Yesterday in discussions with @degregat and @Michael it became clear to me that we have not sufficiently spelled out the various flavours of transaction functions and times during the transaction authoring, solving, and ordering process. I thought it might be useful to write up my current understanding, so this is an attempt at that.

First, let's distinguish between three times which are relevant for each transaction:

- "Creation time", which happens when the user decides that they want to issue an intent to the network. The first computation here happens on the user's local node, and may access state which is private to that user (e.g. shielded resources). From the perspective of the network, all of this is a single point in time (visible to the network as soon as the user's node sends a message to some other node).
- "Solving time", which happens when nodes on the network are searching for ways to compose intents (unbalanced transactions) of which they are aware. This is really multiple points in time, but all of them happen in between the creation time and the execution time, so we can treat them as a block for the reasoning here.
- "Execution time", which happens after

a final ordering of transactions involving the controller in question has been picked. For simplicity, we can assume that the controller themselves is executing the transaction functions, although this need not always be the case.

Second, I think it's necessary to clarify the two kinds of representation in play:

- A transaction object

(the data structure specified in the RM report) is only fully produced during execution time, when all ordering information is known.

- Prior to execution time, we're really just creating, sending around, and processing transaction functions

of various flavours - and technically they don't even return a transaction object

until the final transaction function when it is executed - transaction functions during creation time and solving time just return more transaction functions, in a sort of "distributed partial application" process (there's an old discussion about this)

A further complexity here is that, in order to search for potential matches, a node really must be able to introspect the transaction function and extract the transaction object inside, since there's no way to search for matches to an opaque function. Now, there are two issues here:

- If ordering information is required to determine what the transaction can be matched with, the solver either cannot search at all (since they don't have that information), or must somehow analyze the relationship between ordering and the transaction (whatever computation in the transaction function), and try to find a match which guarantees that the computation will result in a balanced, valid transaction.
- In more common cases, simple post-ordering computations such as incrementing a counter for statistics purposes are independent from the transaction delta, and the solver really just needs to extract the "constant part" of the transaction function to do their analysis, then add back the counter-incrementing code somehow later.

One can imagine that we might want a representation of transaction functions which is easier to decompose and analyze. A first stab would be to simply represent transaction functions as Set TxOrFunc

, where:

type TxOrFunc := | Tx Transaction | Func (() -> Transaction)

and the results of the transactions and functions in the set (applied post-ordering) are to be composed together via the transaction composition operator. Then, a solver node can examine directly the Tx

items in this set to search for matches, and if those independently balance, the functions computed during execution time should result in a valid and balanced transaction.

I don't know if this is the right direction here or not yet, there are probably other options. I also wonder if we need separate names for:

- the structure passed to the node during creation time (maybe "transaction creation function")
- the structure passed around the solver network (maybe "transaction solver function")
- the structure passed to the controller (this one could stay as "transaction function")

… but these are just candidates. Thoughts welcome.