MITLL/NRL Panel Discussion: Understanding Provability and Truth in EasyCrypt

Alley Stoughton

First EasyCrypt Workshop
University of Pennsylvania
July 19, 2013

LINCOLN LABORATORY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

This work is sponsored by the Director of Naitional Intelligence under Air Force Contract FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the author, and are not necessarily endorsed by the United States Government.

Introduction

- When learning EasyCrypt, my biggest challenge was developing good intuitions about whether a goal or claim was:
  - false or
  - true but not provable or
  - provable
- I think this was because:
  - Apart from Santiago Zanella's thesis, the available papers and documentation gave only a rough explanation of truth (model theory)
  - It's nontrivial to apply the model theory in Santiago's thesis to actual EasyCrypt
- I'm going to illustrate my learning process by considering several examples

Random Shuffling

The first two examples involve random shuffling of lists:

```
fun Shuffle(xs : int list) : int list = {
  var ys : int list;
  var i, x : int;
  ys = [];
  while (xs <> []) {
    i = [0 .. length(xs) - 1];
    x = proj(nth(xs, i));
    xs = rm(xs, i);
    ys = x :: ys;
  }
  return ys;
}
```

Shuffling in UCI Client Proof

- Ignoring extraneous detail, we were trying to prove

```
xs = A1();
b = A2(xs);
```

equivalent to

```
xs = A1();
ys = Shuffle(xs);
b = A2(ys);
```

- But this is false: e.g., `A2` can test whether argument is equal to `xs`

Shuffling Lists with Same Elements

- In contrast, is

```
(xs, ys) = A1();
zs = Inter(xs, ys);  (* maintains order of xs *)
ws = Shuffle(zs);
b = A2(ws);
```

equivalent to

```
(xs, ys) = A1();
zs = Inter(ys, xs);  (* maintains order of ys *)
ws = Shuffle(zs);
b = A2(ws);
```

?

LINCOLN LABORATORY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Proof Crux

```
pre   = ={ys} && IsShuffleOf(xs{1}, xs{2}) && xs{1} <> []
stmt1 = 1 : i = [0 .. length(xs) - 1];
stmt2 = 1 : i = [0 .. length(xs) - 1];
post  = let xs_R = rm(xs{2}, i{2}) in
        let xs_L = rm(xs{1}, i{1}) in
        (xs_L <> []) = (xs_R <> []) &&
        proj(nth(xs{1}, i{1})) :: ys{1} =
        proj(nth(xs{2}, i{2})) :: ys{2} &&
        IsShuffleOf(xs_L, xs_R)
```

- **This is handled by**

```
rnd (i -> fst(findPerm(xs{1}, xs{2}))[i]),
    (i -> snd(findPerm(xs{1}, xs{2}))[i]).
```

Permutation Problem

- If `p` is a permutation on `{0, ..., bnd - 1}`, is

```
m = empty_map; i = 0;
while (i < bnd) {
  m[i] = {0, 1}; i = i + 1;
}
```

equivalent to

```
m = empty_map; i = 0;
while (i < bnd) {
  m[p[i]] = {0, 1}; i = i + 1;
}
```

?

Permutation Problem

- **If we introduce an oracle for filling an element of our map, we can solve our problem using lazy random sampling**

```
fun O(n : int) : unit = {
  if (!in_dom(n, m)) {
    m[n] = {0, 1};
  }
}
```

Permutation Problem

- **We transition to:**

```
m = empty_map;
j = 0;
while (j < bnd) {
  O(j);
  j = j + 1;
}
i = 0;
while (i < bnd) {
  O(p[i]);
  i = i + 1;
}
```

- **This works, since the second while loop is redundant**

Permutation Problem

- **Then we use lazy random sampling to swap the while loops:**

```
m = empty_map;
i = 0;
while (i < bnd) {
  O(p[i]);
  i = i + 1;
}
j = 0;
while (j < bnd) {
  O(j);
  j = j + 1;
}
```

- **The second while loop is then redundant, and the first can be turned into our target.**

LINCOLN LABORATORY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Permutation Problem

- **But we'd like to be able to transition from**

```
m = empty_map; i = 0;
while (i < bnd) {
  m[i] = {0, 1}; i = i + 1;
}
```

to

```
m = empty_map; i = 0;
while (i < bnd) {
  m[p[i]] = {0, 1}; i = i + 1;
}
```

within a single equivalence proof

Mismatched Random Assignments

```
pre   = ={xs, n} && n{1} >= 0 && m{2} >= 0
stmt1 = 1 : while (n > 0) {
              i = [1 .. n];
              xs = xs ++ [i];
              n = n - 1;
            }
stmt2 = 1 : while (n + m > 0) {
              i = [1 .. n + m];
              if (i <= n) { xs = xs ++ [i]; n = n - 1; }
              else m = m - 1;
            }
post  = ={xs}
```

- **This is true, but seems not to be provable**
- **pRHL seems wrong framework for proof**

Conclusions

- **Need Coq foundation for EasyCrypt**
  - **Used to understand truth**
  - **Right level for some proofs**
- **Clear informal explanation of model theory also important**
- **Need abstraction mechanism able to take whole-game, multi-step proofs and turn them into tactics**