Building a Billion Dollar Cryptocurrency with Haskell

Duncan Coutts

November 2018, Code Mesh

INPUT | OUTPUT

Well-Typed

About me (briefly)

Computer scientist. Haskell programmer.

Partner at Well-Typed, Haskell consultancy company.

Head of Engineering for Cardano for IOHK

- PhD in computer science (PLT)
- > 10 years in Haskell consulting
- > 20 years programming

What is Cardano

Public blockchain cryptocurrency

- based on “proof of stake” not “proof of work”
- uses lots of peer-reviewed cryptography research
- just over 1 year old

Roadmap including

- full decentralisation
- two smart contract platforms
- native multi-currency support
- compartmentalised design
- loads more stuff at `https://cardanoroadmap.com/`

Challenges in the cryptocurrency space

Public blockchain systems are challenging to design and implement

- highly adversarial deployment environment
- hard real time constraints
- lots of concurrency
- software flaws can be fatal to the system
- cannot control who runs the code or how
- the internet is not friendly for P2P systems

But lots of opportunities to apply good computer science

Delivery is important

Cryptocurrency space has sky high user expectations

Users want all the promised features

But also want it ASAP

And security, reliability, performance etc

Quality is important

Opportunities for failure are everywhere

- flawed protocol design
- incorrect implementation of the design
- typical software mistakes
- amateur cryptography fails
- missing performance deadlines
- collapse under load
- failure to scale
- DoS failures
- vulnerability to economic attacks
- social or voting collapse
- macroeconomic collapse
- …

Cryptocurrencies are multi-disciplinary projects

You need expertise in

- cryptography
- computer science and formal methods
- programming language technology
- software engineering and system design
- secure system design
- concurrent system design with ‘hard’ real-time deadlines
- network protocols, control theory
- incentives, microeconomics and game theory
- voting systems and governance
- macroeconomics, decentralised monetary and fiscal policy
- …

Gap between research and products

Whole IT industry has a problem with the gap between academic research and practical implementation

IOHK funds substantial original research

Cannot just throw papers over the wall

IOHK have introduced a formal methods and prototyping group

- bridges the cultural and intellectual gap
- pipeline from research to practice
- provides feedback to the research

Balancing quality with delivery

No straightforward trade-off between quality and time to deliver

Poor quality software is slow to test, fix, deploy and change

Fixing bugs or design flaws late is slow and expensive

Formal methods

Cryptocurrencies are a great application for formal methods, but many formal methods are slow

We take a twin track approach

- lightweight methods for balance of quality and pace
- follow-up with formal approaches on longer time scales

Examples

- Cardano wallet: following-up with formalising existing specification in Coq and proving the properties.
- new blockchain protocols: following-up with versions expressed in process calculus, with proofs using Isabelle.
- Plutus Core semantics: formalising properties in Agda

Wither agile?

Is this “big upfront design”?

In a complex design space with difficult trade-offs you need to solve the hard problems first

Can still use iterative design, prototyping and simulation, but do it at a high level of abstraction where making mistakes and trying alternatives is cheaper

Semi-formal software development

Example: the Cardano wallet backend

- a precise specification
- mathematical notation and style
- forces one to think clearly and simplify
- highlights tricky issues
- lemmas! but otherwise semi-formal
  - don’t prove everything
  - test the implementation matches the specification

Leads to dramatically simpler and more robust implementations

Wallet state

$$\begin{aligned}(\mathit{utxo}, \mathit{pending}) &\in \mathsf{Wallet} = \mathsf{UTxO} \times \mathsf{Pending} \\ w_{\varnothing} &\in \mathsf{Wallet} = (\varnothing, \varnothing)\end{aligned}$$

Queries

$$\begin{aligned}\mathsf{availableBalance} &= \mathsf{balance} \circ \mathsf{available} \\ \mathsf{totalBalance} &= \mathsf{balance} \circ \mathsf{total}\end{aligned}$$

Atomic updates

$$\begin{aligned}\mathsf{applyBlock}\ b\ (\mathit{utxo}, \mathit{pending}) &= (\mathsf{updateUTxO}\ b\ \mathit{utxo},\ \mathsf{updatePending}\ b\ \mathit{pending}) \\ \mathsf{newPending}\ \mathit{tx}\ (\mathit{utxo}, \mathit{pending}) &= (\mathit{utxo},\ \mathit{pending} \cup \{\mathit{tx}\})\end{aligned}$$

Preconditions

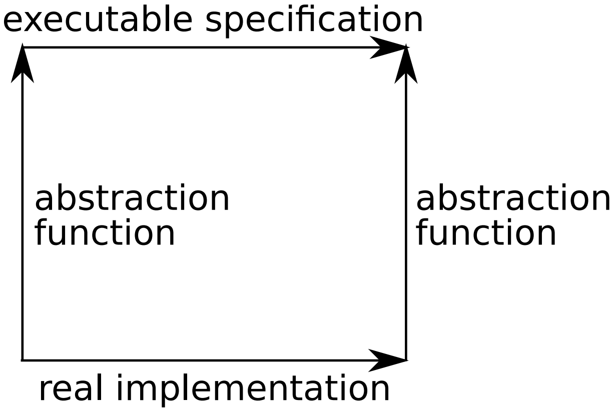
$$\begin{aligned}&\mathsf{newPending}\ (\mathit{ins}, \mathit{outs})\ (\mathit{utxo}, \mathit{pending}) \\ &\qquad \textbf{requires}\ \mathit{ins} \subseteq \mathrm{dom}(\mathsf{available}\ (\mathit{utxo}, \mathit{pending})) \\ &\mathsf{applyBlock}\ b\ (\mathit{utxo}, \mathit{pending}) \\ &\qquad \textbf{requires}\ \ \mathrm{dom}(\mathsf{txouts}\ b) \cap \mathrm{dom}\ \mathit{utxo} = \varnothing\end{aligned}$$

Auxiliary functions

$$\begin{aligned}&\mathsf{available}, \mathsf{total} \in \mathsf{Wallet} \to \mathsf{UTxO} \\ &\mathsf{available}\ (\mathit{utxo}, \mathit{pending}) = \mathsf{txins}\ \mathit{pending} \not\lhd \mathit{utxo} \\ &\mathsf{total}\ (\mathit{utxo}, \mathit{pending}) = \mathsf{available}\ \ (\mathit{utxo}, \mathit{pending}) \cup \mathsf{change}\ \mathit{pending} \\ \\ &\mathsf{change} \in \mathsf{Pending} \to \mathsf{UTxO} \\ &\mathsf{change}\ \mathit{pending} = \mathsf{txouts}\ \mathit{pending} \rhd \mathsf{TxOut}_{\mathsf{ours}} \\ \\ &\mathsf{updateUTxO} \in \mathsf{Block} \to \mathsf{UTxO} \to \mathsf{UTxO} \\ &\mathsf{updateUTxO}\ b\ \mathit{utxo} = \mathsf{txins}\ b \not\lhd (\mathit{utxo} \cup (\mathsf{txouts}\ b \rhd \mathsf{TxOut}_{\mathsf{ours}})) \\ \\ &\mathsf{updatePending} \in \mathsf{Block} \to \mathsf{Pending} \to \mathsf{Pending} \\ &\mathsf{updatePending}\ b\ p = \{\mathit{tx} \mid \mathit{tx} \in p, (\mathit{inputs}, \_) = \mathit{tx}, \mathit{inputs} \cap \mathsf{txins}\ b = \varnothing\}\end{aligned}$$

Figure 3: The basic model

Testing against specifications



Use QuickCheck to test this property

This approach produces great evidence for quality assurance.

Example: new Cardano ledger rules

Ledger rules are being “prototyped” on whiteboards and LaTeX

- write down designs in formal notation
- small size and short review cycles
- allows many brains to be applied

Many issues can be identified and resolved early verification key

signature

$$\frac{\text{verify } vk\ [\![b]\!]\ \sigma \qquad \overbrace{\text{wit}}^{\text{A block has a unique witness}}\ b = (vk\ ,\sigma) \qquad \text{auth}\ \beta\ vk}{\beta \longrightarrow \beta\,;b}$$

vk is authorized to sign a next block given history β

We use Λ for a seq. of txs so maybe we can use β for a sequence of blocks ?

By keeping this relation abstract we can use the same rule for multiple protocols

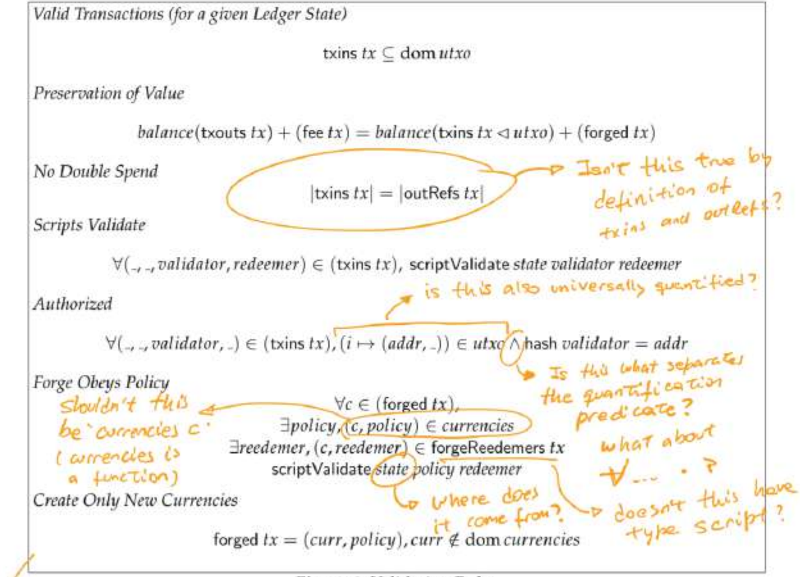
Example: new Cardano ledger rules

Pick the right level of abstraction

- high enough to keep designs small, to fit in brains
- low enough to capture some operational concerns

Operational style for ledger state transition rules

- good guide for implementation
- can analyse space complexity of ledger state
- can analyse time complexity of state transitions



Valid Transactions (for a given Ledger State)

$$\mathsf{txins}\ tx \subseteq \mathsf{dom}\ utxo$$

Preservation of Value

$$balance(\mathsf{txouts}\ tx) + (\mathsf{fee}\ tx) = balance(\mathsf{txins}\ tx \lhd utxo) + (\mathsf{forged}\ tx)$$

No Double Spend

$$|\mathsf{txins}\ tx| = |\mathsf{outRefs}\ tx|$$

Isn't this true by definition of txins and outRefs?

Scripts Validate

$$\forall(\_,\_,validator,redeemer) \in (\mathsf{txins}\ tx),\ \mathsf{scriptValidate}\ state\ validator\ redeemer$$

is this also universally quantified?

Authorized

$$\forall(\_,\_,validator,\_) \in (\mathsf{txins}\ tx), (i \mapsto (addr,\_)) \in utxo \wedge \mathsf{hash}\ validator = addr$$

Is this what separates the quantification predicate?

what about $\forall \ldots .$?

Forge Obeys Policy

$$\begin{gathered}\forall c \in (\mathsf{forged}\ tx),\\ \exists policy, (c, policy) \in currencies\\ \exists reedemer, (c, reedemer) \in \mathsf{forgeReedemers}\ tx\\ \mathsf{scriptValidate}\ state\ policy\ redeemer\end{gathered}$$

Shouldn't this be `currencies c` (currencies is a function)

where does it come from?

doesn't this have type script?

Create Only New Currencies

$$\mathsf{forged}\ tx = (curr, policy), curr \notin \mathsf{dom}\ currencies$$

Figure 4: Validation Rules

Playing to Haskell’s strengths

Haskell is not a silver bullet

You can write terrible code in any language

Key elements to our current approach

- prototype and explore to solve the hard problems first
- pick appropriate maths and CS notations and formalisms
- fanatical devotion to (de)composition, to understand and specify components in relative isolation
- lightweight methods: design, specify and test (but not prove)
- cut corners by doing less, not doing worse

Playing to Haskell's strengths

Haskell makes a CS-heavy approach much easier

"Haskell people" ready to embrace CS-heavy culture

Straightforward translation from mathematical specifications to executable specifications

Facilities for abstraction and composition

Write once for both simulation and execution

Example: new blockchain protocol development

Translation of protocols from papers into Haskell executable specifications

- using an embedded process calculus for precision and simulation
- forces details of plumbing to be set out precisely
- checked with types
- sanity check by running in simulation

Can be quick, and improves understanding

- 2 weeks each to write 2 new protocols
- 1 page descriptions

Example: new network protocol development

Traditional approach starts quite low level

- only runs in IO
- hard to test corner cases

Simulation-first prototyping approach

- iterative design to find low complexity sweet spots
- develop QuickCheck properties in parallel
- concurrency/STM simulation for QC properties
- simulate scenarios that are hard to set up in the real world
- some opportunity to simulate performance

```
class Monad m => MonadFork m where
  fork        :: m () -> m ()

class (MonadFork m, Monad stm) =>
      MonadSTM m stm | m -> stm, stm -> m where
  type TVar m :: * -> *

  atomically  :: stm a -> m a
  newTVar     :: a -> stm (TVar m a)
  readTVar    :: TVar m a -> stm a
  writeTVar   :: TVar m a -> a -> stm ()
  retry       :: stm a

instance MonadFork (SimIO s)
instance MonadSTM  (SimIO s) (SimSTM s)

runSimIO :: forall s. SimIO s () -> ST s Trace
```

Lessons

Lessons

Lots of things worked well

- core system stability has been excellent
- running 24/7 since launch
- globally distributed to survive local failures
- system monitoring gives great insights
- 95% of transactions in the “next block”
- multi-output transactions provide high throughput for exchanges

Lessons

Early stages of a new project are critical

First version of Cardano was developed quickly using traditional methods

- performance requirements need to be clearly understood
- performance design needs to be done early
- distributed concurrency and networking are hard

Corollary: hard problems need more formal approaches

Lessons

New development following new semi-formal approach

- new feature development provides an opportunity to replace components

Example: the wallet backend

- proved not to be good enough for exchanges
- have rewritten the wallet backend from scratch
- have published new specification

Team building and training

Many developers do not have a hard-core CS training

Almost all Haskell devs are eager to learn and improve

Need team buy-in to pursue a spec-heavy approach

Mixed teams also work

Have run training on specification techniques and on testing with QuickCheck

Questions

Extra slides: what’s new in Cardano

Decentralised PoS

First release of Cardano was federated but not decentralised.

Decentralisation in Cardano involves

- mechanisms to delegate stake to stake pools;
- incentives to operate stake pools;
- incentives to delegate to stake pools;
- a robust worldwide P2P network

Delegation and Incentives

Ouroboros was always designed to be decentralised.

To do it in practice has required new research and development

- a research paper on delegation mechanisms
- a research paper on incentives design
- an engineering design for both

This proved remarkably subtle, requiring many design iterations.

Delegation summary

Addresses are associated with stake keys registered on chain.

Stake keys have an associated rewards account.

Stake keys delegate their stake to stake pools.

Stake pools take part in the PoS protocol.

Self staking works by making private stake pools.

Staking rewards are paid out to rewards accounts

- automatically at the end of each epoch
- to stake pool operators
- to stake pool members

Incentives summary

A good incentives scheme requires careful design and analysis using game theory.

The design goals are

- have people operate stake pools, and do so correctly
- have people delegate to stake pools
- have a ‘reasonable’ number of stake pools
- prevent ‘centralisation collapse’ where a few stake pools become dominant

We have a research paper with a design, proofs, and simulations.

The design is based on competition between stake pools to attract delegated stake.

Robust P2P networks are hard

People think P2P networks are easy. They're wrong!

Cryptocurrency P2P networks are especially hard.

- adversaries want to corrupt the system
- adversaries want to waste system resources
- everyone is anonymous or pseudonymous
- home users have firewalls and NAT
- home users have terrible upload speeds
- there are timing requirements
- published attacks on other P2P systems (e.g. Ethereum)
- very little research on designs proved to work

Performance on the internet

Scaling cryptocurrency P2P networks encounter limits.

Time to send a 2MB block from London using TCP/IP

Paris	0.1s
US East coast	1.1s
US West coast	2.5s
Brazil	3.0s
Korea	3.4s
Australia	5.3s

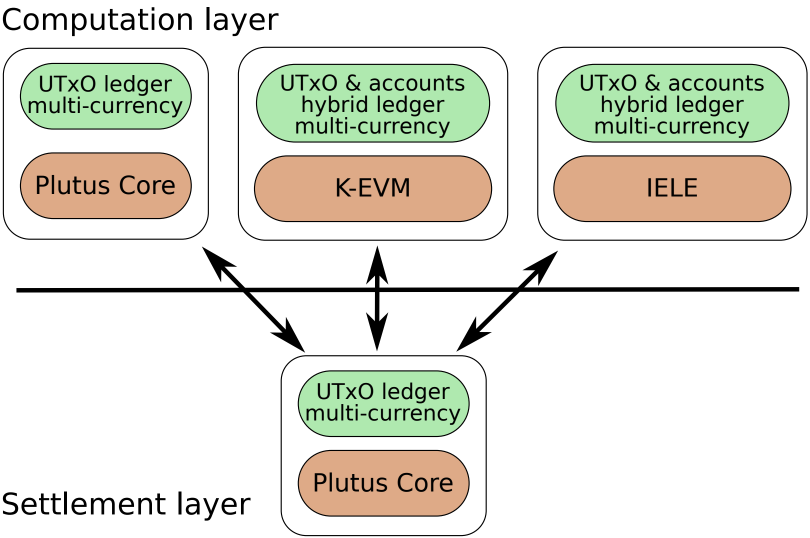
This is not bandwidth constrained.

It is a limitation of TCP and physics.

Our models tell us: roughly 1k – 10k nodes can reliably receive a 2MB block in a 10s time slot.

Beware claims of incredible system performance numbers.

Smart contract platform(s)


Computation layer
UTxO ledger multi-currency
Plutus Core
UTxO & accounts hybrid ledger multi-currency
K-EVM
UTxO & accounts hybrid ledger multi-currency
IELE
UTxO ledger multi-currency
Plutus Core
Settlement layer

Smart contract platform(s)

Two approaches

- covering legacy and traditional: K-EVM & IELE
- more radical: Plutus core, Plutus and Marlowe

Fits with the SL / CL split

- SL stays simple, reliable
- CLs host the more complex program execution environments

Also covers client side code.

A multi-currency ledger

ERC20 provides multi-currency support for Ethereum via smart contracts

Cardano will natively support multiple currencies on the settlement layer (SL).

- a simple extension of UTxO ledger accounting
- safety and simplicity of the SL
- does not require smart contracts for basic functionality
- can be extended by smart contracts
- ERC20 smart contracts on IELE can be backed by native currencies

Hardware wallets

Support for Ledger hardware

- keeps private keys securely in hardware

This will make it easier to support

- mobile clients with hardware key stores;
- exchanges that use off-node transaction signing.

Deposits to prevent economic attacks

An economic attack can happen when costs to system operators are not paid for by fees on system users .

Example: UTxO growth in Bitcoin

- validating nodes need memory (usually RAM) for each unspent transaction output (UTxO entry);
- Bitcoin transaction fees pay for mining but not RAM;
- no user incentive to minimise UTxO usage

Solution: refundable UTxO deposits.

- pay a fee when transactions create more UTxO entries
- negative fee when transactions free UTxO entries
- restore balance between costs and fees

Further into the future

New Ouroboros Genesis implementation

- from-scratch implementation
- incorporating lessons learned
- designed with semi-formal methods from the start
- follow-up with formal methods for even higher assurance
- much clearer understanding and modelling of performance

This will give much better long term flexibility, quality and scaling.