Albert, an intermediate smart-contract language for the Tezos blockchain

Bruno Bernardo, *Raphaël Cauderlier*, Basile Pesin, Julien Tesson

Nomadic Labs

WTSC
February 14, 2020

Tezos

`https://tezos.com`

`https://tezos.gitlab.io`

- written in OCaml (rich static type system)
- liquid proof of stake
- on-chain governance
- formal methods

Michelson: the smart contract language in Tezos

`https://michelson.nomadic-labs.com`

- small stack-based Turing-complete language
- designed with software verification in mind:
  - static typing
  - clear documentation (syntax, typing, semantics)
  - failure is explicit
    - integers do not overflow
    - division returns an option
- implemented using an OCaml GADT
  - subject reduction for free

Michelson example: vote

```
storage (map string int);
parameter string;
code {
      # Check that at least 5tz have been sent
      AMOUNT;
      PUSH mutez 5000000; COMPARE; GT; IF { FAIL } {};

      # Pair and stack manipulation
      DUP; DIP { CDR; DUP }; CAR; DUP;

      DIP { # Get number of votes for chosen option
           GET; IF_NONE { FAIL } {};
           # Increment
           PUSH int 1; ADD; SOME };
      UPDATE; NIL operation; PAIR
    }
```

Mi-Cho-Coq

`https://gitlab.com/nomadic-labs/mi-cho-coq/`

Deep embedding in Coq of the Michelson language

- lexer, parser, macro expander, type checker, evaluator, pretty-printer

Verified smart contracts

- vote example
- default "manager" smart contract
- multisig
  - n persons share the ownership of the contract.
  - they agree on a threshold t (an integer).
  - to do anything with the contract, at least t owners must agree.
  - possible actions:
    - transfer from the multisig contract to somewhere else
    - change the list of owners and the threshold
- spending limit
  - two roles: admin and user
  - user can spend the contract's tokens up-to a stored limit
  - admin can change the limit and authentication keys

High level smart contract languages

Many languages compiled to Michelson:

- Ligo, SmartPy, Fi, Archetype, Morley, Juvix, SCaml, Liquidity, ...

High level smart contract languages

Many languages compiled to Michelson:

- Ligo, SmartPy, Fi, Archetype, Morley, Juvix, SCaml, Liquidity, ...

no certified compiler

The Albert intermediate language

`https://albert-lang.io`

Goals:

- common suffix of most compilers to Michelson
- optimizing
- certified

Choices:

- abstract the stack

The Albert intermediate language

`https://albert-lang.io`

Goals:

- common suffix of most compilers to Michelson
- optimizing
- certified

Choices:

- abstract the stack
- and not much more

Type system

- same types as Michelson + n-ary variants and records
- explicit duplication
- explicit consumption
- implicit ordering

Type system

- same types as Michelson + n-ary variants and records
- explicit duplication
- explicit consumption
- implicit ordering

linear type system

Example: vote in Albert

```
type storage_ty = { threshold : mutez; votes: map string nat }

def vote :
  { param : string ; store : storage_ty } →
  { operations : list operation ; store : storage_ty } =
     {votes = state; threshold = threshold } = store ;
     (state0, state1) = dup state;
     (param0, param1) = dup param;
     prevote_option = state0[param0];
     { res = prevote } = assert_some { opt = prevote_option };
     one = 1; postvote = prevote + one; postvote = Some postvote;
     final_state = update state1 param1 postvote;
     store = {threshold = threshold; votes = final_state};
     operations = ([] : list operation)
```

Example: vote in Albert

```
def guarded_vote :
  { param : string ; store : storage_ty } →
  { operations : list operation ; store : storage_ty } =
    (store0, store1) = dup store;
    threshold = store0.threshold;
    am = amount;
    ok = am >= threshold0;
    match ok with
        False f →failwith "you are so cheap!"
      | True t →drop t;
          voting_parameters = { param = param ; store = store1 };
          vote voting_parameters
    end
```

Ott specification

- syntax, typing, and semantics specified in Ott
- modular specification (one file per language construction)
- from one source
  - OCaml AST
  - Menhir parser
  - Coq AST, typing, and semantic relations
  - LaTeX documentation

Compiler

- compiler written in Coq, certification in progress
- compiler target = Mi-Cho-Coq untyped AST
- proved optimisations at the Michelson level

Compiler pipeline

- inlining of type definitions
- sorting of record labels and variant constructors
- type checking
- function inlining + translation to Michelson

Albert meta theory

Subject reduction:

$$(\Gamma \vdash \mathit{instr} : \mathit{ty} \to \mathit{ty}') \Rightarrow (\Gamma \vdash v : \mathit{ty}) \Rightarrow (E \models \mathit{instr}/v \Rightarrow v') \Rightarrow (\Gamma \vdash v : \mathit{ty}')$$

Progress:

$$(\Gamma \vdash \mathit{instr} : \mathit{ty} \to \mathit{ty}') \Rightarrow (\Gamma \vdash v : \mathit{ty}) \Rightarrow (\exists v', E \models \mathit{instr}/v \Rightarrow v')$$

both proved on a fragment

Conclusion

- The Michelson smart-contract language is formalized in Coq.
- This formalisation can be used to prove interesting Michelson smart-contracts
- and for certified compilation.

Ongoing and Future Work

- prove meta theory
- improve and certify the compiler

Thank you!

Questions?