An Ontology for Smart Contracts

Darryl McAdams
Input Output Hong Kong
Email: darryl.mcadams@iohk.io

I. Introduction

This paper introduces a basic ontology that attempts to capture the essential features of many smart contracts, in order to aid in formal reasoning about their behavior. Section II gives a general overview of the proposed ontology, and Section III gives analyses of a number of interesting smart contracts from the literature, through the lens of this ontology. The proposals here are not intended to be the One True Ontology, but rather a useful ontology. A well-designed blockchain should be able to support arbitrary such ontologies.

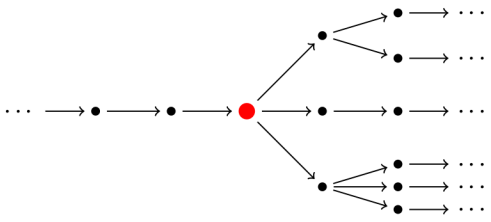

Fig. 1. Relationships between states. The present is larger and red. Arrows represent events that transition one state to another, creating a notion of time.

II. The Ontology

An ontology is a collection of concepts (or kinds, types) which are seen as the most fundamental way of viewing a problem domain, and the relationships that hold among them. In the context of particle physics, the ontology is the Standard Model of particles and the various associated fields.

In the context of smart contracts, a number of concepts arise over and over, which constitute this domain's ontology. This paper assumes that smart contracts fundamentally are stateful computations consisting of some state value which changes over time, and a fixed, unchanging transition function. Precisely what constitutes a state, and how many states there are, is entirely dependent on the smart contract, and might be as simple as a token representing a state in a finite state machine (e.g. q_3) or it might be some richer piece of symbolic data.

The ontology provides concepts for thinking about such contracts, abstracted over the details that are contract-specific or orthogonal to the macroscopic behavior, and are as follows:

Agent An Agent is a participant in a contract, whether that means a person, an organization, a piece of software, or something else. Agents take actions which affect the state of a contract.

Event An Event is a transition from one state to another. Events are brought about by Agents invoking the transition function on the most recent state.

Object An Object is something that is manipulated by a smart contract and may be transferred between Agents. Who can manipulate it and how is determined by the contract. Some Objects might have constraints on usage, such as who they can be transferred to (e.g. a "gift card" that can only be spent at certain retailers).

Time Time is the extrinsic notion of time, which passes at rates independent of the block chain. It is distinct from the internal before-after ordering that blockchains exist to creat. Time is calendar time.

Modality Modalities are higher-order phenomena that describe the relationships between states in the totality of what is logically possible. Modalities quantify over past or future states in relation to a given current state, and allow concepts of possibility, necessity, and so on. Modalities come in two main flavors:

Vertical A Vertical Modality quantifies over the alternative futures that are compatible with the given present state. Concepts like "must" and "may" are fundamentally vertical, because they describe only the existence, or not, of events of some kind. Something "must" occur if every future has that thing occurring, while it "may" occur if at least one future has it occurring.

Horizontal A Horizontal Modality, which may also be called a Temporal Modality, quantifies over sequences of states. Concepts like "eventually" and "never" are fundamentally horizontal because they describe the internal structure of alternative futures. Something "eventually" occurs if at some future, it occurs, but maybe only after many events have taken place. Something "never" occurs if there is no number of events after which it occurs.

The terminology "Vertical" and "Horizontal" come from visualization. We may imagine that a sequence of state transitions is like a time line laid out left to right, as in Figure 1. At any given state, there is a past extending to the left, and a variety of futures extending to the right. Each future is geometrically related to that state by being horizontally offset from it, but is vertically offset relative to other futures of that state. It may even be the case that a state has multiple pasts, as there may be multiple ways to have come to that state.

Some other things which are related to the ontology, but which are considered more fundamental in this paper, are notions like information, knowledge, identity, secrets, and verification. These are all related notions involving information theory and cryptographic protocols, and reasoning about them is extremely difficult. There is also a substantial literature on that subject already, which can provide a better set of tools than this paper could.

III. Analyses of Smart Contracts

This section analyzes a number of smart contracts from the literature. This is not intended to be a comprehensive overview of the full range of ways to apply this ontology, nor of smart contracts. Rather, it is intended to give some intuitions for how the ontology is applied, and how it can be used to describe properties that lead to bugs when they fail to hold.

A. Simple Transfer

In a Simple Transfer, a participant, the Sender, Transfers control of an Item, often some form of money, to another participant, the Recipient, who is known before the sending. In the usual construal of Transfers, this involves the Sender granting to the Recipient the ability to further do with the item as they please.

Using the ontology, we can identify the Sender and Recipient as Agents, the Item is an Object, and the Transfer as an Event, which transitions the system to a state where the Recipient has exclusive ability to further transfer the Item.

The exclusive ability of the Recipient to further use the Item constitutes a modal quality of the system. In particular, we want it to be true that the Recipient *can* use the Item. So we want it to be true that for any possible kind of Event initiated by the Recipient and involving the Item can be initiated from that resultant state. Further, the exclusivity of this ability means that we want it to be true that no other Agent can initiate any kind of Event involving the Item from that resultant state.

B. Multi-Sender Transfer

A Multi-Sender Transfer is like a Simple Transfer, except that there are multiple Sender participants. Each Sender must authorize the Transfer, which usually happens via a sequence of Events that collect up the individual authorizations into an aggregate state. We therefore have a pair of modal properties that are required.

The first is that for any non-final state of the system, any Sender may initiate an Event that adds their authorization to the state (possibly redundantly), and only Senders may do this. In some sense, this is like the multi-recipient example except that here we have multiple actions in a sequence, rather than multiple alternative actions.

The second modal property is the same as for a single-recipient example, namely that the Recipient, and only the Recipient, may initiate any sort of Event from the resultant state of the whole Transfer.

C. Multi-Recipient Transfer

A Multi-Recipient Transfer is like a Simple Transfer, except that there are multiple Recipient participants who have the ability to further transfer the Item. The modal quality we want to hold is therefore that any of the Recipients can initiate an Event, but that no Agent other than one of them can do so.

Both Multi-Sender and Multi-Recipient transactions can be seen as just different versions of the general M-of-N pattern. They're distinguished here just for the sake of highlighting different aspects of the modal content.

D. Deadline-dependent Transfer

A Deadline-dependent Transfer (described in [1] section 2.3 as an example of a financial swap) is like a Multi-Recipient Transfer except that the Recipients have ownership rights during different time periods. In particular, prior to some deadline (perhaps a time, or an event, or some other condition), only Recipient 1 may transfer the Item, and after the deadline, only Recipient 2 may transfer the Item.

A typical use is to ensure that a smart contract, such as in a game where information flow is important, always remains alive, in some sense. That is to say, no participant's failure to act can deadlock the contract and cause others to permanently lose money. The modal content of interest is the claim that there will always be a state where someone completely controls the Item. Every possible future state, and also the current one, is such that there will always be a definite owner of the Item after the deadline has passed.

E. Rock, Paper, Scissors

In a blockchain-based game of Rock, Paper, Scissors (as described in [1] section 4) two players, Player 1 and Player 2, each provide their choice (either rock, paper, or scissors) together with some Item, after which the game moves into a state where one or the other Player can take possession of the Items by being allowed to transfer them.

Through the lens of the ontology, the game consists of two Agents, Player 1 and Player 2, who must each initiate an Event that contributes some Object to the game before the game is officially in play. This is a pair of modal properties, consisting of the possibility for an arbitrary Agent to initiate the first such Event, and then in the resulting state, the possibility for another arbitrary Agent to initiate the second such Event. In this third state, there is a disjunctive possibility, that either its possible for Player 1 exclusively to initiate some Events, or its possible for Player 2 exclusively to do the same.

IV. How to Reason, What to Reason About

The ontology described above is a set of basic conceptual primitives that can be used to reason about smart contracts, but of course we must also have the actual formal tools to reason about smart contracts, and we must have some actual system that can be used to implement smart contracts.

The modal nature of the ontology suggests very strongly that we should make use of coalgebraic models of programming language semantics. The techniques described in [2], for instance, could form the basis of such a framework. We can also make use of proof theoretic tools, such as temporal type theory and lax modal type theory [3], as a framework for reasoning about smart contracts. Work on linear epistemic type theory [4] [5] may also be relevant, as it is the type theory for distributed authorization systems, which is a fairly accurate description of what a blockchain-based smart contract is.

As for the substance of a smart contract, there are many options that are possible. The most straight forward is to explicitly define the coalgebraic information in a normal programming language. For example, suppose that the state is just a token for a finite state machine. We would need to define a type of state tokens, a type of interactions that the participants

in a smart contract can engage in, an initial state for the system to run from, and a transition function that defines how computation proceeds in response to an interaction. In Haskell, then, we might simply declare

```
data StateToken = ...
data Interaction = ...

initialState :: StateToken
initialState = ...

transition :: Interaction
           -> StateToken -> StateToken
transition = ...
```

Provided that we have a good model of the behavior of the host language, this would suffice. This is especially good if the host language is guaranteed to terminate, because then we can be certain than each individual transition will happen in finite time, even if the entire smart contract may proceed indefinitely into the future. A canonical example of such a program is a web server, where you want every request to terminate and return an answer (the web page), but you want the whole server to continue running indefinitely.

Writing a smart contract in this setting may not be ideal, however. It requires that the programming encode the behavior of the contract into the state and the transition function, which may not be the most intuitive representation, even if it's beneficial for computation. And alternative would be to define a domain specific language that more directly captures the nature of smart contracts as the primitives of the language, and enabls direct reasoning. This could then be compiled down into a language that makes execution easy, or an execution environment could be developed specifically for it.

Another option that blends the two would be to develop an embedded domain specific language. If we have a general purpose language such as Haskell, where we can define custom types and functions, the smart contract domain can be represented as just such a collection of those. This is the method typically used for EDSLs in Haskell, such as image and audio libraries.

V. Discussion

As can be seen from these example smart contracts, the use of a modal ontology where states are related by events can capture many important properties. Not only can we express rights that an Agent has by making a modal claim about possible events that the Agent may initiate, we can also express obligations by expressing that inaction can never cause deadlock. We can use these same techniques to express richer notions related to liveness to ensure that a contract can never get stuck in inescapable loops due to attacks.

There are important limitations to what can be expressed here, however. Stateful systems like smart contracts are effectively computers, and without severe restrictions on what constitutes the states of the contract, we may never be able to prove certain things. For instance, one might be tempted to prove that a given smart contract will always reach some final state where the contractual rights end, and obligations have all been fulfilled or else the contract has ended prematurely. However, if the states of the contract consist of Turing Machine tape and control states, then it may be impossible to prove. Since we can embed a Turing Machine into a contract, we run into the Halting Problem.

It is not desirable to limit the state of contracts, however. It is hard to predict just what uses people will have, and perhaps running a Turing Machine is perfectly reasonable. A library of contractual templates can be constructed, however, and these properties can be proven about them independent of particular instantiations, allowing users to easily get verifiable behavior.

Another interesting possibility is to view the stateful modal structure through the lens of virtual machines and compiler design. It may be possible to construct programming languages which have easily checked termination conditions which compile down into transition systems of this sort. The properties may then be easier to prove because they can be applied instead to the programming language not to the state machine that actually implements the contract. Precisely what such a language would look like is a possible future research topic.

Further work may also be needed to incorporate reasoning about knowledge. The ontology given above assumes that knowledge is a more primitive notion than what the ontology aims to capture, but this may not be true. Many smart contracts rely on temporary information hiding to function, and so it may be desirable to express the capabilities of Agents at any one time both with modalities and *also* with claims about how much they know relative to requirements of the system. work on linear epistemic logic for distributed authorization may be relevant here. However, this topic is also very broad, as knowledge and secrets can be construed to include any kind of information leak. Should we wish to express in a smart contract that no information about a number's parity is leaked by a set of Events? Perhaps, but this will require a great deal of information-theoretic proofs, or a very powerful type theory for secure computing. Both of these are very complicated. Security is also an issue for lower levels of blockchains, not just the level of contract logic, and so this may be the wrong part of the system to put that into.

Acknowledgments

I want to thank Lars Brünjes, Vitalik Buterin, Yoichi Hirai, Pablo Lamela, Alex McSherry, and Simon Thompson for their comments and discussion. It was invaluable to have their perspectives, especially where it revealed hidden presuppositions on my part about what the reader knew.

References

[1] Delmolino, K., et al. *Step by Step Towards Creating a Safe Smart Contract: Lessons and Insights from a Cryptocurrency Lab.* https://eprint.iacr.org/2015/460.pdf

[2] Backhouse, R., Crole, R., and Gibbons, J. *Algebraic and Coalgebraic Methods in the Mathematics of Program Construction*.

[3] Pfenning, F., and Davies, R. *A Judgmental Reconstruction of Modal Logic*. https://www.cs.cmu.edu/ fp/papers/mscs00.pdf

[4] Garg, D., and Pfenning, F. *Stateful Authorization Logic – Proof Theory and a Case Study*. http://www.cs.cmu.edu/ fp/papers/jcs12.pdf

[5] DeYoung, H., and Pfenning, F. *Reasoning about the Consequences of Authorization Policies in a Linear Epistemic Logic*. http://www.cs.cmu.edu/ fp/papers/fcs09.pdf