists \pi' \text{ msf } \text{uargs}. (\forall k. \text{get-gu } b \ k = \text{Some } (\text{uargs } k)) \wedge$ 
     $((\pi = \text{pwrite} \wedge \text{get-gs } b = \text{None} \wedge \text{msf} = \{\#\}) \vee (\text{pwrite} = \text{padd } \pi \ \pi' \wedge \text{get-gs}$ 
     $b = \text{Some } (\pi', \text{msf})))$ 
proof (cases π = pwrite)
  case True
    then have get-gs b = None
      using add-gs.simps(2)[of (π, ms)] add-gs-cancellative add-gs-comm assms(1)
assms(4) full-sguard-sum-same
      plus-extract(2)[of x a b]
      by metis
    moreover obtain uargs where  $\bigwedge k. \text{get-gu } x \ k = \text{Some } (\text{uargs } k)$ 
      using all-guards-exists-uargs assms(2) by blast
    moreover have  $\bigwedge k. \text{get-gu } b \ k = \text{Some } (\text{uargs } k)$ 
    proof –
      fix k
      have get-gu a k = None
        using assms(3) by auto
      then show get-gu b k = Some (uargs k)
        by (metis (no-types, opaque-lifting) add-gu-def add-gu-single.simps(1) assms(1)
calculation(2) plus-extract(3))
      qed
    ultimately show ?thesis
      using True by blast
  next
    case False
    then obtain  $\pi' \text{ msf}$  where get-gs b = Some (π', msf)
      by (metis all-guards-def assms(1) assms(2) assms(4) fst-conv option.exhaust-sel
option.sel prod.exhaust-sel sum-gs-one-none)
    moreover obtain v where get-gs x = Some (pwrite, v)
      by (meson all-guards-def assms(2))
    ultimately have pwrite = padd π π'
      by (metis Pair-inject assms(1) assms(4) option.inject sum-gs-one-some)
    then show ?thesis
      by (metis (mono-tags, opaque-lifting) ⟨get-gs b = Some (π', msf)⟩ add-gu-def
add-gu-single.simps(1) all-guards-exists-uargs assms(1) assms(2) assms(3) plus-extract(3))
    qed

```

```

fun add-pwrite-option where
  add-pwrite-option None = None
| add-pwrite-option (Some x) = Some (pwrite, x)

```

```

definition denormalize :: normal-heap ⇒ ('i, 'a) heap where
  denormalize H = ((λl. add-pwrite-option (H l)), None, (λ-. None))

```

```

lemma denormalize-properties:

```

```

shows no-guard (denormalize H)
  and full-ownership (get-fh (denormalize H))
  and normalize (get-fh (denormalize H)) = H
  and full-ownership (get-fh h)  $\wedge$  no-guard h  $\implies$  denormalize (normalize (get-fh
h)) = h
  and full-ownership (get-fh h)  $\implies$  denormalize (normalize (get-fh h)) = re-
move-guards h
  apply (simp add: denormalize-def no-guardI)
  using full-ownershipI[of get-fh (denormalize H)] add-pwrite-option.elims denor-
malize-def fst-conv get-fh.elims option.distinct(1) option.sel apply metis
proof -
  show normalize (get-fh (denormalize H)) = H
  proof (rule ext)
    fix l show normalize (get-fh (denormalize H)) l = H l
    by (metis FractionalHeap.normalize-eq(1) FractionalHeap.normalize-eq(2)
add-pwrite-option.elims denormalize-def fst-conv get-fh.elims)
  qed
  show full-ownership (get-fh h)  $\wedge$  no-guard h  $\implies$  denormalize (FractionalHeap.normalize
(get-fh h)) = h
  proof -
    assume asm0: full-ownership (get-fh h)  $\wedge$  no-guard h
    show denormalize (FractionalHeap.normalize (get-fh h)) = h
    proof (rule heap-ext)
      show get-fh (denormalize (FractionalHeap.normalize (get-fh h))) = get-fh h
      proof (rule ext)
        fix x show get-fh (denormalize (FractionalHeap.normalize (get-fh h))) x =
get-fh h x
        proof (cases get-fh h x)
          case None
            then show ?thesis
              by (metis FractionalHeap.normalize-eq(1) add-pwrite-option.simps(1)
denormalize-def fst-conv get-fh.elims)
          next
            case (Some p)
              then have fst p = pwrite
                by (meson asm0 full-ownership-def)
              then show ?thesis
                by (metis FractionalHeap.normalize-eq(2) Some add-pwrite-option.simps(2)
denormalize-def fst-conv get-fh.elims prod.collapse)
        qed
      qed
      show get-gs (denormalize (FractionalHeap.normalize (get-fh h))) = get-gs h
        by (metis asm0 denormalize-def fst-conv get-gs.elims no-guard-def snd-eqD)
      show get-gu (denormalize (FractionalHeap.normalize (get-fh h))) = get-gu h
        by (metis  $\langle$ get-fh (denormalize (FractionalHeap.normalize (get-fh h))) = get-fh
h $\rangle$   $\langle$ get-gs (denormalize (FractionalHeap.normalize (get-fh h))) = get-gs h $\rangle$  asm0
denormalize-def full-no-guard-same-normalize get-gu.simps no-guard-def snd-conv)
      qed
    qed
  qed

```

```

assume asm0: full-ownership (get-fh h)
show denormalize (FractionalHeap.normalize (get-fh h)) = remove-guards h
proof (rule heap-ext)
  show get-fh (denormalize (FractionalHeap.normalize (get-fh h))) = get-fh (remove-guards
h)
  proof (rule ext)
    fix x show get-fh (denormalize (FractionalHeap.normalize (get-fh h))) x =
get-fh (remove-guards h) x
    proof (cases get-fh h x)
      case None
      then show ?thesis
        by (metis FractionalHeap.normalize-eq(1) add-pwrite-option.simps(1)
denormalize-def fst-eqD get-fh.elims get-fh-remove-guards)
      next
      case (Some p)
      then have fst p = pwrite
        by (meson asm0 full-ownership-def)
      then show ?thesis
        by (metis FractionalHeap.normalize-eq(2) Some add-pwrite-option.simps(2)
denormalize-def fst-conv get-fh.elims get-fh-remove-guards prod.collapse)
      qed
    qed
  show get-gs (denormalize (FractionalHeap.normalize (get-fh h))) = get-gs
(remove-guards h)
  by (simp add: denormalize-def remove-guards-def)
  show get-gu (denormalize (FractionalHeap.normalize (get-fh h))) = get-gu
(remove-guards h)
  by (metis ⟨get-fh (denormalize (FractionalHeap.normalize (get-fh h))) = get-fh
(remove-guards h)⟩ ⟨get-gs (denormalize (FractionalHeap.normalize (get-fh h))) =
get-gs (remove-guards h)⟩ asm0 denormalize-def full-no-guard-same-normalize get-fh-remove-guards
get-gu.simps no-guard-def no-guard-remove-guards snd-conv)
  qed
qed

```

lemma *no-guard-then-sat-star-uguard*:

```

assumes no-guard h ∧ no-guard h'
  and (s, h), (s', h') ⊨ Q
  shows (s, add-uguard-to-no-guard k h (e s)), (s', add-uguard-to-no-guard k h'
(e s')) ⊨ Star Q (UniqueGuard k e)
proof –
  obtain Some (add-uguard-to-no-guard k h (e s)) = Some h ⊕ Some (Map.empty,
None, [k ↦ e s])
    Some (add-uguard-to-no-guard k h' (e s')) = Some h' ⊕ Some (Map.empty,
None, [k ↦ e s'])
  by (simp add: add-uguard-as-sum assms(1))
  moreover have (s, (Map.empty, None, [k ↦ e s])), (s', (Map.empty, None, [k
↦ e s'])) ⊨ UniqueGuard k e
  by simp
  ultimately show ?thesis using assms(2) by fastforce

```


qed

lemma *no-guard-then-sat-star*:

assumes *no-guard h* \wedge *no-guard h'*

and $(s, h), (s', h') \models Q$

shows $(s, \text{add-sguard-to-no-guard } h \ \pi \ (ms \ s)), (s', \text{add-sguard-to-no-guard } h' \ \pi \ (ms \ s')) \models \text{Star } Q \ (\text{SharedGuard } \pi \ ms)$

proof –

obtain $\text{Some } (\text{add-sguard-to-no-guard } h \ \pi \ (ms \ s)) = \text{Some } h \oplus \text{Some } (\text{Map.empty}, \text{Some } (\pi, ms \ s), (\lambda_. \text{None}))$

$\text{Some } (\text{add-sguard-to-no-guard } h' \ \pi \ (ms \ s')) = \text{Some } h' \oplus \text{Some } (\text{Map.empty}, \text{Some } (\pi, ms \ s'), (\lambda_. \text{None}))$

using *add-sguard-as-sum assms(1)* **by** *blast*

moreover have $(s, (\text{Map.empty}, \text{Some } (\pi, ms \ s), (\lambda_. \text{None}))), (s', (\text{Map.empty}, \text{Some } (\pi, ms \ s'), (\lambda_. \text{None}))) \models \text{SharedGuard } \pi \ ms$

by *simp*

ultimately show *?thesis* **using** *assms(2)* **by** *fastforce*

qed

end

4.3 Safety and Hoare Triples

theory *Safety*

imports *Guards*

begin

4.3.1 Preliminaries

definition *sat-inv* :: *store* \Rightarrow $(i, 'a)$ *heap* \Rightarrow $(i, 'a, \text{nat})$ *single-context* \Rightarrow *bool*
where

$\text{sat-inv } s \ hj \ \Gamma \longleftrightarrow (s, hj), (s, hj) \models \text{invariant } \Gamma \wedge \text{no-guard } hj$

lemma *sat-invI*:

assumes $(s, hj), (s, hj) \models \text{invariant } \Gamma$

and *no-guard hj*

shows $\text{sat-inv } s \ hj \ \Gamma$

by $(\text{simp add: assms(1) assms(2) sat-inv-def})$

s and *s'* can differ on variables outside of *vars*, does not change anything.
upper-fvs S vars means that *vars* is an upper-bound of "fv S"

definition *upper-fvs* :: $(\text{store} \times (i, 'a) \text{ heap}) \text{ set} \Rightarrow \text{var set} \Rightarrow \text{bool}$ **where**

$\text{upper-fvs } S \ \text{vars} \longleftrightarrow (\forall s \ s' \ h. (s, h) \in S \wedge \text{agrees vars } s \ s' \longrightarrow (s', h) \in S)$

Only need to agree on *vars*

definition *upperize* **where**

$upperize\ S\ vars = \{ \sigma' \mid \sigma\ \sigma'.\ \sigma \in S \wedge snd\ \sigma = snd\ \sigma' \wedge agrees\ vars\ (fst\ \sigma)\ (fst\ \sigma') \}$

definition *close-var* **where**

$close-var\ S\ x = \{ ((fst\ \sigma)(x := v),\ snd\ \sigma) \mid \sigma\ v.\ \sigma \in S \}$

lemma *upper-fvsI*:

assumes $\bigwedge s\ s'\ h.\ (s, h) \in S \wedge agrees\ vars\ s\ s' \implies (s', h) \in S$

shows *upper-fvs* $S\ vars$

using *assms upper-fvs-def* **by** *blast*

lemma *pair-sat-comm*:

assumes *pair-sat* $S\ S'\ A$

shows *pair-sat* $S'\ S\ A$

proof (*rule pair-satI*)

fix $s\ h\ s'\ h'$ **assume** $(s, h) \in S' \wedge (s', h') \in S$

then show $(s, h), (s', h') \models A$

using *assms pair-sat-def sat-comm* **by** *blast*

qed

lemma *in-upperize*:

$(s', h) \in upperize\ S\ vars \longleftrightarrow (\exists s.\ (s, h) \in S \wedge agrees\ vars\ s\ s') \text{ (is } ?A \longleftrightarrow ?B)$

proof

show $?A \implies ?B$

by (*simp add: upperize-def*)

show $?B \implies ?A$

using *upperize-def* **by** *fastforce*

qed

lemma *upper-fvs-upperize*:

upper-fvs (*upperize* $S\ vars$) $vars$

proof (*rule upper-fvsI*)

fix $s\ s'\ h$

assume $(s, h) \in upperize\ S\ vars \wedge agrees\ vars\ s\ s'$

then obtain s'' **where** $(s'', h) \in S \wedge agrees\ vars\ s''\ s$

by (*meson in-upperize*)

then have $agrees\ vars\ s''\ s'$

using $\langle (s, h) \in upperize\ S\ vars \wedge agrees\ vars\ s\ s' \rangle\ agrees-def[of\ vars\ s\ s']$

$agrees-def[of\ vars\ s''\ s]\ agrees-def[of\ vars\ s''\ s']$

by *simp*

then show $(s', h) \in upperize\ S\ vars$

using $\langle (s'', h) \in S \wedge agrees\ vars\ s''\ s \rangle\ upperize-def$ **by** *fastforce*

qed

lemma *upperize-larger*:

$S \subseteq upperize\ S\ vars$

proof

fix x **assume** $x \in S$

moreover have $agrees\ vars\ (fst\ x)\ (fst\ x)$

using *agrees-def* **by** *blast*
ultimately show $x \in \text{upperize } S \text{ vars}$
by (*metis* (*mono-tags*, *lifting*) *CollectI upperize-def*)
qed

lemma *pair-sat-upperize*:
assumes *pair-sat* $S S' A$
shows *pair-sat* (*upperize* S (*fvA* A)) $S' A$
proof (*rule pair-satI*)
fix $s h s' h'$
assume *asm0*: $(s, h) \in \text{upperize } S \text{ (fvA } A) \wedge (s', h') \in S'$
then obtain s'' **where** *agrees* (*fvA* A) $s s'' (s'', h) \in S$
using *agrees-def*[*of* *fvA* $A s s'$] *in-upperize*[*of* $s h S \text{ fvA } A$]
by (*metis agrees-def*)
then show $(s, h), (s', h') \models A$
using *agrees-same* *asm0* *assms pair-sat-def* **by** *blast*
qed

lemma *in-close-var*:
 $(s', h) \in \text{close-var } S x \longleftrightarrow (\exists s v. (s, h) \in S \wedge s' = s(x := v))$ (**is** $?A \longleftrightarrow ?B$)
proof
show $?A \implies ?B$
using *close-var-def*[*of* $S x$] *mem-Collect-eq prod.inject surjective-pairing*
by *auto*
show $?B \implies ?A$
using *close-var-def* **by** *fastforce*
qed

lemma *pair-sat-close-var*:
assumes $x \notin \text{fvA } A$
and *pair-sat* $S S' A$
shows *pair-sat* (*close-var* $S x$) $S' A$
proof (*rule pair-satI*)
fix $s h s' h'$
assume $(s, h) \in \text{close-var } S x \wedge (s', h') \in S'$
then show $(s, h), (s', h') \models A$
by (*metis* (*no-types*, *lifting*) *agrees-same agrees-update assms in-close-var pair-sat-def*)
qed

lemma *pair-sat-close-var-double*:
assumes *pair-sat* $S S' A$
and $x \notin \text{fvA } A$
shows *pair-sat* (*close-var* $S x$) (*close-var* $S' x$) A
using *assms pair-sat-close-var pair-sat-comm* **by** *blast*

lemma *close-var-subset*:
 $S \subseteq \text{close-var } S x$
proof
fix y **assume** $y \in S$

then have $\text{fst } y = (\text{fst } y)(x := (\text{fst } y \ x))$
by *simp*
then show $y \in \text{close-var } S \ x$
by (*metis* $\langle y \in S \rangle$ *in-close-var prod.exhaust-sel*)
qed

lemma *upper-fvs-close-vars*:

$\text{upper-fvs } (\text{close-var } S \ x) \ (- \ \{x\})$
proof (*rule upper-fvsI*)
fix $s \ s' \ h$
assume $(s, h) \in \text{close-var } S \ x \wedge \text{agrees } (- \ \{x\}) \ s \ s'$
have $s(x := s' \ x) = s'$
proof (*rule ext*)
fix y **show** $(s(x := s' \ x)) \ y = s' \ y$
by (*metis* (*mono-tags, lifting*) *ComplI* $\langle (s, h) \in \text{close-var } S \ x \wedge \text{agrees } (- \ \{x\}) \ s \ s' \rangle$ *agrees-def fun-upd-apply singleton-iff*)
qed
then show $(s', h) \in \text{close-var } S \ x$
by (*metis* $\langle (s, h) \in \text{close-var } S \ x \wedge \text{agrees } (- \ \{x\}) \ s \ s' \rangle$ *fun-upd-upd in-close-var*)
qed

lemma *sat-inv-agrees*:

assumes *sat-inv s hj* Γ
and *agrees (fvA (invariant Γ)) s s'*
shows *sat-inv s' hj* Γ
by (*meson agrees-same assms sat-comm sat-inv-def*)

lemma *abort-iff-fvC*:

assumes *agrees (fvC C) s s'*
shows *aborts C (s, h) \longleftrightarrow aborts C (s', h)*
using *aborts-agrees assms fst-conv snd-eqD*
by (*metis* (*mono-tags, lifting*) *agrees-def*)

lemma *view-function-of-invE*:

assumes *view-function-of-inv* Γ
and *sat-inv s h* Γ
and $(h' :: ('i, 'a) \text{ heap}) \succeq h$
shows $\text{view } \Gamma \ (\text{normalize } (\text{get-fh } h)) = \text{view } \Gamma \ (\text{normalize } (\text{get-fh } h'))$
using *assms(1) assms(2) assms(3) sat-inv-def view-function-of-inv-def* **by** *blast*

4.3.2 Safety

fun *no-abort* $:: ('i, 'a, \text{nat}) \text{ cont} \Rightarrow \text{cmd} \Rightarrow \text{store} \Rightarrow ('i, 'a) \text{ heap} \Rightarrow \text{bool}$ **where**
 $\text{no-abort } \text{None } C \ s \ h \longleftrightarrow (\forall \text{ hf } H. \text{Some } H = \text{Some } h \oplus \text{Some } \text{hf} \wedge \text{full-ownership}$
 $(\text{get-fh } H) \wedge \text{no-guard } H$
 $\longrightarrow \neg \text{aborts } C \ (s, \text{normalize } (\text{get-fh } H)))$
 $|\ \text{no-abort } (\text{Some } \Gamma) \ C \ s \ h \longleftrightarrow (\forall \text{ hf } H \ \text{hj } v0. \text{Some } H = \text{Some } h \oplus \text{Some } \text{hj} \oplus$
 $\text{Some } \text{hf} \wedge \text{full-ownership } (\text{get-fh } H) \wedge$
 $\text{semi-consistent } \Gamma \ v0 \ H \wedge \text{sat-inv } s \ \text{hj } \Gamma$

$\longrightarrow \neg \text{aborts } C (s, \text{normalize } (\text{get-fh } H))$

lemma *no-abortI*:

assumes $\bigwedge (hf :: ('i, 'a) \text{ heap}) (H :: ('i, 'a) \text{ heap}). \text{Some } H = \text{Some } h \oplus \text{Some } hf \wedge \Delta = \text{None} \wedge \text{full-ownership } (\text{get-fh } H) \wedge \text{no-guard } H \implies \neg \text{aborts } C (s, \text{normalize } (\text{get-fh } H))$

and $\bigwedge H hf hj v0 \Gamma. \Delta = \text{Some } \Gamma \wedge \text{Some } H = \text{Some } h \oplus \text{Some } hj \oplus \text{Some } hf \wedge \text{full-ownership } (\text{get-fh } H) \wedge \text{semi-consistent } \Gamma v0 H \wedge \text{sat-inv } s hj \Gamma \implies \neg \text{aborts } C (s, \text{normalize } (\text{get-fh } H))$

shows *no-abort* $\Delta C s (h :: ('i, 'a) \text{ heap})$

apply (*cases* Δ)

using *assms(1) no-abort.simps(1) apply blast*

using *assms(2) no-abort.simps(2) by blast*

lemma *no-abortSomeI*:

assumes $\bigwedge H hf hj v0. \text{Some } H = \text{Some } h \oplus \text{Some } hj \oplus \text{Some } hf \wedge \text{full-ownership } (\text{get-fh } H) \wedge \text{semi-consistent } \Gamma v0 H \wedge \text{sat-inv } s hj \Gamma$

$\implies \neg \text{aborts } C (s, \text{normalize } (\text{get-fh } H))$

shows *no-abort* $(\text{Some } \Gamma) C s (h :: ('i, 'a) \text{ heap})$

using *assms no-abort.simps(2) by blast*

lemma *no-abortNoneI*:

assumes $\bigwedge (hf :: ('i, 'a) \text{ heap}) (H :: ('i, 'a) \text{ heap}). \text{Some } H = \text{Some } h \oplus \text{Some } hf \wedge \text{full-ownership } (\text{get-fh } H) \wedge \text{no-guard } H \implies \neg \text{aborts } C (s, \text{normalize } (\text{get-fh } H))$

shows *no-abort* $(\text{None} :: ('i, 'a, \text{nat}) \text{ cont}) C s (h :: ('i, 'a) \text{ heap})$

using *assms no-abort.simps(1) by blast*

lemma *no-abortE*:

assumes *no-abort* $\Delta C s h$

shows $\text{Some } H = \text{Some } h \oplus \text{Some } hf \implies \Delta = \text{None} \implies \text{full-ownership } (\text{get-fh } H) \implies \text{no-guard } H \implies \neg \text{aborts } C (s, \text{normalize } (\text{get-fh } H))$

and $\Delta = \text{Some } \Gamma \implies \text{Some } H = \text{Some } h \oplus \text{Some } hj \oplus \text{Some } hf \implies \text{sat-inv } s hj \Gamma \implies \text{full-ownership } (\text{get-fh } H) \implies \text{semi-consistent } \Gamma v0 H$

$\implies \neg \text{aborts } C (s, \text{normalize } (\text{get-fh } H))$

using *assms no-abort.simps(1) apply blast*

by (*metis assms no-abort.simps(2)*)

fun *safe* $:: \text{nat} \Rightarrow ('i, 'a, \text{nat}) \text{ cont} \Rightarrow \text{cmd} \Rightarrow (\text{store} \times ('i, 'a) \text{ heap}) \Rightarrow (\text{store} \times ('i, 'a) \text{ heap}) \text{ set} \Rightarrow \text{bool}$ **where**

safe 0 - - - - $\longleftrightarrow \text{True}$

$| \text{safe } (\text{Suc } n) \text{None } C (s, h) S \longleftrightarrow (C = \text{Cskip} \longrightarrow (s, h) \in S) \wedge \text{no-abort } (\text{None} :: ('i, 'a, \text{nat}) \text{ cont}) C s h \wedge$

$(\forall H hf C' s' h'. \text{Some } H = \text{Some } h \oplus \text{Some } hf \wedge \text{full-ownership } (\text{get-fh } H) \wedge \text{no-guard } H$

$\wedge \text{red } C (s, \text{normalize } (\text{get-fh } H)) C' (s', h')$

$\longrightarrow (\exists h'' H'. \text{full-ownership } (\text{get-fh } H') \wedge \text{no-guard } H' \wedge h' = \text{normalize } (\text{get-fh } H') \wedge \text{Some } H' = \text{Some } h'' \oplus \text{Some } hf \wedge \text{safe } n (\text{None} :: ('i, 'a, \text{nat}) \text{ cont}) C' (s',$

$h''') S))$

| $\text{safe } (\text{Suc } n) (\text{Some } \Gamma) C (s, h) S \longleftrightarrow (C = \text{Cskip} \longrightarrow (s, h) \in S) \wedge \text{no-abort}$
 $(\text{Some } \Gamma) C s h \wedge$
 $(\forall H hf C' s' h' hj v0. \text{Some } H = \text{Some } h \oplus \text{Some } hj \oplus \text{Some } hf \wedge \text{full-ownership}$
 $(\text{get-fh } H) \wedge \text{semi-consistent } \Gamma v0 H \wedge \text{sat-inv } s hj \Gamma$
 $\wedge \text{red } C (s, \text{normalize } (\text{get-fh } H)) C' (s', h')$
 $\longrightarrow (\exists h'' H' hj'. \text{full-ownership } (\text{get-fh } H') \wedge \text{semi-consistent } \Gamma v0 H' \wedge \text{sat-inv}$
 $s' hj' \Gamma$
 $\wedge h' = \text{normalize } (\text{get-fh } H') \wedge \text{Some } H' = \text{Some } h'' \oplus \text{Some } hj' \oplus \text{Some } hf \wedge$
 $\text{safe } n (\text{Some } \Gamma) C' (s', h'') S))$

lemma *safeNoneI*:

assumes $C = \text{Cskip} \implies (s, h) \in S$
and $\text{no-abort } \text{None } C s h$
and $\bigwedge H hf C' s' h'. \text{Some } H = \text{Some } h \oplus \text{Some } hf \wedge \text{full-ownership } (\text{get-fh}$
 $H) \wedge \text{no-guard } H \wedge \text{red } C (s, \text{normalize } (\text{get-fh } H)) C' (s', h')$
 $\implies (\exists h'' H'. \text{full-ownership } (\text{get-fh } H') \wedge \text{no-guard } H' \wedge h' = \text{normalize } (\text{get-fh}$
 $H') \wedge \text{Some } H' = \text{Some } h'' \oplus \text{Some } hf \wedge \text{safe } n (\text{None} :: ('i, 'a, \text{nat}) \text{cont}) C' (s',$
 $h'') S)$
shows $\text{safe } (\text{Suc } n) (\text{None} :: ('i, 'a, \text{nat}) \text{cont}) C (s, h :: ('i, 'a) \text{heap}) S$
using *assms by auto*

lemma *safeSomeI*:

assumes $C = \text{Cskip} \implies (s, h) \in S$
and $\text{no-abort } (\text{Some } \Gamma) C s h$
and $\bigwedge H hf C' s' h' hj v0. \text{Some } H = \text{Some } h \oplus \text{Some } hj \oplus \text{Some } hf \wedge$
 $\text{full-ownership } (\text{get-fh } H)$
 $\wedge \text{semi-consistent } \Gamma v0 H \wedge \text{sat-inv } s hj \Gamma \wedge \text{red } C (s, \text{normalize } (\text{get-fh } H))$
 $C' (s', h')$
 $\implies (\exists h'' H' hj'. \text{full-ownership } (\text{get-fh } H') \wedge \text{semi-consistent } \Gamma v0 H' \wedge \text{sat-inv}$
 $s' hj' \Gamma$
 $\wedge h' = \text{normalize } (\text{get-fh } H') \wedge \text{Some } H' = \text{Some } h'' \oplus \text{Some } hj' \oplus \text{Some } hf \wedge$
 $\text{safe } n (\text{Some } \Gamma) C' (s', h'') S)$
shows $\text{safe } (\text{Suc } n) (\text{Some } \Gamma) C (s, h :: ('i, 'a) \text{heap}) S$
using *assms by auto*

lemma *safeI*:

fixes $\Delta :: ('i, 'a, \text{nat}) \text{cont}$
assumes $C = \text{Cskip} \implies (s, h) \in S$
and $\text{no-abort } \Delta C s h$
and $\bigwedge H hf C' s' h'. \Delta = \text{None} \implies \text{Some } H = \text{Some } h \oplus \text{Some } hf \wedge$
 $\text{full-ownership } (\text{get-fh } H) \wedge \text{no-guard } H \wedge \text{red } C (s, \text{normalize } (\text{get-fh } H)) C' (s',$
 $h')$
 $\implies (\exists h'' H'. \text{full-ownership } (\text{get-fh } H') \wedge \text{no-guard } H' \wedge h' = \text{normalize } (\text{get-fh}$
 $H') \wedge \text{Some } H' = \text{Some } h'' \oplus \text{Some } hf \wedge \text{safe } n (\text{None} :: ('i, 'a, \text{nat}) \text{cont}) C' (s',$
 $h'') S)$
and $\bigwedge H hf C' s' h' hj v0 \Gamma. \Delta = \text{Some } \Gamma \implies \text{Some } H = \text{Some } h \oplus \text{Some}$
 $hj \oplus \text{Some } hf \wedge \text{full-ownership } (\text{get-fh } H)$

\wedge *semi-consistent* Γ $v0$ $H \wedge$ *sat-inv* s hj $\Gamma \wedge$ *red* C (s , *normalize* (*get-fh* H))
 $C' (s', h')$
 $\implies (\exists h'' H' hj'. \text{full-ownership } (\text{get-fh } H') \wedge \text{semi-consistent } \Gamma v0 H' \wedge \text{sat-inv } s' hj' \Gamma$
 $\wedge h' = \text{normalize } (\text{get-fh } H') \wedge \text{Some } H' = \text{Some } h'' \oplus \text{Some } hj' \oplus \text{Some } hf \wedge$
 $\text{safe } n (\text{Some } \Gamma) C' (s', h'') S)$
shows *safe* (*Suc* n) Δ $C (s, h :: ('i, 'a) \text{heap}) S$
proof (*cases* Δ)
case *None*
then show *?thesis*
using *assms*(1) *assms*(2) *assms*(3) **by** *auto*
next
case (*Some* Γ)
then show *?thesis* **using** *safeSomeI* *assms*(1) *assms*(2) *assms*(4)
by *simp*
qed

lemma *safeSomeAltI*:

assumes $C = \text{Cskip} \implies (s, h) \in S$
and $\bigwedge H hf hj v0. \text{Some } H = \text{Some } h \oplus \text{Some } hj \oplus \text{Some } hf \wedge \text{full-ownership } (\text{get-fh } H) \wedge \text{semi-consistent } \Gamma v0 H \wedge \text{sat-inv } s hj \Gamma$
 $\implies \neg \text{aborts } C (s, \text{normalize } (\text{get-fh } H))$
and $\bigwedge H hf C' s' h' hj v0. \text{Some } H = \text{Some } h \oplus \text{Some } hj \oplus \text{Some } hf \wedge$
 $\text{full-ownership } (\text{get-fh } H)$
 $\wedge \text{semi-consistent } \Gamma v0 H \wedge \text{sat-inv } s hj \Gamma \implies \text{red } C (s, \text{normalize } (\text{get-fh } H)) C' (s', h')$
 $\implies (\exists h'' H' hj'. \text{full-ownership } (\text{get-fh } H') \wedge \text{semi-consistent } \Gamma v0 H' \wedge \text{sat-inv } s' hj' \Gamma$
 $\wedge h' = \text{normalize } (\text{get-fh } H') \wedge \text{Some } H' = \text{Some } h'' \oplus \text{Some } hj' \oplus \text{Some } hf \wedge$
 $\text{safe } n (\text{Some } \Gamma) C' (s', h'') S)$
shows *safe* (*Suc* n) (*Some* Γ) $C (s, h :: ('i, 'a) \text{heap}) S$
using *assms*(1)
proof (*rule* *safeSomeI*)
show *no-abort* (*Some* Γ) $C s h$ **using** *assms*(2) *no-abortSomeI* **by** *blast*
show $\bigwedge H hf C' s' h' hj v0.$
 $\text{Some } H = \text{Some } h \oplus \text{Some } hj \oplus \text{Some } hf \wedge \text{full-ownership } (\text{get-fh } H)$
 $\wedge \text{semi-consistent } \Gamma v0 H \wedge \text{sat-inv } s hj \Gamma \wedge \text{red } C (s, \text{FractionalHeap.normalize } (\text{get-fh } H)) C' (s', h') \implies$
 $(\exists h'' H' hj'.$
 $\text{full-ownership } (\text{get-fh } H') \wedge$
 $\text{semi-consistent } \Gamma v0 H' \wedge \text{sat-inv } s' hj' \Gamma \wedge h' = \text{FractionalHeap.normalize } (\text{get-fh } H') \wedge \text{Some } H' = \text{Some } h'' \oplus \text{Some } hj' \oplus \text{Some } hf \wedge \text{safe } n (\text{Some } \Gamma) C' (s', h'') S)$
using *assms*(3) **by** *blast*
qed (*simp*)

lemma *safeSomeE*:

assumes $\text{safe } (\text{Suc } n) (\text{Some } \Gamma) C (s, h :: ('i, 'a) \text{ heap}) S$
shows $C = \text{Cskip} \implies (s, h) \in S$
and $\text{no-abort } (\text{Some } \Gamma) C s h$
and $\text{Some } H = \text{Some } h \oplus \text{Some } hj \oplus \text{Some } hf \implies \text{full-ownership } (\text{get-fh } H)$
 $\implies \text{semi-consistent } \Gamma v0 H \implies \text{sat-inv } s hj \Gamma \implies \text{red } C (s, \text{normalize } (\text{get-fh } H)) C' (s', h')$
 $\implies (\exists h'' H' hj'. \text{full-ownership } (\text{get-fh } H') \wedge \text{semi-consistent } \Gamma v0 H' \wedge \text{sat-inv } s' hj' \Gamma$
 $\wedge h' = \text{normalize } (\text{get-fh } H') \wedge \text{Some } H' = \text{Some } h'' \oplus \text{Some } hj' \oplus \text{Some } hf \wedge$
 $\text{safe } n (\text{Some } \Gamma) C' (s', h'') S)$
using $\text{assms safe.simps}(3)[\text{of } n \Gamma C s h S]$ **by** blast+

lemma safeNoneE :

assumes $\text{safe } (\text{Suc } n) (\text{None} :: ('i, 'a, \text{nat}) \text{ cont}) C (s, h :: ('i, 'a) \text{ heap}) S$
shows $C = \text{Cskip} \implies (s, h) \in S$
and $\text{no-abort } (\text{None} :: ('i, 'a, \text{nat}) \text{ cont}) C s h$
and $\text{Some } H = \text{Some } h \oplus \text{Some } hf \implies \text{full-ownership } (\text{get-fh } H) \implies \text{no-guard } H$
 $\implies \text{red } C (s, \text{normalize } (\text{get-fh } H)) C' (s', h')$
 $\implies (\exists h'' H'. \text{full-ownership } (\text{get-fh } H') \wedge \text{no-guard } H' \wedge h' = \text{normalize } (\text{get-fh } H')$
 $\wedge \text{Some } H' = \text{Some } h'' \oplus \text{Some } hf \wedge \text{safe } n (\text{None} :: ('i, 'a, \text{nat}) \text{ cont}) C' (s', h'') S)$
using $\text{assms safe.simps}(2)[\text{of } n C s h S]$ **by** blast+

lemma safeNoneE-bis :

fixes $\text{no-cont} :: ('i, 'a, \text{nat}) \text{ cont}$
assumes $\text{safe } (\text{Suc } n) \text{no-cont } C (s, h :: ('i, 'a) \text{ heap}) S$
and $\text{no-cont} = \text{None}$
shows $C = \text{Cskip} \implies (s, h) \in S$
and $\text{no-abort } \text{no-cont } C s h$
and $\text{Some } H = \text{Some } h \oplus \text{Some } hf \implies \text{full-ownership } (\text{get-fh } H) \implies \text{no-guard } H$
 $\implies \text{red } C (s, \text{normalize } (\text{get-fh } H)) C' (s', h')$
 $\implies (\exists h'' H'. \text{full-ownership } (\text{get-fh } H') \wedge \text{no-guard } H' \wedge h' = \text{normalize } (\text{get-fh } H')$
 $\wedge \text{Some } H' = \text{Some } h'' \oplus \text{Some } hf \wedge \text{safe } n \text{no-cont } C' (s', h'') S)$
using $\text{assms safe.simps}(2)[\text{of } n C s h S]$ **by** blast+

4.3.3 Useful results about safety

lemma no-abort-larger :

assumes $h' \succeq h$
and $\text{no-abort } \Gamma C s h$
shows $\text{no-abort } \Gamma C s h'$

proof (rule no-abortI)

show $\bigwedge hf H. \text{Some } H = \text{Some } h' \oplus \text{Some } hf \wedge \Gamma = \text{None} \wedge \text{full-ownership } (\text{get-fh } H) \wedge \text{no-guard } H \implies \neg \text{aborts } C (s, \text{FractionalHeap.normalize } (\text{get-fh } H))$

using $\text{assms}(1) \text{assms}(2) \text{larger-def larger-trans no-abort.simps}(1)$ **by** blast

show $\bigwedge H hf hj v0 \Gamma'.$

$\Gamma = \text{Some } \Gamma' \wedge \text{Some } H = \text{Some } h' \oplus \text{Some } hj \oplus \text{Some } hf \wedge \text{full-ownership } (\text{get-fh } H) \wedge \text{semi-consistent } \Gamma' v0 H \wedge \text{sat-inv } s hj \Gamma' \implies$
 $\neg \text{aborts } C (s, \text{FractionalHeap.normalize } (\text{get-fh } H))$

proof –
fix H hf hj $v0$ Γ'
assume $asm0$: $\Gamma = \text{Some } \Gamma' \wedge \text{Some } H = \text{Some } h' \oplus \text{Some } hj \oplus \text{Some } hf \wedge$
full-ownership ($get\text{-}fh$ H) \wedge *semi-consistent* $\Gamma' v0 H \wedge sat\text{-}inv$ s hj Γ'
moreover obtain r **where** $\text{Some } h' = \text{Some } h \oplus \text{Some } r$
using $assms(1)$ *larger-def* **by** *blast*
then obtain hf' **where** $\text{Some } hf' = \text{Some } hf \oplus \text{Some } r$
by ($metis$ (*no-types*, *opaque-lifting*) *calculation not-None-eq plus.simps(1)*)
plus-asso plus-comm)
then have $\text{Some } H = \text{Some } h \oplus \text{Some } hj \oplus \text{Some } hf'$
by ($metis$ (*no-types*, *opaque-lifting*) $\langle \text{Some } h' = \text{Some } h \oplus \text{Some } r \rangle$ *calculation*
plus-asso plus-comm)
then show \neg *aborts* C (s , $FractionalHeap.normalize$ ($get\text{-}fh$ H))
using $assms(2)$ *calculation no-abortE(2)* **by** *blast*
qed
qed

lemma *safe-larger-set-aux*:
fixes $\Delta :: ('i, 'a, nat)$ *cont*
assumes *safe n* Δ C (s , h) S
and $S \subseteq S'$
shows *safe n* Δ C (s , h) S'
using $assms$
proof (*induct n arbitrary: s h C*)
case (*Suc n*)
show *?case*
proof (*rule safeI*)
show $C = Cskip \implies (s, h) \in S'$
by ($metis$ (*no-types*, *opaque-lifting*) $Suc.prem(1)$ $assms(2)$ *not-Some-eq*
safeNoneE-bis(1) safeSomeE(1) subset-iff)
show *no-abort* Δ C s h
apply (*cases* Δ)
using $Suc.prem(1)$ *safeNoneE-bis(2)* **apply** *blast*
using $Suc.prem(1)$ *safeSomeE(2)* **by** *blast*

show $\bigwedge H$ hf C' s' h' .
 $\Delta = None \implies$
 $\text{Some } H = \text{Some } h \oplus \text{Some } hf \wedge \text{full-ownership } (get\text{-}fh$ $H) \wedge \text{no-guard } H \wedge$
 red C (s , $FractionalHeap.normalize$ ($get\text{-}fh$ H)) C' (s' , h') \implies
 $\exists h'' H'. \text{full-ownership } (get\text{-}fh$ $H') \wedge \text{no-guard } H' \wedge h' = Fractional\text{-}$
 $Heap.normalize$ ($get\text{-}fh$ $H') \wedge \text{Some } H' = \text{Some } h'' \oplus \text{Some } hf \wedge \text{safe } n$ ($None$
 $:: ('i, 'a, nat)$ *cont*) C' (s' , h') S'
using $Suc.hyps$ $Suc.prem(1)$ $assms(2)$ $safeNoneE(3)$ [*of n C s h*] **by** *blast*

show $\bigwedge H$ hf C' s' h' hj $v0$ Γ .
 $\Delta = \text{Some } \Gamma \implies$
 $\text{Some } H = \text{Some } h \oplus \text{Some } hj \oplus \text{Some } hf \wedge$
 $\text{full-ownership } (get\text{-}fh$ $H) \wedge \text{semi-consistent } \Gamma v0 H \wedge sat\text{-}inv$ s hj $\Gamma \wedge red$ C
 $(s, FractionalHeap.normalize$ ($get\text{-}fh$ H)) C' (s' , h') \implies

$\exists h'' H' hj'$.
full-ownership (*get-fh* H') \wedge
semi-consistent $\Gamma v0 H' \wedge \text{sat-inv } s' hj' \Gamma \wedge h' = \text{FractionalHeap.normalize}$
(*get-fh* H') $\wedge \text{Some } H' = \text{Some } h'' \oplus \text{Some } hj' \oplus \text{Some } hf \wedge \text{safe } n (\text{Some } \Gamma) C'$
(s', h'') S'
proof –
fix $H hf C' s' h' hj v0 \Gamma$
assume *asm0*: $\Delta = \text{Some } \Gamma \text{Some } H = \text{Some } h \oplus \text{Some } hj \oplus \text{Some } hf \wedge$
full-ownership (*get-fh* H) $\wedge \text{semi-consistent } \Gamma v0 H \wedge \text{sat-inv } s hj \Gamma \wedge \text{red } C$
($s, \text{FractionalHeap.normalize } (\text{get-fh } H)) C' (s', h')$
then show $\exists h'' H' hj'. \text{full-ownership } (\text{get-fh } H') \wedge \text{semi-consistent } \Gamma v0 H'$
 $\wedge \text{sat-inv } s' hj' \Gamma$
 $\wedge h' = \text{FractionalHeap.normalize } (\text{get-fh } H') \wedge \text{Some } H' = \text{Some } h'' \oplus \text{Some}$
 $hj' \oplus \text{Some } hf \wedge \text{safe } n (\text{Some } \Gamma) C' (s', h'') S'$
using *safeSomeE*(β)[*of n* $\Gamma C s h S$] *Suc.hyps* *Suc.prem*(1) *assms*(2) **by**
blast
qed
qed
qed (*simp*)

lemma *safe-larger-set*:
assumes *safe n* $\Delta C \sigma S$
and $S \subseteq S'$
shows *safe n* $\Delta C \sigma S'$
using *assms safe-larger-set-aux*[*of n* $\Delta C \text{fst } \sigma \text{snd } \sigma S S'$]
by *auto*

lemma *safe-smaller-aux*:
fixes $\Delta :: ('i, 'a, \text{nat}) \text{cont}$
assumes $m \leq n$
and *safe n* $\Delta C (s, h) S$
shows *safe m* $\Delta C (s, h) S$
using *assms*
proof (*induct n arbitrary: s h C m*)
case (*Suc n*)
show ?*case*
proof (*cases m*)
case (*Suc k*)
then have $k \leq n$
using *Suc.prem*(1) **by** *fastforce*
moreover have *safe* (*Suc k*) $\Delta C (s, h) S$
proof (*rule safeI*)
show $C = C\text{skip} \implies (s, h) \in S$
using *Suc.prem*(2) *safe.elims*(2) **by** *blast*
show *no-abort* $\Delta C s h$
apply (*cases* Δ)
using *Suc.prem*(2) *safeNoneE*(2) **apply** *blast*
using *Suc.prem*(2) *safeSomeE*(2) **by** *blast*
show $\wedge H hf C' s' h'$.

$\Delta = \text{None} \implies$
Some $H = \text{Some } h \oplus \text{Some } hf \wedge \text{full-ownership } (\text{get-fh } H) \wedge \text{no-guard } H \wedge$
red $C (s, \text{FractionalHeap.normalize } (\text{get-fh } H)) C' (s', h') \implies$
 $\exists h'' H'. \text{full-ownership } (\text{get-fh } H') \wedge \text{no-guard } H' \wedge h' = \text{Fractional-Heap.normalize } (\text{get-fh } H') \wedge \text{Some } H' = \text{Some } h'' \oplus \text{Some } hf \wedge \text{safe } k (\text{None} :: ('i, 'a, \text{nat}) \text{cont}) C' (s', h') S$
proof –
fix $H hf C' s' h'$
assume $\text{asm0}: \Delta = \text{None } \text{Some } H = \text{Some } h \oplus \text{Some } hf \wedge \text{full-ownership } (\text{get-fh } H) \wedge \text{no-guard } H \wedge \text{red } C (s, \text{FractionalHeap.normalize } (\text{get-fh } H)) C' (s', h')$
then obtain $h'' H'$ **where** $\text{full-ownership } (\text{get-fh } H') \wedge \text{no-guard } H' \wedge h' = \text{FractionalHeap.normalize } (\text{get-fh } H') \wedge \text{Some } H' = \text{Some } h'' \oplus \text{Some } hf \wedge \text{safe } k (\text{None} :: ('i, 'a, \text{nat}) \text{cont}) C' (s', h'') S$
using *Suc.prem*s(2) *safeNoneE*(3) **by** *blast*
then show $\exists h'' H'. \text{full-ownership } (\text{get-fh } H') \wedge \text{no-guard } H' \wedge h' = \text{FractionalHeap.normalize } (\text{get-fh } H') \wedge \text{Some } H' = \text{Some } h'' \oplus \text{Some } hf \wedge \text{safe } k (\text{None} :: ('i, 'a, \text{nat}) \text{cont}) C' (s', h'') S$
using *Suc.hyps* *asm0*(1) *calculation* **by** *blast*
qed
fix $H hf C' s' h' hj v0 \Gamma$
assume $\text{asm0}: \Delta = \text{Some } \Gamma \text{Some } H = \text{Some } h \oplus \text{Some } hj \oplus \text{Some } hf \wedge \text{full-ownership } (\text{get-fh } H) \wedge \text{semi-consistent } \Gamma v0 H \wedge \text{sat-inv } s hj \Gamma \wedge \text{red } C (s, \text{FractionalHeap.normalize } (\text{get-fh } H)) C' (s', h')$
then show $\exists h'' H' hj'.$
 $\text{full-ownership } (\text{get-fh } H') \wedge$
 $\text{semi-consistent } \Gamma v0 H' \wedge \text{sat-inv } s' hj' \Gamma \wedge h' = \text{FractionalHeap.normalize } (\text{get-fh } H') \wedge \text{Some } H' = \text{Some } h'' \oplus \text{Some } hj' \oplus \text{Some } hf \wedge \text{safe } k (\text{Some } \Gamma) C' (s', h'') S$
using *Suc.prem*s(2) *safeSomeE*(3)[*of* $n \Gamma C s h S H hj hf v0 C' s' h'$]
Suc.hyps
using *calculation* **by** *blast*
qed
ultimately show *?thesis*
using *Suc* **by** *auto*
qed (*simp*)
qed (*simp*)

lemma *safe-smaller*:

assumes $m \leq n$
and *safe* $n \Delta C \sigma S$
shows *safe* $m \Delta C \sigma S$
by (*metis* *assms*(1) *assms*(2) *safe-smaller-aux surj-pair*)

lemma *safe-free-vars-aux*:

fixes $\Delta :: ('i, 'a, \text{nat}) \text{cont}$
assumes *safe* $n \Delta C (s0, h) S$
and *agrees* (*fvC* $C \cup \text{vars}$) $s0 s1$
and *upper-fvs* $S \text{vars}$

and $\bigwedge \Gamma. \Delta = \text{Some } \Gamma \implies \text{agrees } (fvA \text{ (invariant } \Gamma)) \ s0 \ s1$
shows $\text{safe } n \ \Delta \ C \ (s1, h) \ S$
using *assms*
proof (*induct n arbitrary: s0 h s1 C*)
case (*Suc n*)
show *?case*
proof (*rule safeI*)
show $C = Cskip \implies (s1, h) \in S$
by (*metis Suc.prem(1) Suc.prem(2) agrees-union assms(3) not-Some-eq safeNoneE-bis(1) safeSomeE(1) upper-fvs-def*)
show $\text{no-abort } \Delta \ C \ s1 \ h$
proof (*rule no-abortI*)
show $\bigwedge hf \ H. \text{Some } H = \text{Some } h \oplus \text{Some } hf \wedge \Delta = \text{None} \wedge \text{full-ownership } (get\text{-fh } H) \wedge \text{no-guard } H \implies \neg \text{aborts } C \ (s1, \text{FractionalHeap.normalize } (get\text{-fh } H))$
using *Suc.prem(1) Suc.prem(2) abort-iff-fvC agrees-union no-abortE(1) safeNoneE(2) by blast*
show $\bigwedge H \ hf \ hj \ v0 \ \Gamma. \Delta = \text{Some } \Gamma \wedge \text{Some } H = \text{Some } h \oplus \text{Some } hj \oplus \text{Some } hf \wedge \text{full-ownership } (get\text{-fh } H) \wedge \text{semi-consistent } \Gamma \ v0 \ H \wedge \text{sat-inv } s1 \ hj \ \Gamma \implies \neg \text{aborts } C \ (s1, \text{FractionalHeap.normalize } (get\text{-fh } H))$
proof –
fix $H \ hf \ hj \ v0 \ \Gamma$
assume $asm0: \Delta = \text{Some } \Gamma \wedge \text{Some } H = \text{Some } h \oplus \text{Some } hj \oplus \text{Some } hf \wedge \text{full-ownership } (get\text{-fh } H) \wedge \text{semi-consistent } \Gamma \ v0 \ H \wedge \text{sat-inv } s1 \ hj \ \Gamma$
then have $\text{sat-inv } s0 \ hj \ \Gamma$
using *Suc.prem(4) agrees-def sat-inv-agrees*
by (*metis (mono-tags, opaque-lifting)*)
then have $\neg \text{aborts } C \ (s0, \text{FractionalHeap.normalize } (get\text{-fh } H))$
using *Suc.prem(1) asm0 no-abort.simps(2) safeSomeE(2) by blast*
then show $\neg \text{aborts } C \ (s1, \text{FractionalHeap.normalize } (get\text{-fh } H))$
using *Suc.prem(2) abort-iff-fvC agrees-union by blast*
qed
qed
show $\bigwedge H \ hf \ C' \ s1' \ h'.$
 $\Delta = \text{None} \implies$
 $\text{Some } H = \text{Some } h \oplus \text{Some } hf \wedge \text{full-ownership } (get\text{-fh } H) \wedge \text{no-guard } H \wedge \text{red } C \ (s1, \text{FractionalHeap.normalize } (get\text{-fh } H)) \ C' \ (s1', h') \implies$
 $\exists h'' \ H'. \text{full-ownership } (get\text{-fh } H') \wedge \text{no-guard } H' \wedge h' = \text{FractionalHeap.normalize } (get\text{-fh } H') \wedge \text{Some } H' = \text{Some } h'' \oplus \text{Some } hf \wedge \text{safe } n \ (\text{None} :: ('i, 'a, nat) \text{cont}) \ C' \ (s1', h'') \ S$
proof –
fix $H \ hf \ C' \ s1' \ h'$
assume $asm0: \Delta = \text{None}$
 $\text{Some } H = \text{Some } h \oplus \text{Some } hf \wedge \text{full-ownership } (get\text{-fh } H) \wedge \text{no-guard } H \wedge \text{red } C \ (s1, \text{FractionalHeap.normalize } (get\text{-fh } H)) \ C' \ (s1', h')$
then obtain $s0'$ **where** $\text{red } C \ (s0, \text{FractionalHeap.normalize } (get\text{-fh } H)) \ C' \ (s0', h') \text{ agrees } (fvC \ C \cup \text{vars}) \ s1' \ s0'$
using $\text{red-agrees[of } C \ (s1, \text{FractionalHeap.normalize } (get\text{-fh } H)) \ C' \ (s1', h') \ fvC \ C \cup \text{vars}]$
using *Suc.prem(2) agrees-def fst-conv snd-conv sup-ge1*

by (*metis* (*mono-tags*, *lifting*))
then obtain $h'' H'$ **where**
 r : *full-ownership* (*get-fh* H') \wedge *no-guard* $H' \wedge h' = \text{FractionalHeap.normalize}$
(*get-fh* H') \wedge *Some* $H' = \text{Some } h'' \oplus \text{Some } hf \wedge \text{safe } n$ (*None* :: ($'i$, $'a$, *nat*) *cont*)
 $C' (s0', h'') S$
using *Suc.prem*s(1) *asm0*(1) *asm0*(2) *safeNoneE*(3) **by** *blast*
then have *safe* n (*None* :: ($'i$, $'a$, *nat*) *cont*) $C' (s1', h'') S$
using *Suc.hyps*[of $C' s0' h'' s1'$]
using $\langle \text{agrees } (fvC C \cup vars) s1' s0' \rangle$ *agrees-union* *asm0*(1) *asm0*(2)
assms(3) *option.distinct*(1) *red-properties*(1)
by (*metis* (*mono-tags*, *lifting*) *agrees-def subset-iff*)
then show $\exists h'' H'. \text{full-ownership } (get-fh H') \wedge \text{no-guard } H' \wedge h' = \text{FractionalHeap.normalize}$
(*get-fh* H') \wedge *Some* $H' = \text{Some } h'' \oplus \text{Some } hf \wedge \text{safe } n$ (*None* ::
($'i$, $'a$, *nat*) *cont*) $C' (s1', h'') S$
using r **by** *blast*
qed
fix $H hf C' s1' h' hj v0 \Gamma$
assume *asm0*: $\Delta = \text{Some } \Gamma$
 $\text{Some } H = \text{Some } h \oplus \text{Some } hj \oplus \text{Some } hf \wedge \text{full-ownership } (get-fh H) \wedge$
semi-consistent $\Gamma v0 H \wedge \text{sat-inv } s1 hj \Gamma \wedge \text{red } C (s1, \text{normalize } (get-fh H)) C'$
($s1', h'$)
then obtain $s0'$ **where** *red* $C (s0, \text{FractionalHeap.normalize } (get-fh H)) C'$
($s0', h'$) *agrees* ($fvC C \cup vars \cup fvA (\text{invariant } \Gamma)$) $s1' s0'$
using *red-agrees*[of $C (s1, \text{FractionalHeap.normalize } (get-fh H)) C' (s1', h')$]
 $fvC C \cup vars \cup fvA (\text{invariant } \Gamma)$]
using *Suc.prem*s(2) *Suc.prem*s(4) *agrees-comm* *agrees-union* *fst-conv* *snd-conv*
sup-assoc *sup-ge1*
by (*metis* (*no-types*, *lifting*))
moreover have *sat-inv* $s0 hj \Gamma$
using *Suc.prem*s(4) *agrees-comm* *asm0*(1) *asm0*(2) *sat-inv-agrees* **by** *blast*
ultimately obtain $h'' H' hj'$ **where** r : *full-ownership* (*get-fh* H') \wedge *semi-consistent*
 $\Gamma v0 H' \wedge \text{sat-inv } s0' hj' \Gamma$
 $\wedge h' = \text{FractionalHeap.normalize } (get-fh H') \wedge \text{Some } H' = \text{Some } h'' \oplus \text{Some } hj'$
 $\oplus \text{Some } hf \wedge \text{safe } n$ (*Some* Γ) $C' (s0', h'') S$
using *Suc.prem*s(1) *asm0*(1) *asm0*(2) *safeSomeE*(3)[of $n \Gamma C s0 h S H hj$
 hf]
by *blast*
then have *sat-inv* $s1' hj' \Gamma$
using $\langle \text{agrees } (fvC C \cup vars \cup fvA (\text{invariant } \Gamma)) s1' s0' \rangle$ *agrees-comm*
agrees-union *sat-inv-agrees* **by** *blast*
moreover have *safe* n (*Some* Γ) $C' (s1', h'') S$
using *Suc.hyps*[of $C' s0' h'' s1'$] $\langle \text{agrees } (fvC C \cup vars \cup fvA (\text{invariant } \Gamma))$
 $s1' s0' \rangle$ $\langle \text{red } C (s0, \text{FractionalHeap.normalize } (get-fh H)) C' (s0', h') \rangle$
agrees-def *agrees-union* *asm0*(1) *assms*(3) *option.inject* r *red-properties*
by (*metis* (*mono-tags*, *lifting*) *subset-Un-eq*)
ultimately show $\exists h'' H' hj'$.
full-ownership (*get-fh* H') \wedge
semi-consistent $\Gamma v0 H' \wedge$
sat-inv $s1' hj' \Gamma \wedge h' = \text{FractionalHeap.normalize } (get-fh H') \wedge \text{Some } H'$

= *Some* $h'' \oplus$ *Some* $hj' \oplus$ *Some* $hf \wedge$ *safe* n (*Some* Γ) $C' (s1', h')$ S
 using r by *blast*
qed
qed (*simp*)

lemma *safe-free-vars-None*:

assumes *safe* n (*None* :: ($'i, 'a, nat$) *cont*) $C (s, h) S$
and *agrees* ($fvC C \cup vars$) $s s'$
and *upper-fvs* $S vars$
shows *safe* n (*None* :: ($'i, 'a, nat$) *cont*) $C (s', h) S$
by (*meson* *assms*(1) *assms*(2) *assms*(3) *not-Some-eq* *safe-free-vars-aux*)

lemma *safe-free-vars-Some*:

assumes *safe* n (*Some* Γ) $C (s, h) S$
and *agrees* ($fvC C \cup vars \cup fvA$ (*invariant* Γ)) $s s'$
and *upper-fvs* $S vars$
shows *safe* n (*Some* Γ) $C (s', h) S$
by (*metis* *agrees-union* *assms*(1) *assms*(2) *assms*(3) *option.inject* *safe-free-vars-aux*)

lemma *safe-free-vars*:

fixes $\Delta :: ('i, 'a, nat)$ *cont*
assumes *safe* n $\Delta C (s, h) S$
and *agrees* ($fvC C \cup vars$) $s s'$
and *upper-fvs* $S vars$
and $\bigwedge \Gamma. \Delta = \text{Some } \Gamma \implies$ *agrees* (fvA (*invariant* Γ)) $s s'$
shows *safe* n $\Delta C (s', h) S$

proof (*cases* Δ)

case *None*

then show *?thesis*

using *assms*(1) *assms*(2) *assms*(3) *safe-free-vars-None* **by** *blast*

next

case (*Some* Γ)

then show *?thesis*

using *agrees-union* *assms*(1) *assms*(2) *assms*(3) *assms*(4) *safe-free-vars-Some*

by *blast*

qed

4.3.4 Hoare triples

definition *hoare-triple-valid* :: ($'i, 'a, nat$) *cont* \implies ($'i, 'a, nat$) *assertion* \implies *cmd*
 \implies ($'i, 'a, nat$) *assertion* \implies *bool*

($- \models \{-\} - \{-\}$ [*51,0,0*] *81*) **where**

hoare-triple-valid $\Gamma P C Q \longleftrightarrow$ ($\exists \Sigma. (\forall \sigma n. \sigma, \sigma \models P \longrightarrow$ *safe* $n \Gamma C \sigma (\Sigma \sigma))$

\wedge

($\forall \sigma \sigma'. \sigma, \sigma' \models P \longrightarrow$ *pair-sat* ($\Sigma \sigma$) ($\Sigma \sigma'$) Q))

lemma *hoare-triple-validI*:

assumes $\bigwedge s h n. (s, h), (s, h) \models P \implies$ *safe* $n \Gamma C (s, h) (\Sigma (s, h))$

and $\bigwedge s\ h\ s'\ h'. (s, h), (s', h') \models P \implies \text{pair-sat } (\Sigma (s, h)) (\Sigma (s', h')) Q$
shows *hoare-triple-valid* $\Gamma P C Q$
by (*metis* *assms(1)* *assms(2)* *hoare-triple-valid-def* *prod.collapse*)

lemma *hoare-triple-valid-smallerI*:
assumes $\bigwedge \sigma\ n. \sigma, \sigma \models P \implies \text{safe } n\ \Gamma\ C\ \sigma\ (\Sigma\ \sigma)$
and $\bigwedge \sigma\ \sigma'. \sigma, \sigma' \models P \implies \text{pair-sat } (\Sigma\ \sigma)\ (\Sigma\ \sigma')\ Q$
shows *hoare-triple-valid* $\Gamma P C Q$
using *assms* *hoare-triple-valid-def* **by** *metis*

lemma *hoare-triple-validE*:
assumes *hoare-triple-valid* $\Gamma P C Q$
shows $\exists \Sigma. (\forall \sigma\ n. \sigma, \sigma \models P \longrightarrow \text{safe } n\ \Gamma\ C\ \sigma\ (\Sigma\ \sigma)) \wedge$
 $(\forall \sigma\ \sigma'. \sigma, \sigma' \models P \longrightarrow \text{pair-sat } (\Sigma\ \sigma)\ (\Sigma\ \sigma')\ Q)$
using *assms* *hoare-triple-valid-def* **by** *blast*

lemma *hoare-triple-valid-simplerE*:
assumes *hoare-triple-valid* $\Gamma P C Q$
and $\sigma, \sigma' \models P$
shows $\exists S\ S'. \text{safe } n\ \Gamma\ C\ \sigma\ S \wedge \text{safe } n\ \Gamma\ C\ \sigma'\ S' \wedge \text{pair-sat } S\ S'\ Q$
by (*meson* *always-sat-refl* *assms(1)* *assms(2)* *hoare-triple-validE* *sat-comm*)

end

4.4 Soundness of the Rules

theory *Soundness*
imports *Safety* *AbstractCommutativity*
begin

4.4.1 Skip

lemma *safe-skip*:
fixes $\Delta :: ('i, 'a, \text{nat})\ \text{cont}$
assumes $(s, h) \in S$
shows *safe* $n\ \Delta\ C\ \text{skip } (s, h)\ S$
using *assms*
proof (*induct* n)
case (*Suc* n)
then show *?case*
proof (*cases* Δ)
case *None*
then show *?thesis*
by (*simp* *add: Suc.prem*)
next
case (*Some* a)
then show *?thesis*
by (*simp* *add: assms*)
qed

qed (*simp*)

theorem *rule-skip*:

hoare-triple-valid $\Gamma P Cskip P$

proof (*rule hoare-triple-validI*)

let $? \Sigma = \lambda \sigma. \{ \sigma \}$

show $\bigwedge s h n. (s, h), (s, h) \models P \implies safe\ n\ \Gamma\ Cskip\ (s, h)\ (? \Sigma\ (s, h))$

by (*simp add: safe-skip*)

show $\bigwedge s h s' h'. (s, h), (s', h') \models P \implies pair\text{-sat}\ \{(s, h)\}\ \{(s', h')\}\ P$

by (*metis pair-sat-smallerI singleton-iff*)

qed

4.4.2 Assign

inductive-cases *red-assign-cases*: *red* (*Cassign* $x E$) $\sigma C' \sigma'$

inductive-cases *aborts-assign-cases*: *aborts* (*Cassign* $x E$) σ

lemma *safe-assign*:

fixes $\Delta :: ('i, 'a, nat)\ cont$

assumes $\bigwedge \Gamma. \Delta = Some\ \Gamma \implies x \notin fvA\ (invariant\ \Gamma)$

shows *safe* $m\ \Delta\ (Cassign\ x\ E)\ (s, h)\ \{(s(x := edenot\ E\ s), h)\}$

proof (*induct* m)

case (*Suc* n)

show *safe* (*Suc* n) $\Delta\ (Cassign\ x\ E)\ (s, h)\ \{(s(x := edenot\ E\ s), h)\}$

proof (*rule safeI*)

show *no-abort* $\Delta\ (Cassign\ x\ E)\ s\ h$

using *aborts-assign-cases no-abortI* **by** *blast*

show $\bigwedge H\ hf\ C'\ s'\ h'.$

$\Delta = None \implies$

Some $H = Some\ h \oplus Some\ hf \wedge full\text{-ownership}\ (get\text{-fh}\ H) \wedge no\text{-guard}\ H \wedge$
red (*Cassign* $x E$) $(s, FractionalHeap.normalize\ (get\text{-fh}\ H))\ C'\ (s', h') \implies$
 $\exists h''\ H'.$

full-ownership (*get-fh* H') \wedge

no-guard $H' \wedge h' = FractionalHeap.normalize\ (get\text{-fh}\ H') \wedge Some\ H' =$
Some $h'' \oplus Some\ hf \wedge safe\ n\ None\ C'\ (s', h'')\ \{(s(x := edenot\ E\ s), h)\}$

by (*metis Pair-inject insertI1 red-assign-cases safe-skip*)

fix $H\ hf\ C'\ s'\ h'\ hj\ v0\ \Gamma$

assume *asm0*: $\Delta = Some\ \Gamma\ Some\ H = Some\ h \oplus Some\ hj \oplus Some\ hf \wedge$

full-ownership (*get-fh* H) $\wedge semi\text{-consistent}\ \Gamma\ v0\ H \wedge sat\text{-inv}\ s\ hj\ \Gamma \wedge red$
(*Cassign* $x E$) $(s, FractionalHeap.normalize\ (get\text{-fh}\ H))\ C'\ (s', h')$

then have *sat-inv* $(s(x := edenot\ E\ s))\ hj\ \Gamma$

by (*meson agrees-update assms sat-inv-agrees*)

then show $\exists h''\ H'\ hj'. full\text{-ownership}\ (get\text{-fh}\ H') \wedge semi\text{-consistent}\ \Gamma\ v0\ H'$
 $\wedge sat\text{-inv}\ s'\ hj'\ \Gamma \wedge$

$h' = FractionalHeap.normalize\ (get\text{-fh}\ H') \wedge Some\ H' = Some\ h'' \oplus Some$
 $hj' \oplus Some\ hf \wedge safe\ n\ (Some\ \Gamma)\ C'\ (s', h'')\ \{(s(x := edenot\ E\ s), h)\}$

by (*metis* (*no-types*, *lifting*) *asm0(2)* *insertI1* *old.prod.inject* *red-assign-cases* *safe-skip*)
qed (*simp*)
qed (*simp*)

theorem *assign-rule*:

fixes $\Delta :: ('i, 'a, nat) cont$
assumes $\bigwedge \Gamma. \Delta = Some \Gamma \implies x \notin fvA$ (*invariant* Γ)
and *collect-existentials* $P \cap fvE E = \{\}$
shows *hoare-triple-valid* Δ (*subA* $x E P$) (*Cassign* $x E$) P
proof –
define $\Sigma :: store \times ('i, 'a) heap \Rightarrow (store \times ('i, 'a) heap) set$ **where** $\Sigma = (\lambda \sigma. \{ ((fst \sigma)(x := edenot E (fst \sigma)), snd \sigma) \})$

show *?thesis*
proof (*rule* *hoare-triple-validI*)
show $\bigwedge s h n. (s, h), (s, h) \models subA x E P \implies safe n \Delta (Cassign x E) (s, h)$
 $(\Sigma (s, h))$
using *assms* *safe-assign* **by** (*metis* Σ -*def* *fst-eqD* *snd-eqD*)
show $\bigwedge s h s' h'. (s, h), (s', h') \models subA x E P \implies pair\text{-}sat (\Sigma (s, h)) (\Sigma (s', h')) P$
by (*metis* *assms(2)* Σ -*def* *fst-conv* *pair-sat-smallerI* *singleton-iff* *snd-conv* *subA-assign*)
qed
qed

4.4.3 Alloc

inductive-cases *red-alloc-cases*: *red* (*Calloc* $x E$) $\sigma C' \sigma'$

inductive-cases *aborts-alloc-cases*: *aborts* (*Calloc* $x E$) σ

lemma *safe-new-None*:

safe n (*None* :: ('i, 'a, nat) cont) (*Calloc* $x E$) ($s, (Map.empty, gs, gu)$) $\{ (s(x := a), (Map.empty(a \mapsto (pwrite, edenot E s)), gs, gu)) \mid a. True \}$
proof (*induct* n)
case (*Suc* n)
show *?case*
proof (*rule* *safeNoneI*)
show *Calloc* $x E = Cskip \implies (s, Map.empty, gs, gu) \in \{(s(x := a), [a \mapsto (pwrite, edenot E s)], gs, gu) \mid a. True\}$ **by** *simp*
show *no-abort* *None* (*Calloc* $x E$) $s (Map.empty, gs, gu)$
using *aborts-alloc-cases* *no-abort.simps(1)* **by** *blast*
fix $H hf C' s' h'$
assume *asm0*: *Some* $H = Some (Map.empty, gs, gu) \oplus Some hf \wedge$
full-ownership (*get-fh* H) \wedge *no-guard* $H \wedge red (Calloc x E) (s, Fractional\text{-}Heap.normalize (get-fh H)) C' (s', h')$

```

show  $\exists h'' H'$ .
  full-ownership (get-fh H')  $\wedge$ 
  no-guard H'  $\wedge$ 
  h' = FractionalHeap.normalize (get-fh H')  $\wedge$ 
  Some H' = Some h''  $\oplus$  Some hf  $\wedge$  safe n (None :: ('i, 'a, nat) cont) C'
(s', h') {(s(x := a), [a  $\mapsto$  (pwrite, edenot E s)], gs, gu) | a. True}
proof (rule red-alloc-cases)
  show red (Calloc x E) (s, FractionalHeap.normalize (get-fh H)) C' (s', h')
  using asm0 by blast
  fix sa h v
  assume asm1: (s, FractionalHeap.normalize (get-fh H)) = (sa, h) C' = Cskip
(s', h') = (sa(x := v), h(v  $\mapsto$  edenot E sa))
  v  $\notin$  dom h
  then have v  $\notin$  dom (get-fh H)
  by (simp add: dom-normalize)
  then have v  $\notin$  dom (get-fh hf)
  by (metis asm0 fst-conv get-fh.simps no-guard-and-no-heap no-guard-then-smaller-same
no-guards-remove plus-comm)

  moreover have (Map.empty(v  $\mapsto$  (pwrite, edenot E sa)), gs, gu)  $\#\#$  hf
  proof (rule compatibleI)
  show compatible-fract-heaps (get-fh ([v  $\mapsto$  (pwrite, edenot E sa)], gs, gu))
(get-fh hf)
  proof (rule compatible-fract-heapsI)
  fix l p p'
  assume asm0: get-fh ([v  $\mapsto$  (pwrite, edenot E sa)], gs, gu) l = Some p  $\wedge$ 
get-fh hf l = Some p'
  then show pgte pwrite (padd (fst p) (fst p'))
  by (metis calculation domIff fst-conv fun-upd-other get-fh.elims option.
distinct(1))
  show snd p = snd p'
  by (metis asm0 calculation domIff fst-conv fun-upd-other get-fh.elims
option.distinct(1))
  qed
  show  $\bigwedge k$ . get-gu ([v  $\mapsto$  (pwrite, edenot E sa)], gs, gu) k = None  $\vee$  get-gu
hf k = None
  by (metis asm0 compatible-def compatible-eq get-gu.simps option.discI
snd-conv)
  show  $\bigwedge p p'$ . get-gs ([v  $\mapsto$  (pwrite, edenot E sa)], gs, gu) = Some p  $\wedge$  get-gs
hf = Some p'  $\implies$  pgte pwrite (padd (fst p) (fst p'))
  by (metis asm0 no-guard-def no-guard-then-smaller-same option.simps(3)
plus-comm)
  qed
  then obtain H' where Some H' = Some (Map.empty(v  $\mapsto$  (pwrite, edenot
E sa)), gs, gu)  $\oplus$  Some hf
  by auto
  moreover have (s', (Map.empty(v  $\mapsto$  (pwrite, edenot E sa)), gs, gu))  $\in$  {(s(x
:= a), [a  $\mapsto$  (pwrite, edenot E s)], gs, gu) | a. True}
  using asm1(1) asm1(3) by blast

```

```

then have safe n (None :: ('i, 'a, nat) cont) C' (s', (Map.empty(v ↦ (pwrite,
edenot E sa)), gs, gu)) {(s(x := a), [a ↦ (pwrite, edenot E s)], gs, gu) | a. True}
  by (simp add: asm1(2) safe-skip)
moreover have full-ownership (get-fh H') ∧ no-guard H' ∧ h' = Fractional-
Heap.normalize (get-fh H')
proof –
  have full-ownership (get-fh H')
  proof (rule full-ownershipI)
    fix l p
    assume get-fh H' l = Some p
    show fst p = pwrite
    proof (cases l = v)
      case True
        then have get-fh hf l = None
          using calculation(1) by blast
        then have get-fh H' l = (Map.empty(v ↦ (pwrite, edenot E sa))) l
          by (metis calculation(2) fst-conv get-fh.simps sum-second-none-get-fh)
        then show ?thesis
          using True ⟨get-fh H' l = Some p⟩ by fastforce
      next
        case False
          then have get-fh ([v ↦ (pwrite, edenot E sa)], gs, gu) l = None
            by simp
          then show fst p = pwrite
            by (metis (mono-tags, lifting) ⟨get-fh H' l = Some p⟩ asm0 calculation(2)
fst-conv full-ownership-def get-fh.elims sum-first-none-get-fh)
          qed
        qed
    moreover have no-guard H'
    proof –
      have no-guard hf
        by (metis asm0 no-guard-then-smaller-same plus-comm)
      moreover have no-guard (Map.empty, gs, gu)
        using asm0 no-guard-then-smaller-same by blast
      ultimately show ?thesis
        by (metis ⟨Some H' = Some ([v ↦ (pwrite, edenot E sa)], gs,
gu) ⊕ Some hf⟩ decompose-heap-triple no-guard-remove(1) no-guard-remove(2)
no-guards-remove remove-guards-def snd-conv)
      qed
    moreover have h' = FractionalHeap.normalize (get-fh H')
    proof (rule ext)
      fix l show h' l = FractionalHeap.normalize (get-fh H') l
      proof (cases l = v)
        case True
          then have get-fh (Map.empty(v ↦ (pwrite, edenot E sa)), gs, gu) l =
Some (pwrite, edenot E sa)
            by auto
          then have get-fh hf l = None
            using True ⟨v ∉ dom (get-fh hf)⟩ by force

```

```

then show  $h' l = \text{FractionalHeap.normalize (get-fh } H') l$ 
  apply (cases  $h' l$ )
  using True asm1(3) apply auto[1]
  by (metis (no-types, lifting) FractionalHeap.normalize-def True  $\langle \text{Some } H' = \text{Some } ([v \mapsto (\text{pwrite}, \text{edenot } E \text{ sa})], \text{gs}, \text{gu}) \oplus \text{Some } hf \rangle \langle \text{get-fh } ([v \mapsto (\text{pwrite}, \text{edenot } E \text{ sa})], \text{gs}, \text{gu}) l = \text{Some } (\text{pwrite}, \text{edenot } E \text{ sa}) \rangle \text{apply-opt.simps}(2) \text{asm1}(3) \text{fun-upd-same snd-conv sum-second-none-get-fh}$ )
  next
  case False
  then have  $\text{get-fh (Map.empty}(v \mapsto (\text{pwrite}, \text{edenot } E \text{ sa})), \text{gs}, \text{gu}) l = \text{None}$ 
  by simp
  then have  $\text{get-fh } H' l = \text{get-fh } hf l$ 
  using  $\langle \text{Some } H' = \text{Some } ([v \mapsto (\text{pwrite}, \text{edenot } E \text{ sa})], \text{gs}, \text{gu}) \oplus \text{Some } hf \rangle \text{sum-first-none-get-fh}$  by blast
  moreover have  $\text{get-fh } H l = \text{get-fh } hf l$ 
  by (metis asm0 fst-conv get-fh.elims plus-comm sum-second-none-get-fh)
  ultimately show ?thesis
  proof (cases  $\text{get-fh } hf l$ )
  case None
  then show ?thesis
  by (metis False FractionalHeap.normalize-eq(1)  $\langle \text{get-fh } H l = \text{get-fh } hf l \rangle \langle \text{get-fh } H' l = \text{get-fh } hf l \rangle \text{asm1}(1) \text{asm1}(3) \text{fun-upd-apply old.prod.inject}$ )
  next
  case (Some f)
  then show ?thesis
  by (metis (no-types, lifting) False FractionalHeap.normalize-eq(1) FractionalHeap.normalize-eq(2)  $\langle \text{get-fh } H l = \text{get-fh } hf l \rangle \langle \text{get-fh } H' l = \text{get-fh } hf l \rangle \text{asm1}(1) \text{asm1}(3) \text{domD not-in-dom fun-upd-apply old.prod.inject}$ )
  qed
  qed
  qed
  ultimately show ?thesis
  by auto
  qed
  ultimately show  $\exists h'' H'. \text{full-ownership (get-fh } H') \wedge \text{no-guard } H' \wedge h' = \text{FractionalHeap.normalize (get-fh } H') \wedge \text{Some } H' = \text{Some } h'' \oplus \text{Some } hf \wedge \text{safe } n (\text{None} :: ('i, 'a, \text{nat}) \text{cont}) C' (s', h'') \{ (s(x := a), [a \mapsto (\text{pwrite}, \text{edenot } E \text{ s})], \text{gs}, \text{gu}) | a. \text{True} \}$ 
  by blast
  qed
  qed
  qed (simp)

```

lemma *safe-new-Some*:

```

assumes  $x \notin \text{fvA (invariant } \Gamma)$ 
and view-function-of-inv  $\Gamma$ 
shows  $\text{safe } n (\text{Some } \Gamma) (\text{Calloc } x E) (s, (\text{Map.empty}, \text{gs}, \text{gu})) \{ (s(x := a), (\text{Map.empty}(a \mapsto (\text{pwrite}, \text{edenot } E \text{ s})), \text{gs}, \text{gu})) | a. \text{True} \}$ 

```

```

proof (induct n)
  case (Suc n)
  show ?case
  proof (rule safeSomeI)
    show Calloc x E = Cskip  $\implies$  (s, Map.empty, gs, gu)  $\in$  {(s(x := a), [a  $\mapsto$ 
    (pwrite, edenot E s)], gs, gu) | a. True} by simp
    show no-abort (Some  $\Gamma$ ) (Calloc x E) s (Map.empty, gs, gu)
      using aborts-alloc-cases no-abort.simps(2) by blast
    fix H hf C' s' h' hj v0
    assume asm0: Some H = Some (Map.empty, gs, gu)  $\oplus$  Some hj  $\oplus$  Some hf  $\wedge$ 
      full-ownership (get-fh H)  $\wedge$  semi-consistent  $\Gamma$  v0 H  $\wedge$  sat-inv s hj  $\Gamma$   $\wedge$  red
      (Calloc x E) (s, FractionalHeap.normalize (get-fh H)) C' (s', h')

  then obtain hjf where Some hjf = Some hj  $\oplus$  Some hf
    by (metis plus.simps(2) plus.simps(3) plus-asso)
  then have Some H = Some (Map.empty, gs, gu)  $\oplus$  Some hjf
    by (metis asm0 plus-asso)

  show  $\exists$  h'' H' hj'.
    full-ownership (get-fh H')  $\wedge$ 
    semi-consistent  $\Gamma$  v0 H'  $\wedge$ 
    sat-inv s' hj'  $\Gamma$   $\wedge$ 
    h' = FractionalHeap.normalize (get-fh H')  $\wedge$ 
    Some H' = Some h''  $\oplus$  Some hj'  $\oplus$  Some hf  $\wedge$  safe n (Some  $\Gamma$ ) C' (s',
    h'') {(s(x := a), [a  $\mapsto$  (pwrite, edenot E s)], gs, gu) | a. True}
  proof (rule red-alloc-cases)
    show red (Calloc x E) (s, FractionalHeap.normalize (get-fh H)) C' (s', h')
      using asm0 by blast
    fix sa h v
    assume asm1: (s, FractionalHeap.normalize (get-fh H)) = (sa, h) C' = Cskip
      (s', h') = (sa(x := v), h(v  $\mapsto$  edenot E sa))
      v  $\notin$  dom h
    then have v  $\notin$  dom (get-fh H)
      by (simp add: dom-normalize)
    then have v  $\notin$  dom (get-fh hjf)
      by (metis (no-types, lifting)  $\langle$ Some H = Some (Map.empty, gs, gu)  $\oplus$  Some
      hjf $\rangle$  addition-smaller-domain in-mono plus-comm)

  moreover have (Map.empty(v  $\mapsto$  (pwrite, edenot E sa)), gs, gu)  $\#\#$  hjf
  proof (rule compatibleI)
    show compatible-fract-heaps (get-fh ([v  $\mapsto$  (pwrite, edenot E sa)], gs, gu))
    (get-fh hjf)
  proof (rule compatible-fract-heapsI)
    fix l p p'
    assume asm2: get-fh ([v  $\mapsto$  (pwrite, edenot E sa)], gs, gu) l = Some p  $\wedge$ 
    get-fh hjf l = Some p'
    then show pgte pwrite (padd (fst p) (fst p'))
      by (metis calculation domIff fst-conv fun-upd-other get-fh.elims op-

```

```

tion.distinct(1))
  show snd p = snd p'
  using asm2 calculation domIff fst-conv fun-upd-other get-fh.elims option.distinct(1) by metis
qed
show  $\bigwedge k. \text{get-gu} ([v \mapsto (\text{pwrite}, \text{edenot } E \text{ sa})], \text{gs}, \text{gu}) k = \text{None} \vee \text{get-gu} \text{ hjf } k = \text{None}$ 
by (metis  $\langle \text{Some } H = \text{Some} (\text{Map.empty}, \text{gs}, \text{gu}) \oplus \text{Some } \text{hjff} \rangle \text{compatible-def compatible-eq get-gu.simps option.discI snd-conv}$ )
show  $\bigwedge p p'. \text{get-gs} ([v \mapsto (\text{pwrite}, \text{edenot } E \text{ sa})], \text{gs}, \text{gu}) = \text{Some } p \wedge \text{get-gs} \text{ hjff} = \text{Some } p' \implies \text{pgte } \text{pwrite} (\text{padd } (\text{fst } p) (\text{fst } p'))$ 
by (metis  $\langle \text{Some } H = \text{Some} (\text{Map.empty}, \text{gs}, \text{gu}) \oplus \text{Some } \text{hjff} \rangle \text{compatible-def compatible-eq get-gs.simps option.simps(3) snd-eqD}$ )
qed
then obtain  $H'$  where  $\text{Some } H' = \text{Some} (\text{Map.empty}(v \mapsto (\text{pwrite}, \text{edenot } E \text{ sa})), \text{gs}, \text{gu}) \oplus \text{Some } \text{hjff}$ 
by auto
moreover have  $(s', (\text{Map.empty}(v \mapsto (\text{pwrite}, \text{edenot } E \text{ sa})), \text{gs}, \text{gu})) \in \{(s(x := a), [a \mapsto (\text{pwrite}, \text{edenot } E \text{ sa})], \text{gs}, \text{gu}) \mid a. \text{True}\}$ 
using asm1(1) asm1(3) by blast
then have  $\text{safe } n (\text{Some } \Gamma) C' (s', (\text{Map.empty}(v \mapsto (\text{pwrite}, \text{edenot } E \text{ sa})), \text{gs}, \text{gu})) \{(s(x := a), [a \mapsto (\text{pwrite}, \text{edenot } E \text{ sa})], \text{gs}, \text{gu}) \mid a. \text{True}\}$ 
by (simp add: asm1(2) safe-skip)

moreover have  $\text{full-ownership } (\text{get-fh } H') \wedge \text{semi-consistent } \Gamma \text{ v0 } H' \wedge h' = \text{FractionalHeap.normalize } (\text{get-fh } H')$ 
proof -
have  $\text{full-ownership } (\text{get-fh } H')$ 
proof (rule  $\text{full-ownershipI}$ )
fix  $l p$ 
assume  $\text{get-fh } H' l = \text{Some } p$ 
show  $\text{fst } p = \text{pwrite}$ 
proof (cases  $l = v$ )
case True
then have  $\text{get-fh } \text{hjff } l = \text{None}$ 
using  $\text{calculation(1)}$  by blast
then have  $\text{get-fh } H' l = (\text{Map.empty}(v \mapsto (\text{pwrite}, \text{edenot } E \text{ sa}))) l$ 
by (metis  $\text{calculation(2) fst-conv get-fh.simps sum-second-none-get-fh}$ )
then show ?thesis
using  $\text{True} \langle \text{get-fh } H' l = \text{Some } p \rangle$  by fastforce
next
case False
then have  $\text{get-fh } H' l = \text{get-fh } \text{hjff } l$  using  $\text{sum-first-none-get-fh}$ [of  $H' - \text{hjff } l$ ]
using  $\text{calculation(2)}$  by force
then show ?thesis
by (metis (no-types, lifting)  $\langle \text{Some } H = \text{Some} (\text{Map.empty}, \text{gs}, \text{gu}) \oplus \text{Some } \text{hjff} \rangle \langle \text{get-fh } H' l = \text{Some } p \rangle \text{asm0 fst-conv full-ownership-def get-fh.elims plus-comm sum-second-none-get-fh}$ )

```

```

qed
qed
moreover have  $h' = \text{FractionalHeap.normalize (get-fh } H')$ 
proof (rule ext)
  fix l show  $h' l = \text{FractionalHeap.normalize (get-fh } H') l$ 
  proof (cases l = v)
    case True
      then have  $\text{get-fh (Map.empty}(v \mapsto (\text{pwrite}, \text{edenot } E \text{ sa})), \text{gs}, \text{gu}) l =$ 
 $\text{Some (pwrite, edenot } E \text{ sa)}$ 
      by auto
      then have  $\text{get-fh hjf } l = \text{None}$ 
      using  $\text{True } \langle v \notin \text{dom (get-fh hjf)} \rangle$  by force
      then show ?thesis
      apply (cases h' l)
      using True asm1(3) apply auto[1]
      by (metis (no-types, lifting) FractionalHeap.normalize-def True } \langle \text{Some } H'
 $= \text{Some } ([v \mapsto (\text{pwrite}, \text{edenot } E \text{ sa})], \text{gs}, \text{gu}) \oplus \text{Some hjf} \rangle \langle \text{get-fh } ([v \mapsto (\text{pwrite},$ 
 $\text{edenot } E \text{ sa})], \text{gs}, \text{gu}) l = \text{Some (pwrite, edenot } E \text{ sa}) \rangle \text{apply-opt.simps}(2) \text{asm1}(3)$ 
 $\text{fun-upd-same snd-conv sum-second-none-get-fh})$ 
      next
      case False
      then have  $\text{get-fh (Map.empty}(v \mapsto (\text{pwrite}, \text{edenot } E \text{ sa})), \text{gs}, \text{gu}) l =$ 
 $\text{None}$ 
      by simp
      then have  $\text{get-fh } H' l = \text{get-fh hjf } l$ 
      using  $\langle \text{Some } H' = \text{Some } ([v \mapsto (\text{pwrite}, \text{edenot } E \text{ sa})], \text{gs}, \text{gu}) \oplus \text{Some}$ 
 $\text{hjff} \rangle \text{sum-first-none-get-fh}$  by blast
      moreover have  $\text{get-fh } H l = \text{get-fh hjf } l$ 
      by (metis } \langle \text{Some } H = \text{Some (Map.empty, gs, gu)} \oplus \text{Some hjff} \rangle \text{fst-eqD}
 $\text{get-fh.simps sum-first-none-get-fh})$ 
      ultimately show ?thesis
      proof (cases get-fh hjf l)
        case None
          then show ?thesis
          by (metis False FractionalHeap.normalize-eq(1) } \langle \text{get-fh } H l = \text{get-fh}
 $\text{hjff } l \rangle \langle \text{get-fh } H' l = \text{get-fh hjf } l \rangle \text{asm1}(1) \text{asm1}(3) \text{fun-upd-apply old.prod.inject})$ 
          next
          case (Some f)
          then show ?thesis
          by (metis (no-types, lifting) False FractionalHeap.normalize-eq(1)
 $\text{FractionalHeap.normalize-eq}(2) \langle \text{get-fh } H l = \text{get-fh hjf } l \rangle \langle \text{get-fh } H' l = \text{get-fh hjf}$ 
 $l \rangle \text{asm1}(1) \text{asm1}(3) \text{domD not-in-dom fun-upd-apply old.prod.inject})$ 
          qed
        qed
      qed
    moreover have semi-consistent  $\Gamma \ v0 \ H'$ 
    proof (rule semi-consistentI)
      have  $\text{get-gs } H' = \text{get-gs } H$ 
      by (metis } \langle \text{Some } H = \text{Some (Map.empty, gs, gu)} \oplus \text{Some hjff} \rangle \langle \text{Some } H'

```

```

= Some ([v ↦ (pwrite, edenot E sa)], gs, gu) ⊕ Some hjf › fst-conv get-gs.simps
option.discI option.sel plus.simps(3) snd-conv)
  moreover have get-gu H' = get-gu H
  by (metis ‹Some H = Some (Map.empty, gs, gu) ⊕ Some hjf › ‹Some H' =
Some ([v ↦ (pwrite, edenot E sa)], gs, gu) ⊕ Some hjf › get-gu.simps option.discI
option.sel plus.simps(3) snd-conv)
  ultimately show all-guards H'
    by (metis all-guards-def asm0 semi-consistent-def)
  show reachable Γ v0 H'
  proof (rule reachableI)
    fix sargs uargs
    assume get-gs H' = Some (pwrite, sargs) ∧ (∀ k. get-gu H' k = Some
(uargs k))
    then have reachable-value (saction Γ) (uaction Γ) v0 sargs uargs (view
Γ (FractionalHeap.normalize (get-fh H)))
    by (metis ‹get-gs H' = get-gs H › ‹get-gu H' = get-gu H › asm0 reachableE
semi-consistent-def)
    moreover have view Γ (FractionalHeap.normalize (get-fh H)) = view Γ
(FractionalHeap.normalize (get-fh H'))
    proof –
      have view Γ (FractionalHeap.normalize (get-fh H)) = view Γ
(FractionalHeap.normalize (get-fh hj))
      using view-function-of-invE[of Γ s hj H] by (simp add: asm0 assms(2)
larger3)
      moreover have view Γ (FractionalHeap.normalize (get-fh H')) = view
Γ (FractionalHeap.normalize (get-fh hj))
      using view-function-of-invE[of Γ s hj H']
      by (metis ‹Some H' = Some ([v ↦ (pwrite, edenot E sa)], gs, gu) ⊕
Some hjf › ‹Some hjf = Some hj ⊕ Some hf › asm0 assms(2) larger3 plus-comm)
      ultimately show ?thesis by simp
    qed
    ultimately show reachable-value (saction Γ) (uaction Γ) v0 sargs uargs
(view Γ (FractionalHeap.normalize (get-fh H')))
    by simp
  qed
qed
ultimately show ?thesis
  by auto
qed

moreover have sat-inv s' hj Γ
proof (rule sat-invI)
  show no-guard hj
    using asm0 sat-inv-def by blast
  have agrees (fvA (invariant Γ)) s s'
    using asm1(1) asm1(3) assms
    by (simp add: agrees-update)
  then show (s', hj), (s', hj) ⊨ invariant Γ
    using asm0 sat-inv-agrees sat-inv-def by blast

```


qed

ultimately show $\exists h'' H' hj'. \text{full-ownership } (get\text{-fh } H') \wedge \text{semi-consistent } \Gamma$
 $v0 H' \wedge \text{sat-inv } s' hj' \Gamma \wedge h' = \text{FractionalHeap.normalize } (get\text{-fh } H') \wedge$
 $\text{Some } H' = \text{Some } h'' \oplus \text{Some } hj' \oplus \text{Some } hf \wedge \text{safe } n \text{ (Some } \Gamma) C' (s',$
 $h'') \{ (s(x := a), [a \mapsto (pwrite, edenot E s)], gs, gu) \mid a. \text{True} \}$
by $(metis \text{ (no-types, lifting) } \langle \text{Some } hjf = \text{Some } hj \oplus \text{Some } hf \rangle \text{ plus-asso})$
qed
qed
qed $(simp)$

lemma *safe-new*:

fixes $\Delta :: ('i, 'a, nat) \text{ cont}$
assumes $\bigwedge \Gamma. \Delta = \text{Some } \Gamma \implies x \notin \text{fv} A \text{ (invariant } \Gamma) \wedge \text{view-function-of-inv } \Gamma$
shows $\text{safe } n \Delta (\text{Calloc } x E) (s, (\text{Map.empty}, gs, gu)) \{ (s(x := a), (\text{Map.empty}(a$
 $\mapsto (pwrite, edenot E s)), gs, gu)) \mid a. \text{True} \}$
apply $(\text{cases } \Delta)$
using *safe-new-None safe-new-Some assms* **by** *blast+*

theorem *new-rule*:

fixes $\Delta :: ('i, 'a, nat) \text{ cont}$
assumes $x \notin \text{fv} E$
and $\bigwedge \Gamma. \Delta = \text{Some } \Gamma \implies x \notin \text{fv} A \text{ (invariant } \Gamma) \wedge \text{view-function-of-inv } \Gamma$
shows *hoare-triple-valid* $\Delta \text{Emp } (\text{Calloc } x E) (\text{PointsTo } (Evar x) pwrite E)$
proof $(\text{rule hoare-triple-validI})$
define $\Sigma :: \text{store} \times ('i, 'a) \text{ heap} \Rightarrow (\text{store} \times ('i, 'a) \text{ heap}) \text{ set}$ **where** $\Sigma = (\lambda (s,$
 $h). \{ (s(x := a), (\text{Map.empty}(a \mapsto (pwrite, edenot E s))), \text{get-gs } h, \text{get-gu } h)) \mid a. \text{True} \})$

show $\bigwedge s h n. (s, h), (s, h) \models \text{Emp} \implies \text{safe } n \Delta (\text{Calloc } x E) (s, h) (\Sigma (s, h))$
proof $-$
fix $s h n$ **assume** $(s, h :: ('i, 'a) \text{ heap}), (s, h) \models \text{Emp}$ **then have** $\text{get-fh } h =$
 Map.empty
by *simp*
then have $h = (\text{Map.empty}, \text{get-gs } h, \text{get-gu } h)$ **using** *decompose-heap-triple*
by *metis*
moreover have $\text{safe } n \Delta (\text{Calloc } x E) (s, \text{Map.empty}, \text{get-gs } h, \text{get-gu } h) \{ (s(x$
 $:= a), [a \mapsto (pwrite, edenot E s)], \text{get-gs } h, \text{get-gu } h) \mid a. \text{True} \}$
using *safe-new assms(2)* **by** *blast*
moreover have $\Sigma (s, h) = \{ (s(x := a), (\text{Map.empty}(a \mapsto (pwrite, edenot E$
 $s))), \text{get-gs } h, \text{get-gu } h)) \mid a. \text{True} \}$
using $\Sigma\text{-def}$ **by** *force*
ultimately show $\text{safe } n \Delta (\text{Calloc } x E) (s, h) (\Sigma (s, h))$
by *presburger*
qed
fix $s1 h1 s2 h2$
assume $(s1, h1 :: ('i, 'a) \text{ heap}), (s2, h2) \models \text{Emp}$

```

show pair-sat (case (s1, h1) of (s, h) ⇒ {(s(x := a), [a ↦ (pwrite, edenot E
s)], get-gs h, get-gu h) | a. True})
  (case (s2, h2) of (s, h) ⇒ {(s(x := a), [a ↦ (pwrite, edenot E s)], get-gs h,
get-gu h) | a. True}) (PointsTo (Evar x) pwrite E)
proof (rule pair-satI)
  fix s1' h1' s2' h2'
    assume asm0: (s1', h1') ∈ (case (s1, h1) of (s, h) ⇒ {(s(x := a), [a ↦
(pwrite, edenot E s)], get-gs h, get-gu h) | a. True}) ∧
      (s2', h2') ∈ (case (s2, h2) of (s, h) ⇒ {(s(x := a), [a ↦ (pwrite, edenot E
s)], get-gs h, get-gu h) | a. True})
    then obtain a1 a2 where s1' = s1(x := a1) s2' = s2(x := a2) h1' = ([a1
↦ (pwrite, edenot E s1)], get-gs h1, get-gu h1)
      h2' = ([a2 ↦ (pwrite, edenot E s2)], get-gs h2, get-gu h2)
    by blast
    then show (s1', h1'), (s2', h2') ⊨ PointsTo (Evar x) pwrite E
      by (simp add: assms(1))
  qed
qed

```

4.4.4 Write

inductive-cases red-write-cases: red (Cwrite x E) σ C' σ'

inductive-cases aborts-write-cases: aborts (Cwrite x E) σ

lemma safe-write-None:

```

assumes fh (edenot loc s) = Some (pwrite, v)
shows safe n (None :: ('i, 'a, nat) cont) (Cwrite loc E) (s, (fh, gs, gu)) { (s,
(fh(edenot loc s ↦ (pwrite, edenot E s)), gs, gu)) }
using assms
proof (induct n)
  case (Suc n)
  show ?case
  proof (rule safeNoneI)
    show Cwrite loc E = Cskip ⇒ (s, fh, gs, gu) ∈ {(s, fh(edenot loc s ↦ (pwrite,
edenot E s)), gs, gu)}
    by simp
    show no-abort None (Cwrite loc E) s (fh, gs, gu)
  proof (rule no-abortNoneI)
    fix hf H assume asm0: Some H = Some (fh, gs, gu) ⊕ Some hf ∧
full-ownership (get-fh H) ∧ no-guard H
    then have edenot loc s ∈ dom (normalize (get-fh H))
      by (metis (mono-tags, lifting) Suc.premis addition-smaller-domain dom-def
dom-normalize fst-conv get-fh.simps mem-Collect-eq option.discI subsetD)
    then show ¬ aborts (Cwrite loc E) (s, normalize (get-fh H))
      by (metis aborts-write-cases fst-eqD snd-eqD)
  qed

```

fix H hf C' s' h'

assume *asm0*: *Some H = Some (fh, gs, gu) \oplus Some hf \wedge full-ownership (get-fh H) \wedge no-guard H*
 \wedge *red (Cwrite loc E) (s, FractionalHeap.normalize (get-fh H)) C' (s', h')*
then have *get-fh hf (edenot loc s) = None*
proof –
have *compatible-fract-heaps fh (get-fh hf)*
by (*metis asm0 compatible-def compatible-eq fst-conv get-fh.elims option.discI*)
then show *?thesis using compatible-then-dom-disjoint(2)[of fh get-fh hf]*
assms disjoint-iff-not-equal[of dom (get-fh hf) fpdom fh] not-in-dom fp-
dom-def mem-Collect-eq
by *fastforce*
qed

show $\exists h'' H'. \text{full-ownership (get-fh } H') \wedge \text{no-guard } H' \wedge h' = \text{FractionalHeap.normalize (get-fh } H')$
 $\wedge \text{Some } H' = \text{Some } h'' \oplus \text{Some hf} \wedge \text{safe } n \text{ None } C' (s', h'') \{(s, \text{fh(edenot loc } s \mapsto (\text{pwrite, edenot } E \ s)), \text{gs, gu})\}$
proof (*rule red-write-cases*)
show *red (Cwrite loc E) (s, FractionalHeap.normalize (get-fh H)) C' (s', h')*
using *asm0 by blast*
fix *sa h*
assume *asm1*: $(s, \text{FractionalHeap.normalize (get-fh H)}) = (sa, h) \ C' = \text{Cskip}$
 $(s', h') = (sa, h(\text{edenot loc } sa \mapsto \text{edenot } E \ sa))$
then obtain $s = sa \ h' = h(\text{edenot loc } s \mapsto \text{edenot } E \ s)$ **by** *blast*

let *?h = (fh(edenot loc s \mapsto (pwrite, edenot E s)), gs, gu)*
have *?h ## hf*
proof (*rule compatibleI*)
show *compatible-fract-heaps (get-fh (fh(edenot loc s \mapsto (pwrite, edenot E s)), gs, gu)) (get-fh hf)*
proof (*rule compatible-fract-heapsI*)
fix *l p p'* **assume** *asm2*: *get-fh (fh(edenot loc s \mapsto (pwrite, edenot E s)), gs, gu) l = Some p \wedge get-fh hf l = Some p'*
then show *pgte pwrite (padd (fst p) (fst p'))*
apply (*cases l = edenot loc s*)
apply (*metis Suc.prem1 asm0 fst-conv fun-upd-same get-fh.elims option.sel plus-extract-point-fh*)
by (*metis asm0 fst-conv fun-upd-other get-fh.elims plus-extract-point-fh*)
show *snd p = snd p'*
apply (*cases l = edenot loc s*)
using $\langle \text{pgte pwrite (padd (fst p) (fst p'))} \rangle$ *asm2 not-pgte-charact sum-larger*
apply *fastforce*
by (*metis (mono-tags, opaque-lifting) asm0 asm2 fst-eqD get-fh.simps map-upd-Some-unfold plus-extract-point-fh*)
qed
show $\bigwedge k. \text{get-gu (fh(edenot loc } s \mapsto (\text{pwrite, edenot } E \ s)), \text{gs, gu}) } k = \text{None}$
 $\vee \text{get-gu hf } k = \text{None}$
by (*metis asm0 compatible-def compatible-eq get-gu.simps option.discI snd-conv*)

show $\bigwedge p p'. \text{get-gs } (\text{fh}(\text{edenot } \text{loc } s \mapsto (\text{pwrite}, \text{edenot } E \ s)), \text{gs}, \text{gu}) = \text{Some } p \wedge \text{get-gs } \text{hf} = \text{Some } p' \implies \text{pgte } \text{pwrite } (\text{padd } (\text{fst } p) (\text{fst } p'))$
by (*metis asm0 no-guard-def no-guard-then-smaller-same option.simps(3) plus-comm*)
qed
then obtain H' where $\text{Some } H' = \text{Some } ?h \oplus \text{Some } \text{hf}$ **by auto**
moreover have $H' = ((\text{get-fh } H)(\text{edenot } \text{loc } s \mapsto (\text{pwrite}, \text{edenot } E \ s)), \text{get-gs } H, \text{get-gu } H)$
proof (*rule heap-ext*)
show $\text{get-fh } H' = \text{get-fh } ((\text{get-fh } H)(\text{edenot } \text{loc } s \mapsto (\text{pwrite}, \text{edenot } E \ s)), \text{get-gs } H, \text{get-gu } H)$
using *calculation asm0 by (metis <get-fh hf (edenot loc s) = None> add-fh-update add-get-fh fst-conv get-fh.simps)*
show $\text{get-gs } H' = \text{get-gs } ((\text{get-fh } H)(\text{edenot } \text{loc } s \mapsto (\text{pwrite}, \text{edenot } E \ s)), \text{get-gs } H, \text{get-gu } H)$
using *calculation asm0*
by (*metis fst-conv get-gs.simps plus-extract(2) snd-conv*)
show $\text{get-gu } H' = \text{get-gu } ((\text{get-fh } H)(\text{edenot } \text{loc } s \mapsto (\text{pwrite}, \text{edenot } E \ s)), \text{get-gs } H, \text{get-gu } H)$
using *add-fh-update[of get-fh hf edenot E s fh (pwrite, edenot E s)] asm0 calculation*
by (*metis get-gu.elims plus-extract(3) snd-conv*)
qed
moreover have $\text{safe } n \ (\text{None} :: ('i, 'a, \text{nat}) \text{cont}) \ C' \ (s', ?h) \ \{(s, \text{fh}(\text{edenot } \text{loc } s \mapsto (\text{pwrite}, \text{edenot } E \ s)), \text{gs}, \text{gu})\}$
using *<s = sa> asm1(2) asm1(3) safe-skip by fastforce*
moreover have $\text{full-ownership } (\text{get-fh } H') \wedge \text{no-guard } H' \wedge h' = \text{Fractional-Heap.normalize } (\text{get-fh } H')$
proof –
have $\text{full-ownership } (\text{get-fh } H')$
proof (*rule full-ownershipI*)
fix $l \ p$
assume $\text{asm}: \text{get-fh } H' \ l = \text{Some } p$
then show $\text{fst } p = \text{pwrite}$
proof (*cases l = edenot loc s*)
case *True*
then show *?thesis*
using *asm calculation(2) by fastforce*
next
case *False*
then show *?thesis*
by (*metis (mono-tags, lifting) asm asm0 calculation(2) fst-eqD full-ownership-def get-fh.simps map-upd-Some-unfold*)
qed
qed
moreover have $\text{no-guard } H'$ **using** *asm0*
by (*simp add: <H' = ((get-fh H)(edenot loc s mapsto (pwrite, edenot E s)), get-gs H, get-gu H)> no-guard-def*)
moreover have $h' = \text{FractionalHeap.normalize } (\text{get-fh } H')$

```

proof (rule ext)
  fix l show h' l = FractionalHeap.normalize (get-fh H') l
  proof(cases l = edenot loc s)
    case True
      then show ?thesis
        by (metis (no-types, lifting) FractionalHeap.normalize-eq(2) ⟨H'
= ((get-fh H)(edenot loc s ↦ (pwrite, edenot E s)), get-gs H, get-gu H)⟩ ⟨h' =
h(edenot loc s ↦ edenot E s)⟩ fst-conv fun-upd-same get-fh.elims)
      next
        case False
          then have FractionalHeap.normalize (get-fh H') l = Fractional-
Heap.normalize (get-fh H) l
            using FractionalHeap.normalize-eq(2)[of get-fh H' l]
              FractionalHeap.normalize-eq(2)[of get-fh H l] ⟨H' = ((get-fh H)(edenot
loc s ↦ (pwrite, edenot E s)), get-gs H, get-gu H)⟩
                fst-conv fun-upd-other[of l edenot loc s get-fh H] get-fh.simps
option.exhaust
            by metis
          then show ?thesis
            using False ⟨h' = h(edenot loc s ↦ edenot E s)⟩ asm1(1) by force
          qed
        qed
      ultimately show ?thesis
        by auto
      qed
    ultimately show ∃ h'' H'. full-ownership (get-fh H') ∧ no-guard H' ∧ h' =
FractionalHeap.normalize (get-fh H') ∧
      Some H' = Some h'' ⊕ Some hf ∧ safe n None C' (s', h'') {(s, fh(edenot
loc s ↦ (pwrite, edenot E s)), gs, gu)}
    by (metis ⟨Some H' = Some (fh(edenot loc s ↦ (pwrite, edenot E s)), gs,
gu) ⊕ Some hf⟩ ⟨s = sa⟩ asm1(2) asm1(3) fst-conv insertI1 safe-skip)
    qed
  qed
qed (simp)

```

lemma safe-write-Some:

```

assumes fh (edenot loc s) = Some (pwrite, v)
and view-function-of-inv Γ
shows safe n (Some Γ) (Cwrite loc E) (s, (fh, gs, gu)) { (s, (fh(edenot loc s ↦
(pwrite, edenot E s)), gs, gu)) }
using assms
proof (induct n)
  case (Suc n)
  show ?case
  proof (rule safeSomeI)
    show Cwrite loc E = Cskip ⇒ (s, fh, gs, gu) ∈ {(s, fh(edenot loc s ↦ (pwrite,
edenot E s)), gs, gu)}

```

by *simp*
show *no-abort* (*Some* Γ) (*Cwrite loc E*) *s* (*fh*, *gs*, *gu*)
proof (*rule no-abortSomeI*)
fix *H hf hj v0*
assume *asm0*: *Some H = Some (fh, gs, gu) \oplus Some hj \oplus Some hf \wedge*
full-ownership (get-fh H) \wedge semi-consistent Γ v0 H \wedge sat-inv s hj Γ
then have *edenot loc s \in dom (get-fh H)*
by (*metis Un-iff assms(1) domI dom-three-sum fst-conv get-fh.simps*)
then have *edenot loc s \in dom (normalize (get-fh H))*
by (*simp add: dom-normalize*)
then show \neg *aborts (Cwrite loc E) (s, FractionalHeap.normalize (get-fh H))*
by (*metis aborts-write-cases fst-eqD snd-eqD*)
qed

fix *H hf C' s' h' hj v0*

assume *asm0*: *Some H = Some (fh, gs, gu) \oplus Some hj \oplus Some hf \wedge*
full-ownership (get-fh H) \wedge semi-consistent Γ v0 H \wedge sat-inv s hj Γ \wedge red
(Cwrite loc E) (s, FractionalHeap.normalize (get-fh H)) C' (s', h')
then obtain *hjf where hjf-def: Some hjf = Some hj \oplus Some hf*
by (*metis (no-types, opaque-lifting) option.exhaust-sel plus.simps(1) plus-asso*
plus-comm)
then have *asm00: Some H = Some (fh, gs, gu) \oplus Some hjf*
by (*metis asm0 plus-asso*)
then have *get-fh hjf (edenot loc s) = None*
proof –
have *compatible-fract-heaps fh (get-fh hjf)*
by (*metis asm00 compatible-def compatible-eq fst-conv get-fh.elims option.discI*)
then show *?thesis using compatible-then-dom-disjoint(2)[of fh get-fh hjf]*
assms disjoint-iff-not-equal[of dom (get-fh hjf) fpdom fh] not-in-dom
fpdom-def mem-Collect-eq
by *fastforce*
qed

show $\exists h'' H' hj'$. *full-ownership (get-fh H') \wedge semi-consistent Γ v0 H' \wedge sat-inv*
s' hj' Γ \wedge h' = FractionalHeap.normalize (get-fh H') \wedge
Some H' = Some h'' \oplus Some hj' \oplus Some hf \wedge safe n (Some Γ) C' (s',
h'') {(s, fh(edenot loc s \mapsto (pwrite, edenot E s)), gs, gu)}
proof (*rule red-write-cases*)
show *red (Cwrite loc E) (s, FractionalHeap.normalize (get-fh H)) C' (s', h')*
using *asm0 by blast*
fix *sa h*
assume *asm1: (s, FractionalHeap.normalize (get-fh H)) = (sa, h) C' = Cskip*
(s', h') = (sa, h(edenot loc sa \mapsto edenot E sa))
then obtain *s = sa h' = h(edenot loc s \mapsto edenot E s) by blast*

let *?h = (fh(edenot loc s \mapsto (pwrite, edenot E s)), gs, gu)*
have *?h ## hjf*

proof (*rule compatibleI*)
show *compatible-fract-heaps* (*get-fh* ($\text{fh}(\text{edenot loc } s \mapsto (\text{pwrite}, \text{edenot } E \text{ s}))$), *gs*, *gu*) (*get-fh hjf*)
proof (*rule compatible-fract-heapsI*)
fix $l \ p \ p'$ **assume** *asm2*: *get-fh* ($\text{fh}(\text{edenot loc } s \mapsto (\text{pwrite}, \text{edenot } E \text{ s}))$), *gs*, *gu*) $l = \text{Some } p \wedge \text{get-fh } hjf \ l = \text{Some } p'$
then show *pgte pwrite* (*padd* (*fst p*) (*fst p'*))
apply (*cases l = edenot loc s*)
apply (*metis Suc.premis(1) asm00 fst-conv fun-upd-same get-fh.elims option.sel plus-extract-point-fh*)
by (*metis asm00 fst-conv fun-upd-other get-fh.elims plus-extract-point-fh*)
show *snd p = snd p'*
apply (*cases l = edenot loc s*)
using $\langle \text{pgte pwrite } (\text{padd } (\text{fst } p) (\text{fst } p')) \rangle$ *asm2 not-pgte-charact sum-larger*
apply *fastforce*
by (*metis (mono-tags, opaque-lifting) asm00 asm2 fst-eqD get-fh.simps map-upd-Some-unfold plus-extract-point-fh*)
qed
show $\bigwedge k. \text{get-gu } (\text{fh}(\text{edenot loc } s \mapsto (\text{pwrite}, \text{edenot } E \text{ s})), \text{gs}, \text{gu}) \ k = \text{None} \vee \text{get-gu } hjf \ k = \text{None}$
by (*metis asm00 compatible-def compatible-eq get-gu.simps option.discI snd-conv*)
show $\bigwedge p \ p'. \text{get-gs } (\text{fh}(\text{edenot loc } s \mapsto (\text{pwrite}, \text{edenot } E \text{ s})), \text{gs}, \text{gu}) = \text{Some } p \wedge \text{get-gs } hjf = \text{Some } p' \implies \text{pgte pwrite } (\text{padd } (\text{fst } p) (\text{fst } p'))$
by (*metis asm00 compatible-def compatible-eq get-gs.simps option.discI snd-conv*)
qed
then obtain H' **where** $\text{Some } H' = \text{Some } ?h \oplus \text{Some } hjf$ **by** *auto*
moreover have $H' = ((\text{get-fh } H)(\text{edenot loc } s \mapsto (\text{pwrite}, \text{edenot } E \text{ s})), \text{get-gs } H, \text{get-gu } H)$
proof (*rule heap-ext*)
show $\text{get-fh } H' = \text{get-fh } ((\text{get-fh } H)(\text{edenot loc } s \mapsto (\text{pwrite}, \text{edenot } E \text{ s})), \text{get-gs } H, \text{get-gu } H)$
using *asm00 calculation*
by (*metis* $\langle \text{get-fh } hjf (\text{edenot loc } s) = \text{None} \rangle$ *add-fh-update add-get-fh fst-conv get-fh.simps*)
show $\text{get-gs } H' = \text{get-gs } ((\text{get-fh } H)(\text{edenot loc } s \mapsto (\text{pwrite}, \text{edenot } E \text{ s})), \text{get-gs } H, \text{get-gu } H)$
using *asm00 calculation*
by (*metis fst-conv get-gs.simps plus-extract(2) snd-conv*)
show $\text{get-gu } H' = \text{get-gu } ((\text{get-fh } H)(\text{edenot loc } s \mapsto (\text{pwrite}, \text{edenot } E \text{ s})), \text{get-gs } H, \text{get-gu } H)$
using *asm00 calculation get-gu.simps plus-extract(3) snd-conv*
qed
moreover have *safe n* ($\text{Some } \Gamma$) $C' (s', ?h) \{(s, \text{fh}(\text{edenot loc } s \mapsto (\text{pwrite}, \text{edenot } E \text{ s})), \text{gs}, \text{gu})\}$
using $\langle s = sa \rangle$ *asm1(2) asm1(3) safe-skip* **by** *fastforce*
moreover have *full-ownership* ($\text{get-fh } H') \wedge h' = \text{FractionalHeap.normalize } (\text{get-fh } H')$

```

proof –
  have full-ownership (get-fh  $H'$ )
  proof (rule full-ownershipI)
    fix  $l\ p$ 
    assume  $asm: \text{get-fh } H' \ l = \text{Some } p$ 
    then show  $\text{fst } p = \text{pwrite}$ 
    proof (cases  $l = \text{edenot } \text{loc } s$ )
      case True
        then show ?thesis
          using  $asm \text{ calculation}(2)$  by fastforce
      next
        case False
          then show ?thesis
            by (metis (mono-tags, lifting)  $asm \ asm0 \ \text{calculation}(2) \ \text{fst-eqD}$ 
full-ownership-def get-fh.simps map-upd-Some-unfold)
          qed
        qed
      moreover have  $h' = \text{FractionalHeap.normalize } (\text{get-fh } H')$ 
      proof (rule ext)
        fix  $l$  show  $h' \ l = \text{FractionalHeap.normalize } (\text{get-fh } H') \ l$ 
        proof(cases  $l = \text{edenot } \text{loc } s$ )
          case True
            then show ?thesis
              by (metis (no-types, lifting)  $\text{FractionalHeap.normalize-eq}(2) \ \langle H' = ((\text{get-fh } H)(\text{edenot } \text{loc } s \mapsto (\text{pwrite}, \text{edenot } E \ s)), \text{get-gs } H, \text{get-gu } H) \rangle \ \langle h' = h(\text{edenot } \text{loc } s \mapsto \text{edenot } E \ s) \rangle \ \text{fst-conv } \text{fun-upd-same } \text{get-fh.elims}$ )
            next
              case False
                then have  $\text{FractionalHeap.normalize } (\text{get-fh } H') \ l = \text{FractionalHeap.normalize } (\text{get-fh } H) \ l$ 
                using  $\text{FractionalHeap.normalize-eq}(2)[\text{of } \text{get-fh } H' \ l]$ 
                 $\text{FractionalHeap.normalize-eq}(2)[\text{of } \text{get-fh } H \ l] \ \langle H' = ((\text{get-fh } H)(\text{edenot } \text{loc } s \mapsto (\text{pwrite}, \text{edenot } E \ s)), \text{get-gs } H, \text{get-gu } H) \rangle$ 
                 $\text{fst-conv } \text{fun-upd-other}[\text{of } l \ \text{edenot } \text{loc } s \ \text{get-fh } H] \ \text{get-fh.simps}$ 
                option.exhaust
              by metis
            then show ?thesis
              using  $\text{False } \langle h' = h(\text{edenot } \text{loc } s \mapsto \text{edenot } E \ s) \rangle \ \text{asm1}(1)$  by force
          qed
        qed
      ultimately show ?thesis
        by auto
    qed
  moreover have  $\text{Some } H' = \text{Some } ?h \oplus \text{Some } hj \oplus \text{Some } hf$ 
  by (metis  $\text{calculation}(1) \ \text{hjf-def simpler-asso}$ )
  moreover have semi-consistent  $\Gamma \ v0 \ H'$ 
  proof (rule semi-consistentI)
    show all-guards  $H'$ 
    by (metis all-guards-def asm0 calculation(2) fst-conv get-gs.simps get-gu.simps)

```


semi-consistent-def snd-conv
have view Γ (normalize (get-fh H')) = view Γ (normalize (get-fh H))
proof –
have view Γ (normalize (get-fh H')) = view Γ (normalize (get-fh hj))
by (metis asm0 assms(2) calculation(5) larger3 view-function-of-invE)
then show ?thesis **using** assms(2) larger3 view-function-of-invE
by (metis asm0)
qed
then show reachable Γ v0 H'
by (metis asm0 calculation(2) fst-eqD get-gs.simps get-gu.simps reachableE
reachableI semi-consistent-def snd-eqD)
qed
ultimately show $\exists h'' H' hj'$.
full-ownership (get-fh H') \wedge
semi-consistent Γ v0 H' \wedge
sat-inv $s' hj' \Gamma$ \wedge
 $h' = \text{FractionalHeap.normalize (get-fh } H') \wedge$
Some $H' = \text{Some } h'' \oplus \text{Some } hj' \oplus \text{Some } hf \wedge \text{safe } n (\text{Some } \Gamma) C' (s',$
 $h'') \{ (s, fh(\text{edenot } loc\ s \mapsto (\text{pwrite}, \text{edenot } E\ s)), gs, gu) \}$
using $\langle s = sa \rangle$ asm0 asm1(2) asm1(3) **by** blast
qed
qed
qed (simp)

lemma safe-write:

fixes $\Delta :: ('i, 'a, nat)$ cont
assumes fh (edenot loc s) = Some (pwrite, v)
and $\bigwedge \Gamma. \Delta = \text{Some } \Gamma \implies \text{view-function-of-inv } \Gamma$
shows safe $n \Delta$ (Cwrite loc E) (s , (fh, gs , gu)) { (s , (fh(edenot loc $s \mapsto (\text{pwrite},$
edenot $E\ s)), gs, gu)) }
apply (cases Δ)
using safe-write-None safe-write-Some assms **by** blast+$

theorem write-rule:

fixes $\Delta :: ('i, 'a, nat)$ cont
assumes $\bigwedge \Gamma. \Delta = \text{Some } \Gamma \implies \text{view-function-of-inv } \Gamma$
and $v \notin \text{fv}E\ loc$
shows hoare-triple-valid Δ (Exists v (PointsTo loc pwrite (Evar v))) (Cwrite loc
 E) (PointsTo loc pwrite E)
proof (rule hoare-triple-validI)

define $\Sigma :: \text{store} \times ('i, 'a) \text{heap} \Rightarrow (\text{store} \times ('i, 'a) \text{heap}) \text{set}$ **where**
 $\Sigma = (\lambda (s, h). \{ (s, ((\text{get-fh } h)(\text{edenot } loc\ s \mapsto (\text{pwrite}, \text{edenot } E\ s)), \text{get-gs } h,$
get-gu $h)) \}$)

show $\bigwedge s h n. (s, h), (s, h) \models \text{Exists } v (\text{PointsTo } loc\ \text{pwrite } (\text{Evar } v)) \implies \text{safe}$
 $n \Delta$ (Cwrite loc E) (s, h) ($\Sigma (s, h)$)
proof –

```

fix  $s\ h\ n$  assume  $(s, h :: ('i, 'a)\ \text{heap}), (s, h) \models \text{Exists } v\ (\text{PointsTo } \text{loc } \text{pwrite } (\text{Evar } v))$ 
then obtain  $vv$  where  $(s(v := vv), h), (s(v := vv), h) \models \text{PointsTo } \text{loc } \text{pwrite } (\text{Evar } v)$ 
by  $(\text{meson } \text{hyper-sat.simps}(6)\ \text{hyper-sat.simps}(7))$ 
then have  $\text{get-fh } h\ (\text{edenot } \text{loc } (s(v := vv))) = \text{Some } (\text{pwrite}, vv)$ 
by  $\text{simp}$ 
then have  $\text{get-fh } h\ (\text{edenot } \text{loc } s) = \text{Some } (\text{pwrite}, vv)$ 
using  $\text{assms}(2)$  by  $\text{auto}$ 
then show  $\text{safe } n\ \Delta\ (\text{Cwrite } \text{loc } E)\ (s, h)\ (\Sigma\ (s, h))$ 
by  $(\text{metis } (\text{mono-tags}, \text{lifting})\ \Sigma\text{-def } \text{assms}(1)\ \text{decompose-heap-triple } \text{old.prod.case } \text{safe-write})$ 
qed
fix  $s1\ h1\ s2\ h2$ 
assume  $(s1, h1 :: ('i, 'a)\ \text{heap}), (s2, h2) \models \text{Exists } v\ (\text{PointsTo } \text{loc } \text{pwrite } (\text{Evar } v))$ 
then obtain  $v1\ v2$  where  $\text{get-fh } h1\ (\text{edenot } \text{loc } s1) = \text{Some } (\text{pwrite}, v1)$   $\text{get-fh } h2\ (\text{edenot } \text{loc } s2) = \text{Some } (\text{pwrite}, v2)$ 
using  $\text{assms}(2)$  by  $\text{auto}$ 

show  $\text{pair-sat } (\text{case } (s1, h1)\ \text{of } (s, h) \Rightarrow \{(s, (\text{get-fh } h)(\text{edenot } \text{loc } s \mapsto (\text{pwrite}, \text{edenot } E\ s)), \text{get-gs } h, \text{get-gu } h)\})$ 
 $(\text{case } (s2, h2)\ \text{of } (s, h) \Rightarrow \{(s, (\text{get-fh } h)(\text{edenot } \text{loc } s \mapsto (\text{pwrite}, \text{edenot } E\ s)), \text{get-gs } h, \text{get-gu } h)\})\ (\text{PointsTo } \text{loc } \text{pwrite } E)$ 
proof  $(\text{rule } \text{pair-satI})$ 
fix  $s1'\ h1'\ s2'\ h2'$ 
assume  $\text{asm0}: (s1', h1') \in (\text{case } (s1, h1)\ \text{of } (s, h) \Rightarrow \{(s, (\text{get-fh } h)(\text{edenot } \text{loc } s \mapsto (\text{pwrite}, \text{edenot } E\ s)), \text{get-gs } h, \text{get-gu } h)\}) \wedge$ 
 $(s2', h2') \in (\text{case } (s2, h2)\ \text{of } (s, h) \Rightarrow \{(s, (\text{get-fh } h)(\text{edenot } \text{loc } s \mapsto (\text{pwrite}, \text{edenot } E\ s)), \text{get-gs } h, \text{get-gu } h)\})$ 
then show  $(s1', h1'), (s2', h2') \models \text{PointsTo } \text{loc } \text{pwrite } E$ 
using  $\langle (s1, h1), (s2, h2) \models \text{Exists } v\ (\text{PointsTo } \text{loc } \text{pwrite } (\text{Evar } v)) \rangle\ \text{assms}(2)$ 
by  $\text{auto}$ 
qed
qed

```

4.4.5 Read

inductive-cases $\text{red-read-cases}: \text{red } (\text{Cread } x\ E)\ \sigma\ C'\ \sigma'$

inductive-cases $\text{aborts-read-cases}: \text{aborts } (\text{Cread } x\ E)\ \sigma$

lemma safe-read-None :

$\text{safe } n\ (\text{None} :: ('i, 'a, \text{nat})\ \text{cont})\ (\text{Cread } x\ E)\ (s, ([\text{edenot } E\ s \mapsto (\pi, v)], \text{gs}, \text{gu}))$
 $\{ (s(x := v), ([\text{edenot } E\ s \mapsto (\pi, v)], \text{gs}, \text{gu})) \}$

proof $(\text{induct } n)$

case $(\text{Suc } n)$

show $?case$

proof $(\text{rule } \text{safeNoneI})$

show *no-abort* (*None* :: ('i, 'a, nat) cont) (*Cread* x E) s ([*edenot* E s \mapsto (π , v)], *gs*, *gu*)
proof (*rule no-abortNoneI*)
fix *hf* H
assume *asm0*: *Some* H = *Some* ([*edenot* E s \mapsto (π , v)], *gs*, *gu*) \oplus *Some* *hf* \wedge *full-ownership* (*get-fh* H) \wedge *no-guard* H
then have *edenot* E s \in *dom* (*get-fh* H)
by (*metis Un-iff dom-eq-singleton-conv dom-sum-two fst-eqD get-fh.elims insert-iff*)
then have *edenot* E s \in *dom* (*FractionalHeap.normalize* (*get-fh* H))
by (*simp add: dom-normalize*)
then show \neg *aborts* (*Cread* x E) (s, *FractionalHeap.normalize* (*get-fh* H))
by (*metis aborts-read-cases fst-eqD snd-eqD*)
qed
fix H *hf* C' s' h'
assume *asm0*: *Some* H = *Some* ([*edenot* E s \mapsto (π , v)], *gs*, *gu*) \oplus *Some* *hf* \wedge *full-ownership* (*get-fh* H) \wedge *no-guard* H \wedge *red* (*Cread* x E) (s, *FractionalHeap.normalize* (*get-fh* H)) C' (s', h')
let ?S = { (s(x := v), ([*edenot* E s \mapsto (π , v)], *gs*, *gu*)) }

show \exists h'' H'.
full-ownership (*get-fh* H') \wedge
no-guard H' \wedge
h' = *FractionalHeap.normalize* (*get-fh* H') \wedge
Some H' = *Some* h'' \oplus *Some* *hf* \wedge *safe* n (*None* :: ('i, 'a, nat) cont) C' (s', h'') ?S
proof (*rule red-read-cases*)

show *red* (*Cread* x E) (s, *FractionalHeap.normalize* (*get-fh* H)) C' (s', h')
using *asm0* **by** *blast*
fix sa h va
assume (s, *FractionalHeap.normalize* (*get-fh* H)) = (sa, h) C' = *Cskip* (s', h') = (sa(x := va), h)
h (*edenot* E sa) = *Some* va
then have s = sa
by *force*
then have va = v
proof –
have \exists π' . *get-fh* H (*edenot* E s) = *Some* (π' , v)
proof (*rule one-value-sum-same*)
show *Some* H = *Some* ([*edenot* E s \mapsto (π , v)], *gs*, *gu*) \oplus *Some* *hf*
using *asm0* **by** *fastforce*
qed (*simp*)
then show ?thesis
by (*metis FractionalHeap.normalize-eq(2) Pair-inject* \langle (s, *FractionalHeap.normalize* (*get-fh* H)) = (sa, h) \rangle \langle h (*edenot* E sa) = *Some* va \rangle *option.sel*)

qed

then have *safe n* (*None* :: ('i, 'a, nat) cont) *C'* (*s'*, ([edenot *E s* \mapsto (π , *v*)], *gs*, *gu*)) ?*S*
using $\langle (s', h') = (sa(x := va), h) \rangle \langle C' = Cskip \rangle \langle s = sa \rangle$ *safe-skip* **by** *fastforce*

then show $\exists h'' H'$.

full-ownership (*get-fh* *H'*) \wedge *no-guard* *H'* \wedge
 $h' = \text{FractionalHeap.normalize}(\text{get-fh } H') \wedge \text{Some } H' = \text{Some } h'' \oplus \text{Some } hf \wedge \text{safe } n$ (*None* :: ('i, 'a, nat) cont) *C'* (*s'*, *h''*) $\{(s(x := v), [\text{edenot } E s \mapsto (\pi, v)], \text{gs}, \text{gu})\}$
using $\langle (s', h') = (sa(x := va), h) \rangle \langle (s, \text{FractionalHeap.normalize}(\text{get-fh } H)) = (sa, h) \rangle$ *asm0* **by** *blast*
qed
qed (*simp*)
qed (*simp*)

lemma *safe-read-Some*:

assumes *view-function-of-inv* Γ
and $x \notin \text{fv}A$ (*invariant* Γ)
shows *safe n* (*Some* Γ) (*Cread* *x E*) (*s*, ([edenot *E s* \mapsto (π , *v*)], *gs*, *gu*)) $\{(s(x := v), [\text{edenot } E s \mapsto (\pi, v)], \text{gs}, \text{gu})\}$
proof (*induct* *n*)
case (*Suc* *n*)
show ?*case*
proof (*rule safeSomeI*)

show *no-abort* (*Some* Γ) (*Cread* *x E*) *s* ([edenot *E s* \mapsto (π , *v*)], *gs*, *gu*)

proof (*rule no-abortSomeI*)

fix *hf* *H* *hj* *v0*

assume *asm0*: *Some* *H* = *Some* ([edenot *E s* \mapsto (π , *v*)], *gs*, *gu*) \oplus *Some* *hj* \oplus *Some* *hf* \wedge *full-ownership* (*get-fh* *H*) \wedge *semi-consistent* Γ *v0* *H* \wedge *sat-inv* *s* *hj* Γ
then obtain *hjf* **where** *Some* *H* = *Some* ([edenot *E s* \mapsto (π , *v*)], *gs*, *gu*) \oplus *Some* *hjf*

by (*metis* (*no-types*, *lifting*) *plus.simps*(2) *plus.simps*(3) *plus-asso*)

then have *edenot* *E s* \in *dom* (*get-fh* *H*)

by (*metis* *Un-iff* *dom-eq-singleton-conv* *dom-sum-two* *fst-eqD* *get-fh.elims* *insert-iff*)

then have *edenot* *E s* \in *dom* (*FractionalHeap.normalize* (*get-fh* *H*))

by (*simp* *add: dom-normalize*)

then show \neg *aborts* (*Cread* *x E*) (*s*, *FractionalHeap.normalize* (*get-fh* *H*))

by (*metis* *aborts-read-cases* *fst-eqD* *snd-eqD*)

qed

fix *H* *hf* *C'* *s'* *h'* *hj* *v0*

assume *asm0*: *Some* *H* = *Some* ([edenot *E s* \mapsto (π , *v*)], *gs*, *gu*) \oplus *Some* *hj* \oplus *Some* *hf* \wedge

full-ownership (*get-fh* *H*) \wedge *semi-consistent* Γ *v0* *H* \wedge *sat-inv* *s* *hj* Γ \wedge *red* (*Cread* *x E*) (*s*, *FractionalHeap.normalize* (*get-fh* *H*)) *C'* (*s'*, *h'*)

then obtain h_{jf} **where** $\text{Some } h_{jf} = \text{Some } h_j \oplus \text{Some } h_f$
using *compatible-last-two* **by** (*metis plus.simps(3) plus-asso*)
then have $\text{Some } H = \text{Some } ([\text{edenot } E \ s \mapsto (\pi, v)], \text{gs}, \text{gu}) \oplus \text{Some } h_{jf}$
by (*metis asm0 plus-asso*)

let $?S = \{ (s(x := v), ([\text{edenot } E \ s \mapsto (\pi, v)], \text{gs}, \text{gu})) \}$

show $\exists h'' \ H' \ h_{j'}. \text{full-ownership } (\text{get-fh } H') \wedge \text{semi-consistent } \Gamma \ v0 \ H' \wedge \text{sat-inv } s' \ h_{j'} \ \Gamma \wedge h' = \text{FractionalHeap.normalize } (\text{get-fh } H') \wedge$
 $\text{Some } H' = \text{Some } h'' \oplus \text{Some } h_{j'} \oplus \text{Some } h_f \wedge \text{safe } n \ (\text{Some } \Gamma) \ C' \ (s',$
 $h'') \{(s(x := v), [\text{edenot } E \ s \mapsto (\pi, v)], \text{gs}, \text{gu})\}$
proof (*rule red-read-cases*)

show $\text{red } (C \text{read } x \ E) \ (s, \text{FractionalHeap.normalize } (\text{get-fh } H)) \ C' \ (s', h')$
using *asm0* **by** *blast*
fix $sa \ h \ va$
assume $(s, \text{FractionalHeap.normalize } (\text{get-fh } H)) = (sa, h) \ C' = C \text{skip } (s',$
 $h') = (sa(x := va), h)$
 $h \ (\text{edenot } E \ sa) = \text{Some } va$
then have $s = sa$
by *force*
then have $va = v$
proof –
have $\exists \pi'. \text{get-fh } H \ (\text{edenot } E \ s) = \text{Some } (\pi', v)$
proof (*rule one-value-sum-same*)
show $\text{Some } H = \text{Some } ([\text{edenot } E \ s \mapsto (\pi, v)], \text{gs}, \text{gu}) \oplus \text{Some } h_{jf}$
by (*simp add*: $\langle \text{Some } H = \text{Some } ([\text{edenot } E \ s \mapsto (\pi, v)], \text{gs}, \text{gu}) \oplus \text{Some } h_{jf} \rangle$)
qed (*simp*)
then show *?thesis*
by (*metis FractionalHeap.normalize-eq(2) Pair-inject* $\langle (s, \text{FractionalHeap.normalize } (\text{get-fh } H)) = (sa, h) \rangle \langle h \ (\text{edenot } E \ sa) = \text{Some } va \rangle \text{option.sel}$)
qed

then have $\text{safe } n \ (\text{Some } \Gamma) \ C' \ (s', ([\text{edenot } E \ s \mapsto (\pi, v)], \text{gs}, \text{gu})) \ ?S$
using $\langle (s', h') = (sa(x := va), h) \rangle \langle C' = C \text{skip} \rangle \langle s = sa \rangle$ *safe-skip* **by**
fastforce

moreover have $\text{sat-inv } s' \ h_{j'} \ \Gamma$
by (*metis* $\langle (s', h') = (sa(x := va), h) \rangle \langle s = sa \rangle$ *agrees-update asm0 assms(2)*)
prod.inject sat-inv-agrees

ultimately show $\exists h'' \ H' \ h_{j'}. \text{full-ownership } (\text{get-fh } H') \wedge \text{semi-consistent } \Gamma \ v0 \ H' \wedge \text{sat-inv } s' \ h_{j'} \ \Gamma \wedge$
 $h' = \text{FractionalHeap.normalize } (\text{get-fh } H') \wedge$
 $\text{Some } H' = \text{Some } h'' \oplus \text{Some } h_{j'} \oplus \text{Some } h_f \wedge \text{safe } n \ (\text{Some } \Gamma) \ C' \ (s',$
 $h'') \{(s(x := v), [\text{edenot } E \ s \mapsto (\pi, v)], \text{gs}, \text{gu})\}$
using $\langle (s', h') = (sa(x := va), h) \rangle \langle (s, \text{FractionalHeap.normalize } (\text{get-fh } H)) = (sa, h) \rangle$ *asm0* **by** *blast*
qed
qed (*simp*)

qed (*simp*)

lemma *safe-read*:

fixes $\Delta :: ('i, 'a, \text{nat}) \text{ cont}$
assumes $\bigwedge \Gamma. \Delta = \text{Some } \Gamma \implies x \notin \text{fvA } (\text{invariant } \Gamma) \wedge \text{view-function-of-inv } \Gamma$
shows $\text{safe } n \ \Delta \ (\text{Cread } x \ E) \ (s, ([\text{edenot } E \ s \mapsto (\pi, v)], \text{gs}, \text{gu})) \ \{ (s(x := v),$
 $([\text{edenot } E \ s \mapsto (\pi, v)], \text{gs}, \text{gu})) \}$
apply (*cases* Δ)
using *safe-read-None safe-read-Some assms by blast+*

theorem *read-rule*:

fixes $\Delta :: ('i, 'a, \text{nat}) \text{ cont}$
assumes $\bigwedge \Gamma. \Delta = \text{Some } \Gamma \implies x \notin \text{fvA } (\text{invariant } \Gamma) \wedge \text{view-function-of-inv } \Gamma$
and $x \notin \text{fvE } E \cup \text{fvE } e$
shows $\text{hoare-triple-valid } \Delta \ (\text{PointsTo } E \ \pi \ e) \ (\text{Cread } x \ E) \ (\text{And } (\text{PointsTo } E \ \pi$
 $e) \ (\text{Bool } (\text{Beq } (\text{Evar } x) \ e)))$
proof (*rule hoare-triple-validI*)

define $\Sigma :: \text{store} \times ('i, 'a) \text{ heap} \Rightarrow (\text{store} \times ('i, 'a) \text{ heap}) \text{ set}$ **where**

$\Sigma = (\lambda (s, h). \{ (s(x := \text{edenot } e \ s), ([\text{edenot } E \ s \mapsto (\pi, \text{edenot } e \ s)], \text{get-gs } h,$
 $\text{get-gu } h)) \})$

show $\bigwedge s \ h \ n. (s, h), (s, h) \models \text{PointsTo } E \ \pi \ e \implies \text{safe } n \ \Delta \ (\text{Cread } x \ E) \ (s, h)$
 $(\Sigma (s, h))$

proof –

fix $s \ h \ n$

assume $(s, h :: ('i, 'a) \text{ heap}), (s, h) \models \text{PointsTo } E \ \pi \ e$

then have $\text{get-fh } h = [\text{edenot } E \ s \mapsto (\pi, \text{edenot } e \ s)]$

using *sat-points-to by blast*

then have $h = ([\text{edenot } E \ s \mapsto (\pi, \text{edenot } e \ s)], \text{get-gs } h, \text{get-gu } h)$

by (*metis decompose-heap-triple*)

then have $\text{safe } n \ \Delta \ (\text{Cread } x \ E) \ (s, ([\text{edenot } E \ s \mapsto (\pi, \text{edenot } e \ s)], \text{get-gs } h,$
 $\text{get-gu } h))$

$\{ (s(x := \text{edenot } e \ s), ([\text{edenot } E \ s \mapsto (\pi, \text{edenot } e \ s)], \text{get-gs } h, \text{get-gu } h)) \}$

using *assms safe-read by blast*

then show $\text{safe } n \ \Delta \ (\text{Cread } x \ E) \ (s, h) \ (\Sigma (s, h))$

using $\Sigma\text{-def } \langle h = ([\text{edenot } E \ s \mapsto (\pi, \text{edenot } e \ s)], \text{get-gs } h, \text{get-gu } h) \rangle$ **by auto**

qed

fix $s1 \ h1 \ s2 \ h2$

assume $(s1, h1 :: ('i, 'a) \text{ heap}), (s2, h2) \models \text{PointsTo } E \ \pi \ e$

show $\text{pair-sat } (\text{case } (s1, h1) \text{ of } (s, h) \Rightarrow \{(s(x := \text{edenot } e \ s), [\text{edenot } E \ s \mapsto$
 $(\pi, \text{edenot } e \ s)], \text{get-gs } h, \text{get-gu } h)\})$

$(\text{case } (s2, h2) \text{ of } (s, h) \Rightarrow \{(s(x := \text{edenot } e \ s), [\text{edenot } E \ s \mapsto (\pi, \text{edenot } e$
 $s)], \text{get-gs } h, \text{get-gu } h)\}) \ (\text{And } (\text{PointsTo } E \ \pi \ e) \ (\text{Bool } (\text{Beq } (\text{Evar } x) \ e)))$

proofI (*rule pair-satI*)

fix $s1' \ h1' \ s2' \ h2'$

assume $\text{asm0}: (s1', h1') \in (\text{case } (s1, h1) \text{ of } (s, h) \Rightarrow \{(s(x := \text{edenot } e \ s),$

```

[edenot E s  $\mapsto$  ( $\pi$ , edenot e s)], get-gs h, get-gu h))  $\wedge$ 
  (s2', h2')  $\in$  (case (s2, h2) of (s, h)  $\Rightarrow$  {(s(x := edenot e s), [edenot E s  $\mapsto$ 
( $\pi$ , edenot e s)], get-gs h, get-gu h)})
  then obtain s1' = s1(x := edenot e s1) h1' = ([edenot E s1  $\mapsto$  ( $\pi$ , edenot e
s1)], get-gs h1, get-gu h1)
  s2' = s2(x := edenot e s2) h2' = ([edenot E s2  $\mapsto$  ( $\pi$ , edenot e s2)], get-gs
h2, get-gu h2)
  by force
  then show (s1', h1'), (s2', h2')  $\models$  And (PointsTo E  $\pi$  e) (Bool (Beq (Evar
x) e))
  using assms(2) by auto
qed
qed

```

4.4.6 Share

lemma *share-no-abort*:

```

assumes no-abort (Some  $\Gamma$ ) C s (h :: ('i, 'a) heap)
and Some (h' :: ('i, 'a) heap) = Some h  $\oplus$  Some hj
and sat-inv s hj  $\Gamma$ 
and get-gs h = Some (pwrite, sargs)
and  $\bigwedge k$ . get-gu h k = Some (uargs k)
and reachable-value (saction  $\Gamma$ ) (uaction  $\Gamma$ ) v0 sargs uargs (view  $\Gamma$  (normalize
(get-fh hj)))
and view-function-of-inv  $\Gamma$ 
shows no-abort None C s (remove-guards h')
proof (rule no-abortI)
show  $\bigwedge H$  hf hj v0  $\Gamma$ .
  None = Some  $\Gamma$   $\wedge$ 
  Some H = Some (remove-guards h')  $\oplus$  Some hj  $\oplus$  Some hf  $\wedge$  full-ownership
(get-fh H)  $\wedge$  semi-consistent  $\Gamma$  v0 H  $\wedge$  sat-inv s hj  $\Gamma$   $\implies$ 
   $\neg$  aborts C (s, FractionalHeap.normalize (get-fh H)) by blast

```

```

fix hf H :: ('i, 'a) heap
assume asm0: Some H = Some (remove-guards h')  $\oplus$  Some hf  $\wedge$  None = None
 $\wedge$  full-ownership (get-fh H)  $\wedge$  no-guard H

```

```

have compatible h' hf
proof (rule compatibleI)
show compatible-fract-heaps (get-fh h') (get-fh hf)
by (metis asm0 compatible-def compatible-eq fst-eqD get-fh.simps option.distinct(1)
remove-guards-def)
show  $\bigwedge k$ . get-gu h' k = None  $\vee$  get-gu hf k = None
by (metis asm0 no-guard-def no-guard-then-smaller-same plus-comm)
fix p p' assume get-gs h' = Some p  $\wedge$  get-gs hf = Some p'
then show pgte pwrite (padd (fst p) (fst p'))
by (metis asm0 no-guard-def no-guard-then-smaller-same option.distinct(1)
plus-comm)
qed

```

then obtain H' **where** $\text{Some } H' = \text{Some } h' \oplus \text{Some } hf$
by *simp*
then have $\text{get-fh } H' = \text{get-fh } H$
by (*metis asm0 fst-eqD get-fh.elims option.discI remove-guards-def option.sel plus.simps(3)*)

have $\neg \text{aborts } C (s, \text{FractionalHeap.normalize } (\text{get-fh } H'))$
proof (*rule no-abortE(2)*)
show $\text{no-abort } (\text{Some } \Gamma) C s h$
using *assms* **by** *blast*
show $\text{Some } \Gamma = \text{Some } \Gamma$ **by** *blast*
show *full-ownership* ($\text{get-fh } H'$)
using $\langle \text{get-fh } H' = \text{get-fh } H \rangle \text{asm0}$ **by** *presburger*
show *semi-consistent* $\Gamma v0 H'$
proof (*rule semi-consistentI*)
show *all-guards* H'
by (*metis* $\langle \text{Some } H' = \text{Some } h' \oplus \text{Some } hf \rangle$ *all-guards-def all-guards-same assms(2) assms(4) assms(5) option.discI*)

have $\text{view } \Gamma (\text{normalize } (\text{get-fh } hj)) = \text{view } \Gamma (\text{normalize } (\text{get-fh } H'))$
using *assms(7)*
proof (*rule view-function-of-invE*)
show $H' \succeq hj$
using *larger-trans*
by (*simp add:* $\langle \text{Some } H' = \text{Some } h' \oplus \text{Some } hf \rangle$ *assms(2) larger3*)
show *sat-inv* $s hj \Gamma$
by (*simp add:* *assms(3)*)
qed

show *reachable* $\Gamma v0 H'$
proof (*rule reachableI*)
fix *sargs' uargs'*
assume *asm1:* $\text{get-gs } H' = \text{Some } (pwrite, \text{sargs}') \wedge (\forall k. \text{get-gu } H' k = \text{Some } (uargs' k))$
then have $\text{sargs} = \text{sargs}'$
by (*metis* $\langle \text{Some } H' = \text{Some } h' \oplus \text{Some } hf \rangle$ *assms(2) assms(4) full-sguard-sum-same option.inject snd-conv*)
moreover have $\text{uargs} = \text{uargs}'$
proof (*rule ext*)

fix k
show $\text{uargs } k = \text{uargs}' k$
using *full-uguard-sum-same*[*of* $h' k - H' hf$]
by (*metis* $\langle \text{Some } H' = \text{Some } h' \oplus \text{Some } hf \rangle$ *asm1 assms(2) assms(5) full-uguard-sum-same option.inject*)
qed
ultimately show *reachable-value* (*saction* Γ) (*uaction* Γ) $v0 \text{sargs}' \text{uargs}'$
(*view* $\Gamma (\text{FractionalHeap.normalize } (\text{get-fh } H'))$)
using $\langle \text{view } \Gamma (\text{FractionalHeap.normalize } (\text{get-fh } hj)) = \text{view } \Gamma (\text{FractionalHeap.normalize } (\text{get-fh } H')) \rangle$


```

(get-fh H') › assms(6) by presburger
  qed
qed
show Some H' = Some h ⊕ Some hj ⊕ Some hf
  using ⟨Some H' = Some h' ⊕ Some hf › assms(2) by presburger
show sat-inv s hj Γ
  by (simp add: assms(3))
qed

then show  $\neg$  aborts C (s, FractionalHeap.normalize (get-fh H))
  using ⟨get-fh H' = get-fh H › by auto
qed

definition S-after-share where
  S-after-share S Γ v0 = { (s, remove-guards h') | h hj h' s. semi-consistent Γ v0
h' ∧ Some h' = Some h ⊕ Some hj ∧ (s, h) ∈ S ∧ sat-inv s hj Γ }

lemma share-lemma:
  assumes safe n (Some Γ) C (s, h :: ('i, 'a) heap) S
    and Some (h' :: ('i, 'a) heap) = Some h ⊕ Some hj
    and sat-inv s hj Γ
    and semi-consistent Γ v0 h'
    and view-function-of-inv Γ
  shows safe n (None :: ('i, 'a, nat) cont) C (s, remove-guards h') (S-after-share
S Γ v0)
  using assms
proof (induct n arbitrary: C s h h' hj)
  case (Suc n)

  let ?S' = S-after-share S Γ v0

  have is-in-s': ∧h hj h'. Some h' = Some h ⊕ Some hj ∧ (s, h) ∈ S ∧ sat-inv s
hj Γ ∧ semi-consistent Γ v0 h' ⇒ (s, remove-guards h') ∈ ?S'
  proof –
    fix h hj h' assume Some h' = Some h ⊕ Some hj ∧ (s, h) ∈ S ∧ sat-inv s hj
Γ ∧ semi-consistent Γ v0 h'
    then show (s, remove-guards h') ∈ ?S'
      using S-after-share-def[of S Γ v0] mem-Collect-eq by blast
  qed
show ?case
proof (rule safeNoneI)

  show C = Cskip ⇒ (s, remove-guards h') ∈ ?S'
  proof –
    assume C = Cskip
    show (s, remove-guards h') ∈ ?S'
    proof (rule is-in-s')
      show Some h' = Some h ⊕ Some hj ∧ (s, h) ∈ S ∧ sat-inv s hj Γ ∧

```

semi-consistent Γ $v0$ h'
using *Suc.prem*s $\langle C = Cskip \rangle$ *safeSomeE*(1) *sat-inv-def* **by** *blast*
qed
qed

obtain *sargs* *uargs* **where** *get-gs* $h' = \text{Some } (pwrite, sargs) \wedge$
 $(\forall k. \text{get-gu } h' k = \text{Some } (uargs k)) \wedge \text{reachable-value } (saction \Gamma) (uaction \Gamma) v0$
sargs *uargs* (*view* Γ (*FractionalHeap.normalize* (*get-fh* h')))
by (*meson* *Suc.prem*s(4) *semi-consistentE*)
show *no-abort* *None* C s (*remove-guards* h')
proof (*rule share-no-abort*)
show *no-abort* (*Some* Γ) C s h
using *Suc.prem*s(1) *safeSomeE*(2) **by** *blast*
show *Some* $h' = \text{Some } h \oplus \text{Some } hj$
using *Suc.prem*s(2) **by** *blast*
show *sat-inv* s hj Γ
using *Suc.prem*s(3) **by** *auto*
show *get-gs* $h = \text{Some } (pwrite, sargs)$
by (*metis* *Suc.prem*s(2) $\langle \text{get-gs } h' = \text{Some } (pwrite, sargs) \wedge (\forall k. \text{get-gu } h' k = \text{Some } (uargs k)) \wedge \text{reachable-value } (saction \Gamma) (uaction \Gamma) v0 \text{ sargs } uargs \text{ (view } \Gamma \text{ (FractionalHeap.normalize } (get-fh } h')) \rangle \langle \text{sat-inv } s \text{ } hj \text{ } \Gamma \rangle \text{ no-guard-remove(1) sat-inv-def}$)
show $\bigwedge k. \text{get-gu } h k = \text{Some } (uargs k)$
by (*metis* *Suc.prem*s(2) $\langle \text{get-gs } h' = \text{Some } (pwrite, sargs) \wedge (\forall k. \text{get-gu } h' k = \text{Some } (uargs k)) \wedge \text{reachable-value } (saction \Gamma) (uaction \Gamma) v0 \text{ sargs } uargs \text{ (view } \Gamma \text{ (FractionalHeap.normalize } (get-fh } h')) \rangle \langle \text{sat-inv } s \text{ } hj \text{ } \Gamma \rangle \text{ no-guard-remove(2) sat-inv-def}$)
show *reachable-value* (*saction* Γ) (*uaction* Γ) $v0$ *sargs* *uargs* (*view* Γ (*FractionalHeap.normalize* (*get-fh* hj)))
by (*metis* *Suc.prem*s(2) *Suc.prem*s(3) $\langle \text{get-gs } h' = \text{Some } (pwrite, sargs) \wedge (\forall k. \text{get-gu } h' k = \text{Some } (uargs k)) \wedge \text{reachable-value } (saction \Gamma) (uaction \Gamma) v0 \text{ sargs } uargs \text{ (view } \Gamma \text{ (FractionalHeap.normalize } (get-fh } h')) \rangle \text{ assms(5) larger-def plus-comm view-function-of-invE}$)
show *view-function-of-inv* Γ
by (*simp* *add: assms*(5))
qed

fix H hf C' s' $h'a$
assume *asm0*: *Some* $H = \text{Some } (remove-guards } h') \oplus \text{Some } hf \wedge$
 $\text{full-ownership } (get-fh } H) \wedge \text{no-guard } H \wedge \text{red } C (s, \text{FractionalHeap.normalize } (get-fh } H)) C' (s', h'a)$

have *compatible* h' hf
proof (*rule compatibleI*)
show *compatible-fract-heaps* (*get-fh* h') (*get-fh* hf)
by (*metis* *asm0* *compatible-def* *compatible-eq* *fst-eqD* *get-fh.simps* *option.distinct*(1) *remove-guards-def*)
show $\bigwedge k. \text{get-gu } h' k = \text{None} \vee \text{get-gu } hf k = \text{None}$

by (*metis asm0 no-guard-def no-guard-then-smaller-same plus-comm*)
fix $p\ p'$ **assume** $\text{get-gs } h' = \text{Some } p \wedge \text{get-gs } hf = \text{Some } p'$
then show $\text{pgte } \text{pwrite } (\text{padd } (\text{fst } p) (\text{fst } p'))$
by (*metis asm0 no-guard-def no-guard-then-smaller-same option.distinct(1) plus-comm*)
qed
then obtain Hg **where** $\text{Some } Hg = \text{Some } h' \oplus \text{Some } hf$
by *simp*
then have $\text{get-fh } Hg = \text{get-fh } H$
by (*metis asm0 fst-eqD get-fh.elims option.discI remove-guards-def option.sel plus.simps(3)*)

have $\exists h''\ H'\ hj'$.
full-ownership ($\text{get-fh } H'$) \wedge
semi-consistent $\Gamma\ v0\ H'$ \wedge
sat-inv $s'\ hj'\ \Gamma \wedge h'a = \text{FractionalHeap.normalize } (\text{get-fh } H') \wedge \text{Some } H' =$
 $\text{Some } h'' \oplus \text{Some } hj' \oplus \text{Some } hf \wedge \text{safe } n (\text{Some } \Gamma)\ C' (s', h'')\ S$
using *Suc(2)*
proof (*rule safeSomeE(3)[of n $\Gamma\ C\ s\ h\ S\ Hg\ hj\ hf\ v0\ C'\ s'\ h'a$]*)
show $\text{Some } Hg = \text{Some } h \oplus \text{Some } hj \oplus \text{Some } hf$
by (*simp add: Suc.prem(2) $\langle \text{Some } Hg = \text{Some } h' \oplus \text{Some } hf \rangle$*)
show *full-ownership* ($\text{get-fh } Hg$)
using $\langle \text{get-fh } Hg = \text{get-fh } H \rangle$ *asm0* **by** *presburger*
show *sat-inv* $s\ hj\ \Gamma$
by (*simp add: Suc.prem(3)*)
show *red* $C (s, \text{FractionalHeap.normalize } (\text{get-fh } Hg))\ C' (s', h'a)$
using $\langle \text{get-fh } Hg = \text{get-fh } H \rangle$ *asm0* **by** *presburger*
show *semi-consistent* $\Gamma\ v0\ Hg$
proof (*rule semi-consistentI*)
show *all-guards* Hg
by (*meson Suc.prem(4) $\langle \text{Some } Hg = \text{Some } h' \oplus \text{Some } hf \rangle$ all-guards-same semi-consistent-def*)
have *view* $\Gamma (\text{normalize } (\text{get-fh } hj)) = \text{view } \Gamma (\text{normalize } (\text{get-fh } Hg))$
using *assms(5)*
proof (*rule view-function-of-invE*)
show $Hg \succeq hj$
using *larger-trans*
using $\langle \text{Some } Hg = \text{Some } h \oplus \text{Some } hj \oplus \text{Some } hf \rangle$ *larger3* **by** *blast*
show *sat-inv* $s\ hj\ \Gamma$
by (*simp add: $\langle \text{sat-inv } s\ hj\ \Gamma \rangle$*)
qed
show *reachable* $\Gamma\ v0\ Hg$
proof (*rule reachableI*)
fix $\text{sargs}'\ \text{uargs}'$
assume *asm1*: $\text{get-gs } Hg = \text{Some } (\text{pwrite}, \text{sargs}') \wedge (\forall k. \text{get-gu } Hg\ k =$
 $\text{Some } (\text{uargs}'\ k))$
then have $\text{sargs} = \text{sargs}'$
by (*metis Pair-inject $\langle \text{Some } Hg = \text{Some } h' \oplus \text{Some } hf \rangle \langle \text{get-gs } h' = \text{Some } (\text{pwrite}, \text{sargs}) \wedge (\forall k. \text{get-gu } h'\ k = \text{Some } (\text{uargs}'\ k)) \wedge \text{reachable-value}$*)

(saction Γ) (uaction Γ) v0 sargs uargs (view Γ (FractionalHeap.normalize (get-fh h'))) › *full-sguard-sum-same option.inject*
moreover have *uargs = uargs'*
proof (*rule ext*)
fix *k*
show *uargs k = uargs' k*
by (*metis* ‹*Some Hg = Some h' \oplus Some hf*› ‹*get-gs h' = Some (pwrite, sargs) \wedge ($\forall k. \text{get-gu } h' k = \text{Some } (uargs k)$) \wedge reachable-value (saction Γ) (uaction Γ) v0 sargs uargs (view Γ (FractionalHeap.normalize (get-fh h')))*›› *asm1 full-uguard-sum-same option.inject*)
qed
ultimately show *reachable-value (saction Γ) (uaction Γ) v0 sargs' uargs' (view Γ (FractionalHeap.normalize (get-fh Hg)))*
by (*metis* *Suc.prem5(2)* ‹*get-gs h' = Some (pwrite, sargs) \wedge ($\forall k. \text{get-gu } h' k = \text{Some } (uargs k)$) \wedge reachable-value (saction Γ) (uaction Γ) v0 sargs uargs (view Γ (FractionalHeap.normalize (get-fh h')))*›› ‹*sat-inv s hj Γ ›› ‹*view Γ (FractionalHeap.normalize (get-fh hj)) = view Γ (FractionalHeap.normalize (get-fh Hg))*›› *assms(5) larger-def plus-comm view-function-of-invE*)
qed
qed
qed
then obtain *h'' H' hj' where asm1: full-ownership (get-fh H') \wedge semi-consistent Γ v0 H' \wedge sat-inv s' hj' Γ \wedge h'a = FractionalHeap.normalize (get-fh H') \wedge Some H' = Some h'' \oplus Some hj' \oplus Some hf \wedge safe n (Some Γ) C' (s', h'') S*
by *blast*
obtain *hj'' where Some hj'' = Some h'' \oplus Some hj'*
by (*metis* *asm1 not-Some-eq plus.simps(1)*)
moreover obtain *sargs' uargs' where new-guards-def: get-gs H' = Some (pwrite, sargs') \wedge ($\forall k. \text{get-gu } H' k = \text{Some } (uargs' k)$) \wedge reachable-value (saction Γ) (uaction Γ) v0 sargs' uargs' (view Γ (FractionalHeap.normalize (get-fh H')))*
by (*meson* *asm1 semi-consistentE*)

have *safe n (None :: ('i, 'a, nat) cont) C' (s', remove-guards hj'') ?S'*
proof (*rule* *Suc(1)* [of *C' s' h'' hj'' hj'*])

show *safe n (Some Γ) C' (s', h'') S*
using *asm1 by blast*
show *Some hj'' = Some h'' \oplus Some hj'*
using ‹*Some hj'' = Some h'' \oplus Some hj'*› **by** *blast*
show *sat-inv s' hj' Γ*
using *asm1 by fastforce*

have *no-guard hf*
by (*metis* *asm0 no-guard-then-smaller-same plus-comm*)
moreover have *no-guard hj'*
using ‹*sat-inv s' hj' Γ ›› *sat-inv-def by blast***

```

have view  $\Gamma$  (normalize (get-fh hj')) = view  $\Gamma$  (normalize (get-fh H'))
  using assms(5)
proof (rule view-function-of-invE)
  show  $H' \succeq hj'$ 
    using larger-trans
    using asm1 larger3 by blast
  show sat-inv s' hj'  $\Gamma$ 
    by (simp add: asm1)
qed

  obtain uargs' sargs' where args': get-gs H' = Some (pwrite, sargs')  $\wedge$  ( $\forall k$ .
  get-gu H' k = Some (uargs' k))  $\wedge$  reachable-value (saction  $\Gamma$ ) (uaction  $\Gamma$ ) v0 sargs'
  uargs'
  (view  $\Gamma$  (FractionalHeap.normalize
  (get-fh H')))
    using semi-consistentE[of  $\Gamma$  v0 H'] asm1
    by blast
  then have get-gs hj'' = Some (pwrite, sargs')  $\wedge$  ( $\forall k$ . get-gu hj'' k = Some
  (uargs' k))
    by (metis  $\langle$ Some hj'' = Some h''  $\oplus$  Some hj'  $\rangle$  asm1 calculation no-guard-remove(1)
  no-guard-remove(2))

  show semi-consistent  $\Gamma$  v0 hj''
  proof (rule semi-consistentI)

    show all-guards hj''
    by (metis  $\langle$ get-gs hj'' = Some (pwrite, sargs')  $\wedge$  ( $\forall k$ . get-gu hj'' k = Some
  (uargs' k))  $\rangle$  all-guards-def option.discI)
    have view  $\Gamma$  (FractionalHeap.normalize (get-fh H')) = view  $\Gamma$  (FractionalHeap.normalize
  (get-fh hj''))
    by (metis  $\langle$ Some hj'' = Some h''  $\oplus$  Some hj'  $\rangle$   $\langle$ view  $\Gamma$  (FractionalHeap.normalize
  (get-fh hj'')) = view  $\Gamma$  (FractionalHeap.normalize (get-fh H'))  $\rangle$  asm1 assms(5)
  larger-def plus-comm view-function-of-invE)
    then show reachable  $\Gamma$  v0 hj''
      by (metis  $\langle$ get-gs hj'' = Some (pwrite, sargs')  $\wedge$  ( $\forall k$ . get-gu hj'' k
  = Some (uargs' k))  $\rangle$  args' asm1 ext get-fh.simps new-guards-def option.sel reach-
  able-def snd-conv)
    qed
    show view-function-of-inv  $\Gamma$ 
      by (simp add: assms(5))
    qed

  let ?h'' = remove-guards hj''
  have hj'' ## hf
    by (metis asm1 calculation option.simps(3) plus.simps(3))
  then obtain H'' where Some H'' = Some ?h''  $\oplus$  Some hf
    by (simp add: remove-guards-smaller smaller-more-compatible)

```

then have $\text{get-fh } H'' = \text{get-fh } H'$
by (*metis asm1 calculation equiv-sum-get-fh get-fh-remove-guards*)
moreover have $\text{no-guard } H''$
by (*metis* $\langle \text{Some } H'' = \text{Some } (\text{remove-guards } hj'') \oplus \text{Some } hf \rangle \text{ asm0 no-guard-remove-guards}$
no-guard-then-smaller-same plus-comm sum-of-no-guards)

ultimately show $\exists h'' H'$.
 $\text{full-ownership } (\text{get-fh } H') \wedge$
 $\text{no-guard } H' \wedge h'a = \text{FractionalHeap.normalize } (\text{get-fh } H') \wedge \text{Some } H' =$
 $\text{Some } h'' \oplus \text{Some } hf \wedge \text{safe } n \text{ (None :: ('i, 'a, nat) cont) } C' (s', h'') ?S'$
by (*metis* $\langle \text{Some } H'' = \text{Some } (\text{remove-guards } hj'') \oplus \text{Some } hf \rangle \langle \text{safe } n \text{ None}$
 $C' (s', \text{remove-guards } hj'') ?S' \rangle \text{ asm1}$)
qed
qed (*simp*)

definition *no-need-guards where*

$\text{no-need-guards } A \longleftrightarrow (\forall s1 h1 s2 h2. (s1, h1), (s2, h2) \models A \longrightarrow (s1, \text{remove-guards } h1), (s2, \text{remove-guards } h2) \models A)$

lemma *has-guard-then-safe-none:*

assumes $\neg \text{no-guard } h$
and $C = \text{Cskip} \implies (s, h) \in S$
shows $\text{safe } n \text{ (None :: ('i, 'a, nat) cont) } C (s, h) S$
proof (*induct n*)
case (*Suc n*)
show *?case*
proof (*rule safeNoneI*)
show $C = \text{Cskip} \implies (s, h) \in S$
by (*simp add: assms(2)*)
show $\text{no-abort None } C s h$
using *assms(1) no-abortNoneI no-guard-then-smaller-same* **by** *blast*
show $\bigwedge H hf C' s' h'. \text{Some } H = \text{Some } h \oplus \text{Some } hf \wedge \text{full-ownership } (\text{get-fh } H) \wedge \text{no-guard } H \wedge$
 $\text{red } C (s, \text{FractionalHeap.normalize } (\text{get-fh } H)) C' (s', h') \implies$
 $\exists h'' H'. \text{full-ownership } (\text{get-fh } H') \wedge \text{no-guard } H' \wedge h' = \text{FractionalHeap.normalize}$
 $(\text{get-fh } H') \wedge \text{Some } H' = \text{Some } h'' \oplus \text{Some } hf \wedge \text{safe } n \text{ None } C' (s', h'') S$
using *assms(1) no-guard-then-smaller-same* **by** *blast*
qed
qed (*simp*)

theorem *share-rule:*

fixes $\Gamma :: ('i, 'a, nat) \text{ single-context}$
assumes $\Gamma = \langle \text{view} = f, \text{abstract-view} = \alpha, \text{saction} = \text{sact}, \text{uaction} = \text{uact},$
 $\text{invariant} = J \rangle$
and *all-axioms* $\alpha \text{ sact spre uact upre}$

and *hoare-triple-valid* (Some Γ) (Star P *EmptyFullGuards*) C (Star Q (And
(PreSharedGuards (Abs-precondition spre)) (PreUniqueGuards (Abs-indexed-precondition
upre))))
and *view-function-of-inv* Γ
and *unary* $J \wedge$ *precise* J
and *wf-indexed-precondition* upre \wedge *wf-precondition* spre
and $x \notin$ *fvA* J
and *no-guard-assertion* (Star P (LowView ($\alpha \circ f$) J x))
shows *hoare-triple-valid* (None :: ('i, 'a, nat) cont) (Star P (LowView ($\alpha \circ f$)
 J x)) C (Star Q (LowView ($\alpha \circ f$) J x))
proof –
let $?P =$ Star P *EmptyFullGuards*
let $?Q =$ Star Q (And (PreSharedGuards (Abs-precondition spre)) (PreUniqueGuards
(Abs-indexed-precondition upre)))

obtain Σ **where** *asm0*: $\bigwedge \sigma$ n. $\sigma, \sigma \models$ Star P *EmptyFullGuards* \implies *safe* n (Some
 Γ) C σ (Σ σ)
 $\bigwedge \sigma$ σ' . $\sigma, \sigma' \models$ Star P *EmptyFullGuards* \implies *pair-sat* (Σ σ) (Σ σ') (Star Q (And
(PreSharedGuards (Abs-precondition spre)) (PreUniqueGuards (Abs-indexed-precondition
upre))))
using *hoare-triple-validE*[of Some Γ $?P$ C $?Q$] *assms*(3) **by** *blast*

Steps: 1) Remove the *hj* and add *empty-guards* 2) Apply *sigma* 3) Remove the guards and add *hj*, using *S-after-share*

define *input- Σ* **where** *input- Σ* = ($\lambda \sigma$. { (fst σ , *add-empty-guards* hp) | hp hj .
Some (snd σ) = Some $hp \oplus$ Some $hj \wedge$
(fst σ , hp), (fst σ , hp) \models $P \wedge$ *sat-inv* (fst σ) hj Γ })

define Σ' **where** $\Sigma' = (\lambda \sigma$. $\bigcup p \in$ *input- Σ* σ . *S-after-share* (Σ p) Γ (*f* (*normalize*
(*get-fh* (snd σ)))))

show *?thesis*
proof (*rule hoare-triple-validI*)
show $\bigwedge s$ h n. (s , h), (s , h) \models Star P (LowView ($\alpha \circ f$) J x) \implies *safe* n (None
:: ('i, 'a, nat) cont) C (s , h) (Σ' (s , h))
proof –
fix s h n **assume** *asm1*: (s , h), (s , h) \models Star P (LowView ($\alpha \circ f$) J x)
then obtain hp hj **where** *no-guard* h Some $h =$ Some $hp \oplus$ Some hj (s , hp),
(s , hp) \models P
(s , hj), (s , hj) \models LowView ($\alpha \circ f$) J x
by (*meson always-sat-refl* *assms*(8) *hyper-sat.simps*(4) *no-guard-assertion-def*)
then have *sat-inv* s hj Γ
by (*metis LowViewE* *assms*(1) *assms*(7) *no-guard-then-smaller-same plus-comm*
sat-inv-def select-convs(5))
then have (s , *add-empty-guards* hp) \in *input- Σ* (s , h)
using $\langle (s, hp), (s, hp) \models P \rangle$ \langle Some $h =$ Some $hp \oplus$ Some $hj \rangle$ *input- Σ -def*
by *force*

let $?v0 =$ *f* (*normalize* (*get-fh* h))

```

let ?p = (s, add-empty-guards hp)

have safe n (None :: ('i, 'a, nat) cont) C (s, remove-guards (add-empty-guards
h)) (S-after-share (Σ ?p) Γ ?v0)
  proof (rule share-lemma)
    show safe n (Some Γ) C ?p (Σ ?p)
    proof (rule asm0(1))
      show (s, add-empty-guards hp), (s, add-empty-guards hp) ⊨ Star P
EmptyFullGuards
      using ⟨(s, hp), (s, hp) ⊨ P⟩ ⟨Some h = Some hp ⊕ Some hj⟩ ⟨no-guard
h⟩ no-guard-and-sat-p-empty-guards no-guard-then-smaller-same by blast
    qed
    show Some (add-empty-guards h) = Some (add-empty-guards hp) ⊕ Some
hj
  using ⟨Some h = Some hp ⊕ Some hj⟩ ⟨no-guard h⟩ no-guard-add-empty-guards-sum
by blast
  show sat-inv s hj Γ
  using ⟨sat-inv s hj Γ⟩ by auto
  show view-function-of-inv Γ
  by (simp add: assms(4))
show semi-consistent Γ (f (FractionalHeap.normalize (get-fh h))) (add-empty-guards
h)
  by (metis ⟨no-guard h⟩ assms(1) select-convs(1) semi-consistent-empty-no-guard-initial-value)
  qed
moreover have (S-after-share (Σ ?p) Γ ?v0) ⊆ Σ' (s, h)
  using Σ'-def ⟨(s, add-empty-guards hp) ∈ input-Σ (s, h)⟩ by auto
ultimately show safe n (None :: ('i, 'a, nat) cont) C (s, h) (Σ' (s, h))
  by (metis ⟨no-guard h⟩ no-guards-remove-same safe-larger-set)
  qed

fix s1 h1 s2 h2
assume (s1, h1), (s2, h2) ⊨ Star P (LowView (α ∘ f) J x)
then obtain hp1 hj1 hp2 hj2 where asm1: Some h1 = Some hp1 ⊕ Some hj1
  Some h2 = Some hp2 ⊕ Some hj2 (s1, hp1), (s2, hp2) ⊨ P no-guard h1
no-guard h2
  (s1, hj1), (s2, hj2) ⊨ LowView (α ∘ f) J x
  using assms(8) hyper-sat.simps(4) no-guard-assertion-def by blast
  then obtain (s1, hj1), (s2, hj2) ⊨ J α (f (normalize (get-fh hj1))) = α (f
(normalize (get-fh hj2)))
  by (metis LowViewE assms(7) comp-apply)

show pair-sat (Σ' (s1, h1)) (Σ' (s2, h2)) (Star Q (LowView (α ∘ f) J x))
proof (rule pair-satI)
  fix s1' h1' s2' h2'
  assume asm2: (s1', h1') ∈ Σ' (s1, h1) ∧ (s2', h2') ∈ Σ' (s2, h2)
  then obtain p1 p2 where p-assms: p1 ∈ input-Σ (s1, h1) p2 ∈ input-Σ (s2,
h2)

```


$(s1', h1') \in S\text{-after-share } (\Sigma p1) \Gamma (f (\text{normalize } (\text{get-fh } h1)))$
 $(s2', h2') \in S\text{-after-share } (\Sigma p2) \Gamma (f (\text{normalize } (\text{get-fh } h2)))$
using Σ' -def **by force**
moreover have pair-sat $(\Sigma p1) (\Sigma p2) (Star Q (And (PreSharedGuards (Abs-precondition spre)) (PreUniqueGuards (Abs-indexed-precondition upre))))$
proof (rule asm0(2))
obtain $hj1' hj2' hp1' hp2'$ **where** $snd p1 = \text{add-empty-guards } hp1' \text{ snd } p2$
 $= \text{add-empty-guards } hp2'$
 $Some h1 = Some hp1' \oplus Some hj1' \text{ Some } h2 = Some hp2' \oplus Some hj2'$
 $sat\text{-inv } s1 \ hj1' \Gamma \text{ sat}\text{-inv } s2 \ hj2' \Gamma$
 $fst p1 = s1 \text{ fst } p2 = s2$
using p-assms(1) p-assms(2) input- Σ -def **by auto**
moreover have $hj1 = hj1' \wedge hj2 = hj2'$
proof (rule preciseE)
show precise J
by (simp add: assms(5))
show $h1 \succeq hj1' \wedge h1 \succeq hj1 \wedge h2 \succeq hj2' \wedge h2 \succeq hj2$
by (metis asm1(1) asm1(2) calculation(3) calculation(4) larger-def plus-comm)
show $(s1, hj1'), (s2, hj2') \models J \wedge (s1, hj1), (s2, hj2) \models J$
by (metis $\langle (s1, hj1), (s2, hj2) \models J \rangle$ assms(1) assms(5) calculation(5) calculation(6) sat-inv-def select-convs(5) unaryE)
qed
then have $hp1 = hp1' \wedge hp2 = hp2'$
using addition-cancellative asm1(1) asm1(2) calculation(3) calculation(4)
by blast
then show $p1, p2 \models Star P \text{ EmptyFullGuards}$
using no-guard-and-sat-p-empty-guards[of fst p1 snd p1 fst p2 snd p2 P]
by (metis asm1(3) asm1(4) asm1(5) calculation(1) calculation(2) calculation(3) calculation(4) calculation(7) calculation(8) no-guard-and-sat-p-empty-guards no-guard-then-smaller-same prod.exhaust-sel)
qed

let $?v1 = f (\text{normalize } (\text{get-fh } h1))$
let $?v2 = f (\text{normalize } (\text{get-fh } h2))$

obtain $hj1' hg1 H1 hj2' hg2 H2$ **where** $asm3: h1' = \text{remove-guards } H1$
 $semi\text{-consistent } \Gamma \ ?v1 \ H1$
 $Some H1 = Some hg1 \oplus Some hj1' (s1', hg1) \in \Sigma p1 \text{ sat}\text{-inv } s1' \ hj1' \Gamma$
 $h2' = \text{remove-guards } H2 \text{ semi}\text{-consistent } \Gamma \ ?v2 \ H2$
 $Some H2 = Some hg2 \oplus Some hj2' (s2', hg2) \in \Sigma p2 \text{ sat}\text{-inv } s2' \ hj2' \Gamma$
using p-assms(3) S-after-share-def[of $\Sigma p1 \Gamma \ ?v1$] p-assms(4) S-after-share-def[of $\Sigma p2 \Gamma \ ?v2$] **by blast**

then have $(s1', hg1), (s2', hg2) \models Star Q (And (PreSharedGuards (Abs-precondition spre)) (PreUniqueGuards (Abs-indexed-precondition upre)))$
using $\langle \text{pair-sat } (\Sigma p1) (\Sigma p2) (Star Q (And (PreSharedGuards (Abs-precondition spre)) (PreUniqueGuards (Abs-indexed-precondition upre)))) \rangle$ pair-satE **by blast**

then obtain $q1\ g1\ q2\ g2$ **where** $\text{Some } hg1 = \text{Some } q1 \oplus \text{Some } g1\ \text{Some } hg2$
 $= \text{Some } q2 \oplus \text{Some } g2$
 $(s1', q1), (s2', q2) \models Q\ (s1', g1), (s2', g2) \models \text{PreSharedGuards}\ (\text{Abs-precondition}\ \text{spre})$
 $(s1', g1), (s2', g2) \models \text{PreUniqueGuards}\ (\text{Abs-indexed-precondition}\ \text{upre})$
by $(\text{meson}\ \text{hyper-sat.simps}(3)\ \text{hyper-sat.simps}(4))$
moreover have $\text{Rep-precondition}\ (\text{Abs-precondition}\ \text{spre}) = \text{spre} \wedge \text{Rep-indexed-precondition}\ (\text{Abs-indexed-precondition}\ \text{upre}) = \text{upre}$
by $(\text{simp}\ \text{add:}\ \text{assms}(6)\ \text{wf-indexed-precondition-rep-prec}\ \text{wf-precondition-rep-prec})$
ultimately obtain $\text{sargs1}\ \text{sargs2}$ **where**
 $\text{get-gs}\ g1 = \text{Some}\ (\text{pwrite}, \text{sargs1})\ \text{get-gs}\ g2 = \text{Some}\ (\text{pwrite}, \text{sargs2})$
 $\text{PRE-shared-simpler}\ \text{spre}\ \text{sargs1}\ \text{sargs2}$
 $\text{get-fh}\ g1 = \text{Map.empty}\ \text{get-fh}\ g2 = \text{Map.empty}$
by auto
moreover obtain $\text{uargs1}\ \text{uargs2}$ **where**
 $\text{unique-facts: } \bigwedge k. \text{get-gu}\ g1\ k = \text{Some}\ (\text{uargs1}\ k) \wedge \text{get-gu}\ g2\ k = \text{Some}\ (\text{uargs2}\ k) \wedge \text{PRE-unique}\ (\text{upre}\ k)\ (\text{uargs1}\ k)\ (\text{uargs2}\ k)$
using $\text{sat-PreUniqueE}[OF\ \langle (s1', g1), (s2', g2) \models \text{PreUniqueGuards}\ (\text{Abs-indexed-precondition}\ \text{upre}) \rangle]$
by $(\text{metis}\ \langle \text{Rep-precondition}\ (\text{Abs-precondition}\ \text{spre}) = \text{spre} \wedge \text{Rep-indexed-precondition}\ (\text{Abs-indexed-precondition}\ \text{upre}) = \text{upre} \rangle)$
moreover obtain $\text{get-gs}\ H1 = \text{Some}\ (\text{pwrite}, \text{sargs1}) \bigwedge k. \text{get-gu}\ H1\ k = \text{Some}\ (\text{uargs1}\ k)$
by $(\text{metis}\ (\text{no-types}, \text{opaque-lifting})\ \langle \text{Some } hg1 = \text{Some } q1 \oplus \text{Some } g1 \rangle\ \text{asm3}(3)\ \text{calculation}(1)\ \text{calculation}(6)\ \text{full-sguard-sum-same}\ \text{full-uguard-sum-same}\ \text{plus-comm})$
then have $\text{reach1:}\ \text{reachable-value}\ \text{sact}\ \text{uact}\ ?v1\ \text{sargs1}\ \text{uargs1}\ (f\ (\text{normalize}\ (\text{get-fh}\ H1)))$
by $(\text{metis}\ \text{asm3}(2)\ \text{assms}(1)\ \text{reachableE}\ \text{select-convs}(1)\ \text{select-convs}(3)\ \text{select-convs}(4)\ \text{semi-consistent-def})$
moreover obtain $\text{get-gs}\ H2 = \text{Some}\ (\text{pwrite}, \text{sargs2}) \bigwedge k. \text{get-gu}\ H2\ k = \text{Some}\ (\text{uargs2}\ k)$
by $(\text{metis}\ (\text{no-types}, \text{lifting})\ \langle \text{Some } hg2 = \text{Some } q2 \oplus \text{Some } g2 \rangle\ \text{asm3}(8)\ \text{calculation}(2)\ \text{calculation}(6)\ \text{full-sguard-sum-same}\ \text{full-uguard-sum-same}\ \text{plus-comm})$
then have $\text{reach2:}\ \text{reachable-value}\ \text{sact}\ \text{uact}\ ?v2\ \text{sargs2}\ \text{uargs2}\ (f\ (\text{normalize}\ (\text{get-fh}\ H2)))$
by $(\text{metis}\ \text{asm3}(7)\ \text{assms}(1)\ \text{reachableE}\ \text{semi-consistent-def}\ \text{simps}(1)\ \text{simps}(3)\ \text{simps}(4))$
moreover have $\alpha\ (f\ (\text{normalize}\ (\text{get-fh}\ h1))) = \alpha\ (f\ (\text{normalize}\ (\text{get-fh}\ hj1)))$
using $\text{view-function-of-invE}[of\ \Gamma\ s1\ hj1\ h1]$
by $(\text{metis}\ \langle (s1, hj1), (s2, hj2) \models J \rangle\ \text{always-sat-refl}\ \text{asm1}(1)\ \text{asm1}(4)\ \text{assms}(1)\ \text{assms}(4)\ \text{larger-def}\ \text{no-guard-then-smaller-same}\ \text{plus-comm}\ \text{sat-inv-def}\ \text{select-convs}(1)\ \text{select-convs}(5))$
moreover have $\alpha\ (f\ (\text{normalize}\ (\text{get-fh}\ h2))) = \alpha\ (f\ (\text{normalize}\ (\text{get-fh}\ hj2)))$
using $\text{view-function-of-invE}[of\ \Gamma\ s2\ hj2\ h2]$
by $(\text{metis}\ \langle (s1, hj1), (s2, hj2) \models J \rangle\ \text{always-sat-refl}\ \text{asm1}(2)\ \text{asm1}(5)\ \text{assms}(1)\ \text{assms}(4)\ \text{larger-def}\ \text{no-guard-then-smaller-same}\ \text{plus-comm}\ \text{sat-comm}\ \text{sat-inv-def}\ \text{select-convs}(1)\ \text{select-convs}(5))$
ultimately have $\text{low-abstract-view: } \alpha\ (f\ (\text{FractionalHeap.normalize}\ (\text{get-fh}\ H1))) = \alpha\ (f\ (\text{FractionalHeap.normalize}\ (\text{get-fh}\ H2)))$

using *reach1 reach2 main-result*[of *sact uact ?v1 sargs1 uargs1 f (normalize (get-fh H1)) ?v2 sargs2 uargs2 f (normalize (get-fh H2)) spre upre α*]
using $\langle \alpha (f (FractionalHeap.normalize (get-fh hj1))) = \alpha (f (FractionalHeap.normalize (get-fh hj2))) \rangle$ *assms(2)* **by** *presburger*
moreover have $\alpha (f (normalize (get-fh H1))) = \alpha (f (normalize (get-fh hj1)))$
using *view-function-of-invE*[of $\Gamma s1' hj1' H1$]
by (*metis asm3(3) asm3(5) assms(1) assms(4) larger-def plus-comm select-convs(1)*)
moreover have $\alpha (f (normalize (get-fh H2))) = \alpha (f (normalize (get-fh hj2)))$
using *view-function-of-invE*[of $\Gamma s2' hj2' H2$]
by (*metis asm3(10) asm3(8) assms(1) assms(4) larger-def plus-comm select-convs(1)*)
moreover have $(s1', hj1'), (s2', hj2') \models J$
by (*metis asm3(10) asm3(5) assms(1) assms(5) sat-inv-def select-convs(5) unaryE*)
ultimately have $(s1', hj1'), (s2', hj2') \models LowView (\alpha \circ f) J x$
by (*simp add: LowViewI assms(7)*)
moreover have *Some h1' = Some q1 \oplus Some hj1'*
proof –
have *Some h1' = Some (remove-guards hg1) \oplus Some (remove-guards hj1')*
using *asm3(1) asm3(3) remove-guards-sum* **by** *blast*
moreover have *remove-guards hg1 = remove-guards q1*
by (*metis $\langle Some hg1 = Some q1 \oplus Some g1 \rangle \langle get-fh g1 = Map.empty \rangle$*
get-fh-remove-guards no-guard-and-no-heap no-guard-remove-guards no-guards-remove
remove-guards-sum)
moreover have *remove-guards hj1' = hj1'*
by (*metis asm3(5) no-guards-remove sat-inv-def*)
ultimately show *?thesis*
by (*metis $\langle Some hg1 = Some q1 \oplus Some g1 \rangle \langle get-gs g1 = Some (pwrite,$*
sargs1) \rangle unique-facts all-guards-def full-guard-comp-then-no no-guards-remove op-
tion.distinct(1) plus.simps(3) plus-comm)
qed
moreover have *Some h2' = Some q2 \oplus Some hj2'*
proof –
have *Some h2' = Some (remove-guards hg2) \oplus Some (remove-guards hj2')*
using *asm3(6) asm3(8) remove-guards-sum* **by** *blast*
moreover have *remove-guards hg2 = remove-guards q2*
by (*metis $\langle Some hg2 = Some q2 \oplus Some g2 \rangle \langle get-fh g2 = Map.empty \rangle$*
get-fh-remove-guards no-guard-and-no-heap no-guard-remove-guards no-guards-remove
remove-guards-sum)
moreover have *remove-guards hj2' = hj2'*
by (*metis asm3(10) no-guards-remove sat-inv-def*)
ultimately show *?thesis*
by (*metis $\langle Some hg2 = Some q2 \oplus Some g2 \rangle \langle get-gs g2 = Some (pwrite,$*
sargs2) \rangle unique-facts all-guards-def full-guard-comp-then-no no-guards-remove op-
tion.distinct(1) plus.simps(3) plus-comm)
qed

ultimately show $(s1', h1'), (s2', h2') \models Star\ Q\ (LowView\ (\alpha \circ f)\ J\ x)$
by $(meson\ LowViewI\ \langle (s1', q1), (s2', q2) \models Q \rangle\ assms(7)\ hyper-sat.simps(9))$
hyper-sat.simps(4)
qed
qed
qed

4.4.7 Atomic

lemma *red-rtrans-induct*:

assumes *red-rtrans* $C\ \sigma\ C'\ \sigma'$
and $\bigwedge C\ \sigma. P\ C\ \sigma\ C\ \sigma$
and $\bigwedge C\ \sigma\ C'\ \sigma'\ C''\ \sigma''. red\ C\ \sigma\ C'\ \sigma' \implies red-rtrans\ C'\ \sigma'\ C''\ \sigma'' \implies P\ C'\ \sigma'\ C''\ \sigma'' \implies P\ C\ \sigma\ C''\ \sigma''$
shows $P\ C\ \sigma\ C'\ \sigma'$
using *red-red-rtrans.inducts*[of - - - λ - - - . *True P*] **assms** **by** *auto*

lemma *safe-atomic*:

assumes *red-rtrans* $C1\ \sigma1\ C2\ \sigma2$
and $\sigma1 = (s1, H1)$
and $\sigma2 = (s2, H2)$
and $\bigwedge n. safe\ n\ (None :: ('i, 'a, nat)\ cont)\ C1\ (s1, h)\ S$
and $H = denormalize\ H1$
and $Some\ H = Some\ h \oplus Some\ hf$
and *full-ownership* $(get-fh\ H) \wedge no-guard\ H$
shows $\neg aborts\ C2\ \sigma2 \wedge (C2 = Cskip \implies (\exists h1\ H'. Some\ H' = Some\ h1 \oplus Some\ hf \wedge H2 = normalize\ (get-fh\ (H')) \wedge no-guard\ H' \wedge full-ownership\ (get-fh\ H') \wedge (s2, h1) \in S))$
using *assms*
proof (*induction arbitrary: s1 H1 H h rule: red-rtrans-induct*[of $C1\ \sigma1\ C2\ \sigma2$])
case 1
then show *?case* **by** (*simp add: assms(1)*)
next
case $(2\ C\ \sigma)$
then have $\neg aborts\ C\ (s1, FractionalHeap.normalize\ (get-fh\ H))$
using *no-abortE(1) safe.simps(2)* **by** *blast*
then have $\neg aborts\ C\ \sigma$
by (*metis 2.prem(2) 2.prem(5) denormalize-properties(3)*)
moreover have *safe* $(Suc\ 1)\ (None :: ('i, 'a, nat)\ cont)\ C\ (s1, h)\ S$
using *2.prem(4)* **by** *blast*
then have $C = Cskip \implies (s2, h) \in S$
by (*metis 2.prem(2) 2.prem(3) Pair-inject safeNoneE(1)*)
then have $C = Cskip \implies Some\ H = Some\ h \oplus Some\ hf \wedge H2 = FractionalHeap.normalize\ (get-fh\ H) \wedge no-guard\ H \wedge full-ownership\ (get-fh\ H) \wedge (s2, h) \in S$
by (*metis 2.prem(3) 2.prem(2) 2.prem(6) 2.prem(5) 2.prem(7) denormalize-properties(3) old.prod.inject*)
ultimately show *?case*
by *blast*

next
case ($\exists C \sigma C' \sigma' C'' \sigma''$)
obtain $s0 \ H0$ **where** $\sigma' = (s0, H0)$ **using** *prod.exhaust-sel* **by** *blast*

have *safe* (*Suc* 0) (*None* :: ('i, 'a, nat) cont) C (s1, h) S
using 3.prem(4) **by** *force*
then have $\exists h'' H'. \text{full-ownership } (get\text{-fh } H') \wedge \text{no-guard } H' \wedge H0 = \text{FractionalHeap.normalize } (get\text{-fh } H') \wedge \text{Some } H' = \text{Some } h'' \oplus \text{Some } hf \wedge \text{safe } 0$ (*None* :: ('i, 'a, nat) cont) C' (s0, h'') S
proof (*rule safeNoneE*(3)[of 0 C s1 h S H hf C' s0 H0])
show $\text{Some } H = \text{Some } h \oplus \text{Some } hf$ **using** 3.prem(6) **by** *blast*
show *full-ownership* (*get-fh* H) **using** 3.prem(7) **by** *blast*
show *no-guard* H **using** 3.prem(7) **by** *auto*
show *red* C (s1, *FractionalHeap.normalize* (*get-fh* H)) C' (s0, H0)
by (*metis* 3.hyps(1) 3.prem(2) 3.prem(5) $\langle \sigma' = (s0, H0) \rangle$ *denormalize-properties*(3))

qed
then obtain $h0 \ H0'$ **where**
 $r1: \text{full-ownership } (get\text{-fh } H0') \wedge \text{no-guard } H0' \wedge H0 = \text{FractionalHeap.normalize } (get\text{-fh } H0') \wedge \text{Some } H0' = \text{Some } h0 \oplus \text{Some } hf \wedge \text{safe } 0$ (*None* :: ('i, 'a, nat) cont) C' (s0, h0) S
by *blast*
then have $\text{Some } (denormalize \ H0) = \text{Some } h0 \oplus \text{Some } hf$
by (*metis* *denormalize-properties*(4))
have *ih*:
 $\neg \text{aborts } C'' \ \sigma'' \wedge (C'' = \text{Cskip} \longrightarrow$
 $(\exists h1 \ H'. \text{Some } H' = \text{Some } h1 \oplus \text{Some } hf \wedge H2 = \text{FractionalHeap.normalize } (get\text{-fh } H') \wedge \text{no-guard } H' \wedge \text{full-ownership } (get\text{-fh } H') \wedge (s2, h1) \in S))$
proof (*rule* 3(3)[of s0 H0 h0 H0'])
show $\sigma' = (s0, H0)$ **by** (*simp* *add*: $\langle \sigma' = (s0, H0) \rangle$)
show $\sigma'' = (s2, H2)$
by (*simp* *add*: 3.prem(3))
show $H0' = denormalize \ H0$ **by** (*metis* *denormalize-properties*(4) r1)
show $\text{Some } H0' = \text{Some } h0 \oplus \text{Some } hf$ **using** r1 **by** *blast*
show *full-ownership* (*get-fh* H0') \wedge *no-guard* H0' **using** r1 **by** *blast*
show *red-rtrans* C' $\sigma' \ C'' \ \sigma''$
by (*simp* *add*: 3.hyps(2))

fix n
have *safe* (*Suc* n) (*None* :: ('i, 'a, nat) cont) C (s1, h) S
using 3.prem(4) **by** *force*

then have $\exists h'' H'. \text{full-ownership } (get\text{-fh } H') \wedge \text{no-guard } H' \wedge H0 = \text{FractionalHeap.normalize } (get\text{-fh } H') \wedge \text{Some } H' = \text{Some } h'' \oplus \text{Some } hf \wedge \text{safe } n$ (*None* :: ('i, 'a, nat) cont) C' (s0, h'') S
proof (*rule safeNoneE*(3)[of n C s1 h S H hf C' s0 H0])
show $\text{Some } H = \text{Some } h \oplus \text{Some } hf$ **using** 3.prem(6) **by** *blast*
show *full-ownership* (*get-fh* H) **using** 3.prem(7) **by** *blast*
show *no-guard* H **using** 3.prem(7) **by** *auto*

```

    show red C (s1, FractionalHeap.normalize (get-fh H)) C' (s0, H0)
      by (metis 3.hyps(1) 3.prem(2) 3.prem(5) ‹σ' = (s0, H0)› denormal-
ize-properties(3))
    qed
    then obtain h3 H3' where
      r2: full-ownership (get-fh H3') ∧ no-guard H3' ∧ H0 = FractionalHeap.normalize
(get-fh H3') ∧ Some H3' = Some h3 ⊕ Some hf ∧ safe n (None :: ('i, 'a, nat)
cont) C' (s0, h3) S
      by blast
    then have h3 = h0
      by (metis ‹Some (denormalize H0) = Some h0 ⊕ Some hf› addition-cancellative
denormalize-properties(4))
    moreover have H3' = H0'
      by (metis ‹Some H0' = Some h0 ⊕ Some hf› calculation option.inject r2)
    ultimately show safe n (None :: ('i, 'a, nat) cont) C' (s0, h0) S using r2
  by blast
  qed
  then show ?case by blast
qed

theorem atomic-rule-unique:
  fixes Γ :: ('i, 'a, nat) single-context

  fixes map-to-list :: nat ⇒ 'a list
  fixes map-to-arg :: nat ⇒ 'a

  assumes Γ = (| view = f, abstract-view = α, saction = sact, uaction = uact,
invariant = J |)
  and hoare-triple-valid (None :: ('i, 'a, nat) cont) (Star P (View f J (λs. s x)))
    C (Star Q (View f J (λs. uact index (s x) (map-to-arg (s uarg)))))

  and precise J ∧ unary J
  and view-function-of-inv Γ
  and x ∉ fvC C ∪ fvA P ∪ fvA Q ∪ fvA J

  and uarg ∉ fvC C
  and l ∉ fvC C

  and x ∉ fvS (λs. map-to-list (s l))
  and x ∉ fvS (λs. map-to-arg (s uarg) # map-to-list (s l))

  and no-guard-assertion P
  and no-guard-assertion Q

  shows hoare-triple-valid (Some Γ) (Star P (UniqueGuard index (λs. map-to-list
(s l)))) (Catomic C)
    (Star Q (UniqueGuard index (λs. map-to-arg (s uarg) #
map-to-list (s l))))
  proof -

```

```

let ?J = View f J (λs. s x)
let ?J' = View f J (λs. uact index (s x) (map-to-arg (s uarg)))
let ?pre-l = (λs. map-to-list (s l))
let ?G = UniqueGuard index ?pre-l
let ?l = λs. map-to-arg (s uarg) # map-to-list (s l)
let ?G' = UniqueGuard index ?l

have unaries: unary ?J ∧ unary ?J'
  by (simp add: assms(3) unary-inv-then-view)
moreover have precises: precise ?J ∧ precise ?J'
  by (simp add: assms(3) precise-inv-then-view)

obtain Σ where asm0: ∧n σ. σ, σ ⊨ Star P ?J ⇒ safe n (None :: ('i, 'a,
nat) cont) C σ (Σ σ)
  ∧σ σ'. σ, σ' ⊨ Star P ?J ⇒ pair-sat (Σ σ) (Σ σ') (Star Q ?J')
  using assms(2) hoare-triple-valid-def by blast

define start where start = (λσ. { (s, h) | s h hj. agrees (- {x}) (fst σ) s ∧ Some
h = Some (remove-guards (snd σ)) ⊕ Some hj ∧ (s, hj), (s, hj) ⊨ ?J})
define end-qj where end-qj = (λσ. ⋃σ' ∈ start σ. Σ σ')
define Σ' where Σ' = (λσ. { (s, add-uguard-to-no-guard index hq (?l s)) | s hq h
hj. (s, h) ∈ end-qj σ ∧ Some h = Some hq ⊕ Some hj ∧ (s, hj), (s, hj) ⊨ ?J' })

let ?Σ' = λσ. close-var (Σ' σ) x

show hoare-triple-valid (Some Γ) (Star P ?G) (Catomic C) (Star Q ?G')
proof (rule hoare-triple-validI)
  show ∧s h s' h'. (s, h), (s', h') ⊨ Star P ?G ⇒ pair-sat (?Σ' (s, h)) (?Σ' (s',
h')) (Star Q ?G')
  proof -
    fix s1 h1 s2 h2
    assume asm1: (s1, h1), (s2, h2) ⊨ Star P ?G
    then obtain p1 p2 g1 g2 where r0: Some h1 = Some p1 ⊕ Some g1
    Some h2 = Some p2 ⊕ Some g2
    (s1, p1), (s2, p2) ⊨ P (s1, g1), (s2, g2) ⊨ ?G
    using hyper-sat.simps(4) by auto
    then obtain remove-guards h1 = p1 remove-guards h2 = p2
    by (meson assms(10) hyper-sat.simps(13) no-guard-and-no-heap no-guard-assertion-def)

    have pair-sat (Σ' (s1, h1)) (Σ' (s2, h2)) (Star Q ?G')
    proof (rule pair-satI)
      fix s1' hqg1 s2' hqg2 σ2'
      assume asm2: (s1', hqg1) ∈ Σ' (s1, h1) ∧ (s2', hqg2) ∈ Σ' (s2, h2)
      then obtain h1' hj1' h2' hj2' hq1 hq2 where r: (s1', h1') ∈ end-qj (s1,
h1) Some h1' = Some hq1 ⊕ Some hj1'
      (s1', hj1'), (s1', hj1') ⊨ ?J' (s2', h2') ∈ end-qj (s2, h2) Some h2' =
Some hq2 ⊕ Some hj2' (s2', hj2'), (s2', hj2') ⊨ ?J'
      hqg1 = add-uguard-to-no-guard index hq1 (?l s1') hqg2 = add-uguard-to-no-guard
index hq2 (?l s2')

```

using Σ' -def **by** blast
then obtain $\sigma 1' \sigma 2'$ **where** $\sigma 1' \in \text{start } (s1, h1) \sigma 2' \in \text{start } (s2, h2) (s1', h1') \in \Sigma \sigma 1' (s2', h2') \in \Sigma \sigma 2'$
using end-gj-def **by** blast
then obtain $hj1 \ hj2$ **where** $\text{agrees } (- \{x\}) \ s1 \ (\text{fst } \sigma 1') \ \text{Some } (\text{snd } \sigma 1') = \text{Some } p1 \oplus \text{Some } hj1 \ (\text{fst } \sigma 1', \ hj1), \ (\text{fst } \sigma 1', \ hj1) \models ?J$
 $\text{agrees } (- \{x\}) \ s2 \ (\text{fst } \sigma 2') \ \text{Some } (\text{snd } \sigma 2') = \text{Some } p2 \oplus \text{Some } hj2 \ (\text{fst } \sigma 2', \ hj2), \ (\text{fst } \sigma 2', \ hj2) \models ?J$
using start-def $\langle \text{remove-guards } h1 = p1 \rangle \langle \text{remove-guards } h2 = p2 \rangle$ **by** force

moreover have $(\text{fst } \sigma 1', \ hj1), \ (\text{fst } \sigma 2', \ hj2) \models ?J$
using calculation(3) calculation(6) unaries unaryE **by** blast
moreover have $(\text{fst } \sigma 1', \ p1), \ (\text{fst } \sigma 2', \ p2) \models P$
proof –
have $\text{fvA } P \subseteq - \{x\}$
using assms(5) **by** force
then have $\text{agrees } (\text{fvA } P) \ (\text{fst } \sigma 1') \ s1 \wedge \text{agrees } (\text{fvA } P) \ (\text{fst } \sigma 2') \ s2$
using calculation(1) calculation(4)
by (metis agrees-comm agrees-union subset-Un-eq)
then show ?thesis **using** r0(3)
by (meson agrees-same sat-comm)
qed

ultimately have $\sigma 1', \ \sigma 2' \models \text{Star } P \ ?J$ **using** hyper-sat.simps(4)[of $\text{fst } \sigma 1' \ \text{snd } \sigma 1' \ \text{fst } \sigma 2' \ \text{snd } \sigma 2'$] prod.collapse
by metis
then have pair-sat $(\Sigma \sigma 1') \ (\Sigma \sigma 2') \ (\text{Star } Q \ ?J')$
using asm0(2)[of $\sigma 1' \ \sigma 2'$] **by** blast
then have $(s1', \ h1'), \ (s2', \ h2') \models \text{Star } Q \ ?J'$
using $\langle (s1', \ h1') \in \Sigma \sigma 1' \rangle \langle (s2', \ h2') \in \Sigma \sigma 2' \rangle$ pair-sat-def **by** blast
moreover have $(s1', \ hj1'), \ (s2', \ hj2') \models ?J'$
using r(3) r(6) unaries unaryE **by** blast
moreover have $(s1', \ hq1), \ (s2', \ hq2) \models Q$ **using** magic-lemma
using calculation(1) calculation(2) precisifies r(2) r(5) **by** blast
have $(s1', \ \text{add-uguard-to-no-guard } \text{index } hq1 \ (\ ?l \ s1')), \ (s2', \ \text{add-uguard-to-no-guard } \text{index } hq2 \ (\ ?l \ s2')) \models \text{Star } Q \ ?G'$
proof (rule no-guard-then-sat-star-uguard)
show no-guard $hq1 \wedge$ no-guard $hq2$
using $\langle (s1', \ hq1), \ (s2', \ hq2) \models Q \rangle$ assms(11) no-guard-assertion-def **by** blast

show $(s1', \ hq1), \ (s2', \ hq2) \models Q$
using $\langle (s1', \ hq1), \ (s2', \ hq2) \models Q \rangle$ **by** auto
qed
then show $(s1', \ hqg1), \ (s2', \ hqg2) \models \text{Star } Q \ ?G'$
using r(7) r(8) **by** force
qed
then show pair-sat $(\ ?\Sigma' \ (s1, \ h1)) \ (\ ?\Sigma' \ (s2, \ h2)) \ (\text{Star } Q \ ?G')$
proof (rule pair-sat-close-var-double)


```

    show  $x \notin \text{fv}A$  (Star Q (UniqueGuard index ( $\lambda s$ . map-to-arg (s uarg) #
map-to-list (s l))))
    using assms(5) assms(9) by auto
    qed
    qed

fix pre-s h k
assume (pre-s, h), (pre-s, h)  $\models$  Star P ?G
then obtain pp gg where Some h = Some pp  $\oplus$  Some gg (pre-s, pp), (pre-s,
pp)  $\models$  P (pre-s, gg), (pre-s, gg)  $\models$  ?G
    using always-sat-refl hyper-sat.simps(4) by blast
then have remove-guards h = pp
    using assms(10) hyper-sat.simps(13) no-guard-and-no-heap no-guard-assertion-def
by metis
then have (pre-s, remove-guards h), (pre-s, remove-guards h)  $\models$  P
    using  $\langle$ (pre-s, pp), (pre-s, pp)  $\models$  P $\rangle$  hyper-sat.simps(9) by blast
then have (pre-s, remove-guards h), (pre-s, remove-guards h)  $\models$  P
    by (simp add: no-guard-remove-guards)

show safe k (Some  $\Gamma$ ) (Catomic C) (pre-s, h) (? $\Sigma'$  (pre-s, h))
proof (cases k)
case (Suc n)
    moreover have safe (Suc n) (Some  $\Gamma$ ) (Catomic C) (pre-s, h) (? $\Sigma'$  (pre-s,
h))
    proof (rule safeSomeAltI)
        show Catomic C = Cskip  $\implies$  (pre-s, h)  $\in$  ? $\Sigma'$  (pre-s, h) by simp

fix H hf hj v0

assume asm2: Some H = Some h  $\oplus$  Some hj  $\oplus$  Some hf  $\wedge$  full-ownership
(get-fh H)  $\wedge$  semi-consistent  $\Gamma$  v0 H  $\wedge$  sat-inv pre-s hj  $\Gamma$ 

define v where v = f (normalize (get-fh H))
define s where s = pre-s(x := v)
then have v = s x by simp
    moreover have agreements: agrees (fvC C  $\cup$  fvA P  $\cup$  fvA Q  $\cup$  fvA J  $\cup$ 
fvA (UniqueGuard k ?pre-l)) s pre-s
    by (metis UnE agrees-comm agrees-update assms(5) assms(8) fvA.simps(9)
s-def)
    have asm1: (s, h), (s, h)  $\models$  Star P ?G
        using Un-iff[of x]  $\langle$ (pre-s, h), (pre-s, h)  $\models$  Star P (UniqueGuard index
( $\lambda s$ . map-to-list (s l))) $\rangle$ 
        agrees-same agrees-update[of x] always-sat-refl assms(5) assms(8)
fvA.simps(3)[of P UniqueGuard index ( $\lambda s$ . map-to-list (s l))]
        fvA.simps(9)[of index ( $\lambda s$ . map-to-list (s l))] s-def
    by metis
    moreover have asm2-bis: sat-inv s hj  $\Gamma$ 
    proof (rule sat-inv-agrees)
        show sat-inv pre-s hj  $\Gamma$  using asm2 by simp

```

show *agrees* (*fvA* (*invariant* Γ)) *pre-s s*
using *assms*(1) *assms*(5) *s-def*
by (*simp add: agrees-update*)
qed
moreover have (*s, remove-guards h*), (*s, remove-guards h*) $\models P$
by (*meson* \langle *pre-s, remove-guards h* \rangle , (*pre-s, remove-guards h*) $\models P$ \rangle
agreements agrees-same agrees-union always-sat-refl)

moreover have *agrees* ($- \{x\}$) *pre-s s*
proof (*rule agreesI*)
fix *y* **assume** $y \in - \{x\}$
then have $y \neq x$
by *force*
then show *pre-s y = s y*
by (*simp add: s-def*)
qed

moreover obtain (*s, pp*), (*s, pp*) $\models P$ (*s, gg*), (*s, gg*) $\models ?G$
by (*metis* \langle *pre-s, gg* \rangle , (*pre-s, gg*) \models *UniqueGuard index* ($\lambda s.$ *map-to-list* (*s l*) \rangle \langle *remove-guards h = pp* \rangle *agrees-same-aux agrees-update always-sat-refl-aux*
assms(8) *calculation*(4) *fvA.simps*(9) *s-def*)

let $?hf =$ *remove-guards hf*
let $?H =$ *remove-guards H*
let $?h =$ *remove-guards h*

obtain *hhj* **where** *Some hhj = Some h* \oplus *Some hj*
by (*metis asm2 plus.simps*(2) *plus.simps*(3) *plus-comm*)
then have *Some H = Some hhj* \oplus *Some hf*
using *asm2* **by** *presburger*
then have *Some (remove-guards hhj) = Some ?h* \oplus *Some hj*
by (*metis* \langle *Some hhj = Some h* \oplus *Some hj* \rangle *asm2 no-guards-remove*
remove-guards-sum sat-inv-def)

moreover have f (*normalize (get-fh hj)*) = v
proof –
have *view* Γ (*normalize (get-fh hj)*) = *view* Γ (*normalize (get-fh H)*)
using *assms*(4) *view-function-of-invE*
by (*metis* (*no-types, opaque-lifting*) \langle *Some hhj = Some h* \oplus *Some hj* \rangle
asm2 larger-def larger-trans plus-comm)
then show $?thesis$ **using** *assms*(1) *v-def* **by** *fastforce*
qed

then have (*s, hj*), (*s, hj*) $\models ?J$
by (*metis* \langle $v = s x$ \rangle *asm2-bis* *assms*(1) *hyper-sat.simps*(11) *sat-inv-def*
select-convs(5))

ultimately have (*s, remove-guards hhj*), (*s, remove-guards hhj*) \models *Star P*

?J
using $\langle (s, \text{remove-guards } h), (s, \text{remove-guards } h) \models P \rangle \text{ hyper-sat.simps}(4)$
by *blast*

then have *all-safes*: $\bigwedge n. \text{ safe } n \text{ (None :: ('i, 'a, nat) cont) } C \text{ (s, remove-guards hhj) } (\Sigma (s, \text{remove-guards hhj}))$
using *asm0(1)* **by** *blast*
then have $\bigwedge \sigma 1 \ H1 \ \sigma 2 \ H2 \ s2 \ C2. \text{ red-rtrans } C \ \sigma 1 \ C2 \ \sigma 2 \implies \sigma 1 = (s, H1) \implies \sigma 2 = (s2, H2) \implies$
 $\text{?H} = \text{denormalize } H1 \implies$
 $\neg \text{ aborts } C2 \ \sigma 2 \wedge (C2 = \text{Cskip} \longrightarrow (\exists h1 \ H'. \text{ Some } H' = \text{Some } h1 \oplus \text{Some } ?hf \wedge H2 = \text{FractionalHeap.normalize } (\text{get-fh } H') \wedge \text{no-guard } H' \wedge \text{full-ownership } (\text{get-fh } H') \wedge (s2, h1) \in \Sigma (s, \text{remove-guards hhj})))$

proof –
fix $\sigma 1 \ H1 \ \sigma 2 \ H2 \ s2 \ C2$
assume $\text{?H} = \text{denormalize } H1$
assume $\text{red-rtrans } C \ \sigma 1 \ C2 \ \sigma 2 \ \sigma 1 = (s, H1) \ \sigma 2 = (s2, H2)$

then show $\neg \text{ aborts } C2 \ \sigma 2 \wedge (C2 = \text{Cskip} \longrightarrow (\exists h1 \ H'. \text{ Some } H' = \text{Some } h1 \oplus \text{Some } (\text{remove-guards } hf) \wedge H2 = \text{FractionalHeap.normalize } (\text{get-fh } H') \wedge \text{no-guard } H' \wedge \text{full-ownership } (\text{get-fh } H') \wedge (s2, h1) \in \Sigma (s, \text{remove-guards hhj})))$
using *all-safes*
proof (*rule safe-atomic*)
show $\text{?H} = \text{denormalize } H1$ **using** $\langle \text{?H} = \text{denormalize } H1 \rangle$ **by** *simp*
show $\text{Some } \text{?H} = \text{Some } (\text{remove-guards } hhj) \oplus \text{Some } ?hf$
using $\langle \text{Some } H = \text{Some } hhj \oplus \text{Some } hf \rangle \text{ remove-guards-sum}$ **by** *blast*
show $\text{full-ownership } (\text{get-fh } (\text{remove-guards } H)) \wedge \text{no-guard } (\text{remove-guards } H)$
H)
by (*metis asm2 get-fh-remove-guards no-guard-remove-guards*)
qed
qed
moreover have $\text{?H} = \text{denormalize } (\text{normalize } (\text{get-fh } H))$
by (*metis asm2 denormalize-properties(5)*)
ultimately have *safe-atomic-simplified*: $\bigwedge \sigma 2 \ H2 \ s2 \ C2. \text{ red-rtrans } C \text{ (s, normalize } (\text{get-fh } H)) \ C2 \ \sigma 2 \implies \sigma 2 = (s2, H2) \implies \neg \text{ aborts } C2 \ \sigma 2 \wedge (C2 = \text{Cskip} \longrightarrow (\exists h1 \ H'. \text{ Some } H' = \text{Some } h1 \oplus \text{Some } ?hf \wedge H2 = \text{FractionalHeap.normalize } (\text{get-fh } H') \wedge \text{no-guard } H' \wedge \text{full-ownership } (\text{get-fh } H') \wedge (s2, h1) \in \Sigma (s, \text{remove-guards hhj})))$
by *presburger*

have $\neg \text{ aborts } (\text{Catomic } C) \text{ (s, normalize } (\text{get-fh } H))$
proof (*rule ccontr*)

assume $\neg \neg$ *aborts* (*Catomic* *C*) (*s*, *normalize* (*get-fh* *H*))
then obtain $C' \sigma'$ **where** *asm3*: *red-rtrans* *C* (*s*, *FractionalHeap.normalize* (*get-fh* *H*)) $C' \sigma'$
aborts $C' \sigma'$
using *abort-atomic-cases* **by** *blast*
then have \neg *aborts* $C' \sigma'$ **using** *safe-atomic-simplified*[*of* $C' \sigma'$ *fst* σ' *snd* σ'] **by** *simp*
then show *False* **using** *asm3*(2) **by** *simp*
qed
then show \neg *aborts* (*Catomic* *C*) (*pre-s*, *normalize* (*get-fh* *H*))
by (*metis* *agreements* *aborts-agrees* *agrees-comm* *agrees-union* *fst-eqD* *fvC.simps*(11) *snd-conv*)

fix $C' \text{pre-}s' h'$
assume *red* (*Catomic* *C*) (*pre-s*, *FractionalHeap.normalize* (*get-fh* *H*)) C' (*pre-s'*, h')
then obtain s' **where** *red* (*Catomic* *C*) (*s*, *FractionalHeap.normalize* (*get-fh* *H*)) C' (s' , h')
agrees ($-\{x\}$) s' *pre-s'*
by (*metis* (*no-types*, *lifting*) *UnI1* \langle *agrees* ($-\{x\}$) *pre-s* $s\rangle$ *agrees-comm* *assms*(5) *fst-eqD* *fvC.simps*(11) *red-agrees* *snd-conv* *subset-Compl-singleton*)

then obtain $h1 H'$ **where** *asm3*: *Some* $H' = \text{Some } h1 \oplus \text{Some } (\text{remove-guards } hf) C' = \text{Cskip}$
 $h' = \text{FractionalHeap.normalize } (\text{get-fh } H') \text{ no-guard } H' \wedge \text{full-ownership } (\text{get-fh } H') (s', h1) \in \Sigma (s, \text{remove-guards } hhj)$
using *safe-atomic-simplified*[*of* $C' (s', h')$ $s' h'$] **by** (*metis* *red-atomic-cases*)

moreover have $s x = s' x \wedge s' uarg = s uarg \wedge s l = s' l$ **using** *red-not-in-fv-not-touched*
using \langle *red* (*Catomic* *C*) (*s*, *FractionalHeap.normalize* (*get-fh* *H*)) $C' (s', h')\rangle$
by (*metis* *Un-iff* *assms*(5) *assms*(6) *assms*(7) *fst-conv* *fvC.simps*(11))
have $\exists hq' hj'. \text{Some } h1 = \text{Some } hq' \oplus \text{Some } hj' \wedge (s', \text{add-uguard-to-no-guard } \text{index } hq' (?l s')) \in \Sigma' (\text{pre-s}, h) \wedge \text{sat-inv } s' hj' \Gamma$
 $\wedge f (\text{normalize } (\text{get-fh } hj')) = \text{uact } \text{index } (s' x) (\text{map-to-arg } (s' uarg))$
proof –

have *pair-sat* ($\Sigma (s, \text{remove-guards } hhj)$) ($\Sigma (s, \text{remove-guards } hhj)$) (*Star* $Q ?J'$)
using *asm0*(2)[*of* ($s, \text{remove-guards } hhj$) ($s, \text{remove-guards } hhj$)]
using $\langle (s, \text{remove-guards } hhj), (s, \text{remove-guards } hhj) \models \text{Star } P ?J \rangle$ **by** *blast*
then have $(s', h1), (s', h1) \models \text{Star } Q ?J'$
using *asm3*(5) *pair-sat-def* **by** *blast*
then obtain $hq' hj'$ **where** *Some* $h1 = \text{Some } hq' \oplus \text{Some } hj' (s', hq')$, $(s', hq') \models Q (s', hj'), (s', hj') \models ?J'$
using *always-sat-refl* *hyper-sat.simps*(4) **by** *blast*
then have *no-guard* hj'

by (*metis* (*no-types*, *opaque-lifting*) *calculation*(1) *calculation*(4)
no-guard-then-smaller-same plus-comm)
moreover have f (*normalize* (*get-fh* hj')) = *uact index* ($s' x$) (*map-to-arg* ($s' uarg$))
using $\langle (s', hj'), (s', hj') \models \text{View } f J (\lambda s. \text{uact index } (s x) (\text{map-to-arg } (s uarg))) \rangle$ **by** *auto*
moreover have $(s, \text{remove-guards } hhj) \in \text{start } (pre-s, h)$
proof –
have $\text{Some } (\text{remove-guards } hhj) = \text{Some } ?h \oplus \text{Some } hj$
using $\langle \text{Some } (\text{remove-guards } hhj) = \text{Some } (\text{remove-guards } h) \oplus \text{Some } hj \rangle$ **by** *blast*
moreover have $(s, hj), (s, hj) \models ?J$
using $\langle (s, hj), (s, hj) \models ?J \rangle$ **by** *fastforce*
ultimately show *?thesis using start-def*
using $\langle \text{agrees } (- \{x\}) \text{ pre-} s \rangle$ **by** *fastforce*
qed
then have $(s', h1) \in \text{end-qj } (pre-s, h)$
using $\langle \text{end-qj} \equiv \lambda \sigma. \bigcup (\Sigma \text{ 'start } \sigma) \rangle$ *asm3*(5) **by** *blast*

then have $(s', \text{add-uguard-to-no-guard index } hq' (?l s')) \in \Sigma' (pre-s, h)$
using $\Sigma' \text{-def } \langle (s', hj'), (s', hj') \models ?J' \rangle \langle \text{Some } h1 = \text{Some } hq' \oplus \text{Some } hj' \rangle$ **by** *blast*
ultimately show $\exists hq' hj'.$
 $\text{Some } h1 = \text{Some } hq' \oplus \text{Some } hj' \wedge$
 $(s', \text{add-uguard-to-no-guard index } hq' (\text{map-to-arg } (s' uarg) \# \text{map-to-list } (s' l))) \in \Sigma' (pre-s, h) \wedge$
 $\text{sat-inv } s' hj' \Gamma \wedge f (\text{FractionalHeap.normalize } (\text{get-fh } hj')) = \text{uact index } (s' x) (\text{map-to-arg } (s' uarg))$
using $\langle (s', hj'), (s', hj') \models ?J' \rangle \langle \text{Some } h1 = \text{Some } hq' \oplus \text{Some } hj' \rangle$
assms(1) *hyper-sat.simps*(11) *sat-inv-def select-convs*(5)
by *fastforce*
qed
then obtain $hq' hj'$ **where** $\text{Some } h1 = \text{Some } hq' \oplus \text{Some } hj' (s', \text{add-uguard-to-no-guard index } hq' (?l s')) \in \Sigma' (pre-s, h) \text{ sat-inv } s' hj' \Gamma$
 $f (\text{normalize } (\text{get-fh } hj')) = \text{uact index } (s' x) (\text{map-to-arg } (s' uarg))$
by *blast*
then have *safe n* ($\text{Some } \Gamma$) $C' (s', \text{add-uguard-to-no-guard index } hq' (?l s'))$
 $(\Sigma' (pre-s, h))$
using *asm3*(2) *safe-skip* **by** *blast*

moreover have $\exists H''. \text{semi-consistent } \Gamma \ v0 \ H'' \wedge \text{Some } H'' = \text{Some } (\text{add-uguard-to-no-guard index } hq' (?l s')) \oplus \text{Some } hj' \oplus \text{Some } hf$
proof –
have $\text{Some } (\text{add-uguard-to-no-guard index } hq' (?l s')) = \text{Some } hq' \oplus \text{Some } (\text{Map.empty}, \text{None}, [\text{index} \mapsto ?l s'])$
by (*metis* $\langle \text{Some } h1 = \text{Some } hq' \oplus \text{Some } hj' \rangle$ *add-uguard-as-sum* *calculation*(1) *calculation*(4) *no-guard-then-smaller-same*)

obtain hhf **where** $\text{Some } hhf = \text{Some } h \oplus \text{Some } hf$

by (*metis* (*no-types*, *opaque-lifting*) $\langle \text{Some } H = \text{Some } hhj \oplus \text{Some } hf \rangle \langle \text{Some } hhj = \text{Some } h \oplus \text{Some } hj \rangle$ *option.exhaust-sel plus.simps(1) plus-asso plus-comm*)

then have *all-guards hhf*

by (*metis* (*no-types*, *lifting*) *all-guards-no-guard-propagates asm2 plus-asso plus-comm sat-inv-def semi-consistent-def*)

moreover have *get-gs h = None \wedge get-gu h index = Some (?pre-l s)*

proof –

have *no-guard pp*

using $\langle \text{remove-guards } h = pp \rangle$ *no-guard-remove-guards* **by** *blast*

then show *?thesis*

by (*metis* (*no-types*, *lifting*) $\langle \text{Some } h = \text{Some } pp \oplus \text{Some } gg \rangle \langle \bigwedge \text{thesis. } (\llbracket (s, pp), (s, pp) \rrbracket \models P; (s, gg), (s, gg) \rrbracket \models \text{UniqueGuard index } (\lambda s. \text{map-to-list } (s \ l))) \rrbracket \implies \text{thesis} \rangle \implies \text{thesis}$ *full-uguard-sum-same hyper-sat.simps(13) no-guard-remove(1) plus-comm*)

qed

moreover have $\bigwedge i'. i' \neq \text{index} \implies \text{get-gu } h \ i' = \text{None}$

by (*metis* $\langle \text{Some } h = \text{Some } pp \oplus \text{Some } gg \rangle \langle \bigwedge \text{thesis. } (\llbracket (s, pp), (s, pp) \rrbracket \models P; (s, gg), (s, gg) \rrbracket \models \text{UniqueGuard index } (\lambda s. \text{map-to-list } (s \ l))) \rrbracket \implies \text{thesis} \rangle \langle \text{remove-guards } h = pp \rangle$ *hyper-sat.simps(13) no-guard-remove(2) no-guard-remove-guards plus-comm*)

then obtain *sargs* **where** *get-gu hf index = None \wedge get-gs hf = Some (pwrite, sargs)*

by (*metis* (*no-types*, *opaque-lifting*) $\langle \text{Some } hhf = \text{Some } h \oplus \text{Some } hf \rangle$ *add-gs.simps(1) all-guards-def calculation(1) calculation(2) compatible-def compatible-eq option.distinct(1) plus-extract(2)*)

moreover obtain *uargs* **where** $\bigwedge i'. i' \neq \text{index} \implies \text{get-gu } hf \ i' = \text{Some } (uargs \ i')$

by (*metis* (*no-types*, *opaque-lifting*) $\langle \text{Some } hhf = \text{Some } h \oplus \text{Some } hf \rangle \langle \bigwedge i'. i' \neq \text{index} \implies \text{get-gu } h \ i' = \text{None} \rangle$ *add-gu-def add-gu-single.simps(1) all-guards-exists-uargs calculation(1) plus-extract(3)*)

then obtain *ghf* **where** *ghf-def: Some hf = Some (remove-guards hf) \oplus Some ghf*

get-fh ghf = Map.empty *get-gu ghf index = None*

get-gs ghf = Some (pwrite, sargs) $\wedge i'. i' \neq \text{index} \implies \text{get-gu } ghf \ i' = \text{Some } (uargs \ i')$

using *decompose-guard-remove-easy[of hf]*

using *calculation(3)* **by** *auto*

have (*Map.empty*, *None*, [*index* \mapsto ?l s']) **##** *ghf*

proof (*rule compatibleI*)

show *compatible-fract-heaps (get-fh (Map.empty, None, [index \mapsto map-to-arg (s' uarg) # map-to-list (s' l)])) (get-fh ghf)*

using *compatible-fract-heapsI* **by** *fastforce*

show $\bigwedge k. \text{get-gu } (Map.empty, None, [index \mapsto \text{map-to-arg } (s' \ uarg) \ # \ \text{map-to-list } (s' \ l)]) \ k = \text{None} \vee \text{get-gu } ghf \ k = \text{None}$

using *ghf-def(3)* **by** *auto*

qed (*simp*)

then obtain g **where** $g\text{-def}$: $\text{Some } g = \text{Some } (\text{Map.empty}, \text{None}, [\text{index} \mapsto ?l \ s']) \oplus \text{Some } ghf$
by *simp*
moreover have $H' \#\# g$
proof (*rule compatibleI*)
have $\text{get-fh } g = \text{add-fh } \text{Map.empty } \text{Map.empty}$
using $\text{add-get-fh}[\text{of } g \ (\text{Map.empty}, \text{None}, [\text{index} \mapsto ?l \ s']) \ ghf]$
 $g\text{-def} \ \langle \text{get-fh } ghf = \text{Map.empty} \rangle$
by *fastforce*
then have $\text{get-fh } g = \text{Map.empty}$
using *add-fh-map-empty* **by** *auto*
then show *compatible-fract-heaps* ($\text{get-fh } H'$) ($\text{get-fh } g$)
using *compatible-fract-heapsI* **by** *force*
show $\bigwedge k. \text{get-gu } H' \ k = \text{None} \vee \text{get-gu } g \ k = \text{None}$
by (*meson asm3(4) no-guard-def*)
show $\bigwedge p \ p'. \text{get-gs } H' = \text{Some } p \wedge \text{get-gs } g = \text{Some } p' \implies \text{pgte } p \ \text{write}$
(*padd (fst p) (fst p')*)
by (*metis asm3(4) no-guard-def option.simps(3)*)
qed
then obtain H'' **where** $\text{Some } H'' = \text{Some } H' \oplus \text{Some } g$
by *simp*
then have $\text{Some } H'' = \text{Some } (\text{add-uguard-to-no-guard } \text{index } hq' \ (?l \ s'))$
 $\oplus \text{Some } hj' \oplus \text{Some } hf$
proof –
have $\text{Some } H'' = \text{Some } h1 \oplus \text{Some } g \oplus \text{Some } (\text{remove-guards } hf)$
by (*metis* $\langle \text{Some } H'' = \text{Some } H' \oplus \text{Some } g \rangle \text{asm3(1) plus-comm simpler-asso}$)
moreover have $\text{Some } (\text{add-uguard-to-no-guard } \text{index } hq' \ (?l \ s')) = \text{Some } hq' \oplus \text{Some } (\text{Map.empty}, \text{None}, [\text{index} \mapsto ?l \ s'])$
using $\langle \text{Some } (\text{add-uguard-to-no-guard } \text{index } hq' \ (\text{map-to-arg } (s' \ \text{uarg}) \# \text{map-to-list } (s' \ l))) = \text{Some } hq' \oplus \text{Some } (\text{Map.empty}, \text{None}, [\text{index} \mapsto \text{map-to-arg } (s' \ \text{uarg}) \# \text{map-to-list } (s' \ l)]) \rangle$ **by** *blast*
ultimately show *?thesis*
by (*metis* (*no-types, lifting*) $\langle \text{Some } h1 = \text{Some } hq' \oplus \text{Some } hj' \rangle \ g\text{-def } ghf\text{-def}(1) \ \text{plus-comm simpler-asso}$)
qed

moreover have *semi-consistent* $\Gamma \ v0 \ H''$
proof (*rule semi-consistentI*)
have $\text{get-gs } g = \text{Some } (\text{pwrite}, \text{sargs})$
by (*metis full-sguard-sum-same g-def ghf-def(4) plus-comm*)
moreover have $\text{get-gu } g \ \text{index} = \text{Some } (?l \ s')$
proof (*rule full-uguard-sum-same*)
show $\text{get-gu } (\text{Map.empty}, \text{None}, [\text{index} \mapsto ?l \ s']) \ \text{index} = \text{Some } (?l \ s')$
using *get-gu.simps* **by** *auto*
show $\text{Some } g = \text{Some } (\text{Map.empty}, \text{None}, [\text{index} \mapsto ?l \ s']) \oplus \text{Some } ghf$
using $g\text{-def}$ **by** *auto*
qed
moreover have $\bigwedge i'. i' \neq \text{index} \implies \text{get-gu } g \ i' = \text{Some } (\text{uargs } i')$

by (metis full-uguard-sum-same g-def ghf-def(5) plus-comm)
 ultimately have all-guards g
 by (metis all-guardsI option.discI)
 then show all-guards H''
 by (metis ‹Some H'' = Some H' \oplus Some g› all-guards-same plus-comm)
 show reachable Γ v0 H''
 proof (rule reachableI)
 fix sargs' uargs'
 assume get-gs H'' = Some (pwrite, sargs') \wedge ($\forall k.$ get-gu H'' k = Some (uargs' k))
 then have sargs = sargs'
 by (metis (no-types, opaque-lifting) Pair-inject ‹Some H'' = Some H' \oplus Some g› ‹get-gs g = Some (pwrite, sargs)› full-sguard-sum-same option.inject plus-comm)
 moreover have uargs' index = ?l s'
 by (metis ‹Some H'' = Some H' \oplus Some g› ‹get-gs H'' = Some (pwrite, sargs') \wedge ($\forall k.$ get-gu H'' k = Some (uargs' k))› ‹get-gu g index = Some (map-to-arg (s' uarg) # map-to-list (s' l))› asm3(4) no-guard-remove(2) option.inject plus-comm)
 moreover have $\wedge i'. i' \neq \text{index} \implies \text{uargs}' i' = \text{uargs } i'$
 by (metis ‹Some H'' = Some H' \oplus Some g› ‹ $\wedge i'. i' \neq \text{index} \implies \text{get-gu } g \ i' = \text{Some } (\text{uargs } i')$ › ‹get-gs H'' = Some (pwrite, sargs') \wedge ($\forall k.$ get-gu H'' k = Some (uargs' k))› asm3(4) no-guard-remove(2) option.sel plus-comm)
 moreover have view Γ (FractionalHeap.normalize (get-fh hj')) = view Γ (FractionalHeap.normalize (get-fh H''))
 using assms(4) ‹sat-inv s' hj' Γ ›
 proof (rule view-function-of-invE)
 show H'' \succeq hj'
 by (metis (no-types, opaque-lifting) ‹Some H'' = Some H' \oplus Some g› ‹Some h1 = Some hq' \oplus Some hj'› asm3(1) larger-def larger-trans plus-comm)
 qed
 moreover have reachable-value (saction Γ) (uaction Γ) v0 sargs (uargs(index := ?l s')) (uact index (s' x) (map-to-arg (s' uarg)))
 proof –
 have reachable-value (saction Γ) (uaction Γ) v0 sargs (uargs(index := ?pre-l s')) (view Γ (FractionalHeap.normalize (get-fh H)))
 proof –
 have reachable Γ v0 H
 by (meson asm2 semi-consistent-def)
 moreover have get-gs H = Some (pwrite, sargs)
 by (metis ‹Some H = Some hhj \oplus Some hf› ‹get-gu hf index = None \wedge get-gs hf = Some (pwrite, sargs)› full-sguard-sum-same plus-comm)
 moreover have get-gu H index = Some (?pre-l s')
 by (metis ‹Some H = Some hhj \oplus Some hf› ‹Some hhj = Some h \oplus Some hj› ‹get-gs h = None \wedge get-gu h index = Some (map-to-list (s l))› ‹s x = s' x \wedge s' uarg = s uarg \wedge s l = s' l› full-uguard-sum-same)
 moreover have $\wedge i. i \neq \text{index} \implies \text{get-gu } H \ i = \text{Some } (\text{uargs } i)$
 by (metis ‹Some H = Some hhj \oplus Some hf› ‹ $\wedge i'. i' \neq \text{index} \implies$


```

get-gu hf i' = Some (uargs i') › full-uguard-sum-same plus-comm)
  ultimately show ?thesis
    by (simp add: reachable-def)
  qed
  moreover have view  $\Gamma$  (FractionalHeap.normalize (get-fh hj)) = view
 $\Gamma$  (FractionalHeap.normalize (get-fh H))
    using assms(4)
  proof (rule view-function-of-invE)
    show sat-inv s hj  $\Gamma$ 
      by (simp add: asm2-bis)
    show  $H \succeq hj$ 
      by (metis (no-types, opaque-lifting) ‹Some H = Some hhj  $\oplus$  Some
hf› ‹Some hhj = Some h  $\oplus$  Some hj› larger-def larger-trans plus-comm)
    qed
    moreover have s' x = v
      using ‹s x = s' x  $\wedge$  s' uarg = s uarg  $\wedge$  s l = s' l› ‹v = s x› by
presburger
    ultimately have reachable-value (saction  $\Gamma$ ) (uaction  $\Gamma$ ) v0 sargs
(uargs(index := ?pre-l s')) v
      using ‹f (FractionalHeap.normalize (get-fh hj)) = v› assms(1) by
auto
    then show ?thesis
      by (metis UniqueStep ‹s' x = v› assms(1) fun-upd-same fun-upd-upd
select-convs(4))
    qed
    moreover have uargs' = (uargs(index := map-to-arg (s' uarg) #
map-to-list (s' l)))
      proof (rule ext)
        fix i show uargs' i = (uargs(index := map-to-arg (s' uarg) #
map-to-list (s' l))) i
          apply (cases i = index)
          using calculation(2) apply auto[1]
          using calculation(3) by force
        qed
      ultimately show reachable-value (saction  $\Gamma$ ) (uaction  $\Gamma$ ) v0 sargs'
uargs' (view  $\Gamma$  (FractionalHeap.normalize (get-fh H'')))
        using ‹f (FractionalHeap.normalize (get-fh hj')) = uact index (s' x)
(map-to-arg (s' uarg))› assms(1) by force
      qed
    qed
    ultimately show  $\exists H''$ . semi-consistent  $\Gamma$  v0 H''  $\wedge$  Some H'' = Some
(add-uguard-to-no-guard index hq' (map-to-arg (s' uarg) # map-to-list (s' l)))  $\oplus$ 
Some hj'  $\oplus$  Some hf
      by blast
    qed
    ultimately obtain H'' where semi-consistent  $\Gamma$  v0 H''  $\wedge$ 
Some H'' = Some (add-uguard-to-no-guard index hq' (map-to-arg (s' uarg) #
map-to-list (s' l)))  $\oplus$  Some hj'  $\oplus$  Some hf by blast
    moreover have full-ownership (get-fh H'')  $\wedge$  h' = FractionalHeap.normalize

```

$(\text{get-fh } H'')$
proof –
obtain x **where** $\text{Some } x = \text{Some } (\text{add-uguard-to-no-guard index } hq' \text{ } (?l \ s')) \oplus \text{Some } hj'$
by $(\text{metis calculation not-Some-eq plus.simps}(1))$
then have $\text{get-fh } H'' = \text{add-fh } (\text{add-fh } (\text{get-fh } (\text{add-uguard-to-no-guard index } hq' \text{ } (?l \ s')))) (\text{get-fh } hj') (\text{get-fh } hf)$
by $(\text{metis add-get-fh calculation})$
moreover have $\text{get-fh } (\text{add-uguard-to-no-guard index } hq' \text{ } (?l \ s')) = \text{get-fh } hq' \wedge \text{get-fh } hf = \text{get-fh } (\text{remove-guards } hf)$
by $(\text{metis get-fh-add-uguard get-fh-remove-guards})$
ultimately show $?thesis$
by $(\text{metis } \langle \text{Some } h1 = \text{Some } hq' \oplus \text{Some } hj' \rangle \text{ add-get-fh asm3}(1) \text{ asm3}(3) \text{ asm3}(4))$
qed
moreover have $\text{sat-inv pre-s'} \ hj' \ \Gamma$
proof $(\text{rule sat-inv-agrees})$
show $\text{sat-inv } s' \ hj' \ \Gamma$
by $(\text{simp add: } \langle \text{sat-inv } s' \ hj' \ \Gamma \rangle)$
show $\text{agrees } (fvA \ (\text{invariant } \Gamma)) \ s' \ \text{pre-s}'$
using $\text{UnCI } \langle \text{agrees } (- \ \{x\}) \ s' \ \text{pre-s}' \rangle \text{ assms}(1) \text{ assms}(5) \text{ select-convs}(5) \text{ subset-Compl-singleton}$
by $(\text{metis agrees-union sup.orderE})$
qed
moreover have $\text{safe } n \ (\text{Some } \Gamma) \ C' \ (\text{pre-s}', \text{ add-uguard-to-no-guard index } hq' \text{ } (?l \ s')) \ (? \Sigma' \ (\text{pre-s}, h))$
proof $(\text{rule safe-free-vars-Some})$
show $\text{safe } n \ (\text{Some } \Gamma) \ C' \ (s', \text{ add-uguard-to-no-guard index } hq' \text{ } (?l \ s')) \ (? \Sigma' \ (\text{pre-s}, h))$
by $(\text{meson } \langle \text{safe } n \ (\text{Some } \Gamma) \ C' \ (s', \text{ add-uguard-to-no-guard index } hq' \text{ } (?l \ s')) \ (? \Sigma' \ (\text{pre-s}, h)) \rangle \text{ map-to-arg } (s' \ \text{uarg}) \ \# \ \text{map-to-list } (s' \ l)) \ (\Sigma' \ (\text{pre-s}, h)) \ \text{close-var-subset safe-larger-set})$
show $\text{agrees } (fvC \ C' \cup (- \ \{x\}) \cup fvA \ (\text{invariant } \Gamma)) \ s' \ \text{pre-s}'$
by $(\text{metis UnI2 Un-absorb1 } \langle \text{agrees } (- \ \{x\}) \ s' \ \text{pre-s}' \rangle \text{ asm3}(2) \text{ assms}(1) \text{ assms}(5) \text{ empty-iff fvC.simps}(1) \text{ inf-sup-aci}(5) \text{ select-convs}(5) \text{ subset-Compl-singleton})$
show $\text{upper-fvs } (\text{close-var } (\Sigma' \ (\text{pre-s}, h)) \ x) \ (- \ \{x\})$
by $(\text{simp add: upper-fvs-close-vars})$
qed
ultimately show $\exists h'' \ H' \ hj'$.
 $\text{full-ownership } (\text{get-fh } H') \wedge$
 $\text{semi-consistent } \Gamma \ v0 \ H' \wedge$
 $\text{sat-inv pre-s'} \ hj' \ \Gamma \wedge h' = \text{FractionalHeap.normalize } (\text{get-fh } H') \wedge \text{Some } H' = \text{Some } h'' \oplus \text{Some } hj' \oplus \text{Some } hf$
 $\wedge \text{safe } n \ (\text{Some } \Gamma) \ C' \ (\text{pre-s}', h'') \ (? \Sigma' \ (\text{pre-s}, h))$ **using** $\langle \text{sat-inv } s' \ hj' \ \Gamma \rangle$
by blast
qed
ultimately show $\text{safe } k \ (\text{Some } \Gamma) \ (\text{Catomic } C) \ (\text{pre-s}, h) \ (? \Sigma' \ (\text{pre-s}, h))$
by blast
qed (simp)

qed
qed

theorem *atomic-rule-shared*:

fixes $\Gamma :: ('i, 'a, nat)$ *single-context*

fixes *map-to-multiset* $:: nat \Rightarrow 'a$ *multiset*

fixes *map-to-arg* $:: nat \Rightarrow 'a$

assumes $\Gamma = (\mid \text{view} = f, \text{abstract-view} = \alpha, \text{saction} = \text{sact}, \text{uaction} = \text{uact},$
invariant $= J \mid)$

and *hoare-triple-valid* ($None :: ('i, 'a, nat)$ *cont*) ($Star P (View f J (\lambda s. s$
 $x))) C$

($Star Q (View f J (\lambda s. \text{sact} (s x) (\text{map-to-arg} (s \text{sarg}))))$)

and *precise* $J \wedge \text{unary } J$

and *view-function-of-inv* Γ

and $x \notin \text{fv} C \ C \cup \text{fv} A \ P \cup \text{fv} A \ Q \cup \text{fv} A \ J$

and $\text{sarg} \notin \text{fv} C \ C$

and $\text{ms} \notin \text{fv} C \ C$

and $x \notin \text{fv} S (\lambda s. \text{map-to-multiset} (s \text{ms}))$

and $x \notin \text{fv} S (\lambda s. \{\# \text{map-to-arg} (s \text{sarg}) \# \} + \text{map-to-multiset} (s \text{ms}))$

and *no-guard-assertion* P

and *no-guard-assertion* Q

shows *hoare-triple-valid* ($Some \Gamma$) ($Star P (SharedGuard \pi (\lambda s. \text{map-to-multiset}$
 $(s \text{ms})))) (Catomic C)$

($Star Q (SharedGuard \pi (\lambda s. \{\# \text{map-to-arg} (s \text{sarg}) \# \} + \text{map-to-multiset}$
 $(s \text{ms}))))$)

proof –

let $?J = View f J (\lambda s. s x)$

let $?J' = View f J (\lambda s. \text{sact} (s x) (\text{map-to-arg} (s \text{sarg})))$

let $?pre\text{-}ms = \lambda s. \text{map-to-multiset} (s \text{ms})$

let $?G = SharedGuard \pi ?pre\text{-}ms$

let $?ms = \lambda s. \{\# \text{map-to-arg} (s \text{sarg}) \# \} + \text{map-to-multiset} (s \text{ms})$

let $?G' = SharedGuard \pi ?ms$

have *unaries*: $\text{unary } ?J \wedge \text{unary } ?J'$

by (*simp add*: *assms*(\mathcal{J}) *unary-inv-then-view*)

moreover **have** *precises*: $\text{precise } ?J \wedge \text{precise } ?J'$

by (*simp add*: *assms*(\mathcal{J}) *precise-inv-then-view*)

obtain Σ **where** *asm0*: $\bigwedge n \ \sigma. \ \sigma, \sigma \models Star P ?J \implies \text{safe } n (None :: ('i, 'a,$
 $nat) \text{cont}) C \ \sigma (\Sigma \ \sigma)$

$\bigwedge \sigma \ \sigma'. \ \sigma, \sigma' \models Star P ?J \implies \text{pair-sat} (\Sigma \ \sigma) (\Sigma \ \sigma') (Star Q ?J')$

using *assms(2)* *hoare-triple-valid-def* **by** *blast*

define *start* **where** *start* = $(\lambda\sigma. \{ (s, h) \mid s \ h \ hj. \text{ agrees } (- \{x\}) \ (fst \ \sigma) \ s \wedge \text{ Some } h = \text{ Some } (\text{remove-guards } (snd \ \sigma)) \oplus \text{ Some } hj \wedge (s, hj), (s, hj) \models ?J \})$

define *end-qj* **where** *end-qj* = $(\lambda\sigma. \bigcup \sigma' \in \text{start } \sigma. \Sigma \ \sigma')$

define Σ' **where** $\Sigma' = (\lambda\sigma. \{ (s, \text{add-sguard-to-no-guard } hq \ \pi \ (?ms \ s)) \mid s \ hq \ h \ hj. (s, h) \in \text{end-qj } \sigma \wedge \text{ Some } h = \text{ Some } hq \oplus \text{ Some } hj \wedge (s, hj), (s, hj) \models ?J' \})$

let $? \Sigma' = \lambda\sigma. \text{close-var } (\Sigma' \ \sigma) \ x$

show *hoare-triple-valid* (*Some* Γ) (*Star* *P* $?G$) (*Catomic* *C*) (*Star* *Q* $?G'$)

proof (*rule* *hoare-triple-validI*)

show $\bigwedge s \ h \ s' \ h'. (s, h), (s', h') \models \text{Star } P \ ?G \implies \text{pair-sat } (? \Sigma' (s, h)) (? \Sigma' (s', h')) (\text{Star } Q \ ?G')$

proof –

fix *s1* *h1* *s2* *h2*

assume *asm1*: $(s1, h1), (s2, h2) \models \text{Star } P \ ?G$

then obtain *p1* *p2* *g1* *g2* **where** *r0*: $\text{Some } h1 = \text{Some } p1 \oplus \text{Some } g1$
 $\text{Some } h2 = \text{Some } p2 \oplus \text{Some } g2$

$(s1, p1), (s2, p2) \models P \ (s1, g1), (s2, g2) \models ?G$

using *hyper-sat.simps(4)* **by** *auto*

then obtain *remove-guards* *h1* = *p1* *remove-guards* *h2* = *p2*

using *assms(10)* *hyper-sat.simps(12)* *no-guard-and-no-heap* *no-guard-assertion-def* **by** *metis*

have $\text{pair-sat } (\Sigma' (s1, h1)) (\Sigma' (s2, h2)) (\text{Star } Q \ ?G')$

proof (*rule* *pair-satI*)

fix *s1'* *hqq1* *s2'* *hqq2* $\sigma 2'$

assume *asm2*: $(s1', hqq1) \in \Sigma' (s1, h1) \wedge (s2', hqq2) \in \Sigma' (s2, h2)$

then obtain *h1'* *hj1'* *h2'* *hj2'* *hq1* *hq2* **where** *r*: $(s1', h1') \in \text{end-qj } (s1, h1)$ $\text{Some } h1' = \text{Some } hq1 \oplus \text{Some } hj1'$
 $(s1', hj1'), (s1', hj1') \models ?J' (s2', h2') \in \text{end-qj } (s2, h2)$ $\text{Some } h2' = \text{Some } hq2 \oplus \text{Some } hj2' (s2', hj2'), (s2', hj2') \models ?J'$
 $hqq1 = \text{add-sguard-to-no-guard } hq1 \ \pi \ (?ms \ s1') \ hqq2 = \text{add-sguard-to-no-guard } hq2 \ \pi \ (?ms \ s2')$

using Σ' -*def* **by** *blast*

then obtain $\sigma 1' \ \sigma 2'$ **where** $\sigma 1' \in \text{start } (s1, h1) \ \sigma 2' \in \text{start } (s2, h2) (s1', h1') \in \Sigma \ \sigma 1' (s2', h2') \in \Sigma \ \sigma 2'$

using *end-qj-def* **by** *blast*

then obtain *hj1* *hj2* **where** $\text{agrees } (- \{x\}) \ s1 \ (fst \ \sigma 1') \ \text{Some } (snd \ \sigma 1') = \text{Some } p1 \oplus \text{Some } hj1 \ (fst \ \sigma 1', hj1), (fst \ \sigma 1', hj1) \models ?J$
 $\text{agrees } (- \{x\}) \ s2 \ (fst \ \sigma 2') \ \text{Some } (snd \ \sigma 2') = \text{Some } p2 \oplus \text{Some } hj2 \ (fst \ \sigma 2', hj2), (fst \ \sigma 2', hj2) \models ?J$

using *start-def* $\langle \text{remove-guards } h1 = p1 \rangle \langle \text{remove-guards } h2 = p2 \rangle$ **by** *force*

moreover have $(fst \ \sigma 1', hj1), (fst \ \sigma 2', hj2) \models ?J$

using *calculation(3)* *calculation(6)* *unaries* *unaryE* **by** *blast*

moreover have $(fst \ \sigma 1', p1), (fst \ \sigma 2', p2) \models P$

```

proof –
  have  $fvA\ P \subseteq -\ \{x\}$ 
    using assms(5) by force
  then have  $agrees\ (fvA\ P)\ (fst\ \sigma 1')\ s1 \wedge agrees\ (fvA\ P)\ (fst\ \sigma 2')\ s2$ 
    using calculation(1) calculation(4)
    by (metis agrees-comm agrees-union subset-Un-eq)
  then show ?thesis using r0(3)
    by (meson agrees-same sat-comm)
qed

  ultimately have  $\sigma 1', \sigma 2' \models Star\ P\ ?J$  using hyper-sat.simps(4) [of fst  $\sigma 1'$ 
snd  $\sigma 1'$  fst  $\sigma 2'$  snd  $\sigma 2'$ ] prod.collapse
    by metis
  then have  $pair\text{-}sat\ (\Sigma\ \sigma 1')\ (\Sigma\ \sigma 2')\ (Star\ Q\ ?J')$ 
    using asm0(2) [of  $\sigma 1'$   $\sigma 2'$ ] by blast
  then have  $\langle s1', h1' \rangle, \langle s2', h2' \rangle \models Star\ Q\ ?J'$ 
    using  $\langle s1', h1' \rangle \in \Sigma\ \sigma 1' \wedge \langle s2', h2' \rangle \in \Sigma\ \sigma 2'$  pair-sat-def by blast
  moreover have  $\langle s1', hj1' \rangle, \langle s2', hj2' \rangle \models ?J'$ 
    using r(3) r(6) unaries unaryE by blast
  moreover have  $\langle s1', hq1 \rangle, \langle s2', hq2 \rangle \models Q$  using magic-lemma
    using calculation(1) calculation(2) precises r(2) r(5) by blast
  moreover have no-guard hq1  $\wedge$  no-guard hq2
    using assms(11) calculation(3) no-guard-assertion-def by blast
  ultimately show  $\langle s1', hqg1 \rangle, \langle s2', hqg2 \rangle \models Star\ Q\ ?G'$ 
    using no-guard-then-sat-star r(7) r(8)
    by (metis (mono-tags, lifting))
qed
  then show  $pair\text{-}sat\ (? \Sigma'\ (s1, h1))\ (? \Sigma'\ (s2, h2))\ (Star\ Q\ ?G')$ 
proof (rule pair-sat-close-var-double)
    show  $x \notin fvA\ (Star\ Q\ (SharedGuard\ \pi\ (\lambda s. \{\#map\text{-}to\text{-}arg\ (s\ sarg)\ \#\} +$ 
map-to-multiset  $(s\ ms))))$ 
      using assms(5) assms(9) by auto
qed
qed

fix pre-s h k
assume  $(pre\text{-}s, h), (pre\text{-}s, h) \models Star\ P\ ?G$ 
then obtain pp gg where  $Some\ h = Some\ pp \oplus Some\ gg\ (pre\text{-}s, pp), (pre\text{-}s,$ 
pp)  $\models P\ (pre\text{-}s, gg), (pre\text{-}s, gg) \models ?G$ 
    using always-sat-refl hyper-sat.simps(4) by blast
then have remove-guards h = pp
by (meson assms(10) hyper-sat.simps(12) no-guard-and-no-heap no-guard-assertion-def)
then have  $(pre\text{-}s, remove\text{-}guards\ h), (pre\text{-}s, remove\text{-}guards\ h) \models P$ 
    using  $\langle pre\text{-}s, pp \rangle, \langle pre\text{-}s, pp \rangle \models P$  hyper-sat.simps(9) by blast
then have  $(pre\text{-}s, remove\text{-}guards\ h), (pre\text{-}s, remove\text{-}guards\ h) \models P$ 
by (simp add: no-guard-remove-guards)

show  $safe\ k\ (Some\ \Gamma)\ (Catomic\ C)\ (pre\text{-}s, h)\ (? \Sigma'\ (pre\text{-}s, h))$ 
proof (cases k)

```

```

case (Suc n)
moreover have safe (Suc n) (Some  $\Gamma$ ) (Catomic C) (pre-s, h) (? $\Sigma'$  (pre-s,
h))
proof (rule safeSomeAltI)
  show Catomic C = Cskip  $\implies$  (pre-s, h)  $\in$  ? $\Sigma'$  (pre-s, h) by simp

  fix H hf hj v0

  assume asm2: Some H = Some h  $\oplus$  Some hj  $\oplus$  Some hf  $\wedge$  full-ownership
(get-fh H)  $\wedge$  semi-consistent  $\Gamma$  v0 H  $\wedge$  sat-inv pre-s hj  $\Gamma$ 

  define v where v = f (normalize (get-fh H))
  define s where s = pre-s(x := v)
  then have v = s x by simp
  moreover have agreements: agrees (fvC C  $\cup$  fvA P  $\cup$  fvA Q  $\cup$  fvA J  $\cup$ 
fvA (SharedGuard  $\pi$  ( $\lambda$ s. map-to-multiset (s ms)))) s pre-s
    by (metis (mono-tags, lifting) Un-iff agrees-def assms(5) assms(8)
fun-upd-other fvA.simps(8) s-def)
  then have asm1: (s, h), (s, h)  $\models$  Star P ?G
— 10s
  by (metis (mono-tags, lifting)  $\langle$ (pre-s, h), (pre-s, h)  $\models$  Star P (SharedGuard
 $\pi$  ( $\lambda$ s. map-to-multiset (s ms))) $\rangle$  agrees-same agrees-union fvA.simps(3) fvA.simps(8)
sat-comm)
  moreover have asm2-bis: sat-inv s hj  $\Gamma$ 
  proof (rule sat-inv-agrees)
    show sat-inv pre-s hj  $\Gamma$  using asm2 by simp
    show agrees (fvA (invariant  $\Gamma$ )) pre-s s
      using assms(1) assms(5) s-def
    by (simp add: agrees-update)
  qed
  moreover have (s, remove-guards h), (s, remove-guards h)  $\models$  P
    by (meson  $\langle$ (pre-s, remove-guards h), (pre-s, remove-guards h)  $\models$  P $\rangle$ 
agreements agrees-same agrees-union always-sat-refl)
  then have (s, remove-guards h), (s, remove-guards h)  $\models$  P
    by (simp add: no-guard-remove-guards)

  moreover have agrees ( $-$  {x}) pre-s s
  proof (rule agreesI)
    fix y assume y  $\in$   $-$  {x}
    then have y  $\neq$  x
      by force
    then show pre-s y = s y
      by (simp add: s-def)
  qed

  moreover obtain (s, pp), (s, pp)  $\models$  P (s, gg), (s, gg)  $\models$  ?G
    using  $\langle$ (pre-s, gg), (pre-s, gg)  $\models$  SharedGuard  $\pi$  ( $\lambda$ s. map-to-multiset (s
ms)) $\rangle$   $\langle$ remove-guards h = pp $\rangle$  agreements agrees-same agrees-union always-sat-refl-aux
calculation(4) by blast

```

```

let ?hf = remove-guards hf
let ?H = remove-guards H
let ?h = remove-guards h

obtain hhj where Some hhj = Some h  $\oplus$  Some hj
  by (metis asm2 plus.simps(2) plus.simps(3) plus-comm)
then have Some H = Some hhj  $\oplus$  Some hf
  using asm2 by presburger
then have Some (remove-guards hhj) = Some ?h  $\oplus$  Some hj
  by (metis <Some hhj = Some h  $\oplus$  Some hj> asm2 no-guards-remove
remove-guards-sum sat-inv-def)

moreover have f (normalize (get-fh hj)) = v
proof –
  have view  $\Gamma$  (normalize (get-fh hj)) = view  $\Gamma$  (normalize (get-fh H))
  using assms(4) view-function-of-invE
  by (metis (no-types, opaque-lifting) <Some hhj = Some h  $\oplus$  Some hj>
asm2 larger-def larger-trans plus-comm)
  then show ?thesis using assms(1) v-def by fastforce
qed

then have (s, hj), (s, hj)  $\models$  ?J
  by (metis <v = s x> asm2-bis assms(1) hyper-sat.simps(11) sat-inv-def
select-convs(5))

ultimately have (s, remove-guards hhj), (s, remove-guards hhj)  $\models$  Star P
?J
using <(s, remove-guards h), (s, remove-guards h)  $\models$  P> hyper-sat.simps(4)
by blast

then have all-safes:  $\bigwedge n$ . safe n (None :: ('i, 'a, nat) cont) C (s, re-
move-guards hhj) ( $\Sigma$  (s, remove-guards hhj))
  using asm0(1) by blast
then have  $\bigwedge \sigma 1 H1 \sigma 2 H2 s2 C2$ . red-rtrans C  $\sigma 1 C2 \sigma 2 \implies \sigma 1 = (s,$ 
H1)  $\implies \sigma 2 = (s2, H2) \implies$ 
  ?H = denormalize H1  $\implies$ 
 $\neg$  aborts C2  $\sigma 2 \wedge (C2 = Cskip \longrightarrow (\exists h1 H'. \text{Some } H' = \text{Some } h1 \oplus \text{Some } ?hf$ 
 $\wedge H2 = \text{FractionalHeap.normalize (get-fh } H')$ 
 $\wedge \text{no-guard } H' \wedge \text{full-ownership (get-fh } H') \wedge (s2, h1) \in \Sigma (s, \text{remove-guards}$ 
hhj)))
proof –
  fix  $\sigma 1 H1 \sigma 2 H2 s2 C2$ 
  assume ?H = denormalize H1
  assume red-rtrans C  $\sigma 1 C2 \sigma 2 \sigma 1 = (s, H1) \sigma 2 = (s2, H2)$ 

then show  $\neg$  aborts C2  $\sigma 2 \wedge$ 

```

$(C2 = Cskip \longrightarrow$
 $(\exists h1 H'.$
 $\text{Some } H' = \text{Some } h1 \oplus \text{Some } (\text{remove-guards } hf) \wedge$
 $H2 = \text{FractionalHeap.normalize } (\text{get-fh } H') \wedge \text{no-guard } H' \wedge \text{full-ownership}$
 $(\text{get-fh } H') \wedge (s2, h1) \in \Sigma (s, \text{remove-guards } hhj)))$
using *all-safes*
proof (*rule safe-atomic*)
show $?H = \text{denormalize } H1$ **using** $\langle ?H = \text{denormalize } H1 \rangle$ **by** *simp*
show $\text{Some } ?H = \text{Some } (\text{remove-guards } hhj) \oplus \text{Some } ?hf$
using $\langle \text{Some } H = \text{Some } hhj \oplus \text{Some } hf \rangle$ *remove-guards-sum* **by** *blast*
show $\text{full-ownership } (\text{get-fh } (\text{remove-guards } H)) \wedge \text{no-guard } (\text{remove-guards}$
 $H)$
by (*metis asm2 get-fh-remove-guards no-guard-remove-guards*)
qed
moreover have $?H = \text{denormalize } (\text{normalize } (\text{get-fh } H))$
by (*metis asm2 denormalize-properties(5)*)
ultimately have *safe-atomic-simplified*: $\bigwedge \sigma2 H2 s2 C2. \text{red-rtrans } C (s,$
 $\text{normalize } (\text{get-fh } H)) C2 \sigma2$
 $\implies \sigma2 = (s2, H2) \implies \neg \text{aborts } C2 \sigma2 \wedge (C2 = Cskip \longrightarrow (\exists h1 H'. \text{Some}$
 $H' = \text{Some } h1 \oplus \text{Some } ?hf \wedge H2 = \text{FractionalHeap.normalize } (\text{get-fh } H')$
 $\wedge \text{no-guard } H' \wedge \text{full-ownership } (\text{get-fh } H') \wedge (s2, h1) \in \Sigma (s, \text{remove-guards}$
 $hhj)))$
by *presburger*

have $\neg \text{aborts } (\text{Catomic } C) (s, \text{normalize } (\text{get-fh } H))$
proof (*rule ccontr*)
assume $\neg \neg \text{aborts } (\text{Catomic } C) (s, \text{normalize } (\text{get-fh } H))$
then obtain $C' \sigma'$ **where** *asm3*: $\text{red-rtrans } C (s, \text{FractionalHeap.normalize}$
 $(\text{get-fh } H)) C' \sigma'$
 $\text{aborts } C' \sigma'$
using *abort-atomic-cases* **by** *blast*
then have $\neg \text{aborts } C' \sigma'$ **using** *safe-atomic-simplified*[*of* $C' \sigma'$ *fst* σ' *snd*
 σ'] **by** *simp*
then show *False* **using** *asm3(2)* **by** *simp*
qed
then show $\neg \text{aborts } (\text{Catomic } C) (\text{pre-s}, \text{normalize } (\text{get-fh } H))$
by (*metis agreements aborts-agrees agrees-comm agrees-union fst-eqD*
 $\text{fvC.simps(11) snd-conv}$)

fix $C' \text{pre-s}' h'$
assume $\text{red } (\text{Catomic } C) (\text{pre-s}, \text{FractionalHeap.normalize } (\text{get-fh } H)) C'$
 $(\text{pre-s}', h')$
then obtain s' **where** $\text{red } (\text{Catomic } C) (s, \text{FractionalHeap.normalize } (\text{get-fh}$
 $H)) C' (s', h')$
 $\text{agrees } (- \{x\}) s' \text{pre-s}'$
by (*metis (no-types, lifting) UnI1 <agrees (- {x}) pre-s s> agrees-comm*
 $\text{assms(5) fst-eqD fvC.simps(11) red-agrees snd-conv subset-Compl-singleton}$)

then obtain $h1\ H'$ **where** $asm3: Some\ H' = Some\ h1 \oplus Some\ (remove-guards\ hf)\ C' = Cskip$
 $h' = FractionalHeap.normalize\ (get-fh\ H')\ no-guard\ H' \wedge full-ownership\ (get-fh\ H')\ (s',\ h1) \in \Sigma\ (s,\ remove-guards\ hhj)$
using $safe-atomic-simplified[of\ C'\ (s',\ h')\ s'\ h']$ **by** $(metis\ red-atomic-cases)$

moreover have $s\ x = s'\ x \wedge s\ sarg = s'\ sarg \wedge s\ ms = s'\ ms$ **using**
 $red-not-in-fv-not-touched$
using $\langle red\ (Catomic\ C)\ (s,\ FractionalHeap.normalize\ (get-fh\ H))\ C'\ (s',\ h') \rangle$
by $(metis\ UnI1\ assms(5)\ assms(6)\ assms(7)\ fst-eqD\ fvC.simps(11))$

have $\exists\ hq'\ hj'. Some\ h1 = Some\ hq' \oplus Some\ hj' \wedge (s',\ add-sguard-to-no-guard\ hq'\ \pi\ (?ms\ s')) \in \Sigma'\ (pre-s,\ h)$
 $\wedge\ sat-inv\ s'\ hj'\ \Gamma \wedge f\ (normalize\ (get-fh\ hj')) = sact\ v\ (map-to-arg\ (s'\ sarg))$
proof –
have $pair-sat\ (\Sigma\ (s,\ remove-guards\ hhj))\ (\Sigma\ (s,\ remove-guards\ hhj))\ (Star\ Q\ ?J')$
using $asm0(2)[of\ (s,\ remove-guards\ hhj)\ (s,\ remove-guards\ hhj)]$
using $\langle (s,\ remove-guards\ hhj),\ (s,\ remove-guards\ hhj) \models Star\ P\ ?J \rangle$ **by**
 $blast$

then have $(s',\ h1),\ (s',\ h1) \models Star\ Q\ ?J'$
using $asm3(5)\ pair-sat-def$ **by** $blast$
then obtain $hq'\ hj'$ **where** $Some\ h1 = Some\ hq' \oplus Some\ hj'\ (s',\ hq'),$
 $(s',\ hq') \models Q\ (s',\ hj'),\ (s',\ hj') \models ?J'$
using $always-sat-refl\ hyper-sat.simps(4)$ **by** $blast$
then have $no-guard\ hj'$
by $(metis\ (no-types,\ opaque-lifting)\ calculation(1)\ calculation(4)\ no-guard-then-smaller-same-plus-comm)$

moreover have $f\ (normalize\ (get-fh\ hj')) = sact\ v\ (map-to-arg\ (s'\ sarg))$
using $\langle (s',\ hj'),\ (s',\ hj') \models View\ f\ J\ (\lambda s.\ sact\ (s\ x)\ (map-to-arg\ (s\ sarg))) \rangle$
 $\langle s\ x = s'\ x \wedge s\ sarg = s'\ sarg \wedge s\ ms = s'\ ms \rangle \langle v = s\ x \rangle$ **by** $fastforce$
moreover have $(s,\ remove-guards\ hhj) \in start\ (pre-s,\ h)$
proof –
have $Some\ (remove-guards\ hhj) = Some\ ?h \oplus Some\ hj$
using $\langle Some\ (remove-guards\ hhj) = Some\ (remove-guards\ h) \oplus Some\ hj \rangle$ **by** $blast$

moreover have $(s,\ hj),\ (s,\ hj) \models ?J$
using $\langle (s,\ hj),\ (s,\ hj) \models ?J \rangle$ **by** $fastforce$
ultimately show $?thesis$ **using** $start-def$
using $\langle agrees\ (-\ \{x\})\ pre-s\ s \rangle$ **by** $fastforce$

qed
then have $(s',\ h1) \in end-qj\ (pre-s,\ h)$
using $\langle end-qj \equiv \lambda \sigma.\ \bigcup\ (\Sigma\ 'start\ \sigma) \rangle\ asm3(5)$ **by** $blast$

then have $(s',\ add-sguard-to-no-guard\ hq'\ \pi\ (?ms\ s')) \in \Sigma'\ (pre-s,\ h)$
using $\Sigma'-def\ \langle (s',\ hj'),\ (s',\ hj') \models ?J' \rangle \langle Some\ h1 = Some\ hq' \oplus Some\ hj' \rangle$ **by** $blast$
ultimately show $\exists\ hq'\ hj'.$

$\text{Some } h1 = \text{Some } hq' \oplus \text{Some } hj' \wedge$
 $(s', \text{add-sguard-to-no-guard } hq' \pi (\{\#\text{map-to-arg } (s' \text{ sarg})\#\} + \text{map-to-multiset}$
 $(s' \text{ ms}))) \in \Sigma' (\text{pre-s}, h) \wedge$
 $\text{sat-inv } s' hj' \Gamma \wedge f (\text{FractionalHeap.normalize } (\text{get-fh } hj')) = \text{sact } v (\text{map-to-arg}$
 $(s' \text{ sarg}))$
using $\langle (s', hj'), (s', hj') \models \text{View } f J (\lambda s. \text{sact } (s x) (\text{map-to-arg } (s \text{ sarg}))) \rangle$
 $\langle \text{Some } h1 = \text{Some } hq' \oplus \text{Some } hj' \rangle \text{ assms}(1) \text{ sat-inv-def by fastforce}$
qed
then obtain $hq' hj'$ **where** $\text{Some } h1 = \text{Some } hq' \oplus \text{Some } hj' (s',$
 $\text{add-sguard-to-no-guard } hq' \pi (?ms \text{ s}')) \in \Sigma' (\text{pre-s}, h) \text{ sat-inv } s' hj' \Gamma$
 $f (\text{FractionalHeap.normalize } (\text{get-fh } hj')) = \text{sact } v (\text{map-to-arg } (s' \text{ sarg}))$
by blast
then have $\text{safe } n (\text{Some } \Gamma) C' (s', \text{add-sguard-to-no-guard } hq' \pi (?ms \text{ s}'))$
 $(\Sigma' (\text{pre-s}, h))$
using $\text{asm3}(2) \text{ safe-skip by blast}$

moreover have $\exists H''. \text{ semi-consistent } \Gamma v0 H'' \wedge \text{Some } H'' = \text{Some}$
 $(\text{add-sguard-to-no-guard } hq' \pi (?ms \text{ s}')) \oplus \text{Some } hj' \oplus \text{Some } hf$
proof –
have $\text{Some } (\text{add-sguard-to-no-guard } hq' \pi (?ms \text{ s}')) = \text{Some } hq' \oplus \text{Some}$
 $(\text{Map.empty}, \text{Some } (\pi, ?ms \text{ s}'), (\lambda-. \text{None}))$
using $\langle \text{Some } h1 = \text{Some } hq' \oplus \text{Some } hj' \rangle \text{ add-sguard-as-sum asm3}(1)$
 $\text{asm3}(4) \text{ no-guard-then-smaller-same by blast}$

obtain hhf **where** $\text{Some } hhf = \text{Some } h \oplus \text{Some } hf$
by $(\text{metis } (\text{no-types}, \text{opaque-lifting}) \langle \text{Some } H = \text{Some } hhj \oplus \text{Some}$
 $hf \rangle \langle \text{Some } hhj = \text{Some } h \oplus \text{Some } hj \rangle \text{ option.exhaust-sel plus.simps}(1) \text{ plus-asso}$
 $\text{plus-comm})$
then have $\text{all-guards } hhf$
by $(\text{metis } (\text{no-types}, \text{lifting}) \text{ all-guards-no-guard-propagates asm2 plus-asso}$
 $\text{plus-comm sat-inv-def semi-consistent-def})$

moreover have $\text{get-gu } h = (\lambda-. \text{None}) \wedge \text{get-gs } h = \text{Some } (\pi, ?pre-ms \text{ s})$
proof –
have $\text{no-guard } pp$
using $\langle (\text{pre-s}, pp), (\text{pre-s}, pp) \models P \rangle \text{ assms}(10) \text{ no-guard-assertion-def}$
by blast
then show $?thesis$
by $(\text{metis } \langle \text{Some } h = \text{Some } pp \oplus \text{Some } gg \rangle \langle \wedge \text{thesis. } (\llbracket (s, pp), (s, pp) \models$
 $P; (s, gg), (s, gg) \models \text{SharedGuard } \pi (\lambda s. \text{map-to-multiset } (s \text{ ms})) \rrbracket \implies \text{thesis}) \implies$
 $\text{thesis} \rangle \langle \text{remove-guards } h = pp \rangle \text{ decompose-heap-triple fst-conv hyper-sat.simps}(12)$
 $\text{no-guard-remove}(2) \text{ plus-comm remove-guards-def snd-conv sum-gs-one-none})$
qed
then have $\exists \pi' \text{ msf } uargs. (\forall k. \text{get-gu } hf k = \text{Some } (uargs k)) \wedge$
 $(\pi = \text{pwrite} \wedge \text{get-gs } hf = \text{None} \wedge \text{msf} = \{\#\} \vee \text{pwrite} = \text{padd } \pi \pi'$
 $\wedge \text{get-gs } hf = \text{Some } (\pi', \text{msf}))$
using $\text{all-guards-sum-known-one}[of \text{hhf } h \text{ hf } \pi]$
using $\langle \text{Some } hhf = \text{Some } h \oplus \text{Some } hf \rangle \text{ calculation by fastforce}$

then obtain $\pi' \text{ uargs msf}$ **where** $(\forall k. \text{get-gu hf } k = \text{Some } (\text{uargs } k)) \wedge$
 $((\pi = \text{pwrite} \wedge \text{get-gs hf} = \text{None} \wedge \text{msf} = \{\#\}) \vee (\text{pwrite} = \text{padd } \pi \ \pi' \wedge \text{get-gs}$
 $\text{hf} = \text{Some } (\pi', \text{msf})))$
by *blast*

then obtain ghf **where** $ghf\text{-def}: \text{Some hf} = \text{Some } (\text{remove-guards hf}) \oplus$
 Some ghf
 $\text{get-fh ghf} = \text{Map.empty } (\pi = \text{pwrite} \wedge \text{get-gs ghf} = \text{None} \wedge \text{msf} = \{\#\})$
 $\vee (\text{padd } \pi \ \pi' = \text{pwrite} \wedge \text{get-gs ghf} = \text{Some } (\pi', \text{msf}))$
 $\wedge i. \text{get-gu ghf } i = \text{Some } (\text{uargs } i)$
using *decompose-guard-remove-easy*[of hf]
by $(\text{metis fst-conv get-fh.elims get-gs.elims get-gu.simps snd-conv})$

have $(\text{Map.empty}, \text{Some } (\pi, ?ms \ s'), (\lambda-. \text{None})) \#\# ghf$
proof $(\text{rule compatibleI})$
show $\text{compatible-fract-heaps } (\text{get-fh } (\text{Map.empty}, \text{Some } (\pi, ?ms \ s'), (\lambda-. \text{None})))$
 $(\text{get-fh } ghf)$
using *compatible-fract-heapsI* **by** *fastforce*
show $\wedge k. \text{get-gu } (\text{Map.empty}, \text{Some } (\pi, \{\#\text{map-to-arg } (s' \ \text{sarg})\#\}) +$
 $\text{map-to-multiset } (s' \ ms)), \text{Map.empty}) \ k = \text{None} \vee \text{get-gu ghf } k = \text{None}$
by *simp*
fix $p \ p'$
assume $\text{get-gs } (\text{Map.empty}, \text{Some } (\pi, \{\#\text{map-to-arg } (s' \ \text{sarg})\#\}) +$
 $\text{map-to-multiset } (s' \ ms)), \text{Map.empty}) = \text{Some } p \wedge \text{get-gs ghf} = \text{Some } p'$
then have $p = (\pi, ?ms \ s') \wedge p' = (\pi', \text{msf}) \wedge \text{padd } \pi \ \pi' = \text{pwrite}$
using $ghf\text{-def}$ **by** *auto*
then show $\text{pgte pwrite } (\text{padd } (\text{fst } p) \ (\text{fst } p'))$
using *not-pgte-charact pgt-implies-pgte* **by** *auto*
qed
then obtain g **where** $g\text{-def}: \text{Some } g = \text{Some } (\text{Map.empty}, \text{Some } (\pi, ?ms$
 $s'), (\lambda-. \text{None})) \oplus \text{Some ghf}$
by *simp*
moreover have $H' \#\# g$
proof $(\text{rule compatibleI})$
have $\text{get-fh } g = \text{add-fh } \text{Map.empty } \text{Map.empty}$ **using** *add-get-fh*[of g
 $(\text{Map.empty}, \text{Some } (\pi, ?ms \ s'), (\lambda-. \text{None})) \ ghf]$
 $g\text{-def } \langle \text{get-fh } ghf = \text{Map.empty} \rangle$
by *fastforce*
then have $\text{get-fh } g = \text{Map.empty}$
using *add-fh-map-empty* **by** *auto*
then show $\text{compatible-fract-heaps } (\text{get-fh } H') \ (\text{get-fh } g)$
using *compatible-fract-heapsI* **by** *force*
show $\wedge k. \text{get-gu } H' \ k = \text{None} \vee \text{get-gu } g \ k = \text{None}$
by $(\text{meson asm3}(4) \ \text{no-guard-def})$
show $\wedge p \ p'. \text{get-gs } H' = \text{Some } p \wedge \text{get-gs } g = \text{Some } p' \implies \text{pgte pwrite}$
 $(\text{padd } (\text{fst } p) \ (\text{fst } p'))$
by $(\text{metis asm3}(4) \ \text{no-guard-def option.simps}(3))$

qed
then obtain H'' **where** $\text{Some } H'' = \text{Some } H' \oplus \text{Some } g$
by *simp*
then have $\text{Some } H'' = \text{Some } (\text{add-sguard-to-no-guard } hq' \pi \ (?ms \ s^{\wedge})) \oplus$
 $\text{Some } hj' \oplus \text{Some } hf$
proof –
have $\text{Some } H'' = \text{Some } h1 \oplus \text{Some } g \oplus \text{Some } (\text{remove-guards } hf)$
by (*metis* $\langle \text{Some } H'' = \text{Some } H' \oplus \text{Some } g \rangle \text{asm3}(1) \text{ plus-comm}$
simpler-asso)
moreover have $\text{Some } (\text{add-sguard-to-no-guard } hq' \pi \ (?ms \ s^{\wedge})) = \text{Some}$
 $hq' \oplus \text{Some } (\text{Map.empty}, \text{Some } (\pi, \ ?ms \ s^{\wedge}), (\lambda-. \ \text{None}))$
using $\langle \text{Some } (\text{add-sguard-to-no-guard } hq' \pi \ (\{\# \text{map-to-arg } (s'$
 $\text{sarg})\# \} + \text{map-to-multiset } (s' \ ms))) = \text{Some } hq' \oplus \text{Some } (\text{Map.empty}, \text{Some } (\pi,$
 $\{\# \text{map-to-arg } (s' \ \text{sarg})\# \} + \text{map-to-multiset } (s' \ ms)), (\lambda-. \ \text{None})) \rangle$ **by** *blast*
ultimately show *?thesis*
by (*metis* (*no-types, lifting*) $\langle \text{Some } h1 = \text{Some } hq' \oplus \text{Some } hj' \rangle$ *g-def*
ghf-def(1) *plus-comm simpler-asso*)
qed

moreover have *semi-consistent* $\Gamma \ v0 \ H''$
proof (*rule semi-consistentI*)
have *get-gs* $g = \text{Some } (\text{pwrite}, \ ?ms \ s' + \ \text{msf})$
proof (*cases* $\pi = \text{pwrite}$)
case *True*
then have $\pi = \text{pwrite} \wedge \text{get-gs } ghf = \text{None} \wedge \text{msf} = \{\#\}$ **using**
ghf-def(3)
by (*metis not-pgte-charact pgt-implies-pgte sum-larger*)
then show *?thesis*
by (*metis add.right-neutral fst-conv g-def get-gs.simps snd-conv*
sum-gs-one-none)
next
case *False*
then have $\text{padd } \pi \ \pi' = \text{pwrite} \wedge \text{get-gs } ghf = \text{Some } (\pi', \ \text{msf})$
using *ghf-def*(3) **by** *blast*
then show *?thesis*
by (*metis calculation*(2) *fst-conv get-gs.elims snd-conv sum-gs-one-some*)
qed

moreover have $\bigwedge i. \text{get-gu } g \ i = \text{Some } (\text{uargs } i)$
by (*metis full-uguard-sum-same ghf-def*(4) *g-def plus-comm*)
ultimately have *all-guards* g
using *all-guards-def* **by** *blast*
then show *all-guards* H''
by (*metis* $\langle \text{Some } H'' = \text{Some } H' \oplus \text{Some } g \rangle$ *all-guards-same plus-comm*)
show *reachable* $\Gamma \ v0 \ H''$
proof (*rule reachableI*)
fix *sargs* uargs'

assume $get\text{-}gs\ H'' = Some\ (pwrite, sargs) \wedge (\forall k. get\text{-}gu\ H''\ k = Some\ (uargs'\ k))$
then have $sargs = ?ms\ s' + msf$
by $(metis\ (no\text{-}types, opaque\text{-}lifting)\ \langle Some\ H'' = Some\ H' \oplus Some\ g \rangle\ \langle get\text{-}gs\ g = Some\ (pwrite, \{\#\text{map-to-arg}\ (s'\ sarg)\ \#\} + \text{map-to-multiset}\ (s'\ ms) + msf) \rangle\ \text{asm3}(4)\ no\text{-}guard\text{-}remove(1)\ \text{option.inject}\ plus\text{-}comm\ \text{snd}\text{-}conv)$
moreover have $uargs = uargs'$
apply $(rule\ ext)$
by $(metis\ \langle Some\ H'' = Some\ H' \oplus Some\ g \rangle\ \langle \bigwedge i. get\text{-}gu\ g\ i = Some\ (uargs\ i) \rangle\ \langle get\text{-}gs\ H'' = Some\ (pwrite, sargs) \wedge (\forall k. get\text{-}gu\ H''\ k = Some\ (uargs'\ k)) \rangle\ \text{asm3}(4)\ no\text{-}guard\text{-}remove(2)\ \text{option.sel}\ plus\text{-}comm)$
moreover have $view\ \Gamma\ (FractionalHeap.normalize\ (get\text{-}fh\ hj')) = view\ \Gamma\ (FractionalHeap.normalize\ (get\text{-}fh\ H''))$
using $assms(4)\ \langle sat\text{-}inv\ s'\ hj'\ \Gamma \rangle$
proof $(rule\ view\text{-}function\text{-}of\text{-}invE)$
show $H'' \succeq hj'$
by $(metis\ (no\text{-}types, opaque\text{-}lifting)\ \langle Some\ H'' = Some\ H' \oplus Some\ g \rangle\ \langle Some\ h1 = Some\ hq' \oplus Some\ hj' \rangle\ \text{asm3}(1)\ larger\text{-}def\ larger\text{-}trans\ plus\text{-}comm)$
qed
moreover have $reachable\text{-}value\ (saction\ \Gamma)\ (uaction\ \Gamma)\ v0\ (?ms\ s' + msf)\ uargs\ (sact\ v\ (\text{map-to-arg}\ (s'\ sarg)))$
proof –

have $reachable\text{-}value\ (saction\ \Gamma)\ (uaction\ \Gamma)\ v0\ (?pre\text{-}ms\ s + msf)\ uargs\ (view\ \Gamma\ (FractionalHeap.normalize\ (get\text{-}fh\ H)))$
proof –
have $reachable\ \Gamma\ v0\ H$
by $(meson\ \text{asm2}\ semi\text{-}consistent\text{-}def)$
moreover have $get\text{-}gs\ H = Some\ (pwrite, ?pre\text{-}ms\ s + msf) \wedge (\forall k. get\text{-}gu\ H\ k = Some\ (uargs\ k))$
proof $(rule\ conjI)$
show $\forall k. get\text{-}gu\ H\ k = Some\ (uargs\ k)$
by $(metis\ \langle Some\ H = Some\ hhj \oplus Some\ hf \rangle\ full\text{-}uguard\text{-}sum\text{-}same\ ghf\text{-}def(1)\ ghf\text{-}def(4)\ plus\text{-}comm)$

moreover have $get\text{-}gs\ hhj = Some\ (\pi, ?pre\text{-}ms\ s)$
proof –
have $get\text{-}gs\ hj = None$
using $\text{asm2}\ no\text{-}guard\text{-}def\ sat\text{-}inv\text{-}def$ **by** $blast$
moreover have $get\text{-}gs\ h = Some\ (\pi, ?pre\text{-}ms\ s)$
using $\langle get\text{-}gu\ h = Map.empty \wedge get\text{-}gs\ h = Some\ (\pi, \text{map-to-multiset}\ (s\ ms)) \rangle$ **by** $blast$
ultimately show $?thesis$
by $(metis\ \langle Some\ hhj = Some\ h \oplus Some\ hj \rangle\ sum\text{-}gs\text{-}one\text{-}none)$
qed
ultimately show $get\text{-}gs\ H = Some\ (pwrite, ?pre\text{-}ms\ s + msf)$
proof $(cases\ \pi = pwrite)$
case $True$
then have $\pi = pwrite \wedge get\text{-}gs\ ghf = None \wedge msf = \{\#\}$ **using**

```

ghf-def(3)
  by (metis not-pgte-charact pgt-implies-pgte sum-larger)
  then show ?thesis
    by (metis ‹Some H = Some hhj  $\oplus$  Some hf› ‹get-gs hhj =
Some ( $\pi$ , map-to-multiset (s ms))› add.right-neutral full-sguard-sum-same)
  next
  case False
  then have padd  $\pi \pi' = pwrite \wedge get-gs ghf = Some (\pi', msf)$ 
    using ghf-def(3) by blast
  then show ?thesis using ‹Some H = Some hhj  $\oplus$  Some hf›
sum-gs-one-some ghf-def(1)
  ‹get-gs hhj = Some ( $\pi$ , ?pre-ms s)› asm3(1) asm3(4)
no-guard-remove(1)[of hf ghf remove-guards hf] no-guard-then-smaller-same plus-comm
  by metis
  qed
  qed
  ultimately show ?thesis
    by (meson reachableE)
  qed
  moreover have view  $\Gamma (FractionalHeap.normalize (get-fh hj)) = view$ 
 $\Gamma (FractionalHeap.normalize (get-fh H))$ 
    using assms(4)
  proof (rule view-function-of-invE)
  show sat-inv s hj  $\Gamma$ 
    by (simp add: asm2-bis)
  show  $H \succeq hj$ 
    by (metis (no-types, opaque-lifting) ‹Some H = Some hhj  $\oplus$  Some
hf› ‹Some hhj = Some h  $\oplus$  Some hj› larger-def larger-trans plus-comm)
  qed
  ultimately have reachable-value (saction  $\Gamma$ ) (uaction  $\Gamma$ ) v0 (?pre-ms
s + msf) uargs v
    using ‹f (FractionalHeap.normalize (get-fh hj)) = v› assms(1) by
auto
  then show ?thesis
    using SharedStep assms(1)
    using ‹s x = s' x  $\wedge$  s sarg = s' sarg  $\wedge$  s ms = s' ms› by fastforce
  qed
  ultimately show reachable-value (saction  $\Gamma$ ) (uaction  $\Gamma$ ) v0 sargs
uargs' (view  $\Gamma (FractionalHeap.normalize (get-fh H''))$ )
    using ‹f (FractionalHeap.normalize (get-fh hj')) = sact v (map-to-arg
(s' sarg))› assms(1) by force
  qed
  qed
  ultimately show  $\exists H''$ . semi-consistent  $\Gamma v0 H'' \wedge Some H'' = Some$ 
(add-sguard-to-no-guard hq'  $\pi (\{\#map-to-arg (s' sarg)\#} + map-to-multiset (s'$ 
ms)))  $\oplus Some hj' \oplus Some hf$ 
    by blast
  qed
  ultimately obtain  $H''$  where semi-consistent  $\Gamma v0 H'' \wedge Some H'' =$ 

```

Some (add-sguard-to-no-guard $hq' \pi$ (?ms s')) \oplus *Some* hj' \oplus *Some* hf
 \wedge safe n (*Some* Γ) C' (s' , add-sguard-to-no-guard $hq' \pi$ (?ms s')) (Σ'
h)) **by** blast
moreover have full-ownership (get-fh H'') \wedge $h' = \text{FractionalHeap.normalize}$
(get-fh H'')
proof –
obtain x **where** *Some* $x = \text{Some}$ (add-sguard-to-no-guard $hq' \pi$ (?ms s'))
 \oplus *Some* hj'
by (metis calculation not-Some-eq plus.simps(1))
then have get-fh $H'' = \text{add-fh}$ (add-fh (get-fh (add-sguard-to-no-guard hq'
 π (?ms s')) (get-fh hj')) (get-fh hf)
by (metis add-get-fh calculation)
moreover have get-fh (add-sguard-to-no-guard $hq' \pi$ (?ms s')) = get-fh
 $hq' \wedge$ get-fh $hf = \text{get-fh}$ (remove-guards hf)
by (metis get-fh-add-sguard get-fh-remove-guards)
ultimately show ?thesis
by (metis \langle Some $h1 = \text{Some } hq' \oplus \text{Some } hj' \rangle$ add-get-fh asm3(1) asm3(3)
asm3(4))
qed
moreover have sat-inv pre- s' $hj' \Gamma$
proof (rule sat-inv-agrees)
show sat-inv $s' hj' \Gamma$
by (simp add: \langle sat-inv $s' hj' \Gamma \rangle$)
show agrees (fvA (invariant Γ)) $s' \text{pre-}s'$
using UnCI \langle agrees $(- \{x\}) s' \text{pre-}s' \rangle$ assms(1) assms(5) select-convs(5)
subset-Compl-singleton
by (metis (mono-tags, lifting) agrees-def in-mono)
qed
moreover have safe n (*Some* Γ) C' (pre- s' , add-sguard-to-no-guard $hq' \pi$
(?ms s')) (? Σ' (pre-s, h))
proof (rule safe-free-vars-Some)
show safe n (*Some* Γ) C' (s' , add-sguard-to-no-guard $hq' \pi$ (?ms s')) (? Σ'
(pre-s, h))
by (meson \langle safe n (*Some* Γ) C' (s' , add-sguard-to-no-guard $hq' \pi$
($\{\# \text{map-to-arg } (s' \text{ sarg}) \# \} + \text{map-to-multiset } (s' \text{ ms})$)) (Σ' (pre-s, h)) \rangle close-var-subset
safe-larger-set)
show agrees (fvC $C' \cup (- \{x\}) \cup \text{fvA}$ (invariant Γ)) $s' \text{pre-}s'$
by (metis UnI2 Un-absorb1 \langle agrees $(- \{x\}) s' \text{pre-}s' \rangle$ asm3(2) assms(1)
assms(5) empty-iff fvC.simps(1) inf-sup-aci(5) select-convs(5) subset-Compl-singleton)
show upper-fvs (close-var (Σ' (pre-s, h)) x) $(- \{x\})$
by (simp add: upper-fvs-close-vars)
qed
ultimately show $\exists h'' H' hj'$.
full-ownership (get-fh H') \wedge
semi-consistent $\Gamma v0 H' \wedge$
sat-inv pre- $s' hj' \Gamma \wedge h' = \text{FractionalHeap.normalize}$ (get-fh H') \wedge *Some*
 $H' = \text{Some } h'' \oplus \text{Some } hj' \oplus \text{Some } hf$
 \wedge safe n (*Some* Γ) C' (pre- s' , h'') (? Σ' (pre-s, h)) **using** \langle sat-inv $s' hj' \Gamma \rangle$
by blast

```

      qed
      ultimately show safe k (Some  $\Gamma$ ) (Catomic C) (pre-s, h) (? $\Sigma'$  (pre-s, h))
by blast
      qed (simp)
      qed
      qed

```

4.4.8 Parallel

lemma *par-cases*:

```

  assumes red (Cpar C1 C2)  $\sigma$  C'  $\sigma'$ 
  and  $\bigwedge C1'. C' = Cpar C1' C2 \wedge red C1 \sigma C1' \sigma' \implies P$ 
  and  $\bigwedge C2'. C' = Cpar C1 C2' \wedge red C2 \sigma C2' \sigma' \implies P$ 
  and  $C1 = Cskip \wedge C2 = Cskip \wedge C' = Cskip \wedge \sigma = \sigma' \implies P$ 
  shows P
  using assms(1)
  apply (rule red.cases)
  apply blast+
  apply (simp add: assms(2))
  apply (simp add: assms(3))
  apply (simp add: assms(4))
  apply blast+
  done

```

lemma *no-abort-par*:

```

  assumes no-abort  $\Gamma$  C1 s h
  and no-abort  $\Gamma$  C2 s h
  shows no-abort  $\Gamma$  (Cpar C1 C2) s h
proof (rule no-abortI)
  show  $\bigwedge hf H$ .
    Some H = Some h  $\oplus$  Some hf  $\wedge \Gamma = None \wedge full\text{-ownership (get-fh H)} \wedge$ 
no-guard H  $\implies$ 
     $\neg aborts (Cpar C1 C2) (s, FractionalHeap.normalize (get-fh H))$ 
  proof -
    fix hf H assume asm0: Some H = Some h  $\oplus$  Some hf  $\wedge \Gamma = None \wedge$ 
full-ownership (get-fh H)  $\wedge$  no-guard H
    let ?H = FractionalHeap.normalize (get-fh H)
    show  $\neg aborts (Cpar C1 C2) (s, FractionalHeap.normalize (get-fh H))$ 
    proof (rule ccontr)
      assume  $\neg \neg aborts (Cpar C1 C2) (s, FractionalHeap.normalize (get-fh H))$ 
      then have aborts (Cpar C1 C2) (s, FractionalHeap.normalize (get-fh H)) by
simp
      then have aborts C1 (s, ?H)  $\vee$  aborts C2 (s, ?H)
        by (rule aborts.cases) auto
      then show False
        using asm0 assms(1) assms(2) no-abortE(1) by blast
    qed
  qed
  fix H hf hj v0  $\Gamma'$ 

```



```

assume asm0:  $\Gamma = \text{Some } \Gamma' \wedge \text{Some } H = \text{Some } h \oplus \text{Some } hj \oplus \text{Some } hf \wedge$ 
full-ownership (get-fh H)  $\wedge$  semi-consistent  $\Gamma' \text{ v0 } H \wedge \text{sat-inv } s \text{ hj } \Gamma'$ 
let ?H = FractionalHeap.normalize (get-fh H)
show  $\neg \text{aborts } (\text{Cpar } C1 \ C2) (s, \text{FractionalHeap.normalize } (\text{get-fh } H))$ 
proof (rule ccontr)
  assume  $\neg \neg \text{aborts } (\text{Cpar } C1 \ C2) (s, \text{FractionalHeap.normalize } (\text{get-fh } H))$ 
  then have  $\text{aborts } (\text{Cpar } C1 \ C2) (s, \text{FractionalHeap.normalize } (\text{get-fh } H))$  by
simp
  then have  $\text{aborts } C1 (s, ?H) \vee \text{aborts } C2 (s, ?H)$ 
  by (rule aborts.cases) auto
  then show False
  using asm0 assms(1) assms(2) no-abortE(2) by blast
qed
qed

```

lemma *parallel-comp-none*:

```

assumes safe n (None :: ('i, 'a, nat) cont) C1 (s, h1) S1
and safe n (None :: ('i, 'a, nat) cont) C2 (s, h2) S2
and Some h = Some h1  $\oplus$  Some h2

and disjoint (fvC C1  $\cup$  vars1) (wrC C2)
and disjoint (fvC C2  $\cup$  vars2) (wrC C1)

and upper-fvs S1 vars1
and upper-fvs S2 vars2

shows safe n (None :: ('i, 'a, nat) cont) (Cpar C1 C2) (s, h) (add-states S1
S2)
using assms
proof (induct n arbitrary: C1 h1 C2 h2 s h S1 S2)
case (Suc n)
show ?case
proof (rule safeNoneI)
  show Cpar C1 C2 = Cskip  $\implies (s, h) \in \text{add-states } S1 \ S2$ 
  by simp
  show no-abort (None :: ('i, 'a, nat) cont) (Cpar C1 C2) s h
  proof (rule no-abort-par)
    show no-abort (None :: ('i, 'a, nat) cont) C1 s h
    using Suc.prem(1) Suc.prem(3) larger-def no-abort-larger safe.simps(2)
by blast
    have  $h \succeq h2$ 
    by (metis Suc.prem(3) larger-def plus-comm)
    then show no-abort (None :: ('i, 'a, nat) cont) C2 s h
    using Suc.prem(2) no-abort-larger safeNoneE-bis(2) by blast
qed
fix H hf C' s' h'
assume asm0: Some H = Some h  $\oplus$  Some hf  $\wedge$ 
full-ownership (get-fh H)  $\wedge$  no-guard H  $\wedge$  red (Cpar C1 C2) (s, Fractional-
Heap.normalize (get-fh H)) C' (s', h')

```

obtain $hf1$ **where** $Some\ hf1 = Some\ h1 \oplus Some\ hf$
by (*metis* (*no-types*, *opaque-lifting*) *Suc.prem*(3) *asm0 plus.simps*(1) *plus.simps*(3)
plus-asso plus-comm)
then have $Some\ H = Some\ h2 \oplus Some\ hf1$
by (*metis* (*no-types*, *lifting*) *Suc.prem*(3) *asm0 plus-asso plus-comm*)
obtain $hf2$ **where** $Some\ hf2 = Some\ h2 \oplus Some\ hf$
by (*metis* (*no-types*, *opaque-lifting*) $\langle Some\ H = Some\ h2 \oplus Some\ hf1 \rangle \langle Some\ hf1 = Some\ h1 \oplus Some\ hf \rangle$ *option.exhaust-sel plus.simps*(1) *plus-asso plus-comm*)
then have $Some\ H = Some\ h1 \oplus Some\ hf2$
by (*metis* *Suc.prem*(3) *asm0 plus-asso*)

let $?H = normalize\ (get-fh\ H)$

show $\exists h''\ H'$.
full-ownership (*get-fh* H') \wedge
no-guard $H' \wedge h' = FractionalHeap.normalize\ (get-fh\ H') \wedge Some\ H' = Some\ h'' \oplus Some\ hf \wedge safe\ n\ (None :: ('i, 'a, nat)\ cont)\ C'\ (s', h'')$ (*add-states* $S1\ S2$)

proof (*rule par-cases*)
show *red* (*Cpar* $C1\ C2$) ($s, ?H$) $C'\ (s', h')$
using *asm0 by blast*

show $C1 = Cskip \wedge C2 = Cskip \wedge C' = Cskip \wedge (s, FractionalHeap.normalize\ (get-fh\ H)) = (s', h') \implies$
 $\exists h''\ H'.\ full-ownership\ (get-fh\ H') \wedge$
no-guard $H' \wedge h' = FractionalHeap.normalize\ (get-fh\ H') \wedge Some\ H' = Some\ h'' \oplus Some\ hf \wedge safe\ n\ (None :: ('i, 'a, nat)\ cont)\ C'\ (s', h'')$ (*add-states* $S1\ S2$)

proof –
assume *asm1*: $C1 = Cskip \wedge C2 = Cskip \wedge C' = Cskip \wedge (s, FractionalHeap.normalize\ (get-fh\ H)) = (s', h')$
then have $(s, h1) \in S1 \wedge (s, h2) \in S2$
using *Suc.prem*(1) *Suc.prem*(2) *safe.simps*(2) **by** *blast*
moreover have $(s, h) \in add-states\ S1\ S2$
by (*metis* (*mono-tags*, *lifting*) *Suc.prem*(3) *add-states-def calculation mem-Collect-eq*)

ultimately show $\exists h''\ H'$.
full-ownership (*get-fh* H') \wedge
no-guard $H' \wedge h' = FractionalHeap.normalize\ (get-fh\ H') \wedge Some\ H' = Some\ h'' \oplus Some\ hf \wedge safe\ n\ (None :: ('i, 'a, nat)\ cont)\ C'\ (s', h'')$ (*add-states* $S1\ S2$)
by (*metis* *asm0 asm1 old.prod.inject safe-skip*)

qed

show $\bigwedge C1'.\ C' = Cpar\ C1'\ C2 \wedge red\ C1\ (s, FractionalHeap.normalize\ (get-fh\ H))\ C1'\ (s', h') \implies$
 $\exists h''\ H'$.
full-ownership (*get-fh* H') \wedge
no-guard $H' \wedge h' = FractionalHeap.normalize\ (get-fh\ H') \wedge Some\ H'$

$= \text{Some } h'' \oplus \text{Some } hf \wedge \text{safe } n \text{ (None :: ('i, 'a, nat) cont) } C' (s', h'') \text{ (add-states } S1 \text{ } S2)$

proof –
fix $C1'$
assume $asm1: C' = \text{Cpar } C1' \text{ } C2 \wedge \text{red } C1 (s, \text{FractionalHeap.normalize (get-fh } H)) C1' (s', h')$
then obtain $h1' \text{ } H'$ **where** $asm2: \text{full-ownership (get-fh } H') \text{ no-guard } H'$
 $h' = \text{FractionalHeap.normalize (get-fh } H')$
 $\text{Some } H' = \text{Some } h1' \oplus \text{Some } hf2 \text{ safe } n \text{ (None :: ('i, 'a, nat) cont) } C1' (s', h1') \text{ } S1$
using $\text{Suc.prem}(1) \text{ } asm0 \text{ safeNoneE}(3)[\text{of } n \text{ } C1 \text{ } s \text{ } h1 \text{ } S1 \text{ } H \text{ } hf2 \text{ } C1' \text{ } s' \text{ } h']$
 $\langle \text{Some } H = \text{Some } h1 \oplus \text{Some } hf2 \rangle$ **by** *blast*

moreover have $\text{safe } n \text{ (None :: ('i, 'a, nat) cont) } C2 (s, h2) \text{ } S2$
by $(\text{meson } \text{Suc.prem}(2) \text{ } \text{Suc-leD } \text{le-Suc-eq } \text{safe-smaller})$

then have $\text{safe } n \text{ (None :: ('i, 'a, nat) cont) } C2 (s', h2) \text{ } S2$
proof $(\text{rule } \text{safe-free-vars-None})$
show $\text{agrees (fvC } C2 \cup \text{vars2) } s \text{ } s'$
using $\text{Suc.prem}(5) \text{ } \text{agrees-minusD}[\text{of }] \text{ } \text{agrees-comm } asm1 \text{ } \text{fst-eqD}$
 $\text{red-properties}(1) \text{ } \text{disjoint-def } \text{inf-commute}$
by *metis*
show $\text{upper-fvs } S2 \text{ } \text{vars2}$
by $(\text{simp add: } \text{Suc.prem}(7))$
qed

moreover obtain h'' **where** $\text{Some } h'' = \text{Some } h1' \oplus \text{Some } h2$
by $(\text{metis } \langle \text{Some } hf2 = \text{Some } h2 \oplus \text{Some } hf \rangle \text{ } \text{calculation}(4) \text{ } \text{not-Some-eq}$
 $\text{plus.simp}(1) \text{ } \text{plus-asso})$
have $\text{safe } n \text{ (None :: ('i, 'a, nat) cont) (Cpar } C1' \text{ } C2) (s', h'') \text{ (add-states } S1 \text{ } S2)$

proof $(\text{rule } \text{Suc.hyps})$
show $\text{safe } n \text{ (None :: ('i, 'a, nat) cont) } C1' (s', h1') \text{ } S1$
using $\text{calculation}(5)$ **by** *blast*
show $\text{safe } n \text{ (None :: ('i, 'a, nat) cont) } C2 (s', h2) \text{ } S2$
using $\text{calculation}(6)$ **by** *auto*
show $\text{Some } h'' = \text{Some } h1' \oplus \text{Some } h2$
using $\langle \text{Some } h'' = \text{Some } h1' \oplus \text{Some } h2 \rangle$ **by** *blast*
show $\text{disjoint (fvC } C1' \cup \text{vars1) (wrC } C2)$
using $\text{Suc.prem}(4) \text{ } asm1 \text{ } \text{red-properties}(1) \text{ } \text{Un-iff } \text{disjoint-def}[\text{of } \text{fvC } C1 \cup \text{vars1 } \text{wrC } C2]$
 $\text{disjoint-def}[\text{of } \text{fvC } C1' \cup \text{vars1 } \text{wrC } C2]$
 $\text{inf-shunt } \text{subset-iff}$ **by** *blast*
show $\text{disjoint (fvC } C2 \cup \text{vars2) (wrC } C1')$
by $(\text{metis (no-types, lifting) } \text{Suc.prem}(5) \text{ } asm1 \text{ } \text{disjoint-def } \text{inf-commute}$
 $\text{inf-shunt } \text{red-properties}(1) \text{ } \text{subset-Un-eq } \text{sup-assoc})$
show $\text{upper-fvs } S1 \text{ } \text{vars1}$
by $(\text{simp add: } \text{Suc.prem}(6))$
show $\text{upper-fvs } S2 \text{ } \text{vars2}$

by (simp add: Suc.prem(7))
 qed

ultimately show $\exists h'' H'$.
 full-ownership (get-fh H') \wedge
 no-guard $H' \wedge$
 $h' = \text{FractionalHeap.normalize (get-fh } H') \wedge \text{Some } H' = \text{Some } h'' \oplus$
 $\text{Some hf} \wedge \text{safe } n \text{ (None :: ('i, 'a, nat) cont) } C' (s', h'') \text{ (add-states } S1 \ S2)$
 by (metis $\langle \text{Some } h'' = \text{Some } h1' \oplus \text{Some } h2 \rangle \langle \text{Some } hf2 = \text{Some } h2 \oplus$
 $\text{Some hf} \rangle \text{asm1 plus-asso}$)
 qed

show $\bigwedge C2'. C' = \text{Cpar } C1 \ C2' \wedge \text{red } C2 (s, \text{FractionalHeap.normalize (get-fh}$
 $H)) \ C2' (s', h') \implies$
 $\exists h'' H'$.
 full-ownership (get-fh H') \wedge
 no-guard $H' \wedge h' = \text{FractionalHeap.normalize (get-fh } H') \wedge \text{Some } H'$
 $= \text{Some } h'' \oplus \text{Some hf} \wedge \text{safe } n \text{ (None :: ('i, 'a, nat) cont) } C' (s', h'') \text{ (add-states}$
 $S1 \ S2)$

proof –
fix $C2'$
assume $\text{asm1: } C' = \text{Cpar } C1 \ C2' \wedge \text{red } C2 (s, \text{FractionalHeap.normalize}$
 $(\text{get-fh } H)) \ C2' (s', h')$
then obtain $h2' H'$ **where** $\text{asm2: full-ownership (get-fh } H') \text{ no-guard } H'$
 $h' = \text{FractionalHeap.normalize (get-fh } H')$
 $\text{Some } H' = \text{Some } h2' \oplus \text{Some hf1 safe } n \text{ (None :: ('i, 'a, nat) cont) } C2'$
 $(s', h2') \ S2$
using $\text{Suc.prem(1) asm0 safeNoneE(3) Suc.prem(2) } \langle \text{Some } H = \text{Some}$
 $h2 \oplus \text{Some hf1} \rangle$ **by** blast

moreover have $\text{safe } n \text{ (None :: ('i, 'a, nat) cont) } C1 (s, h1) \ S1$
by (meson $\text{Suc.prem(1) Suc-leD le-Suc-eq safe-smaller}$)

then have $\text{safe } n \text{ (None :: ('i, 'a, nat) cont) } C1 (s', h1) \ S1$
proof (rule safe-free-vars-None)
show agrees (fvC $C1 \cup \text{vars1}$) $s \ s'$
using $\text{Suc.prem(4) agrees-comm asm1 fst-eqD red-properties(1) dis-}$
 $\text{joint-def[of fvC } C1 \cup \text{vars1 wrC } C2]$
 agrees-minusD **by** (metis inf-commute)
show upper-fvs $S1 \ \text{vars1}$
by (simp add: Suc.prem(6))
 qed

moreover obtain h'' **where** $\text{Some } h'' = \text{Some } h2' \oplus \text{Some } h1$
by (metis $\langle \text{Some } hf1 = \text{Some } h1 \oplus \text{Some } hf \rangle \text{calculation(4) not-Some-eq}$
 $\text{plus.sims(1) plus-asso}$)
have $\text{safe } n \text{ (None :: ('i, 'a, nat) cont) } (Cpar \ C1 \ C2') (s', h'') \text{ (add-states}$
 $S1 \ S2)$
proof (rule Suc.hyps)
show $\text{safe } n \text{ (None :: ('i, 'a, nat) cont) } C1 (s', h1) \ S1$

using *calculation(6)* **by** *blast*
show *safe n (None :: ('i, 'a, nat) cont) C2' (s', h2') S2*
using *calculation(5)* **by** *auto*
show *Some h'' = Some h1 \oplus Some h2'*
by (*simp add: <Some h'' = Some h2' \oplus Some h1> plus-comm*)
show *disjoint (fvC C2' \cup vars2) (wrC C1)*
using *Suc.prem(5) asm1 disjoint-def[of fvC C2 \cup vars2 wrC C1]*
disjoint-def[of fvC C2' \cup vars2 wrC C1]
inf-shunt inf-sup-aci(5) red-properties(1) subset-Un-eq sup.idem sup-assoc
by *fast*
show *disjoint (fvC C1 \cup vars1) (wrC C2')*
by (*metis (no-types, lifting) Suc.prem(4) asm1 disjoint-def inf-commute*
inf-shunt red-properties(1) subset-Un-eq sup-assoc)
show *upper-fvs S1 vars1*
by (*simp add: Suc.prem(6)*)
show *upper-fvs S2 vars2*
by (*simp add: Suc.prem(7)*)
qed

ultimately show $\exists h'' H'$.
full-ownership (get-fh H') \wedge
no-guard H' \wedge
h' = FractionalHeap.normalize (get-fh H') \wedge Some H' = Some h'' \oplus
Some hf \wedge safe n (None :: ('i, 'a, nat) cont) C' (s', h'') (add-states S1 S2)
by (*metis <Some h'' = Some h2' \oplus Some h1> <Some hf1 = Some h1 \oplus*
Some hf> asm1 plus-asso)
qed
qed
qed
qed (*simp*)

lemma *parallel-comp-some:*

assumes *safe n (Some Γ) C1 (s, h1) S1*
and *safe n (Some Γ) C2 (s, h2) S2*
and *Some h = Some h1 \oplus Some h2*

and *disjoint (fvC C1 \cup vars1) (wrC C2)*
and *disjoint (fvC C2 \cup vars2) (wrC C1)*

and *upper-fvs S1 vars1*
and *upper-fvs S2 vars2*

and *disjoint (fvA (invariant Γ)) (wrC C2)*
and *disjoint (fvA (invariant Γ)) (wrC C1)*

shows *safe n (Some Γ) (Cpar C1 C2) (s, h) (add-states S1 S2)*
using *assms*
proof (*induct n arbitrary: C1 h1 C2 h2 s h S1 S2*)
case (*Suc n*)

```

show ?case
proof (rule safeSomeI)
  show Cpar C1 C2 = Cskip  $\implies$  (s, h)  $\in$  add-states S1 S2
    by simp
  show no-abort (Some  $\Gamma$ ) (Cpar C1 C2) s h
    proof (rule no-abort-par)
      show no-abort (Some  $\Gamma$ ) C1 s h
        using Suc.premis(1) Suc.premis(3) larger-def no-abort-larger safe.simps(3)
by blast
  have h  $\succeq$  h2
    by (metis Suc.premis(3) larger-def plus-comm)
  then show no-abort (Some  $\Gamma$ ) C2 s h
    using Suc.premis(2) no-abort-larger safeSomeE(2) by blast
qed
fix H hf C' s' h' hj v0
  assume asm0: Some H = Some h  $\oplus$  Some hj  $\oplus$  Some hf  $\wedge$  full-ownership
    (get-fh H)  $\wedge$ 
    semi-consistent  $\Gamma$  v0 H  $\wedge$  sat-inv s hj  $\Gamma$   $\wedge$  red (Cpar C1 C2) (s, Fractional-
    Heap.normalize (get-fh H)) C' (s', h')

  obtain hf1 where Some hf1 = Some h1  $\oplus$  Some hf
    by (metis (no-types, opaque-lifting) Suc.premis(3) asm0 plus.simps(1) plus.simps(3)
    plus-asso plus-comm)
  then have Some H = Some h2  $\oplus$  Some hf1  $\oplus$  Some hj
    by (metis (no-types, lifting) Suc.premis(3) asm0 plus-asso plus-comm)
  then have Some H = Some h2  $\oplus$  Some hj  $\oplus$  Some hf1
    by (metis plus-asso plus-comm)
  obtain hf2 where Some hf2 = Some h2  $\oplus$  Some hf
    by (metis (no-types, opaque-lifting)  $\langle$ Some H = Some h2  $\oplus$  Some hf1  $\oplus$ 
    Some hj $\rangle$   $\langle$ Some hf1 = Some h1  $\oplus$  Some hf $\rangle$  not-Some-eq plus.simps(1) plus-asso
    plus-comm)
  then have Some H = Some h1  $\oplus$  Some hf2  $\oplus$  Some hj
    by (metis (no-types, opaque-lifting)  $\langle$ Some H = Some h2  $\oplus$  Some hf1  $\oplus$  Some
    hj $\rangle$   $\langle$ Some hf1 = Some h1  $\oplus$  Some hf $\rangle$  plus-asso plus-comm)
  then have Some H = Some h1  $\oplus$  Some hj  $\oplus$  Some hf2
    by (metis plus-asso plus-comm)

  let ?H = normalize (get-fh H)

  show  $\exists$  h'' H' hj'.
    full-ownership (get-fh H')  $\wedge$ 
    semi-consistent  $\Gamma$  v0 H'  $\wedge$ 
    sat-inv s' hj'  $\Gamma$   $\wedge$ 
    h' = FractionalHeap.normalize (get-fh H')  $\wedge$  Some H' = Some h''  $\oplus$  Some
    hj'  $\oplus$  Some hf  $\wedge$  safe n (Some  $\Gamma$ ) C' (s', h') (add-states S1 S2)
  proof (rule par-cases)
    show red (Cpar C1 C2) (s, ?H) C' (s', h')
      using asm0 by blast

```

show $C1 = Cskip \wedge C2 = Cskip \wedge C' = Cskip \wedge (s, FractionalHeap.normalize (get-fh H)) = (s', h') \implies$
 $\exists h'' H' hj'. full-ownership (get-fh H') \wedge$
 $semi-consistent \Gamma v0 H' \wedge$
 $sat-inv s' hj' \Gamma \wedge h' = FractionalHeap.normalize (get-fh H') \wedge Some H' =$
 $Some h'' \oplus Some hj' \oplus Some hf \wedge safe n (Some \Gamma) C' (s', h'') (add-states S1 S2)$
proof –
assume $asm1: C1 = Cskip \wedge C2 = Cskip \wedge C' = Cskip \wedge (s, FractionalHeap.normalize (get-fh H)) = (s', h')$
then have $(s, h1) \in S1 \wedge (s, h2) \in S2$
using $Suc.prem1(1) Suc.prem1(2) safe.simp1(3)$ **by** *blast*
moreover have $(s, h) \in add-states S1 S2$
by $(metis (mono-tags, lifting) Suc.prem1(3) add-states-def calculation mem-Collect-eq)$
ultimately show $\exists h'' H' hj'. full-ownership (get-fh H') \wedge$
 $semi-consistent \Gamma v0 H' \wedge$
 $sat-inv s' hj' \Gamma \wedge h' = FractionalHeap.normalize (get-fh H') \wedge Some H' =$
 $Some h'' \oplus Some hj' \oplus Some hf \wedge safe n (Some \Gamma) C' (s', h'') (add-states S1 S2)$
by $(metis asm0 asm1 old.prod.inject safe-skip)$
qed

show $\bigwedge C1'. C' = Cpar C1' C2 \wedge red C1 (s, FractionalHeap.normalize (get-fh H)) C1' (s', h') \implies$
 $\exists h'' H' hj'. full-ownership (get-fh H') \wedge semi-consistent \Gamma v0 H' \wedge$
 $sat-inv s' hj' \Gamma \wedge h' = FractionalHeap.normalize (get-fh H') \wedge Some$
 $H' = Some h'' \oplus Some hj' \oplus Some hf \wedge safe n (Some \Gamma) C' (s', h'') (add-states$
 $S1 S2)$
proof –
fix $C1'$
assume $asm1: C' = Cpar C1' C2 \wedge red C1 (s, FractionalHeap.normalize (get-fh H)) C1' (s', h')$
then obtain $h1' H' hj'$ **where** $asm2: full-ownership (get-fh H') h' =$
 $FractionalHeap.normalize (get-fh H')$
 $semi-consistent \Gamma v0 H' sat-inv s' hj' \Gamma Some H' = Some h1' \oplus Some hj'$
 $\oplus Some hf2 safe n (Some \Gamma) C1' (s', h1') S1$
using $safeSomeE(3)[of n \Gamma C1 s h1 S1 H hj hf2 v0 C1' s' h'] Suc.prem1(1)$
 $asm0$
using $\langle Some H = Some h1 \oplus Some hj \oplus Some hf2 \rangle$ **by** *blast*

moreover have $safe n (Some \Gamma) C2 (s, h2) S2$
by $(meson Suc.prem1(2) Suc-leD le-Suc-eq safe-smaller)$

then have $safe n (Some \Gamma) C2 (s', h2) S2$
proof $(rule safe-free-vars-Some)$
show $agrees (fvC C2 \cup vars2 \cup fvA (invariant \Gamma)) s s'$
using $Suc.prem1(5) Suc.prem1(9) agrees-minusD agrees-comm asm1$
 $disjoint-def fst-eqD red-properties(1)$
by $(metis agrees-union inf-commute)$

```

show upper-fvs S2 vars2
  by (simp add: Suc.prem(7))
qed

moreover have h1' ## h2
  by (metis (no-types, opaque-lifting) ‹Some hf2 = Some h2  $\oplus$  Some hf›
  calculation(5) compatible-eq option.discI plus.simps(1) plus-asso plus-comm)
then obtain h'' where Some h'' = Some h1'  $\oplus$  Some h2 by simp

have safe n (Some  $\Gamma$ ) (Cpar C1' C2) (s', h'') (add-states S1 S2)
proof (rule Suc.hyps)
  show safe n (Some  $\Gamma$ ) C1' (s', h1') S1
    using calculation(6) by blast
  show safe n (Some  $\Gamma$ ) C2 (s', h2) S2
    using calculation(7) by auto
  show Some h'' = Some h1'  $\oplus$  Some h2
    using ‹Some h'' = Some h1'  $\oplus$  Some h2› by blast
  show disjoint (fvC C1'  $\cup$  vars1) (wrC C2)
    by (metis (no-types, opaque-lifting) Suc.prem(4) asm1 disjoint-Un1
  disjoint-def disjoint-def red-properties(1) sup.orderE)
  show disjoint (fvC C2  $\cup$  vars2) (wrC C1')
    by (metis (no-types, lifting) Suc.prem(5) asm1 disjoint-def inf-commute
  inf-shunt red-properties(1) subset-Un-eq sup-assoc)
  show upper-fvs S1 vars1
    by (simp add: Suc.prem(6))
  show upper-fvs S2 vars2
    by (simp add: Suc.prem(7))
  show disjoint (fvA (invariant  $\Gamma$ )) (wrC C2)
    by (simp add: Suc.prem(8))
  show disjoint (fvA (invariant  $\Gamma$ )) (wrC C1')
    by (metis (no-types, lifting) Suc.prem(9) asm1 disjoint-def inf-commute
  inf-shunt red-properties(1) subset-Un-eq sup-assoc)
qed
moreover have Some H' = Some h''  $\oplus$  Some hj'  $\oplus$  Some hf
  by (metis (no-types, opaque-lifting) ‹Some h'' = Some h1'  $\oplus$  Some h2›
  ‹Some hf2 = Some h2  $\oplus$  Some hf› calculation(5) plus-asso plus-comm)

ultimately show  $\exists$  h'' H' hj'.
  full-ownership (get-fh H')  $\wedge$ 
  semi-consistent  $\Gamma$  v0 H'  $\wedge$ 
  sat-inv s' hj'  $\Gamma$   $\wedge$  h' = FractionalHeap.normalize (get-fh H')  $\wedge$  Some
  H' = Some h''  $\oplus$  Some hj'  $\oplus$  Some hf  $\wedge$  safe n (Some  $\Gamma$ ) C' (s', h'') (add-states
  S1 S2)
  using asm1 by blast

qed

show  $\bigwedge$  C2'. C' = Cpar C1 C2'  $\wedge$  red C2 (s, FractionalHeap.normalize (get-fh

```


$H)) C2' (s', h') \implies$
 $\exists h'' H' hj'.$
 $full\text{-}ownership (get\text{-}fh H') \wedge$
 $semi\text{-}consistent \Gamma v0 H' \wedge$
 $sat\text{-}inv s' hj' \Gamma \wedge h' = FractionalHeap.normalize (get\text{-}fh H') \wedge Some$
 $H' = Some h'' \oplus Some hj' \oplus Some hf \wedge safe n (Some \Gamma) C' (s', h'') (add\text{-}states$
 $S1 S2)$
proof –
fix $C2'$
assume $asm1: C' = Cpar C1 C2' \wedge red C2 (s, FractionalHeap.normalize$
 $(get\text{-}fh H)) C2' (s', h')$
then obtain $h2' H' hj'$ **where** $asm2: full\text{-}ownership (get\text{-}fh H') h' =$
 $FractionalHeap.normalize (get\text{-}fh H')$
 $semi\text{-}consistent \Gamma v0 H' sat\text{-}inv s' hj' \Gamma Some H' = Some h2' \oplus Some hj'$
 $\oplus Some hf1 safe n (Some \Gamma) C2' (s', h2') S2$
using $safeSomeE(3)[of n \Gamma C2 s h2 S2 H hj hf1 v0 C2' s' h'] Suc.prem(2)$
 $Suc.prem(3)$
using $\langle Some H = Some h2 \oplus Some hj \oplus Some hf1 \rangle asm0$ **by** $blast$
moreover have $safe n (Some \Gamma) C1 (s, h1) S1$
by $(meson Suc.prem(1) Suc\text{-}leD le\text{-}Suc\text{-}eq safe\text{-}smaller)$

then have $safe n (Some \Gamma) C1 (s', h1) S1$
proof $(rule safe\text{-}free\text{-}vars\text{-}Some)$
show $agrees (fvC C1 \cup vars1 \cup fvA (invariant \Gamma)) s s'$
using $Suc.prem(4) Suc.prem(8) agrees\text{-}minusD agrees\text{-}comm asm1$
 $fst\text{-}eqD red\text{-}properties(1)$
by $(metis agrees\text{-}union disjoint\text{-}def inf\text{-}commute)$
show $upper\text{-}fvs S1 vars1$
by $(simp add: Suc.prem(6))$
qed

moreover have $h1 \#\# h2'$
by $(metis (no\text{-}types, opaque\text{-}lifting) \langle Some hf1 = Some h1 \oplus Some hf \rangle$
 $calculation(5) compatible\text{-}eq option.distinct(1) plus.simps(1) plus\text{-}asso plus\text{-}comm)$
then obtain h'' **where** $Some h'' = Some h1 \oplus Some h2'$ **by** $simp$

have $safe n (Some \Gamma) (Cpar C1 C2') (s', h'') (add\text{-}states S1 S2)$
proof $(rule Suc.hyps)$
show $safe n (Some \Gamma) C1 (s', h1) S1$
using $calculation(7)$ **by** $blast$
show $safe n (Some \Gamma) C2' (s', h2') S2$
using $calculation(6)$ **by** $auto$
show $Some h'' = Some h1 \oplus Some h2'$
using $\langle Some h'' = Some h1 \oplus Some h2' \rangle$ **by** $blast$
show $disjoint (fvC C1 \cup vars1) (wrC C2')$
by $(metis (no\text{-}types, lifting) Suc.prem(4) asm1 disjoint\text{-}def inf\text{-}commute$
 $inf\text{-}shunt red\text{-}properties(1) subset\text{-}Un\text{-}eq sup\text{-}assoc)$
show $disjoint (fvC C2' \cup vars2) (wrC C1)$
using $Suc.prem(5) asm1 red\text{-}properties(1)$

by (*metis* (*no-types*, *lifting*) *Un-subset-iff disjoint-def inf-shunt subset-Un-eq*)
show *disjoint (fvA (invariant Γ)) (wrC C2')*
using *Suc.premis(8) asm1 red-properties(1)*
by (*metis* (*no-types*, *lifting*) *Un-subset-iff disjoint-def inf-commute inf-shunt subset-Un-eq*)
show *disjoint (fvA (invariant Γ)) (wrC C1)*
by (*simp add: Suc.premis(9)*)
show *upper-fvs S1 vars1*
by (*simp add: Suc.premis(6)*)
show *upper-fvs S2 vars2*
by (*simp add: Suc.premis(7)*)
qed
moreover have *Some H' = Some h'' \oplus Some hj' \oplus Some hf*
by (*metis* \langle *Some h'' = Some h1 \oplus Some h2* \rangle \langle *Some hf1 = Some h1 \oplus Some hf* \rangle *calculation(5) plus-comm simpler-asso*)
ultimately show $\exists h'' H' hj'$.
full-ownership (get-fh H') \wedge
semi-consistent Γ v0 H' \wedge
sat-inv s' hj' Γ \wedge
h' = FractionalHeap.normalize (get-fh H') \wedge Some H' = Some h'' \oplus
Some hj' \oplus Some hf \wedge safe n (Some Γ) C' (s', h'') (add-states S1 S2)
using *asm1* **by** *blast*
qed
qed
qed
qed (*simp*)

lemma *parallel-comp*:

fixes $\Delta :: ('i, 'a, \text{nat}) \text{cont}$
assumes *safe n Δ C1 (s, h1) S1*
and *safe n Δ C2 (s, h2) S2*
and *Some h = Some h1 \oplus Some h2*
and *disjoint (fvC C1 \cup vars1) (wrC C2)*
and *disjoint (fvC C2 \cup vars2) (wrC C1)*
and *upper-fvs S1 vars1*
and *upper-fvs S2 vars2*

and $\wedge \Gamma. \Delta = \text{Some } \Gamma \implies \text{disjoint (fvA (invariant } \Gamma)) (wrC C2)$
and $\wedge \Gamma. \Delta = \text{Some } \Gamma \implies \text{disjoint (fvA (invariant } \Gamma)) (wrC C1)$

shows *safe n Δ (Cpar C1 C2) (s, h) (add-states S1 S2)*
proof (*cases* Δ)
case *None*
then show *?thesis*
using *assms parallel-comp-none* **by** *blast*
next
case (*Some* Γ)

then show *?thesis*
using *assms parallel-comp-some* **by** *blast*
qed

theorem *rule-par*:

fixes $\Delta :: ('i, 'a, nat)$ *cont*

assumes *hoare-triple-valid* Δ *P1 C1 Q1*

and *hoare-triple-valid* Δ *P2 C2 Q2*

and *disjoint* (*fvA P1* \cup *fvC C1* \cup *fvA Q1*) (*wrC C2*)

and *disjoint* (*fvA P2* \cup *fvC C2* \cup *fvA Q2*) (*wrC C1*)

and $\bigwedge \Gamma. \Delta = \text{Some } \Gamma \implies \text{disjoint } (\text{fvA } (\text{invariant } \Gamma)) (\text{wrC } C2)$

and $\bigwedge \Gamma. \Delta = \text{Some } \Gamma \implies \text{disjoint } (\text{fvA } (\text{invariant } \Gamma)) (\text{wrC } C1)$

and *precise P1* \vee *precise P2*

shows *hoare-triple-valid* Δ (*Star P1 P2*) (*Cpar C1 C2*) (*Star Q1 Q2*)

proof –

obtain $\Sigma 1$ **where** *r1*: $\bigwedge \sigma n. \sigma, \sigma \models P1 \implies \text{safe } n \Delta C1 \sigma (\Sigma 1 \sigma) \bigwedge \sigma \sigma'. \sigma, \sigma' \models P1 \implies \text{pair-sat } (\Sigma 1 \sigma) (\Sigma 1 \sigma') Q1$

using *assms(1) hoare-triple-validE* **by** *blast*

obtain $\Sigma 2$ **where** *r2*: $\bigwedge \sigma n. \sigma, \sigma \models P2 \implies \text{safe } n \Delta C2 \sigma (\Sigma 2 \sigma) \bigwedge \sigma \sigma'. \sigma, \sigma' \models P2 \implies \text{pair-sat } (\Sigma 2 \sigma) (\Sigma 2 \sigma') Q2$

using *assms(2) hoare-triple-validE* **by** *blast*

define *pairs* **where** *pairs* = $(\lambda(s, h). \{ ((s, h1), (s, h2)) \mid h1 \ h2. \text{Some } h = \text{Some } h1 \oplus \text{Some } h2 \wedge (s, h1), (s, h1) \models P1 \wedge (s, h2), (s, h2) \models P2 \})$

define Σ **where** $\Sigma = (\lambda \sigma. \bigcup (\sigma 1, \sigma 2) \in \text{pairs } \sigma. \text{add-states } (\text{upperize } (\Sigma 1 \sigma 1) (\text{fvA } Q1)) (\text{upperize } (\Sigma 2 \sigma 2) (\text{fvA } Q2)))$

show *?thesis*

proof (*rule hoare-triple-validI*)

show $\bigwedge s \ h \ n. (s, h), (s, h) \models \text{Star } P1 \ P2 \implies \text{safe } n \Delta (Cpar \ C1 \ C2) (s, h) (\Sigma (s, h))$

proof –

fix *s h n* **assume** $(s, h), (s, h) \models \text{Star } P1 \ P2$

then obtain *h1 h2* **where** *asm0*: $\text{Some } h = \text{Some } h1 \oplus \text{Some } h2 (s, h1), (s, h1) \models P1$

$(s, h2), (s, h2) \models P2$

using *always-sat-refl hyper-sat.simps(4)* **by** *blast*

then have $((s, h1), (s, h2)) \in \text{pairs } (s, h)$

using *pairs-def* **by** *blast*

then have $\text{add-states } (\text{upperize } (\Sigma 1 (s, h1)) (\text{fvA } Q1)) (\text{upperize } (\Sigma 2 (s, h2)) (\text{fvA } Q2)) \subseteq \Sigma (s, h)$

using Σ -*def* **by** *blast*

moreover have $\text{safe } n \Delta (Cpar \ C1 \ C2) (s, h) (\text{add-states } (\text{upperize } (\Sigma 1 (s,$

```

h1)) (fvA Q1)) (upperize (Σ2 (s, h2)) (fvA Q2)))
proof (rule parallel-comp)
  show safe n Δ C1 (s, h1) (upperize (Σ1 (s, h1)) (fvA Q1))
    by (meson asm0(2) r1(1) safe-larger-set upperize-larger)
  show safe n Δ C2 (s, h2) (upperize (Σ2 (s, h2)) (fvA Q2))
    by (meson asm0(3) r2(1) safe-larger-set upperize-larger)
  show Some h = Some h1 ⊕ Some h2 using asm0 by simp
  show disjoint (fvC C1 ∪ fvA Q1) (wrC C2)
    by (metis Un-subset-iff assms(3) disjoint-def inf-shunt)
  show disjoint (fvC C2 ∪ fvA Q2) (wrC C1)
    by (metis Un-subset-iff assms(4) disjoint-def inf-shunt)
  show upper-fvs (upperize (Σ1 (s, h1)) (fvA Q1)) (fvA Q1)
    by (simp add: upper-fvs-upperize)
  show upper-fvs (upperize (Σ2 (s, h2)) (fvA Q2)) (fvA Q2)
    using upper-fvs-upperize by auto
  show ∧Γ. Δ = Some Γ ⇒ disjoint (fvA (invariant Γ)) (wrC C2)
    using assms(5) by auto
  show ∧Γ. Δ = Some Γ ⇒ disjoint (fvA (invariant Γ)) (wrC C1)
    using assms(6) by blast
qed
ultimately show safe n Δ (Cpar C1 C2) (s, h) (Σ (s, h))
  using safe-larger-set by blast
qed

fix s h s' h'
assume (s, h), (s', h') ⊨ Star P1 P2
then obtain h1 h2 h1' h2' where asm0: Some h = Some h1 ⊕ Some h2 Some
h' = Some h1' ⊕ Some h2'
  (s, h1), (s', h1') ⊨ P1 (s, h2), (s', h2') ⊨ P2
  by auto

show pair-sat (Σ (s, h)) (Σ (s', h')) (Star Q1 Q2)
proof (rule pair-satI)
  fix ss hh ss' hh' assume asm1: (ss, hh) ∈ Σ (s, h) ∧ (ss', hh') ∈ Σ (s', h')

  then obtain σ1 σ2 σ1' σ2' where (σ1, σ2) ∈ pairs (s, h) (σ1', σ2') ∈
pairs (s', h')
  (ss, hh) ∈ add-states (upperize (Σ1 σ1) (fvA Q1)) (upperize (Σ2 σ2) (fvA
Q2))
  (ss', hh') ∈ add-states (upperize (Σ1 σ1') (fvA Q1)) (upperize (Σ2 σ2')
(fvA Q2))
  using Σ-def by blast
  then obtain fst σ1 = s fst σ2 = s fst σ1' = s' fst σ2' = s' Some h = Some
(snd σ1) ⊕ Some (snd σ2)
  Some h' = Some (snd σ1') ⊕ Some (snd σ2')
  (s, snd σ1), (s, snd σ1) ⊨ P1 ∧ (s, snd σ2), (s, snd σ2) ⊨ P2
  (s', snd σ1'), (s', snd σ1') ⊨ P1 ∧ (s', snd σ2'), (s', snd σ2') ⊨ P2
  using case-prod-conv pairs-def by auto

```

moreover have $\text{snd } \sigma 1 = h1 \wedge \text{snd } \sigma 2 = h2 \wedge \text{snd } \sigma 1' = h1' \wedge \text{snd } \sigma 2' = h2'$
proof (cases precise P1)
case True
then have $\text{snd } \sigma 1 = h1 \wedge \text{snd } \sigma 1' = h1'$
proof (rule preciseE)
show $h \succeq h1 \wedge h \succeq \text{snd } \sigma 1 \wedge h' \succeq h1' \wedge h' \succeq \text{snd } \sigma 1'$
using *asm0(1) asm0(2) calculation(5) calculation(6) larger-def* **by** *blast*
show $(s, h1), (s', h1') \models P1 \wedge (s, \text{snd } \sigma 1), (s', \text{snd } \sigma 1') \models P1$
by (metis True $\langle h \succeq h1 \wedge h \succeq \text{snd } \sigma 1 \wedge h' \succeq h1' \wedge h' \succeq \text{snd } \sigma 1' \rangle$
always-sat-refl asm0(3) calculation(7) calculation(8) preciseE sat-comm)
qed
then show ?thesis
by (metis addition-cancellative *asm0(1) asm0(2) calculation(5) calculation(6) plus-comm*)
next
case False
then have precise P2
using *assms(7)* **by** *blast*
then have $\text{snd } \sigma 2 = h2 \wedge \text{snd } \sigma 2' = h2'$
proof (rule preciseE)
show $h \succeq h2 \wedge h \succeq \text{snd } \sigma 2 \wedge h' \succeq h2' \wedge h' \succeq \text{snd } \sigma 2'$
by (metis *asm0(1) asm0(2) calculation(5) calculation(6) larger-def plus-comm*)
show $(s, h2), (s', h2') \models P2 \wedge (s, \text{snd } \sigma 2), (s', \text{snd } \sigma 2') \models P2$
by (metis $\langle h \succeq h2 \wedge h \succeq \text{snd } \sigma 2 \wedge h' \succeq h2' \wedge h' \succeq \text{snd } \sigma 2' \rangle$ \langle precise P2 \rangle *always-sat-refl asm0(4) calculation(7) calculation(8) preciseE sat-comm*)
qed
then show ?thesis
using *addition-cancellative asm0(1) asm0(2) calculation(5) calculation(6)*
by *blast*
qed
ultimately have *pair-sat* $(\Sigma 1 \sigma 1) (\Sigma 1 \sigma 1') Q1 \wedge$ *pair-sat* $(\Sigma 2 \sigma 2) (\Sigma 2 \sigma 2') Q2$
by (metis *asm0(3) asm0(4) prod.exhaust-sel r1(2) r2(2)*)
then show $(ss, hh), (ss', hh') \models \text{Star } Q1 Q2$
by (metis (no-types, opaque-lifting) $\langle (ss', hh') \in \text{add-states } (\text{upperize } (\Sigma 1 \sigma 1') (\text{fvA } Q1)) (\text{upperize } (\Sigma 2 \sigma 2') (\text{fvA } Q2)) \rangle$ $\langle (ss, hh) \in \text{add-states } (\text{upperize } (\Sigma 1 \sigma 1) (\text{fvA } Q1)) (\text{upperize } (\Sigma 2 \sigma 2) (\text{fvA } Q2)) \rangle$ *add-states-sat-star pair-sat-comm pair-sat-def pair-sat-upperize*)
qed
qed
qed

4.4.9 If

lemma *if-cases*:

assumes *red* $(\text{Cif } b \ C1 \ C2) (s, h) \ C' (s', h')$

and $C' = C1 \implies s = s' \wedge h = h' \implies \text{bdenot } b \ s \implies P$

```

    and  $C' = C2 \implies s = s' \wedge h = h' \implies \neg \text{bdenot } b \ s \implies P$ 
  shows  $P$ 
  using  $\text{assms}(1)$ 
  apply (rule  $\text{red.cases}$ )
  apply  $\text{blast+}$ 
  using  $\text{assms}(2)$  apply  $\text{fastforce}$ 
  using  $\text{assms}(3)$  apply  $\text{fastforce}$ 
  apply  $\text{blast+}$ 
  done

lemma if-safe-None:
  fixes  $\Delta :: ('i, 'a, \text{nat}) \text{ cont}$ 

  assumes  $\text{bdenot } b \ s \implies \text{safe } n \ \Delta \ C1 \ (s, h) \ S$ 
    and  $\neg \text{bdenot } b \ s \implies \text{safe } n \ \Delta \ C2 \ (s, h) \ S$ 
    and  $\Delta = \text{None}$ 
  shows  $\text{safe } (\text{Suc } n) \ (\text{None} :: ('i, 'a, \text{nat}) \text{ cont}) \ (\text{Cif } b \ C1 \ C2) \ (s, h) \ S$ 
proof (rule  $\text{safeNoneI}$ )
  show  $\text{Cif } b \ C1 \ C2 = \text{Cskip} \implies (s, h) \in S$  by  $\text{simp}$ 
  show  $\text{no-abort } (\text{None} :: ('i, 'a, \text{nat}) \text{ cont}) \ (\text{Cif } b \ C1 \ C2) \ s \ h$ 
  proof (rule  $\text{no-abortNoneI}$ )
    fix  $hf \ H$  assume  $\text{Some } H = \text{Some } h \oplus \text{Some } hf \wedge \text{full-ownership } (\text{get-fh } H) \wedge$ 
     $\text{no-guard } H$ 
    show  $\neg \text{aborts } (\text{Cif } b \ C1 \ C2) \ (s, \text{FractionalHeap.normalize } (\text{get-fh } H))$ 
    proof (rule  $\text{ccontr}$ )
      assume  $\neg \neg \text{aborts } (\text{Cif } b \ C1 \ C2) \ (s, \text{FractionalHeap.normalize } (\text{get-fh } H))$ 
      then have  $\text{aborts } (\text{Cif } b \ C1 \ C2) \ (s, \text{FractionalHeap.normalize } (\text{get-fh } H))$  by
     $\text{simp}$ 
      then show  $\text{False}$ 
      by (rule  $\text{aborts.cases}$ ) auto
    qed
  qed
  fix  $H \ hf \ C' \ s' \ h'$ 
  assume  $\text{asm0}: \text{Some } H = \text{Some } h \oplus \text{Some } hf \wedge \text{full-ownership } (\text{get-fh } H) \wedge$ 
   $\text{no-guard } H$ 
   $\wedge \text{red } (\text{Cif } b \ C1 \ C2) \ (s, \text{FractionalHeap.normalize } (\text{get-fh } H)) \ C' \ (s', h')$ 
  show  $\exists h'' \ H'$ .
     $\text{full-ownership } (\text{get-fh } H') \wedge$ 
     $\text{no-guard } H' \wedge h' = \text{FractionalHeap.normalize } (\text{get-fh } H') \wedge \text{Some } H' =$ 
     $\text{Some } h'' \oplus \text{Some } hf \wedge \text{safe } n \ (\text{None} :: ('i, 'a, \text{nat}) \text{ cont}) \ C' \ (s', h'') \ S$ 
  by ( $\text{metis } \text{asm0} \ \text{assms}(1) \ \text{assms}(2) \ \text{assms}(3) \ \text{if-cases}$ )
  qed

lemma if-safe-Some:
  assumes  $\text{bdenot } b \ s \implies \text{safe } n \ (\text{Some } \Gamma) \ C1 \ (s, h) \ S$ 
    and  $\neg \text{bdenot } b \ s \implies \text{safe } n \ (\text{Some } \Gamma) \ C2 \ (s, h) \ S$ 
  shows  $\text{safe } (\text{Suc } n) \ (\text{Some } \Gamma) \ (\text{Cif } b \ C1 \ C2) \ (s, h) \ S$ 
proof (rule  $\text{safeSomeI}$ )
  show  $\text{Cif } b \ C1 \ C2 = \text{Cskip} \implies (s, h) \in S$  by  $\text{simp}$ 

```

```

show no-abort (Some  $\Gamma$ ) (Cif b C1 C2) s h
proof (rule no-abortSomeI)
  fix H hf hj v0
  assume asm0: Some H = Some h  $\oplus$  Some hj  $\oplus$  Some hf  $\wedge$  full-ownership
    (get-fh H)  $\wedge$  semi-consistent  $\Gamma$  v0 H  $\wedge$  sat-inv s hj  $\Gamma$ 
  show  $\neg$  aborts (Cif b C1 C2) (s, FractionalHeap.normalize (get-fh H))
  proof (rule ccontr)
    assume  $\neg \neg$  aborts (Cif b C1 C2) (s, FractionalHeap.normalize (get-fh H))
    then have aborts (Cif b C1 C2) (s, FractionalHeap.normalize (get-fh H)) by
      simp
    then show False
      by (rule aborts.cases) auto
    qed
  qed
  fix H hf C' s' h' hj v0
  assume asm0: Some H = Some h  $\oplus$  Some hj  $\oplus$  Some hf  $\wedge$  full-ownership (get-fh
    H)  $\wedge$  semi-consistent  $\Gamma$  v0 H
   $\wedge$  sat-inv s hj  $\Gamma$   $\wedge$  red (Cif b C1 C2) (s, FractionalHeap.normalize (get-fh H))
  C' (s', h')
  show  $\exists$  h'' H' hj'.
    full-ownership (get-fh H')  $\wedge$ 
    semi-consistent  $\Gamma$  v0 H'  $\wedge$ 
    sat-inv s' hj'  $\Gamma$   $\wedge$  h' = FractionalHeap.normalize (get-fh H')  $\wedge$  Some H' =
    Some h''  $\oplus$  Some hj'  $\oplus$  Some hf  $\wedge$  safe n (Some  $\Gamma$ ) C' (s', h'') S
  by (metis asm0 assms(1) assms(2) if-cases)
qed

```

```

lemma if-safe:
  fixes  $\Delta$  :: ('i, 'a, nat) cont
  assumes bdenot b s  $\implies$  safe n  $\Delta$  C1 (s, h) S
  and  $\neg$  bdenot b s  $\implies$  safe n  $\Delta$  C2 (s, h) S
  shows safe (Suc n)  $\Delta$  (Cif b C1 C2) (s, h) S
apply (cases  $\Delta$ )
using assms(1) assms(2) if-safe-None apply blast
using assms(1) assms(2) if-safe-Some by blast

```

```

theorem if1-rule:
  fixes  $\Delta$  :: ('i, 'a, nat) cont
  assumes hoare-triple-valid  $\Delta$  (And P (Bool b)) C1 Q
  and hoare-triple-valid  $\Delta$  (And P (Bool (Bnot b))) C2 Q
  shows hoare-triple-valid  $\Delta$  (And P (Low b)) (Cif b C1 C2) Q
proof –

```

```

  obtain  $\Sigma t$  where safe-t:  $\bigwedge \sigma$  n.  $\sigma, \sigma \models$  And P (Bool b)  $\implies$  safe n  $\Delta$  C1  $\sigma$  ( $\Sigma t$ 
     $\sigma$ )
   $\bigwedge \sigma$   $\sigma'$ .  $\sigma, \sigma' \models$  And P (Bool b)  $\implies$  pair-sat ( $\Sigma t$   $\sigma$ ) ( $\Sigma t$   $\sigma'$ ) Q
  using assms(1) hoare-triple-validE by blast
  obtain  $\Sigma f$  where safe-f:  $\bigwedge \sigma$  n.  $\sigma, \sigma \models$  And P (Bool (Bnot b))  $\implies$  safe n  $\Delta$ 

```

```

C2  $\sigma$  ( $\Sigma f$   $\sigma$ )
   $\bigwedge \sigma \sigma'. \sigma, \sigma' \models \text{And } P \text{ (Bool (Bnot } b)) \implies \text{pair-sat } (\Sigma f \sigma) (\Sigma f \sigma') Q$ 
  using assms(2) hoare-triple-validE by blast

define  $\Sigma$  where  $\Sigma = (\lambda \sigma. \text{if bdenot } b \text{ (fst } \sigma) \text{ then } \Sigma t \sigma \text{ else } \Sigma f \sigma)$ 

show ?thesis
proof (rule hoare-triple-valid-smallerI)

  show  $\bigwedge \sigma n. \sigma, \sigma \models \text{And } P \text{ (Low } b) \implies \text{safe } n \Delta \text{ (Cif } b \text{ C1 C2)} \sigma (\Sigma \sigma)$ 
  proof -
    fix  $\sigma n$ 
    assume asm0:  $\sigma, \sigma \models \text{And } P \text{ (Low } b)$ 
    show  $\text{safe } n \Delta \text{ (Cif } b \text{ C1 C2)} \sigma (\Sigma \sigma)$ 
    proof (cases bdenot b (fst  $\sigma$ ))
      case True
        then have  $\text{safe } n \Delta \text{ C1 } \sigma (\Sigma \sigma)$ 
        by (metis  $\Sigma$ -def asm0 hyper-sat.simps(1) hyper-sat.simps(3) prod.exhaust-sel safe-t(1))
        then show ?thesis
          by (metis (no-types, lifting) Suc-n-not-le-n True if-safe nat-le-linear prod.exhaust-sel safe-smaller)
      next
        case False
          then have  $\text{safe } n \Delta \text{ C2 } \sigma (\Sigma \sigma)$ 
          by (metis  $\Sigma$ -def asm0 bdenot.simps(3) hyper-sat.simps(1) hyper-sat.simps(3) prod.exhaust-sel safe-f(1))
          then show ?thesis
            by (metis (mono-tags) False Suc-n-not-le-n if-safe nat-le-linear prod.exhaust-sel safe-smaller)
    qed
  qed
  fix  $\sigma \sigma'$  assume asm0:  $\sigma, \sigma' \models \text{And } P \text{ (Low } b)$ 
  show  $\text{pair-sat } (\Sigma \sigma) (\Sigma \sigma') Q$ 
  proof (cases bdenot b (fst  $\sigma$ ))
    case True
      then show ?thesis
        by (metis (no-types, lifting)  $\Sigma$ -def asm0 hyper-sat.simps(1) hyper-sat.simps(3) hyper-sat.simps(5) prod.exhaust-sel safe-t(2))
    next
      case False
        then show ?thesis
          by (metis (no-types, lifting)  $\Sigma$ -def asm0 bdenot.simps(3) hyper-sat.simps(1) hyper-sat.simps(3) hyper-sat.simps(5) prod.exhaust-sel safe-f(2))
    qed
  qed
  qed
  theorem if2-rule:

```



```

fixes  $\Delta :: ('i, 'a, nat) cont$ 
assumes hoare-triple-valid  $\Delta (And P (Bool b)) C1 Q$ 
and hoare-triple-valid  $\Delta (And P (Bool (Bnot b))) C2 Q$ 
and unary  $Q$ 
shows hoare-triple-valid  $\Delta P (Cif b C1 C2) Q$ 
proof –
obtain  $\Sigma t$  where safe-t:  $\bigwedge \sigma n. \sigma, \sigma \models And P (Bool b) \implies safe\ n\ \Delta\ C1\ \sigma\ (\Sigma t\ \sigma)$ 
 $\bigwedge \sigma \sigma'. \sigma, \sigma' \models And P (Bool b) \implies pair\text{-}sat\ (\Sigma t\ \sigma)\ (\Sigma t\ \sigma')\ Q$ 
using assms(1) hoare-triple-validE by blast
obtain  $\Sigma f$  where safe-f:  $\bigwedge \sigma n. \sigma, \sigma \models And P (Bool (Bnot b)) \implies safe\ n\ \Delta\ C2\ \sigma\ (\Sigma f\ \sigma)$ 
 $\bigwedge \sigma \sigma'. \sigma, \sigma' \models And P (Bool (Bnot b)) \implies pair\text{-}sat\ (\Sigma f\ \sigma)\ (\Sigma f\ \sigma')\ Q$ 
using assms(2) hoare-triple-validE by blast

define  $\Sigma$  where  $\Sigma = (\lambda \sigma. if\ bdenot\ b\ (fst\ \sigma)\ then\ \Sigma t\ \sigma\ else\ \Sigma f\ \sigma)$ 

show ?thesis
proof (rule hoare-triple-valid-smallerI)

show  $\bigwedge \sigma n. \sigma, \sigma \models P \implies safe\ n\ \Delta\ (Cif\ b\ C1\ C2)\ \sigma\ (\Sigma\ \sigma)$ 
proof –
fix  $\sigma n$ 
assume asm0:  $\sigma, \sigma \models P$ 
show  $safe\ n\ \Delta\ (Cif\ b\ C1\ C2)\ \sigma\ (\Sigma\ \sigma)$ 
proof (cases bdenot b (fst sigma))
case True
then have  $safe\ n\ \Delta\ C1\ \sigma\ (\Sigma\ \sigma)$ 
by (metis  $\Sigma$ -def asm0 hyper-sat.simps(1) hyper-sat.simps(3) prod.exhaust-sel safe-t(1))
then show ?thesis
by (metis (no-types, lifting) Suc-n-not-le-n True if-safe nat-le-linear prod.exhaust-sel safe-smaller)
next
case False
then have  $safe\ n\ \Delta\ C2\ \sigma\ (\Sigma\ \sigma)$ 
by (metis  $\Sigma$ -def asm0 bdenot.simps(3) hyper-sat.simps(1) hyper-sat.simps(3) prod.exhaust-sel safe-f(1))
then show ?thesis
by (metis (mono-tags) False Suc-n-not-le-n if-safe nat-le-linear prod.exhaust-sel safe-smaller)
qed
qed
fix  $\sigma 1\ \sigma 2$  assume asm0:  $\sigma 1, \sigma 2 \models P$ 
then have asm0-bis:  $\sigma 2, \sigma 1 \models P$ 
by (simp add: sat-comm)
show  $pair\text{-}sat\ (\Sigma\ \sigma 1)\ (\Sigma\ \sigma 2)\ Q$ 
proof (rule pair-sat-smallerI)
fix  $\sigma 1'\ \sigma 2'$ 

```

```

assume asm1:  $\sigma 1' \in \Sigma \sigma 1 \wedge \sigma 2' \in \Sigma \sigma 2$ 
then have  $\sigma 1', \sigma 1' \models Q$ 
  apply (cases bdenot b (fst  $\sigma 1$ ))
  apply (metis (no-types, lifting)  $\Sigma$ -def always-sat-refl asm0 hyper-sat.simps(1) hyper-sat.simps(3) pair-sat-def safe-t(2) surjective-pairing)
    by (metis (no-types, lifting)  $\Sigma$ -def always-sat-refl asm0 bdenot.simps(3) hyper-sat.simps(1) hyper-sat.simps(3) pair-satE prod.collapse safe-f(2))
  moreover have  $\sigma 2', \sigma 2' \models Q$ 
    apply (cases bdenot b (fst  $\sigma 2$ ))
    apply (metis (mono-tags)  $\Sigma$ -def always-sat-refl asm0-bis asm1 entailsI entails-def fst-conv hyper-sat.simps(1) hyper-sat.simps(3) old.prod.exhaust pair-sat-def safe-t(2))
      using  $\Sigma$ -def always-sat-refl asm0-bis bdenot.simps(3) hyper-sat.simps(1) hyper-sat.simps(3) pair-satE prod.collapse safe-f(2)
        by (metis (no-types, lifting) asm1)
    ultimately show  $\sigma 1', \sigma 2' \models Q$ 
      by (metis assms(3) eq-fst-iff unaryE)
qed
qed
qed

```

4.4.10 Sequential composition

inductive-cases *red-seq-cases*: *red (Cseq C1 C2) σ C' σ'*

lemma *aborts-seq-aborts-C1*:

```

assumes aborts (Cseq C1 C2)  $\sigma$ 
shows aborts C1  $\sigma$ 
using aborts.simps assms cmd.inject(6) by blast

```

lemma *safe-seq-None*:

```

assumes safe n (None :: ('i, 'a, nat) cont) C1 (s, h) S1
  and  $\bigwedge m s' h'. m \leq n \wedge (s', h') \in S1 \implies \text{safe } m \text{ (None :: ('i, 'a, nat) cont) } C2 (s', h') S2$ 
shows safe n (None :: ('i, 'a, nat) cont) (Cseq C1 C2) (s, h) S2
using assms
proof (induct n arbitrary: C1 s h)
  case (Suc n)
  show ?case
  proof (rule safeNoneI)
    show no-abort (None :: ('i, 'a, nat) cont) (Cseq C1 C2) s h
      by (meson Suc.premis(1) aborts-seq-aborts-C1 no-abort.simps(1) safeNoneE-bis(2))
    fix H hf C' s' h'
    assume asm0: Some H = Some h  $\oplus$  Some hf  $\wedge$ 
      full-ownership (get-fh H)  $\wedge$  no-guard H  $\wedge$  red (Cseq C1 C2) (s, Fractional-Heap.normalize (get-fh H)) C' (s', h')
    show  $\exists h'' H'$ .
      full-ownership (get-fh H')  $\wedge$ 

```

$\text{no-guard } H' \wedge h' = \text{FractionalHeap.normalize } (\text{get-fh } H') \wedge \text{Some } H' =$
 $\text{Some } h'' \oplus \text{Some } hf \wedge \text{safe } n \text{ (None :: ('i, 'a, nat) cont) } C' (s', h'') S2$
proof (rule red-seq-cases)
show red (Cseq C1 C2) (s, FractionalHeap.normalize (get-fh H)) C' (s', h')
using asm0 by blast
show C1 = Cskip \implies
 $C' = C2 \implies$
 $(s', h') = (s, \text{FractionalHeap.normalize } (\text{get-fh } H)) \implies$
 $\exists h'' H'. \text{full-ownership } (\text{get-fh } H') \wedge$
 $\text{no-guard } H' \wedge h' = \text{FractionalHeap.normalize } (\text{get-fh } H') \wedge \text{Some } H' = \text{Some}$
 $h'' \oplus \text{Some } hf \wedge \text{safe } n \text{ (None :: ('i, 'a, nat) cont) } C' (s', h'') S2$
using Suc.prem1(1) Suc.prem2(2) asm0 order-refl prod.inject safeNoneE-bis(1)
by (metis le-SucI)
fix C1' **assume** C' = Cseq C1' C2 red C1 (s, FractionalHeap.normalize
(get-fh H)) C1' (s', h')
obtain H' h'' **where** asm1: full-ownership (get-fh H') no-guard H' h' =
FractionalHeap.normalize (get-fh H')
 $\text{Some } H' = \text{Some } h'' \oplus \text{Some } hf \text{ safe } n \text{ (None :: ('i, 'a, nat) cont) } C1' (s',$
 $h'') S1$
using Suc(2) safeNoneE(3)[of n C1 s h S1 H hf C1' s' h']
using <red C1 (s, FractionalHeap.normalize (get-fh H)) C1' (s', h')> asm0
by blast
moreover **have** safe n (None :: ('i, 'a, nat) cont) (Cseq C1' C2) (s', h'') S2
using Suc.hyps Suc.prem2(2) calculation(5)
using le-Suc-eq **by** presburger
ultimately **show** ?thesis
using <C' = Cseq C1' C2> **by** blast
qed
qed (simp)
qed (simp)

lemma safe-seq-Some:

assumes safe n (Some Γ) C1 (s, h) S1
and $\bigwedge m s' h'. m \leq n \wedge (s', h') \in S1 \implies \text{safe } m \text{ (Some } \Gamma) C2 (s', h'') S2$
shows safe n (Some Γ) (Cseq C1 C2) (s, h) S2
using assms
proof (induct n arbitrary: C1 s h)
case (Suc n)
show ?case
proof (rule safeSomeI)
show no-abort (Some Γ) (Cseq C1 C2) s h
by (meson Suc.prem1(1) aborts-seq-aborts-C1 no-abort.simps(2) safeSomeE(2))
fix H hf C' s' h' hj v0
assume asm0: Some H = Some h \oplus Some hj \oplus Some hf \wedge
full-ownership (get-fh H) \wedge semi-consistent Γ v0 H \wedge sat-inv s hj Γ \wedge red
(Cseq C1 C2) (s, FractionalHeap.normalize (get-fh H)) C' (s', h')
show $\exists h'' H' hj'. \text{full-ownership } (\text{get-fh } H') \wedge$
semi-consistent Γ v0 H' \wedge sat-inv s' hj' Γ \wedge h' = FractionalHeap.normalize
(get-fh H') \wedge Some H' = Some h'' \oplus Some hj' \oplus Some hf \wedge safe n (Some Γ) C'

$(s', h'') S2$
proof (*rule red-seq-cases*)
show $\text{red } (C\text{seq } C1 \ C2) \ (s, \text{FractionalHeap.normalize } (\text{get-fh } H)) \ C' \ (s', h')$
using *asm0 by blast*
show $C1 = C\text{skip} \implies$
 $C' = C2 \implies$
 $(s', h') = (s, \text{FractionalHeap.normalize } (\text{get-fh } H)) \implies \exists h'' \ H' \ hj'. \text{full-ownership}$
 $(\text{get-fh } H') \wedge$
 $\text{semi-consistent } \Gamma \ v0 \ H' \wedge \text{sat-inv } s' \ hj' \ \Gamma \wedge h' = \text{FractionalHeap.normalize}$
 $(\text{get-fh } H')$
 $\wedge \text{Some } H' = \text{Some } h'' \oplus \text{Some } hj' \oplus \text{Some } hf \wedge \text{safe } n \ (\text{Some } \Gamma) \ C' \ (s', h'')$
 $S2$
using *Pair-inject Suc.premis(1) Suc-n-not-le-n asm0 assms(2) not-less-eq-eq*
safeSomeE(1)
by (*metis (no-types, lifting) Suc.premis(2) nat-le-linear*)
fix $C1'$ **assume** $C' = C\text{seq } C1' \ C2 \ \text{red } C1 \ (s, \text{FractionalHeap.normalize}$
 $(\text{get-fh } H)) \ C1' \ (s', h')$
obtain $H' \ h'' \ hj'$ **where** *asm1: full-ownership (get-fh H') \wedge*
 $\text{semi-consistent } \Gamma \ v0 \ H' \wedge \text{sat-inv } s' \ hj' \ \Gamma \wedge h' = \text{FractionalHeap.normalize}$
 $(\text{get-fh } H') \wedge \text{Some } H' = \text{Some } h'' \oplus \text{Some } hj' \oplus \text{Some } hf \wedge \text{safe } n \ (\text{Some } \Gamma) \ C1'$
 $(s', h'') \ S1$
using *Suc(2) safeSomeE(3)[of n $\Gamma \ C1 \ s \ h \ S1 \ H \ hj \ hf \ v0 \ C1' \ s' \ h']$*
using $\langle \text{red } C1 \ (s, \text{FractionalHeap.normalize } (\text{get-fh } H)) \ C1' \ (s', h') \rangle \ \text{asm0}$
by *blast*
moreover **have** $\text{safe } n \ (\text{Some } \Gamma) \ (C\text{seq } C1' \ C2) \ (s', h'') \ S2$
by (*simp add: Suc.hyps Suc.premis(2) calculation*)
ultimately show *?thesis*
using $\langle C' = C\text{seq } C1' \ C2 \rangle$ **by** *blast*
qed
qed (*simp*)
qed (*simp*)

lemma *seq-safe:*

fixes $\Delta :: ('i, 'a, \text{nat}) \ \text{cont}$
assumes $\text{safe } n \ \Delta \ C1 \ (s, h) \ S1$
and $\bigwedge m \ s' \ h'. \ m \leq n \wedge (s', h') \in S1 \implies \text{safe } m \ \Delta \ C2 \ (s', h') \ S2$
shows $\text{safe } n \ \Delta \ (C\text{seq } C1 \ C2) \ (s, h) \ S2$
apply (*cases Δ*)
using *assms(1) assms(2) safe-seq-None* **apply** *blast*
using *assms(1) assms(2) safe-seq-Some* **by** *blast*

theorem *seq-rule:*

fixes $\Delta :: ('i, 'a, \text{nat}) \ \text{cont}$
assumes *hoare-triple-valid $\Delta \ P \ C1 \ R$*
and *hoare-triple-valid $\Delta \ R \ C2 \ Q$*
shows *hoare-triple-valid $\Delta \ P \ (C\text{seq } C1 \ C2) \ Q$*

proof –

obtain $\Sigma 1$ **where** *safe-1: $\bigwedge \sigma \ n. \ \sigma, \sigma \models P \implies \text{safe } n \ \Delta \ C1 \ \sigma \ (\Sigma 1 \ \sigma)$*
 $\bigwedge \sigma \ \sigma'. \ \sigma, \sigma' \models P \implies \text{pair-sat } (\Sigma 1 \ \sigma) \ (\Sigma 1 \ \sigma') \ R$

```

using assms(1) hoare-triple-validE by blast
obtain  $\Sigma 2$  where safe-2:  $\bigwedge \sigma n. \sigma, \sigma \models R \implies \text{safe } n \Delta C2 \sigma (\Sigma 2 \sigma)$ 
 $\bigwedge \sigma \sigma'. \sigma, \sigma' \models R \implies \text{pair-sat } (\Sigma 2 \sigma) (\Sigma 2 \sigma') Q$ 
using assms(2) hoare-triple-validE by blast

define  $\Sigma$  where  $\Sigma = (\lambda \sigma. (\bigcup \sigma' \in \Sigma 1 \sigma. \Sigma 2 \sigma'))$ 

show ?thesis
proof (rule hoare-triple-valid-smallerI)
show  $\bigwedge \sigma n. \sigma, \sigma \models P \implies \text{safe } n \Delta (Cseq C1 C2) \sigma (\Sigma \sigma)$ 
proof –
  fix  $\sigma n$  assume asm0:  $\sigma, \sigma \models P$ 
  then have pair-sat ( $\Sigma 1 \sigma$ ) ( $\Sigma 1 \sigma$ ) R
    using safe-1(2) by blast

  have  $\text{safe } n \Delta (Cseq C1 C2) (fst \sigma, snd \sigma) (\Sigma \sigma)$ 
  proof (rule seq-safe)
    show  $\text{safe } n \Delta C1 (fst \sigma, snd \sigma) (\Sigma 1 \sigma)$ 
    by (simp add: asm0 safe-1(1))
    fix  $m s' h'$ 
    assume  $m \leq n \wedge (s', h') \in \Sigma 1 \sigma$ 
    then show  $\text{safe } m \Delta C2 (s', h') (\Sigma \sigma)$ 
    by (metis (no-types, opaque-lifting) Sup-upper  $\langle \Sigma \equiv \lambda \sigma. \bigcup (\Sigma 2 \text{ ` } \Sigma 1 \sigma) \rangle$ 
 $\langle \text{pair-sat } (\Sigma 1 \sigma) (\Sigma 1 \sigma) R \rangle$  image-iff pair-sat-def safe-2(1) safe-larger-set)
    qed
    then show  $\text{safe } n \Delta (Cseq C1 C2) \sigma (\Sigma \sigma)$  by auto
  qed
  fix  $\sigma 1 \sigma 2$ 
  assume asm0:  $\sigma 1, \sigma 2 \models P$ 
  show pair-sat ( $\Sigma \sigma 1$ ) ( $\Sigma \sigma 2$ ) Q
  proof (rule pair-sat-smallerI)
    fix  $\sigma 1'' \sigma 2''$ 
    assume asm1:  $\sigma 1'' \in \Sigma \sigma 1 \wedge \sigma 2'' \in \Sigma \sigma 2$ 
    then obtain  $\sigma 1' \sigma 2'$  where  $\sigma 1'' \in \Sigma 2 \sigma 1' \sigma 1' \in \Sigma 1 \sigma 1 \sigma 2'' \in \Sigma 2 \sigma 2'$ 
 $\sigma 2' \in \Sigma 1 \sigma 2$ 
    using  $\langle \Sigma \equiv \lambda \sigma. \bigcup (\Sigma 2 \text{ ` } \Sigma 1 \sigma) \rangle$  by blast
    then show  $\sigma 1'', \sigma 2'' \models Q$ 
    by (meson asm0 pair-sat-def safe-1(2) safe-2(2))
  qed
qed
qed

```

4.4.11 Frame rule

lemma *safe-frame-None*:

```

assumes safe  $n$  (None :: ('i, 'a, nat) cont) C (s, h) S
and Some H = Some h  $\oplus$  Some hf0
shows safe  $n$  (None :: ('i, 'a, nat) cont) C (s, H) (add-states S  $\{(s'', hf0) \mid s''.$ 
agrees ( $- wrC C$ ) s s''})

```

```

using assms
proof (induct n arbitrary: s h H C)
  case (Suc n)
    show safe (Suc n) (None :: ('i, 'a, nat) cont) C (s, H) (add-states S {(s'', hf0) | s''. agrees (- wrC C) s s''})
    proof (rule safeNoneI)
      show  $C = Cskip \implies (s, H) \in \text{add-states } S \{(s', hf0) \mid s'. \text{agrees } (- \text{wrC } C) s s'\}$ 
      using CollectI Suc.premis(1) Suc.premis(2) add-states-def agrees-def[of - wrC C s] safeNoneE(1)[of n C s h S]
      by fast
      show no-abort (None :: ('i, 'a, nat) cont) C s H
      using Suc.premis(1) Suc.premis(2) larger-def no-abort-larger safeNoneE(2)
by blast
  fix H1 hf1 C' s' h'
  assume asm0: Some H1 = Some H  $\oplus$  Some hf1  $\wedge$  full-ownership (get-fh H1)  $\wedge$  no-guard H1  $\wedge$  red C (s, FractionalHeap.normalize (get-fh H1)) C' (s', h')
  then obtain hf where Some hf = Some hf0  $\oplus$  Some hf1
  by (metis (no-types, opaque-lifting) Suc.premis(2) option.collapse plus.simps(1) plus-asso plus-comm)
  then have Some H1 = Some h  $\oplus$  Some hf
  by (metis Suc.premis(2) asm0 plus-asso)
  then obtain h'' H' where r: full-ownership (get-fh H') no-guard H' h' = FractionalHeap.normalize (get-fh H') Some H' = Some h''  $\oplus$  Some hf safe n (None :: ('i, 'a, nat) cont) C' (s', h'') S
  using safeNoneE(3)[of n C s h S H1 hf C' s'] Suc.premis(1) asm0 by blast
  then obtain h''' where Some h''' = Some h''  $\oplus$  Some hf0
  by (metis  $\langle$ Some hf = Some hf0  $\oplus$  Some hf1 $\rangle$  not-Some-eq plus.simps(1) plus-asso)
  then have Some H' = Some h'''  $\oplus$  Some hf1
  by (metis  $\langle$ Some hf = Some hf0  $\oplus$  Some hf1 $\rangle$  plus-asso r(4))
  moreover have safe n (None :: ('i, 'a, nat) cont) C' (s', h''') (add-states S {(s'', hf0) | s''. agrees (- wrC C') s' s''})
  proof (rule Suc.hypos)
    show safe n (None :: ('i, 'a, nat) cont) C' (s', h'') S
    using r by simp
    show Some h''' = Some h''  $\oplus$  Some hf0
    by (simp add:  $\langle$ Some h''' = Some h''  $\oplus$  Some hf0 $\rangle$ )
  qed
  moreover have add-states S {(s'', hf0) | s''. agrees (- wrC C') s' s''}  $\subseteq$  add-states S {(s'', hf0) | s''. agrees (- wrC C) s s''}
  proof -
    have wrC C'  $\subseteq$  wrC C
    using asm0 red-properties(1) by blast
    have {(s'', hf0) | s''. agrees (- wrC C') s' s''}  $\subseteq$  {(s'', hf0) | s''. agrees (- wrC C) s s''}
  proof
    fix x assume x  $\in$  {(s'', hf0) | s''. agrees (- wrC C') s' s''}
    then have agrees (- wrC C') s' (fst x)  $\wedge$  snd x = hf0 by force

```

moreover have $fvC\ C' \subseteq fvC\ C \wedge wrC\ C' \subseteq wrC\ C \wedge agrees\ (-\ wrC\ C)$
s' s
using *asm0 red-properties(1)* **by force**

moreover have $agrees\ (-\ wrC\ C)\ s\ (fst\ x)$
proof (*rule agreesI*)
fix *y* **assume** $y \in -\ wrC\ C$
show $s\ y = fst\ x\ y$
by (*metis (no-types, lifting) Compl-subset-Compl-iff* $\langle y \in -\ wrC\ C \rangle$
agrees-def calculation(1) calculation(2) in-mono)
qed
then show $x \in \{(s'', hf0) \mid s''.\ agrees\ (-\ wrC\ C)\ s\ s''\}$
using $\langle agrees\ (-\ wrC\ C')\ s'\ (fst\ x) \wedge snd\ x = hf0 \rangle$ **by force**
qed
then show *?thesis*
by (*metis (no-types, lifting) add-states-comm add-states-subset*)
qed

ultimately have $safe\ n\ (None :: ('i, 'a, nat)\ cont)\ C'\ (s', h''')$ (*add-states S*
 $\{(s'', hf0) \mid s''.\ agrees\ (-\ wrC\ C)\ s\ s''\}$)
using *safe-larger-set* **by blast**
then show $\exists h''\ H'$
 $full-ownership\ (get-fh\ H') \wedge$
 $no-guard\ H' \wedge$
 $h' = FractionalHeap.normalize\ (get-fh\ H') \wedge Some\ H' = Some\ h'' \oplus Some$
 $hf1 \wedge safe\ n\ (None :: ('i, 'a, nat)\ cont)\ C'\ (s', h'')$ (*add-states S* $\{(s'', hf0) \mid s''.$
 $agrees\ (-\ wrC\ C)\ s\ s''\}$)
using $\langle Some\ H' = Some\ h''' \oplus Some\ hf1 \rangle\ r(1)\ r(2)\ r(3)$ **by blast**
qed
qed (*simp*)

lemma *safe-frame-Some*:
assumes $safe\ n\ (Some\ \Gamma)\ C\ (s, h)\ S$
and $Some\ H = Some\ h \oplus Some\ hf0$
shows $safe\ n\ (Some\ \Gamma)\ C\ (s, H)\ (add-states\ S\ \{(s'', hf0) \mid s''.\ agrees\ (-\ wrC$
 $C)\ s\ s''\})$
using *assms*
proof (*induct n arbitrary: s h H C*)
case (*Suc n*)
let $?R = \{(s'', hf0) \mid s''.\ agrees\ (-\ wrC\ C)\ s\ s''\}$
show $safe\ (Suc\ n)\ (Some\ \Gamma)\ C\ (s, H)\ (add-states\ S\ ?R)$
proof (*rule safeSomeI*)
show $C = Cskip \implies (s, H) \in add-states\ S\ ?R$
using *CollectI Suc.prem(1) Suc.prem(2) add-states-def[of S ?R] agrees-def[of*
 $-\ wrC\ C\ s]$
 $safeSomeE(1)[of\ n\ \Gamma\ C\ s\ h\ S]$ **by fast**
show *no-abort* ($Some\ \Gamma$) $C\ s\ H$
using *Suc.prem(1) Suc.prem(2) larger-def no-abort-larger safeSomeE(2)*
by blast
fix $H1\ hf1\ C'\ s'\ h'\ hj\ v0$

assume $asm0$: $Some\ H1 = Some\ H \oplus Some\ hj \oplus Some\ hf1 \wedge$
 $full\text{-ownership}\ (get\text{-fh}\ H1) \wedge semi\text{-consistent}\ \Gamma\ v0\ H1 \wedge sat\text{-inv}\ s\ hj\ \Gamma \wedge red$
 $C\ (s, FractionalHeap.normalize\ (get\text{-fh}\ H1))\ C'\ (s', h')$
then obtain hf **where** $Some\ hf = Some\ hf0 \oplus Some\ hf1$
by $(metis\ (no\text{-types},\ opaque\text{-lifting})\ Suc.prem\ s(2)\ option.collapse\ plus.simps(1))$
 $plus\text{-asso}\ plus\text{-comm}$
then have $Some\ H1 = Some\ h \oplus Some\ hj \oplus Some\ hf$
by $(metis\ (no\text{-types},\ opaque\text{-lifting})\ Suc.prem\ s(2)\ asm0\ plus\text{-asso}\ plus\text{-comm})$

then obtain $h''\ H'\ hj'$ **where** r : $full\text{-ownership}\ (get\text{-fh}\ H') \wedge$
 $semi\text{-consistent}\ \Gamma\ v0\ H' \wedge sat\text{-inv}\ s'\ hj'\ \Gamma \wedge h' = FractionalHeap.normalize$
 $(get\text{-fh}\ H') \wedge Some\ H' = Some\ h'' \oplus Some\ hj' \oplus Some\ hf \wedge safe\ n\ (Some\ \Gamma)\ C'$
 $(s', h'')\ S$
using $safeSomeE(3)[of\ n\ \Gamma\ C\ s\ h\ S\ H1\ hj\ hf\ v0\ C'\ s'\ h'']\ Suc.prem\ s(1)\ asm0$
by $blast$

then obtain h''' **where** $Some\ h''' = Some\ h'' \oplus Some\ hf0$
by $(metis\ (no\text{-types},\ lifting)\ \langle Some\ hf = Some\ hf0 \oplus Some\ hf1 \rangle\ plus.simps(2))$
 $plus.simps(3)\ plus\text{-asso}\ plus\text{-comm}$
then have $Some\ H' = Some\ h''' \oplus Some\ hj' \oplus Some\ hf1$
by $(metis\ (no\text{-types},\ lifting)\ \langle Some\ hf = Some\ hf0 \oplus Some\ hf1 \rangle\ plus\text{-asso}$
 $plus\text{-comm}\ r)$
moreover have $safe\ n\ (Some\ \Gamma)\ C'\ (s', h''')$ $(add\text{-states}\ S\ \{(s'', hf0) | s''.$
 $agrees\ (-\ wrC\ C')\ s'\ s''\})$
proof $(rule\ Suc.hyps)$
show $safe\ n\ (Some\ \Gamma)\ C'\ (s', h'')\ S$
using r **by** $simp$
show $Some\ h''' = Some\ h'' \oplus Some\ hf0$
by $(simp\ add: \langle Some\ h''' = Some\ h'' \oplus Some\ hf0 \rangle)$
qed
moreover have $add\text{-states}\ S\ \{(s'', hf0) | s''.$ $agrees\ (-\ wrC\ C')\ s'\ s''\} \subseteq$
 $add\text{-states}\ S\ \{(s'', hf0) | s''.$ $agrees\ (-\ wrC\ C)\ s\ s''\}$
proof $-$
have $wrC\ C' \subseteq wrC\ C$
using $asm0\ red\text{-properties}(1)$ **by** $blast$
have $\{(s'', hf0) | s''.$ $agrees\ (-\ wrC\ C')\ s'\ s''\} \subseteq \{(s'', hf0) | s''.$ $agrees\ (-$
 $wrC\ C)\ s\ s''\}$
proof
fix x **assume** $x \in \{(s'', hf0) | s''.$ $agrees\ (-\ wrC\ C')\ s'\ s''\}$
then have $agrees\ (-\ wrC\ C')\ s'\ (fst\ x) \wedge snd\ x = hf0$ **by** $force$
moreover have $fvC\ C' \subseteq fvC\ C \wedge wrC\ C' \subseteq wrC\ C \wedge agrees\ (-\ wrC\ C)$
 $s'\ s$
using $asm0\ red\text{-properties}(1)$ **by** $force$
moreover have $agrees\ (-\ wrC\ C)\ s\ (fst\ x)$
proof $(rule\ agreesI)$
fix y **assume** $y \in -\ wrC\ C$
then show $s\ y = fst\ x\ y$
by $(metis\ (mono\text{-tags},\ opaque\text{-lifting})\ Compl\text{-iff}\ agrees\text{-def}\ calculation(1))$

calculation(2) *in-mono*
qed
then show $x \in \{(s'', hf0) \mid s''. \text{ agrees } (- \text{ wrC } C) s s''\}$
using $\langle \text{ agrees } (- \text{ wrC } C') s' (\text{fst } x) \wedge \text{ snd } x = hf0 \rangle$ **by force**
qed
then show *?thesis*
by (*metis* (*no-types*, *lifting*) *add-states-comm* *add-states-subset*)
qed
ultimately have *safe n* (*Some* Γ) $C' (s', h''')$ (*add-states S ?R*)
using *safe-larger-set* **by blast**
then show $\exists h'' H' hj'. \text{ full-ownership } (\text{get-fh } H') \wedge$
semi-consistent $\Gamma v0 H' \wedge$
sat-inv $s' hj' \Gamma \wedge h' = \text{FractionalHeap.normalize } (\text{get-fh } H') \wedge \text{Some } H' =$
Some $h'' \oplus \text{Some } hj' \oplus \text{Some } hf1 \wedge \text{safe n } (\text{Some } \Gamma) C' (s', h'')$ (*add-states S ?R*)
using $\langle \text{Some } H' = \text{Some } h''' \oplus \text{Some } hj' \oplus \text{Some } hf1 \rangle r$ **by blast**
qed
qed (*simp*)

lemma *safe-frame*:

fixes $\Delta :: ('i, 'a, \text{nat}) \text{ cont}$
assumes *safe n* $\Delta C (s, h) S$
and *Some* $H = \text{Some } h \oplus \text{Some } hf0$
shows *safe n* $\Delta C (s, H)$ (*add-states S* $\{(s'', hf0) \mid s''. \text{ agrees } (- \text{ wrC } C) s s''\}$)
apply (*cases* Δ)
using *assms*(1) *assms*(2) *safe-frame-None* **apply blast**
using *assms*(1) *assms*(2) *safe-frame-Some* **by blast**

theorem *frame-rule*:

fixes $\Delta :: ('i, 'a, \text{nat}) \text{ cont}$
assumes *hoare-triple-valid* $\Delta P C Q$
and *disjoint* (*fvA* R) (*wrC* C)
and *precise* $P \vee \text{precise } R$
shows *hoare-triple-valid* $\Delta (\text{Star } P R) C (\text{Star } Q R)$

proof –

obtain Σ **where** *asm0*: $\bigwedge \sigma n. \sigma, \sigma \models P \implies \text{safe n } \Delta C \sigma (\Sigma \sigma) \wedge \sigma \sigma'. \sigma, \sigma' \models P \implies \text{pair-sat } (\Sigma \sigma) (\Sigma \sigma') Q$
using *assms*(1) *hoare-triple-validE* **by blast**

define *pairs* **where** $\text{pairs} = (\lambda \sigma. \{ (p, r) \mid p r. \text{Some } (\text{snd } \sigma) = \text{Some } p \oplus \text{Some } r \wedge (\text{fst } \sigma, p), (\text{fst } \sigma, p) \models P \wedge (\text{fst } \sigma, r), (\text{fst } \sigma, r) \models R \})$

define Σ' **where** $\Sigma' = (\lambda \sigma. (\bigcup (p, r) \in \text{pairs } \sigma. \text{add-states } (\Sigma (\text{fst } \sigma, p)) \{(s'', r) \mid s''. \text{ agrees } (- \text{ wrC } C) (\text{fst } \sigma) s''\}))$

show *?thesis*

proof (*rule* *hoare-triple-validI*)

show $\bigwedge s h n. (s, h), (s, h) \models \text{Star } P R \implies \text{safe n } \Delta C (s, h) (\Sigma' (s, h))$

proof –

fix $s\ h\ n$ **assume** $asm1: (s, h), (s, h) \models Star\ P\ R$
then obtain $p\ r$ **where** $Some\ h = Some\ p \oplus Some\ r$ $(s, p), (s, p) \models P$ $(s, r), (s, r) \models R$
using $always\text{-}sat\text{-}refl\ hyper\text{-}sat\text{-}simps(4)$ **by** $blast$
then have $safe\ n\ \Delta\ C\ (s, p)\ (\Sigma\ (s, p))$
using $asm0(1)$ **by** $blast$
then have $safe\ n\ \Delta\ C\ (s, h)$ $(add\text{-}states\ (\Sigma\ (s, p))\ \{(s'', r) \mid s''.\ agrees\ (-\ wrC\ C)\ s\ s''\})$
using $safe\text{-}frame[of\ n\ \Delta\ C\ s\ p\ \Sigma\ (s, p)\ h\ r]$ $\langle Some\ h = Some\ p \oplus Some\ r \rangle$
by $blast$
moreover have $(add\text{-}states\ (\Sigma\ (s, p))\ \{(s'', r) \mid s''.\ agrees\ (-\ wrC\ C)\ s\ s''\})$
 $\subseteq \Sigma' (s, h)$
proof $-$
have $(p, r) \in pairs\ (s, h)$
using $\langle (s, p), (s, p) \models P \rangle$ $\langle (s, r), (s, r) \models R \rangle$ $\langle Some\ h = Some\ p \oplus Some\ r \rangle$
 $r \rangle$ $pairs\text{-}def$ **by** $force$
then show $?thesis$
using $\Sigma'\text{-}def$ **by** $auto$
qed
ultimately show $safe\ n\ \Delta\ C\ (s, h)\ (\Sigma' (s, h))$
using $safe\text{-}larger\text{-}set$ **by** $blast$
qed

fix $s1\ h1\ s2\ h2$
assume $asm1: (s1, h1), (s2, h2) \models Star\ P\ R$
then obtain $p1\ p2\ r1\ r2$ **where** $Some\ h1 = Some\ p1 \oplus Some\ r1$ $Some\ h2 = Some\ p2 \oplus Some\ r2$
 $(s1, p1), (s2, p2) \models P$ $(s1, r1), (s2, r2) \models R$
by $auto$
then have $(s1, p1), (s1, p1) \models P \wedge (s1, r1), (s1, r1) \models R \wedge (s2, p2), (s2, p2) \models P \wedge (s2, r2), (s2, r2) \models R$
using $always\text{-}sat\text{-}refl\ sat\text{-}comm$ **by** $blast$

show $pair\text{-}sat\ (\Sigma' (s1, h1))\ (\Sigma' (s2, h2))\ (Star\ Q\ R)$
proof $(rule\ pair\text{-}satI)$
fix $s1'\ h1'\ s2'\ h2'$
assume $asm2: (s1', h1') \in \Sigma' (s1, h1) \wedge (s2', h2') \in \Sigma' (s2, h2)$
then obtain $p1'\ r1'\ p2'\ r2'$ **where** $(p1', r1') \in pairs\ (s1, h1)$ $(p2', r2') \in pairs\ (s2, h2)$
 $(s1', h1') \in add\text{-}states\ (\Sigma\ (s1, p1'))\ \{(s'', r1') \mid s''.\ agrees\ (-\ wrC\ C)\ s1\ s''\}$
 $(s2', h2') \in add\text{-}states\ (\Sigma\ (s2, p2'))\ \{(s'', r2') \mid s''.\ agrees\ (-\ wrC\ C)\ s2\ s''\}$
using $\Sigma'\text{-}def$ **by** $force$
moreover obtain $(s1, p1'), (s1, p1') \models P$ $(s1, r1'), (s1, r1') \models R$ $(s2, p2'), (s2, p2') \models P$ $(s2, r2'), (s2, r2') \models R$
 $Some\ h1 = Some\ p1' \oplus Some\ r1'$ $Some\ h2 = Some\ p2' \oplus Some\ r2'$

```

using calculation(1) calculation(2) pairs-def by auto
ultimately have  $p1 = p1' \wedge p2 = p2' \wedge r1 = r1' \wedge r2 = r2'$ 
proof (cases precise P)
  case True
    then have  $p1 = p1' \wedge p2 = p2'$  using preciseE
      by (metis  $\langle (s1, p1), (s1, p1) \models P \wedge (s1, r1), (s1, r1) \models R \wedge (s2, p2),$ 
 $(s2, p2) \models P \wedge (s2, r2), (s2, r2) \models R \rangle \langle \text{Some } h1 = \text{Some } p1 \oplus \text{Some } r1 \rangle \langle \text{Some } h2 = \text{Some } p2 \oplus \text{Some } r2 \rangle$ 
 $\langle \wedge \text{thesis. } (\llbracket (s1, p1'), (s1, p1') \models P; (s1, r1'), (s1, r1') \models R; (s2, p2'), (s2, p2') \models P; (s2, r2'), (s2, r2') \models R; \text{Some } h1 = \text{Some } p1' \oplus \text{Some } r1'; \text{Some } h2 = \text{Some } p2' \oplus \text{Some } r2' \rrbracket \implies \text{thesis}) \implies \text{thesis} \rangle$ 
larger-def)
      then show ?thesis
        by (metis  $\langle \text{Some } h1 = \text{Some } p1 \oplus \text{Some } r1 \rangle \langle \text{Some } h1 = \text{Some } p1' \oplus \text{Some } r1' \rangle \langle \text{Some } h2 = \text{Some } p2 \oplus \text{Some } r2 \rangle \langle \text{Some } h2 = \text{Some } p2' \oplus \text{Some } r2' \rangle$ 
addition-cancellative plus-comm)
    next
      case False
        then have precise R
          using assms(3) by auto
        then show ?thesis
          by (metis (no-types, opaque-lifting)  $\langle (s1, p1), (s1, p1) \models P \wedge (s1, r1), (s1, r1) \models R \wedge (s2, p2), (s2, p2) \models P \wedge (s2, r2), (s2, r2) \models R \rangle \langle \text{Some } h1 = \text{Some } p1 \oplus \text{Some } r1 \rangle \langle \text{Some } h2 = \text{Some } p2 \oplus \text{Some } r2 \rangle$ 
 $\langle \wedge \text{thesis. } (\llbracket (s1, p1'), (s1, p1') \models P; (s1, r1'), (s1, r1') \models R; (s2, p2'), (s2, p2') \models P; (s2, r2'), (s2, r2') \models R; \text{Some } h1 = \text{Some } p1' \oplus \text{Some } r1'; \text{Some } h2 = \text{Some } p2' \oplus \text{Some } r2' \rrbracket \implies \text{thesis}) \implies \text{thesis} \rangle$ 
addition-cancellative larger-def plus-comm preciseE)
        qed
      then have pair-sat  $(\Sigma (s1, p1')) (\Sigma (s2, p2')) Q$ 
        using  $\langle (s1, p1), (s2, p2) \models P \rangle$  asm0(2) by blast
      moreover have pair-sat  $\{(s'', r1') \mid s''. \text{ agrees } (- \text{ wrC } C) s1 s''\} \{(s'', r2') \mid s''. \text{ agrees } (- \text{ wrC } C) s2 s''\} R$ 
        (is pair-sat ?R1 ?R2 R)
      proof (rule pair-satI)
        fix  $s1'' r1'' s2'' r2''$  assume  $(s1'', r1'') \in \{(s'', r1') \mid s''. \text{ agrees } (- \text{ wrC } C) s1 s''\} \wedge (s2'', r2'') \in \{(s'', r2') \mid s''. \text{ agrees } (- \text{ wrC } C) s2 s''\}$ 
        then obtain  $r1'' = r1' r2'' = r2' \text{ agrees } (- \text{ wrC } C) s1 s1'' \text{ agrees } (- \text{ wrC } C) s2 s2''$ 
          by fastforce
        then show  $(s1'', r1''), (s2'', r2'') \models R$ 
          using  $\langle (s1, r1), (s2, r2) \models R \rangle \langle p1 = p1' \wedge p2 = p2' \wedge r1 = r1' \wedge r2 = r2' \rangle$  agrees-minusD agrees-same
          assms(2) sat-comm
        by (metis (no-types, opaque-lifting) disjoint-def inf-commute)
      qed
    ultimately have pair-sat (add-states  $(\Sigma (s1, p1'))$  ?R1) (add-states  $(\Sigma (s2, p2'))$  ?R2) (Star Q R)
      using add-states-sat-star by blast
    then show  $(s1', h1'), (s2', h2') \models \text{Star } Q R$ 
      using  $\langle (s1', h1') \in \text{add-states } (\Sigma (s1, p1')) \{(s'', r1') \mid s''. \text{ agrees } (- \text{ wrC } C) s1 s''\} \rangle \langle (s2', h2') \in \text{add-states } (\Sigma (s2, p2')) \{(s'', r2') \mid s''. \text{ agrees } (- \text{ wrC } C) s2 s''\} \rangle$ 

```

$s2\ s''\} \rangle$ pair-sat-def by blast
 qed
 qed
 qed

4.4.12 Consequence

theorem consequence-rule:
 fixes $\Delta :: ('i, 'a, nat) cont$
 assumes hoare-triple-valid $\Delta\ P'\ C\ Q'$
 and entails $P\ P'$
 and entails $Q'\ Q$
 shows hoare-triple-valid $\Delta\ P\ C\ Q$
proof –
 obtain Σ where asm0: $\bigwedge \sigma\ n.\ \sigma,\ \sigma \models P' \implies safe\ n\ \Delta\ C\ \sigma\ (\Sigma\ \sigma) \bigwedge \sigma\ \sigma'.\ \sigma,\ \sigma' \models P' \implies pair\text{-}sat\ (\Sigma\ \sigma)\ (\Sigma\ \sigma')\ Q'$
 using assms(1) hoare-triple-validE by blast

 show ?thesis
proof (rule hoare-triple-validI)
 show $\bigwedge s\ h\ n.\ (s, h), (s, h) \models P \implies safe\ n\ \Delta\ C\ (s, h)\ (\Sigma\ (s, h))$
 using asm0(1) assms(2) entails-def by blast
 show $\bigwedge s\ h\ s'\ h'.\ (s, h), (s', h') \models P \implies pair\text{-}sat\ (\Sigma\ (s, h))\ (\Sigma\ (s', h'))\ Q$
 by (meson asm0(2) assms(2) assms(3) entails-def pair-sat-def)
 qed
 qed

4.4.13 Existential

theorem existential-rule:
 fixes $\Delta :: ('i, 'a, nat) cont$
 assumes hoare-triple-valid $\Delta\ P\ C\ Q$
 and $x \notin fvC\ C$
 and $\bigwedge \Gamma.\ \Delta = Some\ \Gamma \implies x \notin fvA\ (invariant\ \Gamma)$
 and unambiguous $P\ x$
 shows hoare-triple-valid $\Delta\ (Exists\ x\ P)\ C\ (Exists\ x\ Q)$
proof –
 obtain Σ where asm0: $\bigwedge \sigma\ n.\ \sigma,\ \sigma \models P \implies safe\ n\ \Delta\ C\ \sigma\ (\Sigma\ \sigma) \bigwedge \sigma\ \sigma'.\ \sigma,\ \sigma' \models P \implies pair\text{-}sat\ (\Sigma\ \sigma)\ (\Sigma\ \sigma')\ Q$
 using assms(1) hoare-triple-validE by blast

 define Σ' where $\Sigma' = (\lambda \sigma.\ \bigcup v \in \{ v \mid v.\ ((fst\ \sigma)(x := v), snd\ \sigma), ((fst\ \sigma)(x := v), snd\ \sigma) \models P \}.\ upperize\ (\Sigma\ ((fst\ \sigma)(x := v), snd\ \sigma))\ (fvA\ Q - \{x\}))$

 show ?thesis
proof (rule hoare-triple-validI)
 show $\bigwedge s\ h\ n.\ (s, h), (s, h) \models Exists\ x\ P \implies safe\ n\ \Delta\ C\ (s, h)\ (\Sigma'\ (s, h))$
proof –
 fix $s\ h\ n$ assume $(s, h), (s, h) \models Exists\ x\ P$
 then obtain v where $(s(x := v), h), (s(x := v), h) \models P$

using *always-sat-refl hyper-sat.simps(7)* **by** *blast*
then have $\Sigma (s(x := v), h) \subseteq \Sigma' (s, h)$
using *upperize-larger SUP-upper2 Σ' -def* **by** *fastforce*

moreover have *safe n Δ C* $(s(x := v), h) (\Sigma (s(x := v), h))$
by *(simp add: $\langle (s(x := v), h), (s(x := v), h) \models P \rangle$ asm0(1))*
ultimately have *safe n Δ C* $(s(x := v), h) (\Sigma' (s, h))$
using *safe-larger-set* **by** *blast*
then have *safe n Δ C* $(s, h) (\Sigma' (s, h))$
proof *(rule safe-free-vars)*
show $\bigwedge \Gamma. \Delta = \text{Some } \Gamma \implies \text{agrees } (fvA \text{ (invariant } \Gamma)) (s(x := v)) s$
by *(meson agrees-comm agrees-update assms(3))*
show $\text{agrees } (fvC \ C \cup (fvA \ Q - \{x\})) (s(x := v)) s$
by *(simp add: agrees-def assms(2))*
show *upper-fvs* $(\Sigma' (s, h)) (fvA \ Q - \{x\})$
proof *(rule upper-fvsI)*
fix *sa s' ha*
assume *asm0: (sa, ha) $\in \Sigma' (s, h) \wedge$ agrees (fvA Q - {x}) sa s'*
then obtain *v* **where** $(s(x := v), h), (s(x := v), h) \models P (sa, ha) \in$
upperize $(\Sigma (s(x := v), h)) (fvA \ Q - \{x\})$
using *Σ' -def* **by** *force*
then have $(s', ha) \in \text{upperize } (\Sigma (s(x := v), h)) (fvA \ Q - \{x\})$
using *asm0 upper-fvs-def upper-fvs-upperize* **by** *blast*
then show $(s', ha) \in \Sigma' (s, h)$
using $\langle (s(x := v), h), (s(x := v), h) \models P \rangle$ *Σ' -def* **by** *force*
qed
qed
then show *safe n Δ C* $(s, h) (\Sigma' (s, h))$
by *auto*
qed
fix *s1 h1 s2 h2*
assume *asm1: (s1, h1), (s2, h2) \models Exists x P*
then obtain *v1' v2'* **where** $(s1(x := v1'), h1), (s2(x := v2'), h2) \models P$ **by**
auto
show *pair-sat* $(\Sigma' (s1, h1)) (\Sigma' (s2, h2)) (\text{Exists } x \ Q)$
proof *(rule pair-satI)*
fix *s1' h1' s2' h2'*
assume *asm2: (s1', h1') $\in \Sigma' (s1, h1) \wedge$ (s2', h2') $\in \Sigma' (s2, h2)$*

then obtain *v1 v2* **where**
 $r: (s1(x := v1), h1), (s1(x := v1), h1) \models P (s1', h1') \in \text{upperize } (\Sigma (s1(x$
 $:= v1), h1)) (fvA \ Q - \{x\})$
 $(s2(x := v2), h2), (s2(x := v2), h2) \models P (s2', h2') \in \text{upperize } (\Sigma (s2(x$
 $:= v2), h2)) (fvA \ Q - \{x\})$
using *Σ' -def* **by** *auto*

then obtain *s1'' s2''* **where** $\text{agrees } (fvA \ Q - \{x\}) s1'' s1' (s1'', h1') \in \Sigma$
 $(s1(x := v1), h1)$
 $\text{agrees } (fvA \ Q - \{x\}) s2'' s2' (s2'', h2') \in \Sigma (s2(x := v2), h2)$

```

using in-upperize by (metis (no-types, lifting))

moreover have  $(s1(x := v1), h1), (s2(x := v2), h2) \models P$ 
— Unambiguity needed here
proof –
  have  $v1 = v1'$ 
  using  $\langle (s1(x := v1'), h1), (s2(x := v2'), h2) \models P \rangle$  always-sat-refl assms(4)
r(1) unambiguous-def by blast
  moreover have  $v2 = v2'$ 
  using  $\langle (s1(x := v1'), h1), (s2(x := v2'), h2) \models P \rangle$  always-sat-refl assms(4)
r(3) sat-comm-aux unambiguous-def by blast
  ultimately show ?thesis
    by (simp add:  $\langle (s1(x := v1'), h1), (s2(x := v2'), h2) \models P \rangle$ )
qed
then have pair-sat  $(\Sigma (s1(x := v1), h1)) (\Sigma (s2(x := v2), h2)) Q$ 
  using asm0 by simp
then have  $(s1'', h1'), (s2'', h2') \models Q$ 
  using calculation(2) calculation(4) pair-sat-def by blast
moreover have agrees  $(fvA Q) s1'' (s1'(x := s1'' x))$ 
proof (rule agreesI)
  fix y assume  $y \in fvA Q$ 
  then show  $s1'' y = (s1'(x := s1'' x)) y$ 
    apply (cases  $x = y$ )
    apply auto[1]
  by (metis (mono-tags, lifting) DiffI agrees-def calculation(1) fun-upd-other
singleton-iff)
qed
moreover have agrees  $(fvA Q) s2'' (s2'(x := s2'' x))$ 
proof (rule agreesI)
  fix y assume  $y \in fvA Q$ 
  then show  $s2'' y = (s2'(x := s2'' x)) y$ 
    apply (cases  $x = y$ )
    apply auto[1]
  by (metis (mono-tags, lifting) DiffI agrees-def calculation(3) fun-upd-other
singleton-iff)
qed
ultimately have  $(s1'(x := s1'' x), h1'), (s2'(x := s2'' x), h2') \models Q$ 
  by (meson agrees-same sat-comm)
then show  $(s1', h1'), (s2', h2') \models \text{Exists } x Q$ 
  using hyper-sat.simps(7) by blast
qed
qed
qed

```

4.4.14 While loops

inductive *leads-to-loop* **where**

leads-to-loop $b I \Sigma \sigma \sigma$

$| \llbracket \text{leads-to-loop } b I \Sigma \sigma \sigma' ; \text{bdenot } b (\text{fst } \sigma') ; \sigma'' \in \Sigma \sigma' \rrbracket \implies \text{leads-to-loop } b I$

$\Sigma \sigma \sigma''$

definition *leads-to-loop-set* **where**

leads-to-loop-set $b I \Sigma \sigma = \{ \sigma' \mid \sigma'. \text{leads-to-loop } b I \Sigma \sigma \sigma' \}$

definition *trans- Σ* **where**

trans- Σ $b I \Sigma \sigma = \text{Set.filter } (\lambda \sigma. \neg \text{bdenot } b (\text{fst } \sigma)) (\text{leads-to-loop-set } b I \Sigma \sigma)$

inductive-cases *red-while-cases*: $\text{red } (C\text{while } b s) \sigma C' \sigma'$

inductive-cases *abort-while-cases*: $\text{aborts } (C\text{while } b s) \sigma$

lemma *safe-while-None*:

assumes $\bigwedge \sigma m. \sigma, \sigma \models \text{And } I (\text{Bool } b) \implies \text{safe } n (\text{None} :: ('i, 'a, \text{nat}) \text{cont}) C \sigma (\Sigma \sigma)$

and $\bigwedge \sigma \sigma'. \sigma, \sigma' \models \text{And } I (\text{Bool } b) \implies \text{pair-sat } (\Sigma \sigma) (\Sigma \sigma') I$

and $(s, h), (s, h) \models I$

and *leads-to-loop* $b I \Sigma \sigma (s, h)$

shows $\text{safe } n (\text{None} :: ('i, 'a, \text{nat}) \text{cont}) (C\text{while } b C) (s, h) (\text{trans-}\Sigma b I \Sigma \sigma)$

using *assms*

proof (*induct n arbitrary: s h*)

let $?S = \text{trans-}\Sigma b I \Sigma \sigma$

case (*Suc n*)

show $?case$

proof (*rule safeNoneI*)

show *no-abort* $(\text{None} :: ('i, 'a, \text{nat}) \text{cont}) (C\text{while } b C) s h$

using *abort-while-cases no-abortNoneI* **by** *blast*

fix $H \text{ hf } C' s' h'$

assume *asm0*: $\text{Some } H = \text{Some } h \oplus \text{Some } \text{hf} \wedge \text{full-ownership } (\text{get-fh } H) \wedge \text{no-guard } H \wedge \text{red } (C\text{while } b C) (s, \text{FractionalHeap.normalize } (\text{get-fh } H)) C' (s', h')$

show $\exists h'' H'. \text{full-ownership } (\text{get-fh } H') \wedge \text{no-guard } H' \wedge h' = \text{FractionalHeap.normalize } (\text{get-fh } H')$

$\wedge \text{Some } H' = \text{Some } h'' \oplus \text{Some } \text{hf} \wedge \text{safe } n (\text{None} :: ('i, 'a, \text{nat}) \text{cont}) C' (s', h'') (\text{trans-}\Sigma b I \Sigma \sigma)$

proof (*rule red-while-cases*)

show *red* $(C\text{while } b C) (s, \text{FractionalHeap.normalize } (\text{get-fh } H)) C' (s', h')$

using *asm0* **by** *linarith*

assume *asm1*: $C' = C\text{if } b (C\text{seq } C (C\text{while } b C)) C\text{skip } (s', h') = (s, \text{FractionalHeap.normalize } (\text{get-fh } H))$

have $\text{safe } n (\text{None} :: ('i, 'a, \text{nat}) \text{cont}) C' (s, h) ?S$

proof (*cases n*)

case (*Suc k*)

have $\text{safe } (\text{Suc } k) (\text{None} :: ('i, 'a, \text{nat}) \text{cont}) (C\text{if } b (C\text{seq } C (C\text{while } b C)) C\text{skip}) (s, h) ?S$

proof (*rule if-safe*)

have $\neg \text{bdenot } b s \implies (s, h) \in ?S$

by (*metis CollectI Suc.premis(4) asm1(2) fst-eqD leads-to-loop-set-def member-filter trans- Σ -def*)

```

then show  $\neg \text{bdenot } b \ s \implies \text{safe } k \ (\text{None} :: ('i, 'a, \text{nat}) \text{ cont}) \ Cskip \ (s, h)$ 
  (trans- $\Sigma$  b I  $\Sigma$   $\sigma$ )
  by (metis Pair-inject asm1(2) safe-skip)
assume asm2:  $\text{bdenot } b \ s$ 
then have  $(s, h), (s, h) \models \text{And } I \ (\text{Bool } b)$ 
  by (simp add: Suc.prem3)
then have r:  $\text{safe} \ (Suc \ n) \ (\text{None} :: ('i, 'a, \text{nat}) \text{ cont}) \ C \ (s, h) \ (\Sigma \ (s, h))$ 
  using Suc.prem1 by blast
show  $\text{safe } k \ (\text{None} :: ('i, 'a, \text{nat}) \text{ cont}) \ (Cseq \ C \ (Cwhile \ b \ C)) \ (s, h)$ 
  (trans- $\Sigma$  b I  $\Sigma$   $\sigma$ )
proof (rule seq-safe)
show  $\text{safe } k \ (\text{None} :: ('i, 'a, \text{nat}) \text{ cont}) \ C \ (s, h) \ (\Sigma \ (s, h))$ 
  by (metis Suc Suc-n-not-le-n nat-le-linear r safe-smaller)
fix m s' h' assume asm3:  $m \leq k \wedge (s', h') \in \Sigma \ (s, h)$ 
have  $\text{safe } n \ (\text{None} :: ('i, 'a, \text{nat}) \text{ cont}) \ (Cwhile \ b \ C) \ (s', h')$  (trans- $\Sigma$  b I  $\Sigma$   $\sigma$ )
proof (rule Suc.hyps)
show  $\text{leads-to-loop } b \ I \ \Sigma \ \sigma \ (s', h')$ 
  by (metis Suc.prem4 asm2 asm3 fst-conv leads-to-loop.intros(2))
show  $(s', h'), (s', h') \models I$ 
  using  $\langle (s, h), (s, h) \models \text{And } I \ (\text{Bool } b) \rangle$  asm3 assms(2) pair-satE by blast
show  $\bigwedge \sigma. \sigma, \sigma \models \text{And } I \ (\text{Bool } b) \implies \text{safe } n \ (\text{None} :: ('i, 'a, \text{nat}) \text{ cont}) \ C \ \sigma \ (\Sigma \ \sigma)$ 
  by (meson Suc.prem1 Suc-n-not-le-n nat-le-linear safe-smaller)
qed (auto simp add: assms)
then show  $\text{safe } m \ (\text{None} :: ('i, 'a, \text{nat}) \text{ cont}) \ (Cwhile \ b \ C) \ (s', h')$ 
  (trans- $\Sigma$  b I  $\Sigma$   $\sigma$ )
  using Suc asm3 le-SucI safe-smaller by blast
qed
qed
then show ?thesis
  using Suc asm1(1) by blast
qed (simp)
then show  $\exists h'' H'. \text{full-ownership} \ (\text{get-fh } H') \wedge$ 
   $\text{no-guard } H' \wedge h' = \text{FractionalHeap.normalize} \ (\text{get-fh } H') \wedge \text{Some } H' = \text{Some}$ 
   $h'' \oplus \text{Some } hf \wedge \text{safe } n \ \text{None } C' \ (s', h'')$  (trans- $\Sigma$  b I  $\Sigma$   $\sigma$ )
  using asm0 asm1(2) by blast
qed
qed (simp)
qed (simp)

```

lemma *safe-while-Some*:

```

assumes  $\bigwedge \sigma \ m. \sigma, \sigma \models \text{And } I \ (\text{Bool } b) \implies \text{safe } n \ (\text{Some } \Gamma) \ C \ \sigma \ (\Sigma \ \sigma)$ 
and  $\bigwedge \sigma \ \sigma'. \sigma, \sigma' \models \text{And } I \ (\text{Bool } b) \implies \text{pair-sat} \ (\Sigma \ \sigma) \ (\Sigma \ \sigma') \ I$ 
and  $(s, h), (s, h) \models I$ 
and  $\text{leads-to-loop } b \ I \ \Sigma \ \sigma \ (s, h)$ 
shows  $\text{safe } n \ (\text{Some } \Gamma) \ (Cwhile \ b \ C) \ (s, h)$  (trans- $\Sigma$  b I  $\Sigma$   $\sigma$ )

```



```

using assms
proof (induct n arbitrary: s h)
  let  $?S = \text{trans-}\Sigma \ b \ I \ \Sigma \ \sigma$ 
  case (Suc n)
  show  $?case$ 
  proof (rule safeSomeI)
    show no-abort (Some  $\Gamma$ ) (Cwhile  $b \ C$ )  $s \ h$ 
      using abort-while-cases no-abortSomeI by blast
    fix  $H \ hf \ C' \ s' \ h' \ hj \ v0$ 
    assume asm0:  $\text{Some } H = \text{Some } h \oplus \text{Some } hj \oplus \text{Some } hf \wedge$ 
       $\text{full-ownership } (\text{get-fh } H) \wedge \text{semi-consistent } \Gamma \ v0 \ H \wedge \text{sat-inv } s \ hj \ \Gamma \wedge \text{red}$ 
      (Cwhile  $b \ C$ ) ( $s, \text{FractionalHeap.normalize } (\text{get-fh } H)$ )  $C' \ (s', h')$ 
    show  $\exists h'' \ H' \ hj'. \text{full-ownership } (\text{get-fh } H') \wedge \text{semi-consistent } \Gamma \ v0 \ H' \wedge \text{sat-inv}$ 
       $s' \ hj' \ \Gamma \wedge$ 
       $h' = \text{FractionalHeap.normalize } (\text{get-fh } H') \wedge \text{Some } H' = \text{Some } h'' \oplus \text{Some}$ 
       $hj' \oplus \text{Some } hf \wedge \text{safe } n \ (\text{Some } \Gamma) \ C' \ (s', h') \ (\text{trans-}\Sigma \ b \ I \ \Sigma \ \sigma)$ 
    proof (rule red-while-cases)
      show red (Cwhile  $b \ C$ ) ( $s, \text{FractionalHeap.normalize } (\text{get-fh } H)$ )  $C' \ (s', h')$ 
      using asm0 by linarith
      assume asm1:  $C' = \text{Cif } b \ (\text{Cseq } C \ (\text{Cwhile } b \ C)) \ \text{Cskip } (s', h') = (s,$ 
       $\text{FractionalHeap.normalize } (\text{get-fh } H))$ 
      have  $\text{safe } n \ (\text{Some } \Gamma) \ C' \ (s, h) \ ?S$ 
      proof (cases n)
        case (Suc k)
          have  $\text{safe } (\text{Suc } k) \ (\text{Some } \Gamma) \ (\text{Cif } b \ (\text{Cseq } C \ (\text{Cwhile } b \ C)) \ \text{Cskip}) \ (s, h) \ ?S$ 
          proof (rule if-safe)
            have  $\neg \text{bdenot } b \ s \implies (s, h) \in ?S$ 
            by (metis CollectI Suc.premis(4) asm1(2) fst-eqD leads-to-loop-set-def
            member-filter trans-}\Sigma\text{-def})
            then show  $\neg \text{bdenot } b \ s \implies \text{safe } k \ (\text{Some } \Gamma) \ \text{Cskip } (s, h) \ (\text{trans-}\Sigma \ b \ I \ \Sigma$ 
             $\sigma)$ 
            by (metis Pair-inject asm1(2) safe-skip)
          assume asm2:  $\text{bdenot } b \ s$ 
          then have  $(s, h), (s, h) \models \text{And } I \ (\text{Bool } b)$ 
          by (simp add: Suc.premis(3))
          then have  $r: \text{safe } (\text{Suc } n) \ (\text{Some } \Gamma) \ C \ (s, h) \ (\Sigma \ (s, h))$ 
          using Suc.premis(1) by blast
          show  $\text{safe } k \ (\text{Some } \Gamma) \ (\text{Cseq } C \ (\text{Cwhile } b \ C)) \ (s, h) \ (\text{trans-}\Sigma \ b \ I \ \Sigma \ \sigma)$ 
          proof (rule seq-safe)
            show  $\text{safe } k \ (\text{Some } \Gamma) \ C \ (s, h) \ (\Sigma \ (s, h))$ 
            by (metis Suc Suc-n-not-le-n nat-le-linear r safe-smaller)
          fix  $m \ s' \ h'$  assume asm3:  $m \leq k \wedge (s', h') \in \Sigma \ (s, h)$ 
          have  $\text{safe } n \ (\text{Some } \Gamma) \ (\text{Cwhile } b \ C) \ (s', h') \ (\text{trans-}\Sigma \ b \ I \ \Sigma \ \sigma)$ 
          proof (rule Suc.hyps)
            show  $\text{leads-to-loop } b \ I \ \Sigma \ \sigma \ (s', h')$ 
            by (metis Suc.premis(4) asm2 asm3 fst-conv leads-to-loop.intros(2))
          show  $(s', h'), (s', h') \models I$ 
          using  $\langle (s, h), (s, h) \models \text{And } I \ (\text{Bool } b) \rangle \ \text{asm3} \ \text{assms}(2) \ \text{pair-satE}$  by

```

blast

```

    show  $\bigwedge \sigma. \sigma, \sigma \models \text{And } I \text{ (Bool } b) \implies \text{safe } n \text{ (Some } \Gamma) \text{ } C \sigma \text{ (}\Sigma \sigma)$ 
      by (meson Suc.prem1(1) Suc-n-not-le-n nat-le-linear safe-smaller)
    qed (auto simp add: assms)
    then show safe m (Some  $\Gamma$ ) (Cwhile b C) (s', h') (trans- $\Sigma$  b I  $\Sigma$   $\sigma$ )
      using Suc asm3 le-SucI safe-smaller by blast
    qed
  qed
  then show ?thesis
    using Suc asm1(1) by blast
  qed (simp)
  then show  $\exists h'' H' h_j'$ .
    full-ownership (get-fh H')  $\wedge$ 
    semi-consistent  $\Gamma$  v0 H'  $\wedge$ 
    sat-inv s' h_j'  $\Gamma$   $\wedge$  h' = FractionalHeap.normalize (get-fh H')  $\wedge$  Some H' =
    Some h''  $\oplus$  Some h_j'  $\oplus$  Some hf  $\wedge$  safe n (Some  $\Gamma$ ) C' (s', h'') (trans- $\Sigma$  b I  $\Sigma$   $\sigma$ )
    using asm0 asm1(2) by blast
  qed
  qed (simp)
  qed (simp)

```

lemma *safe-while*:

```

  fixes  $\Delta :: ('i, 'a, nat) \text{ cont}$ 
  assumes  $\bigwedge \sigma m. \sigma, \sigma \models \text{And } I \text{ (Bool } b) \implies \text{safe } n \Delta C \sigma \text{ (}\Sigma \sigma)$ 
    and  $\bigwedge \sigma \sigma'. \sigma, \sigma' \models \text{And } I \text{ (Bool } b) \implies \text{pair-sat } (\Sigma \sigma) (\Sigma \sigma') I$ 
    and  $(s, h), (s, h) \models I$ 
    and leads-to-loop b I  $\Sigma$   $\sigma$  (s, h)
  shows safe n  $\Delta$  (Cwhile b C) (s, h) (trans- $\Sigma$  b I  $\Sigma$   $\sigma$ )
  apply (cases  $\Delta$ )
  using assms safe-while-None apply blast
  using assms safe-while-Some by blast

```

lemma *leads-to-sat-inv-unary*:

```

  assumes leads-to-loop b I  $\Sigma$   $\sigma$   $\sigma'$ 
    and  $\bigwedge \sigma \sigma'. \sigma, \sigma' \models (\text{And } I \text{ (Bool } b)) \implies \text{pair-sat } (\Sigma \sigma) (\Sigma \sigma') I$ 
    and  $\sigma, \sigma \models I$ 
  shows  $\sigma', \sigma' \models I$ 
  using assms
  proof (induct arbitrary: rule: leads-to-loop.induct)
  case (2 b I  $\Sigma$   $\sigma 0$   $\sigma 1$   $\sigma 2$ )
  then have pair-sat ( $\Sigma$   $\sigma 1$ ) ( $\Sigma$   $\sigma 1$ ) I
    by (metis hyper-sat.simps(1) hyper-sat.simps(3) prod.collapse)
  then show ?case
    using 2.hyps(4) pair-sat-def by blast
  qed (simp)

```

theorem *while-rule2*:

```

  fixes  $\Delta :: ('i, 'a, nat) \text{ cont}$ 
  assumes unary I
    and hoare-triple-valid  $\Delta$  (And I (Bool b)) C I

```

```

shows hoare-triple-valid  $\Delta I (Cwhile\ b\ C) (And\ I\ (Bool\ (Bnot\ b)))$ 
proof –
  obtain  $\Sigma$  where  $asm0: \bigwedge \sigma\ n.\ \sigma, \sigma \models (And\ I\ (Bool\ b)) \implies safe\ n\ \Delta\ C\ \sigma\ (\Sigma\ \sigma)$ 
    and  $\bigwedge \sigma\ \sigma'.\ \sigma, \sigma' \models (And\ I\ (Bool\ b)) \implies pair\text{-}sat\ (\Sigma\ \sigma)\ (\Sigma\ \sigma')\ I$ 
    using  $assms(2)$  hoare-triple-validE by blast
  let  $? \Sigma = trans\text{-}\Sigma\ b\ I\ \Sigma$ 
  show ?thesis
  proof (rule hoare-triple-validI)

    show  $\bigwedge s\ h\ s'\ h'.\ (s, h), (s', h') \models I \implies pair\text{-}sat\ (? \Sigma\ (s, h))\ (? \Sigma\ (s', h'))\ (And\ I\ (Bool\ (Bnot\ b)))$ 
    proof –
      fix  $s1\ h1\ s2\ h2$  assume  $asm0: (s1, h1), (s2, h2) \models I$ 
      show  $pair\text{-}sat\ (trans\text{-}\Sigma\ b\ I\ \Sigma\ (s1, h1))\ (trans\text{-}\Sigma\ b\ I\ \Sigma\ (s2, h2))\ (And\ I\ (Bool\ (Bnot\ b)))$ 
      proof (rule pair-satI)
        fix  $s1'\ h1'\ s2'\ h2'$ 
        assume  $asm1: (s1', h1') \in trans\text{-}\Sigma\ b\ I\ \Sigma\ (s1, h1) \wedge (s2', h2') \in trans\text{-}\Sigma\ b\ I\ \Sigma\ (s2, h2)$ 
        then obtain  $leads\text{-}to\text{-}loop\ b\ I\ \Sigma\ (s1, h1)\ (s1', h1') \neg\ bdenot\ b\ s1'$ 
           $leads\text{-}to\text{-}loop\ b\ I\ \Sigma\ (s2, h2)\ (s2', h2') \neg\ bdenot\ b\ s2'$ 
        using  $trans\text{-}\Sigma\text{-}def$   $leads\text{-}to\text{-}loop\text{-}set\text{-}def$ 
        by (metis fst-conv mem-Collect-eq member-filter)
        then have  $(s1', h1'), (s1', h1') \models I \wedge (s2', h2'), (s2', h2') \models I$ 
        by (meson  $\langle \bigwedge \sigma'\ \sigma.\ \sigma, \sigma' \models And\ I\ (Bool\ b) \implies pair\text{-}sat\ (\Sigma\ \sigma)\ (\Sigma\ \sigma')\ I \rangle$ 
           $always\text{-}sat\text{-}refl\ asm0\ leads\text{-}to\text{-}sat\text{-}inv\text{-}unary\ sat\text{-}comm\text{-}aux$ )
        then show  $(s1', h1'), (s2', h2') \models And\ I\ (Bool\ (Bnot\ b))$ 
        by (metis  $\langle \neg\ bdenot\ b\ s1' \rangle\ \langle \neg\ bdenot\ b\ s2' \rangle\ assms(1)\ bdenot.simps(3)$ 
           $hyper\text{-}sat.simps(1)\ hyper\text{-}sat.simps(3)\ unaryE$ )
      qed
    qed
    fix  $s\ h\ n$ 
    assume  $asm1: (s, h), (s, h) \models I$ 

    show  $safe\ n\ \Delta\ (Cwhile\ b\ C)\ (s, h)\ (trans\text{-}\Sigma\ b\ I\ \Sigma\ (s, h))$ 
    proof (rule safe-while)
      show  $\bigwedge \sigma\ \sigma'.\ \sigma, \sigma' \models And\ I\ (Bool\ b) \implies pair\text{-}sat\ (\Sigma\ \sigma)\ (\Sigma\ \sigma')\ I$ 
      by (simp add:  $\langle \bigwedge \sigma'\ \sigma.\ \sigma, \sigma' \models And\ I\ (Bool\ b) \implies pair\text{-}sat\ (\Sigma\ \sigma)\ (\Sigma\ \sigma')\ I \rangle$ )
      show  $(s, h), (s, h) \models I$ 
      using  $asm1$  by auto
      show  $leads\text{-}to\text{-}loop\ b\ I\ \Sigma\ (s, h)\ (s, h)$ 
      by (simp add:  $leads\text{-}to\text{-}loop.intros(1)$ )
      show  $\bigwedge \sigma\ m.\ \sigma, \sigma \models And\ I\ (Bool\ b) \implies safe\ n\ \Delta\ C\ \sigma\ (\Sigma\ \sigma)$ 
      by (simp add:  $asm0$ )
    qed
  qed
qed
fun iterate-sigma ::  $nat \Rightarrow bexp \Rightarrow ('i, 'a, nat)\ assertion \Rightarrow ((store \times ('i, 'a)\ heap)$ 

```

$\Rightarrow (\text{store} \times ('i, 'a) \text{ heap}) \text{ set}) \Rightarrow (\text{store} \times ('i, 'a) \text{ heap}) \Rightarrow (\text{store} \times ('i, 'a) \text{ heap})$
 set

where

$\text{iterate-sigma } 0 \ b \ I \ \Sigma \ \sigma = \{\sigma\}$

$| \text{iterate-sigma } (\text{Suc } n) \ b \ I \ \Sigma \ \sigma = (\bigcup \sigma' \in \text{Set.filter } (\lambda \sigma. \text{bdenot } b \ (\text{fst } \sigma)) \ (\text{iterate-sigma } n \ b \ I \ \Sigma \ \sigma). \ \Sigma \ \sigma')$

lemma *union-of-iterate-sigma-is-leads-to-loop-set:*

assumes *leads-to-loop* $b \ I \ \Sigma \ \sigma \ \sigma'$

shows $\sigma' \in (\bigcup n. \text{iterate-sigma } n \ b \ I \ \Sigma \ \sigma)$

using *assms*

proof (*induct rule: leads-to-loop.induct*)

case $(1 \ b \ I \ \Sigma \ \sigma)$

have $\sigma \in \text{iterate-sigma } 0 \ b \ I \ \Sigma \ \sigma$

by *simp*

then show *?case*

by *blast*

next

case $(2 \ b \ I \ \Sigma \ \sigma \ \sigma' \ \sigma'')$

then obtain n **where** $\sigma' \in \text{iterate-sigma } n \ b \ I \ \Sigma \ \sigma$ **by** *blast*

then have $\sigma'' \in \text{iterate-sigma } (\text{Suc } n) \ b \ I \ \Sigma \ \sigma$ **using** 2 **by** *auto*

then show *?case* **by** *blast*

qed

lemma *trans-included:*

$\text{trans-}\Sigma \ b \ I \ \Sigma \ \sigma \subseteq \text{Set.filter } (\lambda \sigma. \neg \text{bdenot } b \ (\text{fst } \sigma)) \ (\bigcup n. \text{iterate-sigma } n \ b \ I \ \Sigma \ \sigma)$

proof

fix x **assume** $x \in \text{trans-}\Sigma \ b \ I \ \Sigma \ \sigma$

then have $\neg \text{bdenot } b \ (\text{fst } x) \wedge \text{leads-to-loop } b \ I \ \Sigma \ \sigma \ x$

by (*simp add: leads-to-loop-set-def trans-Σ-def*)

then show $x \in \text{Set.filter } (\lambda \sigma. \neg \text{bdenot } b \ (\text{fst } \sigma)) \ (\bigcup n. \text{iterate-sigma } n \ b \ I \ \Sigma \ \sigma)$

by (*metis member-filter union-of-iterate-sigma-is-leads-to-loop-set*)

qed

lemma *iterate-sigma-low-all-sat-I-and-low:*

assumes $\bigwedge \sigma \ \sigma'. \ \sigma, \ \sigma' \models (\text{And } I \ (\text{Bool } b)) \Longrightarrow \text{pair-sat } (\Sigma \ \sigma) \ (\Sigma \ \sigma') \ (\text{And } I \ (\text{Low } b))$

and $\sigma 1, \ \sigma 2 \models I$

and $\text{bdenot } b \ (\text{fst } \sigma 1) = \text{bdenot } b \ (\text{fst } \sigma 2)$

shows $\text{pair-sat } (\text{iterate-sigma } n \ b \ I \ \Sigma \ \sigma 1) \ (\text{iterate-sigma } n \ b \ I \ \Sigma \ \sigma 2) \ (\text{And } I \ (\text{Low } b))$

using *assms*

proof (*induct n*)

case 0

then show *?case*

by (*metis (mono-tags, lifting) hyper-sat.simps(3) hyper-sat.simps(5) iterate-sigma.simps(1)*)

```

pair-satI prod.exhaust-sel singletonD)
next
  case (Suc n)
  show ?case
  proof (rule pair-satI)
    fix s1 h1 s2 h2
    assume asm0: (s1, h1) ∈ iterate-sigma (Suc n) b I Σ σ1 ∧ (s2, h2) ∈
iterate-sigma (Suc n) b I Σ σ2
    then obtain σ1' σ2' where bdenot b (fst σ1') bdenot b (fst σ2')
      σ1' ∈ iterate-sigma n b I Σ σ1 σ2' ∈ iterate-sigma n b I Σ σ2
      (s1, h1) ∈ Σ σ1' (s2, h2) ∈ Σ σ2'
    by auto
    then have pair-sat (iterate-sigma n b I Σ σ1) (iterate-sigma n b I Σ σ2) (And
I (Low b))
      using Suc.hyps
      using Suc.premis(3) assms(1) assms(2) by blast
    moreover have pair-sat (Σ σ1') (Σ σ2') (And I (Low b))
    proof (rule Suc.premis)
      show σ1', σ2' ⊨ And I (Bool b)
        by (metis ⟨σ1' ∈ iterate-sigma n b I Σ σ1⟩ ⟨σ2' ∈ iterate-sigma n b
I Σ σ2⟩ ⟨bdenot b (fst σ1')⟩ ⟨bdenot b (fst σ2')⟩ calculation hyper-sat.simps(1)
hyper-sat.simps(3) pair-sat-def prod.exhaust-sel)
      qed
    ultimately show (s1, h1), (s2, h2) ⊨ And I (Low b)
      using ⟨(s1, h1) ∈ Σ σ1'⟩ ⟨(s2, h2) ∈ Σ σ2'⟩ pair-sat-def by blast
    qed
  qed

```

```

lemma iterate-empty-later-empty:
  assumes iterate-sigma n b I Σ σ = {}
  and m ≥ n
  shows iterate-sigma m b I Σ σ = {}
  using assms
proof (induct m - n arbitrary: n m)
  case (Suc k)
  then obtain mm where m = Suc mm
  by (metis iterate-sigma.elims zero-diff)
  then have iterate-sigma mm b I Σ σ = {}
  by (metis Suc.hyps(1) Suc.hyps(2) Suc.premis(1) Suc.premis(2) Suc-le-mono
diff-Suc-Suc diff-diff-cancel diff-le-self)
  then show ?case
  using ⟨m = Suc mm⟩ by force
qed (simp)

```

```

lemma all-same:
  assumes ∧σ σ'. σ, σ' ⊨ (And I (Bool b)) ⇒ pair-sat (Σ σ) (Σ σ') (And I (Low
b))
  and σ1, σ2 ⊨ I
  and bdenot b (fst σ1) = bdenot b (fst σ2)

```

and $x1 \in \text{iterate-sigma } n \ b \ I \ \Sigma \ \sigma 1$
and $x2 \in \text{iterate-sigma } n \ b \ I \ \Sigma \ \sigma 2$
shows $\text{bdenot } b \ (\text{fst } x1) = \text{bdenot } b \ (\text{fst } x2)$
proof –
have $x1, x2 \models (\text{And } I \ (\text{Low } b))$
using $\text{assms}(1) \ \text{assms}(2) \ \text{assms}(3) \ \text{assms}(4) \ \text{assms}(5) \ \text{iterate-sigma-low-all-sat-I-and-low}$
pair-sat-def **by** *blast*
then show *?thesis*
by (*metis* (*no-types*, *lifting*) *hyper-sat.simps(3)* *hyper-sat.simps(5)* *surjective-pairing*)
qed

lemma *non-empty-at-most-once*:

assumes $\bigwedge \sigma \ \sigma'. \ \sigma, \sigma' \models (\text{And } I \ (\text{Bool } b)) \implies \text{pair-sat } (\Sigma \ \sigma) \ (\Sigma \ \sigma') \ (\text{And } I \ (\text{Low } b))$
and $\sigma, \sigma \models I$
and $\text{Set.filter } (\lambda \sigma. \neg \text{bdenot } b \ (\text{fst } \sigma)) \ (\text{iterate-sigma } n1 \ b \ I \ \Sigma \ \sigma) \neq \{\}$
and $\text{Set.filter } (\lambda \sigma. \neg \text{bdenot } b \ (\text{fst } \sigma)) \ (\text{iterate-sigma } n2 \ b \ I \ \Sigma \ \sigma) \neq \{\}$
shows $n1 = n2$

proof –

let $?n = \min \ n1 \ n2$
obtain σ' **where** $\sigma' \in \text{Set.filter } (\lambda \sigma. \neg \text{bdenot } b \ (\text{fst } \sigma)) \ (\text{iterate-sigma } ?n \ b \ I \ \Sigma \ \sigma)$
by (*metis* $\text{assms}(3) \ \text{assms}(4) \ \text{equals0I} \ \text{min.orderE} \ \text{min-def}$)
then have $\neg \text{bdenot } b \ (\text{fst } \sigma')$
by *fastforce*
moreover have $\text{pair-sat } (\text{iterate-sigma } ?n \ b \ I \ \Sigma \ \sigma) \ (\text{iterate-sigma } ?n \ b \ I \ \Sigma \ \sigma) \ (\text{And } I \ (\text{Low } b))$
using $\text{assms}(1) \ \text{assms}(2) \ \text{assms}(3) \ \text{iterate-sigma-low-all-sat-I-and-low}$ **by** *blast*
then have $r: \bigwedge x. \ x \in \text{iterate-sigma } ?n \ b \ I \ \Sigma \ \sigma \implies \neg \text{bdenot } b \ (\text{fst } x)$
by (*metis* $\langle \sigma' \in \text{Set.filter } (\lambda \sigma. \neg \text{bdenot } b \ (\text{fst } \sigma)) \ (\text{iterate-sigma } (\min \ n1 \ n2) \ b \ I \ \Sigma \ \sigma) \rangle \ \text{all-same} \ \text{assms}(1) \ \text{assms}(2) \ \text{member-filter}$)
then have $\text{iterate-sigma } (\text{Suc } ?n) \ b \ I \ \Sigma \ \sigma = \{\}$ **by** *auto*
then have $\neg (n1 > ?n) \wedge \neg (n2 > ?n)$ **using** *iterate-empty-later-empty[of Suc ?n b I Σ σ]*
assms **by** (*metis* (*no-types*, *lifting*) *Set.filter-def* *empty-Collect-eq* *empty-def* *le-simps(3)* *mem-Collect-eq*)
then show *?thesis* **by** *linarith*
qed

lemma *one-non-empty-union*:

assumes $\bigwedge \sigma \ \sigma'. \ \sigma, \sigma' \models (\text{And } I \ (\text{Bool } b)) \implies \text{pair-sat } (\Sigma \ \sigma) \ (\Sigma \ \sigma') \ (\text{And } I \ (\text{Low } b))$
and $\sigma, \sigma \models I$
and $\text{Set.filter } (\lambda \sigma. \neg \text{bdenot } b \ (\text{fst } \sigma)) \ (\text{iterate-sigma } k \ b \ I \ \Sigma \ \sigma) \neq \{\}$
shows $\text{Set.filter } (\lambda \sigma. \neg \text{bdenot } b \ (\text{fst } \sigma)) \ (\bigcup n. \ \text{iterate-sigma } n \ b \ I \ \Sigma \ \sigma) = \text{Set.filter } (\lambda \sigma. \neg \text{bdenot } b \ (\text{fst } \sigma)) \ (\text{iterate-sigma } k \ b \ I \ \Sigma \ \sigma)$

proof

show $\text{Set.filter } (\lambda \sigma. \neg \text{bdenot } b \ (\text{fst } \sigma)) \ (\text{iterate-sigma } k \ b \ I \ \Sigma \ \sigma) \subseteq \text{Set.filter}$

$(\lambda\sigma. \neg \text{bdenot } b \text{ (fst } \sigma)) (\bigcup n. \text{iterate-sigma } n \text{ b I } \Sigma \sigma)$
by auto
show $\text{Set.filter } (\lambda\sigma. \neg \text{bdenot } b \text{ (fst } \sigma)) (\bigcup n. \text{iterate-sigma } n \text{ b I } \Sigma \sigma) \subseteq \text{Set.filter } (\lambda\sigma. \neg \text{bdenot } b \text{ (fst } \sigma)) (\text{iterate-sigma } k \text{ b I } \Sigma \sigma)$
proof
fix x **assume** $x \in \text{Set.filter } (\lambda\sigma. \neg \text{bdenot } b \text{ (fst } \sigma)) (\bigcup n. \text{iterate-sigma } n \text{ b I } \Sigma \sigma)$
then obtain k' **where** $x \in \text{iterate-sigma } k' \text{ b I } \Sigma \sigma \neg \text{bdenot } b \text{ (fst } x)$
by auto
then have $x \in \text{Set.filter } (\lambda\sigma. \neg \text{bdenot } b \text{ (fst } \sigma)) (\text{iterate-sigma } k' \text{ b I } \Sigma \sigma)$
by fastforce
then have $k = k'$
using *non-empty-at-most-once* *assms(1)* *assms(2)* *assms(3)* **by blast**
then show $x \in \text{Set.filter } (\lambda\sigma. \neg \text{bdenot } b \text{ (fst } \sigma)) (\text{iterate-sigma } k \text{ b I } \Sigma \sigma)$
using $\langle x \in \text{Set.filter } (\lambda\sigma. \neg \text{bdenot } b \text{ (fst } \sigma)) (\text{iterate-sigma } k' \text{ b I } \Sigma \sigma) \rangle$ **by blast**
qed

definition not-set where

not-set $b \text{ } S = \text{Set.filter } (\lambda\sigma. \neg \text{bdenot } b \text{ (fst } \sigma)) S$

lemma union-exists-at-some-point-exactly:

assumes $\bigwedge \sigma \sigma'. \sigma, \sigma' \models (\text{And } I \text{ (Bool } b)) \implies \text{pair-sat } (\Sigma \sigma) (\Sigma \sigma') (\text{And } I \text{ (Low } b))$

and $\sigma 1, \sigma 2 \models I$

and $\text{bdenot } b \text{ (fst } \sigma 1) = \text{bdenot } b \text{ (fst } \sigma 2)$

and $\text{Set.filter } (\lambda\sigma. \neg \text{bdenot } b \text{ (fst } \sigma)) (\bigcup n. \text{iterate-sigma } n \text{ b I } \Sigma \sigma 1) \neq \{\}$

and $\text{Set.filter } (\lambda\sigma. \neg \text{bdenot } b \text{ (fst } \sigma)) (\bigcup n. \text{iterate-sigma } n \text{ b I } \Sigma \sigma 2) \neq \{\}$

shows $\exists k. \text{not-set } b (\bigcup n. \text{iterate-sigma } n \text{ b I } \Sigma \sigma 1) = \text{not-set } b (\text{iterate-sigma } k \text{ b I } \Sigma \sigma 1) \wedge \text{not-set } b (\bigcup n. \text{iterate-sigma } n \text{ b I } \Sigma \sigma 2) = \text{not-set } b (\text{iterate-sigma } k \text{ b I } \Sigma \sigma 2)$

proof –

obtain $k 1$ **where** $\text{Set.filter } (\lambda\sigma. \neg \text{bdenot } b \text{ (fst } \sigma)) (\text{iterate-sigma } k 1 \text{ b I } \Sigma \sigma 1) \neq \{\}$

using *assms(4)* **by fastforce**

moreover obtain $k 2$ **where** $\text{Set.filter } (\lambda\sigma. \neg \text{bdenot } b \text{ (fst } \sigma)) (\text{iterate-sigma } k 2 \text{ b I } \Sigma \sigma 2) \neq \{\}$

using *assms(5)* **by fastforce**

show *?thesis*

proof (*cases* $k 1 \leq k 2$)

case *True*

then have $\text{iterate-sigma } k 1 \text{ b I } \Sigma \sigma 2 \neq \{\}$

by (*metis* (*no-types*, *lifting*) *Collect-cong* *Set.filter-def* $\langle \text{Set.filter } (\lambda\sigma. \neg \text{bdenot } b \text{ (fst } \sigma)) (\text{iterate-sigma } k 2 \text{ b I } \Sigma \sigma 2) \neq \{\} \rangle$ *empty-def* *iterate-empty-later-empty* *mem-Collect-eq*)

then obtain $\sigma 1' \sigma 2'$ **where** $\sigma 1' \in \text{Set.filter } (\lambda\sigma. \neg \text{bdenot } b \text{ (fst } \sigma)) (\text{iterate-sigma } k 1 \text{ b I } \Sigma \sigma 1) \wedge \sigma 2' \in \text{iterate-sigma } k 1 \text{ b I } \Sigma \sigma 2$

using *calculation* **by** *blast*
then have $\neg \text{bdenot } b \text{ (fst } \sigma 1')$
by *fastforce*
moreover have *pair-sat* (*iterate-sigma* $k1$ b I Σ $\sigma 1$) (*iterate-sigma* $k1$ b I Σ $\sigma 2$) (*And* I (*Low* b))
using *assms*(1) *assms*(2) *assms*(3) *iterate-sigma-low-all-sat-I-and-low* **by** *blast*
then have $r: \bigwedge x1\ x2. x1 \in \text{iterate-sigma } k1\ b\ I\ \Sigma\ \sigma 1 \wedge x2 \in \text{iterate-sigma } k1\ b\ I\ \Sigma\ \sigma 2 \implies \text{bdenot } b \text{ (fst } x1) \longleftrightarrow \text{bdenot } b \text{ (fst } x2)$
by (*metis* (*no-types*, *opaque-lifting*) *eq-fst-iff* *hyper-sat.simps*(3) *hyper-sat.simps*(5) *pair-sat-def*)
then have $\neg \text{bdenot } b \text{ (fst } \sigma 2')$
by (*metis* $\langle \sigma 1' \in \text{Set.filter } (\lambda \sigma. \neg \text{bdenot } b \text{ (fst } \sigma)) \text{ (iterate-sigma } k1\ b\ I\ \Sigma\ \sigma 1) \wedge \sigma 2' \in \text{iterate-sigma } k1\ b\ I\ \Sigma\ \sigma 2 \rangle$ *member-filter*)
then have $\bigwedge x1. x1 \in \text{iterate-sigma } k1\ b\ I\ \Sigma\ \sigma 1 \implies \neg \text{bdenot } b \text{ (fst } x1)$
using $\langle \sigma 1' \in \text{Set.filter } (\lambda \sigma. \neg \text{bdenot } b \text{ (fst } \sigma)) \text{ (iterate-sigma } k1\ b\ I\ \Sigma\ \sigma 1) \wedge \sigma 2' \in \text{iterate-sigma } k1\ b\ I\ \Sigma\ \sigma 2 \rangle$ r **by** *blast*
then have *iterate-sigma* (*Suc* $k1$) b I Σ $\sigma 1 = \{\}$ **by** *auto*
moreover have $\bigwedge x2. x2 \in \text{iterate-sigma } k1\ b\ I\ \Sigma\ \sigma 2 \implies \neg \text{bdenot } b \text{ (fst } x2)$
by (*metis* $\langle \sigma 1' \in \text{Set.filter } (\lambda \sigma. \neg \text{bdenot } b \text{ (fst } \sigma)) \text{ (iterate-sigma } k1\ b\ I\ \Sigma\ \sigma 1) \wedge \sigma 2' \in \text{iterate-sigma } k1\ b\ I\ \Sigma\ \sigma 2 \rangle$ *member-filter* r)
then have *iterate-sigma* (*Suc* $k1$) b I Σ $\sigma 2 = \{\}$ **by** *auto*
then have $k1 = k2$
using *True* $\langle \text{Set.filter } (\lambda \sigma. \neg \text{bdenot } b \text{ (fst } \sigma)) \text{ (iterate-sigma } k2\ b\ I\ \Sigma\ \sigma 2) \neq \{\} \rangle$ *dual-order.antisym*[of $k1$ $k2$]
ex-in-conv *iterate-empty-later-empty*[of $-$ b I Σ $\sigma 2$] *member-filter* *not-less-eq-eq*
by *metis*
moreover have *Set.filter* $(\lambda \sigma. \neg \text{bdenot } b \text{ (fst } \sigma)) (\bigcup n. \text{iterate-sigma } n\ b\ I\ \Sigma\ \sigma 1) = \text{Set.filter } (\lambda \sigma. \neg \text{bdenot } b \text{ (fst } \sigma)) \text{ (iterate-sigma } k1\ b\ I\ \Sigma\ \sigma 1)$
using *one-non-empty-union*[of I b Σ $\sigma 1$]
using $\langle \text{Set.filter } (\lambda \sigma. \neg \text{bdenot } b \text{ (fst } \sigma)) \text{ (iterate-sigma } k1\ b\ I\ \Sigma\ \sigma 1) \neq \{\} \rangle$ *always-sat-refl* *assms*(1) *assms*(2) **by** *blast*
moreover have *Set.filter* $(\lambda \sigma. \neg \text{bdenot } b \text{ (fst } \sigma)) (\bigcup n. \text{iterate-sigma } n\ b\ I\ \Sigma\ \sigma 2) = \text{Set.filter } (\lambda \sigma. \neg \text{bdenot } b \text{ (fst } \sigma)) \text{ (iterate-sigma } k1\ b\ I\ \Sigma\ \sigma 2)$
using *one-non-empty-union*[of I b Σ $\sigma 2$]
using $\langle \text{Set.filter } (\lambda \sigma. \neg \text{bdenot } b \text{ (fst } \sigma)) \text{ (iterate-sigma } k2\ b\ I\ \Sigma\ \sigma 2) \neq \{\} \rangle$ *always-sat-refl* *assms*(1) *assms*(2) *calculation*(3) *sat-comm* **by** *blast*
ultimately show *?thesis*
by (*metis* *not-set-def*)
next
case *False*
then have *iterate-sigma* $k2$ b I Σ $\sigma 1 \neq \{\}$
by (*metis* (*no-types*, *lifting*) *Collect-cong* *Set.filter-def* *calculation* *empty-def* *iterate-empty-later-empty* *linorder-le-cases* *mem-Collect-eq*)
then obtain $\sigma 1' \sigma 2'$ **where** $\sigma 1' \in \text{iterate-sigma } k2\ b\ I\ \Sigma\ \sigma 1 \wedge \sigma 2' \in \text{not-set } b \text{ (iterate-sigma } k2\ b\ I\ \Sigma\ \sigma 2)$
by (*metis* $\langle \text{Set.filter } (\lambda \sigma. \neg \text{bdenot } b \text{ (fst } \sigma)) \text{ (iterate-sigma } k2\ b\ I\ \Sigma\ \sigma 2) \neq \{\} \rangle$ *ex-in-conv* *not-set-def*)
then have $\neg \text{bdenot } b \text{ (fst } \sigma 2')$


```

    using not-set-def by fastforce
  then have  $\neg$  bdenot b (fst  $\sigma 1'$ )
    by (metis  $\langle \sigma 1' \in \text{iterate-sigma } k2 \text{ b } I \Sigma \sigma 1 \wedge \sigma 2' \in \text{not-set } b \text{ (iterate-sigma } k2 \text{ b } I \Sigma \sigma 2) \rangle$  all-same assms(1) assms(2) assms(3) member-filter not-set-def)
  then have  $\bigwedge x1. x1 \in \text{iterate-sigma } k2 \text{ b } I \Sigma \sigma 1 \implies \neg$  bdenot b (fst x1)
    using  $\langle \sigma 1' \in \text{iterate-sigma } k2 \text{ b } I \Sigma \sigma 1 \wedge \sigma 2' \in \text{not-set } b \text{ (iterate-sigma } k2 \text{ b } I \Sigma \sigma 2) \rangle$  all-same always-sat-refl assms(1) assms(2) by blast
  then have iterate-sigma (Suc k2) b I  $\Sigma$   $\sigma 1 = \{\}$  by auto
  moreover have  $\bigwedge x2. x2 \in \text{iterate-sigma } k2 \text{ b } I \Sigma \sigma 2 \implies \neg$  bdenot b (fst x2)
    using  $\langle \neg$  bdenot b (fst  $\sigma 1') \rangle \langle \sigma 1' \in \text{iterate-sigma } k2 \text{ b } I \Sigma \sigma 1 \wedge \sigma 2' \in \text{not-set } b \text{ (iterate-sigma } k2 \text{ b } I \Sigma \sigma 2) \rangle$  all-same assms(1) assms(2) assms(3) by blast
  then have iterate-sigma (Suc k2) b I  $\Sigma$   $\sigma 2 = \{\}$  by auto
  then show ?thesis
    by (metis (no-types, lifting) Collect-empty-eq False Set.filter-def  $\langle \text{Set.filter } (\lambda \sigma. \neg$  bdenot b (fst  $\sigma)) \text{ (iterate-sigma } k1 \text{ b } I \Sigma \sigma 1) \neq \{\} \rangle$  calculation empty-iff iterate-empty-later-empty not-less-eq-eq)
  qed
qed

```

theorem while-rule1:

```

  fixes  $\Delta :: ('i, 'a, \text{nat}) \text{ cont}$ 
  assumes hoare-triple-valid  $\Delta$  (And I (Bool b)) C (And I (Low b))
  shows hoare-triple-valid  $\Delta$  (And I (Low b)) (Cwhile b C) (And I (Bool (Bnot b)))
proof -
  obtain  $\Sigma$  where asm0:  $\bigwedge \sigma n. \sigma, \sigma \models (\text{And } I \text{ (Bool } b)) \implies \text{safe } n \Delta C \sigma (\Sigma \sigma)$ 
    and  $\bigwedge \sigma \sigma'. \sigma, \sigma' \models (\text{And } I \text{ (Bool } b)) \implies \text{pair-sat } (\Sigma \sigma) (\Sigma \sigma') (\text{And } I \text{ (Low } b))$ 
  using assms(1) hoare-triple-validE by blast
  let ? $\Sigma = \lambda \sigma. \text{not-set } b (\bigcup n. \text{iterate-sigma } n \text{ b } I \Sigma \sigma)$ 
  show ?thesis
  proof (rule hoare-triple-validI)
    show  $\bigwedge s h s' h'. (s, h), (s', h') \models \text{And } I \text{ (Low } b) \implies \text{pair-sat } (? \Sigma (s, h)) (? \Sigma (s', h')) (\text{And } I \text{ (Bool (Bnot } b)))$ 
    proof -
      fix s1 h1 s2 h2 assume asm0:  $(s1, h1), (s2, h2) \models \text{And } I \text{ (Low } b)$ 
      then have asm0-bis:  $(s1, h1), (s2, h2) \models I \wedge \text{bdenot } b \text{ (fst } (s1, h1)) = \text{bdenot } b \text{ (fst } (s2, h2))$  by auto
      show pair-sat (not-set b  $(\bigcup n. \text{iterate-sigma } n \text{ b } I \Sigma (s1, h1))$ ) (not-set b  $(\bigcup n. \text{iterate-sigma } n \text{ b } I \Sigma (s2, h2))$ ) (And I (Bool (Bnot b)))
      proof (rule pair-satI)
        fix s1' h1' s2' h2'
        assume asm1:  $(s1', h1') \in \text{not-set } b (\bigcup n. \text{iterate-sigma } n \text{ b } I \Sigma (s1, h1))$ 
           $\wedge (s2', h2') \in \text{not-set } b (\bigcup n. \text{iterate-sigma } n \text{ b } I \Sigma (s2, h2))$ 
        then obtain k where not-set b  $(\bigcup n. \text{iterate-sigma } n \text{ b } I \Sigma (s1, h1)) = \text{not-set } b \text{ (iterate-sigma } k \text{ b } I \Sigma (s1, h1))$ 

```

$\text{not-set } b \ (\bigcup n. \text{iterate-sigma } n \ b \ I \ \Sigma \ (s2, h2)) = \text{not-set } b \ (\text{iterate-sigma } k \ b \ I \ \Sigma \ (s2, h2))$
using *union-exists-at-some-point-exactly*[*of I b Σ (s1, h1) (s2, h2)*] *asm0-bis not-set-def*
using $\langle \bigwedge \sigma' \sigma. \sigma, \sigma' \models \text{And } I \ (\text{Bool } b) \implies \text{pair-sat } (\Sigma \sigma) \ (\Sigma \sigma') \ (\text{And } I \ (\text{Low } b)) \rangle$ **by** *blast*
moreover have $\text{pair-sat } (\text{iterate-sigma } k \ b \ I \ \Sigma \ (s1, h1)) \ (\text{iterate-sigma } k \ b \ I \ \Sigma \ (s2, h2)) \ (\text{And } I \ (\text{Low } b))$
using $\langle \bigwedge \sigma' \sigma. \sigma, \sigma' \models \text{And } I \ (\text{Bool } b) \implies \text{pair-sat } (\Sigma \sigma) \ (\Sigma \sigma') \ (\text{And } I \ (\text{Low } b)) \rangle$ *asm0-bis iterate-sigma-low-all-sat-I-and-low* **by** *blast*
ultimately show $(s1', h1'), (s2', h2') \models \text{And } I \ (\text{Bool } (\text{Bnot } b))$
by (*metis (no-types, lifting) asm1 bdenot.simps(3) fst-conv hyper-sat.simps(1) hyper-sat.simps(3) member-filter not-set-def pair-satE*)
qed
qed

fix $s \ h \ n$
assume $\text{asm1}: (s, h), (s, h) \models \text{And } I \ (\text{Low } b)$

have $\text{safe } n \ \Delta \ (\text{Cwhile } b \ C) \ (s, h) \ (\text{trans-}\Sigma \ b \ I \ \Sigma \ (s, h))$
proof (*rule safe-while*)
show $\bigwedge \sigma \sigma'. \sigma, \sigma' \models \text{And } I \ (\text{Bool } b) \implies \text{pair-sat } (\Sigma \sigma) \ (\Sigma \sigma') \ I$
by (*meson* $\langle \bigwedge \sigma' \sigma. \sigma, \sigma' \models \text{And } I \ (\text{Bool } b) \implies \text{pair-sat } (\Sigma \sigma) \ (\Sigma \sigma') \ (\text{And } I \ (\text{Low } b)) \rangle$ *hyper-sat.simps(3) pair-sat-def*)
show $(s, h), (s, h) \models I$
using *asm1* **by** *auto*
show $\text{leads-to-loop } b \ I \ \Sigma \ (s, h) \ (s, h)$
by (*simp add: leads-to-loop.intros(1)*)
show $\bigwedge \sigma \ m. \sigma, \sigma \models \text{And } I \ (\text{Bool } b) \implies \text{safe } n \ \Delta \ C \ \sigma \ (\Sigma \sigma)$
by (*simp add: asm0*)
qed

then show $\text{safe } n \ \Delta \ (\text{Cwhile } b \ C) \ (s, h) \ (\text{not-set } b \ (\bigcup n. \text{iterate-sigma } n \ b \ I \ \Sigma \ (s, h)))$
by (*simp add: not-set-def safe-larger-set trans-included*)
qed
qed

lemma *entails-smallerI:*

assumes $\bigwedge s1 \ h1 \ s2 \ h2. (s1, h1), (s2, h2) \models A \implies (s1, h1), (s2, h2) \models B$
shows *entails A B*
by (*simp add: asms entails-def*)

corollary *while-rule:*

fixes $\Delta :: ('i, 'a, \text{nat}) \text{ cont}$
assumes *entails P (Star P' R)*
and *unary P'*
and $\text{fv } A \ R \cap \text{wr } C \ C = \{\}$

```

    and hoare-triple-valid  $\Delta$  (And  $P'$  (Bool  $e$ ))  $C$   $P'$ 
    and hoare-triple-valid  $\Delta$  (And  $P$  (Bool (Band  $e$   $e'$ )))  $C$  (And  $P$  (Low (Band
 $e$   $e'$ )))
    and precise  $P' \vee$  precise  $R$ 
    shows hoare-triple-valid  $\Delta$  (And  $P$  (Low (Band  $e$   $e'$ ))) (Cseq (Cwhile (Band  $e$ 
 $e'$ )  $C$ ) (Cwhile  $e$   $C$ )) (And (Star  $P' R$ ) (Bool (Bnot  $e$ )))
    proof (rule seq-rule)

    show hoare-triple-valid  $\Delta$  (And  $P$  (Low (Band  $e$   $e'$ ))) (Cwhile (Band  $e$   $e'$ )  $C$ )
(And  $P$  (Bool (Bnot (Band  $e$   $e'$ ))))
    proof (rule while-rule1)
    show hoare-triple-valid  $\Delta$  (And  $P$  (Bool (Band  $e$   $e'$ )))  $C$  (And  $P$  (Low (Band
 $e$   $e'$ )))
    by (simp add: assms(5))
    qed

    show hoare-triple-valid  $\Delta$  (And  $P$  (Bool (Bnot (Band  $e$   $e'$ )))) (Cwhile  $e$   $C$ ) (And
(Star  $P' R$ ) (Bool (Bnot  $e$ )))
    proof (rule consequence-rule)
    show hoare-triple-valid  $\Delta$  (Star  $P' R$ ) (Cwhile  $e$   $C$ ) (Star (And  $P'$  (Bool (Bnot
 $e$ )))  $R$ )
    proof (rule frame-rule)
    show precise  $P' \vee$  precise  $R$ 
    by (simp add: assms(6))
    show disjoint (fvA  $R$ ) (wrC (Cwhile  $e$   $C$ ))
    by (simp add: assms(3) disjoint-def)
    show hoare-triple-valid  $\Delta$   $P'$  (Cwhile  $e$   $C$ ) (And  $P'$  (Bool (Bnot  $e$ )))
    proof (rule while-rule2)
    show hoare-triple-valid  $\Delta$  (And  $P'$  (Bool  $e$ ))  $C$   $P'$ 
    by (simp add: assms(4))
    show unary  $P'$  using assms(2) by auto
    qed
    qed
    show entails (And  $P$  (Bool (Bnot (Band  $e$   $e'$ )))) (Star  $P' R$ )
    using assms(1) entails-def hyper-sat.simps(3) by blast
    show entails (Star (And  $P'$  (Bool (Bnot  $e$ )))  $R$ ) (And (Star  $P' R$ ) (Bool (Bnot
 $e$ )))
    proof (rule entails-smallerI)
    fix  $s1$   $h1$   $s2$   $h2$ 
    assume  $asm0$ :  $(s1, h1), (s2, h2) \models$  Star (And  $P'$  (Bool (Bnot  $e$ )))  $R$ 
    then obtain  $hp1$   $hr1$   $hp2$   $hr2$  where  $\text{Some } h1 = \text{Some } hp1 \oplus \text{Some } hr1$ 
Some  $h2 = \text{Some } hp2 \oplus \text{Some } hr2$ 
     $(s1, hp1), (s2, hp2) \models$  And  $P'$  (Bool (Bnot  $e$ ))  $(s1, hr1), (s2, hr2) \models$   $R$ 
    using hyper-sat.simps(4) by blast
    then show  $(s1, h1), (s2, h2) \models$  And (Star  $P' R$ ) (Bool (Bnot  $e$ ))
    by fastforce
    qed
    qed
    qed

```

4.4.15 CommCSL is sound

theorem *soundness*:

assumes $\Delta \vdash \{P\} C \{Q\}$

shows $\Delta \models \{P\} C \{Q\}$

using *assms*

proof (*induct rule: CommCSL.induct*)

case (*RuleAtomicShared* $\Gamma f \alpha \text{ sact uact } J P Q x C \text{ map-to-arg sarg ms map-to-multiset } \pi$)

then show *?case* **using** *atomic-rule-shared* **by** *blast*

qed (*simp-all add: rule-skip assign-rule new-rule write-rule read-rule share-rule atomic-rule-unique*

rule-par if1-rule if2-rule seq-rule frame-rule consequence-rule existential-rule while-rule1 while-rule2)

4.5 Corollaries

theorem *safety*:

assumes *hoare-triple-valid* (*None* :: (*'i, 'a, nat*) *cont*) $P C Q$

and $(s1, h1), (s2, h2) \models P$

and *Some* $H1 = \text{Some } h1 \oplus \text{Some } hf1 \wedge \text{full-ownership } (\text{get-fh } H1) \wedge \text{no-guard } H1$

— extend $h1$ to a normal state $H1$ without guards

and *Some* $H2 = \text{Some } h2 \oplus \text{Some } hf2 \wedge \text{full-ownership } (\text{get-fh } H2) \wedge \text{no-guard } H2$

— extend $h2$ to a normal state $H2$ without guards

shows $\bigwedge \sigma' C'. \text{red-rtrans } C (s1, \text{normalize } (\text{get-fh } H1)) C' \sigma' \Longrightarrow \neg \text{aborts } C' \sigma'$

and $\bigwedge \sigma' C'. \text{red-rtrans } C (s2, \text{normalize } (\text{get-fh } H2)) C' \sigma' \Longrightarrow \neg \text{aborts } C' \sigma'$

and $\bigwedge \sigma 1' \sigma 2'. \text{red-rtrans } C (s1, \text{normalize } (\text{get-fh } H1)) C \text{skip } \sigma 1' \Longrightarrow \text{red-rtrans } C (s2, \text{normalize } (\text{get-fh } H2)) C \text{skip } \sigma 2' \Longrightarrow (\exists h1' h2' H1' H2'. \text{no-guard } H1' \wedge \text{full-ownership } (\text{get-fh } H1') \wedge \text{snd } \sigma 1' = \text{normalize } (\text{get-fh } H1') \wedge \text{Some } H1' = \text{Some } h1' \oplus \text{Some } hf1 \wedge \text{no-guard } H2' \wedge \text{full-ownership } (\text{get-fh } H2') \wedge \text{snd } \sigma 2' = \text{normalize } (\text{get-fh } H2') \wedge \text{Some } H2' = \text{Some } h2' \oplus \text{Some } hf2 \wedge (\text{fst } \sigma 1', h1'), (\text{fst } \sigma 2', h2') \models Q$

proof —

obtain Σ **where** *asm0*: $\bigwedge \sigma n. \sigma, \sigma \models P \Longrightarrow \text{safe } n (\text{None} :: (\text{'i, 'a, nat}) \text{cont}) C \sigma (\Sigma \sigma)$

$\bigwedge \sigma \sigma'. \sigma, \sigma' \models P \Longrightarrow \text{pair-sat } (\Sigma \sigma) (\Sigma \sigma') Q$

using *assms(1) hoare-triple-validE* **by** *blast*

then have *pair-sat* $(\Sigma (s1, h1)) (\Sigma (s2, h2)) Q$

using *assms(2)* **by** *blast*

moreover have $\bigwedge n. \text{safe } n (\text{None} :: (\text{'i, 'a, nat}) \text{cont}) C (s1, h1) (\Sigma (s1, h1))$

using *always-sat-refl asm0(1) assms(2)* **by** *blast*

then show $\bigwedge \sigma' C'. \text{red-rtrans } C (s1, \text{FractionalHeap.normalize } (\text{get-fh } H1)) C'$

$\sigma' \implies \neg \text{aborts } C' \sigma'$
proof –
fix $\sigma' C'$
assume $\text{red-rtrans } C (s1, \text{FractionalHeap.normalize } (\text{get-fh } H1)) C' \sigma'$
then show $\neg \text{aborts } C' \sigma'$
using $\text{safe-atomic}[\text{of } C (s1, \text{FractionalHeap.normalize } (\text{get-fh } H1)) C' \sigma' s1$
 $\text{FractionalHeap.normalize } (\text{get-fh } H1) \text{fst } \sigma' \text{snd } \sigma']$
by $(\text{metis } \langle \wedge n. \text{safe } n \text{ None } C (s1, h1) (\Sigma (s1, h1)) \rangle \text{assms}(3) \text{denormalize-properties}(4) \text{prod.exhaust-sel})$
qed
moreover have $\wedge n. \text{safe } n (\text{None} :: ('i, 'a, \text{nat}) \text{cont}) C (s2, h2) (\Sigma (s2, h2))$
using $\text{always-sat-refl } \text{asm0}(1) \text{assms}(2) \text{sat-comm-aux}$ **by** blast
then show $\wedge \sigma' C'. \text{red-rtrans } C (s2, \text{FractionalHeap.normalize } (\text{get-fh } H2)) C' \sigma' \implies \neg \text{aborts } C' \sigma'$
proof –
fix $\sigma' C'$
assume $\text{red-rtrans } C (s2, \text{FractionalHeap.normalize } (\text{get-fh } H2)) C' \sigma'$
then show $\neg \text{aborts } C' \sigma'$
using $\text{safe-atomic}[\text{of } C (s2, \text{FractionalHeap.normalize } (\text{get-fh } H2)) C' \sigma' s2$
 $\text{FractionalHeap.normalize } (\text{get-fh } H2) \text{fst } \sigma' \text{snd } \sigma']$
by $(\text{metis } \langle \wedge n. \text{safe } n \text{ None } C (s2, h2) (\Sigma (s2, h2)) \rangle \text{assms}(4) \text{denormalize-properties}(4) \text{prod.exhaust-sel})$
qed
fix $\sigma 1'$
assume $\text{red-rtrans } C (s1, \text{FractionalHeap.normalize } (\text{get-fh } H1)) C \text{skip } \sigma 1'$
then obtain $h1' H1'$ **where** $r1: \text{Some } H1' = \text{Some } h1' \oplus \text{Some } hf1 \text{snd } \sigma 1' =$
 $\text{FractionalHeap.normalize } (\text{get-fh } H1')$
 $\text{no-guard } H1' \wedge \text{full-ownership } (\text{get-fh } H1') (\text{fst } \sigma 1', h1') \in \Sigma (s1, h1)$
using $\text{safe-atomic}[\text{of } C (s1, \text{FractionalHeap.normalize } (\text{get-fh } H1)) C \text{skip } \sigma 1'$
 $s1 - \text{fst } \sigma 1' \text{snd } \sigma 1' h1 \Sigma (s1, h1) H1 hf1]$
by $(\text{metis } \langle \wedge n. \text{safe } n \text{ None } C (s1, h1) (\Sigma (s1, h1)) \rangle \text{assms}(3) \text{denormalize-properties}(4) \text{surjective-pairing})$
fix $\sigma 2'$
assume $\text{red-rtrans } C (s2, \text{FractionalHeap.normalize } (\text{get-fh } H2)) C \text{skip } \sigma 2'$
then obtain $h2' H2'$ **where** $r2: \text{Some } H2' = \text{Some } h2' \oplus \text{Some } hf2 \text{snd } \sigma 2' =$
 $\text{FractionalHeap.normalize } (\text{get-fh } H2')$
 $\text{no-guard } H2' \wedge \text{full-ownership } (\text{get-fh } H2') (\text{fst } \sigma 2', h2') \in \Sigma (s2, h2)$
using $\text{safe-atomic}[\text{of } C (s2, \text{FractionalHeap.normalize } (\text{get-fh } H2)) C \text{skip } \sigma 2'$
 $s2 - \text{fst } \sigma 2' \text{snd } \sigma 2' h2 \Sigma (s2, h2) H2 hf2]$
by $(\text{metis } \langle \wedge n. \text{safe } n \text{ None } C (s2, h2) (\Sigma (s2, h2)) \rangle \text{assms}(4) \text{denormalize-properties}(4) \text{surjective-pairing})$
then have $(\text{fst } \sigma 1', h1'), (\text{fst } \sigma 2', h2') \models Q$
using $\text{calculation}(1) \text{pair-satE } r1(4)$ **by** blast
then show $\exists h1' h2' H1' H2'.$
 $\text{no-guard } H1' \wedge$
 $\text{full-ownership } (\text{get-fh } H1') \wedge$
 $\text{snd } \sigma 1' = \text{FractionalHeap.normalize } (\text{get-fh } H1') \wedge$
 $\text{Some } H1' = \text{Some } h1' \oplus \text{Some } hf1 \wedge$
 $\text{no-guard } H2' \wedge$

$full\text{-}ownership (get\text{-}fh H2') \wedge snd \sigma 2' = FractionalHeap.normalize (get\text{-}fh H2') \wedge Some H2' = Some h2' \oplus Some hf2 \wedge (fst \sigma 1', h1'), (fst \sigma 2', h2') \models Q$
using $r1 r2$ **by** *blast*
qed

lemma *neutral-add*:

$Some h = Some h \oplus Some (Map.empty, None, (\lambda\cdot None))$

proof –

have $h \#\# (Map.empty, None, (\lambda\cdot None))$

by (*metis compatibleI compatible-fract-heapsI empty-heap-def fst-conv get-fh.elims get-gs.simps get-gu.simps option.distinct(1) snd-conv*)

then obtain x **where** $Some x = Some h \oplus Some (Map.empty, None, (\lambda\cdot None))$

by *simp*

moreover have $x = h$

by (*metis (no-types, lifting) addition-cancellative calculation decompose-guard-remove-easy fst-eqD get-gs.simps get-gu.simps no-guard-def no-guards-remove prod.sel(2) simpler-asso*)

ultimately show *?thesis* **by** *blast*

qed

corollary *safety-no-frame*:

assumes *hoare-triple-valid* ($None :: ('i, 'a, nat) cont$) $P C Q$

and $(s1, H1), (s2, H2) \models P$

and *full-ownership* $(get\text{-}fh H1) \wedge no\text{-}guard H1$

and *full-ownership* $(get\text{-}fh H2) \wedge no\text{-}guard H2$

shows $\bigwedge \sigma' C'. red\text{-}rtrans C (s1, normalize (get\text{-}fh H1)) C' \sigma' \implies \neg aborts C' \sigma'$

and $\bigwedge \sigma' C'. red\text{-}rtrans C (s2, normalize (get\text{-}fh H2)) C' \sigma' \implies \neg aborts C' \sigma'$

and $\bigwedge \sigma 1' \sigma 2'. red\text{-}rtrans C (s1, normalize (get\text{-}fh H1)) Cskip \sigma 1'$

$\implies red\text{-}rtrans C (s2, normalize (get\text{-}fh H2)) Cskip \sigma 2'$

$\implies (\exists H1' H2'. no\text{-}guard H1' \wedge full\text{-}ownership (get\text{-}fh H1') \wedge snd \sigma 1' = normalize (get\text{-}fh H1')$

$\wedge no\text{-}guard H2' \wedge full\text{-}ownership (get\text{-}fh H2') \wedge snd \sigma 2' = normalize (get\text{-}fh H2')$

$\wedge (fst \sigma 1', H1'), (fst \sigma 2', H2') \models Q$)

proof –

have $Some H1 = Some H1 \oplus Some (Map.empty, None, (\lambda\cdot None))$

using *neutral-add* **by** *blast*

moreover have $Some H2 = Some H2 \oplus Some (Map.empty, None, (\lambda\cdot None))$

using *neutral-add* **by** *blast*

show $\bigwedge \sigma' C'. red\text{-}rtrans C (s1, FractionalHeap.normalize (get\text{-}fh H1)) C' \sigma' \implies \neg aborts C' \sigma'$

using *always-sat-refl-aux* *assms(1)* *assms(2)* *assms(3)* *calculation* *safety(2)* **by** *blast*

show $\bigwedge \sigma' C'. red\text{-}rtrans C (s2, FractionalHeap.normalize (get\text{-}fh H2)) C' \sigma' \implies \neg aborts C' \sigma'$

```

using ⟨Some H2 = Some H2 ⊕ Some (Map.empty, None, (λ-. None))⟩ assms(1)
assms(2) assms(3) assms(4) calculation safety(2) by blast
fix  $\sigma 1' \sigma 2'$ 
assume red-rtrans C (s1, FractionalHeap.normalize (get-fh H1)) Cskip  $\sigma 1'$ 
      red-rtrans C (s2, FractionalHeap.normalize (get-fh H2)) Cskip  $\sigma 2'$ 

then obtain  $h1' h2' H1' H2'$  where asm0: no-guard H1' ∧ full-ownership
(get-fh H1') ∧ snd  $\sigma 1' = \textit{normalize} (get-fh H1') ∧ Some H1' = Some h1' ⊕ Some
(Map.empty, None, (λ-. None))
  ∧ no-guard H2' ∧ full-ownership (get-fh H2') ∧ snd  $\sigma 2' = \textit{normalize} (get-fh
H2') ∧ Some H2' = Some h2' ⊕ Some (Map.empty, None, (λ-. None))
  ∧ (fst  $\sigma 1'$ , h1'), (fst  $\sigma 2'$ , h2')  $\models Q$ 
using safety[of P C Q s1 H1 s2 H2 H1 (Map.empty, None, (λ-. None)) H2
(Map.empty, None, (λ-. None))] assms
by (metis (no-types, lifting) ⟨Some H2 = Some H2 ⊕ Some (Map.empty, None,
(λ-. None))⟩ calculation)
then have  $H1' = h1'$ 
using addition-cancellative decompose-guard-remove-easy denormalize-properties(4)
denormalize-properties(5)
by (metis denormalize-def get-gs.simps get-gu.simps prod.exhaust-sel snd-conv)
moreover have  $H2' = h2'$ 
by (metis asm0 denormalize-properties(4) denormalize-properties(5) fst-eqD
get-fh.elims no-guard-and-no-heap no-guard-then-smaller-same)

ultimately show  $\exists H1' H2'$ .
  no-guard H1' ∧
  full-ownership (get-fh H1') ∧
  snd  $\sigma 1' = \textit{FractionalHeap.normalize} (get-fh H1') ∧
  no-guard H2' ∧ full-ownership (get-fh H2') ∧ snd  $\sigma 2' = \textit{Fractional-}$ 
Heap.normalize (get-fh H2') ∧ (fst  $\sigma 1'$ , H1'), (fst  $\sigma 2'$ , H2')  $\models Q$ 
using asm0 by blast
qed

end
theory NonInterference
imports Soundness
begin

fun low-list where
  low-list [] = Bool Btrue
| low-list (v # q) = And (LowExp (Evar v)) (low-list q)

lemma low-listE:
assumes (s1, h1), (s2, h2)  $\models \textit{low-list l}$ 
and  $x \in \textit{set l}$ 
shows  $s1\ x = s2\ x$ 
using assms
proof (induct l)
case (Cons a l)$$$ 
```

```

then show ?case
proof (cases x = a)
  case True
    then have (s1, h1), (s2, h2)  $\models$  LowExp (Evar a)
      using Cons.prem(1) by auto
    then show ?thesis
      by (simp add: True)
  next
    case False
    then show ?thesis
      using Cons.hyps Cons.prem(1) Cons.prem(2) by auto
qed
qed (simp)

```

```

lemma low-listI:
  assumes  $\bigwedge x. x \in \text{set } l \implies s1\ x = s2\ x$ 
  shows (s1, h1), (s2, h2)  $\models$  low-list l
  using assms
by (induct l) simp-all

```

corollary non-interference:

```

assumes (None :: ('i, 'a, nat) cont)  $\vdash$  {And P (low-list In)} C {low-list Out}
  and red-rtrans C (s1, normalize (get-fh H1)) Cskip (s1', h1')
  and red-rtrans C (s2, normalize (get-fh H2)) Cskip (s2', h2')
  and  $\bigwedge x. x \in \text{set } In \implies s1\ x = s2\ x$ 
  and x  $\in$  set Out
  and (s1, H1), (s2, H2)  $\models$  P
  and full-ownership (get-fh H1)  $\wedge$  no-guard H1
  and full-ownership (get-fh H2)  $\wedge$  no-guard H2
  shows s1' x = s2' x
proof -
  have  $\exists H1'\ H2'. \text{no-guard } H1' \wedge \text{full-ownership (get-fh } H1') \wedge \text{snd (s1', h1')} =$ 
    FractionalHeap.normalize (get-fh H1')  $\wedge$ 
    no-guard H2'  $\wedge$  full-ownership (get-fh H2')  $\wedge$  snd (s2', h2') = Fractional-
    Heap.normalize (get-fh H2')
     $\wedge$  (fst (s1', h1'), H1'), (fst (s2', h2'), H2')  $\models$  (low-list Out :: ('i, 'a, nat) asser-
    tion)
  proof (rule safety-no-frame(3))
    show (None :: ('i, 'a, nat) cont)  $\models$  {And P (low-list In)} C {low-list Out}
      using assms(1) soundness by blast
    have (s1, H1), (s2, H2)  $\models$  low-list In
      by (simp add: assms(4) low-listI)
    then show (s1, H1), (s2, H2)  $\models$  And P (low-list In)
      by (simp add: assms(6))
    qed ((insert assms; blast)+)
  then show ?thesis
    by (metis assms(5) fst-conv low-listE)
qed

```


definition *heapify where*
 $heapify\ h = (\lambda l. apply-opt\ (\lambda v. (pwrite, v))\ (h\ l),\ None, \lambda-. None)$

lemma *heapify-properties:*
 $full-ownership\ (get-fh\ (heapify\ h))$
 $no-guard\ (heapify\ h)$
 $normalize\ (get-fh\ (heapify\ h)) = h$

proof *(rule full-ownershipI)*
fix $l\ p$ **assume** $get-fh\ (heapify\ h)\ l = Some\ p$
then show $fst\ p = pwrite$
by *(metis apply-opt.elims fst-conv get-fh.elims heapify-def option.sel option.simps(3))*
next
show $no-guard\ (heapify\ h)$
by *(metis addition-cancellative decompose-guard-remove-easy decompose-heap-triple heapify-def neutral-add no-guards-remove snd-conv)*
show $normalize\ (get-fh\ (heapify\ h)) = h$
proof *(rule ext)*
fix l **show** $FractionalHeap.normalize\ (get-fh\ (heapify\ h))\ l = h\ l$
proof *(cases h l)*
case $None$
then show *?thesis*
by *(metis apply-opt.simps(1) domIff dom-normalize fst-conv get-fh.simps heapify-def)*
next
case $(Some\ a)$
then show *?thesis*
by *(simp add: FractionalHeap.normalize-eq(2) heapify-def)*
qed
qed
qed

corollary *non-interference-no-precondition:*
assumes $(None :: ('i, 'a, nat)\ cont) \vdash \{low-list\ In\} C \{low-list\ Out\}$
and $red-rtrans\ C\ (s1, h1)\ Cskip\ (s1', h1')$
and $red-rtrans\ C\ (s2, h2)\ Cskip\ (s2', h2')$
and $\bigwedge x. x \in set\ In \implies s1\ x = s2\ x$
and $x \in set\ Out$
shows $s1'\ x = s2'\ x$

proof *(rule non-interference)*
show $(None :: ('i, 'a, nat)\ cont) \vdash \{And\ (Bool\ Btrue)\ (low-list\ In)\} C \{low-list\ Out\}$
using *RuleCons assms(1) entails-def hyper-sat.simps(3)* **by** *blast*
show $red-rtrans\ C\ (s1, FractionalHeap.normalize\ (get-fh\ (heapify\ h1)))\ Cskip\ (s1', h1')$
by *(metis assms(2) heapify-properties(3))*
show $red-rtrans\ C\ (s2, FractionalHeap.normalize\ (get-fh\ (heapify\ h2)))\ Cskip\ (s2', h2')$
by *(metis assms(3) heapify-properties(3))*
qed *(insert assms heapify-properties; auto)+*

end

References

- [1] M. Eilers, T. Dardinier, and P. Müller. CommCSL: Proving information flow security for concurrent programs using abstract commutativity, 2022.
- [2] V. Vafeiadis. Concurrent separation logic and operational semantics. In M. W. Mislove and J. Ouaknine, editors, *Twenty-seventh Conference on the Mathematical Foundations of Programming Semantics, MFPS 2011, Pittsburgh, PA, USA, May 25-28, 2011*, volume 276 of *Electronic Notes in Theoretical Computer Science*, pages 335–351. Elsevier, 2011.