university of groningen

Basic Approaches to the Semantics of Computation (BaSC)

Lecture 8: HOFL: Syntax, types, eager/lazy semantics

Jorge A. Pérez

Bernoulli Institute for Mathematics, Computer Science, and AI
University of Groningen, Groningen, the Netherlands

December 16, 2025

Previously on MaSC

- IMP: a core language for imperative programming; syntax, op. semantics, rule induction..
- Lambda-notation
- Today we look at HOFL: a core language for typed functional programming

HOFL: Syntax

$$
\begin{array}{rll}
t ::= & x \mid n \mid t_0 \text{ op } t_1 \mid \boxed{\textbf{if } t \textbf{ then } t_0 \textbf{ else } t_1} & \text{ordinary constructs} \\
& \mid (t_0, t_1) \mid \textbf{fst}(t) \mid \textbf{snd}(t) & \text{pairs, projections} \\
& \mid \lambda x.\, t \mid t_0\ t_1 & \text{abstraction, application} \\
& \mid \textbf{rec}\ x.\, t & \text{recursion}
\end{array}
$$

HOFL: Syntax

read: $\mathbf{if}\ t = 0\ \mathbf{then}\ t_0\ \mathbf{else}\ t_1$ (no Bool)

$t ::= x \mid n \mid t_0 \text{ op } t_1 \mid \mathbf{if}\ t\ \mathbf{then}\ t_0\ \mathbf{else}\ t_1$	ordinary constructs
$\mid (t_0, t_1) \mid \mathbf{fst}(t) \mid \mathbf{snd}(t)$	pairs, projections
$\mid \lambda x.\, t \mid t_0\ t_1$	abstraction, application
$\mid \mathbf{rec}\ x.\, t$	recursion

Guess the Meaning

- $\textbf{rec}\, f\,.\,\lambda x\,.\,\textbf{if}\; x\; \textbf{then}\; 1\; \textbf{else}\; x \times (f\;(x-1))$

Guess the Meaning

- $\textbf{rec}\, f\,.\,\lambda x\,.\,\textbf{if}\; x\; \textbf{then}\; 1\; \textbf{else}\; x \times (f\;(x - 1))$ (factorial)

Guess the Meaning

- $\textbf{rec}\, f\,.\,\lambda x\,.\,\textbf{if}\; x\; \textbf{then}\; 1\; \textbf{else}\; x \times (f\;(x-1))$ (factorial)
- $\textbf{rec}\, rep\,.\,\lambda n\,.\,\lambda f\,.\,\lambda x\,.\,\textbf{if}\; n\; \textbf{then}\; x\; \textbf{else}\; f\,(rep\,(n-1)\,f\;x)$

Guess the Meaning

- $\textbf{rec}\, f \,.\, \lambda x \,.\, \textbf{if}\, x \,\textbf{then}\, 1 \,\textbf{else}\, x \times (f\ (x - 1))$ (factorial)
- $\textbf{rec}\, rep \,.\, \lambda n \,.\, \lambda f \,.\, \lambda x \,.\, \textbf{if}\, n \,\textbf{then}\, x \,\textbf{else}\, f\,(rep(n - 1)\ f\ x)$ $(rep\ n\ f\ x = f^{n} x)$

Pre-terms

We call pre-terms the terms generated by the syntax:

$$
\begin{aligned}
t ::= {} & x \mid n \mid t_0 \operatorname{op} t_1 \mid \textbf{if}\ t\ \textbf{then}\ t_0\ \textbf{else}\ t_1 \\
& \mid (t_0, t_1) \mid \textbf{fst}(t) \mid \textbf{snd}(t) \\
& \mid \lambda x.\, t \mid t_0\, t_1 \\
& \mid \textbf{rec}\ x.\, t
\end{aligned}
$$

We have well-formed pre-terms, but also ill-formed ones:

Pre-terms

We call pre-terms the terms generated by the syntax:

$$
\begin{aligned}
t ::= {} & x \mid n \mid t_0 \mathrel{\mathsf{op}} t_1 \mid \textbf{if } t \textbf{ then } t_0 \textbf{ else } t_1 \\
& \mid (t_0, t_1) \mid \textbf{fst}(t) \mid \textbf{snd}(t) \\
& \mid \lambda x \,.\, t \mid t_0\, t_1 \\
& \mid \textbf{rec}\, x \,.\, t
\end{aligned}
$$

We have well-formed pre-terms, but also ill-formed ones:

- $x + 1$

Pre-terms

We call pre-terms the terms generated by the syntax:

$$
\begin{aligned}
t ::= {} & x \mid n \mid t_0 \ \mathsf{op}\ t_1 \mid \textbf{if}\ t\ \textbf{then}\ t_0\ \textbf{else}\ t_1 \\
& \mid (t_0, t_1) \mid \textbf{fst}(t) \mid \textbf{snd}(t) \\
& \mid \lambda x.\, t \mid t_0\ t_1 \\
& \mid \textbf{rec}\ x.\, t
\end{aligned}
$$

We have well-formed pre-terms, but also ill-formed ones:

- $x + 1$ ✓

Pre-terms

We call pre-terms the terms generated by the syntax:

$$
\begin{aligned}
t ::= {} & x \mid n \mid t_0 \ \mathsf{op}\ t_1 \mid \textbf{if}\ t\ \textbf{then}\ t_0\ \textbf{else}\ t_1 \\
& \mid (t_0, t_1) \mid \textbf{fst}(t) \mid \textbf{snd}(t) \\
& \mid \lambda x.\, t \mid t_0\ t_1 \\
& \mid \textbf{rec}\ x.\, t
\end{aligned}
$$

We have well-formed pre-terms, but also ill-formed ones:

- $x + 1$ ✓
- $\textbf{if}\ x\ \textbf{then}\ x + 1\ \textbf{else}\ x - 1$

Pre-terms

We call pre-terms the terms generated by the syntax:

$$
\begin{aligned}
t ::= {} & x \mid n \mid t_0 \; \mathsf{op} \; t_1 \mid \textbf{if } t \textbf{ then } t_0 \textbf{ else } t_1 \\
& \mid (t_0\,, t_1) \mid \textbf{fst}(t) \mid \textbf{snd}(t) \\
& \mid \lambda x \,.\, t \mid t_0 \; t_1 \\
& \mid \textbf{rec } x \,.\, t
\end{aligned}
$$

We have well-formed pre-terms, but also ill-formed ones:

- $x + 1$ ✓
- $\textbf{if } x \textbf{ then } x + 1 \textbf{ else } x - 1$ ✓

Pre-terms

We call pre-terms the terms generated by the syntax:

$$
\begin{aligned}
t ::= {} & x \mid n \mid t_0 \ \mathsf{op}\ t_1 \mid \textbf{if}\ t\ \textbf{then}\ t_0\ \textbf{else}\ t_1 \\
& \mid (t_0, t_1) \mid \textbf{fst}(t) \mid \textbf{snd}(t) \\
& \mid \lambda x.\, t \mid t_0\ t_1 \\
& \mid \textbf{rec}\ x.\, t
\end{aligned}
$$

We have well-formed pre-terms, but also ill-formed ones:

- $x + 1$ ✓
- $\textbf{if}\ x\ \textbf{then}\ x + 1\ \textbf{else}\ x - 1$ ✓
- $1 + (0, 5)$

Pre-terms

We call pre-terms the terms generated by the syntax:

$$\begin{aligned} t ::= {} & x \mid n \mid t_0 \text{ op } t_1 \mid \textbf{if } t \textbf{ then } t_0 \textbf{ else } t_1 \\ & \mid (t_0, t_1) \mid \textbf{fst}(t) \mid \textbf{snd}(t) \\ & \mid \lambda x \,.\, t \mid t_0 \; t_1 \\ & \mid \textbf{rec } x \,.\, t \end{aligned}$$

We have well-formed pre-terms, but also ill-formed ones:

- $x + 1$ ✓
- $\textbf{if } x \textbf{ then } x + 1 \textbf{ else } x - 1$ ✓
- $1 + (0, 5)$ ✗

Pre-terms

We call pre-terms the terms generated by the syntax:

$$\begin{aligned} t ::= {} & x \mid n \mid t_0 \,\mathsf{op}\, t_1 \mid \mathbf{if}\, t \,\mathbf{then}\, t_0 \,\mathbf{else}\, t_1 \\ & \mid (t_0, t_1) \mid \mathbf{fst}(t) \mid \mathbf{snd}(t) \\ & \mid \lambda x.\, t \mid t_0\, t_1 \\ & \mid \mathbf{rec}\, x.\, t \end{aligned}$$

We have well-formed pre-terms, but also ill-formed ones:

- $x + 1$ ✓
- $\mathbf{if}\, x \,\mathbf{then}\, x + 1 \,\mathbf{else}\, x - 1$ ✓
- $1 + (0, 5)$ ✗
- $\mathbf{fst}(3)$

Pre-terms

We call pre-terms the terms generated by the syntax:

$$
\begin{aligned}
t ::= {} & x \mid n \mid t_0 \ \mathsf{op} \ t_1 \mid \textbf{if}\ t\ \textbf{then}\ t_0\ \textbf{else}\ t_1 \\
& \mid (t_0, t_1) \mid \textbf{fst}(t) \mid \textbf{snd}(t) \\
& \mid \lambda x\,.\,t \mid t_0\ t_1 \\
& \mid \textbf{rec}\ x\,.\,t
\end{aligned}
$$

We have well-formed pre-terms, but also ill-formed ones:

- $x + 1$ ✓
- $\textbf{if}\ x\ \textbf{then}\ x + 1\ \textbf{else}\ x - 1$ ✓
- $1 + (0, 5)$ ✗
- $\textbf{fst}(3)$ ✗

Pre-terms

We call pre-terms the terms generated by the syntax:

$$
\begin{aligned}
t ::= {} & x \mid n \mid t_0 \mathbin{\mathsf{op}} t_1 \mid \textbf{if}\ t\ \textbf{then}\ t_0\ \textbf{else}\ t_1 \\
\mid {} & (t_0, t_1) \mid \textbf{fst}(t) \mid \textbf{snd}(t) \\
\mid {} & \lambda x.\, t \mid t_0\ t_1 \\
\mid {} & \textbf{rec}\ x.\, t
\end{aligned}
$$

We have well-formed pre-terms, but also ill-formed ones:

- $x + 1$ ✓
- $\textbf{if}\ x\ \textbf{then}\ x + 1\ \textbf{else}\ x - 1$ ✓
- $1 + (0, 5)$ ✗
- $\textbf{fst}(3)$ ✗

We need a type system!

Type Systems

Syntax of Types

We write $\mathcal{T}$ to denote the set of all types:

$$\tau ::= \texttt{int} \mid \tau_0 * \tau_1 \mid \tau_0 \to \tau_1$$

Syntax of Types

We write $\mathcal{T}$ to denote the set of all types:

Integers (base type)

$$\tau ::= \texttt{int} \mid \tau_0 * \tau_1 \mid \tau_0 \to \tau_1$$

Syntax of Types

We write $\mathcal{T}$ to denote the set of all types:

Integers (base type)

Pair type (aka product, related to conjunction)

$$\tau ::= \texttt{int} \mid \tau_0 * \tau_1 \mid \tau_0 \to \tau_1$$

Syntax of Types

We write $\mathcal{T}$ to denote the set of all types:

Integers (base type)

Pair type (aka product, related to conjunction)

$$\tau ::= \texttt{int} \mid \tau_0 * \tau_1 \mid \tau_0 \to \tau_1$$

Function type (aka arrow, related to implication)

Syntax of Types

$$\tau ::= \texttt{int} \mid \tau_0 * \tau_1 \mid \tau_0 \to \tau_1$$

Example

$\texttt{int} * \texttt{int}$	a pair of integers
$\texttt{int} * (\texttt{int} \to \texttt{int})$	a pair that includes a function
$(\texttt{int} * \texttt{int}) \to \texttt{int}$	function from pairs to integers

Syntax of Types

$$\tau ::= \texttt{int} \mid \tau_0 * \tau_1 \mid \tau_0 \to \tau_1$$

Example

$\texttt{int} * \texttt{int}$	a pair of integers
$\texttt{int} * (\texttt{int} \to \texttt{int})$	a pair that includes a function
$(\texttt{int} * \texttt{int}) \to \texttt{int}$	function from pairs to integers

Relating variables/identifiers to types:

- We assume variables are typed: $\mathrm{Ide} = \{\mathrm{Ide}_\tau\}_{\tau \in \mathcal{T}}$.
- There is a function $\hat{(\cdot)} : \mathrm{Ide} \to \mathcal{T}$. This way, $\hat{x}$ denotes the type of x.
- Typing judgments with formulas $t : \tau$ (read: x has type τ)
- Types are assigned to pre-terms using a set of inference rules

Type System

$$\frac{}{x : \hat{x}} \qquad \frac{}{n : \texttt{int}} \qquad \frac{t_0 : \texttt{int} \qquad t_1 : \texttt{int}}{t_0 \mathbin{\mathsf{op}} t_1 : \texttt{int}} \qquad \frac{t : \texttt{int} \qquad t_0 : \tau \qquad t_1 : \tau}{\textbf{if } t \textbf{ then } t_0 \textbf{ else } t_1 : \tau}$$

$$\frac{t_0 : \tau_0 \qquad t_1 : \tau_1}{(t_0, t_1) : \tau_0 * \tau_1} \qquad \frac{t : \tau_0 * \tau_1}{\textbf{fst}(t) : \tau_0} \qquad \frac{t : \tau_0 * \tau_1}{\textbf{snd}(t) : \tau_1} \qquad \frac{x : \tau_0 \qquad t : \tau_1}{\lambda x \,.\, t : \tau_0 \to \tau_1}$$

$$\frac{t_1 : \tau_0 \to \tau_1 \qquad t_0 : \tau_0}{t_1 \; t_0 : \tau_1} \qquad \frac{x : \tau \qquad t : \tau}{\textbf{rec } x \,.\, t : \tau}$$

Well-formed Terms

$$
\begin{aligned}
t ::= {} & x \mid n \mid t_0 \ \mathsf{op} \ t_1 \mid \mathbf{if}\ t\ \mathbf{then}\ t_0\ \mathbf{else}\ t_1 \\
& \mid (t_0, t_1) \mid \mathbf{fst}(t) \mid \mathbf{snd}(t) \\
& \mid \lambda x.\, t \mid t_0\ t_1 \\
& \mid \mathbf{rec}\ x.\, t
\end{aligned}
$$

$$
\tau ::= \mathtt{int} \mid \tau_0 * \tau_1 \mid \tau_0 \to \tau_1
$$

- A pre-term t is well-formed (well-typed, typable) if $\exists \tau \in \mathcal{T}.\, t : \tau$.
- That is, t is well-formed if we can assign a type to t using the rules.
- We only give semantics to well-formed pre-terms.
- Write T_τ to denote the set of all well-formed terms of type τ.

The Type System is Simple

Consider the example:

$$t \triangleq \textbf{rec}\, p.\, \lambda x.\, (x, p(x+2))$$

- Intuitively, this is a sequence of all even numbers

$$t\,0 \equiv (0, (t2)) \equiv (0, (2, (t4))) \equiv \cdots \equiv (0, (2, (4, \ldots)))$$

- Note: the type system is simple enough to type sequences of integers of finite length, but we have no type for sequences of arbitrary/infinite length.
- (A more powerful tool: recursive types.)

Church-style Typability ($\sim$ Type Checking)

Variables are tagged with (declared) types. We deduce the type of terms by structural induction. Example:

$$fact \triangleq \mathbf{rec}\, f : \texttt{int} \to \texttt{int}.\, \lambda x : \texttt{int}.\, \mathbf{if}\, x\, \mathbf{then}\, 1\, \mathbf{else}\, (x \times (f\,(x - 1)))$$

Using the rules:

$$\dfrac{\dfrac{\hat{f} = \texttt{int} \to \texttt{int}}{f : \texttt{int} \to \texttt{int}} \qquad \dfrac{\dfrac{\hat{x} = \texttt{int}}{x : \texttt{int}} \qquad \dfrac{\dfrac{\hat{x} = \texttt{int}}{x : \texttt{int}} \quad \dfrac{}{1 : \texttt{int}} \quad \dfrac{\dfrac{\hat{x} = \texttt{int}}{x : \texttt{int}} \quad \dfrac{f : \texttt{int} \to \texttt{int} \quad x - 1 : \texttt{int}}{f\,(x - 1) : \texttt{int}}}{(x \times (f\,(x - 1))) : \texttt{int}}}{\mathbf{if}\, x\, \mathbf{then}\, 1\, \mathbf{else}\, (x \times (f\,(x - 1))) : \texttt{int}}}{\lambda x : \texttt{int}.\, \mathbf{if}\, x\, \mathbf{then}\, 1\, \mathbf{else}\, (x \times (f\,(x - 1))) : \texttt{int} \to \texttt{int}}}{fact : \texttt{int} \to \texttt{int}}$$

Church-style Typability ($\sim$ Type Checking)

Variables are tagged with (declared) types. We deduce the type of terms by structural induction.

Alternatively:

$$
\mathit{fact} \triangleq \mathbf{rec}\ \underbrace{f}_{\texttt{int}\to\texttt{int}}\ .\ \underbrace{\lambda \underbrace{x}_{\texttt{int}}\ .\ \underbrace{\mathbf{if}\ \underbrace{x}_{\texttt{int}}\ \mathbf{then}\ \underbrace{1}_{\texttt{int}}\ \mathbf{else}\ \underbrace{(\underbrace{x}_{\texttt{int}} \times (\underbrace{\underbrace{f}_{\texttt{int}\to\texttt{int}}\ (\underbrace{\underbrace{x}_{\texttt{int}} - \underbrace{1}_{\texttt{int}}}_{\texttt{int}})}_{\texttt{int}}))}_{\texttt{int}}}_{\texttt{int}}}_{\texttt{int}\to\texttt{int}}
$$

Curry-style Typability ($\sim$ Type Inference)

- The types of variables are not given. Rather, type rules are used to derive type constraints (type equations).
- Solutions of those equations (via unification, using a signature induced by the syntax of types) define so-called principal types.

Curry-style Typability ($\sim$ Type Inference)

- The types of variables are not given. Rather, type rules are used to derive type constraints (type equations).
- Solutions of those equations (via unification, using a signature induced by the syntax of types) define so-called principal types.

Example

Consider the identity function $\lambda x \,.\, x$. We have:

$$\begin{aligned} \lambda x \,.\, x : \tau &\nwarrow_{\tau = \tau_1 \to \tau_2,\, \hat{x} = \tau_1} x : \tau_2 \\ &\nwarrow_{\hat{x} = \tau_2} \Box \end{aligned}$$

We have $\hat{x} = \tau_1 = \tau_2$. The principal type is $\tau_1 \to \tau_1$, for any arbitrary τ_1.

Curry-style Typability: Example

Recall:

$$t \triangleq \textbf{rec}\, p.\, \lambda x.\, (x, p(x+2))$$

Concisely:

$$t \triangleq \textbf{rec}\ \underbrace{p}_{\texttt{int}\to\tau_4} .\ \underbrace{\lambda\, \underbrace{x}_{\texttt{int}} .\ \underbrace{(\underbrace{x}_{\texttt{int}},\ \underbrace{\underbrace{p}_{\texttt{int}\to\tau_4}\ \underbrace{(\underbrace{x}_{\texttt{int}} + \underbrace{2}_{\texttt{int}})}_{\texttt{int}}}_{\tau_4})}_{\texttt{int} * \tau_4}}_{\texttt{int}\to(\texttt{int} * \tau_4)}$$

But then $\texttt{int} \to (\texttt{int} * \tau_4) \stackrel{?}{=} (\texttt{int} \to \tau_4)$ implies

$$\tau_4 \stackrel{?}{=} (\texttt{int} \to \tau_4)$$

Hence, unification fails (occur-check)!

Curry-style Typability: Example

In goal-oriented style:

$$
\begin{aligned}
\textbf{rec}\, p.\, \lambda x.\, (x, (p(x+2))) : \tau
&\nwarrow_{\hat{p}=\tau} \lambda x.\, (x, p(x+2)) : \tau \\
&\nwarrow_{\tau=\tau_1\to\tau_2, \hat{x}=\tau_1} (x, (p(x+2))) : \tau_2 \\
&\nwarrow_{\tau_2=\tau_3 * \tau_4} x : \tau_3,\ (p(x+2)) : \tau_4 \\
&\nwarrow_{\hat{x}=\tau_3} (p(x+2)) : \tau_4 \\
&\nwarrow\ p : \tau_5 \to \tau_4,\ (x+2) : \tau_5 \\
&\nwarrow_{\hat{p}=\tau_5\to\tau_4} (x+2) : \tau_5 \\
&\nwarrow_{\tau_5=\texttt{int}} x : \texttt{int},\ 2 : \texttt{int} \\
&\nwarrow^{*}_{\hat{x}=\texttt{int}} \square
\end{aligned}
$$

We derive:

- $\hat{x} = \tau_1$, $\hat{x} = \tau_3$, $\hat{x} = \texttt{int}$. Therefore: $\tau_1 = \tau_3 = \texttt{int}$
- $\hat{p} = \tau = \tau_1 \to \tau_2$, $\hat{p} = \tau_5 \to \tau_4$. Therefore, $\tau_1 = \tau_5 = \texttt{int}$ and $\tau_2 = \tau_4$.
- Then from $\tau_2 = \tau_4$ and $\tau_2 = \tau_3 * \tau_4$ we derive fail.

Semantics

Free Variables

$$
\begin{aligned}
\mathsf{fv}(n) &\triangleq \emptyset \\
\mathsf{fv}(x) &\triangleq \{x\} \\
\mathsf{fv}(t_0 \ \mathsf{op}\ t_1) &\triangleq \mathsf{fv}(t_0) \cup \mathsf{fv}(t_1) \\
\mathsf{fv}(\textbf{if}\ t\ \textbf{then}\ t_0\ \textbf{else}\ t_1) &\triangleq \mathsf{fv}(t) \cup \mathsf{fv}(t_0) \cup \mathsf{fv}(t_1) \\
\mathsf{fv}((t_0, t_1)) &\triangleq \mathsf{fv}(t_0) \cup \mathsf{fv}(t_1)
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{fv}(\textbf{fst}(t)) &\triangleq \mathsf{fv}(t) \\
\mathsf{fv}(\textbf{snd}(t)) &\triangleq \mathsf{fv}(t) \\
\mathsf{fv}(\lambda x\,.\,t) &\triangleq \mathsf{fv}(t) \setminus \{x\} \\
\mathsf{fv}((t_0\ t_1)) &\triangleq \mathsf{fv}(t_0) \cup \mathsf{fv}(t_1) \\
\mathsf{fv}(\textbf{rec}\ x\,.\,t) &\triangleq \mathsf{fv}(t) \setminus \{x\}
\end{aligned}
$$

Capture-Avoiding Substitution

$$
\begin{aligned}
n[{}^{t}/_{x}] &\triangleq n \\
y[{}^{t}/_{x}] &\triangleq \text{if } y = x \text{ then } t \text{ else } y \\
(t_0 \text{ op } t_1)[{}^{t}/_{x}] &\triangleq t_0[{}^{t}/_{x}] \text{ op } t_1[{}^{t}/_{x}] \\
(\textbf{if } t' \textbf{ then } t_0 \textbf{ else } t_1)[{}^{t}/_{x}] &\triangleq \textbf{if } t'[{}^{t}/_{x}] \textbf{ then } t_0[{}^{t}/_{x}] \textbf{ else } t_1[{}^{t}/_{x}] \\
(t_0, t_1)[{}^{t}/_{x}] &\triangleq (t_0[{}^{t}/_{x}], t_1[{}^{t}/_{x}]) \\
\textbf{fst}(t')[{}^{t}/_{x}] &\triangleq \textbf{fst}(t'[{}^{t}/_{x}]) \\
\textbf{snd}(t')[{}^{t}/_{x}] &\triangleq \textbf{snd}(t'[{}^{t}/_{x}]) \\
(t_0\ t_1)[{}^{t}/_{x}] &\triangleq (t_0[{}^{t}/_{x}]\ t_1[{}^{t}/_{x}]) \\
(\lambda y.\, t')[{}^{t}/_{x}] &\triangleq \lambda z.\, ((t'[{}^{z}/_{y}])[{}^{t}/_{x}]) \qquad \text{if } z \notin \text{fv}(\lambda y.\, e') \cup \text{fv}(e) \cup \{x\} \\
(\textbf{rec } x.\, t')[{}^{t}/_{x}] &\triangleq \textbf{rec } z.\, (t'[{}^{z}/_{y}])[{}^{t}/_{x}] \qquad \text{if } z \notin \text{fv}(\textbf{rec } y.\, t') \cup \text{fv}(e) \cup \{x\}
\end{aligned}
$$

Substitution Respects Types

Theorem

Given $x_0 : \tau_0$, $t_0 : \tau_0$, and $x : \tau$, then $t[{}^{t_0}/_{x_0}] : \tau$

The proof is by structural induction on the stronger assertion $t[{}^{\tilde{t}}/_{\tilde{x}}] : \tau$.

Big-Step Operational Semantics

Statements of the form:

Reads "reduces to"

$$\boxed{t \longrightarrow c}$$

A closed term

A **canonical term**:
a special kind of term

The semantics formalizes the computation of canonical forms by term manipulation.

Canonical Forms

The set of canonical forms of type τ is denoted $C_\tau \subseteq T_\tau$.

$$\frac{}{n \in C_{\texttt{int}}}$$

$$\frac{t_0 : \tau_0 \quad t_1 : \tau_1 \quad t_0, t_1 \text{ closed}}{(t_0, t_1) \in C_{\tau_0 * \tau_1}}$$

t_0, t_1 may not be in canonical form: **laziness**

$$\frac{\lambda x.\, t : \tau_0 \to \tau_1 \quad \lambda x.\, t \text{ closed}}{\lambda x.\, t \in C_{\tau_0 \to \tau_1}}$$

t not necessarily a closed term

Example: Canonical Forms?

$$\frac{}{n \in C_{\texttt{int}}} \qquad \frac{t_0 : \tau_0 \quad t_1 : \tau_1 \quad t_0, t_1 \text{ closed}}{(t_0, t_1) \in C_{\tau_0 * \tau_1}} \qquad \frac{\lambda x.\, t : \tau_0 \to \tau_1 \quad \lambda x.\, t \text{ closed}}{\lambda x.\, t \in C_{\tau_0 \to \tau_1}}$$

$1 + 2 \times 3$	$\textbf{if}\ 0\ \textbf{then}\ 0\ \textbf{else}\ 0$
$(1,2)$	$\lambda x.\, 1$
$(1+1,2\text{-}1)$	$\lambda x.\, 1 + 2 \times 3$
$\textbf{fst}(1,2)$	$\lambda x.\, \textbf{fst}(1, 2)$

Example: Canonical Forms?

$$\frac{}{n \in C_{\texttt{int}}} \qquad \frac{t_0 : \tau_0 \quad t_1 : \tau_1 \quad t_0, t_1 \text{ closed}}{(t_0, t_1) \in C_{\tau_0 * \tau_1}} \qquad \frac{\lambda x.\, t : \tau_0 \to \tau_1 \quad \lambda x.\, t \text{ closed}}{\lambda x.\, t \in C_{\tau_0 \to \tau_1}}$$

$1 + 2 \times 3$ ✗	**if** 0 **then** 0 **else** 0
(1,2)	$\lambda x.\, 1$
(1+1,2-1)	$\lambda x.\, 1 + 2 \times 3$
fst(1,2)	$\lambda x.$ **fst**$(1, 2)$

Example: Canonical Forms?

$$\frac{}{n \in C_{\texttt{int}}} \qquad \frac{t_0 : \tau_0 \quad t_1 : \tau_1 \quad t_0, t_1 \text{ closed}}{(t_0, t_1) \in C_{\tau_0 * \tau_1}} \qquad \frac{\lambda x\,.\,t : \tau_0 \to \tau_1 \quad \lambda x\,.\,t \text{ closed}}{\lambda x\,.\,t \in C_{\tau_0 \to \tau_1}}$$

$1 + 2 \times 3$ ✗	**if** 0 **then** 0 **else** 0 ✗
(1,2)	$\lambda x\,.\,1$
(1+1,2-1)	$\lambda x\,.\,1 + 2 \times 3$
fst(1,2)	$\lambda x\,.\,$**fst**$(1, 2)$

Example: Canonical Forms?

$$\frac{}{n \in C_{\texttt{int}}} \qquad \frac{t_0 : \tau_0 \quad t_1 : \tau_1 \quad t_0, t_1 \text{ closed}}{(t_0, t_1) \in C_{\tau_0 * \tau_1}} \qquad \frac{\lambda x.\, t : \tau_0 \to \tau_1 \quad \lambda x.\, t \text{ closed}}{\lambda x.\, t \in C_{\tau_0 \to \tau_1}}$$

$1 + 2 \times 3$	✗	**if** 0 **then** 0 **else** 0	✗
(1,2)	✓	$\lambda x.\, 1$	
(1+1,2-1)		$\lambda x.\, 1 + 2 \times 3$	
fst(1,2)		$\lambda x.\,$ **fst**$(1, 2)$	

Example: Canonical Forms?

$$\frac{}{n \in C_{\texttt{int}}} \qquad \frac{t_0 : \tau_0 \quad t_1 : \tau_1 \quad t_0, t_1 \text{ closed}}{(t_0, t_1) \in C_{\tau_0 * \tau_1}} \qquad \frac{\lambda x.\, t : \tau_0 \to \tau_1 \quad \lambda x.\, t \text{ closed}}{\lambda x.\, t \in C_{\tau_0 \to \tau_1}}$$

$1 + 2 \times 3$	✗	**if** 0 **then** 0 **else** 0	✗
(1,2)	✓	$\lambda x.\, 1$	✓
(1+1,2-1)		$\lambda x.\, 1 + 2 \times 3$	
fst(1,2)		$\lambda x.$ **fst**$(1, 2)$	

Example: Canonical Forms?

$$\frac{}{n \in C_{\texttt{int}}} \qquad \frac{t_0 : \tau_0 \quad t_1 : \tau_1 \quad t_0, t_1 \text{ closed}}{(t_0, t_1) \in C_{\tau_0 * \tau_1}} \qquad \frac{\lambda x.\, t : \tau_0 \to \tau_1 \quad \lambda x.\, t \text{ closed}}{\lambda x.\, t \in C_{\tau_0 \to \tau_1}}$$

$1 + 2 \times 3$	✗	**if** 0 **then** 0 **else** 0	✗
(1,2)	✓	$\lambda x.\, 1$	✓
(1+1,2-1)	✓	$\lambda x.\, 1 + 2 \times 3$	✓
fst(1,2)		$\lambda x.$ **fst**$(1, 2)$	

Example: Canonical Forms?

$$\frac{}{n \in C_{\texttt{int}}} \qquad \frac{t_0 : \tau_0 \quad t_1 : \tau_1 \quad t_0, t_1 \text{ closed}}{(t_0, t_1) \in C_{\tau_0 * \tau_1}} \qquad \frac{\lambda x.\, t : \tau_0 \to \tau_1 \quad \lambda x.\, t \text{ closed}}{\lambda x.\, t \in C_{\tau_0 \to \tau_1}}$$

$1 + 2 \times 3$	✗	**if** 0 **then** 0 **else** 0	✗
(1,2)	✓	$\lambda x.\, 1$	✓
(1+1,2-1)	✓	$\lambda x.\, 1 + 2 \times 3$	✓
fst(1,2)	✗	$\lambda x.$ **fst**$(1, 2)$	

Example: Canonical Forms?

$$\frac{}{n \in C_{\texttt{int}}} \qquad \frac{t_0 : \tau_0 \quad t_1 : \tau_1 \quad t_0, t_1 \text{ closed}}{(t_0, t_1) \in C_{\tau_0 * \tau_1}} \qquad \frac{\lambda x\,.\,t : \tau_0 \to \tau_1 \quad \lambda x\,.\,t \text{ closed}}{\lambda x\,.\,t \in C_{\tau_0 \to \tau_1}}$$

$1 + 2 \times 3$	✗	**if** 0 **then** 0 **else** 0	✗
(1,2)	✓	$\lambda x\,.\,1$	✓
(1+1,2-1)	✓	$\lambda x\,.\,1 + 2 \times 3$	✓
fst(1,2)	✗	$\lambda x\,.\,$**fst**$(1, 2)$	✓

Operational Semantics: Axioms

A single rule:

$$\frac{c \in C_\tau}{c \longrightarrow c}$$

Expanding the several cases we have:

Operational Semantics: Axioms

A single rule:

$$\frac{c \in C_\tau}{c \longrightarrow c}$$

Expanding the several cases we have:

$$\frac{}{n \longrightarrow n} \qquad \frac{t_0 : \tau_0 \quad t_1 : \tau_1 \quad t_0, t_1 \text{ closed}}{(t_0, t_1) \longrightarrow (t_0, t_1)} \qquad \frac{\lambda x . t : \tau_0 \to \tau_1 \quad \lambda x . t \text{ closed}}{\lambda x . t \longrightarrow \lambda x . t}$$

That is: integers, pairs, and abstractions are already in canonical form.

Lazy Semantics

$$\frac{}{n \longrightarrow n} \qquad \frac{t_0 : \tau_0 \quad t_1 : \tau_1 \quad t_0, t_1 \text{ closed}}{(t_0, t_1) \longrightarrow (t_0, t_1)} \qquad \frac{\lambda x . t : \tau_0 \to \tau_1 \quad \lambda x . t \text{ closed}}{\lambda x . t \longrightarrow \lambda x . t}$$

$$\frac{t_0 \longrightarrow n_0 \quad t_1 \longrightarrow n_1}{t_0 \text{ op } t_1 \longrightarrow n_0 \, op \, n_1} \qquad \frac{t \longrightarrow 0 \quad t_0 \longrightarrow c_0}{\textbf{if } t \textbf{ then } t_0 \textbf{ else } t_1 \longrightarrow c_0} \qquad \frac{t \longrightarrow (t_0, t_1) \quad t_0 \longrightarrow c_0}{\textbf{fst}(t) \longrightarrow c_0}$$

$$\frac{t[\textbf{rec}\, x . t / x] \longrightarrow c}{\textbf{rec}\, x . t \longrightarrow c} \qquad \frac{t \longrightarrow n \quad n \neq 0 \quad t_1 \longrightarrow c_1}{\textbf{if } t \textbf{ then } t_0 \textbf{ else } t_1 \longrightarrow c_1} \qquad \frac{t \longrightarrow (t_0, t_1) \quad t_1 \longrightarrow c_1}{\textbf{snd}(t) \longrightarrow c_1}$$

(lazy)

$$\frac{t_1 \longrightarrow \lambda x . t_1' \quad t_1'[t_0 / x] \longrightarrow c}{(t_1 \; t_0) \longrightarrow c}$$

Example

Let $t \triangleq \lambda x.\textbf{if}\ \textbf{fst}(x)\ \textbf{then}\ 1\ \textbf{else}\ \textbf{snd}(x) : \texttt{int} * \texttt{int} \to \texttt{int}$.
What is $t\,(1,2) \longrightarrow c$?

Example

Let $t \triangleq \lambda x.\textbf{if}\,\textbf{fst}(x)\,\textbf{then}\,1\,\textbf{else}\,\textbf{snd}(x) : \texttt{int} * \texttt{int} \to \texttt{int}$.
What is $t\,(1,2) \longrightarrow c$?

$$
\begin{aligned}
t\,(1,2) \longrightarrow c \;&\nwarrow\; t \longrightarrow \lambda x'.t',\; t'[{}^{(1,2)}/_{x'}] \longrightarrow c \\
&\nwarrow_{x'=x,\; t'=\text{if}\,\ldots}\; (\textbf{if}\,\textbf{fst}(x)\,\textbf{then}\,1\,\textbf{else}\,\textbf{snd}(x))[{}^{(1,2)}/_{x}] \longrightarrow c \\
&\qquad = \textbf{if}\,\textbf{fst}(1,2)\,\textbf{then}\,1\,\textbf{else}\,\textbf{snd}(1,2) \\
&\nwarrow\; \textbf{fst}(1,2) \longrightarrow n,\; n \neq 0,\; \textbf{snd}(1,2) \longrightarrow c \\
&\nwarrow\; (1,2) \longrightarrow (n_1,n_2),\; n_1 \longrightarrow n,\; n \neq 0,\; \textbf{snd}(1,2) \longrightarrow c \\
&\nwarrow^{*}_{n_1=1,\; n_2=2,\; n=1}\; \textbf{snd}(1,2) \longrightarrow c \\
&\nwarrow\; (1,2) \longrightarrow (n_3,n_4),\; n_4 \longrightarrow c \\
&\nwarrow^{*}_{n_3=1,\; n_4=2,\; c=2}\; \square
\end{aligned}
$$

Hence, $\boxed{t\,(1,2) \longrightarrow 2}$

Example

Let t be defined as follows:

$$t \triangleq \underbrace{\textbf{rec}\ \underbrace{x}_{\tau} \,.\, \underbrace{x}_{\tau}}_{\tau} : \tau$$

We have:

$$\begin{aligned} \textbf{rec}\, x\,.\,x \longrightarrow c \nwarrow\ & x[{}^{\textbf{rec}\, x\,.\,x}/_{x}] \longrightarrow c \\ =\ & \textbf{rec}\, x\,.\,x \longrightarrow c \end{aligned}$$

We reach the same goal from which we started, with no other option to explore: divergence!

Example

Let $fact \triangleq \textbf{rec}\, f\,.\,\lambda x\,.\,\textbf{if}\; x\; \textbf{then}\; 1\; \textbf{else}\; x \times (f\,(x-1))$. We have:

$$
\begin{aligned}
(fact\,1) \longrightarrow c \;&\nwarrow\; fact \longrightarrow \lambda x'.t',\; t'[{}^1\!/x'] \longrightarrow c \\
&\nwarrow^{*}_{x'=x,\; t'=\textsf{if}\,\ldots}\; (\textbf{if}\; x\; \textbf{then}\; 1\; \textbf{else}\; x \times (fact(x-1)))[{}^1\!/x] \longrightarrow c \\
&\qquad = \textbf{if}\; 1\; \textbf{then}\; 1\; \textbf{else}\; 1 \times (fact(1-1)) \longrightarrow c \\
&\nwarrow\; 1 \longrightarrow n,\; n \neq 0,\; 1 \times (fact(1-1)) \longrightarrow c \\
&\nwarrow^{*}_{n=1,\; c=n_1 \underline{\times} n_2}\; 1 \longrightarrow n_1,\; (fact(1-1)) \longrightarrow n_2 \\
&\nwarrow_{n=1}\; fact \longrightarrow \lambda x''.t'',\; t''\; [{}^{1-1}\!/x''] \longrightarrow n_2 \\
&\nwarrow^{*}_{x''=x,\; t''=\textsf{if}\,\ldots}\; (\textbf{if}\; x\; \textbf{then}\; 1\; \textbf{else}\; x \times (fact(x-1)))[{}^{1-1}\!/x] \longrightarrow n_2 \\
&\qquad = \textbf{if}\; 1-1\; \textbf{then}\; 1\; \textbf{else}\; (1-1) \times (fact((1-1)-1)) \\
&\nwarrow\; 1-1 \longrightarrow 0,\; 1 \longrightarrow n_2 \\
&\nwarrow^{*}_{n_2=1}\; \Box
\end{aligned}
$$

Example

Let $\mathit{fact} \triangleq \textbf{rec}\, f\,.\,\lambda x\,.\,\textbf{if}\, x\, \textbf{then}\, 1\, \textbf{else}\, x \times (f\,(x-1))$. We have:

$$
\begin{aligned}
(\mathit{fact}\,1) \longrightarrow c \;&\nwarrow\; \mathit{fact} \longrightarrow \lambda x'.t',\; t'[{}^{1}\!/x'] \longrightarrow c \\
&\nwarrow^{*}_{x'=x,\; t'=\text{if}\ldots}\; (\textbf{if}\, x\, \textbf{then}\, 1\, \textbf{else}\, x \times (\mathit{fact}(x-1)))[{}^{1}\!/x] \longrightarrow c \\
&\qquad = \textbf{if}\, 1\, \textbf{then}\, 1\, \textbf{else}\, 1 \times (\mathit{fact}(1-1)) \longrightarrow c \\
&\nwarrow\; 1 \longrightarrow n,\; n \neq 0,\; 1 \times (\mathit{fact}(1-1)) \longrightarrow c \\
&\nwarrow^{*}_{n=1,\; c=n_1 \times n_2}\; 1 \longrightarrow n_1,\; (\mathit{fact}(1-1)) \longrightarrow n_2 \qquad \text{Laziness evident here!} \\
&\nwarrow_{n=1}\; \mathit{fact} \longrightarrow \lambda x''.t'',\; t''\,[{}^{1-1}\!/x''] \longrightarrow n_2 \\
&\nwarrow^{*}_{x''=x,\; t''=\text{if}\ldots}\; (\textbf{if}\, x\, \textbf{then}\, 1\, \textbf{else}\, x \times (\mathit{fact}(x-1)))[{}^{1-1}\!/x] \longrightarrow n_2 \\
&\qquad = \textbf{if}\, 1-1\, \textbf{then}\, 1\, \textbf{else}\, (1-1) \times (\mathit{fact}((1-1)-1)) \\
&\nwarrow\; 1-1 \longrightarrow 0,\; 1 \longrightarrow n_2 \\
&\nwarrow^{*}_{n_2=1}\; \square
\end{aligned}
$$

Example

Let $\mathit{fact} \triangleq \textbf{rec}\, f\,.\,\lambda x\,.\,\textbf{if}\; x\; \textbf{then}\; 1\; \textbf{else}\; x \times (f\,(x-1))$. We have:

$$
\begin{aligned}
(\mathit{fact}\;1) \longrightarrow c \;&\nwarrow\; \mathit{fact} \longrightarrow \lambda x'.t',\; t'[{}^{1}\!/x'] \longrightarrow c \\
&\nwarrow^{*}_{x'=x,\; t'=\text{if}\ldots}\; (\textbf{if}\; x\; \textbf{then}\; 1\; \textbf{else}\; x \times (\mathit{fact}(x-1)))[{}^{1}\!/x] \longrightarrow c \\
&\qquad = \textbf{if}\; 1\; \textbf{then}\; 1\; \textbf{else}\; 1 \times (\mathit{fact}(1-1)) \longrightarrow c \\
&\nwarrow\; 1 \longrightarrow n,\; n \neq 0,\; 1 \times (\mathit{fact}(1-1)) \longrightarrow c \\
&\nwarrow^{*}_{n=1,\; c=n_1 \underline{\times} n_2}\; 1 \longrightarrow n_1,\; (\mathit{fact}(1-1)) \longrightarrow n_2 \\
&\nwarrow_{n=1}\; \mathit{fact} \longrightarrow \lambda x''.t'',\; t''\,[{}^{1-1}\!/x''] \longrightarrow n_2 \\
&\nwarrow^{*}_{x''=x,\; t''=\text{if}\ldots}\; (\textbf{if}\; x\; \textbf{then}\; 1\; \textbf{else}\; x \times (\mathit{fact}(x-1)))[{}^{1-1}\!/x] \longrightarrow n_2 \\
&\qquad = \textbf{if}\; 1-1\; \textbf{then}\; 1\; \textbf{else}\; (1-1) \times (\mathit{fact}((1-1)-1)) \\
&\nwarrow\; 1-1 \longrightarrow 0,\; 1 \longrightarrow n_2 \\
&\nwarrow^{*}_{n_2=1}\; \square
\end{aligned}
$$

Laziness evident here!

Hence, $\boxed{c = n_1 \underline{\times} n_2 = 1 \underline{\times} 1 = 1.}$

Properties of the Semantics

- Termination? $\forall t.\,\exists c.\, t \longrightarrow c$? ✗
  Counterexample: $\textbf{rec}\, x.\, x$.

Properties of the Semantics

- Termination? $\forall t.\, \exists c.\, t \longrightarrow c$? ✗
  Counterexample: **rec** $x\,.\,x$.
- Determinacy? $\forall t.\, \forall c_1, c_2.\, t \longrightarrow c_1 \wedge t \longrightarrow c_2 \Rightarrow c_1 = c_2$? ✓
  Prove $P(t \longrightarrow c) \triangleq \forall c_1.\, t \longrightarrow c_1 \Rightarrow c_1 = c$ by rule induction.

Properties of the Semantics

- Termination? $\forall t.\, \exists c.\, t \longrightarrow c$? ✗
  Counterexample: **rec** $x\,.\,x$.
- Determinacy? $\forall t.\, \forall c_1, c_2.\, t \longrightarrow c_1 \wedge t \longrightarrow c_2 \Rightarrow c_1 = c_2$? ✓
  Prove $P(t \longrightarrow c) \triangleq \forall c_1.\, t \longrightarrow c_1 \Rightarrow c_1 = c$ by rule induction.
- Subject reduction? $\forall t.\, \forall c.\, \forall \tau.\, t \longrightarrow c \wedge t : \tau \Rightarrow c : \tau$? ✓
  Prove $P(t \longrightarrow c) \triangleq \forall \tau.\, t : \tau \Rightarrow c : \tau$ by rule induction.

Properties of the Semantics

- Termination? $\forall t.\, \exists c.\, t \longrightarrow c$? ✗
  Counterexample: $\textbf{rec}\, x.\, x$.
- Determinacy? $\forall t.\, \forall c_1, c_2.\, t \longrightarrow c_1 \wedge t \longrightarrow c_2 \Rightarrow c_1 = c_2$? ✓
  Prove $P(t \longrightarrow c) \triangleq \forall c_1.\, t \longrightarrow c_1 \Rightarrow c_1 = c$ by rule induction.
- Subject reduction? $\forall t.\, \forall c.\, \forall \tau.\, t \longrightarrow c \wedge t : \tau \Rightarrow c : \tau$? ✓
  Prove $P(t \longrightarrow c) \triangleq \forall \tau.\, t : \tau \Rightarrow c : \tau$ by rule induction.
- Congruence? Given $t_1 \equiv_{\mathrm{op}} t_2$ iff $\forall c.\, (t_1 \longrightarrow c \Leftrightarrow t_2 \longrightarrow c)$.
  Is it a congruence? ✗
  Counterexample: $2 \equiv_{\mathrm{op}} 1 + 1$ but $\lambda x.\, 2 \not\equiv_{\mathrm{op}} \lambda x.\, 1 + 1$.

Lazy vs Eager Semantics

We have seen the rule

(lazy)
$$\frac{t_1 \longrightarrow \lambda x\,.\,t_1' \qquad t_1'[{}^{t_0}/_{x}] \longrightarrow c}{(t_1\ t_0) \longrightarrow c}$$

What would be an eager rule?

Lazy vs Eager Semantics

We have seen the rule

$$\text{(lazy)}\ \frac{t_1 \longrightarrow \lambda x.\, t_1' \qquad t_1'[{}^{t_0}/_{x}] \longrightarrow c}{(t_1\ t_0) \longrightarrow c}$$

What would be an eager rule?

$$\text{(eager)}\ \frac{t_1 \longrightarrow \lambda x.\, t_1' \qquad t_0 \longrightarrow c_0 \qquad t_1'[{}^{c_0}/_{x}] \longrightarrow c}{(t_1\ t_0) \longrightarrow c}$$

Lazy vs Eager Semantics, Compared (1)

$$t \triangleq (\lambda x.\, 1)(\mathbf{rec}\, y.\, y) : \mathtt{int}$$

Lazy vs Eager Semantics, Compared (1)

$$t \triangleq (\lambda x\,.\,1)(\textbf{rec}\, y\,.\,y) \;:\; \texttt{int}$$

Lazy

$$t \longrightarrow c \nwarrow \lambda x\,.\,1 \longrightarrow \lambda x'\,.\,t',\; t'[\textbf{rec}\, y\,.\,y/x'] \longrightarrow c$$

$$\nwarrow_{x'=x,\; t'=1} 1[\textbf{rec}\, y\,.\,y/x] \longrightarrow c$$

$$= 1 \longrightarrow c$$

$$\nwarrow_{c=1} \square$$

Lazy vs Eager Semantics, Compared (1)

$$t \triangleq (\lambda x.\,1)(\mathbf{rec}\; y\,.\,y) \,:\, \mathtt{int}$$

Lazy

$$t \longrightarrow c \;\nwarrow\; \lambda x.\,1 \longrightarrow \lambda x'.\,t',\; t'[\mathbf{rec}\; y\,.\,y/x'] \longrightarrow c$$
$$\nwarrow_{x'=x,\,t'=1}\; 1[\mathbf{rec}\; y\,.\,y/x] \longrightarrow c$$
$$= 1 \longrightarrow c$$
$$\nwarrow_{c=1}\; \square$$

Eager

$$t \longrightarrow c \;\nwarrow\; \lambda x.\,1 \longrightarrow \lambda x'.\,t',\; \mathbf{rec}\; y\,.\,y \longrightarrow c',\; t'[c'/x'] \longrightarrow c$$
$$\nwarrow_{x'=x,\,t'=1}\; \mathbf{rec}\; y\,.\,y \longrightarrow c',\; 1[c'/x'] \longrightarrow c$$
$$\nwarrow\; \mathbf{rec}\; y\,.\,y \longrightarrow c',\; 1[c'/x'] \longrightarrow c$$

divergence!

Lazy vs Eager Semantics, Compared (2)

$$t \triangleq (\lambda x.\, x + 1)(1 \times 2) : \texttt{int}$$

Lazy vs Eager Semantics, Compared (2)

$$t \triangleq (\lambda x.\, x + 1)(1 \times 2) : \texttt{int}$$

Lazy

$$
\begin{array}{l}
t \longrightarrow c \nwarrow \lambda x.\, x + x \longrightarrow \lambda x'.\, t', \;\; t'[^{1 \times 2}/_{x'}] \longrightarrow c \\
\nwarrow_{x'=x,\, t'=x+x} (x + x)[^{1 \times 2}/_{x}] \longrightarrow c \\
\quad = (1 \times 2) + (1 \times 2) \longrightarrow c \\
\nwarrow_{c=c_1 \underline{+} c_2} (1 \times 2) \longrightarrow c_1, \;\; (1 \times 2) \longrightarrow c_2 \\
\nwarrow^{*}_{c=c_1 \underline{+} c_2} \square \qquad\qquad c = c_1 \underline{+} c_2 = 2 \underline{+} 2 = 4
\end{array}
$$

Lazy vs Eager Semantics, Compared (2)

$$t \triangleq (\lambda x.\, x + 1)(1 \times 2) : \texttt{int}$$

Eager

$$
\begin{aligned}
t \longrightarrow c \;&\nwarrow\; \lambda x.\, x + x \longrightarrow \lambda x'.\, t',\;\; 1 \times 2 \longrightarrow c',\;\; t'[c'/x'] \longrightarrow c \\
&\nwarrow_{x'=x,\, t'=x+x}\; 1 \times 2 \longrightarrow c', (x + x)[c'/x] \longrightarrow c \\
&\nwarrow^{*}_{c'=2}\; (x + x)[2/x] \longrightarrow c \\
&\qquad = 2 + 2 \longrightarrow c \\
&\nwarrow^{*}_{c=4}\; \square
\end{aligned}
$$

Summary

We have seen HOFL and a number of different concepts:

- Type system: type checking and inference (Church vs Curry style)
- Big-step operational semantics: canonical forms
- Properties: termination (✗), determinacy (✓), subject reduction (✓), congruence (✗)
- Two evaluation strategies, leading to lazy and eager semantics
- We didn't cover: denotational semantics, which requires domain theory.

Next lectures, by Dan Frumin:

1. Semantics of low-level languages (Thursday, December 18)
2. Mechanized semantics and applications (January 6)

The End