1 Denotational Semantics for FL

So far the most interesting thing we have given a denotational semantics for is the while loop. However, we now have enough machinery to capture higher-order constructs such as mutually recursive functions. We show how to give a semantics for a version of the FL language.

1.1 Syntax

We will work with a simplified version of FL similar to the \(\lambda\)-lifted version from Assignment 3.

\[
p ::= \text{letrec } d \text{ in } e
\]
\[
d ::= f(x_1, \ldots, x_n) = e \mid f(x_1, \ldots, x_n) = e \text{ and } d
\]
\[
e ::= n \mid x \mid \text{let } x = e_1 \text{ in } e_2 \mid f(e_1, \ldots, e_n) \mid \text{ifp } e_0 \text{ then } e_1 \text{ else } e_2
\]
\[
| e_1 + e_2 | \ldots \text{ (other arithmetic operators)}
\]

The syntactic constructs defined by \(d\) are mutually recursive function declarations. These occur at the outermost level only. The conditional test \(\text{ifp}-\text{then}-\text{else}\) expects a number instead of a Boolean for its first argument, and the test succeeds if that number is positive.

For example,

\[
\text{letrec } f_1(n, m) = \text{ifp } m^2 - n \text{ then } 1 \text{ else } (n \text{ mod } m) \cdot f_1(n, m + 1)
\]
\[
\text{and } f_2(n) = \text{ifp } f_1(n, 2) \text{ then } n \text{ else } f_2(n + 1)
\]
\[
in f_2(1000)
\]

In this program, \(f_2(n)\) finds the first prime number \(p \geq n\). The value of \(n \text{ mod } m\) is positive iff \(m\) does not divide \(n\).

1.2 CBV Denotational Semantics for REC

We will interpret an expression \(e\) as a function is \(\llbracket e \rrbracket \in FEnv \rightarrow Env \rightarrow \mathbb{Z}_\perp\), where \(Env\) and \(FEnv\) denote the sets of variable environments and function environments, respectively.

\[
\rho \in Env = \text{Var} \rightarrow \mathbb{Z} \quad \varphi \in FEnv = (\mathbb{Z}^{n_1} \rightarrow \mathbb{Z}_\perp) \times \cdots \times (\mathbb{Z}^{n_k} \rightarrow \mathbb{Z}_\perp)
\]

Here \(\text{Var}\) and \(FVar\) are disjoint countable sets of variables, \(\mathbb{Z}\) is the set of integers, and \(\mathbb{Z}^n = \mathbb{Z} \times \mathbb{Z} \times \cdots \times \mathbb{Z}\) \(n\) times.
\[ n \varphi \rho \triangleq n \]
\[ x \varphi \rho \triangleq \rho(x) \]
\[ (e_1 + e_2) \varphi \rho \triangleq [e_1] \varphi \rho + ^\dagger [e_2] \varphi \rho \quad \text{(similarly for other arithmetic operators)} \]
\[ \text{let } x = e_1 \text{ in } e_2 \varphi \rho \triangleq \text{let } v \in \mathbb{Z} = [e_1] \varphi \rho \text{ in } [e_2] \varphi \rho [v/x] \]
\[ \text{ifp } e_0 \text{ then } e_1 \text{ else } e_2 \varphi \rho \triangleq \text{let } v_0 \in \mathbb{Z} = [e_0] \varphi \rho \text{ in } \begin{cases} e_1 \varphi \rho & \text{if } v_0 > 0 \\ e_2 \varphi \rho & \text{else} \end{cases} \]
\[ f_1(e_1, \ldots, e_n) \varphi \rho \triangleq \text{let } v_1 \in \mathbb{Z} = [e_1] \varphi \rho \text{ in } \begin{cases} \vdots & \\ \text{let } v_n \in \mathbb{Z} = [e_n] \varphi \rho \text{ in } (\pi_n \varphi)(e_1, \ldots, e_n) \end{cases} \]

In the definition of \([e_1 + e_2] \varphi \rho\), the symbol \( + ^\dagger \) refers to the lifted version of addition on \( \mathbb{Z} \). This function takes the value \( \bot \) if either of its arguments is \( \bot \), otherwise returns the sum of its arguments.

The meaning of a program letrec \( d \) in \( e \) is
\[ [\text{letrec } d \text{ in } e] \triangleq [e] \varphi \rho_0, \]
where \( \rho_0 \) is some initial environment containing default values for the variables (say 0), and if the function declarations \( d \) are
\[
\begin{align*}
f_1(x_1, \ldots, x_{n_1}) &= e_1 \quad \text{and} \quad \ldots \quad \text{and} \quad f_k(x_1, \ldots, x_{n_k}) &= e_k,
\end{align*}
\]
then
\[
\varphi = \text{fix } \lambda \psi \in FEnv. \left( \lambda v_1 \in \mathbb{Z}, \ldots, v_{n_1} \in \mathbb{Z}. [e_1] \psi \rho_0 [v_1/x_1, \ldots, v_{n_1}/x_{n_1}], \right.
\]
\[
\vdots
\]
\[
\left. \lambda v_n \in \mathbb{Z}, \ldots, v_{n_k} \in \mathbb{Z}. [e_k] \psi \rho_0 [v_1/x_1, \ldots, v_{n_k}/x_{n_k}], \right)
\]
or more accurately,
\[
\varphi = \text{fix } \lambda \psi \in FEnv. \left( \lambda v \in \mathbb{Z}^{n_1}. [e_1] \psi \rho_0 [\pi_1(v)/x_1, \ldots, \pi_{n_1}(v)/x_{n_1}], \right.
\]
\[
\vdots
\]
\[
\left. \lambda v \in \mathbb{Z}^{n_k}. [e_k] \psi \rho_0 [\pi_1(v)/x_1, \ldots, \pi_{n_k}(v)/x_{n_k}], \right)
\]

For this fixpoint to exist, we need to know that \( FEnv \) is a pointed CPO and that the function \( FEnv \rightarrow FEnv \) to which we are applying fix is continuous. The domain \( FEnv \) is a product, and a product is a pointed CPO when each factor is a pointed CPO. Each factor \( \mathbb{Z}^{n_i} \rightarrow \mathbb{Z}_\bot \) is a pointed CPO, since a function is a pointed CPO when the codomain of that function is a pointed CPO, and \( \mathbb{Z}_\bot \) is a pointed CPO. Therefore, \( FEnv \) is a pointed CPO.

The function \( \tau : FEnv \rightarrow FEnv \) to which we are applying fix is continuous, because it can be written using the metalanguage. Here is the argument. We illustrate with \( k = 2 \) and \( n_1 = n_2 = 1 \) for simplicity, thus we assume the declaration \( d \) is
\[
f_1(x) = e_1 \quad \text{and} \quad f_2(x) = e_2.
\]
Then

\[ \varphi = \text{fix } \lambda \psi \in FEnv. (\lambda v \in \mathbb{Z} \cdot \llbracket e_1 \rrbracket \psi \rho_0[v/x], \lambda v \in \mathbb{Z} \cdot \llbracket e_2 \rrbracket \psi \rho_0[v/x]) \].

This gives the least fixpoint of the operator

\[ \tau = \lambda \psi \in FEnv. (\lambda v \in \mathbb{Z} \cdot \llbracket e_1 \rrbracket \psi \rho_0[v/x], \lambda v \in \mathbb{Z} \cdot \llbracket e_2 \rrbracket \psi \rho_0[v/x]) \]

provided we can show that \( \tau \) is continuous. We can write

\[
\begin{align*}
\tau &= \lambda \psi \in FEnv. (\lambda v \in \mathbb{Z} \cdot \llbracket e_1 \rrbracket \psi \rho_0[v/x], \lambda v \in \mathbb{Z} \cdot \llbracket e_2 \rrbracket \psi \rho_0[v/x]) \\
&= \lambda \psi \in FEnv. (\tau_1(\psi), \tau_2(\psi)) \\
&= \lambda \psi \in FEnv. \langle \tau_1, \tau_2 \rangle (\psi) \\
&= \langle \tau_1, \tau_2 \rangle,
\end{align*}
\]

where \( \tau_i : FEnv \rightarrow FEnv \) is

\[ \tau_i = \lambda \psi \in FEnv. \lambda v \in \mathbb{Z} \cdot \llbracket e_i \rrbracket \psi \rho_0[v/x] \].

Because \( \langle \tau_1, \tau_2 \rangle \) is continuous iff \( \tau_1 \) and \( \tau_2 \) are, it suffices to show that each \( \tau_i \) is continuous. Now we can write \( \tau_i \) in our metalanguage.

\[
\begin{align*}
\tau_i &= \lambda \psi \in FEnv. \lambda v \in \mathbb{Z} \cdot \llbracket e_i \rrbracket \psi \rho_0[v/x] \\
&= \lambda \psi \in FEnv. \lambda v \in \mathbb{Z} \cdot \llbracket e_i \rrbracket \psi (\text{subst } \rho_0 x v) \\
&= \lambda \psi \in FEnv. \lambda v \in \mathbb{Z} \cdot (\llbracket e_i \rrbracket \psi) ((\text{subst } \rho_0 x) v) \\
&= \lambda \psi \in FEnv. \lambda v \in \mathbb{Z} \cdot ((\llbracket e_i \rrbracket \psi) \circ \text{subst } \rho_0 x) v \\
&= \lambda \psi \in FEnv. \text{compose } ((\llbracket e_i \rrbracket \psi, \text{subst } \rho_0 x) \psi) \\
&= \lambda \psi \in FEnv. \text{compose } ((\llbracket e_i \rrbracket, \text{const } \text{subst } \rho_0 x) \psi) \\
&= \lambda \psi \in FEnv. \text{compose } ((\llbracket e_i \rrbracket, \text{const } \text{subst } \rho_0 x)) \psi \\
&= \text{compose } \circ (\llbracket e_i \rrbracket, \text{const } \text{subst } \rho_0 x).
\end{align*}
\]

Now we can argue that \( \tau_1 \) is continuous. The composition of two continuous functions is continuous, so it suffices to know that \text{compose} and \( (\llbracket e_i \rrbracket, \text{const } \text{subst } \rho_0 x) \) are continuous. We argued last time that \text{compose} is continuous. To show \( (\llbracket e_i \rrbracket, \text{const } \text{subst } \rho_0 x) \) is continuous as a function, it suffices to show that both \( \llbracket e_i \rrbracket \) and \text{const } \text{subst } \rho_0 x are continuous as functions. The former is continuous by the induction hypothesis (structural induction on \( e \)). The latter is a constant function on a discrete domain and is therefore continuous.

### 1.3 CBN Denotational Semantics

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

\[
\begin{align*}
\llbracket \text{let } x = e_1 \text{ in } e_2 \rrbracket \varphi & \triangleq \llbracket e_2 \rrbracket \varphi \rho [\llbracket e_1 \rrbracket \varphi \rho / x] \\
\llbracket f_i(e_1, \ldots, e_n) \rrbracket \varphi & \triangleq (\pi_i \varphi)(\llbracket e_1 \rrbracket \varphi \rho, \ldots, \llbracket e_n \rrbracket \varphi \rho).
\end{align*}
\]

We must extend environments and function environments:

\[
\begin{align*}
\text{Env} &= \text{Var} \rightarrow \mathbb{Z}_\perp & \text{FEnv} &= (\mathbb{Z}_\perp^{n_1} \rightarrow \mathbb{Z}_\perp) \times \cdots \times (\mathbb{Z}_\perp^{n_k} \rightarrow \mathbb{Z}_\perp).
\end{align*}
\]