

CS 4110

Programming Languages & Logics

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:

$$\mathcal{T}[\lambda x. e] = \lambda x. \mathcal{T}[e]$$

$$\mathcal{T}[e_1 e_2] = \mathcal{T}[e_1] \mathcal{T}[e_2]$$

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

Example

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:

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

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:

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

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

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

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

$$k_2 = \lambda b. k_1 (b + 3)$$

$$k_3 = \lambda c. k_2 (c + 2)$$

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 $\text{let } x = e \text{ in } e'$ 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 $a = b + 3$ in
let $v = a + 4$ in
 $(\lambda x. x) v$

CPS Transformation

We write $\mathcal{CPS}\llbracket e \rrbracket k = \dots$ instead of $\mathcal{CPS}\llbracket e \rrbracket = \lambda k. \dots$

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

CPS Transformation

$$\mathcal{CPS}\llbracket n \rrbracket k = k\ n$$

We write $\mathcal{CPS}\llbracket e \rrbracket k = \dots$ instead of $\mathcal{CPS}\llbracket e \rrbracket = \lambda k. \dots$

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

CPS Transformation

$$\mathcal{CPS}\llbracket n \rrbracket k = k\ n$$

$$\mathcal{CPS}\llbracket e_1 + e_2 \rrbracket k = \mathcal{CPS}\llbracket e_1 \rrbracket (\lambda n. \mathcal{CPS}\llbracket e_2 \rrbracket (\lambda m. k\ (n + m)))$$

We write $\mathcal{CPS}\llbracket e \rrbracket k = \dots$ instead of $\mathcal{CPS}\llbracket e \rrbracket = \lambda k. \dots$

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

CPS Transformation

$$\mathcal{CPS}\llbracket n \rrbracket k = k\ n$$

$$\mathcal{CPS}\llbracket e_1 + e_2 \rrbracket k = \mathcal{CPS}\llbracket e_1 \rrbracket (\lambda n. \mathcal{CPS}\llbracket e_2 \rrbracket (\lambda m. k\ (n + m)))$$

$$\mathcal{CPS}\llbracket (e_1, e_2) \rrbracket k = \mathcal{CPS}\llbracket e_1 \rrbracket (\lambda v. \mathcal{CPS}\llbracket e_2 \rrbracket (\lambda w. k\ (v, w)))$$

We write $\mathcal{CPS}\llbracket e \rrbracket k = \dots$ instead of $\mathcal{CPS}\llbracket e \rrbracket = \lambda k. \dots$

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

CPS Transformation

$$\mathcal{CPS}\llbracket n \rrbracket k = k\ n$$

$$\mathcal{CPS}\llbracket e_1 + e_2 \rrbracket k = \mathcal{CPS}\llbracket e_1 \rrbracket (\lambda n. \mathcal{CPS}\llbracket e_2 \rrbracket (\lambda m. k\ (n + m)))$$

$$\mathcal{CPS}\llbracket (e_1, e_2) \rrbracket k = \mathcal{CPS}\llbracket e_1 \rrbracket (\lambda v. \mathcal{CPS}\llbracket e_2 \rrbracket (\lambda w. k\ (v, w)))$$

$$\mathcal{CPS}\llbracket \#1\ e \rrbracket k = \mathcal{CPS}\llbracket e \rrbracket (\lambda v. k\ (\#1\ v))$$

We write $\mathcal{CPS}\llbracket e \rrbracket k = \dots$ instead of $\mathcal{CPS}\llbracket e \rrbracket = \lambda k. \dots$

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

CPS Transformation

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

$$\mathcal{CPS}[e_1 + e_2] \ k = \mathcal{CPS}[e_1] \ (\lambda n. \mathcal{CPS}[e_2] \ (\lambda m. k \ (n + m)))$$

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

$$\mathcal{CPS}[\#1 \ e] \ k = \mathcal{CPS}[e] \ (\lambda v. k \ (\#1 \ v))$$

$$\mathcal{CPS}[\#2 \ e] \ k = \mathcal{CPS}[e] \ (\lambda v. k \ (\#2 \ v))$$

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

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

CPS Transformation

$$\mathcal{CPS}\llbracket n \rrbracket k = k\ n$$

$$\mathcal{CPS}\llbracket e_1 + e_2 \rrbracket k = \mathcal{CPS}\llbracket e_1 \rrbracket (\lambda n. \mathcal{CPS}\llbracket e_2 \rrbracket (\lambda m. k\ (n + m)))$$

$$\mathcal{CPS}\llbracket (e_1, e_2) \rrbracket k = \mathcal{CPS}\llbracket e_1 \rrbracket (\lambda v. \mathcal{CPS}\llbracket e_2 \rrbracket (\lambda w. k\ (v, w)))$$

$$\mathcal{CPS}\llbracket \#1\ e \rrbracket k = \mathcal{CPS}\llbracket e \rrbracket (\lambda v. k\ (\#1\ v))$$

$$\mathcal{CPS}\llbracket \#2\ e \rrbracket k = \mathcal{CPS}\llbracket e \rrbracket (\lambda v. k\ (\#2\ v))$$

$$\mathcal{CPS}\llbracket x \rrbracket k = k\ x$$

We write $\mathcal{CPS}\llbracket e \rrbracket k = \dots$ instead of $\mathcal{CPS}\llbracket e \rrbracket = \lambda k. \dots$

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

CPS Transformation

$$\mathcal{CPS}\llbracket n \rrbracket k = k\ n$$

$$\mathcal{CPS}\llbracket e_1 + e_2 \rrbracket k = \mathcal{CPS}\llbracket e_1 \rrbracket (\lambda n. \mathcal{CPS}\llbracket e_2 \rrbracket (\lambda m. k\ (n + m)))$$

$$\mathcal{CPS}\llbracket (e_1, e_2) \rrbracket k = \mathcal{CPS}\llbracket e_1 \rrbracket (\lambda v. \mathcal{CPS}\llbracket e_2 \rrbracket (\lambda w. k\ (v, w)))$$

$$\mathcal{CPS}\llbracket \#1\ e \rrbracket k = \mathcal{CPS}\llbracket e \rrbracket (\lambda v. k\ (\#1\ v))$$

$$\mathcal{CPS}\llbracket \#2\ e \rrbracket k = \mathcal{CPS}\llbracket e \rrbracket (\lambda v. k\ (\#2\ v))$$

$$\mathcal{CPS}\llbracket x \rrbracket k = k\ x$$

$$\mathcal{CPS}\llbracket \lambda x. e \rrbracket k = k\ (\lambda x. \lambda k'. \mathcal{CPS}\llbracket e \rrbracket k')$$

We write $\mathcal{CPS}\llbracket e \rrbracket k = \dots$ instead of $\mathcal{CPS}\llbracket e \rrbracket = \lambda k. \dots$

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

CPS Transformation

$$\mathcal{CPS}\llbracket n \rrbracket k = k\ n$$

$$\mathcal{CPS}\llbracket e_1 + e_2 \rrbracket k = \mathcal{CPS}\llbracket e_1 \rrbracket (\lambda n. \mathcal{CPS}\llbracket e_2 \rrbracket (\lambda m. k\ (n + m)))$$

$$\mathcal{CPS}\llbracket (e_1, e_2) \rrbracket k = \mathcal{CPS}\llbracket e_1 \rrbracket (\lambda v. \mathcal{CPS}\llbracket e_2 \rrbracket (\lambda w. k\ (v, w)))$$

$$\mathcal{CPS}\llbracket \#1\ e \rrbracket k = \mathcal{CPS}\llbracket e \rrbracket (\lambda v. k\ (\#1\ v))$$

$$\mathcal{CPS}\llbracket \#2\ e \rrbracket k = \mathcal{CPS}\llbracket e \rrbracket (\lambda v. k\ (\#2\ v))$$

$$\mathcal{CPS}\llbracket x \rrbracket k = k\ x$$

$$\mathcal{CPS}\llbracket \lambda x. e \rrbracket k = k\ (\lambda x. \lambda k'. \mathcal{CPS}\llbracket e \rrbracket k')$$

$$\mathcal{CPS}\llbracket e_1\ e_2 \rrbracket k = \mathcal{CPS}\llbracket e_1 \rrbracket (\lambda f. \mathcal{CPS}\llbracket e_2 \rrbracket (\lambda v. f\ v\ k))$$

We write $\mathcal{CPS}\llbracket e \rrbracket k = \dots$ instead of $\mathcal{CPS}\llbracket e \rrbracket = \lambda k. \dots$

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