EasyCrypt - Lecture 3
Proving in EasyCrypt and pRHL in practice

Pierre-Yves Strub

Monday November 24th

Proving in EasyCrypt

- EasyCrypt logic is a general higher-order logic
- It comes with a proof engine that allows to prove facts

---

lemma mylemma b1 b2 b3 :
  (b1 ⇒ b2) ⇒ (b2 ⇒ b3) ⇒ b1 ⇒ b3.
proof. *(∗ proof starts here ∗)*

Proving in EasyCrypt

- EasyCrypt logic is a general higher-order logic
- It comes with a proof engine that allows to prove facts

---

lemma mylemma b1 b2 b3 :
  (b1 ⇒ b2) ⇒ (b2 ⇒ b3) ⇒ b1 ⇒ b3.
proof. *(∗ proof starts here ∗)*

$b1 : bool$
$b2 : bool$
$b3 : bool$
} local hypotheses (context)

$\underbrace{(b1 \Rightarrow b2) \Rightarrow (b2 \Rightarrow b3) \Rightarrow b1}_{\text{assumptions}} \Rightarrow \underbrace{b3}_{\text{conclusion}}$ } goal

Proving in EasyCrypt

- Progress is done via tactics that allows the *simplification*, *decomposition* into *subgoals*, or the *resolution* of the goal.

Continuing the proof

---

lemma mylemma b1 b2 b3 : ...
proof.
  move⇒ hb12.

---

b1 : bool
b2 : bool
b3 : bool
hb12 : b1 ⇒ *b2*

(b2 ⇒ b3) ⇒ b1 ⇒ b3

Continuing the proof

---

lemma mylemma b1 b2 b3 : ...
proof.
  move⇒ hb12 bh23 hb1.

---

b1 : bool
b2 : bool
b3 : bool
hb12 : b1 ⇒ b2
hb23 : b2 ⇒ b3
hb1 : b1

═══

b3

Continuing the proof

---

lemma mylemma b1 b2 b3 : ...
proof.
  move⇒ hb12 bh23 hb1.
  apply hb23.

---

b1 : bool
b2 : bool
b3 : bool
hb12 : b1 ⇒ b2
hb23 : b2 ⇒ b3
hb1 : b1

═══

b2

Continuing the proof

```
lemma mylemma b1 b2 b3 : ...
proof.
  move⇒ hb12 bh23 hb1.
  apply hb23.
  apply hb12.
```

```
b1 : bool
b2 : bool
b3 : bool
hb12 : b1 ⇒ b2
hb23 : b2 ⇒ b3
hb1 : b1
```

```
b1
```

Continuing the proof

```
lemma mylemma b1 b2 b3 : ...
proof.
  move⇒ hb12 bh23 hb1.
  apply hb23.
  apply hb12.
  assumption.
```

Proof completed

Continuing the proof

```
lemma mylemma b1 b2 b3 : ...
proof.
  move⇒ hb12 bh23 hb1.
  apply hb23.
  apply hb12.
  assumption.
qed.
```

Main judgments

- Ambient logic
- Hoare Logic $\{\Phi\}\ c\ \{\Psi\}$:

  hoare [c : pre $\Longrightarrow$ post]

- Probabilistic Hoare Logic $\{\Phi\}\ c\ \{\Psi\}\ \diamond\,\delta$:

  phoare [c : pre $\Longrightarrow$ post] $=$ r

- Prob. Relational Hoare Logic $\{\Phi\}\ c_1 \sim c_2\ \{\Psi\}$ (pRHL)

  equiv [c1 $\sim$ c2 : pre $\Longrightarrow$ post]

- Similar ones for functions:

  hoare [M.f : true $\Longrightarrow$ M.x $= 2$]

In this lecture, we focus on

- The ambient logic
- The Prob. Relational Hoare Logic (pRHL)

Plan

Proving in EasyCrypt

Proving in the Ambient Logic

Proving in pRHL

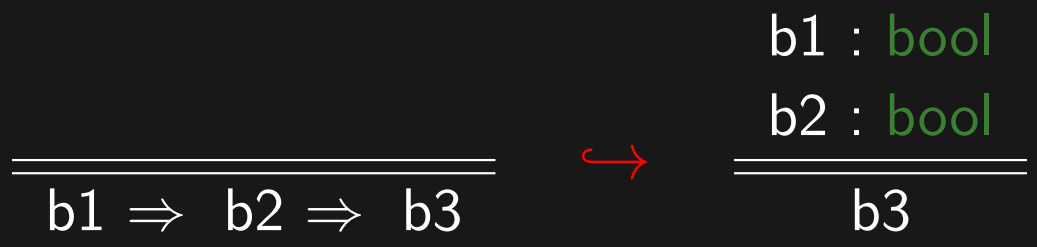
Propositional logic

- $\mathsf{b1} \Rightarrow \mathsf{b2} \Rightarrow \mathsf{b3}$

Propositional logic

- b1 $\Rightarrow$ b2 $\Rightarrow$ b3

As a goal [move$\Rightarrow$ b1 b2]



Propositional logic

- $b1 \Rightarrow b2 \Rightarrow b3$

  As a goal [move⇒ b1 b2]

  As an hypothesis [apply]

$$\frac{h : b1 \Rightarrow b2 \Rightarrow b3}{b3} \hookrightarrow$$

1. $\dfrac{h : b1 \Rightarrow b2 \Rightarrow b3}{b1}$ 2. $\dfrac{h : b1 \Rightarrow b2 \Rightarrow b3}{b2}$

Propositional logic - connectors

- Conjunction: $a \wedge b$

Propositional logic - connectors

- Conjunction: $a \wedge b$

As a goal [split] (prove $a \wedge b$)

$$\frac{}{a \wedge b} \quad \hookrightarrow \quad 1.\ \frac{}{a} \qquad 2.\ \frac{}{b}$$

Propositional logic - connectors

- Conjunction: $a \wedge b$

As a goal [split] (prove $a \wedge b$)

$$\frac{}{a \wedge b} \quad \hookrightarrow \quad 1.\ \frac{}{a} \qquad 2.\ \frac{}{b}$$

As an hypothesis [elim ab] (split $a \wedge b$ in a *and* b)

$$\frac{ab : a \wedge b}{\phi} \quad \hookrightarrow \quad \frac{}{a \Rightarrow b \Rightarrow \phi}$$

Propositional logic - connectors

- Disjunction: $a \vee b$

Propositional logic - connectors

- Disjunction: $a \vee b$

  As a goal

  – [left] (prove $a \vee b$ by proving a)

  $$\frac{}{a \vee b} \hookrightarrow \frac{}{a}$$

  – [right] (prove $a \vee b$ by proving b)

  $$\frac{}{a \vee b} \hookrightarrow \frac{}{b}$$

Propositional logic - connectors

- Disjunction: $a \vee b$

  As a goal

  – [left] (prove $a \vee b$ by proving a)

  $$\frac{}{a \vee b} \hookrightarrow \frac{}{a}$$

  – [right] (prove $a \vee b$ by proving b)

  $$\frac{}{a \vee b} \hookrightarrow \frac{}{b}$$

  As an hypothesis [elim ab] (case analysis on $a \vee b$)

$$\frac{ab : a \vee b}{\phi} \hookrightarrow 1. \ \frac{}{a \Rightarrow \phi} \quad 2. \ \frac{}{b \Rightarrow \phi}$$

Propositional logic - existential

- Existential: *exists* x : t, ϕ (x)

Propositional logic - existential

- Existential: *exists* x : t, ϕ (x)

  As a goal [*exists* v] (prove goal by giving a witness)

$$\frac{}{\textit{exists}\ \mathrm{x} : \mathrm{t}, \phi\,(\mathrm{x})} \quad \hookrightarrow \quad \frac{}{\phi\,(\mathrm{v})}$$

Propositional logic - existential

- Existential: *exists* x : t, ϕ (x)

  As a goal [*exists* v] (prove goal by giving a witness)

$$\frac{}{\textit{exists}\ \mathsf{x} : \mathsf{t}, \phi\,(\mathsf{x})} \quad \hookrightarrow \quad \frac{}{\phi\,(\mathsf{v})}$$

As an hypothesis [elim h] (extract a witness)

$$\frac{\mathsf{h} : \textit{exists}\ \mathsf{x} : \mathsf{t}, \phi\,(\mathsf{x})}{\phi\,'} \quad \hookrightarrow \quad \frac{}{\textit{forall}\ (\mathsf{v} : \mathsf{t}), \phi\,(\mathsf{v}) \Rightarrow \phi\,'}$$

Boolean case analysis

The tactic case allows to do a case analysis on any formula.

Boolean case analysis

The tactic case allows to do a case analysis on any formula.

$$\frac{\begin{array}{l}\mathsf{a} : \mathsf{bool}\\ \mathsf{b} : \mathsf{bool}\end{array}}{\mathsf{a}\ `\mathsf{b} = (\mathsf{a} \wedge\ \ !\mathsf{b}) \,||\, (!\mathsf{a} \wedge\ \ \mathsf{b})}$$

Boolean case analysis

The tactic case allows to do a case analysis on any formula.

$$\frac{\begin{array}{l}a : bool\\ b : bool\end{array}}{a\ `b = (a \wedge\ !b) \,||\, (!a \wedge\ b)}$$

(case a) leads to

1. $$\frac{\begin{array}{l}a : bool\\ b : bool\end{array}}{a \Rightarrow\ true\ `b = (true \wedge\ !b) \vee\ (!true \wedge\ b)}$$

2. $$\frac{\begin{array}{l}a : bool\\ b : bool\end{array}}{!a \Rightarrow\ false\ `b = (false \wedge\ !b) \vee\ (!false \wedge\ b)}$$

Identification up to computations

EasyCrypt comes with a set of simplification rules.

$$\frac{\begin{array}{l}\mathsf{a} : \mathsf{bool} \\ \mathsf{b} : \mathsf{bool}\end{array}}{\mathsf{false}\ \text{`}\mathsf{b} = (\mathsf{false} \wedge\ \ !\mathsf{b}) \vee\ \ (!\mathsf{false} \wedge\ \ \mathsf{b})}$$

Identification up to computations

EasyCrypt comes with a set of simplification rules.

$$\frac{\begin{array}{l}\mathsf{a} : \mathsf{bool}\\ \mathsf{b} : \mathsf{bool}\end{array}}{\mathsf{false}\ \text{`}\mathsf{b} = (\mathsf{false} \wedge\ \ !\mathsf{b}) \vee\ \ (!\mathsf{false} \wedge\ \ \mathsf{b})}$$

simplify leads to

$$\frac{\begin{array}{l}\mathsf{a} : \mathsf{bool}\\ \mathsf{b} : \mathsf{bool}\end{array}}{\mathsf{b} = \mathsf{b}}$$

that can be easily solved by reflexivity.

Identification up to computations

Computations include

- functions applications reduction
- operators body inlining
- logical operators tautology ($a \wedge \ \text{false} \rightarrow \ \text{false}$)

Terms that are equal up to computations are considered as identical

$$\frac{a : \text{bool} \quad b : \text{bool}}{!a \Rightarrow \ \text{false}\ `b = (\text{false} \wedge \ !b) \vee \ (!\text{false} \wedge \ b)}$$

can be directly solved by reflexivity.

Rewrite - replace equals by equals

The tactic rewrite replaces a subterm a of the goal by an equal one b. It takes a proof of $a = b$ or $a \Leftrightarrow b$.

Rewrite - replace equals by equals

The tactic rewrite replaces a subterm a of the goal by an equal one b. It takes a proof of $a = b$ or $a \Leftrightarrow b$.

rewrite h

$$\frac{h : a = b}{\vdash P\ a} \quad \hookrightarrow \quad \frac{h : a = b}{\vdash P\ b}$$

Rewrite - replace equals by equals

$$2 * (a + b) = (b + a) + (a + b)$$

Rewrite - replace equals by equals

$$2 * (a + b) = (b + a) + (a + b)$$

- rewrite {2}addnC

$$2 * (a + b) = (b + a) + (b + a)$$

Rewrite - replace equals by equals

$$2 * (a + b) = (b + a) + (a + b)$$

- rewrite {2}addnC

$$2 * (a + b) = (b + a) + (b + a)$$

- rewrite (addnC b a)

$$2 * (a + b) = (a + b) + (a + b)$$

Rewrite - replace equals by equals

$$2 * (a + b) = (b + a) + (a + b)$$

- rewrite {2}addnC

$$2 * (a + b) = (b + a) + (b + a)$$

- rewrite (addnC b a)

$$2 * (a + b) = (a + b) + (a + b)$$

- rewrite -!addnA

$$2 * (a + b) = b + (a + (a + b))$$

Logical cut

The tactic cut: ϕ allows to do a forward chaining

$$\frac{\text{h: } \ldots}{\phi\,'} \quad \hookrightarrow \quad 1.\ \frac{\text{h: } \ldots}{\phi} \quad 2.\ \frac{\text{h: } \ldots}{\phi \Rightarrow \phi\,'}$$

Logical cut

The tactic cut: ϕ allows to do a forward chaining

$$\frac{\text{h: } \ldots}{\phi\,'} \quad \hookrightarrow \quad 1.\ \frac{\text{h: } \ldots}{\phi} \quad 2.\ \frac{\text{h: } \ldots}{\phi \Rightarrow \phi\,'}$$

It is possible to give a name to the new goal (cut my: ϕ)

$$\frac{\text{h: } \ldots}{\phi\,'} \quad \hookrightarrow \quad 1.\ \frac{\text{h: } \ldots}{\phi} \quad 2.\ \frac{\begin{array}{c}\text{h: } \ldots \\ \text{my: } \phi\end{array}}{\phi\,'}$$

Induction

An induction principle for a type t is any formula of the form:

$$\textit{forall}\ (\mathsf{p} : \mathsf{t} \to\ \mathsf{bool}), \phi_1 \to\ \ldots \to\ \phi_n,$$
$$\textit{forall}\ (\mathsf{x} : \mathsf{t}), \psi_1(\mathsf{x}) \to\ \ldots \to\ \psi_n(\mathsf{x}) \to\ \mathsf{p}\ \mathsf{x}$$

Induction

An induction principle for a type t is any formula of the form:

$$\begin{array}{c} \mathit{forall}\ (\mathsf{p} : \mathsf{t} \to\ \mathsf{bool}), \phi_1 \to\ \ldots \to\ \phi_n, \\ \mathit{forall}\ (\mathsf{x} : \mathsf{t}), \psi_1(\mathsf{x}) \to\ \ldots \to\ \psi_n(\mathsf{x}) \to\ \mathsf{p}\ \mathsf{x} \end{array}$$

For example, for natural numbers:

$$\begin{array}{l} \mathit{forall}\ (\mathsf{p} : \mathsf{int} \to \mathsf{t}),\ \mathsf{p}\ 0 \Rightarrow \\ \quad (\mathit{forall}\ (\mathsf{x} : \mathsf{int}),\ 0 \leq \mathsf{x} \Rightarrow \mathsf{p}\ \mathsf{x} \Rightarrow \mathsf{p}\ (\mathsf{x} + 1)) \Rightarrow \\ \quad\quad \mathit{forall}\ (\mathsf{x} : \mathsf{int}),\ 0 \leq \mathsf{x} \Rightarrow \mathsf{p}\ \mathsf{x} \end{array}$$

Induction

Applying the induction principle via apply can be cumbersome.

Induction

Applying the induction principle via apply can be cumbersome.

The tactic elim eases the applications of such principles.

$$\frac{\mathsf{P}\colon \mathsf{int} \to \mathsf{bool}}{0 \leq \mathsf{x} \Rightarrow \mathsf{P}\ \mathsf{x}} \quad \hookrightarrow \quad \mathsf{elim}/\mathsf{ind}\ \mathsf{x}$$

$$1.\ \frac{\mathsf{P} : \mathsf{int} \to \mathsf{bool}}{\mathsf{P}\ 0} \quad 2.\ \frac{\mathsf{P} : \mathsf{int} \to \mathsf{bool}}{\mathit{forall}\ (\mathsf{x} : \mathsf{int}),\ 0 \leq \mathsf{x} \to \mathsf{P}\ \mathsf{x} \to \mathsf{P}\ (\mathsf{x}+1)}$$

Automation

EasyCrypt comes with some automation tactics:

- progress break the goal by repetead applications of the introduction based tactics (split, move, ...)
- trivial: same as progress, but try to close subgoals.
- smt: try to solve the goal calling external SMT solvers.

Tacticals

Tacticals are operators on tactics.

Tacticals

Tacticals are operators on tactics.

- t1; t2
  apply t1 and then t2 on all generated subgoals

Tacticals

Tacticals are operators on tactics.

- t1; t2
  apply t1 and then t2 on all generated subgoals
- t; [t1|...|tn]
  apply t and then each of the t_i to the i^{th} subgoal

Tacticals

Tacticals are operators on tactics.

- t1; t2
  apply t1 and then t2 on all generated subgoals
- t; [t1|...|tn]
  apply t and then each of the t_i to the i^{th} subgoal
- do t
  repeat t as much as possible, at least one time

  this tactic takes the same multiplier of rewrite
  do! t, do? t, do n! e, do n? t

Tacticals

Tacticals are operators on tactics.

- t1; t2
  apply t1 and then t2 on all generated subgoals
- t; [t1|...|tn]
  apply t and then each of the t_i to the i^{th} subgoal
- do t
  repeat t as much as possible, at least one time
  this tactic takes the same multiplier of rewrite
  do! t, do? t, do n! e, do n? t
- try t
  try to apply t, or nothing if t cannot by applied

Tacticals

Tacticals are operators on tactics.

- t1; t2
  apply t1 and then t2 on all generated subgoals
- t; [t1|...|tn]
  apply t and then each of the t_i to the i^{th} subgoal
- do t
  repeat t as much as possible, at least one time
  this tactic takes the same multiplier of rewrite
  do! t, do? t, do n! e, do n? t
- try t
  try to apply t, or nothing if t cannot by applied
- by t1; ...; tn
  apply t1; ...; tn and then try to close all the subgoals via trivial. fail if all the subgoals cannot be solved.

Tacticals

Tacticals are operators on tactics.

- t1; first t2

  apply t1 and then t2 on the first subgoal
- t1; last t2

  apply t1 and then t2 on the last subgoal
- variants: t1; first n t2, t1; last n t2

Tacticals

Tacticals are operators on tactics.

- t1; first t2

  apply t1 and then t2 on the first subgoal
- t1; last t2

  apply t1 and then t2 on the last subgoal
- variants: t1; first n t2, t1; last n t2
- t; first n last

  apply t and then shift the n first goals to the end

Tacticals - Intros

Tacticals are operators on tactics.

- t $\Rightarrow$ ip1 ... ipn

  apply t and then execute the introduction of ip1 ... ipn

Tacticals - Intros

Tacticals are operators on tactics.

- t ⇒ ip1 ... ipn

  apply t and then execute the introduction of ip1 ... ipn

  - t⇒ x

    introduce a name / an hypothesis

  - t⇒ [ip1|...ipn]

    execute ip_i on the i^{th} subgoal

    + do a case analysis if not done by t

  - t⇒ →

    introduce an equational hypothesis and rewrite it

  - t⇒ {h}

    clear the hypothesis h

  - t⇒ //

    execute trivial

Plan

Proving in EasyCrypt

Proving in the Ambient Logic

Proving in 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
  $(1 + \mathrm{P.r} + \mathrm{x})\{1\}$ is equivalent to $1 + \mathrm{P.r}\{1\} + \mathrm{x}\{1\}$
- Equalities are restricted to variables
  $=\{\mathrm{x},\mathrm{P.r}\}$ stands for $\mathrm{x}\{1\} = \mathrm{x}\{2\} \wedge \mathrm{P.r}\{1\} = \mathrm{P.r}\{2\}$

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

Basic rules: rule of consequence

- exfalso

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

- conseq L | conseq (_ : P' ==> Q)

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

Basic proof rules: case

- case A

$$\frac{\{P \wedge A\}\ c \sim c'\ \{Q\} \qquad \{P \wedge \neg A\}\ c \sim c'\ \{Q\}}{\{P\}\ c \sim c'\ \{Q\}}$$

Basic proof rules: skip and sequence

- skip

$$\frac{P \Rightarrow Q}{\{P\}\ \epsilon \sim \epsilon\ \{Q\}}$$

- seq i j : R

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

where $i = |c_1|$ and $j = |c_1'|$.

Basic proof rules: assignment

Weakest Precondition

- wp - apply the assigmnent rule as much as possible

$$\frac{}{\{Q\{x_{\langle 1\rangle} \leftarrow e_{\langle 1\rangle}\}\}\ x := e \sim \epsilon\ \{Q\}}$$

$$\frac{}{\{Q\{x_{\langle 2\rangle} \leftarrow e_{\langle 2\rangle}\}\}\ \epsilon \sim x := e\ \{Q\}}$$

Basic proof rules: assignment

```
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

- rnd $\{1\}$ - one side rule

$$\frac{P = \text{lossless } d \wedge \forall v \in \text{supp } d, Q\{x_{\langle 1 \rangle} \leftarrow v\}}{\{P\}\ x = \$d \sim \epsilon\ \{Q\}}$$

- rnd f f^{-1} – two side rule – f and f^{-1} are optional

$$\frac{Q' = \forall v \in \text{supp } d, Q\{(x_{\langle 1 \rangle}, x'_{\langle 2 \rangle}) \leftarrow (v, f\ v)}{\{Q'\}\ x = \$d \sim x' = \$d'\ \{Q\}}$$

  - f is a bijection $(\text{supp}\, d)$ and $(\text{supp}\, d')$
  - $\forall x \in \text{supp}\, d.\, d\ x = d'(fx)$

Example

---

pre = true

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

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

---

rnd (*fun* x, x + 2) (*fun* x, x − 2).

---

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

Example

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

The function f is (*fun* x, x+2) with (*fun* x, x−2) as its inverse.

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: conditionals

- **if** $\{1\}$ | **if** $\{2\}$ - one side rule

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

- **if** - two side rule

$$\frac{\begin{array}{c} P \Rightarrow e_{\langle 1 \rangle} \Leftrightarrow e'_{\langle 2 \rangle} \\ \{P \wedge e_{\langle 1 \rangle}\}\ c_t \sim c'_t\ \{Q\} \qquad \{P \wedge \neg e_{\langle 1 \rangle}\}\ c_f \sim c'_f\ \{Q\} \end{array}}{\{P\}\ \textbf{if}\ e\ \textbf{then}\ c_t\ \textbf{else}\ c_f \sim \textbf{if}\ e'\ \textbf{then}\ c'_t\ \textbf{else}\ c'_f\ \{Q\}}$$

- note: works only when the **if** is the first instruction

Basic proof rules: while

- **while** I $=$ two side rule (simplified)

$$\frac{\{e_{\langle 1\rangle} \wedge e'_{\langle 2\rangle} \wedge I\}\ c \sim c'\ \{I'\} \qquad I' \equiv e_{\langle 1\rangle} \Leftrightarrow e'_{\langle 2\rangle} \wedge I}{\{I'\}\ \textbf{while}\ e\{c\} \sim \textbf{while}\ e'\{c'\}\ \{\neg e_{\langle 1\rangle} \wedge \neg e'_{\langle 2\rangle} \wedge I\}}$$

- a one side version exists

Basic proof rules: call

- simplified version

$$\frac{\{P_f\}\ f \sim f'\ \{Q_f\} \qquad P \Rightarrow P_f\{x_{\langle 1\rangle}, x'_{\langle 2\rangle} \leftarrow e_{\langle 1\rangle}, e'_{\langle 2\rangle}\} \qquad \forall\ r\ r'.\ Q_f\{\mathsf{res}_{\langle 1\rangle}, \mathsf{res}_{\langle 2\rangle} \leftarrow r, r'\} \Rightarrow Q\{y_{\langle 1\rangle}, y'_{\langle 2\rangle} \leftarrow r, r'\}}{\{P\}\ y = f(e) \sim y' = f'(e')\ \{Q\}}$$

  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:

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

  where c_1 and c_2 are semantically equivalent.

- c_2 is automatically generated by the rule (syntax directed).

Program transformations: swap

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

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

  where 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

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'}
```

Other tactics based on program transformation

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

From functions to statements

- **proc**

$$\frac{\{P\}\ c_f \sim c_g\ \{Q\{\mathsf{res}_{\langle 1\rangle}, \mathsf{res}_{\langle 2\rangle} \leftarrow r_{f\langle 1\rangle}, r_{g\langle 2\rangle}\}\}}{\{P\}\ f \sim g\ \{Q\}}$$

  - 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

- this rule only works for concrete functions

From pRHL to probabilities

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

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

From pRHL to probabilities

- in EasyCrypt

---

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

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

- variant: byequiv (_ : P ==> Q).