

VERIFIED EXTRACTION FROM COQ TO A LAMBDA-CALCULUS

COQ WORKSHOP TALK

Yannick Forster and Fabian Kunze

SAARLAND UNIVERSITY
Programming Systems Lab

A VERIFIED SELF INTERPRETER IN THE λ -CALCULUS

Definition $\text{Eva} :=$

$$\begin{aligned} &R (\lambda (\lambda (\lambda \\ &\quad ((0 (\lambda \text{none})) \\ &\quad\quad (\lambda (\lambda \\ &\quad\quad\quad (3 \text{none}) \\ &\quad\quad\quad\quad (\lambda \\ &\quad\quad\quad\quad\quad (((5 0) 2) \\ &\quad\quad\quad\quad\quad\quad (\lambda \\ &\quad\quad\quad\quad\quad\quad\quad (((6 1) 2) \\ &\quad\quad\quad\quad\quad\quad\quad\quad (\lambda \\ &\quad\quad\quad\quad\quad\quad\quad\quad\quad ((1 (\lambda \text{none})) (\lambda (\lambda \text{none})))) (\lambda (8 3) (((\text{Subst } 0) \text{Zero}) 1)))) \\ &\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \text{none})) \text{none})))) (\lambda \text{some } (\text{Lam } 0)))))) \end{aligned}$$

Lemma $\text{Eva_correct } k \ s : \text{Eva } (\text{enc } k) (\text{tenc } s) \equiv \text{oenc } (\text{eva } k \ s).$

Proof.

(including lemmas: 75 lines correctness proof *)*

Qed.

... AND WITH OUR FRAMEWORK

Instance term_eva : internalized eva.

Proof.

internalizeR. revert y0. induction y; intros[]; recStep P; crush.
repeat (destruct _ ; crush).

Defined.

- ▶ Framework to extract Coq terms to λ -terms
- ▶ Semi-automatic verification (only briefly mentioned in this talk)
- ▶ Development of computability theory in this framework

SYNTAX AND SEMANTICS OF OUR λ -CALCULUS

De Bruijn Terms:

$$s, t ::= n \mid s t \mid \lambda s \quad (n \in \mathbb{N})$$

Reduction:

$$\frac{}{(\lambda s)(\lambda t) \succ s_{\lambda t}^0} \quad \frac{s \succ s'}{st \succ s't} \quad \frac{t \succ t'}{st \succ st'}$$

\succ^* denotes the reflexive, transitive closure of \succ .

\equiv the equivalence closure.

BOOLEANS AND NATURAL NUMBERS

SCOTT ENCODING:

$$\overline{true} := \lambda x y.x$$

$$\overline{false} := \lambda x y.y$$

$$\text{if } b \text{ then } s \text{ else } t \implies \overline{b} s t$$

$$\overline{0} := \lambda z s.z$$

$$\overline{Sn} := \lambda z s.s \overline{n}$$

$$\text{match } n \text{ with } 0 \Rightarrow s \mid S n' \Rightarrow t \implies \overline{n} s(\lambda n'.t)$$

EXTRACTION

EXAMPLE: ADDITION

```

fix plus (n m : ℕ) {struct n} : ℕ :=
  match n with
  | 0 ⇒ m
  | S p ⇒ S (plus p m)
  end

```

$$\ulcorner S \urcorner := \lambda n z s. s n$$

$$\ulcorner plus \urcorner := \rho(\lambda A n m. n m (\lambda p. \ulcorner S \urcorner (A p m)))$$

EXISTING EXTRACTIONS

1. Write down a Coq term
2. Prove it to be correct (or use dependent type)
3. Extract to programming language

How to know that the extracted term is correct?
Trust or prove the extraction mechanism!

OUR EXTRACTION

1. Write down a Coq term
2. Prove it to be correct (or use dependent type)
3. Extract to lambda-calculus
4. Use semi-automatic verification to verify the correctness in Coq?

How to know that the extracted term is correct?
It's proven in Coq!

TYPICAL EXTRACTION PROCESS

Preliminaries:

1. Register relevant encoding functions
2. Extract all occurring functions

Automated extraction:

1. Generate an inductive representation from a Coq term
2. Eliminate non-computational parts
3. Extract to L-term
4. Generate correctness statement

Verification:

1. Verify the term semi-automatically

SEEN THIS BEFORE?

Definition $\text{dec } (X : \mathbf{Prop}) : \mathbf{Type} := \{X\} + \{\neg X\}$.

Existing **Class** dec .

Definition $\text{decision } (X : \mathbf{Prop}) (D : \text{dec } X) : \text{dec } X := D$.

Arguments $\text{decision } X \{D\}$.

SEEN THIS BEFORE?

Essentially the same:

```
Typeclass dec (X : Prop) : Type :=mk_dec {  
  decider (X : Prop) : Type :={X} + {¬ X}  
}
```

Definition decision (X : Prop) (D : dec X) : dec X :=decider.
Arguments decision X {D}.

A TYPECLASS FOR ENCODINGS

```
Class registered (X : Type) :=mk_registered
{
  enc_f : X → term ; (* the encoding function for X *)
  proc_enc : ∀ x, proc (enc_f x) (* encodings need to be a procedure *)
}.
Arguments enc_f X {registered} _.
```

REGISTRATION OF BOOL AND NAT

Instance register_bool : registered bool.

Proof.

register bool_enc.

Defined.

Instance register_ℕ : registered ℕ.

Proof.

register ℕ_enc.

Defined.

THE SAME TRICK AGAIN

Definition $\text{enc } (X : \mathbf{Type}) (H:\text{registered } X) : X \rightarrow \text{term} := \text{enc_f } X$.
 Global Arguments $\text{enc } \{X\} \{H\} _ : \text{simpl never}$.

Compute (enc 0, enc false, enc 2).

$((\lambda (\lambda 1)), (\lambda (\lambda 0)), (\lambda (\lambda 0 (\lambda (\lambda 0 (\lambda (\lambda 1)))))))$
 : *term * term * term*

TEMPLATE COQ

“Template Coq is a quoting library for Coq. It takes Coq terms and constructs a representation of their syntax tree as a Coq inductive data type.”

TEMPLATE COQ'S REPRESENTATION

Inductive term : Type :=

- | tRel : $\mathbb{N} \rightarrow \text{term}$
- | tVar : $\text{ident} \rightarrow \text{term}$
- | tMeta : $\mathbb{N} \rightarrow \text{term}$
- | tEvar : $\mathbb{N} \rightarrow \text{term}$
- | tSort : $\text{sort} \rightarrow \text{term}$
- | tCast : $\text{term} \rightarrow \text{cast_kind} \rightarrow \text{term} \rightarrow \text{term}$
- | tProd : $\text{name} \rightarrow \text{term} (** \textit{the type} **) \rightarrow \text{term} \rightarrow \text{term}$
- | tLambda : $\text{name} \rightarrow \text{term} (** \textit{the type} **) \rightarrow \text{term} \rightarrow \text{term}$
- | tLetIn : $\text{name} \rightarrow \text{term} (** \textit{the type} **) \rightarrow \text{term} \rightarrow \text{term} \rightarrow \text{term}$
- | tApp : $\text{term} \rightarrow \text{list term} \rightarrow \text{term}$
- | tConst : $\text{string} \rightarrow \text{term}$
- | tInd : $\text{inductive} \rightarrow \text{term}$
- | tConstruct : $\text{inductive} \rightarrow \mathbb{N} \rightarrow \text{term}$
- | tCase : $\mathbb{N} \rightarrow \text{term} \rightarrow \text{term} \rightarrow \text{list term} \rightarrow \text{term}$
- | tFix : $\text{mfixpoint term} \rightarrow \mathbb{N} \rightarrow \text{term}$
- | tUnknown : $\text{string} \rightarrow \text{term}$.

INTERMEDIATE REPRESENTATION

Inductive $iTerm : Prop :=$

- $iApp : iTerm \rightarrow iTerm \rightarrow iTerm$ (** application of two terms **)
- $iLam : iTerm \rightarrow iTerm$ (** fun **)
- $iFix : iTerm \rightarrow iTerm$ (** fix **)
- $iConst (X:Type) : X \rightarrow iTerm$ (** not unfolded constants **)
- $iMatch : iTerm \rightarrow list\ iTerm \rightarrow iTerm$ (** matches with all the cases **)
- $iVar : \mathbb{N} \rightarrow \mathbb{N} \rightarrow iTerm$ (** variables **)
- $iType : iTerm.$ (** eliminated terms **)

Straightforward/seen in the introduction:

- ▶ fun
- ▶ var
- ▶ app
- ▶ match
- ▶ eliminated terms

FIX

Use function ρ with

$$(\rho u) t \succ^* u (\rho u) t$$

A TYPECLASS FOR INTERNALIZATION

```
Class internalized (X : Type) (x : X) :=  
{ internalizer : term ;  
  proc_t : proc internalizer  
}.
```

Definition int (X : **Type**) (x : X) (H : internalized x) :=internalizer.
Global Arguments int {X} {ty} x {H} : simpl never.

GENERATING CORRECTNESS STATEMENTS

Correctness statement for $\ulcorner plus \urcorner$:

$$\ulcorner plus \urcorner \bar{n} \bar{m} \succ^* \overline{n + m}$$

Correctness statement for $\ulcorner f \urcorner$ with $f : X \rightarrow Y \rightarrow Z$:

$$\ulcorner f \urcorner \bar{x} \bar{y} \succ^* \overline{f x y}$$

Idea: Correctness statement can be generated from the type

THE TT TYPE

An inductive representation for types using HOAS:

Inductive $TT : \mathbf{Type} \rightarrow \mathbf{Type} :=$

 TyB t (H : registered t) : TT t

 | TyElim t : TT t

 | TyAll t (ttt : TT t) (f : t \rightarrow **Type**) (ftt : $\forall x : t, TT (f x)$)
 : TT ($\forall (x:t), f x$).

Arguments TyB _ {_}.

Arguments TyAll {_} _ {_} _.

Notation " $X \rightsquigarrow Y$ " := (TyAll X (fun _ \Rightarrow Y)) (right associativity, at level 70).

EXAMPLE

TT representation for

$\forall x y : \mathbb{N}, \{x = y\} + \{x \neq y\}$ is

```
TyAll (TyB  $\mathbb{N}$ )
  (fun x :  $\mathbb{N} \Rightarrow$ 
    TyAll (TyV  $\mathbb{N}$ ) (fun y :  $\mathbb{N} \Rightarrow$  TyB (  $\{x = y\} + \{x \neq y\}$  ) ) )
  : TT ( $\forall x y : \mathbb{N}, \{x = y\} + \{x \neq y\}$ )
```


GENERATING CORRECTNESS STATEMENTS

Generate statements using a function:

Definition $\text{internalizesF} (p : \text{Lvw.term}) t (ty : \text{TT } t) (f : t) : \text{Prop}$.

revert p. induction ty as [t H p | t H p | t ty internalizesHyp R ftt internalizesF'];

simpl in *; intros.

– exact (p >* enc f).

– exact (p >* I).

– exact ($\forall (y : t) u, \text{proc } u \rightarrow \text{internalizesHyp } y \ u \rightarrow \text{internalizesF}' _ (f \ y) (\text{app } p \ u)$).

Defined.

EXEMPLARY CORRECTNESS STATEMENTS

Correctness statement that t internalizes ...

▶ ... a term $n : \mathbb{N}$: $t >^* \text{enc } n$

▶ ... a term $X : \text{Type}$: $t >^* I$

▶ ... a term $f : X \rightarrow Y$:

$\forall u (x : X), \text{internalizesF } u \ X_x \rightarrow \text{internalizesF } (t \ u) \ Y_ (f \ x)$

INTERNALIZEDCLASS

```
Class internalizedClass (X : Type) (ty : TT X) (x : X) :=  
{  
  internalizer : term ;  
  proc_t : proc internalizer ;  
  correct_t : internalizesF internalizer ty x  
}
```

Definition int (X : **Type**) (ty : TT X) (x : X) (H : internalizedClass ty x) :=internalizer.
Global Arguments int {X} {ty} x {H} : simpl never.

A FINAL HACK

Instance `term_eva` :

`internalizedClass (TyB $\mathbb{N} \rightsquigarrow$ TyB term \rightsquigarrow TyB (option term)) eva.`

Better:

Notation "'internalized' f" :=

`(internalizedClass ltac:(let t :=type of f in let x :=toTT t in exact x) f)`
(at level 100, only parsing).

Instance `term_eva` : `internalized eva.`

COMPUTABILITY THEORY

Formalization	Thesis	Framework
Natural Numbers	110	60
Equality on terms and \mathbb{N}	85	46
Lists	230	113
Substitution and Self Interpretation	209	74
Inverse Encoding of \mathbb{N}	37	9
In Total	777	319

Thanks!

Code:

`ps.uni-saarland.de/~forster/coq-workshop-16/`