Theorems For Free: a Sneak Peek at type theory

1 Our Purpose

Our goal today is to introduce some of the core concepts of type theory, and see how we can apply them in real life. Some of the core concepts we will see today are

- ◇ Syntax/Semantics of type expressions
- ◇ Structural type inductions
- ◇ Types as relations

2 Problem Statement

Write a function with the signature.

```
ident :: forall a. a -> a
```

In haskell, there is an obvious approach

```
ident x = x
```

The key question is, are there any others? What else could you possibly do? Likewise, consider the signature

```
class Functor t where
  fmap :: forall a b. (a -> b) -> t a -> t b
```

What should be true of a functor instance? Given an `fmap`, what laws should it satisfy?

3 Our type language

We won't answer this question for haskell types for several weeks. Instead, I want to do what the Theorems For Free paper does. Let's start by defining a simple type system, and work with set-functions rather than haskell type systems, and haskell terms. This will simplify the problem setting without losing the key idea.

We will consider types that have

- ◇ a primitive Int and $Bool$.
- ◇ products, written (a, b)
- ◇ arrows, written $a \to b$
- ◇ lists, written a^*
- ◇ type parameters, written $\forall a, \mathbf{T}a$

This creates a language of types, $\mathcal{T}$. A one naive semantics would be to interpret the types directly as sets. These are the terms of a given type:

- ◇ $[\![Int]\!] = \mathbf{Z}$, $[\![Int]\!] = \{T, F\}$
- ◇ $[\![(a, b)]\!] = [\![a]\!] \times [\![b]\!]$

- $\diamond$ $[\![a \to b]\!] = [\![b]\!]^{[\![a]\!]}$
- $\diamond$ $[\![a^*]\!] = \bigcup_{n\in\mathbb{N}} [\![a]\!]^n$
- $\diamond$ $[\![\forall a, \mathbf{T}a]\!] = \bigcap_{s\in\mathcal{T}} [\![Ts]\!]$

Sadly this doesn't make sense for the type parameter case.

$$\begin{aligned}
[\![\forall a, a \to a]\!] &= \bigcap_s [\![s \to s]\!] \\
&\subset [\![Int \to Int]\!] \cap [\![Bool \to Bool]\!] \\
&= Int^{Int} \cap Bool^{Bool} \\
&= \emptyset
\end{aligned}$$

But we really want there to be a term in $[\![\forall a, a \to a]\!]$ since its so easy to write one in haskell. We can instead consider ̈polymorphic functions; which in this case means something more like a function over powersets since types are interpreted as sets. The exact details of this is not that important, since when we revisit this with a real type system it will be unambiguous.

So fine, we know what the terms are. But that doesn't help us answer our questions. Instead, we do something terribly clever. We interpret types into a relation.

- $\diamond$ $[\![Int]\!] = \{(x, x) \mid x \in \mathbf{Z}\}$, $[\![Bool]\!] = \{(x, x) \mid x \in \{\mathbf{T}, \mathbf{F}\}\}$,
- $\diamond$ $[\![(a, b)]\!] = \{((a, b), (x, y)) \mid (a, x) \in [\![a]\!] \wedge (b, y) \in [\![b]\!]\}$
- $\diamond$ $[\![a \to b]\!] = \{(f, g) \mid \forall (x, y) \in [\![a]\!], (f(x), g(y)) \in [\![b]\!]\}$
- $\diamond$ $[\![a^*]\!] = \bigcup_{n\in\mathbb{N}} \{([a1, ..., an], [b1, ..., bn]) \mid \forall i, (a_i, b_i) \in [\![a]\!]\}$
- $\diamond$ $[\![\forall a, \mathbf{T}a]\!] = \{(f, g) \mid \forall a, b, R \subset a \times b, (f_a, g_b) \in [\![\mathbf{T}R]\!]\}$

Again, type parameters are the interesting case. We can think of f and g here as taking an extra set as an argument, and the subscript is applying that argument.

The main result we care about is that for all types a, and any term t, we will find

$$t \text{ is a term of } a \iff (t, t) \in [\![a]\!]$$

The proof is straightforward, mostly. The base case are the primitives, true by definition. The case for products and lists follows trivially. For arrows and parameters, consider the following. If f is a term of $a \to b$, then forall $(a, b) \in [\![a]\!]$, then $a = b$. So $(f(a), f(b)) \in b$, and so $f \in [\![a \to b]\!]$.

4 What's next

Ok, so now we know that this relation is, what can we say? Time to turn to the Theorems for free paper