An Under-Approximate Relational Logic

Toby Murray

March 17, 2025

Abstract

Recently, authors have proposed *under-approximate* logics for reasoning about programs [4, 2]. So far, all such logics have been confined to reasoning about individual program behaviours. Yet there exist many over-approximate *relational* logics for reasoning about pairs of programs and relating their behaviours.

We present the first under-approximate relational logic, for the simple imperative language IMP. We prove our logic is both sound and complete. Additionally, we show how reasoning in this logic can be decomposed into non-relational reasoning in an under-approximate Hoare logic, mirroring Beringer's result for over-approximate relational logics. We illustrate the application of our logic on some small examples in which we provably demonstrate the presence of insecurity.

These proofs accompany a paper [3] that explains the results in more detail.

Contents

theory *RelationalIncorrectness*
imports *HOL−IMP.Big-Step*
begin

1 Under-Approximate Relational Judgement

This is the relational analogue of OHearn's [4] and de Vries & Koutavas' [2] judgements.

Note that in our case it doesn't really make sense to talk about "erroneous" states: the presence of an error can be seen only by the violation of a relation. Unlike O'Hearn, we cannot encode it directly into the semantics of our programs, without giving them a relational semantics. We use the standard big step semantics of IMP unchanged.

type-synonym *rassn* = *state* ⇒ *state* ⇒ *bool*

definition
ir-valid :: *rassn* ⇒ *com* ⇒ *com* ⇒ *rassn* ⇒ *bool*
where
$ir\text{-}valid\ P\ c\ c'\ Q \equiv (\forall\ t\ t'.\ Q\ t\ t' \longrightarrow (\exists\ s\ s'.\ P\ s\ s' \land (c,s) \Rightarrow t \land (c',s') \Rightarrow t'))$

2 Rules of the Logic

definition
flip :: *rassn* ⇒ *rassn*
where
$flip\ P \equiv \lambda s\ s'.\ P\ s'\ s$

inductive
ir-hoare :: *rassn* ⇒ *com* ⇒ *com* ⇒ *rassn* ⇒ *bool*
where
$ir\text{-}Skip$: $(\bigwedge t\ t'.\ Q\ t\ t' \Longrightarrow \exists\ s'.\ P\ t\ s' \land (c',s') \Rightarrow t') \Longrightarrow$
$ir\text{-}hoare\ P\ SKIP\ c'\ Q$ |
$ir\text{-}If\text{-}True$: $ir\text{-}hoare\ (\lambda s\ s'.\ P\ s\ s' \land bval\ b\ s)\ c_1\ c'\ Q \Longrightarrow$
$ir\text{-}hoare\ P\ (IF\ b\ THEN\ c_1\ ELSE\ c_2)\ c'\ Q$ |
$ir\text{-}If\text{-}False$: $ir\text{-}hoare\ (\lambda s\ s'.\ P\ s\ s' \land \neg\ bval\ b\ s)\ c_2\ c'\ Q \Longrightarrow$
$ir\text{-}hoare\ P\ (IF\ b\ THEN\ c_1\ ELSE\ c_2)\ c'\ Q$ |
$ir\text{-}Seq1$: $ir\text{-}hoare\ P\ c\ c'\ Q \Longrightarrow ir\text{-}hoare\ Q\ d\ SKIP\ R \Longrightarrow ir\text{-}hoare\ P\ (Seq\ c\ d)\ c'\ R$ |

ir-Assign: *ir-hoare* $(\lambda t\ t'.\ \exists\ v.\ P\ (t(x := v))\ t' \wedge (t\ x) = aval\ e\ (t(x := v)))$ *SKIP* $c'\ Q \Longrightarrow$
ir-hoare P (*Assign* $x\ e$) $c'\ Q$ |
ir-While-False: *ir-hoare* $(\lambda s\ s'.\ P\ s\ s' \wedge \neg\ bval\ b\ s)$ *SKIP* $c'\ Q \Longrightarrow$
ir-hoare P (*WHILE* b *DO* c) $c'\ Q$ |
ir-While-True: *ir-hoare* $(\lambda s\ s'.\ P\ s\ s' \wedge bval\ b\ s)$ (c;; *WHILE* b *DO* c) $c'\ Q \Longrightarrow$
ir-hoare P (*WHILE* b *DO* c) $c'\ Q$ |
ir-While-backwards-frontier: $(\bigwedge n.$ *ir-hoare* $(\lambda\ s\ s'.\ P\ n\ s\ s' \wedge bval\ b\ s)\ c$ *SKIP* $(P\ (Suc\ n))) \Longrightarrow$
ir-hoare $(\lambda s\ s'.\ \exists n.\ P\ n\ s\ s')$ (*WHILE* b *DO* c) $c'\ Q \Longrightarrow$
ir-hoare $(P\ 0)$ (*WHILE* b *DO* c) $c'\ Q$ |
ir-conseq: *ir-hoare* $P\ c\ c'\ Q \Longrightarrow (\bigwedge s\ s'.\ P\ s\ s' \Longrightarrow P'\ s\ s') \Longrightarrow (\bigwedge s\ s'.\ Q'\ s\ s' \Longrightarrow Q\ s\ s') \Longrightarrow$
ir-hoare $P'\ c\ c'\ Q'$ |
ir-disj: *ir-hoare* $P_1\ c\ c'\ Q_1 \Longrightarrow$ *ir-hoare* $P_2\ c\ c'\ Q_2 \Longrightarrow$
ir-hoare $(\lambda s\ s'.\ P_1\ s\ s' \vee P_2\ s\ s')\ c\ c'\ (\lambda\ t\ t'.\ Q_1\ t\ t' \vee Q_2\ t\ t')$ |
ir-sym: *ir-hoare* (*flip* P) $c\ c'$ (*flip* Q) $\Longrightarrow$ *ir-hoare* $P\ c'\ c\ Q$

3 Simple Derived Rules

lemma *meh-simp*[*simp*]: $(SKIP,\ s') \Rightarrow t' = (s' = t')$
⟨*proof*⟩

lemma *ir-pre*: *ir-hoare* $P\ c\ c'\ Q \Longrightarrow (\bigwedge s\ s'.\ P\ s\ s' \Longrightarrow P'\ s\ s') \Longrightarrow$
ir-hoare $P'\ c\ c'\ Q$
⟨*proof*⟩

lemma *ir-post*: *ir-hoare* $P\ c\ c'\ Q \Longrightarrow (\bigwedge s\ s'.\ Q'\ s\ s' \Longrightarrow Q\ s\ s') \Longrightarrow$
ir-hoare $P\ c\ c'\ Q'$
⟨*proof*⟩

4 Soundness

lemma *Skip-ir-valid*[*intro*]:
$(\bigwedge t\ t'.\ Q\ t\ t' \Longrightarrow \exists s'.\ P\ t\ s' \wedge (c',\ s') \Rightarrow t') \Longrightarrow$ *ir-valid* P *SKIP* $c'\ Q$
⟨*proof*⟩

lemma *sym-ir-valid*[*intro*]:
ir-valid (*flip* P) $c'\ c$ (*flip* Q) $\Longrightarrow$ *ir-valid* $P\ c\ c'\ Q$
⟨*proof*⟩

lemma *If-True-ir-valid*[*intro*]:
ir-valid $(\lambda a\ c.\ P\ a\ c \wedge bval\ b\ a)\ c_1\ c'\ Q \Longrightarrow$
ir-valid P (*IF* b *THEN* c_1 *ELSE* c_2) $c'\ Q$
⟨*proof*⟩

lemma *If-False-ir-valid*[*intro*]:

$ir\text{-}valid\ (\lambda a\ c.\ P\ a\ c \wedge \neg\ bval\ b\ a)\ c_2\ c'\ Q \Longrightarrow$
$ir\text{-}valid\ P\ (IF\ b\ THEN\ c_1\ ELSE\ c_2)\ c'\ Q$
⟨*proof*⟩

lemma *Seq1-ir-valid*[*intro*]:
$ir\text{-}valid\ P\ c\ c'\ Q \Longrightarrow ir\text{-}valid\ Q\ d\ SKIP\ R \Longrightarrow ir\text{-}valid\ P\ (c;;\ d)\ c'\ R$
⟨*proof*⟩

lemma *Seq2-ir-valid*[*intro*]:
$ir\text{-}valid\ P\ c\ SKIP\ Q \Longrightarrow ir\text{-}valid\ Q\ d\ c'\ R \Longrightarrow ir\text{-}valid\ P\ (c;;\ d)\ c'\ R$
⟨*proof*⟩

lemma *Seq-ir-valid*[*intro*]:
$ir\text{-}valid\ P\ c\ c'\ Q \Longrightarrow ir\text{-}valid\ Q\ d\ d'\ R \Longrightarrow ir\text{-}valid\ P\ (c;;\ d)\ (c';;\ d')\ R$
⟨*proof*⟩

lemma *Assign-blah*[*intro*]:
$t\ x = aval\ e\ (t(x := v))$
$\Longrightarrow (x ::= e,\ t(x := v)) \Rightarrow t$
⟨*proof*⟩

lemma *Assign-ir-valid*[*intro*]:
$ir\text{-}valid\ (\lambda t\ t'.\ \exists\ v.\ P\ (t(x := v))\ t' \wedge (t\ x) = aval\ e\ (t(x := v)))\ SKIP\ c'\ Q \Longrightarrow$
$ir\text{-}valid\ P\ (Assign\ x\ e)\ c'\ Q$
⟨*proof*⟩

lemma *While-False-ir-valid*[*intro*]:
$ir\text{-}valid\ (\lambda s\ s'.\ P\ s\ s' \wedge \neg\ bval\ b\ s)\ SKIP\ c'\ Q \Longrightarrow$
$ir\text{-}valid\ P\ (WHILE\ b\ DO\ c)\ c'\ Q$
⟨*proof*⟩

lemma *While-True-ir-valid*[*intro*]:
$ir\text{-}valid\ (\lambda s\ s'.\ P\ s\ s' \wedge bval\ b\ s)\ (Seq\ c\ (WHILE\ b\ DO\ c))\ c'\ Q \Longrightarrow$
$ir\text{-}valid\ P\ (WHILE\ b\ DO\ c)\ c'\ Q$
⟨*proof*⟩

lemma *While-backwards-frontier-ir-valid'*:
assumes *asm*: $\bigwedge n.\ \forall\ t\ t'.\ P\ (k + Suc\ n)\ t\ t' \longrightarrow$
$(\exists\ s.\ P\ (k + n)\ s\ t' \wedge bval\ b\ s \wedge (c,\ s) \Rightarrow t)$
assumes *last*: $\forall\ t\ t'.\ Q\ t\ t' \longrightarrow (\exists\ s\ s'.\ (\exists\ n.\ P\ (k + n)\ s\ s') \wedge (WHILE\ b\ DO\ c,\ s) \Rightarrow t \wedge (c',\ s') \Rightarrow t')$
assumes *post*: $Q\ t\ t'$
shows $\exists\ s\ s'.\ P\ k\ s\ s' \wedge (WHILE\ b\ DO\ c,\ s) \Rightarrow t \wedge (c',\ s') \Rightarrow t'$
⟨*proof*⟩

lemma *While-backwards-frontier-ir-valid*[*intro*]:
$(\bigwedge n.\ ir\text{-}valid\ (\lambda\ s\ s'.\ P\ n\ s\ s' \wedge bval\ b\ s)\ c\ SKIP\ (P\ (Suc\ n))) \Longrightarrow$

$ir\text{-}valid\ (\lambda s\ s'.\ \exists n.\ P\ n\ s\ s')\ (WHILE\ b\ DO\ c)\ c'\ Q \Longrightarrow$
$ir\text{-}valid\ (P\ 0)\ (WHILE\ b\ DO\ c)\ c'\ Q$
⟨*proof*⟩

lemma *conseq-ir-valid*:
$ir\text{-}valid\ P\ c\ c'\ Q \Longrightarrow (\bigwedge s\ s'.\ P\ s\ s' \Longrightarrow P'\ s\ s') \Longrightarrow (\bigwedge s\ s'.\ Q'\ s\ s' \Longrightarrow Q\ s\ s') \Longrightarrow$
$ir\text{-}valid\ P'\ c\ c'\ Q'$
⟨*proof*⟩

lemma *disj-ir-valid*[*intro*]:
$ir\text{-}valid\ P_1\ c\ c'\ Q_1 \Longrightarrow ir\text{-}valid\ P_2\ c\ c'\ Q_2 \Longrightarrow$
$ir\text{-}valid\ (\lambda s\ s'.\ P_1\ s\ s' \vee P_2\ s\ s')\ c\ c'\ (\lambda\ t\ t'.\ Q_1\ t\ t' \vee Q_2\ t\ t')$
⟨*proof*⟩

theorem *soundness*:
$ir\text{-}hoare\ P\ c\ c'\ Q \Longrightarrow ir\text{-}valid\ P\ c\ c'\ Q$
⟨*proof*⟩

5 Completeness

lemma *ir-Skip-Skip*[*intro*]:
$ir\text{-}hoare\ P\ SKIP\ SKIP\ P$
⟨*proof*⟩

lemma *ir-hoare-Skip-Skip*[*simp*]:
$ir\text{-}hoare\ P\ SKIP\ SKIP\ Q = (\forall s\ s'.\ Q\ s\ s' \longrightarrow P\ s\ s')$
⟨*proof*⟩

lemma *ir-valid-Seq1*:
$ir\text{-}valid\ P\ (c1;;\ c2)\ c'\ Q \Longrightarrow ir\text{-}valid\ P\ c1\ c'\ (\lambda t\ t'.\ \exists s\ s'.\ P\ s\ s' \wedge (c1,s) \Rightarrow t \wedge (c',s') \Rightarrow t' \wedge (\exists u.\ (c2,t) \Rightarrow u \wedge Q\ u\ t'))$
⟨*proof*⟩

lemma *ir-valid-Seq1'*:
$ir\text{-}valid\ P\ (c1;;\ c2)\ c'\ Q \Longrightarrow ir\text{-}valid\ (\lambda t\ t'.\ \exists s\ s'.\ P\ s\ s' \wedge (c1,s) \Rightarrow t \wedge (c',s') \Rightarrow t' \wedge (\exists u.\ (c2,t) \Rightarrow u \wedge Q\ u\ t'))\ c2\ SKIP\ Q$
⟨*proof*⟩

lemma *ir-valid-track-history*:
$ir\text{-}valid\ P\ c\ c'\ Q \Longrightarrow$
$ir\text{-}valid\ P\ c\ c'\ (\lambda t\ t'.\ Q\ s\ s' \wedge (\exists s\ s'.\ P\ s\ s' \wedge (c,s) \Rightarrow t \wedge (c',s') \Rightarrow t'))$
⟨*proof*⟩

lemma *ir-valid-If*:
$ir\text{-}valid\ (\lambda s\ s'.\ P\ s\ s')\ (IF\ b\ THEN\ c1\ ELSE\ c2)\ c'\ Q \Longrightarrow$
$ir\text{-}valid\ (\lambda s\ s'.\ P\ s\ s' \wedge bval\ b\ s)\ c1\ c'\ (\lambda t\ t'.\ Q\ t\ t' \wedge (\exists s\ s'.\ P\ s\ s' \wedge (c1,s) \Rightarrow t \wedge (c',s') \Rightarrow t' \wedge bval\ b\ s)) \wedge$

ir-valid $(\lambda s\ s'.\ P\ s\ s' \wedge \neg\ bval\ b\ s)\ c2\ c'\ (\lambda t\ t'.\ Q\ t\ t' \wedge (\exists s\ s'.\ P\ s\ s' \wedge (c2,s) \Rightarrow t \wedge (c',s') \Rightarrow t' \wedge \neg\ bval\ b\ s))$
⟨*proof*⟩

Inspired by the "$p(n) = \{\sigma \mid$ *you can get back from* σ *to some state in* p *by executing* C *backwards* n *times*$\}$" part of OHearn [4].

primrec *get-back* **where**
get-back $P\ b\ c\ 0 = (\lambda t\ t'.\ P\ t\ t')\ |$
get-back $P\ b\ c\ (Suc\ n) = (\lambda t\ t'.\ \exists s.\ (c,s) \Rightarrow t \wedge bval\ b\ s \wedge$ *get-back* $P\ b\ c\ n\ s\ t')$

lemma *ir-valid-get-back*:
ir-valid (*get-back* $P\ b\ c\ (Suc\ k)$) $(WHILE\ b\ DO\ c)\ c'\ Q \Longrightarrow$
ir-valid (*get-back* $P\ b\ c\ k$) $(WHILE\ b\ DO\ c)\ c'\ Q$
⟨*proof*⟩

lemma *ir-valid-While1*:
ir-valid (*get-back* $P\ b\ c\ k$) $(WHILE\ b\ DO\ c)\ c'\ Q \Longrightarrow$
(*ir-valid* $(\lambda s\ s'.$ *get-back* $P\ b\ c\ k\ s\ s' \wedge bval\ b\ s)\ c\ SKIP\ (\lambda t\ t'.$ *get-back* $P\ b\ c\ (Suc\ k)\ t\ t' \wedge (\exists u\ u'.\ (WHILE\ b\ DO\ c,t) \Rightarrow u \wedge (c',t') \Rightarrow u' \wedge Q\ u\ u')))$
⟨*proof*⟩

lemma *ir-valid-While3*:
ir-valid (*get-back* $P\ b\ c\ k$) $(WHILE\ b\ DO\ c)\ c'\ Q \Longrightarrow$
(*ir-valid* $(\lambda s\ s'.$ *get-back* $P\ b\ c\ k\ s\ s' \wedge bval\ b\ s)\ c\ c'\ (\lambda t\ t'.\ (\exists s'.\ (c',s') \Rightarrow t' \wedge$ *get-back* $P\ b\ c\ (Suc\ k)\ t\ s' \wedge (\exists u.\ (WHILE\ b\ DO\ c,t) \Rightarrow u \wedge\ Q\ u\ t'))))$
⟨*proof*⟩

lemma *ir-valid-While2*:
ir-valid $P\ (WHILE\ b\ DO\ c)\ c'\ Q \Longrightarrow$
ir-valid $(\lambda s\ s'.\ P\ s\ s' \wedge \neg\ bval\ b\ s)\ SKIP\ c'\ (\lambda t\ t'.\ Q\ t\ t' \wedge (\exists s'.\ (c',s') \Rightarrow t' \wedge P\ t\ s' \wedge \neg\ bval\ b\ t))$
⟨*proof*⟩

lemma *assign-upd-blah*:
$(\lambda a.\ if\ a = x1\ then\ s\ x1\ else\ (s(x1 := aval\ x2\ s))\ a) = s$
⟨*proof*⟩

lemma *Assign-complete*:
assumes v: *ir-valid* $P\ (x1 ::= x2)\ c'\ Q$
assumes q: $Q\ t\ t'$
shows $\exists s'.\ (\exists v.\ P\ (t(x1 := v))\ s' \wedge t\ x1 = aval\ x2\ (t(x1 := v))) \wedge (c', s') \Rightarrow t'$
⟨*proof*⟩

lemmas *ir-Skip-sym* = *ir-sym*[*OF ir-Skip*, *simplified flip-def*]

theorem *completeness*:
$\textit{ir-valid}\ P\ c\ c'\ Q \Longrightarrow \textit{ir-hoare}\ P\ c\ c'\ Q$
⟨*proof*⟩

6 A Decomposition Principle: Proofs via Under-Approximate Hoare Logic

We show the under-approximate analogue holds for Beringer's [1] decomposition principle for over-approximate relational logic.

definition
$\textit{decomp} :: \textit{rassn} \Rightarrow \textit{com} \Rightarrow \textit{com} \Rightarrow \textit{rassn} \Rightarrow \textit{rassn}$ **where**
$\textit{decomp}\ P\ c\ c'\ Q \equiv \lambda t\ s'.\ \exists s\ t'.\ P\ s\ s' \wedge (c,s) \Rightarrow t \wedge (c',s') \Rightarrow t' \wedge Q\ t\ t'$

lemma *ir-valid-decomp1*:
$\textit{ir-valid}\ P\ c\ c'\ Q \Longrightarrow \textit{ir-valid}\ P\ c\ \textit{SKIP}\ (\textit{decomp}\ P\ c\ c'\ Q) \wedge \textit{ir-valid}\ (\textit{decomp}\ P\ c\ c'\ Q)\ \textit{SKIP}\ c'\ Q$
⟨*proof*⟩

lemma *ir-valid-decomp2*:
$\textit{ir-valid}\ P\ c\ \textit{SKIP}\ R \wedge \textit{ir-valid}\ R\ \textit{SKIP}\ c'\ Q \Longrightarrow \textit{ir-valid}\ P\ c\ c'\ Q$
⟨*proof*⟩

lemma *ir-valid-decomp*:
$\textit{ir-valid}\ P\ c\ c'\ Q = (\textit{ir-valid}\ P\ c\ \textit{SKIP}\ (\textit{decomp}\ P\ c\ c'\ Q) \wedge \textit{ir-valid}\ (\textit{decomp}\ P\ c\ c'\ Q)\ \textit{SKIP}\ c'\ Q)$
⟨*proof*⟩

Completeness with soundness means we can prove proof rules about *ir-hoare* in terms of *ir-valid*.

lemma *ir-to-Skip*:
$\textit{ir-hoare}\ P\ c\ c'\ Q = (\textit{ir-hoare}\ P\ c\ \textit{SKIP}\ (\textit{decomp}\ P\ c\ c'\ Q) \wedge \textit{ir-hoare}\ (\textit{decomp}\ P\ c\ c'\ Q)\ \textit{SKIP}\ c'\ Q)$
⟨*proof*⟩

O'Hearn's under-approximate Hoare triple, for the "ok" case (since that is the only case we have) This is also likely the same as from the "Reverse Hoare Logic" paper (SEFM).

type-synonym $\textit{assn} = \textit{state} \Rightarrow \textit{bool}$
definition
$\textit{ohearn} :: \textit{assn} \Rightarrow \textit{com} \Rightarrow \textit{assn} \Rightarrow \textit{bool}$
where
$\textit{ohearn}\ P\ c\ Q \equiv (\forall t.\ Q\ t \longrightarrow (\exists s.\ P\ s \wedge (c,s) \Rightarrow t))$

lemma *fold-ohearn1*:
$\textit{ir-valid}\ P\ c\ \textit{SKIP}\ Q = (\forall t'.\ \textit{ohearn}\ (\lambda t.\ P\ t\ t')\ c\ (\lambda t.\ Q\ t\ t'))$
⟨*proof*⟩

lemma *fold-ohearn2*:
$ir\text{-}valid\ P\ SKIP\ c'\ Q = (\forall\, t.\ ohearn\ (P\ t)\ c'\ (Q\ t))$
⟨*proof*⟩

theorem *relational-via-hoare*:
$ir\text{-}hoare\ P\ c\ c'\ Q = ((\forall\, t'.\ ohearn\ (\lambda t.\ P\ t\ t')\ c\ (\lambda t.\ decomp\ P\ c\ c'\ Q\ t\ t')) \wedge (\forall\, t.\ ohearn\ (decomp\ P\ c\ c'\ Q\ t)\ c'\ (Q\ t)))$
⟨*proof*⟩

7 Deriving Proof Rules from Completeness

Note that we can more easily derive proof rules sometimes by appealing to the corresponding properties of *ir-valid* than from the proof rules directly.

We see more examples of this later on when we consider examples.

lemma *ir-Seq2*:
$ir\text{-}hoare\ P\ c\ SKIP\ Q \Longrightarrow ir\text{-}hoare\ Q\ d\ c'\ R \Longrightarrow ir\text{-}hoare\ P\ (Seq\ c\ d)\ c'\ R$
⟨*proof*⟩

lemma *ir-Seq*:
$ir\text{-}hoare\ P\ c\ c'\ Q \Longrightarrow ir\text{-}hoare\ Q\ d\ d'\ R \Longrightarrow ir\text{-}hoare\ P\ (Seq\ c\ d)\ (Seq\ c'\ d')\ R$
⟨*proof*⟩

8 Examples

8.1 Some Derived Proof Rules

First derive some proof rules – here not by appealing to completeness but just using the existing rules

lemma *ir-If-True-False*:
$ir\text{-}hoare\ (\lambda s\ s'.\ P\ s\ s' \wedge bval\ b\ s \wedge \neg\ bval\ b'\ s')\ c_1\ c_2{}'\ Q \Longrightarrow$
$ir\text{-}hoare\ P\ (IF\ b\ THEN\ c_1\ ELSE\ c_2)\ (IF\ b'\ THEN\ c_1{}'\ ELSE\ c_2{}')\ Q$
⟨*proof*⟩

lemma *ir-Assign-Assign*:
$ir\text{-}hoare\ P\ (x ::= e)\ (x' ::= e')\ (\lambda t\ t'.\ \exists\, v\ v'.\ P\ (t(x := v))\ (t'(x' := v')) \wedge t\ x = aval\ e\ (t(x := v)) \wedge t'\ x' = aval\ e'\ (t'(x' := v')))$
⟨*proof*⟩

8.2 prog1

A tiny insecure program. Note that we only need to reason on one path through this program to detect that it is insecure.

abbreviation *low-eq* :: *rassn* **where** $low\text{-}eq\ s\ s' \equiv s\ ''low'' = s'\ ''low''$
abbreviation *low-neq* :: *rassn* **where** $low\text{-}neq\ s\ s' \equiv \neg\ low\text{-}eq\ s\ s'$

definition *prog1* :: *com* **where**
prog1 ≡ (*IF* (*Less* (*N 0*) (*V ''x''*)) *THEN* (*Assign ''low''* (*N 1*)) *ELSE* (*Assign ''low''* (*N 0*)))

We prove that *prog1* is definitely insecure. To do that, we need to find some non-empty post-relation that implies insecurity. The following property encodes the idea that the post-relation is non-empty, i.e. represents a feasible pair of execution paths.

definition
nontrivial :: *rassn* ⇒ *bool*
where
nontrivial Q ≡ (∃ *t t'*. *Q t t'*)

Note the property we prove here explicitly encodes the fact that the postcondition can be anything that implies insecurity, i.e. implies $\lambda s\ s'.\ s\ ''low'' \neq s'\ ''low''$. In particular we should not necessarily expect it to cover the entirely of all states that satisfy $\lambda s\ s'.\ s\ ''low'' \neq s'\ ''low''$.

Also note that we also have to prove that the postcondition is non-trivial. This is necessary to make sure that the violation we find is not an infeasible path.

lemma *prog1*:
∃ *Q*. *ir-hoare low-eq prog1 prog1 Q* ∧ (∀ *s s'*. *Q s s'* ⟶ *low-neq s s'*) ∧ *nontrivial Q*
⟨*proof*⟩

8.3 More Derived Proof Rules for Examples

definition *BEq* :: *aexp* ⇒ *aexp* ⇒ *bexp* **where**
BEq a b ≡ *And* (*Less a* (*Plus b* (*N 1*))) (*Less b* (*Plus a* (*N 1*)))

lemma *BEq-aval*[*simp*]:
bval (*BEq a b*) *s* = ((*aval a s*) = (*aval b s*))
⟨*proof*⟩

lemma *ir-If-True-True*:
ir-hoare ($\lambda s\ s'$. *P s s'* ∧ *bval b s* ∧ *bval b' s'*) c_1 c_1' Q_1 ⟹
ir-hoare P (*IF b THEN* c_1 *ELSE* c_2) (*IF b' THEN* c_1' *ELSE* c_2') ($\lambda t\ t'$. Q_1 *t t'*)
⟨*proof*⟩

lemma *ir-If-False-False*:
ir-hoare ($\lambda s\ s'$. *P s s'* ∧ ¬ *bval b s* ∧ ¬ *bval b' s'*) c_2 c_2' Q_2 ⟹
ir-hoare P (*IF b THEN* c_1 *ELSE* c_2) (*IF b' THEN* c_1' *ELSE* c_2') ($\lambda t\ t'$. Q_2 *t t'*)
⟨*proof*⟩

lemma *ir-If'*:
ir-hoare ($\lambda s\ s'$. *P s s'* ∧ *bval b s* ∧ *bval b' s'*) c_1 c_1' Q_1 ⟹
ir-hoare ($\lambda s\ s'$. *P s s'* ∧ ¬ *bval b s* ∧ ¬ *bval b' s'*) c_2 c_2' Q_2 ⟹

ir-hoare P (*IF* b *THEN* c_1 *ELSE* c_2) (*IF* b' *THEN* $c_1{}'$ *ELSE* $c_2{}'$) ($\lambda t\ t'.\ Q_1\ t\ t' \vee Q_2\ t\ t'$)
⟨*proof*⟩

lemma *ir-While-triv*:
ir-hoare ($\lambda s\ s'.\ P\ s\ s' \wedge \neg$ *bval* $b\ s \wedge \neg$ *bval* $b'\ s'$) *SKIP SKIP* $Q_2 \Longrightarrow$
ir-hoare P (*WHILE* b *DO* c) (*WHILE* b' *DO* c') ($\lambda s\ s'.\ (Q_2\ s\ s')$)
⟨*proof*⟩

lemma *ir-While′*:
ir-hoare ($\lambda s\ s'.\ P\ s\ s' \wedge$ *bval* $b\ s \wedge$ *bval* $b'\ s'$) (c;;*WHILE* b *DO* c) (c';;*WHILE* b' *DO* c') $Q_1 \Longrightarrow$
ir-hoare ($\lambda s\ s'.\ P\ s\ s' \wedge \neg$ *bval* $b\ s \wedge \neg$ *bval* $b'\ s'$) *SKIP SKIP* $Q_2 \Longrightarrow$
ir-hoare P (*WHILE* b *DO* c) (*WHILE* b' *DO* c') ($\lambda s\ s'.\ (Q_1\ s\ s' \vee Q_2\ s\ s')$)
⟨*proof*⟩

8.4 client0

definition *low-eq-strong* **where**
low-eq-strong $s\ s' \equiv$ (s ''*high*'' $\neq s'$ ''*high*'') $\wedge$ *low-eq* $s\ s'$

lemma *low-eq-strong-upd*[*simp*]:
var $\neq$ ''*high*'' $\wedge$ *var* $\neq$ ''*low*'' $\Longrightarrow$ *low-eq-strong*(s(*var* := v)) (s'(*var* := v')) = *low-eq-strong* $s\ s'$
⟨*proof*⟩

A variation on client0 from O'Hearn [4]: how to reason about loops via a single unfolding

definition *client0* :: *com* **where**
client0 ≡ (*Assign* ''*x*'' (*N 0*);;
(*While* (*Less* (*N 0*) (*V* ''*n*''))
((*Assign* ''*x*'' (*Plus* (*V* ''*x*'') (*V* ''*n*'')));;
(*Assign* ''*n*'' (*V* ''*nondet*''))));;
(*If* (*BEq* (*V* ''*x*'') (*N 2000000*)) (*Assign* ''*low*'' (*V* ''*high*'')) *SKIP*))

lemma *client0*:
$\exists\ Q$. *ir-hoare low-eq client0 client0* Q $\wedge$ ($\forall\ s\ s'.\ Q\ s\ s' \longrightarrow$ *low-neq* $s\ s'$) $\wedge$ *nontrivial* Q
⟨*proof*⟩

lemma *ir-While-backwards*:
($\bigwedge n$. *ir-hoare* ($\lambda\ s\ s'.\ P\ n\ s\ s' \wedge$ *bval* $b\ s$) c *SKIP* (P (*Suc* n))) $\Longrightarrow$
ir-hoare ($\lambda s\ s'.\ \exists\ n.\ P\ n\ s\ s' \wedge \neg$ *bval* $b\ s$) *SKIP* c' $Q \Longrightarrow$
ir-hoare (P *0*) (*WHILE* b *DO* c) c' Q
⟨*proof*⟩

8.5 Derive a variant of the backwards variant rule

Here we appeal to completeness again to derive this rule from the corresponding property about *ir-valid*.

8.6 A variant of the frontier rule

Agin we derive this rule by appealing to completeness and the corresponding property of *ir-valid*

lemma *While-backwards-frontier-both-ir-valid'*:
assumes *asm*: $\bigwedge n.\ \forall t\ t'.\ P\ (k + Suc\ n)\ t\ t' \longrightarrow (\exists s\ s'.\ P\ (k + n)\ s\ s' \wedge bval\ b\ s \wedge bval\ b'\ s' \wedge (c, s) \Rightarrow t \wedge (c', s') \Rightarrow t')$
assumes *last*: $\forall t\ t'.\ Q\ t\ t' \longrightarrow (\exists s\ s'.\ (\exists n.\ P\ (k + n)\ s\ s') \wedge (WHILE\ b\ DO\ c, s) \Rightarrow t \wedge (WHILE\ b'\ DO\ c', s') \Rightarrow t')$
assumes *post*: $Q\ t\ t'$
shows $\exists s\ s'.\ P\ k\ s\ s' \wedge (WHILE\ b\ DO\ c, s) \Rightarrow t \wedge (WHILE\ b'\ DO\ c', s') \Rightarrow t'$
⟨*proof*⟩

lemma *While-backwards-frontier-both-ir-valid*[*intro*]:
$(\bigwedge n.\ ir\text{-}valid\ (\lambda\ s\ s'.\ P\ n\ s\ s' \wedge bval\ b\ s \wedge bval\ b'\ s')\ c\ c'\ (P\ (Suc\ n))) \Longrightarrow$
$ir\text{-}valid\ (\lambda s\ s'.\ \exists n.\ P\ n\ s\ s')\ (WHILE\ b\ DO\ c)\ (WHILE\ b'\ DO\ c')\ Q \Longrightarrow$
$ir\text{-}valid\ (P\ 0)\ (WHILE\ b\ DO\ c)\ (WHILE\ b'\ DO\ c')\ (\lambda s\ s'.\ Q\ s\ s')$
⟨*proof*⟩

lemma *ir-While-backwards-frontier-both*:
$[\![\bigwedge n.\ ir\text{-}hoare\ (\lambda s\ s'.\ P\ n\ s\ s' \wedge bval\ b\ s \wedge bval\ b'\ s')\ c\ c'\ (P\ (Suc\ n));$
$ir\text{-}hoare\ (\lambda s\ s'.\ \exists n.\ P\ n\ s\ s')\ (WHILE\ b\ DO\ c)\ (WHILE\ b'\ DO\ c')\ Q]\!]$
$\Longrightarrow ir\text{-}hoare\ (P\ 0)\ (WHILE\ b\ DO\ c)\ (WHILE\ b'\ DO\ c')\ (\lambda s\ s'.\ Q\ s\ s')$
⟨*proof*⟩

The following rule then follows easily as a special case

lemma *ir-While-backwards-both*:
$(\bigwedge n.\ ir\text{-}hoare\ (\lambda\ s\ s'.\ P\ n\ s\ s' \wedge bval\ b\ s \wedge bval\ b'\ s')\ c\ c'\ (P\ (Suc\ n))) \Longrightarrow ir\text{-}hoare\ (P\ 0)\ (WHILE\ b\ DO\ c)\ (WHILE\ b'\ DO\ c')\ (\lambda s\ s'.\ \exists n.\ P\ n\ s\ s' \wedge \neg\ bval\ b\ s \wedge \neg\ bval\ b'\ s')$
⟨*proof*⟩

8.7 client1

An example roughly equivalent to cient1 from O'Hearn [4]0

In particular we use the backwards variant rule to reason about the loop.

definition *client1* :: *com* **where**
client1 ≡ (*Assign* ''x'' (*N 0*);;
(*While* (*Less* (*V* ''x'') (*V* ''n''))
((*Assign* ''x'' (*Plus* (*V* ''x'') (*N 1*))));;
(*If* (*BEq* (*V* ''x'') (*N 2000000*)) (*Assign* ''low'' (*V* ''high'')) *SKIP*))

lemma *client1*:
$\exists Q.$ *ir-hoare low-eq client1 client1* $Q \wedge (\forall s\ s'.\ Q\ s\ s' \longrightarrow$ *low-neq* $s\ s') \wedge$ *nontrivial* Q
⟨*proof*⟩

8.8 client2

An example akin to client2 from O'Hearn [4].

Note that this example is carefully written to show use of the frontier rule first to reason up to the broken loop iteration, and then we unfold the loop at that point to reason about the broken iteration, and then use the plain backwards variant rule to reason over the remainder of the loop.

definition *client2* :: *com* **where**
client2 ≡ (*Assign* ''*x*'' (*N 0*);;
(*While* (*Less* (*V* ''*x*'') (*N 4000000*))
((*Assign* ''*x*'' (*Plus* (*V* ''*x*'') (*N 1*)));;
(*If* (*BEq* (*V* ''*x*'') (*N 2000000*)) (*Assign* ''*low*'' (*V* ''*high*'')) *SKIP*))
)
)

lemma *client2*:
$\exists Q.$ *ir-hoare low-eq client2 client2* $Q \wedge (\forall s\ s'.\ Q\ s\ s' \longrightarrow$ *low-neq* $s\ s') \wedge$ *nontrivial* Q
⟨*proof*⟩

end

References

[1] L. Beringer. Relational decomposition. In *International Conference on Interactive Theorem Proving (ITP)*, pages 39–54. Springer, 2011.

[2] E. De Vries and V. Koutavas. Reverse hoare logic. In *International Conference on Software Engineering and Formal Methods (SEFM)*, pages 155–171. Springer, 2011.

[3] T. Murray. An under-approximate relational logic: Heralding logics of insecurity, incorrect implementation & more. arXiv eprint arXiv:2003.04791 [cs.LO], 2020. https://arxiv.org/abs/2003.04791.

[4] P. W. O'Hearn. Incorrectness logic. *Proceedings of the ACM on Programming Languages*, 4(POPL):1–32, 2019.