Lecture 2 : Interactive Proofs in EasyCrypt

July 16th, 2013

The Ambient Logic

EasyCrypt ambient logic is a general higher-order logic.

In this talk

- How define facts about user defined operators
- How to prove them when automatic techniques do not work

Plan

Types

EasyCrypt is a typed language:

- It comes with a set of core types

  unit, bool, int, real, tuple, lists ...

  Some of these types are polymorphic (type constructor)

- Possibility to create type aliases

  type $\alpha\, u$ $= \alpha * \alpha$
  type v $=$ int u
  type w $=$ int list

- Possibility to create abstract types

  type t
  type $\alpha\, u$

Types

EasyCrypt is a typed language:

- It comes with a set of core types

  unit, bool, int, real, tuple, lists ...

  Some of these types are polymorphic (type constructor)

- Possibility to create type aliases

  type $\alpha\ u$ $= \alpha * \alpha$
  type v $=$ int u
  type w $=$ int list

- Possibility to create abstract types

  type t
  type $\alpha\ u$

Types

EasyCrypt is a typed language:

- It comes with a set of core types

  unit, bool, int, real, tuple, lists ...

  Some of these types are polymorphic (type constructor)

- Possibility to create type aliases

  type $\alpha\, u$ $= \alpha * \alpha$
  type v $=$ int u
  type w $=$ int list

- Possibility to create abstract types

  type t
  type $\alpha\, u$

Types

EasyCrypt is a typed language:

- It comes with a set of core types

  unit, bool, int, real, tuple, lists ...

  Some of these types are polymorphic (type constructor)

- Possibility to create type aliases

  type $\alpha\, u$ $= \alpha * \alpha$
  type v $=$ int u
  type w $=$ int list

- Possibility to create abstract types

  type t
  type $\alpha\, u$

Expressions - Functional language

EasyCrypt comes with a functional language:

- concrete operators:

```
op f1 (b : bool) (x y : int)   = b ? (x − y) : (x + y).
op f2 (xs : int list) (x : int) = map (lambda (z : int), z + x) xs.
op f3 (xs : 'a list)            = fold (lambda v _, v + 1) 0 xs.
```

- abstract operators:

$$\begin{aligned} \text{map} &: (\alpha \to \beta) \to \alpha\ \mathit{list} \to \beta\ \mathit{list} \\ \text{fold} &: (\alpha \to \beta) \to \alpha\ \mathit{list} \to \beta\ \mathit{list} \end{aligned}$$

Expressions - Functional language

EasyCrypt comes with a functional language:

- concrete operators:

```
op f1 (b : bool) (x y : int)  = b ? (x − y) : (x + y).
op f2 (xs : int list) (x : int) = map (lambda (z : int), z + x) xs.
op f3 (xs : 'a list)           = fold (lambda v _, v + 1) 0 xs.
```

- abstract operators:

$$\begin{aligned} \text{map} &: (\alpha \to \beta) \to \alpha\ \mathit{list} \to \beta\ \mathit{list} \\ \text{fold} &: (\alpha \to \beta) \to \alpha\ \mathit{list} \to \beta\ \mathit{list} \end{aligned}$$

Expressions - Functional language

EasyCrypt comes with a functional language:

- concrete operators:

```
op f1 (b : bool) (x y : int)  = b ? (x − y) : (x + y).
op f2 (xs : int list) (x : int) = map (lambda (z : int), z + x) xs.
op f3 (xs : 'a list)            = fold (lambda v _, v + 1) 0 xs.
```

- abstract operators:

$$\begin{aligned} \text{map} &: (\alpha \rightarrow \beta) \rightarrow \alpha\ \textit{list} \rightarrow \beta\ \textit{list} \\ \text{fold} &: (\alpha \rightarrow \beta) \rightarrow \alpha\ \textit{list} \rightarrow \beta\ \textit{list} \end{aligned}$$

Predicates / Formulas

- Predicates are boolean operators:

$$\textbf{op}\ \text{mypred} : \text{int} \to \text{int} \to \text{bool}.$$

- These predicates can be defined:

$$\textbf{pred}\ \text{mypred}\ (x\ y : \text{int}) = (0 \leq x) \wedge (0 \leq y) \wedge (2 * x \leq y)$$

- Formulas constructors:

forall $(x : t), \phi$	$(\forall (x : t), \phi)$	*exists* $(x : t), \phi$	$(\exists (x : t), \phi)$
$\phi_1 /\backslash\ \phi_2$	$(\phi_1 \wedge \phi_2)$	$\phi_1 \backslash/\ \phi_2$	$(\phi_1 \vee \phi_2)$
$\phi_1 => \phi_2$	$(\phi_1 \Rightarrow \phi_2)$	$\phi_1 <=> \phi_2$	$(\phi_1 \Leftrightarrow \phi_2)$
$!\,\phi$	$(\neg\, \phi)$	+ dedicated formulas for p(R)HL	

Axioms / Lemmas

- Formulas for operators axiomatization:

```
op count : 'a list -> int.

axiom count_nil  : count [] = 0.
axiom count_cons : forall (x : 'a) (xs : 'a list),
                      count (x :: xs) = 1 + (count xs).
```

- Formulas for stating facts:

```
lemma fact (x y : int): x ≤ 0 → y ≤ 0 → 0 ≤ x * y.
```

Axioms / Lemmas

- Formulas for operators axiomatization:

```
op count : 'a list -> int.

axiom count_nil  : count [] = 0.
axiom count_cons : forall (x : 'a) (xs : 'a list),
                     count (x :: xs) = 1 + (count xs).
```

- Formulas for stating facts:

```
lemma fact (x y : int): x ≤ 0 → y ≤ 0 → 0 ≤ x * y.
```

Plan

Stating a theorem

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

$$\left.\begin{array}{l} b1 : bool \\ b2 : bool \\ b3 : bool \end{array}\right\}\text{local hypotheses (context)}$$

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

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

Stating a theorem

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

$$\left.\begin{array}{l} b1 : bool \\ b2 : bool \\ b3 : bool \end{array}\right\}\text{local hypotheses (context)}$$

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

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

Stating a theorem

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

$$\left.\begin{array}{l} b1 : bool \\ b2 : bool \\ b3 : bool \end{array}\right\}\text{local hypotheses (context)}$$

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

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.
  intros⇒ hb12.
```

$$\begin{array}{l} b1 : \text{bool} \\ b2 : \text{bool} \\ b3 : \text{bool} \\ hb12 : b1 \Rightarrow b2 \end{array}$$

$$\overline{\overline{\qquad\qquad\qquad\qquad}}$$

$$(b2 \Rightarrow b3) \Rightarrow b1 \Rightarrow b3$$

Continuing the proof

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

$$\begin{array}{l}
b1 : bool \\
b2 : bool \\
b3 : bool \\
hb12 : b1 \Rightarrow b2 \\
hb23 : b2 \Rightarrow b3 \\
hb1 : b1 \\
\hline\hline
b3
\end{array}$$

Continuing the proof

```
lemma mylemma b1 b2 b3 : ...
proof.
  intros=> hb12 bh23 hb1.
  apply hb23.
```

$$
\begin{array}{l}
b1 : bool \\
b2 : bool \\
b3 : bool \\
hb12 : b1 \Rightarrow b2 \\
hb23 : b2 \Rightarrow b3 \\
hb1 : b1 \\
\hline\hline
b2
\end{array}
$$

Continuing the proof

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

b1 : bool
b2 : bool
b3 : bool
hb12 : b1 $\Rightarrow$ b2
hb23 : b2 $\Rightarrow$ b3
hb1 : b1

---

b1

Continuing the proof

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

Proof completed

Continuing the proof

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

Propositional logic

- $b1 \Rightarrow b2 \Rightarrow b3$

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

  As an hypothesis [apply]

Propositional logic

- $b1 \Rightarrow b2 \Rightarrow b3$

  As a goal [intros⇒ b1 b2]

$$\frac{}{b1 \Rightarrow b2 \Rightarrow b3} \hookrightarrow \frac{\begin{array}{c} b1 : bool \\ b2 : bool \end{array}}{b3}$$

  As an hypothesis [apply]

Propositional logic

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

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

  As an hypothesis [apply]

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

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

Propositional logic - connectors

- Conjunction: $a \wedge b$

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

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

Propositional logic - connectors

- Conjunction: a ∧ b

As a goal [split] (prove a ∧ b)

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

As an hypothesis [elim ab] (destruct a ∧ b in a *and* b)

Propositional logic - connectors

- Conjunction: a $\wedge$ b

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

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

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

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

Propositional logic - connectors

- Disjunction: a ∨ b

  As a goal

  As an hypothesis [elim ab] (case analysis on a ∨ b)

Propositional logic - connectors

- Disjunction: a ∨ b

  As a goal

  - [left] (prove a ∨ b by proving a)

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

  - [right] (prove a ∨ b by proving b)

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

  As an hypothesis [elim ab] (case analysis on a ∨ b)

Propositional logic - connectors

- Disjunction: $a \vee b$

  As a goal

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

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

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

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

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

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

Propositional logic - existential

- Existential: *exists* $x : t, \phi(x)$

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

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

Propositional logic - existential

- Existential: *exists* x : t, ϕ (x)

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

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

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

Propositional logic - existential

- Existential: *exists* x : t, ϕ (x)

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

$$\frac{}{\text{exists x : t}, \phi(\text{x})} \hookrightarrow \frac{}{\phi(\text{v})}$$

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

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

Boolean case analysis

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

$$\frac{\begin{array}{l}a : \text{bool}\\ b : \text{bool}\end{array}}{a \oplus b = (a \wedge !b) \mid\mid (!a \wedge b)}$$

(case a) leads to

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

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

Boolean case analysis

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

$$\frac{\begin{array}{l} a : \text{bool} \\ b : \text{bool} \end{array}}{a \oplus b = (a \wedge !b) \mid\mid (!a \wedge b)}$$

(case a) leads to

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

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

Boolean case analysis

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

$$\frac{\begin{array}{l} a : \text{bool} \\ b : \text{bool} \end{array}}{a \oplus b = (a \wedge !b) \mid\mid (!a \wedge b)}$$

(case a) leads to

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

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

Identification up to computations

EasyCrypt comes with a set of simplification rules.

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

simplify leads to

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

that can be easily solved by reflexivity.

Identification up to computations

EasyCrypt comes with a set of simplification rules.

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

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 ($a \land \text{false} \to \text{false}$)

Terms that are equal up to computations are considered as identical

$$\dfrac{\begin{array}{l} a : \text{bool} \\ b : \text{bool} \end{array}}{!a \Rightarrow \text{false} \oplus b = (\text{false} \land !b) \lor (!\text{false} \land b)}$$

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 h

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

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

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

Rewrite - replace equals by equals

rewrite comes in different flavor:

- rewrite −h : from right to left
- rewrite mh where m is a multiplier

?	as many times as possible
!	as many times as possible, at least one
n?	at most n times
n!	exactly n times

- rewrite {o}h where o is a sequence of positive integers

  Rewrites the o^{th} occurrences only.

Rewrite - replace equals by equals

rewrite comes in different flavor:

- rewrite −h : from right to left
- rewrite mh where m is a multiplier

?	as many times as possible
!	as many times as possible, at least one
n?	at most n times
n!	exactly n times

- rewrite {o}h where o is a sequence of positive integers

  Rewrites the o^{th} occurrences only.

Rewrite - replace equals by equals

rewrite comes in different flavor:

- rewrite −h : from right to left
- rewrite mh where m is a multiplier

?	as many times as possible
!	as many times as possible, at least one
n?	at most n times
n!	exactly n times

- rewrite {o}h where o is a sequence of positive integers

  Rewrites the o^{th} occurrences only.

Rewrite - replace equals by equals

rewrite comes in different flavor:

- rewrite $-$h : from right to left
- rewrite mh where m is a multiplier

?	as many times as possible
!	as many times as possible, at least one
n?	at most n times
n!	exactly n times

- rewrite {o}h where o is a sequence of positive integers

  Rewrites the o^{th} occurrences only.

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

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

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

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

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

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

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

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

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

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

Logical cut

The tactic cut: ϕ allows to do a forward chaining

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

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

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

Induction

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

$$\forall(\mathsf{p} : \mathsf{t} \rightarrow \mathsf{bool}), \phi_1 \rightarrow \ldots \rightarrow \phi_n, \forall(\mathsf{x} : \mathsf{t}), \mathsf{psi1}(\mathsf{x}) \rightarrow \ldots \rightarrow \mathsf{psin}(\mathsf{x}) \rightarrow \mathsf{p}\ \mathsf{x}$$

For example, for natural numbers:

forall (p : int $\rightarrow$ t), p 0 $\Rightarrow$
  (*forall* (x : int), 0 $\leq$ x $\Rightarrow$ p x $\Rightarrow$ p (x + 1)) $\Rightarrow$
    forall (x : int), 0 $\leq$ x $\Rightarrow$ p x

Induction

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

$$\forall (\mathsf{p} : \mathsf{t} \to \mathsf{bool}), \phi_1 \to \ldots \to \phi_n, \forall (\mathsf{x} : \mathsf{t}), \mathsf{psi1}(\mathsf{x}) \to \ldots \to \mathsf{psin}(\mathsf{x}) \to \mathsf{p}\ \mathsf{x}$$

For example, for natural numbers:

forall (p : int → t), p 0 ⇒
  (*forall* (x : int), $0 \leq$ x ⇒ p x ⇒ p (x + 1)) ⇒
    forall (x : int), $0 \leq$ x ⇒ p x

Induction

Applying the induction principle via apply can be cumbersome.

The tactic elimT eases the applications of such principles.

$$\dfrac{\text{P: int} \to \text{bool}}{0 \leq \text{x} \Rightarrow \text{P x}} \quad \hookrightarrow \quad \text{elimT ind x}$$

1. $\dfrac{\text{P : int} \to \text{bool}}{\text{P } 0}$ 2. $\dfrac{\text{P : int} \to \text{bool}}{\forall(\text{x : int}),\ 0 \leq \text{x} \to \text{P x} \to \text{P (x+1)}}$

Induction

Applying the induction principle via apply can be cumbersome.

The tactic elimT eases the applications of such principles.

$$\frac{\text{P: int} \to \text{bool}}{0 \leq \text{x} \Rightarrow \text{P x}} \quad \hookrightarrow \quad \text{elimT ind x}$$

1. $$\frac{\text{P : int} \to \text{bool}}{\text{P } 0}$$

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

Automation

EasyCrypt comes with some automation tactics:

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

Plan

Tacticals

Tacticals are operators on tactics.

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; 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; 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; 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; 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
- t; first n last

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

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

  - t$\Rightarrow$ x

    introduce a name / an hypothesis
  - t$\Rightarrow$ [ip1|...ipn]

    execute ip$_i$ on the i^{th} subgoal

    $+$ do a case analysis if not done by t
  - t$\Rightarrow$ $\rightarrow$

    introduce an equational hypothesis and rewrite it
  - t$\Rightarrow$ {h}

    clear the hypothesis h
  - t$\Rightarrow$ //

    execute trivial

Tacticals - Intros

Tacticals are operators on tactics.

- t $\Rightarrow$ ip1 ... ipn

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

  - t$\Rightarrow$ x

    introduce a name / an hypothesis
  - t$\Rightarrow$ [ip1|...ipn]

    execute ip_i on the i^{th} subgoal

    + do a case analysis if not done by t
  - t$\Rightarrow$ $\rightarrow$

    introduce an equational hypothesis and rewrite it
  - t$\Rightarrow$ {h}

    clear the hypothesis h
  - t$\Rightarrow$ //

    execute trivial

Plan

trade-off between interactive / automatic proof

- EasyCrypt has now two kinds of tactics
  - low-level, interactive ones
  - the SMT hammer

  The difficulty is to find the right trade-off between the two.
  - SMT goal resolution success can be very unstable
  - SMT can be very good in solving large or numerous problems generated by p(R)HL judgments

- qed does not mark the end of the proof.