More Symple Types
Progress And Preservation

Armando Solar-Lezama
Computer Science and Artificial Intelligence Laboratory
M.I.T.

September 28, 2015
Formalizing a Type System
Recap
Static Semantics

• Typing rules
  – Typing rules tell us how to derive typing judgments
  – Very similar to derivation rules in Big Step OS

\[
\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \quad \frac{\Gamma \vdash N : int}{\Gamma \vdash N : int} \quad \frac{\Gamma \vdash e_1 : int \quad \Gamma \vdash e_2 : int}{\Gamma \vdash e_1 + e_2 : int}
\]

• Ex. Language of Expressions
Ex. Language of Expressions

\[ \frac{x: T \in \Gamma}{\Gamma \vdash x : T} \quad \frac{\Gamma \vdash N : int}{\Gamma \vdash e_1 : int \quad \Gamma \vdash e_2 : int} \quad \frac{\Gamma \vdash e_1 + e_2 : int}{\Gamma \vdash e_1 + e_2 : int} \]

- Show that the following Judgment is valid

\[ x: int, y: int \vdash x + (y + 5) : int \]

\[ x: int, y: int \vdash x: int \quad x: int, y: int \vdash (y + 5) : int \]

\[ x: int, y: int \vdash x + (y + 5) : int \]

\[ x: int \in x: int, y: int \quad x: int, y: int \vdash y: int \quad x: int, y: int \vdash 5 : int \]

\[ x: int, y: int \vdash x: int \quad x: int, y: int \vdash (y + 5) : int \]

\[ x: int, y: int \vdash x + (y + 5) : int \]
Simply Typed $\lambda$ Calculus ($F_1$)

- **Basic Typing Rules**

- **Extensions**
Example

• Is this a valid typing judgment?

\[ \vdash (\lambda x: \text{bool} \ \lambda y: \text{int} \ \text{if } x \ \text{then } y \ \text{else } y + 1): \text{bool} \rightarrow \text{int} \rightarrow \text{int} \]

• How about this one?

\[ \vdash (\lambda x: \text{int} \ \lambda y: \text{bool} \ x + y): \text{int} \rightarrow \text{bool} \rightarrow \text{int} \]
Example

• What’s the type of this function?

\((\lambda f. \lambda x. \text{if } x = 1 \text{ then } x \text{ else } (f \ f \ (x-1)) \ast x)\)

- Hint: This IS a trick question
Simply Typed $\lambda$ Calculus ($F_1$)

- We have defined a really strong type system on $\lambda$-calculus
  - It’s so strong, it won’t even let us write non-terminating computation
  - We can actually prove this!
Progress and Preservation
What makes a type system “correct”

• “Well typed programs never go wrong”

• Inductive argument
  – **Preservation**: If a program is well typed it will stay well typed in the next step of evaluation
  – **Progress**: If a program is well typed now, it won’t go wrong in the next step of evaluation

• What do we mean by “step of evaluation”?
Preservation

- Using Big-Step semantics we can argue global preservation

\[ \Gamma \vdash e_1: \tau \land e_1 \rightarrow e_2 \Rightarrow \Gamma \vdash e_2: \tau \]

- Prove by induction on the structure of derivation of \( e_1 \rightarrow e_2 \)
Proof by induction on Structure of Evaluation

• Base cases: trivial

\[ x \rightarrow x \quad \quad \quad \quad \lambda x. e \rightarrow \lambda x. e \]

• Inductive case is a little trickier

\[
\begin{align*}
e_1 \rightarrow \lambda x. e'_1 & \quad e'_1[e_2/x] \rightarrow e_3 \\
e_1 e_2 & \rightarrow e_3
\end{align*}
\]
Induction on the Structure of the Derivation

- Inductive case

\[
\begin{align*}
\frac{e_1 \rightarrow \lambda x. e'_1}{e_1[e'_2/x] \rightarrow e_3} & \quad e_1 \ e_2 \rightarrow e_3 \\
\end{align*}
\]

- Given \( \Gamma \vdash e_1 e_2 : \tau_{e12} \) we want to show that \( \Gamma \vdash e_3 : \tau_{e12} \)
- By our typing rule, we have

\[
\frac{\Gamma \vdash e_1 : \tau' \rightarrow \tau_{e12} \quad \Gamma \vdash e_2 : \tau'}{\Gamma \vdash e_1 e_2 : \tau_{e12}}
\]

- And by the IH, we have that \( \lambda x. e'_1 : \tau' \rightarrow \tau_{e12} \)
- Which again by the typing rule

\[
\frac{\Gamma, x : \tau' \vdash e'_1 : \tau_{e12}}{\Gamma \vdash (\lambda x : \tau' e'_1) : \tau \rightarrow \tau_{e12}}
\]

- Now, we need to show that

\[
\Gamma, x : \tau' \vdash e'_1 : \tau_{e12} \land \Gamma \vdash e_2 : \tau' \Rightarrow \Gamma \vdash e'_1[e_2/x] : \tau_{e12}
\]

- And from our IH

\[
\Gamma \vdash e'_1[e_2/x] : \tau_{e12} \Rightarrow \Gamma \vdash e_3 : \tau_{e12}
\]
Small Step Semantics

- Big step goes directly from initial program to result

- Small Step evaluates one step at a time
Small Step Example

• Contexts
  
  \[ H ::= o | H \, e_1 | H + e | n + H | \]
  \[ \text{if } H \text{ then } e_1 \text{ else } e_2 | \]
  \[ H == e_1 | n == H \]

• Local Reduction Rules
  
  – \( n_1 + n_2 \rightarrow n \) (where \( n = \text{plus} \, n_1 \, n_2 \))
  – \( n_1 == n_2 \rightarrow b \) (where \( b = \text{equals} \, n_1 \, n_2 \))
  – \( \text{if true then } e_1 \text{ else } e_2 \rightarrow e_1 \)
  – \( \text{if false then } e_1 \text{ else } e_2 \rightarrow e_2 \)
  – \( (\lambda x: \tau . e_1) \, v_2 \rightarrow [v_2/x] \, e_1 \)

• Global Reduction Rules
  
  – \( H[r] \rightarrow H[e] \) iff \( r \rightarrow e \)
The proof strategy

- **Progress Theorem**
  
  If $\vdash e : \tau$ and $e$ is not a value, then there is an $e'$ s.t. $e \rightarrow e'$

- **We can prove this through a decomposition lemma**
  
  - If $\vdash e : \tau$ and $e$ is not a value, then there are $H$ and $r$ s.t. $e = H[r]$
  
  - This guarantees one step of progress
Proving the Progress Theorem

If \( \vdash e : \tau \) and \( e \) is not a value, then there is an \( e' \) s.t. 
\( e \to e' \)

or equivalently, \( e = \text{H}[r] \)

• Proved by induction on the derivation of 
\( \vdash e : \tau \)

• Base case:
  – Irreducible values

\[
\begin{align*}
\Gamma \vdash false : bool & \quad \Gamma \vdash N : int \\
\Gamma \vdash x : \tau & \quad \Gamma \vdash true : bool \\
\Gamma, x: \tau_1 \vdash e: \tau_2 & \\
\Gamma \vdash (\lambda x: \tau_1 e) : \tau_1 \to \tau_2
\end{align*}
\]
Proving the Progress Theorem

• Inductive case

\[
\begin{align*}
\Gamma \vdash e : \text{bool} & \quad \Gamma \vdash e_t : \tau & \quad \Gamma \vdash e_f : \tau \\
\hline \\
\Gamma \vdash \text{if } e \text{ then } e_t \text{ else } e_f : \tau
\end{align*}
\]

– by the IH, e can be irreducible,
  • in which case it must be true or false and the whole thing is a redex

– Or, it can be decomposed into \( H[r] \)
  • in which case if \( H \) then \( e_1 \) else \( e_2 \) is a valid context.