Lecture 10: The Simply Typed λ-Calculus

Typechecker implementation and some extra types

Polyvios Pratikakis

Computer Science Department, University of Crete

Type Systems and Programming Languages

Last time

- Function types: $T \to T'$, $\to$ is a *type constructor*
- Add booleans as a *base type*: inductive type language T
- Add type declarations for function arguments $\lambda x : T.e$
- Typing relation $\Gamma \vdash e : T$
- ...uses *context* Γ to type free variables (α-renamed to be unique)
- Also called typing *judgement*
- Defined recursively (with type rules)

Typed λ-calculus definition

$$
\begin{array}{lrcl}
\text{Term language} & e & ::= & e\ e \mid \lambda x : T.e \mid x \\
 & & \mid & \mathsf{true} \mid \mathsf{false} \mid \mathsf{if}\ e\ \mathsf{then}\ e\ \mathsf{else}\ e \\
\text{Values} & v & ::= & \lambda x : T.e \mid \mathsf{true} \mid \mathsf{false} \\
\text{Type language} & T & ::= & T \to T \mid \mathit{Bool}
\end{array}
$$

$$
[\text{T-Abs}]\ \frac{\Gamma, x : T_1 \vdash e_2 : T_2}{\Gamma \vdash \lambda x : T_1.e_2 : T_1 \to T_2} \qquad [\text{T-Var}]\ \frac{x : T \in \Gamma}{\Gamma \vdash x : T}
$$

$$
[\text{T-App}]\ \frac{\Gamma \vdash e_1 : T \to T' \quad \Gamma \vdash e_2 : T}{\Gamma \vdash e_1\ e_2 : T'} \qquad [\text{T-If}]\ \frac{\Gamma \vdash e_1 : \mathit{Bool} \quad \Gamma \vdash e_2 : T \quad \Gamma \vdash e_3 : T}{\Gamma \vdash \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 : T}
$$

$$
[\text{T-True}]\ \frac{}{\Gamma \vdash \mathsf{true} : \mathit{Bool}} \qquad [\text{T-False}]\ \frac{}{\Gamma \vdash \mathsf{false} : \mathit{Bool}}
$$

Metatheory

- Progress theorem
  - If $\vdash e : T$ then either e is a value, or it can take a step $e \rightarrow e'$ to some e'
  - Proof by induction on $\Gamma \vdash e : T$
- Preservation theorem
  - If $\vdash e : T$ and $e \rightarrow e'$ then $e' : T$
  - Proof by induction on $e \rightarrow e'$, using lemmas
  - Permutation lemma
    - ★ If $\Gamma \vdash e : T$ and Γ' is a permutation of Γ, then $\Gamma' \vdash e : T$
    - ★ Proof by induction on $\Gamma \vdash e : T$
  - Weakening lemma (by induction on $\Gamma \vdash e : T$)
    - ★ If $\Gamma \vdash e : T$ and $x \notin dom(\Gamma)$, then $\Gamma, x : T' \vdash e : T$
  - Substitution lemma (by induction on $\Gamma, x : T' \vdash e : T$)
    - ★ If $\Gamma, x : T' \vdash e : T$ and $\Gamma \vdash e' : T'$, then $\Gamma \vdash e[e'/x] : T$

Typecheck functions

- Lambda term:

$$[\text{T-ABS}]\frac{\Gamma, x : T_1 \vdash e_2 : T_2}{\Gamma \vdash \lambda x : T_1 . e_2 : T_1 \to T_2}$$

- If the program is a lambda term of the form $\lambda x : T_1 . e_2$
- Add $x : T_1$ to the input environment Γ and using that, find the type of e_2
  - Before adding $x : T_1$ to Γ, make sure it is not already bound
- If the type of $e_2 : T_2$, then return the type $T_1 \to T_2$

Typecheck variables

- Variable:

$$[\text{T-VAR}]\frac{x : T \in \Gamma}{\Gamma \vdash x : T}$$

- If the program is a variable x
- Check in the input environment Γ for any binding of x: $(x : T) \in \Gamma$
- If there is one, return type T
- If not, the variable x is undeclared in the program: type error

Typecheck application

- If the program is an application $e_1\ e_2$

$$[\text{T-App}]\frac{\Gamma \vdash e_1 : T \to T' \quad \Gamma \vdash e_2 : T}{\Gamma \vdash e_1\ e_2 : T'}$$

- Find the type T_1 of e_1 using the input environment Γ
- Find the type T_2 of e_2 using the same input environment Γ
- Check whether T_1 is a function type
- If it is, check that the argument type matches T_2
- If it does, return the result type of the function T_1

- ...and so on, for the other cases
- If you cannot apply a rule to compute a type: type error

A small revision

- Prove substitution lemma (one case each)
- If $\Gamma, x : T' \vdash e : T$ and $\Gamma \vdash e' : T'$, then $\Gamma \vdash e[e'/x] : T$
- Proof by induction on $\Gamma, x : T' \vdash e : T$

Adding more types

- The unit type
- Pairs (the product type)
- Binary unions (the sum type)

The unit type

- Syntax

$$\begin{array}{rcl} e & ::= & \ldots \mid () \mid e_1; e_2 \\ v & ::= & \ldots \mid () \\ T & ::= & \ldots \mid \mathit{Unit} \end{array}$$

- Typing

$$[\text{T-Unit}]\frac{}{\Gamma \vdash () : \mathit{Unit}} \qquad [\text{T-Seq}]\frac{\Gamma \vdash e_1 : \mathit{Unit} \quad \Gamma \vdash e_2 : T}{\Gamma \vdash e_1; e_2 : T}$$

- Semantics

$$\frac{}{(); e_2 \to e_2} \qquad \frac{e_1 \to e_1'}{e_1; e_2 \to e_1'; e_2}$$

Pairs: the product type

- Syntax

$$\begin{array}{rcl} e & ::= & \ldots \mid e.1 \mid e.2 \mid (e, e) \\ v & ::= & \ldots \mid (v, v) \\ T & ::= & \ldots \mid T \times T \end{array}$$

- Typing

$$[\text{T-Pair}]\frac{\Gamma \vdash e_1 : T_1 \qquad \Gamma \vdash e_2 : T_2}{\Gamma \vdash (e_1, e_2) : T_1 \times T_2}$$

$$[\text{T-Fst}]\frac{\Gamma \vdash e : T \times T'}{\Gamma \vdash e.1 : T} \qquad [\text{T-Snd}]\frac{\Gamma \vdash e : T \times T'}{\Gamma \vdash e.2 : T'}$$

Pairs: the product type (cont'd)

- Semantics

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

$$\frac{e_1 \to e_1'}{(e_1, e_2) \to (e_1', e_2)} \qquad \frac{e_2 \to e_2'}{(v_1, e_2) \to (v_1, e_2')}$$

$$\frac{e \to e'}{e.1 \to e'.1} \qquad \frac{e \to e'}{e.2 \to e'.2}$$

- Adapt progress and preservation ...

Binary unions: the sum type

- Syntax

$$
\begin{array}{rcl}
e & ::= & \ldots \mid \mathsf{inl}\ e \mid \mathsf{inr}\ e \mid \mathsf{case}\ e\ \mathsf{of}\ \mathsf{inl}\ x \Rightarrow e \mid \mathsf{inr}\ x \Rightarrow e \\
v & ::= & \ldots \mid \mathsf{inl}\ v \mid \mathsf{inr}\ v \\
T & ::= & \ldots \mid T + T
\end{array}
$$

- Typing

$$[\text{T-INL}]\ \frac{\Gamma \vdash e : T_1}{\Gamma \vdash \mathsf{inl}\ e : T_1 + T_2} \qquad [\text{T-INR}]\ \frac{\Gamma \vdash e : T_2}{\Gamma \vdash \mathsf{inr}\ e : T_1 + T_2}$$

$$[\text{T-CASE}]\ \frac{\begin{array}{c}\Gamma \vdash e : T_1 + T_2 \\ \Gamma, x_1 : T_1 \vdash e_1 : T \\ \Gamma, x_2 : T_2 \vdash e_2 : T\end{array}}{\Gamma \vdash \mathsf{case}\ e\ \mathsf{of}\ \mathsf{inl}\ x_1 \Rightarrow e_1 \mid \mathsf{inr}\ x_2 \Rightarrow e_2 : T}$$

Binary unions: the sum type (cont'd)

- Semantics

$$\frac{e \to e'}{\text{inl } e \to \text{inl } e'} \qquad \frac{e \to e'}{\text{inr } e \to \text{inr } e'}$$

$$\frac{e \to e'}{\begin{array}{c}\text{case } e \text{ of inl } x_1 => e_1 \mid \text{inr } x_2 => e_2 \to \\ \text{case } e' \text{ of inl } x_1 => e_1 \mid \text{inr } x_2 => e_2\end{array}}$$

$$\frac{}{\text{case } (\text{inl } v) \text{ of inl } x_1 => e_1 \mid \text{inr } x_2 => e_2 \to e_1[v/x_1]}$$

$$\frac{}{\text{case } (\text{inr } v) \text{ of inl } x_1 => e_1 \mid \text{inr } x_2 => e_2 \to e_2[v/x_2]}$$