Scribe: Adam Barth and Stanislaw Dziobiak Lecturer: Radu Rugina

1 Syntax

The syntax of the *REC* language is built from the following syntactic domains:

- Integers, $n \in \mathbb{Z}$
- Variables, $v \in Var$
- Function variables, $f \in Fvar$

1.1 Grammar

A program, p, is built using the following grammar:

$$
\begin{array}{rcl}
p & ::= & de \\
d & ::= & f(x_{1,1}, \ldots, x_{1,a_1}) = e_{f_1} \\
 & & \ldots \\
 & & f(x_{k,1}, \ldots, x_{k,a_k}) = e_{f_k} \\
e & ::= & n \;\mid\; x \;\mid\; e_0 + e_1 \;\mid\; e_0 - e_1 \;\mid\; e_0 * e_1 \;\mid\; e_0 \leq e_1 \\
 & & \;\mid\; \mathit{ifnz}\ e\ \mathit{then}\ e_0\ \mathit{else}\ e_1 \;\mid\; f_i(e_{i,1}, \ldots, e_{i,a_i})
\end{array}
$$

Where $e_0 \leq e_1$ is understood to have the value 1 when it is the case that e_0 evaluates to a value less than or equal to the value to which e_1 evaluates and 0 otherwise. $\mathit{ifnz}\ e\ \mathit{then}\ e_0\ \mathit{else}\ e_1$ is understood to evaluate to the value of e_0 if e evaluates to a non-zero integer and e_1 otherwise.

1.2 Examples

The following are examples of programs in the *REC* language:

(a) Computes the max of x_1, x_2, x_3, and x_4:
$f_1(x, y) = \mathit{ifnz}\ x \leq y\ \mathit{then}\ y\ \mathit{else}\ x$
$f_1(f_1(x_1, x_2), f_1(x_3, x_4))$

(b) Computes the factorial of 10:
$\mathit{fact}(n) = \mathit{ifnz}\ n \leq 1\ \mathit{then}\ 1\ \mathit{else}\ n * \mathit{fact}(n-1)$
$\mathit{fact}(10)$

(c) Computes the greatest common divisor of 21 and 14:
$f_1'(x, y) = \mathit{ifnz}\ x \leq y\ \mathit{then}\ x\ \mathit{else}\ y$
$f_2(x', y') = \mathit{ifnz}\ x' \leq y'\ \mathit{then}\ y' - x'\ \mathit{else}\ x' - y'$
$f_3(x'', y'') = \mathit{ifnz}\ x'' - y''\ \mathit{then}\ f_3(f_1'(x'', y''), f_2(x'', y''))\ \mathit{else}\ x''$
$f_3(21, 14)$

2 Semantics

There is a choice of two different semantics for the language *REC*. The issue is how we handle parameter passing. We can either evaluate each parameter to a function and then invoke the function on the values, or we can pass each of the parameters as pieces of syntax, to be evaluated later. The first approach is known as call-by-value (CBV) whereas the second approach is known as call-by-name (CBN).

In an imperative language, the method of parameter passing has a major effect on the semantics of the language. Evaluating expressions at different times can result in different values because the values of the variables change over time. In our side-effect free language, the choice of parameter passing may also affect semantics. Consider the following program:

$$\begin{array}{rcl} f(x) & = & f(x)+1 \\ g(y) & = & 1 \\ g(f(1)) & & \end{array}$$

With call-by-value semantics, the program will not terminate. However, with call-by-name semantics, the program will terminate as f will never be evaluated.

2.1 Large-Step Operational Semantics for Call-by-Value

Our operational semantics will take the form:

$$e \to n$$

where $e \in \mathit{Expr}$ and $n \in \mathbb{Z}$. We define $\to$ inductively with inference rules.

$$\frac{}{n \to n} \qquad \frac{e_0 \to n_0 \quad e_1 \to n_1}{e_0 \ \mathit{op} \ e_1 \to n} \ \text{where } n = n_0 \ \mathit{op} \ n_1 \text{ and } \mathit{op} \in \{+,-,* \le\}$$

$$\frac{e \to n \quad e_0 \to n_0}{\mathit{ifnz} \ e \ \mathit{then} \ e_0 \ \mathit{else} \ e_1 \to n_0} \qquad \frac{e \to 0 \quad e_1 \to n_1}{\mathit{ifnz} \ e \ \mathit{then} \ e_0 \ \mathit{else} \ e_1 \to n_1} \ \text{where } n \neq 0$$

$$\frac{e_{i,1} \to n_1 \quad \ldots \quad e_{i,a_i} \to n_{a_i} \quad e[x_{i,1} \mapsto n_1]\ldots[x_{i,a_i} \mapsto n_{a_i}] \to n}{f_i(e_{i,1},\ldots,e_{i,a_i}) \to n}$$

There are no rules for evaluating variables because we assume all variables are bound. In this case we never have to evaluate a variable by itself.

2.2 Denotational Semantics for Call-by-Value

For denotational semantics, we introduce an environment construct that can be taken to represent the execution stack. We have $\mathit{Env} = \mathit{Var} \to \mathbb{Z}$. For an expression e, its denotation is $[\![e]\!] \in \mathit{Env} \to \mathbb{Z}_\perp$. We define this denotation as follows (here $\rho \in \mathit{Env}$ is an environment for variables):

$$\begin{array}{rcl} [\![n]\!]\rho & = & \lfloor n \rfloor \\ [\![x]\!]\rho & = & \lfloor \rho(x) \rfloor \\ [\![e_0 \ \mathit{op} \ e_1]\!]\rho & = & \mathit{op}_\perp([\![e_0]\!]\rho, [\![e_1]\!]\rho), \ \text{where } \mathit{op} \in \{+,*,-,\le\} \end{array}$$

Here we understand

$$\mathit{op}_\perp(n_1,n_2) = \begin{cases} \lfloor n_0' \ \mathit{op} \ n_1' \rfloor & \text{if } n_0 = \lfloor n_0' \rfloor \text{ and } n_1 = \lfloor n_1' \rfloor \\ \perp & \text{otherwise} \end{cases}$$

The rules for ifnz and f_i are more complex.

$$\begin{array}{rcl} [\![\mathit{ifnz} \ e \ \mathit{then} \ e_0 \ \mathit{else} \ e_1]\!]\rho & = & \begin{cases} [\![e_0]\!]\rho & \text{if } [\![e]\!]\rho = \lfloor n \rfloor \text{ and } n \neq 0 \\ [\![e_1]\!]\rho & \text{if } [\![e]\!]\rho = \lfloor 0 \rfloor \\ \perp & \text{otherwise} \end{cases} \\ [\![f_i(e_{i,1},\ldots,e_{i,a_i})]\!]\rho & = & \begin{cases} [\![e_{f_i}[x_{i,1} \mapsto [\![e_{i,1}]\!]\rho]\ldots[x_{i,a_i} \mapsto [\![e_{i,a_i}]\!]\rho]]\!]\rho & \text{all } [\![e_{i,k}]\!]\rho \neq \perp \\ \perp & \text{otherwise} \end{cases} \end{array}$$

We claim $[\![.]\!]$ is continuous. Most of the proof is routine, however there are a couple interesting points.

- First, we need to show that the functions $op_\perp$ are continuous. This can be accomplished by noting that
$$op_\perp(n_1, n_2) = \mathit{let}\ (n_1' = n_1)(n_2' = n_2)\ .\ n_1'\ op\ n_2'$$
if we understand $\mathit{let}\ (x = e)\ .\ e'$ to be $(\lambda x.e')^*(e)$.

- We might also worry about the *ifnz* case. Recall from our study of *IMP* the function $\mathit{cond} \in (\mathbb{T} \times E \times E) \to E$. We define a similiar function $\mathit{condnz} \in (\mathbb{Z}_\perp \times \mathbb{Z}_\perp \times \mathbb{Z}_\perp) \to \mathbb{Z}_\perp$.
$$f_{n_1} \in \{n \mid n \neq 0\} \to \mathbb{Z}_\perp,\ f_{n_1}(n) = n_1$$
$$f_{n_2} \in \{0\} \to \mathbb{Z}_\perp,\ f_{n_2}(0) = n_2$$
Now we have
$$[f_{n_1}, f_{n_2}](n) = \begin{cases} n_1 & \text{if } n \neq 0 \\ n_2 & \text{if } n = 0 \end{cases}$$
However, this does not handle $\perp$. We define $\mathit{condnz}(n, n_1, n_2) = [f_{n_1}, f_{n_2}]^*(n)$ which is continuous.

- For function application, we note
$$[\![f_i(e_{i,1}, \ldots, e_{i,a_i})]\!]\rho = \mathit{let}\ (n_1 = [\![e_{i,1}]\!]\rho) \ldots (n_{a_i} = [\![e_{i,a_i}]\!]\rho)\ .\ [\![e_{f_i}[x_{i,1} \mapsto n_{i,1}] \ldots [x_{i,a_i} \mapsto n_{i,a_i}]]\!]\rho$$

We can then use a routine proof by structural induction to show that $[\![e]\!]$ is continuous for any e.

2.3 Determinism

By determinism, we mean the question
$$\forall e, \rho, n, n'\ .\ ([\![e]\!]\rho = n\ \wedge\ [\![e]\!]\rho = n') \longrightarrow n = n'$$
The proof of determinism is left as an exercise.