An exercise in formalisation (and what that gets you): blockchain transactions

Work started at Data61, ATP, Sydney in September 2018 and continued at INESC TEC/HASlab, Minho, Braga in October/November 2018

Steve Reeves

Department of Computer Science
University of Waikato
Hamilton
New Zealand

THE UNIVERSITY OF WAIKATO
Te Whare Wānanga o Waikato

Introduction

- Three aims:

Introduction

- Three aims:
- using Z and PVS to formalise, in very abstract terms, different accounting systems (classical, UTXO...)

Introduction

- Three aims:
- using Z and PVS to formalise, in very abstract terms, different accounting systems (classical, UTXO...)
- using PVS to reproduce work on formalising an abstraction of Ethereum transactions

Introduction

- Three aims:
- using Z and PVS to formalise, in very abstract terms, different accounting systems (classical, UTXO...)
- using PVS to reproduce work on formalising an abstraction of Ethereum transactions
- looking at the connection (if any) between refinement (in general) and theory interpretations (in PVS)

Introduction

- Three aims:
- using Z and PVS to formalise, in very abstract terms, different accounting systems (classical, UTXO...)
- using PVS to reproduce work on formalising an abstraction of Ethereum transactions
- looking at the connection (if any) between refinement (in general) and theory interpretations (in PVS)
- NOTE: we are ignoring the questions of security and how consensus is reached...it turns out that even if all that is perfect, there are currently problems

Aim One—Formalisation

- What general properties should blockchains have? Especially relative to existing accounting systems....

Aim One—Formalisation

- What general properties should blockchains have? Especially relative to existing accounting systems....
- Initially independent from any particular "version"

Aim One—Formalisation

- What general properties should blockchains have? Especially relative to existing accounting systems....
- Initially independent from any particular "version"
- Help manage complexity and provide a coherent view

Aim One—Formalisation

- What general properties should blockchains have? Especially relative to existing accounting systems....
- Initially independent from any particular "version"
- Help manage complexity and provide a coherent view
- Express properties of BC

Aim One—Formalisation

- What general properties should blockchains have? Especially relative to existing accounting systems....
- Initially independent from any particular "version"
- Help manage complexity and provide a coherent view
- Express properties of BC
- Then build models that have those properties

Aim One—Formalisation

- What general properties should blockchains have? Especially relative to existing accounting systems....
- Initially independent from any particular "version"
- Help manage complexity and provide a coherent view
- Express properties of BC
- Then build models that have those properties
- Then, for any particular system, try to show that it is a refinement of the abstract system with known properties

Aim One—Formalisation

- What general properties should blockchains have? Especially relative to existing accounting systems....
- Initially independent from any particular "version"
- Help manage complexity and provide a coherent view
- Express properties of BC
- Then build models that have those properties
- Then, for any particular system, try to show that it is a refinement of the abstract system with known properties
- *Property-driven development*

Refinement

- Express a model abstractly, then move towards a more concrete version (and ultimately a program) in steps which provably preserve correctness relative to the abstract model

Refinement

- Express a model abstractly, then move towards a more concrete version (and ultimately a program) in steps which provably preserve correctness relative to the abstract model
- Principle of Substitutivity

Refinement

- Express a model abstractly, then move towards a more concrete version (and ultimately a program) in steps which provably preserve correctness relative to the abstract model
- Principle of Substitutivity
- Forward simulation rules in Z, for example

$$\forall\, CState' \bullet CInit \Rightarrow \exists\, AState' \bullet AInit \land R'$$

$$\forall\, CState;\ AState \bullet R \land \operatorname{pre} AOp \Rightarrow \operatorname{pre} COp$$

$$\forall\, AState;\ CState;\ CState' \bullet R \land \operatorname{pre} AOp \land COp \Rightarrow \exists\, AState' \bullet AOp \land R'$$

Second Aim—Exploring current BC/DLT systems, with an eye on the future

- Past work has been looking at existing contracts or the EVM

Second Aim—Exploring current BC/DLT systems, with an eye on the future

- Past work has been looking at existing contracts or the EVM
- Aim to (1) reproduce that and (2) expand it to the whole of EtherLite

Second Aim—Exploring current BC/DLT systems, with an eye on the future

- Past work has been looking at existing contracts or the EVM
- Aim to (1) reproduce that and (2) expand it to the whole of EtherLite
- A model of a trivial blockchain in PVS

Second Aim—Exploring current BC/DLT systems, with an eye on the future

- Past work has been looking at existing contracts or the EVM
- Aim to (1) reproduce that and (2) expand it to the whole of EtherLite
- A model of a trivial blockchain in PVS
- Some proofs of simple properties—which guide the model in a modelling/validation cycle

Second Aim—Exploring current BC/DLT systems, with an eye on the future

- Past work has been looking at existing contracts or the EVM
- Aim to (1) reproduce that and (2) expand it to the whole of EtherLite
- A model of a trivial blockchain in PVS
- Some proofs of simple properties—which guide the model in a modelling/validation cycle
- The simplified Etherlite in PVS (Nikolić et al.)

Second Aim—Exploring current BC/DLT systems, with an eye on the future

- Past work has been looking at existing contracts or the EVM
- Aim to (1) reproduce that and (2) expand it to the whole of EtherLite
- A model of a trivial blockchain in PVS
- Some proofs of simple properties—which guide the model in a modelling/validation cycle
- The simplified Etherlite in PVS (Nikolić et al.)
- Full Etherlite in PVS (Luu et al.)

Second Aim—Exploring current BC/DLT systems, with an eye on the future

- Past work has been looking at existing contracts or the EVM
- Aim to (1) reproduce that and (2) expand it to the whole of EtherLite
- A model of a trivial blockchain in PVS
- Some proofs of simple properties—which guide the model in a modelling/validation cycle
- The simplified Etherlite in PVS (Nikolić et al.)
- Full Etherlite in PVS (Luu et al.)
- Denotational rather than the operational semantics of EtherLite

Second Aim—Exploring current BC/DLT systems, with an eye on the future

- Past work has been looking at existing contracts or the EVM
- Aim to (1) reproduce that and (2) expand it to the whole of EtherLite
- A model of a trivial blockchain in PVS
- Some proofs of simple properties—which guide the model in a modelling/validation cycle
- The simplified Etherlite in PVS (Nikolić et al.)
- Full Etherlite in PVS (Luu et al.)
- Denotational rather than the operational semantics of EtherLite
- Try to formulate general properties of BCs from all this experimentation and reproduction

Aim Three—Refinement/Theory Interpretations

Is the connection stated by the PVS guys useful and interesting for me?

Using PVS

- Long pedigree

Using PVS

- Long pedigree
- Functional programming with dependent types, and therefore a proof theory—and therefore all the support that goes with those

Using PVS

- Long pedigree
- Functional programming with dependent types, and therefore a proof theory—and therefore all the support that goes with those
- It means there is a theorem-prover sitting there...which is useful

Using PVS

- Long pedigree
- Functional programming with dependent types, and therefore a proof theory—and therefore all the support that goes with those
- It means there is a theorem-prover sitting there...which is useful
- Some PVS....

Example of what formalisation gives—EtherLite

- Greedy, Prodigal and Suicidal Contracts (Nikolić et al., Singapore, UK) using MAIAN

Example of what formalisation gives—EtherLite

- Greedy, Prodigal and Suicidal Contracts (Nikolić et al., Singapore, UK) using MAIAN

$$
\begin{array}{lll}
\text{Configuration} & \delta & \triangleq \langle A, \sigma \rangle \\
\text{Execution stack} & A & \triangleq \langle M, id, pc, s, m \rangle \cdot A \mid \varepsilon \\
\text{Message} & m & \triangleq \{ sender \mapsto id;\ value : \mathbb{N};\ data \mapsto \ldots \} \\
\text{Blockchain state} & \sigma & \triangleq \overline{id \mapsto \{ bal : \mathbb{N};\ code? \mapsto M;\ \overline{f? \mapsto v} \}}
\end{array}
$$

Example of what formalisation gives—EtherLite

- Greedy, Prodigal and Suicidal Contracts (Nikolić et al., Singapore, UK) using MAIAN

$$
\begin{array}{lrcl}
\text{Configuration} & \delta & \triangleq & \langle A, \sigma \rangle \\
\text{Execution stack} & A & \triangleq & \langle M, id, pc, s, m \rangle \cdot A \mid \varepsilon \\
\text{Message} & m & \triangleq & \{sender \mapsto id;\ value : \mathbb{N};\ data \mapsto \ldots\} \\
\text{Blockchain state} & \sigma & \triangleq & \overline{id \mapsto \left\{bal : \mathbb{N};\ code? \mapsto M;\ \overline{f? \mapsto v}\right\}}
\end{array}
$$

SSTORE

$$\frac{M[pc] = \texttt{SSTORE} \qquad \sigma' = \sigma[id][f \mapsto v]}{\langle \langle M, id, pc, f \cdot v \cdot s, m \rangle \cdot A, \sigma \rangle \xrightarrow{\mathsf{sstore}(f,\ v)} \langle \langle M, id, pc+1, s, m \rangle \cdot A, \sigma' \rangle}$$

SLOAD

$$\frac{M[pc] = \texttt{SLOAD} \qquad v = \sigma[id][f]}{\langle \langle M, id, pc, f \cdot s, m \rangle \cdot A, \sigma \rangle \xrightarrow{\mathsf{sload}(f,\ v)} \langle \langle M, id, pc+1, v \cdot s, m \rangle \cdot A, \sigma \rangle}$$

CALL

$$\frac{\begin{array}{c} M[pc] = \texttt{CALL} \qquad \sigma[id][bal] \geq z \\ s = id' \cdot z \cdot args \cdot s' \qquad a = \langle M, id, pc+1, s', m \rangle \\ m' = \{sender \mapsto id; value \mapsto z; data \mapsto args\} \qquad M' = \sigma[id'][code] \\ \sigma' = \sigma[id][bal \mapsto \sigma[id][bal] - z] \quad \sigma'' = \sigma'[id'][bal \mapsto \sigma'[id'][bal] + z] \end{array}}{\langle \langle M, id, pc, s, m \rangle \cdot A, \sigma \rangle \xrightarrow{\mathsf{call}(id',\ m')} \langle \langle M', id', 0, \varepsilon, m' \rangle \cdot a \cdot A, \sigma'' \rangle}$$

Example of what formalisation gives—EtherLite

- A contract is *prodigal* if it can engage in a sequence of messages which drives the configuration through a trace that sends Ether to an arbitrary sender
- pre-condition P true of initial configuration, side-condition R true of each configuration, post-condition Q is true of final configuration

$$
\begin{array}{lcl}
P(M, \langle \rho, \ell \rangle, m) & \triangleq & m[sender] \notin \mathsf{im}(\rho) \wedge m[value] = 0 \\
R(\langle \rho, \ell \rangle, m) & \triangleq & \mathsf{True} \\
Q(\langle \rho, \ell \rangle, m) & \triangleq & \ell = \mathsf{call}(m[sender], m') \wedge m'[value] > 0
\end{array}
$$

- Similarly formalise *suicidal* and *greedy* contracts
- Scan the Ethereum BC, decompile EVM byte-code, find contracts that have these properties

THE UNIVERSITY OF WAIKATO
Te Whare Wānanga o Waikato

Example of what formalisation gives—EtherLite

- Analysed 970, 898 contracts (from whole BC up to 26th December 2017)
- Used Etherscan to get source for 9,825

Category	**#Candidates flagged** (*distinct*)	Candidates without source	#Validated	% of true positives
Prodigal	**1504** (*438*)	1487	1253	*97*
Suicidal	**1495** (*403*)	1487	1423	*99*
Greedy	**31,201** (*1524*)	31,045	1083	*69*
Total	34,200 (2,365)	34,019	3,759	89

- More than 11,000 Ether trapped or perhaps lost

The general case for formalisation...

- Models that are *simpler* than the thing being modelled, due to abstraction

The general case for formalisation...

- Models that are *simpler* than the thing being modelled, due to abstraction
- Impose as little rigidity as possible

The general case for formalisation...

- Models that are *simpler* than the thing being modelled, due to abstraction
- Impose as little rigidity as possible
- Specialise, make more concrete, less abstract, if required, via refinement...a provable and controlled progression

Conclusions

- Use what gets learned in this reconstruction to express general properties that BC and their languages should have

Conclusions

- Use what gets learned in this reconstruction to express general properties that BC and their languages should have
- The study is a template for designing or even *judging* future BC implementations

Conclusions

- Use what gets learned in this reconstruction to express general properties that BC and their languages should have
- The study is a template for designing or even *judging* future BC implementations
- This then suggests the question: what do we do when "BC goes wrong"?

Conclusions

- Use what gets learned in this reconstruction to express general properties that BC and their languages should have
- The study is a template for designing or even *judging* future BC implementations
- This then suggests the question: what do we do when "BC goes wrong"?
- We will have a way of defining "wrong"
- Also....."We"???? —

Conclusions

- Use what gets learned in this reconstruction to express general properties that BC and their languages should have
- The study is a template for designing or even *judging* future BC implementations
- This then suggests the question: what do we do when "BC goes wrong"?
- We will have a way of defining "wrong"
- Also....."We"???? —
- Governance!

Conclusions

- Use what gets learned in this reconstruction to express general properties that BC and their languages should have
- The study is a template for designing or even *judging* future BC implementations
- This then suggests the question: what do we do when "BC goes wrong"?
- We will have a way of defining "wrong"
- Also....."We"???? —
- Governance!
- This will bite: (restart) and use Aletheia for recording cultural artefacts (tāonga like whakapapa)