1 Introduction

So far the most interesting thing we have given a denotational semantics for is the **while** loop. What about functions? We now have enough machinery to capture some of their semantics, even for mutually recursive functions. We show how to give a semantics for the language REC, found in Winskel Ch. 9.

2 Denotational Semantics for REC

2.1 REC Syntax

$$
\begin{array}{rcl}
p & ::= & \textbf{let } d \textbf{ in } e \\
d & ::= & f_1(x_1, \ldots, x_{a_1}) = e_1 \\
 & & \vdots \\
 & & f_n(x_1, \ldots, x_{a_n}) = e_n \\
e & ::= & n \mid x \mid e_1 \oplus e_2 \mid \textbf{let } x = e_1 \textbf{ in } e_2 \mid \textbf{ifp } e_0 \textbf{ then } e_1 \textbf{ else } e_2 \mid f_i(e_1, \ldots, e_{a_i})
\end{array}
$$

The functions in d ("declarations") are mutually recursive. It is reasonable to expect that under most semantics, **let** $f_1(x_1) = f_1(x_1)$ **in** $f_1(0)$ will loop infinitely, but **let** $f_1(x_1) = f_1(x_1)$ **in** 0 will halt and return 0.

For example,

```
let
  f1(n,m) = if m^2 - n
            then 1
            else ifp (n-m*(n div m))
                 then f1(n,m+1)
                 then 0
  f2(n) = if f1(n,2) then n else f2(n+1)
in
  f2(1000)
```

In this REC program, $f_2(n)$ finds the first prime number $p \geq n$. (The value of $n - m * (n \textbf{ div } m)$ is positive iff m does not divide n).

2.2 CBV Denotational Semantics for REC

The meaning function is $[\![e]\!] \in \mathit{FEnv} \to \mathit{Env} \to \mathbb{Z}_\perp$, where *FEnv* and *Env* denote the sets of variable environments and function environments, respectively, as used in REC.

$$
\begin{array}{rcl}
\rho & \in & \mathit{Env} = \textbf{Var} \to \mathbb{Z} \\
\phi & \in & \mathit{FEnv} = (\mathbb{Z}^{a_1} \to \mathbb{Z}_\perp) \times \cdots \times (\mathbb{Z}^{a_n} \to \mathbb{Z}_\perp)
\end{array}
$$

Here **Var** is a countable set of variables, $\mathbb{Z}$ is the set of integers, which are the values that can be bound to a variable in an environment, and $\mathbb{Z}^m = \underbrace{\mathbb{Z} \times \mathbb{Z} \times \cdots \times \mathbb{Z}}_{m \text{ times}}$.

$$
\begin{aligned}
[\![n]\!]\phi\rho &= \lfloor n \rfloor \\
[\![x]\!]\phi\rho &= \lfloor \rho(x) \rfloor \\
[\![e_1 \ \oplus \ e_2]\!]\phi\rho &= \text{let } v_1 \in \mathbb{Z} = [\![e_1]\!]\phi\rho \text{ in} \\
&\qquad \text{let } v_2 \in \mathbb{Z} = [\![e_2]\!]\phi\rho \text{ in} \\
&\qquad\quad v_1 \ \oplus \ v_2 \\
&= [\![e_1]\!]\phi\rho \ \ \oplus_\perp \ \ [\![e_2]\!]\phi\rho \\
[\![\textbf{let } x = e_1 \textbf{ in } e_2]\!]\phi\rho &= \text{let } y \in \mathbb{Z} = [\![e_1]\!]\phi\rho \text{ in} \\
&\qquad [\![e_2]\!]\,\phi\,\rho[x \mapsto y] \\
[\![\textbf{ifp } e_0 \textbf{ then } e_1 \textbf{ else } e_2]\!]\phi\rho &= \text{let } v_0 \in \mathbb{Z} = [\![e_0]\!]\phi\rho \text{ in} \\
&\qquad \text{if } v_0 > 0 \text{ then } [\![e_1]\!]\phi\rho \text{ else } [\![e_2]\!]\phi\rho \\
[\![f_i(e_1, \ldots, e_{a_i})]\!]\phi\rho &= \text{let } v_1 \in \mathbb{Z} = [\![e_1]\!]\phi\rho \text{ in} \\
&\qquad \vdots \\
&\qquad \text{let } v_{a_i} \in \mathbb{Z} = [\![e_{a_i}]\!]\phi\rho \text{ in} \\
&\qquad\quad (\pi_i \ \phi)\langle v_1, \ldots, v_{a_i} \rangle
\end{aligned}
$$

The meaning of a program **let** d **in** e is $[\![e]\!]\phi\rho$, where ρ is some initial environment containing default values for the variables, and

$$
\begin{aligned}
\phi \quad = \quad & \mathit{fix}\ \lambda F \in \mathit{FEnv}\,.\,\langle \lambda v \in \mathbb{Z}^{a_1}\,.\,[\![e_1]\!]\,F\,\rho[x_1 \mapsto \pi_1(v), \ldots, x_{a_1} \mapsto \pi_{a_1}(v)], \\
& \qquad \vdots \\
& \lambda v \in \mathbb{Z}^{a_n}\,.\,[\![e_n]\!]\,F\,\rho[x_1 \mapsto \pi_1(v_1), \ldots, x_{a_n} \mapsto \pi_{a_n}(v)]\rangle.
\end{aligned}
$$

For this fixed point to exist, we need to know that *FEnv* is a pointed CPO. But *FEnv* is a product, and a product is a pointed CPO when each factor is a pointed CPO. Each factor $\mathbb{Z}^{a_i} \to \mathbb{Z}_\perp$ is a pointed CPO, since a function is a pointed CPO when the codomain of that function is a pointed CPO, and $\mathbb{Z}_\perp$ is a pointed CPO. Therefore, *FEnv* is a pointed CPO.

We also need to know that the function $\mathit{FEnv} \to \mathit{FEnv}$ to which we are applying *fix* is continuous. It is because is written using the metalanguage.

2.3 CBN Denotational Semantics

The denotational semantics for CBN is the same as for CBV with two exceptions:

$$
\begin{aligned}
[\![\textbf{let } x = e_1 \textbf{ in } e_2]\!]\phi\rho &= [\![e_2]\!]\,\phi\,\rho[x \mapsto [\![e_1]\!]\phi\rho] \\
[\![f_i(e_1, \ldots, e_{a_i})]\!]\phi\rho &= (\pi_i \ \phi)\langle [\![e_1]\!]\phi\rho, \ldots, [\![e_{a_i}]\!]\phi\rho \rangle.
\end{aligned}
$$

In fact, the lazy semantics is shorter than the eager semantics. This is because we had to explicitly "code up" strictness, whereas it came from free in the math representation, since mathematics is happy to ignore nontermination.

2.4 Comments

These semantics tell us something some interesting things about the language. To ensure that we could take a fixed point, we had to make the codomains of the representations of the f_i pointed—we were forced to recognize the possibility of nontermination. This is one of the nice properties of denotational semantics.

Fixed-point semantics also tell us something interesting about compilation. As a rule of thumb, the presence of a fixed point tells us when we will have to build a cyclical data structure within our compiler, and backpatching will be needed to create the cycle. We can see this if we imagine what data structure within the compiler we would need to use to represent the structure ϕ. The code for each of the f_i, which *phi* would point to, in turn points back to ϕ. In a sense, the ability to take a fixed point tells us when we will succeed in backpatching the pointer that makes the cycle before that pointer needs to be used!