Towards safer smart contract languages

Danil Annenkov

Aarhus University, Concordium Blockchain Research Centre

AU Digital Innovation Festival, April 8, 2019

Smart contracts

- A concept proposed by Nick Szabo in 90s.
- (Wikipedia) A smart contract is a computer protocol intended to digitally facilitate, verify, or enforce the negotiation or performance of a contract.
- Usually thought as self-enforcing, self-executing entities.

Smart contracts

- A concept proposed by Nick Szabo in 90s.
- (Wikipedia) A smart contract is a computer protocol intended to digitally facilitate, verify, or enforce the negotiation or performance of a contract.
- Usually thought as self-enforcing, self-executing entities.

This is not what "smart contracts" on blockchains are!

Smart contracts are neither

At least, currently:

Smart contracts are programs in a general purpose language running "on a blockchain".

Smart contracts are neither

At least, currently:

Smart contracts are programs in a general purpose language running "on a blockchain".

Why neither smart nor contracts?

- Connection to the legal contracts is not clear.
- Smart contracts mix together specification and execution.
- Can go terribly wrong.

Fritz Henglein. Smart contracts are neither.
Cyber Security, Privacy and Blockchain High Tech Summit, DTU, 2017

Smart Contracts: The Evolution

- First generation: Bitcoin script.
- Second generation: Ethereum EVM and Solidity.
- Third generation: functional languages + limited inter-contract communication patterns.

Ethereum and Solidity

- Solidity is a high level java/javascript-like imperative language.
- One of the most widely used smart contract languages.
- Compiles to EVM byte-code.
- Each contract has state, which can be modified during the execution of any of contract's methods.
- Contracts can interact with other contracts by calling methods and sending money.
- Calls can happen in any point of the program execution.

Is Solidity really solid?

Plenty of vulnerabilities have been found:

- Adrian Manning. *Solidity Security: Comprehensive list of known attack vectors and common anti-patterns*
  16 Solidity Hacks/Vulnerabilities
- Luu et al. *Making Smart Contracts Smarter*.
  19366 contracts analysed, 8833 of them have vulnerabilities.
- Ilya Sergey, Aquinas Hobor. *A Concurrent Perspective on Smart Contracts*.
  Multiple issues related to (non-obvious) concurrent behaviour

Towards safer smart contract languages

Why designing safe smart contract languages is crucially important?
At least, because:

- Many smart contract developers with different backgrounds ("coding" is becoming a mass culture).
- Once deployed, contract code cannot be changed.
- Contract execution is irreversible ("Code is Law").
- Flaws in a smart contract may result in huge financial losses (infamous DAO smart contract on Ethereum).

Towards safer smart contract languages

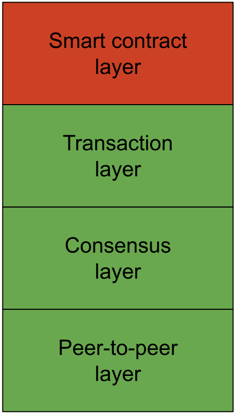
Why designing safe smart contract languages is crucially important?
At least, because:

- Many smart contract developers with different backgrounds ("coding" is becoming a mass culture).
- Once deployed, contract code cannot be changed.
- Contract execution is irreversible ("Code is Law").
- Flaws in a smart contract may result in huge financial losses (infamous DAO smart contract on Ethereum).

Safe languages should make shooting yourself in the foot if not impossible, but at least hard!

The smart contract layer

- Imagine, we have verified all other layers.
- We put some badly designed language on top.
- It's like having your pension depend on a javascript program.
- And any bug is law!



A functional perspective on smart contracts

How we can address the issues? Functional languages to the rescue!

- Based on variants of typed λ-calculi.
- Well-studied **formal semantics**.
- Well-suited for reasoning.
- Proof assistants are based on typed λ-calculi as well!

Semantics matters

Why do we care about formal semantics?

- Meta-theory of a language:
  - type soundness "well-typed programs can't go wrong";
  - termination;
  - compiler correctness;
- Program correctness.

Meta-theory of polymorphic λ-calculus (a.k.a System F) is well developed.

Theoretical foundation of: Haskell, OCaml, Standard ML, F#, ...

Functional core, imperative shell

It's all is good, **but**

- We cannot get rid of stateful computations completely — blockchains are inherently stateful.
- However, we can limit ways of modifying the state.
- Contracts are pure functions transforming the state:

```
contract : state * parameters -> state * operation list
```

Functional core, imperative shell

It's all is good, **but**

- We cannot get rid of stateful computations completely — blockchains are inherently stateful.
- However, we can limit ways of modifying the state.
- Contracts are pure functions transforming the state:
  `contract : state * parameters -> state * operation list`

Examples of languages with the functional "core".

- Simplicity
- Plutus
- **Liquidity**
- Scilla
- **Oak**

DEMO

Towards formal verification

Proof assistants — special software for developing machine-checkable proofs.

- Allows for developing proofs for mathematics and computer science.
- Proofs are developed by interacting with users.
- Proof automation: tactics, decision procedures, SAT/SMT integration.

Towards formal verification

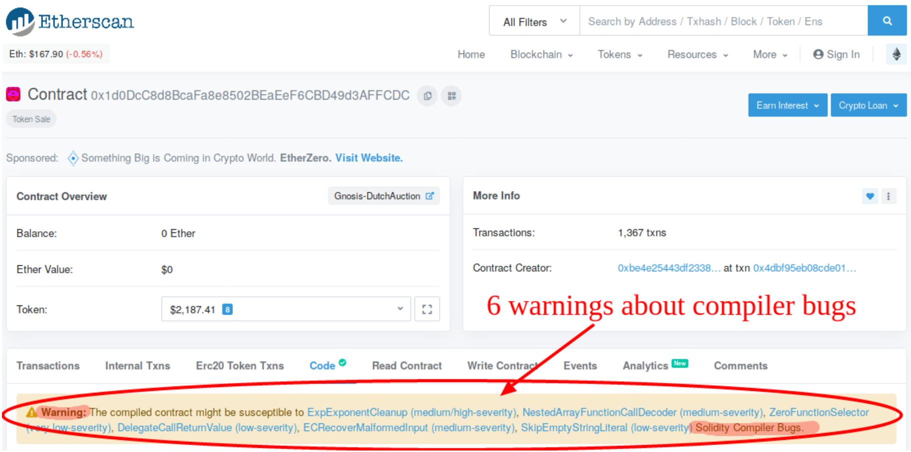
Proof assistants — special software for developing machine-checkable proofs.

- Allows for developing proofs for mathematics and computer science.
- Proofs are developed by interacting with users.
- Proof automation: tactics, decision procedures, SAT/SMT integration.

In particular:

- Formalisation of the programming language's meta-theory.
- Proving correctness of compilers, interpreters, type checking/type inference, etc.
- "Extraction" of bug-free implementation.

Compiler correctness matters



Proof assistants

Proof assistants like Coq, Isabelle/HOL have been successfully applied in large-scale projects

- CompCert — verified C compiler.
- CakeML — verified implemtation of Standard ML.
- seL4 — formal verification of an OS kernel.

Smart contracts formalisation

- Simplicity language — simple language formalised in Coq.
- Ongoing project at *Concordium Research Center*: formalisation of a more expressive smart contract language: the *Oak* language.

Proof assistants

Proof assistants like Coq, Isabelle/HOL have been successfully applied in large-scale projects

- CompCert — verified C compiler.
- CakeML — verified implemtation of Standard ML.
- seL4 — formal verification of an OS kernel.

Smart contracts formalisation

- Simplicity language — simple language formalised in Coq.
- Ongoing project at *Concordium Research Center*: formalisation of a more expressive smart contract language: the *Oak* language.

But formal verification still remains challenging.
Careful design of smart contract languages is a key factor.

> Programming languages semanticists should be the obstetricians of programming languages, not their coroners.
>
> — John C. Reynolds