A SIMPLE MODEL OF THE LAMBDA CALCULUS

DANNY GRATZER

Hi folks. Today I'd like to talk about something that came up during the talks Dana Scott gave this week at CMU. He was discussing a new way of integrating randomness into models of the lambda calculus, but the model which he was building this all on was surprisingly simple and nice. In an attempt to consolidate my notes I'm writing down everything that I can remember about what are called "graph models" of the lambda calculus. Hopefully it'll be helpful to someone else.

Since these are just my notes they come with a huge disclaimer, they're only correct to the best of my knowledge

1. Observations about Models of The Lambda Calculus

If we want to construct a model of the lambda calculus we need to construct some domain of discourse, D with operations on it letting us model the critical rules of the lambda calculus. Specifically we need to be able to simulate one particularly troublesome rule.

$$(\lambda x.\ E)E' \equiv [E'/x]E$$

What makes this rule such a bother is that it forces us construct some way to convert $d \in D$ into a function upon D. Furthermore, in order to properly interpret lambdas into the model we'll want to find some way to turn a function on D back into a term on D. It doesn't necessarily need to be an isomorphism, but it certainly needs to be an embedding. That is, if we have $\text{enc} : (D \to D) \to D$ and

$\text{dec} : D \to (D \to D)$ it really needs to be the case that $\text{dec} \circ \text{enc} = 1$. It would be even better though to be able to construct a model where these two collections are isomorphic, meaning that all functions on our domain are computable functions and all elements of our domain are computable functions. This is not as easy as one would hope though, if our domain has 2 distinct elements (it needs to in order to be a reasonable model) then $D \to D \cong \mathcal{P}(D)$. Then Cantor pops up and we know that we won't be able to just build some set so that $\text{dec} \circ \text{enc} = 1$.

This circularity is going to come back up in pretty much any model you attempt to build and this won't be any exception.

2. Using Graphs

In order to construct our model we're going to think of lambda terms as graphs. They'll pair their input to it's output rendering the term into one huge stinking table. This is basically how a set theorist would describe their functions so it's not

completely out of no where. If a lambda term doesn't terminate on a particular input that's fine, we'll just neglect to add an entry there. This matches our intuition that two functions are equivalent if they coterminate and are equivalent on all the cases where they terminate.

So then we can think of how to define enc and dec. You should think of enc as a function squashing a function into the appropriate look up table.

$$\text{enc}(F) = \{(m,n) \mid n = F(m)\} \qquad \text{dec}(d) = m \mapsto \begin{cases} n & (m,n) \in d \\ \bot & \text{otherwise} \end{cases}$$

There I just mean $\bot$ for undefined, it's notation not some specific element of our model. This looks so great and yet it's completely broken. The whole thing starts to unravel as soon as we ask the question "what are the entries of our tables?" Then we'll notice that our tables are built of.. tables, made of tables, made of tables, .. off and off and off to infinity. A little set theory is enough to convince us that this will lead to paradoxes. In particular it shouldn't be tricky to show that we couldn't actually realize a set of such lookup tables.

So now we've reached the main technical challenge, this is the circularity I've hinted at before; the model we want to define is just too big to exist. In order to make sure that we don't make the same mistake twice let's get a little more serious about our math and fix some concrete set to work with. Specifically, let's look at the powerset of natural numbers. There's nothing special about the natural numbers other than them having an infinite number of them.

Our clever idea here is that while we definitely can't encode tables of tables as sets of numbers, we can encode pairs of natural numbers as natural numbers. It's not so hard, just

$$(n,m) \triangleq 2^n(2m+1)$$

("Fun" exercise, convince yourself that this is injective)

Pairs are a good start, but we need more than that, we need a finite set of pairs. But looking to lisp, we can easily see we can encode finite sequences with our pairing operation as cons and nil is just encoded as 0 (our pairing operation never gives 0). And from sequences there's a simple but woefully non-injective correspondence between finite sets of pairs and sequence pairs. These can be mapped to sequences of natural numbers, which are just natural numbers. I'll use the notation $\mathbf{set}(n)$ to indicate the finite set of natural numbers n denotes by this process.

So we've Gödel numbered our way to being able to think of finite tables as numbers. So, since we're talking about $\mathcal{P}(\mathbb{N})$, that means we could regard each element really as an approximation of the structure we were talking about before where each time we go down a level in the table we go from infinitely entries at the top to finitely many entries, and then fewer and fewer until we're completely out. We're forced into a much smaller model than we had before but it actually exists which is a step up I suppose.

Now we have to convince ourselves that we can work in such a restricted model. Let's define a few helpful concepts to help us show this. Let

$$X^* \triangleq \{n \mid \mathbf{set}(n) \subseteq X\}$$

We then know that $X^* = \{\mathbf{set}^{-1}(F) \mid F \text{ finite} \wedge F \subseteq X\}$. We can see X^* as a collection of approximations of X. It is after all just an encoding of a set of sets which all describe some finite fragment of X. It's not a terribly efficient way of describing X because there's lots of duplication between approximations but having all the finite approximations of a set is enough to uniquely determine it (just union the approximations).

What we do then is define dec and enc by piecing together all of the approximations of relevant sets.

$$\text{enc}(F) = \{(n, m) \mid m \in F(\mathbf{set}(n))^*\} \cup \{0\}$$
$$\text{dec}(d) = X \mapsto \bigcup\{\mathbf{set}(m) \mid \exists n \in X^*.\ (n, m) \in d\}$$

So enc creates a table mapping a fragment of input to the appropriate fragment of output. We also tack on $\{0\}$ which is our Gödel encoding of $\emptyset$ because "completely undefined" is always a valid approximation of a function, if a poor one. To reverse the process and get a function out of a lookup table we take our input and rip it apart into all its valid approximations and piece together what the lookup table gives us.

The natural question to ask is whether these functions cohere with each other. Is it the case that $\text{dec} \circ \text{enc} = 1$ for example? The short answer is no for a variety of reasons. Remember that while we're encoding everything into a lookup table we can only ever encode finite entries, infinite sets can't have a Gödel number. In order for us to be able prove that $\text{dec} \circ \text{enc}$ could *ever* be an identity it needs to be possible to determine the behavior of our function from only the finite cases.

This idea is a driving idea between a lot of Scott's work on lattice theoretic models. It stems from the observation that computation is smooth. In the more general sense we can state this property with the preservation of least upper bounds of directed sets. In our specific case we can use the more simplistic definition that F is continuous if

$$F(X) = \bigcup\{F(Y) \mid Y \subseteq X \wedge Y \text{ finite}\}$$

If we apply our trick with Gödel numbers this is equivalent to

$$F(X) = \bigcup\{F(\mathbf{set}(n)) \mid n \in X^*\}$$

It's not hard to prove that if F is continuous then $\text{dec}(\text{enc}(F)) = F$. The reverse has some complications, while $\text{enc} \circ \text{dec}$ will "morally" work on the appropriate class of sets, it won't return the exact same set on the nose. It may return any of a number of tables which describe the same function in subtly different ways.