Blockchain Semantics in Agda

Laying the foundations for the formal verification of smart contracts

ORESTIS MELKONIAN, School of Informatics, University of Edinburgh, UK

This report presents the progress made during the third year of my PhD studies, supervised by Prof. Philip Wadler and co-supervised by Dr. Brian Campbell and Prof. Aggelos Kiayias.

1 PROGRESS: 3^{rd} YEAR

This section documents what I have accomplished during the third year of my PhD studies. For the sake of brevity, all formalisations are referenced through blue hypelinks inside the text and/or links to Github repositories, and paper submission/publication info appears in boxes. For reference, I have attached previous yearly reports and relevant papers as separate PDF files.

1.1 BitML

For the 3^{rd} year, my top priority was mostly the BitML formalization. I will not go through the myriads of tedious technicalities I came across and fixed or worked around, but rather outline the most fundamental or conceptually important ones.

Coherence. It turned out that coherence was immensely difficult to fully formalize; the same pattern as last year's continue to manifest, namely using Agda at such a scale where typechecking becomes almost impossible.

I managed to however overcome all such obstacles and have a completely mechanized definition without any postulates or axioms. It is now probably the most complex (as in scaled-up) inductive datatype in a public Agda repository, at least from what I've seen (which might be both a good and a bad thing).

The most involved part that was missing was something hidden in a mere sentence of the paper in natural language:

> "In R^s, we find that C has GC as its ancestor advertisement."

This appear as a hypothesis on some of the inductive cases of coherence, but constructively proving this required non-trivial proofs of temporal hyper-properties, i.e. properties on BitML traces in the operational semantics that involve multiple states at different times and not merely a state invariant or a contiguous sub-trace.

The abstraction issue. With a full definition of coherence finally in place, I can prove things about it by induction. But a thing happened I would never assume possible: simply performing a case analysis on a coherence proof, even if it is a trivial fact, makes the Agda type-checker hang‼ To clarify, it's not that the execution of my program does not terminate, but the type-checking itself. This is because of the complex expressions involved on the type level, which get in the way when Agda is unifying them. Therefore, I found myself in this strange situation where I have to care about the (type-checking) performance of my programs/proofs, just as a developer developer cares about low-level optimizations.

Long story short, I managed to temporarily work around this by marking most of those huge proofs that the definition of coherence contains as abstract, therefore ignoring their implementation details during unification.

Author's address: Orestis Melkonian, Laboratory for Foundations of Computer Science, School of Informatics, University of Edinburgh, Edinburgh, UK.

Lemmas 1 & 6. Moving on, my first attempt was to prove something meaningful about the coherence relation, and the only ones found in the BitML paper are Lemma 1 and Lemma 6, which essentially require a correspondence between names used in BitML contracts and transaction outputs in Bitcoin, all in the context of a symbolic BitML run coherent with a Bitcoin computational run.

Lemma 1 concerns deposit names on BitML, while Lemma 6 deals with contract names. In fact, the statement could be unified in a single lemma, due to the general treatment of names my formulation enjoys.

However, it turned out that it's not possible to prove this for some abstract proofs contained within coherence; these proofs are actually computationally relevant! To clarify with a demonstrative example, some of these proofs help construct the correspondence mappings that coherence relies on, which is exactly what Lemma 1 & 6 try to equate.

I eventually found a way to overcome this by marking only *certain* proofs as abstract, essentially finding a sweet spot between type-checking performance and access to hypotheses for my proofs.

Even after overcoming the abstraction issue, the lemma was suprisingly tricky and time-consuming to prove, so I decided to cover some demonstrative cases (1 trivial, 1 non-trivial, 1 hard) and re-focus my attention to the big picture once again instead of spending even more time on something that is not even clear where it fits in the big picture.

What made this endeavour really hard is the fact that we have to reason about meta-properties of lists, that is properties on how (computationally-relevant) properties of lists interact with one another. For instance, we care about how proofs of membership in a list interact with proofs of list permutation, since we need the index that results from this interaction to perform our computation.

Statement of the final theorem. I then wrote down the statement, or type, of computational correctness, which only required an extra formulation of backtranslation.

From the shape of computational soundness, I saw a clear separation of two individual parts for the first time:

- **Parsing:** Computational soundness states the existence of a (coherent) symbolic run for any possible computational run. This means that we are given a sequence of transaction submissions and messages/broadcasts that contain bitstrings, and we have to reconstruct a meaningful sequence of symbolic steps (both in the operational semantics of BitML and the inductive steps of the coherence relation itself), which might involve many parameters and complicated proofs about those parameters.
  One immediately begins to wonder: how deep down do we want to formalise? Is encoding and decoding bitstrings really that interesting?
  For now, I have formulated a type-safe interface for what encoding/decoding should be and what properties we expect to hold, but I have decided to just postulate them for now and only later fully define them if there is time.
  I have setup the proof infrastructure and proven some easy cases, but there is still hard work to be done involving all the proofs one needs to manufacture in a lot of cases.
- **Conformance to strategies:** This is where **backtranslation** appears, since we need to convert symbolic strategies to computational ones. Note that this inverse direction arises from the fact that strategies are functions from the current run to a set of possible next moves, therefore the direction in which we have to translate appears in a *contravariant* position. Backtranslation boils down to translating symbolic actions to computational ones, where most cases are immediate but there are still a handful that I have not defined.

When backtranslation is defined, one can properly specify that our parsing stage has preserved conformance to the strategies via backtranslation, however it is not entirely clear if this part of the theorem is actually meaningful and useful.

Backup plan. The separation of the *computational soundness* theorem into two manageable chunks also suggests a reasonable fallback scenario in case I run out of time without having the whole proof. It is certainly still significant to be able to produce a coherence symbolic run given any computational run.

1.2 Separation logic for blockchain ledgers

We've been making progress, along with Wouter Swierstra (Utrecht University) and James Chapman (IOG), towards a separation logic for blockchain ledgers.

Our previous model (last year's work) has been iteratively refined to be as elegant, simple, and compositional as possible, altough still restricted to a simple linear bank ledger without any notion of blockchain. The key idea that made this possible is to have a more *fine-grained* notion of separation on *monetary values*, rather than our previous *coarse-grained* separation on sets of *participants*. This allows us, for instance, to modularly reason about a central entity transacting with muliple individuals, even if there is an overlap between involved participants. Compositionality, in this case, arises from the monoidal structure of values being exchanged and the definitions and proofs of the standard Hoare rules for separation logic flow effortlessly from that.

Our next step is to expand this to cover UTxO as well, alas the explicit naming of addresses via hashes breaks the aforementioned compositionality. The main idea I want to pursue to remedy this is defining a more abstract version of UTxO, let's call it *Abstract UTxO (AUTxO)*, which avoids referencing transaction outputs by hash/index and works on sets instead, while employing a value-centric approach. I believe this would allow us to regain compositionality, but I currently only have the ledger definition (i.e. denotational, axiomatic, and operational semantics) and still have to investigate the Hoare logic on top of that and its properties.

1.3 Nominal Agda

After several discussions with Jamie Gabbay on our related work on EUTxO metatheory, I decided to educate myself on *nominal techniques* [**? ?**], and what better way to do this than from one of the co-founders of this theory, Jamie!

This is certainly relevant to both BitML and my EUTxO work, since nominal techiques give a more elegant treatment of names which are heavily used in my BitML formalization. For EUTxO in particular, this allows for a more natural treatment of addresses that enjoys more principled mathematical properties, and this is how Jamie came up with his *Idealised UTxO (IUTxO)* formulation [**? ?**].

The project I have been slowly making progress on is an Agda formalization of nominal sets, as well as a useful framework to work with them in a (constructive) proof setting. The main challenges here involve the gap between the classical/set-theoretic definitions that Jamie explains to me and the computational/type-theoretical formulation I need to represent and reason about.

My ultimate goal is to port the chapter of Phi's PLFA book on the meta-theory of untyped λ-calculus (ULC) which uses de-Bruijn indices, hopefully resulting in a simpler, more elegant way to achieve the same thing.

Let me give a brief overview of what I've covered so far:

- A definition of abstract **atoms** (i.e. names) and the properties they should have. For instance, the type of atoms can sensibly be instantiated to the natural numbers.

- The fundamental notion of **swapping** two atoms within an arbitrary structure that contains atoms/names, as well as a compile-time meta-program that generates a swap action on any inductive datatype along with a proof that it behaves as expected (swapping should always distribute functorially). This is naturally generalized to atom **permutation** actions and the corresponding meta-theory of *G-sets*.
- The notion of **abstractions** over atoms, and correspondingly **concretion**, to a specific atom. This allows me to define the inductive type of λ-terms as without using de-Bruijn indices.
- A definition for the **support** of a structure, which essentially tracks the set of atoms that are *relevant* to the structure. This leads to a notion of **freshness** of a atom w.r.t. to a given structure.
- Now, we have all the ingredients for the crucial `ν` quantifier, that reads as "for all fresh atoms", which we use to define *α-equivalence* of λ-terms and prove that it actually forms an equivalence relation.
- This finally allows us to define *substitution* without having to deal with any capture-avoiding de-Bruijn trickery, which let us define ULC's dynamic semantics / reduction rules.

1.4 AGDA2HS

I have been maintaining AGDA2HS since last year, and striving to get a paper published about it jointly with my collaborators. After two disheartening rejections from CPP and ICFP, we finally got the paper accepted [**?**] at the Haskell Symposium:

> **Reasonable Agda Is Correct Haskell: Writing Verified Haskell using agda2hs**
> *J.Cockx, O.Melkonian, L.Escot, J.Chapman, U.Norell*
> `[agda2hs.pdf]`

1.5 setup-agda

I have not been actively adding new features to the CI tool I had developed for Agda repositories. However, I have stabilized it and released it publicly to the Github marketplace. It is currently being used by 12 different Github projects/repos‼ … but there are all mine :(

1.6 EUTxO (collaboration with IOG)

I have not actively continued researching the metatheory of EUTxO, but rather focused on BitML this year. However, I have participated in different projects with different sets of people within IOG as a more minor figure, although I hope to have been influential nonetheless. These collaborations have resulted in the following 3 short papers in this year's International Workshop on Formal Methods for Blockchains (FMBC), co-located with FLoC 2022:

Formal ledger model (with the Formal Methods team). IOG's formal methods team is currently migrating from an informal LaTeX specification to a mechanized Agda one that compiles to readable LaTeX and executable Haskell. In order to make this work, a crucial automation is to turn relational specifications (as idiomatically used both on paper and dependently-typed theorem proving) to decision procedures/functions that the generated Haskell should execute and production code should be tested against. In Agda, we realise this via compile-time meta-programming, namely a *macro* that given any inductive relation automatically defines the corresponding decision procedure and the proof that it is sound & complete w.r.t. to the relation.

Human and machine-readable models of state machines for the Cardano ledger
A.Knispel, O.Melkonian, J.Chapman, P.Vinogradova
[formal-ledger-model.pdf]

Account simulation in EUTxO (with the Plutus team). The Plutus team is currently investigating how to write co-operating smart contracts that communicate, akin to how a distributed system works, and several challenges and design decisions arise. We are strongly leaning towards a message-passing idiom as most appropriate for general smart contracts. A first demonstrative example is an Ethereum-like account system with the ability to deposit/withdraw funds and transfer funds between participants. We have outlined the design space for such a contract by specifying four different ways to model this in EUTxO using our previous notion of *constraint emitting machines*:

(1) A **naive** *single* state machine that unfortunately has to carry the whole map of accounts in each transaction.
(2) An **optimized** *single* state machine that utilized Merkle tries to efficient communicate account information in much less transaction space.
(3) A **direct** *multi-threaded* version, where every participant runs a separate state machine and is able to transfer funds by synchronizing with another participant/machine and executing the transaction in a single-shot manner.
(4) A **message-passing** *multi-threaded* version, where participants asynchronously transfer funds between them, and communication happens in a mailbox-style message-passing system that we have modelled entirely within the EUTxO model.

Designing EUTxO smart contracts as communicating state machines: the case of simulating accounts
P.Vinogradova, M.Chakravarty, J.Chapman, T.Ferariu M.P.Jones, J.Krijnen
[account-simulations.pdf]

Ledger determinism (with the Formal Methods team). There have been several diffrent notions of *ledger determinism* across the blockchain space, although without precise formal definitions backing them and a mathematical theory to compare them. We are making small steps to remedy this, by defining and comparing the two types of determinism on top of a blockchain-agnostic abstract model of ledgers:

(1) **order-determinism:** permuting the order of submitted transactions does not affect the final state of the ledger
(2) **update-determinism:** two transactions that reach the same ledger state given a valid source state, also behave equivalently on *all* valid source states.

We can give precise definition and furthermore study their relation, which we do using the concept of differentation in the *theory of changes*.

Determinism of ledger updates
P.Vinogradova, A.Knispel, J.Chapman, O.Melkonian
[ledger-determinism.pdf]

1.7 Academic Experience

This year I was invited to review papers and artifacts in different venues, which was undeniably a rewarding experience!

Reviews.

- **TYPES:** reviewed a paper that uses Agda to formally verify Bitcoin smart contracts based on weakest preconditions.

AEC. I was also a member of the following *Artifact Evaluation Committees*:

- **ESOP (European Symposium on Programming)**: reviewed a paper on adding lineary and uniqueness to the Granule programming language and an Agda framework for mechanizing the metatheory of substructural systems generically.
- **ICFP (International Conference on Functional Programming):** reviewed an Agda paper on datatype-generic programming, an Agda formalization of two-level type theory (2LTT) for modelling staging and meta-programming, and a Coq formalization of gradual type theory.

AIM. I am organising the next **Agda Implementors' Meeting (AIM)** in Edinburgh this November; it is a week-long event where Agda practisioners give talks, hack on the Agda compiler or some related tool, form research collaborations (c.f. my involvement in AGDA2HS), and have discussions about language design decisions.

2 TIMEPLAN: 4^{th} YEAR

Let me finally outline my plans for the final year of my PhD studies, aimed at producing a thesis disseration at the end of the year.

2.1 OCT'22 - MAR'23 | Closure

For the first half of my 4^{th} year I would like to reach as much closure as possible to each of my projects. Concretely:

- My main priority will be the BitML formalisation, namely completing at least the definition of parsing computational runs to symbolic ones, which constitutes a significant individual part of the final theorem of *computational soundness*, which is orthogonal to the game-theoretic concerns about participant strategies.
- Extend our separation logic from simple linear ledgers to UTxO-based blockchain ones and publish a paper about it. As a backup plan, in case I find no adequate extension that retains all nice mathematical properties of our simple model, try to publish a paper on just the simple fragment to urge the community to start thinking in those terms.
- Slowly add new features to nominal Agda and have a full formalization of the PLFA chapter on ULC. If time permits, investigate how my existing formalizations can be re-formulated in this framework and then conclude if this was worth it.
- This is slightly more far-fetched, but it would be nice to use AGDA2HS on my own formalizations and generate executable Haskell code (e.g. a BitML compiler to Bitcoin transactions, a CEM state machine library for Plutus, etc...). However, the transition from arbitrary dependent types that my work employs to a simpler fragment that systematically uses erasure is bound to be painful and might even require tweaking the Agda type-checker itself.

Hopefully, these attempts will result in individual publications for each topic. Needless to say, if one of those happen to be at a top-tier venue such as POPL, ICFP or CPP, I would be immensely satified.

2.2 MAR'23 - AUG'23 | Thesis write-up

For the second half of the year I am planning to write down my thesis, reflecting on the various challenges I faced during the past 3.5 years and highlighting the novel ideas that helped me overcome them.

I believe I have enough work to write a disseration adequate for a PhD, which would probably consist of an amalgam of my publications during all this period.