

CS 4110

Programming Languages & Logics

Lecture 18
Definitional Translation



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.

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

Multi-Argument λ -calculus

We can define a CBV operational semantics:

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

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

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

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

Definitional Translation

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

Definitional Translation

The multi-argument λ -calculus isn't any more expressive than 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 than 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.

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

$$\mathcal{T}[\lambda x_1, \dots, x_n. e] = \lambda x_1. \dots \lambda x_n. \mathcal{T}[e]$$

$$\mathcal{T}[e_0 e_1 e_2 \dots e_n] = (\dots ((\mathcal{T}[e_0] \mathcal{T}[e_1]) \mathcal{T}[e_2]) \dots \mathcal{T}[e_n])$$

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

Products (Pairs) and Let

Syntax

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

Products (Pairs) 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 (Pairs) 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 (Pairs) and Let

Translation

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

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

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

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

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

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

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

Laziness

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

Syntax

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

Semantics

$$\frac{e_1 \rightarrow e'_1}{e_1 e_2 \rightarrow e'_1 e_2} \qquad \frac{}{(\lambda x. e_1) e_2 \rightarrow e_1 \{e_2/x\}} \beta$$

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

$$e ::= x$$
$$| \lambda x. e$$
$$| e_0 e_1$$
$$v ::= \lambda x. e$$

References

Syntax

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

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

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

References

Syntax

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

References

Evaluation Contexts

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

References

Evaluation Contexts

$$E ::= [\cdot]$$
$$| E e$$
$$| v E$$
$$| \text{ref } E$$

References

Evaluation Contexts

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

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

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

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

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

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)

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