Lecture 17
Definitional Translation & Continuations

3 October 2016
Announcements

- Homework 4 returned: out of 41, $\bar{x} = 35.3$, $\sigma = 5.3$, median 36
- Wednesday: Prelim I
- My office hours today: 1–2pm, instead of right after class
Products and Let

Syntax

\[ e ::= x \]
\[ \quad | \lambda x. e \]
\[ \quad | e_1 e_2 \]
\[ \quad | (e_1, e_2) \]
\[ \quad | \#1 e \]
\[ \quad | \#2 e \]
\[ \quad | \text{let } x = e_1 \text{ in } e_2 \]

\[ v ::= \lambda x. e \]
\[ \quad | (v_1, v_2) \]
Evaluation Contexts

\[ E ::= [\cdot] \]
\[ | E \ e \]
\[ | v \ E \]
\[ | (E, e) \]
\[ | (v, E) \]
\[ | \#1 \ E \]
\[ | \#2 \ E \]
\[ | \text{let } x = E \text{ in } e_2 \]
Products and Let

Semantics

\[
\begin{align*}
  e & \rightarrow e' \\
  E[e] & \rightarrow E[e'] \\
  (\lambda x. e) \, v & \rightarrow e\{v/x\} \\
  \#1 \, (v_1, v_2) & \rightarrow v_1 \\
  \#2 \, (v_1, v_2) & \rightarrow v_2 \\
  \text{let } x = v \text{ in } e & \rightarrow e\{v/x\}
\end{align*}
\]
Products and Let

Translation

\[ T[x] = x \]
\[ T[\lambda x. e] = \lambda x. T[e] \]
\[ T[e_1 e_2] = T[e_1] T[e_2] \]
\[ T[(e_1, e_2)] = (\lambda x. \lambda y. \lambda f. f x y) T[e_1] T[e_2] \]
\[ T[#1 e] = T[e] (\lambda x. \lambda y. x) \]
\[ T[#2 e] = T[e] (\lambda x. \lambda y. y) \]
\[ T[\text{let } x = e_1 \text{ in } e_2] = (\lambda x. T[e_2]) T[e_1] \]
Consider the call-by-name $\lambda$-calculus...

**Syntax**

$$ e ::= x $$

$$ | e_1 e_2 $$

$$ | \lambda x. e $$

$$ \nu ::= \lambda x. e $$

**Semantics**

$$ e_1 \rightarrow e'_1 $$

$$ e_1 e_2 \rightarrow e'_1 e_2 $$

$$ (\lambda x. e_1) e_2 \rightarrow e_1[e_2/x] $$
Translation

\[ \mathcal{T}[x] = x (\lambda y. y) \]
\[ \mathcal{T}[\lambda x. e] = \lambda x. \mathcal{T}[e] \]
\[ \mathcal{T}[e_1 e_2] = \mathcal{T}[e_1] (\lambda z. \mathcal{T}[e_2]) \quad \text{z is not a free variable of } e_2 \]
Syntax

\[
e ::= x \\
  \quad \mid \lambda x. e \\
  \quad \mid e_0 \ e_1
\]

\[
\nu ::= \lambda x. e
\]
References

**Syntax**

\[
e ::= x \\
\quad | \lambda x. e \\
\quad | e_0 \ e_1 \\
\quad | \text{ref } e
\]

\[
\nu ::= \lambda x. e
\]
Syntax

\[ e ::= x \]

\[ \mid \lambda x. e \]

\[ \mid e_0 e_1 \]

\[ \mid \text{ref } e \]

\[ \mid !e \]

\[ v ::= \lambda x. e \]
Syntax

\[
e ::= x \\
   | \lambda x. e \\
   | e_0 e_1 \\
   | \text{ref} e \\
   | !e \\
   | e_1 ::= e_2
\]

\[
\nu ::= \lambda x. e
\]
Syntax

\[ e ::= x \]

\[ \quad | \lambda x. e \]

\[ \quad | e_0 e_1 \]

\[ \quad | \text{ref } e \]

\[ \quad | !e \]

\[ \quad | e_1 ::= e_2 \]

\[ \quad | \ell \]

\[ \nu ::= \lambda x. e \]
Syntax

\[
e ::= x \quad | \quad \lambda x. e \quad | \quad e_0 \; e_1 \quad | \quad \text{ref } e \quad | \quad !e \quad | \quad e_1 ::= e_2 \quad | \quad \ell
\]

\[
v ::= \lambda x. e \quad | \quad \ell
\]
Evaluation Contexts

\[ E ::= [ \cdot ] \]
\[ \mid E \cdot e \]
\[ \mid v \cdot E \]
References

Evaluation Contexts

\[
E ::= [\cdot] \\
| E e \\
| v E \\
| \text{ref } E
\]
Evaluation Contexts

\[ E ::= [\cdot] \]
\[ \mid E \ e \]
\[ \mid \nu E \]
\[ \mid \text{ref} E \]
\[ \mid !E \]
Evaluation Contexts

\[ E ::= [.] \]
\[ \mid E e \]
\[ \mid \nu E \]
\[ \mid \text{ref } E \]
\[ \mid !E \]
\[ \mid E ::= e \]
Evaluation Contexts

\[ E ::= [\cdot] \]
\[ | \quad E \ e \]
\[ | \quad v \ E \]
\[ | \quad \text{ref} \ E \]
\[ | \quad !E \]
\[ | \quad E ::= e \]
\[ | \quad v ::= E \]
Semantics

\[
\frac{\langle \sigma, e \rangle \rightarrow \langle \sigma', e' \rangle}{\langle \sigma, E[e] \rangle \rightarrow \langle \sigma', E[e'] \rangle}
\]

\[
\langle \sigma, (\lambda x. e) v \rangle \rightarrow \langle \sigma, e[v/x] \rangle
depth
\]

\[
\frac{\ell \not\in \text{dom}(\sigma)}{\langle \sigma, \text{ref } v \rangle \rightarrow \langle \sigma[\ell \mapsto v], \ell \rangle}
\]

\[
\frac{\sigma(\ell) = v}{\langle \sigma, !\ell \rangle \rightarrow \langle \sigma, v \rangle}
\]

\[
\frac{\langle \sigma, \ell := v \rangle \rightarrow \langle \sigma[\ell \mapsto v], v \rangle}{\}
\]
Translation

...left as an exercise to the reader. ;-)
Adequacy

How do we know if a translation is correct?
Adequacy

How do we know if a translation is correct?

Every target evaluation should represent a source evaluation...

**Definition (Soundness)**

\[ \forall e \in \text{Exp}_{\text{src}} . \text{if } \mathcal{T}[e] \to^*_{\text{trg}} v' \text{ then } \exists v . e \to^*_{\text{src}} v \]

and \( v' \) equivalent to \( v \)
Adequacy

How do we know if a translation is correct?

Every target evaluation should represent a source evaluation...

**Definition (Soundness)**

\[ \forall e \in \text{Exp}_{\text{src}}. \text{if } T[e] \rightarrow^*_{\text{trg}} v' \text{ then } \exists \nu. e \rightarrow^*_{\text{src}} \nu \]

and \( v' \) equivalent to \( \nu \)

...and every source evaluation should have a target evaluation:

**Definition (Completeness)**

\[ \forall e \in \text{Exp}_{\text{src}}. \text{if } e \rightarrow^*_{\text{src}} \nu \text{ then } \exists \nu'. T[e] \rightarrow^*_{\text{trg}} \nu' \]

and \( \nu' \) equivalent to \( \nu \)
Continuations

In the preceding translations, the control structure of the source language was translated directly into the corresponding control structure in the target language.

For example:

\[ T[\lambda x. e] = \lambda x. T[e] \]
\[ T[e_1 e_2] = T[e_1] T[e_2] \]

What can go wrong with this approach?
Continuations

- A snippet of code that represents “the rest of the program”
- Can be used directly by programmers...
- ...or in program transformations by a compiler
- Make the control flow of the program explicit
- Also useful for defining the meaning of features like exceptions
Consider the following expression:

\[(\lambda x. x) ((1 + 2) + 3) + 4\]
Example

Consider the following expression:

\[(\lambda x. x) \ (\ (1 + 2) + 3 ) + 4\]

If we make all of the continuations explicit, we obtain the following:

\[k_0 = \lambda \nu. (\lambda x. x) \ \nu\]
Consider the following expression:

\[(\lambda x. x) \left( (1 + 2) + 3 \right) + 4 \]

If we make all of the continuations explicit, we obtain the following:

\[ k_0 = \lambda v. (\lambda x. x) \, v \]
Example

Consider the following expression:

$$(\lambda x. x) ((1 + 2) + 3) + 4$$

If we make all of the continuations explicit, we obtain the following:

$$k_0 = \lambda v. (\lambda x. x) v$$
$$k_1 = \lambda a. k_0 (a + 4)$$
Consider the following expression:

$$(\lambda x. x) ((1 + 2) + 3) + 4$$

If we make all of the continuations explicit, we obtain the following:

$$k_0 = \lambda v. (\lambda x. x) v$$
$$k_1 = \lambda a. k_0 (a + 4)$$
$$k_2 = \lambda b. k_1 (b + 3)$$
Consider the following expression:

\[(\lambda x. x) \left((1 + 2) + 3\right) + 4\]

If we make all of the continuations explicit, we obtain the following:

\[
k_0 = \lambda \nu. \left(\lambda x. x\right) \nu \]
\[
k_1 = \lambda a. k_0 \left(a + 4\right) \]
\[
k_2 = \lambda b. k_1 \left(b + 3\right) \]
\[
k_3 = \lambda c. k_2 \left(c + 2\right) \]
Example

Consider the following expression:

\[(\lambda x. x) \ ((1 + 2) + 3) + 4\]

If we make all of the continuations explicit, we obtain the following:

\[
k_0 = \lambda v. \ (\lambda x. x) \ v
\]
\[
k_1 = \lambda a. \ k_0 \ (a + 4)
\]
\[
k_2 = \lambda b. \ k_1 \ (b + 3)
\]
\[
k_3 = \lambda c. \ k_2 \ (c + 2)
\]

The original expression is equivalent to \( k_3 \ 1 \), or:

\[
(\lambda c. \ (\lambda b. \ (\lambda a. \ (\lambda v. \ (\lambda x. x) \ v) \ (a + 4)) \ (b + 3)) \ (c + 2)) \ 1
\]
Example (Continued)

Recall that let \( x = e \) in \( e' \) is syntactic sugar for \((\lambda x. e') e\).

Hence, we can rewrite the expression with continuations more succinctly as

\[
\begin{align*}
\text{let } c &= 1 \text{ in} \\
\text{let } b &= c + 2 \text{ in} \\
\text{let } a &= b + 3 \text{ in} \\
\text{let } v &= a + 4 \text{ in} \\
(\lambda x. x) v
\end{align*}
\]
CPS Transformation

We write $\text{CPS}[e]k = \ldots$ instead of $\text{CPS}[e] = \lambda k. \ldots$

We assume that the new variables introduced are “fresh”.
CPS Transformation

\[ CPS[n] k = k n \]

We write \( CPS[e] k = \ldots \) instead of \( CPS[e] = \lambda k. \ldots \)

We assume that the new variables introduced are “fresh”.
CPS Transformation

\[ CPS[n] \ k = k \ n \]
\[ CPS[e_1 + e_2] \ k = CPS[e_1] (\lambda n. CPS[e_2] (\lambda m. k (n + m))) \]

We write \( CPS[e] \ k = \ldots \) instead of \( CPS[e] = \lambda k. \ldots \)

We assume that the new variables introduced are “fresh”.
CPS Transformation

\[
\begin{align*}
CPS[n] k &= kn \\
CPS[e_1 + e_2] k &= CPS[e_1] (\lambda n. CPS[e_2] (\lambda m. k (n + m)))
\end{align*}
\]

We write \( CPS[e] k = \ldots \) instead of \( CPS[e] = \lambda k. \ldots \)

We assume that the new variables introduced are “fresh”.

CPS Transformation

\[ CPS[n] k = k \, n \]
\[ CPS[e_1 + e_2] k = CPS[e_1] (\lambda n. CPS[e_2] (\lambda m. k \, (n + m))) \]
\[ CPS[(e_1, e_2)] k = CPS[e_1] (\lambda v. CPS[e_2] (\lambda w. k \, (v, w))) \]
\[ CPS[#1 \, e] k = CPS[e] (\lambda v. k \, (#1 \, v)) \]

We write \( CPS[e] k = \ldots \) instead of \( CPS[e] = \lambda k. \ldots \)

We assume that the new variables introduced are “fresh”.
CPS Transformation

\[ \text{CPS}[n] k = kn \]
\[ \text{CPS}[e_1 + e_2] k = \text{CPS}[e_1] (\lambda n. \text{CPS}[e_2] (\lambda m. k (n + m))) \]
\[ \text{CPS}[(e_1, e_2)] k = \text{CPS}[e_1] (\lambda v. \text{CPS}[e_2] (\lambda w. k (v, w))) \]
\[ \text{CPS}[\#1 e] k = \text{CPS}[e] (\lambda v. k (\#1 v)) \]
\[ \text{CPS}[\#2 e] k = \text{CPS}[e] (\lambda v. k (\#2 v)) \]

We write \( \text{CPS}[e] k = \ldots \) instead of \( \text{CPS}[e] = \lambda k. \ldots \)

We assume that the new variables introduced are “fresh”.
CPS Transformation

\[
\text{CPS}[n] \; k = kn \\
\text{CPS}[e_1 + e_2] \; k = \text{CPS}[e_1] (\lambda n. \text{CPS}[e_2] (\lambda m. k (n + m))) \\
\text{CPS}[(e_1, e_2)] \; k = \text{CPS}[e_1] (\lambda v. \text{CPS}[e_2] (\lambda w. k (v, w))) \\
\text{CPS}[#1 e] \; k = \text{CPS}[e] (\lambda v. k (#1 v)) \\
\text{CPS}[#2 e] \; k = \text{CPS}[e] (\lambda v. k (#2 v)) \\
\text{CPS}[x] \; k = kx
\]

We write \(\text{CPS}[e] \; k = \ldots\) instead of \(\text{CPS}[e] = \lambda k. \ldots\)

We assume that the new variables introduced are “fresh”.

CPS Transformation

\[
\begin{align*}
\text{CPS}[n] \; k &= kn \\
\text{CPS}[e_1 + e_2] \; k &= \text{CPS}[e_1] \; (\lambda n. \text{CPS}[e_2] \; (\lambda m. k \; (n + m))) \\
\text{CPS}[(e_1, e_2)] \; k &= \text{CPS}[e_1] \; (\lambda v. \text{CPS}[e_2] \; (\lambda w. k \; (v, w))) \\
\text{CPS}[\#1 \; e] \; k &= \text{CPS}[e] \; (\lambda v. k \; (#1 \; v)) \\
\text{CPS}[\#2 \; e] \; k &= \text{CPS}[e] \; (\lambda v. k \; (#2 \; v)) \\
\text{CPS}[x] \; k &= k \; x \\
\text{CPS}[\lambda x. \; e] \; k &= k \; (\lambda x. \; \lambda k'. \text{CPS}[e] \; k')
\end{align*}
\]

We write \( \text{CPS}[e] \; k = \ldots \) instead of \( \text{CPS}[e] = \lambda k. \ldots \)

We assume that the new variables introduced are “fresh”.
CPS Transformation

\[ CPS[n] k = kn \]
\[ CPS[e_1 + e_2] k = CPS[e_1] (\lambda n. CPS[e_2] (\lambda m. k (n + m))) \]
\[ CPS[(e_1, e_2)] k = CPS[e_1] (\lambda v. CPS[e_2] (\lambda w. k (v, w))) \]
\[ CPS[#1 e] k = CPS[e] (\lambda v. k (#1 v)) \]
\[ CPS[#2 e] k = CPS[e] (\lambda v. k (#2 v)) \]
\[ CPS[x] k = kx \]
\[ CPS[\lambda x. e] k = k (\lambda x. \lambda k'. CPS[e] k') \]
\[ CPS[e_1 e_2] k = CPS[e_1] (\lambda f. CPS[e_2] (\lambda v. f v k)) \]

We write \( CPS[e] k = \ldots \) instead of \( CPS[e] = \lambda k. \ldots \)

We assume that the new variables introduced are “fresh”.