CS 6110 — Advanced Programming Languages

Lecture 11
A Functional Language

18 February 2011

Syntax

Expressions

$$
\begin{array}{rcl}
e & ::= & x \mid \lambda x_1 \ldots x_n . e \mid e_0 \cdots e_n \\
& \mid & e_1 \oplus e_2 \mid n \\
& \mid & \text{true} \mid \text{false} \mid \text{if } e_0 \text{ then } e_1 \text{ else } e_2 \\
& \mid & (e_1, \ldots, e_n) \mid \#n\, e \\
& \mid & \text{null} \\
& \mid & \text{let } x = e_1 \text{ in } e_2 \\
& \mid & \text{letrec } f_1 = \lambda x_1 . e_1 \text{ and } \ldots \text{ and } f_n = \lambda x_n . e_n \text{ in } e
\end{array}
$$

Syntax

Values

$$
\begin{array}{rcl}
v & ::= & \lambda x_1 \ldots x_n . e \\
& | & n \\
& | & \text{true} \mid \text{false} \\
& | & (v_1, \ldots, v_n) \\
& | & \text{null}
\end{array}
$$

Structural Operational Semantics

Evaluation Order Rule

$$\frac{e \xrightarrow{1} e'}{E[e] \xrightarrow{1} E[e']}$$

Evaluation Contexts

$$\begin{array}{rcl} E & ::= & [\cdot] \\ & \mid & v_0 \cdots v_m \, E \, e_{m+2} \cdots e_n \\ & \mid & \text{if } E \text{ then } e_1 \text{ else } e_2 \\ & \mid & \#n \, E \\ & \mid & \text{let } x = E \text{ in } e \\ & \mid & (v_1, \ldots, v_m, E, e_{m+2}, \ldots, e_n) \end{array}$$

Structural Operational Semantics

Reduction Rules

$$
\begin{array}{rcll}
(\lambda x_1 \ldots x_n . e)\, v_1 \cdots v_n & \rightarrow & e\{v_1/x_1\}\{v_2/x_2\} \cdots \{v_n/x_n\} & \\
n_1 \oplus n_2 & \rightarrow & n_3 & \text{where } n_1 \oplus n_2 = n_3 \\
\#n\,(v_1, \ldots, v_m) & \rightarrow & v_n, & \text{where } 1 \leq n \leq m \\
\text{if true then } e_1 \text{ else } e_2 & \rightarrow & e_1 & \\
\text{if false then } e_1 \text{ else } e_2 & \rightarrow & e_2 & \\
\text{let } x = v \text{ in } e & \rightarrow & e\{v/x\} & \\
\text{"}\mathit{letrec}\text{"} \ldots & \rightarrow & \mathit{to\ be\ continued} &
\end{array}
$$

Semantics via Translation

$$
\begin{aligned}
[\![\lambda x_1 \ldots x_n . e]\!] &\triangleq \lambda x_1 \ldots x_n . [\![e]\!] \\
[\![e_0 \cdots e_n]\!] &\triangleq [\![e_0]\!]\,[\![e_1]\!]\,[\![e_2]\!] \cdots [\![e_n]\!] \\
[\![x]\!] &\triangleq x \\
[\![n]\!] &\triangleq \lambda f x . f^n x \\
[\![\mathsf{null}]\!] &\triangleq \mathsf{id} \\
[\![\mathsf{true}]\!] &\triangleq \lambda x y . x\,\mathsf{id} \\
[\![\mathsf{false}]\!] &\triangleq \lambda x y . y\,\mathsf{id} \\
[\![\mathsf{if}\ e_0\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2]\!] &\triangleq [\![e_0]\!]\,(\lambda z . [\![e_1]\!])\,(\lambda z . [\![e_2]\!]).
\end{aligned}
$$

Tuples

We have already seen how to represent lists in the λ-calculus using the functions list, head, tail, nil, and empty...

$$
\begin{aligned}
[\![()]\!] &\triangleq \mathsf{nil} \\
[\![(e_1, e_2, \ldots, e_n)]\!] &\triangleq \mathsf{list}\,[\![e_1]\!]\,[\![(e_2, \ldots, e_n)]\!] \\
[\![\#n\,e]\!] &\triangleq \mathsf{head}\,(\mathsf{tail}^{n-1}\,[\![e]\!]).
\end{aligned}
$$

Let Bindings

The let construct can be translated as follows:

$$[\![\text{let } x = e_1 \text{ in } e_2]\!] \stackrel{\triangle}{=} (\lambda x.\, e_2)e_1.$$

Let Bindings

The let construct can be translated as follows:

$$[\![\text{let } x = e_1 \text{ in } e_2]\!] \stackrel{\triangle}{=} (\lambda x.\, e_2)e_1.$$

The letrec construct can be used to define blocks of n mutually recursive functions. For simplicity, we show the case where $n = 1$.

$$[\![\text{letrec } f = \lambda x.\, e_1 \text{ in } e_2]\!] \stackrel{\triangle}{=} (\lambda f.\, [\![e_2]\!])\,(Y\,(\lambda f.\, [\![\lambda x.\, e_1]\!])).$$

A Problem

Unfortunately, there is a serious problem with this translation...

A Problem

Unfortunately, there is a serious problem with this translation... It is unsound!

The translation of $[\![\text{if } 3 \text{ then } 1 \text{ else } 0]\!]$ is

$$(\lambda fx.\, f\,(f\,(f\,x)))\;(\lambda z.\, \lambda f\, x.\, fx)\;(\lambda z.\, z),$$

which reduces to a value under CBV.

But this value does not correspond to the stuck non-value (if 3 then 1 else 0) in source language. It is meaningless gibberish!

Run-time Types

We'll represent types using integers,

$$\begin{array}{lll} \text{Err} \triangleq 0 & \text{Null} \triangleq 1 & \text{Bool} \triangleq 2 \\ \text{Num} \triangleq 3 & \text{Tuple} \triangleq 4 & \text{Func} \triangleq 5 \end{array}$$

but any distinct set of values would do.

Instrumentation

Extend expressions with a special error value:

$$e ::= \ldots \mid \text{error}$$

Define a translation from the language to itself:

$$\begin{aligned}
\mathcal{E}[\![\text{null}]\!] &\triangleq (\text{Null}, \text{null}) \\
\mathcal{E}[\![t]\!] &\triangleq (\text{Bool}, t),\ t \in \{\text{true}, \text{false}\} \\
\mathcal{E}[\![n]\!] &\triangleq (\text{Num}, n) \\
\mathcal{E}[\![(e_1, \ldots, e_n)]\!] &\triangleq (\text{Tuple}, n, (\mathcal{E}[\![e_1]\!], \ldots, \mathcal{E}[\![e_n]\!])) \\
\mathcal{E}[\![\text{error}]\!] &\triangleq (\text{Err}, \text{error}) \\
\mathcal{E}[\![\lambda x_1, \ldots, x_n . e]\!] &\triangleq (\text{Func}, n, \lambda x_1, \ldots, x_n . \mathcal{E}[\![e]\!])
\end{aligned}$$

Instrumentation

$$
\begin{array}{l}
\mathcal{E}[\![\text{if } e_0 \text{ then } e_1 \text{ else } e_2]\!] \stackrel{\triangle}{=} \\
\quad \text{let } z = \mathcal{E}[\![e_0]\!] \text{ in} \\
\quad \text{if } \#1\, z = \text{Bool} \\
\quad\quad \text{then } (\text{if } \#2\, z \text{ then } \mathcal{E}[\![e_1]\!] \text{ else } \mathcal{E}[\![e_2]\!]) \\
\quad\quad \text{else } \mathcal{E}[\![\text{error}]\!]
\end{array}
$$

Instrumentation

$$
\begin{aligned}
&\mathcal{E}[\![e_1 \oplus e_2]\!] \stackrel{\triangle}{=} \\
&\quad \text{let } z_1 = \mathcal{E}[\![e_1]\!] \text{ in} \\
&\quad \text{let } z_2 = \mathcal{E}[\![e_2]\!] \text{ in} \\
&\quad \text{if } (\#1\, z_1 = \text{Num}) \wedge (\#1\, z_2 = \text{Num}) \\
&\qquad \text{then } (\#2\, z_1) \oplus (\#2\, z_2) \\
&\qquad \text{else } \mathcal{E}[\![\text{error}]\!]
\end{aligned}
$$

Instrumentation

$$
\begin{array}{l}
\mathcal{E}[\![\mathsf{e}_0\,\mathsf{e}_1\,\cdots\,\mathsf{e}_n]\!] \stackrel{\triangle}{=} \\
\quad \mathsf{let}\ z = \mathcal{E}[\![\mathsf{e}_0]\!]\ \mathsf{in} \\
\quad \mathsf{if}\ (\#1\,z = \mathsf{Func}) \wedge (\#2\,z = n) \\
\quad\quad \mathsf{then}\ \#3\,z\,\mathcal{E}[\![\mathsf{e}_1]\!]\,\cdots\,\mathcal{E}[\![\mathsf{e}_n]\!] \\
\quad\quad \mathsf{else}\ \mathcal{E}[\![\mathsf{error}]\!]
\end{array}
$$