Scribe: Jonathan Kaldor, Ian Kash Lecturer: Andrew Myers

1 Introduction

In this lecture we discuss the denotational semantics of REC+ (a language introduced in the previous lecture) and uML

2 Denotational Semantics of REC+

Although REC+ was introduced in the last lecture, we will first review the structure of a program in the language:

$$\begin{aligned}
\textit{program} &::= d \;\mathsf{in}\; e \\
d &::= f_1(x_1, ...x_{a_1}) = e_1 \;...\; f_n(x_1, ...x_{a_n}) = e_n \\
e &::= n \mid x \mid e_1 + e_2 \mid \mathsf{ifp}\; e_0 \;\mathsf{then}\; e_1 \;\mathsf{else}\; e_2 \mid \mathsf{let}\; x = e_1 \;\mathsf{in}\; e_2 \mid f_i(e_1, ...e_{a_i})
\end{aligned}$$

As in all denotational semantics, we begin by assigning reasonable domains to language features. Since all functions in REC+ exist at the top level of the program, we will need a domain *FEnv* to represent all of the functions in a given program. It is quite natural to think of an element of this domain as being a tuple of a_n functions, each taking a_i *Arg*s to a *Result*, where *Arg* and *Result* are domains that we have not yet defined but whose purpose should be obvious. Thus, we have

$$\phi \in \mathit{FEnv} = (\mathit{Arg}^{a_1} \to \mathit{Result}) \times ... \times (\mathit{Arg}^{a_n} \to \mathit{Result})$$

We will also need a domain *Env* which will contain all naming environments ρ. Again, an element of this domain should be a function that maps a *Var* to an *Arg*. It follows that

$$\rho \in \mathit{Env} = \mathit{Var} \to \mathit{Arg}$$

As for the definition of the domain *Arg*, it depends on whether we choose to have REC+ be a CBV or CBN language. If it is CBV, then every argument to a function must terminate to a value, so $\mathit{Arg} = \mathbb{Z}$. If it is CBN, however, argument expressions don't necessarily need to terminate, and so $\mathit{Arg} = \mathbb{Z}_\perp$. In all cases, the domain $\mathit{Result} = \mathbb{Z}_\perp$.

Finally, we need two functions to produce the actual denotational semantics for a program:

$$\begin{aligned}
\mathcal{D}[\![d]\!] &: \mathit{FEnv} \\
\mathcal{E}[.] &: \mathit{Exp}_{REC+} \to \mathit{FEnv} \to \mathit{Env} \to \mathit{Result}
\end{aligned}$$

Thus, the denotational semantics of a program $p = d \;\mathsf{in}\; e$ is equal to:

$$\mathcal{E}[\![e]\!]\mathcal{D}[\![d]\!]\ (\lambda x \in \mathit{Var}.\ 0)$$

2.1 Changing the properties of REC+

Depending on how $\mathcal{D}$ is defined, the properties of the language REC+ change:

Eagerness

A Call by value

B Call by name

Function scope

1 Only in later functions

2 In self and later

3 Everywhere

This leads to six total combinations of properties. Before we begin, recall that $\mathcal{D}[\![d]\!] = \langle F_1, ...F_n \rangle$.

A1 and B1

Define each F_i such that

$$F_i = \lambda y \in \mathbb{Z}, ...y_{a_i} \in \mathbb{Z}.\ \mathcal{E}[\![e_i]\!]\langle F_1, ...F_{i-1}\rangle \rho_0[x_1 \mapsto y_1, ...x_{a_i} \mapsto y_{a_i}]$$

Since nontermination is impossible (there is no recursion), the domain of *Result* can be changed to $\mathit{Result} = \mathbb{Z}$. As a result, this language would not be Turing complete. This definition holds for both the CBN and CBV variations **A1** and **B1**

A2 and B2

We would like F_i to be defined such that:

$$F_i = \lambda y_1 \in \mathbb{Z}, ..., y_{a_i} \in \mathbb{Z}.\mathcal{E}[\![e_i]\!]\langle F_1, ...F_{i-1}, f\rangle \rho_0[x_j \mapsto y_j]$$

where $f = F_i$. This implies that we need to take a fixed point of F_i:

$$F_i = \mathit{fix}\ \lambda f \in \mathit{Arg}^{a_i} \to \mathit{Result}.\ \lambda y_1 \in \mathbb{Z}, ...y_{a_i} \in \mathbb{Z}.\ \mathcal{E}[\![e_i]\!]\langle F_1, ...F_{i-1}, f\rangle \rho_0[x_j \mapsto y_j]$$

However, $\mathit{Arg}^{a_i} \to \mathit{Result}$ must be pointed for this to be valid. Additionally, since nontermination is now possible, $\mathit{Result} = \mathbb{Z}_\perp$. This translation is identical for CBN, except that $\mathit{Arg} = \mathbb{Z}_\perp$.

A3 and B3

$$\begin{aligned} \mathcal{D}[\![d]\!] &= \langle F_1, ...F_n\rangle \\ &= \mathit{fix}\ \lambda \phi \in \mathit{FEnv}.\ \langle \lambda y_1 ... y_{a_i} \in \mathbb{Z}.\mathcal{E}[\![e_1]\!]\ \phi\ \rho_0[x_j \mapsto y_j],\ ...\ \lambda y_1 ... y_{a_n} \in \mathbb{Z}.\mathcal{E}[\![e_n]\!]\ \phi\ \rho_0[x_j \mapsto y_j]\rangle \end{aligned}$$

Since the codomain of each function F_i is $\mathbb{Z}_\perp$, it follows that each F_i is pointed, and thus so is their cross product, and thus *FEnv* is a CPO. This translation is identical for CBN, except $\mathit{Arg} = \mathbb{Z}_\perp$ instead of $\mathbb{Z}$

3 Denotational Semantics of UML

Recall the definition of UML:

$$\begin{aligned} e ::=\ & n \mid x \mid e_1 + e_2 \mid \lambda x.e \mid e_0\ e_1 \mid \mathsf{true} \mid \mathsf{false} \mid \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 \mid \#n\ e \mid \\ & (e_1, ..., e_n) \mid \mathsf{letrec}\ f_1 = \lambda x_1.\ e_1\ ...\ f_n = \lambda x_n.\ e_n\ \mathsf{in}\ e \mid \mathsf{if}\ e_0\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \end{aligned}$$

As above, we need to come up with definitions for the domains of this language. An initial guess results in the following:

$$
\begin{aligned}
\mathit{Value} &= \mathbb{T} + \mathbb{Z} + \mathit{Tuple} + \mathit{Function} + \mathit{Error} \\
\mathit{Error} &= \mathbb{U} \\
\mathit{Result} &= \mathit{Value}_\perp \\
\mathit{Function} &= \mathit{Value} \to \mathit{Result} \\
\mathit{Tuple} &= \mathit{Value} + (\mathit{Value} \times \mathit{Tuple}) \\
\mathit{EnvVar} &\to \mathit{Value}
\end{aligned}
$$

However, both *Function* and *Tuple* define their domains in terms of themselves. The above are actually domain equations which need to be solved for each one of the domains. Is that possible?

The answer is "yes", but we need to take a fixed point on domains to solve the equations. We've already seen this problem before when we talked about constructing a denotational model for the untyped lambda calculus. There we noted that we couldn't construct an isomorphism between D and $D \to D$ (except for the trivial solution $D = \mathbb{U}$). Now we know we only need to find a solution that makes D isomorphic to the continuous functions from D to D, which addresses the cardinality problems we saw earlier. So we can express the equation $D \cong [D \to D]$ as $D \cong \mathcal{F}(D)$ where $\mathcal{F}$ is a function on domains defined as $\mathcal{F}(E) = [E \to E]$. Then the solution we are looking for is a fixed point of $\mathcal{F}$!

We will state (without proof) that the domain equations can be solved as long as the right hand side consists of constructions of $D + E$, $D \times E$, $D \to E$, $D_\perp$. For more information, read Winskel, Chapter 12 for a discussion of *information systems*, which give a way to find fixed points on domains defined using these constructions.

Now that we have the domain equations we can write a direct semantics:

$$
\begin{aligned}
[\![e]\!] \quad & : \mathit{Env} \to \mathit{Result} \\
[\![n]\!]\rho \quad & = \lfloor \mathit{in}_2(n) \rfloor \\
[\![x]\!]\rho \quad & = \lfloor \rho(x) \rfloor \\
[\![e_1 + e_3]\!] \quad & = \mathsf{let}\ v_1 = [\![e_1]\!]\rho\ \mathsf{in}\ \mathsf{let}\ v_2 = [\![e_2]\!]\rho\ \mathsf{in}\ \mathsf{case} v_1\ \mathsf{of} \\
& \qquad \mathit{in}_1(b).\mathit{error}(= \lfloor \mathit{in}_5(\mathit{unit}) \rfloor) \\
& \qquad \mid \mathit{in}_2(n_1).\mathsf{case}\ v_2\ \mathsf{of} \\
& \qquad\qquad \mathit{in}_1(b).\mathit{error} \\
& \qquad\qquad \mid \mathit{in2}(n_2).\lfloor \mathit{in}_2(n_1 + n_2) \rfloor \\
& \qquad\qquad \mid \mathit{in}_3(t).\mathit{error} \\
& \qquad\qquad \mid \mathit{in}_4(f).\mathit{error} \\
& \qquad\qquad \mid \mathit{in}_5(\mathit{unit}).\mathit{error} \\
& \qquad \mid \mathit{in}_3(t).\mathit{error} \\
& \qquad \mid \mathit{in}_4(f).\mathit{error} \\
& \qquad \mid \mathit{in}_5(\mathit{unit}).\mathit{error}
\end{aligned}
$$

Note that this works without excess lifting because let is strict so it takes care of the delifting for us. This format is very wordy because we have to explicitly handle every case. So we will introduce a scase that allows us to do mre powerful ML-style pattern matching for the sake of brevity. This makes the remaining semantics:

$$
\begin{aligned}
[\![e_1 + e_2]\!]\rho &= \mathsf{scase}\ [\![e_1]\!]\rho\ \mathsf{of}\ \mathbb{Z}(n_1)(\mathsf{scase}\ [\![e_2]\!]\rho\ \mathsf{of}\ \mathbb{Z}(n_2).\lfloor in_2(n_1+n_2)\rfloor\ \mathsf{else}\ error)\ \mathsf{else}\ error\\
[\![true]\!]\rho &= \lfloor in_1(in_1(true))\rfloor\\
[\![false]\!]\rho &= \lfloor in_1(in_1(\mathit{false}))\rfloor\\
[\![\lambda x.\, e]\!]\rho &= \lfloor in_4(\lambda y \in \mathit{Value}.\, [\![e]\!]\rho[x \mapsto y])\rfloor\\
[\![e_0 e_1]\!]\rho &= \mathsf{scase}\ [\![e_0]\!]\rho\ \mathsf{of}\ \mathit{Function}(f).(\mathsf{scase}\ [\![e_1]\!]\rho\ \mathsf{of}\ \mathit{Error}.error \mid \mathsf{else}\ (v).f(v))\ \mathsf{else}\ error\\
[\![\mathsf{if}\ e_0\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2]\!]\rho &= \mathsf{scase}\ [\![e_0]\!]\rho\ \mathsf{of}\ \mathbb{T}(true).[\![e_1]\!]\rho \mid \mathbb{T}(\mathit{false}).[\![e_2]\!]\rho \mid \mathsf{else}\ error\\
[\![\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2]\!]\rho &= \mathsf{let}\ v = [\![e_1]\!]\rho\ \mathsf{in}\ \mathsf{scase}\ v\ \mathsf{of}\ \mathit{Error}.error \mid \mathsf{else}\ [\![e_2]\!]\rho[x \mapsto v]\\
[\![\#1\ e]\!]\rho &= \mathsf{scase}\ [\![e]\!]\rho\ \mathsf{of}\ \mathit{Tuple}(t).\mathsf{case}\ (\mathit{unfold}_{\mathit{Tuple}}\ t)\ \mathsf{of}\ in_1(v).\lfloor v\rfloor \mid in_2(\langle v, t\rangle).\lfloor v\rfloor\\
[\![\#n\ e]\!]\rho &= \mathsf{scase}\ [\![e]\!]\rho\ \mathsf{of}\ \mathit{Tuple}(t).(\mathit{fix}\ \lambda f \in \mathit{Tuple} \to \mathbb{Z} \to \mathit{Result}.\, \lambda t' \in \mathit{Tuple}.\, \lambda n \in \mathbb{Z}.\, \mathsf{scase}\ [\![e]\!]\rho\ \mathsf{of}\\
&\qquad \mathit{Tuple}(t).\mathsf{case}\ (\mathit{unfold}_{\mathit{Tuple}} t')\ \mathsf{of}\ in_1(v).\mathsf{if}\ n = 1\ \mathsf{then}\ \lfloor v\rfloor \mathsf{else}\ error\\
&\qquad \mid in_2(\langle v, t''\rangle).\mathsf{if}\ n = 1\ \mathsf{then}\ \lfloor v\rfloor\ \mathsf{else}\ f t''(n-1))t\\
[\![(e_1, \ldots, e_n)]\!]\rho &= \mathsf{let}\ v_1 = [\![e_1]\!]\rho\ \mathsf{in}\ \ldots\ \mathsf{let}\ v_n = [\![e_n]\!]\rho\ \mathsf{in}\ \lfloor in_3(\mathit{fold}(in_2\langle v_1, \mathit{fold}(in_2\langle v_2, \ldots \mathit{fold}(in_1(v_n))))))\rfloor\\
[\![\mathsf{letrec}\ f_i = \lambda x_i.\, e_i\ \mathsf{in}\ e]\!]\rho &= (\lambda F \in \mathit{Function}^n\, .\, [\![e]\!]\rho[f_1 \mapsto \pi_1 F, \ldots, f_n \mapsto \pi_n F])\\
&\qquad (\mathit{fix}_{\mathit{Function}^n}\ (\lambda F \in \mathit{Function}^n\, .\\
&\qquad\qquad \langle \lambda y \in \mathit{Value}\, .\, [\![e_1]\!]\rho[x_1 \mapsto y, f_1 \mapsto \pi_1 F, \ldots, f_n \mapsto \pi_n F], \ldots,\\
&\qquad\qquad \lambda y \in \mathit{Value}\, .\, [\![e_n]\!]\rho[x_n \mapsto y, f_1 \mapsto \pi_1 F, \ldots, f_n \mapsto \pi_n F]\rangle))
\end{aligned}
$$

Note that the construction for letrec works because $\mathit{Function} \times \mathit{Function} \times \ldots \times \mathit{Function}$ is pointed.