Probabilistic Relational Hoare Logic
Main judgments

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

$$\text{hoare} [ c : \text{pre} \Rightarrow \text{post}]$$

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

$$\text{bd}_\text{hoare} [c : \text{pre} \Rightarrow \text{post}] = r$$

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

$$\text{equiv} [c_1 \sim c_2 : \text{pre} \Rightarrow \text{post}]$$

Judgments consider statements; similar ones for functions

$$\text{hoare} [M.f : \text{true} \Rightarrow M.x = 2]$$

In this lecture, we will focus on pRHL
Some syntax

```plaintext
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 $x{1} = x{2} \land P.r{1} = P.r{2}$
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

\[ c_1 \sim c_2 : \text{false} \implies Q \]

Syntax: exfalso

\[ c_1 \sim c_2 : P' \implies Q' \quad P \implies P' \quad Q' \implies Q \]

Syntax:
- \text{conseq L}
- \text{conseq (\_ : P' ==> Q')}
Basic proof rules: case

\[
\begin{align*}
    c \sim c' : P \land A &\implies Q \\
    c \sim c' : P \land \neg A &\implies Q \\
    c \sim c' : P &\implies Q
\end{align*}
\]

Syntax: case A
Basic proof rules: skip and sequence

\[
P \Rightarrow Q
\]

\[
\text{skip} \sim \text{skip}: P \implies Q
\]

Syntax: skip

\[
c_1 \sim c'_1 : P \implies R \quad c_2 \sim c'_2 : R \implies Q
\]

\[
c_1; c_2 \sim c'_1; c'_2 : P \implies Q
\]

Syntax: seq \( i \) \( j : R \)

- \( i \) is the length of \( c_1 \)
- \( j \) is the length of \( c'_1 \)
Basic proof rules: assignment

\[ x = e \sim \text{skip} : Q \{x\langle1\rangle := e\langle1\rangle\} \implies Q \]

\[ \text{skip} \sim x = e : Q \{x\langle2\rangle := e\langle2\rangle\} \implies Q \]

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

\[ \text{pre} = \text{true} \]

\[ b = \{0, 1\} \quad (1) \quad z = 3 \]
\[ x = 1 \quad (2) \]
\[ y = 2 \quad (3) \]

\[ \text{post} = x^{1} + y^{1} = z^{2} \]

wp.

\[ \text{pre} = \text{true} \]

\[ b = \{0, 1\} \quad (1) \]

\[ \text{post} = 1 + 2 = 3 \]
Basic proof rules: random assignment

One side rule

\[
P = \text{lossless } d \land \forall v \in \text{supp } d, \ Q \{x\langle 1 \rangle := v\}
\]

\[
x = d \sim \text{skip} : P \implies Q
\]

Syntax: \text{rnd}\{1\}

Remark: This is not the rule used in practice (relational).
Basic proof rules: random assignment

Two-sided rule

\[ Q' = \forall v \in supp \, d, \ Q \{x\langle 1\rangle, \ x'\langle 2\rangle := v, \ f \ v\} \]
\[ x = \$d \sim x' = \$d' : \ Q' \implies Q \]

where

- \( f \) is 1-1 from \( supp \, d \) to \( supp \, d' \)
- for all \( x \in supp \, d \), \( d \ x = d' \ (f \ x) \)

Syntax:

- \( \text{rnd f finv} \)
- \( \text{rnd f} \)
- \( \text{rnd} \)
Example

\[\text{pre} = \text{true}\]

\[x \in [0..10] \quad (1) \quad x \in [2..12]\]

\[\text{post} = x[1] + 2 = x[2]\]

\[\text{rnd} (\lambda x, x + 2) (\lambda x, x - 2). \quad \text{beta.}\]

\[\text{pre} = \text{true}\]

\[\text{post} = \forall (x_L, x_R : \text{int}), \text{in_supp} x_L [0..10] \implies \text{in_supp} x_R [2..12] \implies \mu_x [0..10] x_L = \mu_x [2..12] (x_L + 2) \land \text{in_supp} (x_R - 2) [0..10] \land x_L + 2 - 2 = x_L \land x_R - 2 + 2 = x_R \land x_L + 2 = x_L + 2\]
Explanation

\[
\text{post} = x\{1\} + 2 = x\{2\}
\]
\[
\text{rnd} (\text{lambda } x, x + 2) (\text{lambda } x, x - 2).
\]

The function \( f \) is \( \lambda x, x + 2 \) and its inverse \( f^{-1} \) is \( \lambda x, x - 2 \)

For all \( x_L \) \( x_R \) in the support of [0..10] and [2..12]

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

To finish the proof: skip;smt
Basic proof rules: conditional

One sided version

\[
\begin{align*}
  & c_t \sim c : P \land e(1) \Rightarrow Q \\
  & c_f \sim c : P \land \neg e(1) \Rightarrow Q \\
  \text{if } e \text{ then } c_t \text{ else } c_f \sim c : P \Rightarrow Q
\end{align*}
\]

Syntax: \texttt{if\{1\}, if\{2\}}

Two sided version

\[
\begin{align*}
  & P \Rightarrow e(1) \Leftrightarrow e'(2) \\
  & c_t \sim c'_t : P \land e(1) \Rightarrow Q \\
  & c_f \sim c'_f : P \land \neg e(1) \Rightarrow Q \\
  \text{if } e \text{ then } c_t \text{ else } c_f \sim \text{if } e' \text{ then } c'_t \text{ else } c'_f : P \Rightarrow Q
\end{align*}
\]

Syntax: \texttt{if}

Remark: works only when the \texttt{if} is the first instruction
Basic proof rules: while

Two sided version (simplified):

\[
I' = e\langle 1 \rangle \iff e'\langle 2 \rangle \land I
\]

\[
c \sim c' : e\langle 1 \rangle \land e'\langle 2 \rangle \land I \implies I'
\]

while e do c \sim while e' do c' : I' \implies \neg e\langle 1 \rangle \land \neg e'\langle 2 \rangle \land I

Syntax: while I

A one sided version exists
Basic proof rules: call

simplified version:

\[ f \sim f' : P_f \implies Q_f \]
\[ P \Rightarrow P_f \{ x\langle 1 \rangle , x'\langle 2 \rangle := e\langle 1 \rangle , e'\langle 2 \rangle \} \]
\[ \forall r \, r', Q_f \{ res\langle 1 \rangle , res\langle 2 \rangle := r , r' \} \Rightarrow Q \{ y\langle 1 \rangle , y'\langle 2 \rangle := r , r' \} \]
\[ y = f(e) \sim y' = f'(e') : P \implies Q \]

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:

\[
\frac{c_2 \sim c' : P \implies Q}{c_1 \sim c' : P \implies Q}
\]

Where \(c_1\) and \(c_2\) are semantically equivalent.

\(c_2\) is automatically generated by the rule.
Program transformations: swap

\[
\begin{align*}
&c_1; c_3; c_2; c_4 \sim c' : P \implies Q \\
&c_1; c_2; c_3; c_4 \sim c' : P \implies Q
\end{align*}
\]

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} \hspace{1cm} (1) b' = \${0,1}

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

post = ={b, b'}

swap{2} 1 1

pre = true

b = \${0,1} \hspace{1cm} (1) b = \${0,1}

b' = \${0,1} \hspace{1cm} (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

\[
c_f \sim c_g : P \implies Q \{ \text{res}\langle 1 \rangle, \text{res}\langle 2 \rangle := r_f\langle 1 \rangle, r_g\langle 2 \rangle \}
\]

\[
f \sim g : P \implies Q
\]

[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 \( \text{res}\langle 1 \rangle, \text{res}\langle 2 \rangle \) 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

\[ f \sim g : P \implies Q \quad P \quad m_1 \quad m_2 \quad \forall m_1 \quad m_2, Q \quad m_1 \quad m_2 \implies A \quad m_1 \iff B \quad m_2 \]

\[ \Pr[f, m_1 : A] = \Pr[g, m_2 : B] \]

\[ f \sim g : P \implies Q \quad P \quad m_1 \quad m_2 \quad \forall m_1 \quad m_2, Q \quad m_1 \quad m_2 \implies A \quad m_1 \implies B \quad m_2 \]

\[ \Pr[f, m_1 : A] \leq \Pr[g, m_2 : B] \]

In EasyCrypt

\[ \text{lemma} \ E : \text{equiv} \ [M.f \sim N.g : P \implies Q]. \]

\[ \text{lemma} \ L : \Pr[M.f() @ \&m1 : A] = \Pr[N.g() @ \&m2 : B]. \]

\[ \text{proof.} \]
\[ \quad \text{equiv_deno} \ E. \]

Variant: \[ \text{equiv_deno} \ (\_ : P \implies Q). \]
Try by yourself !