Deep and Shallow Embeddings in Coq

Danil Annenkov  Bas Spitters

Aarhus University, Concordium Blockchain Research Center

TYPES
June 13, 2019
Oslo
Motivation

- We want to reason about functional languages using proof assistants.
- New challenge: smart contract languages.
- But many modern smart contract languages have a functional core.
- We need a convenient and principled way of embedding functional languages into a proof assistant.
Deep embedding VS shallow embedding in proof assistants

Deep embedding:
- AST as an algebraic data type.
- Semantics: big step, small step, definitional interpreter etc.
- Full control over evaluation, features, etc.
- Suitable for **meta-theoretical reasoning**.

Shallow embedding:
- Proof assistants usually come with a built-in functional language (a host language).
- Programming language constructs can be represented using the host language constructs.
- Works better if the languages are similar.
- Convenient for **proving properties of concrete programs**.
Deep embedding AND shallow embedding

We want both!

- AST for a language we want to reason about: for meta-theory.
- Some way of converting AST to functions in Coq.

Ways of converting AST to functions:

- Interpret directly in NbE style ($\text{eval} : \text{Env } \Gamma \rightarrow \text{Expr } \Gamma A \rightarrow A$)
  - $\times$ complicated for the features we want in our language;
  - $\times$ resulting program cab be far from the “natural” representation.
  - $\checkmark$ direct way of proving soundness of the embedding ($\text{eval}$ is a function).

- Use meta-programming approach:
  - $\checkmark$ “naturally”-looking programs;
  - $\checkmark$ flexible in terms of language features;
  - $\times$ proofs of soundness require formalised meta-theory of the host language (we will address this later)
Our approach

- We use meta-programming facilities of MetaCoq.
- Smart Contract AST $\longrightarrow$ MetaCoq AST $\xrightarrow{\text{unquote}}$ Coq function.
- To prove soundness we use formalisation of Coq’s meta-theory in Coq.
Our approach

- We use meta-programming facilities of MetaCoq.
- Smart Contract AST $\rightarrow$ MetaCoq AST $\xrightarrow{\text{unquote}}$ Coq function.
- To prove soundness we use formalisation of Coq’s meta-theory in Coq.

Why not hs-to-coq (or coq-of-ocaml)?

- We want stronger correctness guarantees.
- We want meta-theory to be formalised as well.
- Meta-theory should be “in sync” with the representation in Coq.
MetaCoq project

- Adds metaprogramming facilities to Coq (quote/unquote).
- Implements the kernel of Coq.
- Develops meta-theory of Coq (typing, reduction, etc.)
- Allows for writing Coq plugins within Coq.
- Allows for implementing syntactic translations.
- Allows for proving correctness of plugins, translations, etc.

We will use MetaCoq for embedding of a functional core of a smart contract language.
MetaCoq project

- Adds metaprogramming facilities to Coq (quote/unquote).
- Implements the kernel of Coq.
- Develops meta-theory of Coq (typing, reduction, etc.)
- Allows for writing Coq plugins within Coq.
- Allows for implementing syntactic translations.
- Allows for proving correctness of plugins, translations, etc.

We will use MetaCoq for embedding of a functional core of a smart contract language.
The Oak-light Language

We keep our embedded functional language close to Oak — a smart contract language developed at the Concordium Foundation.

\[
\text{Inductive } \text{expr} : \text{Set} := \\
| \text{eRel} & : \text{nat} \to \text{expr} \\
| \text{eVar} & : \text{name} \to \text{expr} \\
| \text{eLambda} & : \text{name} \to \text{type} \to \text{expr} \to \text{expr} \\
| \text{eLetIn} & : \text{name} \to \text{expr} \to \text{type} \to \text{expr} \to \text{expr} \\
| \text{eApp} & : \text{expr} \to \text{expr} \to \text{expr} \\
| \text{eConstr} & : \text{inductive} \to \text{name} \to \text{expr} \\
| \text{eConst} & : \text{name} \to \text{expr} \\
| \text{eCase} & : (\text{inductive} \times \text{nat}) \to \text{type} \to \text{expr} \to \text{list (pat} \times \text{expr}) \to \text{expr} \\
| \text{eFix} & : \text{name} \to \text{name} \to \text{type} \to \text{type} \to \text{expr} \to \text{expr}. \\
\]
We formalise the semantics of the language in the definitional-interpreter style.

We define our interpreter using a *fuel idiom*: by structural recursion on an additional argument (a natural number).

The interpreter works for both named and nameless representations of terms.

We define a translation of Oak-light to MetaCoq terms.

We want to show that our embedding is sound on terminating programs.
Examples

(* Define a program using Custom Entries for parsing *)
Definition plus_syn : expr :=
  [  fix "plus" (x : Nat) : Nat → Nat :=
      case x : Nat return Nat → Nat of
      |  Z → \y : Nat → y
      |  Suc y → \z : Nat → Suc ("plus" y z) |].

(* Unquoting the translated syntax into a Coq function *)
Make Definition my_plus :=
  Eval compute in (expr_to_term (indexify plus_syn)).

(* Proving correctness by comparing with Coq’s addition on nat *)
Lemma my_plus_correct n m : my_plus n m = n + m.
Proof. induction n; simpl; auto. Qed.

(* Computing with the interpreter *)
Compute (eval 10 [| {plus_syn} 1 1 |]).
Computational soundness: we compare our interpreter with the call-by-value evaluation (CbV) relation of MetaCoq.

The CbV relation is a sub-relation of the reflexive transitive closure of the one-step Coq’s reduction relation.

Complications: closures should be converted to expression by substituting the closed environments, n-ary application of MetaCoq vs unary in our language.
Deep embedding: syntax and (executable) semantics for Oak-light.
Shallow embedding: programs in Gallina language of Coq from the Oak-light syntax.
Computational soundness proof — WIP.
Some small things: customised embedded syntax using Custom Entries notation feature.
Future Work

- Develop more meta-theory of Oak-light.
- Add support for primitives: bounded integers, addresses, hashes, etc.
- Take into account a cost semantics and reasoning about “gas”.
- Integrate with the execution framework for reasoning about inter-contract communication.