Coq for Programming Language Proofs — A Personal Experience

Ximeng Li

Language-Based Technology
DTU Compute

February 25, 2015

Coq — A Brief Overview

- A proof assistant developed by INRIA (`https://coq.inria.fr/`)
- Coq provides:
  - a formal language to write mathematical definitions, executable algorithms and theorems
  - an environment for semi-interactive development of machine-checked proofs
  - a tactic definition language for extensibility
  - the ability to use dependent types

Slide adapted from one by Xinyu Feng (USTC, China)

Coq — Recent Gain in Popularity

MORE ACM AWARD WINNERS

AWARDS

Software System Award

ABOUT THIS AWARD

Awarded to an institution or individual(s) recognized for developing a software system that has had a lasting influence, reflected in contributions to concepts, in commercial acceptance, or both. The Software System Award carries a prize of $35,000. Financial support for the Software System Award is provided by IBM.

Coq Selected As Recipient Of The 2013 Software System Award

Other recipients of the award:
Unix、TCP/IP、World-Wide Web、Java、Make、VMWare、Eclipse、LLVM …

Slide from Xinyu Feng (USTC, China)

Coq for Information Flow Proofs

Hanne Riis Nielson, Flemming Nielson, and Ximeng Li.
Disjunctive Information Flow.

- The technical development:
  - small concurrent language (where the flows arise from)
  - instrumented semantics (what the flows are)
  - security type system (how to ensure that the flows are secure)
  - soundness (why the assurance is faithful)
- The Coq development:
  - parallels that of the formulations in the paper
  - more verbose/precise
  - might not be as comprehensible

Coq for Information Flow Proofs

Hanne Riis Nielson, Flemming Nielson, and Ximeng Li.
Disjunctive Information Flow.

- The technical development:
  - small concurrent language (where the flows arise from)
  - instrumented semantics (what the flows are)
  - security type system (how to ensure that the flows are secure)
  - soundness (why the assurance is faithful)
- The Coq development:
  - parallels that of the formulations in the paper
  - more verbose/precise
  - might not be as comprehensible

Syntax

On paper

$$S ::= \ldots \mid \text{if } {}^{\ell}b \text{ then } S_1 \text{ else } S_2 \mid \ldots \mid \{X\}\, S \mid \ldots$$

In Coq

```
Inductive com : Type :=
  ...
  | CIf : lbl → bexp → com → com → com
  ...
  | CBlk : block → com → com.

...
Notation "l ':IF' b 'THEN' S1 'ELSE' S2 'FI'" :=
  (CIf l b S1 S2) (at level 80, right associativity).
Notation " X 'OVER' S" :=
  (CBlk X S) (at level 60, right associativity).
...
```

Syntax

On paper

$$S ::= \ldots \mid \text{if } {}^{\ell}b \text{ then } S_1 \text{ else } S_2 \mid \ldots \mid \{X\}\, S \mid \ldots$$

In Coq

```
Inductive com : Type :=
  ...
  | CIf : lbl → bexp → com → com → com
  ...
  | CBlk : block → com → com.

...
Notation "l ':IF' b 'THEN' S1 'ELSE' S2 'FI'" :=
  (CIf l b S1 S2) (at level 80, right associativity).
Notation " X 'OVER' S" :=
  (CBlk X S) (at level 60, right associativity).
...
```

Semantics

On paper

$$\vdash_p \langle \text{if } {}^{\ell} b \text{ then } S_1 \text{ else } S_2; \sigma\rangle \xrightarrow[\tau]{F} \langle \{\mathsf{FV}(b)\}\, S_1; \sigma\rangle \quad \text{if } \mathcal{B}[\![b]\!]\sigma = \mathbf{tt} \text{ and } F = (\mathsf{FV}(b) \cup \{p\}) \times \{p\}$$

In Coq

```
Definition fls_if (p: pr) (b: bexp) : flow :=
  (car_prod (Union src_snk (to_ss (vvb b)) (Singleton src_snk (Pr p)))
         (Singleton src_snk (Pr p))).
...
Inductive com_step : pr → (prod com state) → flow → trans_label →
                (prod com state) → Prop :=
  ...
  | S_IfTrue : ∀ l p (fl:flow) b st S1 S2,
    (beval st b = true) → (fl = fls_if p b) →
    (l:IF b THEN S1 ELSE S2 FI) / st ⇒ [p,fl,Trans_LTau] ((vvb b) OVER S1) / st
  ...
 where " S '/' st '⇒' '[' P ',' F ',' alpha ']' S' '/' st' " :=
     (com_step (P) (S,st) F alpha (S',st')) : sem_scope.
```

Semantics

On paper

$$\vdash_p \langle \text{if }^{\ell} b \text{ then } S_1 \text{ else } S_2; \sigma\rangle \xrightarrow[\tau]{F} \langle \{\text{FV}(b)\}\, S_1; \sigma\rangle \quad \text{if } \mathcal{B}[\![b]\!]\sigma = \mathbf{tt} \text{ and } F = (\text{FV}(b) \cup \{p\}) \times \{p\}$$

In Coq

```
Definition fls_if (p: pr) (b: bexp) : flow :=
  (car_prod (Union src_snk (to_ss (vvb b)) (Singleton src_snk (Pr p)))
         (Singleton src_snk (Pr p))).
...
Inductive com_step : pr → (prod com state) → flow → trans_label →
                  (prod com state) → Prop :=
  ...
  | S_IfTrue : ∀ l p (fl:flow) b st S1 S2,
     (beval st b = true) → (fl = fls_if p b) →
     (l:IF b THEN S1 ELSE S2 FI) / st ⇒ [p,fl,Trans_LTau] ((vvb b) OVER S1) / st
  ...
 where " S '/' st '⇒' '[' P ',' F ',' alpha ']' S' '/' st' " :=
      (com_step (P) (S,st) F alpha (S',st')) : sem_scope.
```

Semantics

Computational Definition in Coq

```
Fixpoint beval (st : state) (b : bexp) : bool :=
  match b with
  | BTrue       ⇒ true
  | BFalse      ⇒ false
  | BEq a1 a2   ⇒ beq_nat (aeval st a1) (aeval st a2)
  | BLe a1 a2   ⇒ ble_nat (aeval st a1) (aeval st a2)
  | BNot b1     ⇒ negb (beval st b1)
  | BAnd b1 b2  ⇒ andb (beval st b1) (beval st b2)
  end.
```

Typing

On paper

$$\frac{X \cup \mathrm{FV}(b) \vdash_p \{\phi \wedge b\} S_1 \{\psi\} \quad X \cup \mathrm{FV}(b) \vdash_p \{\phi \wedge \neg b\} S_2 \{\psi\}}{X \vdash_p \{\phi\} \text{if } {}^{\ell} b \text{ then } S_1 \text{ else } S_2 \{\psi\}}$$

$$\text{if} \quad \Phi(\ell) = \phi \text{ and } \forall P \in \mathcal{P} : (\phi \ \wedge \ \bigwedge_y y \in P_{\mathrm{V}}(y)) \Rightarrow p \,\mathsf{E}\, P_{\mathrm{R}}[\mathrm{FV}(b) \cup X]$$

Typing

In Coq

```
Inductive well_typed np k base (dop : pr → sec_dom_t base)
        (pols: Ensemble (policy_t base)) (PPhi: lbl→ Assertion) (eta: var_thread_t) :
  (Ensemble id) → pr → Assertion → com → Assertion → Prop :=
 | TP_IF : ∀ l b S1 S2 X p phi psi,
          (np, k, base, dop, pols, PPhi, eta, (X |U| (vvb b)), p |–
           (fun st ⇒ (phi st /\ (bassn b st))), S1, psi) →
          (np, k, base, dop, pols, PPhi, eta, (X |U| (vvb b)), p |–
           (fun st ⇒ (phi st /\ ~(bassn b st))), S2, psi) →
          (PPhi l) << − >> phi →
             (∀ ipb cpb vp (sts:states),
               (length sts = np /\
                Ensembles.In _ pols (pair (pair ipb cpb) vp) /\ phi (sts@ k) /\
                (∀ (u:id), eta u < length sts →
                   Ensembles.In nat (vp (CV_Var u)) ((sts@ eta u) u))) →
               Ensembles.In _ (effective_c base
                   (cl_over_set base cpb (x_to_cv (vvb b |U| X)))) p)
          → (np,k,base,dop,pols,PPhi,eta,X,p |– phi,(l:IF b THEN S1 ELSE S2 FI),psi)
 where "np','k','base','dop','pols','PPhi','eta','X','p '|–' phi ',' S ',' psi"
     := (well_typed np k base dop pols PPhi eta X p phi S psi).
```

Security

On paper

$$
\begin{aligned}
&\textsf{sec}(P, F, P') \ \text{ iff} \\
&\forall (p, u') \in F : p \mathrel{\mathsf{E}} P'_{\mathrm{I}}(u') \ \wedge \\
&\forall (u, u') \in F : (P_{\mathrm{I}}(u) \sqsubseteq P'_{\mathrm{I}}(u') \wedge P_{\mathrm{R}}(u) \sqsubseteq P'_{\mathrm{R}}(u')) \ \wedge \\
&\forall (u, p') \in F : p' \mathrel{\mathsf{E}} P_{\mathrm{R}}(u) \ \wedge \\
&\forall y \in \overline{\mathsf{snd}(F)} : (P_{\mathrm{I}}(y) \sqsubseteq P'_{\mathrm{I}}(y) \wedge P_{\mathrm{R}}(y) \sqsubseteq P'_{\mathrm{R}}(y))
\end{aligned}
$$

In Coq

```
Definition sec base (ipb ipb': i_pol base) (cpb cpb': c_pol base)
           (vp vp':v_pol) (fl:flow) : Prop :=
  (∀ (p z: src_snk), (is_cv z = true) → (is_cv p = false) →
                  Ensembles.In src_to_snk fl (pair p z) →
                  Ensembles.In pr (effective_i base (ipb' (get_cv z))) (get_pr p))
  /\
  (∀ (y z: src_snk), (is_cv y = true) → (is_cv z = true) →
                  Ensembles.In src_to_snk fl (pair y z) →
                  (ipb (get_cv y) <=I[base] ipb' (get_cv z) /\
                   cpb (get_cv y) <=C[base] cpb' (get_cv z)))
  /\
  (∀ (y q: src_snk), (is_cv y = true) → (is_cv q = false) →
                  Ensembles.In src_to_snk fl (pair y q) →
                  Ensembles.In pr (effective_c base (cpb (get_cv y))) (get_pr q))
  /\
  (∀ (x:id), ~Ensembles.In _ (snds fl) (Var x) →
             ipb (CV_Var x) <=I[base] ipb' (CV_Var x) /\
             cpb (CV_Var x) <=C[base] cpb' (CV_Var x)).
```

Security

On paper

$$
\begin{aligned}
&\textcolor{red}{\mathsf{sec}(P, F, P')} \ \ \text{iff} \\
&\forall (p, u') \in F : p \,E\, P'_{\mathsf{I}}(u') \wedge \\
&\forall (u, u') \in F : (P_{\mathsf{I}}(u) \sqsubseteq P'_{\mathsf{I}}(u') \wedge P_{\mathsf{R}}(u) \sqsubseteq P'_{\mathsf{R}}(u')) \wedge \\
&\forall (u, p') \in F : p' \,E\, P_{\mathsf{R}}(u) \wedge \\
&\forall y \in \overline{\mathsf{snd}(F)} : (P_{\mathsf{I}}(y) \sqsubseteq P'_{\mathsf{I}}(y) \wedge P_{\mathsf{R}}(y) \sqsubseteq P'_{\mathsf{R}}(y))
\end{aligned}
$$

In Coq

```
Definition sec base (ipb ipb': i_pol base) (cpb cpb': c_pol base)
             (vp vp':v_pol) (fl:flow) : Prop :=
  (∀ (p z: src_snk), (is_cv z = true) → (is_cv p = false) →
                    Ensembles.In src_to_snk fl (pair p z) →
                    Ensembles.In pr (effective_i base (ipb' (get_cv z))) (get_pr p))
  /\
  (∀ (y z: src_snk), (is_cv y = true) → (is_cv z = true) →
                    Ensembles.In src_to_snk fl (pair y z) →
                    (ipb (get_cv y) <=I[base] ipb' (get_cv z) /\
                     cpb (get_cv y) <=C[base] cpb' (get_cv z)))
  /\
  (∀ (y q: src_snk), (is_cv y = true) → (is_cv q = false) →
                    Ensembles.In src_to_snk fl (pair y q) →
                    Ensembles.In pr (effective_c base (cpb (get_cv y))) (get_pr q))
  /\
  (∀ (x:id), ~Ensembles.In _ (snds fl) (Var x) →
               ipb (CV_Var x) <=I[base] ipb' (CV_Var x) /\
               cpb (CV_Var x) <=C[base] cpb' (CV_Var x)).
```

Soundness

Lemma (Subject Reduction/Soundness)

Assume $X \vdash_p \{\Phi(\mathsf{fst}(S))\}S\{\psi\}$ *and* $\sigma \models \Phi(\mathsf{fst}(S))$ *and* $P \in \mathcal{P}$ *and* $\sigma \models P_{\mathsf{V}}$. *Then there exists* $P' \in \mathcal{P}$ *such that:*

- *If* $\vdash_p \langle S; \sigma\rangle \xrightarrow[\alpha]{F} \langle S'; \sigma'\rangle$ *and* $(\alpha = ch?\vec{v}) \Rightarrow \bigwedge_i v_i \in P_{\mathsf{V}}(ch.i)$ *then* $X \vdash_p \{\Phi(\mathsf{fst}(S'))\}S'\{\psi\}$ *and* $\sigma' \models \Phi(\mathsf{fst}(S'))$ *and* $\sigma' \models P'_{\mathsf{V}}$ *and* $\mathsf{sec}(P, \{X\}\ F, P')$ *and* $(\alpha = ch!\vec{v}) \Rightarrow \bigwedge_i v_i \in P_{\mathsf{V}}(ch.i)$.
- *If* $\vdash_p \langle S; \sigma\rangle \xrightarrow[\alpha]{F} \sigma'$ *and* $(\alpha = ch?\vec{v}) \Rightarrow \bigwedge_i v_i \in P_{\mathsf{V}}(ch.i)$ *then* $\sigma' \models \psi$ *and* $\sigma' \models P'_{\mathsf{V}}$ *and* $\mathsf{sec}(P, \{X\}\ F, P')$ *and* $(\alpha = ch!\vec{v}) \Rightarrow \bigwedge_i v_i \in P_{\mathsf{V}}(ch.i)$.

Soundness

```
Lemma subject_reduction :
 ∀ np k dop S (sts: states) X p pols PPhi (psi: Assertion)
     ipb cpb vp st,
  np, k, base, dop, pols, PPhi, eta, X, p   ⊢   (PPhi (fst_lbl S)), S, psi →
  length sts = np →
  (
   0<=k → k<length sts → st = (sts@ k) → (all_sat_v eta sts vp) →
   (∼is_prefixed_stop S → (PPhi (fst_lbl S)) st) →
   (Ensembles.In _ pols (pair (pair ipb cpb) vp)) →
   ∃ ipb' cpb' vp', (Ensembles.In _ pols (pair (pair ipb' cpb') vp')) /\
     ∀ S' st' fl alpha,
    ((com_step p (pair S st) fl alpha (pair S' st')) →
     (∀ (x:id), Ensembles.In src_snk (snds fl) (Var x) → k = eta x) →
     (∀ c vl,
       is_input alpha c vl → ∀ i, 0<=i → i<(length vl) →
             Ensembles.In nat (vp (CV_Chan (Chan c (i + 1)))) (nth i vl 0))
     →
     (∼is_prefixed_stop S' →
      np, k, base, dop, pols, PPhi, eta, X, p   ⊢   (PPhi (fst_lbl S')), S', psi /\
      ((PPhi (fst_lbl S')) st')) /\
     (is_prefixed_stop S' → psi st') /\
     (∀ sts', update_list _ sts k st' empty_state sts' → all_sat_v eta sts' vp') /\
     (sec base ipb ipb' cpb cpb' vp vp' (fl |U| (car_prod (to_ss X) (snds fl)))) /\
     (∀ c vl,
       is_output alpha c vl →
       ∀ i, 0<=i → i<(length vl) →
             Ensembles.In nat (vp' (CV_Chan (Chan c (i + 1)))) (nth i vl 0))
    )
  ).
```

Soundness Proof

```
Proof.
 intros.
 remember psi as post_cond.
 generalize dependent psi.
 well_typed_cases (induction H) Case.
 ...
   Case "TP_IF".
   rename H into H_TP_S1. rename H7 into H_TP_S2.
   intros.
   ∃ ipb. ∃ cpb. ∃ vp.
   split. assumption.
   intros S' st' fl alpha H_If_trans H_mod_lvar H_input_rely.
   assert (~ is_prefixed_stop (l :IF b THEN S1 ELSE S2 FI)) as H_n_pf_stop_IF
     by (intro Contra; inversion Contra).
 ...
Qed.
```

Similar Efforts

Some similar efforts from other people, in the same area, are:

- Daniel Hedin, Andrei Sabelfeld: Information-Flow Security for a Core of JavaScript. CSF 2012.
- Frdric Besson, Nataliia Bielova, Thomas P. Jensen: Hybrid Information Flow Monitoring against Web Tracking. CSF 2013.
- Benot Montagu, Benjamin C. Pierce, and Randy Pollack. A Theory of Information-Flow Labels. CSF 2013.

Conclusion

- Several flaws spotted
- More structural formulation should help with comprehensibility
- More proof automation is desirable
  - Ltac (an untyped tactic definition language)
- Proofs should be made robust
  - "adaptive proof style" advocated by Adam Chlipala
  - "I believe that the best ways to manage significant Coq developments are far from settled." — Adam Chlipala