CS611 Lecture 19 **Call by Name Semantics of REC** 10 October 2003

Scribe: Pavel Dmitriev Lecturer: Radu Rugina

1 Review

Last time we started talking about a new language, REC, which includes functions. We mentioned that there are two types of semantics available for this language, depending on how we pass parameters to the functions: Call By Value (CBV), and Call By Name (CBN). We also defined CBV Denotational and Operational semantics for REC.

On the last lecture, there was some confusion about whether we can express the **if** $(x \leq y)$ **then** e_0 **else** e_1 construct in our language. The answer is yes, we can do that. This construct is equivalen to

$$\textbf{ifz } g(y-x, x-y) \textbf{ then } e_0 \textbf{ else } e_1$$

where

$$\begin{aligned} g(u,v) \quad &= \quad \textbf{ifz } u \textbf{ then } 0 \textbf{ else} \\ & \qquad \textbf{ifz } v \textbf{ then } 1 \textbf{ else } g(u-1, v-1) \end{aligned}$$

Let us summarize briefly the semantics that we defined on the last lecture.

1.1 CBV Operational Semantics

Almost all the rules for the CBV large step operational semantics for REC were similar to the ones we defined earlier for IMP. The only interesting one was the rule for functions:

$$\frac{e_{i1} \to n_1, \ldots, e_{ia_i} \to n_{a_i} \;\; e_{f_i}[x_{i1} \to n_{i1}, \ldots, x_{ia_i} \to n_{ia_i}] \to n}{f_i(e_{i1}, \ldots, e_{ia_i}) \to n}$$

The formal parameters are replaced with the values of the e_{ij}. So, we first evaluate the parameters, and then apply the body of the function.

1.2 CBV Denotational Semantics

We defined our denotation for e as

$$[\![\mathrm{e}]\!] : \mathbf{Env} \to \mathbf{FEnv} \to \mathbb{Z}_\perp$$

where

$$\begin{aligned} \mathbf{Env} \quad &: \quad \mathbf{Var} \to \mathbb{Z} \\ \mathbf{FEnv} \quad &: \quad (\mathbb{Z}^{a_1} \to \mathbb{Z}_\perp) \times \cdots \times (\mathbb{Z}^{a_k} \to \mathbb{Z}_\perp) \end{aligned}$$

We inductively defined the denotation for expressions, basing on their structure:

$$\begin{aligned} [\![n]\!]\rho\varphi \quad &= \quad \lfloor n \rfloor \\ [\![x]\!]\rho\varphi \quad &= \quad \lfloor \rho(x) \rfloor \\ [\![e_0 \;\oplus\; e_1]\!]\rho\varphi \quad &= \quad [\![e_0]\!]\rho\varphi \;\oplus_\perp\; [\![e_1]\!]\rho\varphi \end{aligned}$$

$$where \;\; n_1 \;\oplus_\perp\; n_2 = \begin{cases} \lfloor n_1' \oplus n_2' \rfloor, \ \textbf{if } n_1 = \lfloor n_1' \rfloor, \ n_2 = \lfloor n_2' \rfloor \\ \perp, \ \textbf{if } n_1 = \perp \text{ or } n_2 = \perp \end{cases}$$

$$[\![\textbf{ifz } e \textbf{ then } e_0 \textbf{ else } e_1]\!]\rho\varphi \;\;=\;\; \textbf{condnz}([\![e]\!]\rho\varphi, [\![e_0]\!]\rho\varphi, [\![e_1]\!]\rho\varphi)$$

$$where \;\; \textbf{condnz} \;:\; (\mathbb{Z}_\perp \times \mathbb{Z}_\perp \times \mathbb{Z}_\perp) \to \mathbb{Z}_\perp \quad \textbf{condnz}(n_0, n_1, n_2) \;=\; [f_1, f_2]^*(n_0)$$

$$\begin{array}{rcl} f_1 & : & 0 \to \mathbb{Z}_\perp \quad f_1(0) = n_1 \\ f_2 & : & \{n, \ n \neq 0\} \to \mathbb{Z}_\perp \quad f_2(n) = n_2 \end{array}$$

$$[\![f_i(e_{i1}, \ldots, e_{ia_i})]\!]\rho\varphi = \textbf{let } n_{i1} \Leftarrow [\![e_{i1}]\!]\rho\varphi, \ldots, n_{ia_i} \Leftarrow [\![e_{ia_i}]\!]\rho\varphi \;.\; \varphi_i(n_{i1}, \ldots, n_{ia_i})$$

Here we used the **let** construct, to be consistent with the textbook. If one of the arguments of **let** evaluates to $\perp$, then the whole construct evaluates to $\perp$.

$$\textbf{let } x \Leftarrow e_1.e \;\;\equiv\;\; (\lambda x.e)^*(e_1)$$

It can be shown that let is continuous.

Now, let us move to the CBN semantics for REC.

2 Call-by-Name Semantics for REC

2.1 Operational Semantics

All the rules for CBN large step operational semantics will be the same as for CBV semantics, except the rule for functions. Here we do not evaluate the parameters, we just pass them to the body of the function, so that they are evaluated when and if they are needed. The rule for functions looks as follows:

$$\frac{e_{f_i}[x_{i1} \to e_{i1}, \ldots, x_{ia_i} \to e_{ia_i}] \to n}{f_i(e_{i1}, \ldots, e_{ia_i}) \to n}$$

2.2 Denotational Semantics

What we have here is very similar to what we had for CBV semantics. The main difference is, again, in the function rule. Also, because we pass unevaluated parameters to functions, we may be passing bottom as well (imagine a function that takes as a parameter another function that might not terminate). Thus, our denotation now is

$$[\![\mathrm{e}]\!] : \textbf{Env} \to \textbf{FEnv} \to \mathbb{Z}_\perp$$

where

$$\begin{array}{rcl} \textbf{Env} & : & \textbf{Var} \to \mathbb{Z}_\perp \\ \textbf{FEnv} & : & (\mathbb{Z}_\perp^{a_1} \to \mathbb{Z}_\perp) \times \cdots \times (\mathbb{Z}_\perp^{a_k} \to \mathbb{Z}_\perp) \end{array}$$

The denotations for expressions are as follows:

$$\begin{array}{rcl} [\![n]\!]\rho\varphi & = & \lfloor n \rfloor \\ [\![x]\!]\rho\varphi & = & \rho(x) \\ [\![e_0 \;\oplus\; e_1]\!]\rho\varphi & = & [\![e_0]\!]\rho\varphi \;\oplus_\perp\; [\![e_1]\!]\rho\varphi \\ [\![\textbf{ifz } e \textbf{ then } e_0 \textbf{ else } e_1]\!]\rho\varphi & = & \textbf{condnz}([\![e]\!]\rho\varphi, [\![e_0]\!]\rho\varphi, [\![e_1]\!]\rho\varphi) \\ [\![f_i(e_{i1}, \ldots, e_{ia_i})]\!]\rho\varphi & = & \varphi_i([\![e_{i1}]\!]\rho\varphi, \ldots, [\![e_{ia_i}]\!]\rho\varphi) \end{array}$$

Note that we do not need to lift the result of evaluation of X in **Var**. Also note that though we used the same $[\![.]\!]$ symbol for describing both CBN and CBV denotational semantics, they are two different systems of rules. Ideally, we should have used explicit subscripts, like $[\![.]\!]_{cbv}$ and $[\![.]\!]_{cbn}$, to indicate which semantics we are using.

Now, the question is how do we construct the function environment **FEnv**? Our definitions are a system of recursive equations. Evaluation of a function involves evaluation of its arguments, which can, in turn, involve evaluation of the same function. We see a situation similar to the one we had with the while loop in IMP. We will use the same mechanism - fixed points - to solve our problem with REC. We will define a higher order function that constructs successive approximations of the recursive equation, and will take the least fixed point of it. We will only do it for CBN semantics, the recursive system for CBV can be solved similarly.

$$F \ : \ \mathbf{FEnv} \to \mathbf{FEnv}$$
$$F((\varphi_1, \ldots, \varphi_k)) \ = \ (\varphi'_1, \ldots, \varphi'_k)$$

How do we define φ'_i? Recall that our program looks as follows.

$$\begin{gathered} f_1(x_{11}, \ldots, x_{1a_1}) = e_{f_1} \\ \ldots \\ f_k(x_{k1}, \ldots, x_{ka_k}) = e_{f_k} \\ e \end{gathered}$$

We will define φ'_i as follows:

$$\begin{aligned} \varphi'_1 \quad &= \quad \lambda n_{11}, \ldots, n_{1a_1} \in \mathbb{Z}_\perp \ . \ [\![e_{f_1}]\!]_{cbn}\rho[x_{11} \to n_{11}, \ldots, x_{1a_1} \to n_{1a_1}]\varphi \\ &\quad\ \ \ldots \\ \varphi'_k \quad &= \quad \lambda n_{k1}, \ldots, n_{ka_k} \in \mathbb{Z}_\perp \ . \ [\![e_{f_k}]\!]_{cbn}\rho[x_{k1} \to n_{k1}, \ldots, x_{ka_k} \to n_{ka_k}]\varphi \end{aligned}$$

Function F is continuous, so it has the list fixed point: $\varphi^{fix} \ = \ fix(F)$. For executing the program, we evaluate e using this fixed point as the function environment:

$$[\![e]\!]_{cbn}\rho\varphi^{fix}$$

3 Example

Let us consider the following example program:

$$\begin{aligned} f(x) \quad &= \quad f(x) + 1 \\ g(y) \quad &= \quad 2 \\ g(f(3))& \end{aligned}$$

Intuitively, we expect that evaluating it under CBV semantics will not terminate, and that evaluation of CBN semantics will terminate, and return 2. We will only consider CBN here, evaluating this example under CBV semantics is left as an excercise.

First, we need to construct our function $F \ : \ \mathbf{FEnv} \to \mathbf{FEnv}$. In our case, $\mathbf{FEnv} \ = \ (\mathbb{Z}_\perp \to \mathbb{Z}_\perp) \times (\mathbb{Z}_\perp \to \mathbb{Z}_\perp)$.

$$F(\underbrace{(\varphi_1, \varphi_2)}_{\varphi}) \ = \ (\varphi'_1, \varphi'_2)$$

$$\begin{array}{rcl}
\varphi_1'(n) & = & [\![f(x)\ +\ 1]\!]\rho[x \rightarrow n]\varphi\ = \\
& = & [\![f(x)]\!]\rho[x \rightarrow n]\varphi\ +_\perp\ \lfloor 1 \rfloor\ = \\
& = & \varphi_1(n)\ +_\perp\ \lfloor 1 \rfloor \\
\varphi_2'(n) & = & [\![2]\!]\rho[y \rightarrow n]\varphi\ =\ 2
\end{array}$$

Hence, writing φ functions vertically, we have:

$$F \left(\begin{array}{c} \lambda n.\varphi_1(n) \\ \lambda m.\varphi_2(m) \end{array} \right) = \left(\begin{array}{c} \lambda n.\varphi_1(n)\ +_\perp\ \lfloor 1 \rfloor \\ \lambda m.\lfloor 2 \rfloor \end{array} \right)$$

It is easy to see that the list fixed point of this function is

$$\varphi^{fix} = \left(\begin{array}{c} \lambda n.\perp \\ \lambda m.\lfloor 2 \rfloor \end{array} \right)$$

Now we can evaluate $g(f(3))$.

$$\begin{array}{rcl}
[\![g(f(3))]\!]\rho\varphi^{fix} & = & \varphi_2^{fix}([\![f(3)]\!]\rho\varphi^{fix})\ = \\
& = & \varphi_2^{fix}(\varphi_1^{fix}(\lfloor 3 \rfloor))\ = \\
& = & \varphi_2^{fix}(\perp)\ =\ \lfloor 2 \rfloor
\end{array}$$