CS 4110

Programming Languages & Logics

Lecture 18
Evaluation Contexts and Definitional Translation

Review: Call-by-Value

Here are the syntax and CBV semantics of λ-calculus:

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

$$
\frac{e_1 \rightarrow e_1'}{e_1\, e_2 \rightarrow e_1'\, e_2} \qquad \frac{e \rightarrow e'}{v\, e \rightarrow v\, e'}
$$

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

There are two kinds of rules: *congruence rules* that specify evaluation order and *computation rules* that specify the "interesting" reductions.

Evaluation Contexts

Evaluation contexts let us separate out these two kinds of rules.

Evaluation Contexts

Evaluation contexts let us separate out these two kinds of rules.

An evaluation context E is an expression with a "hole" in it: a single occurrence of the special symbol $[\cdot]$ in place of a subexpression.

$$E ::= [\cdot] \mid E\ e \mid v\ E$$

Evaluation Contexts

Evaluation contexts let us separate out these two kinds of rules.

An evaluation context E is an expression with a "hole" in it: a single occurrence of the special symbol $[\cdot]$ in place of a subexpression.

$$E ::= [\cdot] \mid E\ e \mid v\ E$$

We write $E[e]$ to mean the evaluation context E where the hole has been replaced with the expression e.

Examples

$$E_1 = [\cdot]\,(\lambda x.\, x)$$
$$E_1[\lambda y.\, y\, y] = (\lambda y.\, y\, y)\, \lambda x.\, x$$

Examples

$$
\begin{aligned}
E_1 &= [\cdot]\,(\lambda x.\, x) \\
E_1[\lambda y.\, y\, y] &= (\lambda y.\, y\, y)\ \lambda x.\, x
\end{aligned}
$$

$$
\begin{aligned}
E_2 &= (\lambda z.\, z\, z)\,[\cdot] \\
E_2[\lambda x.\, \lambda y.\, x] &= (\lambda z.\, z\, z)\,(\lambda x.\, \lambda y.\, x)
\end{aligned}
$$

Examples

$$E_1 = [\cdot]\,(\lambda x.\, x)$$
$$E_1[\lambda y.\, y\, y] = (\lambda y.\, y\, y)\; \lambda x.\, x$$

$$E_2 = (\lambda z.\, z\, z)\,[\cdot]$$
$$E_2[\lambda x.\, \lambda y.\, x] = (\lambda z.\, z\, z)\,(\lambda x.\, \lambda y.\, x)$$

$$E_3 = ([\cdot]\; \lambda x.\, x\, x)\,((\lambda y.\, y)\,(\lambda y.\, y))$$
$$E_3[\lambda f.\, \lambda g.\, f\, g] = ((\lambda f.\, \lambda g.\, f\, g)\; \lambda x.\, x\, x)\,((\lambda y.\, y)\,(\lambda y.\, y))$$

CBV With Evaluation Contexts

With evaluation contexts, we can define the evaluation semantics for the CBV λ-calculus with just two rules: one for evaluation contexts, and one for β-reduction.

CBV With Evaluation Contexts

With evaluation contexts, we can define the evaluation semantics for the CBV λ-calculus with just two rules: one for evaluation contexts, and one for β-reduction.

With this syntax:

$$E ::= [\cdot] \mid E\, e \mid v\, E$$

The small-step rules are:

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

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

CBN With Evaluation Contexts

We can also define the semantics of CBN λ-calculus with evaluation contexts.

CBN With Evaluation Contexts

We can also define the semantics of CBN λ-calculus with evaluation contexts.

For call-by-name, the syntax for evaluation contexts is different:

$$E ::= [\cdot] \mid E\ e$$

CBN With Evaluation Contexts

We can also define the semantics of CBN λ-calculus with evaluation contexts.

For call-by-name, the syntax for evaluation contexts is different:

$$E ::= [\cdot] \mid E\, e$$

But the small-step rules are the same:

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

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

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:

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

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

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

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.

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

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

Products (Pairs) and Let

Syntax

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

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

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

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

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

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

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

Products (Pairs) and Let

Translation

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

Laziness

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

Syntax

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

Semantics

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

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

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

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

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

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

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

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

References

Semantics

$$\frac{\langle \sigma, e\rangle \rightarrow \langle \sigma', e'\rangle}{\langle \sigma, E[e]\rangle \rightarrow \langle \sigma', E[e']\rangle}$$

$$\frac{}{\langle \sigma, (\lambda x.\, e)\; v\rangle \rightarrow \langle \sigma, e\{v/x\}\rangle}\;\beta$$

$$\frac{\ell \notin dom(\sigma)}{\langle \sigma, \mathsf{ref}\; v\rangle \rightarrow \langle \sigma[\ell \mapsto v], \ell\rangle}$$

$$\frac{\sigma(\ell) = v}{\langle \sigma,\; !\ell\rangle \rightarrow \langle \sigma, v\rangle}$$

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

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)

$\forall e \in \mathbf{Exp}_{\mathrm{src}}.$ 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