Formalization of Hyper Hoare Logic: A Logic to (Dis-)Prove Program Hyperproperties

Thibault Dardinier
Department of Computer Science
ETH Zurich, Switzerland

September 13, 2023

Abstract

Hoare logics [5, 6] are proof systems that allow one to formally establish properties of computer programs. Traditional Hoare logics prove properties of individual program executions (so-called trace properties, such as functional correctness). On the one hand, Hoare logic has been generalized to prove properties of multiple executions of a program (so-called hyperproperties [1], such as determinism or non-interference). These program logics prove the absence of (bad combinations of) executions. On the other hand, program logics similar to Hoare logic have been proposed to disprove program properties (e.g., Incorrectness Logic [8]), by proving the existence of (bad combinations of) executions. All of these logics have in common that they specify program properties using assertions over a fixed number of states, for instance, a single pre- and post-state for functional properties or pairs of pre- and post-states for non-interference.

In this entry, we formalize Hyper Hoare Logic [2], a generalization of Hoare logic that lifts assertions to properties of arbitrary sets of states. The resulting logic is simple yet expressive: its judgments can express arbitrary trace- and hyperproperties over the terminating executions of a program. By allowing assertions to reason about sets of states, Hyper Hoare Logic can reason about both the absence and the existence of (combinations of) executions, and, thereby, supports both proving and disproving program (hyper-)properties within the same logic. In fact, we prove that Hyper Hoare Logic subsumes the properties handled by numerous existing correctness and incorrectness logics, and can express hyperproperties that no existing Hoare logic can. We also prove that Hyper Hoare Logic is sound and complete.

Contents

1 Language and Semantics

In this file, we formalize the programming language from section III, and the extended states and semantics from section IV of the paper [2]. We also prove the useful properties described by Lemma 1.

theory *Language*
  imports *Main*
begin

1.1 Language

Definition 1

type-synonym $('var, 'val)\ pstate = 'var \Rightarrow 'val$

Definition 2

type-synonym $('var, 'val)\ bexp = ('var, 'val)\ pstate \Rightarrow bool$

type-synonym ('*var*, '*val*) *exp* = ('*var*, '*val*) *pstate* ⇒ '*val*

datatype ('*var*, '*val*) *stmt* =
  Assign '*var* ('*var*, '*val*) *exp*
  | *Seq* ('*var*, '*val*) *stmt* ('*var*, '*val*) *stmt*
  | *If* ('*var*, '*val*) *stmt* ('*var*, '*val*) *stmt*
  | *Skip*
  | *Havoc* '*var*
  | *Assume* ('*var*, '*val*) *bexp*
  | *While* ('*var*, '*val*) *stmt*

1.2 Semantics

Figure 2

inductive *single-sem* :: ('*var*, '*val*) *stmt* ⇒ ('*var*, '*val*) *pstate* ⇒ ('*var*, '*val*) *pstate* ⇒ *bool*
  (⟨-, -⟩ → - [*51*,*0*] *81*)
  where
  SemSkip: $\langle Skip, \sigma\rangle \rightarrow \sigma$
| *SemAssign*: $\langle Assign\ var\ e, \sigma\rangle \rightarrow \sigma(var := (e\ \sigma))$
| *SemSeq*: $[\![\ \langle C1, \sigma\rangle \rightarrow \sigma 1; \langle C2, \sigma 1\rangle \rightarrow \sigma 2\ ]\!] \Longrightarrow \langle Seq\ C1\ C2, \sigma\rangle \rightarrow \sigma 2$
| *SemIf1*: $\langle C1, \sigma\rangle \rightarrow \sigma 1 \Longrightarrow \langle If\ C1\ C2, \sigma\rangle \rightarrow \sigma 1$
| *SemIf2*: $\langle C2, \sigma\rangle \rightarrow \sigma 2 \Longrightarrow \langle If\ C1\ C2, \sigma\rangle \rightarrow \sigma 2$
| *SemHavoc*: $\langle Havoc\ var, \sigma\rangle \rightarrow \sigma(var := v)$
| *SemAssume*: $b\ \sigma \Longrightarrow \langle Assume\ b, \sigma\rangle \rightarrow \sigma$
| *SemWhileIter*: $[\![\ \langle C, \sigma\rangle \rightarrow \sigma'\ ; \langle While\ C, \sigma'\rangle \rightarrow \sigma''\ ]\!] \Longrightarrow \langle While\ C, \sigma\rangle \rightarrow \sigma''$
| *SemWhileExit*: $\langle While\ C, \sigma\rangle \rightarrow \sigma$

inductive-cases *single-sem-Seq-elim*[*elim*!]: $\langle Seq\ C1\ C2, \sigma\rangle \rightarrow \sigma'$
inductive-cases *single-sem-Skip-elim*[*elim*!]: $\langle Skip, \sigma\rangle \rightarrow \sigma'$
inductive-cases *single-sem-While-elim*: $\langle While\ C, \sigma\rangle \rightarrow \sigma'$
inductive-cases *single-sem-If-elim*[*elim*!]: $\langle If\ C1\ C2, \sigma\rangle \rightarrow \sigma'$
inductive-cases *single-sem-Assume-elim*[*elim*!]: $\langle Assume\ b, \sigma\rangle \rightarrow \sigma'$
inductive-cases *single-sem-Assign-elim*[*elim*!]: $\langle Assign\ x\ e, \sigma\rangle \rightarrow \sigma'$
inductive-cases *single-sem-Havoc-elim*[*elim*!]: $\langle Havoc\ x, \sigma\rangle \rightarrow \sigma'$

2 Extended States and Extended Semantics

Definition 3

type-synonym ('*lvar*, '*lval*, '*pvar*, '*pval*) *state* = ('*lvar* ⇒ '*lval*) × ('*pvar*, '*pval*) *pstate*

Definition 5

definition *sem* :: ('*pvar*, '*pval*) *stmt* ⇒ ('*lvar*, '*lval*, '*pvar*, '*pval*) *state set* ⇒ ('*lvar*, '*lval*, '*pvar*, '*pval*) *state set* **where**
  $sem\ C\ S = \{\ (l, \sigma')\ |\sigma'\ \sigma\ l.\ (l, \sigma) \in S \wedge \langle C, \sigma\rangle \rightarrow \sigma'\ \}$

lemma *in-sem*:
$\varphi \in sem\ C\ S \longleftrightarrow (\exists\sigma.\ (fst\ \varphi,\ \sigma) \in S \wedge single\text{-}sem\ C\ \sigma\ (snd\ \varphi))$ (**is** $?A \longleftrightarrow ?B$)
proof
  assume $?A$
  then obtain $\sigma'\ \sigma\ l$ **where** $\varphi = (l,\ \sigma')\ (l,\ \sigma) \in S \wedge \langle C,\ \sigma\rangle \rightarrow \sigma'$
    using *sem-def*[*of C S*] **by** *auto*
  then show $?B$
    by *auto*
next
  show $?B \Longrightarrow ?A$
    by (*metis* (*mono-tags*, *lifting*) *CollectI prod.collapse sem-def*)
qed

Useful properties

lemma *sem-seq*:
$sem\ (Seq\ C1\ C2)\ S = sem\ C2\ (sem\ C1\ S)$ (**is** $?A = ?B$)
proof
  show $?A \subseteq ?B$
  proof
    fix $x2$ **assume** $x2 \in ?A$
    then obtain $x0$ **where** $(fst\ x2,\ x0) \in S\ single\text{-}sem\ (Seq\ C1\ C2)\ x0\ (snd\ x2)$
      by (*metis in-sem*)
    then obtain $x1$ **where** $single\text{-}sem\ C1\ x0\ x1\ single\text{-}sem\ C2\ x1\ (snd\ x2)$
      using *single-sem-Seq-elim*[*of C1 C2 x0 snd x2*]
      by *blast*
    then show $x2 \in ?B$
      by (*metis* ‹$(fst\ x2,\ x0) \in S$› *fst-conv in-sem snd-conv*)
  qed
  show $?B \subseteq ?A$
  proof
    fix $x2$ **assume** $x2 \in ?B$
    then obtain $x1$ **where** $(fst\ x2,\ x1) \in sem\ C1\ S\ single\text{-}sem\ C2\ x1\ (snd\ x2)$
      by (*metis in-sem*)
    then obtain $x0$ **where** $(fst\ x2,\ x0) \in S\ single\text{-}sem\ C1\ x0\ x1$
      by (*metis fst-conv in-sem snd-conv*)
    then have $single\text{-}sem\ (Seq\ C1\ C2)\ x0\ (snd\ x2)$
      by (*simp add*: *SemSeq* ‹$\langle C2,\ x1\rangle \rightarrow snd\ x2$›)
    then show $x2 \in ?A$
      by (*meson* ‹$(fst\ x2,\ x0) \in S$› *in-sem*)
  qed
qed

lemma *sem-skip*:
$sem\ Skip\ S = S$
  using *single-sem-Skip-elim SemSkip in-sem*[*of - Skip S*]
  by *fastforce*

```
lemma sem-union:
  sem C (S1 ∪ S2) = sem C S1 ∪ sem C S2 (is ?A = ?B)
proof
  show ?A ⊆ ?B
  proof
    fix x assume x ∈ ?A
    then obtain y where (fst x, y) ∈ S1 ∪ S2 single-sem C y (snd x)
      using in-sem by blast
    then show x ∈ ?B
      by (metis Un-iff in-sem)
  qed
  show ?B ⊆ ?A
  proof
    fix x assume x ∈ ?B
    show x ∈ ?A
    proof (cases x ∈ sem C S1)
      case True
      then show ?thesis
        by (metis IntD2 Un-Int-eq(3) in-sem)
    next
      case False
      then show ?thesis
        by (metis Un-iff ‹x ∈ sem C S1 ∪ sem C S2› in-sem)
    qed
  qed
qed

lemma sem-union-general:
  sem C (⋃x. f x) = (⋃x. sem C (f x)) (is ?A = ?B)
proof
  show ?A ⊆ ?B
  proof
    fix b assume b ∈ ?A
    then obtain a where a ∈ (⋃x. f x) fst a = fst b single-sem C (snd a) (snd b)
      by (metis fst-conv in-sem snd-conv)
    then obtain x where a ∈ f x by blast
    then have b ∈ sem C (f x)
      by (metis ‹⟨C, snd a⟩ → snd b› ‹fst a = fst b› in-sem surjective-pairing)
    then show b ∈ ?B
      by blast
  qed
  show ?B ⊆ ?A
  proof
    fix y assume y ∈ ?B
    then obtain x where y ∈ sem C (f x)
      by blast
    then show y ∈ ?A
      by (meson UN-I in-sem iso-tuple-UNIV-I)
  qed
```

qed

lemma *sem-monotonic*:
  assumes $S \subseteq S'$
  shows *sem* C $S \subseteq$ *sem* C S'
  by (*metis assms sem-union subset-Un-eq*)

lemma *subsetPairI*:
  assumes $\bigwedge l\ \sigma.\ (l,\ \sigma) \in A \Longrightarrow (l,\ \sigma) \in B$
  shows $A \subseteq B$
  by (*simp add*: *assms subrelI*)

lemma *sem-if*:
  sem (*If C1 C2*) S = *sem C1* $S \cup$ *sem C2* S (**is** *?A* = *?B*)
proof
  show *?A* $\subseteq$ *?B*
  proof (*rule subsetPairI*)
    fix l y **assume** $(l,\ y) \in$ *?A*
    then obtain x **where** $(l,\ x) \in S$ *single-sem* (*If C1 C2*) x y
      by (*metis fst-conv in-sem snd-conv*)
    then show $(l,\ y) \in$ *?B* **using** *single-sem-If-elim*
      UnI1 UnI2 in-sem
      by (*metis fst-conv snd-conv*)
  qed
  show *?B* $\subseteq$ *?A*
    using *SemIf1 SemIf2 in-sem*
    by (*metis* (*no-types*, *lifting*) *Un-subset-iff subsetI*)
qed

lemma *sem-assume*:
  sem (*Assume* b) S = $\{\ (l,\ \sigma)\ |l\ \sigma.\ (l,\ \sigma) \in S \wedge b\ \sigma\ \}$ (**is** *?A* = *?B*)
proof
  show *?A* $\subseteq$ *?B*
  proof (*rule subsetPairI*)
   fix l y **assume** $(l,\ y) \in$ *?A* **then obtain** x **where** $(l,\ x) \in S$ *single-sem* (*Assume* b) x y
      using *in-sem*
      by (*metis fst-conv snd-conv*)
    then show $(l,\ y) \in$ *?B* **using** *single-sem-Assume-elim* **by** *blast*
  qed
  show *?B* $\subseteq$ *?A*
  proof (*rule subsetPairI*)
    fix l σ **assume** *asm0*: $(l,\ \sigma) \in \{(l,\ \sigma)\ |l\ \sigma.\ (l,\ \sigma) \in S \wedge b\ \sigma\}$
    then have $(l,\ \sigma) \in S$ b σ **by** *simp-all*
    then show $(l,\ \sigma) \in$ *sem* (*Assume* b) S
      by (*metis SemAssume fst-eqD in-sem snd-eqD*)
  qed
qed

```
lemma while-then-reaches:
  assumes (single-sem C)** σ σ''
  shows single-sem (While C) σ σ''
  using assms
proof (induct rule: converse-rtranclp-induct)
  case base
  then show ?case
    by (simp add: SemWhileExit)
next
  case (step y z)
  then show ?case
    using SemWhileIter by blast
qed

lemma in-closure-then-while:
  assumes single-sem C' σ σ''
  shows ⋀C. C' = While C ⟹ (single-sem C)** σ σ''
  using assms
proof (induct rule: single-sem.induct)
  case (SemWhileIter σ C' σ' σ'')
  then show ?case
     by (metis (no-types, lifting) rtranclp.rtrancl-into-rtrancl rtranclp.rtrancl-refl
rtranclp-trans stmt.inject(6))
next
  case (SemWhileExit σ C')
  then show ?case
    by blast
qed (auto)

theorem loop-equiv:
  single-sem (While C) σ σ' ⟷ (single-sem C)** σ σ'
  using in-closure-then-while while-then-reaches by blast


fun iterate-sem where
  iterate-sem 0 - S = S
| iterate-sem (Suc n) C S = sem C (iterate-sem n C S)

lemma in-iterate-then-in-trans:
  assumes (l, σ'') ∈ iterate-sem n C S
  shows ∃σ. (l, σ) ∈ S ∧ (single-sem C)** σ σ''
  using assms
proof (induct n arbitrary: σ'' S)
  case 0
  then show ?case
    using iterate-sem.simps(1) by blast
next
  case (Suc n)
```

then show *?case*
using *in-sem rtranclp.rtrancl-into-rtrancl*
by (*metis* (*mono-tags*, *lifting*) *fst-conv iterate-sem.simps*(*2*) *snd-conv*)
qed

lemma *reciprocal*:
assumes (*single-sem C*)** σ σ''
and (l, σ) $\in$ S
shows $\exists\, n$. (l, σ'') $\in$ *iterate-sem n C S*
using *assms*
proof (*induct rule*: *rtranclp-induct*)
case *base*
then show *?case*
using *iterate-sem.simps*(*1*) **by** *blast*
next
case (*step y z*)
then obtain n **where** (l, y) $\in$ *iterate-sem n C S* **by** *blast*
then show *?case*
using *in-sem iterate-sem.simps*(*2*) *step.hyps*(*2*)
by (*metis fst-eqD snd-eqD*)
qed

lemma *union-iterate-sem-trans*:
(l, σ'') $\in$ ($\bigcup n$. *iterate-sem n C S*) $\longleftrightarrow$ ($\exists\sigma$. (l, σ) $\in$ S $\wedge$ (*single-sem C*)** σ σ'') (**is** *?A* $\longleftrightarrow$ *?B*)
using *in-iterate-then-in-trans reciprocal* **by** *auto*

lemma *sem-while*:
sem (*While C*) S = ($\bigcup n$. *iterate-sem n C S*) (**is** *?A* = *?B*)
proof
show *?A* $\subseteq$ *?B*
proof (*rule subsetPairI*)
fix *l y* **assume** (l, y) $\in$ *?A*
then obtain x **where** *x-def*: (l, x) $\in$ S (*single-sem C*)** x y
using *in-closure-then-while in-sem*
by (*metis fst-eqD snd-conv*)
then have *single-sem* (*While C*) x y
using *while-then-reaches* **by** *blast*
then show (l, y) $\in$ *?B*
by (*metis x-def union-iterate-sem-trans*)
qed
show *?B* $\subseteq$ *?A*
proof (*rule subsetPairI*)
fix *l y* **assume** (l, y) $\in$ *?B*
then obtain x **where** (l, x) $\in$ S (*single-sem C*)** x y
using *union-iterate-sem-trans* **by** *blast*
then show (l, y) $\in$ *?A*
using *in-sem while-then-reaches* **by** *fastforce*
qed

qed

lemma *assume-sem*:
sem (*Assume b*) *S* = *Set.filter* (*b* ∘ *snd*) *S* (**is** *?A* = *?B*)
proof
show *?A* ⊆ *?B*
proof (*rule subsetPairI*)
fix *l σ*
assume *asm0*: (*l*, *σ*) ∈ *sem* (*Assume b*) *S*
then show (*l*, *σ*) ∈ *Set.filter* (*b* ∘ *snd*) *S*
by (*metis comp-apply fst-conv in-sem member-filter single-sem-Assume-elim snd-conv*)
qed
show *?B* ⊆ *?A*
by (*metis* (*mono-tags*, *opaque-lifting*) *SemAssume comp-apply in-sem member-filter prod.collapse subsetI*)
qed

lemma *sem-split-general*:
sem C (⋃*x*. *F x*) = (⋃*x*. *sem C* (*F x*)) (**is** *?A* = *?B*)
proof
show *?A* ⊆ *?B*
proof (*rule subsetPairI*)
fix *l σ'*
assume *asm0*: (*l*, *σ'*) ∈ *sem C* (⋃ (*range F*))
then obtain *x σ* **where** (*l*, *σ*) ∈ *F x single-sem C σ σ'*
by (*metis* (*no-types*, *lifting*) *UN-iff fst-conv in-sem snd-conv*)
then show (*l*, *σ'*) ∈ (⋃*x*. *sem C* (*F x*))
using *asm0 sem-union-general* **by** *blast*
qed
show *?B* ⊆ *?A*
by (*simp add*: *SUP-least Sup-upper sem-monotonic*)
qed

end

3 Hyper Hoare Logic

In this file, we define concepts from the logic (section IV): hyper-assertions, hyper-triples, and the syntactic rules. We also prove soundness (theorem 1), completeness (theorem 2), the ability to disprove hyper-triples in the logic (theorem 4), and the synchronized if rule from appendix C.

theory *Logic*
imports *Language*
begin

Definition 4

type-synonym $'a$ *hyperassertion* $= ('a\ set \Rightarrow bool)$

definition *entails* **where**
entails $A\ B \longleftrightarrow (\forall S.\ A\ S \longrightarrow B\ S)$

lemma *entailsI*:
assumes $\bigwedge S.\ A\ S \Longrightarrow B\ S$
shows *entails* $A\ B$
by (*simp add*: *assms entails-def*)

lemma *entailsE*:
assumes *entails* $A\ B$
and $A\ x$
shows $B\ x$
by (*meson assms*(*1*) *assms*(*2*) *entails-def*)

lemma *bientails-equal*:
assumes *entails* $A\ B$
and *entails* $B\ A$
shows $A = B$
proof (*rule ext*)
fix S **show** $A\ S = B\ S$
by (*meson assms*(*1*) *assms*(*2*) *entailsE*)
qed

lemma *entails-trans*:
assumes *entails* $A\ B$
and *entails* $B\ C$
shows *entails* $A\ C$
by (*metis assms*(*1*) *assms*(*2*) *entails-def*)

definition *setify-prop* **where**
setify-prop $b = \{\ (l, \sigma)\ |l\ \sigma.\ b\ \sigma\}$

lemma *sem-assume-setify*:
sem (*Assume* b) $S = S \cap$ *setify-prop* b (**is** $?A = ?B$)
proof $-$
have $\bigwedge l\ \sigma.\ (l, \sigma) \in ?A \longleftrightarrow (l, \sigma) \in ?B$
proof $-$
fix $l\ \sigma$
have $(l, \sigma) \in ?A \longleftrightarrow (l, \sigma) \in S \land b\ \sigma$
by (*simp add*: *assume-sem*)
then show $(l, \sigma) \in ?A \longleftrightarrow (l, \sigma) \in ?B$
by (*simp add*: *setify-prop-def*)
qed
then show *?thesis*
by *auto*

qed

definition *over-approx* :: $'a\ set \Rightarrow 'a\ hyperassertion$ **where**
over-approx $P\ S \longleftrightarrow S \subseteq P$

definition *lower-closed* :: $'a\ hyperassertion \Rightarrow bool$ **where**
lower-closed $P \longleftrightarrow (\forall S\ S'.\ P\ S \wedge S' \subseteq S \longrightarrow P\ S')$

lemma *over-approx-lower-closed*:
lower-closed (*over-approx* P)
by (*metis* (*full-types*) *lower-closed-def order-trans over-approx-def*)

definition *under-approx* :: $'a\ set \Rightarrow 'a\ hyperassertion$ **where**
under-approx $P\ S \longleftrightarrow P \subseteq S$

definition *upper-closed* :: $'a\ hyperassertion \Rightarrow bool$ **where**
upper-closed $P \longleftrightarrow (\forall S\ S'.\ P\ S \wedge S \subseteq S' \longrightarrow P\ S')$

lemma *under-approx-upper-closed*:
upper-closed (*under-approx* P)
by (*metis* (*no-types*, *lifting*) *order.trans under-approx-def upper-closed-def*)

definition *closed-by-union* :: $'a\ hyperassertion \Rightarrow bool$ **where**
closed-by-union $P \longleftrightarrow (\forall S\ S'.\ P\ S \wedge P\ S' \longrightarrow P\ (S \cup S'))$

lemma *closed-by-unionI*:
assumes $\bigwedge a\ b.\ P\ a \Longrightarrow P\ b \Longrightarrow P\ (a \cup b)$
shows *closed-by-union* P
by (*simp add*: *assms closed-by-union-def*)

lemma *closed-by-union-over*:
closed-by-union (*over-approx* P)
by (*simp add*: *closed-by-union-def over-approx-def*)

lemma *closed-by-union-under*:
closed-by-union (*under-approx* P)
by (*simp add*: *closed-by-union-def sup.coboundedI1 under-approx-def*)

definition *conj* **where**
conj $P\ Q\ S \longleftrightarrow P\ S \wedge Q\ S$

definition *disj* **where**
disj $P\ Q\ S \longleftrightarrow P\ S \vee Q\ S$

definition *exists* :: $('c \Rightarrow 'a\ hyperassertion) \Rightarrow 'a\ hyperassertion$ **where**
exists $P\ S \longleftrightarrow (\exists x.\ P\ x\ S)$

definition *forall* :: $('c \Rightarrow 'a\ hyperassertion) \Rightarrow 'a\ hyperassertion$ **where**
forall $P\ S \longleftrightarrow (\forall x.\ P\ x\ S)$

lemma *over-inter*:
  entails (*over-approx* ($P \cap Q$)) (*conj* (*over-approx* P) (*over-approx* Q))
  by (*simp add*: *conj-def entails-def over-approx-def*)

lemma *over-union*:
  entails (*disj* (*over-approx* P) (*over-approx* Q)) (*over-approx* ($P \cup Q$))
  by (*metis disj-def entailsI le-supI1 le-supI2 over-approx-def*)

lemma *under-union*:
  entails (*under-approx* ($P \cup Q$)) (*disj* (*under-approx* P) (*under-approx* Q))
  by (*simp add*: *disj-def entails-def under-approx-def*)

lemma *under-inter*:
  entails (*conj* (*under-approx* P) (*under-approx* Q)) (*under-approx* ($P \cap Q$))
  by (*simp add*: *conj-def entails-def le-infI1 under-approx-def*)

Notation 1

definition *join* :: *'a hyperassertion* $\Rightarrow$ *'a hyperassertion* $\Rightarrow$ *'a hyperassertion* **where**
  join $A\ B\ S \longleftrightarrow (\exists\, SA\ SB.\ A\ SA \wedge B\ SB \wedge S = SA \cup SB)$

definition *general-join* :: (*'b* $\Rightarrow$ *'a hyperassertion*) $\Rightarrow$ *'a hyperassertion* **where**
  general-join $f\ S \longleftrightarrow (\exists\, F.\ S = (\bigcup x.\ F\ x) \wedge (\forall\, x.\ f\ x\ (F\ x)))$

lemma *join-closed-by-union*:
  assumes *closed-by-union* Q
  shows *join* $Q\ Q = Q$
proof
  fix S
  show *join* $Q\ Q\ S \longleftrightarrow Q\ S$
    by (*metis assms closed-by-union-def join-def sup-idem*)
qed

lemma *entails-join-entails*:
  assumes *entails A1 B1*
      and *entails A2 B2*
    shows *entails* (*join A1 A2*) (*join B1 B2*)
proof (*rule entailsI*)
  fix S **assume** *join A1 A2 S*
  then obtain *S1 S2* **where** *A1 S1 A2 S2* $S = S1 \cup S2$
    by (*metis join-def*)
  then show *join B1 B2 S*
    by (*metis assms*(*1*) *assms*(*2*) *entailsE join-def*)
qed

Notation 2

definition *natural-partition* **where**
  natural-partition $I\ S \longleftrightarrow (\exists\, F.\ S = (\bigcup n.\ F\ n) \wedge (\forall\, n.\ I\ n\ (F\ n)))$

lemma *natural-partitionI*:
  assumes $S = (\bigcup n.\ F\ n)$
    and $\bigwedge n.\ I\ n\ (F\ n)$
   shows *natural-partition I S*
  using *assms*(*1*) *assms*(*2*) *natural-partition-def* **by** *blast*

lemma *natural-partitionE*:
  assumes *natural-partition I S*
  obtains F **where** $S = (\bigcup n.\ F\ n)$ $\bigwedge n.\ I\ n\ (F\ n)$
  by (*meson assms natural-partition-def*)

3.1 Rules of the Logic

Rules from figure 3

inductive *syntactic-HHT* ::
$((\prime lvar,\ \prime lval,\ \prime pvar,\ \prime pval)\ state\ hyperassertion) \Rightarrow (\prime pvar,\ \prime pval)\ stmt \Rightarrow ((\prime lvar,\ \prime lval,\ \prime pvar,\ \prime pval)\ state\ hyperassertion) \Rightarrow bool$
  $(\vdash \{\text{-}\}\ \text{-}\ \{\text{-}\}\ [51,0,0]\ 81)$ **where**
  RuleSkip: $\vdash \{P\}\ Skip\ \{P\}$
| *RuleCons*: $[\![\ entails\ P\ P'\ ;\ entails\ Q'\ Q\ ;\ \vdash \{P'\}\ C\ \{Q'\}\ ]\!] \Longrightarrow \vdash \{P\}\ C\ \{Q\}$
| *RuleSeq*: $[\![\ \vdash \{P\}\ C1\ \{R\}\ ;\ \vdash \{R\}\ C2\ \{Q\}\ ]\!] \Longrightarrow \vdash \{P\}\ (Seq\ C1\ C2)\ \{Q\}$
| *RuleIf*: $[\![\ \vdash \{P\}\ C1\ \{Q1\}\ ;\ \vdash \{P\}\ C2\ \{Q2\}\ ]\!] \Longrightarrow \vdash \{P\}\ (If\ C1\ C2)\ \{join\ Q1\ Q2\}$
| *RuleWhile*: $[\![\ \bigwedge n. \vdash \{I\ n\}\ C\ \{I\ (Suc\ n)\}\ ]\!] \Longrightarrow \vdash \{I\ 0\}\ (While\ C)\ \{natural\text{-}partition\ I\}$
| *RuleAssume*: $\vdash \{\ (\lambda S.\ P\ (Set.filter\ (b \circ snd)\ S))\ \}\ (Assume\ b)\ \{P\}$
| *RuleAssign*: $\vdash \{\ (\lambda S.\ P\ \{\ (l,\ \sigma(x := e\ \sigma))\ |l\ \sigma.\ (l,\ \sigma) \in S\ \})\ \}\ (Assign\ x\ e)\ \{P\}$
| *RuleHavoc*: $\vdash \{\ (\lambda S.\ P\ \{\ (l,\ \sigma(x := v))\ |l\ \sigma\ v.\ (l,\ \sigma) \in S\ \})\ \}\ (Havoc\ x)\ \{P\}$
| *RuleExistsSet*: $[\![\bigwedge x{::}(\prime lvar,\ \prime lval,\ \prime pvar,\ \prime pval)\ state\ set. \vdash \{P\ x\}\ C\ \{Q\ x\}]\!] \Longrightarrow \vdash \{exists\ P\}\ C\ \{exists\ Q\}$

3.2 Soundness

Definition 6: Hyper-Triples

definition *hyper-hoare-triple* $(\models \{\text{-}\}\ \text{-}\ \{\text{-}\}\ [51,0,0]\ 81)$ **where**
  $\models \{P\}\ C\ \{Q\} \longleftrightarrow (\forall S.\ P\ S \longrightarrow Q\ (sem\ C\ S))$

lemma *hyper-hoare-tripleI*:
  assumes $\bigwedge S.\ P\ S \Longrightarrow Q\ (sem\ C\ S)$
  shows $\models \{P\}\ C\ \{Q\}$
  by (*simp add*: *assms hyper-hoare-triple-def*)

lemma *hyper-hoare-tripleE*:
  assumes $\models \{P\}\ C\ \{Q\}$
    and $P\ S$
   shows $Q\ (sem\ C\ S)$
  using *assms*(*1*) *assms*(*2*) *hyper-hoare-triple-def*

by *metis*

lemma *consequence-rule*:
assumes *entails P P′*
and *entails Q′ Q*
and $\models$ {*P′*} *C* {*Q′*}
shows $\models$ {*P*} *C* {*Q*}
by (*metis* (*no-types*, *opaque-lifting*) *assms*(*1*) *assms*(*2*) *assms*(*3*) *entails-def hyper-hoare-triple-def*)

lemma *skip-rule*:
$\models$ {*P*} *Skip* {*P*}
by (*simp add*: *hyper-hoare-triple-def sem-skip*)

lemma *assume-rule*:
$\models$ { (λ*S*. *P* (*Set.filter* (*b* $\circ$ *snd*) *S*)) } (*Assume b*) {*P*}
proof (*rule hyper-hoare-tripleI*)
fix *S* **assume** *P* (*Set.filter* (*b* $\circ$ *snd*) *S*)
then show *P* (*sem* (*Assume b*) *S*)
by (*simp add*: *assume-sem*)
qed

lemma *seq-rule*:
assumes $\models$ {*P*} *C1* {*R*}
and $\models$ {*R*} *C2* {*Q*}
shows $\models$ {*P*} *Seq C1 C2* {*Q*}
using *assms*(*1*) *assms*(*2*) *hyper-hoare-triple-def sem-seq*
by *metis*

lemma *if-rule*:
assumes $\models$ {*P*} *C1* {*Q1*}
and $\models$ {*P*} *C2* {*Q2*}
shows $\models$ {*P*} *If C1 C2* {*join Q1 Q2*}
by (*metis* (*full-types*) *assms*(*1*) *assms*(*2*) *hyper-hoare-triple-def join-def sem-if*)

lemma *sem-assign*:
sem (*Assign x e*) *S* = {(*l*, σ(*x* := *e* σ)) |*l* σ. (*l*, σ) $\in$ *S*} (**is** *?A* = *?B*)
proof
show *?A* $\subseteq$ *?B*
proof (*rule subsetPairI*)
fix *l* σ'
assume (*l*, σ') $\in$ *sem* (*Assign x e*) *S*
then obtain σ **where** (*l*, σ) $\in$ *S single-sem* (*Assign x e*) σ σ'
by (*metis fst-eqD in-sem snd-conv*)
then show (*l*, σ') $\in$ {(*l*, σ(*x* := *e* σ)) |*l* σ. (*l*, σ) $\in$ *S*}
by *blast*
qed
show *?B* $\subseteq$ *?A*
proof (*rule subsetPairI*)

fix l σ'
assume $(l, \sigma') \in ?B$
then obtain σ **where** $\sigma' = \sigma(x := e\ \sigma)$ $(l, \sigma) \in S$
by *blast*
then show $(l, \sigma') \in ?A$
by (*metis SemAssign fst-eqD in-sem snd-conv*)
qed
qed

lemma *assign-rule*:
$\models \{ (\lambda S.\ P \{ (l, \sigma(x := e\ \sigma)) \mid l\ \sigma.\ (l, \sigma) \in S \}) \}\ (Assign\ x\ e)\ \{P\}$
proof (*rule hyper-hoare-tripleI*)
fix S **assume** $P \{(l, \sigma(x := e\ \sigma)) \mid l\ \sigma.\ (l, \sigma) \in S\}$
then show P (*sem* (*Assign x e*) S) **using** *sem-assign*
by *metis*
qed

lemma *sem-havoc*:
sem (*Havoc x*) $S = \{(l, \sigma(x := v)) \mid l\ \sigma\ v.\ (l, \sigma) \in S\}$ (**is** $?A = ?B$)
proof
show $?A \subseteq ?B$
proof (*rule subsetPairI*)
fix l σ'
assume $(l, \sigma') \in$ *sem* (*Havoc x*) S
then obtain σ **where** $(l, \sigma) \in S$ *single-sem* (*Havoc x*) σ σ'
by (*metis fst-eqD in-sem snd-conv*)
then show $(l, \sigma') \in \{(l, \sigma(x := v)) \mid l\ \sigma\ v.\ (l, \sigma) \in S\}$
by *blast*
qed
show $?B \subseteq ?A$
proof (*rule subsetPairI*)
fix l σ'
assume $(l, \sigma') \in ?B$
then obtain σ v **where** $\sigma' = \sigma(x := v)$ $(l, \sigma) \in S$
by *blast*
then show $(l, \sigma') \in ?A$
by (*metis SemHavoc fst-eqD in-sem snd-conv*)
qed
qed

lemma *havoc-rule*:
$\models \{ (\lambda S.\ P \{ (l, \sigma(x := v)) \mid l\ \sigma\ v.\ (l, \sigma) \in S \}) \}\ (Havoc\ x)\ \{P\}$
proof (*rule hyper-hoare-tripleI*)
fix S **assume** $P \{ (l, \sigma(x := v)) \mid l\ \sigma\ v.\ (l, \sigma) \in S \}$
then show P (*sem* (*Havoc x*) S) **using** *sem-havoc* **by** *metis*
qed

Loops

lemma *indexed-invariant-then-power*:

```
  assumes ⋀n. hyper-hoare-triple (I n) C (I (Suc n))
      and I 0 S
  shows I n (iterate-sem n C S)
  using assms
proof (induct n arbitrary: S)
next
  case (Suc n)
  then have I n (iterate-sem n C S)
    by blast
  then have I (Suc n) (sem C (iterate-sem n C S))
    using Suc.prems(1) hyper-hoare-tripleE by blast
  then show ?case
    by (simp add: Suc.hyps Suc.prems(1))
qed (auto)

lemma while-rule:
  assumes ⋀n. hyper-hoare-triple (I n) C (I (Suc n))
  shows hyper-hoare-triple (I 0) (While C) (natural-partition I)
proof (rule hyper-hoare-tripleI)
  fix S assume asm0: I 0 S
  show natural-partition I (sem (While C) S)
  proof (rule natural-partitionI)
    show sem (While C) S = ⋃ (range (λn. iterate-sem n C S))
      by (simp add: sem-while)
    fix n show I n (iterate-sem n C S)
      by (simp add: asm0 assms indexed-invariant-then-power)
  qed
qed
```

Additional rules

```
lemma empty-pre:
  hyper-hoare-triple (λ-. False) C QQ
  by (simp add: hyper-hoare-triple-def)

lemma full-post:
  hyper-hoare-triple P C (λ-. True)
  by (simp add: hyper-hoare-triple-def)


lemma rule-join:
  assumes ⊨ {P} C {Q}
      and hyper-hoare-triple P' C Q'
    shows hyper-hoare-triple (join P P') C (join Q Q')
proof (rule hyper-hoare-tripleI)
  fix S assume asm0: join P P' S
  then obtain S1 S2 where S = S1 ∪ S2 P S1 P' S2
    by (metis join-def)
  then have sem C S = sem C S1 ∪ sem C S2
    using sem-union by auto
```

then show *join* Q Q' (*sem* C S)
by (*metis* ‹P *S1*› ‹P' *S2*› *assms*(*1*) *assms*(*2*) *hyper-hoare-tripleE join-def*)
qed

lemma *rule-general-join*:
assumes $\bigwedge x.$ $\models$ {P x} C {Q x}
shows *hyper-hoare-triple* (*general-join* P) C (*general-join* Q)
proof (*rule hyper-hoare-tripleI*)
fix S **assume** *general-join* P S
then obtain F **where** *asm0*: $S = (\bigcup x.\ F\ x)$ $\bigwedge x.\ P\ x\ (F\ x)$
by (*meson general-join-def*)
have *sem* C S = ($\bigcup x.$ *sem* C (F x))
by (*simp add*: *asm0*(*1*) *sem-split-general*)
moreover have $\bigwedge x.$ Q x (*sem* C (F x))
using *asm0*(*2*) *assms hyper-hoare-tripleE* **by** *blast*
ultimately show *general-join* Q (*sem* C S)
by (*metis general-join-def*)
qed

lemma *rule-conj*:
assumes $\models$ {P} C {Q}
and *hyper-hoare-triple* P' C Q'
shows *hyper-hoare-triple* (*conj* P P') C (*conj* Q Q')
proof (*rule hyper-hoare-tripleI*)
fix S **assume** *Logic.conj* P P' S
then show *Logic.conj* Q Q' (*sem* C S)
by (*metis assms*(*1*) *assms*(*2*) *conj-def hyper-hoare-tripleE*)
qed

Generalization

lemma *rule-forall*:
assumes $\bigwedge x.$ $\models$ {P x} C {Q x}
shows *hyper-hoare-triple* (*forall* P) C (*forall* Q)
by (*metis assms forall-def hyper-hoare-triple-def*)

lemma *rule-disj*:
assumes $\models$ {P} C {Q}
and $\models$ {P'} C {Q'}
shows *hyper-hoare-triple* (*disj* P P') C (*disj* Q Q')
by (*metis assms*(*1*) *assms*(*2*) *disj-def hyper-hoare-triple-def*)

Generalization

lemma *rule-exists*:
assumes $\bigwedge x.$ $\models$ {P x} C {Q x}
shows $\models$ {*exists* P} C {*exists* Q}
by (*metis assms exists-def hyper-hoare-triple-def*)

corollary *variant-if-rule*:
assumes *hyper-hoare-triple* P *C1* Q

and *hyper-hoare-triple P C2 Q*
and *closed-by-union Q*
shows *hyper-hoare-triple P* (*If C1 C2*) *Q*
by (*metis assms*(*1*) *assms*(*2*) *assms*(*3*) *if-rule join-closed-by-union*)

Simplifying the rule

definition *stable-by-infinite-union* :: *'a hyperassertion* ⇒ *bool* **where**
stable-by-infinite-union I ⟷ (∀ *F*. (∀ *S* ∈ *F*. *I S*) ⟶ *I* (⋃ *S* ∈ *F*. *S*))

lemma *stable-by-infinite-unionE*:
assumes *stable-by-infinite-union I*
and ⋀*S*. *S* ∈ *F* ⟹ *I S*
shows *I* (⋃ *S* ∈ *F*. *S*)
using *assms*(*1*) *assms*(*2*) *stable-by-infinite-union-def* **by** *blast*

lemma *stable-by-union-and-constant-then-I*:
assumes ⋀*n*. *I n* = *I'*
and *stable-by-infinite-union I'*
shows *natural-partition I* = *I'*
proof (*rule ext*)
fix *S* **show** *natural-partition I S* = *I' S*
proof
show *I' S* ⟹ *natural-partition I S*
proof −
assume *I' S*
show *natural-partition I S*
proof (*rule natural-partitionI*)
show *S* = ⋃ (*range* (λ*n*. *S*))
by *simp*
fix *n* **show** *I n S*
by (*simp add*: ‹*I' S*› *assms*(*1*))
qed
qed
assume *asm0*: *natural-partition I S*
then obtain *F* **where** *S* = (⋃ *n*. *F n*) ⋀*n*. *I n* (*F n*)
using *natural-partitionE* **by** *blast*
let *?F* = {*F n* |*n*. *True*}
have *I'* (⋃ *S*∈*?F*. *S*)
using *assms*(*2*)
proof (*rule stable-by-infinite-unionE*[*of I'*])
fix *S* **assume** *S* ∈ {*F n* |*n*. *True*}
then show *I' S*
using ‹⋀*n*. *I n* (*F n*)› *assms*(*1*) **by** *force*
qed
moreover have (⋃ *S*∈*?F*. *S*) = *S*
using ‹*S* = (⋃ *n*. *F n*)› **by** *auto*
ultimately show *I' S* **by** *blast*
qed
qed

corollary *simpler-rule-while*:
  assumes *hyper-hoare-triple I C I*
    and *stable-by-infinite-union I*
  shows *hyper-hoare-triple I* (*While C*) *I*
proof −
  let *?I* = $\lambda n.$ *I*
  have *hyper-hoare-triple* (*?I 0*) (*While C*) (*natural-partition ?I*)
    using *while-rule*[*of ?I C*]
    by (*simp add*: *assms*(*1*) *assms*(*2*) *stable-by-union-and-constant-then-I*)
  then show *?thesis*
    by (*simp add*: *assms*(*2*) *stable-by-union-and-constant-then-I*)
qed

Theorem 1

theorem *soundness*:
  assumes $\vdash \{A\}\ C\ \{B\}$
    shows $\models \{A\}\ C\ \{B\}$
  using *assms*
proof (*induct rule*: *syntactic-HHT.induct*)
  case (*RuleSkip P*)
  then show *?case*
    using *skip-rule* **by** *auto*
next
  case (*RuleCons P P′ Q′ Q C*)
  then show *?case*
    using *consequence-rule* **by** *blast*
next
  case (*RuleExistsSet P C Q*)
  then show *?case*
    using *rule-exists* **by** *blast*
next
  case (*RuleSeq P C1 R C2 Q*)
  then show *?case*
    using *seq-rule* **by** *meson*
next
  case (*RuleIf P C1 Q1 C2 Q2*)
  then show *?case*
    using *if-rule* **by** *blast*
next
  case (*RuleAssume P b*)
  then show *?case*
    by (*simp add*: *assume-rule*)
next
  case (*RuleWhile I C*)
  then show *?case*
    using *while-rule* **by** *blast*
next
  case (*RuleAssign x e*)

then show *?case*
by (*simp add*: *assign-rule*)
next
case (*RuleHavoc x*)
then show *?case*
using *havoc-rule* **by** *fastforce*
qed

3.3 Completeness

definition *complete*
where
$complete\ P\ C\ Q \longleftrightarrow (\models \{P\}\ C\ \{Q\} \longrightarrow \vdash \{P\}\ C\ \{Q\})$

lemma *completeI*:
assumes $\models \{P\}\ C\ \{Q\} \Longrightarrow \vdash \{P\}\ C\ \{Q\}$
shows *complete P C Q*
by (*simp add*: *assms complete-def*)

lemma *completeE*:
assumes *complete P C Q*
and $\models \{P\}\ C\ \{Q\}$
shows $\vdash \{P\}\ C\ \{Q\}$
using *assms complete-def* **by** *auto*

lemma *complete-if-aux*:
assumes *hyper-hoare-triple A* (*If C1 C2*) *B*
shows *entails* $(\lambda S'.\ \exists S.\ A\ S \wedge S' = sem\ C1\ S \cup sem\ C2\ S)\ B$
proof (*rule entailsI*)
fix S' **assume** $\exists S.\ A\ S \wedge S' = sem\ C1\ S \cup sem\ C2\ S$
then show $B\ S'$
by (*metis assms hyper-hoare-tripleE sem-if*)
qed

lemma *complete-if*:
fixes $P\ Q$ :: (*'lvar*, *'lval*, *'pvar*, *'pval*) *state hyperassertion*
assumes $\bigwedge P1\ Q1$ :: (*'lvar*, *'lval*, *'pvar*, *'pval*) *state hyperassertion*. *complete P1 C1 Q1*
and $\bigwedge P2\ Q2$ :: (*'lvar*, *'lval*, *'pvar*, *'pval*) *state hyperassertion*. *complete P2 C2 Q2*
shows *complete P* (*If C1 C2*) *Q*
proof (*rule completeI*)
assume *asm0*: $\models \{P\}$ *If C1 C2* $\{Q\}$

show $\vdash \{P\}$ *stmt.If C1 C2* $\{Q\}$
proof (*rule RuleCons*)
show $\vdash \{exists\ (\lambda V\ S.\ P\ S \wedge S = V)\}$ *stmt.If C1 C2* $\{exists\ (\lambda V.\ join\ (\lambda S.\ S = sem\ C1\ V \wedge P\ V)\ (\lambda S.\ S = sem\ C2\ V))\}$
proof (*rule RuleExistsSet*)

fix V
show $\vdash \{(\lambda S.\ P\ S \wedge S = V)\}\ \mathit{stmt.If}\ C1\ C2\ \{\mathit{join}\ (\lambda S.\ S = \mathit{sem}\ C1\ V \wedge P\ V)\ (\lambda S.\ S = \mathit{sem}\ C2\ V)\}$
proof $(\mathit{rule}\ \mathit{RuleIf})$
show $\vdash \{(\lambda S.\ P\ S \wedge S = V)\}\ C1\ \{\lambda S.\ S = \mathit{sem}\ C1\ V \wedge P\ V\}$
by $(\mathit{simp}\ \mathit{add}\colon \mathit{assms}(1)\ \mathit{completeE}\ \mathit{hyper\text{-}hoare\text{-}triple\text{-}def})$
show $\vdash \{(\lambda S.\ P\ S \wedge S = V)\}\ C2\ \{\lambda S.\ S = \mathit{sem}\ C2\ V\}$
by $(\mathit{simp}\ \mathit{add}\colon \mathit{assms}(2)\ \mathit{completeE}\ \mathit{hyper\text{-}hoare\text{-}triple\text{-}def})$
qed
qed
show $\mathit{entails}\ P\ (\mathit{exists}\ (\lambda V\ S.\ P\ S \wedge S = V))$
by $(\mathit{simp}\ \mathit{add}\colon \mathit{entailsI}\ \mathit{exists\text{-}def})$
show $\mathit{entails}\ (\mathit{exists}\ (\lambda V.\ \mathit{join}\ (\lambda S.\ S = \mathit{sem}\ C1\ V \wedge P\ V)\ (\lambda S.\ S = \mathit{sem}\ C2\ V)))\ Q$
proof $(\mathit{rule}\ \mathit{entailsI})$
fix S **assume** $\mathit{exists}\ (\lambda V.\ \mathit{join}\ (\lambda S.\ S = \mathit{sem}\ C1\ V \wedge P\ V)\ (\lambda S.\ S = \mathit{sem}\ C2\ V))\ S$
then obtain V **where** $\mathit{join}\ (\lambda S.\ S = \mathit{sem}\ C1\ V \wedge P\ V)\ (\lambda S.\ S = \mathit{sem}\ C2\ V)\ S$
by $(\mathit{meson}\ \mathit{exists\text{-}def})$
then obtain $S1\ S2$ **where** $S = S1 \cup S2\ S1 = \mathit{sem}\ C1\ V \wedge P\ V\ S2 = \mathit{sem}\ C2\ V$
by $(\mathit{simp}\ \mathit{add}\colon \mathit{join\text{-}def})$
then show $Q\ S$
by $(\mathit{metis}\ \mathit{asm0}\ \mathit{hyper\text{-}hoare\text{-}tripleE}\ \mathit{sem\text{-}if})$
qed
qed
qed

lemma *complete-seq-aux*:
assumes $\mathit{hyper\text{-}hoare\text{-}triple}\ A\ (\mathit{Seq}\ C1\ C2)\ B$
shows $\exists R.\ \mathit{hyper\text{-}hoare\text{-}triple}\ A\ C1\ R \wedge \mathit{hyper\text{-}hoare\text{-}triple}\ R\ C2\ B$
proof $-$
let $?R = \lambda S.\ \exists S'.\ A\ S' \wedge S = \mathit{sem}\ C1\ S'$
have $\mathit{hyper\text{-}hoare\text{-}triple}\ A\ C1\ ?R$
using *hyper-hoare-triple-def* **by** *blast*
moreover have $\mathit{hyper\text{-}hoare\text{-}triple}\ ?R\ C2\ B$
proof $(\mathit{rule}\ \mathit{hyper\text{-}hoare\text{-}tripleI})$
fix S **assume** $\exists S'.\ A\ S' \wedge S = \mathit{sem}\ C1\ S'$
then obtain S' **where** $\mathit{asm0}\colon A\ S'\ S = \mathit{sem}\ C1\ S'$
by *blast*
then show $B\ (\mathit{sem}\ C2\ S)$
by $(\mathit{metis}\ \mathit{assms}\ \mathit{hyper\text{-}hoare\text{-}tripleE}\ \mathit{sem\text{-}seq})$
qed
ultimately show *?thesis* **by** *blast*
qed

lemma *complete-assume*:

complete P (*Assume b*) *Q*
proof (*rule completeI*)
  assume *asm0*: $\models$ {*P*} *Assume b* {*Q*}
  show $\vdash$ {*P*} *Assume b* {*Q*}
  proof (*rule RuleCons*)
    show $\vdash$ { ($\lambda S.$ *Q* (*Set.filter* ($b \circ snd$) *S*)) } (*Assume b*) {*Q*}
      by (*simp add*: *RuleAssume*)
    show *entails P* ($\lambda S.$ *Q* (*Set.filter* ($b \circ snd$) *S*))
    by (*metis* (*mono-tags*, *lifting*) *asm0 assume-sem entails-def hyper-hoare-tripleE*)
    show *entails Q Q*
      by (*simp add*: *entailsI*)
  qed
qed

lemma *complete-skip*:
  complete P Skip Q
  using *completeI RuleSkip*
  by (*metis* (*mono-tags*, *lifting*) *entails-def hyper-hoare-triple-def sem-skip Rule-Cons*)

lemma *complete-assign*:
  complete P (*Assign x e*) *Q*
proof (*rule completeI*)
  assume *asm0*: $\models$ {*P*} *Assign x e* {*Q*}
  show $\vdash$ {*P*} *Assign x e* {*Q*}
  proof (*rule RuleCons*)
    show $\vdash$ {($\lambda S.$ *Q* $\{(l, \sigma(x := e\ \sigma)) \mid l\ \sigma.\ (l, \sigma) \in S\}$)} *Assign x e* {*Q*}
      by (*simp add*: *RuleAssign*)
    show *entails P* ($\lambda S.$ *Q* $\{(l, \sigma(x := e\ \sigma)) \mid l\ \sigma.\ (l, \sigma) \in S\}$)
    proof (*rule entailsI*)
      fix *S* **assume** *P S*
      then show *Q* $\{(l, \sigma(x := e\ \sigma)) \mid l\ \sigma.\ (l, \sigma) \in S\}$
        by (*metis asm0 hyper-hoare-triple-def sem-assign*)
    qed
    show *entails Q Q*
      by (*simp add*: *entailsI*)
  qed
qed

lemma *complete-havoc*:
  complete P (*Havoc x*) *Q*
proof (*rule completeI*)
  assume *asm0*: $\models$ {*P*} *Havoc x* {*Q*}
  show $\vdash$ {*P*} *Havoc x* {*Q*}
  proof (*rule RuleCons*)
    show $\vdash$ { ($\lambda S.$ *Q* $\{ (l, \sigma(x := v)) \mid l\ \sigma\ v.\ (l, \sigma) \in S \}$) } (*Havoc x*) {*Q*}
      using *RuleHavoc* **by** *fast*
    show *entails P* ($\lambda S.$ *Q* $\{(l, \sigma(x := v)) \mid l\ \sigma\ v.\ (l, \sigma) \in S\}$)
    proof (*rule entailsI*)

```
    fix S assume P S
    then show Q {(l, σ(x := v)) |l σ v. (l, σ) ∈ S}
      by (metis asm0 hyper-hoare-triple-def sem-havoc)
  qed
  show entails Q Q
    by (simp add: entailsI)
 qed
qed

lemma complete-seq:
 assumes ⋀R. complete P C1 R
    and ⋀R. complete R C2 Q
  shows complete P (Seq C1 C2) Q
 by (meson RuleSeq assms(1) assms(2) completeE completeI complete-seq-aux)



fun construct-inv
 where
 construct-inv P C 0 = P
| construct-inv P C (Suc n) = (λS. (∃S'. S = sem C S' ∧ construct-inv P C n
S'))

lemma iterate-sem-ind:
 assumes construct-inv P C n S'
 shows ∃S. P S ∧ S' = iterate-sem n C S
 using assms
by (induct n arbitrary: S') (auto)


lemma complete-while-aux:
 assumes hyper-hoare-triple (λS. P S ∧ S = V) (While C) Q
 shows entails (natural-partition (construct-inv (λS. P S ∧ S = V) C)) Q
proof (rule entailsI)
 fix S assume natural-partition (construct-inv (λS. P S ∧ S = V) C) S

 then obtain F where asm0: S = (⋃n. F n) ⋀n. construct-inv (λS. P S ∧ S
= V) C n (F n)
   using natural-partitionE by blast
 then have P (F 0) ∧ F 0 = V
   by (metis (mono-tags, lifting) construct-inv.simps(1))
 then have Q (⋃n. iterate-sem n C (F 0))
   using assms hyper-hoare-triple-def[of λS. P S ∧ S = V While C Q] sem-while
   by metis
 moreover have ⋀n. F n = iterate-sem n C V
 proof -
   fix n
   obtain S' where P S' ∧ S' = V F n = iterate-sem n C S'
     using asm0(2) iterate-sem-ind by blast
```

then show $F\ n = \mathit{iterate\text{-}sem}\ n\ C\ V$
by *simp*
qed
ultimately show $Q\ S$
using *asm0*(*1*) **by** *auto*
qed

lemma *complete-while*:
fixes $P\ Q$ :: ($'lvar$, $'lval$, $'pvar$, $'pval$) *state hyperassertion*
assumes $\bigwedge P'\ Q'$ :: ($'lvar$, $'lval$, $'pvar$, $'pval$) *state hyperassertion. complete* P' $C\ Q'$
shows *complete* P (*While* C) Q
proof (*rule completeI*)
assume *asm0*: *hyper-hoare-triple* P (*While* C) Q

let $?I = \lambda V.\ \mathit{construct\text{-}inv}\ (\lambda S.\ P\ S \wedge S = V)\ C$

have *r*: $\bigwedge V.$ *syntactic-HHT* ($?I\ V\ 0$) (*While* C) (*natural-partition* ($?I\ V$))
proof (*rule RuleWhile*)
fix $V\ n$ **show** *syntactic-HHT* (*construct-inv* ($\lambda S.\ P\ S \wedge S = V$) $C\ n$) C (*construct-inv* ($\lambda S.\ P\ S \wedge S = V$) C (*Suc* n))
by (*meson assms completeE construct-inv.simps*(*2*) *hyper-hoare-tripleI*)
qed

show *syntactic-HHT* P (*While* C) Q
proof (*rule RuleCons*)
show *syntactic-HHT* (*exists* ($\lambda V.\ ?I\ V\ 0$)) (*While* C) (*exists* ($\lambda V.$ ((*natural-partition* ($?I\ V$)))))
using *r* **by** (*rule RuleExistsSet*)
show *entails* P (*exists* ($\lambda V.$ *construct-inv* ($\lambda S.\ P\ S \wedge S = V$) $C\ 0$))
by (*simp add*: *entailsI exists-def*)
show *entails* (*exists* ($\lambda V.$ *natural-partition* (*construct-inv* ($\lambda S.\ P\ S \wedge S = V$) C))) Q
proof (*rule entailsI*)
fix S' **assume** *exists* ($\lambda V.$ *natural-partition* (*construct-inv* ($\lambda S.\ P\ S \wedge S = V$) C)) S'
then obtain V **where** *natural-partition* (*construct-inv* ($\lambda S.\ P\ S \wedge S = V$) C) S'
by (*meson exists-def*)
moreover have *entails* (*natural-partition* (*construct-inv* ($\lambda S.\ P\ S \wedge S = V$) C)) Q
proof (*rule complete-while-aux*)
show *hyper-hoare-triple* ($\lambda S.\ P\ S \wedge S = V$) (*While* C) Q
using *asm0 hyper-hoare-triple-def*[*of* $\lambda S.\ P\ S \wedge S = V$]
hyper-hoare-triple-def[*of* P *While* $C\ Q$] **by** *auto*
qed
ultimately show $Q\ S'$
by (*simp add*: *entails-def*)
qed

qed
qed

Theorem 2

theorem *completeness*:
fixes P Q :: (*'lvar*, *'lval*, *'pvar*, *'pval*) *state hyperassertion*
assumes $\models \{P\}\ C\ \{Q\}$
shows $\vdash \{P\}\ C\ \{Q\}$
using *assms*
proof (*induct C arbitrary*: P Q)
case (*Assign x1 x2*)
then show *?case*
using *completeE complete-assign* **by** *fast*
next
case (*Seq C1 C2*)
then show *?case*
using *complete-def complete-seq* **by** *meson*
next
case (*If C1 C2*)
then show *?case*
using *complete-def complete-if* **by** *meson*
next
case *Skip*
then show *?case*
using *complete-def complete-skip* **by** *meson*
next
case (*Havoc x*)
then show *?case*
by (*simp add*: *completeE complete-havoc*)
next
case (*Assume b*)
then show *?case*
by (*simp add*: *completeE complete-assume*)
next
case (*While C*)
then show *?case*
using *complete-def complete-while* **by** *blast*
qed

3.4 Disproving Hyper-Triples

definition *sat* **where** *sat* $P \longleftrightarrow (\exists S.\ P\ S)$

Theorem 4

theorem *disproving-triple*:
$\neg \models \{P\}\ C\ \{Q\} \longleftrightarrow (\exists P'.\ sat\ P' \land entails\ P'\ P \land \models \{P'\}\ C\ \{\lambda S.\ \neg\ Q\ S\})$ (**is** *?A* $\longleftrightarrow$ *?B*)
proof
assume $\neg \models \{P\}\ C\ \{Q\}$

then obtain S **where** *asm0*: $P\ S\ \neg\ Q\ (sem\ C\ S)$
  using *hyper-hoare-triple-def* **by** *blast*
let $?P = \lambda S'.\ S = S'$
have *entails* $?P\ P$
  by (*simp add*: *asm0*(*1*) *entails-def*)
moreover have $\models \{?P\}\ C\ \{\lambda S.\ \neg\ Q\ S\}$
  by (*simp add*: *asm0*(*2*) *hyper-hoare-triple-def*)
moreover have *sat* $?P$
  by (*simp add*: *sat-def*)
ultimately show $?B$ **by** *blast*
next
assume $\exists P'.\ sat\ P' \wedge entails\ P'\ P \wedge \models \{P'\}\ C\ \{\lambda S.\ \neg\ Q\ S\}$
then obtain P' **where** *asm0*: $sat\ P'\ entails\ P'\ P \models \{P'\}\ C\ \{\lambda S.\ \neg\ Q\ S\}$
  by *blast*
then obtain S **where** $P'\ S$
  by (*meson sat-def*)
then show $?A$
  using *asm0*(*2*) *asm0*(*3*) *entailsE hyper-hoare-tripleE*
  by (*metis* (*no-types*, *lifting*))
qed

definition *differ-only-by* **where**
  $differ\text{-}only\text{-}by\ a\ b\ x \longleftrightarrow (\forall\, y.\ y \neq x \longrightarrow a\ y = b\ y)$

lemma *differ-only-byI*:
  assumes $\bigwedge y.\ y \neq x \Longrightarrow a\ y = b\ y$
  shows *differ-only-by* $a\ b\ x$
  by (*simp add*: *assms differ-only-by-def*)

lemma *diff-by-update*:
  $differ\text{-}only\text{-}by\ (a(x := v))\ a\ x$
  by (*simp add*: *differ-only-by-def*)

lemma *diff-by-comm*:
  $differ\text{-}only\text{-}by\ a\ b\ x \longleftrightarrow differ\text{-}only\text{-}by\ b\ a\ x$
  by (*metis* (*mono-tags*, *lifting*) *differ-only-by-def*)

lemma *diff-by-trans*:
  assumes *differ-only-by* $a\ b\ x$
    and *differ-only-by* $b\ c\ x$
  shows *differ-only-by* $a\ c\ x$
  by (*metis assms*(*1*) *assms*(*2*) *differ-only-by-def*)

definition *not-free-var-of* **where**
  $not\text{-}free\text{-}var\text{-}of\ P\ x \longleftrightarrow (\forall\, states\ states'.$
  $(\forall\, i.\ differ\text{-}only\text{-}by\ (fst\ (states\ i))\ (fst\ (states'\ i))\ x \wedge snd\ (states\ i) = snd\ (states'\ i))$
  $\longrightarrow (states \in P \longleftrightarrow states' \in P))$

lemma *not-free-var-ofE*:
assumes *not-free-var-of P x*
and $\bigwedge i$. *differ-only-by* (*fst* (*states i*)) (*fst* (*states' i*)) *x*
and $\bigwedge i$. *snd* (*states i*) = *snd* (*states' i*)
and *states* $\in$ *P*
shows *states'* $\in$ *P*
using *not-free-var-of-def*[*of P x*] *assms* **by** *blast*

3.5 Synchronized Rule for Branching

definition *combine* **where**
combine from-nat x P1 P2 S $\longleftrightarrow$ *P1* (*Set.filter* ($\lambda\varphi$. *fst* φ *x* = *from-nat 1*) *S*) $\wedge$ *P2* (*Set.filter* ($\lambda\varphi$. *fst* φ *x* = *from-nat 2*) *S*)

lemma *combineI*:
assumes *P1* (*Set.filter* ($\lambda\varphi$. *fst* φ *x* = *from-nat 1*) *S*) $\wedge$ *P2* (*Set.filter* ($\lambda\varphi$. *fst* φ *x* = *from-nat 2*) *S*)
shows *combine from-nat x P1 P2 S*
by (*simp add*: *assms combine-def*)

definition *modify-lvar-to* **where**
modify-lvar-to x v φ = ((*fst* φ)(*x* := *v*), *snd* φ)

lemma *logical-var-in-sem-same*:
assumes $\bigwedge\varphi$. $\varphi \in S \Longrightarrow$ *fst* φ *x* = *a*
and $\varphi' \in$ *sem C S*
shows *fst* φ' *x* = *a*
by (*metis assms*(*1*) *assms*(*2*) *fst-conv in-sem*)

lemma *recover-after-sem*:
assumes $a \neq b$
and $\bigwedge\varphi$. $\varphi \in S1 \Longrightarrow$ *fst* φ *x* = *a*
and $\bigwedge\varphi$. $\varphi \in S2 \Longrightarrow$ *fst* φ *x* = *b*
shows *sem C S1* = *Set.filter* ($\lambda\varphi$. *fst* φ *x* = *a*) (*sem C* (*S1* $\cup$ *S2*)) (**is** *?A* = *?B*)
proof
have *r*: *sem C* (*S1* $\cup$ *S2*) = *sem C S1* $\cup$ *sem C S2*
by (*simp add*: *sem-union*)
moreover have *r1*: $\bigwedge\varphi'$. $\varphi' \in$ *sem C S1* $\Longrightarrow$ *fst* φ' *x* = *a*
by (*metis assms*(*2*) *fst-conv in-sem*)
moreover have *r2*: $\bigwedge\varphi'$. $\varphi' \in$ *sem C S2* $\Longrightarrow$ *fst* φ' *x* = *b*
by (*metis assms*(*3*) *fst-conv in-sem*)
show *?B* $\subseteq$ *?A*
proof (*rule subsetPairI*)
fix *l* σ
assume (*l*, σ) $\in$ *Set.filter* ($\lambda\varphi$. *fst* φ *x* = *a*) (*sem C* (*S1* $\cup$ *S2*))
then show (*l*, σ) $\in$ *sem C S1*

    using *assms*(*1*) *r r2* **by** *auto*
  qed
  show *?A* ⊆ *?B*
    by (*simp add*: *r r1 subsetI*)
qed

lemma *injective-then-ok*:
  assumes $a \neq b$
      and $S1' = (\textit{modify-lvar-to } x\ a)\ `\ S1$
      and $S2' = (\textit{modify-lvar-to } x\ b)\ `\ S2$
    shows *Set.filter* (λφ. *fst* φ *x* = *a*) (*S1′* ∪ *S2′*) = *S1′* (**is** *?A* = *?B*)
proof
  show *?B* ⊆ *?A*
  proof (*rule subsetI*)
    fix *y* **assume** *y* ∈ *S1′*
    then have *fst y x* = *a* **using** *modify-lvar-to-def assms*(*2*)
      by (*metis* (*mono-tags*, *lifting*) *fst-conv fun-upd-same image-iff*)
    then show *y* ∈ *Set.filter* (λφ. *fst* φ *x* = *a*) (*S1′* ∪ *S2′*)
      by (*simp add*: ‹*y* ∈ *S1′*›)
  qed
  show *?A* ⊆ *?B*
  proof
    fix *y* **assume** *y* ∈ *?A*
    then have *y* ∉ *S2′*
        by (*metis* (*mono-tags*, *lifting*) *assms*(*1*) *assms*(*3*) *fun-upd-same image-iff member-filter modify-lvar-to-def prod.sel*(*1*))
    then show *y* ∈ *?B*
      using ‹*y* ∈ *Set.filter* (λφ. *fst* φ *x* = *a*) (*S1′* ∪ *S2′*)› **by** *auto*
  qed
qed

definition *not-free-var-hyper* **where**
  not-free-var-hyper x P ⟷ (∀ *S v*. *P S* ⟷ *P* ((*modify-lvar-to x v*) ‘ *S*))

definition *injective* **where**
  injective f ⟷ (∀ *a b*. *a* ≠ *b* ⟶ *f a* ≠ *f b*)

lemma *sem-of-modify-lvar*:
  sem C ((*modify-lvar-to r v*) ‘ *S*) = (*modify-lvar-to r v*) ‘ (*sem C S*) (**is** *?A* = *?B*)
proof
  show *?A* ⊆ *?B*
  proof (*rule subsetI*)
    fix *y* **assume** *asm0*: *y* ∈ *?A*
    then obtain *x* **where** *x* ∈ (*modify-lvar-to r v*) ‘ *S single-sem C* (*snd x*) (*snd y*) *fst x* = *fst y*
      by (*metis fst-conv in-sem snd-conv*)
    then obtain *xx* **where** *xx* ∈ *S x* = *modify-lvar-to r v xx*
      by *blast*
    then have (*fst xx*, *snd y*) ∈ *sem C S*

by (*metis* ‹⟨*C*, *snd x*⟩ → *snd y*› *fst-conv in-sem modify-lvar-to-def prod.collapse snd-conv*)
then show $y \in$ *?B*
by (*metis* ‹*fst x* = *fst y*› ‹*x* = *modify-lvar-to r v xx*› *fst-eqD modify-lvar-to-def prod.exhaust-sel rev-image-eqI snd-eqD*)
qed
show *?B* $\subseteq$ *?A*
proof (*rule subsetI*)
fix *y* **assume** $y \in$ *modify-lvar-to r v* ‘ *sem C S*
then obtain *yy* **where** *y* = *modify-lvar-to r v yy yy* $\in$ *sem C S*
by *blast*
then obtain *x* **where** $x \in S$ *fst x* = *fst yy single-sem C* (*snd x*) (*snd yy*)
by (*metis fst-conv in-sem snd-conv*)
then have *fst* (*modify-lvar-to r v x*) = *fst y*
by (*simp add*: ‹*y* = *modify-lvar-to r v yy*› *modify-lvar-to-def*)
then show $y \in$ *sem C* (*modify-lvar-to r v* ‘ *S*)
by (*metis* (*mono-tags*, *lifting*) ‹⟨*C*, *snd x*⟩ → *snd yy*› ‹$x \in S$› ‹*y* = *modify-lvar-to r v yy*› *fst-conv*
image-eqI in-sem modify-lvar-to-def snd-conv)
qed
qed

Proposition 15 (appendix C).

theorem *if-sync-rule*:
assumes $\models$ {*P*} *C1* {*P1*}
and $\models$ {*P*} *C2* {*P2*}
and $\models$ {*combine from-nat x P1 P2*} *C* {*combine from-nat x R1 R2*}
and $\models$ {*R1*} *C1′* {*Q1*}
and $\models$ {*R2*} *C2′* {*Q2*}

and *not-free-var-hyper x P1*
and *not-free-var-hyper x P2*
and *injective* (*from-nat* :: *nat* ⇒ ′*a*)

and *not-free-var-hyper x R1*
and *not-free-var-hyper x R2*

shows $\models$ {*P*} *If* (*Seq C1* (*Seq C C1′*)) (*Seq C2* (*Seq C C2′*)) {*join Q1 Q2*}
proof (*rule hyper-hoare-tripleI*)
fix *S* **assume** *asm0*: *P S*
have *r0*: *sem* (*stmt.If* (*Seq C1* (*Seq C C1′*)) (*Seq C2* (*Seq C C2′*))) *S*
= *sem C1′* (*sem C* (*sem C1 S*)) $\cup$ *sem C2′* (*sem C* (*sem C2 S*))
by (*simp add*: *sem-if sem-seq*)
moreover have *P1* (*sem C1 S*) $\wedge$ *P2* (*sem C2 S*)
using *asm0 assms*(*1*) *assms*(*2*) *hyper-hoare-tripleE* **by** *blast*

let *?S1* = (*modify-lvar-to x* (*from-nat 1*)) ‘ (*sem C1 S*)
let *?S2* = (*modify-lvar-to x* (*from-nat 2*)) ‘ (*sem C2 S*)
let *?f1* = *Set.filter* (λφ. *fst* φ *x* = *from-nat 1*)

let *?f2* = *Set.filter* (λφ. *fst* φ *x* = *from-nat 2*)

have *r*: *from-nat 1* ≠ *from-nat 2*
by (*metis Suc-1 assms*(*8*) *injective-def n-not-Suc-n*)

have *P1 ?S1* ∧ *P2 ?S2*
by (*meson* ‹*P1* (*sem C1 S*) ∧ *P2* (*sem C2 S*)› *assms*(*6*) *assms*(*7*) *not-free-var-hyper-def*)
moreover have *rr1*: *Set.filter* (λφ. *fst* φ *x* = *from-nat 1*) (*?S1* ∪ *?S2*) = *?S1*
using *injective-then-ok*[*of from-nat 1 from-nat 2 ?S1 x*]
by (*metis* (*no-types*, *lifting*) *assms*(*8*) *injective-def num.simps*(*4*) *one-eq-numeral-iff*)
moreover have *rr2*: *Set.filter* (λφ. *fst* φ *x* = *from-nat 2*) (*?S1* ∪ *?S2*) = *?S2*
using *injective-then-ok*[*of from-nat 2 from-nat 1 ?S2 x*]
by (*metis* (*no-types*, *lifting*) *assms*(*8*) *injective-def one-eq-numeral-iff sup-commute verit-eq-simplify*(*10*))
ultimately have *combine from-nat x P1 P2* (*?S1* ∪ *?S2*)
by (*metis combineI*)
then have *combine from-nat x R1 R2* (*sem C* (*?S1* ∪ *?S2*))
using *assms*(*3*) *hyper-hoare-tripleE* **by** *blast*
moreover have *?f1* (*sem C* (*?S1* ∪ *?S2*)) = *sem C ?S1*
using *recover-after-sem*[*of from-nat 1 from-nat 2 ?S1 x ?S2*] *r rr1 rr2*
member-filter[*of* - λφ. *fst* φ *x* = *from-nat 1*] *member-filter*[*of* - λφ. *fst* φ *x* = *from-nat 2*]
by *metis*
then have *R1* (*sem C ?S1*)
by (*metis* (*mono-tags*) *calculation combine-def*)
then have *R1* (*sem C* (*sem C1 S*))
by (*metis assms*(*9*) *not-free-var-hyper-def sem-of-modify-lvar*)
moreover have *?f2* (*sem C* (*?S1* ∪ *?S2*)) = *sem C ?S2*
using *recover-after-sem*[*of from-nat 2 from-nat 1 ?S2 x ?S1*] *r rr1 rr2 sup-commute*[*of*
]
member-filter[*of* - λφ. *fst* φ *x* = *from-nat 1 ?S1* ∪ *?S2*] *member-filter*[*of* - λφ. *fst* φ *x* = *from-nat 2 ?S1* ∪ *?S2*]
by *metis*
then have *R2* (*sem C ?S2*)
by (*metis* (*mono-tags*) *calculation*(*1*) *combine-def*)
then have *R2* (*sem C* (*sem C2 S*))
by (*metis assms*(*10*) *not-free-var-hyper-def sem-of-modify-lvar*)

then show *join Q1 Q2* (*sem* (*stmt.If* (*Seq C1* (*Seq C C1'*)) (*Seq C2* (*Seq C C2'*))) *S*)
by (*metis* (*full-types*) *r0 assms*(*4*) *assms*(*5*) *calculation*(*2*) *hyper-hoare-tripleE join-def*)
qed

end

4 Examples

In this file, we prove that the two examples from section IV satisfy resp. violate GNI, using the proof outlines from appendix A.

theory *Examples*
  imports *Logic*
begin

definition *GNI* **where**
  $GNI\ l\ h\ S \longleftrightarrow (\forall \varphi 1\ \varphi 2.\ \varphi 1 \in S \wedge \varphi 2 \in S$
  $\longrightarrow (\exists \varphi \in S.\ snd\ \varphi\ h = snd\ \varphi 1\ h \wedge snd\ \varphi\ l = snd\ \varphi 2\ l))$

lemma *GNI-I*:
  assumes $\bigwedge \varphi 1\ \varphi 2.\ \varphi 1 \in S \wedge \varphi 2 \in S$
  $\Longrightarrow (\exists \varphi \in S.\ snd\ \varphi\ h = snd\ \varphi 1\ h \wedge snd\ \varphi\ l = snd\ \varphi 2\ l)$
  shows *GNI l h S*
  by (*simp add*: *GNI-def assms*)

lemma *program-1-sat-gni*:
  assumes $y \neq l \wedge y \neq h \wedge l \neq h$
  shows $\vdash \{\ (\lambda S.\ True)\ \}\ Seq\ (Havoc\ y)\ (Assign\ l\ (\lambda\sigma.\ (\sigma\ h :: int) + \sigma\ y))\ \{\ GNI\ l\ h\ \}$
proof (*rule RuleSeq*)
  let $?R = \lambda S.\ \forall \varphi 1\ \varphi 2.\ \varphi 1 \in S \wedge \varphi 2 \in S$
  $\longrightarrow (\exists \varphi \in S.\ (snd\ \varphi\ h :: int) = snd\ \varphi 1\ h \wedge snd\ \varphi\ h + snd\ \varphi\ y = snd\ \varphi 2\ h + snd\ \varphi 2\ y\ )$

  show $\vdash \{(\lambda S.\ True)\}\ Havoc\ y\ \{?R\}$
  proof (*rule RuleCons*)
    show $\vdash \{(\lambda S.\ ?R\ \{(l,\ \sigma(y := v))\ |l\ \sigma\ v.\ (l,\ \sigma) \in S\})\}\ Havoc\ y\ \{?R\}$
      using *RuleHavoc*[*of ?R*] **by** *blast*
    show $entails\ (\lambda S.\ True)\ (\lambda S.\ ?R\ \{(l,\ \sigma(y := v))\ |l\ \sigma\ (v :: int).\ (l,\ \sigma) \in S\})$
    proof (*rule entailsI*)
      fix *S*
      show $?R\ \{(l,\ \sigma(y := v))\ |l\ \sigma\ (v :: int).\ (l,\ \sigma) \in S\}$
      proof (*clarify*)
        fix $a\ b\ aa\ ba\ l\ la\ \sigma\ \sigma'\ v\ va$
        assume $asm0$: $(l,\ \sigma) \in S\ (la,\ \sigma') \in S$
        let $?v = (\sigma'(y := va))\ h + (\sigma'(y := va))\ y + -\ \sigma\ h$
        let $?\varphi = (l,\ \sigma(y := ?v))$
        have $snd\ ?\varphi\ h = snd\ (l,\ \sigma(y := v))\ h \wedge snd\ ?\varphi\ h + snd\ ?\varphi\ y = snd\ (la,\ \sigma'(y := va))\ h + snd\ (la,\ \sigma'(y := va))\ y$
          using *assms* **by** *force*
        then show $\exists \varphi \in \{(l,\ \sigma(y := v))\ |l\ \sigma\ v.\ (l,\ \sigma) \in S\}.$
          $snd\ \varphi\ h = snd\ (l,\ \sigma(y := v))\ h \wedge snd\ \varphi\ h + snd\ \varphi\ y = snd\ (la,\ \sigma'(y := va))\ h + snd\ (la,\ \sigma'(y := va))\ y$
          using *asm0*(*1*) **by** *blast*
      qed
    qed

show *entails ?R ?R*
  by (*meson entailsI*)
qed
show $\vdash$ {*?R*} (*Assign l* ($\lambda\sigma$. σ *h* + σ *y*)) {*GNI l h*}
proof (*rule RuleCons*)
  show $\vdash$ {(λS. *GNI l h* {(*la*, σ(*l* := σ *h* + σ *y*)) |*la* σ. (*la*, σ) $\in$ *S*})} *Assign l* ($\lambda\sigma$. σ *h* + σ *y*) {*GNI l h*}
    using *RuleAssign*[*of GNI l h l* $\lambda\sigma$. σ *h* + σ *y*] **by** *blast*
  show *entails* (*GNI l h*) (*GNI l h*)
    by (*simp add*: *entails-def*)

show *entails ?R* (λS. *GNI l h* {(*la*, σ(*l* := σ *h* + σ *y*)) |*la* σ. (*la*, σ) $\in$ *S*})
proof (*rule entailsI*)
  fix *S*
  assume *asm0*: $\forall\varphi 1\ \varphi 2.\ \varphi 1 \in S \wedge \varphi 2 \in S \longrightarrow (\exists\varphi\in S.\ snd\ \varphi\ h = snd\ \varphi 1\ h \wedge snd\ \varphi\ h + snd\ \varphi\ y = snd\ \varphi 2\ h + snd\ \varphi 2\ y)$
  show *GNI l h* {(*la*, σ(*l* := σ *h* + σ *y*)) |*la* σ. (*la*, σ) $\in$ *S*}
  proof (*rule GNI-I*)
    fix $\varphi 1$ $\varphi 2$
    assume *asm1*: $\varphi 1 \in$ {(*la*, σ(*l* := σ *h* + σ *y*)) |*la* σ. (*la*, σ) $\in$ *S*} $\wedge$ $\varphi 2 \in$ {(*la*, σ(*l* := σ *h* + σ *y*)) |*la* σ. (*la*, σ) $\in$ *S*}
    then obtain *la* σ *la'* σ' **where** (*la*, σ) $\in$ *S* (*la'*, σ') $\in$ *S* $\varphi 1$ = (*la*, σ(*l* := σ *h* + σ *y*)) $\varphi 2$ = (*la'*, σ'(*l* := σ' *h* + σ' *y*))
      by *blast*
    then obtain φ **where** $\varphi\in S$ *snd* φ *h* = σ *h* *snd* φ *h* + *snd* φ *y* = σ' *h* + σ' *y*
      using *asm0 snd-conv* **by** *force*
    let $?\varphi$ = (*fst* φ, (*snd* φ)(*l* := *snd* φ *h* + *snd* φ *y*))
    have *snd* $?\varphi$ *h* = *snd* $\varphi 1$ *h* $\wedge$ *snd* $?\varphi$ *l* = *snd* $\varphi 2$ *l*
      using ‹$\varphi 1$ = (*la*, σ(*l* := σ *h* + σ *y*))› ‹$\varphi 2$ = (*la'*, σ'(*l* := σ' *h* + σ' *y*))›
        ‹*snd* φ *h* + *snd* φ *y* = σ' *h* + σ' *y*› ‹*snd* φ *h* = σ *h*› *assms* **by** *force*
    then show $\exists\varphi\in$ {(*la*, σ(*l* := σ *h* + σ *y*)) |*la* σ. (*la*, σ) $\in$ *S*}. *snd* φ *h* = *snd* $\varphi 1$ *h* $\wedge$ *snd* φ *l* = *snd* $\varphi 2$ *l*
      using ‹$\varphi \in S$› *mem-Collect-eq*[*of* $?\varphi$]
      by (*metis* (*mono-tags*, *lifting*) *prod.collapse*)
  qed
qed
qed
qed

lemma *program-2-violates-gni*:
  assumes $y \neq l \wedge y \neq h \wedge l \neq h$
  shows $\vdash$ { (λS. $\exists\, a \in S$. $\exists\, b \in S$. (*snd a h* :: *nat*) $\neq$ *snd b h*) }
  Seq (*Seq* (*Havoc y*) (*Assume* ($\lambda\sigma$. σ *y* $\geq$ (*0* :: *nat*) $\wedge$ σ *y* $\leq$ (*100* :: *nat*)))) (*Assign l* ($\lambda\sigma$. σ *h* + σ *y*))
  {λ(*S* :: (('*lvar* $\Rightarrow$ '*lval*) $\times$ ('*a* $\Rightarrow$ *nat*)) *set*). $\neg$ *GNI l h S*}
proof (*rule RuleSeq*)

let *?R0* = λ(*S* :: (('*lvar* $\Rightarrow$ '*lval*) $\times$ ('*a* $\Rightarrow$ *nat*)) *set*).
($\exists$ *a* $\in$ *S*. $\exists$ *b* $\in$ *S*. *snd b h* > *snd a h* $\wedge$ *snd a y* $\geq$ (*0* :: *nat*) $\wedge$ *snd a y* $\leq$ *100* $\wedge$ *snd b y* = *100*)
let *?R1* = λ(*S* :: (('*lvar* $\Rightarrow$ '*lval*) $\times$ ('*a* $\Rightarrow$ *nat*)) *set*).
($\exists$ *a* $\in$ *S*. $\exists$ *b* $\in$ *S*. *snd b h* > *snd a h* $\wedge$ *snd b y* = *100*) $\wedge$ ($\forall$ *c* $\in$ *S*. *snd c y* $\leq$ *100*)
let *?R2* = λ(*S* :: (('*lvar* $\Rightarrow$ '*lval*) $\times$ ('*a* $\Rightarrow$ *nat*)) *set*).
$\exists$ *a* $\in$ *S*. $\exists$ *b* $\in$ *S*. $\forall$ *c* $\in$ *S*. *snd c h* = *snd a h* $\longrightarrow$ *snd c h* + *snd c y* = *snd b h* + *snd b y*

show $\vdash$ {(λ*S*. $\exists$ *a*$\in$*S*. $\exists$ *b*$\in$*S*. *snd a h* $\neq$ *snd b h*)} *Seq* (*Havoc y*) (*Assume* ($\lambda\sigma$. *0* $\leq$ σ *y* $\wedge$ σ *y* $\leq$ (*100* :: *nat*))) {*?R1*}
proof (*rule RuleSeq*)
show $\vdash$ {(λ*S*. $\exists$ *a*$\in$*S*. $\exists$ *b*$\in$*S*. *snd a h* $\neq$ *snd b h*)} *Havoc y* { *?R0* }
proof (*rule RuleCons*)
show $\vdash$ {(λ*S*. *?R0* {(*l*, σ(*y* := *v*)) |*l* σ *v*. (*l*, σ) $\in$ *S*})} *Havoc y* {*?R0*}
using *RuleHavoc*[*of - y*] **by** *fast*
show *entails ?R0 ?R0*
by (*simp add*: *entailsI*)
show *entails* (λ*S*. $\exists$ *a*$\in$*S*. $\exists$ *b*$\in$*S*. *snd a h* $\neq$ *snd b h*) (λ*S*. *?R0* {(*l*, σ(*y* := *v*)) |*l* σ *v*. (*l*, σ) $\in$ *S*})
proof (*rule entailsI*)
fix *S* :: (('*lvar* $\Rightarrow$ '*lval*) $\times$ ('*a* $\Rightarrow$ *nat*)) *set*
assume $\exists$ *a*$\in$*S*. $\exists$ *b*$\in$*S*. *snd a h* $\neq$ *snd b h*
then obtain *a b* **where** *a* $\in$ *S b* $\in$ *S snd b h* > *snd a h*
by (*meson linorder-neq-iff*)
let *?a* = (*fst a*, (*snd a*)(*y* := *100*))
let *?b* = (*fst b*, (*snd b*)(*y* := *100*))
have *?a* $\in$ {(*l*, σ(*y* := *v*)) |*l* σ *v*. (*l*, σ) $\in$ *S*} $\wedge$ *?b* $\in$ {(*l*, σ(*y* := *v*)) |*l* σ *v*. (*l*, σ) $\in$ *S*}
using ‹*a* $\in$ *S*› ‹*b* $\in$ *S*› **by** *fastforce*
moreover have *snd ?b h* > *snd ?a h* $\wedge$ *snd ?a y* $\geq$ (*0* :: *nat*) $\wedge$ *snd ?a y* $\leq$ *100* $\wedge$ *snd ?b y* = *100*
using ‹*snd a h* < *snd b h*› *assms* **by** *force*
ultimately show *?R0* {(*l*, σ(*y* := *v*)) |*l* σ *v*. (*l*, σ) $\in$ *S*} **by** *blast*
qed
qed
show $\vdash$ {*?R0*} *Assume* ($\lambda\sigma$. *0* $\leq$ σ *y* $\wedge$ σ *y* $\leq$ *100*) {*?R1*}
proof (*rule RuleCons*)
show $\vdash$ {(λ*S*. *?R1* (*Set.filter* (($\lambda\sigma$. *0* $\leq$ σ *y* $\wedge$ σ *y* $\leq$ *100*) $\circ$ *snd*) *S*))} *Assume* ($\lambda\sigma$. *0* $\leq$ σ *y* $\wedge$ σ *y* $\leq$ *100*) {*?R1*}
using *RuleAssume*[*of -* $\lambda\sigma$. *0* $\leq$ σ *y* $\wedge$ σ *y* $\leq$ *100*]
by *fast*
show *entails ?R1 ?R1*
by (*simp add*: *entailsI*)
show *entails ?R0* (λ*S*. *?R1* (*Set.filter* (($\lambda\sigma$. *0* $\leq$ σ *y* $\wedge$ σ *y* $\leq$ *100*) $\circ$ *snd*) *S*))
proof (*rule entailsI*)

fix S :: (('$lvar$ $\Rightarrow$ '$lval$) $\times$ ('a $\Rightarrow$ nat)) set
assume $asm0$: $?R0\ S$
then obtain $a\ b$ **where** $a{\in}S$ $b{\in}S$ $snd\ a\ h < snd\ b\ h \wedge 0 \leq snd\ a\ y \wedge snd\ a\ y \leq (100 :: nat) \wedge snd\ b\ y = 100$
by $blast$
then have $a \in Set.filter\ ((\lambda\sigma.\ 0 \leq \sigma\ y \wedge \sigma\ y \leq 100) \circ snd)\ S \wedge b \in Set.filter\ ((\lambda\sigma.\ 0 \leq \sigma\ y \wedge \sigma\ y \leq 100) \circ snd)\ S$
by ($simp\ add$: ‹$a \in S$› ‹$b \in S$›)
then show $?R1\ (Set.filter\ ((\lambda\sigma.\ 0 \leq \sigma\ y \wedge \sigma\ y \leq 100) \circ snd)\ S)$
using ‹$snd\ a\ h < snd\ b\ h \wedge 0 \leq snd\ a\ y \wedge snd\ a\ y \leq 100 \wedge snd\ b\ y = 100$› **by** $force$
qed
qed
qed
show $\vdash$ { $?R1$ } $Assign\ l\ (\lambda\sigma.\ \sigma\ h + \sigma\ y)$ $\{\lambda S.\ \neg\ GNI\ l\ h\ S\}$
proof ($rule\ RuleCons$)
show $\vdash \{(\lambda S.\ \neg\ GNI\ l\ h\ \{(la, \sigma(l := \sigma\ h + \sigma\ y))\ |la\ \sigma.\ (la, \sigma) \in S\})\}$ $Assign\ l\ (\lambda\sigma.\ \sigma\ h + \sigma\ y)$ $\{\lambda S.\ \neg\ GNI\ l\ h\ S\}$
using $RuleAssign[of\ \lambda S.\ \neg\ GNI\ l\ h\ S\ l\ \lambda\sigma.\ \sigma\ h + \sigma\ y]$
by $blast$
show $entails\ (\lambda S.\ \neg\ GNI\ l\ h\ S)\ (\lambda S.\ \neg\ GNI\ l\ h\ S)$
by ($simp\ add$: $entails\text{-}def$)
show $entails\ (\lambda S.\ (\exists\, a{\in}S.\ \exists\, b{\in}S.\ snd\ a\ h < snd\ b\ h \wedge snd\ b\ y = 100) \wedge (\forall\, c{\in}S.\ snd\ c\ y \leq 100))$
$(\lambda(S :: (('lvar \Rightarrow 'lval) \times ('a \Rightarrow nat))\ set).\ \neg\ GNI\ l\ h\ \{(la, \sigma(l := \sigma\ h + \sigma\ y))\ |la\ \sigma.\ (la, \sigma) \in S\})$
proof ($rule\ entailsI$)
fix S :: (('$lvar$ $\Rightarrow$ '$lval$) $\times$ ('a $\Rightarrow$ nat)) set
assume $asm0$: $(\exists\, a{\in}S.\ \exists\, b{\in}S.\ snd\ a\ h < snd\ b\ h \wedge snd\ b\ y = 100) \wedge (\forall\, c{\in}S.\ snd\ c\ y \leq 100)$
then obtain $a\ b$ **where** $asm1$: $a{\in}S$ $b{\in}S$ $snd\ a\ h < snd\ b\ h \wedge snd\ b\ y = 100$ **by** $blast$
let $?a = (fst\ a, (snd\ a)(l := snd\ a\ h + snd\ a\ y))$
let $?b = (fst\ b, (snd\ b)(l := snd\ b\ h + snd\ b\ y))$
have $\bigwedge la\ \sigma.\ (la, \sigma) \in S \Longrightarrow (\sigma(l := \sigma\ h + \sigma\ y))\ h = snd\ ?a\ h \Longrightarrow (\sigma(l := \sigma\ h + \sigma\ y))\ l \neq snd\ ?b\ l$
using $asm0\ asm1(3)\ assms$ **by** $fastforce$
moreover have r: $?a \in \{(la, \sigma(l := \sigma\ h + \sigma\ y))\ |la\ \sigma.\ (la, \sigma) \in S\} \wedge ?b \in \{(la, \sigma(l := \sigma\ h + \sigma\ y))\ |la\ \sigma.\ (la, \sigma) \in S\}$
using $asm1(1)\ asm1(2)$ **by** $fastforce$
show $\neg\ GNI\ l\ h\ \{(la, \sigma(l := \sigma\ h + \sigma\ y))\ |la\ \sigma.\ (la, \sigma) \in S\}$
proof ($rule\ ccontr$)
assume $\neg\ \neg\ GNI\ l\ h\ \{(la, \sigma(l := \sigma\ h + \sigma\ y))\ |la\ \sigma.\ (la, \sigma) \in S\}$
then have $GNI\ l\ h\ \{(la, \sigma(l := \sigma\ h + \sigma\ y))\ |la\ \sigma.\ (la, \sigma) \in S\}$
by $blast$
then obtain φ **where** $\varphi \in \{(la, \sigma(l := \sigma\ h + \sigma\ y))\ |la\ \sigma.\ (la, \sigma) \in S\}$ $snd\ \varphi\ h = snd\ ?a\ h$ $snd\ \varphi\ l = snd\ ?b\ l$
using $GNI\text{-}def[of\ l\ h\ \{(la, \sigma(l := \sigma\ h + \sigma\ y))\ |la\ \sigma.\ (la, \sigma) \in S\}]\ r$
by $meson$

```
        then show False
          using calculation by auto
      qed
    qed
  qed
qed

end
```

5 Expressivity of Hyper Hoare Logic

In this file, we define program hyperproperties (definition 7), and prove theorem 3.

5.1 Program Hyperproperties

```
theory ProgramHyperproperties
  imports Logic
begin
```

Definition 7

```
type-synonym 'a hyperproperty = ('a × 'a) set ⇒ bool

definition set-of-traces where
  set-of-traces C = { (σ, σ') |σ σ'. ⟨C, σ⟩ → σ' }

definition hypersat where
  hypersat C H ⟷ H (set-of-traces C)

definition copy-p-state where
  copy-p-state to-pvar to-lval σ x = to-lval (σ (to-pvar x))

definition recover-p-state where
  recover-p-state to-pval to-lvar l x = to-pval (l (to-lvar x))

lemma injective-then-exists-inverse:
  assumes injective to-lvar
  shows ∃ to-pvar. (∀ x. to-pvar (to-lvar x) = x)
proof -
  let ?to-pvar = λy. SOME x. to-lvar x = y
  have ⋀x. ?to-pvar (to-lvar x) = x
    by (metis (mono-tags, lifting) assms injective-def someI)
  then show ?thesis
    by force
qed

lemma single-step-then-in-sem:
  assumes single-sem C σ σ'
```

and $(l, \sigma) \in S$
shows $(l, \sigma') \in$ *sem C S*
using *assms(1) assms(2) in-sem* **by** *fastforce*

lemma *in-set-of-traces*:
$(\sigma, \sigma') \in$ *set-of-traces C* $\longleftrightarrow \langle C, \sigma\rangle \to \sigma'$
by (*simp add*: *set-of-traces-def*)

lemma *in-set-of-traces-then-in-sem*:
assumes $(\sigma, \sigma') \in$ *set-of-traces C*
and $(l, \sigma) \in S$
shows $(l, \sigma') \in$ *sem C S*
using *in-set-of-traces assms single-step-then-in-sem* **by** *metis*

lemma *set-of-traces-same*:
assumes $\bigwedge x.$ *to-pvar* (*to-lvar x*) = *x*
and $\bigwedge x.$ *to-pval* (*to-lval x*) = *x*
and *S* = {(*copy-p-state to-pvar to-lval* σ, σ) |σ. *True*}
shows {(*recover-p-state to-pval to-lvar l*, σ') |*l* σ'. (*l*, σ') $\in$ *sem C S*} = *set-of-traces C*
(**is** *?A* = *?B*)
proof
show *?A* $\subseteq$ *?B*
proof (*rule subsetPairI*)
fix σ σ' **assume** *asm0*: $(\sigma, \sigma') \in$ {(*recover-p-state to-pval to-lvar l*, σ') |*l* σ'. (*l*, σ') $\in$ *sem C S*}
then obtain *l* **where** σ = *recover-p-state to-pval to-lvar l* (*l*, σ') $\in$ *sem C S*
by *blast*
then obtain *x* **where** (*l*, *x*) $\in$ *S* $\langle C, x\rangle \to \sigma'$
by (*metis fst-conv in-sem snd-conv*)
then have *l* = *copy-p-state to-pvar to-lval x*
using *assms(3)* **by** *blast*
moreover have $\sigma = x$
proof (*rule ext*)
fix *y* **show** $\sigma\ y = x\ y$
by (*simp add*: ‹σ = *recover-p-state to-pval to-lvar l*› *assms(1) assms(2) calculation copy-p-state-def recover-p-state-def*)
qed
ultimately show $(\sigma, \sigma') \in$ *set-of-traces C*
by (*simp add*: ‹$\langle C, x\rangle \to \sigma'$› *set-of-traces-def*)
qed
show *?B* $\subseteq$ *?A*
proof (*rule subsetPairI*)
fix σ σ' **assume** *asm0*: $(\sigma, \sigma') \in$ *set-of-traces C*
let *?l* = *copy-p-state to-pvar to-lval* σ
have (*?l*, σ) $\in$ *S*
using *assms(3)* **by** *blast*
then have (*?l*, σ') $\in$ *sem C S*

using *asm0 in-set-of-traces-then-in-sem* **by** *blast*
moreover have *recover-p-state to-pval to-lvar ?l* $= \sigma$
proof (*rule ext*)
fix x **show** *recover-p-state to-pval to-lvar* (*copy-p-state to-pvar to-lval* σ) $x = \sigma\ x$
by (*simp add*: *assms*(*1*) *assms*(*2*) *copy-p-state-def recover-p-state-def*)
qed
ultimately show $(\sigma, \sigma') \in \{($*recover-p-state to-pval to-lvar* $l, \sigma') \mid l\ \sigma'.\ (l, \sigma') \in$ *sem* $C\ S\}$
by *force*
qed
qed

Theorem 3

theorem *proving-hyperproperties*:

fixes *to-lvar* :: $'pvar \Rightarrow 'lvar$
fixes *to-lval* :: $'pval \Rightarrow 'lval$

assumes *injective to-lvar*
and *injective to-lval*

shows $\exists P\ Q$::($'lvar$, $'lval$, $'pvar$, $'pval$) *state hyperassertion*. ($\forall C$. *hypersat* $C\ H \longleftrightarrow \models \{P\}\ C\ \{Q\}$)
proof $-$

obtain *to-pval* :: $'lval \Rightarrow 'pval$ **where** *r1*: $\bigwedge x$. *to-pval* (*to-lval* x) $= x$
using *assms*(*2*) *injective-then-exists-inverse* **by** *blast*

obtain *to-pvar* :: $'lvar \Rightarrow 'pvar$ **where** *r2*: $\bigwedge x$. *to-pvar* (*to-lvar* x) $= x$
using *assms*(*1*) *injective-then-exists-inverse* **by** *blast*

let *?P* $= \lambda S.\ S = \{($*copy-p-state to-pvar to-lval* $\sigma, \sigma) \mid \sigma.$ *True* $\}$
let *?Q* $= \lambda S.\ H\ \{\ ($*recover-p-state to-pval to-lvar* $l, \sigma') \mid l\ \sigma'.\ (l, \sigma') \in S\ \}$

have $\bigwedge C$. *hypersat* $C\ H \longleftrightarrow \models \{$*?P*$\}\ C\ \{$*?Q*$\}$
proof
fix C
assume *hypersat* $C\ H$
show $\models \{$*?P*$\}\ C\ \{$*?Q*$\}$
proof (*rule hyper-hoare-tripleI*)
fix S **assume** $S = \{($*copy-p-state to-pvar to-lval* $\sigma, \sigma) \mid \sigma.$ *True*$\}$
have $\{($*recover-p-state to-pval to-lvar* $l, \sigma') \mid l\ \sigma'.\ (l, \sigma') \in$ *sem* $C\ S\}$ $=$ *set-of-traces* C
using ‹$S = \{($*copy-p-state to-pvar to-lval* $\sigma, \sigma) \mid \sigma.$ *True*$\}$› *set-of-traces-same*[*of to-pvar to-lvar to-pval to-lval*]
r1 r2 **by** *presburger*
then show $H\ \{($*recover-p-state to-pval to-lvar* $l, \sigma') \mid l\ \sigma'.\ (l, \sigma') \in$ *sem* $C\ S\}$
using ‹*hypersat* $C\ H$› *hypersat-def* **by** *metis*

qed
next
fix C
let $?S = \{(copy\text{-}p\text{-}state\ to\text{-}pvar\ to\text{-}lval\ \sigma,\ \sigma)\ |\sigma.\ True\ \}$
assume $\models \{?P\}\ C\ \{?Q\}$
then have $?Q\ (sem\ C\ ?S)$
by $(simp\ add\text{:}\ hyper\text{-}hoare\text{-}triple\text{-}def)$
moreover have $\{(recover\text{-}p\text{-}state\ to\text{-}pval\ to\text{-}lvar\ l,\ \sigma')\ |l\ \sigma'.\ (l,\ \sigma') \in sem\ C\ ?S\} = set\text{-}of\text{-}traces\ C$
using $r1\ r2\ set\text{-}of\text{-}traces\text{-}same[of\ to\text{-}pvar\ to\text{-}lvar\ to\text{-}pval\ to\text{-}lval]$
by $presburger$
ultimately show $hypersat\ C\ H$
by $(simp\ add\text{:}\ hypersat\text{-}def)$
qed
then show $?thesis$
by $auto$
qed

Hypersafety, hyperliveness

definition $max\text{-}k$ **where**
$max\text{-}k\ k\ S \longleftrightarrow finite\ S \wedge card\ S \leq k$

definition $hypersafety$ **where**
$hypersafety\ P \longleftrightarrow (\forall S.\ \neg\ P\ S \longrightarrow (\forall S'.\ S \subseteq S' \longrightarrow \neg\ P\ S'))$

definition $k\text{-}hypersafety$ **where**
$k\text{-}hypersafety\ k\ P \longleftrightarrow (\forall S.\ \neg\ P\ S \longrightarrow (\exists S'.\ S' \subseteq S \wedge max\text{-}k\ k\ S' \wedge (\forall S''.\ S' \subseteq S'' \longrightarrow \neg\ P\ S'')))$

definition $hyperliveness$ **where**
$hyperliveness\ P \longleftrightarrow (\forall S.\ \exists S'.\ S \subseteq S' \wedge P\ S')$

lemma $k\text{-}hypersafetyI$:
assumes $\bigwedge S.\ \neg\ P\ S \Longrightarrow \exists S'.\ S' \subseteq S \wedge max\text{-}k\ k\ S' \wedge (\forall S''.\ S' \subseteq S'' \longrightarrow \neg\ P\ S'')$
shows $k\text{-}hypersafety\ k\ P$
by $(simp\ add\text{:}\ assms\ k\text{-}hypersafety\text{-}def)$

lemma $hypersafetyI$:
assumes $\bigwedge S\ S'.\ \neg\ P\ S \Longrightarrow S \subseteq S' \Longrightarrow \neg\ P\ S'$
shows $hypersafety\ P$
by $(metis\ assms\ hypersafety\text{-}def)$

lemma $hyperlivenessI$:
assumes $\bigwedge S.\ \exists S'.\ S \subseteq S' \wedge P\ S'$
shows $hyperliveness\ P$
using $assms\ hyperliveness\text{-}def$ **by** $blast$

lemma *k-hypersafe-is-hypersafe*:
assumes *k-hypersafety k P*
shows *hypersafety P*
by (*metis* (*full-types*) *assms dual-order.trans hypersafety-def k-hypersafety-def*)

lemma *one-safety-equiv*:
assumes *sat H*
shows *k-hypersafety 1 H* $\longleftrightarrow$ ($\exists P. \forall S.\ H\ S \longleftrightarrow (\forall \tau \in S.\ P\ \tau)$) (**is** *?A* $\longleftrightarrow$ *?B*)
proof
assume *?B*
then obtain *P* **where** *asm0*: $\bigwedge S.\ H\ S \longleftrightarrow (\forall \tau \in S.\ P\ \tau)$
by *auto*
show *?A*
proof (*rule k-hypersafetyI*)
fix *S*
assume *asm1*: $\neg$ *H S*
then obtain τ **where** $\tau \in S$ $\neg$ $P\ \tau$
using *asm0* **by** *blast*
let $?S = \{\tau\}$
have $?S \subseteq S \wedge$ *max-k 1 ?S* $\wedge (\forall S''.\ ?S \subseteq S'' \longrightarrow \neg H\ S'')$
using ‹¬ $P\ \tau$› ‹$\tau \in S$› *asm0 max-k-def* **by** *fastforce*
then show $\exists S' \subseteq S.$ *max-k 1* $S' \wedge (\forall S''.\ S' \subseteq S'' \longrightarrow \neg H\ S'')$ **by** *blast*
qed
next
assume *?A*
let $?P = \lambda\tau.\ H\ \{\tau\}$
have $\bigwedge S.\ H\ S \longleftrightarrow (\forall \tau \in S.\ ?P\ \tau)$
proof
fix *S* **assume** *H S*
then show $\forall \tau \in S.\ ?P\ \tau$
using ‹*k-hypersafety 1 H*› *hypersafety-def k-hypersafe-is-hypersafe* **by** *auto*
next
fix *S* **assume** *asm0*: $\forall \tau \in S.\ ?P\ \tau$
show *H S*
proof (*rule ccontr*)
assume ¬ *H S*
then obtain S' **where** $S' \subseteq S \wedge$ *max-k 1* $S' \wedge (\forall S''.\ S' \subseteq S'' \longrightarrow \neg H\ S'')$
by (*metis* ‹*k-hypersafety 1 H*› *k-hypersafety-def*)
then show *False*
proof (*cases* $S' = \{\}$)
case *True*
then show *?thesis*
by (*metis* ‹$S' \subseteq S \wedge$ *max-k 1* $S' \wedge (\forall S''.\ S' \subseteq S'' \longrightarrow \neg H\ S'')$› *assms empty-subsetI sat-def*)
next
case *False*
then obtain τ **where** $\tau \in S'$

by *blast*
then have *card* $S' = 1$
by (*metis False One-nat-def Suc-leI* ‹$S' \subseteq S \wedge$ *max-k 1* $S' \wedge (\forall S''.\ S' \subseteq S'' \longrightarrow \neg\ H\ S'')$› *card-gt-0-iff le-antisym max-k-def*)
then have $S' = \{\tau\}$
using ‹$\tau \in S'$› *card-1-singletonE* **by** *auto*
then show *?thesis*
using ‹$S' \subseteq S \wedge$ *max-k 1* $S' \wedge (\forall S''.\ S' \subseteq S'' \longrightarrow \neg\ H\ S'')$› *asm0* **by** *fastforce*
qed
qed
qed
then show *?B* **by** *blast*
qed

definition *hoarify* **where**
hoarify $P\ Q\ S \longleftrightarrow (\forall p \in S.$ *fst* $p \in P \longrightarrow$ *snd* $p \in Q)$

lemma *hoarify-hypersafety*:
hypersafety (*hoarify* $P\ Q$)
by (*metis* (*no-types*, *opaque-lifting*) *hoarify-def hypersafetyI subsetD*)

theorem *hypersafety-1-hoare-logic*:
k-hypersafety 1 (*hoarify* $P\ Q$)
proof (*rule k-hypersafetyI*)
fix S **assume** $\neg$ *hoarify* $P\ Q\ S$
then obtain τ **where** $\tau \in S$ *fst* $\tau \in P$ *snd* $\tau \notin Q$
using *hoarify-def* **by** *blast*
let $?S = \{\tau\}$
have $?S \subseteq S \wedge$ *max-k 1* $?S \wedge (\forall S''.\ ?S \subseteq S'' \longrightarrow \neg$ *hoarify* $P\ Q\ S'')$
by (*metis Compl-iff One-nat-def* ‹$\tau \in S$› ‹*fst* $\tau \in P$› ‹*snd* $\tau \notin Q$› *card.empty card.insert compl-le-compl-iff empty-not-insert finite.intros*(*1*) *finite.intros*(*2*) *hoarify-def insert-absorb le-numeral-extra*(*4*) *max-k-def subset-Compl-singleton*)
then show $\exists S' \subseteq S.$ *max-k 1* $S' \wedge (\forall S''.\ S' \subseteq S'' \longrightarrow \neg$ *hoarify* $P\ Q\ S'')$
by *meson*
qed

definition *incorrectnessify* **where**
incorrectnessify $P\ Q\ S \longleftrightarrow (\forall \sigma' \in Q.\ \exists \sigma \in P.\ (\sigma, \sigma') \in S)$

lemma *incorrectnessify-liveness*:
assumes $P \neq \{\}$
shows *hyperliveness* (*incorrectnessify* $P\ Q$)
proof (*rule hyperlivenessI*)
fix S

obtain σ **where** *asm0*: $\sigma \in P$
using *assms* **by** *blast*
let $?S = S \cup \{(\sigma, \sigma') \mid \sigma'. \sigma' \in Q\}$
have *incorrectnessify* P Q $?S$
using *asm0 incorrectnessify-def* **by** *force*
then show $\exists S'. S \subseteq S' \wedge$ *incorrectnessify* P Q S'
using *sup.cobounded1* **by** *blast*
qed

definition *real-incorrectnessify* **where**
real-incorrectnessify P Q $S \longleftrightarrow (\forall \sigma \in P. \exists \sigma' \in Q. (\sigma, \sigma') \in S)$

lemma *real-incorrectnessify-liveness*:
assumes $Q \neq \{\}$
shows *hyperliveness* (*real-incorrectnessify* P Q)
by (*metis UNIV-I assms equals0I hyperliveness-def real-incorrectnessify-def subsetI*)

Verifying GNI

definition *gni-hyperassertion* :: $'n \Rightarrow 'n \Rightarrow ('n \Rightarrow 'v)$ *hyperassertion* **where**
gni-hyperassertion h l $S \longleftrightarrow (\forall \sigma \in S. \forall v. \exists \sigma' \in S. \sigma'\ h = v \wedge \sigma\ l = \sigma'\ l)$

definition *semify* **where**
semify Σ $S = \{ (l, \sigma') \mid \sigma'\ \sigma\ l. (l, \sigma) \in S \wedge (\sigma, \sigma') \in \Sigma \}$

definition *hyperprop-hht* **where**
hyperprop-hht P Q $\Sigma \longleftrightarrow (\forall S. P\ S \longrightarrow Q$ (*semify* Σ S))

Footnote 4

theorem *any-hht-hyperprop*:
$\models \{P\}\ C\ \{Q\} \longleftrightarrow$ *hypersat* C (*hyperprop-hht* P Q) (**is** $?A \longleftrightarrow ?B$)
proof
have $\bigwedge S$. *semify* (*set-of-traces* C) $S =$ *sem* C S
proof −
fix S
have $\bigwedge l\ \sigma'. (l, \sigma') \in$ *sem* C $S \longleftrightarrow (l, \sigma') \in$ *semify* (*set-of-traces* C) S
proof −
fix l σ'
have $(l, \sigma') \in$ *sem* C $S \longleftrightarrow (\exists \sigma. (l, \sigma) \in S \wedge \langle C, \sigma\rangle \rightarrow \sigma')$
by (*simp add*: *in-sem*)
also have ... $\longleftrightarrow (\exists \sigma. (l, \sigma) \in S \wedge (\sigma, \sigma') \in$ *set-of-traces* C)
using *set-of-traces-def* **by** *fastforce*
then show $(l, \sigma') \in$ *sem* C $S \longleftrightarrow (l, \sigma') \in$ *semify* (*set-of-traces* C) S
by (*simp add*: *calculation semify-def*)
qed
then show *semify* (*set-of-traces* C) $S =$ *sem* C S
by *auto*
qed
show $?A \Longrightarrow ?B$

by (*simp add*: ‹$\bigwedge S$. *semify* (*set-of-traces* C) S = *sem* C S› *hyper-hoare-tripleE hyperprop-hht-def hypersat-def*)
show *?B* $\Longrightarrow$ *?A*
by (*simp add*: ‹$\bigwedge S$. *semify* (*set-of-traces* C) S = *sem* C S› *hyper-hoare-triple-def hyperprop-hht-def hypersat-def*)
qed

end

In this file, we prove most results of section V: hyper-triples subsume many other triples, as well as example 4.

theory *Expressivity*
imports *ProgramHyperproperties*
begin

5.2 Hoare Logic (HL) [6]

Definition 8

definition *HL* **where**
HL P C Q $\longleftrightarrow$ ($\forall \sigma\ \sigma'\ l.\ (l, \sigma) \in P \wedge (\langle C, \sigma\rangle \to \sigma') \longrightarrow (l, \sigma') \in Q$)

lemma *HLI*:
assumes $\bigwedge \sigma\ \sigma'\ l.\ (l, \sigma) \in P \Longrightarrow \langle C, \sigma\rangle \to \sigma' \Longrightarrow (l, \sigma') \in Q$
shows *HL* P C Q
using *assms HL-def* **by** *blast*

lemma *hoarifyI*:
assumes $\bigwedge \sigma\ \sigma'.\ (\sigma, \sigma') \in S \Longrightarrow \sigma \in P \Longrightarrow \sigma' \in Q$
shows *hoarify* P Q S
by (*metis assms hoarify-def prod.collapse*)

definition *HL-hyperprop* **where**
HL-hyperprop P Q S $\longleftrightarrow$ ($\forall l.\ \forall p \in S.\ (l, \mathit{fst}\ p) \in P \longrightarrow (l, \mathit{snd}\ p) \in Q$)

lemma *connection-HL*:
HL P C Q $\longleftrightarrow$ *HL-hyperprop* P Q (*set-of-traces* C) (**is** *?A* $\longleftrightarrow$ *?B*)
proof
assume *?A*
then show *?B*
by (*simp add*: *HL-def HL-hyperprop-def set-of-traces-def*)
next
assume *?B*
show *?A*
proof (*rule HLI*)
fix σ σ' l **assume** *asm0*: $(l, \sigma) \in P$ $\langle C, \sigma\rangle \to \sigma'$
then have $(\sigma, \sigma') \in$ *set-of-traces* C
by (*simp add*: *set-of-traces-def*)
then show $(l, \sigma') \in Q$

using ‹*HL-hyperprop P Q* (*set-of-traces C*)› *asm0*(*1*) *HL-hyperprop-def* **by** *fastforce*
qed
qed

Proposition 1

theorem *HL-expresses-hyperproperties*:
$\exists\, H.\ (\forall\, C.\ \textit{hypersat}\ C\ H \longleftrightarrow \textit{HL}\ P\ C\ Q) \wedge \textit{k-hypersafety}\ 1\ H$
proof −
let *?H* = *HL-hyperprop P Q*
have $\bigwedge C.\ \textit{hypersat}\ C\ ?H \longleftrightarrow \textit{HL}\ P\ C\ Q$
by (*simp add*: *connection-HL hypersat-def*)
moreover have *k-hypersafety 1 ?H*
proof (*rule k-hypersafetyI*)
fix *S* **assume** *asm0*: ¬ *HL-hyperprop P Q S*
then obtain *l p* **where** $p \in S\ (l,\ \textit{fst}\ p) \in P\ (l,\ \textit{snd}\ p) \notin Q$
using *HL-hyperprop-def* **by** *blast*
let *?S* = $\{p\}$
have $\textit{max-k}\ 1\ ?S \wedge (\forall\, S''.\ ?S \subseteq S'' \longrightarrow \neg\ \textit{HL-hyperprop}\ P\ Q\ S'')$
by (*metis* (*no-types*, *lifting*) *One-nat-def* ‹$(l,\ \textit{fst}\ p) \in P$› ‹$(l,\ \textit{snd}\ p) \notin Q$› *card.empty card.insert*
empty-iff finite.intros(*1*) *finite.intros*(*2*) *le-numeral-extra*(*4*) *max-k-def* *HL-hyperprop-def singletonI subsetD*)
then show $\exists\, S' \subseteq S.\ \textit{max-k}\ 1\ S' \wedge (\forall\, S''.\ S' \subseteq S'' \longrightarrow \neg\ \textit{HL-hyperprop}\ P\ Q\ S'')$
by (*meson* ‹$p \in S$› *empty-subsetI insert-subsetI*)
qed
ultimately show *?thesis*
by *blast*
qed

Proposition 2

theorem *encoding-HL*:
$\textit{HL}\ P\ C\ Q \longleftrightarrow (\textit{hyper-hoare-triple}\ (\textit{over-approx}\ P)\ C\ (\textit{over-approx}\ Q))$ (**is** $?A \longleftrightarrow ?B$)
proof (*rule iffI*)
show $?B \Longrightarrow ?A$
proof −
assume *asm0*: *?B*
show *?A*
proof (*rule HLI*)
fix $\sigma\ \sigma'\ l$
assume *asm1*: $(l,\ \sigma) \in P\ \langle C,\ \sigma\rangle \rightarrow \sigma'$
then have $\textit{over-approx}\ P\ \{(l,\ \sigma)\}$
by (*simp add*: *over-approx-def*)
then have $(\textit{over-approx}\ Q)\ (\textit{sem}\ C\ \{(l,\ \sigma)\})$
using *asm0 hyper-hoare-tripleE* **by** *auto*
then show $(l,\ \sigma') \in Q$
by (*simp add*: *asm1*(*2*) *in-mono in-sem over-approx-def*)
qed

qed
next
assume *r*: *?A*
show *?B*
proof (*rule hyper-hoare-tripleI*)
fix *S* **assume** *asm0*: *over-approx P S*
then have $S \subseteq P$
by (*simp add*: *over-approx-def*)
then have *sem C S* $\subseteq$ *sem C P*
by (*simp add*: *sem-monotonic*)
then have *sem C S* $\subseteq$ *Q*
using *r HL-def*[*of P C Q*]
by (*metis* (*no-types*, *lifting*) *fst-conv in-mono in-sem snd-conv subrelI*)
then show *over-approx Q* (*sem C S*)
by (*simp add*: *over-approx-def*)
qed
qed

lemma *entailment-order-hoare*:
assumes $P \subseteq P'$
shows *entails* (*over-approx P*) (*over-approx P'*)
by (*simp add*: *assms entails-def over-approx-def subset-trans*)

5.3 Cartesian Hoare Logic (CHL) [9]

Notation 3

definition *k-sem* **where**
k-sem C states states' $\longleftrightarrow$ ($\forall$ *i*. (*fst* (*states i*) = *fst* (*states' i*) $\wedge$ *single-sem C* (*snd* (*states i*)) (*snd* (*states' i*))))

lemma *k-semI*:
assumes $\bigwedge$*i*. (*fst* (*states i*) = *fst* (*states' i*) $\wedge$ *single-sem C* (*snd* (*states i*)) (*snd* (*states' i*)))
shows *k-sem C states states'*
by (*simp add*: *assms k-sem-def*)

lemma *k-semE*:
assumes *k-sem C states states'*
shows *fst* (*states i*) = *fst* (*states' i*) $\wedge$ *single-sem C* (*snd* (*states i*)) (*snd* (*states' i*))
using *assms k-sem-def* **by** *fastforce*

Definition 9

definition *CHL* **where**
CHL P C Q $\longleftrightarrow$ ($\forall$ *states*. *states* $\in$ *P* $\longrightarrow$ ($\forall$ *states'*. *k-sem C states states'* $\longrightarrow$ *states'* $\in$ *Q*))

lemma *CHLI*:
assumes $\bigwedge$*states states'*. *states* $\in$ *P* $\Longrightarrow$ *k-sem C states states'* $\Longrightarrow$ *states'* $\in$ *Q*

shows $CHL\ P\ C\ Q$
by (*simp add*: *assms CHL-def*)

lemma *CHLE*:
assumes $CHL\ P\ C\ Q$
and $states \in P$
and $k\text{-}sem\ C\ states\ states'$
shows $states' \in Q$
using *assms*(*1*) *assms*(*2*) *assms*(*3*) *CHL-def* **by** *fast*

definition *encode-CHL* **where**
$encode\text{-}CHL\ from\text{-}nat\ x\ P\ S \longleftrightarrow (\forall states.\ (\forall i.\ states\ i \in S \wedge fst\ (states\ i)\ x = from\text{-}nat\ i) \longrightarrow states \in P)$

lemma *encode-CHLI*:
assumes $\bigwedge states.\ (\forall i.\ states\ i \in S \wedge fst\ (states\ i)\ x = from\text{-}nat\ i) \Longrightarrow states \in P$
shows $encode\text{-}CHL\ from\text{-}nat\ x\ P\ S$
using *assms*(*1*) *encode-CHL-def* **by** *force*

lemma *encode-CHLE*:
assumes $encode\text{-}CHL\ from\text{-}nat\ x\ P\ S$
and $\bigwedge i.\ states\ i \in S$
and $\bigwedge i.\ fst\ (states\ i)\ x = from\text{-}nat\ i$
shows $states \in P$
by (*metis assms*(*1*) *assms*(*2*) *assms*(*3*) *encode-CHL-def*)

lemma *equal-change-lvar*:
assumes $fst\ \varphi\ x = y$
shows $\varphi = ((fst\ \varphi)(x := y),\ snd\ \varphi)$
using *assms* **by** *fastforce*

Proposition 3

theorem *encoding-CHL*:
assumes $not\text{-}free\text{-}var\text{-}of\ P\ x$
and $not\text{-}free\text{-}var\text{-}of\ Q\ x$
and $injective\ from\text{-}nat$
shows $CHL\ P\ C\ Q \longleftrightarrow\ \models \{encode\text{-}CHL\ from\text{-}nat\ x\ P\}\ C\ \{encode\text{-}CHL\ from\text{-}nat\ x\ Q\}$ (**is** $?A \longleftrightarrow ?B$)
proof
assume $?A$
show $?B$
proof (*rule hyper-hoare-tripleI*)
fix S **assume** $encode\text{-}CHL\ from\text{-}nat\ x\ P\ S$
then obtain *asm0*: $\bigwedge states\ states'.\ (\bigwedge i.\ states\ i \in S) \Longrightarrow (\bigwedge i.\ fst\ (states\ i)\ x = from\text{-}nat\ i) \Longrightarrow states \in P$
by (*simp add*: *encode-CHLE*)

show *encode-CHL from-nat x Q* (*sem C S*)
proof (*rule encode-CHLI*)
fix *states'*
assume *asm1*: $\forall$ *i*. *states' i* $\in$ *sem C S* $\wedge$ *fst* (*states' i*) *x* = *from-nat i*

let *?states* = λ*i*. (*fst* (*states' i*), *SOME* σ. (*fst* (*states' i*), σ) $\in$ *S* $\wedge$ *single-sem C* σ (*snd* (*states' i*)))

show *states'* $\in$ *Q*
using ‹*?A*›
proof (*rule CHLE*)
show *?states* $\in$ *P*
proof (*rule asm0*)
fix *i*
let *?*σ = *SOME* σ. ((*fst* (*states' i*), σ) $\in$ *S* $\wedge$ $\langle C, \sigma\rangle \rightarrow$ *snd* (*states' i*))
have *r*: (*fst* (*states' i*), *?*σ) $\in$ *S* $\wedge$ $\langle C$, *?*$\sigma\rangle \rightarrow$ *snd* (*states' i*)
using *someI-ex*[*of* $\lambda\sigma$. (*fst* (*states' i*), σ) $\in$ *S* $\wedge$ $\langle C, \sigma\rangle \rightarrow$ *snd* (*states' i*)] *asm1 in-sem* **by** *blast*
then show *?states i* $\in$ *S*
by *blast*
show *fst* (*?states i*) *x* = *from-nat i*
by (*simp add*: *asm1*)
qed
show *k-sem C ?states states'*
proof (*rule k-semI*)
fix *i*
let *?*σ = *SOME* σ. ((*fst* (*states' i*), σ) $\in$ *S* $\wedge$ $\langle C, \sigma\rangle \rightarrow$ *snd* (*states' i*))
have *r*: (*fst* (*states' i*), *?*σ) $\in$ *S* $\wedge$ $\langle C$, *?*$\sigma\rangle \rightarrow$ *snd* (*states' i*)
using *someI-ex*[*of* $\lambda\sigma$. (*fst* (*states' i*), σ) $\in$ *S* $\wedge$ $\langle C, \sigma\rangle \rightarrow$ *snd* (*states' i*)] *asm1 in-sem* **by** *blast*
then show *fst* (*?states i*) = *fst* (*states' i*) $\wedge$ $\langle C$, *snd* (*?states i*)$\rangle \rightarrow$ *snd* (*states' i*)
by *simp*
qed
qed
qed
qed
next
assume *asm0*: $\models$ {*encode-CHL from-nat x P*} *C* {*encode-CHL from-nat x Q*}

show *CHL P C Q*
proof (*rule CHLI*)
fix *states states'*
assume *asm1*: *states* $\in$ *P k-sem C states states'*

let *?states* = λ*i*. ((*fst* (*states i*))(*x* := *from-nat i*), *snd* (*states i*))
let *?states'* = λ*i*. ((*fst* (*states i*))(*x* := *from-nat i*), *snd* (*states' i*))
let *?S* = *range ?states*

```
  have encode-CHL from-nat x Q (sem C ?S)
    using asm0
  proof (rule hyper-hoare-tripleE)
    show encode-CHL from-nat x P ?S
    proof (rule encode-CHLI)
      fix f assume asm2: ∀ i. f i ∈ ?S ∧ fst (f i) x = from-nat i
      have f = ?states
      proof (rule ext)
        fix i
        obtain j where j-def: f i = ((fst (states j))(x := from-nat j), snd (states
j))
          using asm2 by fastforce
        then have from-nat j = from-nat i
          by (metis asm2 fst-conv fun-upd-same)
        then show f i = ((fst (states i))(x := from-nat i), snd (states i))
          by (metis j-def assms(3) injective-def)
      qed
      moreover have ?states ∈ P
        using assms(1)
      proof (rule not-free-var-ofE)
        show states ∈ P
          using asm1(1) by simp
        fix i
        show differ-only-by (fst (states i)) (fst ((fst (states i))(x := from-nat i),
snd (states i))) x
          by (simp add: differ-only-by-def)
        show snd (states i) = snd ((fst (states i))(x := from-nat i), snd (states i))
          by simp
      qed
      ultimately show f ∈ P
        by meson
    qed
  qed
  then have ?states' ∈ Q
  proof (rule encode-CHLE)
    fix i
    show fst ((fst (states i))(x := from-nat i), snd (states' i)) x = from-nat i
      by simp
    moreover have single-sem C (snd (?states i)) (snd (?states' i))
      using asm1(2) k-sem-def by fastforce
    ultimately show ((fst (states i))(x := from-nat i), snd (states' i)) ∈ sem C
?S
      using in-sem by fastforce
  qed
  show states' ∈ Q
    using assms(2)
  proof (rule not-free-var-ofE[of Q x])
    show ?states' ∈ Q
      by (simp add: ‹(λi. ((fst (states i))(x := from-nat i), snd (states' i))) ∈ Q›)
```

fix i **show** *differ-only-by* $(fst\ ((fst\ (states\ i))(x := from\text{-}nat\ i),\ snd\ (states'\ i)))\ (fst\ (states'\ i))\ x$
by (*metis asm1*(*2*) *diff-by-update fst-conv k-sem-def*)
qed (*auto*)
qed
qed

definition *CHL-hyperprop* **where**
$CHL\text{-}hyperprop\ P\ Q\ S \longleftrightarrow (\forall l\ p.\ (\forall i.\ p\ i \in S) \land (\lambda i.\ (l\ i,\ fst\ (p\ i))) \in P \longrightarrow (\lambda i.\ (l\ i,\ snd\ (p\ i))) \in Q)$

lemma *CHL-hyperpropI*:
assumes $\bigwedge l\ p.\ (\forall i.\ p\ i \in S) \land (\lambda i.\ (l\ i,\ fst\ (p\ i))) \in P \Longrightarrow (\lambda i.\ (l\ i,\ snd\ (p\ i))) \in Q$
shows *CHL-hyperprop P Q S*
by (*simp add*: *assms CHL-hyperprop-def*)

lemma *CHL-hyperpropE*:
assumes *CHL-hyperprop P Q S*
and $\bigwedge i.\ p\ i \in S$
and $(\lambda i.\ (l\ i,\ fst\ (p\ i))) \in P$
shows $(\lambda i.\ (l\ i,\ snd\ (p\ i))) \in Q$
using *assms*(*1*) *assms*(*2*) *assms*(*3*) *CHL-hyperprop-def* **by** *blast*

Proposition 10

theorem *CHL-hyperproperty*:
hypersat C (*CHL-hyperprop P Q*) $\longleftrightarrow$ *CHL P C Q* (**is** *?A* $\longleftrightarrow$ *?B*)
proof
assume *?A*
show *?B*
proof (*rule CHLI*)
fix *states states'*
assume *asm0*: $states \in P$ *k-sem C states states'*
let $?p = \lambda i.\ (snd\ (states\ i),\ snd\ (states'\ i))$
let $?l = \lambda i.\ fst\ (states\ i)$

have *range ?p* $\subseteq$ *set-of-traces C*
proof (*rule subsetI*)
fix x **assume** $x \in range\ ?p$
then obtain i **where** $x = (snd\ (states\ i),\ snd\ (states'\ i))$
by *blast*
then show $x \in$ *set-of-traces C*
by (*metis* (*mono-tags*, *lifting*) *CollectI asm0*(*2*) *k-sem-def set-of-traces-def*)
qed
have $(\lambda i.\ (?l\ i,\ snd\ (?p\ i))) \in Q$
proof (*rule CHL-hyperpropE*)
show *CHL-hyperprop P Q* (*range ?p*)
proof (*rule CHL-hyperpropI*)
fix $l\ p$ **assume** *asm1*: $(\forall i.\ p\ i \in range\ (\lambda i.\ (snd\ (states\ i),\ snd\ (states'\ i))))$

$\wedge$ ($\lambda i.$ (l i, *fst* (p i))) $\in P$
then show ($\lambda i.$ (l i, *snd* (p i))) $\in Q$
using *CHL-hyperprop-def*[*of P Q set-of-traces C*] ‹*hypersat C* (*CHL-hyperprop P Q*)›
‹*range* ($\lambda i.$ (*snd* (*states* i), *snd* (*states'* i))) $\subseteq$ *set-of-traces C*› *hypersat-def subset-iff*
by *blast*
qed
show ($\lambda i.$ (*fst* (*states* i), *fst* (*snd* (*states* i), *snd* (*states'* i)))) $\in P$
by (*simp add*: *asm0*(*1*))
fix i **show** (*snd* (*states* i), *snd* (*states'* i)) $\in$ *range* ($\lambda i.$ (*snd* (*states* i), *snd* (*states'* i)))
by *blast*
qed
moreover have *states'* = ($\lambda i.$ (*?l* i, *snd* (*?p* i)))
proof (*rule ext*)
fix i **show** *states'* i = (*fst* (*states* i), *snd* (*snd* (*states* i), *snd* (*states'* i)))
by (*metis asm0*(*2*) *k-sem-def prod.exhaust-sel sndI*)
qed
ultimately show *states'* $\in Q$
by *auto*
qed
next
assume *asm0*: *CHL P C Q*
have *CHL-hyperprop P Q* (*set-of-traces C*)
proof (*rule CHL-hyperpropI*)
fix *l p* **assume** *asm1*: ($\forall i.$ p i $\in$ *set-of-traces C*) $\wedge$ ($\lambda i.$ (l i, *fst* (p i))) $\in P$

show ($\lambda i.$ (l i, *snd* (p i))) $\in Q$
using *asm0*
proof (*rule CHLE*)
show ($\lambda i.$ (l i, *fst* (p i))) $\in P$
by (*simp add*: *asm1*)
show *k-sem C* ($\lambda i.$ (l i, *fst* (p i))) ($\lambda i.$ (l i, *snd* (p i)))
proof (*rule k-semI*)
fix i **show** *fst* (l i, *fst* (p i)) = *fst* (l i, *snd* (p i)) $\wedge$ $\langle C$, *snd* (l i, *fst* (p i))$\rangle$ $\rightarrow$ *snd* (l i, *snd* (p i))
using *asm1 in-set-of-traces* **by** *fastforce*
qed
qed
qed
then show *hypersat C* (*CHL-hyperprop P Q*)
by (*simp add*: *hypersat-def*)
qed

theorem *k-hypersafety-HL-hyperprop*:
fixes P :: ('i $\Rightarrow$ ('*lvar*, '*lval*, '*pvar*, '*pval*) *state*) *set*

assumes *finite* $(UNIV :: 'i\ set)$
and *card* $(UNIV :: 'i\ set) = k$
shows *k-hypersafety* k (*CHL-hyperprop* $P\ Q$)
proof (*rule k-hypersafetyI*)
fix S
assume $\neg$ *CHL-hyperprop* $P\ Q\ S$
then obtain $l\ p$ **where** *asm0*: $\forall i.\ p\ i \in S\ (\lambda i.\ (l\ i,\ fst\ (p\ i))) \in P$
$(\lambda i.\ (l\ i,\ snd\ (p\ i))) \notin Q$
using *CHL-hyperprop-def* **by** *blast*
let $?S = range\ p$
have *max-k* $k\ ?S$
by (*metis assms*(*1*) *assms*(*2*) *card-image-le finite-imageI max-k-def*)
moreover have $\bigwedge S''.\ ?S \subseteq S'' \Longrightarrow \neg$ *CHL-hyperprop* $P\ Q\ S''$
by (*meson asm0*(*2*) *asm0*(*3*) *CHL-hyperprop-def range-subsetD*)
ultimately show $\exists S' \subseteq S.$ *max-k* $k\ S' \wedge (\forall S''.\ S' \subseteq S'' \longrightarrow \neg$ *CHL-hyperprop* $P\ Q\ S'')$
by (*meson asm0*(*1*) *image-subsetI*)
qed

5.4 Incorrectness Logic [8] or Reverse Hoare Logic [3] (IL))

Definition 11

definition *IL* **where**
$IL\ P\ C\ Q \longleftrightarrow Q \subseteq sem\ C\ P$

lemma *equiv-def-incorrectness*:
$IL\ P\ C\ Q \longleftrightarrow (\forall l\ \sigma'.\ (l, \sigma') \in Q \longrightarrow (\exists \sigma.\ (l, \sigma) \in P \wedge \langle C, \sigma\rangle \rightarrow \sigma'))$
by (*simp add*: *in-sem IL-def subset-iff*)

definition *IL-hyperprop* **where**
IL-hyperprop $P\ Q\ S \longleftrightarrow (\forall l\ \sigma'.\ (l, \sigma') \in Q \longrightarrow (\exists \sigma.\ (l, \sigma) \in P \wedge (\sigma, \sigma') \in S))$

lemma *IL-hyperpropI*:
assumes $\bigwedge l\ \sigma'.\ (l, \sigma') \in Q \Longrightarrow (\exists \sigma.\ (l, \sigma) \in P \wedge (\sigma, \sigma') \in S)$
shows *IL-hyperprop* $P\ Q\ S$
by (*simp add*: *assms IL-hyperprop-def*)

Proposition 11

lemma *IL-expresses-hyperproperties*:
$IL\ P\ C\ Q \longleftrightarrow$ *IL-hyperprop* $P\ Q$ (*set-of-traces* C) (**is** $?A \longleftrightarrow ?B$)
proof
assume $?A$
show $?B$
proof (*rule IL-hyperpropI*)
fix $l\ \sigma'$ **assume** *asm0*: $(l, \sigma') \in Q$
then obtain σ **where** $(l, \sigma) \in P$ *single-sem* $C\ \sigma\ \sigma'$
using ‹$IL\ P\ C\ Q$› *equiv-def-incorrectness* **by** *blast*
then show $\exists \sigma.\ (l, \sigma) \in P \wedge (\sigma, \sigma') \in$ *set-of-traces* C

using *set-of-traces-def* **by** *auto*
qed
next
assume *?B*
have $Q \subseteq$ *sem C P*
proof (*rule subsetPairI*)
fix $l\ \sigma'$ **assume** $(l, \sigma') \in Q$
then obtain σ **where** $(\sigma, \sigma') \in$ *set-of-traces C* $(l, \sigma) \in P$
by (*meson* ‹*IL-hyperprop P Q* (*set-of-traces C*)› *IL-hyperprop-def*)
then show $(l, \sigma') \in$ *sem C P*
using *in-set-of-traces-then-in-sem* **by** *blast*
qed
then show *?A*
by (*simp add*: *IL-def*)
qed

lemma *IL-consequence*:
assumes *IL P C Q*
and $(l, \sigma') \in Q$
shows $\exists \sigma.\ (l, \sigma) \in P \wedge$ *single-sem C* $\sigma\ \sigma'$
using *assms*(*1*) *assms*(*2*) *equiv-def-incorrectness* **by** *blast*

Proposition 4

theorem *encoding-IL*:
IL P C Q $\longleftrightarrow$ (*hyper-hoare-triple* (*under-approx P*) *C* (*under-approx Q*)) (**is** *?A* $\longleftrightarrow$ *?B*)
proof (*rule iffI*)
show *?B* $\Longrightarrow$ *?A*
proof −
assume *?B*
then have *under-approx Q* (*sem C P*)
by (*simp add*: *hyper-hoare-triple-def under-approx-def*)
then show *?A*
by (*simp add*: *IL-def under-approx-def*)
qed
assume *?A*
then show *?B*
by (*simp add*: *hyper-hoare-triple-def sem-monotonic IL-def under-approx-def subset-trans*)
qed

lemma *entailment-order-reverse-hoare*:
assumes $P \subseteq P'$
shows *entails* (*under-approx* P') (*under-approx P*)
by (*simp add*: *assms dual-order.trans entails-def under-approx-def*)

definition *incorrectify* **where**
incorrectify p = *under-approx* $\{ \sigma\ |\sigma.\ p\ \sigma\}$

lemma *incorrectifyI*:
  assumes $\bigwedge\sigma.\ p\ \sigma \Longrightarrow \sigma \in S$
  shows *incorrectify p S*
  by (*metis assms incorrectify-def mem-Collect-eq subsetI under-approx-def*)

lemma *incorrectifyE*:
  assumes *incorrectify p S*
      and $p\ \sigma$
    shows $\sigma \in S$
  by (*metis CollectI assms*(*1*) *assms*(*2*) *in-mono incorrectify-def under-approx-def*)

lemma *simple-while-incorrectness*:
  assumes $\bigwedge n.$ *hyper-hoare-triple* (*incorrectify* (*p n*)) *C* (*incorrectify* (*p* (*Suc n*)))
  shows *hyper-hoare-triple* (*incorrectify* (*p 0*)) (*While C*) (*incorrectify* ($\lambda\sigma.\ \exists n.\ p$ $n\ \sigma$))
proof (*rule consequence-rule*)
  show *entails* (*incorrectify* (*p 0*)) (*incorrectify* (*p 0*))
    by (*simp add*: *entailsI*)
  show *hyper-hoare-triple* (*incorrectify* (*p 0*)) (*While C*) (*natural-partition* ($\lambda n.$ *incorrectify* (*p n*)))
    by (*meson assms while-rule*)

  have *entails* (*incorrectify* ($\lambda\sigma.\ \exists n.\ p\ n\ \sigma$)) (*natural-partition* ($\lambda n.$ *incorrectify* (*p* *n*)))
  proof (*rule entailsI*)
    fix *S* **assume** *asm0*: *incorrectify* ($\lambda\sigma.\ \exists n.\ p\ n\ \sigma$) *S*
    then have *under-approx* $\{\ \sigma\ |\sigma\ n.\ p\ n\ \sigma\}$ *S*
      by (*metis incorrectify-def*)
    let $?F = \lambda n.\ S$
    show *natural-partition* ($\lambda n.$ *incorrectify* (*p n*)) *S*
    proof (*rule natural-partitionI*)
      show $\bigwedge n.$ *incorrectify* (*p n*) (*?F n*)
        by (*metis asm0 incorrectifyE incorrectifyI*)
      show $S = \bigcup$ (*range ?F*)
        by *simp*
    qed
  qed

  show *entails* (*natural-partition* ($\lambda n.$ *incorrectify* (*p n*))) (*incorrectify* ($\lambda\sigma.\ \exists n.\ p$ $n\ \sigma$))
  proof (*rule entailsI*)
    fix *S* **assume** *asm0*: *natural-partition* ($\lambda n.$ *incorrectify* (*p n*)) *S*
    then obtain *F* **where** $S = (\bigcup n.\ F\ n)$ $\bigwedge n.$ *incorrectify* (*p n*) (*F n*)
      using *natural-partitionE* **by** *blast*
    show *incorrectify* ($\lambda\sigma.\ \exists n.\ p\ n\ \sigma$) *S*
    proof (*rule incorrectifyI*)
      fix σ **assume** $\exists n.\ p\ n\ \sigma$

then obtain n **where** $p\ n\ \sigma$
by *blast*
then have $\sigma \in F\ n$
by (*meson* ‹$\bigwedge n$. *incorrectify* ($p\ n$) ($F\ n$)› *incorrectifyE*)
then show $\sigma \in S$
using ‹$S = \bigcup$ (*range* F)› **by** *blast*
qed
qed
qed

definition *sat-for-l* **where**
sat-for-l $l\ P \longleftrightarrow (\exists \sigma.\ (l, \sigma) \in P)$

theorem *incorrectness-hyperliveness*:
assumes $\bigwedge l$. *sat-for-l* $l\ Q \Longrightarrow$ *sat-for-l* $l\ P$
shows *hyperliveness* (*IL-hyperprop* $P\ Q$)
proof (*rule hyperlivenessI*)
fix S
let $?S = S \cup \{(\sigma, \sigma') \mid \sigma\ \sigma'\ l.\ (l, \sigma') \in Q \land (l, \sigma) \in P\}$
have *IL-hyperprop* $P\ Q\ ?S$
proof (*rule IL-hyperpropI*)
fix $l\ \sigma'$
assume *asm0*: $(l, \sigma') \in Q$
then obtain σ **where** $(l, \sigma) \in P$
by (*meson assms sat-for-l-def*)
then show $\exists \sigma.\ (l, \sigma) \in P \land (\sigma, \sigma') \in ?S$
using *asm0* **by** *auto*
qed
then show $\exists S'.\ S \subseteq S' \land$ *IL-hyperprop* $P\ Q\ S'$
by *auto*
qed

5.5 Relational Incorrectness Logic [7] (RIL)

Definition 11

definition *RIL* **where**
RIL $P\ C\ Q \longleftrightarrow (\forall\ states' \in Q.\ \exists\ states \in P.$ *k-sem* $C\ states\ states')$

lemma *RILI*:
assumes $\bigwedge states'.\ states' \in Q \Longrightarrow (\exists\ states \in P.$ *k-sem* $C\ states\ states')$
shows *RIL* $P\ C\ Q$
by (*simp add*: *assms RIL-def*)

lemma *RILE*:
assumes *RIL* $P\ C\ Q$
and $states' \in Q$
shows $\exists\ states \in P.$ *k-sem* $C\ states\ states'$
using *assms*(1) *assms*(2) *RIL-def* **by** *blast*

definition *RIL-hyperprop* **where**
$RIL\text{-}hyperprop\ P\ Q\ S \longleftrightarrow (\forall\, l\ states'.\ (\lambda i.\ (l\ i,\ states'\ i)) \in Q$
$\longrightarrow (\exists\, states.\ (\lambda i.\ (l\ i,\ states\ i)) \in P \wedge (\forall\, i.\ (states\ i,\ states'\ i) \in S)))$

lemma *RIL-hyperpropI*:
assumes $\bigwedge l\ states'.\ (\lambda i.\ (l\ i,\ states'\ i)) \in Q \Longrightarrow (\exists\, states.\ (\lambda i.\ (l\ i,\ states\ i)) \in P \wedge (\forall\, i.\ (states\ i,\ states'\ i) \in S))$
shows *RIL-hyperprop P Q S*
by (*simp add*: *assms RIL-hyperprop-def*)

lemma *RIL-hyperpropE*:
assumes *RIL-hyperprop P Q S*
and $(\lambda i.\ (l\ i,\ states'\ i)) \in Q$
shows $\exists\, states.\ (\lambda i.\ (l\ i,\ states\ i)) \in P \wedge (\forall\, i.\ (states\ i,\ states'\ i) \in S)$
using *assms*(*1*) *assms*(*2*) *RIL-hyperprop-def* **by** *blast*

lemma *useful*:
$states' = (\lambda i.\ ((fst \circ states')\ i,\ (snd \circ states')\ i))$
proof (*rule ext*)
fix *i* **show** $states'\ i = ((fst \circ states')\ i,\ (snd \circ states')\ i)$
by *auto*
qed

Proposition 12

theorem *RIL-expresses-hyperproperties*:
$hypersat\ C\ (RIL\text{-}hyperprop\ P\ Q) \longleftrightarrow RIL\ P\ C\ Q$ (**is** $?A \longleftrightarrow ?B$)
proof
assume *?A*
show *?B*
proof (*rule RILI*)
fix *states'* **assume** *asm0*: $states' \in Q$
then obtain *states* **where** *asm0*: $(\lambda i.\ ((fst \circ states')\ i,\ states\ i)) \in P \wedge (\forall\, i.\ (states\ i,\ (snd \circ states')\ i) \in set\text{-}of\text{-}traces\ C)$
using *RIL-hyperpropE*[*of P Q set-of-traces C fst* ∘ *states' snd* ∘ *states'*] ‹*?A*›
using *hypersat-def* **by** *auto*

moreover have $k\text{-}sem\ C\ (\lambda i.\ ((fst \circ states')\ i,\ states\ i))\ states'$
proof (*rule k-semI*)
fix *i*
have $\langle C,\ snd\ ((fst \circ states')\ i,\ states\ i)\rangle \rightarrow snd\ (states'\ i)$
using *calculation set-of-traces-def* **by** *auto*
then show $fst\ ((fst \circ states')\ i,\ states\ i) = fst\ (states'\ i) \wedge \langle C,\ snd\ ((fst \circ states')\ i,\ states\ i)\rangle \rightarrow snd\ (states'\ i)$
by *simp*
qed
ultimately show $\exists\, states{\in}P.\ k\text{-}sem\ C\ states\ states'$
by *fast*

qed
next
assume *?B*
have *RIL-hyperprop P Q* (*set-of-traces C*)
proof (*rule RIL-hyperpropI*)
fix *l states'*
assume *asm0*: $(\lambda i.\ (l\ i,\ states'\ i)) \in Q$
then obtain *states* **where** $states \in P$ *k-sem C states* $(\lambda i.\ (l\ i,\ states'\ i))$
using ‹*RIL P C Q*› *RILE* **by** *blast*
moreover have $(\lambda i.\ (l\ i,\ (snd \circ states)\ i)) = states$
proof (*rule ext*)
fix *i* **show** $(l\ i,\ (snd \circ states)\ i) = states\ i$
by (*metis calculation*(*2*) *comp-apply fst-conv k-sem-def surjective-pairing*)
qed
moreover have $\bigwedge i.\ ((snd \circ states)\ i,\ states'\ i) \in$ *set-of-traces C*
by (*metis* (*mono-tags*, *lifting*) *calculation*(*2*) *comp-apply in-set-of-traces k-sem-def snd-conv*)
ultimately show $\exists\, states.\ (\lambda i.\ (l\ i,\ states\ i)) \in P \wedge (\forall\, i.\ (states\ i,\ states'\ i) \in$ *set-of-traces C*)
by *force*
qed
then show *?A*
using *hypersat-def* **by** *blast*
qed

definition *k-sat-for-l* **where**
k-sat-for-l l P $\longleftrightarrow (\exists\, \sigma.\ (\lambda i.\ (l\ i,\ \sigma\ i)) \in P)$

theorem *RIL-hyperprop-hyperlive*:
assumes $\bigwedge l.$ *k-sat-for-l l Q* $\Longrightarrow$ *k-sat-for-l l P*
shows *hyperliveness* (*RIL-hyperprop P Q*)
proof (*rule hyperlivenessI*)
fix *S*
have *RIL-hyperprop P Q UNIV*
by (*meson assms RIL-hyperpropI iso-tuple-UNIV-I k-sat-for-l-def*)
then show $\exists\, S'.\ S \subseteq S' \wedge$ *RIL-hyperprop P Q S'*
by *blast*
qed

definition *strong-pre-insec* **where**
strong-pre-insec from-nat x c P S $\longleftrightarrow (\forall\, states \in P.$
$(\forall\, i.\ fst\ (states\ i)\ x = from\text{-}nat\ i) \longrightarrow (\exists\, r.\ \forall\, i.\ ((fst\ (states\ i))(c := r),\ snd\ (states\ i)) \in S)) \wedge$
$(\forall\, states.\ (\forall\, i.\ states\ i \in S) \wedge (\forall\, i.\ fst\ (states\ i)\ x = from\text{-}nat\ i) \wedge$
$(\forall\, i\ j.\ fst\ (states\ i)\ c = fst\ (states\ j)\ c) \longrightarrow states \in P)$

lemma *strong-pre-insecI*:
assumes $\bigwedge states.\ states \in P \Longrightarrow (\forall\, i.\ fst\ (states\ i)\ x = from\text{-}nat\ i)$

$\Longrightarrow (\exists\, r.\ \forall\, i.\ ((\mathit{fst}\ (\mathit{states}\ i))(c := r),\ \mathit{snd}\ (\mathit{states}\ i)) \in S)$
and $\bigwedge \mathit{states}.\ (\forall\, i.\ \mathit{states}\ i \in S) \Longrightarrow (\forall\, i.\ \mathit{fst}\ (\mathit{states}\ i)\ x = \mathit{from\text{-}nat}\ i) \Longrightarrow$
$(\forall\, i\ j.\ \mathit{fst}\ (\mathit{states}\ i)\ c = \mathit{fst}\ (\mathit{states}\ j)\ c) \Longrightarrow \mathit{states} \in P$
shows *strong-pre-insec from-nat x c P S*
by (*simp add*: *assms*(*1*) *assms*(*2*) *strong-pre-insec-def*)

lemma *strong-pre-insecE*:
assumes *strong-pre-insec from-nat x c P S*
and $\bigwedge i.\ \mathit{states}\ i \in S$
and $\bigwedge i.\ \mathit{fst}\ (\mathit{states}\ i)\ x = \mathit{from\text{-}nat}\ i$
and $\bigwedge i\ j.\ \mathit{fst}\ (\mathit{states}\ i)\ c = \mathit{fst}\ (\mathit{states}\ j)\ c$
shows $\mathit{states} \in P$
by (*meson assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *strong-pre-insec-def*)

definition *pre-insec* **where**
$\mathit{pre\text{-}insec}\ \mathit{from\text{-}nat}\ x\ c\ P\ S \longleftrightarrow (\forall\, \mathit{states} \in P.$
$(\forall\, i.\ \mathit{fst}\ (\mathit{states}\ i)\ x = \mathit{from\text{-}nat}\ i)$
$\longrightarrow (\exists\, r.\ \forall\, i.\ ((\mathit{fst}\ (\mathit{states}\ i))(c := r),\ \mathit{snd}\ (\mathit{states}\ i)) \in S))$

lemma *pre-insecI*:
assumes $\bigwedge \mathit{states}.\ \mathit{states} \in P \Longrightarrow (\forall\, i.\ \mathit{fst}\ (\mathit{states}\ i)\ x = \mathit{from\text{-}nat}\ i)$
$\Longrightarrow (\exists\, r.\ \forall\, i.\ ((\mathit{fst}\ (\mathit{states}\ i))(c := r),\ \mathit{snd}\ (\mathit{states}\ i)) \in S)$
shows *pre-insec from-nat x c P S*
by (*simp add*: *assms*(*1*) *pre-insec-def*)

lemma *strong-pre-implies-pre*:
assumes *strong-pre-insec from-nat x c P S*
shows *pre-insec from-nat x c P S*
by (*meson assms pre-insecI strong-pre-insec-def*)

lemma *pre-insecE*:
assumes *pre-insec from-nat x c P S*
and $\mathit{states} \in P$
and $\bigwedge i.\ \mathit{fst}\ (\mathit{states}\ i)\ x = \mathit{from\text{-}nat}\ i$
shows $\exists\, r.\ \forall\, i.\ ((\mathit{fst}\ (\mathit{states}\ i))(c := r),\ \mathit{snd}\ (\mathit{states}\ i)) \in S$
by (*meson assms*(*1*) *assms*(*2*) *assms*(*3*) *pre-insec-def*)

definition *post-insec* **where**
$\mathit{post\text{-}insec}\ \mathit{from\text{-}nat}\ x\ c\ Q\ S \longleftrightarrow (\forall\, \mathit{states} \in Q.\ (\forall\, i.\ \mathit{fst}\ (\mathit{states}\ i)\ x = \mathit{from\text{-}nat}$
$i)$
$\longrightarrow (\exists\, r.\ (\forall\, i.\ ((\mathit{fst}\ (\mathit{states}\ i))(c := r),\ \mathit{snd}\ (\mathit{states}\ i)) \in S)))$

lemma *post-insecE*:
assumes *post-insec from-nat x c Q S*
and $\mathit{states} \in Q$

and $\bigwedge i.\ \mathit{fst}\ (\mathit{states}\ i)\ x = \mathit{from\text{-}nat}\ i$
shows $\exists\, r.\ (\forall\, i.\ ((\mathit{fst}\ (\mathit{states}\ i))(c := r),\ \mathit{snd}\ (\mathit{states}\ i)) \in S)$
by (*meson assms(1) assms(2) assms(3) post-insec-def*)

lemma *post-insecI*:
assumes $\bigwedge \mathit{states}.\ \mathit{states} \in Q \Longrightarrow (\forall\, i.\ \mathit{fst}\ (\mathit{states}\ i)\ x = \mathit{from\text{-}nat}\ i)$
$\Longrightarrow (\exists\, r.\ (\forall\, i.\ ((\mathit{fst}\ (\mathit{states}\ i))(c := r),\ \mathit{snd}\ (\mathit{states}\ i)) \in S))$
shows *post-insec from-nat x c Q S*
by (*simp add*: *assms post-insec-def*)

lemma *same-pre-post*:
pre-insec from-nat x c Q S $\longleftrightarrow$ *post-insec from-nat x c Q S*
by (*simp add*: *post-insec-def pre-insec-def*)

theorem *can-be-sat*:
fixes $x :: {'}\mathit{lvar}$
assumes $\bigwedge l\ l'\ \sigma.\ (\lambda i.\ (l\ i,\ \sigma\ i)) \in P \longleftrightarrow (\lambda i.\ (l'\ i,\ \sigma\ i)) \in P$
and $\mathit{injective}\ (\mathit{indexify} :: (({'}a \Rightarrow ({'}\mathit{pvar} \Rightarrow {'}\mathit{pval})) \Rightarrow {'}\mathit{lval}))$
and $x \neq c$
and *injective from-nat*
shows $\mathit{sat}\ (\mathit{strong\text{-}pre\text{-}insec}\ \mathit{from\text{-}nat}\ x\ c\ (P :: ({'}a \Rightarrow ({'}\mathit{lvar} \Rightarrow {'}\mathit{lval}) \times ({'}\mathit{pvar} \Rightarrow {'}\mathit{pval}))\ \mathit{set}))$
proof −

let $?S = \bigcup \mathit{states}{\in}P.\ \{\ (((\mathit{fst}\ (\mathit{states}\ i))(x := \mathit{from\text{-}nat}\ i))(c := \mathit{indexify}\ (\lambda i.\ \mathit{snd}\ (\mathit{states}\ i))),\ \mathit{snd}\ (\mathit{states}\ i))\ |i.\ \mathit{True}\}$

have *strong-pre-insec from-nat x c P ?S*
proof (*rule strong-pre-insecI*)
fix *states*
assume *asm0*: $\mathit{states} \in P\ \forall\, i.\ \mathit{fst}\ (\mathit{states}\ i)\ x = \mathit{from\text{-}nat}\ i$
define *r* **where** $r = \mathit{indexify}\ (\lambda i.\ \mathit{snd}\ (\mathit{states}\ i))$
have $\bigwedge i.\ ((\mathit{fst}\ (\mathit{states}\ i))(c := r),\ \mathit{snd}\ (\mathit{states}\ i)) \in \{\ (((\mathit{fst}\ (\mathit{states}\ i))(x := \mathit{from\text{-}nat}\ i))(c := \mathit{indexify}\ (\lambda i.\ \mathit{snd}\ (\mathit{states}\ i))),\ \mathit{snd}\ (\mathit{states}\ i))\ |i.\ \mathit{True}\}$
proof −
fix *i*
show $((\mathit{fst}\ (\mathit{states}\ i))(c := r),\ \mathit{snd}\ (\mathit{states}\ i)) \in \{\ (((\mathit{fst}\ (\mathit{states}\ i))(x := \mathit{from\text{-}nat}\ i))(c := \mathit{indexify}\ (\lambda i.\ \mathit{snd}\ (\mathit{states}\ i))),\ \mathit{snd}\ (\mathit{states}\ i))\ |i.\ \mathit{True}\}$
using *asm0(2) r-def* **by** *fastforce*
qed
then show $\exists\, r.\ \forall\, i.\ ((\mathit{fst}\ (\mathit{states}\ i))(c := r),\ \mathit{snd}\ (\mathit{states}\ i)) \in\ ?S$
by (*meson UN-I asm0(1)*)
next
fix *states*

assume *asm0*: $\forall\, i.\ \mathit{states}\ i \in\ ?S\ \forall\, i.\ \mathit{fst}\ (\mathit{states}\ i)\ x = \mathit{from\text{-}nat}\ i\ \forall\, i\ j.\ \mathit{fst}\ (\mathit{states}\ i)\ c = \mathit{fst}\ (\mathit{states}\ j)\ c$

let $?P = \lambda i\ \mathit{states}'.\ \mathit{states}' \in P \wedge \mathit{states}\ i \in \{\ (((\mathit{fst}\ (\mathit{states}'\ i))(x := \mathit{from\text{-}nat}$

i))(*c* := *indexify* (λ*i*. *snd* (*states′ i*))), *snd* (*states′ i*)) |*i*. *True*}

let *?states* = λ*i*. *SOME states′*. *?P i states′*
have *r*: ⋀*i*. *?P i* (*?states i*)
proof −
  fix *i*
  show *?P i* (*?states i*)
  proof (*rule someI-ex*[*of ?P i*])
    show ∃ *states′*. *states′* ∈ *P* ∧ *states i* ∈ { (((*fst* (*states′ i*))(*x* := *from-nat i*))(*c* := *indexify* (λ*i*. *snd* (*states′ i*))), *snd* (*states′ i*)) |*i*. *True*}
      using *asm0*(*1*) **by** *fastforce*
  qed
qed
moreover have *rr*: ⋀*i*. *fst* (*states i*) *c* = *indexify* (λ*j*. *snd* (*?states i j*)) ∧ *snd* (*?states i i*) = *snd* (*states i*)
proof −
  fix *i*
  obtain *j* **where** *j-def*: *states i* = (((*fst* ((*?states i*) *j*))(*x* := *from-nat j*))(*c* := *indexify* (λ*k*. *snd* ((*?states i*) *k*))), *snd* ((*?states i*) *j*))
    using *r*[*of i*] **by** *blast*
  then have *r1*: *snd* (*?states i j*) = *snd* (*states i*)
    by (*metis* (*no-types*, *lifting*) *snd-conv*)
  then have *from-nat i* = *from-nat j*
    by (*metis* (*no-types*, *lifting*) *j-def asm0*(*2*) *assms*(*3*) *fst-conv fun-upd-same fun-upd-twist*)
  then have *i* = *j*
    by (*meson assms*(*4*) *injective-def*)
  show *fst* (*states i*) *c* = *indexify* (λ*j*. *snd* (*?states i j*)) ∧ *snd* (*?states i i*) = *snd* (*states i*)
  proof
    show *fst* (*states i*) *c* = *indexify* (λ*j*. *snd* (*?states i j*))
      by (*metis* (*no-types*, *lifting*) *j-def fst-conv fun-upd-same*)
    show *snd* (*?states i i*) = *snd* (*states i*)
      using ‹*i* = *j*› *r1* **by** *blast*
  qed
qed
moreover have *r0*: ⋀*i j*. (λ*n*. *snd* (*?states i n*)) = (λ*n*. *snd* (*?states j n*))
proof −
  fix *i j*
  have *indexify* (λ*n*. *snd* (*?states i n*)) = *indexify* (λ*n*. *snd* (*?states j n*))
    using *asm0*(*3*) *rr* **by** *fastforce*
  then show (λ*n*. *snd* (*?states i n*)) = (λ*n*. *snd* (*?states j n*))
    by (*meson assms*(*2*) *injective-def*)
qed
obtain *k* :: *′a* **where** *True* **by** *blast*
then have *?states k* ∈ *P*
  using *UN-iff*[*of* - λ*states*. {((*fst* (*states i*))(*x* := *from-nat i*, *c* := *indexify* (λ*i*. *snd* (*states i*))), *snd* (*states i*)) |*i*. *True*} *P*]
    asm0(*1*) *someI-ex*[*of* λ*y*. *y* ∈ *P* ∧ *states k* ∈ {((*fst* (*y i*))(*x* := *from-nat i*,

$c := indexify\ (\lambda i.\ snd\ (y\ i))),\ snd\ (y\ i))\ |i.\ True\}]$
    by *fast*
  moreover have $\bigwedge i.\ snd\ (?states\ k\ i) = snd\ (states\ i)$
  proof −
    fix i
    have $snd\ (?states\ i\ i) = snd\ (states\ i)$
      using *rr* **by** *blast*
    moreover have $(\lambda n.\ snd\ (?states\ i\ n))\ i = (\lambda n.\ snd\ (?states\ k\ n))\ i$
      by (*metis r0*)
    ultimately show $snd\ (?states\ k\ i) = snd\ (states\ i)$
      by *auto*
  qed
  moreover have $(\lambda i.\ ((\lambda i.\ fst\ (?states\ k\ i))\ i,\ (\lambda i.\ snd\ (states\ i))\ i)) \in P \longleftrightarrow (\lambda i.\ ((\lambda i.\ fst\ (states\ i))\ i,\ (\lambda i.\ snd\ (states\ i))\ i)) \in P$
    using *assms*(*1*) **by** *fast*
  moreover have $(\lambda i.\ ((\lambda i.\ fst\ (states\ i))\ i,\ (\lambda i.\ snd\ (states\ i))\ i)) = states$
  proof (*rule ext*)
    fix i **show** $(fst\ (states\ i),\ snd\ (states\ i)) = states\ i$
      by *simp*
  qed
  moreover have $(\lambda i.\ ((\lambda i.\ fst\ (?states\ k\ i))\ i,\ (\lambda i.\ snd\ (states\ i))\ i)) = ?states\ k$
  proof (*rule ext*)
    fix i **show** $(\lambda i.\ ((\lambda i.\ fst\ (?states\ k\ i))\ i,\ (\lambda i.\ snd\ (states\ i))\ i))\ i = ?states\ k\ i$
      by (*metis* (*no-types*, *lifting*) *calculation*(*4*) *prod.exhaust-sel*)
  qed
  ultimately show $states \in P$ **by** *argo*
qed
then show $sat\ (strong\text{-}pre\text{-}insec\ from\text{-}nat\ x\ c\ P)$
  by (*meson sat-def*)
qed

theorem *encode-insec*:
  assumes *injective from-nat*
    and $sat\ (strong\text{-}pre\text{-}insec\ from\text{-}nat\ x\ c\ (P :: ('a \Rightarrow ('lvar \Rightarrow 'lval) \times ('pvar \Rightarrow 'pval))\ set))$
    and $not\text{-}free\text{-}var\text{-}of\ P\ x \wedge not\text{-}free\text{-}var\text{-}of\ P\ c$
    and $not\text{-}free\text{-}var\text{-}of\ Q\ x \wedge not\text{-}free\text{-}var\text{-}of\ Q\ c$
    and $c \neq x$

  shows $RIL\ P\ C\ Q \longleftrightarrow \models \{pre\text{-}insec\ from\text{-}nat\ x\ c\ P\}\ C\ \{post\text{-}insec\ from\text{-}nat\ x\ c\ Q\}$ (**is** $?A \longleftrightarrow ?B$)
proof
  assume $?A$
  show $?B$
  proof (*rule hyper-hoare-tripleI*)
    fix S **assume** *asm0*: $pre\text{-}insec\ from\text{-}nat\ x\ c\ P\ S$

    show $post\text{-}insec\ from\text{-}nat\ x\ c\ Q\ (sem\ C\ S)$

proof (*rule post-insecI*)
fix *states'* **assume** *asm1*: *states'* $\in$ *Q* $\forall$ *i*. *fst* (*states' i*) *x* = *from-nat i*
then obtain *states* **where** *states* $\in$ *P k-sem C states states'*
using ‹*RIL P C Q*› *RILE* **by** *blast*
then obtain *r* **where** *asm2*: $\bigwedge$*i*. ((*fst* (*states i*))(*c* := *r*), *snd* (*states i*)) $\in$ *S*
using *pre-insecE*[*of from-nat x c P S states*]
by (*metis asm0 asm1*(*2*) *k-sem-def*)
then show $\exists$ *r*. $\forall$ *i*. ((*fst* (*states' i*))(*c* := *r*), *snd* (*states' i*)) $\in$ *sem C S*
by (*metis* (*mono-tags*, *opaque-lifting*) ‹*k-sem C states states'*› *k-sem-def single-step-then-in-sem*)
qed
qed
next
assume *asm0*: *?B*
show *?A*
proof (*rule RILI*)
fix *states'* **assume** *asm1*: *states'* $\in$ *Q*
obtain *S* **where** *asm2*: *strong-pre-insec from-nat x c P S*
by (*meson assms*(*2*) *sat-def*)
then have *asm3*: *post-insec from-nat x c Q* (*sem C S*)
by (*meson asm0 hyper-hoare-tripleE strong-pre-implies-pre*)

let *?states'* = λ*i*. ((*fst* (*states' i*))(*x* := *from-nat i*), *snd* (*states' i*))
have *?states'* $\in$ *Q*
by (*metis* (*no-types*, *lifting*) *asm1 assms*(*4*) *diff-by-update fstI not-free-var-of-def snd-conv*)
then obtain *r* **where** *r-def*: $\bigwedge$*i*. ((*fst* (*?states' i*))(*c* := *r*), *snd* (*?states' i*)) $\in$ *sem C S*
using *asm3 post-insecE*[*of from-nat x c Q*] **by** *fastforce*

let *?states* = λ*i*. *SOME* σ. ((*fst* (*?states' i*))(*c* := *r*), σ) $\in$ *S* $\wedge$ *single-sem C* σ (*snd* (*?states' i*))

have *asm4*: $\bigwedge$*i*. ((*fst* (*?states' i*))(*c* := *r*), (*?states i*)) $\in$ *S* $\wedge$ *single-sem C* (*?states i*) (*snd* (*?states' i*))
proof −
fix *i*
have $\exists$ σ. ((*fst* (*?states' i*))(*c* := *r*), σ) $\in$ *S* $\wedge$ *single-sem C* σ (*snd* (*?states' i*))
by (*metis r-def fst-conv in-sem snd-conv*)
then show ((*fst* (*?states' i*))(*c* := *r*), (*?states i*)) $\in$ *S* $\wedge$ *single-sem C* (*?states i*) (*snd* (*?states' i*))
using *someI-ex*[*of* $\lambda\sigma$. ((*fst* (*?states' i*))(*c* := *r*), σ) $\in$ *S* $\wedge$ *single-sem C* σ (*snd* (*?states' i*))]
by *blast*
qed
moreover have *r0*: (λ*i*. ((*fst* (*?states' i*))(*c* := *r*), (*?states i*))) $\in$ *P*
using *asm2*
proof (*rule strong-pre-insecE*)

fix i
show $(\lambda i.\ ((fst\ (?states'\ i))(c := r),\ (?states\ i)))\ i \in S$
using *calculation* **by** *blast*
show $fst\ ((\lambda i.\ ((fst\ (?states'\ i))(c := r),\ (?states\ i)))\ i)\ x = from\text{-}nat\ i$
using *assms*(*5*) **by** *auto*
fix j
show $fst\ ((\lambda i.\ ((fst\ (?states'\ i))(c := r),\ (?states\ i)))\ i)\ c = fst\ ((\lambda i.\ ((fst\ (?states'\ i))(c := r),\ (?states\ i)))\ j)\ c$
by *fastforce*
qed
have *r1*: $(\lambda i.\ (((fst\ (?states'\ i))(c := r))(x := fst\ (states'\ i)\ x),\ (?states\ i))) \in P$
proof (*rule not-free-var-ofE*[*of P x*])
show $(\lambda i.\ ((fst\ (?states'\ i))(c := r),\ (?states\ i))) \in P$
using *r0* **by** *fastforce*
show *not-free-var-of P x*
by (*simp add*: *assms*(*3*))
fix i
show *differ-only-by*
$(fst\ ((fst\ ((fst\ (states'\ i))(x := from\text{-}nat\ i),\ snd\ (states'\ i)))(c := r),\ ?states\ i))$
$(fst\ ((fst\ ((fst\ (states'\ i))(x := from\text{-}nat\ i),\ snd\ (states'\ i)))(c := r,\ x := fst\ (states'\ i)\ x),\ ?states\ i))\ x$
by (*metis* (*mono-tags*, *lifting*) *diff-by-comm diff-by-update fst-conv*)
qed (*auto*)
have $(\lambda i.\ ((((fst\ (?states'\ i))(c := r))(x := fst\ (states'\ i)\ x))(c := fst\ (states'\ i)\ c),\ (?states\ i))) \in P$
proof (*rule not-free-var-ofE*)
show $(\lambda i.\ (((fst\ (?states'\ i))(c := r))(x := fst\ (states'\ i)\ x),\ (?states\ i))) \in P$
using *r1* **by** *fastforce*
show *not-free-var-of P c*
by (*simp add*: *assms*(*3*))
fix i **show** *differ-only-by*
$(fst\ ((fst\ ((fst\ (states'\ i))(x := from\text{-}nat\ i),\ snd\ (states'\ i)))(c := r,\ x := fst\ (states'\ i)\ x),\ ?states\ i))$
$(fst\ ((fst\ ((fst\ (states'\ i))(x := from\text{-}nat\ i),\ snd\ (states'\ i)))(c := r,\ x := fst\ (states'\ i)\ x,\ c := fst\ (states'\ i)\ c),\ ?states\ i))$
c
by (*metis* (*mono-tags*, *lifting*) *diff-by-comm diff-by-update fst-conv*)
qed (*auto*)
moreover have $(\lambda i.\ ((((fst\ (?states'\ i))(c := r))(x := fst\ (states'\ i)\ x))(c := fst\ (states'\ i)\ c),\ (?states\ i)))$
$= (\lambda i.\ (fst\ (states'\ i),\ (?states\ i)))$
proof (*rule ext*)
fix i **show** $(\lambda i.\ ((((fst\ (?states'\ i))(c := r))(x := fst\ (states'\ i)\ x))(c := fst\ (states'\ i)\ c),\ (?states\ i)))\ i$
$= (\lambda i.\ (fst\ (states'\ i),\ (?states\ i)))\ i$
by *force*
qed

moreover have *k-sem C* $(\lambda i.\ (fst\ (states'\ i),\ (?states\ i)))$ *states'*
proof (*rule k-semI*)
fix *i*
show $(fst\ ((\lambda i.\ (fst\ (states'\ i),\ (?states\ i)))\ i) = fst\ (states'\ i) \wedge$
single-sem C $(snd\ ((\lambda i.\ (fst\ (states'\ i),\ (?states\ i)))\ i))\ (snd\ (states'\ i)))$
using *asm4* **by** *auto*
qed
ultimately show $\exists\, states \in P.$ *k-sem C states states'*
by *auto*
qed
qed

Proposition 5

theorem *encoding-RIL*:
fixes $x :: 'lvar$
assumes $\bigwedge l\ l'\ \sigma.\ (\lambda i.\ (l\ i,\ \sigma\ i)) \in P \longleftrightarrow (\lambda i.\ (l'\ i,\ \sigma\ i)) \in P$
and *injective* $(indexify :: (('a \Rightarrow ('pvar \Rightarrow 'pval)) \Rightarrow 'lval))$
and $c \neq x$
and *injective from-nat*
and *not-free-var-of* $(P :: ('a \Rightarrow ('lvar \Rightarrow 'lval) \times ('pvar \Rightarrow 'pval))\ set)\ x \wedge$ *not-free-var-of P c*
and *not-free-var-of Q x* $\wedge$ *not-free-var-of Q c*
shows *RIL P C Q* $\longleftrightarrow\ \models$ {*pre-insec from-nat x c P*} *C* {*post-insec from-nat x c Q*} (**is** $?A \longleftrightarrow ?B$)
proof (*rule encode-insec*)
show *sat* (*strong-pre-insec from-nat x c* $(P :: ('a \Rightarrow ('lvar \Rightarrow 'lval) \times ('pvar \Rightarrow 'pval))\ set))$
proof (*rule can-be-sat*)
show *injective* $(indexify :: (('a \Rightarrow ('pvar \Rightarrow 'pval)) \Rightarrow 'lval))$
by (*simp add*: *assms*(*2*))
show $x \neq c$
using *assms*(*3*) **by** *auto*
qed (*auto simp add*: *assms*)
qed (*auto simp add*: *assms*)

5.6 Forward Underapproximation (FU)

As employed by Outcome Logic [10]

Definition 12

definition *FU* **where**
$FU\ P\ C\ Q \longleftrightarrow (\forall l.\ \forall \sigma.\ (l,\ \sigma) \in P \longrightarrow (\exists \sigma'.\ single\text{-}sem\ C\ \sigma\ \sigma' \wedge (l,\ \sigma') \in Q))$

lemma *FUI*:
assumes $\bigwedge \sigma\ l.\ (l,\ \sigma) \in P \Longrightarrow (\exists \sigma'.\ single\text{-}sem\ C\ \sigma\ \sigma' \wedge (l,\ \sigma') \in Q)$
shows *FU P C Q*
by (*simp add*: *assms FU-def*)

definition *encode-FU* **where**

encode-FU P S $\longleftrightarrow$ $P \cap S \neq \{\}$

Proposition 6

theorem *encoding-FU*:
FU P C Q $\longleftrightarrow$ $\models$ {*encode-FU P*} *C* {*encode-FU Q*} (**is** *?A* $\longleftrightarrow$ *?B*)
proof
show *?B* $\Longrightarrow$ *?A*
proof −
assume *?B*
show *?A*
proof (*rule FUI*)
fix σ *l*
assume *asm*: $(l, \sigma) \in P$
then have *encode-FU P* $\{(l, \sigma)\}$
by (*simp add*: *encode-FU-def*)
then have $Q \cap$ *sem C* $\{(l, \sigma)\} \neq \{\}$
using ‹$\models$ {*encode-FU P*} *C* {*encode-FU Q*}› *hyper-hoare-tripleE encode-FU-def* **by** *blast*
then obtain φ' **where** $\varphi' \in Q$ $\varphi' \in$ *sem C* $\{(l, \sigma)\}$
by *blast*
then show $\exists \sigma'$. *single-sem C* σ σ' $\wedge$ $(l, \sigma') \in Q$
by (*metis fst-conv in-sem prod.collapse singletonD snd-conv*)
qed
qed
assume *?A*
show *?B*
proof (*rule hyper-hoare-tripleI*)
fix *S* **assume** *encode-FU P S*
then obtain *l* σ **where** $(l, \sigma) \in P \cap S$
by (*metis Expressivity.encode-FU-def ex-in-conv surj-pair*)
then obtain σ' **where** *single-sem C* σ σ' $\wedge$ $(l, \sigma') \in Q$
by (*meson IntD1* ‹*FU P C Q*› *FU-def*)
then show *encode-FU Q* (*sem C S*)
using *Expressivity.encode-FU-def* ‹$(l, \sigma) \in P \cap S$› *sem-def* **by** *fastforce*
qed
qed

definition *hyperprop-FU* **where**
hyperprop-FU P Q S $\longleftrightarrow$ $(\forall l\ \sigma.\ (l, \sigma) \in P \longrightarrow (\exists \sigma'.\ (l, \sigma') \in Q \wedge (\sigma, \sigma') \in S))$

lemma *hyperprop-FUI*:
assumes $\bigwedge l\ \sigma.\ (l, \sigma) \in P \Longrightarrow (\exists \sigma'.\ (l, \sigma') \in Q \wedge (\sigma, \sigma') \in S)$
shows *hyperprop-FU P Q S*
by (*simp add*: *hyperprop-FU-def assms*)

lemma *hyperprop-FUE*:
assumes *hyperprop-FU P Q S*
and $(l, \sigma) \in P$
shows $\exists \sigma'.\ (l, \sigma') \in Q \wedge (\sigma, \sigma') \in S$

using *hyperprop-FU-def assms*(*1*) *assms*(*2*) **by** *fastforce*

theorem *FU-expresses-hyperproperties*:
hypersat C (*hyperprop-FU P Q*) $\longleftrightarrow$ *FU P C Q* (**is** *?A* $\longleftrightarrow$ *?B*)
proof
assume *?A*
show *?B*
proof (*rule FUI*)
fix σ *l* **assume** (l, σ) $\in$ *P*
then obtain σ' **where** *asm0*: (l, σ') $\in$ *Q* $\wedge$ (σ, σ') $\in$ *set-of-traces C*
by (*meson* ‹*hypersat C* (*hyperprop-FU P Q*)› *hyperprop-FUE hypersat-def*)
then show $\exists\sigma'$. ($\langle C, \sigma\rangle \rightarrow \sigma'$) $\wedge$ (l, σ') $\in$ *Q*
using *in-set-of-traces* **by** *blast*
qed
next
assume *?B*
have *hyperprop-FU P Q* (*set-of-traces C*)
proof (*rule hyperprop-FUI*)
fix *l* σ
assume *asm0*: (l, σ) $\in$ *P*
then show $\exists\sigma'$. (l, σ') $\in$ *Q* $\wedge$ (σ, σ') $\in$ *set-of-traces C*
by (*metis* (*mono-tags*, *lifting*) *CollectI* ‹*FU P C Q*› *FU-def set-of-traces-def*)
qed
then show *?A*
by (*simp add*: *hypersat-def*)
qed

theorem *hyperliveness-hyperprop-FU*:
assumes $\bigwedge l$. *sat-for-l l P* $\Longrightarrow$ *sat-for-l l Q*
shows *hyperliveness* (*hyperprop-FU P Q*)
proof (*rule hyperlivenessI*)
fix *S* **show** $\exists S'$. $S \subseteq S'$ $\wedge$ *hyperprop-FU P Q* S'
by (*meson UNIV-I hyperprop-FU-def assms sat-for-l-def subsetI*)
qed

No relationship between incorrectness and forward underapproximation

lemma *incorrectness-does-not-imply-FU*:
assumes *injective from-nat*
assumes $P = \{(l, \sigma) \mid \sigma\ l.\ \sigma\ x = \textit{from-nat}\ (0 :: \textit{nat}) \vee \sigma\ x = \textit{from-nat}\ 1\}$
and $Q = \{(l, \sigma) \mid \sigma\ l.\ \sigma\ x = \textit{from-nat}\ 1\}$
and $C = \textit{Assume}\ (\lambda\sigma.\ \sigma\ x = \textit{from-nat}\ 1)$
shows *IL P C Q*
and $\neg$ *FU P C Q*
proof −
have $Q \subseteq$ *sem C P*
proof (*rule subsetPairI*)
fix *l* σ **assume** (l, σ) $\in$ *Q*
then have $\sigma\ x = \textit{from-nat}\ 1$
using *assms*(*3*) **by** *blast*

    then have $(l, \sigma) \in P$
      by $(simp\ add\colon assms(2))$
    then show $(l, \sigma) \in sem\ C\ P$
      by $(simp\ add\colon \langle\sigma\ x = from\text{-}nat\ 1\rangle\ assms(4)\ sem\text{-}assume)$
  qed
  then show $IL\ P\ C\ Q$
    by $(simp\ add\colon IL\text{-}def)$
  show $\neg\ FU\ P\ C\ Q$
  proof $(rule\ ccontr)$
    assume $\neg\ \neg\ FU\ P\ C\ Q$
    then have $FU\ P\ C\ Q$
      by *blast*
    obtain σ **where** $\sigma\ x = from\text{-}nat\ 0$
      by *simp*
    then obtain l **where** $(l, \sigma) \in P$
      using $assms(2)$ **by** *blast*
    then obtain σ' **where** $single\text{-}sem\ C\ \sigma\ \sigma'\ (l, \sigma') \in Q$
      by $(meson\ \langle FU\ P\ C\ Q\rangle\ FU\text{-}def)$
    then have $\sigma'\ x = from\text{-}nat\ 0$
      using $\langle\sigma\ x = from\text{-}nat\ 0\rangle\ assms(4)$ **by** *blast*
    then have $from\text{-}nat\ 0 = from\text{-}nat\ 1$
      using $\langle\langle C, \sigma\rangle \rightarrow \sigma'\rangle\ assms(4)$ **by** *force*
    then show *False*
      using $assms(1)\ injective\text{-}def[of\ from\text{-}nat]$ **by** *auto*
  qed
qed

lemma *FU-does-not-imply-incorrectness*:
  assumes $P = \{(l, \sigma)\ |\sigma\ l.\ \sigma\ x = from\text{-}nat\ (0 :: nat) \lor \sigma\ x = from\text{-}nat\ 1\}$
    and $Q = \{(l, \sigma)\ |\sigma\ l.\ \sigma\ x = from\text{-}nat\ 1\}$
  assumes *injective from-nat*
  shows $\neg\ IL\ Q\ Skip\ P$
    and $FU\ Q\ Skip\ P$
proof −
  show $FU\ Q\ Skip\ P$
  proof $(rule\ FUI)$
    fix $\sigma\ l$
    assume $(l, \sigma) \in Q$
    then show $\exists\sigma'.\ (\langle Skip, \sigma\rangle \rightarrow \sigma') \land (l, \sigma') \in P$
      using $SemSkip\ assms(1)\ assms(2)$ **by** *fastforce*
  qed
  obtain σ **where** $\sigma\ x = from\text{-}nat\ 0$
    by *simp*
  then obtain l **where** $(l, \sigma) \in P$
    using $assms(1)$ **by** *blast*
  moreover have $\sigma\ x \neq from\text{-}nat\ 1$
    by $(metis\ \langle\sigma\ x = from\text{-}nat\ 0\rangle\ assms(3)\ injective\text{-}def\ one\text{-}neq\text{-}zero)$
  then have $(l, \sigma) \notin Q$
    using $assms(2)$ **by** *blast*

ultimately show $\neg$ *IL Q Skip P*
using *IL-consequence* **by** *blast*
qed

5.7 Relational Forward Underapproximate logic

Definition 13

definition *RFU* **where**
RFU P C Q $\longleftrightarrow$ ($\forall$ *states* $\in$ *P*. $\exists$ *states'* $\in$ *Q*. *k-sem C states states'*)

lemma *RFUI*:
assumes $\bigwedge$*states*. *states* $\in$ *P* $\Longrightarrow$ ($\exists$ *states'* $\in$ *Q*. *k-sem C states states'*)
shows *RFU P C Q*
by (*simp add*: *assms RFU-def*)

lemma *RFUE*:
assumes *RFU P C Q*
and *states* $\in$ *P*
shows $\exists$ *states'* $\in$ *Q*. *k-sem C states states'*
using *assms*(*1*) *assms*(*2*) *RFU-def* **by** *blast*

definition *encode-RFU* **where**
encode-RFU from-nat x P S $\longleftrightarrow$ ($\exists$ *states* $\in$ *P*. ($\forall$ *i*. *states i* $\in$ *S* $\wedge$ *fst* (*states i*) *x* = *from-nat i*))

Proposition 7

theorem *encode-RFU*:
assumes *not-free-var-of P x*
and *not-free-var-of Q x*
and *injective from-nat*
shows *RFU P C Q* $\longleftrightarrow$ $\models$ {*encode-RFU from-nat x P*} *C* {*encode-RFU from-nat x Q*}
(**is** *?A* $\longleftrightarrow$ *?B*)
proof
assume *?A*
show *?B*
proof (*rule hyper-hoare-tripleI*)
fix *S* **assume** *encode-RFU from-nat x P S*
then obtain *states* **where** *asm0*: *states* $\in$ *P* $\bigwedge$*i*. *states i* $\in$ *S* $\wedge$ *fst* (*states i*) *x* = *from-nat i*
by (*meson encode-RFU-def*)
then obtain *states'* **where** *states'* $\in$ *Q k-sem C states states'*
using ‹*RFU P C Q*› *RFUE* **by** *blast*
then have $\bigwedge$*i*. *states' i* $\in$ *sem C S* $\wedge$ *fst* (*states' i*) *x* = *from-nat i*
by (*metis* (*mono-tags*, *lifting*) *asm0*(*2*) *in-sem k-sem-def prod.collapse*)
then show *encode-RFU from-nat x Q* (*sem C S*)
by (*meson* ‹*states'* $\in$ *Q*› *encode-RFU-def*)
qed
next

assume *?B*
show *?A*
proof (*rule RFUI*)
fix *states* **assume** *asm0*: $states \in P$
let $?states = \lambda i.\ ((fst\ (states\ i))(x := from\text{-}nat\ i),\ snd\ (states\ i))$

have $?states \in P$
using *assms*(*1*)
proof (*rule not-free-var-ofE*)
show $states \in P$ **using** *asm0* **by** *simp*
fix *i* **show** *differ-only-by* (*fst* (*states i*)) (*fst* ((*fst* (*states i*))(*x* := *from-nat i*), *snd* (*states i*))) *x*
using *diff-by-comm diff-by-update* **by** *fastforce*
qed (*auto*)
then have *encode-RFU from-nat x P* (*range ?states*)
using *encode-RFU-def* **by** *fastforce*
then have *encode-RFU from-nat x Q* (*sem C* (*range ?states*))
using ‹$\models$ {*encode-RFU from-nat x P*} *C* {*encode-RFU from-nat x Q*}› *hyper-hoare-tripleE* **by** *blast*
then obtain *states'* **where** *states'-def*: $states' \in Q$ $\bigwedge i.\ states'\ i \in sem\ C\ (range\ ?states) \wedge fst\ (states'\ i)\ x = from\text{-}nat\ i$
by (*meson encode-RFU-def*)

let $?states' = \lambda i.\ ((fst\ (states'\ i))(x := fst\ (states\ i)\ x),\ snd\ (states'\ i))$

have $?states' \in Q$
using *assms*(*2*)
proof (*rule not-free-var-ofE*)
show $states' \in Q$ **using** ‹$states' \in Q$› **by** *simp*
fix *i* **show** *differ-only-by* (*fst* (*states' i*)) (*fst* ((*fst* (*states' i*))(*x* := *fst* (*states i*) *x*), *snd* (*states' i*))) *x*
using *diff-by-comm diff-by-update* **by** *fastforce*
qed (*auto*)
moreover obtain *to-pvar* **where** *to-pvar-def*: $\bigwedge i.\ to\text{-}pvar\ (from\text{-}nat\ i) = i$
using *assms*(*3*) *injective-then-exists-inverse* **by** *blast*
then have *inj*: $\bigwedge i\ j.\ from\text{-}nat\ i = from\text{-}nat\ j \Longrightarrow i = j$
by *metis*

moreover have *k-sem C states ?states'*
proof (*rule k-semI*)
fix *i*
obtain σ **where** $(fst\ (states'\ i),\ \sigma) \in range\ (\lambda i.\ ((fst\ (states\ i))(x := from\text{-}nat\ i),\ snd\ (states\ i))) \wedge \langle C,\ \sigma\rangle \rightarrow snd\ (states'\ i)$
using *states'-def*(*2*) *in-sem* **by** *blast*
moreover have $fst\ (states'\ i)\ x = from\text{-}nat\ i$
by (*simp add*: *states'-def*)
then have *r*: $((fst\ (states\ (inv\ ?states\ (fst\ (states'\ i),\ \sigma))))$ $(x := from\text{-}nat\ (inv\ ?states\ (fst\ (states'\ i),\ \sigma))),\ snd\ (states\ (inv\ ?states\ (fst\ (states'\ i),\ \sigma))))$

$= (\mathit{fst}\ (\mathit{states}'\ i),\ \sigma)$
by (*metis* (*mono-tags*, *lifting*) *calculation f-inv-into-f*)
then have *single-sem* C (*snd* (*states* i)) (*snd* (*states*$'$ i))
using ‹*fst* (*states*$'$ i) x = *from-nat* i› *calculation inj* **by** *fastforce*
moreover have *fst* (*?states* i) = *fst* (*states*$'$ i)
by (*metis* (*mono-tags*, *lifting*)*r* ‹*fst* (*states*$'$ i) x = *from-nat* i› *fst-conv fun-upd-same inj*)
ultimately show *fst* (*states* i) = *fst* ((*fst* (*states*$'$ i))(x := *fst* (*states* i) x), *snd* (*states*$'$ i)) $\wedge$
$\langle C$, *snd* (*states* i)$\rangle \rightarrow$ *snd* ((*fst* (*states*$'$ i))(x := *fst* (*states* i) x), *snd* (*states*$'$ i))
by (*metis* (*mono-tags*, *lifting*) *fst-conv fun-upd-triv fun-upd-upd snd-conv*)
qed
ultimately show $\exists$ *states*$'\in Q$. *k-sem* C *states states*$'$ **by** *blast*
qed
qed

definition *RFU-hyperprop* **where**
RFU-hyperprop P Q S $\longleftrightarrow$ ($\forall$ l *states*. (λi. (l i, *states* i)) $\in P$
$\longrightarrow$ ($\exists$ *states*$'$. (λi. (l i, *states*$'$ i)) $\in Q$ $\wedge$ ($\forall$ i. (*states* i, *states*$'$ i) $\in S$)))

lemma *RFU-hyperpropI*:
assumes $\bigwedge l$ *states*. (λi. (l i, *states* i)) $\in P \Longrightarrow$ ($\exists$ *states*$'$. (λi. (l i, *states*$'$ i)) $\in Q$ $\wedge$ ($\forall$ i. (*states* i, *states*$'$ i) $\in S$))
shows *RFU-hyperprop* P Q S
by (*simp add*: *assms RFU-hyperprop-def*)

lemma *RFU-hyperpropE*:
assumes *RFU-hyperprop* P Q S
and (λi. (l i, *states* i)) $\in P$
shows $\exists$ *states*$'$. (λi. (l i, *states*$'$ i)) $\in Q$ $\wedge$ ($\forall$ i. (*states* i, *states*$'$ i) $\in S$)
using *assms*(*1*) *assms*(*2*) *RFU-hyperprop-def* **by** *blast*

Proposition 13

theorem *RFU-captures-hyperproperties*:
hypersat C (*RFU-hyperprop* P Q) $\longleftrightarrow$ *RFU* P C Q (**is** *?A* $\longleftrightarrow$ *?B*)
proof
assume *?A*
show *?B*
proof (*rule RFUI*)
fix *states* **assume** *states* $\in P$
moreover have (λi. ((*fst* $\circ$ *states*) i, (*snd* $\circ$ *states*) i)) = *states* **by** *simp*
ultimately obtain *states*$'$ **where** *asm0*: (λi. ((*fst* $\circ$ *states*) i, *states*$'$ i)) $\in Q$
$\bigwedge i$. ((*snd* $\circ$ *states*) i, *states*$'$ i) $\in$ *set-of-traces* C
using *RFU-hyperpropE*[*of* P Q *set-of-traces* C *fst* $\circ$ *states snd* $\circ$ *states*]
using ‹*hypersat* C (*RFU-hyperprop* P Q)› *hypersat-def* **by** *auto*

moreover have *k-sem* C *states* (λi. ((*fst* $\circ$ *states*) i, *states*$'$ i))
proof (*rule k-semI*)

fix i
have $\langle C,\ snd\ (states\ i)\rangle \rightarrow states'\ i$
using *calculation(2) in-set-of-traces* **by** *fastforce*
then show $fst\ (states\ i) = fst\ ((fst \circ states)\ i,\ states'\ i) \wedge \langle C,\ snd\ (states\ i)\rangle \rightarrow snd\ ((fst \circ states)\ i,\ states'\ i)$
by *simp*
qed
ultimately show $\exists\, states' {\in} Q.\ k\text{-}sem\ C\ states\ states'$
by *fast*
qed
next
assume *?B*
have $RFU\text{-}hyperprop\ P\ Q\ (set\text{-}of\text{-}traces\ C)$
proof (*rule RFU-hyperpropI*)
fix *l states*
assume $(\lambda i.\ (l\ i,\ states\ i)) \in P$
then obtain $states'$ **where** *asm0*: $states' \in Q\ k\text{-}sem\ C\ (\lambda i.\ (l\ i,\ states\ i))\ states'$
using ‹*RFU P C Q*› *RFUE* **by** *blast*
then have $\bigwedge i.\ fst\ (states'\ i) = l\ i$
by (*simp add*: *k-sem-def*)
moreover have $(\lambda i.\ (l\ i,\ (snd \circ states')\ i)) = states'$
proof (*rule ext*)
fix i **show** $(l\ i,\ (snd \circ states')\ i) = states'\ i$
by (*metis calculation comp-apply surjective-pairing*)
qed
moreover have $\bigwedge i.\ (states\ i,\ (snd \circ states')\ i) \in set\text{-}of\text{-}traces\ C$
proof –
fix i **show** $(states\ i,\ (snd \circ states')\ i) \in set\text{-}of\text{-}traces\ C$
using *asm0(2) comp-apply*[*of snd states'*] *in-set-of-traces k-sem-def*[*of C* $\lambda i.\ (l\ i,\ states\ i)$ *states'*] *snd-conv*
by *fastforce*
qed
ultimately show $\exists\, states'.\ (\lambda i.\ (l\ i,\ states'\ i)) \in Q \wedge (\forall\, i.\ (states\ i,\ states'\ i) \in set\text{-}of\text{-}traces\ C)$
using *asm0(1)* **by** *force*
qed
then show *?A*
by (*simp add*: *hypersat-def*)
qed

theorem *hyperliveness-encode-RFU*:
assumes $\bigwedge l.\ k\text{-}sat\text{-}for\text{-}l\ l\ P \Longrightarrow k\text{-}sat\text{-}for\text{-}l\ l\ Q$
shows $hyperliveness\ (RFU\text{-}hyperprop\ P\ Q)$
proof (*rule hyperlivenessI*)
fix S
have $RFU\text{-}hyperprop\ P\ Q\ UNIV$
proof (*rule RFU-hyperpropI*)
fix *l states* **assume** *asm0*: $(\lambda i.\ (l\ i,\ states\ i)) \in P$

then obtain *states'* **where** $(\lambda i.\ (l\ i,\ states'\ i)) \in Q$
by (*metis assms k-sat-for-l-def*)
then show $\exists\, states'.\ (\lambda i.\ (l\ i,\ states'\ i)) \in Q \wedge (\forall\, i.\ (states\ i,\ states'\ i) \in UNIV)$
by *blast*
qed
then show $\exists\, S'.\ S \subseteq S' \wedge \mathit{RFU\text{-}hyperprop}\ P\ Q\ S'$
by *blast*
qed

5.8 Relational Universal Existential (RUE) [4]

Definition 14

definition *RUE* **where**
$RUE\ P\ C\ Q \longleftrightarrow (\forall\,(\sigma 1,\ \sigma 2) \in P.\ \forall\,\sigma 1'.\ k\text{-}sem\ C\ \sigma 1\ \sigma 1' \longrightarrow (\exists\,\sigma 2'.\ k\text{-}sem\ C\ \sigma 2\ \sigma 2' \wedge (\sigma 1',\ \sigma 2') \in Q))$

lemma *RUE-I*:
assumes $\bigwedge \sigma 1\ \sigma 2\ \sigma 1'.\ (\sigma 1,\ \sigma 2) \in P \Longrightarrow k\text{-}sem\ C\ \sigma 1\ \sigma 1' \Longrightarrow (\exists\,\sigma 2'.\ k\text{-}sem\ C\ \sigma 2\ \sigma 2' \wedge (\sigma 1',\ \sigma 2') \in Q)$
shows *RUE P C Q*
using *assms RUE-def* **by** *fastforce*

lemma *RUE-E*:
assumes *RUE P C Q*
and $(\sigma 1,\ \sigma 2) \in P$
and $k\text{-}sem\ C\ \sigma 1\ \sigma 1'$
shows $\exists\,\sigma 2'.\ k\text{-}sem\ C\ \sigma 2\ \sigma 2' \wedge (\sigma 1',\ \sigma 2') \in Q$
using *RUE-def assms(1) assms(2) assms(3)* **by** *blast*

Hyperproperty

definition *hyperprop-RUE* **where**
$\mathit{hyperprop\text{-}RUE}\ P\ Q\ S \longleftrightarrow (\forall\, l1\ l2\ \sigma 1\ \sigma 2\ \sigma 1'.\ (\lambda i.\ (l1\ i,\ \sigma 1\ i),\ \lambda k.\ (l2\ k,\ \sigma 2\ k)) \in P \wedge$
$(\forall\, i.\ (\sigma 1\ i,\ \sigma 1'\ i) \in S) \longrightarrow (\exists\,\sigma 2'.\ (\forall\, k.\ (\sigma 2\ k,\ \sigma 2'\ k) \in S) \wedge (\lambda i.\ (l1\ i,\ \sigma 1'\ i),\ \lambda k.\ (l2\ k,\ \sigma 2'\ k)) \in Q))$

lemma *hyperprop-RUE-I*:
assumes $\bigwedge l1\ l2\ \sigma 1\ \sigma 2\ \sigma 1'.\ (\lambda i.\ (l1\ i,\ \sigma 1\ i),\ \lambda k.\ (l2\ k,\ \sigma 2\ k)) \in P \Longrightarrow$
$(\forall\, i.\ (\sigma 1\ i,\ \sigma 1'\ i) \in S) \Longrightarrow (\exists\,\sigma 2'.\ (\forall\, k.\ (\sigma 2\ k,\ \sigma 2'\ k) \in S) \wedge (\lambda i.\ (l1\ i,\ \sigma 1'\ i),\ \lambda k.\ (l2\ k,\ \sigma 2'\ k)) \in Q)$
shows *hyperprop-RUE P Q S*
using *assms hyperprop-RUE-def[of P Q S]*
by *force*

lemma *hyperprop-RUE-E*:
assumes *hyperprop-RUE P Q S*
and $(\lambda i.\ (l1\ i,\ \sigma 1\ i),\ \lambda k.\ (l2\ k,\ \sigma 2\ k)) \in P$
and $\bigwedge i.\ (\sigma 1\ i,\ \sigma 1'\ i) \in S$

shows $\exists \sigma 2'. (\forall k. (\sigma 2\ k, \sigma 2'\ k) \in S) \wedge (\lambda i. (l1\ i, \sigma 1'\ i), \lambda k. (l2\ k, \sigma 2'\ k)) \in Q$
using *assms*(*1*) *assms*(*2*) *assms*(*3*) *hyperprop-RUE-def* **by** *blast*

Proposition 14

theorem *RUE-express-hyperproperties*:
RUE P C Q $\longleftrightarrow$ *hypersat C* (*hyperprop-RUE P Q*) (**is** *?A* $\longleftrightarrow$ *?B*)
proof
assume *asm0*: *?A*
have *hyperprop-RUE P Q* (*set-of-traces C*)
proof (*rule hyperprop-RUE-I*)
fix *l1 l2* $\sigma 1\ \sigma 2\ \sigma 1'$
assume *asm1*: $(\lambda i. (l1\ i, \sigma 1\ i), \lambda k. (l2\ k, \sigma 2\ k)) \in P$ $\forall i. (\sigma 1\ i, \sigma 1'\ i) \in$ *set-of-traces C*

let $?x1 = \lambda i. (l1\ i, \sigma 1\ i)$
let $?x2 = \lambda k. (l2\ k, \sigma 2\ k)$

let $?x1' = \lambda i. (l1\ i, \sigma 1'\ i)$

have $\exists \sigma 2'.$ *k-sem C* $(\lambda k. (l2\ k, \sigma 2\ k))\ \sigma 2' \wedge (?x1', \sigma 2') \in Q$
using *asm0 asm1*(*1*)
proof (*rule RUE-E*)
show *k-sem C* $(\lambda i. (l1\ i, \sigma 1\ i))\ (\lambda i. (l1\ i, \sigma 1'\ i))$
proof (*rule k-semI*)
fix *i*
have *single-sem C* $(\sigma 1\ i)\ (\sigma 1'\ i)$ **using** *asm1*(*2*)
by (*simp add*: *set-of-traces-def*)
then show $fst\ (l1\ i, \sigma 1\ i) = fst\ (l1\ i, \sigma 1'\ i) \wedge \langle C, snd\ (l1\ i, \sigma 1\ i)\rangle \rightarrow snd\ (l1\ i, \sigma 1'\ i)$
by *simp*
qed
qed
then obtain $\sigma 2'$ **where** *asm2*: *k-sem C* $(\lambda k. (l2\ k, \sigma 2\ k))\ \sigma 2'\ (?x1', \sigma 2') \in Q$
by *blast*
let $?\sigma 2' = \lambda k.\ snd\ (\sigma 2'\ k)$

have $\bigwedge k. (\sigma 2\ k, ?\sigma 2'\ k) \in$ *set-of-traces C*
by (*metis* (*mono-tags*, *lifting*) *asm2*(*1*) *in-set-of-traces k-sem-def snd-conv*)
moreover have $(\lambda k. (l2\ k, ?\sigma 2'\ k)) = \sigma 2'$
proof (*rule ext*)
fix *k* **show** $(l2\ k, snd\ (\sigma 2'\ k)) = \sigma 2'\ k$
by (*metis* (*mono-tags*, *lifting*) *asm2*(*1*) *fst-eqD k-sem-def surjective-pairing*)
qed
ultimately show $\exists \sigma 2'. (\forall k. (\sigma 2\ k, \sigma 2'\ k) \in$ *set-of-traces C*$) \wedge (\lambda i. (l1\ i, \sigma 1'\ i), \lambda k. (l2\ k, \sigma 2'\ k)) \in Q$
using *asm2*(*2*) **by** *fastforce*
qed

then show *?B*
by (*simp add*: *hypersat-def*)
next
assume *?B* **then have** *asm0*: *hyperprop-RUE P Q* (*set-of-traces C*)
by (*simp add*: *hypersat-def*)
show *?A*
proof (*rule RUE-I*)
fix $\sigma 1$ $\sigma 2$ $\sigma 1'$
assume *asm1*: $(\sigma 1, \sigma 2) \in P$ *k-sem C* $\sigma 1$ $\sigma 1'$
then have $\bigwedge i.$ *fst* $(\sigma 1\ i) =$ *fst* $(\sigma 1'\ i)$
by (*simp add*: *k-sem-def*)
have $\exists \sigma 2'. (\forall k.$ (*snd* $(\sigma 2\ k), \sigma 2'\ k) \in$ *set-of-traces C*) $\wedge$ ($\lambda i.$ (*fst* $(\sigma 1\ i)$, *snd* $(\sigma 1'\ i)$), $\lambda k.$ (*fst* $(\sigma 2\ k), \sigma 2'\ k)) \in Q$
using *asm0*
proof (*rule hyperprop-RUE-E*[*of P Q set-of-traces C* $\lambda i.$ *fst* $(\sigma 1\ i)$ $\lambda i.$ *snd* $(\sigma 1\ i)$ $\lambda k.$ *fst* $(\sigma 2\ k)$ $\lambda k.$ *snd* $(\sigma 2\ k)$ $\lambda i.$ *snd* $(\sigma 1'\ i)$])
show ($\lambda i.$ (*fst* $(\sigma 1\ i)$, *snd* $(\sigma 1\ i)$), $\lambda k.$ (*fst* $(\sigma 2\ k)$, *snd* $(\sigma 2\ k))) \in P$
by (*simp add*: *asm1*(*1*))
fix *i* **show** (*snd* $(\sigma 1\ i)$, *snd* $(\sigma 1'\ i)) \in$ *set-of-traces C*
by (*metis* (*mono-tags*, *lifting*) *CollectI asm1*(*2*) *k-sem-def set-of-traces-def*)
qed
then obtain $\sigma 2'$ **where** *asm2*: $\bigwedge k.$ (*snd* $(\sigma 2\ k), \sigma 2'\ k) \in$ *set-of-traces C* ($\lambda i.$ (*fst* $(\sigma 1\ i)$, *snd* $(\sigma 1'\ i)$), $\lambda k.$ (*fst* $(\sigma 2\ k), \sigma 2'\ k)) \in Q$
by *blast*
moreover have *k-sem C* $\sigma 2$ ($\lambda k.$ (*fst* $(\sigma 2\ k), \sigma 2'\ k)$)
proof (*rule k-semI*)
fix *i* **show** *fst* $(\sigma 2\ i) =$ *fst* (*fst* $(\sigma 2\ i), \sigma 2'\ i) \wedge \langle C,$ *snd* $(\sigma 2\ i)\rangle \rightarrow$ *snd* (*fst* $(\sigma 2\ i), \sigma 2'\ i)$
using *calculation*(*1*) *in-set-of-traces* **by** *auto*
qed
ultimately show $\exists \sigma 2'.$ *k-sem C* $\sigma 2$ $\sigma 2' \wedge (\sigma 1', \sigma 2') \in Q$
using ‹$\bigwedge i.$ *fst* $(\sigma 1\ i) =$ *fst* $(\sigma 1'\ i)$› **by** *auto*
qed
qed

definition *is-type* **where**
is-type type fn x t S $\sigma \longleftrightarrow (\forall i.\ \sigma\ i \in S \wedge$ *fst* $(\sigma\ i)\ t =$ *type* $\wedge$ *fst* $(\sigma\ i)\ x =$ *fn i*$)$

lemma *is-typeI*:
assumes $\bigwedge i.\ \sigma\ i \in S$
and $\bigwedge i.$ *fst* $(\sigma\ i)\ t =$ *type*
and $\bigwedge i.$ *fst* $(\sigma\ i)\ x =$ *fn i*
shows *is-type type fn x t S* σ
by (*simp add*: *assms*(*1*) *assms*(*2*) *assms*(*3*) *is-type-def*)

lemma *is-type-E*:
assumes *is-type type fn x t S* σ
shows $\sigma\ i \in S \wedge$ *fst* $(\sigma\ i)\ t =$ *type* $\wedge$ *fst* $(\sigma\ i)\ x =$ *fn i*
by (*meson assms is-type-def*)

definition *encode-RUE-1* **where**
encode-RUE-1 fn fn1 fn2 x t P S $\longleftrightarrow$ $(\forall k.\ \exists \sigma \in S.\ fst\ \sigma\ x = fn2\ k \wedge fst\ \sigma\ t = fn\ 2)$
$\wedge$ $(\forall \sigma\ \sigma'.\ is\text{-}type\ (fn\ 1)\ fn1\ x\ t\ S\ \sigma \wedge is\text{-}type\ (fn\ 2)\ fn2\ x\ t\ S\ \sigma'$
$\longrightarrow (\sigma, \sigma') \in P)$

lemma *encode-RUE-1-I*:
assumes $\bigwedge k.\ \exists \sigma \in S.\ fst\ \sigma\ x = fn2\ k \wedge fst\ \sigma\ t = fn\ 2$
and $\bigwedge \sigma\ \sigma'.\ is\text{-}type\ (fn\ 1)\ fn1\ x\ t\ S\ \sigma \wedge is\text{-}type\ (fn\ 2)\ fn2\ x\ t\ S\ \sigma'$
$\Longrightarrow (\sigma, \sigma') \in P$
shows *encode-RUE-1 fn fn1 fn2 x t P S*
by (*simp add*: *assms*(*1*) *assms*(*2*) *encode-RUE-1-def*)

lemma *encode-RUE-1-E1*:
assumes *encode-RUE-1 fn fn1 fn2 x t P S*
shows $\exists \sigma \in S.\ fst\ \sigma\ x = fn2\ k \wedge fst\ \sigma\ t = fn\ 2$
by (*meson assms encode-RUE-1-def*)

lemma *encode-RUE-1-E2*:
assumes *encode-RUE-1 fn fn1 fn2 x t P S*
and $is\text{-}type\ (fn\ 1)\ fn1\ x\ t\ S\ \sigma$
and $is\text{-}type\ (fn\ 2)\ fn2\ x\ t\ S\ \sigma'$
shows $(\sigma, \sigma') \in P$
by (*meson assms encode-RUE-1-def*)

definition *encode-RUE-2* **where**
encode-RUE-2 fn fn1 fn2 x t Q S $\longleftrightarrow$ $(\forall \sigma.\ is\text{-}type\ (fn\ 1)\ fn1\ x\ t\ S\ \sigma \longrightarrow (\exists \sigma'.\ is\text{-}type\ (fn\ 2)\ fn2\ x\ t\ S\ \sigma' \wedge (\sigma, \sigma') \in Q))$

lemma *encode-RUE-2I*:
assumes $\bigwedge \sigma.\ is\text{-}type\ (fn\ 1)\ fn1\ x\ t\ S\ \sigma \Longrightarrow (\exists \sigma'.\ is\text{-}type\ (fn\ 2)\ fn2\ x\ t\ S\ \sigma' \wedge (\sigma, \sigma') \in Q)$
shows *encode-RUE-2 fn fn1 fn2 x t Q S*
by (*simp add*: *assms encode-RUE-2-def*)

lemma *encode-RUE-2-E*:
assumes *encode-RUE-2 fn fn1 fn2 x t Q S*
and $is\text{-}type\ (fn\ 1)\ fn1\ x\ t\ S\ \sigma$
shows $\exists \sigma'.\ is\text{-}type\ (fn\ 2)\ fn2\ x\ t\ S\ \sigma' \wedge (\sigma, \sigma') \in Q$
by (*meson assms*(*1*) *assms*(*2*) *encode-RUE-2-def*)

definition *differ-only-by-set* **where**
differ-only-by-set vars a b $\longleftrightarrow$ $(\forall x.\ x \notin vars \longrightarrow a\ x = b\ x)$

definition *differ-only-by-lset* **where**
differ-only-by-lset vars a b $\longleftrightarrow$ $(\forall i.\ snd\ (a\ i) = snd\ (b\ i) \wedge$ *differ-only-by-set*

vars (*fst* (*a i*)) (*fst* (*b i*)))

lemma *differ-only-by-lsetI*:
assumes $\bigwedge i.$ *snd* (*a i*) = *snd* (*b i*)
and $\bigwedge i.$ *differ-only-by-set vars* (*fst* (*a i*)) (*fst* (*b i*))
shows *differ-only-by-lset vars a b*
using *assms*(*1*) *assms*(*2*) *differ-only-by-lset-def* **by** *blast*

definition *not-in-free-vars-double* **where**
not-in-free-vars-double vars P $\longleftrightarrow$ ($\forall\sigma\ \sigma'$. *differ-only-by-lset vars* (*fst* σ) (*fst* σ')
$\wedge$
differ-only-by-lset vars (*snd* σ) (*snd* σ') $\longrightarrow$ ($\sigma \in P \longleftrightarrow \sigma' \in P$))

lemma *not-in-free-vars-doubleE*:
assumes *not-in-free-vars-double vars P*
and *differ-only-by-lset vars* (*fst* σ) (*fst* σ')
and *differ-only-by-lset vars* (*snd* σ) (*snd* σ')
and $\sigma \in P$
shows $\sigma' \in P$
by (*meson assms not-in-free-vars-double-def*)

Proposition 8

theorem *encoding-RUE*:
assumes *injective fn* $\wedge$ *injective fn1* $\wedge$ *injective fn2*
and $t \neq x$

and *injective* (*fn* :: *nat* $\Rightarrow$ *'a*)
and *injective fn1*
and *injective fn2*

and *not-in-free-vars-double* $\{x, t\}$ *P*
and *not-in-free-vars-double* $\{x, t\}$ *Q*

shows *RUE P C Q* $\longleftrightarrow$ $\models$ {*encode-RUE-1 fn fn1 fn2 x t P*} *C* {*encode-RUE-2 fn fn1 fn2 x t Q*}
(**is** *?A* $\longleftrightarrow$ *?B*)
proof
assume *asm0*: *?A*
show *?B*
proof (*rule hyper-hoare-tripleI*)
fix *S* **assume** *asm1*: *encode-RUE-1 fn fn1 fn2 x t P S*

show *encode-RUE-2 fn fn1 fn2 x t Q* (*sem C S*)
proof (*rule encode-RUE-2I*)
fix $\sigma 1'$ **assume** *asm2*: *is-type* (*fn 1*) *fn1 x t* (*sem C S*) $\sigma 1'$

let $?\sigma 2 = \lambda k.$ *SOME* σ'. $\sigma' \in S \wedge$ *fst* σ' *x* = *fn2 k* $\wedge$ *fst* σ' *t* = *fn 2*
have *r2*: $\bigwedge k.$ $?\sigma 2\ k \in S \wedge$ *fst* ($?\sigma 2\ k$) *x* = *fn2 k* $\wedge$ *fst* ($?\sigma 2\ k$) *t* = *fn 2*
proof –

fix k
show $?\sigma 2\ k \in S \wedge fst\ (?\sigma 2\ k)\ x = fn2\ k \wedge fst\ (?\sigma 2\ k)\ t = fn\ 2$
proof (*rule someI-ex*)
show $\exists xa.\ xa \in S \wedge fst\ xa\ x = fn2\ k \wedge fst\ xa\ t = fn\ 2$
by (*meson asm1 encode-RUE-1-E1*)
qed
qed
let $?\sigma 1 = \lambda i.\ SOME\ \sigma.\ (fst\ (\sigma 1'\ i), \sigma) \in S \wedge single\text{-}sem\ C\ \sigma\ (snd\ (\sigma 1'\ i))$
have $r1$: $\bigwedge i.\ (fst\ (\sigma 1'\ i),\ ?\sigma 1\ i) \in S \wedge single\text{-}sem\ C\ (?\sigma 1\ i)\ (snd\ (\sigma 1'\ i))$
proof −
fix i
show $(fst\ (\sigma 1'\ i),\ ?\sigma 1\ i) \in S \wedge single\text{-}sem\ C\ (?\sigma 1\ i)\ (snd\ (\sigma 1'\ i))$
proof (*rule someI-ex*[*of* $\lambda\sigma.\ (fst\ (\sigma 1'\ i), \sigma) \in S \wedge single\text{-}sem\ C\ \sigma\ (snd\ (\sigma 1'\ i))$])
show $\exists \sigma.\ (fst\ (\sigma 1'\ i), \sigma) \in S \wedge \langle C, \sigma\rangle \rightarrow snd\ (\sigma 1'\ i)$
by (*meson asm2 in-sem is-type-def*)
qed
qed
have $(\lambda i.\ (fst\ (\sigma 1'\ i),\ ?\sigma 1\ i),\ ?\sigma 2) \in P$
using *asm1*
proof (*rule encode-RUE-1-E2*)
show *is-type* $(fn\ 1)\ fn1\ x\ t\ S\ (\lambda i.\ (fst\ (\sigma 1'\ i),\ ?\sigma 1\ i))$
using *asm2 fst-conv is-type-def*[*of fn 1 fn1 x t S*] *is-type-def*[*of fn 1 fn1 x t sem C S*] *r1*
by *force*
show *is-type* $(fn\ 2)\ fn2\ x\ t\ S\ ?\sigma 2$
by (*simp add*: *is-type-def r2*)
qed
moreover have $\exists \sigma 2'.\ k\text{-}sem\ C\ ?\sigma 2\ \sigma 2' \wedge (\sigma 1', \sigma 2') \in Q$
using *asm0*
proof (*rule RUE-E*)
show $(\lambda i.\ (fst\ (\sigma 1'\ i),\ ?\sigma 1\ i),\ ?\sigma 2) \in P$
using *calculation* **by** *auto*
show $k\text{-}sem\ C\ (\lambda i.\ (fst\ (\sigma 1'\ i),\ SOME\ \sigma.\ (fst\ (\sigma 1'\ i), \sigma) \in S \wedge \langle C, \sigma\rangle \rightarrow snd\ (\sigma 1'\ i)))\ \sigma 1'$
by (*simp add*: *k-sem-def r1*)
qed
then obtain $\sigma 2'$ **where** $\sigma 2'$-*def*: $k\text{-}sem\ C\ ?\sigma 2\ \sigma 2' \wedge (\sigma 1', \sigma 2') \in Q$ **by** *blast*
then have *is-type* $(fn\ 2)\ fn2\ x\ t\ (sem\ C\ S)\ \sigma 2'$
using *in-sem*[*of - C S*] *k-semE*[*of C ?σ2 σ2'*] *prod.collapse r2 is-type-def*[*of fn 2 fn2 x t S*] *is-type-def*[*of fn 2 fn2 x t sem C S*]
by (*metis* (*no-types*, *lifting*))
then show $\exists \sigma 2'.\ is\text{-}type\ (fn\ 2)\ fn2\ x\ t\ (sem\ C\ S)\ \sigma 2' \wedge (\sigma 1', \sigma 2') \in Q$
using $\sigma 2'$-*def* **by** *blast*
qed
qed
next

assume *asm0*: $\models$ {*encode-RUE-1 fn fn1 fn2 x t P*} *C* {*encode-RUE-2 fn fn1 fn2 x t Q*}
show *?A*
proof (*rule RUE-I*)
fix $\sigma 1$ $\sigma 2$ $\sigma 1'$
assume *asm1*: $(\sigma 1, \sigma 2) \in P$ *k-sem C* $\sigma 1$ $\sigma 1'$

let $?\sigma 1 = \lambda i.$ $((((fst\ (\sigma 1\ i))(t := fn\ 1))(x := fn1\ i)), snd\ (\sigma 1\ i))$
let $?\sigma 2 = \lambda k.$ $((((fst\ (\sigma 2\ k))(t := fn\ 2))(x := fn2\ k)), snd\ (\sigma 2\ k))$

let $?S1 = \{\ ?\sigma 1\ i\ |i.\ True\ \}$
let $?S2 = \{\ ?\sigma 2\ k\ |k.\ True\ \}$
let $?S = ?S1 \cup ?S2$

let $?\sigma 1' = \lambda i.$ $((((fst\ (\sigma 1'\ i))(t := fn\ 1))(x := fn1\ i)), snd\ (\sigma 1'\ i))$

have *encode-RUE-2 fn fn1 fn2 x t Q* (*sem C ?S*)
using *asm0*
proof (*rule hyper-hoare-tripleE*)
show *encode-RUE-1 fn fn1 fn2 x t P ?S*
proof (*rule encode-RUE-1-I*)
fix *k*
let $?\sigma = ((((fst\ (\sigma 2\ k))(t := fn\ 2))(x := fn2\ k)), snd\ (\sigma 2\ k))$
have $?\sigma \in ?S2$
by *auto*
moreover have $fst\ ?\sigma\ x = fn2\ k$
by *simp*
moreover have $fst\ ?\sigma\ t = fn\ 2$
by (*simp add*: *assms*(*2*))
ultimately show $\exists \sigma \in ?S1 \cup ?S2.\ fst\ \sigma\ x = fn2\ k \wedge fst\ \sigma\ t = fn\ 2$
by *blast*
next
fix σ σ'
assume *asm2*: *is-type* $(fn\ (1 :: nat))$ *fn1 x t* $(?S1 \cup ?S2)$ $\sigma \wedge$ *is-type* (*fn* *2*) *fn2 x t* $(?S1 \cup ?S2)$ σ'
moreover have *r1*: $\bigwedge i.\ \sigma\ i = ((fst\ (\sigma 1\ i))(t := fn\ 1, x := fn1\ i), snd\ (\sigma 1\ i))$
proof −
fix *i*
have $fst\ (\sigma\ i)\ t = fn\ 1$
by (*meson calculation is-type-def*)
moreover have $\sigma\ i \in ?S1$
proof (*rule ccontr*)
assume $\neg\ \sigma\ i \in ?S1$
moreover have $\sigma\ i \in ?S1 \cup ?S2$
using *asm2 is-type-def*[*of fn 1 fn1 x t*]
by (*metis* (*no-types*, *lifting*))
ultimately have $\sigma\ i \in ?S2$ **by** *simp*
then have $fst\ (\sigma\ i)\ t = fn\ 2$

using *assms*(*2*) **by** *auto*
then show *False*
by (*metis Suc-1 Suc-eq-numeral* ‹*fst* (σ *i*) *t* = *fn 1*› *assms*(*3*) *injective-def numeral-One one-neq-zero pred-numeral-simps*(*1*))
qed
then obtain *j* **where** σ *i* = ((*fst* (σ*1 j*))(*t* := *fn 1*, *x* := *fn1 j*), *snd* (σ*1 j*))
by *blast*
moreover have *i* = *j*
by (*metis* (*mono-tags*, *lifting*) *asm2 assms*(*4*) *calculation*(*2*) *fst-conv fun-upd-same injective-def is-type-def*)
ultimately show σ *i* = ((*fst* (σ*1 i*))(*t* := *fn 1*, *x* := *fn1 i*), *snd* (σ*1 i*))
by *blast*
qed
moreover have $\bigwedge$*i*. σ' *i* = ((*fst* (σ*2 i*))(*t* := *fn 2*, *x* := *fn2 i*), *snd* (σ*2 i*))
proof −
fix *i*
have *fst* (σ' *i*) *t* = *fn 2*
by (*meson calculation is-type-def*)
moreover have σ' *i* ∈ *?S2*
proof (*rule ccontr*)
assume ¬ σ' *i* ∈ *?S2*
moreover have σ' *i* ∈ *?S1* ∪ *?S2*
using *asm2 is-type-def*[*of fn 2 fn2 x t*]
by (*metis* (*no-types*, *lifting*))
ultimately have σ' *i* ∈ *?S1* **by** *simp*
then have *fst* (σ' *i*) *t* = *fn 1*
using *assms*(*2*) **by** *auto*
then show *False*
by (*metis Suc-1 Suc-eq-numeral* ‹*fst* (σ' *i*) *t* = *fn 2*› *assms*(*3*) *injective-def numeral-One one-neq-zero pred-numeral-simps*(*1*))
qed
then obtain *j* **where** σ' *i* = ((*fst* (σ*2 j*))(*t* := *fn 2*, *x* := *fn2 j*), *snd* (σ*2 j*))
by *blast*
moreover have *i* = *j*
by (*metis* (*mono-tags*, *lifting*) *asm2 assms*(*5*) *calculation*(*2*) *fst-conv fun-upd-same injective-def is-type-def*)
ultimately show σ' *i* = ((*fst* (σ*2 i*))(*t* := *fn 2*, *x* := *fn2 i*), *snd* (σ*2 i*))
by *blast*
qed
moreover have (*?*σ*1*, *?*σ*2*) ∈ *P*
using *assms*(*6*)
proof (*rule not-in-free-vars-doubleE*)
show (σ*1*, σ*2*) ∈ *P*
by (*simp add*: *asm1*(*1*))
show *differ-only-by-lset* {*x*, *t*} (*fst* (σ*1*, σ*2*)) (*fst* (*?*σ*1*, *?*σ*2*))
by (*rule differ-only-by-lsetI*) (*simp-all add*: *differ-only-by-set-def*)
show *differ-only-by-lset* {*x*, *t*} (*snd* (σ*1*, σ*2*)) (*snd* (*?*σ*1*, *?*σ*2*))

by (*rule differ-only-by-lsetI*) (*simp-all add*: *differ-only-by-set-def*)
qed
ultimately show $(\sigma, \sigma') \in P$
by *presburger*
qed
qed
then have $\exists \sigma'$. *is-type* (*fn 2*) *fn2 x t* (*sem C ?S*) $\sigma' \wedge (?\sigma 1', \sigma') \in Q$
proof (*rule encode-RUE-2-E*)
show *is-type* (*fn 1*) *fn1 x t* (*sem C ?S*) $?\sigma 1'$
proof (*rule is-typeI*)
fix *i* **show** *fst* ((*fst* ($\sigma 1'$ *i*))(*t* := *fn 1*, *x* := *fn1 i*), *snd* ($\sigma 1'$ *i*)) *t* = *fn 1*
by (*simp add*: *assms*(*2*))
show ((*fst* ($\sigma 1'$ *i*))(*t* := *fn 1*, *x* := *fn1 i*), *snd* ($\sigma 1'$ *i*)) $\in$ *sem C ?S*
using *UnI1*[*of - ?S1 ?S2*]
asm1(*2*) *k-semE*[*of C* $\sigma 1$ $\sigma 1'$ *i*]
single-step-then-in-sem[*of C snd* ($\sigma 1$ *i*) *snd* ($\sigma 1'$ *i*) *- ?S*]
by *force*
qed (*auto*)
qed
then obtain $\sigma 2'$ **where** *r*: *is-type* (*fn 2*) *fn2 x t* (*sem C ?S*) $\sigma 2' \wedge (?\sigma 1', \sigma 2')$ $\in Q$
by *blast*
let $?\sigma 2' = \lambda k.$ ((*fst* ($\sigma 2'$ *k*))(*x* := *fst* ($\sigma 2$ *k*) *x*, *t* := *fst* ($\sigma 2$ *k*) *t*), *snd* ($\sigma 2'$ *k*))
have $(\sigma 1', ?\sigma 2') \in Q$
using *assms*(*7*)
proof (*rule not-in-free-vars-doubleE*)
show $(?\sigma 1', \sigma 2') \in Q$
using *r* **by** *blast*
show *differ-only-by-lset* $\{x, t\}$ (*fst* $(?\sigma 1', \sigma 2')$) (*fst* $(\sigma 1', ?\sigma 2')$)
by (*rule differ-only-by-lsetI*) (*simp-all add*: *differ-only-by-set-def*)
show *differ-only-by-lset* $\{x, t\}$ (*snd* $(?\sigma 1', \sigma 2')$) (*snd* $(\sigma 1', ?\sigma 2')$)
by (*rule differ-only-by-lsetI*) (*simp-all add*: *differ-only-by-set-def*)
qed
moreover have *k-sem C* $\sigma 2$ $?\sigma 2'$
proof (*rule k-semI*)
fix *i*
obtain *y* **where** *y-def*: $y \in ?S$ *fst y* = *fst* ($\sigma 2'$ *i*) *single-sem C* (*snd y*) (*snd* ($\sigma 2'$ *i*))
using *r in-sem*[*of* $\sigma 2'$ *i C ?S*]
is-type-E[*of fn 2 fn2 x t sem C ?S* $\sigma 2'$ *i*]
by (*metis* (*no-types*, *lifting*) *fst-conv snd-conv*)
then have *fst y t* = *fn 2*
by (*metis* (*no-types*, *lifting*) *is-type-def r*)
moreover have *fn 1* $\neq$ *fn 2*
by (*metis Suc-1 assms*(*3*) *injective-def n-not-Suc-n*)
then have $y \notin ?S1$
using *assms*(*2*) *calculation* **by** *fastforce*
then have $y \in ?S2$
using *y-def*(*1*) **by** *blast*

show $fst\ (\sigma 2\ i) = fst\ ((fst\ (\sigma 2'\ i))(x := fst\ (\sigma 2\ i)\ x,\ t := fst\ (\sigma 2\ i)\ t),\ snd\ (\sigma 2'\ i)) \wedge$
$\langle C,\ snd\ (\sigma 2\ i)\rangle \rightarrow snd\ ((fst\ (\sigma 2'\ i))(x := fst\ (\sigma 2\ i)\ x,\ t := fst\ (\sigma 2\ i)\ t),\ snd\ (\sigma 2'\ i))$
proof
have $r1$: $\sigma 2'\ i \in sem\ C\ ?S \wedge fst\ (\sigma 2'\ i)\ t = fn\ 2 \wedge fst\ (\sigma 2'\ i)\ x = fn2\ i$
proof (*rule is-type-E*[*of fn 2 fn2 x t sem C ?S* $\sigma 2'$ *i*])
show *is-type* (*fn 2*) *fn2 x t* (*sem C ?S*) $\sigma 2'$
using *r* **by** *blast*
qed
then obtain σ **where** $(fst\ (\sigma 2'\ i),\ \sigma) \in\ ?S\ single\text{-}sem\ C\ \sigma\ (snd\ (\sigma 2'\ i))$
by (*meson in-sem*)
then have $(fst\ (\sigma 2'\ i),\ \sigma) \in\ ?S2$
using *r1* ‹$fn\ 1 \neq fn\ 2$› *assms*(*2*) **by** *fastforce*
then obtain *k* **where** $fst\ (\sigma 2'\ i) = (fst\ (\sigma 2\ k))(t := fn\ 2,\ x := fn2\ k)$ **and** $\sigma = snd\ (\sigma 2\ k)$
by *blast*
then have $k = i$
by (*metis r1 assms*(*5*) *fun-upd-same injective-def*)
then show $\langle C,\ snd\ (\sigma 2\ i)\rangle \rightarrow snd\ ((fst\ (\sigma 2'\ i))(x := fst\ (\sigma 2\ i)\ x,\ t := fst\ (\sigma 2\ i)\ t),\ snd\ (\sigma 2'\ i))$
using ‹$\langle C,\ \sigma\rangle \rightarrow snd\ (\sigma 2'\ i)$› ‹$\sigma = snd\ (\sigma 2\ k)$› **by** *auto*
show $fst\ (\sigma 2\ i) = fst\ ((fst\ (\sigma 2'\ i))(x := fst\ (\sigma 2\ i)\ x,\ t := fst\ (\sigma 2\ i)\ t),\ snd\ (\sigma 2'\ i))$
by (*simp add*: ‹$fst\ (\sigma 2'\ i) = (fst\ (\sigma 2\ k))(t := fn\ 2,\ x := fn2\ k)$› ‹$k = i$›)
qed
qed
ultimately show $\exists \sigma 2'.\ k\text{-}sem\ C\ \sigma 2\ \sigma 2' \wedge (\sigma 1',\ \sigma 2') \in Q$
by *blast*
qed
qed

5.9 Program Refinement

lemma *sem-assign-single*:
$sem\ (Assign\ x\ e)\ \{(l,\ \sigma)\} = \{(l,\ \sigma(x := e\ \sigma))\}$ (**is** $?A =\ ?B$)
proof
show $?A \subseteq\ ?B$
proof (*rule subsetPairI*)
fix *la* σ'
assume $(la,\ \sigma') \in sem\ (Assign\ x\ e)\ \{(l,\ \sigma)\}$
then show $(la,\ \sigma') \in \{(l,\ \sigma(x := e\ \sigma))\}$
by (*metis* (*mono-tags*, *lifting*) *in-sem prod.sel*(*1*) *prod.sel*(*2*) *single-sem-Assign-elim singleton-iff*)
qed
show $?B \subseteq\ ?A$
by (*simp add*: *SemAssign in-sem*)
qed

definition *refinement* **where**
refinement C1 C2 $\longleftrightarrow$ (*set-of-traces C1* $\subseteq$ *set-of-traces C2*)

definition *not-free-var-stmt* **where**
not-free-var-stmt x C $\longleftrightarrow$ ($\forall \sigma\ \sigma'\ v.\ (\sigma, \sigma') \in$ *set-of-traces C* $\longrightarrow (\sigma(x := v), \sigma'(x := v)) \in$ *set-of-traces C*)
$\wedge$ ($\forall \sigma\ \sigma'$. *single-sem C* $\sigma\ \sigma' \longrightarrow \sigma\ x = \sigma'\ x$)

lemma *not-free-var-stmtE-1*:
assumes *not-free-var-stmt x C*
and $(\sigma, \sigma') \in$ *set-of-traces C*
shows $(\sigma(x := v), \sigma'(x := v)) \in$ *set-of-traces C*
using *assms(1) assms(2) not-free-var-stmt-def* **by** *force*

lemma *not-free-in-sem-same-val*:
assumes *not-free-var-stmt x C*
and *single-sem C* $\sigma\ \sigma'$
shows $\sigma\ x = \sigma'\ x$
using *assms(1) assms(2) not-free-var-stmt-def* **by** *fastforce*

lemma *not-free-in-sem-equiv*:
assumes *not-free-var-stmt x C*
and *single-sem C* $\sigma\ \sigma'$
shows *single-sem C* $(\sigma(x := v))\ (\sigma'(x := v))$
by (*meson assms(1) assms(2) in-set-of-traces not-free-var-stmtE-1*)

Example 4

theorem *encoding-refinement*:
fixes P :: $(('lvar \Rightarrow 'lval) \times ('pvar \Rightarrow 'pval))$ *set* $\Rightarrow$ *bool*
assumes $(a :: 'pval) \neq b$

and $P = (\lambda S.\ card\ S = 1)$
and $Q = (\lambda S.$
$\forall \varphi \in S.\ snd\ \varphi\ x = a \longrightarrow (fst\ \varphi, (snd\ \varphi)(x := b)) \in S)$
and *not-free-var-stmt x C1*
and *not-free-var-stmt x C2*
shows *refinement C1 C2* $\longleftrightarrow$
$\models$ { P } *If* (*Seq* (*Assign* $(x :: 'pvar)$ (λ-. a)) *C1*) (*Seq* (*Assign* x (λ-. b)) *C2*) {
Q }
(**is** *?A* $\longleftrightarrow$ *?B*)
proof
assume *?A*
show *?B*
proof (*rule hyper-hoare-tripleI*)
fix S **assume** P $(S :: (('lvar \Rightarrow 'lval) \times ('pvar \Rightarrow 'pval))$ *set*)

then obtain σ l **where** *asm0*: $S = \{(l, \sigma)\}$
by (*metis assms(2) card-1-singletonE surj-pair*)

let *?C* = *If* (*Seq* (*Assign* *x* (λ-. *a*)) *C1*) (*Seq* (*Assign* *x* (λ-. *b*)) *C2*)
let *?a* = (*l*, σ(*x* := *a*))
let *?b* = (*l*, σ(*x* := *b*))

have *if-sem*: *sem* *?C* *S* = *sem* *C1* {*?a*} ∪ *sem* *C2* {*?b*}
by (*simp add*: *asm0 sem-assign-single sem-if sem-seq*)
then have ⋀φ. φ ∈ *sem* *?C* *S* ⟹ *snd* φ *x* = *a* ⟹ (*fst* φ, (*snd* φ)(*x* := *b*)) ∈ *sem* *?C* *S*
proof −
fix φ **assume** *asm1*: φ ∈ *sem* *?C* *S* *snd* φ *x* = *a*
have φ ∈ *sem* *C1* {*?a*}
proof (*rule ccontr*)
assume φ ∉ *sem* *C1* {(*l*, σ(*x* := *a*))}
then have φ ∈ *sem* *C2* {(*l*, σ(*x* := *b*))}
using *if-sem asm1*(*1*) **by** *force*
then have *snd* φ *x* = *b*
using *assms*(*5*) *fun-upd-same in-sem not-free-in-sem-same-val*[*of x C2* σ(*x* := *b*) *snd* φ] *singletonD snd-conv*
by *metis*
then show *False*
using *asm1*(*2*) *assms*(*1*) **by** *blast*
qed
then have (σ(*x* := *a*), *snd* φ) ∈ *set-of-traces* *C1*
by (*simp add*: *in-sem set-of-traces-def*)
then have (σ(*x* := *a*), *snd* φ) ∈ *set-of-traces* *C2*
using ‹*refinement C1 C2*› *refinement-def* **by** *fastforce*
then have ((σ(*x* := *a*))(*x* := *b*), (*snd* φ)(*x* := *b*)) ∈ *set-of-traces* *C2*
by (*meson assms*(*5*) *not-free-var-stmtE-1*)
then have *single-sem* *C2* (σ(*x* := *b*)) ((*snd* φ)(*x* := *b*))
by (*simp add*: *set-of-traces-def*)
then have (*fst* φ, (*snd* φ)(*x* := *b*)) ∈ *sem* *C2* {*?b*}
by (*metis* ‹φ ∈ *sem* *C1* {(*l*, σ(*x* := *a*))}› *fst-eqD in-sem singleton-iff snd-eqD*)
then show (*fst* φ, (*snd* φ)(*x* := *b*)) ∈ *sem* *?C* *S*
by (*simp add*: *if-sem*)
qed
then show *Q* (*sem* *?C* *S*)
using *assms*(*3*) **by** *blast*
qed
next
assume *asm0*: *?B*

have *set-of-traces* *C1* ⊆ *set-of-traces* *C2*
proof (*rule subsetPairI*)
fix σ σ′ **assume** *asm1*: (σ, σ′) ∈ *set-of-traces* *C1*
obtain *l* *S* **where** (*S* :: ((′*lvar* ⇒ ′*lval*) × (′*pvar* ⇒ ′*pval*)) *set*) = { (*l*, σ) }

by *simp*

let *?a* = (*l*, σ(*x* := *a*))

```
    let ?b = (l, σ(x := b))

    let ?C = If (Seq (Assign (x :: 'pvar) (λ-. a)) C1) (Seq (Assign x (λ-. b)) C2)
    have Q (sem ?C S)
    proof (rule hyper-hoare-tripleE)
      show P S
        by (simp add: ‹S = {(l, σ)}› assms(2))
      show ?B using asm0 by simp
    qed
    moreover have (l, σ'(x := a)) ∈ sem ?C S
    proof –
      have single-sem (Seq (Assign x (λ-. a)) C1) σ (σ'(x := a))
      by (meson SemAssign SemSeq asm1 assms(4) in-set-of-traces not-free-in-sem-equiv)
      then show ?thesis
        by (simp add: SemIf1 ‹S = {(l, σ)}› in-sem)
    qed
    then have (l, σ'(x := b)) ∈ sem ?C S
      using assms(3) calculation by fastforce
    moreover have (l, σ'(x := b)) ∈ sem (Seq (Assign x (λ-. b)) C2) S
    proof (rule ccontr)
      assume ¬ (l, σ'(x := b)) ∈ sem (Seq (Assign x (λ-. b)) C2) S
      then have (l, σ'(x := b)) ∈ sem (Seq (Assign x (λ-. a)) C1) S
        using calculation(2) sem-if by auto
      then have (l, σ'(x := b)) ∈ sem C1 {?a}
        by (simp add: ‹S = {(l, σ)}› sem-assign-single sem-seq)
      then show False
        using assms(1) assms(4) fun-upd-same in-sem not-free-in-sem-same-val[of
x C1  σ(x := a) σ'(x := b)] singletonD snd-conv
        by metis
    qed
    then have single-sem (Seq (Assign x (λ-. b)) C2) σ (σ'(x := b))
      by (simp add: ‹S = {(l, σ)}› in-sem)
    then have single-sem C2 (σ(x := b)) (σ'(x := b))
      by blast
    then have (σ(x := b), σ'(x := b)) ∈ set-of-traces C2
      by (simp add: set-of-traces-def)
    then have ((σ(x := b))(x := σ x), (σ'(x := b))(x := σ x)) ∈ set-of-traces C2
      by (meson assms(5) not-free-var-stmtE-1)
    then show (σ, σ') ∈ set-of-traces C2
     by (metis asm1 assms(4) fun-upd-triv fun-upd-upd in-set-of-traces not-free-in-sem-same-val)
  qed
  then show ?A
    by (simp add: refinement-def)
qed

end
```

References

[1] Michael R. Clarkson and Fred B. Schneider. Hyperproperties. In *21st IEEE Computer Security Foundations Symposium*, pages 51–65, 2008.

[2] Thibault Dardinier and Peter Müller. Hyper Hoare Logic: (dis-)proving program hyperproperties (extended version). *arXiv preprint arXiv:2301.10037*, 2023.

[3] Edsko de Vries and Vasileios Koutavas. Reverse Hoare Logic. In Gilles Barthe, Alberto Pardo, and Gerardo Schneider, editors, *Software Engineering and Formal Methods*, pages 155–171, 2011.

[4] Robert Dickerson, Qianchuan Ye, Michael K. Zhang, and Benjamin Delaware. RHLE: Modular deductive verification ofărelational ∀∃ properties. In *Programming Languages and Systems: 20th Asian Symposium, APLAS 2022, Auckland, New Zealand, December 5, 2022, Proceedings*, page 6787, 2022.

[5] Robert W. Floyd. Assigning meanings to programs. *Proceedings of Symposium in Applied Mathematics*, pages 19–32, 1967.

[6] C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576580, oct 1969.

[7] Toby Murray. An under-approximate relational logic: Heralding logics of insecurity, incorrect implementation and more. *arXiv preprint arXiv:2003.04791*, 2020.

[8] Peter W. O'Hearn. Incorrectness Logic. *Proc. ACM Program. Lang.*, 4(POPL), dec 2019.

[9] Marcelo Sousa and Isil Dillig. Cartesian Hoare Logic for verifying k-safety properties. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '16, page 5769, New York, NY, USA, 2016. Association for Computing Machinery.

[10] Noam Zilberstein, Derek Dreyer, and Alexandra Silva. Outcome Logic: A unifying foundation for correctness and incorrectness reasoning. *Proc. ACM Program. Lang.*, 7(OOPSLA1), april 2023.