Lecture 19
Continuations
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:

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

Consider the following expression:

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

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

$$k_0 = \lambda v. (\lambda x. x) v$$
Example

Consider the following expression:

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

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

$$k_0 = \lambda v. (\lambda x. x) v$$

$$k_1 = \lambda a. k_0 (a + 4)$$
Example

Consider the following expression:

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

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

$$k_0 = \lambda v. (\lambda x. x) \ v$$
$$k_1 = \lambda a. k_0 \ (a + 4)$$
$$k_2 = \lambda b. k_1 \ (b + 3)$$
Example

Consider the following expression:

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

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

- $$k_0 = \lambda v. \ (\lambda x. x) \ v$$
- $$k_1 = \lambda a. \ k_0 \ (a + 4)$$
- $$k_2 = \lambda b. \ k_1 \ (b + 3)$$
- $$k_3 = \lambda c. \ k_2 \ (c + 2)$$
Example

Consider the following expression:

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

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

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

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

\[
\begin{align*}
&\text{let } c = 1 \text{ in} \\
&\quad \text{let } b = c + 2 \text{ in} \\
&\quad \text{let } a = b + 3 \text{ in} \\
&\quad \text{let } v = a + 4 \text{ in} \\
&\quad (\lambda x. x) v
\end{align*}
\]
We write $CPS[e] \ k = \ldots$ instead of $CPS[e] = \lambda k. \ldots$

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

\[ \text{CPS}[n] \ k = kn \]

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

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

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

We assume that the new variables introduced are “fresh.”
\[
CP\!\!S[n] k = kn
\]
\[
CP\!\!S[e_1 + e_2] k = CP\!\!S[e_1] (\lambda n. CP\!\!S[e_2] (\lambda m. k (n + m)))
\]
\[
CP\!\!S[(e_1, e_2)] k = CP\!\!S[e_1] (\lambda v. CP\!\!S[e_2] (\lambda w. k (v, w)))
\]

We write \( CP\!\!S[e] k = \ldots \) instead of \( CP\!\!S[e] = \lambda k. \ldots \)

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

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

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

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

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

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

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

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

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

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

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

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