Simplicity

A New Language for Blockchains

Russell O'Connor

`roconnor@blockstream.com`

Paris Cryptofinance Seminar
January 17, 2018

Blockchains and Smart Contracts

- Crypto-currencies use a blockchain as a distributed ledger.
- Appending transactions to this ledger transfers funds between participants.
- Funds are locked by small programs.
- These programs use cryptography to authorize transactions.
- More complex programs allow for escrow, covenants, digital swaps, et cetera.

Language Design for Blockchains

Why not use JavaScript?

Language Design for Blockchains

Why not use JavaScript?

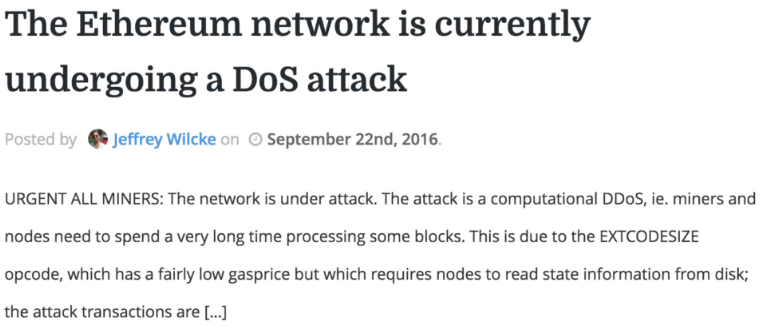
- Ethereum's primary smart-contract language, Solidity, is syntactically similar to JavaScript.
- Solidity is compiled to machine code for an abstract machine called EVM.

Language Design for Blockchains

Why not use JavaScript?

- Ethereum's primary smart-contract language, Solidity, is syntactically similar to JavaScript.
- Solidity is compiled to machine code for an abstract machine called EVM.

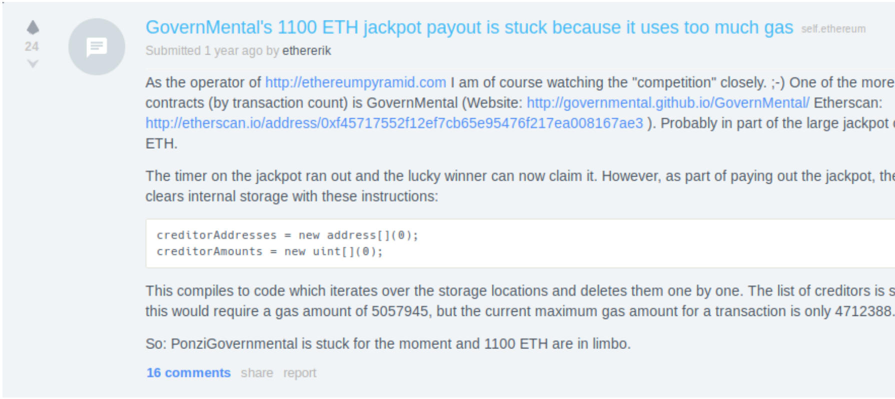
What could go wrong?



The Ethereum network is currently undergoing a DoS attack

Posted by Jeffrey Wilcke on September 22nd, 2016.

URGENT ALL MINERS: The network is under attack. The attack is a computational DDoS, ie. miners and nodes need to spend a very long time processing some blocks. This is due to the EXTCODESIZE opcode, which has a fairly low gasprice but which requires nodes to read state information from disk; the attack transactions are [...]


24

GovernMental's 1100 ETH jackpot payout is stuck because it uses too much gas self.ethereum

Submitted 1 year ago by ethererik

As the operator of http://ethereumpyramid.com I am of course watching the "competition" closely. ;-) One of the more
contracts (by transaction count) is GovernMental (Website: http://governmental.github.io/GovernMental/ Etherscan:
http://etherscan.io/address/0xf45717552f12ef7cb65e95476f217ea008167ae3). Probably in part of the large jackpot
ETH.

The timer on the jackpot ran out and the lucky winner can now claim it. However, as part of paying out the jackpot, the
clears internal storage with these instructions:

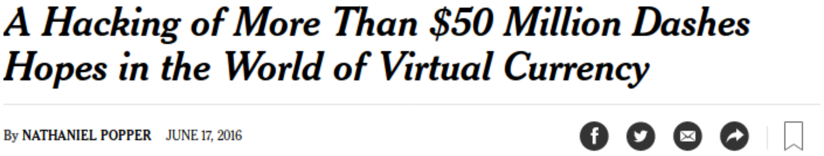
```
creditorAddresses = new address[](0);
creditorAmounts = new uint[](0);
```

This compiles to code which iterates over the storage locations and deletes them one by one. The list of creditors is s
this would require a gas amount of 5057945, but the current maximum gas amount for a transaction is only 4712388.

So: PonziGovernmental is stuck for the moment and 1100 ETH are in limbo.

16 comments share report

The DAO incident



A Hacking of More Than $50 Million Dashes Hopes in the World of Virtual Currency

By NATHANIEL POPPER JUNE 17, 2016

Problems with EVM

- Difficult to assign costs to operations.
- Difficult to bound the costs of programs.
- Complex and informal language semantics makes reasoning difficult.

Problems with EVM

- Difficult to assign costs to operations.
- Difficult to bound the costs of programs.
- Complex and informal language semantics makes reasoning difficult.

But things are better now, right?

$30 million below Parity: Ethereum wallet bug fingered in mass heist

Crypto-cash leak made possible by software stuff-up

By Richard Chirgwin 20 Jul 2017 at 00:55 18 SHARE ▼

Parity's multisig wallet deleted

2017-11-07: TODO

What about Bitcoin?

- Bitcoin's programming language is too inexpressive.
    - e.g. multiplication has been disabled since 2010.

A language design problem

Tony Hoare

There are two methods in software design. One is to make the program so simple, there are obviously no errors. The other is to make it so complicated, there are no obvious errors.

What we need is Simplicity

to create a solid foundation for building smart contracts upon.

What is Simplicity?

Simplicity is low-level language designed for specifying user-defined programs to be evaluated within an adversarial environment.

Simplicity's Features

- Typed combinator language
- Finitarily-complete instead of Turing-complete
- Simple, formal denotational semantics
- Formal operational semantics
- Easy static analysis of computational costs

Type System

- $\mathbb{1}$ - Unit type
- $A + B$ - Sum types
- $A \times B$ - Product types

Expressions

Every Simplicity expression, t, is a function from an input type, A, to an output type, B.

$$t : A \vdash B$$

Simplicity Language

$$\frac{}{\mathrm{iden} : A \vdash A} \qquad \frac{s : A \vdash B \qquad t : B \vdash C}{\mathrm{comp}\ s\ t : A \vdash C}$$

$$\frac{}{\mathrm{unit} : A \vdash \mathbb{1}}$$

$$\frac{t : A \vdash B}{\mathrm{injl}\ t : A \vdash B + C} \qquad \frac{t : A \vdash C}{\mathrm{injr}\ t : A \vdash B + C}$$

$$\frac{s : A \times C \vdash D \qquad t : B \times C \vdash D}{\mathrm{case}\ s\ t : (A + B) \times C \vdash D} \qquad \frac{s : A \vdash B \qquad t : A \vdash C}{\mathrm{pair}\ s\ t : A \vdash B \times C}$$

$$\frac{t : A \vdash C}{\mathrm{take}\ t : A \times B \vdash C} \qquad \frac{t : B \vdash C}{\mathrm{drop}\ t : A \times B \vdash C}$$

Typing rules for the terms of core Simplicity.

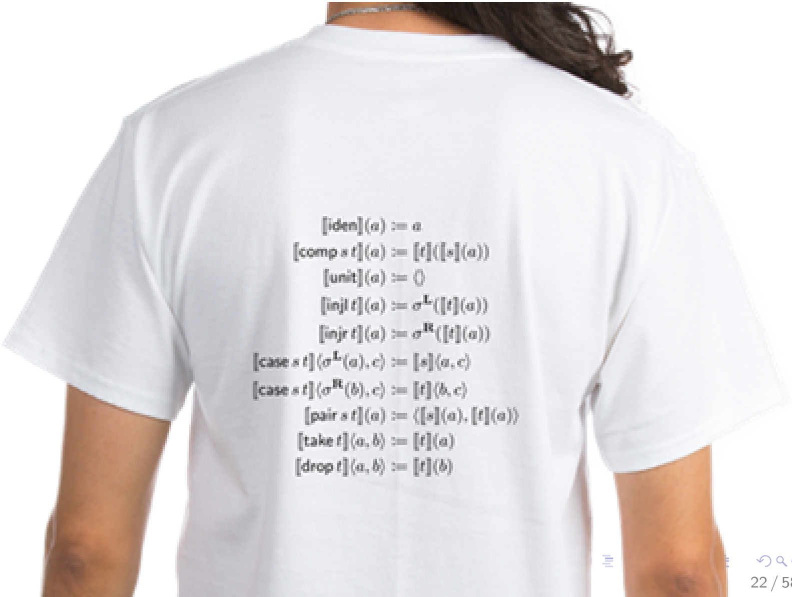
Denotational Semantics

Formally, the language and semantics are defined in the Coq proof assistant. Informally the semantics are as follows:

$$\begin{aligned}
[\![\mathsf{iden}]\!](a) &:= a \\
[\![\mathsf{comp}\, s\, t]\!](a) &:= [\![t]\!]([\![s]\!](a)) \\
[\![\mathsf{unit}]\!](a) &:= \langle\rangle \\
[\![\mathsf{injl}\, t]\!](a) &:= \sigma^{\mathbf{L}}([\![t]\!](a)) \\
[\![\mathsf{injr}\, t]\!](a) &:= \sigma^{\mathbf{R}}([\![t]\!](a)) \\
[\![\mathsf{case}\, s\, t]\!]\langle\sigma^{\mathbf{L}}(a), c\rangle &:= [\![s]\!]\langle a, c\rangle \\
[\![\mathsf{case}\, s\, t]\!]\langle\sigma^{\mathbf{R}}(b), c\rangle &:= [\![t]\!]\langle b, c\rangle \\
[\![\mathsf{pair}\, s\, t]\!](a) &:= \langle[\![s]\!](a), [\![t]\!](a)\rangle \\
[\![\mathsf{take}\, t]\!]\langle a, b\rangle &:= [\![t]\!](a) \\
[\![\mathsf{drop}\, t]\!]\langle a, b\rangle &:= [\![t]\!](b)
\end{aligned}$$

Denotational Semantics

So simple, it fits on a T-shirt.



$$
\begin{aligned}
[\![\mathsf{iden}]\!](a) &:= a \\
[\![\mathsf{comp}\ s\ t]\!](a) &:= [\![t]\!]([\![s]\!](a)) \\
[\![\mathsf{unit}]\!](a) &:= \langle\rangle \\
[\![\mathsf{injl}\ t]\!](a) &:= \sigma^{\mathbf{L}}([\![t]\!](a)) \\
[\![\mathsf{injr}\ t]\!](a) &:= \sigma^{\mathbf{R}}([\![t]\!](a)) \\
[\![\mathsf{case}\ s\ t]\!]\langle\sigma^{\mathbf{L}}(a), c\rangle &:= [\![s]\!]\langle a, c\rangle \\
[\![\mathsf{case}\ s\ t]\!]\langle\sigma^{\mathbf{R}}(b), c\rangle &:= [\![t]\!]\langle b, c\rangle \\
[\![\mathsf{pair}\ s\ t]\!](a) &:= \langle[\![s]\!](a), [\![t]\!](a)\rangle \\
[\![\mathsf{take}\ t]\!]\langle a, b\rangle &:= [\![t]\!](a) \\
[\![\mathsf{drop}\ t]\!]\langle a, b\rangle &:= [\![t]\!](b)
\end{aligned}
$$

Example Expressions

Bits

$$
\begin{aligned}
\mathbb{2} &:= \mathbb{1} + \mathbb{1} \\
\mathbb{2}^2 &:= \mathbb{2} \times \mathbb{2} \\
\mathbb{2}^4 &:= \mathbb{2}^2 \times \mathbb{2}^2 \\
&\vdots
\end{aligned}
$$

Half-Adder

$$\text{half-adder} : 2 \times 2 \vdash 2^2$$

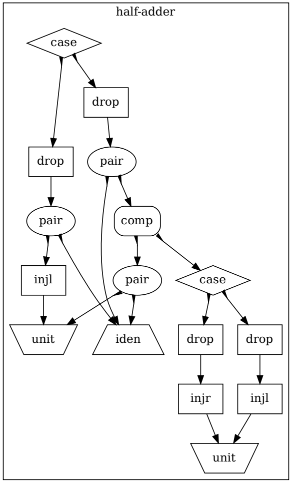
$$\begin{aligned}\text{half-adder} := \text{case}\ &(\text{drop}\,(\text{pair}\,(\text{injl unit})\ \text{iden}))\\ &(\text{drop}\,(\text{pair iden}\,(\text{comp}\ (\text{pair iden unit})\\ &\qquad\qquad\qquad\qquad\quad (\text{case}\,(\text{injr unit})\,(\text{injl unit})))))\end{aligned}$$

SHA-256 Block Compression

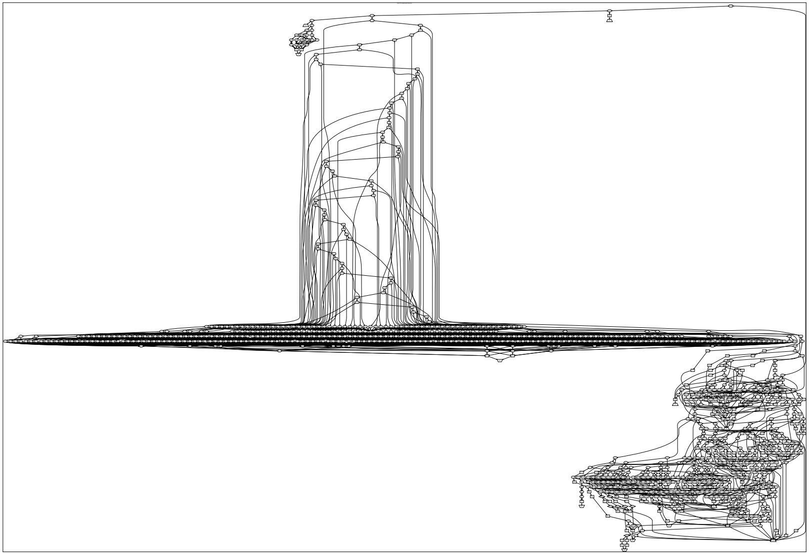
$$\text{sha-256-block} : 2^{256} \times 2^{512} \vdash 2^{256}$$

We use the formal semantics of Simplicity to prove in Coq that our implementation of sha-256-block is correct.

Expression DAGs



Expression DAGs



Simplicity in a Blockchain

Commitment

By recursively hashing the DAG of a Simplicity program, we compute a Merkle root that commits to the program.

Merkle Roots

$$\begin{aligned}
\#(\mathsf{iden}) &:= \mathsf{SHA256Block}\langle \mathsf{tag}_{\mathsf{iden}}, \lfloor 0 \rfloor_{2^{512}} \rangle \\
\#(\mathsf{comp}\ s\ t) &:= \mathsf{SHA256Block}\langle \mathsf{tag}_{\mathsf{comp}}, \langle \#(s), \#(t) \rangle \rangle \\
\#(\mathsf{unit}) &:= \mathsf{SHA256Block}\langle \mathsf{tag}_{\mathsf{unit}}, \lfloor 0 \rfloor_{2^{512}} \rangle \\
\#(\mathsf{injl}\ t) &:= \mathsf{SHA256Block}\langle \mathsf{tag}_{\mathsf{injl}}, \langle \#(t), \lfloor 0 \rfloor_{2^{256}} \rangle \rangle \\
\#(\mathsf{injr}\ t) &:= \mathsf{SHA256Block}\langle \mathsf{tag}_{\mathsf{injr}}, \langle \#(t), \lfloor 0 \rfloor_{2^{256}} \rangle \rangle \\
\#(\mathsf{case}\ s\ t) &:= \mathsf{SHA256Block}\langle \mathsf{tag}_{\mathsf{case}}, \langle \#(s), \#(t) \rangle \rangle \\
\#(\mathsf{pair}\ s\ t) &:= \mathsf{SHA256Block}\langle \mathsf{tag}_{\mathsf{pair}}, \langle \#(s), \#(t) \rangle \rangle \\
\#(\mathsf{take}\ t) &:= \mathsf{SHA256Block}\langle \mathsf{tag}_{\mathsf{take}}, \langle \#(t), \lfloor 0 \rfloor_{2^{256}} \rangle \rangle \\
\#(\mathsf{drop}\ t) &:= \mathsf{SHA256Block}\langle \mathsf{tag}_{\mathsf{drop}}, \langle \#(t), \lfloor 0 \rfloor_{2^{256}} \rangle \rangle
\end{aligned}$$

Witness Values

Simplicity adds a special witness combinator that directly provides input values such as digital signatures.

$$\frac{b : B}{\text{witness}\, b : A \vdash B}$$

These witness values are not committed as part of the commitment Merkle root.

$$\#(\text{witness}\, b) := \text{SHA256Block}\langle \text{tag}_{\text{witness}}, \lfloor 0 \rfloor_{2^{512}} \rangle$$

Redemption

At redemption time, one provides a full Simplicity DAG, including witness values.

- The system checks that the Merkle root matches the commitment.
- The system evaluates the Simplicity program an ensures that it succeeds.

Pruning

During redemption unused branches of the Simplicity program can be pruned.

Types are not committed, they are inferred. Pruning may result is smaller types being inferred and reduce memory usage.

More Features

Full Simplicity is intended to support:

- Signature aggregation
- Covenants
- Delegation

Operational Semantics

Bit Machine

An abstract machine, called the Bit Machine, evaluates Simplicity expressions.

read frame stack	write frame stack
[100$\underline{1}$1??110101000]	[11??1101$\underline{]}$
[$\underline{0}$000]	[111$\underline{?}$?]
[$\underline{]}$	
[$\underline{1}$0]	

Example state for the Bit Machine

Simple instructions for the Bit Machine manipulate the two stacks.

Simplicity Costs

The cost (or weight) of a Simplicity program is determined by

- the size of the programs DAG.
- the number of steps the Bit Machine model takes.
- the amount of memory needed by the Bit Machine model.

This costs can be quickly bounded using static analysis.

Jets

read frame stack	write frame stack
[1001$\underline{1}$??110101000]	[1$\underline{?}$??????]
[$\underline{0}$000]	[111$\underline{?}$?]
[$\underline{\ }$]	
[$\underline{1}$0]	

Evaluating a Simplicity subexpression can only read from a fragment of the active read frame and can only write to a fragment of the active write frame.

Jets

Jets allow the Simplicity interpreter to recognize common subexpressions and preform the computation with C code instead.

Discounted jets allow reduced costs for Simplicity programs that use these jets.

Jets

read frame stack	write frame stack
[1001$\underline{1}$??110101000]	[1011??1$\underline{?}$]
[$\underline{0}$000]	[111$\underline{?}$?]
[$\underline{\ }$]	
[$\underline{1}$0]	

The jet writes the answer directly to the active write frame.

Formal Verification

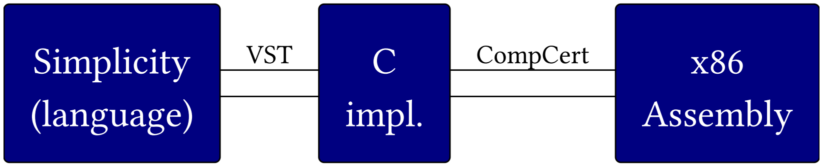
Formal Verification

We want to be able to build smart contracts that

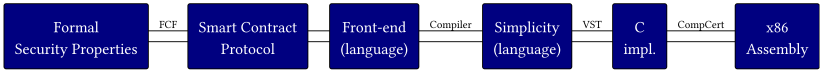
- manage millions of dollars worth of funds.
- are custom tailored.
- may be single purpose.
- run on a public blockchain.

Formal verification is the only conceivable way of achieving the safety required.

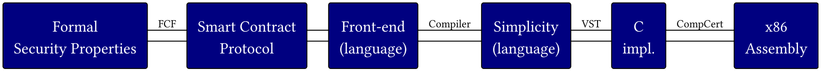
Formal Verification



Formal Verification



Formal Verification



Playing Go Off-chain with Smart Contracts

A Concept Illustration

Playing Go

Both players pay funds into a smart-contract that knows the rules of Go, and the two player's public keys.

Image credit: *By mangpages (Flickr: problem) [CC BY 2.0], via Wikimedia Commons*

Playing Go

Off-chain, players append their moves to a log, digitally signing each move they make.

Image credit: *By mangpages (Flickr: problem) [CC BY 2.0], via Wikimedia Commons*

Playing Go

When cooperating, the player who forfeits signs the funds over to the winner.

Image credit: *By mangpages (Flickr: problem) [CC BY 2.0], via Wikimedia Commons*

Playing Go

When a player cheats, the honest player posts the illegal move, verified by the smart contract, and redeems the funds.

Image credit: *By mangpages (Flickr: problem) [CC BY 2.0], via Wikimedia Commons*

Playing Go

When a player abandons, the honest player posts the last pair of moves to force an on-chain move within a fixed time period.

Image credit: *By mangpages (Flickr: problem) [CC BY 2.0], via Wikimedia Commons*

Playing Go

In all cases at most the last two moves ever need to be posted.

Image credit: *By mangpages (Flickr: problem) [CC BY 2.0], via Wikimedia Commons*

Blockchain as Judiciary

Smart contracts act as a mechanical judiciary to resolve disputes.

Image credit: *By Chris Potter (Flickr: 3D Judges Gavel) [CC BY 2.0], via Wikimedia Commons*

Blockchain as Judiciary

Bonds can be allocated to cover fees.

Image credit: *By Chris Potter (Flickr: 3D Judges Gavel) [CC BY 2.0], via Wikimedia Commons*

Blockchain as Judiciary

Ideally, participants know in advance the judicial outcome, it never needs to be invoked.

Image credit: *By Chris Potter (Flickr: 3D Judges Gavel) [CC BY 2.0], via Wikimedia Commons*

Solutions provided by Simplicity

- Avoids denial of service:
  - The Bit Machine provides a simple model for computational resource costs.
- Avoids running out of gas:
  - Simple static analysis algorithms can bound resource costs of arbitrary programs.
- Avoids hacks:
  - Practical proofs of program correctness made possible by formal semantics in Coq.
- Add privacy:
  - Unused code is pruned before appearing on the Blockchain.

Read more about Simplicity

https://arxiv.org/abs/1711.03028