Introduction

I think it would be very helpful to have a discrete mathematical model of intents that captures the essence which we want to capture without unnecessary implementation detail, but which is specific enough to clarify what we do and do not mean by the term “intent”. The best candidate I have thought of personally is that of an “intent machine”. In this post, I will walk through what an intent machine is, what definition of “intent” it provides, and (broadly) how this definition relates to Anoma’s protocol components.

Definitions

Fix a state type T

(can be equivalently understood as a possibly infinite set of possible states).

An intent

I_T

is a function of type T \to T \to { 0 | 1 }

.

An intent machine

IM_T

is a potentially non-deterministic function IM_f

of type (T, Set\ I) \to (T, Set\ I)

. Informally, the first tuple consists of the prior state and a set of intents which could possibly be processed, and the second tuple consists of the posterior state and a set of intents which was actually processed. Let’s name the arguments for clarity:

IM_f :: (prior :: T, available :: Set\ I) \to (posterior :: T, processed :: Set\ I)

In order to qualify as an intent machine, the function must preserve the property that all intents actually processed were satisfied:

\forall i \in processed . i\ prior\ posterior = 1

.

Let’s call this property “intent adherence”.

IM_f

is required to satisfy this property, but it is not required to be deterministic.

Analysis

Without loss of generality, IM_f

can be decomposed into two steps and one constraint.

Two steps

1. computing a set of (candidate state, processed intents) tuples which satisfy intent adherence
2. selecting one of the tuples to return

Constraint

IM_f

may additionally constrain which state transitions are considered to be valid (potentially returned). This constraint can be modelled as a singular “system intent” which must always be satisfied, and it may itself change from one invocation of the machine to the next, so we shall model it as a distinct component of state:

T := (I_T, T')

.

IM_f :: ((I_{{system}*{prior}}, prior'), Set\ I) \to ((I*{{system}_{posterior}}, posterior'), Set\ I)

IM_f

then satisfies:

I_{{system}_{prior}} \ prior\ posterior = 1

.

Let’s call this property system intent adherence

.

Note that I_{{system}_{posterior}}

is not checked, but it will constrain future invocations of the intent machine.

Let’s go through the two steps in detail:

1 - Enumeration

The enumeration

step enumerates all possible valid return pairs of (T, Set\ I)

.

{(posterior, processed)\ |\ \forall i \in processed . i\ prior\ posterior = 1, processed \subset available, I_{{system}_{prior}}\ prior\ posterior = 1}

2 - Selection

The selection

step picks one pair from the set of valid options.

choose :: Set\ (T, Set\ I) \to (T, Set\ I)

.

Selection

All of the interesting structure happens in the selection phase. To aid the intuition, here are some example choose

functions:

- “Pure chaos”
- Select at random a valid return pair
- Select at random a valid return pair
- “Pareto-efficient chaos”
- Select the return pair which satisfies the most intents, break ties with randomness.
- Select the return pair which satisfies the most intents, break ties with randomness.
- “Utility maximization”
- Select the return pair which maximizes some scalar function U_T :: T \to Nat

.

- Select the return pair which maximizes some scalar function U_T :: T \to Nat

.

- “Profit maximization”
- Utility maximization with U_T

as the balance of some specific token owned by the operator’s address in the posterior state.

- Utility maximization with U_T

as the balance of some specific token owned by the operator’s address in the posterior state.

- “Expected utility maximization”
- Select the return pair which maximizes U_{{T}_{expected}}

, given some probability distribution over future intents conditional on the posterior state.

- Select the return pair which maximizes U_{{T}_{expected}}

, given some probability distribution over future intents conditional on the posterior state.

Relation to Anoma

A resource machine

is an intent machine with a system intent that interprets the state as a set of created/consumed resources, checks resource logic adherence, and enforces linearity.

Taiga is a concrete instantiation of a resource machine (currently using Halo 2).

Typhon, roughly, is a concrete instantiation of a distributed intent machine. This needs to be treated in combination with composition and will be covered in a follow-up post.