Ownership types

Martin Lundfall

June 30, 2017

For some time, the idea of using linear logic or linear type theory to model ownership on the blockchain has been floating around (See for example Merediths' *Linear types can change the blockchain*). But outlining how this would work in detail is not trivial, and I have yet to see anybody explain on the level of detail I would like to see.
Here are my own attempts to make sense of the concept. The setting is mainly based on Krishnaswami's *Integrating Linear and Dependent types*. The key realization that I am pursuing here is that handling ownership is a lot like handling pointers; ownership is an association of an actor to an item, whereas a pointer is the association of a location to some data. Both the actor/location and item/data are not linear types themselves, they are ordinary types (what I will call **cartesian**) to which we may refer to (or use) many times in a derivation, but the actual **ownership** (or pointer) is a linear resource.

1 Rules

A **cartesian context**, Γ consists of the execution variables

- `msg.sender`,
- `tx.origin`
- etc

together with the input data of the tx.

2 Overview

The general form of a derivation/function is:

$$\Gamma; \Delta \vdash a : A; b : B$$

where Γ is an (ordinary) **cartesian context**, A an (ordinary) **cartesian** type, Δ a **linear** context, and B a **linear** type. We can think of the constituents of Γ as **conditions** that need to be met in order for an action to execute, the constituents of Δ as the resources needed, the term $a : A$ as the functional output and the term $b : B$ as the resulting state of resources. A function that is the identity function on the linear part of the derivation is called **pure**. We may omit writing out the linear part of such functions.
For some functions, we are not interested in the cartesian result. In those cases, we may simply write $\Gamma; \Delta \vdash b : B$.
Thus when our derivation is of the form: $\Gamma \vdash a : A$, we mean that A is a cartesian type, whereas $\Gamma; \Delta \vdash a : A$ implies that A is linear. We require constructor functions to have empty linear contexts.

3 Ownership

At the most fundamental level, an ownership is the association of an **actor** to an **item**.
Ownerships can be created, destroyed or transferred. As it is important to distinguish the former two from the latter, we call the former two **ownership management**, and the latter **ownership morphisms**.
The basic Ownership type, $[t \mapsto A]$, associates an actor t, with some term of type A:

$$\frac{\Gamma \vdash t : \mathit{actor} \quad \Gamma \vdash A \ \text{type}}{\Gamma \vdash [t \mapsto A] \ \text{linear}} \ \texttt{Own-Formation}$$

We call terms of $[t \mapsto A]$ **ownerships**. Whenever an ownership exists, we can refer to the item being owned via the following rule:

$$\frac{\Gamma; \Delta \vdash m : [t \mapsto A] \quad \Gamma \vdash t : \mathit{actor}}{\Gamma; \Delta \vdash \mathit{get}(t, m) : A; \Delta} \ \texttt{get-item}$$

or the more complicated (Krishnaswami version):

$$\frac{\Gamma \vdash t : \mathit{actor} \quad \Gamma; \Delta \vdash m : [t \mapsto A] \quad \Gamma, x : A; \Delta', y : [t \mapsto A] \vdash c : C}{\Gamma; \Delta, \Delta' \vdash \texttt{let } (x, y) \texttt{ be 'get } (t, m)\texttt{' in } c : C}$$

3.1 Ownership management

Ownership management are usually rarer than ownership morphisms. In a vanilla token contract without any minting functions, there is only one instance of an ownership managing operation: At the creation of the contract, the contact creator is given the initial supply of tokens:

```
function Token(uint256 supply) {
        _balances[msg.sender] = supply;
    }
```

In our type system, ownership creation is the introduction rule of the Ownership type.

$$\frac{\begin{array}{c}\Gamma, x : actor, y : A \vdash C \ \text{ type} \\ \Gamma \vdash t : actor \quad \Gamma \vdash a : A \\ \Gamma \vdash c : C[t/x, a/x]\end{array}}{\Gamma; \cdot \vdash (\lambda t.a) : [t \mapsto A]} \ \texttt{Own-Creation}$$

As we can choose the type C freely, we may choose it to be a condition which is hard to satisfy, such as an equality type between actors:

$$C := msg.sender =_{actor} owner$$

ensuring that only the owner of the contract can mint tokens.
The condition can also depend on the item being owned, as well as the owner. For example, we may want to allow everyone to hold a balance of 0 tokens by setting C to $x =_{\mathbb{N}} 0$.
The destruction of an ownership is similarly given by:

$$\frac{\begin{array}{c}\Gamma; \Delta \vdash b : B; m : [t \mapsto A] \\ \Gamma \vdash c : C\end{array}}{\Gamma; \Delta \vdash I} \ \texttt{Own-Destruct}$$

3.2 Ownership morphisms

Transferring ownership is a simple linear function:

$$\frac{\begin{array}{c}\Gamma \vdash s : actor \\ \Gamma; \Delta \vdash m : [t \mapsto A]\end{array}}{\Gamma; \Delta \vdash trans(m) : [s \mapsto A]} \ \texttt{Own-transfer}$$

This is fine for the cases where A is some kind of record, or unique item, but how would we model ownership of tokens? An actor t having balance of n tokens could be represented by the term $(\lambda t.n) : [t \mapsto \mathbb{N}]$, but the above function does not allow us to transfer part of a balance. Instead, we could model ownership of all of the individual tokens, and say that a balance of n tokens consists of a term of the type $[t \mapsto \mathbf{1}]^n$, representing the n-fold tensor product $[t \mapsto \mathbf{1}]^n := \underbrace{[t \mapsto \mathbf{1}] \otimes [t \mapsto \mathbf{1}] \otimes \dots [t \mapsto \mathbf{1}]}_{n \text{ times}}$.
Transferring m of your $m + n$ tokens would be an m-fold application of `Own-transfer`, yielding the linear function:

$$transfer(n, s) : [t \mapsto \mathbf{1}]^{m+n} \multimap ([t \mapsto \mathbf{1}]^n \otimes [s \mapsto \mathbf{1}]^m)$$

Another way of would be to introduce another a class of primitve transfer functions, parametric on n, which can only be applied to Ownership types of the form $[t \mapsto \mathbb{N}]$:

$$\frac{\begin{array}{c}\Gamma \vdash s, t : actor \\ \Gamma; \Delta \vdash b : [t \mapsto \mathbb{N}] \\ \Gamma \vdash n \leq getBal(b, t)\end{array}}{\Gamma; \Delta \vdash ((s, n) \otimes (t, getBal(b) - n)) : [s \mapsto \mathbb{N}] \otimes [t \mapsto \mathbb{N}]} \ \text{transfer-n}$$

4 Minimal Example: Token

TODO

5 Questions

Are there any other actors but addresses?
What sort of interaction is necessary between linear and cartesian types? The a modality ! sending linear types to cartesian is probably pretty useful.