Harvard School of Engineering and Applied Sciences — CS 152: Programming Languages

Lambda calculus encodings and Recursion; Definitional translations (Lectures 8–9) Section and Practice Problems

Week 5: Tue Feb 20-Fri 23, 2018

---

1 Lambda calculus encodings

(a) Evaluate $AND\ FALSE\ TRUE$ under CBV semantics.

Answer:
Recall that:

$$TRUE \triangleq \lambda x.\, \lambda y.\, x$$
$$FALSE \triangleq \lambda x.\, \lambda y.\, y$$
$$AND \triangleq \lambda b_1.\, \lambda b_2.\, b_1\ b_2\ FALSE$$

We can substitute these definitions and evaluate:

$$\begin{aligned} AND\ FALSE\ TRUE &\triangleq (\lambda b_1.\, \lambda b_2.\, b_1\ b_2\ FALSE)\ (\lambda x.\, \lambda y.\, y)\ (\lambda x.\, \lambda y.\, x) \\ &\longrightarrow (\lambda b_2.\, (\lambda x.\, \lambda y.\, y)\ b_2\ FALSE)\ (\lambda x.\, \lambda y.\, x) \\ &\longrightarrow (\lambda x.\, \lambda y.\, y)\ (\lambda x.\, \lambda y.\, x)\ FALSE \\ &\longrightarrow (\lambda y.\, y)\ FALSE \\ &\longrightarrow FALSE \end{aligned}$$

(b) Evaluate $IF\ FALSE\ \Omega\ \lambda x.\, x$ under CBN semantics. What happens when you evaluated it under CBV semantics?

Answer:
Recall that:

$$IF \triangleq \lambda b.\, \lambda t.\, \lambda f.\, b\ t\ f$$
$$\Omega \triangleq (\lambda x.\, x\ x)\ (\lambda x.\, x\ x)$$

We can substitute these definitions and evaluate under CBN semantics:

$$\begin{aligned} IF\ FALSE\ \Omega\ (\lambda x.\, x) &\triangleq (\lambda b.\, \lambda t.\, \lambda f.\, b\ t\ f)\ (\lambda x.\, \lambda y.\, y)\ ((\lambda x.\, x\ x)\ (\lambda x.\, x\ x))\ (\lambda x.\, x) \\ &\longrightarrow (\lambda t.\, \lambda f.\, (\lambda x.\, \lambda y.\, y)\ t\ f)\ ((\lambda x.\, x\ x)\ (\lambda x.\, x\ x))\ (\lambda x.\, x) \\ &\longrightarrow (\lambda f.\, (\lambda x.\, \lambda y.\, y)\ ((\lambda x.\, x\ x)\ (\lambda x.\, x\ x))\ f)\ (\lambda x.\, x) \\ &\longrightarrow (\lambda x.\, \lambda y.\, y)\ ((\lambda x.\, x\ x)\ (\lambda x.\, x\ x))\ (\lambda x.\, x) \\ &\longrightarrow (\lambda y.\, y)\ (\lambda x.\, x) \\ &\longrightarrow (\lambda x.\, x) \end{aligned}$$

If we evaluate this expression under CBV semantics, we need to evaluate the Omega expression before we can apply the lambda abstraction. Since the Omega expression evaluates indefinitely, the overall expression never terminates.

(c) Evaluate $ADD\ \overline{2}\ \overline{1}$ under CBV semantics. (Make sure you know what the Church encoding of 1 and 2 are, and check that the answer is equal to the Church encoding of 3.)

Answer:
Recall that:

$$\overline{1} \triangleq \lambda f.\, \lambda x.\, f\ x$$
$$\overline{2} \triangleq \lambda f.\, \lambda x.\, f\ (f\ x)$$
$$PLUS \triangleq \lambda n_1.\, \lambda n_2.\, n_1\ SUCC\ n_2$$
$$SUCC \triangleq \lambda n.\, \lambda f.\, \lambda x.\, f\ (n\ f\ x)$$

We can substitute these definitions and evaluate under CBV semantics:

$$\begin{aligned}
ADD\ \overline{2}\ \overline{1} &\triangleq (\lambda n_1.\, \lambda n_2.\, n_1\ SUCC\ n_2)\ \overline{2}\ \overline{1} \\
&\longrightarrow (\lambda n_2.\, \overline{2}\ SUCC\ n_2)\ \overline{1} \\
&\longrightarrow (\overline{2}\ SUCC\ \overline{1}) \\
&\longrightarrow (\lambda f.\, \lambda x.\, f\ (f\ x))\ SUCC\ \overline{1} \\
&\longrightarrow (\lambda x.\, SUCC\ (SUCC\ x))\ \overline{1} \\
&\longrightarrow SUCC\ (SUCC\ \overline{1}) \\
&\longrightarrow SUCC\ ((\lambda n.\, \lambda f.\, \lambda x.\, f\ (n\ f\ x))\ \overline{1}) \\
&\longrightarrow SUCC\ (\lambda f.\, \lambda x.\, f\ (\overline{1}\ f\ x)) \\
&\longrightarrow \lambda f.\, \lambda x.\, f\ ((\lambda f.\, \lambda x.\, f\ (\overline{1}\ f\ x))\ f\ x)
\end{aligned}$$

This is functionally equivalent to $\overline{3}$

(d) In class we made use of a combinator $ISZERO$, which takes a Church encoding of a natural number n, and evaluates to $TRUE$ if n is zero, and $FALSE$ if n is not zero. (We don't care what $ISZERO$ does if it is applied to a lambda term that is not a Church encoding of a natural number.)

Define $ISZERO$.

Answer: *We define ISZERO to be the following:*

$$ISZERO \triangleq \lambda n.\, n\ (\lambda x.\, FALSE)\ TRUE$$

2 Recursion

Assume we have an applied lambda calculus with integers, booleans, conditionals, etc. Consider the following higher-order function H.

$$H \triangleq \lambda f.\, \lambda n.\, \text{if } n = 1 \text{ then true else if } n = 0 \text{ then false else not } (f\ (n-1))$$

(a) Suppose that g is the fixed point of H. What does g compute?

Answer: *g computes whether a number is odd.*

(b) Compute $Y\ H$ under CBN semantics. What has happened to the function call $f\ (n-1)$?

Answer: *Recall that:*

$$Y \triangleq \lambda f.\,(\lambda x.\,f\ (x\ x))\ (\lambda x.\,f\ (x\ x))$$

We can substitute this definition and evaluate under CBN semantics:

$$\begin{aligned}
ISODD = Y\ H &\triangleq (\lambda f.\,(\lambda x.\,f\ (x\ x))\ (\lambda x.\,f\ (x\ x)))\ H \\
&\longrightarrow (\lambda x.\,H\ (x\ x))\ (\lambda x.\,H\ (x\ x)) \\
&\longrightarrow H\ ((\lambda x.\,H\ (x\ x))\ (\lambda x.\,H\ (x\ x))) \\
&\longrightarrow (\lambda f.\,\lambda n.\ \text{if } n=1 \text{ then true else if } n=0 \text{ then false else not } (f\ (n-1)))\ ((\lambda x.\,H\ (x\ x))\ (\lambda x.\,H\ (x\ x))) \\
&\longrightarrow (\lambda n.\ \text{if } n=1 \text{ then true else if } n=0 \text{ then false else not } (((\lambda x.\,H\ (x\ x))\ (\lambda x.\,H\ (x\ x)))\ (n-1))) \\
&=_{\beta} (\lambda n.\ \text{if } n=1 \text{ then true else if } n=0 \text{ then false else not } (ISODD\ (n-1)))
\end{aligned}$$

Here, note that $((\lambda x.\,H\ (x\ x))\ (\lambda x.\,H\ (x\ x)))$ was the result of beta-reducing $Y\ H$ and it has replaced f in the evaluation. Thus, f becomes our recursive call ISODD.

(c) Compute $(Y\ H)\ 2$ under CBN semantics.

Answer: *From above:*

$$\begin{aligned}
&(\lambda n.\ \text{if } n=1 \text{ then true else if } n=0 \text{ then false else not } (ISODD\ (n-1)))\ 2 \\
\longrightarrow\ &\text{if } 2=1 \text{ then true else if } 2=0 \text{ then false else not } (ISODD\ (2-1)) \\
\longrightarrow\ &\text{if false then true else if } 2=0 \text{ then false else not } (ISODD\ (2-1)) \\
\longrightarrow\ &\text{if } 2=0 \text{ then false else not } (ISODD\ (2-1)) \\
\longrightarrow\ &\text{if false then false else not } (ISODD\ (2-1)) \\
\longrightarrow\ &\text{not } (ISODD\ (2-1)) \\
\longrightarrow\ &\text{not } (ISODD\ 1) \\
\longrightarrow^{*}\ &\text{not true} \\
\longrightarrow\ &\text{false}
\end{aligned}$$

(d) Use the "recursion removal trick" to write another function that behaves the same as the fixed point of H.

Answer: *Define a function H'*

$$\begin{aligned}
H' &\triangleq \lambda f.\,\lambda n.\ \text{if } n=1 \text{ then true else if } n=0 \text{ then false else not } (f\ f\ (n-1)) \\
g &= H'\ H'
\end{aligned}$$

3 Evaluation context

Consider the lambda calculus with let expressions and pairs (§1.3 of Lecture 9), and a semantics defined using evaluation contexts. For each of the following expressions, show one step of evaluation. Be clear about what the evaluation context is.

(a) $(\lambda x.\, x)\ (\lambda y.\, y)\ (\lambda z.\, z)$

Answer:

$$E = [\cdot](\lambda z.\, z)$$
$$E[(\lambda x.\, x)\ (\lambda y.\, y)] \longrightarrow (\lambda y.\, y)(\lambda z.\, z)$$

(b) $\mathsf{let}\ x = 5\ \mathsf{in}\ (\lambda y.\, y + x)\ 9$

Answer:

$$E = [\cdot]$$
$$E[\mathit{let}\ x = 5\ \mathit{in}\ (\lambda y.\, y + x)\ 9] \longrightarrow (\lambda y.\, y + 5)\ 9$$

(c) $(4, ((\lambda x.\, x)\ 8, 9))$

Answer:

$$E = (4, ([\cdot], 9))$$
$$E[(\lambda x.\, x)\ 8] \longrightarrow (4, (8, 9))$$

(d) $\mathsf{let}\ x = \#1\ ((\lambda y.\, y)\ (3, 4))\ \mathsf{in}\ x + 2$

Answer:

$$E = \mathit{let}\ x = \#1\ [\cdot]\ \mathit{in}\ x + 2)$$
$$E[(\lambda y.\, y)\ (3, 4)] \longrightarrow \mathit{let}\ x = \#1\ (3, 4)\ \mathit{in}\ x + 2)$$

4 Definitional translations

Consider an applied lambda calculus with booleans, conjunction, a few constant natural numbers, and addition, whose syntax is defined as follows.

$$e ::= x \mid \lambda x.\, e \mid e_1\ e_2 \mid \mathsf{true} \mid \mathsf{false} \mid e_1\ \mathsf{and}\ e_2 \mid 0 \mid 1 \mid 2 \mid e_1 + e_2 \mid \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3$$

Give a translation to the pure lambda calculus. Use the encodings of booleans and natural numbers that we considered in class.

Answer:

$$
\begin{aligned}
\mathcal{T}[\![x]\!] &= x \\
\mathcal{T}[\![\lambda x.\, e]\!] &= \lambda x.\, \mathcal{T}[\![e]\!] \\
\mathcal{T}[\![e_1\ e_2]\!] &= \mathcal{T}[\![e_1]\!]\ \mathcal{T}[\![e_2]\!] \\
\mathcal{T}[\![\mathsf{true}]\!] &= \lambda x.\, \lambda y.\, x \\
\mathcal{T}[\![\mathsf{false}]\!] &= \lambda x.\, \lambda y.\, y \\
\mathcal{T}[\![e_1 \text{ and } e_2]\!] &= (\lambda b_1.\, \lambda b_2.\, b_1\ b_2\ \mathcal{T}[\![\mathsf{false}]\!])\ \mathcal{T}[\![e_1]\!]\ \mathcal{T}[\![e_2]\!] \\
\mathcal{T}[\![0]\!] &= \lambda f.\, \lambda x.\, x \\
\mathcal{T}[\![1]\!] &= \lambda f.\, \lambda x.\, f\ x \\
\mathcal{T}[\![2]\!] &= \lambda f.\, \lambda x.\, f\ (f\ x) \\
\mathcal{T}[\![e_1 + e_2]\!] &= (\lambda n_1.\, \lambda n_2.\, n_1\ \mathcal{T}[\![SUCC]\!]\ n_2)\ \mathcal{T}[\![e_1]\!]\ \mathcal{T}[\![e_2]\!] \\
\mathcal{T}[\![\text{if } e_1 \text{ then } e_2 \text{ else } e_3]\!] &= (\mathcal{T}[\![e_1]\!]\ (\lambda z.\, \mathcal{T}[\![e_2]\!])\ (\lambda z.\, \mathcal{T}[\![e_3]\!]))\ \lambda x.\, x \textit{ where } z \notin FV(\mathcal{T}[\![e_2]\!]) \cup FV(\mathcal{T}[\![e_3]\!])
\end{aligned}
$$

$$
\mathcal{T}[\![SUCC]\!] = \lambda n.\, \lambda f.\, \lambda x.\, f\ (n\ f\ x)
$$

Note that in the translation of if e_1 then e_2 else e_3*, we need to thunk e_2 and e_3 to ensure that only one of them is evaluated.*