Recitation 11:
Stacks

15-312: Principles of Programming Languages

Wednesday, April 2, 2014

For this recitation, we will use the construction described in lecture and in Chapter 28 of PFPL for control stacks. As a refresher, here is the relevant portion of our syntax for stacks in the language:

\[
\text{Sort} \quad s ::= k \triangleright e \\
\text{State} \quad s ::= k \triangleleft e \\
\text{Stack} \quad k ::= e \\
\text{Frame} \quad f ::= \square (e_2) \\
\]

The direction of the triangle is used to indicate whether the expression in a state might need further evaluation. For \( k \triangleright e \), we interpret this as some stack \( k \) waiting for the evaluation of \( e \) before preceding. For \( k \triangleleft e \), we interpret this as some stack \( k \) ready to use \( e \) where \( e \ \text{val} \). The \( k \downarrow \) case is used to reflect exceptions which are passed on to the stack.

If we want to step some expression \( e \) in this language, we start with the state \( \epsilon \triangleright e \). When the computation is complete, our state should be of the form \( \epsilon \triangleleft e' \) where \( e \mapsto^* e' \) and \( e \ \text{val} \) in the case of success and \( \epsilon \downarrow \) in the case of failure.

Recall from class the rules for stepping states which relate to function application:

\[
\begin{align*}
(\triangleright) & : k \triangleright (x; e_2) & k; \square e_2 & \triangleright e_1 \\
(\triangleleft) & : k; \square e_2 \triangleleft e_1 & k; e_1 (\square) & \triangleright e_2 \\
(\downarrow) & : k; (\lambda(x) e) (\square) \triangleleft e_2 & k \triangleright [e_2/x]e
\end{align*}
\]

Task 1  What do we need to add to our syntax for frame to support evaluation of \( z, \lambda(x) e \), and \( \text{fix}[\tau](x.e) \)?

Task 2  What about \( s(e), \text{ifz}(e_0; e_1; x.e_2), \text{raise} \), and \( \text{try } e \text{ otherwise } e' \)?

Task 3  Now give the relevant rules for stepping states from which use expressions from Task 1.
**Task 4**  Do the same with Task 2.

**Task 5**  Show the steps involved in evaluating \((\lambda(x) x) \(z\))\) using traditional dynamics and again using stacks.