1 Review of CBN and CBV Semantics

\[ e ::= x \mid \lambda x e \mid e_0 e_1 \]
\[ v ::= \lambda x e \]

Call By Name Semantics:

\[ C[(\lambda x e_0) e_1] \mapsto C[e_0 \{e_1/x\}] \]
\[ C ::= [\cdot] \mid C e \]

Call By Value Semantics:

\[ C[(\lambda x e) v] \mapsto C[e \{v/x\}] \]
\[ C ::= [\cdot] \mid C e \mid v C \]

2 CBV Translation and Notes

Call By Name to Call By Value Translation For compactness we omit the name of the translation and just treat \([\cdot]\) as a semantic function itself

\[
\begin{align*}
[x] &= x I = x (\lambda y y) \\
[\lambda x e] &= \lambda x [e] \\
[[e_0 e_1]] &= [[e_0]] (\lambda z [[[e_1]]])
\end{align*}
\]

Expressing semantics through translations is a style of semantics known as denotational semantics, although the target language is usually mathematical functions rather than \(\lambda\)-calculus terms. We’ll see true denotational semantics later in the course.

3 Soundness and Adequacy in CBV semantics

**Soundness:** \(e \mapsto^* v \Rightarrow \exists v'[e] \mapsto^* v' \land v' \approx [v] \)

**Adequacy:** \(\exists v' e \mapsto^* v \land v' \approx [v] \Leftarrow [e] \mapsto^* v' \)

Basic idea of soundness is saying that the operational semantics doesn’t break the meaning (with respect to the translation) of the program as it executes.

**Proof of soundness**

We will show that if \(e \mapsto e'\) in CBN then \([e] \approx [e']\)

We will prove this by induction on the form of \(C_N\).
For \( C_N = \{ \} \) we have \( C_N[(\lambda \ x \ e_0) \ e_1] \mapsto C_N[e_0\{e_1/x\}] \) or equivalently \( (\lambda \ x \ e_0) \ e_1 \mapsto e_0\{e_1/x\} \). So we have to show that \( [e_0]\{\lambda \ z \ [e_1]/x\} \approx [e_0\{e_1/x\}] \).

We will show this by structural induction on \( e_0 \).

If \( e_0 = x \) then we have:

\[
[e_0]\{\lambda \ z \ [e_1]/x\} \approx (x \ I)\{\lambda \ z \ [e_1]/x\} \approx \lambda \ z \ [e_1] I \approx [e_1] \approx [x\{e_1/x\}] \approx [e_0\{e_1/x\}]
\]

If \( e_0 = y \) with \( y \neq x \) then we have:

\[
[e_0]\{\lambda \ z \ [e_1]/x\} \approx (y \ I)\{\lambda \ z \ [e_1]/x\} \approx y \ I \approx [y] \approx [y\{e_1/x\}] \approx [e_0\{e_1/x\}]
\]

If \( e_0 = \lambda \ x \ e_2 \) we have:

\[
[\lambda \ x \ e_2]\{\lambda \ z \ [e_1]/x\} \approx (\lambda \ x \ [e_2])\{\lambda \ z \ [e_1]/x\} \approx [(\lambda \ x \ e_2)\{e_1/x\}] \approx [e_0\{e_1/x\}]
\]

If \( e_0 = \lambda \ y \ e_2 \) we have:

\[
[\lambda \ y \ e_2]\{\lambda \ z \ [e_1]/x\} \approx (\lambda \ y \ [e_2])\{\lambda \ z \ [e_1]/x\}
\]

Given that \( e_2 \) is a subexpression of \( e_0 \) we can apply the induction hypothesis obtaining:

\[
\lambda \ y \ ([e_2]\{\lambda \ z \ [e_1]/x\}) \approx \lambda \ y \ [e_2\{e_1/x\}] \approx [(\lambda \ y \ e_2)\{e_1/x\}] \approx [e_0\{e_1/x\}]
\]

If \( e_0 = e_2 \) then we have:

\[
[e_2]\{\lambda \ z \ [e_1]/x\} \approx ([e_2]\{\lambda \ z \ [e_3]/x\})\{\lambda \ z \ [e_1]/x\} \approx ([e_2]\{\lambda \ z \ [e_1]/x\})(\lambda \ z \ [e_3])\{\lambda \ z \ [e_1]/x\}) \approx \\
\approx ([e_2]\{\lambda \ z \ [e_1]/x\})(\lambda \ z \ [e_3])\{\lambda \ z \ [e_1]/x\}) \approx \\
\approx [e_2\{e_3\}(e_1/x)] \approx [e_0\{e_1/x\}]
\]

This concludes our proof that \( [e_0]\{\lambda \ z \ [e_1]/x\} \approx [e_0\{e_1/x\}] \).

Now, for \( C_N = C'_N \ e'' \) we have \( C'_N[(\lambda \ x \ e_0) \ e_1]e'' \mapsto C'_N[e_0\{e_1/x\}]e'' \). Because \( C'_N \) is a subexpression of \( C_N \) we have \( [C'_N[(\lambda \ x \ e_0) \ e_1]] \approx [C'_N[e_0\{e_1/x\}]] \) according to the induction hypothesis. So we have (with induction on structure of \( C_N \), now):

\[
[C'_N((\lambda \ x \ e_0) \ e_1)e''] \approx [C'_N((\lambda \ x \ e_0)\{e_1\})](\lambda \ z \ [e'']) \approx [C'_N[e_0\{e_1/x\}]](\lambda \ z \ [e'']) \approx [C'_N[e_0\{e_1/x\}]]e''
\]

4 Extending the CBV Lambda Calculus

4.1 Adding If’s and booleans

\[
e ::= \ldots \mid \#t \mid \#f \mid \text{if } e_0 \text{ then } e_1 \text{ else } e_2
\]

\[
v ::= \ldots \mid \#t \mid \#f
\]

SOS:

\[
C ::= \ldots \mid \text{if } C \text{ then } e_1 \text{ else } e_2
\]

\[
\frac{e \mapsto e'}{C[e] \mapsto C[e']}
\]

\[
\frac{\text{if } t \text{ then } e_1 \text{ else } e_2 \mapsto e_1}{\text{if } \#t \text{ then } e_1 \text{ else } e_2 \mapsto e_1}
\]

\[
\frac{\text{if } f \text{ then } e_1 \text{ else } e_2 \mapsto e_2}{\text{if } \#f \text{ then } e_1 \text{ else } e_2 \mapsto e_2}
\]

\[
[\#t] = \lambda \ x \ y \ (x \ I)
\]

\[
[\#f] = \lambda \ x \ y \ (y \ I)
\]

\[
[\text{if } e_0 \text{ then } e_1 \text{ else } e_2] = [e_0] (\lambda \ z \ [e_1]) (\lambda \ z \ [e_2])
\]

Note that this translation has no error checking for the case where \#t or \#f are not first argument to an if expression.
4.2 Adding Let’s

\[ e ::= \ldots \mid \text{let } x = e_1 \text{ in } e_2 \]

SOS:

\[ C ::= \ldots \mid \text{let } x = C \text{ in } e_2 \]

\[
\text{let } x = v \text{ in } e \rightarrow e\{v/x\}
\]

\[[\text{let } x = e_1 \text{ in } e_2] = (\lambda x \ [e_2]) \ [e_1]\]

4.3 Adding Pairs

\[ e ::= \ldots \mid \langle e_1, e_2 \rangle \mid \text{left } e \mid \text{right } e \]
\[ v ::= \ldots \mid \langle v_1, v_2 \rangle \]

SOS:

\[ C ::= \ldots \mid \langle C, e \rangle \mid \langle v, C \rangle \mid \text{left } C \mid \text{right } C \]

\[
\text{left } \langle v_1, v_2 \rangle \rightarrow v_1 \quad \text{right } \langle v_1, v_2 \rangle \rightarrow v_2
\]

\[[\langle e_1, e_2 \rangle] = (\lambda x \lambda y \lambda f \ f x y) \ [e_1] \ [e_2] \]
\[[\text{left } e] = [e] (\lambda x \lambda y \ x) \]
\[[\text{right } e] = [e] (\lambda x \lambda y \ y) \]