What is an EUTxO blockchain?

LFCS seminar series, Edinburgh Informatics

Murdoch J. Gabbay

23 June 2020

Thanks

Thanks to Ohad Kammar for the invitation to speak.

This talk is based on joint work with Lars Brünjes.

Implementation

What follows has been implemented in Haskell using the Nominal Datatypes Package:

- Code at: `tinyurl.com/nomeutxo`
- Package at: `tinyurl.com/nominaldata`
- Fetch source: `git clone https://github.com/bellissimogiorno/nominal.git`

(Can you run `stack` in `repl.it`? Please tell me how.)

Call these equations idealised EUTxO, and a solution a model or an algebra of chunks

$$
\begin{aligned}
\mathsf{Input} &= \mathbb{A} \times \alpha \\
\mathsf{Validator} &\subseteq \mathit{pow}(\mathsf{Transaction}) \\
\mathsf{Output} &= \mathbb{A} \times \mathsf{Validator} \\
\mathsf{Transaction} &\subseteq [\mathsf{Input}] \times [\mathsf{Output}]
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{Chunk} &\subseteq [\mathsf{Transaction}] \\
\mathsf{Blockchain} &= \{\mathit{ch} \in \mathsf{Chunk} \mid \mathit{utxi}(\mathit{ch}) = \varnothing\}
\end{aligned}
$$

Above, [-] means 'list of -', as per Haskell notation.

You might stare at the 'Blockchain' type, but you should *also* pay attention to the 'Chunk' type. More on that later.

Warning: this figure elides some details. That's OK.

Base types: α and $\mathbb{A}$

$$
\begin{aligned}
\mathsf{Input} &= \mathbb{A} \times \alpha \\
\mathsf{Validator} &\subseteq \mathit{pow}(\mathsf{Transaction}) \\
\mathsf{Output} &= \mathbb{A} \times \mathsf{Validator} \\
\mathsf{Transaction} &\subseteq [\mathsf{Input}] \times [\mathsf{Output}] \\
\mathsf{Chunk} &\subseteq [\mathsf{Transaction}]
\end{aligned}
$$

- α is a base data type for our blockchain. Assuming sufficient Gödel encoding & disregarding efficiency, we could take $\alpha = \mathbb{N}$.
- $\mathbb{A}$ is a countably infinite set of location IDs. Each Input will get a unique ID; as will each Output.
  Atoms are Fraenkel-Mostowski style atoms, as per nominal techniques (relevant to reading the Haskell implementation).

Input, Output, Validator, Transaction

$$
\begin{aligned}
\mathsf{Input} &= \mathbb{A} \times \alpha \\
\mathsf{Validator} &\subseteq \mathit{pow}(\mathsf{Transaction}) \\
\mathsf{Output} &= \mathbb{A} \times \mathsf{Validator} \\
\mathsf{Transaction} &\subseteq [\mathsf{Input}] \times [\mathsf{Output}] \\
\mathsf{Chunk} &\subseteq [\mathsf{Transaction}]
\end{aligned}
$$

An input is an α, located at some $\mathbb{A}$-position.

A output is a validator, located at some $\mathbb{A}$-position.

A validator specifies a set of 'valid transactions'. We don't take the full powerset — e.g. we expect validity to be computable (this also avoids cardinality issues).

A transaction is a list of inputs, and a list of outputs. Also subject to validity constraints (more on this later).

Chunk

$$\begin{aligned} \text{Input} &= \mathbb{A} \times \alpha \\ \text{Output} &\subseteq \mathbb{A} \times \mathit{pow}([\text{Input}] \times [\text{Output}]) \\ \text{Chunk} &\subseteq [\,[\text{Input}]\times[\text{Output}]\,] \end{aligned}$$

A Chunk is a transaction-list, subject to validity constraints:

- Each input in a chunk must have a unique position amongst inputs, likewise for outputs.
- If an output shares a position with an input then:
  - the pair must be unique with that position; and
  - occur in the order output-input, so a later input points to at most one earlier output; and
  - an output pointed to must validate the pointing input's transaction, with that input moved to the list head.

We can α-rename positions of output-input pairs (see paper).

UTxIs and UTxOs

Chunks are similar to a datatype of abstract syntax with binding. They have notions of:

- dangling / free / unspent inputs (those inputs not bound to some earlier output), and
- dangling / free / unspent outputs (those outputs not bound to some later input), and therefore
- α-equivalence on positions of bound output-input pairs.

Call an unspent input a UTxI and an unspent output a UTxO.

A blockchain is a chunk with no free inputs (*left-closed*):

$$\begin{aligned} \mathsf{Chunk} &\subseteq [\mathsf{Transaction}] \\ \mathsf{Blockchain} &= \{ ch \in \mathsf{Chunk} \mid utxi(ch) = \varnothing \} \end{aligned}$$

Chunks have nice properties:

The algebra of chunks

Chunks form a partially-ordered partial monoid.

- The unit is the empty chunk (the chunk consisting of no transactions).
- Composition is list concatenation, subject to validity conditions (no name-clash with positions, no failed validators).
  We may create α-bindings: UTxOs dangling *right* from the *left* chunk may bind to UTxIs dangling *left* from the *right* chunk.
- This partial monoid is partially-ordered by sublist inclusion. It is a fact that validity is preserved by taking sublists.

Operationally as well as mathematically, chunks can be nicer to work with than blockchains.

I discovered this from the Haskell implementation: to be elegant, the code wanted chunks and a partial monoid structure. The maths followed.

In these slides:

- We consider the *structural* aspects of blockchain.
  The challenge of securing these structures cryptographically, is not considered. Yet, at least conceptually, we still provide a clean abstraction to which the crypto side can attach.
- This talk is a mathematical abstraction. Practical implementation is more complex, of course.
  Yet, the type equations and their algebraic structure brings clarity which I (at least) find helpful.
- Even with these elisions, our Haskell implementation demonstrates that this idealisation still has operational content and yields executable code.

What's original?

- The notion of (E)UTxO blockchain is established machinery (Bitcoin; "The extended UTXO model").
- The mathematical idealisation on Slide 4 is new.
- The focus on chunks, their partial monoid structure, and the (minor, but explicit) idea of UTxIs, is new, so far as I am aware.
- Equating names of output-input pairs explicitly with α-equivalence (like in syntax), and applying a nominal model to their operational semantics (cf. the code), is new.

Let's use our maths to express some results

Notation 1. Let variables named ch range over $Chunk$.

Definition 2. Write $fa(ch)$ for the free atoms of $ch \in$ Chunk. Thus, $fa(ch) = utxi(ch) \cup utxo(ch)$.

Definition 3. Write $ch\#ch'$ when $fa(ch) \cap fa(ch') = \varnothing$.

Remark 4. $ch\#ch'$ is a strong orthogonality assertion. The inputs and outputs of ch and ch' cannot connect, and they can't communicate or compete for UTxOs or UTxIs of other chunks.

Let's use our maths to express some results

Notation 5. Write $ch \bullet ch'$ when $ch \cdot ch'$ is defined.

Lemma 6 (simple). $ch \# ch'$ implies $ch \bullet ch' \wedge ch' \bullet ch$.
Sketch proof: If they don't share positions they can't interact: there can't be name-clash between them, and their validators can't fail on one another, because their validators can't be referenced, because they don't know one another's positions.

Lemma 7 (slightly harder). $ch \bullet ch' \wedge ch' \bullet ch$ implies $ch \# ch'$.
Sketch proof: If an input in ch' points to an output in ch then $\neg(ch \bullet ch')$, because this would violate that an input must point to an earlier output position. So they can't share positions.

We don't develop observational equivalence in these slides, but if we did then using Lemmas 6 and 7 we would identify $ch \# ch'$ with commutativity. See also next Theorem:

More results

Theorem 8. Suppose $ch \bullet ch_1$, $ch \bullet ch_2$, and $ch \bullet ch_2 \bullet ch_1$. Then

$$utxi(ch \cdot ch_1) = utxi(ch \cdot ch_2 \cdot ch_1) \Rightarrow ch_1 \# ch_2$$

Technical as this may seem, it is an important purity result.

Consider the special case that UTxIs are $\varnothing$ (so: blockchains), and ch is the current chain. Then *if* we can append ch_1 to the chain now, and we can also append it later (after some ch_2 attaches), then the UTxOs that ch_2 references are necessarily apart from those referenced by inputs of ch_1.

So ch_2 might cause ch_1 to fail to attach to ch, but if it doesn't then it *can't* interact with ch_1; ch_2 might block ch_1 from attaching, but has no effect on ch_1's outcome if successful.

More at `http://gabbay.org.uk/papers.html#utxabs`

A simple example: it counts!

Take $\alpha = \mathbb{N}$. *(An even simpler possiblity is* $\alpha = \{*\}$*, but I want non-trivial validators).*

We will now choose subsets:

$$\begin{aligned}
\text{Input} &= \mathbb{A} \times \mathbb{N} \\
\text{Validator} &\subseteq \mathit{pow}(\text{Transaction}) \\
\text{Output} &= \mathbb{A} \times \text{Validator} \\
\text{Transaction} &\subseteq [\text{Input}] \times [\text{Output}] \\
\text{Chunk} &\subseteq [\text{Transaction}]
\end{aligned}$$

A simple example: it counts!

We admit transactions $\mathrm{Transaction} \subseteq [\mathrm{Input}] \times [\mathrm{Output}]$ of the form

$$succ_{i,p,p'} = \Big(\ \big[(p,i)\big],\ \ \big[(p', val_{p',i'})\big]\ \Big)$$
$$\text{where} \quad val_{p',i'} = ([(p',i')], \_) \longmapsto i' = i{+}1$$

for $i \in \mathbb{N}$ and $p, p' \in \mathbb{A}$ distinct. That's a singleton input $[(p,i)]$ and a singleton output $[(p', val_{p',i'})]$, where *val* validates a transaction iff its input points to p' and carries $i{+}1$. Thus:

$$\mathrm{Transaction} = \Big\{ succ_{i,p,p'} \mid i \in \mathbb{N},\ p{\neq}p' \in \mathbb{A} \Big\}.$$

Admit any chunk, if positions match up and validators are satisfied (in particular, at most one UTxI and UTxO). So:

- Composition is list concatenation; and
- the unit is the empty chunk [].

A simple example: it counts!

Proposition 9. There is a homomorphism of partial monoids from $(\mathrm{Chunk}, [], \cdot)$ to $(\mathbb{N}, 0, +)$, given by mapping a chunk to its length as a transaction-list:

- The empty chunk maps to 0.
- Composition — attaching an n-chunk to an n'-chunk — maps to addition $n + n'$.

Thus, our example counts; each transaction is visibly a 'successor' operation, subject to solving the puzzle of knowing the position of the end UTxO of the left-hand n-chunk, and knowing its final value. That's fine: we expect partiality and this is just part of the 'crypto' aspect of the model.

Note: Proposition 9 holds for any chunk system (not only this example).

A simple example

The previous model doesn't have any blockchains (left-closed chunks), because we did not admit a genesis block (a transaction without inputs).

I don't see this as a problem — what system doesn't allow users to download partial blockchains nowadays? — but it's also easily fixed. Admit zero transactions

$$0_p = ([], [(p,\ ([(p, i)], \_) \mapsto i = 0)])$$

and admit a chunk provided it contains *at most* one zero transaction.

A simple example

Is this a simple example? Yes! But I propose that more complex examples are, mathematically, just fancy-pants versions of this one.

That is not to say that blockchains are simple, nor that we have a accounted for all complexity and extensions; quite the contrary.

But what we have done is simplify complexity, abstract detail, and obtain a clear model to guide us.

Example: restoring some complexity, to see how it's done

In practice, $\text{Validator} \subseteq \mathit{pow}(\text{Transaction})$ is supplied not as a set but as the action of a script — in the literature, validators are taken to *be* scripts, which makes perfect operational sense.

A script is propagated along outputs and should not necessarily change with every transaction. To reflect this in the maths we just require an additional type parameter β:

$$\begin{aligned}
\text{Input} &= \mathbb{A} \times \alpha \\
\text{Validator} &\subseteq \mathit{pow}(\beta \times \text{Transaction}) \\
\text{Output} &= \mathbb{A} \times \beta \times \text{Validator} \\
\text{Transaction} &\subseteq [\text{Input}] \times [\text{Output}] \\
\text{Chunk} &\subseteq [\text{Transaction}]
\end{aligned}$$

The β in Output tells us which part of $v \in \text{Validator}$ (thought of as a script parameterised over a β-value) to reference.

Conclusions

We've seen a simple, abstract presentation of the (E)UTxO model and sketched its properties.

We noted that *chunks* have algebraic properties, and form a partially-ordered partial monoid.

Name-binding corresponds to linking UTxOs to UTxIs and this connection is non-superficial in the sense that notions of support and apartness $\#$ correspond to commutativity and equivalence properties of chunks.

There's more to say, that I haven't covered.

Conclusions

A paper is here:
http://gabbay.org.uk/papers.html#utxabs
with more to follow.

The package is here:
https://github.com/bellissimogiorno/nominal

An example of future work would be to expand on the algebraic treatment of chunks, and hopefully extract a sound and complete axiomatisation of our type equations as an algebraic structure — i.e. a univeral algebra of chunks.

Thanks for listening.