Formalizing Abstract Interpretation in Coq

Seungcheol Shin

Korea University of Technology and Education

2008 SIGPL Winter School

This Talk is not about

- Framework of Abstract Interpretation
- Theorem Proving
- Type Theory
- Coq the Proof Assistant

Why AI + Proof Assistant

- POPLmark
- Machine checked metatheory for AI
- Certified Program Generation

Contents

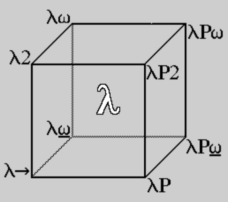
Coq the proof assistant from INRIA

Coq provides features for

- Defining systems (software, math, formal, ...)
- Stating math theorems and software specifications
- Developing interactively formal proofs
  - Checking existing proofs
  - Reusing pre-developed specifications and proofs
- Extracting programs from proofs

Theories behind Coq

- Calculus of (Co)Inductive Constructions(CIC)



- Curry-Howard Isomorphism:

Propositions as types

Proofs as λ-terms

WHILE language syntax

- by hand

$$e ::= n \mid x \mid e{+}e \qquad b ::= e < e$$

$$i ::= \mathsf{skip} \mid x{:=}e \mid i;i \mid \mathsf{while}\ b\ \mathsf{do}\ i\ \mathsf{done}$$

WHILE language syntax

- by hand

$$e ::= n \mid x \mid e{+}e \qquad b ::= e{<}e$$

$$i ::= \text{skip} \mid x{:=}e \mid i;i \mid \text{while } b \text{ do } i \text{ done}$$

- by Coq

Inductive aexpr : Type := avar (s : string) | anum (n : Z) | aplus (a_1 a_2 :aexpr).

Inductive bexpr : Type := blt (a_1 a_2 : aexpr).

Inductive instr : Type := skip | assign (s: string)(e:aexpr) | sequence (i_1 i_2:instr) | while (b:bexpr)(i:instr).

WHILE natural semantics

- by hand

$$\frac{}{\rho \vdash \mathsf{skip} \rightsquigarrow \rho} \qquad \frac{\rho \vdash e \longrightarrow n \qquad \rho \vdash x, n \mapsto \rho'}{\rho \vdash x{:=}e \rightsquigarrow \rho'}$$

$$\frac{\rho \vdash i_1 \rightsquigarrow \rho' \qquad \rho' \vdash i_2 \rightsquigarrow \rho''}{\rho \vdash i_1; i_2 \rightsquigarrow \rho''} \qquad \frac{\rho \vdash b \longrightarrow \mathsf{false}}{\rho \vdash \mathsf{while}\ b\ \mathsf{do}\ i\ \mathsf{done} \rightsquigarrow \rho}$$

$$\frac{\rho \vdash b \longrightarrow \mathsf{true} \quad \rho \vdash i \rightsquigarrow \rho' \quad \rho' \vdash \mathsf{while}\ b\ \mathsf{do}\ i\ \mathsf{done} \rightsquigarrow \rho''}{\rho \vdash \mathsf{while}\ b\ \mathsf{do}\ i\ \mathsf{done} \rightsquigarrow \rho''}$$

WHILE natural semantics

- by hand

$$\frac{}{\rho \vdash \mathsf{skip} \rightsquigarrow \rho} \qquad \frac{\rho \vdash e \longrightarrow n \qquad \rho \vdash x, n \mapsto \rho'}{\rho \vdash x{:=}e \rightsquigarrow \rho'}$$

$$\frac{\rho \vdash i_1 \rightsquigarrow \rho' \qquad \rho' \vdash i_2 \rightsquigarrow \rho''}{\rho \vdash i_1;i_2 \rightsquigarrow \rho''} \qquad \frac{\rho \vdash b \longrightarrow \mathsf{false}}{\rho \vdash \mathsf{while}\ b\ \mathsf{do}\ i\ \mathsf{done} \rightsquigarrow \rho}$$

$$\frac{\rho \vdash b \longrightarrow \mathsf{true} \quad \rho \vdash i \rightsquigarrow \rho' \quad \rho' \vdash \mathsf{while}\ b\ \mathsf{do}\ i\ \mathsf{done} \rightsquigarrow \rho''}{\rho \vdash \mathsf{while}\ b\ \mathsf{do}\ i\ \mathsf{done} \rightsquigarrow \rho''}$$

- by Coq

Inductive exec : env → instr → env → Prop :=
| SN1: ∀ r, exec r skip r
| SN2: ∀ r r' x e v, aeval r e v → s_update r x v r' → exec r (assign x e) r'
| SN3: ∀ r r' r'' i1 i2, exec r i1 r' → exec r' i2 r'' → exec r (sequence i1 i2) r''
| SN4: ∀ r b i, beval r b false → exec r (while b i) r
| SN5: ∀ r r' r'' b i, beval r b true → exec r i r' → exec r' (while b i) r'' → exec r (while b i) r''.

WHILE denotational semantics

- by hand

$$
\begin{array}{l}
DS[\![\mathrm{skip}]\!]\rho = \rho \\
DS[\![\mathrm{x} := \mathrm{e}]\!]\rho = \rho[A[\![e]\!]\rho/x] \\
DS[\![\mathrm{i1; i2}]\!]\rho = DS[\![\mathrm{i2}]\!](DS[\![\mathrm{i1}]\!]\rho) \\
DS[\![\mathrm{while\ e\ do\ i\ done}]\!]\rho = \phi_{b,i}(\rho) \\
\text{ where } \phi_{b,i}(\rho) = \begin{cases} \rho & \text{if } B[\![b]\!]\rho = \text{false} \\ \phi_{b,i}(\rho') & \text{if } B[\![b]\!]\rho = \text{true and } DS[\![\mathrm{i}]\!]\rho = \rho' \\ \perp & \text{otherwise} \end{cases}
\end{array}
$$

WHILE denotational semantics

- by hand

$$
\begin{aligned}
&DS[\![\text{skip}]\!]\rho = \rho\\
&DS[\![\text{x} := \text{e}]\!]\rho = \rho[A[\![e]\!]\rho/x]\\
&DS[\![\text{i1; i2}]\!]\rho = DS[\![\text{i2}]\!](DS[\![\text{i1}]\!]\rho)\\
&DS[\![\text{while e do i done}]\!]\rho = \phi_{b,i}(\rho)
\end{aligned}
$$

$$
\text{where } \phi_{b,i}(\rho) = \begin{cases} \rho & \text{if } B[\![b]\!]\rho = \text{false} \\ \phi_{b,i}(\rho') & \text{if } B[\![b]\!]\rho = \text{true and } DS[\![\text{i}]\!]\rho = \rho' \\ \bot & \text{otherwise} \end{cases}
$$

- by Coq

```
Fixpoint ds(i:instr) : env → option env :=
match i with
skip ⇒ fun r ⇒ Some r
| assign x e ⇒ fun l ⇒ bind (af l e) (fun v ⇒ uf l x v)
| sequence i1 i2 ⇒ fun r ⇒ bind (ds i1 r) (ds i2)
| while e i ⇒ fun l ⇒ phi (fun l' ⇒ bf l' e)(ds i) l
end.
```

System Specifications

- Equivalence
  - by hand
    For every instruction i and environments ρ and ρ',

$$\text{if} \;\; DS[\![i]\!]\rho = \rho' \;\; \text{then} \;\; \rho \vdash \mathrm{i} \rightsquigarrow \rho'$$

System Specifications

- Equivalence
  - by hand
    For every instruction i and environments ρ and ρ',

$$\text{if } \; DS[\![i]\!]\rho = \rho' \;\text{ then }\; \rho \vdash \text{i} \rightsquigarrow \rho'$$

  - by Coq

    Theorem ds_sn : forall i l l', ds i l = Some l' $\rightarrow$ exec l i l'.

System Specifications

- Equivalence
  - by hand
    For every instruction i and environments ρ and ρ',

$$\text{if } \; DS[\![i]\!]\rho = \rho' \; \text{ then } \; \rho \vdash \mathrm{i} \rightsquigarrow \rho'$$

  - by Coq

    Theorem ds_sn : forall i l l', ds i l = Some l' → exec l i l'.

  - Proof in Coq

```
induction i.
intros l l'; simpl; unfold bind. generalize (af_eval l a).
case' (af l a).
intros v He. generalize (uf_s l s v). intros Hu Heq. rewrite Heq in Hu. eauto.
simpl; unfold comp_right; intros l l'. generalize (IHi1 l); case' (ds i1 l).
......
......
simpl; intros l l' Heq; injection Heq; intros; subst; apply SN1.
Qed.
```

Interval Analysis

- Abstract Domain

```
Inductive ext_Z :Type := cZ : Z -> ext_Z
                       | minfty : ext_Z
                       | pinfty : ext_Z.

Definition t := (ext_Z*ext_Z).
```

Interval Analysis

- Abstract Domain

```
Inductive ext_Z :Type := cZ : Z -> ext_Z
                       | minfty : ext_Z
                       | pinfty : ext_Z.

Definition t := (ext_Z*ext_Z).
```

- Abstract Operations

```
Definition plus (i1 i2:ext_Z*ext_Z) : ext_Z*ext_Z :=
  let (l1,u1) := i1 in  let (l2,u2) := i2
  in (comp_add l1 l2, comp_add u1 u2).

Definition join (i1 i2:ext_Z*ext_Z) : ext_Z*ext_Z :=
  let (l1, u1):= i1 in let (l2, u2) := i2
  in (cp_min l1 l2, cp_max u1 u2).
```

Abstract Semantics

```
Fixpoint abstract_i (i:instr)(l:ab_env) {struct i}:a_instr*option ab_env :=
match i with
  skip => (prec (to_a l) a_skip, Some l)
| assign x e =>
    (prec (to_a l)(a_assign x e), Some(ab_update l x (ab_eval (ab_lookup l) e)))
| sequence i1 i2 =>
    let (i'1, l') := abstract_i i1 l in
    match l' with
        None => (a_sequence i'1 (prec false_assert (mark i2)), None)
    | Some l' => let (i'2, l'') := abstract_i i2 l' in (a_sequence i'1 i'2, l'')
    end
| while b i =>
    match intersect_env true l b with
      None =>
       (prec (to_a l)(a_while b (a_conj (a_not (a_b b))(to_a l))(mark i)), Some l)
    | Some l' =>
       let (i',l'') := fp l b i (abstract_i i) in
       match l'' with
          None => (prec (to_a l) (a_while b (to_a l) i'), intersect_env false l b)
        | Some l'' =>
           (prec (to_a l)(a_while b (to_a l'') i'), intersect_env false l'' b)
    end
    end
end.
```

Proving Correctness of AI

- Fixed point theorem

```
Theorem Tarski_least_fixpoint :
            exists phi: A, least_fixpoint f phi.
```

Proving Correctness of AI

- Fixed point theorem

```
Theorem Tarski_least_fixpoint :
            exists phi: A, least_fixpoint f phi.
```

- Soundness of Abstract Interpretation

```
Theorem abstract_i_sound :
            forall i e i' e',
               abstract_i i e = (i', e') ->
                  forall g, i_lc m  g (vcg i' (to_a' e')).
```

Certified Abstract Interpreter

- How to Extract

Certified Abstract Interpreter

- How to Extract

```
Extraction "interp.ml" Zopp denot.denot ax ab.
```

Certified Abstract Interpreter

- How to Extract

```
Extraction "interp.ml" Zopp denot.denot ax ab.
```

- Certified Analyzer

```
let rec abstract_i i l =
  match i with
    | Skip -> Pair ((Prec ((to_a l), A_skip)), (Some l))
    | Assign (x, e) -> Pair ((Prec ((to_a l), (A_assign (x, e)))),
        (Some (ab_update l x (ab_eval (ab_lookup l) e))))
    | Sequence (i1, i2) ->
        let Pair (i'1, l') = abstract_i i1 l in ......
    | While (b, i0) ->
        (match intersect_env True l b with
        ......
```

Starting points

- POPLmark
  (http://alliance.seas.upenn.edu/~plclub/cgi-bin/poplmark/)
- Coq HQ (http://coq.inria.fr/)
- Coq resource wiki (http://logical.futurs.inria.fr/cocorico/)
- Interactive Theorem Proving and Program Development
  Coq'Art: The Calculus of Inductive Constructions,
  Yves Bertot and Pierre Castéran, Springer-Verlag, 2004
- Yves Bertot, Theorem proving support in programming language semantics, RR-6242, INRIA, 2007
- Using Proof Assistants for Programming Language Research or, How to write your next POPL paper in Coq, 2008 POPL tutorial,
  (http://www.cis.upenn.edu/~plclub/popl08-tutorial/)