Verified Extraction from Coq to a Lambda-Calculus

Research Immersion Lab - Final Talk

Yannick Forster
Joint Work with Fabian Kunze

Advisor: Prof. Dr. Gert Smolka

Saarland University
Programming Systems Lab

SAARLAND UNIVERSITY
COMPUTER SCIENCE

BEFORE

```
Definition Eva :=
  R (λ (λ (λ
    ((0 (λ none))
      (λ (λ
      (3 none)
        (λ
        (((5 0) 2)
          (λ
          (((6 1) 2)
            (λ
            ((1 (λ none)) (λ (λ none))) (λ (8 3) (((Subst 0) Zero) 1))))
          none)) none)))) (λ some (Lam 0)))))

Lemma Eva_correct k s : Eva (enc k) (tenc s) ≡ oenc (eva k s).
Proof.
  (* including lemmas: 75 lines correctness proof *)
Qed.
```

After

```
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 L terms
- Semi-automatic verification (not part of this talk)
- Development of computability theory redone in this framework

The Language

Syntax and Semantics of L

De Bruijn Terms:

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

Reduction:

$$\frac{}{(\lambda s)(\lambda t) \succ s^0_{\lambda t}} \qquad \frac{s \succ s'}{st \succ s't} \qquad \frac{t \succ t'}{st \succ st'}$$

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

[Plotkin, 1975], [Niehren, 1996], [Dal Lago & Martini, 2008]

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Booleans and natural numbers

Scott encoding:

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

$$\textbf{if}\ \mathrm{b}\ \textbf{then}\ \mathrm{s}\ \textbf{else}\ \mathrm{t} \Longrightarrow \overline{b}\ s\ t$$

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

$$\textbf{match}\ \mathrm{n}\ \textbf{with}\ \mathrm{O} \Rightarrow \mathrm{s} \mid \mathrm{S}\ \mathrm{n}' \Rightarrow \mathrm{t} \Longrightarrow \overline{n}\ s(\lambda n'.t)$$

[Curry, Hindley, Seldin, 1972]

VERIFICATION

EXAMPLE: ADDITION

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

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

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

OUTLINE

The framework should be able to:

- Generate and register encoding functions for constructor types
- Generate and register internalizations for Coq functions
- Generate and verify correctness statements

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Overview

1. Register relevant encoding functions
2. Extract all occuring functions
3. Generate an inductive representation from a Coq term
4. Eliminate non-computational parts
5. Extract to L-term
6. Generate correctness statement
7. Verify the term semi-automatically

Encodings

REMEMBER?

Definition dec (X : **Prop**) : **Type** :={X} + {¬ X}.
Existing **Class** dec.

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

REMEMBER?

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 enc (X : **Type**) (H:registered X) : X → term :=enc_f X.
Global Arguments enc {X} {H} _ : simpl never.

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

= ((λ (λ 1)), (λ (λ 0)), (λ (λ O (λ (λ O (λ (λ 1)))))))
: term ∗ term ∗ term

Representation

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."

[Malecha, 2014]

Template Coq's representation

```
Inductive term : Type :=
  | tRel : ℕ → term
  | tVar : ident → term
  | tMeta : ℕ → term
  | tEvar : ℕ → term
  | tSort : sort → term
  | tCast : term → cast_kind → term → term
  | tProd : name → term (** the type **) → term → term
  | tLambda : name → term (** the type **) → term → term
  | tLetIn : name → term (** the type **) → term → term → term
  | tApp : term → list term → term
  | tConst : string → term
  | tInd : inductive → term
  | tConstruct : inductive → ℕ → term
  | tCase : ℕ → term → term → list term → term
  | tFix : mfixpoint term → ℕ → term
  | tUnknown : string → term.
```

Intermediate Representation

```
Inductive iTerm : Prop :=
  iApp : iTerm → iTerm → iTerm (* application of two terms *)
| iLam : iTerm → iTerm (* fun *)
| iFix : iTerm → iTerm (* fix *)
| iConst (X:Type) : X → iTerm (* not unfolded constants *)
| iMatch : iTerm → list iTerm → iTerm (* matches with all the cases *)
| iVar : ℕ→ ℕ→ iTerm (* variables *)
| iType : iTerm. (* eliminated terms *)
```

Internalization

Straightforward / seen in the introduction:

- fun
- var
- app
- match
- eliminated terms

FIX

Use function ρ with

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

Be careful, ρ introduces additional lambdas

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 $\ulcorner plus \urcorner$:

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

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

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

Idea: Correctness statement can be generated from the type

The TT Type

An inductive representation for types using HOAS:

```
Inductive TT : Type → Type :=
  TyB t (H : registered t) : TT t
| TyElim t : TT t
| TyAll t (ttt : TT t) (f : t → Type) (ftt : ∀ x : t, TT (f x))
  : TT (∀  (x:t), f x).

Arguments TyB _ {_}.
Arguments TyAll {_} _ {_} _.

Notation "! X" :=(TyB X) (at level 69).
Notation "X ⇝Y" :=(TyAll X (fun _ ⇒Y)) (right associativity, at level 70).
```

Example

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

$$\begin{aligned}&\text{TyAll}\ (!\ \mathbb{N})\\ &\quad(\textbf{fun}\ x : \mathbb{N} \Rightarrow\\ &\quad\ \text{TyAll}\ (!\ \mathbb{N})\ (\textbf{fun}\ y : \mathbb{N} \Rightarrow !\ \{x = y\} + \{x \neq y\}))\\ &: \text{TT}\ (\forall\ x\ y : \mathbb{N}, \{x = y\} + \{x \neq y\})\end{aligned}$$

SAARLAND UNIVERSITY
COMPUTER SCIENCE

GENERATING CORRECTNESS STATEMENTS

Generate statements using a function:

```
Definition internalizesF (p : Lvw.term) t (ty : TT t) (f : t) : 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 (∀ (y : t) u, proc u → internalizesHyp y u → internalizesF' _ (f y) (app p u)).
Defined.
```

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 (! ℕ⇝! term ⇝! option term) eva.

Better:

Notation "'internalized' f" :=
(internalizedClass $(**let** t :=**type of** f **in let** x :=toTT t **in** exact x)$ f)
(at level 100, only parsing).

Instance term_eva : internalized eva.

In Practice

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
Enumeration of terms	143	26
Inverse Encoding of $\mathbb{N}$	37	9
In Total	777	319