1 The π-calculus

$$
\begin{array}{rcl}
\pi & ::= & \tau \mid \bar{x}\langle y\rangle \mid x(y) \mid [x = y]\pi \\
M, N & ::= & 0 \mid \pi.P \mid M + N \\
P & ::= & M \mid P_1|P_2 \mid \nu x P \mid !P
\end{array}
$$

Polyadic encoding and recursion were also covered in the previous lecture.

2 Encoding Booleans

$$
\begin{array}{rcl}
True(l) & \triangleq & l(t, f).\bar{t} \\
False(l) & \triangleq & l(t, f).\bar{f} \\
Cond(P, Q)(l) & \triangleq & \nu t, f(\bar{l}\langle t, f\rangle.(t.P + f.Q))
\end{array}
$$

3 Encoding Lists

$$
\begin{array}{rcl}
Nil(k) & \triangleq & k(n, c).\bar{k} \\
Cons(V, L)(k) & \triangleq & \nu v, l(k(n, c).\bar{c}\langle v, l\rangle \mid V\langle v\rangle \mid L\langle l\rangle) \\
IsNil(L)(r) & \triangleq & \nu l, n, c(L\langle l\rangle \mid \bar{l}\langle n, v\rangle.(n.True\langle r\rangle + c(n, t).False\langle r\rangle))
\end{array}
$$

Introduce a *case* form:

$$
\begin{array}{rl}
\text{case } l \text{ of} & \\
Nil? & \Rightarrow P \\
Cons? & \Rightarrow Q
\end{array}
$$

3.1 Destructive List Copy

$$
\begin{array}{rcl}
Copy\langle l, m\rangle & \triangleq & \text{case } l \text{ of} \\
& & Nil? \Rightarrow Nil\langle m\rangle \\
& & Cons?(h, t) \Rightarrow \nu m'(m(n, c).\bar{c}\langle n, m'\rangle \mid Copy\langle r, m'\rangle)
\end{array}
$$

3.2 Destructive List Join

$$\begin{array}{rcl} Join\langle k, l, m\rangle & \triangleq & \text{case } k \text{ of} \\ & & Nil? \Rightarrow Copy\langle l, m\rangle \\ & & Cons?(h,t) \Rightarrow \nu m'(m(n,c).\bar{c}\langle n, m'\rangle \mid Join\langle t, l, m'\rangle) \end{array}$$

A $Double$ operation cannot be defined as follows because $Join$ is destructive

$$Double(l, m) \triangleq Join(l, lm)$$

4 Encoding Persistent Datatypes

We put a ! in front of processes to turn them into servers create arbitrary numbers of the original process. This prevents their destruction after sending or receiving a message. This is generally used on "connectors" – the processes that acts as "cons cell" equivalents. The List encoding can be rewritten as:

$$\begin{array}{rcl} Nil(k) & \triangleq & !k(n,c).\bar{k} \\ Cons(V, L)(k) & \triangleq & \nu v, l(!k(n,c).\bar{c}\langle v, l\rangle \mid V\langle v\rangle \mid L\langle l\rangle) \\ IsNil(L)(r) & \triangleq & \nu l, n, c(L\langle l\rangle \mid \bar{l}\langle n, v\rangle.(n.True\langle r\rangle + c(n,t).False\langle r\rangle)) \end{array}$$

5 Reference Cells for Mutable Data

$$\begin{array}{rcl} NullRef(r) & \triangleq & r(g,s,t,i).(s(v).Ref\langle v, r\rangle + t.NullRef\langle r\rangle + i(b).True\langle b\rangle) \\ Ref(v,r) & \triangleq & r(g,s,t,i).(\bar{g}(v) + s(v').Ref\langle v', r\rangle + t.NullRef\langle r\rangle + i(b).False\langle b\rangle) \end{array}$$

6 Encoding the λ-calculus

We can express the syntax of the pure untyped λ-calculus as:

$$e \ ::= \ x \mid e_1\, e_2 \mid \lambda x.\, e$$

The semantics can be expressed as:

$$\frac{e_1 \to e_1'}{e_1 e_2 \to e_1' e_2} \qquad \frac{}{(\lambda x.\, e_1)e_2 \to e_1[e_2/x]}$$

The above semantics can be encoded in the π-calculus as:

$$\begin{array}{rcl} [\![x]\!](u) & \triangleq & \bar{x}\langle u\rangle \\ [\![\lambda x.\, e]\!](u) & \triangleq & u(x,y).[\![e]\!]\langle v\rangle \\ [\![e_1, e_2]\!](u) & \triangleq & \nu y([\![e_1]\!]\langle y\rangle \mid \nu x(\bar{y}\langle x, u\rangle \mid !x(w).[\![e_2]\!]\langle w\rangle)) \end{array}$$