Chapter 9: Simply Typed Lambda-Calculus

Function Types

The Typing Relation

Properties of Typing

The Curry-Howard Correspondence

Erasure and Typability

Function Types

- T1→T2
  - classifying functions that expect arguments of type T1 and return results of type T2.

  (The type constructor → is right-associative.

  T1→T2→T3 stands for T1→(T2→T3))

- We will consider Booleans with lambda calculus
  - T ::= Bool

    T → T

- Examples
  - Bool→Bool
  - (Bool→Bool) → (Bool→Bool)

$\lambda\rightarrow$

Syntax

t	::=		*terms:*
		x	*variable*
		$\lambda x{:}T.t$	*abstraction*
		$t\ t$	*application*
v	::=		*values:*
		$\lambda x{:}T.t$	*abstraction value*
T	::=		*types:*
		$T\to T$	*type of functions*
Γ	::=		*contexts:*
		$\varnothing$	*empty context*
		$\Gamma, x{:}T$	*term variable binding*

Assume all variables in Γ are different

Evaluation $\boxed{t \longrightarrow t'}$

$$\frac{t_1 \longrightarrow t_1'}{t_1\ t_2 \longrightarrow t_1'\ t_2} \quad \text{(E-APP1)}$$

$$\frac{t_2 \longrightarrow t_2'}{v_1\ t_2 \longrightarrow v_1\ t_2'} \quad \text{(E-APP2)}$$

$$(\lambda x{:}T_{11}.t_{12})\ v_2 \longrightarrow [x \mapsto v_2]t_{12} \quad \text{(E-APPABS)}$$

Typing $\boxed{\Gamma \vdash t : T}$

$$\frac{x{:}T \in \Gamma}{\Gamma \vdash x : T} \quad \text{(T-VAR)}$$

$$\frac{\Gamma, x{:}T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x{:}T_1.t_2 : T_1\to T_2} \quad \text{(T-ABS)}$$

$$\frac{\Gamma \vdash t_1 : T_{11}\to T_{12} \qquad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12}} \quad \text{(T-APP)}$$

Type Derivation Tree

$$\dfrac{\dfrac{\dfrac{\texttt{x:Bool} \in \texttt{x:Bool}}{\texttt{x:Bool} \vdash \texttt{x} : \texttt{Bool}}\,\textsc{T-Var}}{\vdash \lambda\texttt{x:Bool.x} : \texttt{Bool}\to\texttt{Bool}}\,\textsc{T-Abs} \qquad \dfrac{}{\vdash \texttt{true} : \texttt{Bool}}\,\textsc{T-True}}{\vdash (\lambda\texttt{x:Bool.x})\ \texttt{true} : \texttt{Bool}}\,\textsc{T-App}$$

Properties of Typing

Inversion Lemma

Uniqueness of Types

Canonical Forms

Safety: Progress + Preservation

Inversion Lemma

LEMMA [INVERSION OF THE TYPING RELATION]:

1. If $\Gamma \vdash \mathtt{x} : \mathsf{R}$, then $\mathtt{x}:\mathsf{R} \in \Gamma$.
2. If $\Gamma \vdash \lambda \mathtt{x}:\mathsf{T}_1 .\ \mathtt{t}_2 : \mathsf{R}$, then $\mathsf{R} = \mathsf{T}_1 \rightarrow \mathsf{R}_2$ for some R_2 with $\Gamma, \mathtt{x}:\mathsf{T}_1 \vdash \mathtt{t}_2 : \mathsf{R}_2$.
3. If $\Gamma \vdash \mathtt{t}_1\ \mathtt{t}_2 : \mathsf{R}$, then there is some type T_{11} such that $\Gamma \vdash \mathtt{t}_1 : \mathsf{T}_{11} \rightarrow \mathsf{R}$ and $\Gamma \vdash \mathtt{t}_2 : \mathsf{T}_{11}$.
4. If $\Gamma \vdash \mathtt{true} : \mathsf{R}$, then $\mathsf{R} = \mathtt{Bool}$.
5. If $\Gamma \vdash \mathtt{false} : \mathsf{R}$, then $\mathsf{R} = \mathtt{Bool}$.
6. If $\Gamma \vdash \mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3 : \mathsf{R}$, then $\Gamma \vdash \mathtt{t}_1 : \mathtt{Bool}$ and $\Gamma \vdash \mathtt{t}_2, \mathtt{t}_3 : \mathsf{R}$. $\square$

Exercise: Is there any context Γ and type T such that $\Gamma \vdash x\ x:T$?

Uniqueness of Types

- **Theorem** [Uniqueness of Types]: In a given typing context Γ, a term t (with free variables all in the domain of Γ) has at most one type. Moreover, there is just one derivation of this typing built from the inference rules that generate the typing relation.

Canonical Form

- **Lemma** [Canonical Forms]:
  - If v is a value of type Bool, then v is either true or false.
  - If v is a value of type $T_1 \rightarrow T_2$, then v = $\lambda x{:}T_1.t_2$.

Progress

- **Theorem** [Progress]: Suppose t is a closed, well-typed term (that is, $\vdash t : T$ for some T). Then either t is a value or else there is some t′ with $t \to t'$.

Proof: By induction on typing derivations.

Two Structural Lemmas

- **Lemma** [Permutation]: If $\Gamma \vdash t : T$ and Δ is a permutation of Γ, then $\Delta \vdash t : T$.

- **Lemma** [Weakening]: If $\Gamma \vdash t{:}T$ and x is not in dom(Γ), then $\Gamma, x{:}S \vdash t{:}T$.

Note: All can be easily proved by induction on derivation

Preservation

- **Lemma** [Preservation of types under substitution]: If Γ, $x{:}S \vdash t{:}T$ and $\Gamma \vdash s{:}S$,

then $\Gamma \vdash [x \to s]t{:}T$.

Proof: By induction on derivation of $\Gamma, x{:}S \vdash t : T$.

- **Theorem** [Preservation]:

If $\Gamma \vdash t{:}T$ and $t \to t'$, then $\Gamma \vdash t' : T$.

The Curry-Howard Correspondence

- A connection between logic and type theory

LOGIC	PROGRAMMING LANGUAGES
propositions	types
proposition $P \supset Q$	type $\mathsf{P} \rightarrow \mathsf{Q}$
proposition $P \wedge Q$	type $\mathsf{P} \times \mathsf{Q}$ (see §11.6)
proof of proposition P	term t of type P
proposition P is provable	type P is inhabited (by some term)

Erasure and Typability

- Types are used during type checking, but do not appear in the compiled form of the program.

DEFINITION: The *erasure* of a simply typed term t is defined as follows:

$$
\begin{aligned}
erase(x) &= x \\
erase(\lambda x{:}T_1.\ t_2) &= \lambda x.\ erase(t_2) \\
erase(t_1\ t_2) &= erase(t_1)\ erase(t_2)
\end{aligned}
$$

THEOREM:

1. If $t \longrightarrow t'$ under the typed evaluation relation, then $erase(t) \longrightarrow erase(t')$.
2. If $erase(t) \longrightarrow m'$ under the typed evaluation relation, then there is a simply typed term t' such that $t \longrightarrow t'$ and $erase(t') = m'$. □

Untyped?

Curry-Style vs. Church-Style

- Curry Style
  - Syntax → Semantics → Typing
  - Semantics is defined on untyped terms
  - Often used for implicit typed languages

- Church Style
  - Syntax → Typing → Semantics
  - Sematnics is defined only on well-typed terms
  - Often used for explicit typed languages

Homework

- Read Chapter 9.
- Do Exercise 9.3.9.

9.3.9 THEOREM [PRESERVATION]: If $\Gamma \vdash t : T$ and $t \longrightarrow t'$, then $\Gamma \vdash t' : T$. □

Proof: EXERCISE [RECOMMENDED, ★★★]. The structure is very similar to the proof of the type preservation theorem for arithmetic expressions (8.3.3), except for the use of the substitution lemma. □