Motivation

This post breaks down @cwgoes talk (interpreted transcript) from Research Day NYC. I think it is worthwhile to read through the talk or at least have it organized in text format for the sake of information transmission purposes. Also, I hope this post kicks off more discourse on the topic.

- Slides
- Talk
- Synopsis

Disclaimer: Please note that slides and content herein are “extremely provisional” and we will release a write-up on the topic in the next few weeks which is better edited & more comprehensive.

Part 1 - Introduction, Inspiration & What Even is a VM?

Anoma envisions its role in a broader ecosystem as doing things which are complimentary, like doing things which other people aren’t doing based on what constraints we might have that are different. One thing that is different about Anoma as compared to projects in the Ethereum ecosystem is that we are willing to ditch backwards compatibility.

In both Ethereum and in Cosmos those systems embedded a lot of old design assumptions- made based on our world views from 5 years ago. Some of these assumptions have proven to be right and some if not wrong, at least orthogonal to what has actually happened.

Anoma has been working on this intent-centric idea in vague terms for a little while. Now its permeated its way through all the layers of the protocol stack to what you might call a VM.

Inspiration - What is an intent?

There is no one true, holy answer, to what is an intent? This is our answer to how can intents be represented and satisfies design criteria that many of the projects represented at research day share.

Talk was inspired by Andrew Miller who said

“Intents are just txs, let’s fight”

If you squint really hard intents/txs feel like some data you send to the distributed system and something happens. But that is maybe squinting too hard.

Inspiration Part Deux

Thanks to Xyn from Flashbots

They have no idea intents are higher-order commitments with extensive literature defining its semantics

Interestingly, the extensive literature is filled with folk theorems, ie. where people seem to all know something, but it took a while to write something down.

This Talk Is Not About Your Grandma’s VM

This talk is about a general intent-centric virtual machine. VM is used to refer to many things which are not in the same category, so we will clarify what we mean by VM.

Von Neumann Architecture

A long time ago people were building computers, and they wanted computers to execute programs and those computers had hardware constraints.They had a processing unit, instruction set, registers/stack, volatile memory, non-volatile storage. And they wanted to execute programs sequentially. This is 1960 no one’s thinking about MEV.

The distinction between control and arithmetic/logic units doesn’t matter anymore. There is a communication system between the CPU and it is executing instructions sequentially. As a result you run through your programs accessing memory and I/O when you need to.

The EVM

The EVM in the grand von Neumann style is a von Neumann VM. It has a program counter, an instruction set, it has different layers of memory and storage, and it goes through and runs programs sequentially.

What is a VM, Anyways?

We are going to talk about a VM designed to do something else. It's not competing with the EVM, but it is doing something which is more suitable to the intent-centric world. In particular, this VM is designed to match conditional commitments atomically.

Commitments involve programs. You might say well you need something that executes programs, why not use a von Neumann VM? Yes, you can do that, but it's not quite the relevant problem. What you want to do in matching commitments is to execute several commitments atomically.

In an intent-centric world you care about whether the result of executing those intents or those commitments is satisfactory to all involved parties. Maybe you care a bit less about the specific path of execution to get there. There is a slightly different design constraint. The EVM also includes other things which are not exactly von Neumann things. Message passing between programs which includes scheduling, those are relevant.

But… Turing-completeness

Something people sometimes bring up in the discourse is Turing-completeness. If you want something why don't you just emulate it on this Turing-complete VM which we already have? Yes, you can emulate the thing which we're talking about on the EVM.

The question we're interested in is; what does the execution environment look like for intents? How you emulate that is interesting from a performance standpoint but a separate question. You could also emulate on a non-EVM von Neumann machine & it wouldn't change that much.

Part 2 - Research Literature

Intents are binding conditional commitments from 50-year old game theory.

Supergames (A Non-cooperative Equilibrium for Supergames, Friedman 1971)

From the 50 year old game theory, the first folk theory to be written down was by Friedman in 1971, the equilibrium for "Supergames." The result established in the paper was that repeated interaction can result in any feasible, individually rational payoff.

Program Equilibria

You can achieve this same result if you have users instead of taking actions themselves, they use commitment devices or programs and commit themselves to a strategy. Because this provides a credible guarantee of how users will act then you can do stuff like this in the prisoner's Dilemma.

If MyProgram = OtherPlayerProgram DO(coop); else DO(defect)

Program Equilibria are what you might call the closest near term research basis for intents. The research literature here describes the problem pretty well from a mathematical perspective in terms of what we want to get out of these systems and why it's interesting.

What if we just sent around commitments and settle them somewhere?

Forget blockchains. Commitments are functions, published to a blockchain somewhere. The blockchain somehow calculates an equilibrium which is individually rational and feasible.

Four Challenges

We argue this runs into four challenges if you take the research literature and try to implement it.

- Termination/fix-point finding
- Function introspection
- Nominal/structural identities
- Unclear information flow

(1) Termination/fix-point finding

If you use the most powerful representation in the research literature which gives you the best game theoretic results, you need to use higher order commitments, as in commitments which are dependent on other players commitments. Eventually you want your execution to terminate, which means that the commitment system runs forever unless someone short circuits somehow. There are 2 approaches in the research literature to short-circuiting.

Randomness - a shared randomness beacon, where you say 1% of the time I'll cooperate else I'll call the other function and that function will call me, and I'll call that other function until some randomness happens, and we'll both cooperate and then it will terminate. Randomness adds an assumption which you maybe don't want, and there is an annoying trade-off where you have to run for a lot of time until your randomness causes the execution to terminate.

When your execution is replicated, that seems like a very expensive thing to do. This is a more serious implementation problem than it appears to be. Another way to do this is add a separate payoff is to bribe someone to make you terminate, and that also works but creates unnecessary MEV.

(2) Function Introspections

Program equilibria deals with this problem by reading the source code for the other guy's program.

If MyProgram = OtherPlayerProgram DO(coop); else DO(defect)

The problem is the = sign is an open Type theory research problem, in the general case. In particular if you want to deal with programmatic preferences which are going to encode complex things, and you care about specific structures, you would need full dependent types.

You need to prove arbitrary properties of theses other functions to figure out what your fixed point is. This seems very heavyweight. Ideally, we wouldn't need to embed dependent types as a design requirement for our intent-centric architecture.

(3) Nominal vs. Structural Identity

In these formalisms all the players are known, they are referred to by index. If you have a formalism that uses commitment devices these devices refer to each other by index. This is a closed world model.

This does not match what we need in the typical distributed blockchain setting. We are dealing with this open world where we don't necessarily know all of the identities of the players (we don't want to know for privacy reasons) and the set is very large.

Anyway, we typically don't care because we want to find counterparties on the basis of structural identity, what capabilities they have. Something like owning a token is a capability. Maybe we are using these systems to prevent war and we want credible actions that some party took action in the external world. So we need to change the basis of identity for this literature to include all actions that matter in our context.

(4) Unclear Information Flow

Typically these constructions in the research literature rely on some magic logically centralized commitment-executing computer. It needs to take all these commitments, calculate the equilibrium, and then it needs to output that state.

Unfortunately this requires a single logical point. This doesn't give you a fine grain information control. You may want to know things like I have a bunch of these commitments which are not codependent, so I can deal with that separately. This framework doesn't give you a way to natively check that.

Part 3 - Resource Logic to the Rescue!

Now we discuss the specific construction we have in mind called resource logic. We chose this term bc there seems to be similarities to distributed linear logic. We care about no double spends in the blockchain context and we like the idea of modeling everything as resources.

Sequential Commitment Execution

There is a basic trick that resource logic uses to solve or redress several of these concerns. That trick is to forget some information or not care about some information.That information is the path of execution. In a typical commitment scheme you have commitments; you do one commitment then check the next commitment then check the next commitment. It's like executing small state transitions one at a time.

One Weird Trick

We can get atomicity by changing the type slightly. Instead of having commitments be higher order functions we can instead pretend that we already executed everything. Commitments will say yes I'm okay with this or no I'm not. Still they are issued by particular parties who have the ability to authorize particular actions. This is a very high level view.

No One Cared About the Path Anyways

If we do this we can not care about the path.The earlier talks talked about these different intent-centric vertical specialized systems like account abstraction. One thing these systems often differ in is where the execution is specifically happening. Ultimately there is still some verification on chain but sometimes someone is executing to do some meta-transaction.

If we can architect our VM in a way which doesn't care about where the execution happens as long as the final result is in alignment with everyone's preferences and satisfies the state transition constraints of the system, then we generalize all of these cases. As in people can choose a specific topology for where the specific components of execution happen at runtime, and all that we need to check on the final blockchain that we've agreed to trust for consensus and state custody purposes is that everyone who took an action in this transaction is happy with the final outcome.

Basic Structure: Resource

So how do we encode this? In Anoma we encode this in something we call Resources. Resources are a little bit like smart contracts but not quite. If you squint you could also call them smart contracts but that term has become associated with many things and some of those things resources are not.

In particular resources do not encode imperative execution logic. Smart contracts typically encode start at state 1 do some computation end at state 2. Resources don't encode this. Instead resources encode constraints.

In Anoma each resource has what we call logic, where logic is this predicate function over partial transactions, maybe some arguments, Prefix, Suffix, Quantity, and some arbitrary value.

data Change = Created | Consumed

data ResourceLogic = ResourceLogic { predicate :: PartialTx -> Change -> Bool, arguments :: ByteString }

data Resource = Resource { logic :: ResourceLogic, prefix :: ByteString, suffix :: Nonce, quantity :: Integer, value :: ByteString }

If you want to squint and think about them as smart contracts the logic is the code and the value is like the state in the Ethereum system. But we have this built in way to encode fungibility into the VM and resources don't specify any kind of execution engine.

Basic structure: PartialTx

The units of the system, the closest thing to an intent is what we call a PartialTx

.

data PartialTx = PartialT× { consumed :: Set Resource, created :: Set Resource, extradata :: ByteString }

A partial transaction includes 2 sets of resources. A set of resources which are consumed and a set of new resources that were created. And it includes some arbitrary extra data such as signatures.

Basic Structure: Validity

And the idea behind PartialTxs is we can have this resource validity check at the partial transaction level.

valid ptx@(PartialTx consumed created _) = all (map (\r -> logic r pt Consumed) consumed) && all (map (\r -> logic r pt Created) created)

A PartialTx

might not be entirely balanced. We might spend resources that we don't actually have or create resources that aren't yet consumed. But we can check that all of the predicates are satisfied, just by mapping them.

Basic Structure: Balance

Then we can separate out what we call the balance check. The balance check relies on different denominations of resources. The denomination of resources is calculated based on the logic, prefix, and static data.

type Balance = [(ByteString, Integer)] denomination :: Resource -> ByteString denomination (Resource logic prefix _ _ _) = hash (logic, prefix)

delta :: Resource -> Balance delta r = [(denomination r, quantity r)]

balanced :: PartialTx -> Bool balanced px@(PartialTx consumed created _) = sum (map delta created) - sum (map deita consumed) = 0

What we require for a transaction to be balanced is that there is no change. The sum of the delta of all the created resources less the sum of the delta of all the consumed resources is zero. This is the linear logic check.

Properties

This has several nice properties. PartialTxs compose. If you take two 2 valid PartialTxs and you join them, their created and consumed resources, you end up with another valid PartialTx, the system is compositional. A valid FullTx

is just a PartialTx

that in fact happens to be balanced. There is no hard distinction between intents and txs anymore.

When you create these PartialTxs and compose them and append to them you can do all of that execution wherever you want. It just needs to end in this valid FullTx which you publish to the blockchain.

Example: Prisoners Dilemma

It's almost like you just pretend you can do some execution which you actually can't do.You pretend that you're the other player, and you consume a resource that your counterparty cooperates. You create a resource that says you cooperate, which you have permissions to do.

Furthermore, you don't have the permissions to create a resource that says your counterparty cooperates, but you can consume one.Where then, this balance check differs the check of the validity of the whole thing to the Tx level instead of the PartialTx level.

Symmetric PartialTx - If you create a resource that says you cooperate and consume a resource that says your counterparty cooperates and your counterparty creates the inverse then these will balance because the denominations will cancel out.

Example: Token Swap

If you use this for something like a token swap & you consume the resource representing the tokens you already have & want to pay, you create a resource of new tokens assigned to you. Similarly, symmetric PartialTxs will balance w/ some slack (MEV question).

Part 4 - Addressing Challenges & Spicy Takes

Addressing Termination & Introspection

This model addresses these concerns with faithfully translating this conditional commitment concept. In particular it addresses this kind of termination/introspection difficulty just by splitting computation and verification

.

When you make a PartialTx

, you can make some state changes which you don't even have permission to do, they are far down the line. You can send the PartialTx

to someone else who could make the state changes which if the whole system were sequential would've happened first.

But they don't need to happen first because we've made our VM agnostic to the actual ordering of computation. All it needs to care about is that everything checks out okay in verification.

Addressing Nominal vs. Structural Identities

To deal with nominal & structural identities when we specify interactions on the basis of resources; they already connote the ability to do something bc they are owned. So the inclusion of other players is explicit in this model. If you want to check that your counterparty cooperates, you put that check right in the validity check of the PartialTx.

You consume the resource that says your counterparty cooperates and the whole tx is only valid if your counterparty in fact sees these and creates the resource that says they cooperate.

Addressing Information Flow

This is a VM, not a language for information flow. Because validity conditions are separate from the balance check this helps a lot with building a good substrate for information flow. In particular it means that you can make the proofs of validity separately and prior to checking balance.

If I spend my tokens in a PartialTx and the whole thing is unbalanced but I can make a proof for that spend, I can send the

PartialTx to you and you can see what constraints have to be satisfied in order to balance it without seeing the path of execution that allowed me to create that potential constraints in the first place. The validity constraints are forwarded and they can be satisfied at any point during execution as long as they are all satisfied in the end.

Spicy Take

This resource structure is inevitable. What you really want in order to generalize intents is to build a kind of VM or execution environment which is agnostic to where execution happened. You don't want the VM to care about the path, only the end result.

You can still use the EVM to do execution, that's fine. You would end up with something on top of the EVM which would do intent matching and end up splitting out validity and balance checks in this way.

If you end up with this then the EVM does things you don't need, like sequential message passing execution which is fine but doesn't solve the problem that matching conditional commitments wants to solve because the EVM is path dependent, and we don't care about the path.

Part 5 - Towards a Substrate for Information Flow Control

How we think this resource model can act as a substrate for information flow control. What we described thus far is not a language for information flow control, it's kind of like a runtime.

Viaduct [Recto 2009]

The way this paper, Viaduct, structures things is they have a high level source language that describes in a declarative way, information flow control policies like constraints, who is trusted, integrity assumptions, cryptographic assumptions, etc.

Viaduct is a compiler that transforms high-level programs into secure, efficient distributed realizations.

Language vs. Runtime

Then that language is compiled somehow into instructions and run using actual cryptographic primitives. The runtime execution engine is responsible for running the primitives in the correct sequence with real data. This is what the resource model is suitable for, as in it can be a part of specific cryptographic primitives this kind of runtime that executes PartialTxs with real data.

If constructed correctly, it can enforce information flow control policies by calling the primitives in the right order. You can have many different higher level languages which compiled and run on something like this.This system is helpful because it is very amenable to what we call the "least powerful primitive."

There is a power ranking of cryptography primitives where you have very powerful primitives that do everything but are extremely slow, and you have very fast primitives which do only one thing like ZKPs which are much faster. Typically you want to architect your system so that you can use the least powerful primitive you need for the specific task you are trying to do under the specific information flow constraints that you have. It will make your system more efficient faster and amenable to replicated verification in the case of ZKPs.

Example: Solver Selection

How would you use this resource logic runtime to provide the information flow control people want is Solver selection?

In the system you still have to reveal some info to solvers in order for them to match ur intents or put together PartialTxs. Maybe you care about which solvers you send those intents to. Because we have this **separation of validity from balance** you can do things like; find me a counterparty and don't forward it and don't reveal it to someone else.

Because we have this separation they can do that. Once they find a counterparty they can create another PartialTx which already doesn't reveal your personal data anymore. Since we have compositionality we can get these nice information flow control properties.

Example: Batching ("Penumbra-On-Anoma")

How would you do it?

You would consume tokens to create threshold encrypted resources which would be queued in some type of block batch. Those resources are threshold decrypted next round, next block, and they have logic such that their logic only allows them to be consumed in this batch. The validators still have to attest to what was included in the batch, they provide data availability.

The validators attest to what was included in the batch and the logic checks that all the resources are consumed, that the fairness conditions like optimal arbitrage was satisfied.

Example: Threshold FHE (Aggregate Statistics)

Another thing you might want to do is something like aggregate statistics. In a privacy first world where all of your transactions are private, sometimes you might want information flow control to decrypt aggregates that still allow you to reason about aggregate properties of a user, de-anonymizing specific users.

For example if you have a private bridging system you may want to reason about economic security which requires that you know how much of say some assets are secured by the POS assets of some particular chain. In order to do that in a world where bridges are private, you need to decrypt some aggregate amounts of assets.

Resources don't magically solve any of the cryptography problems for you, but they create this nice separation of different parts of state and logic. Then you can enforce those different information flow control policies on those different parts of state. This is why it's an amenable substrate.

Part 6 - The Case for the Resource Model & Conclusion

Resources package data and logic together very cleanly. They separate out execution so you don't need to worry about execution paths, only results. They make state inclusion very explicit. Instead of having implicit global state, you only access the state that you need to validate for the purposes of your specific application. There is no path dependence, which is helpful for information flow control.

Conclusions and Future Directions

Three points to remember if you only remember 3 things from this thread/talk;

- The Intent-centric VM design is not a von Neumann problem, we are trying to build something else. Its not competing, it is orthogonal.
- Speculative execution or this kind of executing things you can't authorize yourself + atomicity is a powerful tool. Users care about the final output state, the game theoretic equilibria, not the path it took to get there. Often, you don't even need to compute the path.
- The resource model is probably inevitable. You can call it something else, change some specific decisions about variables and code. But the atomic execution of conditional commitments and the separation out of the execution path so it can be determined at runtime are things that any intent-centric architecture is going to want.

Open Questions

Anoma open research questions for ResearchDay NYC

• Better formal languages for information flow control particularly amenable to conditional disclosure

• Most suitable abstract IR for programs

• Solver privacy improvements using TEEs in combination with ZKPs in efficient ways