ConCert: A Smart Contract Certification Framework in Coq

Danil Annenkov, Jakob Botsch Nielsen and Bas Spitters

Aarhus University, Concordium Blockchain Research Center

CPP2020, January 21, 2020

COBRA
CONCORDIUM BLOCKCHAIN
RESEARCH CENTER AARHUS

AARHUS
UNIVERSITY

What are smart contracts?

- A concept of smart contracts was proposed by Nick Szabo in 90s.
- This is different from smart contracts *on blockchains*.

Fritz Henglein: smart contracts are neither (smart nor contracts).

They are

programs in a general-purpose language running "on a blockchain"

- Check conditions, change global state (account balances) and local state (user-defined contract state).
- Code and calls to contracts *from users* are recorded in blocks.
- Can call other contracts containing possibly malicious code.
- Each node executes the calls and maintains state.
- Contracts are not self-executing: someone has to call.

What is so special about smart contracts?

- They often manage money: auctions, crowdfunding campaigns, multi-signature wallets, DAOs.
- Once deployed, contract code cannot be changed.
- Code is Law.
- Flaws may result in huge financial losses:
  - The DAO $\sim$ \$50M — hacker attack.
  - Parity's multi-signature wallet $\sim$ \$280M — a bug in the library code.

Functional smart contract languages

- Contracts are programs in a functional language transforming the state:

  `contract : CallCtx * Msg * State -> State * Action list`
- But blockchains are stateful.
- Contracts are used as transition functions.
- A *scheduler* handles transfers and calls to other contracts in `Action list`.

Examples of such languages:

- Liquidity, LIGO (Tezos)
- Scilla (Zilliqa)
- **Acorn(Concordium)**

ACORN

- ACORN is a smart contract language for the Concordium blockchain.
- Explicitly typed System F_2, inductive types, general recursion.
- The Concordium blockchain interprets ACORN programs.
- That's what we are going to verify!

Our contributions

- Deep embedding (AST + semantics): for meta-theory.
- Shallow embedding (Coq functions): for convenient reasoning about programs.

- Combine deep and shallow embeddings.
- λ_{smart}: explicitly typed System F + ADT + structural recursion ($\sim$ $\textsc{Acorn}$, or a pure subset of ML-like functional language).
- Shallow embedding of λ_{smart} programs through meta-programming facilities of MetaCoq.
- Soundness using the formalisation of Coq's meta-theory in Coq.
- Integration of the shallow embedding with the execution model (scheduler).

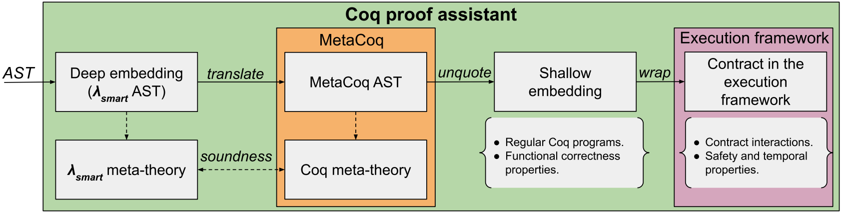
The MetaCoq project

MetaCoq: Metaprogramming in Coq

Consists of several subprojects. Relevant for our project:

- Template Coq — adds meta-programming facilities to Coq:
  - quote: from Coq's definitions to AST as an inductive data type.
  - **unquote**: from AST to back to a Coq definition.
- PCUIC — formalisation of Coq's meta-theory.

ConCert: A Smart Contract Certification Framework



- "translate": a compiler from λ_{smart} (System F + inductives) into PCUIC.
- A bit of λ_{smart} meta-theory: a "fueled" definitional interpreter.

$$\texttt{eval} : \mathbb{N} \to \texttt{global\_env} \to \texttt{env} \to \texttt{expr} \to \texttt{res val}$$

Embedding ACORN code into Coq

Code in ACORN

```
definition foldr a b (f :: a → b → b) (initVal :: b) =
    letrec go (xs :: List a) :: b =
      case xs of
        Nil → initVal
        Cons x xs' → f x (go xs')
     in go
```

A fragment of λ_{smart} AST (deep embedding)

```
Definition Functions := [( "foldr", eTyLam "A" (eTyLam "A" (eLambda "x" (
     tyArr (tyRel 1) (tyArr (tyRel 0) (tyRel 0))) (eLambda "x" (tyRel 0)
      (eLetIn "f" (eFix "rec" "x" ))))));
```

Code in Coq

```
(** Run the translation in the Template Monad (translate and unquote) *)
Run TemplateProgram (translateDefs gEnv Functions).

Print foldr.
(* fun (A A0 : Set)(x : A → A0 → A0) (x0 : A0) ⇒
   fix rec (x1 : List A) : A0 :=
      match x1 with
      | @Nil_coq _ ⇒ x0
      | @Cons_coq _ x2 x3 ⇒ x x2 (rec x3)
      end *)
```

Computational soundness

Translation: $[\![-]\!]^t_\Sigma : \lambda_{\text{smart}} \to \texttt{PCUIC}$

Theorem

For any closed λ_{smart} expression e if $\texttt{eval}^n_{\Sigma,[]}(e) = \texttt{Ok}\ v$, then there is a derivation in MetaCoq's CBV big-step evaluation relation:

$$[\![e]\!]^t_\Sigma \Downarrow [\![\texttt{of\_val}(v)]\!]^t_\Sigma$$

- We support only structurally recursive λ_{smart} programs.
- We rely on correctness of MetaCoq's `unquote`.

Execution model

- Remember the signature
  `contract : CallCtx * Msg * State → State * Action list`?
- `Actions` can be transfers, calls to other contracts (including self calls), contract deployments.
- The execution model formalises the *scheduler*:
  - blockchain state updates (account balances, contract deployments);
  - executing the calls in the `Action list` in *some* order;
  - adding new blocks.
- Provides a reasoning framework on traces — chains of one-step executions.
- Outgoing actions might be arbitrary reordered.
- Implementation and partial correctness of the Congress contract (simplified DAO).

Jakob Botsch Nielsen, Bas Spitters. Smart Contract Interactions in Coq. FMBC'19

Crowdfunding

Crowdfunding: a smart contract allowing arbitrary users to donate money within a deadline.

- Will the users get their money back if the campaign is not funded (goal is not reached)?
- Are all contributions recorded correctly in the contract?
- Does the contract have enough money at the account to cover all contributions?
- ...

```
(* The contract balance "on a blockchain" is consistent the sum of
    individual contributions *)
Corollary cf_donations_backed_after_block {_ : ChainBuilderType}
  prev hd acts new cf_addr lstate :

  builder_add_block prev hd acts = Some new →
(* [cf_contract] - produced from the deep embedding *)
  env_contracts new cf_addr = Some cf_contract  →
  cf_state new cf_addr = Some lstate →
  ∼ lstate.(done_coq) →

  account_balance (env_chain new) cf_addr >=
                  sum_map (lstate.( donations_coq)).
```

Conclusions

- Deep and shallow embeddings in one framework.
- Soundness through the PCUIC formalisation.
- Acorn code verification: parts of the standard library and simple contracts.
- Example: properties of a crowdfunding contract.
- Integration with the execution model.
- Extraction to a functional smart contract language.

ConCert on GitHub:

`https://github.com/AU-COBRA/ConCert`

Future work

- Static semantics of λ_{smart}.
- Gas analysis.
- Connect the development to CertiCoq.