CS611 Lecture 25 **Error Handling** 31 October 2003

Scribe: Hui Tan Lecturer: Radu Rugina

In the FUN language, the evaluation of an expression may run into errors. This is different than other languages we have seen, such as IMP, REC, where there was no possibility of errors. Today we are going to discuss how to use Denotational Semantics in the FUN language.

1 Domains

We can recursively define domains as follows:

$$\begin{array}{rcl} v \in Value & = & \mathbb{Z} + \mathbb{T} + Pair + Function \\ n \in \mathbb{Z} & = & \{\cdots, -1, 0, 1, \cdots\} \\ t \in \mathbb{T} & = & \{true, false\} \\ p \in Pair & = & \{Value \times Value\} \\ f \in Function & = & Value \to Result \\ r \in Result & = & (Value + Error)_\perp \\ Error & = & \{error\} \end{array}$$

The denotation of an expression e in environment ρ get a result in *Result* domain, which distinguishes the difference between value, error and non-termination.

$$\mathcal{C}[\![e]\!] : Env \to Result$$

$$Env = Var \to Value$$

2 Case Constructs

To distinguish between values in different domains, such as $\mathbb{Z}, \mathbb{T}, Pair, Function, Error$, we use a case construct.

- Case construct 1. Consider the following construct:

$$\begin{array}{llll} case & d & of & in_1(x_1) \Rightarrow e_1 \\ & & | & in_2(x_2) \Rightarrow e_2 \\ & & \vdots & \\ & & | & in_k(x_k) \Rightarrow e_k \end{array}$$

  where each x_i belongs to different domain D_i. The functions in_i's are injection functions:

$$in_i : D_i \to (D_1 + \cdots + D_k)_\perp$$

  Using domain constructions, we can express the *case* construct as follows:

$$case : (D_1 + \cdots + D_k)_\perp \to D$$

$$case = [\lambda x_1.e_1, \cdots, \lambda x_k.e_k]^* \text{ (here * indicates lifting)}$$

$$\lambda x_i.e_i : D_i \to D, 1 \le i \le k$$

- Case construct 2. Next, consider another case construct which has a default clause. Given $j < k$, this construct has cases for all $i \le j$; all other cases are handled by the else clause.

$$\begin{array}{llll} case & d & of & in_1(x_1) \Rightarrow e_1 \\ & & | & in_2(x_2) \Rightarrow e_2 \\ & & \vdots & \\ & & | & in_j(x_j) \Rightarrow e_j \\ & & | & else\ e \end{array}$$

Hence, using domain constructions, we can write this *case* construct as:

$$case = [\lambda x_1.e_1, \cdots, \lambda x_j.e_j, \lambda x.e]^*$$

$$\lambda x_i.e_i : D_i \to D, 1 \le i \le j$$

$$\lambda x.e : (D_{j+1} + \cdots + D_k) \to D$$

- Case construct 3. This last construct *ecase* is a case construct which returns an error for all non-listed cases. This is the one we are going to use in the Denotational Semantics of FUN language.

$$\begin{array}{llll} ecase & d & of & in_1(x_1) \Rightarrow e_1 \\ & & | & in_2(x_2) \Rightarrow e_2 \\ & & \vdots & \\ & & | & in_j(x_j) \Rightarrow e_j \end{array}$$

Using domain constructions, *ecase* is:

$$ecase = [\lambda x_1.e_1, \cdots, \lambda x_j.e_j, \lambda x.error]^*$$

$$\lambda x_i.e_i : D_i \to D, 1 \le i \le j$$

$$\lambda x.error : (D_{j+1} + \cdots + D_k) \to D$$

3 Denotational Semantics of FUN language

The denotational semantics of FUN language are in the following form:

$$\mathcal{C}[\![e]\!]\rho \in Result$$

Since many of the domains defined earlier are sum domains and lifted domains, one must use injection functions and lifting to express values in these domains. For instance, a number n in the *Result* domain must be written as:

$$\lfloor in_{Value \to Value+Error}(in_{\mathbb{Z} \to Value(n)}) \rfloor$$

To make the denotations less verbose and more readable, we will simply write this as $\mathbb{Z}(n)$, and we will do so only in the cases of *ecase* constructs. All of the omitted injection functions and lifting brackets are otherwise considered to be implicit.

- The semantics for constants and variables:

$$\mathcal{C}[\![n]\!]\rho = n$$

$$\mathcal{C}[\![true]\!]\rho = true$$

$$\mathcal{C}[\![false]\!]\rho = false$$

$$\mathcal{C}[\![x]\!]\rho = \rho(x)$$

- The semantics for arithmetic and comparison operations are:

$$\begin{array}{ll} \mathcal{C}[\![e_0 \oplus e_1]\!]\rho = & \text{ecase } \mathcal{C}[\![e_0]\!]\rho \text{ of } \mathbb{Z}(n_0) \Rightarrow \\ & \text{ecase } \mathcal{C}[\![e_1]\!]\rho \text{ of } \mathbb{Z}(n_1) \Rightarrow \ n_0 \oplus n_1 \end{array}$$

Here, $\oplus \in \{+, -, *, \le, =\}$

- The semantics for pairs are:

$$\begin{aligned}\mathcal{C}[\![\langle e_0, e_1\rangle]\!]\rho = \quad & \text{ecase } \mathcal{C}[\![e_0]\!]\rho \text{ of } Value(v_0) \Rightarrow \\ & \text{ecase } \mathcal{C}[\![e_1]\!]\rho \text{ of } Value(v_1) \Rightarrow \ (v_0, v_1)\end{aligned}$$

$$\mathcal{C}[\![\text{fst } e]\!]\rho = \text{ecase } \mathcal{C}[\![e]\!]\rho \text{ of } Pair(p) \Rightarrow \pi_1(p)$$

$$\mathcal{C}[\![\text{snd } e]\!]\rho = \text{ecase } \mathcal{C}[\![e]\!]\rho \text{ of } Pair(p) \Rightarrow \pi_2(p)$$

- The CBV semantics for functions are:

$$\begin{aligned}\mathcal{C}[\![e_0 e_1]\!]\rho = \quad & \text{ecase } \mathcal{C}[\![e_0]\!]\rho \text{ of } Function(f) \Rightarrow \\ & \text{ecase } \mathcal{C}[\![e_1]\!]\rho \text{ of } Value(v) \Rightarrow \ f(v)\end{aligned}$$

$$\mathcal{C}[\![\lambda x.e]\!]\rho = \lambda v \in Value.\mathcal{C}[\![e]\!]\rho[x \to v]$$

$$\mathcal{C}[\![\text{rec } y(\lambda x.e)]\!]\rho = fix(\lambda f \in Function.\mathcal{C}[\![\lambda x.e]\!]\rho[y \to f])$$

- The semantics for the remaining expressions are:

$$\mathcal{C}[\![\text{let } x = e_0 \text{ in } e_1]\!]\rho = \quad \text{ecase } \mathcal{C}[\![e_0]\!]\rho \text{ of } Value(v) \Rightarrow \ \mathcal{C}[\![e_1]\!]\rho[x \to v]$$

$$\begin{array}{lclll}\mathcal{C}[\![\text{if } e_0 \text{ then } e_1 \text{ else } e_2]\!]\rho & = & \text{ecase } \mathcal{C}[\![e_0]\!]\rho & \text{of} & \mathbb{T}(true) \Rightarrow \mathcal{C}[\![e_1]\!]\rho \\ & & & | & \mathbb{T}(false) \Rightarrow \mathcal{C}[\![e_2]\!]\rho\end{array}$$