CS 4110

Programming Languages & Logics

Lecture 20

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:

$$\begin{aligned}
\mathcal{T}[\![\lambda x.\, e]\!] &= \lambda x.\, \mathcal{T}[\![e]\!] \\
\mathcal{T}[\![e_1\; e_2]\!] &= \mathcal{T}[\![e_1]\!]\; \mathcal{T}[\![e_2]\!]
\end{aligned}$$

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:

$$(\lambda x.\, x)\; ((3 * (1 + 2)) - 4)$$

Example

Consider the following expression:

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

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

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

Example

Consider the following expression:

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

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

$$\begin{aligned} k_0 &= \lambda v.\, (\lambda x.\, x)\, v \\ k_1 &= \lambda a.\, k_0\, (a - 4) \end{aligned}$$

Example

Consider the following expression:

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

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

$$\begin{aligned} k_0 &= \lambda v.\, (\lambda x.\, x)\, v \\ k_1 &= \lambda a.\, k_0\, (a - 4) \\ k_2 &= \lambda b.\, k_1\, (3 * b) \end{aligned}$$

Example

Consider the following expression:

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

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

$$\begin{aligned}
k_0 &= \lambda v.\, (\lambda x.\, x)\, v \\
k_1 &= \lambda a.\, k_0\, (a - 4) \\
k_2 &= \lambda b.\, k_1\, (3 * b) \\
k_3 &= \lambda c.\, k_2\, (c + 2)
\end{aligned}$$

Example

Consider the following expression:

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

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

$$\begin{aligned}
k_0 &= \lambda v.\, (\lambda x.\, x)\, v \\
k_1 &= \lambda a.\, k_0\, (a - 4) \\
k_2 &= \lambda b.\, k_1\, (3 * b) \\
k_3 &= \lambda c.\, k_2\, (c + 2)
\end{aligned}$$

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

$$(\lambda c.\, (\lambda b.\, (\lambda a.\, (\lambda v.\, (\lambda x.\, x)\, v)\, (a - 4))\, (3 * b))\, (c + 2))\ 1$$

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

$$\begin{array}{l} \text{let } c = 1 \text{ in} \\ \text{let } b = c + 2 \text{ in} \\ \text{let } a = 3 * b \text{ in} \\ \text{let } v = a - 4 \text{ in} \\ (\lambda x.\, x)\, v \end{array}$$

CPS Transformation

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

We assume that the new variables introduced are "fresh."

CPS Transformation

$$\mathcal{CPS}[\![n]\!]\, k = k\, n$$

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

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

CPS Transformation

$$\mathcal{CPS}[\![n]\!]\,k = k\,n$$
$$\mathcal{CPS}[\![x]\!]\,k = k\,x$$

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

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

CPS Transformation

$$\begin{aligned}
\mathcal{CPS}[\![n]\!]\, k &= k\, n \\
\mathcal{CPS}[\![x]\!]\, k &= k\, x \\
\mathcal{CPS}[\![\mathsf{succ}\; e]\!]\, k &= \mathcal{CPS}[\![e]\!]\, (\lambda n.\, k\, (\mathsf{succ}\; n))
\end{aligned}$$

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

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

CPS Transformation

$$
\begin{aligned}
\mathcal{CPS}[\![n]\!]\,k &= k\,n \\
\mathcal{CPS}[\![x]\!]\,k &= k\,x \\
\mathcal{CPS}[\![\mathrm{succ}\;e]\!]\,k &= \mathcal{CPS}[\![e]\!]\,(\lambda n.\,k\,(\mathrm{succ}\;n)) \\
\mathcal{CPS}[\![e_1 + e_2]\!]\,k &= \mathcal{CPS}[\![e_1]\!]\,(\lambda n.\,\mathcal{CPS}[\![e_2]\!]\,(\lambda m.\,k\,(n + m)))
\end{aligned}
$$

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

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

CPS Transformation

$$
\begin{aligned}
\mathcal{CPS}[\![n]\!]\,k &= k\,n \\
\mathcal{CPS}[\![x]\!]\,k &= k\,x \\
\mathcal{CPS}[\![\mathsf{succ}\ e]\!]\,k &= \mathcal{CPS}[\![e]\!]\,(\lambda n.\,k\,(\mathsf{succ}\ n)) \\
\mathcal{CPS}[\![e_1 + e_2]\!]\,k &= \mathcal{CPS}[\![e_1]\!]\,(\lambda n.\,\mathcal{CPS}[\![e_2]\!]\,(\lambda m.\,k\,(n+m))) \\
\mathcal{CPS}[\![\lambda x.\,e]\!]\,k &= k\,(\lambda x.\,\lambda k'.\,\mathcal{CPS}[\![e]\!]\,k')
\end{aligned}
$$

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

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

CPS Transformation

$$
\begin{aligned}
\mathcal{CPS}[\![n]\!]\,k &= k\,n \\
\mathcal{CPS}[\![x]\!]\,k &= k\,x \\
\mathcal{CPS}[\![\mathsf{succ}\,e]\!]\,k &= \mathcal{CPS}[\![e]\!]\,(\lambda n.\,k\,(\mathsf{succ}\,n)) \\
\mathcal{CPS}[\![e_1 + e_2]\!]\,k &= \mathcal{CPS}[\![e_1]\!]\,(\lambda n.\,\mathcal{CPS}[\![e_2]\!]\,(\lambda m.\,k\,(n+m))) \\
\mathcal{CPS}[\![\lambda x.\,e]\!]\,k &= k\,(\lambda x.\,\lambda k'.\,\mathcal{CPS}[\![e]\!]\,k') \\
\mathcal{CPS}[\![e_1\,e_2]\!]\,k &= \mathcal{CPS}[\![e_1]\!]\,(\lambda f.\,\mathcal{CPS}[\![e_2]\!]\,(\lambda v.\,f\,v\,k))
\end{aligned}
$$

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

We assume that the new variables introduced are "fresh."

CPS Transformation, Extended

We can also translate other language features, like products:

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

CPS Transformation, Extended

We can also translate other language features, like products:

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

$$\mathcal{CPS}[\![(e_1, e_2)]\!]\, k = \mathcal{CPS}[\![e_1]\!]\, (\lambda v.\, \mathcal{CPS}[\![e_2]\!]\, (\lambda w.\, k\, (v, w)))$$

CPS Transformation, Extended

We can also translate other language features, like products:

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

$$\begin{aligned}\mathcal{CPS}[\![(e_1, e_2)]\!]\, k &= \mathcal{CPS}[\![e_1]\!]\, (\lambda v.\, \mathcal{CPS}[\![e_2]\!]\, (\lambda w.\, k\, (v, w)))\\ \mathcal{CPS}[\![\#1\, e]\!]\, k &= \mathcal{CPS}[\![e]\!]\, (\lambda v.\, k\, (\#1\, v))\end{aligned}$$

CPS Transformation, Extended

We can also translate other language features, like products:

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

$$\begin{aligned}
\mathcal{CPS}[\![(e_1, e_2)]\!]\, k &= \mathcal{CPS}[\![e_1]\!]\, (\lambda v.\, \mathcal{CPS}[\![e_2]\!]\, (\lambda w.\, k\, (v, w))) \\
\mathcal{CPS}[\![\#1\, e]\!]\, k &= \mathcal{CPS}[\![e]\!]\, (\lambda v.\, k\, (\#1\, v)) \\
\mathcal{CPS}[\![\#2\, e]\!]\, k &= \mathcal{CPS}[\![e]\!]\, (\lambda v.\, k\, (\#2\, v))
\end{aligned}$$