

CS 4110

Programming Languages & Logics

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

Products and Let

Syntax

$$\begin{aligned} e ::= & x \\ & | \lambda x. e \\ & | e_1 e_2 \\ & | (e_1, e_2) \\ & | \#1 e \\ & | \#2 e \\ & | \text{let } x = e_1 \text{ in } e_2 \\ v ::= & \lambda x. e \\ & | (v_1, v_2) \end{aligned}$$

Products and Let

Evaluation Contexts

$$\begin{aligned} E ::= & [\cdot] \\ & | E e \\ & | v E \\ & | (E, e) \\ & | (v, E) \\ & | \#1 E \\ & | \#2 E \\ & | \text{let } x = E \text{ in } e_2 \end{aligned}$$

Products and Let

Semantics

$$\frac{e \rightarrow e'}{E[e] \rightarrow E[e']}$$

$$\overline{(\lambda x. e) v \rightarrow e\{v/x\}}^{\beta}$$

$$\overline{\#1 (v_1, v_2) \rightarrow v_1}$$

$$\overline{\#2 (v_1, v_2) \rightarrow v_2}$$

$$\overline{\text{let } x = v \text{ in } e \rightarrow e\{v/x\}}$$

Products and Let

Translation

$$\mathcal{T}[\![x]\!] = x$$

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

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

$$\mathcal{T}[\![(e_1, e_2)]\!] = (\lambda x. \lambda y. \lambda f. f x y) \mathcal{T}[\![e_1]\!] \mathcal{T}[\![e_2]\!]$$

$$\mathcal{T}[\![\#1 e]\!] = \mathcal{T}[\![e]\!] (\lambda x. \lambda y. x)$$

$$\mathcal{T}[\![\#2 e]\!] = \mathcal{T}[\![e]\!] (\lambda x. \lambda y. y)$$

$$\mathcal{T}[\![\text{let } x = e_1 \text{ in } e_2]\!] = (\lambda x. \mathcal{T}[\![e_2]\!]) \mathcal{T}[\![e_1]\!]$$

Laziness

Consider the call-by-name λ -calculus...

Syntax

$$\begin{aligned} e &::= x \\ &\quad | e_1 e_2 \\ &\quad | \lambda x. e \\ v &::= \lambda x. e \end{aligned}$$

Semantics

$$\frac{e_1 \rightarrow e'_1}{e_1 e_2 \rightarrow e'_1 e_2}$$

$$\frac{}{(\lambda x. e_1) e_2 \rightarrow e_1 \{e_2/x\}} \beta$$

Laziness

Translation

$$\mathcal{T}[\![x]\!] = x (\lambda y. y)$$

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

$$\mathcal{T}[\![e_1 e_2]\!] = \mathcal{T}[\![e_1]\!] (\lambda z. \mathcal{T}[\![e_2]\!]) \quad z \text{ is not a free variable of } e_2$$

References

Syntax

$$\begin{aligned} e ::= & x \\ & | \lambda x. e \\ & | e_0 e_1 \end{aligned}$$
$$v ::= \lambda x. e$$

References

Syntax

$$\begin{aligned} e ::= & x \\ & | \lambda x. e \\ & | e_0 e_1 \\ & | \text{ref } e \end{aligned}$$
$$v ::= \lambda x. e$$

References

Syntax

$$\begin{aligned} e ::= & x \\ & | \lambda x. e \\ & | e_0 e_1 \\ & | \text{ref } e \\ & | !e \end{aligned}$$
$$v ::= \lambda x. e$$

References

Syntax

$$\begin{aligned} e ::= & x \\ & | \lambda x. e \\ & | e_0 e_1 \\ & | \text{ref } e \\ & | !e \\ & | e_1 := e_2 \end{aligned}$$
$$v ::= \lambda x. e$$

References

Syntax

$$\begin{aligned} e ::= & x \\ & | \lambda x. e \\ & | e_0 e_1 \\ & | \text{ref } e \\ & | !e \\ & | e_1 := e_2 \\ & | \ell \end{aligned}$$
$$v ::= \lambda x. e$$

References

Syntax

$$\begin{aligned} e ::= & x \\ & | \lambda x. e \\ & | e_0 e_1 \\ & | \text{ref } e \\ & | !e \\ & | e_1 := e_2 \\ & | \ell \end{aligned}$$
$$\begin{aligned} v ::= & \lambda x. e \\ & | \ell \end{aligned}$$

References

Evaluation Contexts

$$\begin{aligned} E ::= & [\cdot] \\ & | E e \\ & | v E \end{aligned}$$

References

Evaluation Contexts

$$\begin{aligned} E ::= & [\cdot] \\ & | E e \\ & | v E \\ & | \text{ref } E \end{aligned}$$

References

Evaluation Contexts

$$\begin{aligned} E ::= & [\cdot] \\ & | E e \\ & | v E \\ & | \text{ref } E \\ & | !E \end{aligned}$$

References

Evaluation Contexts

$$\begin{aligned} E ::= & [\cdot] \\ & | E e \\ & | v E \\ & | \text{ref } E \\ & | !E \\ & | E := e \end{aligned}$$

References

Evaluation Contexts

$$\begin{aligned} E ::= & [\cdot] \\ & | E e \\ & | v E \\ & | \text{ref } E \\ & | !E \\ & | E := e \\ & | v := E \end{aligned}$$

References

Semantics

$$\frac{\langle \sigma, e \rangle \rightarrow \langle \sigma', e' \rangle}{\langle \sigma, E[e] \rangle \rightarrow \langle \sigma', E[e'] \rangle} \qquad \frac{}{\langle \sigma, (\lambda x. e) \, v \rangle \rightarrow \langle \sigma, e\{v/x\} \rangle} \beta$$

$$\frac{\ell \notin \text{dom}(\sigma)}{\langle \sigma, \text{ref } v \rangle \rightarrow \langle \sigma[\ell \mapsto v], \ell \rangle} \qquad \frac{\sigma(\ell) = v}{\langle \sigma, !\ell \rangle \rightarrow \langle \sigma, v \rangle}$$

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

References

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)

$\forall e \in \mathbf{Exp}_{\text{src}}.$ if $\mathcal{T}[\![e]\!] \rightarrow_{\text{trg}}^* v'$ then $\exists v. e \rightarrow_{\text{src}}^* v$
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)

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

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

Definition (Completeness)

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

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 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:

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

Example

Consider the following expression:

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

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

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

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

Example

Consider the following expression:

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

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

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

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

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

Example

Consider the following expression:

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

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

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

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

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

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

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 $\text{let } x = e \text{ in } e'$ 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

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

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

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

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

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

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

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

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

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

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

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

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

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”.