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 $('var, 'val)\ exp = ('var, 'val)\ pstate \Rightarrow 'val$

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

1.2 Semantics

Figure 2

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

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

2 Extended States and Extended Semantics

Definition 3

type-synonym $('lvar, 'lval, 'pvar, 'pval)\ state = ('lvar \Rightarrow 'lval) \times ('pvar, 'pval)\ pstate$

Definition 5

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

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** $?A \longleftrightarrow ?B$)
$\langle proof \rangle$

Useful properties

lemma *sem-seq*:
$sem\ (Seq\ C1\ C2)\ S = sem\ C2\ (sem\ C1\ S)$ (**is** $?A = ?B$)
$\langle proof \rangle$

lemma *sem-skip*:
$sem\ Skip\ S = S$
$\langle proof \rangle$

lemma *sem-union*:
$sem\ C\ (S1 \cup S2) = sem\ C\ S1 \cup sem\ C\ S2$ (**is** $?A = ?B$)
$\langle proof \rangle$

lemma *sem-union-general*:
$sem\ C\ (\bigcup x.\ f\ x) = (\bigcup x.\ sem\ C\ (f\ x))$ (**is** $?A = ?B$)
$\langle proof \rangle$

lemma *sem-monotonic*:
assumes $S \subseteq S'$
shows $sem\ C\ S \subseteq sem\ C\ S'$
$\langle proof \rangle$

lemma *subsetPairI*:
assumes $\bigwedge l\ \sigma.\ (l, \sigma) \in A \Longrightarrow (l, \sigma) \in B$
shows $A \subseteq B$
$\langle proof \rangle$

lemma *sem-if*:
$sem\ (If\ C1\ C2)\ S = sem\ C1\ S \cup sem\ C2\ S$ (**is** $?A = ?B$)
$\langle proof \rangle$

lemma *sem-assume*:
$sem\ (Assume\ b)\ S = \{\ (l, \sigma)\ |l\ \sigma.\ (l, \sigma) \in S \wedge b\ \sigma\ \}$ (**is** $?A = ?B$)
$\langle proof \rangle$

lemma *while-then-reaches*:
assumes $(single\text{-}sem\ C)^{**}\ \sigma\ \sigma''$
shows $single\text{-}sem\ (While\ C)\ \sigma\ \sigma''$
$\langle proof \rangle$

lemma *in-closure-then-while*:
assumes $single\text{-}sem\ C'\ \sigma\ \sigma''$

shows $\bigwedge C.\ C' = \mathit{While}\ C \Longrightarrow (\mathit{single\text{-}sem}\ C)^{**}\ \sigma\ \sigma''$
⟨*proof*⟩

theorem *loop-equiv*:
$\mathit{single\text{-}sem}\ (\mathit{While}\ C)\ \sigma\ \sigma' \longleftrightarrow (\mathit{single\text{-}sem}\ C)^{**}\ \sigma\ \sigma'$
⟨*proof*⟩

fun *iterate-sem* **where**
$\mathit{iterate\text{-}sem}\ 0\ \text{-}\ S = S$
$|\ \mathit{iterate\text{-}sem}\ (\mathit{Suc}\ n)\ C\ S = \mathit{sem}\ C\ (\mathit{iterate\text{-}sem}\ n\ C\ S)$

lemma *in-iterate-then-in-trans*:
assumes $(l, \sigma'') \in \mathit{iterate\text{-}sem}\ n\ C\ S$
shows $\exists \sigma.\ (l, \sigma) \in S \wedge (\mathit{single\text{-}sem}\ C)^{**}\ \sigma\ \sigma''$
⟨*proof*⟩

lemma *reciprocal*:
assumes $(\mathit{single\text{-}sem}\ C)^{**}\ \sigma\ \sigma''$
and $(l, \sigma) \in S$
shows $\exists n.\ (l, \sigma'') \in \mathit{iterate\text{-}sem}\ n\ C\ S$
⟨*proof*⟩

lemma *union-iterate-sem-trans*:
$(l, \sigma'') \in (\bigcup n.\ \mathit{iterate\text{-}sem}\ n\ C\ S) \longleftrightarrow (\exists \sigma.\ (l, \sigma) \in S \wedge (\mathit{single\text{-}sem}\ C)^{**}\ \sigma\ \sigma'')$ (**is** $?A \longleftrightarrow ?B$)
⟨*proof*⟩

lemma *sem-while*:
$\mathit{sem}\ (\mathit{While}\ C)\ S = (\bigcup n.\ \mathit{iterate\text{-}sem}\ n\ C\ S)$ (**is** $?A = ?B$)
⟨*proof*⟩

lemma *assume-sem*:
$\mathit{sem}\ (\mathit{Assume}\ b)\ S = \mathit{Set.filter}\ (b \circ \mathit{snd})\ S$ (**is** $?A = ?B$)
⟨*proof*⟩

lemma *sem-split-general*:
$\mathit{sem}\ C\ (\bigcup x.\ F\ x) = (\bigcup x.\ \mathit{sem}\ C\ (F\ x))$ (**is** $?A = ?B$)
⟨*proof*⟩

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 $entails\ A\ B$
  ⟨*proof*⟩

lemma *entailsE*:
  assumes $entails\ A\ B$
    and $A\ x$
  shows $B\ x$
  ⟨*proof*⟩

lemma *bientails-equal*:
  assumes $entails\ A\ B$
    and $entails\ B\ A$
  shows $A = B$
⟨*proof*⟩

lemma *entails-trans*:
  assumes $entails\ A\ B$
    and $entails\ B\ C$
  shows $entails\ A\ C$
  ⟨*proof*⟩

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

lemma *sem-assume-setify*:
  $sem\ (Assume\ b)\ S = S \cap setify\text{-}prop\ b$ (**is** $?A = ?B$)
⟨*proof*⟩

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

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

lemma *over-approx-lower-closed*:
  lower-closed (*over-approx* P)
  ⟨*proof*⟩

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)
  ⟨*proof*⟩

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
  ⟨*proof*⟩

lemma *closed-by-union-over*:
  closed-by-union (*over-approx* P)
  ⟨*proof*⟩

lemma *closed-by-union-under*:
  closed-by-union (*under-approx* P)
  ⟨*proof*⟩

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))
  ⟨*proof*⟩

lemma *over-union*:
  entails (*disj* (*over-approx* P) (*over-approx* Q)) (*over-approx* $(P \cup Q)$)
  ⟨*proof*⟩

lemma *under-union*:
$\mathit{entails}\ (\mathit{under\text{-}approx}\ (P \cup Q))\ (\mathit{disj}\ (\mathit{under\text{-}approx}\ P)\ (\mathit{under\text{-}approx}\ Q))$
⟨*proof*⟩

lemma *under-inter*:
$\mathit{entails}\ (\mathit{conj}\ (\mathit{under\text{-}approx}\ P)\ (\mathit{under\text{-}approx}\ Q))\ (\mathit{under\text{-}approx}\ (P \cap Q))$
⟨*proof*⟩

Notation 1

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

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

lemma *join-closed-by-union*:
assumes $\mathit{closed\text{-}by\text{-}union}\ Q$
shows $\mathit{join}\ Q\ Q = Q$
⟨*proof*⟩

lemma *entails-join-entails*:
assumes $\mathit{entails}\ A1\ B1$
and $\mathit{entails}\ A2\ B2$
shows $\mathit{entails}\ (\mathit{join}\ A1\ A2)\ (\mathit{join}\ B1\ B2)$
⟨*proof*⟩

Notation 2

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

lemma *natural-partitionI*:
assumes $S = (\bigcup n.\ F\ n)$
and $\bigwedge n.\ I\ n\ (F\ n)$
shows $\mathit{natural\text{-}partition}\ I\ S$
⟨*proof*⟩

lemma *natural-partitionE*:
assumes $\mathit{natural\text{-}partition}\ I\ S$
obtains F **where** $S = (\bigcup n.\ F\ n)$ $\bigwedge n.\ I\ n\ (F\ n)$
⟨*proof*⟩

3.1 Rules of the Logic

Rules from figure 3

inductive *syntactic-HHT* ::
$(({}'\mathit{lvar}, {}'\mathit{lval}, {}'\mathit{pvar}, {}'\mathit{pval})\ \mathit{state}\ \mathit{hyperassertion}) \Rightarrow ({}'\mathit{pvar}, {}'\mathit{pval})\ \mathit{stmt} \Rightarrow (({}'\mathit{lvar}, {}'\mathit{lval}, {}'\mathit{pvar}, {}'\mathit{pval})\ \mathit{state}\ \mathit{hyperassertion}) \Rightarrow \mathit{bool}$

($\vdash$ {-} - {-} [*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-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*: $[\![\bigwedge x$::('*lvar*, '*lval*, '*pvar*, '*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* ($\models$ {-} - {-} [*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 $\models \{P\}\ C\ \{Q\}$
⟨*proof*⟩

lemma *hyper-hoare-tripleE*:
assumes $\models \{P\}\ C\ \{Q\}$
and $P\ S$
shows $Q\ (sem\ C\ S)$
⟨*proof*⟩

lemma *consequence-rule*:
assumes *entails* $P\ P'$
and *entails* $Q'\ Q$
and $\models \{P'\}\ C\ \{Q'\}$
shows $\models \{P\}\ C\ \{Q\}$
⟨*proof*⟩

lemma *skip-rule*:
$\models \{P\}$ *Skip* $\{P\}$
⟨*proof*⟩

lemma *assume-rule*:
$\models$ { ($\lambda S.\ P$ (*Set.filter* ($b \circ snd$) S)) } (*Assume b*) $\{P\}$
⟨*proof*⟩

lemma *seq-rule*:
assumes $\models \{P\}\ C1\ \{R\}$
and $\models \{R\}\ C2\ \{Q\}$

shows $\models \{P\}$ *Seq C1 C2* $\{Q\}$
⟨*proof*⟩

lemma *if-rule*:
assumes $\models \{P\}$ *C1* $\{Q1\}$
and $\models \{P\}$ *C2* $\{Q2\}$
shows $\models \{P\}$ *If C1 C2* $\{join\ Q1\ Q2\}$
⟨*proof*⟩

lemma *sem-assign*:
$sem\ (Assign\ x\ e)\ S = \{(l, \sigma(x := e\ \sigma)) \mid l\ \sigma.\ (l, \sigma) \in S\}$ (**is** $?A = ?B$)
⟨*proof*⟩

lemma *assign-rule*:
$\models \{\ (\lambda S.\ P\ \{\ (l, \sigma(x := e\ \sigma)) \mid l\ \sigma.\ (l, \sigma) \in S\ \})\ \}\ (Assign\ x\ e)\ \{P\}$
⟨*proof*⟩

lemma *sem-havoc*:
$sem\ (Havoc\ x)\ S = \{(l, \sigma(x := v)) \mid l\ \sigma\ v.\ (l, \sigma) \in S\}$ (**is** $?A = ?B$)
⟨*proof*⟩

lemma *havoc-rule*:
$\models \{\ (\lambda S.\ P\ \{\ (l, \sigma(x := v)) \mid l\ \sigma\ v.\ (l, \sigma) \in S\ \})\ \}\ (Havoc\ x)\ \{P\}$
⟨*proof*⟩

Loops

lemma *indexed-invariant-then-power*:
assumes $\bigwedge n.$ *hyper-hoare-triple* $(I\ n)\ C\ (I\ (Suc\ n))$
and $I\ 0\ S$
shows $I\ n$ (*iterate-sem* $n\ C\ S$)
⟨*proof*⟩

lemma *while-rule*:
assumes $\bigwedge n.$ *hyper-hoare-triple* $(I\ n)\ C\ (I\ (Suc\ n))$
shows *hyper-hoare-triple* $(I\ 0)$ (*While* C) (*natural-partition* I)
⟨*proof*⟩

Additional rules

lemma *empty-pre*:
hyper-hoare-triple (λ-. *False*) C QQ
⟨*proof*⟩

lemma *full-post*:
hyper-hoare-triple P C (λ-. *True*)
⟨*proof*⟩

lemma *rule-join*:
assumes $\models \{P\}\ C\ \{Q\}$

and *hyper-hoare-triple* P' C Q'
shows *hyper-hoare-triple* (*join* P P') C (*join* Q Q')
⟨*proof*⟩

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*⟩

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*⟩

Generalization

lemma *rule-forall*:
assumes $\bigwedge x.\ \models \{P\ x\}\ C\ \{Q\ x\}$
shows *hyper-hoare-triple* (*forall* P) C (*forall* Q)
⟨*proof*⟩

lemma *rule-disj*:
assumes $\models \{P\}\ C\ \{Q\}$
and $\models \{P'\}\ C\ \{Q'\}$
shows *hyper-hoare-triple* (*disj* P P') C (*disj* Q Q')
⟨*proof*⟩

Generalization

lemma *rule-exists*:
assumes $\bigwedge x.\ \models \{P\ x\}\ C\ \{Q\ x\}$
shows $\models$ {*exists* P} C {*exists* Q}
⟨*proof*⟩

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
⟨*proof*⟩

Simplifying the rule

definition *stable-by-infinite-union* :: *'a hyperassertion* ⇒ *bool* **where**
stable-by-infinite-union $I \longleftrightarrow (\forall F.\ (\forall S \in F.\ I\ S) \longrightarrow I\ (\bigcup S \in F.\ S))$

lemma *stable-by-infinite-unionE*:
assumes *stable-by-infinite-union* I
and $\bigwedge S.\ S \in F \Longrightarrow I\ S$
shows $I\ (\bigcup S \in F.\ S)$
⟨*proof*⟩

lemma *stable-by-union-and-constant-then-I*:
assumes $\bigwedge n.\ I\ n = I'$
and *stable-by-infinite-union* I'
shows *natural-partition* $I = I'$
⟨*proof*⟩

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*⟩

Theorem 1

theorem *soundness*:
assumes $\vdash \{A\}\ C\ \{B\}$
shows $\models \{A\}\ C\ \{B\}$
⟨*proof*⟩

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* $P\ C\ Q$
⟨*proof*⟩

lemma *completeE*:
assumes *complete* $P\ C\ Q$
and $\models \{P\}\ C\ \{Q\}$
shows $\vdash \{P\}\ C\ \{Q\}$
⟨*proof*⟩

lemma *complete-if-aux*:
assumes *hyper-hoare-triple* A (*If* $C1\ C2$) B
shows *entails* $(\lambda S'.\ \exists S.\ A\ S \wedge S' = \mathit{sem}\ C1\ S \cup \mathit{sem}\ C2\ S)\ B$
⟨*proof*⟩

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*⟩

lemma *complete-seq-aux*:
assumes *hyper-hoare-triple A* (*Seq C1 C2*) *B*
shows $\exists$ *R. hyper-hoare-triple A C1 R* $\wedge$ *hyper-hoare-triple R C2 B*
⟨*proof*⟩

lemma *complete-assume*:
complete P (*Assume b*) *Q*
⟨*proof*⟩

lemma *complete-skip*:
complete P Skip Q
⟨*proof*⟩

lemma *complete-assign*:
complete P (*Assign x e*) *Q*
⟨*proof*⟩

lemma *complete-havoc*:
complete P (*Havoc x*) *Q*
⟨*proof*⟩

lemma *complete-seq*:
assumes $\bigwedge$*R. complete P C1 R*
and $\bigwedge$*R. complete R C2 Q*
shows *complete P* (*Seq C1 C2*) *Q*
⟨*proof*⟩

fun *construct-inv*
where
construct-inv P C 0 = *P*
| *construct-inv P C* (*Suc n*) = (λ*S.* ($\exists$ *S'*. *S* = *sem C S'* $\wedge$ *construct-inv P C n S'*))

lemma *iterate-sem-ind*:
assumes *construct-inv P C n S'*
shows $\exists$ *S. P S* $\wedge$ *S'* = *iterate-sem n C S*
⟨*proof*⟩

lemma *complete-while-aux*:
assumes *hyper-hoare-triple* (λ*S. P S* $\wedge$ *S* = *V*) (*While C*) *Q*
shows *entails* (*natural-partition* (*construct-inv* (λ*S. P S* $\wedge$ *S* = *V*) *C*)) *Q*
⟨*proof*⟩

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*⟩

Theorem 2

theorem *completeness*:
fixes $P\ Q$ :: $('lvar, 'lval, 'pvar, 'pval)$ *state hyperassertion*
assumes $\models \{P\}\ C\ \{Q\}$
shows $\vdash \{P\}\ C\ \{Q\}$
⟨*proof*⟩

3.4 Disproving Hyper-Triples

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

Theorem 4

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

definition *differ-only-by* **where**
differ-only-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* $a\ b\ x$
⟨*proof*⟩

lemma *diff-by-update*:
differ-only-by $(a(x := v))\ a\ x$
⟨*proof*⟩

lemma *diff-by-comm*:
differ-only-by $a\ b\ x \longleftrightarrow$ *differ-only-by* $b\ a\ x$
⟨*proof*⟩

lemma *diff-by-trans*:
assumes *differ-only-by* $a\ b\ x$
and *differ-only-by* $b\ c\ x$
shows *differ-only-by* $a\ c\ x$
⟨*proof*⟩

definition *not-free-var-of* **where**
not-free-var-of $P\ x \longleftrightarrow (\forall\ states\ states'.$

$(\forall\, i.\ \textit{differ-only-by}\ (\textit{fst}\ (\textit{states}\ i))\ (\textit{fst}\ (\textit{states}'\ i))\ x \wedge \textit{snd}\ (\textit{states}\ i) = \textit{snd}\ (\textit{states}'\ i))$
$\longrightarrow (\textit{states} \in P \longleftrightarrow \textit{states}' \in P))$

lemma *not-free-var-ofE*:
assumes *not-free-var-of P x*
and $\bigwedge i.\ \textit{differ-only-by}\ (\textit{fst}\ (\textit{states}\ i))\ (\textit{fst}\ (\textit{states}'\ i))\ x$
and $\bigwedge i.\ \textit{snd}\ (\textit{states}\ i) = \textit{snd}\ (\textit{states}'\ i)$
and $\textit{states} \in P$
shows $\textit{states}' \in P$
⟨*proof*⟩

3.5 Synchronized Rule for Branching

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

lemma *combineI*:
assumes $P1\ (\textit{Set.filter}\ (\lambda\varphi.\ \textit{fst}\ \varphi\ x = \textit{from-nat}\ 1)\ S) \wedge P2\ (\textit{Set.filter}\ (\lambda\varphi.\ \textit{fst}\ \varphi\ x = \textit{from-nat}\ 2)\ S)$
shows *combine from-nat x P1 P2 S*
⟨*proof*⟩

definition *modify-lvar-to* **where**
$\textit{modify-lvar-to}\ x\ v\ \varphi = ((\textit{fst}\ \varphi)(x := v),\ \textit{snd}\ \varphi)$

lemma *logical-var-in-sem-same*:
assumes $\bigwedge\varphi.\ \varphi \in S \Longrightarrow \textit{fst}\ \varphi\ x = a$
and $\varphi' \in \textit{sem}\ C\ S$
shows $\textit{fst}\ \varphi'\ x = a$
⟨*proof*⟩

lemma *recover-after-sem*:
assumes $a \neq b$
and $\bigwedge\varphi.\ \varphi \in S1 \Longrightarrow \textit{fst}\ \varphi\ x = a$
and $\bigwedge\varphi.\ \varphi \in S2 \Longrightarrow \textit{fst}\ \varphi\ x = b$
shows $\textit{sem}\ C\ S1 = \textit{Set.filter}\ (\lambda\varphi.\ \textit{fst}\ \varphi\ x = a)\ (\textit{sem}\ C\ (S1 \cup S2))$ (**is** $?A = ?B$)
⟨*proof*⟩

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 $\textit{Set.filter}\ (\lambda\varphi.\ \textit{fst}\ \varphi\ x = a)\ (S1' \cup S2') = S1'$ (**is** $?A = ?B$)
⟨*proof*⟩

definition *not-free-var-hyper* **where**
$\mathit{not\text{-}free\text{-}var\text{-}hyper}\ x\ P \longleftrightarrow (\forall S\ v.\ P\ S \longleftrightarrow P\ ((\mathit{modify\text{-}lvar\text{-}to}\ x\ v)\ `\ S))$

definition *injective* **where**
$\mathit{injective}\ f \longleftrightarrow (\forall a\ b.\ a \neq b \longrightarrow f\ a \neq f\ b)$

lemma *sem-of-modify-lvar*:
$\mathit{sem}\ C\ ((\mathit{modify\text{-}lvar\text{-}to}\ r\ v)\ `\ S) = (\mathit{modify\text{-}lvar\text{-}to}\ r\ v)\ `\ (\mathit{sem}\ C\ S)$ (**is** $?A = ?B$)
$\langle \mathit{proof} \rangle$

Proposition 15 (appendix C).

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

and $\mathit{not\text{-}free\text{-}var\text{-}hyper}\ x\ P1$
and $\mathit{not\text{-}free\text{-}var\text{-}hyper}\ x\ P2$
and $\mathit{injective}\ (\mathit{from\text{-}nat} :: \mathit{nat} \Rightarrow 'a)$

and $\mathit{not\text{-}free\text{-}var\text{-}hyper}\ x\ R1$
and $\mathit{not\text{-}free\text{-}var\text{-}hyper}\ x\ R2$

shows $\models \{P\}\ \mathit{If}\ (\mathit{Seq}\ C1\ (\mathit{Seq}\ C\ C1'))\ (\mathit{Seq}\ C2\ (\mathit{Seq}\ C\ C2'))\ \{\mathit{join}\ Q1\ Q2\}$
$\langle \mathit{proof} \rangle$

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

lemma *GNI-I*:
assumes $\bigwedge \varphi 1\ \varphi 2.\ \varphi 1 \in S \wedge \varphi 2 \in S$
$\Longrightarrow (\exists \varphi \in S.\ \mathit{snd}\ \varphi\ h = \mathit{snd}\ \varphi 1\ h \wedge \mathit{snd}\ \varphi\ l = \mathit{snd}\ \varphi 2\ l)$
shows $\mathit{GNI}\ l\ h\ S$
$\langle \mathit{proof} \rangle$

lemma *program-1-sat-gni*:
assumes $y \neq l \wedge y \neq h \wedge l \neq h$
shows $\vdash \{ (\lambda S.\ True) \}\ Seq\ (Havoc\ y)\ (Assign\ l\ (\lambda\sigma.\ (\sigma\ h :: int) + \sigma\ y))\ \{ GNI\ l\ h \}$
$\langle proof \rangle$

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

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 $'a\ hyperproperty = ('a \times 'a)\ set \Rightarrow bool$

definition *set-of-traces* **where**
$set\text{-}of\text{-}traces\ C = \{ (\sigma, \sigma') \mid \sigma\ \sigma'.\ \langle C, \sigma\rangle \rightarrow \sigma' \}$

definition *hypersat* **where**
$hypersat\ C\ H \longleftrightarrow H\ (set\text{-}of\text{-}traces\ C)$

definition *copy-p-state* **where**
$copy\text{-}p\text{-}state\ to\text{-}pvar\ to\text{-}lval\ \sigma\ x = to\text{-}lval\ (\sigma\ (to\text{-}pvar\ x))$

definition *recover-p-state* **where**
$recover\text{-}p\text{-}state\ to\text{-}pval\ to\text{-}lvar\ l\ x = to\text{-}pval\ (l\ (to\text{-}lvar\ x))$

lemma *injective-then-exists-inverse*:
assumes *injective to-lvar*
shows $\exists\, to\text{-}pvar.\ (\forall x.\ to\text{-}pvar\ (to\text{-}lvar\ x) = x)$
$\langle proof \rangle$

lemma *single-step-then-in-sem*:
  assumes *single-sem* $C\ \sigma\ \sigma'$
    and $(l, \sigma) \in S$
   shows $(l, \sigma') \in$ *sem* $C\ S$
  $\langle$*proof*$\rangle$

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

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* $C\ S$
  $\langle$*proof*$\rangle$

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* $\sigma, \sigma)\ |\sigma.$ *True*$\}$
   shows $\{($*recover-p-state to-pval to-lvar* $l, \sigma')\ |l\ \sigma'.\ (l, \sigma') \in$ *sem* $C\ S\} =$ *set-of-traces* C
(**is** $?A = ?B$)
$\langle$*proof*$\rangle$

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* $C\ H \longleftrightarrow\ \models \{P\}\ C\ \{Q\}$)
$\langle$*proof*$\rangle$

Hypersafety, hyperliveness

definition *max-k* **where**
  max-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-hypersafety* **where**
  k-hypersafety $k\ P \longleftrightarrow (\forall\, S.\ \neg\ P\ S \longrightarrow (\exists\, S'.\ S' \subseteq S \wedge$ *max-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-hypersafetyI*:
assumes $\bigwedge S. \neg P\ S \Longrightarrow \exists S'. S' \subseteq S \wedge$ *max-k* $k\ S' \wedge (\forall S''. S' \subseteq S'' \longrightarrow \neg P\ S'')$
shows *k-hypersafety* $k\ P$
⟨*proof*⟩

lemma *hypersafetyI*:
assumes $\bigwedge S\ S'. \neg P\ S \Longrightarrow S \subseteq S' \Longrightarrow \neg P\ S'$
shows *hypersafety* P
⟨*proof*⟩

lemma *hyperlivenessI*:
assumes $\bigwedge S. \exists S'. S \subseteq S' \wedge P\ S'$
shows *hyperliveness* P
⟨*proof*⟩

lemma *k-hypersafe-is-hypersafe*:
assumes *k-hypersafety* $k\ P$
shows *hypersafety* P
⟨*proof*⟩

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*⟩

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* $P\ Q$)
⟨*proof*⟩

theorem *hypersafety-1-hoare-logic*:
k-hypersafety 1 (*hoarify* $P\ Q$)
⟨*proof*⟩

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*⟩

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

lemma *real-incorrectnessify-liveness*:
assumes $Q \neq \{\}$
shows $hyperliveness\ (real\text{-}incorrectnessify\ P\ Q)$
⟨*proof*⟩

Verifying GNI

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

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

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

Footnote 4

theorem *any-hht-hyperprop*:
$\models \{P\}\ C\ \{Q\} \longleftrightarrow hypersat\ C\ (hyperprop\text{-}hht\ P\ Q)$ (**is** $?A \longleftrightarrow ?B$)
⟨*proof*⟩

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 $HL\ P\ C\ Q$
⟨*proof*⟩

lemma *hoarifyI*:
  assumes $\bigwedge\sigma\ \sigma'.\ (\sigma,\ \sigma') \in S \Longrightarrow \sigma \in P \Longrightarrow \sigma' \in Q$
  shows *hoarify P Q S*
  ⟨*proof*⟩

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

lemma *connection-HL*:
  $\mathit{HL}\ P\ C\ Q \longleftrightarrow \mathit{HL\text{-}hyperprop}\ P\ Q\ (\mathit{set\text{-}of\text{-}traces}\ C)$ (**is** $?A \longleftrightarrow\ ?B$)
⟨*proof*⟩

Proposition 1

theorem *HL-expresses-hyperproperties*:
  $\exists\, H.\ (\forall\, C.\ \mathit{hypersat}\ C\ H \longleftrightarrow \mathit{HL}\ P\ C\ Q) \wedge \mathit{k\text{-}hypersafety}\ 1\ H$
⟨*proof*⟩

Proposition 2

theorem *encoding-HL*:
  $\mathit{HL}\ P\ C\ Q \longleftrightarrow (\mathit{hyper\text{-}hoare\text{-}triple}\ (\mathit{over\text{-}approx}\ P)\ C\ (\mathit{over\text{-}approx}\ Q))$ (**is** $?A \longleftrightarrow\ ?B$)
⟨*proof*⟩

lemma *entailment-order-hoare*:
  assumes $P \subseteq P'$
  shows $\mathit{entails}\ (\mathit{over\text{-}approx}\ P)\ (\mathit{over\text{-}approx}\ P')$
  ⟨*proof*⟩

5.3 Cartesian Hoare Logic (CHL) [9]

Notation 3

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

lemma *k-semI*:
  assumes $\bigwedge i.\ (\mathit{fst}\ (\mathit{states}\ i) = \mathit{fst}\ (\mathit{states}'\ i) \wedge \mathit{single\text{-}sem}\ C\ (\mathit{snd}\ (\mathit{states}\ i))\ (\mathit{snd}\ (\mathit{states}'\ i)))$
  shows $\mathit{k\text{-}sem}\ C\ \mathit{states}\ \mathit{states}'$
  ⟨*proof*⟩

lemma *k-semE*:
  assumes $\mathit{k\text{-}sem}\ C\ \mathit{states}\ \mathit{states}'$
  shows $\mathit{fst}\ (\mathit{states}\ i) = \mathit{fst}\ (\mathit{states}'\ i) \wedge \mathit{single\text{-}sem}\ C\ (\mathit{snd}\ (\mathit{states}\ i))\ (\mathit{snd}\ (\mathit{states}'\ i))$
  ⟨*proof*⟩

Definition 9

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

lemma *CHLI*:
assumes $\bigwedge states\ states'.\ states \in P \Longrightarrow k\text{-}sem\ C\ states\ states' \Longrightarrow states' \in Q$
shows $CHL\ P\ C\ Q$
⟨*proof*⟩

lemma *CHLE*:
assumes $CHL\ P\ C\ Q$
and $states \in P$
and $k\text{-}sem\ C\ states\ states'$
shows $states' \in Q$
⟨*proof*⟩

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

lemma *encode-CHLI*:
assumes $\bigwedge states.\ (\forall\, i.\ states\ i \in S \land fst\ (states\ i)\ x = from\text{-}nat\ i) \Longrightarrow states \in P$
shows $encode\text{-}CHL\ from\text{-}nat\ x\ P\ S$
⟨*proof*⟩

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 $states \in P$
⟨*proof*⟩

lemma *equal-change-lvar*:
assumes $fst\ \varphi\ x = y$
shows $\varphi = ((fst\ \varphi)(x := y),\ snd\ \varphi)$
⟨*proof*⟩

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*⟩

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

lemma *CHL-hyperpropI*:
assumes $\bigwedge l\ p.\ (\forall i.\ p\ i \in S) \wedge (\lambda i.\ (l\ i,\ fst\ (p\ i))) \in P \Longrightarrow (\lambda i.\ (l\ i,\ snd\ (p\ i))) \in Q$
shows $CHL\text{-}hyperprop\ P\ Q\ S$
$\langle proof \rangle$

lemma *CHL-hyperpropE*:
assumes $CHL\text{-}hyperprop\ P\ Q\ S$
and $\bigwedge i.\ p\ i \in S$
and $(\lambda i.\ (l\ i,\ fst\ (p\ i))) \in P$
shows $(\lambda i.\ (l\ i,\ snd\ (p\ i))) \in Q$
$\langle proof \rangle$

Proposition 10

theorem *CHL-hyperproperty*:
$hypersat\ C\ (CHL\text{-}hyperprop\ P\ Q) \longleftrightarrow CHL\ P\ C\ Q$ (**is** $?A \longleftrightarrow ?B$)
$\langle proof \rangle$

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 $k\text{-}hypersafety\ k\ (CHL\text{-}hyperprop\ P\ Q)$
$\langle proof \rangle$

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*:
$IL\ P\ C\ Q \longleftrightarrow (\forall l\ \sigma'.\ (l,\ \sigma') \in Q \longrightarrow (\exists \sigma.\ (l,\ \sigma) \in P \wedge \langle C,\ \sigma\rangle \rightarrow \sigma'))$
$\langle proof \rangle$

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

lemma *IL-hyperpropI*:
assumes $\bigwedge l\ \sigma'.\ (l,\ \sigma') \in Q \Longrightarrow (\exists \sigma.\ (l,\ \sigma) \in P \wedge (\sigma,\ \sigma') \in S)$
shows $IL\text{-}hyperprop\ P\ Q\ S$
$\langle proof \rangle$

Proposition 11

lemma *IL-expresses-hyperproperties*:
$IL\ P\ C\ Q \longleftrightarrow IL\text{-}hyperprop\ P\ Q\ (set\text{-}of\text{-}traces\ C)$ (**is** $?A \longleftrightarrow ?B$)
⟨*proof*⟩

lemma *IL-consequence*:
assumes $IL\ P\ C\ Q$
and $(l, \sigma') \in Q$
shows $\exists \sigma.\ (l, \sigma) \in P \wedge single\text{-}sem\ C\ \sigma\ \sigma'$
⟨*proof*⟩

Proposition 4

theorem *encoding-IL*:
$IL\ P\ C\ Q \longleftrightarrow (hyper\text{-}hoare\text{-}triple\ (under\text{-}approx\ P)\ C\ (under\text{-}approx\ Q))$ (**is** $?A \longleftrightarrow ?B$)
⟨*proof*⟩

lemma *entailment-order-reverse-hoare*:
assumes $P \subseteq P'$
shows $entails\ (under\text{-}approx\ P')\ (under\text{-}approx\ P)$
⟨*proof*⟩

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

lemma *incorrectifyI*:
assumes $\bigwedge\sigma.\ p\ \sigma \Longrightarrow \sigma \in S$
shows $incorrectify\ p\ S$
⟨*proof*⟩

lemma *incorrectifyE*:
assumes $incorrectify\ p\ S$
and $p\ \sigma$
shows $\sigma \in S$
⟨*proof*⟩

lemma *simple-while-incorrectness*:
assumes $\bigwedge n.\ hyper\text{-}hoare\text{-}triple\ (incorrectify\ (p\ n))\ C\ (incorrectify\ (p\ (Suc\ n)))$
shows $hyper\text{-}hoare\text{-}triple\ (incorrectify\ (p\ 0))\ (While\ C)\ (incorrectify\ (\lambda\sigma.\ \exists n.\ p\ n\ \sigma))$
⟨*proof*⟩

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

theorem *incorrectness-hyperliveness*:
assumes $\bigwedge l.\ sat\text{-}for\text{-}l\ l\ Q \Longrightarrow sat\text{-}for\text{-}l\ l\ P$
shows $hyperliveness\ (IL\text{-}hyperprop\ P\ Q)$

$\langle proof \rangle$

5.5 Relational Incorrectness Logic [7] (RIL)

Definition 11

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

lemma *RILI*:
assumes $\bigwedge states'.\ states' \in Q \Longrightarrow (\exists\, states \in P.\ k\text{-}sem\ C\ states\ states')$
shows $RIL\ P\ C\ Q$
$\langle proof \rangle$

lemma *RILE*:
assumes $RIL\ P\ C\ Q$
and $states' \in Q$
shows $\exists\, states \in P.\ k\text{-}sem\ C\ states\ states'$
$\langle proof \rangle$

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

lemma *RIL-hyperpropI*:
assumes $\bigwedge l\ states'.\ (\lambda i.\ (l\ i,\ states'\ i)) \in Q \Longrightarrow (\exists\, states.\ (\lambda i.\ (l\ i,\ states\ i)) \in P \land (\forall\, i.\ (states\ i,\ states'\ i) \in S))$
shows $RIL\text{-}hyperprop\ P\ Q\ S$
$\langle proof \rangle$

lemma *RIL-hyperpropE*:
assumes $RIL\text{-}hyperprop\ P\ Q\ S$
and $(\lambda i.\ (l\ i,\ states'\ i)) \in Q$
shows $\exists\, states.\ (\lambda i.\ (l\ i,\ states\ i)) \in P \land (\forall\, i.\ (states\ i,\ states'\ i) \in S)$
$\langle proof \rangle$

lemma *useful*:
$states' = (\lambda i.\ ((fst \circ states')\ i,\ (snd \circ states')\ i))$
$\langle proof \rangle$

Proposition 12

theorem *RIL-expresses-hyperproperties*:
$hypersat\ C\ (RIL\text{-}hyperprop\ P\ Q) \longleftrightarrow RIL\ P\ C\ Q$ (**is** $?A \longleftrightarrow ?B$)
$\langle proof \rangle$

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

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*⟩

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-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-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-nat i*)
  $\Longrightarrow$ ($\exists$ *r*. $\forall$ *i*. ((*fst* (*states i*))(*c* := *r*), *snd* (*states i*)) $\in$ *S*)
      and $\bigwedge$*states*. ($\forall$ *i*. *states i* $\in$ *S*) $\Longrightarrow$ ($\forall$ *i*. *fst* (*states i*) *x* = *from-nat i*) $\Longrightarrow$
($\forall$ *i j*. *fst* (*states i*) *c* = *fst* (*states j*) *c*) $\Longrightarrow$ *states* $\in$ *P*
    shows *strong-pre-insec from-nat x c P S*
  ⟨*proof*⟩

lemma *strong-pre-insecE*:
  assumes *strong-pre-insec from-nat x c P S*
      and $\bigwedge$*i*. *states i* $\in$ *S*
      and $\bigwedge$*i*. *fst* (*states i*) *x* = *from-nat i*
      and $\bigwedge$*i j*. *fst* (*states i*) *c* = *fst* (*states j*) *c*
    shows *states* $\in$ *P*
  ⟨*proof*⟩

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

lemma *pre-insecI*:
  assumes $\bigwedge$*states*. *states* $\in$ *P* $\Longrightarrow$ ($\forall$ *i*. *fst* (*states i*) *x* = *from-nat i*)
  $\Longrightarrow$ ($\exists$ *r*. $\forall$ *i*. ((*fst* (*states i*))(*c* := *r*), *snd* (*states i*)) $\in$ *S*)
    shows *pre-insec from-nat x c P S*
  ⟨*proof*⟩

lemma *strong-pre-implies-pre*:
  assumes *strong-pre-insec from-nat x c P S*
  shows *pre-insec from-nat x c P S*
  ⟨*proof*⟩

lemma *pre-insecE*:
  assumes *pre-insec from-nat x c P S*
      and *states* $\in$ *P*

and $\bigwedge i.\ \mathit{fst}\ (\mathit{states}\ i)\ x = \mathit{from\text{-}nat}\ i$
shows $\exists\, r.\ \forall\, i.\ ((\mathit{fst}\ (\mathit{states}\ i))(c := r),\ \mathit{snd}\ (\mathit{states}\ i)) \in S$
$\langle \mathit{proof} \rangle$

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 $\mathit{post\text{-}insec}\ \mathit{from\text{-}nat}\ x\ c\ Q\ S$
and $\mathit{states} \in Q$
and $\bigwedge i.\ \mathit{fst}\ (\mathit{states}\ i)\ x = \mathit{from\text{-}nat}\ i$
shows $\exists\, r.\ (\forall\, i.\ ((\mathit{fst}\ (\mathit{states}\ i))(c := r),\ \mathit{snd}\ (\mathit{states}\ i)) \in S)$
$\langle \mathit{proof} \rangle$

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 $\mathit{post\text{-}insec}\ \mathit{from\text{-}nat}\ x\ c\ Q\ S$
$\langle \mathit{proof} \rangle$

lemma *same-pre-post*:
$\mathit{pre\text{-}insec}\ \mathit{from\text{-}nat}\ x\ c\ Q\ S \longleftrightarrow \mathit{post\text{-}insec}\ \mathit{from\text{-}nat}\ x\ c\ Q\ S$
$\langle \mathit{proof} \rangle$

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 $\mathit{injective}\ \mathit{from\text{-}nat}$
shows $\mathit{sat}\ (\mathit{strong\text{-}pre\text{-}insec}\ \mathit{from\text{-}nat}\ x\ c\ (P :: ({}'a \Rightarrow ({}'\mathit{lvar} \Rightarrow {}'\mathit{lval}) \times ({}'\mathit{pvar} \Rightarrow {}'\mathit{pval}))\ \mathit{set}))$
$\langle \mathit{proof} \rangle$

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

shows $\mathit{RIL}\ P\ C\ Q \longleftrightarrow\ \models \{\mathit{pre\text{-}insec}\ \mathit{from\text{-}nat}\ x\ c\ P\}\ C\ \{\mathit{post\text{-}insec}\ \mathit{from\text{-}nat}\ x\ c\ Q\}$ (**is** $?A \longleftrightarrow ?B$)

$\langle proof \rangle$

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 $injective\ from\text{-}nat$
and $not\text{-}free\text{-}var\text{-}of\ (P :: ('a \Rightarrow ('lvar \Rightarrow 'lval) \times ('pvar \Rightarrow 'pval))\ set)\ x \wedge not\text{-}free\text{-}var\text{-}of\ P\ c$
and $not\text{-}free\text{-}var\text{-}of\ Q\ x \wedge not\text{-}free\text{-}var\text{-}of\ Q\ c$
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** $?A \longleftrightarrow ?B$)
$\langle proof \rangle$

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 $FU\ P\ C\ Q$
$\langle proof \rangle$

definition *encode-FU* **where**
$encode\text{-}FU\ P\ S \longleftrightarrow P \cap S \neq \{\}$

Proposition 6

theorem *encoding-FU*:
$FU\ P\ C\ Q \longleftrightarrow \models \{encode\text{-}FU\ P\}\ C\ \{encode\text{-}FU\ Q\}$ (**is** $?A \longleftrightarrow ?B$)
$\langle proof \rangle$

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

lemma *hyperprop-FUI*:
assumes $\bigwedge l\ \sigma.\ (l,\ \sigma) \in P \Longrightarrow (\exists \sigma'.\ (l,\ \sigma') \in Q \wedge (\sigma,\ \sigma') \in S)$
shows $hyperprop\text{-}FU\ P\ Q\ S$
$\langle proof \rangle$

lemma *hyperprop-FUE*:
assumes $hyperprop\text{-}FU\ P\ Q\ S$
and $(l,\ \sigma) \in P$

shows $\exists\,\sigma'.\ (l,\ \sigma') \in Q \wedge (\sigma,\ \sigma') \in S$
$\langle proof \rangle$

theorem *FU-expresses-hyperproperties*:
$hypersat\ C\ (hyperprop\text{-}FU\ P\ Q) \longleftrightarrow FU\ P\ C\ Q$ (**is** $?A \longleftrightarrow\ ?B$)
$\langle proof \rangle$

theorem *hyperliveness-hyperprop-FU*:
assumes $\bigwedge l.\ sat\text{-}for\text{-}l\ l\ P \Longrightarrow sat\text{-}for\text{-}l\ l\ Q$
shows $hyperliveness\ (hyperprop\text{-}FU\ P\ Q)$
$\langle proof \rangle$

No relationship between incorrectness and forward underapproximation

lemma *incorrectness-does-not-imply-FU*:
assumes *injective from-nat*
assumes $P = \{(l,\ \sigma)\ |\sigma\ l.\ \sigma\ x = from\text{-}nat\ (0 :: nat) \vee \sigma\ x = from\text{-}nat\ 1\}$
and $Q = \{(l,\ \sigma)\ |\sigma\ l.\ \sigma\ x = from\text{-}nat\ 1\}$
and $C = Assume\ (\lambda\sigma.\ \sigma\ x = from\text{-}nat\ 1)$
shows $IL\ P\ C\ Q$
and $\neg\ FU\ P\ C\ Q$
$\langle proof \rangle$

lemma *FU-does-not-imply-incorrectness*:
assumes $P = \{(l,\ \sigma)\ |\sigma\ l.\ \sigma\ x = from\text{-}nat\ (0 :: nat) \vee \sigma\ x = from\text{-}nat\ 1\}$
and $Q = \{(l,\ \sigma)\ |\sigma\ l.\ \sigma\ x = from\text{-}nat\ 1\}$
assumes *injective from-nat*
shows $\neg\ IL\ Q\ Skip\ P$
and $FU\ Q\ Skip\ P$
$\langle proof \rangle$

5.7 Relational Forward Underapproximate logic

Definition 13

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

lemma *RFUI*:
assumes $\bigwedge states.\ states \in P \Longrightarrow (\exists\, states' \in Q.\ k\text{-}sem\ C\ states\ states')$
shows $RFU\ P\ C\ Q$
$\langle proof \rangle$

lemma *RFUE*:
assumes $RFU\ P\ C\ Q$
and $states \in P$
shows $\exists\, states' \in Q.\ k\text{-}sem\ C\ states\ states'$
$\langle proof \rangle$

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*)
$\langle$*proof*$\rangle$

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

lemma *RFU-hyperpropI*:
assumes $\bigwedge$*l states*. (λ*i*. (*l i*, *states i*)) $\in$ *P* $\Longrightarrow$ ($\exists$ *states'*. (λ*i*. (*l i*, *states' i*)) $\in$ *Q* $\wedge$ ($\forall$ *i*. (*states i*, *states' i*) $\in$ *S*))
shows *RFU-hyperprop P Q S*
$\langle$*proof*$\rangle$

lemma *RFU-hyperpropE*:
assumes *RFU-hyperprop P Q S*
and (λ*i*. (*l i*, *states i*)) $\in$ *P*
shows $\exists$ *states'*. (λ*i*. (*l i*, *states' i*)) $\in$ *Q* $\wedge$ ($\forall$ *i*. (*states i*, *states' i*) $\in$ *S*)
$\langle$*proof*$\rangle$

Proposition 13

theorem *RFU-captures-hyperproperties*:
hypersat C (*RFU-hyperprop P Q*) $\longleftrightarrow$ *RFU P C Q* (**is** *?A* $\longleftrightarrow$ *?B*)
$\langle$*proof*$\rangle$

theorem *hyperliveness-encode-RFU*:
assumes $\bigwedge$*l*. *k-sat-for-l l P* $\Longrightarrow$ *k-sat-for-l l Q*
shows *hyperliveness* (*RFU-hyperprop P Q*)
$\langle$*proof*$\rangle$

5.8 Relational Universal Existential (RUE) [4]

Definition 14

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

lemma *RUE-I*:
assumes $\bigwedge$$\sigma$*1* σ*2* σ*1'*. (σ*1*, σ*2*) $\in$ *P* $\Longrightarrow$ *k-sem C* σ*1* σ*1'* $\Longrightarrow$ ($\exists$ σ*2'*. *k-sem C* σ*2* σ*2'* $\wedge$ (σ*1'*, σ*2'*) $\in$ *Q*)

shows $RUE\ P\ C\ Q$
$\langle proof \rangle$

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 $\exists \sigma 2'.\ k\text{-}sem\ C\ \sigma 2\ \sigma 2' \wedge (\sigma 1', \sigma 2') \in Q$
$\langle proof \rangle$

Hyperproperty

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

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 $hyperprop\text{-}RUE\ P\ Q\ S$
$\langle proof \rangle$

lemma *hyperprop-RUE-E*:
assumes $hyperprop\text{-}RUE\ P\ Q\ S$
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 $\exists \sigma 2'.\ (\forall k.\ (\sigma 2\ k, \sigma 2'\ k) \in S) \wedge (\lambda i.\ (l1\ i, \sigma 1'\ i), \lambda k.\ (l2\ k, \sigma 2'\ k)) \in Q$
$\langle proof \rangle$

Proposition 14

theorem *RUE-express-hyperproperties*:
$RUE\ P\ C\ Q \longleftrightarrow hypersat\ C\ (hyperprop\text{-}RUE\ P\ Q)$ (**is** $?A \longleftrightarrow ?B$)
$\langle proof \rangle$

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

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 $is\text{-}type\ type\ fn\ x\ t\ S\ \sigma$
$\langle proof \rangle$

lemma *is-type-E*:
assumes $is\text{-}type\ type\ fn\ x\ t\ S\ \sigma$

shows $\sigma\ i \in S \wedge \mathit{fst}\ (\sigma\ i)\ t = \mathit{type} \wedge \mathit{fst}\ (\sigma\ i)\ x = \mathit{fn}\ i$
$\langle \mathit{proof} \rangle$

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

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

lemma *encode-RUE-1-E1*:
assumes $\mathit{encode\text{-}RUE\text{-}1}\ \mathit{fn}\ \mathit{fn1}\ \mathit{fn2}\ x\ t\ P\ S$
shows $\exists \sigma \in S.\ \mathit{fst}\ \sigma\ x = \mathit{fn2}\ k \wedge \mathit{fst}\ \sigma\ t = \mathit{fn}\ 2$
$\langle \mathit{proof} \rangle$

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

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

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

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

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

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*
⟨*proof*⟩

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 $\sigma' \in P$
⟨*proof*⟩

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*⟩

5.9 Program Refinement

lemma *sem-assign-single*:
sem (*Assign x e*) $\{(l, \sigma)\} = \{(l, \sigma(x := e\ \sigma))\}$ (**is** *?A* = *?B*)
⟨*proof*⟩

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

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

lemma *not-free-var-stmtE-1*:
assumes $not\text{-}free\text{-}var\text{-}stmt\ x\ C$
and $(\sigma, \sigma') \in set\text{-}of\text{-}traces\ C$
shows $(\sigma(x := v), \sigma'(x := v)) \in set\text{-}of\text{-}traces\ C$
$\langle proof \rangle$

lemma *not-free-in-sem-same-val*:
assumes $not\text{-}free\text{-}var\text{-}stmt\ x\ C$
and $single\text{-}sem\ C\ \sigma\ \sigma'$
shows $\sigma\ x = \sigma'\ x$
$\langle proof \rangle$

lemma *not-free-in-sem-equiv*:
assumes $not\text{-}free\text{-}var\text{-}stmt\ x\ C$
and $single\text{-}sem\ C\ \sigma\ \sigma'$
shows $single\text{-}sem\ C\ (\sigma(x := v))\ (\sigma'(x := v))$
$\langle proof \rangle$

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 $not\text{-}free\text{-}var\text{-}stmt\ x\ C1$
and $not\text{-}free\text{-}var\text{-}stmt\ x\ C2$
shows $refinement\ C1\ C2 \longleftrightarrow$
$\models \{\ P\ \}\ If\ (Seq\ (Assign\ (x :: 'pvar)\ (\lambda\text{-}.\ a))\ C1)\ (Seq\ (Assign\ x\ (\lambda\text{-}.\ b))\ C2)\ \{\ Q\ \}$
(**is** $?A \longleftrightarrow ?B$)
$\langle proof \rangle$

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.