SmartSpec:
End-to-end specification of Smart Contract and Services

Georges Gonthier, Inria

Why verify end-to-end?

- Smart Contracts are high-value targets
  - > Large liability / small area
  - > Attack incentive / reduced cost
- Verification is a gateway to acceptance
  - > Commercial argument (Tezos, Cardano, …)
  - > Regulatory requirement (Concordium,…)
    - Regulators may step in as soon as verification is feasible (RGPD)
- Smart Contracts are but one (key) component implementing *services*
  - > Asset management, booking, payment/delivery, directory,…
  - > Alongside non-chain code (apps, plugins), third parties (oracles), party obligations (contracts, regulations)
- Contracting parties are realy concerned with end-to-end execution of services
  - > Code-only unit specifications overly technical (e.g., protocol description)

Inria

Technical challenges

- Specifications cover both Smart Contracts and general soft/hard/wet – ware
  - > Need transparent specifications for non-code components
  - > General expressiveness, unhindered by implementation constraints (with Smart Contract subset)
- Reasoning about concurrency and distribution
  - > Handle shared and fragmented state
  - > Distributed actions ⇢ synchronous execution
  - > Temporal modalities
- Reasoning about flows of information, trust, cash, gas, …
  - > Belief modalities (BAN logic), source/auhtority modalities (IF logic)
- Probabilistic correctness
  - > Crypto (sub) protocols, secrecy/privacy
- Economic correctness
  - > Game theory (e.g., auctions, escrow schemes)

Inria

Technical assets

- Scilla
  - > Award-winning msart contract language design, adhering to a minimal transition model
  - > Asynchronous, implementation-oriented execution model
- TLA
  - > Turing award winning abstract specification language with synchronous transitions and time modalities
  - > Based on set theory (less suited to functional and data-oriented programming)
- Coq
  - > Fully general proof system with support for developping and embedding arbitrary theories
- MathComp
  - > Modular Coq theory library for general mathematics and combinatorics (though no crypto/game theory)
- Michelson and its compilers
  - > Execution for code fragments, simulation in general
  - > Leverage existing compilers (LIGO, SmartPy, Morley, …)

Inria

Scilla

From ERC223 sample

```
(*spender: address which owner wants to give access to transfer tokens*)
transition approve (spender : ByStr20, value : Uint128)
sender_map <- allowed[_sender];
match sender_map with
| Some m =>
allowed[_sender][spender] := value;
e = { _eventname : "spenderapproved"; _recipient : _sender; _amount : Uint128 0; message : "spender approved" };
event e
| None =>
allowed[_sender][spender] := value;
e = { _eventname : "spenderapproved"; _recipient : _sender; _amount : Uint128 0; message : "spender approved" };
event e
end
```

TLA

From Dijkstra ring termination sample

```
SendMsg(i) ==
  /\ active[i]
  /\ \E j \in Nodes \ {i}  :
  /\ active' = [active EXCEPT ![j] = TRUE]
  /\ color' = [color EXCEPT ![i] = IF j>i THEN "black" ELSE @]
  /\ UNCHANGED <<tpos, tcolor>>

Deactivate(i) ==
  /\ active[i]
  /\ active' = [active EXCEPT ![i] = FALSE]
  /\ UNCHANGED <<color, tpos, tcolor>>

Environment == \E i \in Nodes : SendMsg(i) \/ Deactivate(i)
```

Workplan

- Language Design
  - > Combine features of Scilla (transition monad), TLA (synchronous transition) with logical assertions/refinements maing use of extended theories.
  - > Select features and semantics to balance expressiveness with formal verification feasibility.
  - > Collaboration (visiting position) with Ilya Serguei (Scilla)
- Evaluation and illustration
  - > Develop examples that test, inform, and showcase the language design decisions.
- Verification support
  - > Embed the language semantics in Coq, using MathComp; embed or compile the syntax
  - > Extend MathComp with the required relevant theories, such as concurrency with communication and synchronisation, information flow and modalities, game semantics, and probabilities and expectations.
- Implementation
  - > Compilation of smart contract code specification to Michelson (via a higher level language).
  - > Simulation of non-code specification fragments.
  - > Compilation of assertions to verification conditions for model and conformance-checking tools.

Merci !

Any questions?