编程语言的设计原理

Design Principles of Programming Languages

Zhenjiang Hu, Haiyan Zhao,

胡振江　赵海燕

Peking University, Spring, 2022

Recap

The Typing Relation

t : T

Types

- Values have two possible "shapes": they are either *booleans* or *numbers*.

T ::=		*types*
	Bool	*type of booleans*
	Nat	*type of numbers*

Typing Rules

$$\texttt{true} : \texttt{Bool} \qquad \text{(T-True)}$$

$$\texttt{false} : \texttt{Bool} \qquad \text{(T-False)}$$

$$\frac{\texttt{t}_1 : \texttt{Bool} \qquad \texttt{t}_2 : \texttt{T} \qquad \texttt{t}_3 : \texttt{T}}{\texttt{if } \texttt{t}_1 \texttt{ then } \texttt{t}_2 \texttt{ else } \texttt{t}_3 : \texttt{T}} \qquad \text{(T-If)}$$

$$\texttt{0} : \texttt{Nat} \qquad \text{(T-Zero)}$$

$$\frac{\texttt{t}_1 : \texttt{Nat}}{\texttt{succ } \texttt{t}_1 : \texttt{Nat}} \qquad \text{(T-Succ)}$$

$$\frac{\texttt{t}_1 : \texttt{Nat}}{\texttt{pred } \texttt{t}_1 : \texttt{Nat}} \qquad \text{(T-Pred)}$$

$$\frac{\texttt{t}_1 : \texttt{Nat}}{\texttt{iszero } \texttt{t}_1 : \texttt{Bool}} \qquad \text{(T-IsZero)}$$

Typing Relation: Formal Definition

- **Definition**:

  the *typing relation* for arithmetic expressions is the *smallest binary relation* between *terms* and *types* satisfying **all instances** of the typing rules.

- A term t is *typable* (or *well typed*) if there is some T such that $t : T$.

Chapter 9:
Simply Typed Lambda-Calculus

Function Types

The Typing Relation

Properties of Typing

The Curry-Howard Correspondence

Erasure and Typability

The simply typed lambda-calculus

- The system we are about to define is commonly called the *simply typed lambda-calculus*, $\lambda_{\rightarrow}$, for short.
- Unlike the *untyped lambda-calculus*, the "pure" form of $\lambda_{\rightarrow}$ (with no primitive values or operations) is not very interesting; to talk about $\lambda_{\rightarrow}$, we always begin with some set of "base types."
  - So, strictly speaking, there are many variants of $\lambda_{\rightarrow}$, depending on *the choice of base types*.
  - For now, we'll work with *a variant constructed over the booleans*.

Function Types

- $T_1 \longrightarrow T_2$
  - classifying functions that expect arguments of type T1 and return results of type T2.
- the type constructor $\longrightarrow$ is *right-associative*, e.g.,

  $T_1 \longrightarrow T_2 \longrightarrow T_3$ stands for $T_1 \longrightarrow (T_2 \longrightarrow T_3)$

- Let's consider *Booleans* with lambda calculus

T ::=		types :
	Bool	type of booleans
	T ⟶ T	type of functions

- Examples
  - Bool ⟶ Bool
  - (Bool ⟶ Bool) ⟶ (Bool ⟶ Bool)

Typing rules

$$\texttt{true} : \texttt{Bool} \qquad \text{(T-True)}$$

$$\texttt{false} : \texttt{Bool} \qquad \text{(T-False)}$$

$$\frac{t_1 : \texttt{Bool} \quad t_2 : T \quad t_3 : T}{\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 : T} \qquad \text{(T-If)}$$

$$\frac{???}{\lambda x{:}\, T_1.\, t_2 : T_1 \longrightarrow T_2} \qquad \text{(T-Abs)}$$

$\lambda_{\rightarrow}$

Syntax

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

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

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

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

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

Assume:

all variables in Γ are different via renaming/internal

$\lambda_{\rightarrow}$

- What is the relation between these two statements?
  1. $t : T$
  2. $\vdash t : T$

  these two relations are completely different things.
- We are dealing with *several different small programming languages*, each with *its own typing relation* (between terms in that language and types in that language)
  - for the *simple language of numbers* and *booleans*, typing is a *binary relation* between terms and types ($t : T$).
  - for $\lambda_{\rightarrow}$, typing is a *ternary relation* between *contexts*, *terms*, and *types* ($\Gamma \vdash t : T$, $\vdash t : T$ if $\Gamma = \emptyset$)

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

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

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.

Progress

- **Theorem** [Progress]:

  Suppose *t* is a *closed*, *well-typed term*. Then either *t* is a value or else there is some t′ with $t \rightarrow t'$.

  Proof: By induction on typing derivations.

  – The cases for *Boolean constants* and *conditions* are the same as before.

  – The *variable case* is trivial (cannot occur because t is closed).

  – The *abstraction case* is immediate, since abstractions are values.

  – The *case for application*, by induction.

- *Closed*: No free variable
- *Well-typed*: $\vdash t : T$ for some T

Preservation

- **Theorem** [Preservation]:

  If $\Gamma \vdash t: T$ and $t \longrightarrow t'$, then $\Gamma \vdash t' : T$.

  Proof: By induction on typing derivations.

- **Substitution Lemma** [Preservation of types under substitution]:

  if $\Gamma, x: S \vdash t: T$ and $\Gamma \vdash s: S$,

  Then $\Gamma \vdash [x \mapsto s]\, t: T$.

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

  cases on the possible *shape of t*.

The Curry-Howard Correspondence

- A connection between logic and type theory

Logic	Programming languages
propositions	types
proposition $P \supset Q$	type P→Q
proposition $P \land Q$	type P × 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 need to appear* in the compiled form of the program.
- Terms in $\lambda_{\rightarrow}$ can be transformed to terms of *the untyped lambda-calculus* simply by *erasing type annotations* on lambda-abstractions.

$$
\begin{aligned}
erase(\mathtt{x}) &= \mathtt{x} \\
erase(\lambda \mathtt{x}{:}\mathtt{T}_1.\ \mathtt{t}_2) &= \lambda \mathtt{x}.\ erase(\mathtt{t}_2) \\
erase(\mathtt{t}_1\ \mathtt{t}_2) &= erase(\mathtt{t}_1)\ erase(\mathtt{t}_2)
\end{aligned}
$$

Erasure and Typability

- Conversely, an untyped λ-term m is said to be *typable* if there is some term t in the simply typed λ-calculus, some type T, and some context Γ such that

$$erase(t) = m \quad \text{and} \quad \Gamma \vdash t : T$$

This process is called *type reconstruction* or *type inference*.

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
  - Semantics is defined only on *well-typed terms*
  - Often used for *explicit* typed languages

Homework

- Read through Chapter 9.
- Do Exercise 9.3.9.

THEOREM [PRESERVATION]: If $\Gamma \vdash \mathtt{t} : \mathsf{T}$ and $\mathtt{t} \longrightarrow \mathtt{t}'$, then $\Gamma \vdash \mathtt{t}' : \mathsf{T}$. $\square$

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. $\square$