Probabilistic Relational Hoare Logic

Main judgments

Hoare Logic $c : \Phi \implies \Psi$:

hoare [c : pre ==> post]

Probabilistic Hoare Logic $[c : \Phi \implies \Psi] = \delta$ (see Lecture 6):

bd_hoare [c : pre ==> post] = r

Probabilistic Relational Hoare Logic $c_1 \sim c_2 : \Phi \implies \Psi$ (pRHL):

equiv [c1 ~ c2 : pre ==> post]

Judgments consider statements; similar ones for functions

hoare [M.f : true ==> M.x = 2]

In this lecture, we will focus on pRHL

Some syntax

```
module P = {
  var r: int
  fun f(x:int, y:int) : int { return r + x + y }
}.
module M = {
  fun g(x:int, w:int) : int { return P.r + x + w }
}.
lemma L1 :
  equiv [ P.f ~ M.g :
      y{1} = w{2} /\ ={x, P.r} ==> ={res, P.r}].
```

- Tags apply to expressions
  (1 + P.r + x){1} is equivalent to 1 + P.r{1} + x{1}
- Equalities are restricted to variables
  ={x,P.r} stands for x{1} = x{2} ∧ P.r{1} = P.r{2}

Different kinds of rules

- For each instruction of the language there exists a corresponding logical rule
- Most of the rules are a composition of the sequence rule and the corresponding basic rule
- Also high level rules based on program transformation
- Some automation, composition of basic rules (in progress)

Basic rules: rule of consequence

$$\frac{}{c_1 \sim c_2 : \mathit{false} \Longrightarrow Q}$$

Syntax: exfalso

$$\frac{c_1 \sim c_2 : P' \Longrightarrow Q' \qquad P \Rightarrow P' \qquad Q' \Rightarrow Q}{c_1 \sim c_2 : P \Longrightarrow Q}$$

Syntax:

- conseq L
- conseq (_ : P’ ==> Q’)

Basic proof rules: case

$$\frac{c \sim c' : P \land A \Longrightarrow Q \quad c \sim c' : P \land \neg A \Longrightarrow Q}{c \sim c' : P \Longrightarrow Q}$$

Syntax: case A

Basic proof rules: skip and sequence

$$\frac{P \Rightarrow Q}{\text{skip} \sim \text{skip} : P \Longrightarrow Q}$$

Syntax: skip

$$\frac{c_1 \sim c_1' : P \Longrightarrow R \qquad c_2 \sim c_2' : R \Longrightarrow Q}{c_1; c_2 \sim c_1'; c_2' : P \Longrightarrow Q}$$

Syntax: seq i j : R

- i is the length of c_1
- j is the length of c_1'

Basic proof rules: assignment

$$\frac{}{x = e \sim \mathsf{skip} : Q\,\{x\langle 1\rangle := e\langle 1\rangle\} \Longrightarrow Q}$$

$$\frac{}{\mathsf{skip} \sim x = e : Q\,\{x\langle 2\rangle := e\langle 2\rangle\} \Longrightarrow Q}$$

Syntax: wp
Applies the assignment rule as much as possible.

Example

```
pre = true

b =${0,1}     (1) z = 3
x = 1         (2)
y = 2         (3)

post = x{1} + y{1} = z{2}
```

wp.

```
pre = true

b =${0,1}     (1)

post = 1 + 2 = 3
```

Basic proof rules: random assignment

One side rule

$$\frac{P = \textit{lossless } d \wedge \forall v \in \textit{supp } d, Q\{x\langle 1\rangle := v\}}{x = \$d \sim \text{skip} : P \Longrightarrow Q}$$

Syntax: rnd{1}

Remark: This is not the rule used in practice (relational).

Basic proof rules: random assignment

Two-sided rule

$$\frac{Q' = \forall v \in \mathit{supp}\ d, Q\{x\langle 1\rangle, x'\langle 2\rangle := v, f\ v\}}{x = \$d \sim x' = \$d' : Q' \Longrightarrow Q}$$

where

- f is 1-1 from *supp* d to *supp* d'
- for all $x \in \mathit{supp}\ d$, $d\ x = d'\ (f\ x)$

Syntax:

- rnd f finv
- rnd f
- rnd

Example

---

pre = true

x =$[0..10]   (1) x =$[2..12]

post = x{1} + 2 = x{2}

---

rnd (*lambda* x, x + 2) (*lambda* x, x − 2).   beta.

---

pre = true

post =
  forall (xL xR : int), in_supp xL [0..10] => in_supp xR [2..12] =>
    mu_x [0..10] xL = mu_x [2..12] (xL + 2) /\
    in_supp (xR − 2) [0..10] /\
     xL + 2 − 2 = xL /\ xR − 2 + 2 = xR /\
     xL + 2 = xL + 2

Explanation

post = x{1} + 2 = x{2}
rnd (*lambda* x, x + 2) (*lambda* x, x − 2).

The function f is λ x, x + 2 and its inverse f^{-1} is λ x, x − 2

For all xL xR in the support of [0..10] and [2..12]

- f preserves the probability of each element
  mu_x [0..10] xL = mu_x [2..12] (xL + 2)
- f^{-1} maps an element of [2..12] to an element of [0..10]
  in_supp (xR − 2) [0..10]
- f is a bijection $f\ (f^{-1}\ xL) = xL$ and $f^{-1}(f\ xR) = xR$
  xL + 2 − 2 = xL / xR − 2 + 2 = xR
- the original post-condition is valid for all xL and $(f\ xL)$
  xL + 2 = xL + 2

To finish the proof: skip;smt

Basic proof rules: conditional

One sided version

$$\frac{c_t \sim c : P \wedge e\langle 1\rangle \Longrightarrow Q \qquad c_f \sim c : P \wedge \neg e\langle 1\rangle \Longrightarrow Q}{\text{if } e \text{ then } c_t \text{ else } c_f \sim c : P \Longrightarrow Q}$$

Syntax: **if**{1}, **if**{2}

Two sided version

$$\frac{\begin{array}{c} P \Rightarrow e\langle 1\rangle \Leftrightarrow e'\langle 2\rangle \\ c_t \sim c'_t : P \wedge e\langle 1\rangle \Longrightarrow Q \quad c_f \sim c'_f : P \wedge \neg e\langle 1\rangle \Longrightarrow Q \end{array}}{\text{if } e \text{ then } c_t \text{ else } c_f \sim \text{if } e' \text{ then } c'_t \text{ else } c'_f : P \Longrightarrow Q}$$

Syntax: **if**

Remark : works only when the *if* is the first instruction

Basic proof rules: while

Two sided version (simplified):

$$\frac{\begin{array}{l} I' = e\langle 1\rangle \Leftrightarrow e'\langle 2\rangle \wedge I \\ c \sim c' : e\langle 1\rangle \wedge e'\langle 2\rangle \wedge I \Longrightarrow I' \end{array}}{\text{while } e \text{ do } c \sim \text{while } e' \text{ do } c' : I' \Longrightarrow \neg e\langle 1\rangle \wedge \neg e'\langle 2\rangle \wedge I}$$

Syntax: **while** I

A one sided version exists

Basic proof rules: call

simplified version:

$$\frac{\begin{array}{l} f \sim f' : P_f \Longrightarrow Q_f \\ P \Rightarrow P_f \{x\langle 1\rangle, x'\langle 2\rangle := e\langle 1\rangle, e'\langle 2\rangle\} \\ \forall\ r\ r', Q_f \{res\langle 1\rangle, res\langle 2\rangle := r, r'\} \Rightarrow Q \{y\langle 1\rangle, y'\langle 2\rangle := r, r'\} \end{array}}{y = f(e) \sim y' = f'(e') : P \Longrightarrow Q}$$

where x (resp. x') is the parameter of f (resp. f').

A one-sided version also exists (based on probabilistic hoare logic)

Rules based on program transformations

The generic form is:

$$\frac{c_2 \sim c' : P \Longrightarrow Q}{c_1 \sim c' : P \Longrightarrow Q}$$

Where c_1 and c_2 are semantically equivalent.

c_2 is automatically generated by the rule.

Program transformations: swap

$$\frac{c_1\,;\,c_3\,;\,c_2\,;\,c_4 \sim c' : P \Longrightarrow Q}{c_1\,;\,c_2\,;\,c_3\,;\,c_4 \sim c' : P \Longrightarrow Q}$$

Side condition: c_2 and c_3 are *independent*
Sufficient conditions

- c_2 does not write variables read by c_3
- c_3 does not write variables read by c_2
- they do not write a common variable

They are automatically checked by the tool

Syntax:

- swap{1} i k
- swap{1} [i .. j] k

Example

```
pre = true

b  = ${0,1}        (1) b' = ${0,1}
b' = ${0,1}        (2) b  = ${0,1}

post = ={b, b'}
```

```
swap{2} 1 1
```

```
pre = true

b  = ${0,1}        (1) b  = ${0,1}
b' = ${0,1}         (2) b' = ${0,1}


post = ={b, b'}
```

To finish: do !rnd => //.

Other tactics based on program transformation

- inline, rcondt, rcondf
- unroll, splitwhile, (loop)fusion, (loop)fission
- kill
- eqobs_in

From functions to statements

$$\frac{c_f \sim c_g : P \Longrightarrow Q \left\{\text{res}\langle 1\rangle, \text{res}\langle 2\rangle := r_f\langle 1\rangle, r_g\langle 2\rangle\right\}}{f \sim g : P \Longrightarrow Q}\ [\text{Fun}]$$

- The rule allows proving a specification on functions by proving it on their bodies
- c_f and c_g correspond to the statement bodies of the functions
- the special variables res{1},res{2} are replaced by the return expression of the functions

Syntax: fun

Remark: this rule only works for concrete functions (see tomorrow)

From pRHL to probabilities

$$\frac{f \sim g : P \Longrightarrow Q \qquad P\ m_1\ m_2 \qquad \forall m_1\ m_2, Q\ m_1\ m_2 \Rightarrow A\ m_1 \Leftrightarrow B\ m_2}{\Pr[f, m_1 : A] = \Pr[g, m_2 : B]}$$

$$\frac{f \sim g : P \Longrightarrow Q \qquad P\ m_1\ m_2 \qquad \forall m_1\ m_2, Q\ m_1\ m_2 \Rightarrow A\ m_1 \Rightarrow B\ m_2}{\Pr[f, m_1 : A] \leq \Pr[g, m_2 : B]}$$

In EasyCrypt

```
lemma E : equiv [M.f ~ N.g : P ==> Q].

lemma L : Pr[M.f() @ &m1 : A] = Pr[N.g() @ &m2 : B].
proof.
  equiv_deno E.
```

Variant: `equiv_deno (_ : P ==> Q).`

Try by yourself !