Lecture 19
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:

\[
\mathcal{T}\left[\lambda x. e\right] = \lambda x. \mathcal{T}\left[e\right]
\]
\[
\mathcal{T}\left[e_1 e_2\right] = \mathcal{T}\left[e_1\right] \mathcal{T}\left[e_2\right]
\]

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
Example

Consider the following expression:

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

Consider the following expression:

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

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

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

Consider the following expression:

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

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:

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

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

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

Consider the following expression:

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

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

\[
\begin{align*}
k_0 &= \lambda v. \ (\lambda x. \ x) \ v \\
k_1 &= \lambda a. \ k_0 \ (a - 4) \\
k_2 &= \lambda b. \ k_1 \ (3 \times b) \\
k_3 &= \lambda c. \ k_2 \ (c + 2)
\end{align*}
\]
Example

Consider the following expression:

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

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

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

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

\[(\lambda c. (\lambda b. (\lambda a. (\lambda v. (\lambda x. x) \; v) \; (a - 4)) \; (3 \ast b)) \; (c + 2)) \; 1\]
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

```
let c = 1 in
let b = c + 2 in
let a = 3 * b in
let v = a - 4 in
(\lambda x. x) v
```
CPS Transformation

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

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

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

\[ CPS[n] k = kn \]
\[ CPS[x] k = kx \]

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

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

\[
\text{CPS}[n] k = kn \\
\text{CPS}[x] k = kx \\
\text{CPS}[\text{succ } e] k = \text{CPS}[e] (\lambda n. k (\text{succ } n))
\]

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

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

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

\[
\begin{align*}
CPS[n] k &= kn \\
CPS[x] k &= kx \\
CPS[\text{succ } e] k &= CPS[e] (\lambda n. k (\text{succ } n)) \\
CPS[e_1 + e_2] k &= CPS[e_1] (\lambda n. CPS[e_2] (\lambda m. k (n + m))) \\
CPS[\lambda x. e] k &= k (\lambda x. \lambda k'. CPS[e] k')
\end{align*}
\]

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

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

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

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

We assume that the new variables introduced are “fresh.”
We can also translate other language features, like products:

\[ e ::= \cdots \mid (e_1, e_2) \mid \#1\, e \mid \#2\, e \]
CPS Transformation, Extended

We can also translate other language features, like products:

\[ e ::= \cdots \mid (e_1, e_2) \mid \#1 e \mid \#2 e \]

\[
\begin{align*}
\mathcal{CPS}[(e_1, e_2)](\lambda v. \mathcal{CPS}[e_2](\lambda w. k(v, w))) &= \\mathcal{CPS}[e_1] \mathcal{CPS}[e_2] \mathcal{CPS}(\lambda v. \lambda w. k(v, w))
\end{align*}
\]
CPS Transformation, Extended

We can also translate other language features, like products:

\[ e ::= \cdots \mid (e_1, e_2) \mid \#1 \, e \mid \#2 \, e \]

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

We can also translate other language features, like products:

\[ e ::= \cdots | (e_1, e_2) | \#1 \ e | \#2 \ e \]

\[
\begin{align*}
\text{CPS}[(e_1, e_2)] \ k &= \text{CPS}[e_1] (\lambda v. \text{CPS}[e_2] (\lambda w. k (v, w))) \\
\text{CPS}[\#1 \ e] \ k &= \text{CPS}[e] (\lambda v. k (\#1 v)) \\
\text{CPS}[\#2 \ e] \ k &= \text{CPS}[e] (\lambda v. k (\#2 v))
\end{align*}
\]