CS 4110

Programming Languages & Logics

Lecture 17
Definitional Translation & Continuations

Definitional Translation

We know how to encode Booleans, conditionals, natural numbers, and recursion in λ-calculus.

Can we define a *real* programming language by translating everything in it into the λ-calculus?

Definitional Translation

We know how to encode Booleans, conditionals, natural numbers, and recursion in λ-calculus.

Can we define a *real* programming language by translating everything in it into the λ-calculus?

In definitional translation, we define a denotational semantics where the target is a simpler programming language instead of mathematical objects.

Multi-Argument λ-calculus

Let's define a version of the λ-calculus that allows functions to take multiple arguments.

$$e ::= x \mid \lambda x_1, \ldots, x_n.\, e \mid e_0\ e_1\ \ldots\ e_n$$

Multi-Argument λ-calculus

We can define a CBV operational semantics:

$$E ::= [\cdot] \mid v_0 \ \ldots \ v_{i-1}\, E\, e_{i+1} \ \ldots \ e_n$$

$$\frac{e \to e'}{E[e] \to E[e']} \text{ CONTEXT}$$

$$\frac{}{(\lambda x_1, \ldots, x_n.\, e_0)\; v_1 \ \ldots \ v_n \to e_0\{v_1/x_1\}\{v_2/x_2\} \ldots \{v_n/x_n\}}\ \beta$$

The evaluation contexts ensure that we evaluate multi-argument applications $e_0\; e_1 \ \ldots \ e_n$ from left to right.

Definitional Translation

The multi-argument λ-calculus isn't any more expressive that the pure λ-calculus.

Definitional Translation

The multi-argument λ-calculus isn't any more expressive that the pure λ-calculus.

We can define a translation $\mathcal{T}[\![\cdot]\!]$ that takes an expression in the multi-argument λ-calculus and returns an equivalent expression in the pure λ-calculus.

Definitional Translation

The multi-argument λ-calculus isn't any more expressive that the pure λ-calculus.

We can define a translation $\mathcal{T}[\![\cdot]\!]$ that takes an expression in the multi-argument λ-calculus and returns an equivalent expression in the pure λ-calculus.

$$
\begin{aligned}
\mathcal{T}[\![x]\!] &\triangleq x \\
\mathcal{T}[\![\lambda x_1, \ldots, x_n.\, e]\!] &\triangleq \lambda x_1.\ \ldots\ \lambda x_n.\, \mathcal{T}[\![e]\!] \\
\mathcal{T}[\![e_0\ e_1\ e_2\ \ldots\ e_n]\!] &\triangleq (\ldots((\mathcal{T}[\![e_0]\!]\ \mathcal{T}[\![e_1]\!])\ \mathcal{T}[\![e_2]\!])\ \ldots\ \mathcal{T}[\![e_n]\!])
\end{aligned}
$$

This translation *curries* the multi-argument λ-calculus.

$\Longleftrightarrow$

Products (Pairs) and Let

Syntax

$$
\begin{aligned}
e ::= {} & x \\
\mid {} & \lambda x.\, e \\
\mid {} & e_1\ e_2 \\
\mid {} & (e_1, e_2) \\
\mid {} & \#1\ e \\
\mid {} & \#2\ e \\
\mid {} & \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2
\end{aligned}
$$

$$
\begin{aligned}
v ::= {} & \lambda x.\, e \\
\mid {} & (v_1, v_2)
\end{aligned}
$$

Products (Pairs) and Let

Evaluation Contexts

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

Products (Pairs) and Let

Semantics

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

$$\frac{}{(\lambda x.\, e)\; v \to e\{v/x\}}\;\beta$$

$$\frac{}{\#1\;(v_1, v_2) \to v_1} \qquad \frac{}{\#2\;(v_1, v_2) \to v_2}$$

$$\frac{}{\mathsf{let}\; x = v \;\mathsf{in}\; e \to e\{v/x\}}$$

Products (Pairs) and Let

Translation

$$
\begin{aligned}
\mathcal{T}[\![x]\!] &= x \\
\mathcal{T}[\![\lambda x.\, e]\!] &= \lambda x.\, \mathcal{T}[\![e]\!] \\
\mathcal{T}[\![e_1\, e_2]\!] &= \mathcal{T}[\![e_1]\!]\, \mathcal{T}[\![e_2]\!] \\
\mathcal{T}[\![(e_1, e_2)]\!] &= (\lambda x.\, \lambda y.\, \lambda f.\, f\, x\, y)\, \mathcal{T}[\![e_1]\!]\, \mathcal{T}[\![e_2]\!] \\
\mathcal{T}[\![\#1\, e]\!] &= \mathcal{T}[\![e]\!]\, (\lambda x.\, \lambda y.\, x) \\
\mathcal{T}[\![\#2\, e]\!] &= \mathcal{T}[\![e]\!]\, (\lambda x.\, \lambda y.\, y) \\
\mathcal{T}[\![\mathrm{let}\, x = e_1 \,\mathrm{in}\, e_2]\!] &= (\lambda x.\, \mathcal{T}[\![e_2]\!])\, \mathcal{T}[\![e_1]\!]
\end{aligned}
$$

Laziness

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

Syntax

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

$$v ::= \lambda x.\, e$$

Semantics

$$\frac{e_1 \to e_1'}{e_1\ e_2 \to e_1'\ e_2} \qquad \frac{}{(\lambda x.\, e_1)\ e_2 \to e_1\{e_2/x\}}\ \beta$$

Laziness

Translation

$$\begin{aligned}
\mathcal{T}[\![x]\!] &= x\,(\lambda y.\, y) \\
\mathcal{T}[\![\lambda x.\, e]\!] &= \lambda x.\, \mathcal{T}[\![e]\!] \\
\mathcal{T}[\![e_1\; e_2]\!] &= \mathcal{T}[\![e_1]\!]\,(\lambda z.\, \mathcal{T}[\![e_2]\!]) \qquad z \text{ is not a free variable of } e_2
\end{aligned}$$

References

Syntax

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

$$v ::= \lambda x.\, e$$

References

Syntax

$$
\begin{aligned}
e ::= {} & x \\
\mid {} & \lambda x.\, e \\
\mid {} & e_0\; e_1 \\
\mid {} & \mathsf{ref}\; e
\end{aligned}
$$

$$v ::= \lambda x.\, e$$

References

Syntax

$$
\begin{aligned}
e ::= &\ x \\
&\mid \lambda x.\, e \\
&\mid e_0\ e_1 \\
&\mid \mathsf{ref}\ e \\
&\mid\ !e
\end{aligned}
$$

$$v ::= \lambda x.\, e$$

References

Syntax

$$
\begin{aligned}
e ::= {} & x \\
\mid {} & \lambda x.\, e \\
\mid {} & e_0\ e_1 \\
\mid {} & \mathsf{ref}\ e \\
\mid {} & \; !e \\
\mid {} & e_1 := e_2
\end{aligned}
$$

$$
v ::= \lambda x.\, e
$$

References

Syntax

$\ell \in Loc$

$$
\begin{aligned}
e ::= {} & x \\
\mid {} & \lambda x.\, e \\
\mid {} & e_0\ e_1 \\
\mid {} & \mathsf{ref}\ e \\
\mid {} & \ !e \\
\mid {} & e_1 := e_2 \\
\mid {} & \ell
\end{aligned}
$$

$$v ::= \lambda x.\, e$$

References

Syntax

$$
\begin{aligned}
e ::= \;& x \\
\mid\;& \lambda x.\, e \\
\mid\;& e_0\; e_1 \\
\mid\;& \mathsf{ref}\; e \\
\mid\;& !e \\
\mid\;& e_1 := e_2 \\
\mid\;& \ell
\end{aligned}
$$

$$
\begin{aligned}
v ::= \;& \lambda x.\, e \\
\mid\;& \ell
\end{aligned}
$$

References

Evaluation Contexts

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

References

Evaluation Contexts

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

References

Evaluation Contexts

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

References

Evaluation Contexts

$$\begin{aligned} E ::= {} & [\cdot] \\ \mid {} & E\, e \\ \mid {} & v\, E \\ \mid {} & \mathrm{ref}\ E \\ \mid {} & \,!E \\ \mid {} & E := e \end{aligned}$$

References

Evaluation Contexts

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

References

Semantics

$\sigma : Loc \to Val$

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

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

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

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)

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

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

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

Definition (Completeness)

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

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

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:

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

Example

Consider the following expression:

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

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

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

Example

Consider the following expression:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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