Recitation 12:
Call by Need

15-312: Principles of Programming Languages
Wednesday, April 16, 2014

1 By-Need Evaluation

By-need evaluation of functions uses memoization to record the value of a computation so that any future use of the same computation may return the previously computed value (or compute it from scratch if there is none). This is achieved by “naming” each deferred computation with a symbol, which is then used to access its value whenever it is used. A memo table records the deferred computation associated to each symbol \((a \mapsto b)\) until such time as it is evaluated, after which it records the value of that computation \((a \mapsto c)\). Thus naming implements sharing, and the memo table ensures irredundancy.

The rules in the dynamics are thus:

\[
\begin{align*}
    s &\mapsto s' \\
    \nu \Sigma \{s \parallel \varphi\} &\mapsto \nu \Sigma \{s' \parallel \varphi\} \quad (L_1) \\
    \nu \Sigma, a \sim \tau \{(k \triangleright a) \parallel \varphi \otimes (a \mapsto \triangleright e)\} &\mapsto \nu \Sigma, a \sim \tau \{(k; \triangleright a \triangleleft v) \parallel \varphi \otimes (a \mapsto \triangleleft v)\} \quad (L_2) \\
    \nu \Sigma, a \sim \tau \{(k; \triangleright a \triangleleft v) \parallel \varphi \otimes (a \mapsto \bullet)\} &\mapsto \nu \Sigma, a \sim \tau \{(k \triangleleft v) \parallel \varphi \otimes (a \mapsto \triangleleft v)\} \quad (L_3) \\
    \nu \Sigma, a \sim \tau \{(k \triangleright a) \parallel \varphi \otimes (a \mapsto \triangleright v)\} &\mapsto \nu \Sigma, a \sim \tau \{(k \triangleleft v) \parallel \varphi \otimes (a \mapsto \triangleleft v)\} \quad (L_4) \\
    \nu \Sigma, a \sim \tau \{(k \triangleright a) \parallel \varphi \otimes (a \mapsto \bullet)\} &\mapsto \nu \Sigma, a \sim \tau \{(k \triangleright a) \parallel \varphi \otimes (a \mapsto \bullet)\} \quad (L_5) \\
    \nu \Sigma \{(k \triangleright \text{fix}(\tau(x.e)) \parallel \varphi\} &\mapsto \nu \Sigma, a \sim \tau \{(k \triangleright \text{fix}(\tau(x.e)) \parallel \varphi \otimes (a \mapsto \triangleright [\text{fix}(\tau(x.e)]))\} \quad (L_6) \\
    \nu \Sigma \{(k \triangleright \text{s}(e)) \parallel \varphi\} &\mapsto \nu \Sigma, a \sim \text{nat} \{(k \triangleleft \text{s}(\tau(a)) \parallel \varphi \otimes (a \mapsto \triangleleft e)\} \quad (L_7)
\end{align*}
\]

**Task 1.1** (100%). Does \(\text{fix}(\tau)(f \cdot \lambda(x : \text{nat}) f x)\) loop? Show the evaluation trace.

2 Suspensions

Another way to introduce laziness is to consolidate the machinery of the by-need dynamics into a single type whose values are possibly unevaluated, memoized computations. The type of suspensions of type \(\tau\),

\[
\text{suspensions} = \tau 
\]
written \( \tau \) susp, has as introductory form \( \text{susp}[\tau](x.e) \) representing the suspended computation \( e \) of type \( \tau \).

For convenience, suspended computations can be self-referential – that is the purpose of the abstractor. The key rules are as follows:

\[
\frac{k \triangleright \text{susp}[a]}{\Sigma} \quad k \triangleleft \text{susp}[a] \quad (S_1) \\
\frac{k \triangleright \text{force}(e)}{\Sigma} \quad k; \text{force}(\square) \triangleright e \quad (S_2)
\]

\[
\nu \Sigma \{ (k \triangleright \text{susp}[\tau](x.e)) \parallel \varphi \} \rightarrow \nu \Sigma, a \sim \tau \{ (k \triangleleft \text{susp}[a]) \parallel \varphi \otimes (a \rightarrow \triangleright \text{delay}[a/x]) \} \quad (S_3)
\]

\[
\nu \Sigma, a \sim \tau \{ (k; \text{force}(\square) \triangleleft \text{susp}[a]) \parallel \varphi \otimes (a \rightarrow \triangleright e) \} \rightarrow \nu \Sigma, a \sim \tau \{ (k; \square a \triangleright e) \parallel \varphi \otimes (a \rightarrow \bullet) \} \quad (S_4)
\]

\[
\nu \Sigma, a \sim \tau \{ (k; \square a \triangleright v) \parallel \varphi \otimes (a \rightarrow \bullet) \} \rightarrow \nu \Sigma, a \sim \tau \{ (k \triangleright v) \parallel \varphi \otimes (a \rightarrow \triangleright v) \} \quad (S_5)
\]

\[
\nu \Sigma, a \sim \tau \{ (k; \text{force}(\square) \triangleleft \text{susp}[a]) \parallel \varphi \otimes (a \rightarrow \triangleright v) \} \rightarrow \nu \Sigma, a \sim \tau \{ (k \triangleright v) \parallel \varphi \otimes (a \rightarrow \triangleright v) \} \quad (S_6)
\]

\[
\nu \Sigma, a \sim \tau \{ (k; \text{force}(\square) \triangleleft \text{susp}[a]) \parallel \varphi \otimes (a \rightarrow \bullet) \} \text{ loops} \quad (S_7)
\]

Let’s implement suspensions in SML:

```ml
signature SUSP = sig
  type 'a susp
  val force : 'a susp -> 'a
  val delay : (unit -> 'a) -> 'a susp
end
```